Appendices A and B

NCHRP Web-Only Document 387

Safety Prediction Model for Freeway Facilities with High Occupancy Lanes

NCHRP Project 17-89A

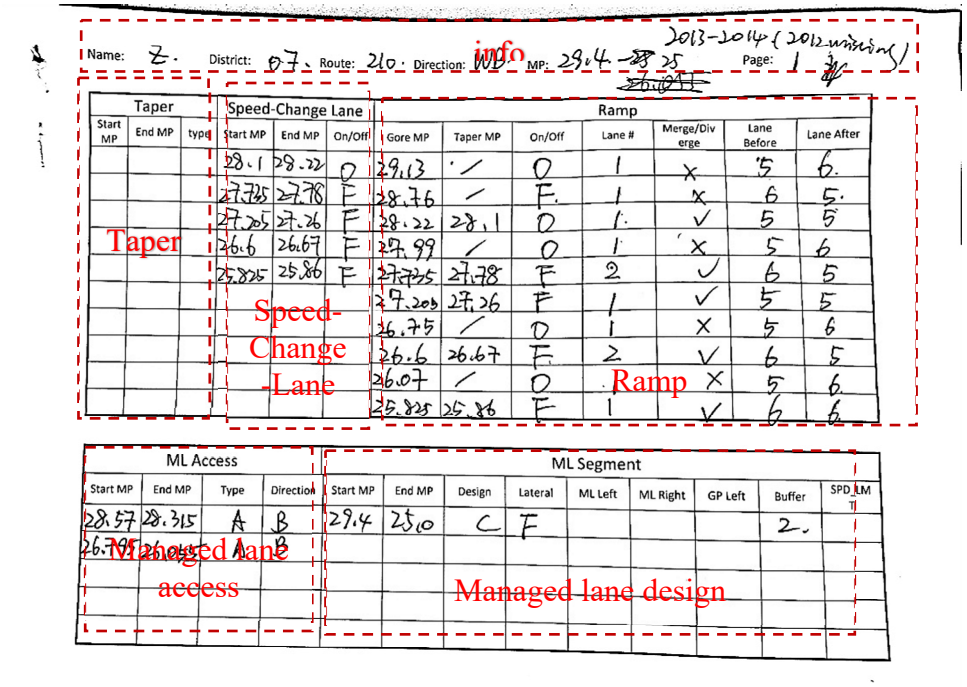
December 2021

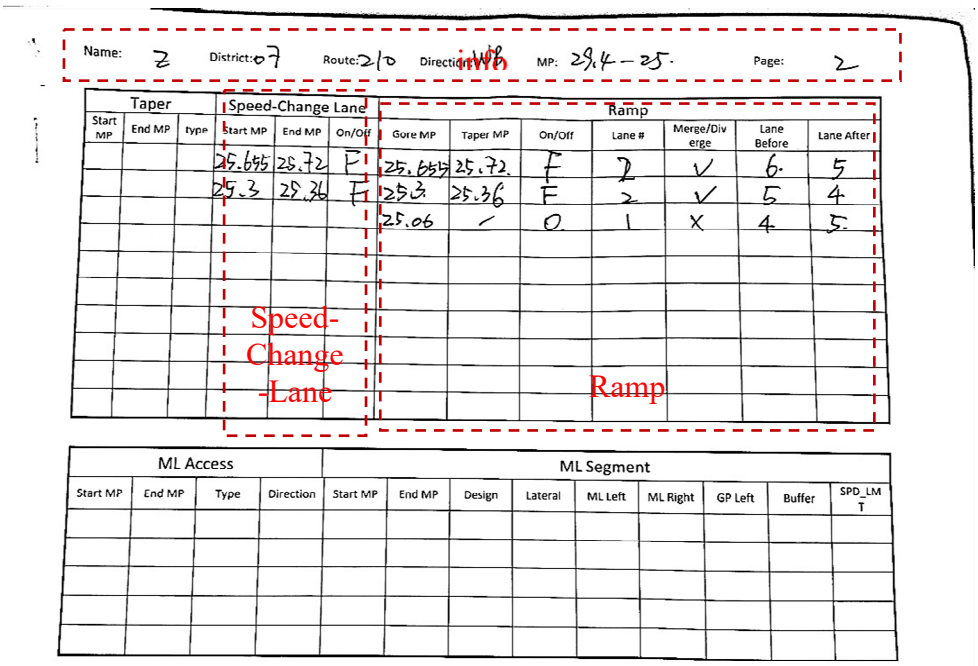
APPENDIX A

Manual Data Collection Example

Appendix A: Manual Data Collection Example

In this appendix, we explain how to create the input file with the collected data by showing an example corridor. When the engineers went through the satellite imagery and collected the geometric features of the corridor, a data collection form is required to be filled in. Then an input file of the corridor for the segment matching code will be created. Figure 1 shows the filled data collection form for the corridor from MP 29.4 to MP 25 on I-210 WB in District 7 in California.





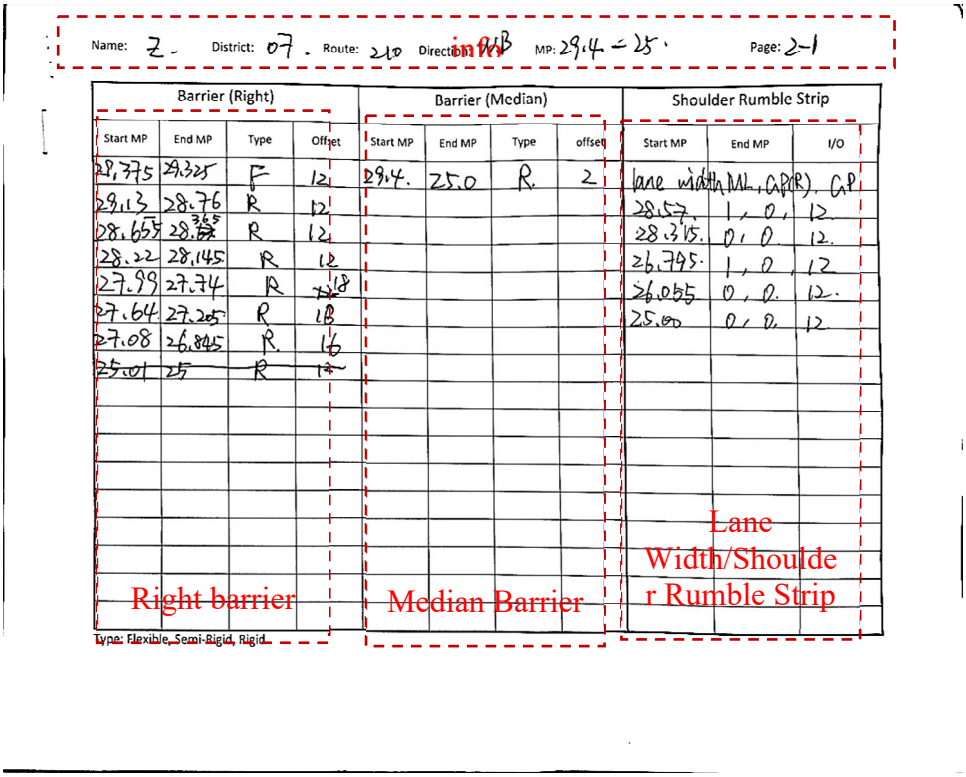


Figure 1 Example Data Collection Form

The input file is a *.xlsx file named by the district and route number that the corridor locates. There are ten spreadsheets in the input file, including *info, ML_ACCESS, ML_SEGMENT, CURVE, RAMP, SCL, TAPER, BARRIER_M, BARRIER_R, and LANE_WIDTH. Info*

The *info* spreadsheet provides the general information of the corridor, such as the start and end mileposts, speed limit of the corridor, the year when the ML design was adopted, the last year when the ML design was adopted (since we are using HSIS data from 2010 to 2014, the last year is always 2014 for CA), and ML restriction. The speed limit and restrictions are provided by Caltrans. The *info* spreadsheet is as follows:

Start_MP	29.4
End_MP	25
Speed_Limit	70
Begin_Year	2013
End_Year	2014
Restriction	24H

ML_SEGMENT

In the data collection form, the ML on the study corridor starts from MP 29.4 and ends at 25.0. The traffic on ML and GP lanes are concurrent (C) and separated by flush buffer (F), and the average width of flush buffer approximates 2 ft. Therefore, in the input file, the *ML_SEGMENT* spreadsheet is as follows:

ID	Start_MP	End_MP	DESIGN	LATERAL	ML_Left_Shoulder	ML_RIGHT_SHOULI	GP_Left_Shoulder	BUFFER_WIDTH	SPD_LIMIT
1	29.4	25	C	F	0	0	0	2	70

ML_ACCESS

According the records in the data collection form, there are two at-grade (Type A) accesses on the corridor: the first access is from MP 28.57 to MP 28.315, and the second one is from MP 26.795 to MP 26.055. Traffic can ingress and egress (Direction B) the ML. The *ML_ACCESS* spreadsheet is therefore written as:

ID	START_MP	END_MP	TYPE	DIRECTION
1	28.57	28.315	A	B
2	26.795	26.055	A	B

CURVE

The horizontal curves are measured in Google Map and recorded separately. On the example corridor, there are 2 horizontal curves: one is from MP 28.8 to MP 28.5 with the radius of 2800 ft, and the second one is from MP 29.3 to MP 28.9 with the radius of 3050 ft. The *CURVE* spreadsheet is as:

ID	START_MI	END_MP	RADIUS	LENGTH
1	28.5	28.8	2800	1584
2	28.9	29.3	3050	2112

RAMP

There are 13 ramps (both on-ramp and off-ramp) on the study corridor. The ramp AADT information is provided by HSIS Ramp data. However, in HSIS California data, the direction (Eastbound or Westbound) and type (on-ramp or off-ramp) of each ramp are not clarified. So we manually compared the observed ramps with the HSIS ramp data by milepost and identified the corresponding HSIS records. The *RAMP* spreadsheet is as follows:

ID	milepost	RMP_ADT	COUNTY	on_offrp	lane_num	merge	lane_before	lane_after
1	29.13	8500	19	O	1	no	5	6
2	28.76	6900	19	F	1	no	6	5
3	28.22	8400	19	O	1	yes	5	5
4	27.99	8400	19	O	1	no	5	6
5	27.735	7000	19	F	2	yes	6	5
6	27.205	6300	19	F	1	yes	5	5
7	26.75	15200	19	O	1	no	5	6
8	26.6	15800	19	F	2	yes	6	5
9	26.07	15200	19	O	1	no	5	6
10	25.825	7800	19	F	1	yes	6	6
11	25.655	55000	19	F	2	yes	6	5
12	25.3	14100	19	F	2	yes	5	4
13	25.06	10700	19	O	1	no	4	5

SCL

There are seven ramps (one on-ramp and six off-ramps) with both gore point and taper point records, where exists speed-change-lane. The *SCL* spreadsheet is as:

ID	START_MP	END_MP	TYPE
1	28.22	28.1	O
2	27.78	27.735	F
3	27.26	27.205	F
4	26.67	26.6	F
5	25.86	25.825	F
6	25.72	25.655	F
7	25.36	25.3	F

TAPER

Although there is no merging or shifting taper observed on the example corridor, there still exists a blank *TAPER* spreadsheet, which is as:

ID	START_MP	END_MP	TYPE

BARRIER_M

The example corridor is separated with the traffic of opposite direction by rigid barrier (Type R), which was observed on the entire corridor. Also, the offset of the median barrier constantly approximates 2 ft, so there is only one record on the *BARRIER_M* spreadsheet:

ID	start_mp	end_mp	type	offset
1	29.4	25	R	2

BARRIER_R

There are seven segments with right barrier on the example corridor, and the offsets of these right barriers range from 12 ft to 18 ft. One segment has flexible type right barrier (F), and the rest segment have rigid right barrier (R). Since the data collection participants observed the right barrier from the Google Earth satellite imagery, it may be difficult to identify the barrier type accurately. For example, some barriers resemble semi-rigid barrier from the satellite imagery of 2014, but in the street view, where only the records of 2018 or 2019 is available, they were built on top of a concrete base. In this case, the engineers would use their judgement to decide the barrier type. The *BARRIER_R* spreadsheet is as:

ID	start_mp	end_mp	type	offset
1	29.375	29.325	F	12
2	29.13	28.76	R	12
3	28.655	28.365	R	12
4	28.22	28.145	R	12
5	27.99	27.74	R	18
6	27.64	27.205	R	16
7	27.08	26.845	R	16

LANE_WIDTH

The width of ML (HOV) changes several times along the example corridor, but the width of GP lanes (both rightmost GP (GP) and the GP lanes next to the ML (GP_Width)) remains the same along the entire corridor. So, the *LANE_WIDTH* spreadsheet is as follows:

ID	MP	HOV	GP	GP_Width
1	29.4	1	0	12
2	28.57	1	0	12
3	28.315	0	0	12
4	26.795	1	0	12
5	26.055	0	0	12
6	25	0	0	12

APPENDIX B

Python Script for Data Retrieval, Quality Assessment and Integration

Appendix B: Python Script for Data Retrieval, Quality Assessment and Integration

Road_Segment: element in data processing and final output

```
class Road_Segment:
    def
__init__(self,district,route,direction,begmp,endmp,lane_number,ave_lane_width,area,city,ml_restriction,
\

inside_shoulder_width,inside_pave_shoulder,outside_shoulder_width,outside_pave_shoulder,\
            travel_width,bi_aadt,median_width,speed_lmt,county,ml_lane_num):
        self.district = district
        self.route = route
        self.begmp = begmp
        self.endmp = endmp

        self.length = abs(begmp-endmp)
        self.lane_number = lane_number
        self.ave_lane_width = ave_lane_width
        self.area = area
        self.city = city
        self.ml_restriction = ml_restriction
        self.inside_shoulder_width = inside_shoulder_width
        self.inside_pave_shoulder = inside_pave_shoulder
        self.outside_shoulder_width = outside_shoulder_width
        self.outside_pave_shoulder = outside_pave_shoulder
        self.travel_width = travel_width
        self.bi_aadt = bi_aadt
        self.median_width = median_width
        self.speed_lmt = speed_lmt
        self.county = county
        self.ml_lane_num = ml_lane_num
```

1. Width

```
def Widths(self,lane_width_list,taper_list,ramp_list):
    width_count = len(lane_width_list)
    found = False
    for i in range(width_count-1):
        start_mp = lane_width_list[i].mp
        end_mp = lane_width_list[i+1].mp
        if (self.begmp-self.endmp)*(start_mp-end_mp)<0:
            print('Lane Width Error')
            raw_input()
        else:
            if OVERLAP([start_mp,end_mp],[self.begmp,self.endmp]):
                found = True
                width_info = lane_width_list[i]
                break
    if not found:
        width_info = lane_width_list[-1]
    hov_factor,gp_factor1,gp_width1 = width_info.hov,width_info.gp,width_info.gp_width
    hov_width = gp_width1+hov_factor
    gp_width2 = gp_width1+gp_factor1
    gp_lane_number = self.lane_number
    travel_width_estimate = hov_width*self.ml_lane_num+gp_width1*(gp_lane_number-
2)+gp_width2
    return(hov_width,gp_width1,gp_width2)
```

2. Truck_Station

```
def Truck_Station(self):
        direction = ''
        if self.begmp<self.endmp:
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'E.xlsx'):
                direction = 'E'
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'N.xlsx'):
                direction = 'N'
        if self.begmp>self.endmp:
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'W.xlsx'):
                direction = 'W'
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'S.xlsx'):
                direction = 'S'
        if len(direction)==0:
            print 'no file founded',self.district,self.route,self.begmp,self.endmp
            return('Error')
        else:
            station_file =
'./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+direction+'.xlsx'
            Station_WB = openpyxl.load_workbook(station_file)
            Station_Sheet = Station_WB['Report Data']
            row_num = Station_Sheet.max_row
            HOV_Station_List = []
            station_distance = 1000
            truck_station = 'N/A'
            for i in range(row_num-1):
                month = Station_Sheet.cell(row = i+2,column = 1).value
                fwy = Station_Sheet.cell(row = i+2,column = 2).value
                CAPM = Station_Sheet.cell(row = i+2,column = 5).value

                VDS = Station_Sheet.cell(row = i+2,column = 8).value
                Name = Station_Sheet.cell(row = i+2,column = 9).value
```

```
        Type = Station_Sheet.cell(row = i+2,column = 11).value
        Daily = Station_Sheet.cell(row = i+2,column = 12).value
        K = Station_Sheet.cell(row = i+2,column = 13).value
        if not CAPM.isdigit():
            if '.' in CAPM:
                capm = re.findall("[+-]?\d+\.\d+",CAPM)

                if len(capm)==0:
                    capm = re.findall(".\d+",CAPM)
                capm = capm[0]
            else:
                 capm = re.sub("[^0-9]", "", CAPM)
        else:
            capm = CAPM
        PM = float(capm)
        if str(Type) =='Mainline' and abs(PM-self.endmp*0.5-0.5*self.begmp)<station_distance:
            station_distance = abs(PM-self.endmp*0.5-0.5*self.begmp)
            truck_station = str(VDS)
    return(truck_station)
```

3. Operation_HOV

```
def Operation_HOV(self):
        direction = ''
        if self.begmp<self.endmp:
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'E.xlsx'):
                direction = 'E'
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'N.xlsx'):
                direction = 'N'
        if self.begmp>self.endmp:
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'W.xlsx'):
                direction = 'W'
            if
os.path.exists('./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+'S.xlsx'):
                direction = 'S'
        if len(direction)==0:
            print 'no file founded',self.district,self.route,self.begmp,self.endmp
            return('Error')
        else:
            station_file =
'./Stations/All_Stations/Dis'+str(int(self.district))+'RT'+str(int(self.route))+direction+'.xlsx'
            Station_WB = openpyxl.load_workbook(station_file)
            Station_Sheet = Station_WB['Report Data']
            row_num = Station_Sheet.max_row
            HOV_Station_List = []
            station_distance = 1000
            truck_station = 'N/A'
            for i in range(row_num-1):
                month = Station_Sheet.cell(row = i+2,column = 1).value
                fwy = Station_Sheet.cell(row = i+2,column = 2).value
                CAPM = Station_Sheet.cell(row = i+2,column = 5).value
                VDS = Station_Sheet.cell(row = i+2,column = 8).value
                Name = Station_Sheet.cell(row = i+2,column = 9).value
```

```
            Type = Station_Sheet.cell(row = i+2,column = 11).value
            Daily = Station_Sheet.cell(row = i+2,column = 12).value
            K = Station_Sheet.cell(row = i+2,column = 13).value
            if not CAPM.isdigit():
               if '.' in CAPM:
                  capm = re.findall("[+-]?\d+\.\d+",CAPM)
                  if len(capm)==0:
                     capm = re.findall(".\d+",CAPM)
                  capm = capm[0]
               else:
                   capm = re.sub("[^0-9]", "", CAPM)
            else:
               capm = CAPM
            PM = float(capm)
            if int(self.district)==4:
               if str(Type) =='Mainline' and abs(PM-self.endmp*0.5-
0.5*self.begmp)<station_distance:

                  check_file = './Stations/Operation_w_Lane/'+'Operation_'+str(VDS)+'_w_Lane.xlsx'

                  if os.path.exists(check_file):
                     station_distance = abs(PM-self.endmp*0.5-0.5*self.begmp)
                     truck_station = str(VDS)
            else:
               if str(Type) == 'HOV' and abs(PM-self.endmp*0.5-0.5*self.begmp)<station_distance:
                  operation_hov = str(VDS)
                  station_distance = abs(PM-self.endmp*0.5-self.begmp*0.5)
         return(operation_hov)
```

4. MultiLane

```
def MultiLane(self,multilane_list):
    lane_number = 1
    for multi in multilane_list:
        check_dist = multi.district
        check_route = multi.route
        check_county = multi.county
        check_start = multi.start_mp
        check_end = multi.end_mp
        check_lane = multi.lane_num
        if int(self.district)==check_dist and int(self.route)==check_route and \
        ((self.begmp>=check_start and self.begmp<=check_end)or(self.begmp<=check_start and
self.begmp>=check_end)):
            lane_number = check_lane
            break
    return(lane_number)
```

5. Curve

```
def Curve(self,curve_list):
    found = False
    overlap_length = -1
    curve_length = -1
    radius = -1
    for curve in curve_list:
        segment1 = [curve.start_mp,curve.end_mp]
        segment2 = [self.begmp,self.endmp]
        if OVERLAP(segment1, segment2):
            if OVERLAP_LEN(segment1,segment2)>overlap_length:
                overlap_length = OVERLAP_LEN(segment1,segment2)
                curve_length = overlap_length*5280
                radius = curve.radius
                found = True
    if found:
      return(curve_length,radius)
    else:
      return('N/A','N/A')
```

6. Lane_Number

```
def Lane_Number(self,taper_list,ramp_list):
    found = False
    up_distance = 1000
    if self.begmp>self.endmp:
        for ramp in ramp_list:
            if ramp.mp>self.begmp and ramp.mp-self.begmp<up_distance:
                up_distance = ramp.mp-self.begmp
                lane_number = ramp.lane_after
                #print self.begmp,self.endmp,lane_number
                found = True
        if found and len(taper_list)>0:
            for taper in taper_list:
                if taper.start_mp>self.begmp and taper.start_mp-self.begmp<up_distance:
                    up_distance = taper.start_mp-self.begmp
                    if taper.taper_type=='D':
                        lane_number -=1
                    if taper.taper_type == 'A':
                        lane_number+=1
    else:
        for ramp in ramp_list:
            if ramp.mp<self.begmp and self.begmp-ramp.mp<up_distance:

                up_distance = self.begmp-ramp.mp
                lane_number = ramp.lane_after
                #print self.begmp,self.endmp,lane_number
                found = True
        if found and len(taper_list)>0:
            for taper in taper_list:
                #print taper.start_mp,taper.end_mp,taper.taper_type,self.begmp,up_distance
                #continue
                if taper.start_mp<self.begmp and self.begmp-taper.start_mp<up_distance:
                    up_distance = self.begmp-taper.start_mp
                    if taper.taper_type =='D':
```

```
                    lane_number-=1
                if taper.taper_type =='A':
                    lane_number+=1
        if found:
            return(lane_number)
        else:
            return('N/A')
```

7. Taper

```
def Taper(self,taper_list):
        found = False
        overlap_length = -1
        taper_length = -1
        for taper in taper_list:
            segment1 = [taper.start_mp,taper.end_mp]
            segment2 = [self.begmp,self.endmp]
            if OVERLAP(segment1,segment2):
                if OVERLAP_LEN(segment1,segment2)>overlap_length:
                    overlap_length = OVERLAP_LEN(segment1,segment2)
                    taper_start_mp = taper.start_mp
                    taper_end_mp = taper.end_mp
                    taper_type = taper.taper_type
                    found = True
        if found:
            return(taper_type)
        else:
            return('N/A')
```

8. SCL

```
def SCL(self,scl_list):
    found = False
    overlap_length = -1
    for scl in scl_list:
        segment1 = [scl.start_mp,scl.end_mp]
        segment2 = [self.begmp,self.endmp]
        if OVERLAP(segment1,segment2):
            if OVERLAP_LEN(segment1,segment2)>1e-6:
                overlap_length = OVERLAP_LEN(segment1,segment2)
                scl_start_mp = scl.start_mp
                scl_end_mp = scl.end_mp
                scl_type = scl.scl_type
                scl_function = scl.scl_function
                found = True
    if found:
        return(scl_start_mp,scl_end_mp,scl_type,scl_function)
    else:
        return('N/A','N/A','N/A','N/A')
```

9. Barrier

```
def Barrier(self,barrier_list):
    found = False
    overlap_length = -1
    for barrier in barrier_list:
        segment1 = [barrier.start_mp,barrier.end_mp]
        segment2 = [self.begmp,self.endmp]

        if OVERLAP(segment1,segment2):
            temp_length = OVERLAP_LEN(segment1,segment2)
            if temp_length>overlap_length:
                overlap_length = temp_length
                barrier_type = barrier.barrier_type
                barrier_offset = barrier.offset
                found = True
    if found:
        return(barrier_type,barrier_offset)
    else:
        return('N/A','N/A')
```

10. ML_Segment

```
def ML_Segment(self,ml_segment_list):
    found = False
    overlap_length = -1
    for ml_segment in ml_segment_list:
        segment1 = [ml_segment.start_mp,ml_segment.end_mp]
        segment2 = [self.begmp,self.endmp]
        temp_length = OVERLAP_LEN(segment1,segment2)
        if temp_length>overlap_length:
            overlap_length = temp_length
            ml_design_type = ml_segment.design_type
            ml_lateral_type = ml_segment.lateral_type
            ml_left_shoulder_width = ml_segment.left_shoulder_width
            ml_right_shoulder_width = ml_segment.right_shoulder_width
            ml_buffer_width = ml_segment.buffer_width
            found = True
    if found:

return(ml_design_type,ml_lateral_type,ml_left_shoulder_width,ml_right_shoulder_width,ml_buffer_wi
dth)
    else:
        return('N/A','N/A','N/A','N/A','N/A')
```

11. Up_Ramp

```
    def Up_Ramp(self,ramp_list):
        up_found = False
        up_distance = 10000
        up_aadt = 0
        if self.begmp>self.endmp:
            for ramp in ramp_list:
                if ramp.ramp_type=='O' and ramp.mp>self.begmp and ramp.mp-
self.begmp<up_distance:
                    up_distance = ramp.mp-self.begmp
                    up_aadt = ramp.aadt
                    up_found = True
            if up_found:
                return(up_distance,up_aadt)
            else:
                return('N/A','N/A')
        else:
            for ramp in ramp_list:
                if ramp.ramp_type=='O' and ramp.mp<self.begmp and self.begmp-
ramp.mp<up_distance:
                    up_distance = self.begmp-ramp.mp
                    up_aadt = ramp.aadt
                    up_found = True
            if up_found:
                return(up_distance,up_aadt)
            else:
                return('N/A','N/A')
```

12. Down_Ramp

```
        down_found = False
        down_distance = 10000
        down_aadt = 0
        if self.begmp>self.endmp:
            for ramp in ramp_list:
                if ramp.ramp_type=='F' and ramp.mp<self.endmp and self.endmp-
ramp.mp<down_distance:
                    down_distance = self.endmp-ramp.mp
                    down_aadt = ramp.aadt
                    down_found = True
            if down_found:
                return(down_distance,down_aadt)
            else:
                return('N/A','N/A')
        else:
            for ramp in ramp_list:
                if ramp.ramp_type =='F' and ramp.mp>self.endmp and ramp.mp-
self.endmp<down_distance:
                    down_distance = ramp.mp-self.endmp
                    down_aadt = ramp.aadt
                    down_found = True
            if down_found:
                return(down_distance,down_aadt)
            else:
                return('N/A','N/A')
```

13. ML_ACCESS_Up

```
def ML_ACCESS_Up(self,access_list,ramp_list):
    up_found = False
    up_distance = 10000
    if self.begmp>self.endmp:
        for access in access_list:
            if OVERLAP([self.begmp,self.endmp],[access.start_mp,access.end_mp]):
                if access.direction=='B' or access.direction=='T':
                    up_distance = 0
                    up_found = True
                    up_access_type = access.access_type
                    up_access_direction = access.direction
                    up_access_length = abs(access.start_mp-access.end_mp)
                    if up_access_length==0:
                        up_access_length =99999
                    if access.access_type!='C':

                        up_access_ramp_distance,up_access_ramp_aadt = access.Up_Ramp(ramp_list)


                    else:
                        up_access_length =99999
                        up_access_ramp_distance,up_access_ramp_aadt = self.Up_Ramp(ramp_list)
                        up_access_ramp_distance=0
            else:
                if (access.direction=='B' or access.direction=='T')and\
                    access.end_mp>self.begmp and access.end_mp-self.begmp<up_distance:
                    up_distance = access.end_mp-self.begmp
                    up_found = True
                    up_access_type = access.access_type
                    up_access_direction = access.direction

                    up_access_length = abs(access.start_mp-access.end_mp)
                    if up_access_length==0:
                        up_access_length =99999
```

```
            if access.access_type!='C':
                up_access_ramp_distance,up_access_ramp_aadt = access.Up_Ramp(ramp_list)

            else:
                up_access_length =99999
                up_access_ramp_distance,up_access_ramp_aadt = self.Up_Ramp(ramp_list)
                up_access_ramp_distance=0

    else:
        for access in access_list:
            if OVERLAP([self.begmp,self.endmp],[access.start_mp,access.end_mp]):
                if access.direction=='B' or access.direction=='T':
                    up_distance = 0
                    up_found = True
                    up_access_type = access.access_type
                    up_access_direction = access.direction
                    up_access_length = abs(access.start_mp-access.end_mp)
                    if up_access_length==0:
                        up_access_length =99999
                    if access.access_type!='C':

                        up_access_ramp_distance,up_access_ramp_aadt = access.Up_Ramp(ramp_list)

                    else:
                        up_access_length =99999
                        up_access_ramp_distance,up_access_ramp_aadt = self.Up_Ramp(ramp_list)
                        up_access_ramp_distance=0
                else:
                    if (access.direction=='B' or access.direction=='T')and\
                        access.end_mp<self.begmp and -access.end_mp+self.begmp<up_distance:
                        up_distance = -access.end_mp+self.begmp
                        up_found = True
                        up_access_type = access.access_type
                        up_access_direction = access.direction
```

```
                up_access_length = abs(access.start_mp-access.end_mp)
                if up_access_length==0:
                    up_access_length =99999
                if access.access_type!='C':
                    up_access_ramp_distance,up_access_ramp_aadt = access.Up_Ramp(ramp_list)

                else:
                    up_access_length =99999
                    up_access_ramp_distance,up_access_ramp_aadt = self.Up_Ramp(ramp_list)
                    up_access_ramp_distance=0

        if up_found:


return(up_distance,up_access_type,up_access_length,up_access_direction,up_access_ramp_distance,up_
access_ramp_aadt)
        else:
            return('N/A','N/A','N/A','N/A','N/A','N/A')
```

14. ML_ACCESS_Down

```
    def ML_ACCESS_Down(self,access_list,ramp_list):
        down_found = False
        down_distance = 10000

        if self.begmp>self.endmp:
            for access in access_list:
                #print access.start_mp,access.end_mp,self.begmp,self.endmp
                if OVERLAP([self.begmp,self.endmp],[access.start_mp,access.end_mp]):
                    #print 'downstream',access.direction
                    if access.direction=='B' or access.direction=='X':
                        down_distance = 0
                        down_found = True
                        down_access_type = access.access_type
                        down_access_direction = access.direction
                        down_access_length = abs(access.start_mp-access.end_mp)
                        if down_access_length ==0:
                            down_access_length=99999
                        if access.access_type!='C':
                            down_access_ramp_distance,down_access_ramp_aadt =
access.Down_Ramp(ramp_list)


                        else:
                            down_access_length = 99999
                            down_access_ramp_distance,down_access_ramp_aadt = self.Down_Ramp(ramp_list)
                            down_access_ramp_distance=0
                else:
                    if (access.direction=='B' or access.direction=='X') and\
                        access.start_mp<self.endmp and self.endmp-access.start_mp<down_distance:
                        ######################revision 3
                        down_distance = self.endmp-access.start_mp
                        down_found = True
                        down_access_type = access.access_type
                        down_access_direction = access.direction
                        down_access_length = abs(access.start_mp-access.end_mp)
```

```
                if down_access_length ==0:
                    down_access_length=99999
                if access.access_type!='C':
                    down_access_ramp_distance,down_access_ramp_aadt =
access.Down_Ramp(ramp_list)

                else:
                    down_access_length = 99999
                    down_access_ramp_distance,down_access_ramp_aadt = self.Down_Ramp(ramp_list)
                    down_access_ramp_distance=0

        else:
            for access in access_list:
                if OVERLAP([self.begmp,self.endmp],[access.start_mp,access.end_mp]):
                    if access.direction=='B' or access.direction=='X':
                        down_distance = 0
                        down_found = True
                        down_access_type = access.access_type
                        down_access_direction = access.direction
                        down_access_length = abs(access.start_mp-access.end_mp)
                        if down_access_length ==0:
                            down_access_length=99999
                        if access.access_type!='C':
                            down_access_ramp_distance,down_access_ramp_aadt =
access.Down_Ramp(ramp_list)

                        else:
                            down_access_length = 99999
                            down_access_ramp_distance,down_access_ramp_aadt = self.Down_Ramp(ramp_list)
                            down_access_ramp_distance=0
                    else:
                        if (access.direction=='B' or access.direction=='X') and\
                            access.start_mp>self.endmp and -self.endmp+access.start_mp<down_distance:
                            ######################revision 3
                            down_distance = -self.endmp+access.start_mp
```

```
                down_found = True
                down_access_type = access.access_type
                down_access_direction = access.direction
                down_access_length = abs(access.start_mp-access.end_mp)
                if down_access_length ==0:
                    down_access_length=99999
                if access.access_type!='C':
                    down_access_ramp_distance,down_access_ramp_aadt =
access.Down_Ramp(ramp_list)

                else:
                    down_access_length = 99999
                    down_access_ramp_distance,down_access_ramp_aadt = self.Down_Ramp(ramp_list)
                    down_access_ramp_distance=0


    if down_found:


return(down_distance,down_access_type,down_access_direction,down_access_ramp_distance,down_ac
cess_ramp_aadt)
    else:
        return('N/A','N/A','N/A','N/A','N/A')
```

15. Weaving

```
    def Weaving(self,ramp_list):
        up_found = False
        down_found = False
        up_distance = 10000
        down_distance = 10000
        if self.begmp>self.endmp:
            for ramp in ramp_list:
                if ramp.mp>=self.begmp and (ramp.mp-self.begmp)<up_distance and
ramp.ramp_type=='O':
                    up_distance = ramp.mp-self.begmp
                    up_found = True
                    up_ramp = ramp
                    up_ramp_aadt = ramp.aadt
                if ramp.mp<=self.endmp and (self.endmp-ramp.mp)<down_distance and
ramp.ramp_type=='F':
                    down_distance = self.endmp-ramp.mp
                    down_found = True
                    down_ramp = ramp
                    down_ramp_aadt = ramp.aadt
        else:
            for ramp in ramp_list:
                if ramp.mp<=self.begmp and (self.begmp-ramp.mp)<up_distance and
ramp.ramp_type=='O':
                    up_distance = self.begmp-ramp.mp
                    up_found = True
                    up_ramp = ramp
                    up_ramp_aadt = ramp.aadt
                if ramp.mp>=self.endmp and (ramp.mp-self.endmp)<down_distance and
ramp.ramp_type =='F':
                    down_distance = ramp.mp-self.endmp
                    down_found = True
                    down_ramp = ramp
                    down_ramp_aadt = ramp.aadt
        if up_found and down_found and up_ramp.ramp_type =='O' \
            and down_ramp.ramp_type=='F' and abs(up_ramp.mp-down_ramp.mp)<0.85:
```

```
            inbound_on = up_ramp.ramp_lane_num
            if up_ramp.merge=='yes':
                outbound_on = inbound_on
            else:
                outbound_on = inbound_on+1
            outbound_off = down_ramp.ramp_lane_num
            if down_ramp.merge =='yes':
                inbound_off = outbound_off
            else:
                inbound_off = outbound_off+1
            inbound = abs(inbound_on-inbound_off)
            outbound = abs(outbound_on-outbound_off)
            if min(inbound,outbound)==0:
                if (inbound+outbound)<=1:
                    weave_type = 'B'
                else:
                   weave_type = 'C'
            else:
                weave_type = 'A'


            return('True',weave_type,abs(up_ramp.mp-
down_ramp.mp),up_distance,down_distance,up_ramp.aadt,down_ramp_aadt)


        else:
            if not up_found:
                up_distance='N/A'
                up_ramp_aadt = 'N/A'
            if not down_found:
                down_distance = 'N/A'
                down_ramp_aadt = 'N/A'
            return('N/A','N/A','N/A',up_distance,down_distance,up_ramp_aadt,down_ramp_aadt)
```

Feature Class

```
class CURVATURE:
    def __init__(self,start_mp,end_mp,length,radius):
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.length = length
        self.radius = radius
class ML_SEGMENT:
    def __init__(self,start_mp,end_mp,design_type,lateral_type,spd_limit,\
                left_shoulder_width,right_shoulder_width,gp_left_shoulder,buffer_width):
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.design_type = design_type
        self.lateral_type = lateral_type
        self.spd_limit = spd_limit
        self.right_shoulder_width = right_shoulder_width
        self.left_shoulder_width = left_shoulder_width
        self.gp_left_shoulder = gp_left_shoulder
        self.buffer_width = buffer_width
class MULTILANE:
    def __init__(self,district,route,county,start_mp,end_mp,lane_num):
        self.district = district
        self.route = route
        self.county = county
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.lane_num = lane_num
class ML_ACCESS:
    def __init__(self,start_mp,end_mp,access_type,direction):
        self.start_mp=start_mp
        self.end_mp = end_mp
        self.access_type = access_type
        self.direction = direction
    def Up_Ramp(self,ramp_list):
```

```
        found = False
        ramp_distance = 10000
        if self.start_mp<self.end_mp:
            for ramp in ramp_list:
                if ramp.ramp_type=='O' and ramp.mp<=self.start_mp and (self.start_mp-
ramp.mp)<ramp_distance:
                    found = True
                    ramp_distance = self.start_mp-ramp.mp
                    ramp_aadt = ramp.aadt
            if found:
                return(ramp_distance,ramp_aadt)
            else:
                return('N/A','N/A')
        else:
            for ramp in ramp_list:
                if ramp.ramp_type=='O' and ramp.mp>=self.start_mp and (ramp.mp-
self.start_mp)<ramp_distance:
                    found = True
                    ramp_distance = ramp.mp-self.start_mp
                    ramp_aadt = ramp.aadt
            if found:
                return(ramp_distance,ramp_aadt)
            else:
                return('N/A','N/A')
    def Down_Ramp(self,ramp_list):
        found = False
        ramp_distance = 10000
        if self.start_mp<self.end_mp:
            for ramp in ramp_list:
                #print ramp.ramp_type,ramp.mp
                if ramp.ramp_type=='F' and ramp.mp>self.end_mp and (ramp.mp-
self.end_mp)<ramp_distance:
                    found = True
                    ramp_distance = ramp.mp-self.end_mp
                    ramp_aadt = ramp.aadt
```

```
            if found:
                return(ramp_distance,ramp_aadt)
            else:
                return('N/A','N/A')
        else:
            for ramp in ramp_list:
                if ramp.ramp_type =='F' and ramp.mp<self.end_mp and (self.end_mp-
ramp.mp)<ramp_distance:
                    found = True
                    ramp_distance = self.end_mp-ramp.mp
                    ramp_aadt = ramp.aadt
            if found:
                return(ramp_distance,ramp_aadt)
            else:
                return('N/A','N/A')
class LOOP_RECORD:
    def __init__(self,loop_mp,aadt_value,lane_width):
        self.loop_mp = loop_mp
        self.aadt_value = aadt_value
        self.lane_width = lane_width
class GP_RAMP:
    def __init__(self,mp,aadt,ramp_type,ramp_lane_num,merge,lane_before,lane_after):
        self.mp = mp
        self.aadt = aadt
        self.ramp_type = ramp_type # on /off
        self.ramp_lane_num = ramp_lane_num # number of lanes on the ramp
        self.merge= merge
        self.lane_before = lane_before
        self.lane_after = lane_after# if the lanes merge or split immediately
class BARRIER:
    def __init__(self,start_mp,end_mp,barrier_type,offset):
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.barrier_type = barrier_type
        self.offset = offset
```

```
class TAPER:
    def __init__(self,start_mp,end_mp,taper_type):
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.taper_type = taper_type
class SPEED_CHANGE_LANE:
    def __init__(self,start_mp,end_mp,scl_type,scl_function):
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.scl_type = scl_type
        self.scl_function = scl_function
class LANE_WIDTH:
    def __init__(self,mp,hov,gp,gp_width):
        self.mp = mp
        self.hov = hov
        self.gp = gp
        self.gp_width = gp_width
def STATUS_OUTPUT(error_output,text):
    print >> error_output,text
    print ("##########################",text,"################################")
def OVERLAP(seg1,seg2):
    start1 = seg1[0]
    start2 = seg2[0]
    end1 = seg1[1]
    end2 = seg2[1]
    overlap = True

    points = [start1,start2,end1,end2]
    points.sort()
    if max(points.index(start1),points.index(end1))<min(points.index(start2),points.index(end2))
or\
     max(points.index(start2),points.index(end2))<min(points.index(start1),points.index(end1)):
         overlap = False
    return(overlap)
def OVERLAP_LEN(seg1,seg2):
```

```
    if not OVERLAP(seg1,seg2):
        return(0)
    else:
        start1 = seg1[0]
        start2 = seg2[0]
        end1 = seg1[1]
        end2 = seg2[1]
        points = seg1+seg2
        points.sort()
        return(abs(points[2]-points[1]))

def OVERLAP_RATE(seg1,seg2):
    if not OVERLAP(seg1,seg2):
        return(0)
    else:
        if abs(seg1[0]-seg1[1])<1e-6:
            return(1)
        else:
            rate = OVERLAP_LEN(seg1,seg2)/abs(seg1[0]-seg1[1])
            return(rate)
def OVERLAP_SEG(seg1,seg2):
    points = seg1+seg2
    if seg1[0]<seg1[1]:
        points.sort()
    else:
        points.sort(reverse=True)
    return(points[1],points[2])

def OVERLAP_CHECKING(SEGMENT_LIST):
    found_overlap = False

    for i in range(len(SEGMENT_LIST)-1):
        for j in range(len(SEGMENT_LIST)-i-1):
            #print i,j
```

```
            seg1_mp1 = SEGMENT_LIST[i].begmp
            seg1_mp2 = SEGMENT_LIST[i].endmp
            seg2_mp1 = SEGMENT_LIST[i+j+1].begmp
            seg2_mp2 = SEGMENT_LIST[i+j+1].endmp
            if OVERLAP_LEN([seg1_mp1,seg1_mp2],[seg2_mp1,seg2_mp2])>1e-4:
                #print >> overlap_output,input_file,seg1_mp1,seg1_mp2,seg2_mp1,seg2_mp2
                #print >> overlap_output,input_file
                found_overlap = True
                break
        if found_overlap:
                break
    return(found_overlap)
def OLD_CHECK_STATISTICS(segment_list,feature,feature_list):
    segment1 = segment_list[0]
    func = getattr(segment1,feature)
    temp = func(feature_list)
    var_count = len(temp)
    observe = [[] for x in range(var_count)]
    for segment in segment_list:
        func = getattr(segment,feature)
        temp = func(feature_list)
        if temp[0]!='N/A':
            for i in range(var_count):
                item = temp[i]
                #print type(item)
                observe[i].append(item)
    for i in range(var_count):
        code = False
        for j in range(len(observe[i])):
            if type(observe[i][j])!='float' and type(observe[i][j])!='int':
                code = True
                break
        if code:
            print observe[i],'unicode'
```

```
        else:
            print observe[i]
            print sum(observe[i])/len(observe[i])
    return

def BETWEEN(segment,point):
    point1 = segment[0]
    point2 = segment[1]
    if (point1<point and point<point2) or (point2<point and point<point1):
        return True
    else:
        return False
   # raw_input()
def Check_Statistics(file_name):
    check_wb = openpyxl.load_workbook(file_name)
    sheet_names = ['Normal','SCL_On','SCL_Off']
    for name in sheet_names:
        check_sheet = check_wb[name]
        feature_number = check_sheet.max_column
        print feature_number
        observe_number = check_sheet.max_row-1
        for i in range(feature_number):
            feature =str(check_sheet.cell(row =1, column = i+1).value)
            print feature
            vars()[feature] = [" for x in range(observe_number)]
            for j in range(observe_number):

                vars()[feature][j] = check_sheet.cell(row=j+2,column=i+1).value
            vars()[feature] = [x for x in vars()[feature] if x!='N/A' ]
            print vars()[feature]
            #raw_input()
    return
```

```
def CUT_SCL(segment_list,scl_list):
    Finish = False
    while not Finish:
        Finish = True
        for segment in segment_list:
            for scl in scl_list:
                seg = [segment.begmp,segment.endmp]
                point1 = scl.start_mp
                point2 = scl.end_mp
                if BETWEEN(seg,point1) and BETWEEN(seg,point2):
                    if segment.begmp>segment.endmp:

                        new_seg1 = copy.deepcopy(segment)
                        new_seg1.begmp = segment.begmp
                        new_seg1.endmp = max(point1,point2)
                        new_seg2 = copy.deepcopy(segment)
                        new_seg2.begmp = max(point1,point2)
                        new_seg2.endmp = min(point1,point2)
                        new_seg3 = copy.deepcopy(segment)
                        new_seg3.begmp = min(point1,point2)
                        new_seg3.endmp = segment.endmp
                        #print 'processed',segment.begmp,segment.endmp,point1,point2
                        segment_list.remove(segment)
                        segment_list.append(new_seg1)
                        segment_list.append(new_seg2)
                        segment_list.append(new_seg3)
                        Finish = False
                        #print len(segment_list)
                        #raw_input()
                        break
                    else:
                        new_seg1,new_seg2,new_seg3 =
copy.deepcopy(segment),copy.deepcopy(segment),copy.deepcopy(segment)
                        new_seg1.begmp = segment.begmp
                        new_seg1.endmp = min(point1,point2)
```

```
            new_seg2.begmp = new_seg1.endmp
            new_seg2.endmp = max(point1,point2)
            new_seg3.begmp = new_seg2.endmp
            new_seg3.endmp = segment.endmp
            #print 'processed',segment.begmp,segment.endmp,point1,point2
            segment_list.remove(segment)
            segment_list.append(new_seg1)
            segment_list.append(new_seg2)
            segment_list.append(new_seg3)
            Finish = False
            #print len(segment_list)
            break

        else:
            if BETWEEN(seg,point1) and not BETWEEN(seg,point2):
                new_seg1,new_seg2 = copy.deepcopy(segment),copy.deepcopy(segment)
                new_seg1.begmp = segment.begmp
                new_seg1.endmp = point1
                new_seg2.begmp = point1
                new_seg2.endmp = segment.endmp
                #print 'processed',segment.begmp,segment.endmp,point1,point2
                segment_list.remove(segment)
                segment_list.append(new_seg1)
                segment_list.append(new_seg2)
                Finish = False
                #print len(segment_list)
                break
            else:
                if not BETWEEN(seg,point1) and BETWEEN(seg,point2):
                    new_seg1,new_seg2 = copy.deepcopy(segment),copy.deepcopy(segment)
                    new_seg1.begmp = segment.begmp
                    new_seg1.endmp = point2
                    new_seg2.begmp = point2
                    new_seg2.endmp = segment.endmp
```

```
                    #print 'processed',segment.begmp,segment.endmp,point1,point2
                    segment_list.remove(segment)
                    segment_list.append(new_seg1)
                    segment_list.append(new_seg2)
                    Finish = False
                    #print len(segment_list)
                    break
    return(segment_list)

import copy
import openpyxl
```

CODE_Data_Process_Segment_Based

```
import openpyxl
from Feature_Class import *
from CLASS_DEFINE import *
import os
#from pathlib import Path
FINISHED_FILE_LIST = []
monitor_file = open('./file_status.txt','r')
for line in monitor_file.readlines():
   FINISHED_FILE_LIST.append(line[:-1])
monitor_file.close()


error_output = open('ERROR_OVERLAP.txt','w')
filelist = os.listdir('./Raw_Data_III/')
file_summary = open('./File_Summary.txt','w')
#print filelist
multilane_input = open('Multilane_Record.txt','r')
Multilane_List = []
for line in multilane_input.readlines():
   temp = [str(x) for x in line.split()]
   check_dist,check_route,check_county = int(temp[0]),int(temp[1]),int(temp[2])
   check_start,check_end = float(temp[3]),float(temp[4])
   check_lane = int(temp[5])
   new_multi =
MULTILANE(check_dist,check_route,check_county,check_start,check_end,check_lane)
   Multilane_List.append(new_multi)
multilane_input.close()
#for item in Multilane_List:
#    print item.district,item.route,item.county,item.start_mp,item.end_mp,item.lane_num
#raw_input()
for input_file in filelist:
   if input_file.endswith('.xlsx'):#
      if input_file in FINISHED_FILE_LIST:
         print input_file
         continue
```

```
    if input_file!="DIS12_RT91_SEG2.xlsx":
        continue# and input_file!='DIS04_RT04_SEG1.xlsx':
    #if not (input_file.startswith('DIS12')): #or input_file.startswith('DIS11')) :#or input_file in
FINISHED_FILE_LIST:
    #    continue
    #print input_file
    monitor_file = open('./file_status.txt','a+')
    #continue
    name1,name2,name3 = [x for x in input_file.split('_')]
    district = name1[3:]
    route_num = name2[2:]
    seg_id = int(name3[3:4])
    local_input_file = './Raw_Data_III/'+input_file
    INPUT_WB = openpyxl.load_workbook(local_input_file,data_only = True)
    Info_Sheet = INPUT_WB['Info']

    MP_start = float(Info_Sheet.cell(row = 1,column = 2).value)
    MP_end = float(Info_Sheet.cell(row = 2, column = 2).value)
    road_spd_limit = int(Info_Sheet.cell(row = 3, column = 2).value)
    YEAR1 = Info_Sheet.cell(row = 4, column = 2).value
    YEAR2 = Info_Sheet.cell(row = 5, column = 2).value
    ml_restrict = Info_Sheet.cell(row = 6,column = 2).value
    LOCAL_YEARS = []
    for local_year in range(4):
        if local_year+2010>=YEAR1 and local_year+2010<=YEAR2:
            LOCAL_YEARS.append(local_year+2010)

    #print Info_Sheet.cell(row =7, column =1).value

    if Info_Sheet.cell(row =7, column =1).value!=None:
        county_flag = Info_Sheet.cell(row = 7, column = 1).value
        county=Info_Sheet.cell(row = 7,column = 2).value
    else:
        county=0
    #print county
```

```
#print county_flag,len(county_flag),county
#raw_input()
if MP_start>MP_end:
   road_direction = 'S'
else:
   road_direction = 'N'
FILE_ID = 'DIS'+district+'_RT'+route_num+'_RD'+str(seg_id)
print district,route_num,seg_id,MP_start,MP_end
print >> file_summary,district,route_num,seg_id,MP_start,MP_end,county
#continue
##########################initialize variables#######################

############################### read-in values from road file#########################
HSIS_Seg = openpyxl.load_workbook("./HSIS_CA/Processed_ca_road.xlsx",data_only = True)
names = HSIS_Seg.sheetnames
sheet = HSIS_Seg[names[0]]
row_count = sheet.max_row
column_count = sheet.max_column
FEATURES = []
SEGMENT_LIST = []
for i in range(column_count):
   feature = sheet.cell(row = 1,column = i+1).value
   #print feature
   FEATURES.append(feature)

seg_count=0
for i in range(row_count-1):
   for j in range(column_count):
      feature = sheet.cell(row = 1, column = j+1).value
      vars()[feature] = sheet.cell(row = i+2,column = j+1).value
   LOCAL_AADT = []
   #print i, HWY_GRP, trktot,cntyrte,NO_LANES,AADT,district,route_num,BEGMP,ENDMP
   if int(AADT)>0:
      LOCAL_AADT.append(int(AADT))
```

```
        for local_year in LOCAL_YEARS:
            if int(vars()['aadt'+str(local_year)])!=0 and
int(vars()['aadt'+str(local_year)])<2*LOCAL_AADT[0]\
            and int(vars()['aadt'+str(local_year)])>=0.5*LOCAL_AADT[0]:
                LOCAL_AADT.append(int(vars()['aadt'+str(local_year)]))
        #print LOCAL_AADT
        AADT = sum(LOCAL_AADT)/len(LOCAL_AADT)
        if int(DISTRICT)==int(district) and int(RTE_NBR) == int(route_num) and
OVERLAP([BEGMP,ENDMP],[MP_start,MP_end])\
            and OVERLAP_RATE([BEGMP,ENDMP],[MP_start,MP_end])>=0.5 and
(int(county)==int(COUNTY) or county==0)\
            and (not RTE_SUF=='S') and (not RTE_SUF=='U'):
                if road_direction=='N' or road_direction =='E':
                    new_segment
=Road_Segment(district,route_num,road_direction,BEGMP,ENDMP,no_lane1,SURF_WID,RURURB,
CITY,ml_restrict,\

LSHLDWID,PAV_WDL,PAV_WIDR,RSHLDWID,SURF_WID,AADT,\
                                    MEDWID,road_spd_limit,COUNTY,1)
                if road_direction=='S' or road_direction == 'W':
                    new_segment =
Road_Segment(district,route_num,road_direction,ENDMP,BEGMP,no_lane2,SURF_WD2,RURURB,C
ITY,ml_restrict,\

LSHL_WD2,PAV_WDL2,PAV_WDR2,RSHL_WD2,SURF_WD2,AADT,\
                                    MEDWID,road_spd_limit,COUNTY,1)
                new_segment.ml_lane_num = new_segment.MultiLane(Multilane_List)
                #print new_segment.begmp, new_segment.endmp
                SEGMENT_LIST.append(new_segment)
                #print 'found new segment',seg_count,new_segment.begmp,new_segment.endmp
                seg_count+=1
            #print (seg_count,BEGMP[i],ENDMP[i])
    #x = input('Segment Check. If OK, press any key to proceed...')
    STATUS_OUTPUT(error_output,input_file)
    STATUS_OUTPUT(error_output,'SEGMENT CREATED')
    STATUS_OUTPUT(error_output,'Number of Segments:'+str(len(SEGMENT_LIST)))
    if len(SEGMENT_LIST)==0:
```

```
            continue
        #continue
        ########################################Check
overlapping#####################################
        if OVERLAP_CHECKING(SEGMENT_LIST):
            print >> error_output,input_file
            print input_file,"overlap found"
            continue
        #continue

#############################################################################################



        #raw_input()
        ########################load input data#######################################


        ################################Speed-Change
Lane#################################
        SCL_Sheet = INPUT_WB['SCL']
        scl_count = SCL_Sheet.max_row-1
        SCL_List = []
        if scl_count>0:
            for i in range(scl_count):


                start_mp,end_mp,scl_type,scl_function = [SCL_Sheet.cell(row = i+2,column = x+2).value for
x in range(4)]
                if scl_function=='G':

SCL_List.append(SPEED_CHANGE_LANE(float(start_mp),float(end_mp),scl_type,scl_function))
        SEGMENT_LIST = CUT_SCL(SEGMENT_LIST,SCL_List)
        #CHECK_STATISTICS(SEGMENT_LIST,'SCL',SCL_List)
        STATUS_OUTPUT(error_output,"Speed Change Lane Done")
        #raw_input()


        ########################Horizontal Alignment#################################
        Curve_Sheet = INPUT_WB['CURVE']
```

```
    Curve_List = []
    curve_count = Curve_Sheet.max_row-1
    if curve_count>0:
        for i in range(curve_count):
            start_mp, end_mp,radius,length = [Curve_Sheet.cell(row =i+2,column = x+2).value for x in
range(4)]
            Curve_List.append(CURVATURE(start_mp,end_mp,length,radius))



    STATUS_OUTPUT(error_output,'Horizontal Alignment Input Done')
    #x = input('press to continue')

########################################Taper#########################################
    Taper_Sheet = INPUT_WB['TAPER']
    taper_count= Taper_Sheet.max_row-1
    Taper_List = []
    if taper_count>0:
        for i in range(taper_count):
            start_mp,end_mp,taper_type = [Taper_Sheet.cell(row = i+2,column = x+2).value for x in
range(3)]
            if start_mp!=None:
                Taper_List.append(TAPER(start_mp,end_mp,taper_type))
            #print(start_mp,end_mp,taper_type  )
    #print(len(Taper_List),'taper length')
    #continue
    STATUS_OUTPUT(error_output,"Taper Done")
    #raw_input('Taper')

##############################Outside_Barrier#########################################
    RBarrier_Sheet = INPUT_WB['BARRIER_R']


    RBarrier_List = []
    right_barrier_count = RBarrier_Sheet.max_row-1
    if right_barrier_count>0:
        for i in range(right_barrier_count):
```

```
            start_mp,end_mp,barrier_type,barrier_offset = \
            [RBarrier_Sheet.cell(row = i+2,column = x+2).value for x in range(4)]
            RBarrier_List.append(BARRIER(start_mp,end_mp,barrier_type,barrier_offset))



    STATUS_OUTPUT(error_output,'Right Barrier')
    #raw_input('Right Barrier')
    ####################################Median Barrier#################################
    MBarrier_Sheet = INPUT_WB['BARRIER_M']

    MBarrier_List = []
    median_barrier_count = MBarrier_Sheet.max_row-1
    if median_barrier_count>0:
       for i in range(median_barrier_count):
          start_mp,end_mp,barrier_type,barrier_offset = \
          [MBarrier_Sheet.cell(row = i+2,column = x+2).value for x in range(4)]
          MBarrier_List.append(BARRIER(start_mp,end_mp,barrier_type,barrier_offset))

    #raw_input('Median Barrier')
    ########################### ML Information ##########################



    ML_Sheet = INPUT_WB['ML_SEGMENT']
    ML_Segment_List = []
    lateral_factor = False
    ml_count = ML_Sheet.max_row-1
    if ml_count>0:
       for i in range(ml_count):

start_mp,end_mp,design_type,lateral_type,left_shoulder,shoulder_width,gp_left_shoulder,buffer_width,
spd_lmt = \
            [ML_Sheet.cell(row = i+2, column = x+2).value for x in range(9)]
          if start_mp==None:
```

```
            continue

ML_Segment_List.append(ML_SEGMENT(start_mp,end_mp,design_type,lateral_type,spd_lmt,left_sho
ulder,shoulder_width,gp_left_shoulder,buffer_width))


    #raw_input('ML_SEGMENT')



    STATUS_OUTPUT(error_output,'Manage Lane Type Input Done')
    #x = input('press to continue')
    ################################ Managed Lane Access
##########################################
    Access_Sheet = INPUT_WB['ML_ACCESS']
    Access_List = []
    access_count = Access_Sheet.max_row-1
    if access_count>0:
       for i in range(access_count):
          start_mp,end_mp,access_type,direction = \
             [Access_Sheet.cell(row=i+2,column = x+2).value for x in range(4)]
          #print start_mp,end_mp,access_type,direction
          if start_mp==None:
             continue
          Access_List.append(ML_ACCESS(float(start_mp),float(end_mp),access_type,direction))
    STATUS_OUTPUT(error_output,'ACCESS Done')
    ######################################Ramp info
##########################################################
    Ramp_Sheet = INPUT_WB['RAMP']
    ramp_count = Ramp_Sheet.max_row-1
    #print (ramp_count)
    rp_column_count = Ramp_Sheet.max_column


    Ramp_List = []


    for i in range(ramp_count):
       for j in range(rp_column_count):
```

```
            feature ='RP_'+Ramp_Sheet.cell(row = 1, column = j+1).value
            #print feature
            #print(feature)
            #print i+2,j+1,sheet.cell(row = i+2,column = j+1).value
            vars()[feature] = Ramp_Sheet.cell(row = i+2,column = j+1).value
        if RP_RMP_ADT==None:
            continue
        ramp_aadt_short_list = []
        ramp_aadt_short_list.append(RP_RMP_ADT)
        for k in range(4):
            if (k+2010>=YEAR1) and (k+2010<=YEAR2) and vars()['RP_AADT'+str(k+2010)]!=None:
                ramp_aadt_short_list.append(vars()['RP_AADT'+str(k+2010)])
        RP_RMP_ADT = sum(ramp_aadt_short_list)/len(ramp_aadt_short_list)

        new_ramp = GP_RAMP(RP_milepost,float(RP_RMP_ADT),RP_on_offrp,RP_lane_num,\
                           RP_merge,RP_lane_before,RP_lane_after)
        Ramp_List.append(new_ramp)
    #print(len(Ramp_List))
    #continue
    #for segment in SEGMENT_LIST:0
     #   temp1,temp2,temp3,temp4,temp5 = segment.Weaving(Ramp_List)
      #  print temp1,temp2,temp3,temp4,temp5


    STATUS_OUTPUT(error_output,'RAMP Done')
    STATUS_OUTPUT(error_output,'Weaving Done')
    #raw_input()
    ######################## LANE WIDTH INPUT############################

    Width_Sheet = INPUT_WB['LANE_WIDTH']
    width_count = Width_Sheet.max_row-1
    Width_List = []
    for i in range(width_count):
        width_mp,width_hov,width_gp,width_gp_std = \
```

```
        [Width_Sheet.cell(row = i+2,column = x+2).value for x in range(4)]
        new_width_seg = LANE_WIDTH(width_mp,width_hov,width_gp,width_gp_std)
        Width_List.append(new_width_seg)
    STATUS_OUTPUT(error_output,'LANE WIDTH DONE')

    ################################################

    OUTPUT_FEATURES =
['ID','DISTRICT','RTE_NBR','ROAD_DIRECTION','START_MP','END_MP','LENGTH'\
                ,'AREA','COUNTY','Speed_Limit','Lane_Number_GP','ML_Lane_Number'\
                ,'HOV_Lane_Width','GP_Inside_Lane_Width','GP_Outside_Lane_Width'\

,'ONR_Distance','OFR_Distance','Up_ONR_AADT','Down_OFR_AADT','Up_Acc_Ramp_Distance','D
own_Acc_Ramp_Distance'\
                ,'Taper_Type','Median_Barrier_Type','Median_Width'\
                ,'Inside_Shoulder_Width','Inside_Pave_Shoulder'\
                ,'Outside_Shoulder_Width','Outside_Pave_Shoulder'\
                ,'Travel_Width'\
                ,'Right_Barrier_Type','Right_Barrier_Offset'\
                ,'Median_Type','Median_Offset'\
                ,'Curve_Length','Curve_Radius'\
                ,'Inside_Rumble','Inside_Rumble_Length','Outside_Rumble','Outside_Rumble_Length'\
                ,'Weave_Exist','Weave_Type','Weave_Length'\
                ,'ML_Design_Type','ML_Access_Type','ML_Access_Direction'\
                ,'ML_Lat_Type','ML_Buffer_Width'\
                ,'ML_Left_Shoulder','ML_Right_Shoulder'\
                ,'ML_Inter_Access_Type','ML_Inter_Access_Length'\
                ,'ML_AADT','ML_Lane_Width','ML_Up_Distance','ML_Down_Distance'\
                ,'Bi_AADT','Start_Year','End_Year'\
                ,'ML_Peak_Vol','ML_Restrict','Prop_AADT_Restrict','Heavy_Veh_Ave'\
                ,'Operation_GP','Operation_HOV'\
                ,'Auto_Enforce','Motor_Share','ML_GP_Speed_Diff'\
                ,'Speed_Diff_Ave','Ramp_AADT(SCL_Only)'\
                ,'Direct_AADT']
```

```
        Output_wb = openpyxl.Workbook()
        #Output_Sheet = Output_wb.active
        sheet_names = ['Normal','SCL_On','SCL_Off']
        output_sheet1 = Output_wb.create_sheet(sheet_names[0])
        output_sheet2 = Output_wb.create_sheet(sheet_names[1])
        output_sheet3 = Output_wb.create_sheet(sheet_names[2])

        output_sheet1.append(OUTPUT_FEATURES)
        output_sheet2.append(OUTPUT_FEATURES)
        output_sheet3.append(OUTPUT_FEATURES)

        record_count_1,record_count_2,record_count_3=1,1,1

        for segment in SEGMENT_LIST:
            #print segment.begmp,segment.endmp,segment.Truck_Station()
            #continue
            segment_output = []
            scl_start,scl_end,scl_type,scl_function = segment.SCL(SCL_List)
            taper_type = segment.Taper(Taper_List)
            curve_length,curve_radius = segment.Curve(Curve_List)

            lane_number = segment.Lane_Number(Taper_List,Ramp_List)
            if lane_number=='N/A':
                lane_number=segment.lane_number
            ml_lane_number = segment.MultiLane(Multilane_List)
            hov_lane_width,gp1_lane_width,gp2_lane_width = segment.Widths(Width_List
,Taper_List,Ramp_List)


ml_design_type,ml_lateral_type,ml_left_shoulder_width,ml_right_shoulder_width,ml_buffer_width \
                = segment.ML_Segment(ML_Segment_List)

            right_barrier_type,right_barrier_offset = segment.Barrier(RBarrier_List)
```

```
        median_barrier_type,median_barrier_offset = segment.Barrier(MBarrier_List)


up_distance,ml_itm_access_type,ml_itm_access_length,up_access_direction,up_access_ramp_distance,\
            up_access_ramp_aadt= segment.ML_ACCESS_Up(Access_List,Ramp_List)

        down_distance,down_access_type,down_access_direction,down_access_ramp_distance,\
                down_access_ramp_aadt= segment.ML_ACCESS_Down(Access_List,Ramp_List)

        weave_exist,weave_type,weave_length,ONR_distance,OFR_distance,ONR_aadt,OFR_aadt =
segment.Weaving(Ramp_List)

        truck_station,operation_gp,operation_hov =
segment.Truck_Station(),segment.Operation_GP(),segment.Operation_HOV()

        if scl_type =='N/A':
           record_count_1+=1
           record_count = record_count_1
        if scl_type =='O':
           record_count_2+=1
           record_count = record_count_2
        if scl_type =='F':
           record_count_2+=1
           record_count = record_count_3

        segment_output.extend([record_count,district,route_num,road_direction])
        segment_output.extend([segment.begmp,segment.endmp,segment.length,\
                        segment.area,segment.county,segment.speed_lmt])
        #if weave_exist:
        #    segment_output.extend([lane_number-1])
        #else:
        #    segment_output.extend([lane_number])
        segment_output.extend([lane_number])
        segment_output.extend([ml_lane_number])
```

```
        segment_output.extend([hov_lane_width,gp1_lane_width,gp2_lane_width])

segment_output.extend([ONR_distance,OFR_distance,ONR_aadt,OFR_aadt,up_access_ramp_distance,
down_access_ramp_distance])
        segment_output.extend([taper_type])
        segment_output.extend([median_barrier_type,segment.median_width])
        segment_output.extend([min(segment.inside_shoulder_width,median_barrier_offset)])
        segment_output.extend([min(segment.inside_pave_shoulder,median_barrier_offset)])
        segment_output.extend([min(segment.outside_shoulder_width,right_barrier_offset)])
        segment_output.extend([min(segment.outside_pave_shoulder,right_barrier_offset)])
        segment_output.extend([segment.travel_width])
        segment_output.extend([right_barrier_type,right_barrier_offset])
        segment_output.extend([median_barrier_type,median_barrier_offset])
        segment_output.extend([curve_length,curve_radius])
        segment_output.extend(['N/A','N/A','N/A','N/A'])
        segment_output.extend([weave_exist,weave_type,weave_length])
        segment_output.extend([ml_design_type,ml_itm_access_type,up_access_direction,\
                        ml_lateral_type,ml_buffer_width])
        segment_output.extend([ml_left_shoulder_width,ml_right_shoulder_width])
        segment_output.extend([ml_itm_access_type,ml_itm_access_length])
        segment_output.extend(['N/A','N/A'])
        segment_output.extend([up_distance,down_distance])
        #segment_output.extend(scl_type,abs(scl_start-scl_end))
        segment_output.extend([segment.bi_aadt,YEAR1,YEAR2,'N/A',ml_restrict,'N/A'])
        segment_output.extend([truck_station,operation_gp,operation_hov])
        segment_output.extend(['N/A','N/A','N/A','N/A'])

        if scl_type =='N/A':
           segment_output.extend(['N/A'])
           segment_output.extend([operation_gp])
           output_sheet1.append(segment_output)
        if scl_type =='O':
           up_ramp_distance,up_ramp_aadt = segment.Up_Ramp(Ramp_List)
           segment_output.extend([up_ramp_aadt])
           segment_output.extend([operation_gp])
```

```
            output_sheet2.append(segment_output)
        if scl_type =='F':
            down_ramp_distance,down_ramp_aadt = segment.Down_Ramp(Ramp_List)
            segment_output.extend([down_ramp_aadt])
            segment_output.extend([operation_gp])
            output_sheet3.append(segment_output)
        #segment_output.extend(['N/A'])
#x = input('press'
    Output_wb.remove(Output_wb['Sheet'])
    Output_wb.save('./RESULTS/'+FILE_ID+'_OUTPUT.xlsx')
    #os.rename('./Raw_Data'+input_file,'./RESULTS/'+input_file)
    # Check_Statistics('./RESULTS/'+FILE_ID+'_OUTPUT.xlsx')
    # break
    #FINISHED_FILE_LIST.append(input_file)
    print >> monitor_file,input_file
    monitor_file.close()
error_output.close()
file_summary.close()
```

CODE_Station_Download

```
import webbrowser
import codecs
import time
import os
import shutil
import openpyxl
import re

if not True:

    filelist = os.listdir('./RESULTS/')
    station_list= []
    for input_file in filelist:
        if input_file.endswith('.xlsx') and (input_file.startswith('DIS')):# or input_file.startswith('DIS07')):
```

```
        local_input_file = './RESULTS/'+input_file
        print local_input_file
        workbook  = openpyxl.load_workbook(local_input_file)
        sheetnames = ['Normal','SCL_On','SCL_Off']
        for name in sheetnames:
            worksheet = workbook[name]
            observe_count = worksheet.max_row
            if observe_count>1:
                for i in range(observe_count-1):
                    gp_station = worksheet.cell(row = i+2,column = 63).value
                    hov_station = worksheet.cell(row = i+2,column = 64).value
                    gp_station = str(gp_station)

                    hov_station = str(hov_station)
                   # if gp_station=='406574' or hov_station =='406574':
                    #    print 'find',local_input_file
                       #raw_input()
                    if not gp_station in station_list:
                        station_list.append(gp_station)
                    if not hov_station in station_list:
                        station_list.append(hov_station)

    print len(station_list)
    station_list_file = open('GP_HOV_Station.list','w')
    for item in station_list:
        print >> station_list_file,item
    station_list_file.close()
    for item in station_list:
        print item
station_list = []
input_station_file = open('./GP_HOV_Station.list','r')
for line in input_station_file.readlines():

    station_list.append(int(line))
```

```
input_station_file.close()
#raw_input()
YEARS = [2010,2011,2012,2013,2014]
#Start_Time_ID = ['1275350400','1306886400','1338508800','1370044800','1401580800']
#Start_Time =
['06%2F01%2F2010+00%3A00','06%2F01%2F2011+00%3A00','06%2F01%2F2012+00%3A00',\
#            '06%2F01%2F2013+00%3A00','06%2F01%2F2014+00%3A00']
#End_Time_ID = ['1283299140','1314835140','1346457540','1377993540','1409529540']
#End_Time =
['08%2F31%2F2010+23%3A59','08%2F31%2F2011+23%3A59','08%2F31%2F2012+23%3A59',\
#           '08%2F31%2F2013+23%3A59','08%2F31%2F2014+23%3A59']
Start_Time_ID = ['1394409600','1402876800','1410739200','1418601600']
Start_Time =
['03%2F10%2F2014+00%3A00','06%2F16%2F2014+00%3A00','09%2F15%2F2014+00%3A00',\
           '12%2F15%2F2014+00%3A00']
End_Time_ID = ['1395014340','1403481540','1411343940', '1419206340']
End_Time =
['03%2F16%2F2014+23%3A59','06%2F22%2F2014+23%3A59','09%2F21%2F2014+23%3A59',\
          '12%2F21%2F2014+23%3A59']
for ii in range(2):
   year = 2014
   i = ii+1
   #if year!=2014:
   #    continue
   start_time_id = Start_Time_ID[i]
   start_time = Start_Time[i]
   end_time_id = End_Time_ID[i]
   end_time = End_Time[i]
   for vds in station_list:
      print vds
      #new_name = './Stations/FLOW_ALL/Flow_'+str(vds)+'_'+str(year)+'.xlsx'
      #new_name2 = './Stations/Sep_HOV/Operation_'+str(vds)+'.xlsx'
      #if not os.path.exists(new_name):
      if True:
         #url =
"http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=analysis&tab=aadt&export=xls"+\
```

```
#"&station_id="+str(vds)+"&s_time_id=1401580800&s_time_id_f=06%2F2014&e_time_id=1401667140&e_time_id_f=06%2F2014"
        #url = "http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=analysis&tab=aadt&export=xls"+\
        #"&station_id="+str(vds)+"&s_time_id=1262304000&s_time_id_f=01%2F2010&e_time_id=1417478340&e_time_id_f=12%2F2014"
        url = "http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=loops&tab=det_timeseries&export=xls"+\

"&station_id="+str(vds)+"&s_time_id="+start_time_id+"&s_time_id_f="+start_time+\
                "&e_time_id="+end_time_id+"&e_time_id_f="+end_time+\

"&tod=all&tod_from=0&tod_to=0&dow_0=on&dow_1=on&dow_2=on&dow_3=on"+\

"&dow_4=on&dow_5=on&dow_6=on&holidays=on&q=flow&q2=speed&gn=5min&agg=on&lane1=on&lane2=on&lane3=on&lane4=on&lane5=on&lane6=on"
        webbrowser.open(url)
        #new_name = './Stations/Operation_w_Lane/'+'Operation_'+str(vds)+'_w_Lane.xlsx'
        #March "&station_id="+str(vds)+"&s_time_id=1394496000&s_time_id_f=03%2F11%2F2014+00%3A00&e_time_id=1394755140&e_time_id_f=03%2F13%2F2014+23%3A59"+\
        #June "&station_id="+str(vds)+"&s_time_id=1402963200&s_time_id_f=06%2F17%2F2014+00%3A00&e_time_id=1403222340&e_time_id_f=06%2F19%2F2014+23%3A59"+\
        local_new_name = './TEMP2/Weekly_'+str(vds)+'_'+str(i)+'.xlsx'
        time.sleep(30)
        down_sleep_count = 0
        found_download_file = False
        while (down_sleep_count<8 and not found_download_file):
           if os.path.exists("./TEMP2/pems_output.xlsx"):
              found_download_file = True
              shutil.move("./TEMP2/pems_output.xlsx",local_new_name)
              print "Done With",vds

              sleep_count=0
              found_check_file = False
```

```
            while (not found_check_file and sleep_count<8):
               time.sleep(1)
               sleep_count+=1
               if os.path.exists(local_new_name):
                  found_check_file = True
               else:
                  time.sleep(9)
            #       check_workbook = openpyxl.load_workbook(new_name)
            #       check_sheet = check_workbook['PeMS Report Description']
            #       check_seg = check_sheet['C12']
            #       print >> station_output,district,route,seg_id,start_mp,end_mp,PM,str(vds),
"CHECK",check_seg
            #       found_check_file = True
            #    else:
             #       time.sleep(2)
             #       sleep_count+=1
          else:
            time.sleep(10)
            down_sleep_count+=1
```

CODE_OPERATION_PROCESS

```
import time
import os
import shutil
import openpyxl
import statistics
def average_value(file_name):
    workbook = openpyxl.load_workbook(file_name)
    worksheet = workbook['Report Data']
    observe_count = worksheet.max_row
    result_list = []
    if observe_count>1:
        col_count = worksheet.max_column
        lane_count= 0
        lane_found = False
        for i in range(col_count):
            string_context = worksheet.cell(row = 1,column = i+1).value
            if "Flow (Veh/5 Minutes)" in string_context:
                flow_id = i
            if  "Speed (mph)" in string_context:
                speed_id = i
            if "# Lane Points" in string_context:
                lane_id = i
                lane_found = True
        if not lane_found:
            return([])
        else:
            lane_record=0

            for i in range(observe_count-1):
                flow_record = worksheet.cell(row=i+2,column =flow_id+1).value
                speed_record = worksheet.cell(row=i+2,column =speed_id+1 ).value
                lane_record = worksheet.cell(row=i+2,column = lane_id+1).value
                if lane_record!=0:
                    #print flow_record,lane_record,speed_record
                    flow_record = float(flow_record)/float(lane_record)*12
                    result_list.append([flow_record,speed_record])
                else:

                    raw_input('ERROR')

    return(result_list)
def truck_average(file_name):

    workbook = openpyxl.load_workbook(file_name)
    worksheet = workbook['Report Data']
```

```
    observe_count = worksheet.max_row
    result_list = []
    if observe_count>1:
        col_count = worksheet.max_column
        lane_count=0
        lane_found = False
        for i in range(col_count):
            string_context = worksheet.cell(row = 1,column = i+1).value
            if "Truck Prop (%)" in string_context and not "Lane" in string_context:
                truck_id = i
                lane_found = True
        if not lane_found:
            return([])
        else:
            lane_record=0
            #print flow_id,speed_id,lane_id
            #raw_input()
            for i in range(observe_count-1):
                truck_record = worksheet.cell(row=i+2,column = truck_id+1).value
                result_list.append(truck_record)


    return(result_list)
def merge_file():
    output_file = open('merged_file.txt','w')
    temp_list = [[]for x in range(4)]
    for i in range(4):
        location = LOCATIONS[i]

        input_file = open('./TEMP/'+location+'_diff.txt','r')
        for line in input_file.readlines():
            temp_value = [str(x) for x in line.split()]
            temp_list[i].append(temp_value)
    print len(temp_list[0]),len(temp_list[1]),len(temp_list[2]),len(temp_list[3])
    for i in range(len(temp_list[0])):

        print >> output_file,temp_list[0][i],temp_list[1][i],temp_list[2][i],temp_list[3][i]
    output_file.close()
    return


def aadt_value(file_name):
    workbook = openpyxl.load_workbook(file_name)
    worksheet = workbook['Report Data']
    max_col = worksheet.max_column
    for i in range(max_col):
        if "Sum of 24 Annual Avg Hours" in worksheet.cell(row = 1, column=i+1).value:
```

```
            aadt_id = i
    return(worksheet.cell(row=2,column =aadt_id+1 )).value



LOCATIONS = ['March','June','Sep','Dec']
#merge_file()
#raw_input()
station_input = open('./example_stations.txt','r')
station_list = []
for line in station_input.readlines():
    temp = [str(x) for x in line.split()]
    sta_pair = [temp[0],temp[2]]
    #print sta_pair
    station_list.append(sta_pair)
#raw_input()
filelist = os.listdir('./RESULTS/')
station_list= []
for input_file in filelist:
    if input_file.endswith('.xlsx') and input_file.startswith('DIS07_RT105_RD1_OUTPUT'):# or
input_file.startswith('DIS07')):

        local_input_file = './RESULTS/'+input_file
        print local_input_file
        workbook  = openpyxl.load_workbook(local_input_file)
        worksheet = workbook['Normal']
        variable_count = worksheet.max_column
        observe_count = worksheet.max_row
        for i in range(variable_count):
            if worksheet.cell(row = 1,column = i+1).value=='Heavy_Veh_Ave':
                truck_id = i+1
                gp_id = i+1+1
                hov_id = i+1+2
        for i in range(observe_count-1):
            truck_station = worksheet.cell(row=i+2,column = truck_id).value
            gp_station = truck_station
            hov_station = worksheet.cell(row = i+2,column = hov_id).value
            new_observation = []
            for j in range(9):
                new_observation.extend([worksheet.cell(row = i+2,column = j+1).value])
            for location in LOCATIONS:
                file_path = './Stations/'+location+'/Operation_'+gp_station+'./xlsx'
                file_path2 = './Stations/'+location+'/Operation_'+hov_station+'./xlsx'
                aadt_path = './Stations/AADT/AADT_'+gp_station+'.xlsx'
                aadt_path2 = './Stations/AADT/AADT_'+hov_station+'.xlsx'
                truck_path = './Stations/Truck_Prop/Truck_Prop_'+gp_station+'.xlsx'
                if gp_station!=hov_station:
```

```
                gp_list = average_value(file_path,'a')
                hov_list = average_value(file_path2,'a')
            else:
                gp_list = average_value(file_path,'f')
                hov_list = average_value(file_path2,'a')
            for j in range(len(truck_list)):
                truck_list[j] = truck_list[j]/100.0*gp_list[j][0]
            count_num = len(gp_list)
            flow_diff = []
            speed_diff = []
            for j in range(count_num):
                delta_flow = gp_list[i][0]-hov_list[i][0]
                delta_speed = gp_list[i][1]-hov_list[i][1]
                flow_diff.append(delta_flow)
                speed_diff.append(delta_speed)



for location in LOCATIONS:
    output_file = open('./TEMP/'+location+'_diff.txt','w')
    print location
    for station_pair  in station_list:
        #print station_pair
        station1 = station_pair[0]
        station2 = station_pair[1]

        file_path = './TEMP/'+'Operation_'+station1+'_'+location+'.xlsx'
        file_path2 = './TEMP/'+'Operation_'+station2+'_'+location+'.xlsx'
        aadt_path = './TEMP/AADT_'+station1+'_Sep.xlsx'
        aadt_path2 = './TEMP/AADT_'+station2+'_Sep.xlsx'
        truck_file_path = './TEMP/'+'Truck_'+station1+'_Sep.xlsx'
        print aadt_value(aadt_path)+aadt_value(aadt_path2),aadt_value(aadt_path2)
        gp_list = average_value(file_path)
        hov_list = average_value(file_path2)
        truck_list = truck_average(truck_file_path)
        for i in range(len(truck_list)):

            truck_list[i] = truck_list[i]/100.0*gp_list[i][0]
        #for item in gp_list:
          # print item
        #raw_input()
        count_num = len(gp_list)
        flow_diff = []
        speed_diff = []

        for i in range(count_num):
            #print station1,station2,gp_list[i],hov_list[i]
```

```
            #raw_input()
            delta_flow = gp_list[i][0]-hov_list[i][0]
            delta_speed = gp_list[i][1]-hov_list[i][1]
            flow_diff.append(delta_flow)
            speed_diff.append(delta_speed)
            #print delta_flow,delta_speed
        high_vol_count=0
        for item in gp_list:
            #print item
            if item[0]>1000:
                high_vol_count+=1
        print >>
output_file,sum(flow_diff)/len(flow_diff),sum(speed_diff)/len(speed_diff),statistics.stdev(flow_diff),stat
istics.stdev(speed_diff),high_vol_count/float(len(gp_list)),sum(truck_list)/len(truck_list)
    output_file.close()

merge_file()
```

CODE_CRASH

```
class SEGMENT:
    def __init__(self,district,route,county,direction,milepost,record_list):
        self.route = route
        self.district = district
        self.county = county
        self.direction = direction
        self.milepost = milepost
        self.record_list = record_list
    def speed_diff_ave(self):

        diff_list = []
        if len(self.record_list)==0:
            #print 'Error', route,direction,milepost
            #raw_input()
            return('NA')
        for item in self.record_list:
            gp_speed = item[2]
            hov_speed = item[4]
            diff_list.append(gp_speed-hov_speed)
        return(statistics.mean(diff_list))
    def speed_diff_sd(self):
        diff_list = []
        if len(self.record_list)==0:
            return('NA')
        for item in self.record_list:
            gp_speed = item[2]
            hov_speed = item[4]
            diff_list.append(gp_speed-hov_speed)
        return(statistics.stdev(diff_list))
    def flow_diff_ave(self):
        diff_list = []
        if len(self.record_list)==0:
            return('NA')
        for item in self.record_list:
            gp_flow = item[1]
            hov_flow = item[3]
            #print item,gp_flow,hov_flow
            diff_list.append(gp_flow-hov_flow)
        return(statistics.mean(diff_list))
    def flow_diff_sd(self):
        diff_list = []
        if len(self.record_list)==0:
            return('NA')
        for item in self.record_list:
            gp_flow = item[1]
```

```
            hov_flow = item[3]
            diff_list.append(gp_flow-hov_flow)
        return(statistics.stdev(diff_list))
    def high_vol_duration(self):
        if len(self.record_list)==0:
            return('NA')
        high_vol_count=0
        for item in self.record_list:
            total_vol = item[1]
            if total_vol>1000:
                high_vol_count+=1
        return(high_vol_count*1.0/len(self.record_list))
class INCIDENT:
    def __init__(self,district,route,county,milepost,caseid,direction,severity,veh_count,veh_invl,acctype):
        self.district = district
        self.route = route
        self.county = county
        self.caseid = caseid
        self.milepost = milepost
        self.direction = direction
        self.severity = severity
        self.veh_count = veh_count
        self.veh_invl = veh_invl
        self.acctype = acctype
    def letter(self):
        if self.severity==1:
            return('K')

        if self.severity==2:
            return('A')
        if self.severity==3:
            return('B')
        if self.severity==4:
            return('C')
        return('O')
class VEHICLE:
    def __init__(self,case_id,direction):
        self.case_id = case_id
        self.direction = direction


import os
import openpyxl
import statistics
import winsound
import sys
Finished_File = []
```

```
totallength=0
input_finish = open('./file_status.txt','r')
for line in input_finish.readlines():
    old_file = line[:-1]
    Finished_File.append(old_file)
    #print old_file
input_finish.close()
RESTRICT_RECORD = []
restrict_input = open('./restriction_table.txt','r')
restrict_input.readline()
for line in restrict_input.readlines():
    temp = [str(x) for x in line.split(',')]
    temp[-1]=temp[-1][:-1]

    RESTRICT_RECORD.append(temp)
restrict_input.close()

filelist = os.listdir('./RESULTS/')
Missing_Station = []
frequency = 2500
duration = 200
#vehicle_wb = openpyxl.load_workbook('./ca14veh_part.xlsx')
#vehicle_ws = vehicle_wb['ca14veh']
#print "Vehicle File Loaded"
#veh_count = vehicle_ws.max_row-1
#feature_count = vehicle_ws.max_column
#Date_List = []
#TEMP = []
#for i in range(feature_count):
#    TEMP.append(str(vehicle_ws.cell(row = 1,column=i+1).value))
#for i in range(veh_count):
#    temp=[]
#    for j in range(feature_count):
#        temp.append(str(vehicle_ws.cell(row = i+2,column=j+1).value))
#    current_date = temp[0][:8]
#    if i%100==0:
#        print i,current_date
#    continue
#    if current_date in Date_List:
#        print >> vars()['F'+current_date],','.join(temp)
#    else:
#        vars()['F'+current_date] = open('./VEH/'+current_date+'.txt','w')
#        print >> vars()['F'+current_date],','.join(TEMP)
#        print >> vars()['F'+current_date],','.join(temp)
#        Date_List.append(current_date)
#for item in Date_List:
#    vars()['F'+item].close()
```

```
AADT_RECORD = []
aadt_input_file = open('./AADT_Summary.txt','r')
aadt_input_file.readline()
for line in aadt_input_file.readlines():
    temp = [int(x) for x in line.split()]

    AADT_RECORD.append(temp)
aadt_input_file.close()
input_finish = open('./file_status.txt','a+')
for input_file in filelist:

    if not input_file.endswith('_OUTPUT.xlsx'):
        continue

    local_input_file = './RESULTS/'+input_file
    workbook = openpyxl.load_workbook(local_input_file)

    Output_WB = openpyxl.Workbook()
    SheetNames = workbook.sheetnames
    file_info = [str(x) for x in input_file.split('_')]
    district = file_info[0][3:]
    route = file_info[1][2:]


    segment = int(file_info[2][2:])
    normal_sheet = workbook['Normal']
    county = normal_sheet.cell(row=2,column = 9).value
    year1 = normal_sheet.cell(row = 2,column = 57).value
    year2 = normal_sheet.cell(row=2,column = 58).value
    restrict = normal_sheet.cell(row=2,column = 60).value
    for restrict_rule in RESTRICT_RECORD:
        if restrict == restrict_rule[0]:
            restrict_hours = [restrict_rule[x+1] for x in range(len(restrict_rule)-1)]
    #print restrict_hours
    min_mp = 10000
    max_mp = -1000
    segment_count = normal_sheet.max_row-1
    for i in range(segment_count):
        mp1 = float(normal_sheet.cell(row=i+2,column = 5).value)
        mp2 = float(normal_sheet.cell(row=i+2,column = 6).value)
        if mp1>max_mp:
            max_mp = mp1
        if mp1<min_mp:
            min_mp = mp1
        if mp2>max_mp:
            max_mp = mp2
        if mp2<min_mp:
```

```
            min_mp = mp2

        if mp1<=mp2:
            direction = 'NE'
        else:
            direction = 'SW'

        print district, county,route,segment,direction,min_mp,max_mp



        Incident_List = []
        for year_incre in range(int(year2)-int(year1)+1):
            year = year_incre+int(year1)
            if year==2012:
                continue
            incident_input = open('./ACC/REV_DIS'+str(district)+'_RT'+str('00'+route)[-
3:]+'_ACC'+str(year)[-2:]+'.txt')
            #print 'incident file opened'
            #incident_wb = openpyxl.load_workbook('./ca14acc.xlsx')
            #incident_ws = incident_wb['ca'+str(year)[2:]+'acc']
            #print "Incident & Vehicle File Opened"

            incident_fea_line = incident_input.readline()
            incident_fea_line = incident_fea_line[:-1]
            incident_features = [str(x) for x in incident_fea_line.split(',')]
           #print incident_features

            Found_Incident = False

            Vehicle_List = []
            #vehicle_count = vehicle_ws.max_row
            Directions = ['N','S','W','E']

            for line in incident_input.readlines():
                line = line[:-1]
                item = [str(x) for x in line.split(',')]
                for i in range(len(item)):
                    vars()['T_'+incident_features[i].lower()] = item[i]
                try:
                    float(T_milepost)
                    T_milepost_flag = False
                except ValueError:
                    T_milepost_flag = True
                if not T_rte_nbr.isdigit() or not T_district.isdigit() or T_milepost_flag:
                    continue
```

```
        incident_route = int(T_rte_nbr)
        #print incident_route
        incident_district = int(T_district)
        incident_county = int(T_county)
        incident_milepost = float(T_milepost)

        if incident_route ==int(route) and incident_district==int(district) and
incident_county==int(county) and\
        incident_milepost>=min_mp and incident_milepost<max_mp:
            Found_Incident = True
            if not T_severity in ['0','1','2','3','4']:
                T_severity ='0'
            incident_severity = int(T_severity)


            if not str(T_numvehs).isdigit():
#                 print T_caseno,T_numvehs,T_veh_invl,T_acctype
                T_numvehs=1
            incident_id = str(T_caseno)
            if str(T_direction) in Directions:

                incident_direction = str(T_direction).upper()
            else:
                continue

            incident_vehnum = int(T_numvehs)
            #print incident_vehnum
            incident_veh_invl = T_veh_invl
            incident_acctype = T_acctype
            #print "Incident_Found",incident_id
            #print 'test',incident_route,incident_direction,incident_milepost
            new_incident =
INCIDENT(incident_district,incident_route,incident_county,incident_milepost,\
                                incident_id,incident_direction,incident_severity,incident_vehnum,\
                                incident_veh_invl,incident_acctype)
            #print "--Incident Found for current corridor"
            Incident_List.append(new_incident)
            Found_Case= False
            target_date = str(new_incident.caseid[:6])
            vehicle_input_file = open('./VEH/'+target_date+'.txt')
            vehicle_input_file.readline()
            for vehicle_line in vehicle_input_file.readlines():
                vehicle_temp = [str(x) for x in vehicle_line.split(',')]
                vehicle_id = vehicle_temp[0]
                print vehicle_id,new_incident.caseid
                if vehicle_id==new_incident.caseid:
```

```
                print "----Vehicle Found"
                vehicle_direction = vehicle_temp[11]
                if vehicle_direction in Directions:
                    new_vehicle = VEHICLE(vehicle_id,vehicle_direction)
                    if vehicle_direction in direction:
                    new_incident.direction=vehicle_direction
                    Vehicle_List.append(new_vehicle)
                    Found_Case = True
                    print "------Incident Direction Found"
                    break
            vehicle_input_file.close()
            if not Found_Case:
                print '--vehicle not found',incident.caseid
            else:
                print "--Vehicle Found on Case",incident.caseid
        incident_input.close()
        print len(Incident_List)
    if not Found_Incident:
        print "No incident found on this segment",
       # raw_input()



    incident_wo_direct = 0
    for incident in Incident_List:
        if incident.direction==":
            incident_wo_direct+=1
    print len(Incident_List),incident_wo_direct



    print "incident number",len(Incident_List)

    for item in Incident_List:
        print item.route,item.milepost,item.direction,item.severity
    print "Incident Input Done!",len(Incident_List)
    GP_Station_Info = []
    HOV_Station_Info = []
    Target_Months = ['March','June','Sep','Dec']


    # for target_month in Target_Months:
     #    gp_input = open('../GP_LOOP/RT_'+str(route)+'_'+direction+'_'+target_month+'.txt','r')
    #     hov_input = open('../HOV_LOOP/RT_'+str(route)+'_'+direction+'_'+target_month+'.txt','r')
    #     GP_Stations = []
    #     HOV_Stations = []
```

```
    #     for line in gp_input.readlines():
    #         temp = [str(x) for x in line.split()]
    #         GP_Stations.append(temp)
    #     for line in hov_input.readlines():
    #         temp = [str(x) for x in line.split()]
     #        HOV_Stations.append(temp)
    #     gp_input.close()
    #     hov_input.close()
    #     GP_Station_Info.append(GP_Stations)
    #     HOV_Station_Info.append(HOV_Stations)
#
#
#
#
    for sheetname in SheetNames:
        #print sheetname
        total_incident = 0
        worksheet = workbook[sheetname]
        observe_count = worksheet.max_row-1
        totallength+=observe_count
        print totallength

        variable_count = worksheet.max_column
        OUTPUT_FEATURES = []
        output_sheet = Output_WB.create_sheet(sheetname)
        for i in range(7):
            OUTPUT_FEATURES.extend([worksheet.cell(row = 1, column=i+1).value])
#
#
#
        #print district,route,segment,restriction


        Variable_List =
['Flow_Diff_Ave','Speed_Diff_Ave','Flow_Diff_SD','Speed_Diff_SD','High_Vol_Duration']


        for month in Target_Months:
            for variable_name in Variable_List:
                new_variable_name = variable_name+'('+month+')'
                OUTPUT_FEATURES.extend([new_variable_name])

        OUTPUT_FEATURES.extend(['Truck_Prop','AADT','Restrict_Vol'])
        OUTPUT_FEATURES.extend(['K','A','B','C','O','ALL'])
        OUTPUT_FEATURES.extend(['Single-Veh','Multi-Veh'])
```

```
OUTPUT_FEATURES.extend(['SV_animal_FI','SV_animal_PDO','SV_fixed_FI','SV_fixed_PDO','SV_
other_FI','SV_other_PDO'])
    OUTPUT_FEATURES.extend(['Other_SV_FI','Other_SV_PDO'])
    OUTPUT_FEATURES.extend(['MV_K','MV_A','MV_B','MV_C','MV_O'])

OUTPUT_FEATURES.extend(['MV_Head_FI','MV_Head_PDO','MV_Right_FI','MV_Right_PDO','M
V_Rear_FI','MV_Rear_PDO','MV_Side_FI','MV_Side_PDO','MV_Other_FI','MV_Other_PDO'])
    output_sheet.append(OUTPUT_FEATURES)
    #print OUTPUT_FEATURES
    all_incident_count = 0
    for i in range(observe_count):
        local_incident_list  = []
        print input_file,i, '/',observe_count
        winsound.Beep(frequency,duration)
        OUTPUT_FEATURES = []
        for j in range(7):
            OUTPUT_FEATURES.extend([worksheet.cell(row=i+2,column=j+1).value])

        start_mp = float(worksheet.cell(row = i+2,column=5).value)
        end_mp = float(worksheet.cell(row = i+2,column=6).value)
        #OUTPUT_FEATURES.extend([route,direction,start_mp,end_mp])
        milepost = 0.5*start_mp+0.5*end_mp
        #truck_station = str(worksheet.cell(row = i+2,column = 62).value)
        gp_station = str(worksheet.cell(row = i+2,column = 63).value)
        hov_station = str(worksheet.cell(row = i+2,column = 64).value)
        truck_station =str(worksheet.cell(row = i+2,column = 62).value)
        for month in Target_Months:
            if gp_station=='Error' or hov_station=='Error':
                OUTPUT_FEATURES.extend(['N/A','N/A','N/A','N/A','N/A'])
                #print "Operatinal processed"
                continue
            gp_exists = os.path.isfile('./Stations/'+month+'/Operation_'+gp_station+'.xlsx')
            hov_exists = os.path.isfile('./Stations/'+month+'/Operation_'+hov_station+'.xlsx')
            if not gp_exists or not hov_exists:
                OUTPUT_FEATURES.extend(['N/A','N/A','N/A','N/A','N/A'])
                continue

            gp_wb = openpyxl.load_workbook('./Stations/'+month+'/Operation_'+gp_station+'.xlsx')
            hov_wb = openpyxl.load_workbook('./Stations/'+month+'/Operation_'+hov_station+'.xlsx')
            gp_ws = gp_wb['Report Data']
            gp_observe_num = gp_ws.max_row-1
            gp_feature_num = gp_ws.max_column
            for k in range(gp_feature_num):
                feature_name = str(gp_ws.cell(row=1,column = k+1).value)
                if feature_name=="Flow (Veh/5 Minutes)":
                    gp_flow_feature_num = k
```

```
            if feature_name == "Speed (mph)":
                gp_speed_feature_num = k

        gp_record = []
        for gp_id in range(gp_observe_num):
            date_time = str(gp_ws.cell(row=gp_id+2,column = 1).value)
            gp_flow = float(gp_ws.cell(row = gp_id+2,column = gp_flow_feature_num+1).value)
            gp_speed = float(gp_ws.cell(row = gp_id+2,column = gp_speed_feature_num+1).value)
            gp_lane = float(gp_ws.cell(row = gp_id+2,column = gp_speed_feature_num+2).value)
            temp = [date_time,gp_flow/gp_lane*12,gp_speed]
            gp_record.append(temp)
        hov_ws = hov_wb['Report Data']
        hov_observe_num = hov_ws.max_row-1
        hov_feature_num = hov_ws.max_column

        for k in range(hov_feature_num):
            feature_name = str(hov_ws.cell(row=1,column = k+1).value)
            if int(district)==4:
                if feature_name=="Lane 1 Flow (Veh/5 Minutes)":
                    hov_flow_feature_num = k
                if feature_name=="Lane 1 Speed (mph)":
                    hov_speed_feature_num = k


            else:
                if feature_name=="Flow (Veh/5 Minutes)":
                    hov_flow_feature_num = k
                if feature_name == "Speed (mph)":
                    hov_speed_feature_num = k
        hov_record = []
        for hov_id in range(hov_observe_num):
            date_time = str(hov_ws.cell(row = hov_id+2,column = 1).value)
            hov_flow = float(hov_ws.cell(row = hov_id+2,column = hov_flow_feature_num+1).value)
            hov_speed = float(hov_ws.cell(row = hov_id+2,column =
hov_speed_feature_num+1).value)
            temp = [date_time,hov_flow*12,hov_speed]

            hov_record.append(temp)
        if len(hov_record)!=len(gp_record):
            #print 'Record Error',gp_station,hov_station
            Record_List=[]
        else:
            Record_List = []
            for k in range(len(hov_record)):
```

```
Record_List.append(([gp_record[k][0],gp_record[k][1],gp_record[k][2],hov_record[k][1],hov_record[k]
[2]]))


            new_segment = SEGMENT(district,route,county,direction,milepost,Record_List)


OUTPUT_FEATURES.extend([new_segment.flow_diff_ave(),new_segment.speed_diff_ave(),\
                        new_segment.flow_diff_sd(),new_segment.speed_diff_sd(),\
                        new_segment.high_vol_duration()])


        exists = os.path.isfile('./Stations/Truck_Prop/Truck_Prop_'+truck_station+'.xlsx')
        if exists :
            truck_wb =
openpyxl.load_workbook('./Stations/Truck_Prop/Truck_Prop_'+truck_station+'.xlsx')
            truck_ws = truck_wb['Report Data']
            truck_feature_num = truck_ws.max_column
            truck_observe_num = truck_ws.max_row-1
            for k in range(truck_feature_num):
                feature_title = str(truck_ws.cell(row = 1,column = k+1).value)
                if feature_title =='Truck Prop (%)':
                    truck_prop_feature_num = k
                    break
            truck_record = []
            for k in range(truck_observe_num):
                if truck_ws.cell(row =k+2,column = truck_prop_feature_num+1).value!=None:
                    temp_value= float(truck_ws.cell(row =k+2,column = truck_prop_feature_num+1).value)
                    #OUTPUT_FEATURES.extend([statistics.mean(truck_record)])
                else:
                    temp_value = 0
                    #OUTPUT_FEATURES.extend(['N/A'])
                #print temp_value

                truck_record.append(temp_value)
        else:
            truck_record = [0]
            Missing_Station.append(truck_station)
        if len(truck_record)>0:
            OUTPUT_FEATURES.extend([statistics.mean(truck_record)])
        else:
            OUTPUT_FEATURES.extend(['N/A'])
        print truck_record
        exists = os.path.isfile('./Stations/AADT_More/AADT_'+truck_station+'.xlsx')
```

```
        if exists :
            aadt_wb = openpyxl.load_workbook('./Stations/AADT_More/AADT_'+truck_station+'.xlsx')
            aadt_ws = aadt_wb['Report Data']
            aadt_feature_count =aadt_ws.max_column
            for k in range(aadt_feature_count):
                feature_title = str(aadt_ws.cell(row = 1,column = k+1).value)
                if feature_title=='Sum of 24 Annual Avg Hours':
                    aadt_feature_num = k
                    break
            print aadt_ws.cell(row = 2,column = aadt_feature_num+1).value
            if aadt_ws.cell(row = 2,column = aadt_feature_num+1).value==None:
                aadt_value = 'N/A'
            else:
                aadt_value = float(aadt_ws.cell(row = 2,column = aadt_feature_num+1).value)

        else:
            aadt_value = 'N/A'
            Missing_Station.append(truck_station)
        aadt_found = False
        if truck_station.isdigit():
            for aadt_record in AADT_RECORD:
                if aadt_record[0]==int(truck_station):
                    aadt_sum = 0
                    aadt_count = 0
                    for kk in range(5):
                        if aadt_record[kk+1]!=0:
                            aadt_sum +=aadt_record[kk+1]
                            aadt_count+=1
                            aadt_found=True
                    if aadt_found:
                        break
        if aadt_found:
            aadt_value = aadt_sum/aadt_count
        else:
            aadt_value = 'NA'
        OUTPUT_FEATURES.extend([aadt_value])
        total_vol = 0
        restrict_vol = 0
        hour1,min1,hour2,min2 =
int(restrict_hours[0]),int(restrict_hours[1]),int(restrict_hours[2]),int(restrict_hours[3])
        #print restrict[4],restrict[5]
        if restrict_hours[4]!='':
            hour3,min3,hour4,min4 =
int(restrict_hours[4]),int(restrict_hours[5]),int(restrict_hours[6]),int(restrict_hours[7])
        else:
            hour3,min3,hour4,min4 = -1,0,-1,0
        if not os.path.isfile('./Stations/HOURLY_VOL/HOURLY_VOL_'+str(truck_station)+'.txt'):
```

```
            restrict_pp = 'NA'
        else:
            hourly_vol = open('./Stations/HOURLY_VOL/HOURLY_VOL_'+str(truck_station)+'.txt')
            for vol_line in hourly_vol.readlines():
                temp = [int(x) for x in vol_line.split()]
                if temp[1]>=year1 and temp[1]<=year2:
                    if (temp[2]>=hour1 and temp[2]<=hour2) or (temp[2]>=hour3 and temp[2]<=hour4):
                        if (temp[2]==hour1 and min1!=0)or(temp[2]==hour2 and min2!=0) or
(temp[2]==hour3 and min3!=0) or (temp[2]==hour4 and min4!=0):
                            restrict_vol+=temp[3]*0.5
                            total_vol+=temp[3]
                        else:
                            restrict_vol+=temp[3]
                            total_vol+=temp[3]
                    else:
                        total_vol+=temp[3]
            if total_vol!=0:
                restrict_pp = 1.0*restrict_vol/total_vol
        OUTPUT_FEATURES.extend([restrict_pp])



        hourly_vol.close()

        incident_history = ''
        K_count=0
        A_count=0
        B_count=0
        C_count=0
        O_count=0
        all_count = 0
        local_incident_list = []
        for incident in Incident_List:
            #print incident.milepost,start_mp,end_mp

            if ((incident.milepost>=start_mp and incident.milepost<end_mp and start_mp<end_mp)or\
            (incident.milepost<start_mp and incident.milepost>=end_mp and start_mp>end_mp))and
(incident.direction in direction):
                #print incident.letter()
                #print 'found one',direction,incident.direction,incident.severity
                local_incident_list.append(incident)
                incident_history+=incident.letter()
        #print incident_history
        all_incident_count+=len(local_incident_list)
        for letter in incident_history:
            vars()[letter+'_count']+=1
        all_count = len(incident_history)
```

```
        OUTPUT_FEATURES.extend([K_count,A_count,B_count,C_count,O_count,all_count])
        #if all_count!=len(local_incident_list):
        #    print 'count_error'
        #    raw_input()
        SV_count= 0
        MV_count = 0
        MV_K,MV_A,MV_B,MV_C,MV_O = 0,0,0,0,0

SV_animal_FI,SV_animal_PDO,SV_fixed_FI,SV_fixed_PDO,SV_other_FI,SV_other_PDO,Other_SV_
PDO,Other_SV_FI = 0,0,0,0,0,0,0,0

MV_head_FI,MV_head_PDO,MV_right_FI,MV_right_PDO,MV_rear_FI,MV_rear_PDO,MV_side_FI,
MV_side_PDO,MV_other_FI,MV_other_PDO = 0,0,0,0,0,0,0,0,0,0

        #print 'incident list length local ',len(local_incident_list),len(incident_history)

        for incident in local_incident_list:
            #print incident.veh_count,incident.acctype,incident.veh_invl,incident.letter()
            incident_sev = incident.letter()
            #print 'vehicle count',incident.veh_count
            if incident.veh_count==1:
                SV_count+=1
                #print 'new sv incident found',SV_count
                if incident.veh_invl=='H':
                    if incident_sev=='O':
                        SV_animal_PDO+=1
                    else:
                        SV_animal_FI+=1
                else:
                    if incident.veh_invl=='I':
                        if incident_sev=='O':
                            SV_fixed_PDO+=1
                        else:
                            SV_fixed_FI+=1
                    else:
                        if incident.veh_invl=='A':
                            if incident_sev =='O':
                                Other_SV_PDO +=1
                            else:
                                Other_SV_FI +=1
                        else:
                            if incident_sev=='O':
                                SV_other_PDO+=1
                            else:
                                SV_other_FI+=1

            else:
```

```
                MV_count+=1
               #print 'mv incident found',MV_count
                vars()['MV_'+incident.letter()]+=1

                if incident.acctype=='A':
                   if incident_sev=='O':
                      MV_head_PDO+=1
                   else:
                      MV_head_FI+=1
                else:
                   if incident.acctype=='D':
                      if incident_sev=='O':
                         MV_right_PDO +=1
                      else:
                         MV_right_FI+=1
                   else:
                      if incident.acctype=='C':
                         if incident_sev=='O':
                            MV_rear_PDO+=1
                         else:
                            MV_rear_FI+=1
                      else:
                         if incident.acctype=='B':
                            if incident_sev =='O':
                               MV_side_PDO+=1
                            else:
                               MV_side_FI+=1
                         else:
                            if incident_sev=='O':
                               MV_other_PDO+=1
                            else:
                               MV_other_FI+=1
            #print SV_count,MV_count,len(local_incident_list)
            if SV_count+MV_count!=len(local_incident_list):
               print 'error'
               raw_input()

            OUTPUT_FEATURES.extend([SV_count, MV_count])


            total_incident +=SV_count+MV_count

OUTPUT_FEATURES.extend([SV_animal_FI,SV_animal_PDO,SV_fixed_FI,SV_fixed_PDO,SV_othe
r_FI,SV_other_PDO,Other_SV_FI,Other_SV_PDO])
            OUTPUT_FEATURES.extend([MV_K,MV_A,MV_B,MV_C,MV_O])
```

```
OUTPUT_FEATURES.extend([MV_head_FI,MV_head_PDO,MV_right_FI,MV_right_PDO,MV_rear_
FI,MV_rear_PDO])
        OUTPUT_FEATURES.extend([MV_side_FI,MV_side_PDO,MV_other_FI,MV_other_PDO])
        #print OUTPUT_FEATURES


        output_sheet.append(OUTPUT_FEATURES)
        print total_incident

        #print all_incident_count
   #continue
   Output_WB.remove(Output_WB['Sheet'])

Output_WB.save('./RESULTS/DIS'+str(district)+'_RT'+str(route)+'_RD'+str(segment)+'_Operation_Sho
rt.xlsx')
   print >> input_finish,input_file
   input_finish.flush()

output_missing = open('./MISSING_STATIONS.txt','w')
for item in Missing_Station:
   print >> output_missing,item
   #print item
output_missing.close()
input_finish.close()
```

CODE_AADT

```
import os
import openpyxl
filelist = os.listdir('./RESULTS')
AADT_RECORD = []
AADT_RECORD2 = []
aadt_input = open('./AADT_Summary.txt','r')
aadt_input.readline()
for line in aadt_input.readlines():
    temp = [str(x) for x in line.split()]
    AADT_RECORD.append(temp)
aadt_input.close()
aadt_input = open('./AADT_Summary_II.txt','r')
aadt_input.readline()
for line in aadt_input.readlines():
    temp = [str(x) for x in line.split()]
    AADT_RECORD2.append(temp)
aadt_input.close()

for output_file in filelist:
    if output_file.endswith('Operation.xlsx') or output_file.endswith('AADT.xlsx'):
        continue

    new_aadt_wb = openpyxl.Workbook()

    local_output_file = './RESULTS/'+output_file
    output_wb = openpyxl.load_workbook(local_output_file)
    sheetnames = output_wb.sheetnames
    operation_wb = openpyxl.load_workbook('./RESULTS/'+output_file[:-11]+'Operation.xlsx')

    print sheetnames
    for sheet in sheetnames:
        new_aadt_ws = new_aadt_wb.create_sheet(sheet)
        OUTPUT_FEATURES = []
        OUTPUT_FEATURES.extend(['start_mp','end_mp','HOV_AADT','GP_AADT','Total_AADT'])
        new_aadt_ws.append(OUTPUT_FEATURES)
        output_ws = output_wb[sheet]
        operation_ws = operation_wb[sheet]
        observe_count = output_ws.max_row-1
        print observe_count
        if observe_count==0:
            continue
        #print output_file,sheet,operation_ws.cell(row = 2,column = 2).value,output_ws.max_row-1
        district = int(operation_ws.cell(row = 2,column = 2).value)
        feature_count = output_ws.max_column
        for i  in range(feature_count):
```

```
    feature_name = output_ws.cell(row = 1,column = i+1).value
    if feature_name=='Operation_GP':
       gp_id = i
    if feature_name =='Operation_HOV':
       hov_id = i
    if feature_name =='Start_Year':
       year1_id = i
    if feature_name == 'End_Year':
       year2_id = i

#print observe_count
if observe_count>0:
    for i in range(observe_count):
       hov_station = output_ws.cell(row = i+2,column = hov_id+1).value
       gp_station = output_ws.cell(row = i+2,column = gp_id+1).value
       year1 = output_ws.cell(row = i+2,column = year1_id+1).value
       year2 = output_ws.cell(row = i+2,column = year2_id+1).value
       start_mp = output_ws.cell(row = i+2,column = 5).value
       end_mp = output_ws.cell(row = i+2,column = 6).value
       existing_aadt = str(operation_ws.cell(row = i+2,column = 29 ).value)
       print "aadt is",output_file,sheet,existing_aadt
       # hov_station,gp_station
       if hov_station!=gp_station:
          RECORD_LIST = AADT_RECORD
       else:
          RECORD_LIST = AADT_RECORD2
       for item in RECORD_LIST:
          if item[0]==str(hov_station):
             aadt2010 = int(item[1])
             aadt2011 = int(item[2])
             aadt2012 = int(item[3])
             aadt2013 = int(item[4])
             aadt2014 = int(item[5])
             sum_aadt = 0
             count_aadt = 0
             for year_inc in range(5):
                if year_inc+2010>=year1 and year_inc+2010<=year2:
                   if vars()['aadt'+str(year_inc+2010)]!=0:
                      sum_aadt+=vars()['aadt'+str(year_inc+2010)]
                      count_aadt+=1

             if count_aadt!=0 and existing_aadt.isdigit():
                existing_aadt = int(existing_aadt)
                if district==4:
```

```
                    OUTPUT_FEATURES=[start_mp,end_mp,sum_aadt*1.0/count_aadt,existing_aadt-
sum_aadt*1.0/count_aadt,existing_aadt]
                else:

OUTPUT_FEATURES=[start_mp,end_mp,sum_aadt*1.0/count_aadt,existing_aadt,existing_aadt+sum_
aadt*1.0/count_aadt]
                    print sum_aadt*1.0/count_aadt
                else:
                    if district==4:
                        OUTPUT_FEATURES = [start_mp,end_mp,'NA','NA',existing_aadt]
                    else:
                        OUTPUT_FEATURES = [start_mp,end_mp,'NA',existing_aadt,'NA']
                new_aadt_ws.append(OUTPUT_FEATURES)
                break
    new_aadt_wb.remove(new_aadt_wb['Sheet'])
    new_aadt_wb.save('./RESULTS/'+output_file[:-11]+'AADT.xlsx')
```

```
CODE_MERGE# -*- coding: utf-8 -*-
import os
import openpyxl
filelist = os.listdir('./RESULTS/')
overall_count = 0
error_count = 0
for output_file in filelist:



    if output_file.endswith('OUTPUT.xlsx'):
        print "file opened"
        local_output_file = './RESULTS/'+output_file
        operation_file = output_file[:-11]+'Operation.xlsx'
        aadt_file = output_file[:-11]+'AADT.xlsx'
        local_operation_file = './RESULTS/'+operation_file
        local_aadt_file = './RESULTS/'+aadt_file
       #print output_file,operation_file#,local_output_file,local_operation_file
        exist = os.path.isfile(local_operation_file)
        exist2 = os.path.isfile(local_aadt_file)

        if exist and exist2:
            Final_wb = openpyxl.Workbook()
            output_wb = openpyxl.load_workbook(local_output_file)
            operation_wb = openpyxl.load_workbook(local_operation_file)
            aadt_wb = openpyxl.load_workbook(local_aadt_file)
            sheetnames = output_wb.sheetnames
```

```
for sheet in sheetnames:
    Final_ws = Final_wb.create_sheet(sheet)
    feature_output_file = open('./Feature_List_'+str(sheet)+'.txt','r')
    Feature_List = []
    for line in feature_output_file.readlines():
        Feature_List.append(str(line[:-1]))
    #print Feature_List
    output_ws = output_wb[sheet]
    operation_ws = operation_wb[sheet]
    aadt_ws = aadt_wb[sheet]
    output_feature_count = output_ws.max_column
    output_row_count = output_ws.max_row
    overall_count+=output_row_count
    operation_feature_count = operation_ws.max_column
    operation_row_count = operation_ws.max_row
    aadt_feature_count = aadt_ws.max_column
    aadt_row_count = aadt_ws.max_row
    if output_row_count!=operation_row_count or output_row_count!=aadt_row_count:
        print 'ERROR',output_file,operation_file,aadt_file
        raw_input()
    Feature_Output = []
    Feature_Operation = []
    Feature_AADT = []
    column_count=0
    for i in range(output_feature_count):
        Feature_Output.append(output_ws.cell(row=1,column = i+1).value)
```

```
        for i in range(len(Feature_Output)):
            feature_name = Feature_Output[i]



            if 'G_'+feature_name in Feature_List:
                column_count+=1
                for j in range(output_row_count):

                    Final_ws.cell(row = j+1,column = column_count).value =
output_ws.cell(row=j+1,column = i+1).value



        #error_count=0
        for i in range(operation_feature_count):
            Feature_Operation.append(operation_ws.cell(row = 1,column = i+1).value)
        for i in range(len(Feature_Operation)):
            feature_name = Feature_Operation[i]
            if 'O_'+feature_name in Feature_List:

                #print output_file


                column_count+=1
                for j in range(operation_row_count):
                    Final_ws.cell(row = j+1,column = column_count).value =
operation_ws.cell(row=j+1,column = i+1).value
                    if feature_name=='ALL' and j>0:
                        #print operation_ws.cell(row=j+1,column = i+1).value
```

```
                        error_count+=int(operation_ws.cell(row=j+1,column = i+1).value)

            for i in range(aadt_feature_count):

                Feature_AADT.append(aadt_ws.cell(row =1,column = i+1).value)

            for i in range(len(Feature_AADT)):

                feature_name = Feature_AADT[i]

                if 'A_'+feature_name in Feature_List:

                    #if feature_name=='Total_AADT' and not str(aadt_ws.cell(row = j+1,column =
i+1).value).isdigit():

                    #    print output_file

                    #    error_count+=1

                    column_count+=1

                    for j in range(aadt_row_count):

                        Final_ws.cell(row = j+1,column= column_count).value = aadt_ws.cell(row =
j+1,column = i+1).value

            print output_file,error_count

            feature_output_file.close()

        Final_wb.remove(Final_wb['Sheet'])

        Final_wb.save('./RESULTS/'+output_file[:-11]+'_FINAL.xlsx')


print error_count,'/',overall_count
```

CODE_ACC_SPLIT

```
input_file = open('./HSIS_CA/ca10acc.txt','r')
temp = input_file.readline()
FEATURES = [str(x) for x in temp.split("\t")]
#print FEATURES
PAIR_LIST = []
record_count = 0
for line in input_file.readlines():
    record_count+=1
    temp = [str(x) for x in line.split("\t")]
    #print record_count,",".join(temp)[:-1]
    #route = temp[4]
    route = temp[5]# for 2010
    #route = str(int(temp[1])[])
    #district = temp[3]
    district = temp[4]#for 2010
    pair = [district,route]
    print record_count, pair
    if pair not in PAIR_LIST:
        PAIR_LIST.append(pair)
        vars()['D'+district+'R'+route]
=open("./ACC/DIS"+district+"_RT"+route+"_ACC10.txt",'w')
        print "./ACC/DIS"+district+"_RT"+route+"_ACC14.txt Opened"
        print >> vars()['D'+district+'R'+route],",".join(FEATURES)[:-1]
        print >> vars()['D'+district+'R'+route],",".join(temp)[:-1]
    else:
```

```
        print >> vars()['D'+district+'R'+route],",".join(temp)[:-1]
for pair in PAIR_LIST:
    vars()['D'+district+'R'+route].close()
    print 'D'+district+'R'+route,'closed'
input_file.close()
```

CODE_AVERAGE_RAMP

```
class ramp_record:
    def __init__(self,cntyrte,county,district,rte_nbr,on_offrp,begmp_offset,aadt):
        self.cntyrte = cntyrte
        self.county = county
        self.district = district
        self.rte_nbr = rte_nbr
        self.on_offrp = on_offrp
        self.begmp_offset = begmp_offset
        self.aadt = aadt
import os
import openpyxl
import shutil
filelist = os.listdir('./Raw_Data_II/')
YEARS = [10,11,12,13,14]
for year in YEARS:
    vars()['Ramp_List'+str(year)] = []
    wb = openpyxl.load_workbook('./HSIS_CA/ca'+str(year)+'ramp.xlsx')
    ws = wb['ca'+str(year)+'ramp']
    row_count = ws.max_row
    for i in range(row_count-1):
        cntyrte = ws.cell(row = i+2,column = 1).value
        county = ws.cell(row = i+2,column = 16).value
        district = ws.cell(row = i+2,column = 6).value
        rte_nbr = ws.cell(row = i+2,column = 7).value
        on_offrp  = ws.cell(row = i+2,column = 17).value
        begmp_offset = ws.cell(row = i+2,column = 5).value
        aadt = ws.cell(row = i+2,column = 14).value
        new_record = ramp_record(cntyrte,county,district,rte_nbr,on_offrp,begmp_offset,aadt)
        vars()['Ramp_List'+str(year)].append(new_record)
    print 'Year',year,'finished'
    print len(vars()['Ramp_List'+str(year)])
for input_file in filelist:
    if input_file.startswith('DIS') and input_file.endswith('.xlsx'):
        shutil.copyfile('./Raw_Data_II/'+input_file,'./Raw_Data_III/'+input_file)
        wb = openpyxl.load_workbook('./Raw_Data_III/'+input_file)
        ws = wb['RAMP']
        row_count = ws.max_row
        feature_count = ws.max_column

        for i in range(5):
            ws.cell(row = 1,column = feature_count+1+i).value = 'AADT'+str(2010+i)
            for j in range(row_count-1):
                std_cntyrte = ws.cell(row = j+2,column = 2).value
                std_begmp_offset = ws.cell(row = j+2,column = 6).value
                std_district = ws.cell(row = j+2,column = 7).value
                std_rte_nbt = ws.cell(row = j+2,column = 8).value
```

```
        std_county  =ws.cell(row = j+2,column = 17).value
        std_on_offrp = ws.cell(row = j+2,column = 18).value
        found = False
        for item in vars()['Ramp_List'+str(10+i)]:
            if item.cntyrte ==std_cntyrte \
            and item.begmp_offset ==std_begmp_offset\
            and item.district == std_district\
            and item.county == std_county\
            and item.rte_nbr == std_rte_nbt\
            and item.begmp_offset == std_begmp_offset:
                ws.cell(row = j+2,column = feature_count+1+i).value = item.aadt
                found = True
                break
        #print found
    wb.save('./Raw_Data_III/'+input_file)
    print input_file,'is done'
```

CODE_COMBINE_SEGMENTS

```
def can_be_merge(first_segment,second_segment):
   #'normal'
#    constant_pool = [1,2,3,7,8,9,10,11,12,13,14,17,18,19,20,21,22,23,24,25,26,27,28,29,30,\
#                     31,32,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,\
#                     53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,104,105,106]
   #SCL
   if abs(first_segment[5]-second_segment[4])>0.00001:
      return False
   constant_pool = [1,2,3,7,8,9,10,11,12,13,14,17,18,19,20,21,22,23,24,25,26,27,28,29,30,\
                    31,32,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,\
                    53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,105,106]

   for feature_id in constant_pool:
      if first_segment[feature_id]!=second_segment[feature_id]:

         #print first_segment[4],first_segment[5],second_segment[4],second_segment[5],"not
consistent at",feature_id
         #print len(first_segment),len(second_segment)
         return False
   if not (first_segment[15]=='N/A' or second_segment[15]=='N/A'):
      if not first_segment[15]<=second_segment[15]:
         return False
   if not (first_segment[16]=='N/A' or second_segment[16]=='N/A'):
      if not first_segment[16]>=second_segment[16]:
         return False
   return True
def merged_segment(first_segment,second_segment):
# normal
#    constant_pool = [1,2,3,7,8,9,10,11,12,13,14,17,18,19,20,21,22,23,24,25,26,27,28,29,30,\
#                     31,32,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,\
#                     53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,104,105,106]
#    vector_pool =
[6,33,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,\
#                 97,98,99,100,101,102,103]
    #SCL
   constant_pool = [1,2,3,7,8,9,10,11,12,13,14,17,18,19,20,21,22,23,24,25,26,27,28,29,30,\
                    31,32,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,\
                    53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,105,106]
   vector_pool = [6,33,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,\
                  97,98,99,100,101,102,103,104]
   output_segment = []
   for i in range(len(first_segment)):
      if i==0:
         new_value = first_segment[i]
      if i in constant_pool:
```

```
            new_value = first_segment[i]
        if i in vector_pool:
            new_value = first_segment[i]+second_segment[i]
        if i==15 or i==0:
            new_value = first_segment[i]
        if i==16:
            new_value = second_segment[i]
        if i==4:
            new_value = first_segment[i]
        if i==5:
            new_value = second_segment[i]

        output_segment.append(new_value)
    return(output_segment)

class SEGMENT:
    def __init__(self,district,route,start_mp,end_mp,variables, constants,vectors):
        self.district = district
        self.route = route
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.variables = variables
        self.constants = constants
        self.vectors = vectors
import os
import openpyxl

directory = './RESULTS/'
variable_pool = []
constant_pool = []
vector_pool = []
#sheetnames = ['Normal']
sheetnames = ['SCL_Off','SCL_On']
for file_id in os.listdir(directory):
    if file_id.endswith('_FINAL.xlsx') and file_id.startswith('DIS'):
        print file_id
        WB = openpyxl.load_workbook(directory+file_id)

#       temp_WS = WB['Normal_merged']

#       WB.remove_sheet(temp_WS)
        for sheet_name in sheetnames:
            print sheet_name
            if sheet_name+'_merged' in WB.sheetnames:
                WB.remove_sheet(WB[sheet_name+'_merged'])
            WS = WB[sheet_name]
            feature_titles = []
```

```
feature_count = WS.max_column
observe_count = WS.max_row
if observe_count==1:
   continue
else:
   record_list = []
   observe_count-=1
   for i in range(feature_count):
      new_title = WS.cell(row = 1,column = i+1).value
      feature_titles.append(new_title)


   for i in range(observe_count):

      new_record = []
      for j in range(feature_count):
         temp_value = WS.cell(row = i+2,column = j+1).value
         if j==1:
            district = temp_value
         if j==2:
            route = temp_value
         if j==3:
            start_mp = temp_value
         if j==4:
            start_mp = temp_value
         new_record.append(temp_value)
      #print len(new_record)
      #print new_record[4],new_record[5]
      if len(new_record)!=107:
         print "error",new_record
         raw_input()
      record_list.append(new_record)
   print "there are ",len(record_list),'segments'
   if record_list[0][4]<record_list[0][5]:
      new_list = sorted(record_list,key = lambda x: x[4])
   else:
      new_list = sorted(record_list,key = lambda x: x[4],reverse=True)
   Mergeable = True
   while Mergeable:
      Mergeable = False
      segment_count = len(new_list)
      for i in range(segment_count):
         seg_id = segment_count-i-1
         first_segment = new_list[seg_id-1]
         second_segment = new_list[seg_id]
         if can_be_merge(first_segment,second_segment):
```

```
                    print
first_segment[4],first_segment[5],second_segment[4],second_segment[5]
                    #raw_input()
                    new_segment = merged_segment(first_segment,second_segment)
                    new_list[seg_id-1]=new_segment
                    del new_list[seg_id]
                    Mergeable = True
                    break
            OWS = WB.create_sheet(sheet_name+'_merged')

            for i in range(len(feature_titles)):
               OWS.cell(row = 1,column=i+1).value = feature_titles[i]
            for i in range(len(new_list)):
               OWS.cell(row = i+2,column=1).value = i+1
               for j in range(feature_count-1):
                  OWS.cell(row = i+2,column=j+2).value = new_list[i][j+1]
      WB.save(directory+file_id)
```

CODE_AADT_CHECK

```
import openpyxl
import os
direct = "./RESULTS_Backup/02192020_Backup/"
filelist = os.listdir(direct)
output_file = open(direct+'aadt_check.txt','w')
for result_file in filelist:
    if result_file.endswith('_FINAL.xlsx'):

        WB = openpyxl.load_workbook(direct+result_file)

        ws_list = WB.sheetnames

        for ws_name in ws_list:
            WS = WB[ws_name]
            aadt_find = False
            feature_num = WS.max_column
            observe_num = WS.max_row
            for i in range(feature_num):
                local_name = WS.cell(row = 1,column = i+1).value

                if local_name=='GP_AADT':
                    target_cell = i+1
                    aadt_find = True
                    break
            if not aadt_find:
```

```
            raw_input('ERROR')
        for i in range(observe_num):
            local_aadt = WS.cell(row = i+2,column = target_cell).value
            if local_aadt<20000 and local_aadt!='None':
                mp1 = WS.cell(row = i+2,column = 5).value
                mp2 = WS.cell(row = i+2,column = 6).value
                print result_file,ws_name,mp1,mp2,local_aadt
                print >> output_file,result_file,ws_name,mp1,mp2,local_aadt
    #feature_num =
output_file.close()
```

CODE_STATION_DOWNLOADimport webbrowser

```
import codecs
import time
import os
import shutil
import openpyxl
import re


if not True:


    filelist = os.listdir('./RESULTS/')
    station_list= []
    for input_file in filelist:
        if input_file.endswith('.xlsx') and (input_file.startswith('DIS')):# or input_file.startswith('DIS07')):


            local_input_file = './RESULTS/'+input_file
            print local_input_file
            workbook  = openpyxl.load_workbook(local_input_file)
            sheetnames = ['Normal','SCL_On','SCL_Off']
            for name in sheetnames:
                worksheet = workbook[name]
                observe_count = worksheet.max_row
                if observe_count>1:
                    for i in range(observe_count-1):
                        gp_station = worksheet.cell(row = i+2,column = 63).value
                        hov_station = worksheet.cell(row = i+2,column = 64).value
```

```
                gp_station = str(gp_station)


                hov_station = str(hov_station)
               # if gp_station=='406574' or hov_station =='406574':
                #    print 'find',local_input_file
                   #raw_input()
                if not gp_station in station_list:
                   station_list.append(gp_station)
                if not hov_station in station_list:
                   station_list.append(hov_station)


   print len(station_list)
   station_list_file = open('GP_HOV_Station.list','w')
   for item in station_list:
      print >> station_list_file,item
   station_list_file.close()
   for item in station_list:
      print item
station_list = []
input_station_file = open('./GP_HOV_Station.list','r')
for line in input_station_file.readlines():


   station_list.append(int(line))
input_station_file.close()
#raw_input()
YEARS = [2010,2011,2012,2013,2014]
```

```
#Start_Time_ID = ['1275350400','1306886400','1338508800','1370044800','1401580800']

#Start_Time =
['06%2F01%2F2010+00%3A00','06%2F01%2F2011+00%3A00','06%2F01%2F2012+00%3A00',\

#             '06%2F01%2F2013+00%3A00','06%2F01%2F2014+00%3A00']

#End_Time_ID = ['1283299140','1314835140','1346457540','1377993540','1409529540']

#End_Time =
['08%2F31%2F2010+23%3A59','08%2F31%2F2011+23%3A59','08%2F31%2F2012+23%3A59',\

#           '08%2F31%2F2013+23%3A59','08%2F31%2F2014+23%3A59']

Start_Time_ID = ['1394409600','1402876800','1410739200','1418601600']

Start_Time =
['03%2F10%2F2014+00%3A00','06%2F16%2F2014+00%3A00','09%2F15%2F2014+00%3A00',\

           '12%2F15%2F2014+00%3A00']

End_Time_ID = ['1395014340','1403481540','1411343940', '1419206340']

End_Time =
['03%2F16%2F2014+23%3A59','06%2F22%2F2014+23%3A59','09%2F21%2F2014+23%3A59',\

          '12%2F21%2F2014+23%3A59']

for ii in range(2):

   year = 2014

   i = ii+1

   #if year!=2014:

   #    continue

   start_time_id = Start_Time_ID[i]

   start_time = Start_Time[i]

   end_time_id = End_Time_ID[i]

   end_time = End_Time[i]

   for vds in station_list:

      print vds

      #new_name = './Stations/FLOW_ALL/Flow_'+str(vds)+'_'+str(year)+'.xlsx'
```

```
        #new_name2 = './Stations/Sep_HOV/Operation_'+str(vds)+'.xlsx'

        #if not os.path.exists(new_name):

        if True:

            #url =
"http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=analysis&tab=aadt&export=xls"+\


#"&station_id="+str(vds)+"&s_time_id=1401580800&s_time_id_f=06%2F2014&e_time_id=14016671
40&e_time_id_f=06%2F2014"

            #url =
"http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=analysis&tab=aadt&export=xls"+\

            #
"&station_id="+str(vds)+"&s_time_id=1262304000&s_time_id_f=01%2F2010&e_time_id=141747834
0&e_time_id_f=12%2F2014"

            url =
"http://pems.dot.ca.gov/?report_form=1&dnode=VDS&content=loops&tab=det_timeseries&export=xls"
+\


"&station_id="+str(vds)+"&s_time_id="+start_time_id+"&s_time_id_f="+start_time+\

                        "&e_time_id="+end_time_id+"&e_time_id_f="+end_time+\


"&tod=all&tod_from=0&tod_to=0&dow_0=on&dow_1=on&dow_2=on&dow_3=on"+\


"&dow_4=on&dow_5=on&dow_6=on&holidays=on&q=flow&q2=speed&gn=5min&agg=on&lane1=o
n&lane2=on&lane3=on&lane4=on&lane5=on&lane6=on"

            webbrowser.open(url)

            #new_name = './Stations/Operation_w_Lane/'+'Operation_'+str(vds)+'_w_Lane.xlsx'

            #March
"&station_id="+str(vds)+"&s_time_id=1394496000&s_time_id_f=03%2F11%2F2014+00%3A00&e_ti
me_id=1394755140&e_time_id_f=03%2F13%2F2014+23%3A59"+\

            #June
"&station_id="+str(vds)+"&s_time_id=1402963200&s_time_id_f=06%2F17%2F2014+00%3A00&e_ti
me_id=1403222340&e_time_id_f=06%2F19%2F2014+23%3A59"+\

            local_new_name = './TEMP2/Weekly_'+str(vds)+'_'+str(i)+'.xlsx'

            time.sleep(30)
```

```
        down_sleep_count = 0
        found_download_file = False
        while (down_sleep_count<8 and not found_download_file):
           if os.path.exists("./TEMP2/pems_output.xlsx"):
              found_download_file = True
              shutil.move("./TEMP2/pems_output.xlsx",local_new_name)
              print "Done With",vds


              sleep_count=0
              found_check_file = False


              while (not found_check_file and sleep_count<8):
                 time.sleep(1)
                 sleep_count+=1
                 if os.path.exists(local_new_name):
                    found_check_file = True
                 else:
                    time.sleep(9)
              #       check_workbook = openpyxl.load_workbook(new_name)
              #       check_sheet = check_workbook['PeMS Report Description']
              #       check_seg = check_sheet['C12']
              #       print >> station_output,district,route,seg_id,start_mp,end_mp,PM,str(vds),
"CHECK",check_seg
              #       found_check_file = True
              #    else:
               #       time.sleep(2)
```

```
                #       sleep_count+=1
            else:
                time.sleep(10)
down_sleep_count+=1
```

CODE_ADD_DIRECTION

```
import os
#route_list =
[4,101,237,280,580,680,80,84,85,87,880,92,105,110,118,134,14,170,210,57,60,10,215,71,91,15]
route_list = [10,15]

filelist = os.listdir('./')
for input_file in filelist:
   if input_file.startswith('DIS')and input_file.endswith('.txt'):
      print input_file
      district = input_file.split('_')[0][3:]
      route = input_file.split('_')[1][2:]
      if (not int(route) in route_list) or (not int(district) in [8]):
         continue
      local_input_file = './'+input_file
      Input_File = open(local_input_file,'r')
      if os.path.isfile('./REV_'+input_file):
         continue
      Output_File =open( './REV_'+input_file,'w')
      line = Input_File.readline()
      temp = [str(x) for x in line.split(',')]
      caseno_id = 0
      for i in range(len(temp)):
         item = temp[i]

         if item.lower()=='caseno':
            caseno_id=i
            break
      #print caseno_id,input_file
      print >>Output_File,line[:-1]+',direction'
      for line in Input_File.readlines():
         temp = [str(x) for x in line.split(',')]
         temp[-1] = temp[-1][:-1]
         #print temp
         #print caseno_id
         caseno = temp[caseno_id]
         if len(caseno)<10:
            print caseno,input_file
            continue
         datetime = caseno[:8]
         #print datetime
         vehicle_input = open('../VEH/'+datetime+'.txt','r')
         first_line = vehicle_input.readline()
         veh_temp = [str(x) for x in first_line.split(',')]

         incident_dir = ''
```

```
    direction_id = -1
    for i in range(len(veh_temp)):
        item = veh_temp[i]

        if item.lower()=='dir_trvl':
            direction_id = i
            break
    if direction_id==-1:
        print veh_temp
        raw_input()
    for veh_line in vehicle_input.readlines():
        veh_temp=[str(x) for x in veh_line.split(',')]
        if veh_temp[0]==caseno:
            incident_dir = veh_temp[direction_id]
            if not incident_dir in ['N','S','E','W']:

                print direction_id,veh_temp
                break
    vehicle_input.close()
    temp.append(incident_dir)
    print temp
    print >>Output_File,','.join(temp)

Output_File.close()
Input_File.close()
```

CODE_DIRECTION_CHECK

```
import os

filelist = os.listdir('./')
DIRECTIONS = ['E','W','N','S']
for input_file in filelist:
   if input_file.startswith('REV')and input_file.endswith('.txt'):
      local_input_file = './'+input_file
      input_rev = open(local_input_file,'r')
      input_rev.readline()
      error_count=0
      incident_count = 0
      for line in input_rev.readlines():
         temp = [str(x) for x in line.split(',')]
         milepost = str(temp[1])
         incident_count+=1
         if milepost.isdigit():

            error_count+=1
            #print direction
      if 1.0*error_count/incident_count>0.1:
         print local_input_file,error_count,incident_count
input_rev.close()
```

CODE_HOURLY_FLOW

```
class aadt_record:
    def __init__(self,station_id,aadt10,aadt11,aadt12,aadt13,aadt14):
        self.station_id = station_id
        self.aadt10 = aadt10
        self.aadt11 = aadt11
        self.aadt12 = aadt12
        self.aadt13 = aadt13
        self.aadt14 = aadt14
def HOURLY_ESTIMATE(station_id,year):
    if year=='2010':
        print station_id,year
    wb_op =
openpyxl.load_workbook('../FLOW_ALL/Flow_'+str(station_id)+'_'+str(year)+'.xlsx')
    ws_op = wb_op['Report Data']
    op_feature_count = ws_op.max_column
    op_row_count = ws_op.max_row

    for j in range(op_feature_count):
        #print ws_op.cell(row = 1,column = j+1).value,station_id,year

        if ws_op.cell(row = 1,column = j+1).value =='Flow (Veh/5 Minutes)':

            op_target_col = j+1
            break
    for i in range(24):
        vars()['hour'+str(i)] = []
    VOL_RECORD = []
    for i in range(op_row_count-1):
        vol_per_5 = ws_op.cell(row = i+2,column = op_target_col).value
        datetime = str(ws_op.cell(row = i+2,column  =1).value)
        temp = [str(x) for x in datetime.split()]
        temp1 = [str(x) for x in temp[1].split(':')]

        hour = int(temp1[0])
        if vol_per_5!=None:
            vars()['hour'+str(hour)].append(int(vol_per_5))
    for i in range(24):
        if len(vars()['hour'+str(i)])!=0:
            VOL_RECORD.append(sum(vars()['hour'+str(i)])/len(vars()['hour'+str(i)])*12)
        else:
            VOL_RECORD.append(0)
    return(VOL_RECORD)

import os
import shutil
```

```
import openpyxl
filelist = os.listdir('./')
AADT_RECORD = []
YEARS = [2010,2011,2012,2013,2014]


STATION_LIST= []
for input_file in filelist:
   if input_file.startswith('Flow_') and input_file.endswith('.xlsx'):
      station_id = input_file.split('.')[0].split('_')[1]
      if not station_id in STATION_LIST:
         STATION_LIST.append(station_id)
for station_id in STATION_LIST:
   output_file=open('../HOURLY_VOL/HOURLY_VOL_'+station_id+'.txt','w')
   for year in YEARS:


      record_list = HOURLY_ESTIMATE(station_id,str(year))
      if len(record_list)!=24:
         print station_id,year,'missing hourly data'
         raw_input()
      else:
         for i in range(24):
            print >> output_file,station_id,year,i,record_list[i]

   output_file.close()
```

```
class Washington_Loop_Record:
   def __init__(self,route,milepost,direction,month,records):
      self.route = route
      self.milepost = milepost
      self.direction = direction
      self.month = month
      self.records = records

def FOUND_LOOP(loop_list,route,milepost,direction,month):
   for item in loop_list:
      if item.route ==route and item.milepost==milepost and item.direction==direction\
         and item.month ==month:
            return(item)
   print 'ERROR'
   exit()



YEARS = [2014]
LOOP_LIST= []
ALL_RECORDS = []
MONTHS= ['03','06','09','12']
ROUTE_LIST = []
for year in YEARS:
   input_file_name = "../Loop_data_"+str(year)+"_for_NCHRP.csv"
   input_file = open(input_file_name,'r')
   input_file.readline()
   for line in input_file.readlines():
      print line
      if not ',' in line:
         route,milepost,direction,date,time,speed, volumn = [str(x) for x in line.split()]

      else:
         route,milepost,direction,time,speed, volumn = [str(x) for x in line.split(',')]

         date,time = time.split(' ')
      route = str(int(route))
      milepost = float(milepost)
      speed = float(speed)
      volumn = int(volumn)
      #print date,' time,',time
      temp,month,day = date.split('-')
      hour,minute,second = time.split(':')
```

```
        #print route,milepost, direction, month,day,hour,minute, speed,volumn
        new_record = [route,milepost,direction,month]
        new_route = [route,direction,month]
        #print route+direction+str(year)+str(month)
        if not new_route in ROUTE_LIST:
            ROUTE_LIST.append(new_route)


            vars()[route+direction+str(year)+str(month)] =
open('../RT_'+str(route)+'_'+direction+'_'+str(year)+'_'+str(month)+'.txt','w')
            print >>
vars()[route+direction+str(year)+str(month)],route,direction,milepost,month,day,hour,minute,speed,volu
mn
        else:
            #print 'test'
            print >>
vars()[route+direction+str(year)+str(month)],route,direction,milepost,month,day,hour,minute,speed,volu
mn
            vars()[route+direction+str(year)+str(month)].flush()
        if not new_record in LOOP_LIST:
            LOOP_LIST.append(new_record)
            new_loop = Washington_Loop_Record(route,milepost,direction,month,[])
            #print 'New monthly record detected',route,milepost,direction,month
            new_loop.records.append([day,hour,minute,speed,volumn])
            ALL_RECORDS.append(new_loop)
        else:
            new_loop = FOUND_LOOP(ALL_RECORDS,route,milepost,direction,month)
            new_loop.records.append([day,hour,minute,speed,volumn])

    input_file.close()
```

CODE_MERGE_HSIS

```
class historical_record:
    def __init__(self,district,route,start_mp,end_mp,aadt):
        self.district = district
        self.route = route
        self.start_mp = start_mp
        self.end_mp = end_mp
        self.aadt = aadt

YEARS = [10,11,12,13]
STD_RECORD = []

input_stand = open('./ca14road.txt','r')
line = input_stand.readline()
features = [str(x) for x in line.split('\t')]
for i in range(len(features)):
    print i,features[i]
features[-1]=features[-1][:-1]
features.append('aadt2010')
features.append('aadt2011')
features.append('aadt2012')
features.append('aadt2013')
std_record_count = 0
for line in input_stand.readlines():
    temp = [str(x) for x in line.split('\t')]
    temp[-1] = temp[-1][:-1]
    #print temp[23]

    STD_RECORD.append(temp)
input_stand.close()
#for item in STD_RECORD:
#    print item[23]
print "Standard AADT Read-in"
print features
print len(features)
raw_input()
#print std_record_count
for year in YEARS:
    print "PROCESSING",year
    RECORD_ANNUAL = []
    input_file = open('./ca'+str(year)+'road.txt','r')
    line = input_file.readline()
    features = [str(x) for x in line.split()]
    #print features
    #print len(features)
```

```
    for line in input_file.readlines():
        temp = [str(x) for x in line.split('\t')]

       # print temp
        #print temp
        #print 'Year',year,temp[7],temp[8],temp[2],temp[3],temp[23]
        start_mp = float(temp[2])
        end_mp = float(temp[3])
        district = int(temp[7])

        route = int(temp[8])
        if temp[23]!='':
            aadt = int(temp[23])
        else:
            aadt=0

        new_record  = historical_record(district,route,start_mp,end_mp,aadt)
        RECORD_ANNUAL.append(new_record)
    input_file.close()
    for item in STD_RECORD:
        std_district = int(item[7])
        std_route = int(item[8])
        std_start_mp = float(item[2])
        std_end_mp = float(item[3])
        if item[23]!='':
            std_aadt = int(temp[23])
        else:
            std_aadt = 0
        found = False
        for record in RECORD_ANNUAL:
            if record.district==std_district and record.route==std_route and
record.start_mp==std_start_mp\
            and record.end_mp==std_end_mp:
                item.append(record.aadt)
                found = True
                break
        if not found:
            item.append(0)
empty_count = 0

print empty_count
output_file = open('./Processed_ca_road.txt','w')
print >> output_file,','.join(features)
for item in STD_RECORD:

    print >> output_file,','.join((str(x) for x in item))
output_file.close()
```