Report - Lab 10

Aarushi Karnany
Dhanusha Varik
Group ID - 27

1. Approach

We were given individual files for timestamp, flow, speed, occupancy and probability for each zone. We used different files depending on the approach we were doing.

For Method 1 (Linear Regression) we created measurement vectors = {flow, probability} for all the rows in the files and for each detector(column). We also calculated the median of flow values, row wise as they represent flow captured at the same time in different lanes. We train the Linear Regression model with flow values for all lanes as input and median as ground truth using Scikit's Linear Regression function. To predict flow values for one lane, we used the nearby lane(s) as testing data. If a lane had only one nearby lane, then that was used. Else, we took the average of both the nearby lanes (on left and right) as testing data. The same process was done to predict the confidence of the value with probability values. The confidence and predicted flow values were then saved in a tsv file, to be used later. The predicted flow values were stored as float, and then rounded off to int later after the final merging.

For Method 2 (Nearby timestamps) we calculated predicted flow value for each lane using weighted sum of previous and next timestamp flow measurement for that lane as:

Predicted(flow) = w1*Preceding_Measured(flow) + w2*Following_Measured(flow)

Here the weights were calculated using the probability density values of that particular timestamp's flow. If there was no previous/next timestamp, or if the timestamps were more than 10 minutes apart from the current timestamp, we gave them a weight of zero in the above formula so as to not consider them.
The confidence of the predicted value was taken as min of previous and next flow's probability density value.

For Method 3 we keep the flow measurements as they are, without changing them. So the predicted flow value was the flow value column in measurement vector and confidence was the probability column.

Now, to merge all three methods together, we used the weighted sum of all three methods as follows -

Merged(Flow) = w1*Pred_1(flow) + w2*Pred_2(flow) + w3*Pred_3(flow)

where w1,w2,w3 are weights defined as and c1,c2,c3 are confidence values from part1,part2 and part3 respectively.

2. Intuitions

For Linear Regression we used median flow value for a particular timestamp as ground truth because it doesn't get affected by the outliers, while mean does. We also dropped the NA flow values during training, so that it doesn't affect our model.

For the nearby timestamps, a weight of zero was assigned to previous or next value if timestamp was more than 10 minutes apart from current timestamp as we assumed it should not contribute in predicting the current timestamp since it was recorded more than 10 minutes after or before the current flow's timestamp.

In method3, we assigned a weight of zero to the flows that had NA values, since we did not want to consider them in the weighted sum of all three methods and hence we got a predicted value for these NA flows as weighted sum of methods 1 and 2. So after combining all the 3 method's predicted flows, we kept the predicted flows for the NA flow values in the output file. We did not replace this by empty strings, since our methods predicted the flow for these values using nearby lanes/ nearby timestamps (method 1 and 2).

3. Implementation Steps

For Method 1 (Linear Regression) we created measurement vectors = {flow, probability} for all the rows in the files and for each detector(column). We also calculated the median of flow values, row wise as they represent flow captured at the same time in different lanes. We appended the median as a new column to our measurement vector. We train the Linear Regression model with flow values for all lanes as input and median as ground truth using Scikit's Linear Regression function. We dropped NA flow values while training. To predict flow values for one lane, we used the nearby lane(s) as testing data. If a lane had only one nearby lane, then that was used. Else, we took the average of both the nearby lanes (on left and right) as testing data. The same process was done to predict the confidence of the value with probability values. The confidence and predicted flow values were then saved in a tsv file, to be used later. The predicted flow values were stored as float as of now, and rounded off later after the final merging.

For Method 2 (Nearby timestamps) we calculated predicted flow value for each lane using weighted sum of previous and next timestamp flow measurement. We created a vector for each lane's flow value and then shifted it up by one to get next flow and shifted it down by one to get next preceding flow and saved these 3 columns in a data frame. Similar shifting was done to get timestamp values for the preceding and next flow values. We did this instead of iterating row-wise since it is much faster and pandas shifts columns in a few seconds whereas manually iterating through the rows to get next and previous values using a loop was taking hours. We then calculated difference between prev timestamp column and current timestamp column and if it was more than 10 mins we assigned weight of zero to that timestamp's flow. Similar for next timestamp column. The confidence of the predicted value was taken as min of previous and next flow's probability density value.

For Method 3, no change was required in the flow and probability values. They were used as it is for predicted flow and confidence.

For merging all the methods together, we read the values from Method 1, Method 2, and Method 3 and use the formula as above. While calculating weights, we found that some predicted probabilities were 0 for all three methods. So we assigned a weight 0 to that value. We calculated the predicted flow value and rounded it off to int and stored it column wise for each lane, as the given input files (flow.tsv etc.)

All of this was done using 2 python files which were then called from a shell script with arguments to the python files being path to the zone folder and the zone number. First python file cimputed method1 and 2's flow values and dumped them in 2 tsv files. Second python file read original flow values and these 2 tsv files created by 1st python script and calculated final predicted flow values. The shell script took the path to the zone folder as an argument, then extracted the zone number and called the 2 python files. The 2nd python file output is a zone_number.flow.txt file in the current directory.

4. Challenges

The first challenge we faced included figuring out what ground truth should be used to train the Linear Regression model. We first thought to use the mean but then decided median was better, since mean is influenced by outliers while median is not. Another thing we had to decide was the nearby lane while predicting the value. One choice was to just consider the next lane, if it existed. But it seemed more reasonable to us to average out the left and right lanes both, if they existed. This was because both the lanes were nearby and would affect the flow for the lane in consideration.

For nearby timestamps method we initially iterated through all the rows to calculate the difference between previous and next but this way was taking hours since there are millions of rows. We then used panda's shift method on the timestamp and flow columns and we able to perform the calculations on entire columns instead of iterating through each row, and this way took just a few seconds.

During the merging step, we were getting memory issues as initially we had written one python script to perform all 3 methods computations. We then decided to split this into 2 python scripts. First one to compute and write the output of methods 1 and 2 into tsv files, and second python script reads those tsv files and computes final predicted flow.

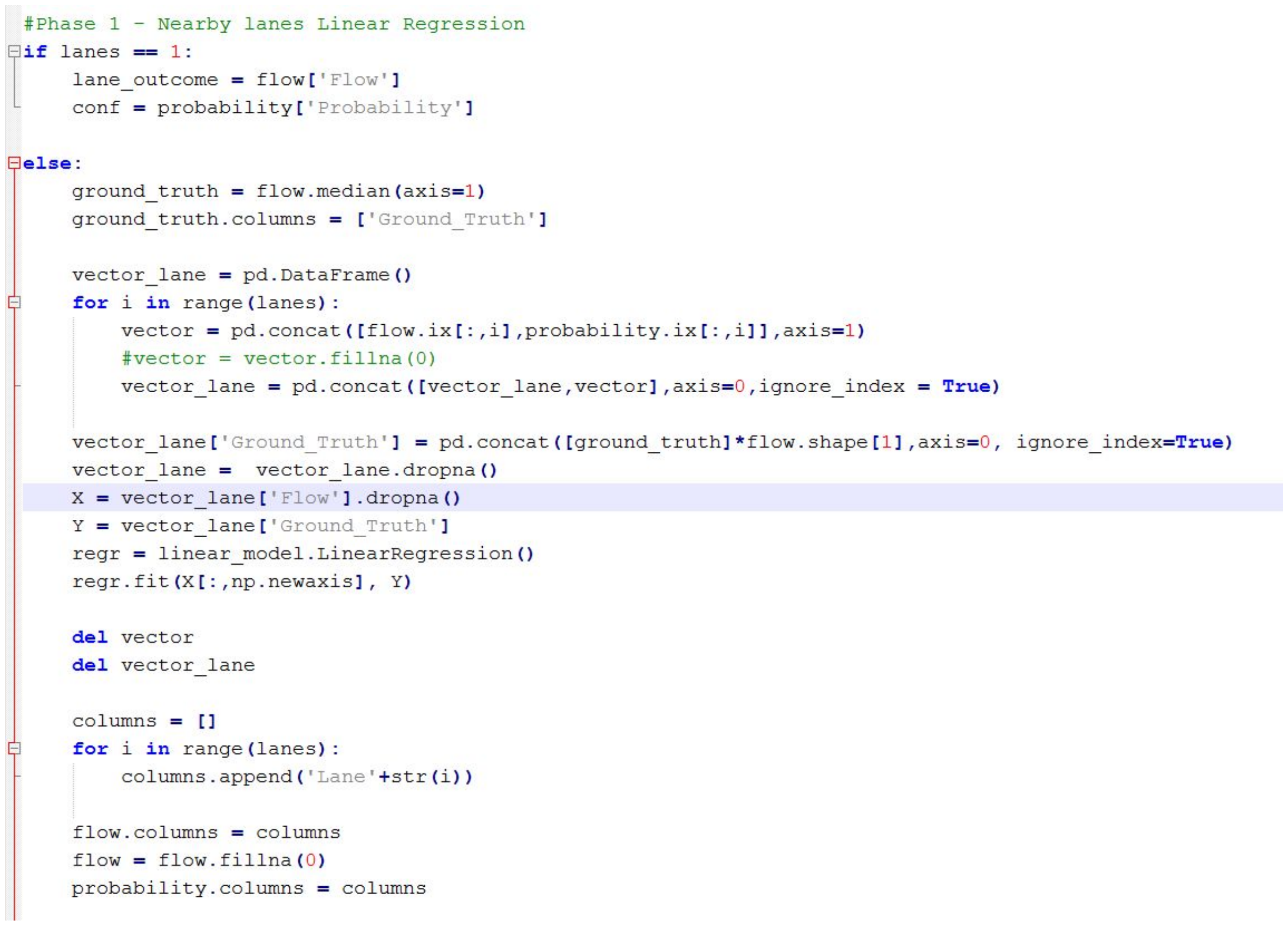
5. Self-Evaluation and Improvements in Final Result

For Linear Regression model we first thought to use the mean as ground truth, but then decided median was better, since mean is influenced by outliers while median is not. Another thing we had to decide was which nearby lane to use while predicting the flow value. One choice was to just consider the next lane, if it existed. But it seemed more reasonable to us to average out the left and right lanes flow values both, if they existed. This was because both the lanes were nearby and would better affect the flow for the lane in consideration.

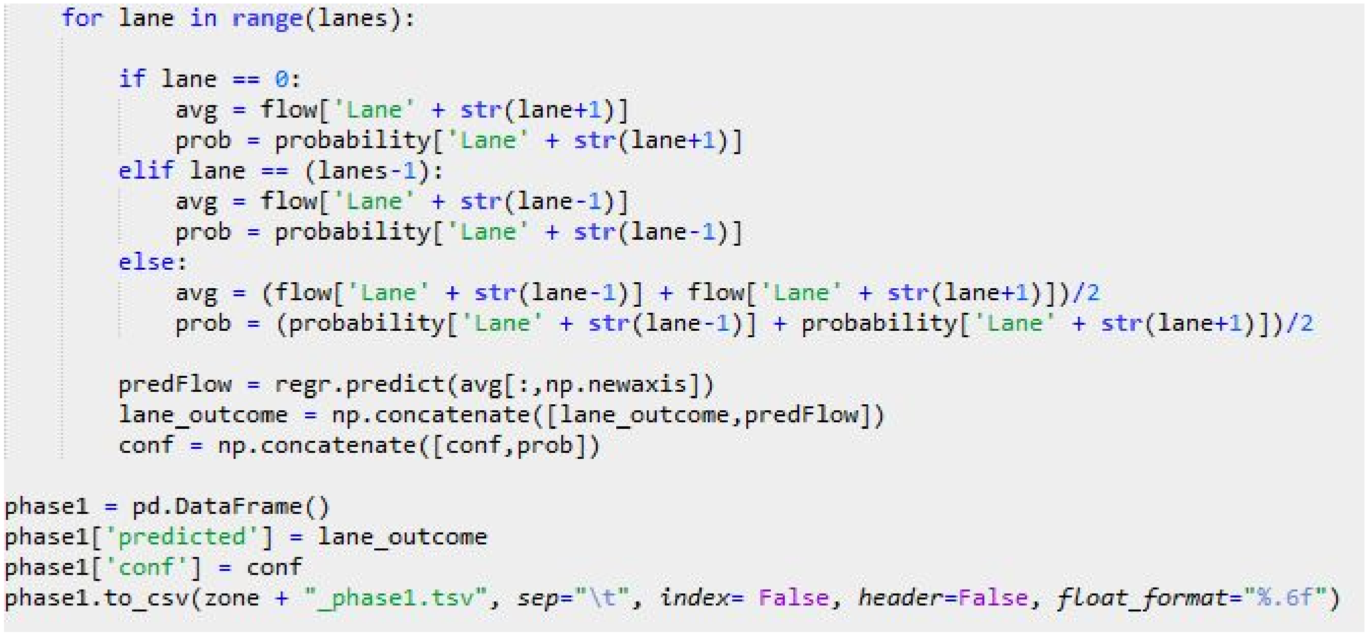
We also optimised our code so that it runs faster and takes less memory. The total time taken to run the entire code on any zone and get the output predicted flow text file is around 5-7 minutes.

Slides describing the 3 Methods to get predicted flow values-:

1. **Method 1 - Linear Regression of nearby lanes**

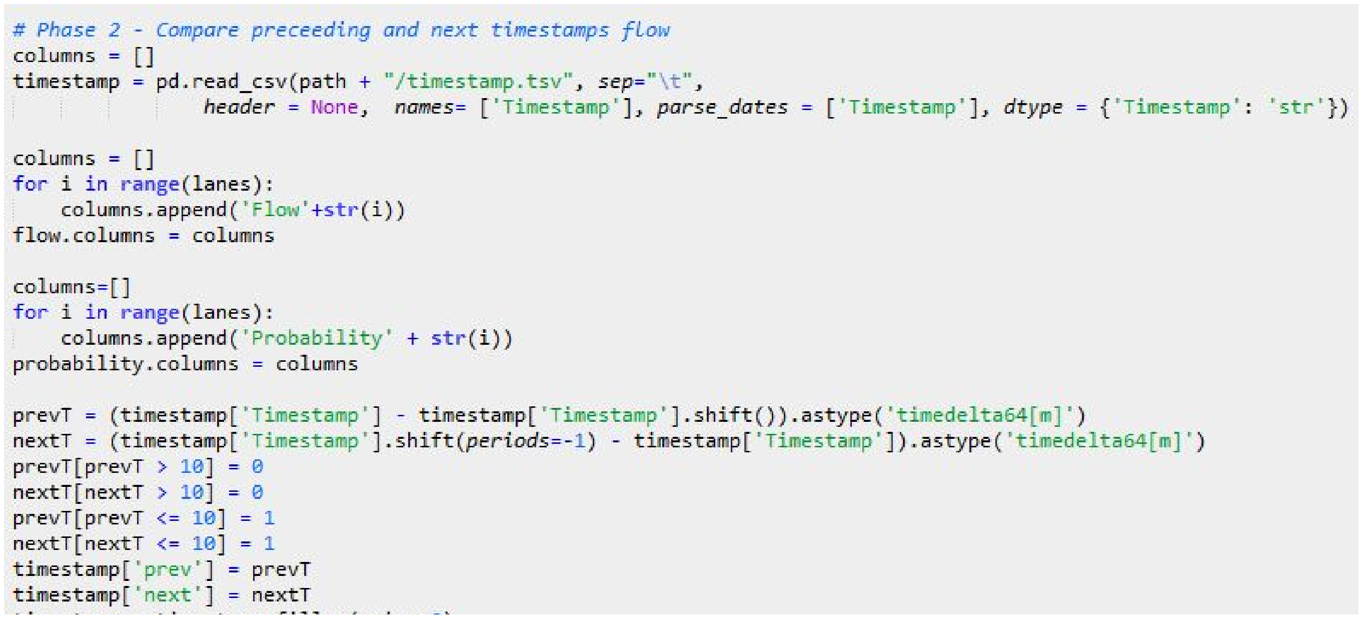


```
#Phase 1 - Nearby lanes Linear Regression
if lanes == 1:
    lane_outcome = flow['Flow']
    conf = probability['Probability']

else:
    ground_truth = flow.median(axis=1)
    ground_truth.columns = ['Ground_Truth']

    vector_lane = pd.DataFrame()
    for i in range(lanes):
        vector = pd.concat([flow.ix[:,i],probability.ix[:,i]],axis=1)
        #vector = vector.fillna(0)
        vector_lane = pd.concat([vector_lane,vector],axis=0,ignore_index = True)

    vector_lane['Ground_Truth'] = pd.concat([ground_truth]*flow.shape[1],axis=0, ignore_index=True)
    vector_lane =  vector_lane.dropna()
    X = vector_lane['Flow'].dropna()
    Y = vector_lane['Ground_Truth']
    regr = linear_model.LinearRegression()
    regr.fit(X[:,np.newaxis], Y)

    del vector
    del vector_lane

    columns = []
    for i in range(lanes):
        columns.append('Lane'+str(i))

    flow.columns = columns
    flow = flow.fillna(0)
    probability.columns = columns
```

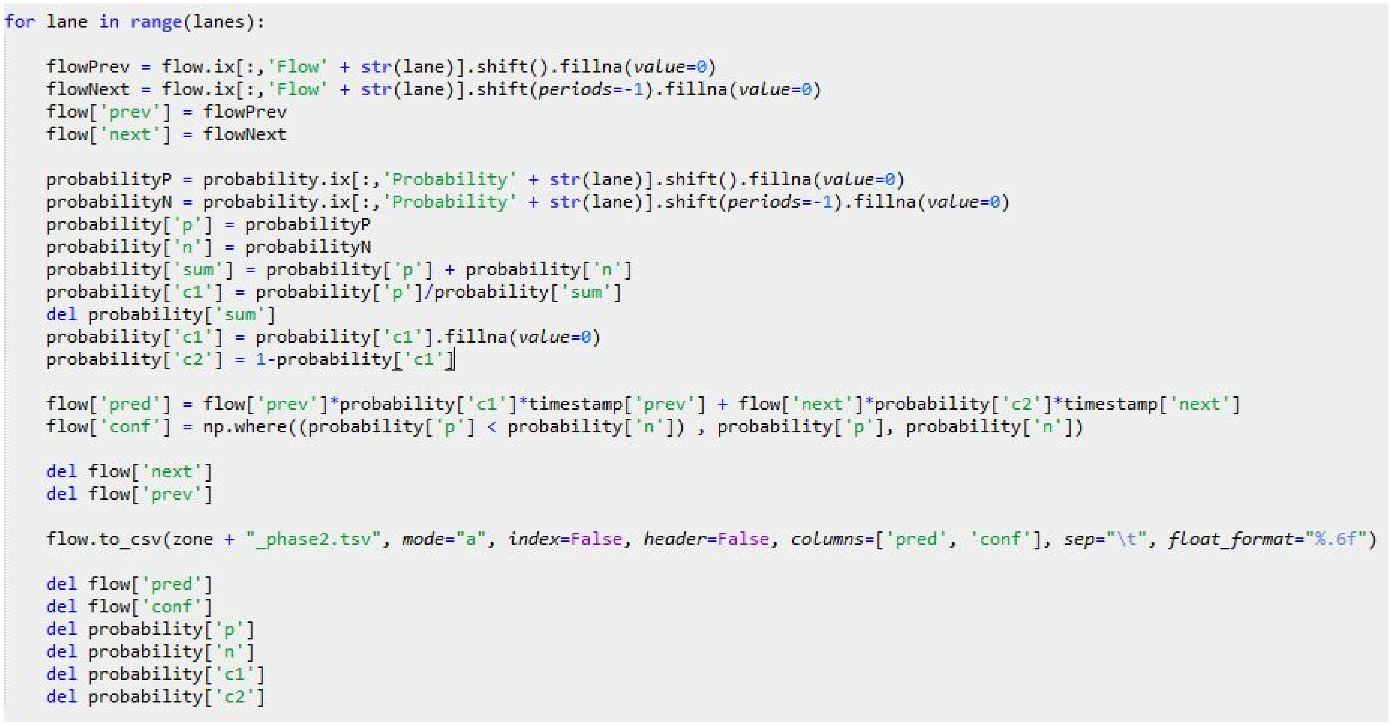


```python
    for lane in range(lanes):

        if lane == 0:
            avg = flow['Lane' + str(lane+1)]
            prob = probability['Lane' + str(lane+1)]
        elif lane == (lanes-1):
            avg = flow['Lane' + str(lane-1)]
            prob = probability['Lane' + str(lane-1)]
        else:
            avg = (flow['Lane' + str(lane-1)] + flow['Lane' + str(lane+1)])/2
            prob = (probability['Lane' + str(lane-1)] + probability['Lane' + str(lane+1)])/2

        predFlow = regr.predict(avg[:,np.newaxis])
        lane_outcome = np.concatenate([lane_outcome,predFlow])
        conf = np.concatenate([conf,prob])

phase1 = pd.DataFrame()
phase1['predicted'] = lane_outcome
phase1['conf'] = conf
phase1.to_csv(zone + "_phase1.tsv", sep="\t", index= False, header=False, float_format="%.6f")
```

2. **Method 2 - Nearby Timestamps**

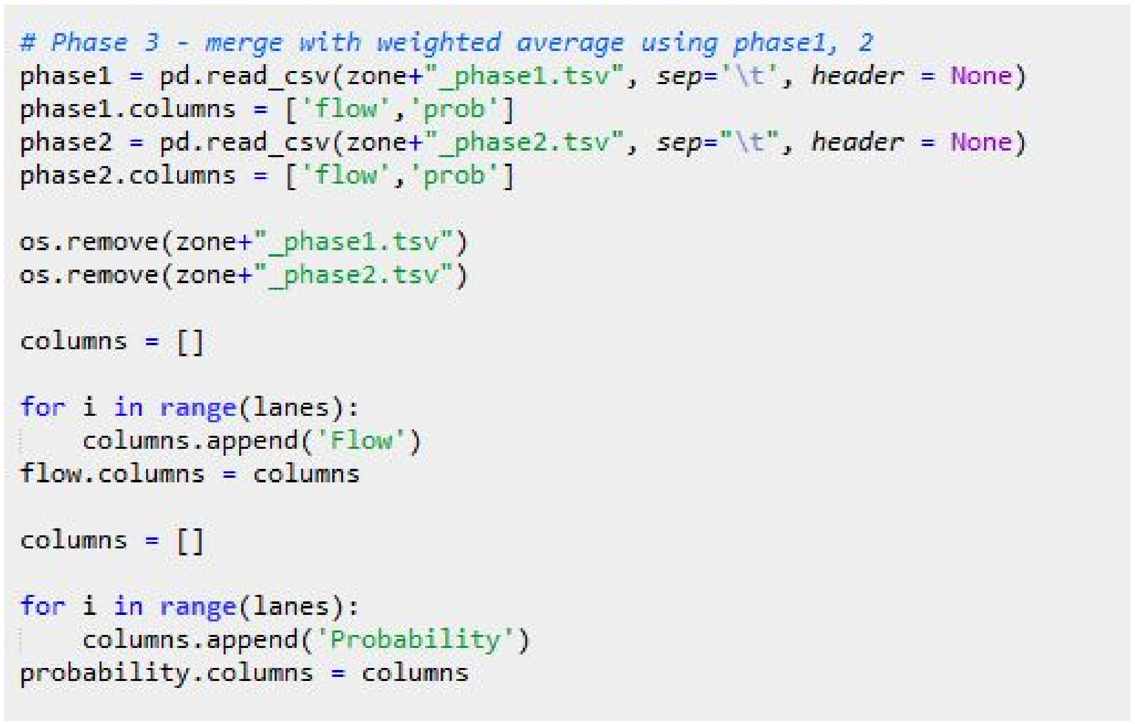


```python
# Phase 2 - Compare preceeding and next timestamps flow
columns = []
timestamp = pd.read_csv(path + "/timestamp.tsv", sep="\t",
                        header = None,  names= ['Timestamp'], parse_dates = ['Timestamp'], dtype = {'Timestamp': 'str'})

columns = []
for i in range(lanes):
    columns.append('Flow'+str(i))
flow.columns = columns

columns=[]
for i in range(lanes):
    columns.append('Probability' + str(i))
probability.columns = columns

prevT = (timestamp['Timestamp'] - timestamp['Timestamp'].shift()).astype('timedelta64[m]')
nextT = (timestamp['Timestamp'].shift(periods=-1) - timestamp['Timestamp']).astype('timedelta64[m]')
prevT[prevT > 10] = 0
nextT[nextT > 10] = 0
prevT[prevT <= 10] = 1
nextT[nextT <= 10] = 1
timestamp['prev'] = prevT
timestamp['next'] = nextT
```



```
for lane in range(lanes):

    flowPrev = flow.ix[:,'Flow' + str(lane)].shift().fillna(value=0)
    flowNext = flow.ix[:,'Flow' + str(lane)].shift(periods=-1).fillna(value=0)
    flow['prev'] = flowPrev
    flow['next'] = flowNext

    probabilityP = probability.ix[:,'Probability' + str(lane)].shift().fillna(value=0)
    probabilityN = probability.ix[:,'Probability' + str(lane)].shift(periods=-1).fillna(value=0)
    probability['p'] = probabilityP
    probability['n'] = probabilityN
    probability['sum'] = probability['p'] + probability['n']
    probability['c1'] = probability['p']/probability['sum']
    del probability['sum']
    probability['c1'] = probability['c1'].fillna(value=0)
    probability['c2'] = 1-probability['c1']

    flow['pred'] = flow['prev']*probability['c1']*timestamp['prev'] + flow['next']*probability['c2']*timestamp['next']
    flow['conf'] = np.where((probability['p'] < probability['n']) , probability['p'], probability['n'])

    del flow['next']
    del flow['prev']

    flow.to_csv(zone + "_phase2.tsv", mode="a", index=False, header=False, columns=['pred', 'conf'], sep="\t", float_format="%.6f")

    del flow['pred']
    del flow['conf']
    del probability['p']
    del probability['n']
    del probability['c1']
    del probability['c2']
```

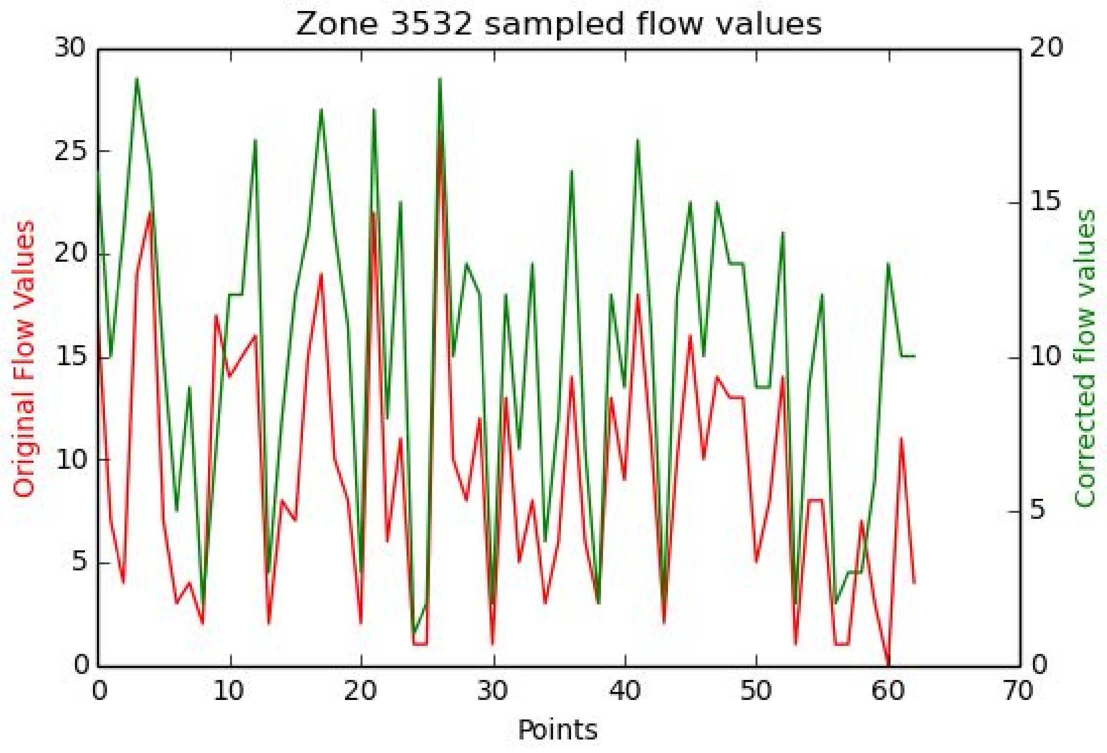
3. Method3 + Merging all the methods together

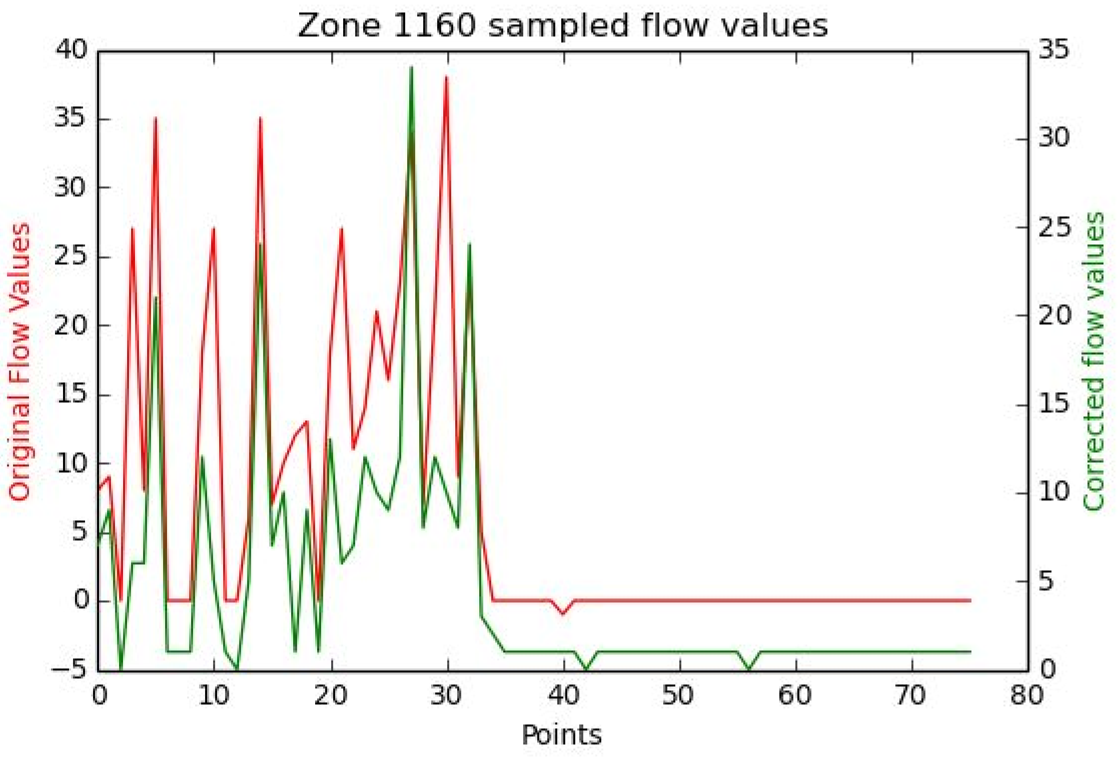


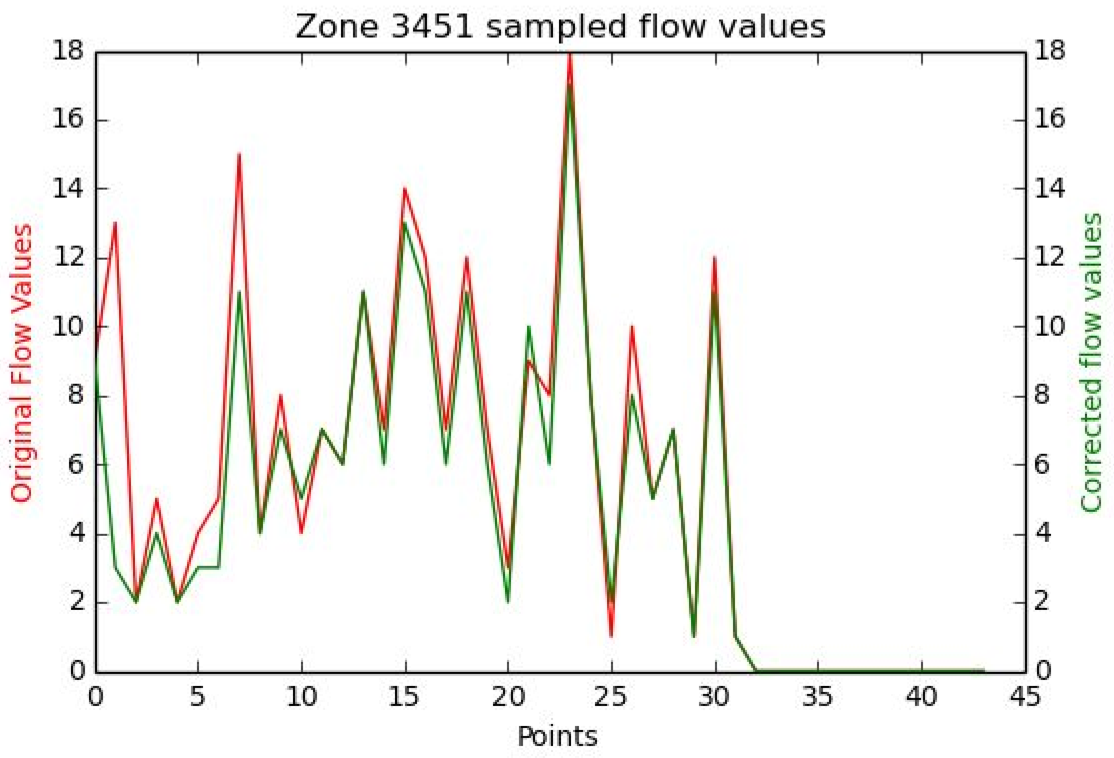
```
# Phase 3 - merge with weighted average using phase1, 2
phase1 = pd.read_csv(zone+"_phase1.tsv", sep='\t', header = None)
phase1.columns = ['flow','prob']
phase2 = pd.read_csv(zone+"_phase2.tsv", sep="\t", header = None)
phase2.columns = ['flow','prob']

os.remove(zone+"_phase1.tsv")
os.remove(zone+"_phase2.tsv")

columns = []

for i in range(lanes):
    columns.append('Flow')
flow.columns = columns

columns = []

for i in range(lanes):
    columns.append('Probability')
probability.columns = columns
```

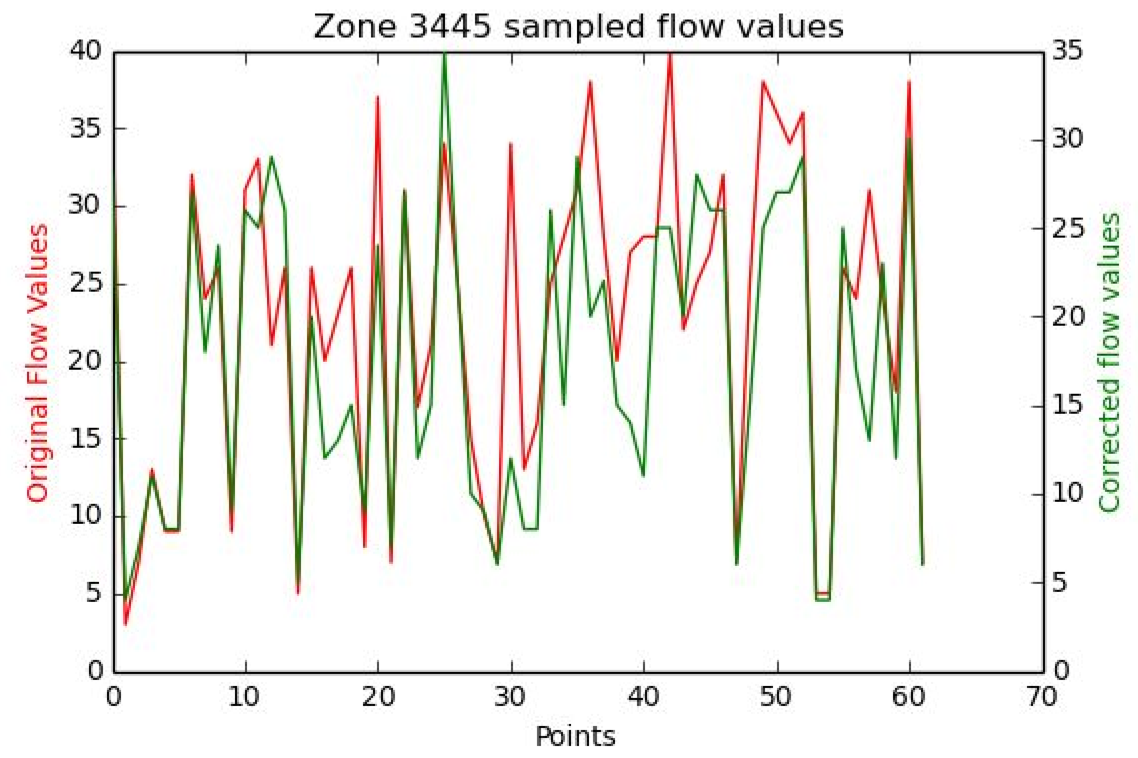
```
phase1['w'] = phase1['prob']/(phase1['prob'] + phase2['prob'] + phase3['Probability'])
phase1['w'] = phase1['w'].fillna(0)
phase2['w'] = phase2['prob']/(phase1['prob'] + phase2['prob'] + phase3['Probability'])
phase2['w'] = phase2['w'].fillna(0)
phase3['w'] = 1 - (phase1['w'] + phase2['w'])

merged = pd.DataFrame()
merged['Merged_Flow'] = phase1['w']*phase1['flow']
del phase1
merged['Merged_Flow'] = merged['Merged_Flow'] + phase2['w']*phase2['flow']
del phase2
merged['Merged_Flow'] = merged['Merged_Flow'] + phase3['w']*phase3['Flow']
del phase3

final = pd.DataFrame()
ctr = 0
for i in range(lanes):
    final = pd.concat([final,merged.ix[ctr:ctr+(length-1),:].reset_index(drop=True)],axis=1,ignore_index=True)
    ctr = ctr + (length)
final = final.astype(int)

del merged
final.to_csv(zone + ".flow.txt", sep="\t", index = False, header=False)
```

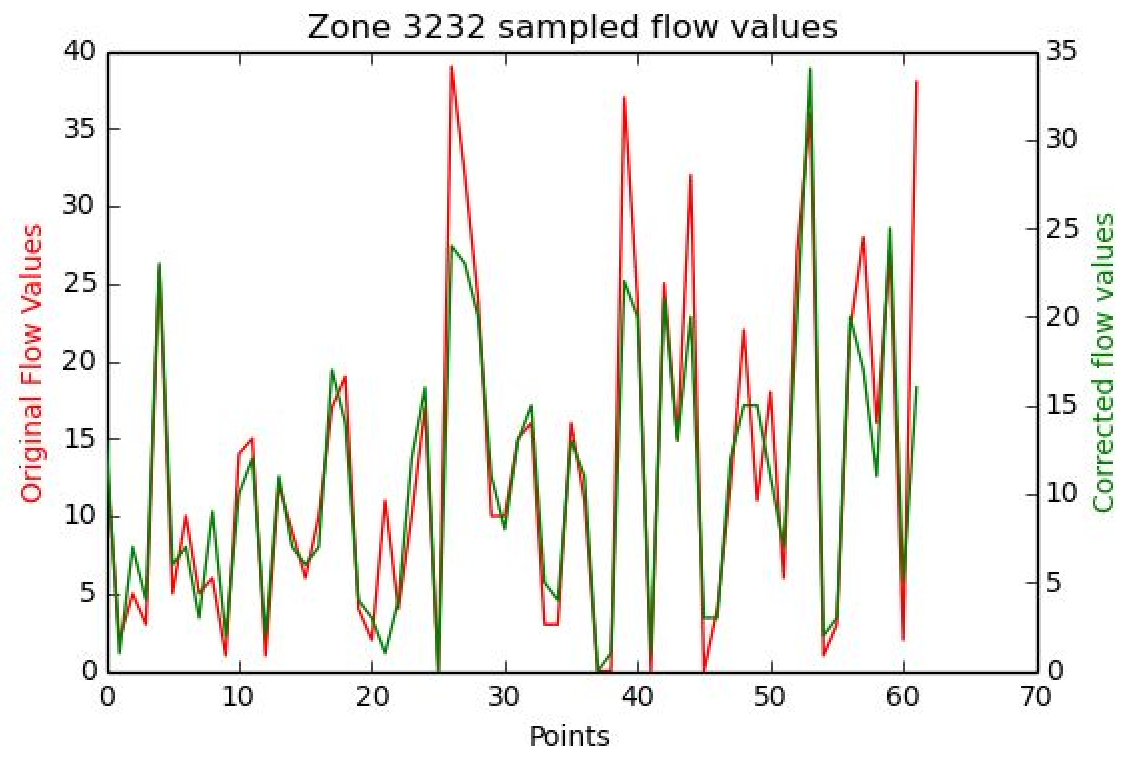
Plots of original flow v/s corrected ones for each zone:




Zone 1160 sampled flow values
Original Flow Values
Corrected flow values
Points


Zone 3451 sampled flow values
Original Flow Values
Corrected flow values
Points


Zone 3445 sampled flow values
Original Flow Values
Corrected flow values
Points
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30
35
0
10
20
30
40
50
60
70


Zone 3232 sampled flow values
Original Flow Values
Corrected flow values
Points
0
5
10
15
20
25
30
35
40
0
5
10
15
20
25
30
35
0
10
20
30
40
50
60
70