INF 264 Project 2: Traffic predictor

Ketil Fagerli Iversen

October 2020

0.1 The functions i need

Onehot encoder functions and helper functions:

The Jupyter notebook contains all of the helpful functions that i used for this project. If i were to have a figure for each one including comments, this document would have far too many pages. So i will just explain shortly the input and output, and what each function does instead.

easter_sunday(year)
NOTE: The **easter_sunday()** function is based on the algorithm known as the Meeus / Jones / Butcher algorithm. It is thus **NOT** something i came up with on my own and all credit goes to it's original creator. This function **easter_sunday()** takes a year as input, and outputs the month and the day of when Easter Sunday occurs in that year. This is used to determine when the easter holiday occurs as it starts about 7 days before Easter Sunday and lasts about til the day after. The reason i need this is that Easter starts on different dates depending on the year, and the formula to compute this is quite complicated. The source for the algorithm is near the bottom of the following article;

https://en.wikipedia.org/wiki/Computus

I could unfortunately not find a library similar to the datetime library that would do this for me. Hopefully this will not be regarded as an attempt at cheating as it is a known algorithm which is not directly connected to this project other than the fact i wanted to include Easter as a holiday for my onehot encoder.

is_date_holiday(X_row)
This function takes as input a single sample of feature data, and outputs wether or not the sample corresponds to a holiday. It outputs either **False** if the sample is not in a holiday, or the name of the current holiday as a string.

is_date_weekend(X_row)
This function takes as input a single sample of feature data, and outputs wether or not the sample corresponds to a weekend. It outputs an integer 1 if the sample is in a weekend, and 0 if it is not. This function uses the date().weekday() function. Given a date, it will output an integer from 0 to 6 which corresponds to the days of the week. This is how i can tell what day of the week it is, given any date.

is_date_rush_hour(X_row)
This function takes as input a single sample of feature data and outputs wether or not the sample corresponds to rush-hour or not. I have arbitrarily chosen rush-hour to be between 6AM and 8AM and between 2PM and 4PM. This can be justified by how the weekly patterns look in section 1.2.

is_date_daytime(X_row)
This function takes as input a single sample of feature data and outputs wether or not the sample corresponds to daytime or not. I have arbitrarily chosen daytime to be anything that is not between 22PM and 4AM. This could also be justified by looking at the weekly patterns in section 1.2.

modify_features_onehot(X, weekends, holiday, rush_hours, daytimes) This function takes one or several feature samples **X** as input, which onehot columns the user wants to add, and then outputs a modified feature matrix **modified_X**. This modified feature matrix will have extra discrete feature columns appended to it which will add some extra information for our estimators to digest.

data_splitter(X,y)
This function takes the feature matrix **X** and the target values **y** as inputs and outputs training, validation and testing features and targets. This function uses sklearn's **train_test_split()** function with shuffling set to **False**. I also let the training data be the first 70% of the dataset, the validation data be the next 15%, then the testing data be the final 15%.

plot_holiday_domains(X)
This function takes the feature data **X** and plots coloured vertical lines corresponding to the start and end of each major holiday. This is only used to justify something in section 1.2.

train_eval_plot_model(model_list, X_train, y_train_coln, X_val, y_val_coln, week)
This function takes as input a list of models (with different hyperparameters), the data used for training, the validation data used for computing the score, then the argument 'week' which is used to determine which week to plot. It then loops through all the candidate models and finds the best one out of them. It then plots the best model's predicted values vs the actual validation values. Then it returns the best model and the best score achieved.

1.1 Data:

We load the main dataset by first storing it as a pandas dataframe, then converting it into a numpy array. The pandas dataframe is only used for printing as it also gives the names of the relevant columns.

```
# Load the dataset as a pandas dataframe
dataset = pd.read_csv('data.csv')

# Extract features to a pandas dataframe X then convert to a numpy array
X = dataset.iloc[:,:-3].to_numpy()

# Extract features to a pandas dataframe y then convert to a numpy array
y = dataset.iloc[:,-3:].to_numpy()

print('Number of samples: ', X.shape[0])
print('Number of features: ', X.shape[-1])
print('Number of targets: ', y.shape[-1])
```

```
Number of samples:  35192
Number of features:  4
Number of targets:  3
```

Let us get a feel for what the data looks like;

```
print(dataset)
```

```
          År  Måned  Dag  Fra_time  Volum til SNTR  Volum til DNP  Volum totalt
0       2015     12   16        11             265            232           497
1       2015     12   16        12             243            223           466
2       2015     12   16        13             251            289           540
3       2015     12   16        14             369            409           778
4       2015     12   16        15             283            365           648
...      ...    ...  ...       ...             ...            ...           ...
35187   2019     12   31        19              77             72           149
35188   2019     12   31        20              58             61           119
35189   2019     12   31        21              52             51           103
35190   2019     12   31        22              51             39            90
35191   2019     12   31        23              34             81           115

[35192 rows x 7 columns]
```

As we can see, our first column **År** describes the year of the given sample, the second column the month, third the day. The fourth column **Fra_time** is the hour of the day ranging from 0 to 23. These four columns will be our base features. The three last columns **Volum til SNTR**, **Volum til DNP** and **Volum totalt**, describe the volume of cars going to the center, to Danmarksplass and the total volume during one hour. They are our target values which we wish to predict. We will train models seperately on each target column.

1.2 Preprocessing:

1.2.1 Visualizing the data

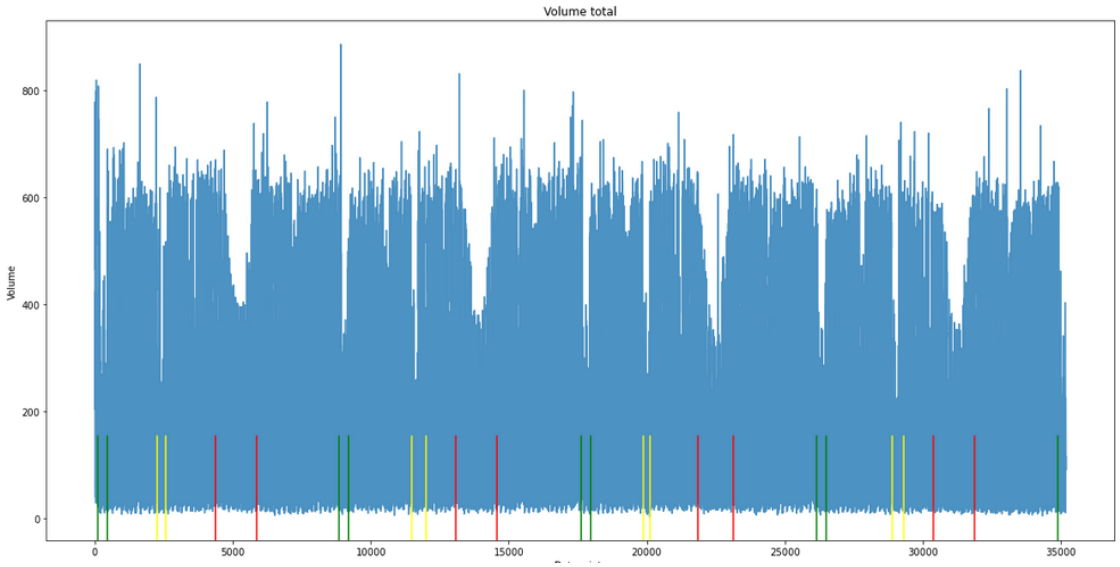
To get a better understanding of how the features affect the target values, i will visualize the data. Since this dataset is quite large (over 35 thousand samples), i will have a look at both local (weekly) structure, and global (yearly) structure seperately.

We know that we have three different sets of target values. The volume of cars heading towards the center, Volum til SNTR, the volume of cars heading to Danmarksplass, Volum til DNP, and the total volume of cars per hour Volum totalt. These targets occupy columns 0, 1 and 2 respectively in the target matrix y.

Global patterns

We begin with the global structure. In this PDF I will only plot the total volume. This is because the volume towards Danmarksplass and the center are very similar globally. The Jupyter notebook will have a global plot of all three target values however.

```
# Initialize a figure of certain size
plt.figure(figsize=(20,10))

# Plot the targets in column 2 of the target matrix
plt.plot(y[:,2], alpha=0.8)

# Set labels and title
plt.xlabel('Datapoints')
plt.ylabel('Volume')
plt.title('Volume total')

# Plot the domains of holidays
plot_holiday_domains(X)

plt.show()
```



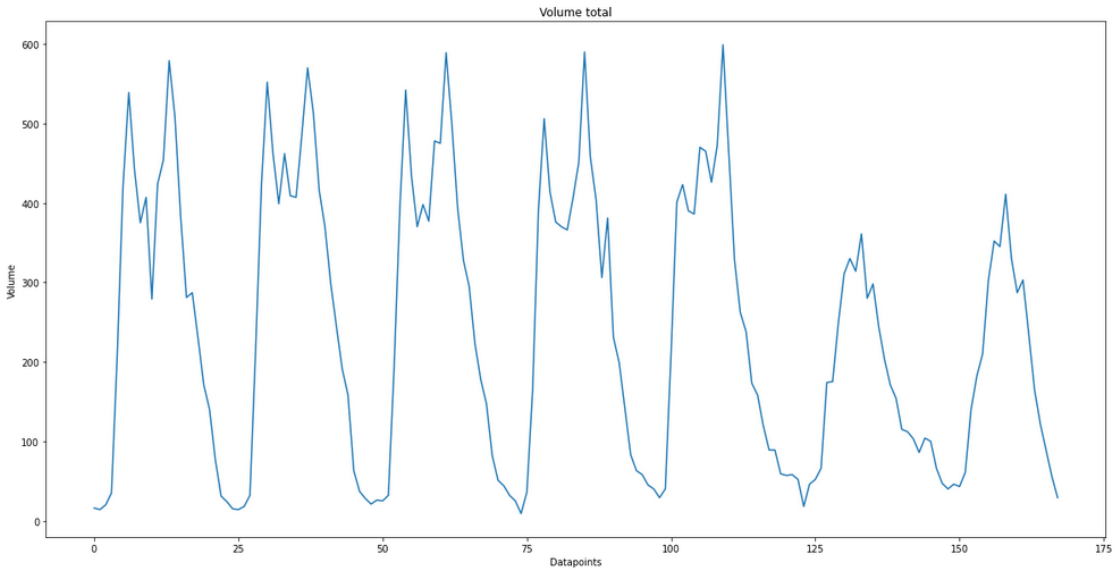
Here we see the total amount of cars over for each sample. This dataset spans over a period of approximately four years. I have not plotted the specific dates, but keep in mind that a period of one year is around 8760 samples. We can clearly see some regularities in the pattern. We have an average volume of about 600 interrupted with regular dips down to 400 or less each year. By looking at the date where these larger dips occur, i find that all the larger dips in volume are caused by the major holidays.

By using the **plot_holiday_domains(X)** function, we get several coloured vertical lines on our plot. A pair of green lines corresponds to Christmas. A pair of yellow lines correspond to Easter. A pair of red lines corresponds to summer. The dates for Christmas and summer are hard coded, but Easter changes date depending on the year. That is why i had to resort to the **easter_sunday()** function to determine when Easter is. These lines lign up well with the dips in the data, thus justifying my suspicion that traffic goes down during the holidays.

Weekly patterns

I now want to have a look at local structure. I choose an arbitrary week which is somewhat far away from any holiday. In the following example i look at the target values from sample 1118 to 1286. These correspond to the data from the 1st of Feburary 2016 to the 8th of Feburary 2016.

```
# Initialize a figure of certain size
plt.figure(figsize=(20,10))

# I plot the values for the 1th column between the sample #1118 and #1286 which is a weeks worth of data.
plt.plot(y[1118:1286,2], alpha=1)

# Set labels and title
plt.xlabel('Datapoints')
plt.ylabel('Volume')
plt.title('Volume total')

plt.show()
```

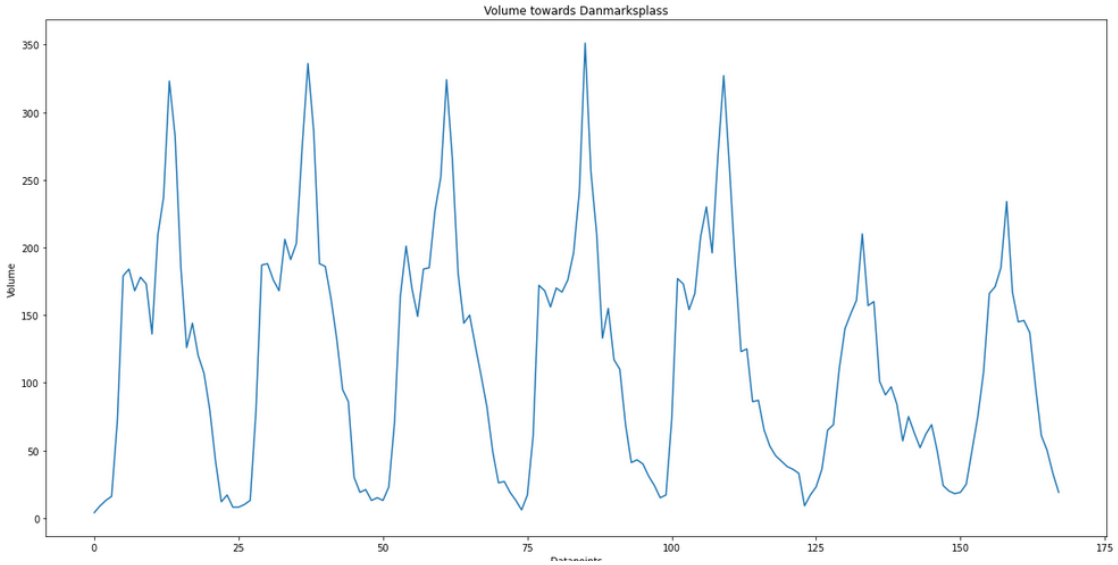


Here we see seven peaks. These peaks correspond to each day of the week. Also, many of the peaks contain sub-peaks. These sub-peaks correspond to rush-hour in the morning, and then rush-hour later in the afternoon. We also see that the first and second sub-peak are about the same size however this will not be the case for the other target columns. The reason why the sub-peaks are about the same size, is because there are about the same amount of cars heading towards the center in the morning, as people leaving from work in the evening.

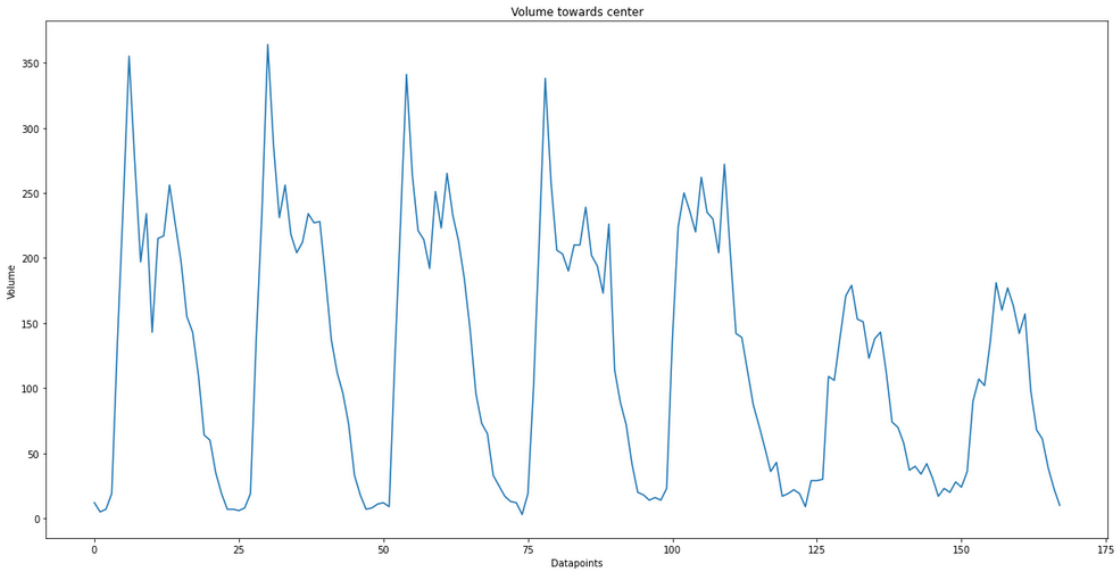
We can also see that during the week, there are five big peaks, then two smaller ones. This corresponds to the fact, that traffic will be a lot heavier during workdays, as opposed to the weekend.

I also plot the volumes towards Danmarksplass and the volume towards the center for the same week.

```
# Initialize a figure of certain size
plt.figure(figsize=(20,10))

# I plot the values for the 1th column between the sample #1118 and #1286 which is a weeks worth of data.
plt.plot(y[1118:1286,1], alpha=1)

# Set labels and title
plt.xlabel('Datapoints')
plt.ylabel('Volume')
plt.title('Volume towards Danmarksplass')

plt.show()
```




```
# Initialize a figure of certain size
plt.figure(figsize=(20,10))

# I plot the values for the 0th column between the sample #1118 and #1286 which is a weeks worth of data.
plt.plot(y[1118:1286,0], alpha=1)

# Set labels and title
plt.xlabel('Datapoints')
plt.ylabel('Volume')
plt.title('Volume towards center')

plt.show()
```



Here we see the same general patterns. The major difference is that in each peak, the first and second sub-peak differ in size significantly. This reveals that in the morning,

more cars are headed towards the center. And then in the evening as people are finishing work, there are more cars headed towards Danmarksplass.

1.2.2 Training-, test- and validationdata

Now, we need to consider how to split our data **X** and **y** into training, validation and test data and targets. Our data is continous in time and thus, for plotting purposes, a single datapoint can only be appreciated together with neighboring datapoints. If we were to shuffle the dataset, then plotting would give us a mess of fluctuating values without the patterns we have looked at. I would prefer to keep my data in sequence. Thus, i will not suffle my data before splitting it. Consequently, the first years will be the training data, then the last few months will be the validation and test data respectively. Specifically, the first 70% of data will be training data, then the next $15\% + 15\%$ will be validation and test data respectively.

```
X_train, X_val, X_test, y_train, y_val, y_test = data_splitter(X,y)
```

This should not adversely affect the performance of the learning algorithm as we have enough datapoints to span a wide array of different dates for each dataset. And if we assume that there are no major differences in traffic between 2015-2019, then splitting this way will not make too much of a difference. This is more evident by looking at the global plot for the total volume. The traffic patterns remain somewhat consistent throughout the years.

1.2.3 Onehot encoding

As we are free to modify the dataset as we wish, I might want to help the learning algorithm by adding some extra information via one-hot encoding. The idea is to add extra columns to our feature data. I think the most important features will be ones which can tell us wether or not the sample we look at is;

1. A major holiday or not. Holidays have a major effect on the average value for volume.
2. A weekend or not. Weekends have on average a lower volume, and they tend to not have a sub-peak structure.
3. A time with high or low traffic. Rushhour has high traffic. Nighttime has very little traffic.

I will compare the score of a learning algorithm trained on unmodified data to one trained on one-hot-encoded data to justify my choices of features.

I have created a function **modify_features_onehot(X, weekends, holidays, rush_hours, daytimes)**, which takes the featuredata **X**, and adds extra columns (features) to it. One column is the 'weekend' column, it contains 0s for workdays, and 1s for weekends. Another column is the 'holiday' column. It contains 0s for when the sample is not a holiday and 1s for when the sample is one of the major holidays. Then i have the 'Rush-hour' column. It contains 1s if the time is between 6-8 AM or 2-4 PM and 0s otherwise. Finally the 'daytime' column. It contains 0s for nighttime (22 PM to 4AM) and 1s otherwise.

This function also takes the arguments **weekends**, **holidays**, **rush_hours** and **daytimes**. These will by default be set to true. If any of them are false, then the corresponding column will not be added to the modified feature matrix **modified_X**. This was used during the prototyping phase. I would add and remove features and check which ones gave me an overall better score. I concluded that keeping all of them set to true was the best. The reason i did not have this as a part of my model selection process is that some of the estimators take several minutes to train on the data. If i were to loop through all the different possible combinations of feature data to include, a cell would take forever to complete.

After passing the feature data, **X_train** for example, through this function, we get a modified featureset **modified_X_train**. Here is an example of what a modified featureset would look like;

```
print(X_modified_train[80:100])
```

```
[[2015   12   19   19    1    0    0    1]
 [2015   12   19   20    1    0    0    1]
 [2015   12   19   21    1    0    0    1]
 [2015   12   19   22    1    0    0    0]
 [2015   12   19   23    1    0    0    0]
 [2015   12   20    0    1    1    0    0]
 [2015   12   20    1    1    1    0    0]
 [2015   12   20    2    1    1    0    0]
 [2015   12   20    3    1    1    0    0]
 [2015   12   20    4    1    1    0    0]
 [2015   12   20    5    1    1    0    1]
 [2015   12   20    6    1    1    1    1]
 [2015   12   20    7    1    1    1    1]
 [2015   12   20    8    1    1    1    1]
 [2015   12   20    9    1    1    0    1]
 [2015   12   20   10    1    1    0    1]
 [2015   12   20   11    1    1    0    1]
 [2015   12   20   12    1    1    0    1]
 [2015   12   20   13    1    1    0    1]
 [2015   12   20   14    1    1    1    1]]
```

The first four columns are the dates of the samples, the last four columns are the extra discrete features ive added.

1.3 Modelling and evaluation:

We want to train atleast three different models that should predict the total number of cars crossing Gamlenygårdsbro during one hour given the date and time. We begin by first training our models on the total volume.

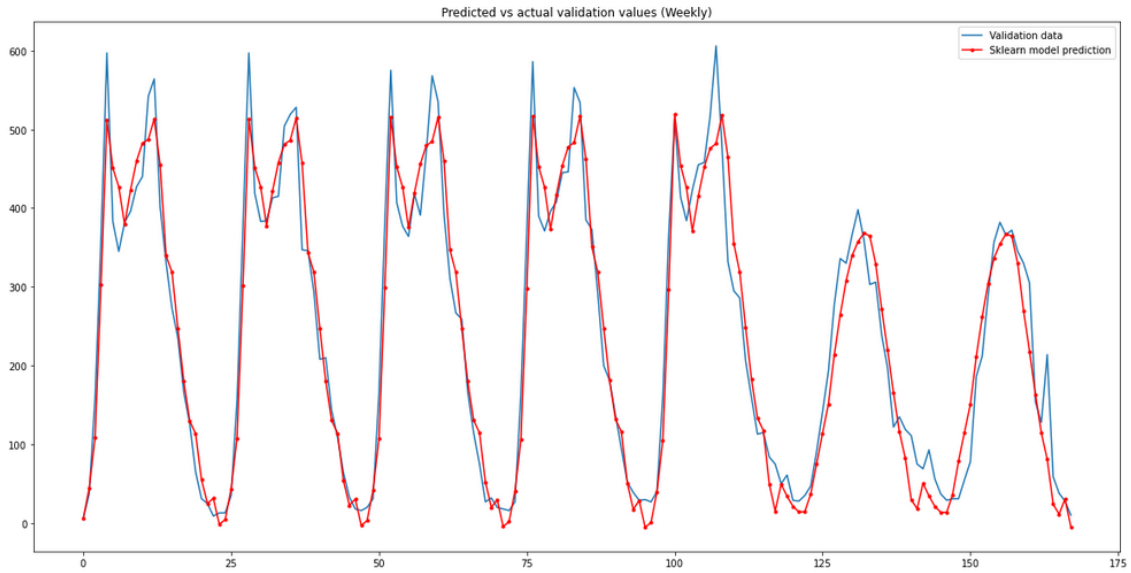
1.3.1 Model selection: Total volume

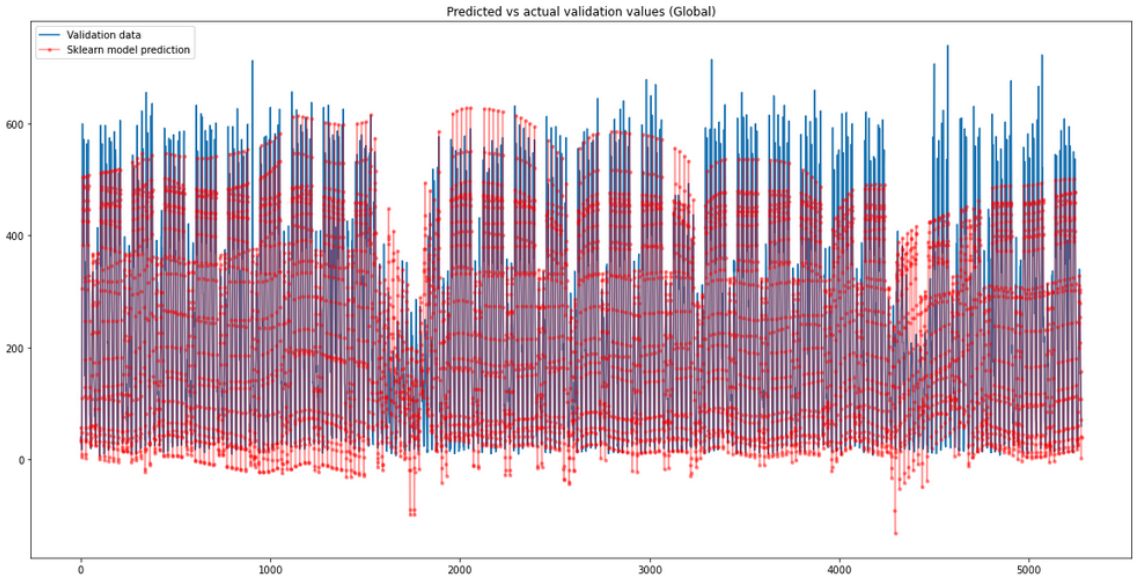
1.3.1 a) LinearRegression()

The first model i wanted to try was the sklearn's **LinearRegression()** model, by first transforming the data with **PolynomialFeatures()**. As our target values oscillate and they are cyclical in the range 0 to 23, i thought it could be an ok model for the job.

I create a loop which changes the degree d on each iteration and initializes a new model with these hyperparameters. I then use the **train_eval_plot_model()** function which will loop through all the candidate models and find the best one. It then plots the results of this best model.

```
# Initialize a list of hyperparameters to vary
d_list = [1,2,3,4,5,6]

# Initialize the best score achieved
best_score = 0

# Initialize the model list
model_list = []

# Extract the targets we are going to train/evaluate with
y_train_coln = y_train[:,2]
y_val_coln = y_val[:,2]

# Loop through all of the hyperparameters and store each model in a list
for d in d_list:
        model_list.append(make_pipeline(PolynomialFeatures(d), LinearRegression()))

best_poly_model, best_poly_score = train_eval_plot_model(model_list, X_modified_train, y_train_coln, X_modified_val, y_val_col
```





By looking at the weekly (local) plot, we could almost say that it predicts the volume perfectly, however as we look at the entire validation set, we see that it consistently underestimates the volume during rushour, overestimates the volume for vacations and even predicts negative values. This is obviously wrong.

Next, i print the score for the best model and the hyperparameters that were used to achieve this score:

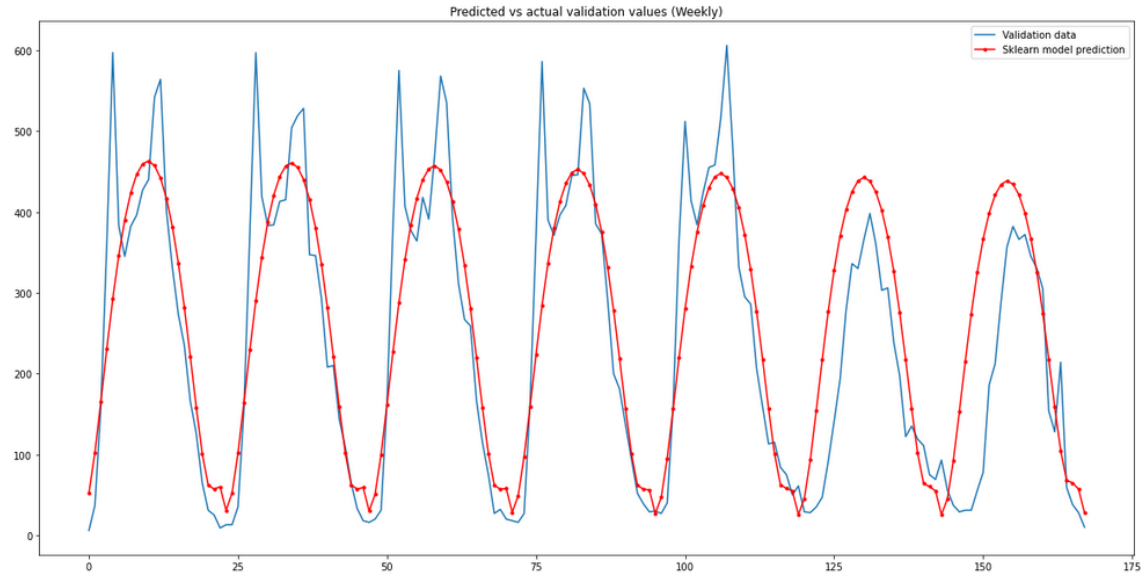
```
score = best_poly_model.score(X_modified_val,y_val[:,2])

print('The best polynomial model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_poly_model.get_params(deep=False)
```

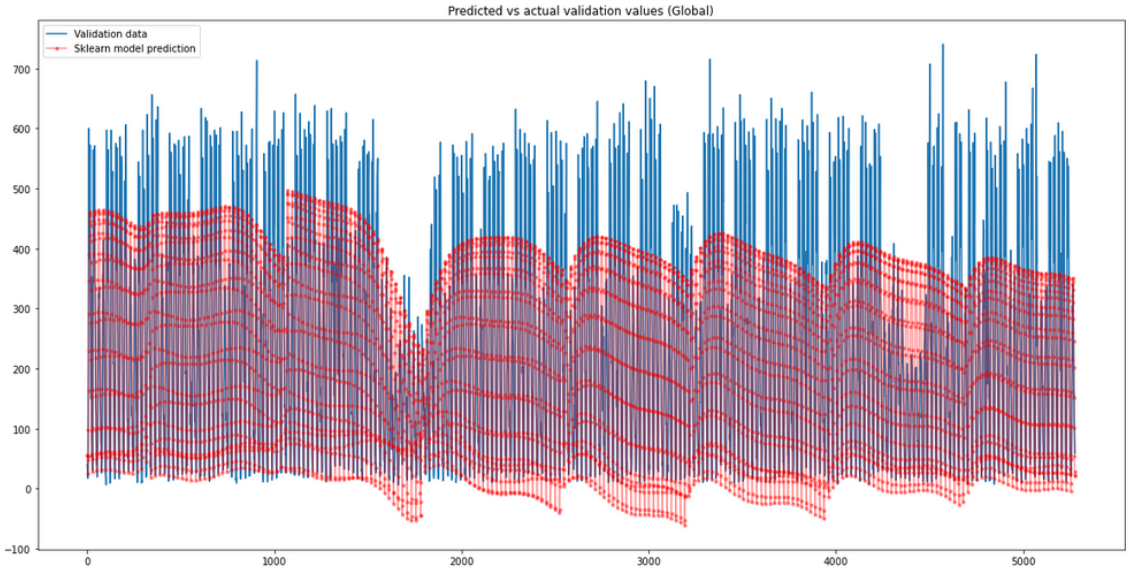
```
The best polynomial model achieved a score of: 0.8834511902500924
It achieved this score with the hyperparameters:

{'memory': None,
 'steps': [('polynomialfeatures', PolynomialFeatures(degree=5)),
  ('linearregression', LinearRegression())],
 'verbose': False}
```

We get a very formidable score of 0.8835. The optimal degree was $d = 5$. The performance is largely attributed to the onehot encoding I performed on the feature data in the preprocessing stage.

Tangent: Justification for my choice of discrete features

Why did I select weekends, holidays, rush-hour and daytime as the discrete features for onehot encoding? I simply selected them as they gave me very large increases in performance for the models that i tested. If I train the LinearRegressor model on the unmodified **X_train** data, i get the following plots;





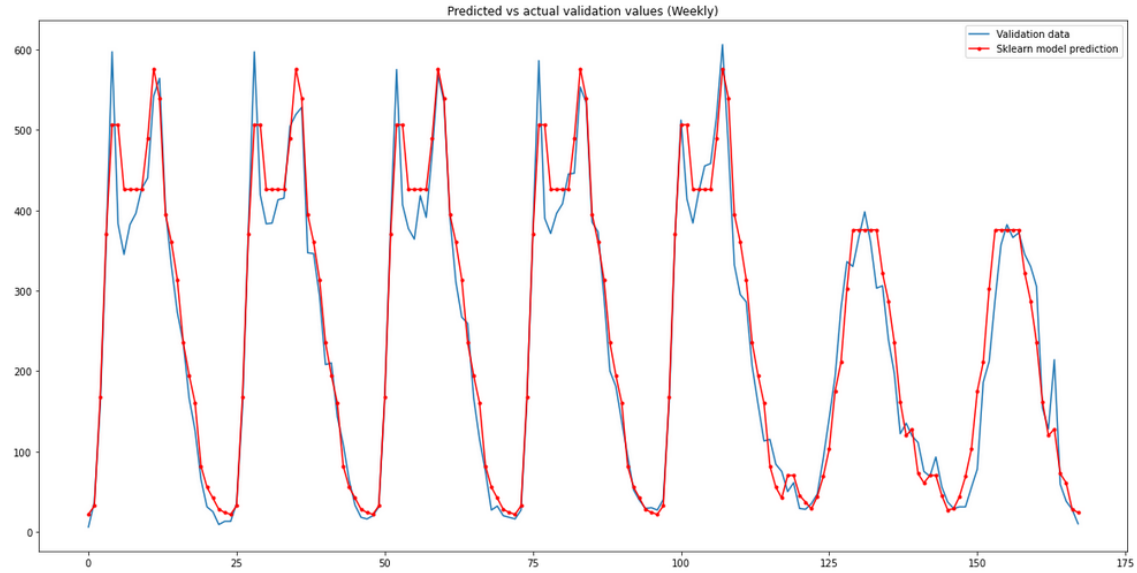

```
score = poly_model_ex.score(X_val,y_val[:,2])

print('The best polynomial model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
poly_model_ex.get_params(deep=False)
```

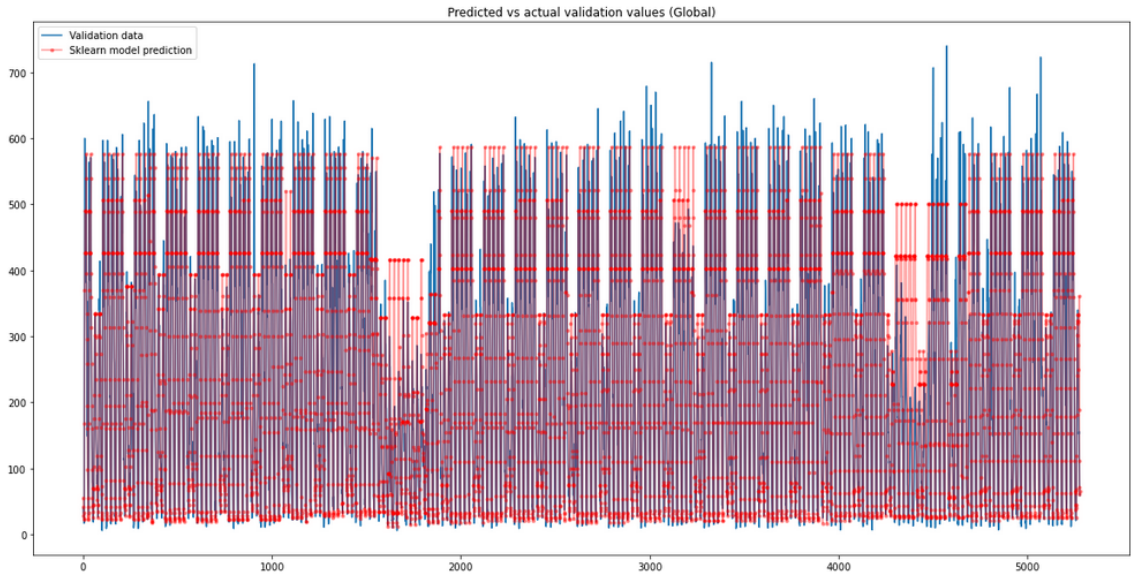
```
The best polynomial model achieved a score of: 0.6566753024080358
It achieved this score with the hyperparameters:

{'memory': None,
 'steps': [('polynomialfeatures', PolynomialFeatures(degree=6)),
  ('linearregression', LinearRegression())],
 'verbose': False}
```

We only get a score of 0.6567 this time. The addition of my choice of discrete features gave us an increase in score of 0.2268.

1.3.1 b) Decision Tree Regressor

The next regressor I wanted to try was sklearn's **DecisionTreeRegressor()**. I wish to do model selection on the parameter **max_depth** as it decides the amount of pre-pruning. The code i wrote for the model selection is largely identical to the previous bit of code. The only difference is that i loop over the hyperparameters **max_depth** instead. I let the maximum depths be a list covering a wide range of values [**None, 500, 200, 100, 50, 20, 10, 8, 6, 4, 2**]. By running the cell i get the following graphs;





In the local plot, we see that the estimator quite accurately predicts the double-peak behaviour. It is also able to capture how the shape of the Saturday/Sunday peaks are smaller and simpler. In the global plot we see that it still underestimates the rushhour peaks, but not as badly as the **LinearRegressor()** based model. However it still overestimates the volume during the vacations. This model does in the very least not predict negative values.

I print the score and the hyperparameter used to achieve the score;

```
score = best_DTR_model.score(X_modified_val,y_val[:,2])

print('The best Decision Tree Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_DTR_model.get_params(deep=False)
```

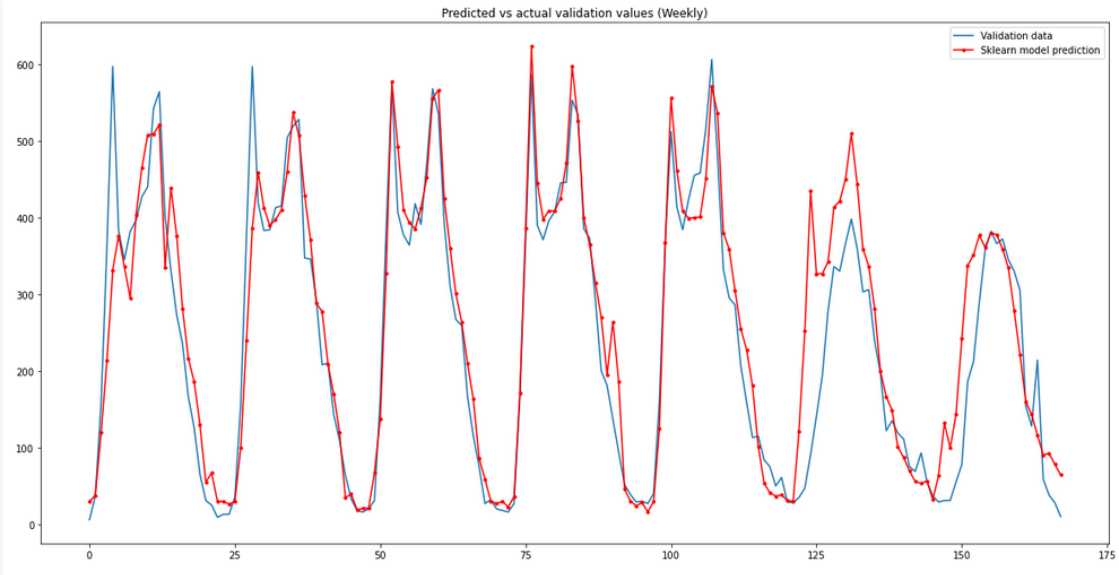
```
The best Decision Tree Regressor model achieved a score of: 0.8899497181242353
It achieved this score with the hyperparameters:

{'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 10,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 0,
 'splitter': 'best'}
```

We now get a score of 0.8899. This is really good. But hopefully we can do even better. This score was achieved with a maximum depth of 10.

1.3.1 c) K-Nearest Neighbors Regressor

Now i want to try a KNN Regressor. I will let the hyperparameters be the number of neighbors to account for, and how to weigh the neighbors. I let the number of neighbors vary in a list as [**1, 2, 3, 4, 5, 10, 15, 20, 25**], then i let the weighting vary as [**'uniform', 'distance'**]. Here a uniform weighting means that all neighbors are equally weighted, wheras the distance weighting means that the prediction is weighted more towards near points.

Running the cell gives me the following plots;



The local plot shows us that it does not always manage to predict the correct shape for the daily peaks. In this case, the Monday peak has three peaks, and the Saturday peak has two peaks. Looking at the global plot however we see that it sometimes overestimates rushour traffic and sometimes underestimates it. Surprisingly this model almost perfectly predicted the dip of the first vacation in the validation dataset (around sample 1700). The second one not at all however (the dip around sample 4300).

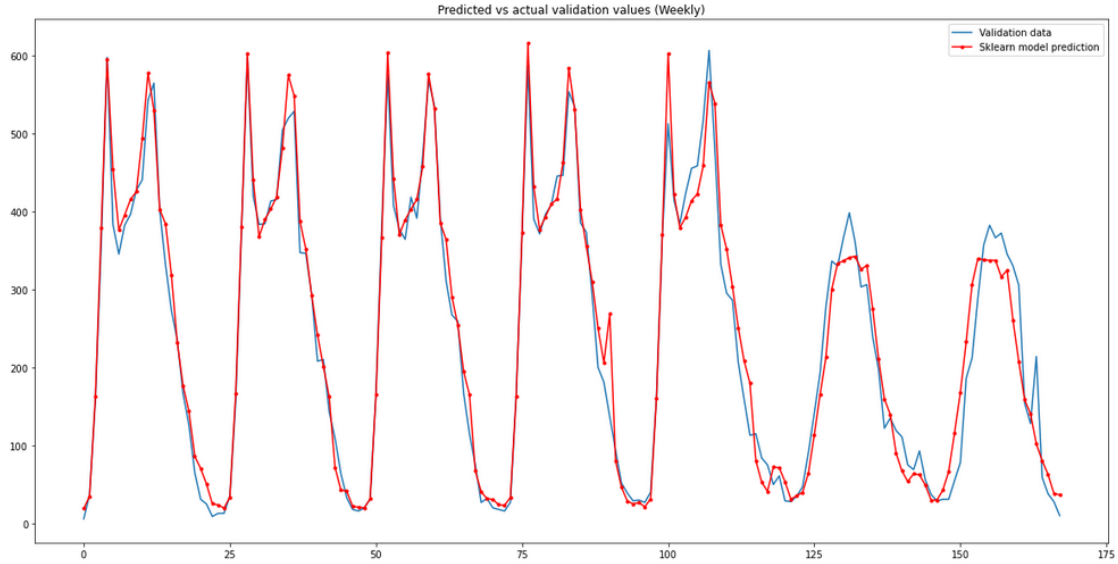
```
score = best_KNNR_model.score(X_modified_val,y_val[:,2])

print('The best KNN Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_KNNR_model.get_params(deep=False)
```

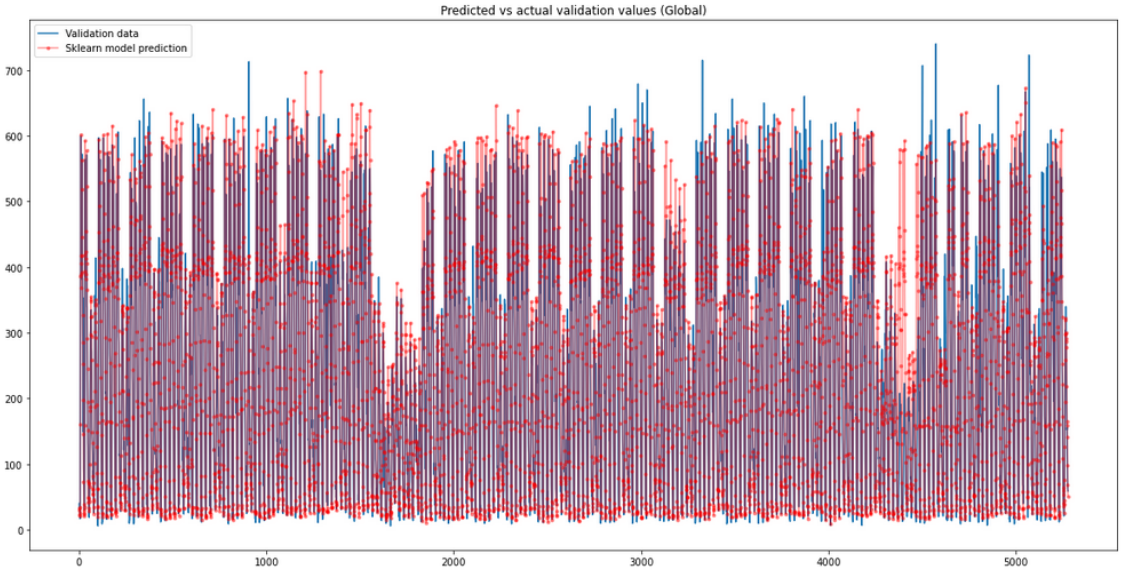
```
The best KNN Regressor model achieved a score of: 0.819891865139968
It achieved this score with the hyperparameters:

{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 3,
 'p': 2,
 'weights': 'distance'}
```

We see that this model performs the worst so far with a score of 0.8199. It achieved this score with number of neighbors = 3 and the weighting being based on the distance.

1.3.1 d) Random Forest Regressor

Finally i would like to try a random forest regressor. I will vary the hyperparemeters for number of estimators to use, and the maximum depth to have for each tree. I let the maximum depth vary as [5, 10, 20, 25, None] and the number of estimators vary as [25, 50, 100, 150, 200, 250].





Both the local and global plots look very promising. It just about perfectly predicts the behaviour locally. Globally it underestimates rushhour a lot less and it handled the first holiday very well, but overestimated the second vacation somewhat.

```
score = best_RND_Forest.score(X_modified_val,y_val[:,2])

print('The best Random Forest Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_RND_Forest.get_params(deep=False)
```

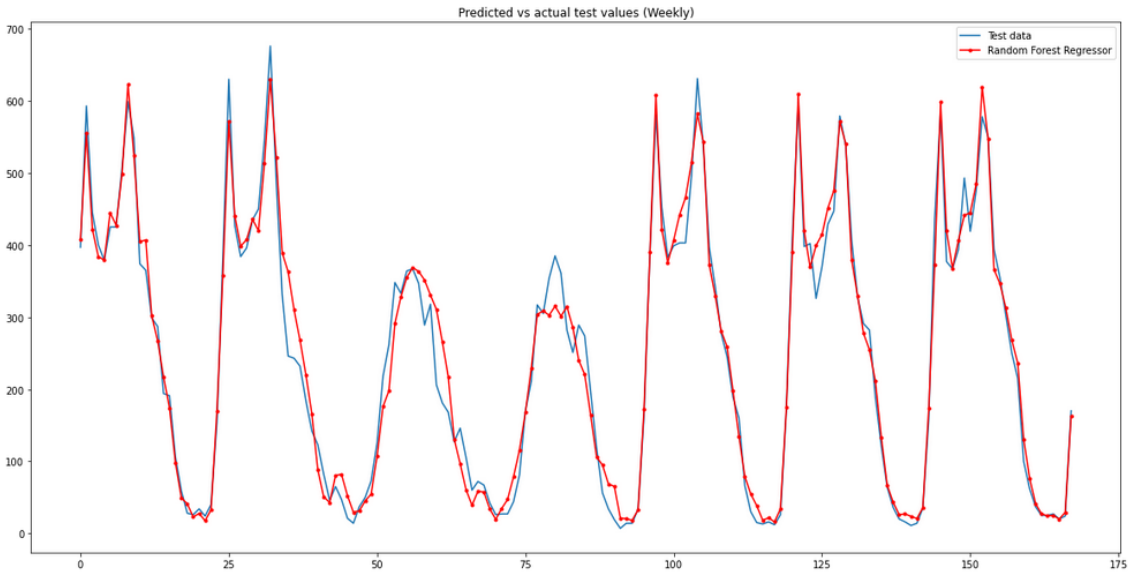
```
The best Random Forest Regressor model achieved a score of: 0.9220053847883242
It achieved this score with the hyperparameters:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 20,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 200,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 999,
 'verbose': 0,
 'warm_start': False}
```

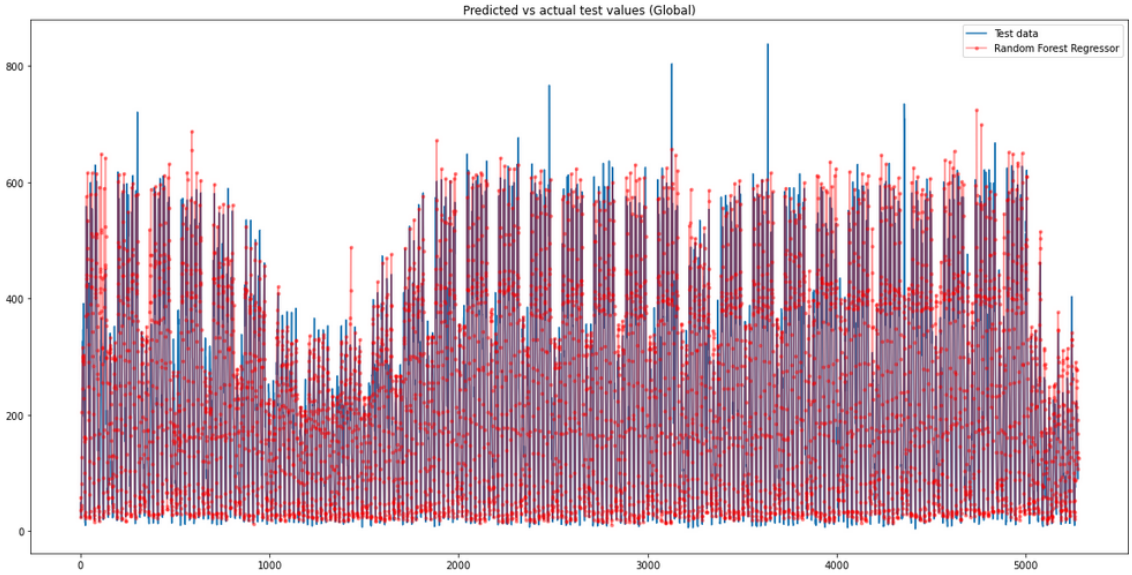
With this model i get a very good score of 0.9223. This was achieved with 200 estimators and a maximum depth of 20. This was the best score i could achieve with all of my testing. Thus i will select the Random Forest Regressor with the hyperparameters **n_estimators** = 200 and **max_depth** = 20 as my model.

1.3.1 e) The best overall model on total volume

We can see that so far, the best model is the Random Forest regressor with the hyper parameters **n_estimators** = 200 and **max_depth** = 20. This model had a score of 0.9223 on the validation data which is very good! I thus select this as my prefered model for predicting total volume.

I now evaluate my selected model on the test data to estimate its performance on unseen data. I get the following;






```
score_test = best_RND_Forest.score(X_modified_test,y_test[:,2])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_test) + ' on the test data.')
```

```
The best Random Forest Regressor model achieved a score of: 0.9250183137886531 on the test data.
```

Our selected model gets an astonishing score of 0.9250 on unseen data (the test data) for the total volume. This score is surprisingly larger than the validation score used in model selection. We know that validation score is supposed to be an overestimate of testing performance. One reason for this might be that since the dataset is not shuffled, the test dataset might be a bit simpler to predict for. A reason for this could perhaps be that the test data includes the big dip of the summer holidays. And since the summer holidays last a lot longer than any of the other holidays, then there will be more datapoints to associate with a lower traffic.

1.3.2 Model selection: Volume towards the center

We now want to do the same procedure as before but for the volume towards the center instead. I will not show the local and global plots as they take up a lot of space. I will instead just show the results for each of the four models. All the plots will be in the Jupyter notebook however.

1.3.2 a) LinearRegression()

We do the same model selection on the linear regression model with the 0th column of the target matrix. We get the following score;

```
score = best_poly_model.score(X_modified_val,y_val[:,0])

print('The best polynomial model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_poly_model.get_params(False)
```

```
The best polynomial model achieved a score of: 0.844094939968034
It achieved this score with the hyperparameters:

{'memory': None,
 'steps': [('polynomialfeatures', PolynomialFeatures(degree=5)),
  ('linearregression', LinearRegression())],
 'verbose': False}
```

We get a score of 0.8441 with the hyperparameter degree = 5.

1.3.2 b) Decision Tree Regressor

Similarly for the decision tree regressor;

```
score = best_DTR_model.score(X_modified_val,y_val[:,0])

print('The best Decision Tree Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_DTR_model.get_params(False)
```

```
The best Decision Tree Regressor model achieved a score of: 0.8903687266761225
It achieved this score with the hyperparameters:

{'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 10,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 0,
 'splitter': 'best'}
```

Here the decision tree regressor achieves a score of 0.8903 with the hyperparameter maximum depth = 10.

1.3.2 c) K Nearest Neighbor Regressor

```
score = best_KNNR_model.score(X_modified_val,y_val[:,0])

print('The best KNN Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_KNNR_model.get_params(False)
```

```
The best KNN Regressor model achieved a score of: 0.7976278379003706
It achieved this score with the hyperparameters:

{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 3,
 'p': 2,
 'weights': 'distance'}
```

The KNN Regressor achieves a score of 0.7977 with the hyperparameters number of neighbors = 3 and weights = distance.

1.3.2 d) Random Forest Regressor

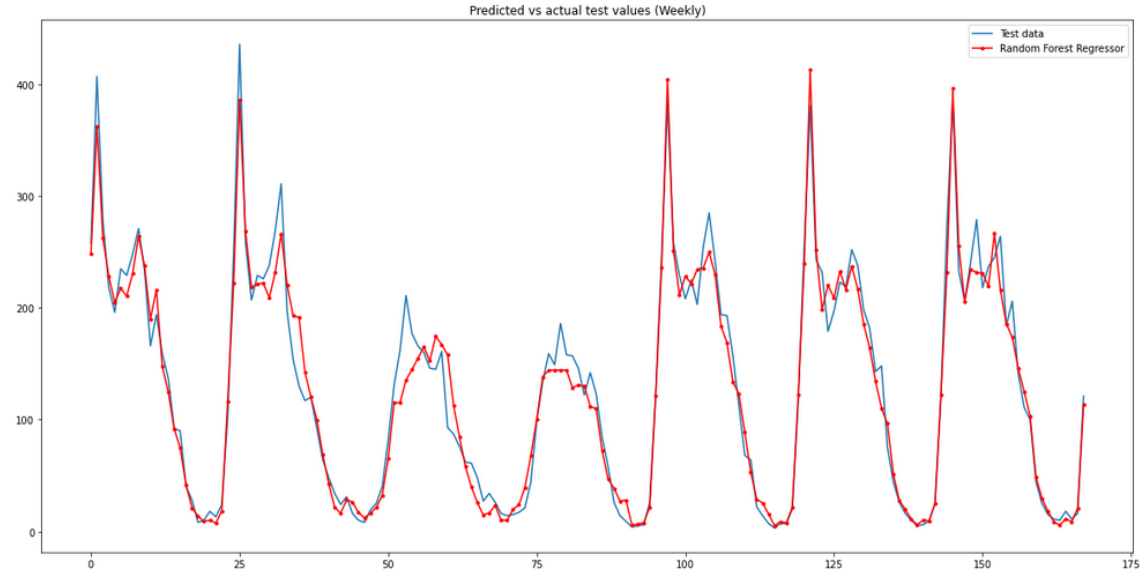
```
score = best_RND_Forest.score(X_modified_val,y_val[:,0])

print('The best Random Forest Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_RND_Forest.get_params(False)
```

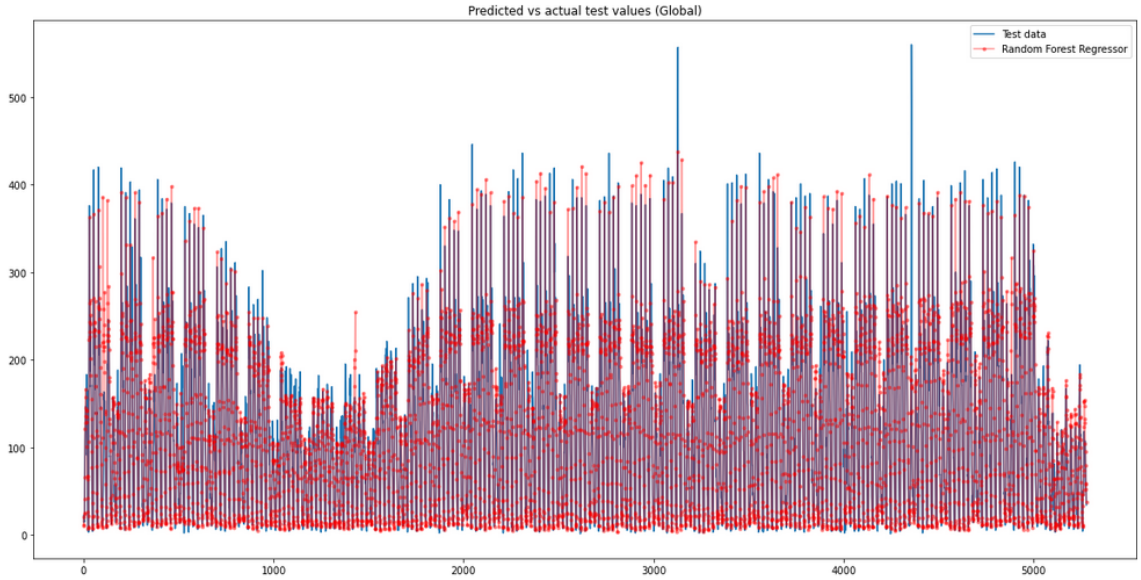
```
The best Random Forest Regressor model achieved a score of: 0.9185178652377399
It achieved this score with the hyperparameters:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 20,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 200,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 999,
 'verbose': 0,
 'warm_start': False}
```

The Random Forest Regressor achieves a score of 0.9185 with the hyperparameters number of estimators = 200 and maximum depth = 20.

1.3.2 e) The best overall model on volume towards center

We can see that again, the best model is the Random Forest regressor with the hyper parameters **n_estimators** = 200 and **max_depth** = 20. This model had a score of 0.9185 on the validation data. I thus select this as my prefered model.






```
score_test = best_RND_Forest.score(X_modified_test,y_test[:,0])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_test) + ' on the test data.')
```

```
The best Random Forest Regressor model achieved a score of: 0.9296178545497278 on the test data.
```

The selected model has a score of 0.9296 on the unseen test data for the volume towards the center. The model also correctly predicts that the first sub-peak is larger than the second sub-peak. Saturdays and Sundays are also smaller peaks with no distinguishing sub-peaks. Note that in this specific case, the Saturday and Sunday peaks are the 3rd and 4th peak in the graph respectively. This is because the week has been offset after the split.

1.3.3 Model selection: Volume towards Danmarksplass

Same procedure as before but for the volume towards Danmarksplass.

1.3.3 a) LinearRegression()

We do the same model selection on the linear regression model with the 1th column of the target matrix. We get the following score;

```
score = best_poly_model.score(X_modified_val,y_val[:,1])

print('The best polynomial model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_poly_model.get_params(deep=False)
```

```
The best polynomial model achieved a score of: 0.8649173668120111
It achieved this score with the hyperparameters:

{'memory': None,
 'steps': [('polynomialfeatures', PolynomialFeatures(degree=5)),
  ('linearregression', LinearRegression())],
 'verbose': False}
```

We get a score of 0.8649 with the hyperparameter degree = 5.

1.3.3 b) Decision Tree Regressor

Similarly for the decision tree regressor;

```
score = best_DTR_model.score(X_modified_val,y_val[:,1])

print('The best Decision Tree Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_DTR_model.get_params(deep=False)
```

```
The best Decision Tree Regressor model achieved a score of: 0.8770480134288866
It achieved this score with the hyperparameters:

{'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 10,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 0,
 'splitter': 'best'}
```

Here the decision tree regressor achieves a score of 0.8770 with the hyperparameter maximum depth = 10.

1.3.3 c) K Nearest Neighbor Regressor

```
score = best_KNNR_model.score(X_modified_val,y_val[:,1])

print('The best KNN Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_KNNR_model.get_params(deep=False)
```

```
The best KNN Regressor model achieved a score of: 0.8134770264708939
It achieved this score with the hyperparameters:

{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 10,
 'p': 2,
 'weights': 'distance'}
```

The KNN Regressor achieves a score of 0.8135 with the hyperparameters number of neighbors = 10 and weights = distance.

1.3.3 d) Random Forest Regressor

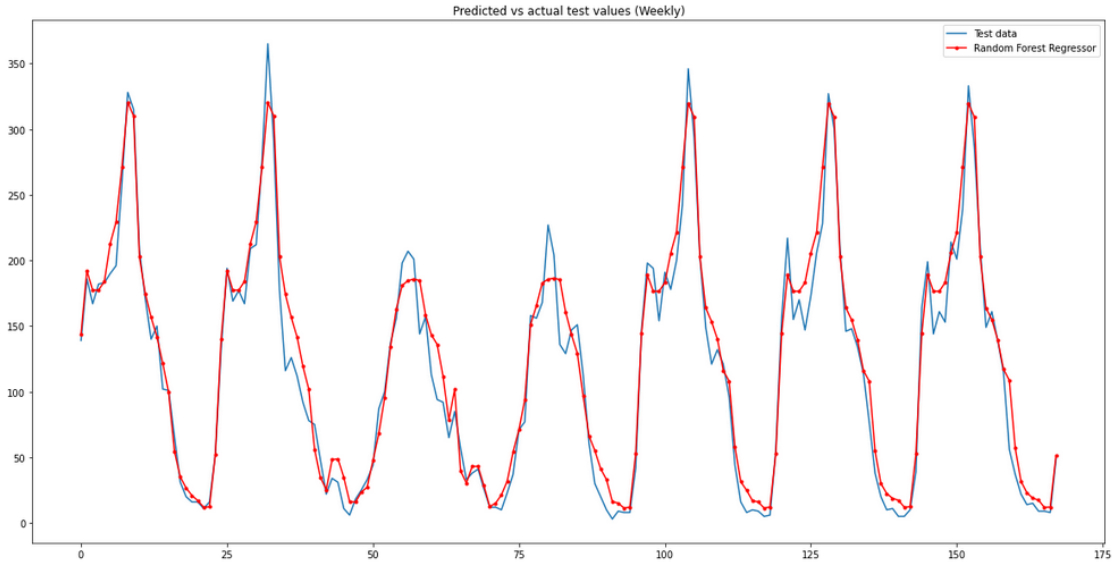
```
score = best_RND_Forest.score(X_modified_val,y_val[:,1])

print('The best Random Forest Regressor model achieved a score of: ' + str(score))
print('It achieved this score with the hyperparameters:')
best_RND_Forest.get_params(deep=False)
```

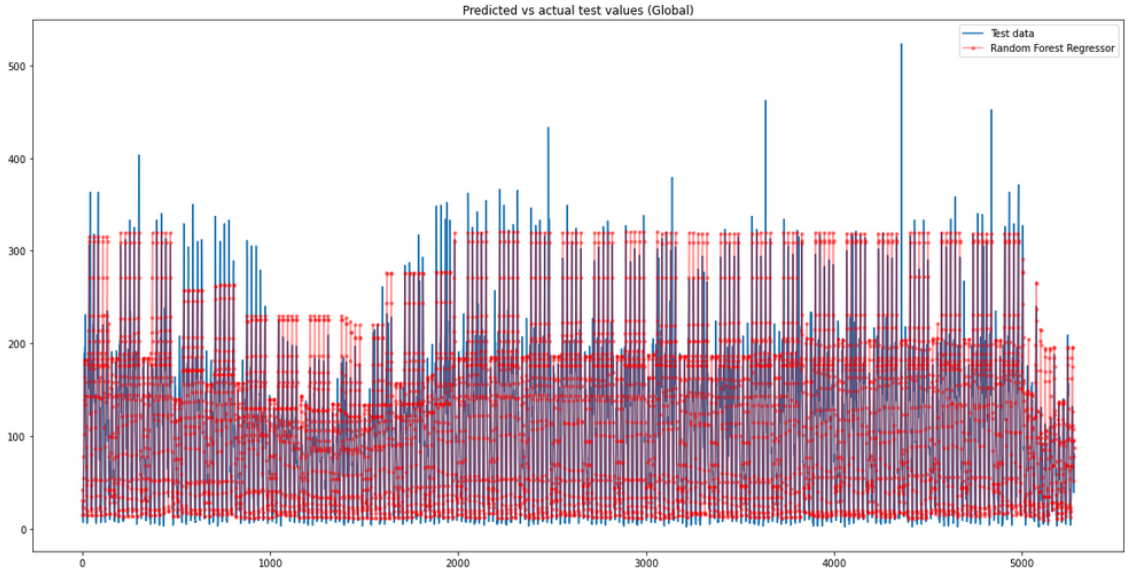
```
The best Random Forest Regressor model achieved a score of: 0.8910185916093275
It achieved this score with the hyperparameters:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'mse',
 'max_depth': 10,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 50,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 999,
 'verbose': 0,
 'warm_start': False}
```

The Random Forest Regressor achieves a score of 0.8910 with the hyperparameters number of estimators = 50 and maximum depth = 10.

1.3.3 e) The best overall model on volume towards Danmarksplass

We can see that now, the best model is the Random Forest regressor with the hyper parameters **n_estimators** = 50 and **max_depth** = 10. This model had a score of 0.8910 on the validation data. I thus select this as my prefered model.






```
score_test = best_RND_Forest.score(X_modified_test,y_test[:,1])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_test) + ' on the test data.')
```

```
The best Random Forest Regressor model achieved a score of: 0.8704053299460695 on the test data.
```

The selected model has a score of 0.8704 on the unseen test data for the volume towards Danmarksplass. This model correctly predicts that the second sub-peak is larger than the first. It also is able to predict the dip of the summer holiday.

1.3 Conclusion so far

As I have already discussed in section 1.2, the main differences between the patterns towards Sentrum and Danmarksplass is that on average, towards Sentrum has a high first sub-peak and a low second sub-peak, wheras Danmarksplass has a low first sub-peak and a high second sub-peak.

After training, evaluating and selecting models for all three targets, we get that the best models are;

Targets	Model	n_estimators	max_depth	Score on test data
Total volume	**RandomForestRegressor()**	200	20	0.9250
Sentrum	**RandomForestRegressor()**	200	20	0.9296
Danmarksplass	**RandomForestRegressor()**	50	10	0.8704

For all three targets, the most optimal estimator was a random forest regressor. The only difference being the number of estimators and the maximum depth of each tree. By looking at the global plots of these tree estimators on the test data, we can see they all do a very good job at estimating the number of cars per hour, even being able to account for the large dip in traffic due to the summer holidays (between sample 800 and 1800) and christmas (the very end). The models are mostly bad at predicting the highest peaks of rush-hour, we can see for all the three targets that we have certain high peaks that our models underestimate.

One limitation of my models are that they are very reliant on the extra features i added with onehot encoding. As we saw in the tangent section, the score was a lot lower without them. And a lot of the added features i included in the onehot encoder are ones that i felt were logical. Of course there could be others that could have helped the learners even more to make accurate predictions. For example, i did not include the autumn holidays in my **is_date_holiday()** function. This is because the dates changes depending on the year and also which commune you are in. That is why i did not add it. The models are also dependent on the fact that the traffic patterns do not change abruptly by some unforseen events not accounted for in the features. The model will only stay accurate if the traffic pattern stays somewhat similar to what the models have trained on. We will look at what happens when the traffic patterns abruptly change in the next section.

1.3.4 Evaluating my model on the 2020 dataset

Now that we have a best model for all the 3 different target values, we want to deploy them on the 2020 dataset. I load the 2020 dataset similarly to before;

```
# Load the 2020 dataset as a pandas dataframe
dataset_2020 = pd.read_csv('data_2020.csv')

# Extract features to a pandas dataframe X_2020 then convert to a numpy array
X_2020 = dataset_2020.iloc[:,:-3].to_numpy()

# Extract features to a pandas dataframe y_2020 then convert to a numpy array
y_2020 = dataset_2020.iloc[:,-3:].to_numpy()

print('Number of samples: ', X_2020.shape[0])
print('Number of features: ', X_2020.shape[-1])
print('Number of targets: ', y_2020.shape[-1])
```

```
Number of samples:  5230
Number of features:  4
Number of targets:  3
```

We see that the 2020 dataset is of similar form as the previous dataset.

```
print(dataset_2020)
```

```
        År  Måned  Dag  Fra_time  Volum til SNTR  Volum til DNP  Volum totalt
0     2020      1    1         0              58             59           117
1     2020      1    1         1              44             48            92
2     2020      1    1         2              36             42            78
3     2020      1    1         3              34             21            55
4     2020      1    1         4              28             19            47
...    ...    ...  ...       ...             ...            ...           ...
5225  2020      8    5        17             133            138           271
5226  2020      8    5        18             141            133           274
5227  2020      8    5        19              81             76           157
5228  2020      8    5        20              64             71           135
5229  2020      8    5        21              47             30            77

[5230 rows x 7 columns]
```

I run the feature matrix **X_2020** through the onehot encoder to add the extra features. Then retrain the three best models with their respective features and targets. Remember i retrain on the **same** training data as before. The only reason i do this is so that i make sure i have the correct models.

```
best_RND_Forest_total = RandomForestRegressor(n_estimators=200, max_depth=20, random_state=999)
best_RND_Forest_total.fit(X_modified_train, y_train[:,2])
```

```
RandomForestRegressor(max_depth=20, n_estimators=200, random_state=999)
```

```
best_RND_Forest_center = RandomForestRegressor(n_estimators=200, max_depth=20, random_state=999)
best_RND_Forest_center.fit(X_modified_train, y_train[:,0])
```

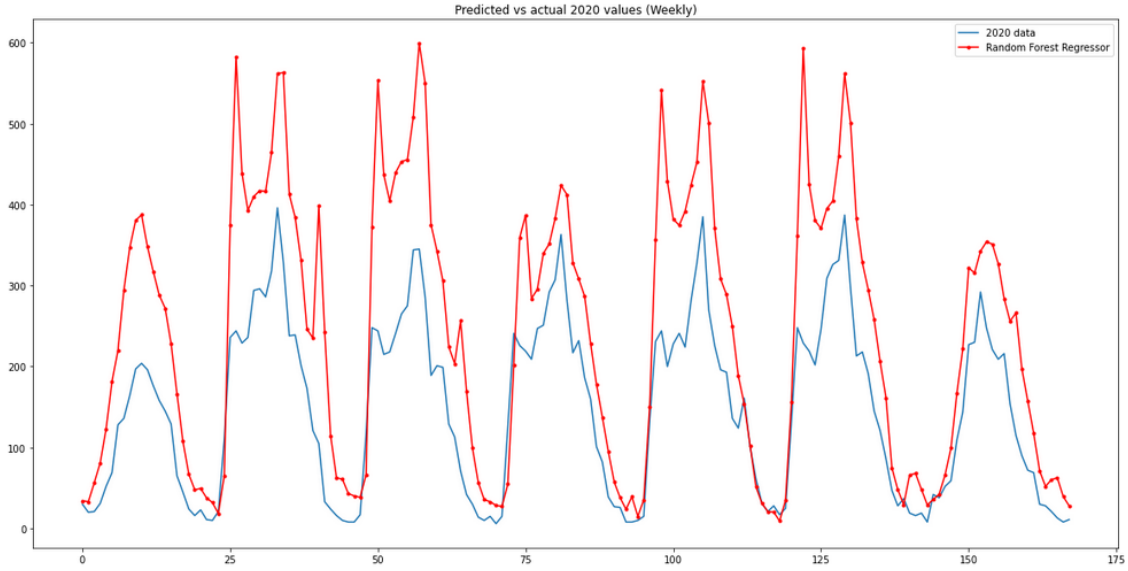
```
RandomForestRegressor(max_depth=20, n_estimators=200, random_state=999)
```

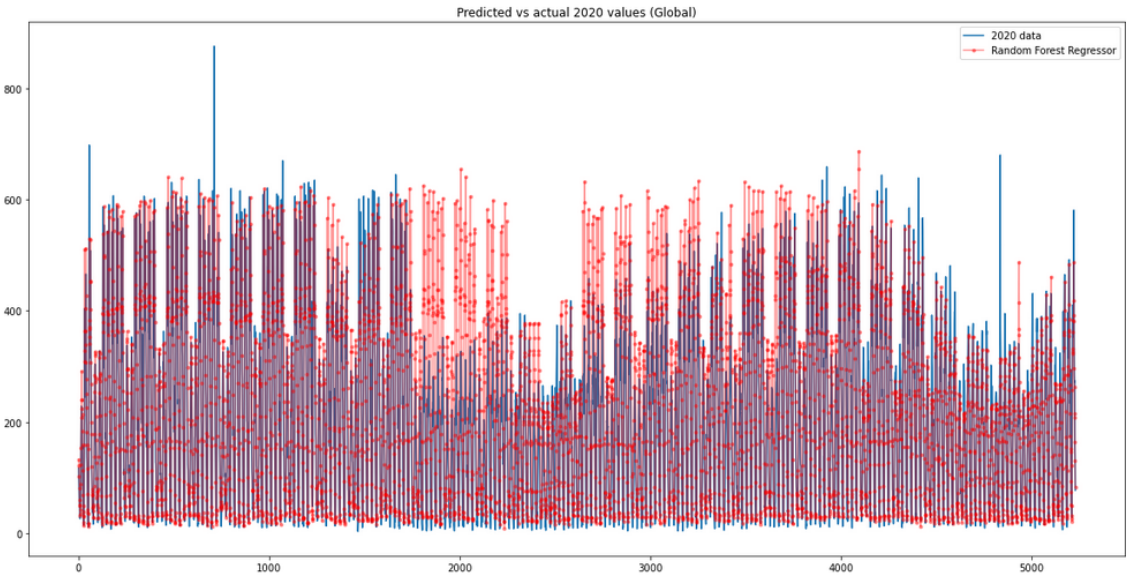
```
best_RND_Forest_DNMKP = RandomForestRegressor(n_estimators=50, max_depth=10, random_state=999)
best_RND_Forest_DNMKP.fit(X_modified_train, y_train[:,1])
```

```
RandomForestRegressor(max_depth=10, n_estimators=50, random_state=999)
```

I will now evaluate and plot the predicted vs actual values for the 2020 dataset for all three targets.

1.3.4 a) Evaluation and plotting for total volume 2020



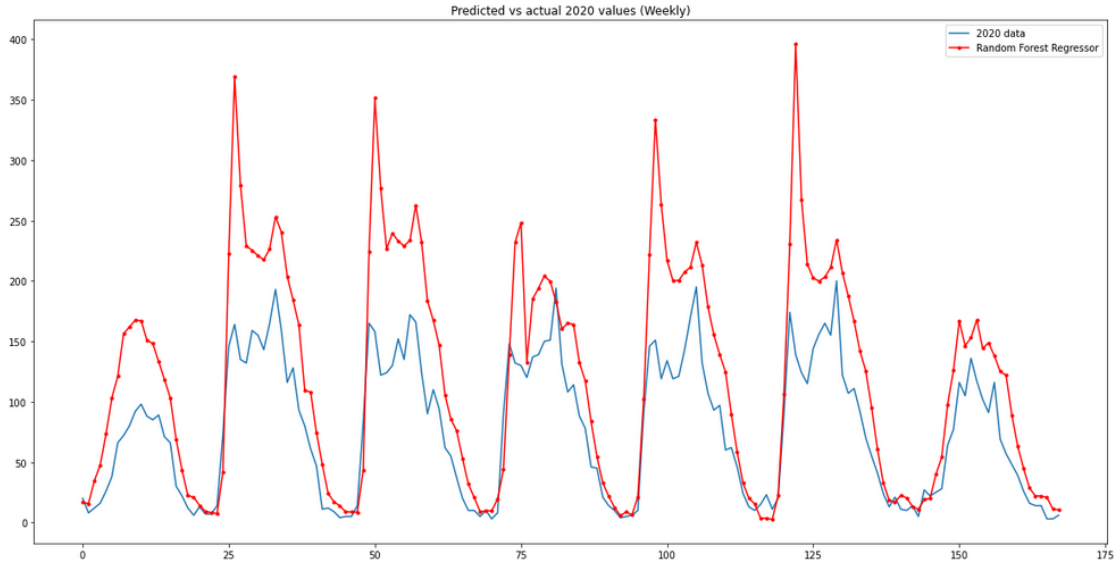


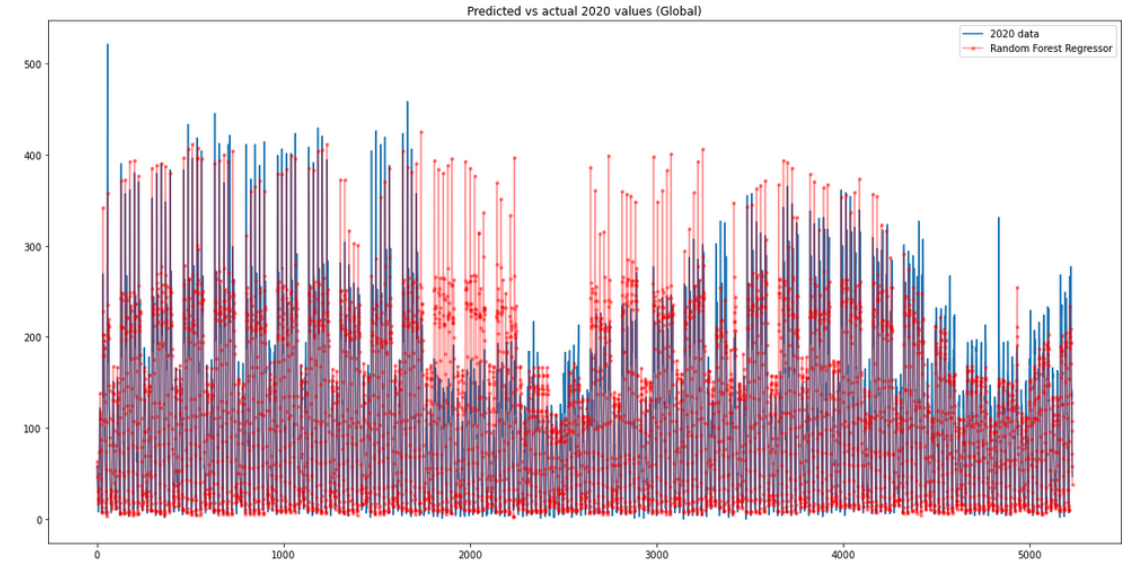

```
score_2020_total = best_RND_Forest_total.score(X_modified_2020, y_2020[:,2])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_2020_total) + ' on the 2020 dataset.')
```

```
The best Random Forest Regressor model achieved a score of: 0.8084384772400769 on the 2020 dataset.
```

For the total volume, the model gets a score of 0.8084, which is quite a bit worse than expected.

1.3.4 b) Evaluation and plotting for volume towards center 2020



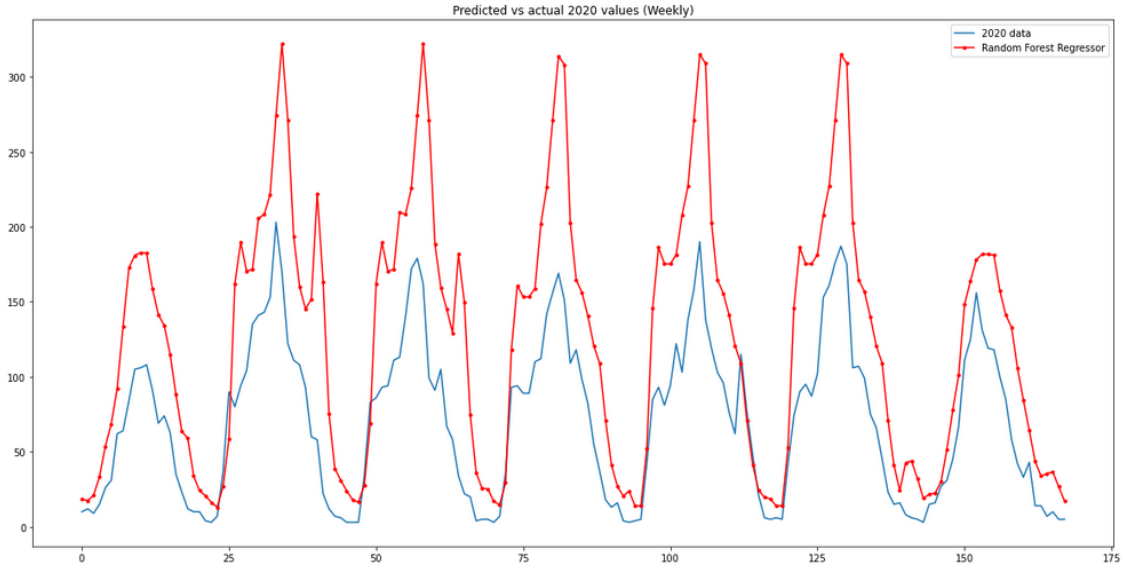


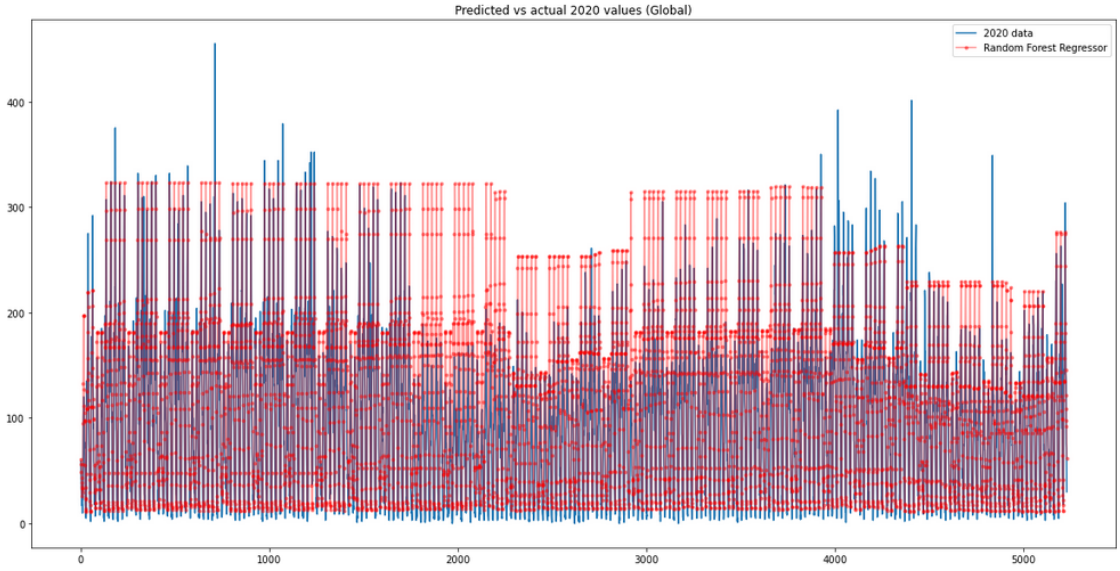

```
score_2020_center = best_RND_Forest_center.score(X_modified_2020, y_2020[:,0])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_2020_center) + ' on the 2020 dataset.')
```

```
The best Random Forest Regressor model achieved a score of: 0.8294054359764303 on the 2020 dataset.
```

For the volume towards the center, the model gets a score of 0.8294.

1.3.4 c) Evaluation and plotting for volume towards Danmarksplass 2020






```
score_2020_DNMKP = best_RND_Forest_DNMKP.score(X_modified_2020, y_2020[:,1])

print('The best Random Forest Regressor model achieved a score of: ' + str(score_2020_DNMKP) + ' on the 2020 dataset.')
```

```
The best Random Forest Regressor model achieved a score of: 0.7364265196804303 on the 2020 dataset.
```

And finally, for the volume towards the center the best model achieves a score of 0.7364.

By using the scores we computed by evaluting the models on the test data to compare, we get the following table.

Model	Score on test data	Score on 2020 data	Difference
best_RND_Forest_total	0.9250	0.8084	-0.1166
best_RND_Forest_center	0.9296	0.8294	-0.1002
best_RND_Forest_DNMKP	0.8704	0.7364	-0.1340

Here we see that the models do a bit worse on the 2020 dataset than expected. The reason for this is that the estimator greatly overestimates the traffic during the lockdown of the 2020 COVID-19 pandemic. The first 11 weeks before lockdown, we can see that the estimator does very well and accurately predicts the traffic as expected. However once lockdown hits, it overestimates traffic by a lot. Curiously there is a sudden dip in the predicted values for the 15th and 16th week. This is because that the Easter holiday occurs in that period, and our models will try to account for that. Once the Easter holiday is over, it starts overestimating again until the start of the summer holiday (around sample 4000).

We also see that the model predicting volume towards Danmarksplass performs significantly worse on the 2020 dataset than the two others.

Note: The local plots are of a week within the COVID-19 lockdown, as I wanted to show how the models are overestimating the traffic.

We have learnt that despite having all this data to train on, our models can not predict abrupt changes due to unforseen events which have not been accounted for in the training data.

We have also seen the benefit of ensemble methods. The random forest regressor won out every single time. The general nature of a decision tree usually leads to overfitting, but with having several 'lower-quality' trees we can have a better generalized performance because we base our prediction on the majority rule. This is why i had a decision tree and a random forest as two of my four candidate models. I wanted to see how the majority rule would benefit the generalized performance of our estimator.