An Analysis of On Time Performance: SEPTA

Trains a crucial lifeline for many, especially in a big city like Philly. SEPTA (Philly's Transportation Authority) has an On-Time-Performance of around 91 percent but even a few late trains can be very frustrating for commuters! This post tries to look at trends in train delays using a dataset that SEPTA uploaded on Kaggle (SEPTA Train Times Dataset)

We will first look at overall trends, then look at a particular station - University City, then look at a particular train- Train 395 and finally try to use a model to predict train delays.

Overall Trends

To start off the analysis, we use a data table that has information about train stations, dates, and delays and cleaned it.

	train_id	direction	origin	next_station	date	status	timeStamp
0	778	N	Trenton	Stenton	2016-03-23	1 min	2016-03-23 00:01:47

We removed data for trains that were suspended, had None values and non-numeric Id's.

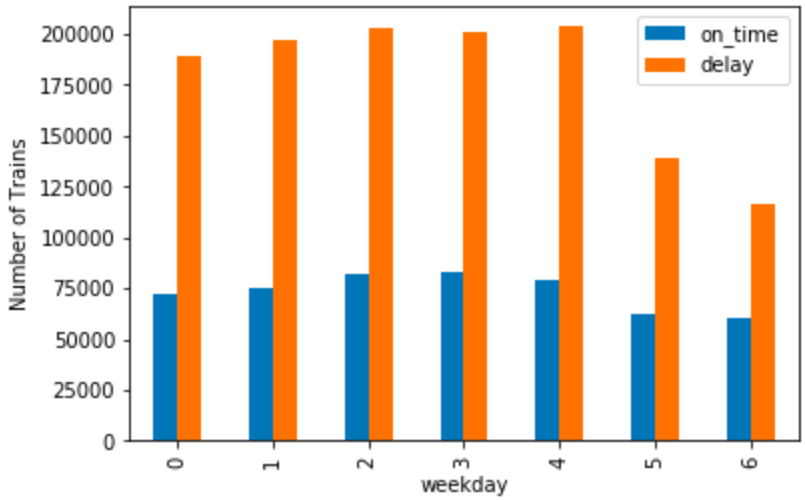
```
otp_df = otp_df[otp_df.train_id.apply(lambda x: x.isnumeric())]
otp_df['train_id'] = otp_df['train_id'].astype(str).astype(int)
otp_df = otp_df[otp_df['origin'] != 'None']
otp_df = otp_df[otp_df['next_station'] != 'None']
otp_df
```

After cleaning out the data, we extract features important to delay times such as weekday- labelled O to 6 (where O represents Monday) and the hour of the day.

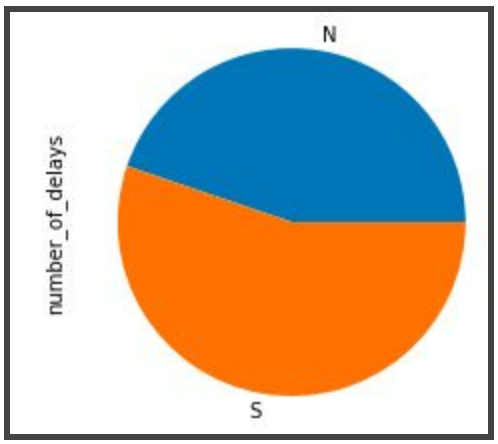
```
otp_df['weekday'] = otp_df['timeStamp'].apply(lambda x: x.weekday())
otp_df['hour'] = otp_df['timeStamp'].dt.hour
```

Well now we are ready to look at what makes Philly's trains run late! Here are a few chosen graphs that provide us important insights

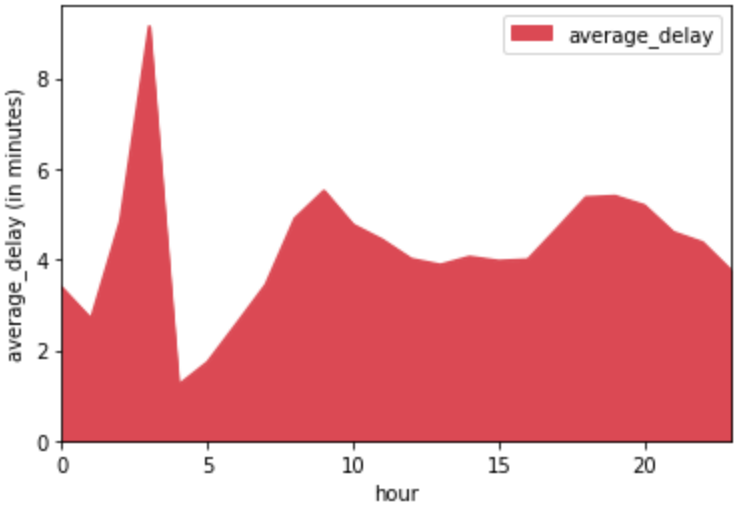
Graph 1: Number of Trains delayed by week day



Graph 2: Number of delays for southbound and northbound trains



Graph 3: Average delay by hour of the day



The first graph shows that on any weekday, the number of trains delayed are more than the number of trains on time. Also beware of travelling between 1-4 AM as you might have to wait for a long..long time! The third graph shows that trains that are North Bound are delayed lesser than South Bound trains.

This data was gathered by using basic SQL queries and graphed using matplotlib functionality:

```
q = """SELECT direction, COUNT(status) AS number_of_delays
    FROM otp_df
    WHERE status > 0
    GROUP BY direction"""
result = pysqldf(q)
```

```
result = result.set_index('direction')
result['number_of_delays'].plot.pie(subplots=True, figsize=(8, 4))
```

University City Station

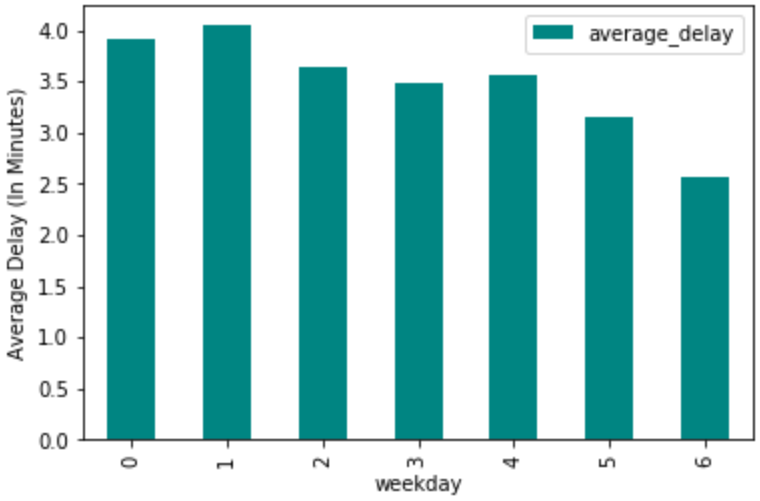
We can also analyse the on-time-performance at specific stations on the SEPTA line. For example, let us look at some of the important trends for the University City Station.

We first extract the rows which have their 'next_station' attribute as 'University City' like so:

```
otp_df = otp_df[otp_df['next_station'] == 'University City']
otp_df
```

Here are some insights into train delays at University City:

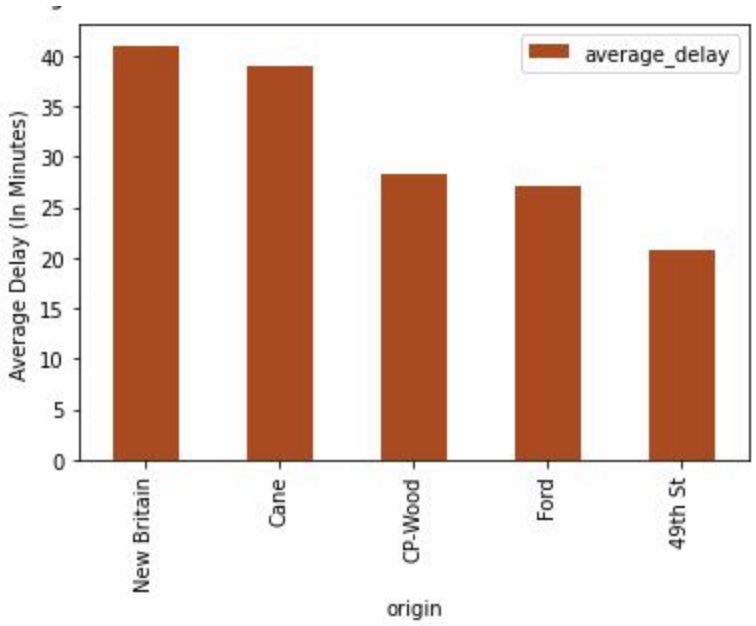
Graph 4: Average delay by weekday at University City



Graph 5: Covariance matrix for status,weekday,hour at U-City

	status	weekday	hour
status	1.000000	-0.066425	0.022997
weekday	-0.066425	1.000000	0.009861
hour	0.022997	0.009861	1.000000

Graph 6: Average delay by origin



We can see that average delays are highest at University City on Tuesdays (the Tuesday morning rush does exist!). We can infer from the correlation matrix that the delay generally increases with an increase in the hours of the day, i.e as we go from early morning to night and the delay decreases with an increase in weekday, i.e towards the weekend. It is also clear that with average delays of around 40 minutes, taking a train from New Britain passing through University City is never a great option!

This data was obtained using SQL queries and python functions for covariance matrices:

```
covariance_df = station_df[['status','weekday','hour']]
corr_matrix = covariance_df.corr()
corr_matrix.style.background_gradient()
```

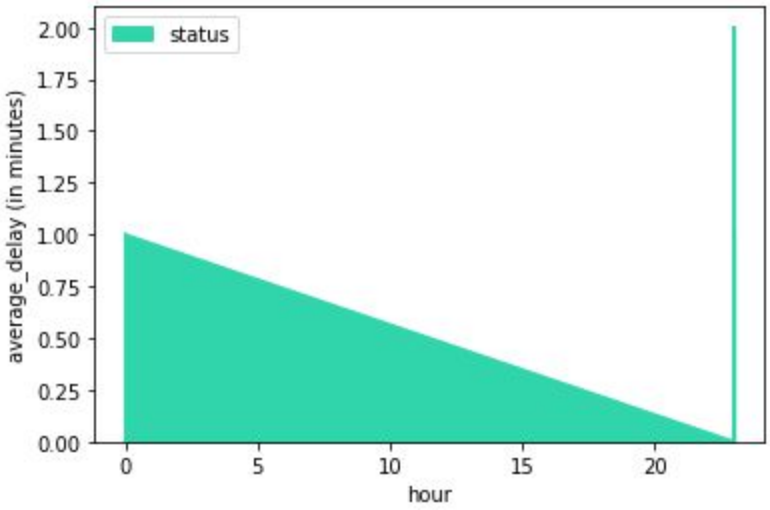
```
q = """SELECT origin, AVG(status) AS average_delay
    FROM station_df
    GROUP BY origin
    ORDER BY average_delay DESC LIMIT 5"""
result = pysqldf(q)
```

Train 395

Why did I pick train 395, is it because it is my favourite number? No, it was just one of the trains passing through University City :)

For each train, we can analyse how delays vary throughout the day and how each station contributes to the delay:

Graph 7: Delay of train 395 by hour of the day



This is the data for the status of the train per hour on a random day of the year (April 08 2016)

We can see that overall, the train is not late by more than 2 minutes and it is only late at around 11:30 PM by around 2 minutes.

We can also look at the incremental contributions of each station to the delays (Note that negative contributions mean that the train actually speeds up here and positive contributions mean that the delay has increased).

We see that Yardley reduces delays the most and Woodbourne increases the delays by the most:

	next_station	incremental_contribution
0	30th Street Station	-0.210526
1	Bethayres	-0.680851
2	Fern Rock TC	0.747899
3	Forest Hills	-1.053191
4	Jefferson Station	0.978723
5	Jenkintown-Wyncote	0.255319
6	Langhorne	-0.042553
7	Meadowbrook	-0.063158
8	Neshaminy Falls	-0.138298
9	Noble	-0.397849
10	Philmont	-0.021053
11	Rydal	0.483516
12	Somerton	-1.084211
13	Suburban Station	-1.361702
14	Temple U	-0.179688
15	Trevose	0.797872
16	University City	0.144444
17	Woodbourne	3.542553
18	Yardley	-1.815217

Model to Predict Delays

Now that we know what the data looks like, let's get down to business. Let us predict when SEPTA trains are delayed!

But before we do this, let us convert the direction, origin and next_station variable from strings to a numerical form. This is done using the get_dummies function like so:

```
otp_df = pd.get_dummies(otp_df, columns = ['direction','origin','next_station'])
otp_df
```

As we have 1.7 million rows in our dataframe, we will be using Apache Spark instead of scitkitlearn, to speed up computations. So, we store our pandas dataframe to a .csv file and read it as a spark dataframe:

```
combined_sdf = spark.read.format('csv').options/
(header='true', inferSchema='true').load('categorical_data.csv')
```

For our machine learning algorithm, our input features will be the weekday, hour (of the day), origin (one-hot-encoded), next_station(one-hot-encoded), and direction(one-hot-encoded)
We put all of these into a single column called 'features' using a vector assembler like so:

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=columns_to_use, outputCol="features")
combined_sdf = assembler.transform(combined_sdf)
combined_sdf.select("features").show()
```

We then split the data into training and test sets:

```
train_sdf, test_sdf = combined_sdf.\
randomSplit([0.8, 0.2], seed = 2018)
```

The next step is to scale the data and we do this using StandardScaler:

```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features", outputCol="scaled_features")
scaler_model = scaler.fit(train_sdf)
train_sdf = scaler_model.transform(train_sdf)
test_sdf = scaler_model.transform(test_sdf)
```

Now to the interesting part, as we have a lot of features, we can try to use Principal Component Analysis. Sadly, in this case it does not yield fruitful results! This is because the variance that each principal component can explain has a maximum of 0.092, which means that we do not derive much utility from this:

```
DenseVector([0.0092, 0.0055, 0.0054, 0.0053, 0.0049,
```

Another question is whether to use regression or classification. We decide to go with classification and the short reason as to why is that regression does not produce great results! On trying linear regression for example, we get:

```
RMSE: 6.215932
r2: 0.087687   TEST RMSE: 6.220979671284873
```

This is not great for two reasons:

- The standard deviation of the column status which represents the delay time is 6.5, so the randomly drawing a status would give roughly the same answers
- R^2 shows that our model only explains 8 percent of the variance in the status variable

So, we use classification and assign the status variable to 0 when there is a delay of less than 2 minutes and 1 when there is a delay of more than 2

minutes (2 is the 50th percentile value of variable status and hence was chosen as the threshold):

```
otp_df['status'] = otp_df['status'].apply(\
lambda x: 0 if (x <= 2) else 1)
otp_df['status']
```

We use a RandomForestClassifier and try out different values for the 'numTree' and 'maxDepth' parameters. We find out that the best parameters are numTree = 10 and maxDepth = 3

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'scaled_features', labelCol = 'label',\
                            numTrees = 20, maxDepth = 3)
rfModel = rf.fit(train_sdf)
predictions = rfModel.transform(test_sdf)
```

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'scaled_features', labelCol = 'label',\
                            numTrees = 10, maxDepth = 3)
rfModel = rf.fit(train_sdf)
predictions = rfModel.transform(test_sdf)
```

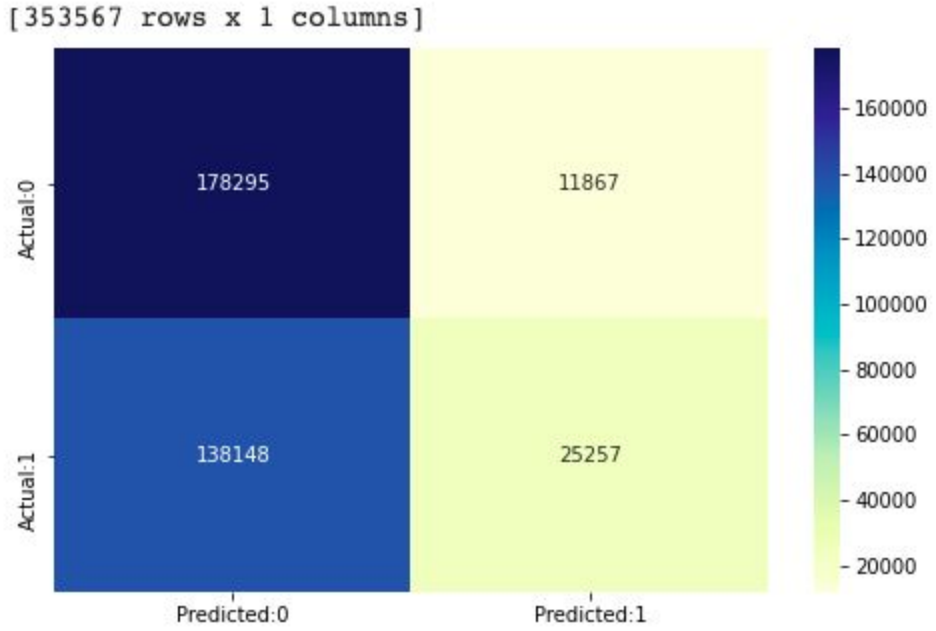
On running this, we find that the area under the ROC (Receiver Operating Characteristic) curve gets maximised:

```
Test Area Under ROC: 0.6051391487395327
```

A classifier is bad if the area is 0.5 and the best if the area is 1. So, the model is doing moderately well in terms of predicting delays.

We also display the confusion matrix for the classifier:

Graph 8: Confusion matrix for the binary classifier



This shows that the number of true positives is greater than the number of false positives and the number of false negatives are greater than the number of false positives

The reason why this classifier does not do exceedingly well, is that there are no clear and obvious correlations between the features and the label. For example, it is difficult to find out how the station of origin affects the delay time.

So why is this classifier useful? The classifier, although not perfect, provides a good estimate of when a train will be delayed. If we can know that the train arrives with less than 2 minutes of delay, that is very useful information.