AASD 4001 Final Report

TTC Subway Delay Dataset

Harsh Gupta - 101371646

Jonie Caisip - 101363744

Leandra Lai - 101367265

Naman Sharma - 101369174

Ziyad El Amrani-Joutey - 101366815

October 5, 2021

Table of Contents

Introduction

The TTC network has many advantages over other modes of transit (last longer, offers great capacity, typically moves at a higher speed, protected from the weather, etc.) However, they have one significant disadvantage which is expensive to build, usually estimated by the TTC at about $300 million per kilometer. Underground stations are also typically far more expensive. This project is based on practical application. We are going to use a publicly available dataset that describes every delay encountered in the TTC subway system in Toronto from January 2014 till June 2021.

The Problem Statement

Toronto subway is a rapid transit system serving Toronto and the city of Vaughan north of Toronto. Operated by the Toronto Transit Commission (TTC), the TTC subway is a multimodal rail network operating predominantly underground, and one elevated medium-capacity rail line. There are currently light rail transit (LRT) lines, which will operate both at-grade and underground, that are under construction. Delays in transit schedules happen day-to-day, and the TTC commuters may be familiar with occasional delays of the subway train. For those with tight schedules, delays can be a big problem and planning around delay times will be necessary. In this project we are predicting the delay of the TTC (Toronto Transit Commission) subway trains using the regression model.

The Dataset

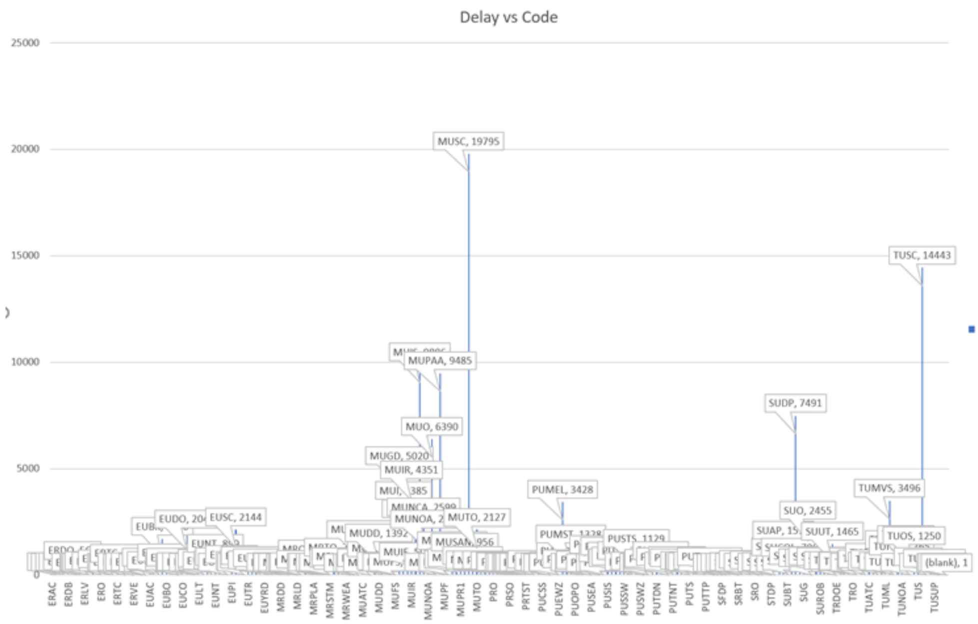
The dataset we chose for this project was found on Kaggle (https://www.kaggle.com/jsun13/toronto-subway-delay-data?select=Toronto-Subway-Delay-Jan-2014-Jun-2021.csv) which is a merge of all the data on train arrivals submitted on TTC's official website from January 2014 to June 2021.

It has over 140 thousand records and contains 10 features including: Date, Time, Day, Station, Code (the kind of the incident), Minutes delay (the delay to the schedule for the following train), Minute Gap (the total scheduled time from the train ahead of the following train), Line of the train and finally the vehicle code. That represents 7 categorical features and 3 integral features.

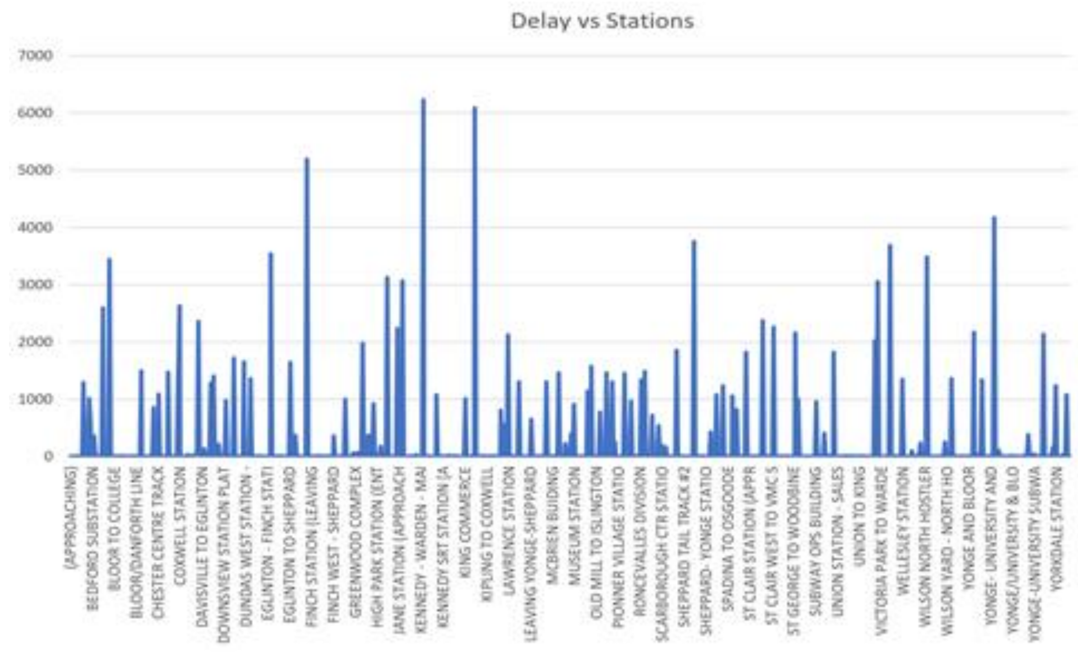
	Date	Time	Day	Station	Code	Min Delay	Min Gap	Bound	Line	Vehicle
0	1/1/2014	2:06	Wednesday	HIGH PARK STATION	SUDP	3	7	W	BD	5001
1	1/1/2014	2:40	Wednesday	SHEPPARD STATION	MUNCA	0	0	NaN	YU	0
2	1/1/2014	3:10	Wednesday	LANSDOWNE STATION	SUDP	3	8	W	BD	5116
3	1/1/2014	3:20	Wednesday	BLOOR STATION	MUSAN	5	10	S	YU	5386
4	1/1/2014	3:29	Wednesday	DUFFERIN STATION	MUPAA	0	0	E	BD	5174
...	...	...	...	...	...	...	...	...	...	...
143135	6/30/2021	1:23	Wednesday	ST CLAIR STATION	MUIS	0	0	NaN	YU	0
143136	6/30/2021	6:00	Wednesday	TORONTO TRANSIT COMMIS	MUO	0	0	NaN	SHP	0
143137	6/30/2021	12:40	Wednesday	LESLIE STATION	MUIS	0	0	NaN	SHP	0
143138	6/30/2021	20:50	Wednesday	LESLIE STATION	MUTD	9	14	E	SHP	6171
143139	6/30/2021	0:45	Wednesday	LESLIE STATION	TUMVS	5	10	E	SHP	6166

143140 rows × 10 columns

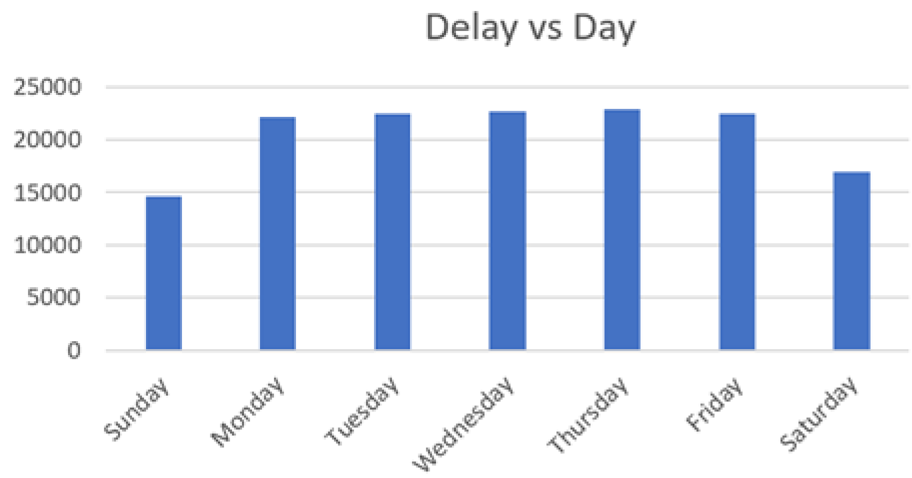
Here are some graphs to establish some statistics about our dataset. Below, you can find a graph that compares the codes to the number of delays that happened. Miscellaneous Speed Control and Operator Over speeding seem to be the most common reasons for a train delay.



The same was done to compare stations and the number of delays. And we found that Kennedy, King Commerce and Finch Station have the most delay problems.

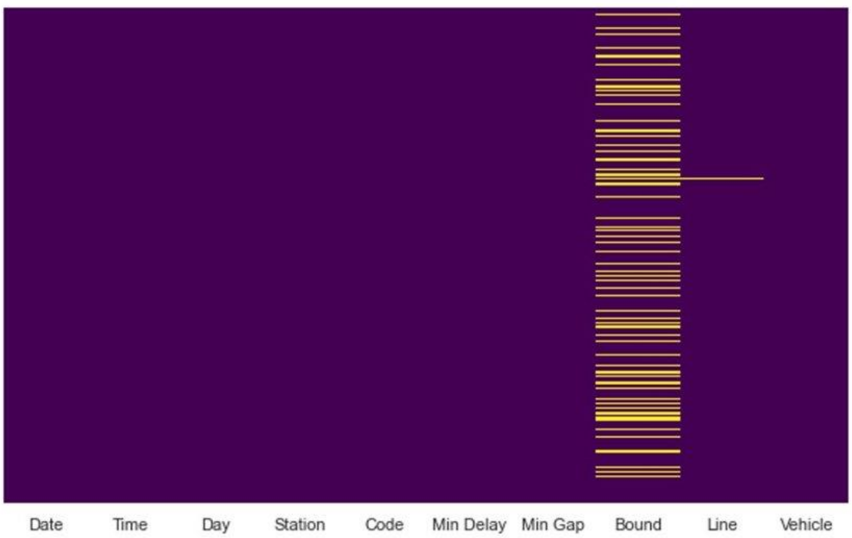


Finally, there seems to be a little correlation between the day and delays since Thursday, Wednesday and Tuesday seem to have the greatest number of delays.



Data Cleaning

During the data cleaning process, the outliers of 'Min Delay' (any extreme data samples that is larger than 30) was first removed since any delay that is more than 30 mins is rare and should consider as abnormal. Then, removed the 'Bound' column as it contains over 22.5% missing data and does not have much value when we test the model. Finally, the null and/or missing value was removed to keep the model's accuracy and performance. The heat map below shows the missing value in yellow.



Here is how the dataset looks after data cleaning.

	Date	Time	Day	Station	Code	Min Delay	Min Gap	Line	Vehicle
0	1/1/2014	2:06	Wednesday	HIGH PARK STATION	SUDP	3	7	BD	5001
1	1/1/2014	2:40	Wednesday	SHEPPARD STATION	MUNCA	0	0	YU	0
2	1/1/2014	3:10	Wednesday	LANSDOWNE STATION	SUDP	3	8	BD	5116
3	1/1/2014	3:20	Wednesday	BLOOR STATION	MUSAN	5	10	YU	5386
4	1/1/2014	3:29	Wednesday	DUFFERIN STATION	MUPAA	0	0	BD	5174
...	...	...	...	...	...	...	...	...	...
143135	6/30/2021	1:23	Wednesday	ST CLAIR STATION	MUIS	0	0	YU	0
143136	6/30/2021	6:00	Wednesday	TORONTO TRANSIT COMMIS	MUO	0	0	SHP	0
143137	6/30/2021	12:40	Wednesday	LESLIE STATION	MUIS	0	0	SHP	0
143138	6/30/2021	20:50	Wednesday	LESLIE STATION	MUTD	9	14	SHP	6171
143139	6/30/2021	0:45	Wednesday	LESLIE STATION	TUMVS	5	10	SHP	6166

142595 rows × 9 columns

Data Preprocessing

Categorical Encoding

Since the dataset contains 7 categorical features and some features can have over 1000 different values, label encoding was decided to use for categorical encoding for this project. With all those additional columns, the model could be hard to run and requires a lot of computational power.

	Date	Time	Day	Station	Code	Min Delay	Min Gap	Line	Vehicle
0	0	966	6	206	181	3	7	39	5001
1	0	1000	6	417	91	0	0	55	0
2	0	1030	6	296	181	3	8	39	5116
3	0	1040	6	27	101	5	10	55	5386
4	0	1049	6	106	95	0	0	39	5174

Feature Selection

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model

Feature Selection Methods:

1. Pearson Correlation

The Pearson correlation method is the most common method to use for numerical variables; it assigns a value between − 1 and 1, where 0 is no correlation, 1 is total positive correlation, and − 1 is total negative correlation. Using this method, we got the correlations stated below:

	feature	cor_value
0	Date	0.000723
1	Day	-0.003113
2	Time	-0.002384
3	Station	-0.007034
4	Line	-0.022629
5	Code	0.915531
6	Vehicle	0.014343
7	Min Gap	0.113221

As per the coefficients above we found that Code has the highest correlation followed by the min gap and other. This concluded that our output is highly dependent on the Code and other features are not that important. But we will see further that this is not the case.

2. Recursive Feature Elimination

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. ... RFE requires a specified number of features to keep, however it is often not known in advance how many features are valid. Selected Features:

```
Recursive Feature Elimination: ['Date', 'Time', 'Station', 'Code', 'Min Gap', 'Line']
```

3. Embedded RandomForestClassifier

Embedded methods combine the qualities of filter and wrapper methods. It's implemented by algorithms that have their own built-in feature selection methods. Some of the most popular examples of these methods are LASSO and RIDGE regression which have inbuilt penalization functions to reduce overfitting. Selected Features:

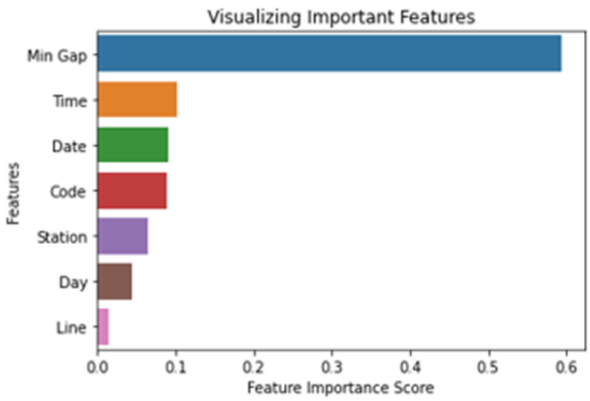
```
Embedded RandomForestClassifier: ['Min Gap']
```

4. Chi-Squared

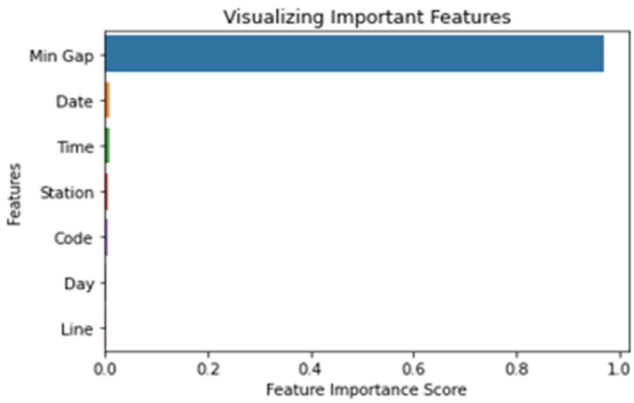
A chi-square test is used in statistics to test the independence of two events. Given the data of two variables, we can get observed count O and expected count E. Chi-Square measures how expected count E and observed count O deviates each other. Selected Features:

```
Chi-squared: ['Time', 'Day', 'Station', 'Code', 'Min Gap', 'Vehicle']
```

After running all the methods, we found that output is most highly dependent on the feature "Min Gap" and not "Code". Listed below are the visualizations of the feature importance as per the used model /algorithm to predict the output.



Random Forest Classifier



Random Forest Regressor

Finally, as per the feature selection methods and the calculated accuracy we select 5 features listed below:

```
# Define X for selected features from Feature Selection Methods

X = df[['Date', 'Time', 'Station', 'Min Gap', 'Code']]
X
```

	Date	Time	Station	Min Gap	Code
0	0	966	206	7	181
1	0	1000	417	0	91
2	0	1030	296	8	181
3	0	1040	27	10	101
4	0	1049	106	0	95
...	...	...	...	...	...
143135	2045	683	463	0	90
143136	2045	1199	516	0	93
143137	2045	220	317	0	90
143138	2045	770	317	14	103
143139	2045	45	317	10	204

142595 rows × 5 columns

Models

Train and Test Data Split

Separating data into training and testing sets is an important part of evaluating data mining models. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct.

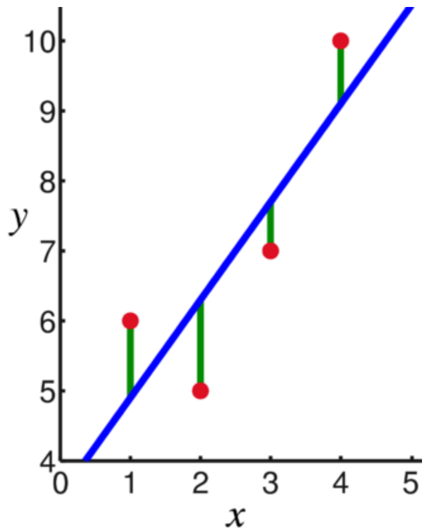
Method to Split: The method used to split our dataset is by train_test_split from the scikit learn library. Random state is set to 0 for the reproducibility of our data split. A test size of 25% was chosen for our model after a comparison of different splits were tested. More on this in the Results section.

```
# Train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                     test_size = 0.25,
                                                   random_state=0)


print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(106946, 5)
(35649, 5)
(106946,)
(35649,)
```

Models Used - Linear Regression

Linear Regression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.



The first algorithm we used to train our model is Linear Regression with the default hyperparameters. Linear regression model has returned an impressive 89.02% test accuracy.

```
# Algorithm - Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

lin_model = LinearRegression()

lin_model = lin_model.fit(X_train, y_train)
lin_model

lin_r2 = r2_score(y_test, lin_model.predict(X_test))
print(f' Linear Regression Test Accuracy: {(lin_r2)}')
```

```
Linear Regression Test Accuracy: 0.8901907365208289
```

Models Used - Decision Tree Regressor

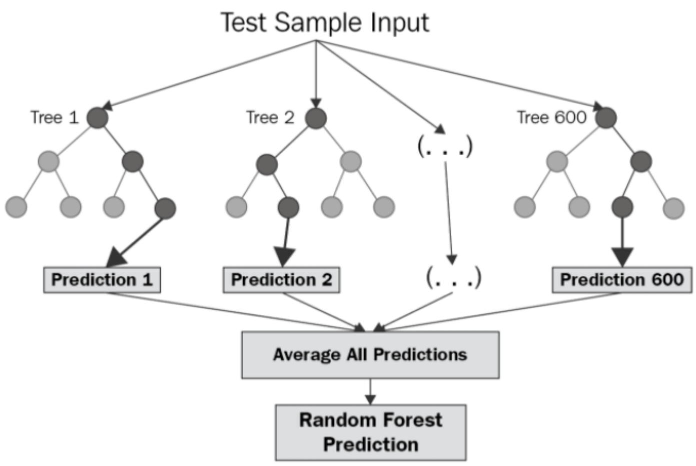
Decision Tree - Regression. Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. This is the next algorithm we implemented for our model, and the decision tree regression algorithm obtained a slightly higher accuracy than linear regression with a 90.84% accuracy.

```
# Algorithm - Decision Tree Regressor

from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor(random_state=0)
dtr.fit(X_train, y_train)
dtr

dtr_r2 = r2_score(y_test, dtr.predict(X_test))
print(f' Decision Tree Regressor Test Accuracy: {(dtr_r2)}')
```

```
 Decision Tree Regressor Test Accuracy: 0.9083528731811269
```

Models Used - Random Forest Regressor

Random Forest Regression is a supervised learning algorithm that uses ensemble learning method for regression. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.



The final algorithm we employed is Random Forest Regressor with number of estimators at 100, random state at 0. With this model we achieved the highest accuracy among the regression models at 94.53%. Following these, the Random Forest Regressor is our model of choice.

```
# Algorithm - Random Forest Regressor

from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(n_estimators=100, random_state =0, n_jobs=-1)
rfr.fit(X_train, y_train)
rfr

rfr_r2 = r2_score(y_test, rfr.predict(X_test))
print(f' Random Forest Regressor Test Accuracy: {(rfr_r2)}')
```

```
 Random Forest Regressor Test Accuracy: 0.9452888781323181
```

Best Random Forest Model - Hyperparameter tuning using RandomizedSearchCV

To select the best hyperparameters for our Random Forest model, we used RandomizedSearchCV as our cross-validation method to evaluate the models. A 30% test size is used for this model as this test size has the highest accuracy of the benchmarked regression models in different test sizes. The selected hyperparameters are n_estimators of 437, max_depth of 9, and min_samples_split at 10. Fitting the model with these hyperparameters and with a 30% test size produced the highest accuracy at 94.86%.

```
param_grid = {'n_estimators': np.arange(100,500),
              'max_depth': np.arange(3,10),
              'min_samples_split': [2, 5, 10]}
```

```
from sklearn.model_selection import RandomizedSearchCV
rscv = RandomizedSearchCV(rfr2, param_grid, n_iter=50, n_jobs=-1)
```

```
rfr_model = rscv.fit(X_train, y_train)
```

```
rfr_model.best_estimator_
```

```
RandomForestRegressor(max_depth=9, min_samples_split=10, n_estimators=437,
                      n_jobs=-1, random_state=0)
```

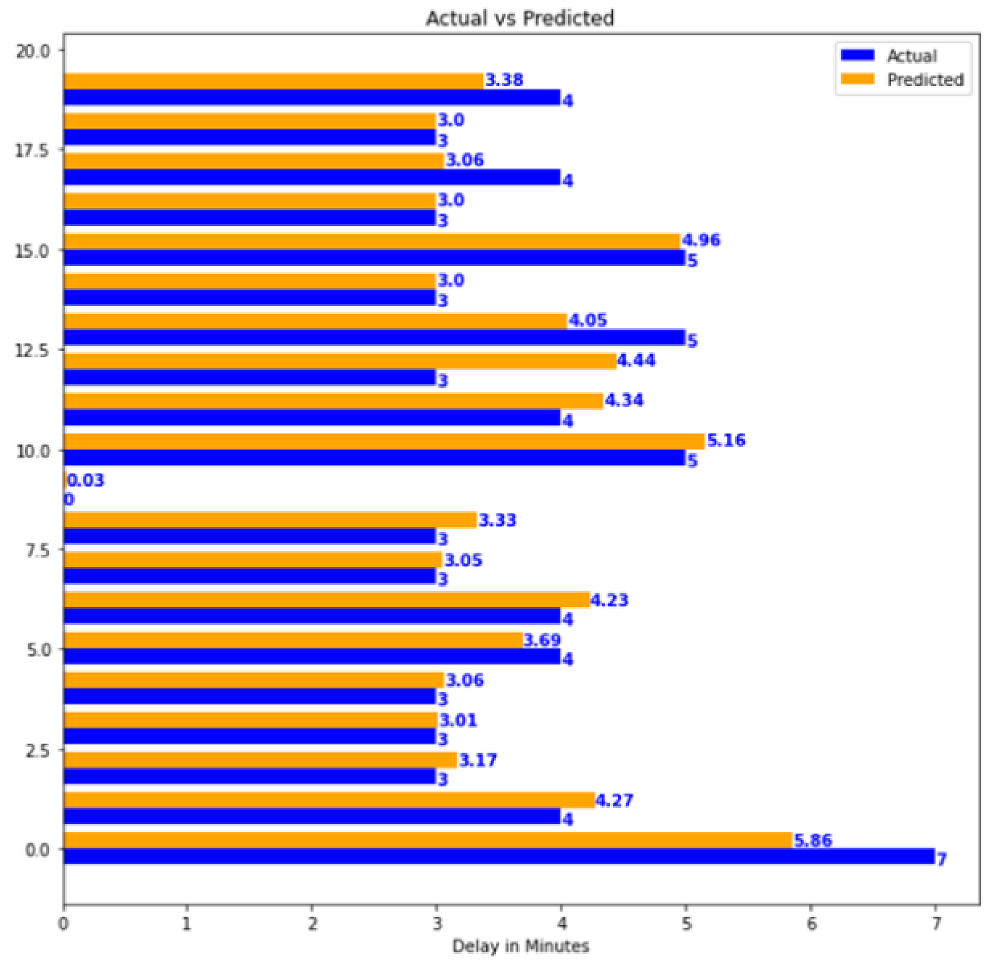
```
rfr_best = rfr_model.best_estimator_.fit(X_train, y_train)
```

```
rfrbest_r2 = r2_score(y_test, rfr_best.predict(X_test))
print(f' Best Random Forest Regressor Test Accuracy: {(rfrbest_r2)}')
```

```
 Best Random Forest Regressor Test Accuracy: 0.9486170215682657
```

Results

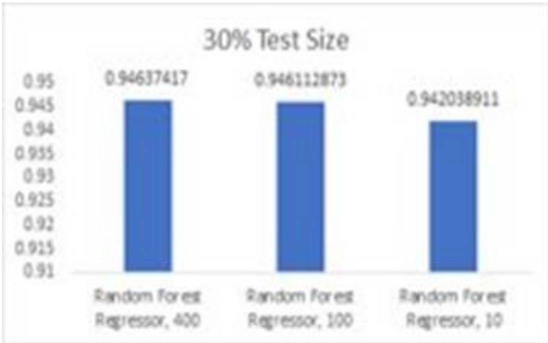
The highest test accuracy achieved by the Random Forest Regressor with using a **30%** train-test-split at 437 estimators is **94.8%**. In case of the Linear Regression model, we achieved an accuracy of **89.17%**. In case of Decision Tree, we got an accuracy of **90.83%**. Below is the plot between the actual value of the result and the value our model predicted.



Here are the outcomes of using Random Forest Regression, Linear Regression and Decision Tree algorithms with a different percentage of train-test-split.

After analyzing these graphs, we can justify that **Random Forest Regression** is the algorithm which is giving out the highest accuracy among the others irrespective of the train-test-split percentage.

Here are the outcomes of using different number of estimators for the Random Forest Regression with a different percentage of train-test-split.



After analyzing these graphs, we can justify that using **437** as the number of estimators for Random Forest Regression gives the highest accuracy among the others irrespective of the train-test-split percentage.

Different Approach

With the regression model, the aim is to accurately predict the train's delay time in minutes, which can be helpful to assess and prepare for expected delays when commuting on the subway. On the problem of delay, classification can also be used based on predicting whether a delay will happen or not.

Classification – Predict on two classes for delay: **True** if Min Delay is > 0 and **False** if Min Delay is 0. In the dataset, the column 'Delayed' is added which contains the labels for our classifier model.

	Date	Time	Day	Station	Code	Min Delay	Min Gap	Line	Vehicle	Delayed
0	1/1/2014	2:06	Wednesday	HIGH PARK STATION	SUDP	3	7	BD	5001	True
1	1/1/2014	2:40	Wednesday	SHEPPARD STATION	MUNCA	0	0	YU	0	False
2	1/1/2014	3:10	Wednesday	LANSDOWNE STATION	SUDP	3	8	BD	5116	True
3	1/1/2014	3:20	Wednesday	BLOOR STATION	MUSAN	5	10	YU	5386	True
4	1/1/2014	3:29	Wednesday	DUFFERIN STATION	MUPAA	0	0	BD	5174	False
...	...	...	...	...	...	...	...	...	...	...
143135	6/30/2021	1:23	Wednesday	ST CLAIR STATION	MUIS	0	0	YU	0	False
143136	6/30/2021	6:00	Wednesday	TORONTO TRANSIT COMMIS	MUO	0	0	SHP	0	False
143137	6/30/2021	12:40	Wednesday	LESLIE STATION	MUIS	0	0	SHP	0	False
143138	6/30/2021	20:50	Wednesday	LESLIE STATION	MUTD	9	14	SHP	6171	True
143139	6/30/2021	0:45	Wednesday	LESLIE STATION	TUMVS	5	10	SHP	6166	True

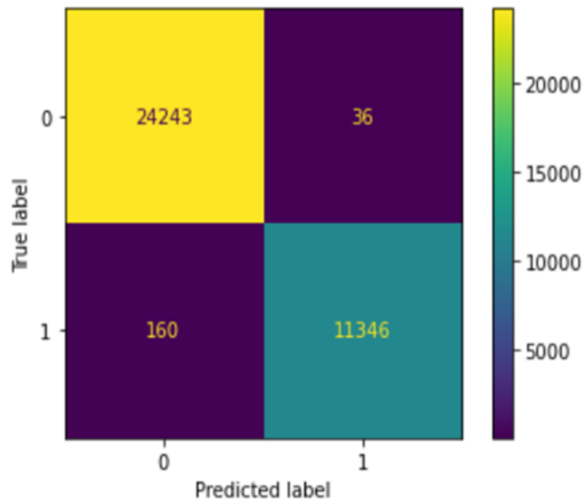
143140 rows × 10 columns

After applying feature selection and preprocessing steps, and training our model with Random Forest Classifier, we were able to achieve a very high accuracy of 99.5%. Iterations may be necessary to evaluate and adjust for overfitting. So far, this result is quite promising for further optimization.

```
# Algorithm - Random Forest Classifier


from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
rfc = rfc.fit(X_train, y_train)
rfc

rfc_accuracy = accuracy_score(y_test, rfc.predict(X_test))
print(f' Random Forest Classifier Test Accuracy: {(rfc_accuracy)}')
```

```
 Random Forest Classifier Test Accuracy: 0.9950629751185166
```

A confusion matrix is a technique for summarizing the performance of a classification algorithm. In the Confusion matrix for Random Forest Classifier, the model was able to classify 24243 correct True values and 11346 correct False values.



Conclusions

In conclusion, based on the output of our problem, we selected **Random Forest Regression** instead of a Random Forest Classifier because what we are trying to predict is **time** (minute delay) which is continuous variable. Currently the **437** is the best no. of estimators we have used to build the most accurate model and the highest test accuracy achieved is **94.8%**.

Future work

An interesting step would be to add open-source historical seasonal data to the TTC dataset to make it possible to data-driven analysis to determine if there is a correlation between the time of delays and seasonal weather (For example minute delay would be higher in winters compared to summers due to less no. of passengers and so on).

Reference

https://ckan0.cf.opendata.inter.prod-toronto.ca/tr/dataset/ttc-subway-delay-data

https://www.kaggle.com/jsun13/toronto-subway-delay-data?select=Toronto-Subway-Delay-Jan-2014-Jun-2021.csv

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