Module Code: CS3AI18

Assignment report Title: Coursework

Student Number: 29015642

Date (when the work completed): 20/03/2023

Actual hrs spent for the assignment: 15

Assignment evaluation (3 key points):

- Learning and implementing Ann's were very interesting and exciting to use.
- The use of google colab's GPU was a great benefactor when doing large neural network models.
- The practicals were very beneficial in learning of machine learning.

Contents

1.0 Abstract

In this study, we address the problem of predicting taxi trip durations using a dataset comprising of various features such as geocoordinates and dates. We implement and compare two different models: a deep learning model using an Artificial Neural Network (ANN) and a traditional machine learning model using Random Forest Regression. The dataset undergoes extensive pre-processing, including feature extraction, dimensionality reduction, and removal of outliers, ultimately leading to high accuracy in predictions, as evidenced by the impressive R2 score. Our findings indicate that the combination of advanced data processing techniques and the appropriate choice of modelling methods can significantly improve taxi trip duration predictions, with potential applications in transportation planning and optimization.

2.0 Background

Predicting taxi trip durations is an essential task in the transportation industry, as it streamlines the optimization of taxi services, reduces passenger waiting times, and enhances overall transportation efficiency. Numerous factors, including traffic conditions, weather, and geographic features, contribute to the challenge of accurately predicting taxi trip durations.

Existing research on taxi trip duration prediction has utilized a variety of machine learning techniques, ranging from traditional approaches such as linear regression and decision trees to more advanced methods like deep learning [1]. However, there is no agreement on the most effective strategy. The role of data pre-processing and feature engineering in improving prediction accuracy remains an active area of investigation [2].

This study aims to augment the current body of knowledge by comparing the performance of an Artificial Neural Network (ANN)-based deep learning model and a Random Forest Regression model in predicting taxi trip durations. Moreover, we examine the impact of advanced data pre-processing techniques, such as dimensionality reduction and time data transformation, on the performance of these models.

In conclusion, our work seeks not only to determine the most effective modelling technique for predicting taxi trip durations but also to shed light on the importance of data pre-processing and feature engineering in achieving high prediction accuracy.

3.0 Exploratory data analysis

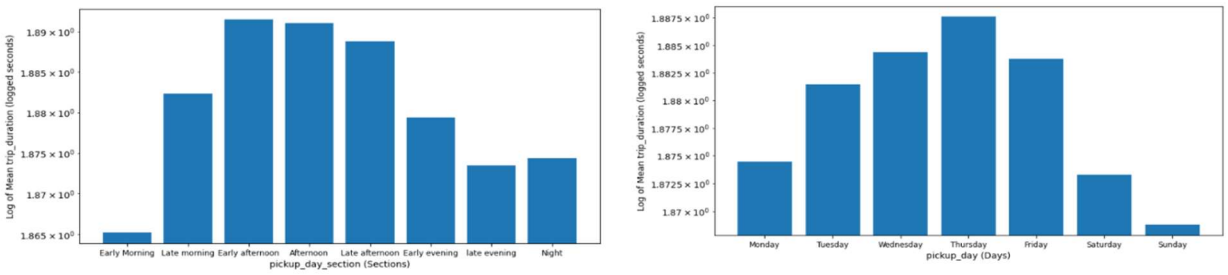
Exploratory Data Analysis (EDA) is crucial for any machine learning model. It is very helpful when gaining insights into the data to identify patterns and trends that can be useful for the modelling process. In this study, we analysed the " NYC Taxi trip duration" dataset from Kaggle, which contains 1,458,644 trip records with 11 different features [3]. (Information regarding each feature can be found within the appendix of this report as expressed in a table).

By looking into the statistical summary (see appendix), there are a lot of issues regarding some of the columns. trip_duration has a maximum value of 3526282 and a minimum value of 1 which are clearly outliers. It is also worth noting that the minimum value for passenger_count is 0 which means there are taxi rides that do not contain any passengers.

The dataset did not contain any Null or NaN values, hence there was no need to do any clean-up of these data types. The columns id and store_and_fwd_flag do not give any useful information or correlation to our target value; hence these columns have been removed. It was found that Vendor ID “2” take on average 213 seconds longer than Vendor ID “1”. This could be a very useful feature when training our machine learning model. (see appendix)

By looking at specific days or parts of the day we can clearly see that there is a trend with the trip duration. Figures 1 shows the mean log of trip_duration against the part of the day. During the early afternoon/afternoon the mean taxi trip duration is much longer than the early morning or Night. The same could be said about Thursday and Sunday. Reasons for these include the reduction of traffic during this time and the less active the roads are, meaning the taxi drivers can get from pickup to dropoff much faster than during the day. On Sunday, the roads tend to be a lot quieter, meaning the time taken is considerably less.

Figure 1 (mean log trip_duration against pickup_day_section and pickup_day)



Some trips had passenger count as 0, so these rows were removed, 60 to be exact.

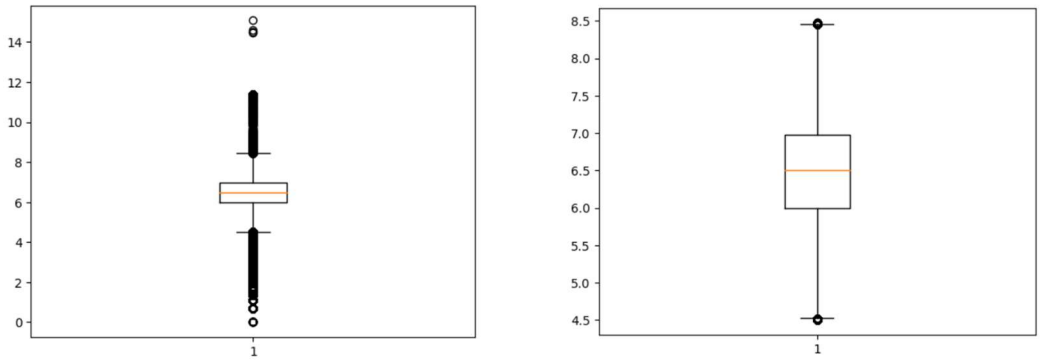
It was found that after calculating distance from the longatitude and latitude values, distance had a linear relationship with trip_duration, implying that this was going to be a very important feature when we go to train our model.(see appendix)

4.0 Data pre-processing and Feature Selection

Data pre-processing and feature selection are critical steps in machine learning, as they help to improve the quality of the data and identify relevant features that are predictive of the target variable [4]. In this study, we performed a comprehensive data pre-processing and feature selection process to prepare the "Taxi trip duration" dataset for modelling.

Firstly, outliers with the original data had to be removed. To quickly remove outliers within any data, Interquartile range (IQR) [5] can be implemented on a relevant column to remove data that does not follow any trends with most of the data. For example, time taken has a max value of 3526282 seconds, whereas most of the data Is within the 400-1000 range. To visualise the outliers, we log scaled the trip_duration variable and plotted it into a boxplot. Figure 2 clearly shows that there are many outliers that do not belong with most of the data. By using IQR's standard 25% to 75%, we can remove the outliers from our data. The result of doing IQR can be shown on the right boxplot where the range of values of the data has been significantly lowered.

Figure 2 (Box plot showing range of trip_duration before and after IQR implementation)



For feature extraction, we calculated the distance in kilometres using the Haversine formula [6] from the longitude and latitude values. We also reduced the dimensionality of the time and date data into more appropriate categorical data, including the month, day, part of the day, and rush hour. We used a reference to a New York personal blog that provided us vital information about the rush hour times in New York, and they were labelled accordingly [20].

Below is a code snippet haversine's formula implemented as a python function.

```
def distance(lat1, lon1, lat2, lon2):
    R = 6371.0
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)
    delta_lat = lat2_rad - lat1_rad
    delta_lon = lon2_rad - lon1_rad
    a = math.sin(delta_lat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(delta_lon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = R * c
    return d
```

Additionally, from the cyclical data, we converted the features into their cos and sin representations to account for the circular nature of the data. However, after evaluating the impact of this feature on the model, the decision to not use it as the final features was made. This is because both neural networks and random forest perform better with categorical or binary data and the performance of the models are hindered with these types of representations [7].

When creating the distance feature through haversine's formula, some of the distances can be classified as outliers as some distances are as small as 0 and some distances are as large as 2000 kilometres. To remove these outliers, a threshold value for both distance and trip_duration was made. In addition, a new column "avg_speed" was made from calculation of distance / trip_duration to calculate the average speed of the taxi trip. An average speed of below 6 kilometers per hour and above 40 kilometres per hour was considered an outlier. (The python code for this implementation can be found in the appendix)

After all the pre-processing, I was left with 1,240,758 rows with 7 columns. Figure 3 contains a snapshot of the dataframe in its fully cleaned and transformed state.

Figure 3 (taxi dataframe displayed)

	vendor_id	trip_duration	distance	pickup_day	pickup_month	pickup_day_section	pickup_rush_hour
0	2	6.120297	0.404478	1	3	5	2
1	1	6.496775	0.590842	7	6	8	3
2	2	7.661056	1.853967	2	1	2	3
3	2	6.061457	0.395750	3	4	6	2
4	2	6.075346	0.172766	6	3	3	2
...	...	...	...	...	...	...	...
1458637	1	6.633318	1.263436	7	4	6	2
1458638	2	6.025866	0.840467	2	2	8	3
1458640	1	6.484635	1.800031	7	1	1	3
1458641	2	6.638568	2.057273	5	4	1	3
1458642	1	5.921578	0.088527	2	1	3	2

1240758 rows × 7 columns

Overall, our data pre-processing and feature extraction process has resulted in clean and relevant features that will be used to train and test our models.

5.0 Machine learning model

Using the fully pre-processed data, numerous machine learning models were trained and tested using sklearn's test train split functionality [8].

After experimenting with various machine learning algorithms such as decision tree regression and n nearest neighbour regression, it was found that random forest regression yielded the best accuracy and efficiency when compare to the other models. A table in the appendix Containing the results of the different models can be found. Similarly, the same approach was done for artificial neural networks at which alternations of sequential networks were tested to find the ideal network with n number of deep layers. The most highly performing neural network was a sequential network with 4 layers containing dense relu activation layers, 64, 32, 16 and 1 with an optimizer of "adam". (The results table alongside the code snippets for each model can be found in the appendix)

5.1 Random Forest Regression

Random Forest Regression was the chosen model for this study. Random Forest Regression is an ensemble learning method that takes multiple decision trees to improve the performance of the model. It randomly selects a subset of features and build a decision tree based on that subset. The processes are repeated x number of times where the final performance score of the model is based on the average of the predictions of all the decision trees. [9]

Random Forest Regression has several advantages over other machine learning algorithms. These advantages include its ability to handle many features and its robustness to noise and outliers. With the fact that our dataset contains a significant amount of rows and columns, Random Forest Regression is well-suited to identify the most important features and provide accurate predictions as well as provide the best computational efficiency when compared to other models such as k nearest neighbour and gradient booster regression. By randomly selecting a subset of features and building decision trees based on them, this type of regression can achieve high accuracy while avoiding overfitting to the noise and outliers in the data [10].

Below are two code snippet taken from maching_learning.ipynb showing how to train and test a Random Forest Regression model and displaying the results of the model.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, r
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

```
Mean Squared Error: 0.09285868207296308
R-squared score: 0.7748166735174706
```

5.2 Artificial Neural Network

An Artificial Neural Network (ANN) is a machine learning algorithm that draws inspiration from the structure and function of the human brain. Much like biological neural networks, ANNs consist of interconnected nodes or units, referred to as neurons [11]. These neurons are arranged in layers, allowing the network to process and learn complex relationships between input and output variables.

Artificial Neural Networks offer several advantages over other machine learning algorithms, making them an ideal choice for various applications. These advantages include their ability to handle nonlinear relationships and their robustness to noise and outliers. Given that the taxi dataset comprises a considerable number of rows and columns, ANNs are well-suited for portraying the underlying patterns and providing accurate predictions. Using multiple hidden layers and nonlinear activation functions, ANNs can achieve high accuracy without overfitting to noise and outliers in the data.

Below are two code snippet taken from deep_learning.ipynb showing how the testing and training of an artificial neural network and displaying the results of the model.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense

X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_s
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=5, batch_size=64)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/5
13571/13571 [==============================] - 28s 2ms/step - loss: 0.1611
Epoch 2/5
13571/13571 [==============================] - 27s 2ms/step - loss: 0.0974
Epoch 3/5
13571/13571 [==============================] - 27s 2ms/step - loss: 0.0960
Epoch 4/5
13571/13571 [==============================] - 27s 2ms/step - loss: 0.0955
Epoch 5/5
13571/13571 [==============================] - 28s 2ms/step - loss: 0.0949
11633/11633 [==============================] - 16s 1ms/step
R2 score: 0.7724502420791968
MSE: 0.0938345256578618
```

6.0 Evaluation

The performance metrics used in this study are explained below.

MSE – How close a regression line is to a set of points; it is calculated by taking the distances from the points to the regression line and squaring them [12].

R2 – An evaluation measure for regression-based machine models. It is also known as the coefficient of determination. It is the most important metric that can be used to evaluate the performance of a regression model. It is a calculation of the difference between the true values in the dataset and the predictions made by the model [13].

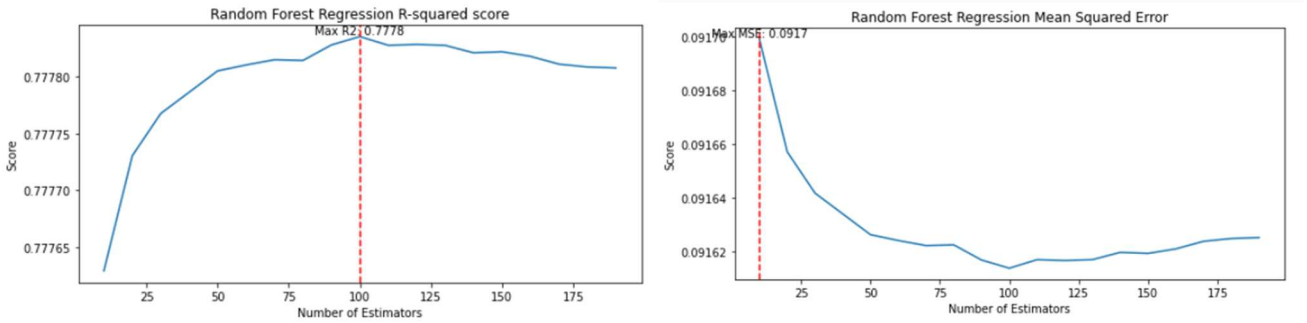
Before we compare the performance of the two models, it is important that we evaluate each model by tweaking their hyperparameters and apply cross validation testing. Hyperparameters are certain requirements that can change the performance and efficiency of a model. A model can perform very poorly if these values are not configured correctly. To deploy hyperparameter tweaking, we can iterate through different hyperparameter values and test and train a model with each different value. After the iterations, we can plot the performances against the parameter values, and we identify the optimum hyperparameter value.

6.1 Machine Learning Hyperparameters

For the machine learning model, we investigate the hyperparamters ‘n_estimators’ and ‘depth’

The number of estimators parameter is responsible for the number of individual decision trees the model will create. For example, if the model has n_estimators = 10 then the model will have 10 decision trees. Figure 4 reveals the results of iterations of n_estimators and its affect on R2 and MSE scores (code can be found in the appendix). The model is performing at its best when the r2 score is at its maximum and when the mse is at the minimum. In this case, this value is 100, therefore the best value for n_estimators is 100.

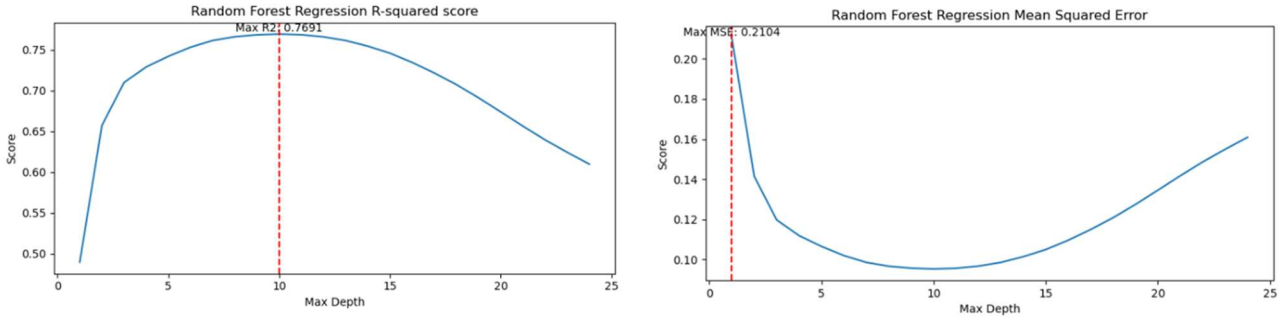
Figure 4 (Hyper parameter number of estimators against score metrics)



Another hyperparameter called depth was also investigated. Depth is referred as the maximum number of levels in each decision tree that makes up the forest. Increasing the depth over a certain value can cause overfitting, therefore careful tweaking of this value is required to fully optimise the model.

Figure 5 shows the results of increments of 1 for the depth. The graphs show the effect on r2 and mse scores. As clearly shown in the graph, the optimum depth value is 10 as r2 score is at its highest and mse is at its lowest.

Figure 5 (Hyper parameter depth against score metrics)



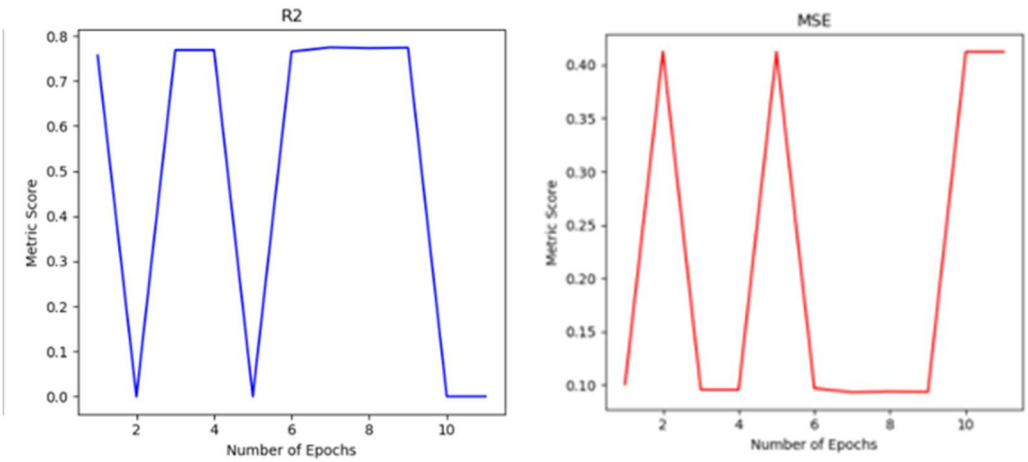
6.1 Neural Network Hyperparameters

For the neural network model, we investigate the hyperparameters 'epochs and 'batch_size'.

An epoch is defined as the number of times the dataset is passed through the training process of a neural network. The effect of the epoch size on the performance of ANN models is investigated by training several models with different epoch values for a given dataset.

Figures 6 visualises the iteration process by plotting a line of best fit through each iteration point. The R2 score and MSE are plotted against each iteration. The graphs show a very skewed performance at which the model did the best at approximately epoch =7.

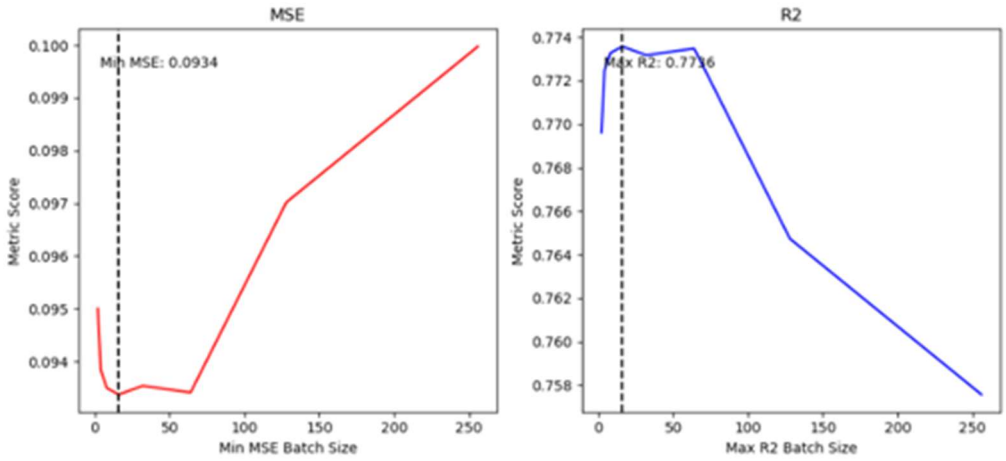
Figure 6 (Hyper parameter epochs against score metrics)



Batch size refers to the number of training samples that are processed at each iteration of the model's training sequence. By dividing the data into batches, the model can update its weights more frequently and see more samples in each epoch. This significantly improves the processing time per epoch as there is much less data to process. Having a smaller value for batch size results in a longer processing time as the number refers to how many divisions the original data is being split into.

This hyperparameter was also tested through a range of 2 to 250, at which the most optimum value was batch_size = 16.

Figure 7 (Hyper parameter batch size against score metrics)



6.2 K-Fold Cross-Validation

Now that the models have had their hyperparameters tweaked, it is time to now compare the performances of the two models. However, before we compare the performances, we must apply k-fold cross validation on both models to see their genuine performance. Cross-validation using k-fold is a popular machine learning technique that involves the splitting of the dataset into k equal-sized folds [14]. An average of the performance of each K fold is then taken to determine the overall performance of the model. As a result, this process provides a much more reliable assessment of the performance of any machine learning model. By only relying on one test-train split for the model, the performance of the model may be highly dependent on that instance of a single test-train split. This may lead to overfitting whereby the model performs worse on unseen data. Therefore the need for cross validation is very important for any machine learning model. The results of k-fold cross validation of both models can be shown in figures 8 and 9.

Figure 8 (K-fold cross validation on neural network)

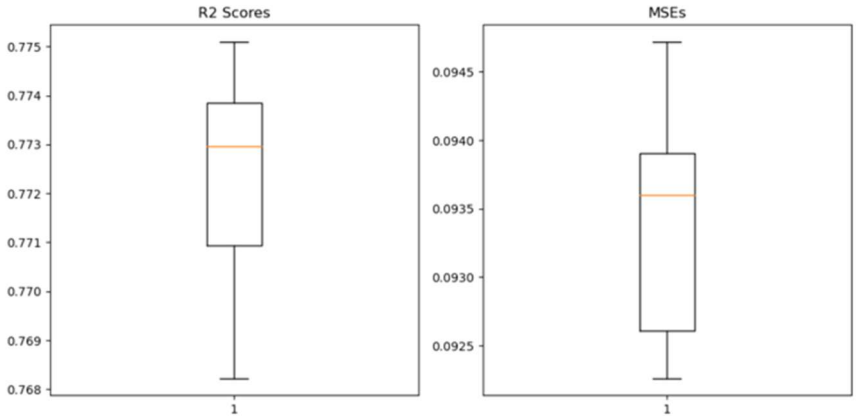


Figure 9 (K-fold cross validation on Random Forest)

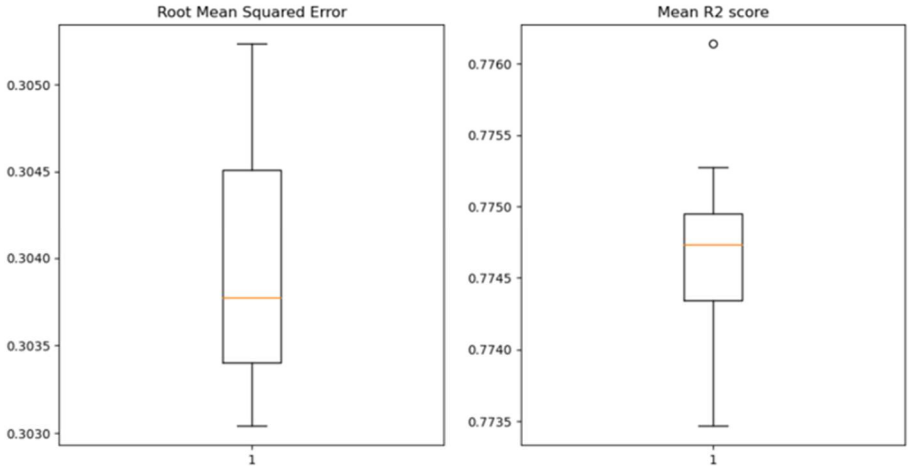
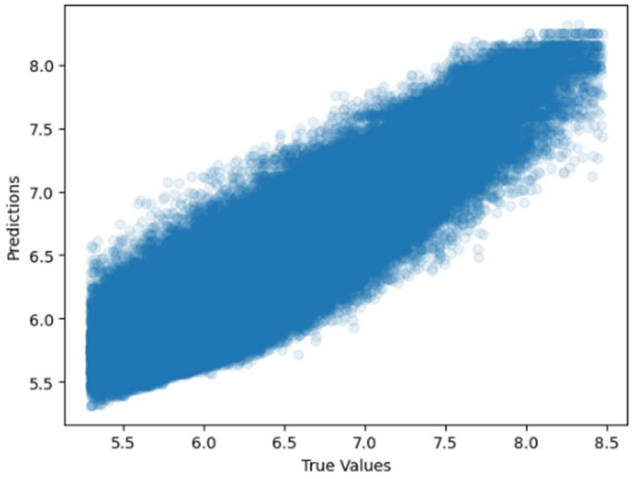


Table 1 (Performance metrics from the k-fold cross validation)

Model	Mean R2 Score	Mean MSE Score
Artificial Neural Network	0.7723	0.0933
Random Forest Regression	0.7746	0.0924

It was found that Random Forest Regression performs slightly better than the artificial neural network as it has a higher r2 and lower mse score. The IQR for r2 score for Random Forest was significantly smaller than the neural network. This could be a good indicator that the model is performing well in a set range of r2 scores, rather the neural network is more skewed in terms of average performance.

Figure 10 (True vs Random Forest predictions)



To conclude, a plot of the best performing model of true vs predicted is shown below. Showing a linear relationship between the true values and Random Forest's predictions. The graph reveals that the model can accurately predict the trip duration of a taxi rides in New York City. Therefore, giving the potential use and deployment for transportation companies to help optimise their services and give useful information to customers.

6.3 Limitations

One limitation of using artificial neural networks is the computational complexity. ANNs use up a lot of resources and take a very long time to train. A reason for this is because ANNs require many iterations that are used to optimise the weights of the model which get more computationally expensive for larger datasets. This is especially true for our dataset as it contains 1.2 million rows (after pre-processing). Random forest was a better choice as it was much faster and computationally efficient than an ANN.

Some other limitations of the models include the lack of useful features such as weather conditions, traffic jams, road works and special events. These types of features can be difficult to measure and/or predict in real time, making it challenging to implement into a machine learning model. All real-world factors that can heavily affect the model's performance should be included with the model to help improve accuracy and help deal more with reasonings for why some taxi trips took longer than they should have.

In addition, the model may not perform well with places outside New York, as the data has been specifically trained on information that is taken from New York City. The model may fail to generalise other locations where there might be different factors that can affect the trip duration of the taxi.

7.0 Conclusion, future work recommendations

After hyperparameter tweaking, both the machine learning and neural network models performed similarly to each other, but Random Forest yielded better results in terms of performance metrics and computational cost. Random Forest processed significantly faster than ANN, resulting in a more robust and efficient model.

Further work in this area could include the implementation of geo mapping [15] the longitude and latitude values to get a better representation of where the locations are in New York. Pairing this with Dijkstra's shortest path algorithm [16] could be used to find the optimum route from pickup to drop-off location. Currently, the distance is calculated as a single vector, whereas the implementation of remapping alongside Dijkstra's algorithm will significantly improve the distance calculation by considering pathing routes instead of a straight line.

In conclusion, our model has shown promising results in predicting taxi trip durations in New York City. Future work could focus on implementing additional features such as weather conditions, traffic jams, and special events to improve accuracy further. Kaggle offers a dataset that contains the weather information for New York in 2016 [17]. This means that both datasets can be merged to create new features for the machine learning model to further improve the performance.

8.0 references

[1] Zhao, Z., Chi, Y., Ding, Z., Chang, M. and Cai, Z. (2023). Latent Semantic Sequence Coding Applied to Taxi Travel Time Estimation. ISPRS International Journal of Geo-Information, [online] 12(2), p.44. Available at: https://www.mdpi.com/2220-9964/12/2/44/pdf [Accessed 23 Mar. 2023].

[2] MIT Press. (n.d.). Fundamentals of Machine Learning for Predictive Data Analytics. [online] Available at: https://mitpress.mit.edu/9780262029445/fundamentals-of-machine-learning-for-predictive-data-analytics/ [Accessed 2 Mar. 2023].

[3] kaggle.com. (n.d.). New York City Taxi Trip Duration. [online] Available at: https://www.kaggle.com/competitions/nyc-taxi-trip-duration [Accessed 23 Mar. 2023].

[4] KDnuggets. (n.d.). Importance of Pre-Processing in Machine Learning. [online] Available at: https://www.kdnuggets.com/2023/02/importance-preprocessing-machine-learning.html [Accessed 23 Mar. 2023].

[5] Wikipedia Contributors (2019). Interquartile range. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Interquartile_range.

[6] Wikipedia Contributors (2019). Haversine formula. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Haversine_formula.

[7] Stack Overflow. (n.d.). machine learning - Cyclic ordinal features in random forest. [online] Available at: https://stackoverflow.com/questions/47350569/cyclic-ordinal-features-in-random-forest [Accessed 23 Mar. 2023].

[8] Scikit-learn.org. (2018). sklearn.model_selection.train_test_split — scikit-learn 0.20.3 documentation. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[9] Team, G.L. (2020). Random Forest Algorithm- An Overview | Understanding Random Forest. [online] GreatLearning. Available at: https://www.mygreatlearning.com/blog/random-forest-algorithm/.

[10] Ellis, C. (2021). Random forest overfitting. [online] Crunching the Data. Available at: https://crunchingthedata.com/random-forest-overfitting/#:~:text=In%20general%2C%20random%20forests%20are [Accessed 23 Mar. 2023].

[11] Analytics Vidhya. (2021). Artificial Neural Network | How does Artificial Neural Network Work. [online] Available at: https://www.analyticsvidhya.com/blog/2021/04/artificial-neural-network-its-inspiration-and-the-working-mechanism/.

[12] Statistics How To. (n.d.). Mean Squared Error: Definition and Example. [online] Available at: https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/.

[13] Kharwal, A. (2021). R2 Score in Machine Learning. [online] Data Science | Machine Learning | Python | C++ | Coding | Programming | JavaScript. Available at: https://thecleverprogrammer.com/2021/06/22/r2-score-in-machine-learning/.

[14] SciKit-Learn (2009). 3.1. Cross-validation: evaluating estimator performance — scikit-learn 0.21.3 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/cross_validation.html.

[15] Stewart, R. (2018). GeoPandas 101: Plot any data with a latitude and longitude on a map. [online] Medium. Available at: https://towardsdatascience.com/geopandas-101-plot-any-data-with-a-latitude-and-longitude-on-a-map-98e01944b972.

[16] freeCodeCamp.org. (2020). Dijkstra's Shortest Path Algorithm - A Detailed and Visual Introduction. [online] Available at: https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/#:~:text=Dijkstra.

[17] www.kaggle.com. (n.d.). Weather data in New York City - 2016. [online] Available at: https://www.kaggle.com/datasets/mathijs/weather-data-in-new-york-city-2016 [Accessed 23 Mar. 2023].

[18] IBM (n.d.). What is Machine Learning? | IBM. [online] www.ibm.com. Available at: https://www.ibm.com/uk-en/topics/machine-learning.

[19] The Editors of Encyclopedia Britannica (2019). latitude and longitude | Description & Diagrams. In: Encyclopædia Britannica. [online] Available at: https://www.britannica.com/science/latitude.

[20] Knispel, J. (2022). Worst Traffic Times in New York City. [online] Law Offices of Jay S. Knispel Personal Injury Lawyers. Available at: https://jknylaw.com/blog/worst-traffic-times-in-new-york-city/.

Appendix

Deep Learning files located on google colab

https://colab.research.google.com/drive/14-MtJZrDjMBvq5m6R2PeFYfcfpJ6m7ej?usp=sharing

Link to fully preprocessed data as well as the raw data

https://livereadingac-my.sharepoint.com/:f:/g/personal/jz015642_student_reading_ac_uk/EnXpjL79KDdOueocVjuIiRUBURU4A69QPmSITLVVrAoO7g?e=3TdCRB

3.0

In [425]:
```
integer_summary = taxi_train.describe().round().astype(int)
print(integer_summary)
```

```
       vendor_id  passenger_count  pickup_longitude  pickup_latitude  \
count    1458644          1458644           1458644          1458644
mean           2                2               -74               41
std            0                1                 0                0
min            1                0              -122               34
25%            1                1               -74               41
50%            2                1               -74               41
75%            2                2               -74               41
max            2                9               -61               52

       dropoff_longitude  dropoff_latitude  trip_duration
count            1458644           1458644        1458644
mean                 -74                41            959
std                    0                 0           5237
min                 -122                32              1
25%                  -74                41            397
50%                  -74                41            662
75%                  -74                41           1075
max                  -61                44        3526282
```

:
```
vendor_ids = taxi_train['vendor_id'].unique()
for vendor_id in vendor_ids:
    mean_duration = taxi_train.loc[taxi_train['vendor_id'] == vendor_id, 'trip_duration'].mean()
    print(f"Vendor ID {vendor_id}: Mean Trip Duration = {mean_duration:.2f}")
```

```
Vendor ID 2: Mean Trip Duration = 1058.64
Vendor ID 1: Mean Trip Duration = 845.44
```

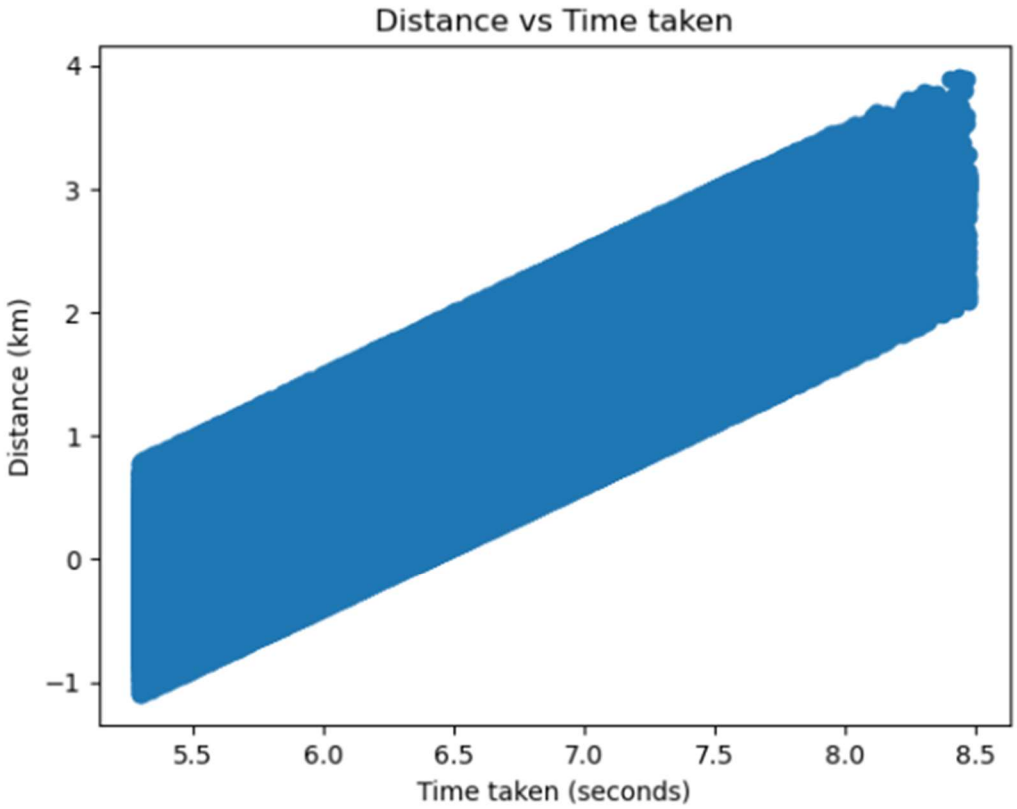
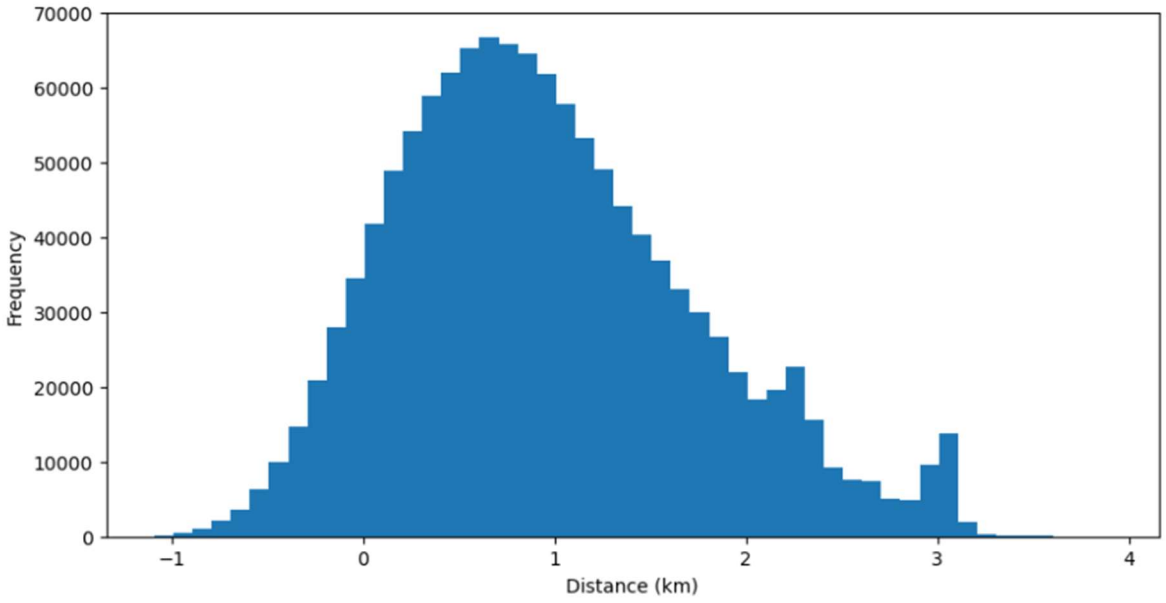
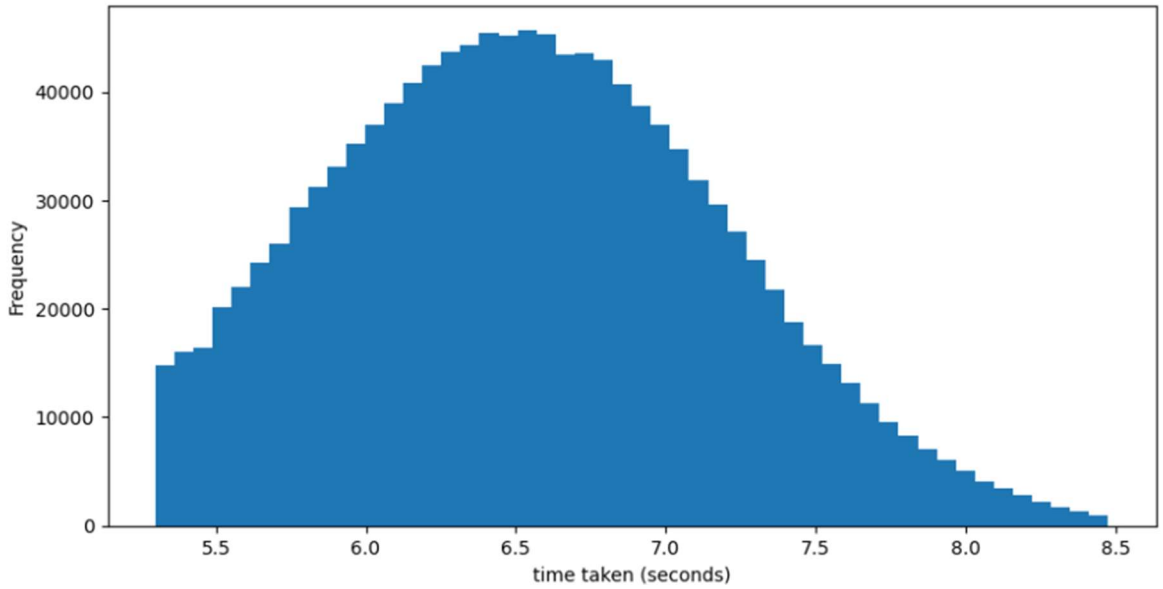


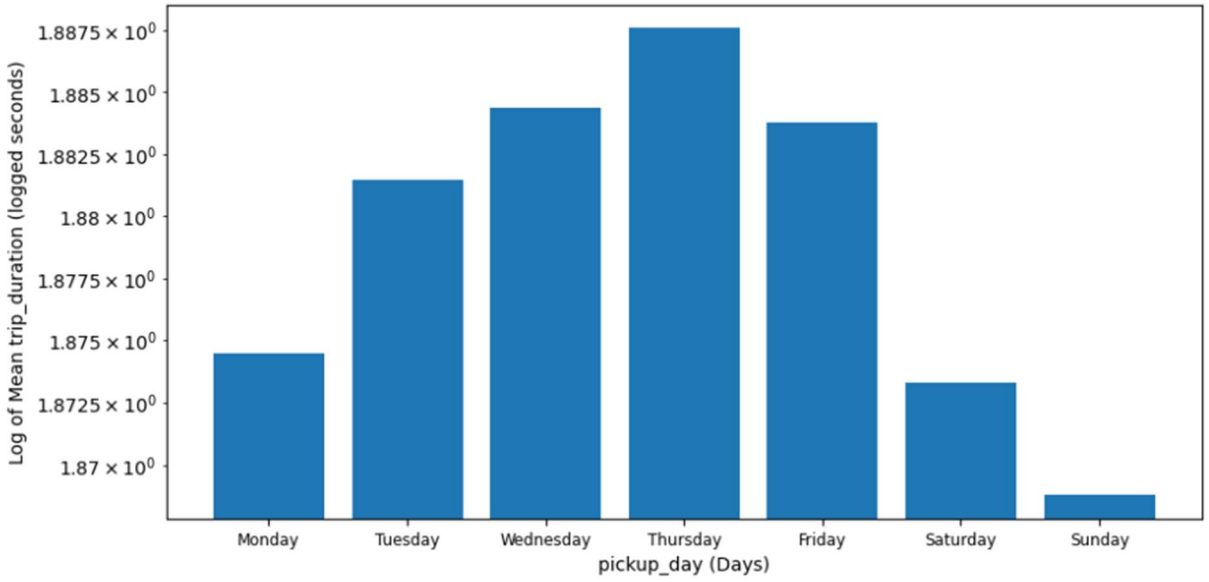
Table showing the features with their relevant description.

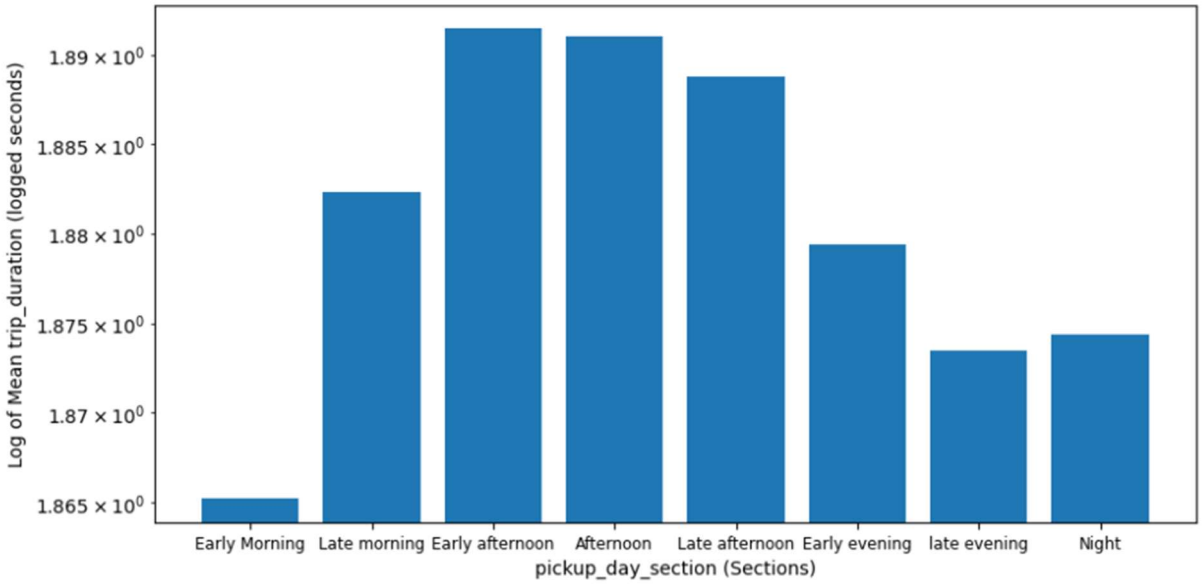
id	a unique identifier for each trip
vendor_id	a code indicating the provider associated with the trip record
pickup_datetime	date and time when the meter was engaged
dropoff_datetime	date and time when the meter was disengaged
passenger	
Pickup_longitude	the longitude where the meter was engaged
Pickup_latitude	the latitude where the meter was engaged
dropoff_longitude	the longitude where the meter was disengaged
dropoff_latitude	the latitude where the meter was disengaged
store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip
trip_duration	duration of the trip in seconds

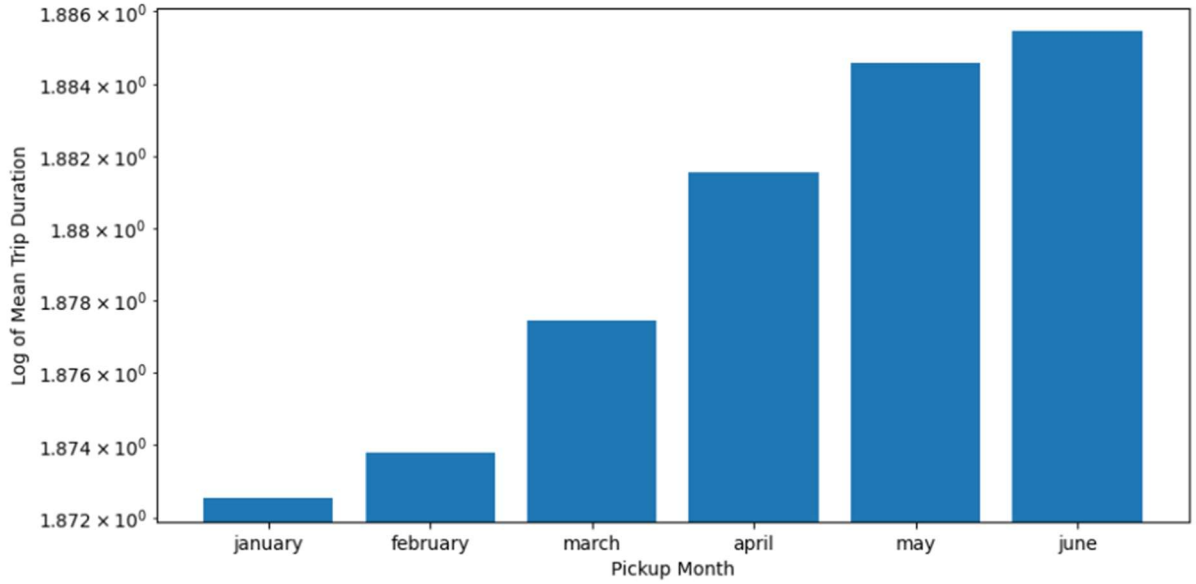
4.0

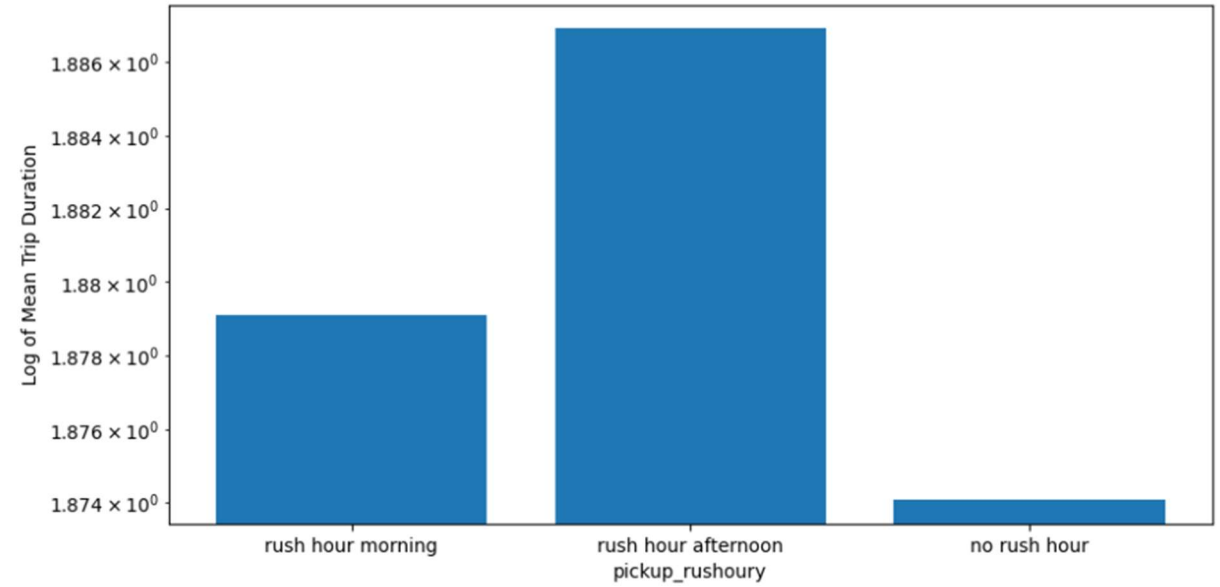

70000
60000
50000
40000
30000
20000
10000
0
Frequency
−1
0
1
2
3
4
Distance (km)


40000
30000
20000
10000
0
Frequency
5.5
6.0
6.5
7.0
7.5
8.0
8.5
time taken (seconds)


1.8875 × 10^0
1.885 × 10^0
1.8825 × 10^0
1.88 × 10^0
1.8775 × 10^0
1.875 × 10^0
1.8725 × 10^0
1.87 × 10^0
Log of Mean trip_duration (logged seconds)
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
pickup_day (Days)


Log of Mean trip_duration (logged seconds)
1.89 × 10^0
1.885 × 10^0
1.88 × 10^0
1.875 × 10^0
1.87 × 10^0
1.865 × 10^0
Early Morning
Late morning
Early afternoon
Afternoon
Late afternoon
Early evening
late evening
Night
pickup_day_section (Sections)


Log of Mean Trip Duration
1.886 × 10^0
1.884 × 10^0
1.882 × 10^0
1.88 × 10^0
1.878 × 10^0
1.876 × 10^0
1.874 × 10^0
1.872 × 10^0
january
february
march
april
may
june
Pickup Month


Log of Mean Trip Duration
1.886 × 10^0
1.884 × 10^0
1.882 × 10^0
1.88 × 10^0
1.878 × 10^0
1.876 × 10^0
1.874 × 10^0
rush hour morning
rush hour afternoon
no rush hour
pickup_rushoury

```
df = df[df['distance'] >= 0.25]
df['avg_speed'] = df['distance'] / (df['trip_duration'] / 3600)
df = df[df['trip_duration'] >= 200]
df = df[df['avg_speed'] <= 40]
df = df[df['avg_speed'] >= 6]
df = df.drop('avg_speed',axis=1)
```

```
def get_section_of_day(param_time):
    '''This function is used to convert time in the format "hour:minute:second" into the part of the day, eg Morning or afternoon
    The function uses https://www.britannica.com/dictionary/eb/qa/parts-of-the-day-early-morning-late-morning-etc as a standared
    measure for the allocation of the part of the day'''
    hour = param_time.hour
    if hour >= 5 and hour <= 8:
        #return 'Early morning'
        return 1
    elif hour >= 8 and hour <= 12:
        #return 'Late morning'
        return 2
    elif hour >= 12 and hour <= 15:
        #return 'Early afternoon'
        return 3
    elif hour >= 15 and hour <= 16:
        #return 'Afternoon'
        return 4
    elif hour >= 16 and hour <= 17:
        #return 'Late afternoon'
        return 5
    elif hour >= 17 and hour <= 19:
        #return 'Early evening'
        return 6
    elif hour >= 19 and hour <= 21:
        #return 'Late evening'
        return 7
    elif hour >= 21 or hour < 5:
        #return 'Night'
        return 8

taxi_train['pickup_day_section'] = taxi_train['pickup_time'].apply(get_section_of_day)
taxi_train['dropoff_day_section'] = taxi_train['dropoff_time'].apply(get_section_of_day)
taxi_train
```

In [436]:
```
import math

def distance(lat1, lon1, lat2, lon2):
    R = 6371.0
    lat1_rad = math.radians(lat1)
    lon1_rad = math.radians(lon1)
    lat2_rad = math.radians(lat2)
    lon2_rad = math.radians(lon2)
    delta_lat = lat2_rad - lat1_rad
    delta_lon = lon2_rad - lon1_rad
    a = math.sin(delta_lat / 2)**2 + math.cos(lat1_rad) * math.cos(lat2_rad) * math.sin(delta_lon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
    d = R * c
    return d
```

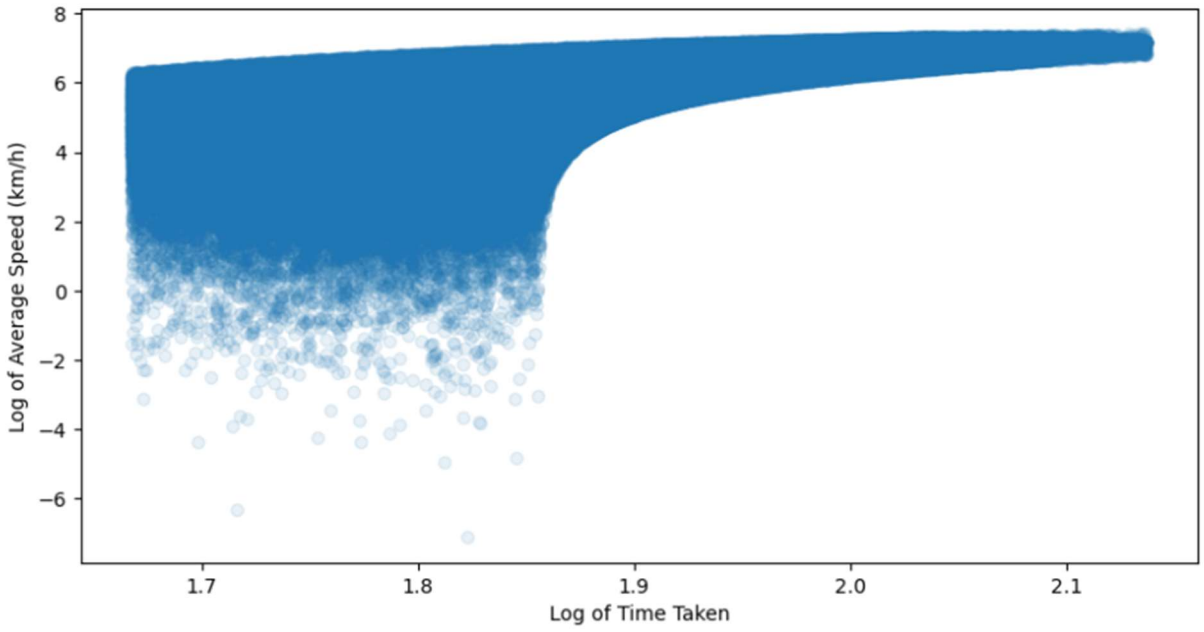
In [437]:
```
taxi_train['distance'] = taxi_train.apply(lambda x: distance(x['pickup_latitude'], x['pickup_longitude'], x['dropoff_latitude'],
```

4]:
```
def get_rush_hour(param_time):
    hour = param_time.hour
    if hour >=8 and hour <= 9:
        #return 'Morning rush hour'
        return 1
    elif hour >= 13 and hour <= 19:
        #return 'Afternoon rush hour'
        return 2
    else:
        return 3
        #return 'No rush hour'
```

5]:
```
taxi_train['pickup_rush_hour'] = taxi_train['pickup_time'].apply(get_rush_hour)
taxi_train['dropoff_rush_hour'] = taxi_train['dropoff_time'].apply(get_rush_hour)
```

```python
def convert_weekday_to_integer(param_weekdayName):
    week_days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']
    return week_days.index(param_weekdayName.lower()) +1
#df =df.drop('store_and_fwd_flag',axis=1)
df['pickup_day'] = df['pickup_day'].apply(convert_weekday_to_integer)
df['dropoff_day'] = df['dropoff_day'].apply(convert_weekday_to_integer)
def convert_month_to_integer(param_monthName):
    months = ['january', 'february', 'march', 'april', 'may', 'june', 'july', 'august', 'september', 'october', 'november', 'dece
    return months.index(param_monthName.lower()) + 1
```

```python
# Convert pickup_day and dropoff_day to cosine and sine values
df['pickup_day_cos'] = np.cos(2 * np.pi * df['pickup_day'] / 7)
df['pickup_day_sin'] = np.sin(2 * np.pi * df['pickup_day'] / 7)


# Convert pickup_month and dropoff_month to cosine and sine values
df['pickup_month_cos'] = np.cos(2 * np.pi * df['pickup_month'] / 12)
df['pickup_month_sin'] = np.sin(2 * np.pi * df['pickup_month'] / 12)


# Convert pickup_day_section and dropoff_day_section to cosine and sine values
df['pickup_day_section_cos'] = np.cos(2 * np.pi * df['pickup_day_section'] / 8)
df['pickup_day_section_sin'] = np.sin(2 * np.pi * df['pickup_day_section'] / 8)


# Convert pickup_rush_hour and dropoff_rush_hour to cosine and sine values
df['pickup_rush_hour_cos'] = np.cos(2 * np.pi * df['pickup_rush_hour'] / 3)
df['pickup_rush_hour_sin'] = np.sin(2 * np.pi * df['pickup_rush_hour'] / 3)
```



5.0

Machine learning table

Model	R2 score	mse
Random Forest Regression	0.7748166735174706	0.09285868207296308
Decision tree regression	0.7728617024124562	0.09366485206404565
N nearest neighbour regression	0.7334185864182337	0.10992998068296528
Ada booster regression	0.7529829809524184	0.10186222575466765

Gradient booster regression	0.7758346869884016	0.09243888460962599

Machine learning code

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, rando
rf = RandomForestRegressor(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

In [479]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, rando
tree = DecisionTreeRegressor(max_depth=10, random_state=42)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

```
Mean Squared Error: 0.09366485206404565
R-squared score: 0.7728617024124562
```

In [480]:
```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, rando
ab = AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=5), n_estimators=100, learning_rate=0.1, random_state=42)
ab.fit(X_train, y_train)
y_pred = ab.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

```
Mean Squared Error: 0.10186222575466765
R-squared score: 0.7529829809524184
```

In [481]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, rando
gb = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
gb.fit(X_train, y_train)
y_pred = gb.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

```
Mean Squared Error: 0.09243888460962599
R-squared score: 0.7758346869884016
```

In [482]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
X_train, X_test, y_train, y_test = train_test_split(df.drop(['trip_duration'], axis=1), df['trip_duration'], test_size=0.3, rand
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
R2 score: 0.7334185864182337
MSE: 0.10992998068296528
```

Artificial neural network table

Model	**Number of layers**	**Layer**	**R2 score (3.sf)**	**MSE (3.sf)**
Sequential	6	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(4, activation='relu')) (Dense(1, activation='linear'))	0.773	0.0923
Sequential	2	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(1, activation='linear')	0.766	0.0965
Sequential	3	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(1, activation='linear'))	0.774	0.0932
Sequential	4	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(1, activation='linear'))	0.775	0.0929
Sequential	5	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(1, activation='linear'))	0.772	0.0937
Sequential	7	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(4, activation='relu')) (Dense(2, activation='relu'))	-7.87e-06	0.412

		(Dense(1, activation='linear'))		

ANN code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/10
27142/27142 [==============================] - 102s 4ms/step - loss: 0.1964
Epoch 2/10
27142/27142 [==============================] - 92s 3ms/step - loss: 0.0982
Epoch 3/10
27142/27142 [==============================] - 97s 4ms/step - loss: 0.0969
Epoch 4/10
27142/27142 [==============================] - 92s 3ms/step - loss: 0.0964
Epoch 5/10
27142/27142 [==============================] - 92s 3ms/step - loss: 0.0957
Epoch 6/10
27142/27142 [==============================] - 96s 4ms/step - loss: 0.0954
Epoch 7/10
27142/27142 [==============================] - 93s 3ms/step - loss: 0.0949
Epoch 8/10
27142/27142 [==============================] - 96s 4ms/step - loss: 0.0945
Epoch 9/10
27142/27142 [==============================] - 96s 4ms/step - loss: 0.0943
Epoch 10/10
27142/27142 [==============================] - 98s 4ms/step - loss: 0.0941
11633/11633 [==============================] - 19s 2ms/step
R2 score: 0.7748309103386745
MSE: 0.09285281124551527
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/10
27142/27142 [==============================] - 74s 3ms/step - loss: 0.1642
Epoch 2/10
27142/27142 [==============================] - 72s 3ms/step - loss: 0.0975
Epoch 3/10
27142/27142 [==============================] - 72s 3ms/step - loss: 0.0964
Epoch 4/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0958
Epoch 5/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0954
Epoch 6/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0951
Epoch 7/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0949
Epoch 8/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0947
Epoch 9/10
27142/27142 [==============================] - 71s 3ms/step - loss: 0.0947
Epoch 10/10
27142/27142 [==============================] - 72s 3ms/step - loss: 0.0946
11633/11633 [==============================] - 18s 2ms/step
R2 score: 0.7658740586809175
MSE: 0.09654634155015297
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/10
27142/27142 [==============================] - 86s 3ms/step - loss: 0.1333
Epoch 2/10
27142/27142 [==============================] - 81s 3ms/step - loss: 0.0972
Epoch 3/10
27142/27142 [==============================] - 79s 3ms/step - loss: 0.0957
Epoch 4/10
27142/27142 [==============================] - 85s 3ms/step - loss: 0.0950
Epoch 5/10
27142/27142 [==============================] - 86s 3ms/step - loss: 0.0946
Epoch 6/10
27142/27142 [==============================] - 90s 3ms/step - loss: 0.0945
Epoch 7/10
27142/27142 [==============================] - 77s 3ms/step - loss: 0.0941
Epoch 8/10
27142/27142 [==============================] - 75s 3ms/step - loss: 0.0939
Epoch 9/10
27142/27142 [==============================] - 78s 3ms/step - loss: 0.0937
Epoch 10/10
27142/27142 [==============================] - 77s 3ms/step - loss: 0.0937
11633/11633 [==============================] - 17s 1ms/step
R2 score: 0.7738138207655503
MSE: 0.09327222772179597
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/10
27142/27142 [==============================] - 83s 3ms/step - loss: 0.1321
Epoch 2/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0973
Epoch 3/10
27142/27142 [==============================] - 81s 3ms/step - loss: 0.0959
Epoch 4/10
27142/27142 [==============================] - 81s 3ms/step - loss: 0.0951
Epoch 5/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0947
Epoch 6/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0944
Epoch 7/10
27142/27142 [==============================] - 81s 3ms/step - loss: 0.0941
Epoch 8/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0940
Epoch 9/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0939
Epoch 10/10
27142/27142 [==============================] - 82s 3ms/step - loss: 0.0937
11633/11633 [==============================] - 18s 2ms/step
R2 score: 0.7746210813716057
MSE: 0.09293933826173402
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

```
Epoch 1/10
27142/27142 [==============================] - 90s 3ms/step - loss: 0.1443
Epoch 2/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0967
Epoch 3/10
27142/27142 [==============================] - 89s 3ms/step - loss: 0.0951
Epoch 4/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0945
Epoch 5/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0941
Epoch 6/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0939
Epoch 7/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0937
Epoch 8/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0936
Epoch 9/10
27142/27142 [==============================] - 89s 3ms/step - loss: 0.0935
Epoch 10/10
27142/27142 [==============================] - 88s 3ms/step - loss: 0.0934
11633/11633 [==============================] - 19s 2ms/step
R2 score: 0.7726110395953949
MSE: 0.09376821770483529
```

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=32)
y_pred = model.predict(X_test)
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print('R2 score:', r2)
print('MSE:', mse)
```

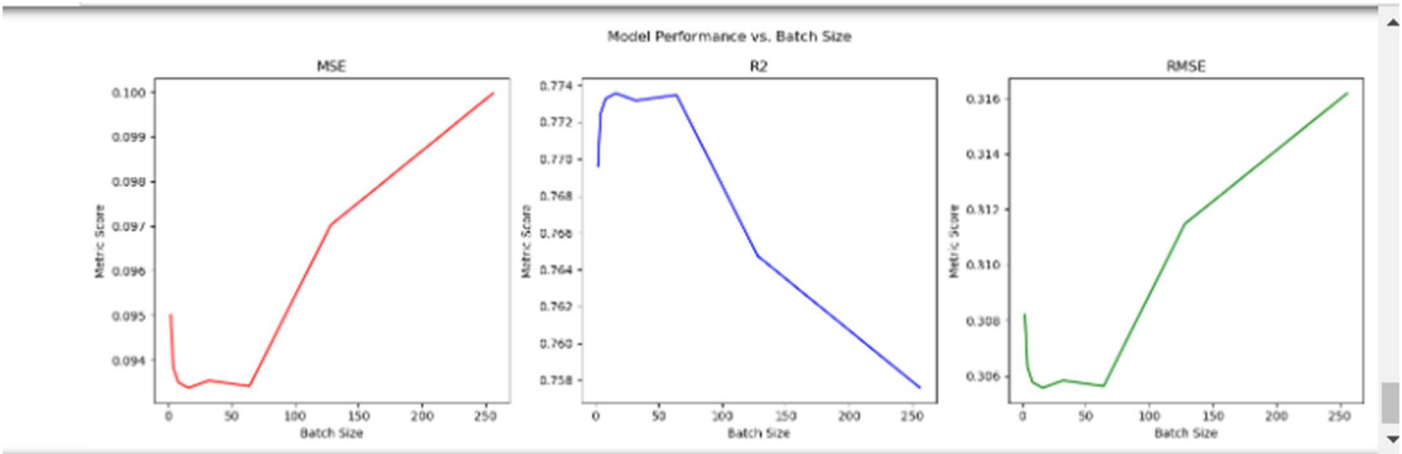
```
Epoch 1/10
27142/27142 [==============================] - 94s 3ms/step - loss: 4.3015
Epoch 2/10
27142/27142 [==============================] - 94s 3ms/step - loss: 0.4092
Epoch 3/10
27142/27142 [==============================] - 94s 3ms/step - loss: 0.4092
Epoch 4/10
27142/27142 [==============================] - 93s 3ms/step - loss: 0.4092
Epoch 5/10
27142/27142 [==============================] - 96s 4ms/step - loss: 0.4092
Epoch 6/10
27142/27142 [==============================] - 95s 4ms/step - loss: 0.4092
Epoch 7/10
27142/27142 [==============================] - 94s 3ms/step - loss: 0.4092
Epoch 8/10
27142/27142 [==============================] - 96s 4ms/step - loss: 0.4092
Epoch 9/10
27142/27142 [==============================] - 95s 4ms/step - loss: 0.4092
Epoch 10/10
27142/27142 [==============================] - 95s 3ms/step - loss: 0.4092
11633/11633 [==============================] - 19s 2ms/step
R2 score: -7.873200537300562e-06
MSE: 0.4123725082957821
```

6.0

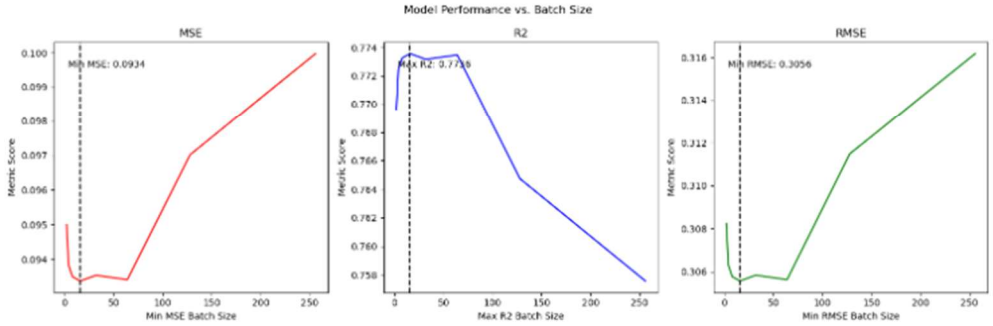
Batch size hyper parametisation

In [10]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt


X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_
def create_model(batch_size):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=10, batch_size=batch_size)
    return model
mse_list = []
r2_list = []
rmse_list = []

batch_sizes = [4, 16, 32, 64, 128, 256]
for batch_size in batch_sizes:
    model = create_model(batch_size)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mse)
    mse_list.append(mse)
    r2_list.append(r2)
    rmse_list.append(rmse)
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].plot(batch_sizes, mse_list, 'r')
axs[0].set_title('MSE')
axs[0].set_xlabel('Batch Size')
axs[0].set_ylabel('Metric Score')
axs[1].plot(batch_sizes, r2_list, 'b')
axs[1].set_title('R2')
axs[1].set_xlabel('Batch Size')
axs[1].set_ylabel('Metric Score')
axs[2].plot(batch_sizes, rmse_list, 'g')
axs[2].set_title('RMSE')
axs[2].set_xlabel('Batch Size')
axs[2].set_ylabel('Metric Score')
plt.suptitle('Model Performance vs. Batch Size')
plt.tight_layout()
plt.show()
```

```
11633/11633 [==============================] - 23s 2ms/step
```



In [20]:
```
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
max_r2_index = np.argmax(r2_list)
max_r2_score = r2_list[max_r2_index]
min_mse_index = np.argmin(mse_list)
min_mse = mse_list[min_mse_index]
min_rmse_index = np.argmin(rmse_list)
min_rmse = rmse_list[min_rmse_index]
axs[0].plot(batch_sizes, mse_list, 'r')
axs[0].set_title('MSE')
axs[0].set_xlabel('Batch Size')
axs[0].set_ylabel('Metric Score')
axs[0].axvline(x=batch_sizes[min_mse_index], color='k', linestyle='--')
axs[0].text(0.05, 0.9, f'Min MSE: {min_mse:.4f}', transform=axs[0].transAxes)
axs[0].set_xlabel('Min MSE Batch Size')
axs[1].plot(batch_sizes, r2_list, 'b')
axs[1].set_title('R2')
axs[1].set_xlabel('Batch Size')
axs[1].set_ylabel('Metric Score')
axs[1].axvline(x=batch_sizes[max_r2_index], color='k', linestyle='--')
axs[1].text(0.05, 0.9, f'Max R2: {max_r2_score:.4f}', transform=axs[1].transAxes)
axs[1].set_xlabel('Max R2 Batch Size')
axs[2].plot(batch_sizes, rmse_list, 'g')
axs[2].set_title('RMSE')
axs[2].set_xlabel('Batch Size')
axs[2].set_ylabel('Metric Score')
axs[2].axvline(x=batch_sizes[min_rmse_index], color='k', linestyle='--')
axs[2].text(0.05, 0.9, f'Min RMSE: {min_rmse:.4f}', transform=axs[2].transAxes)
axs[2].set_xlabel('Min RMSE Batch Size')
plt.suptitle('Model Performance vs. Batch Size')
plt.tight_layout()
plt.show()
```

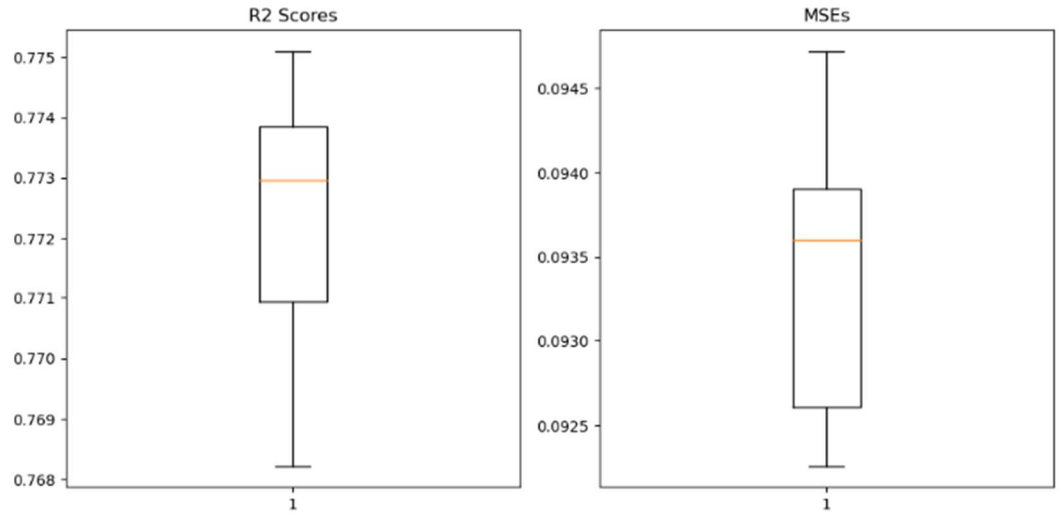


K fold for best model

In [24]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import r2_score
from keras.metrics import MeanSquaredError, MeanAbsoluteError, RootMeanSquaredError, MeanAbsolutePercentageError
k = 10
kf = KFold(n_splits=k, random_state=42, shuffle=True)
r2_scores = []
mses = []
for train_index, test_index in kf.split(taxi_data):
    X_train, X_test = taxi_data.iloc[train_index].drop(['trip_duration'], axis=1), taxi_data.iloc[test_index].drop(['trip_durati
    y_train, y_test = taxi_data.iloc[train_index]['trip_duration'], taxi_data.iloc[test_index]['trip_duration']
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=7, batch_size=16)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)s
    r2_scores.append(r2)
    mses.append(mse)
mean_r2 = np.mean(r2_scores)
std_r2 = np.std(r2_scores)
mean_mse = np.mean(mses)
std_mse = np.std(mses)
print('Mean R2 score:', mean_r2)
print('Standard deviation of R2 scores:', std_r2)
print('Mean MSE:', mean_mse)
print('Standard deviation of MSEs:', std_mse)
```

Standard deviation of MSEs: 0.000787802955271857

In [30]:
```
import matplotlib.pyplot as plt
mean_r2 = np.mean(r2_scores)
mean_mse = np.mean(mses)
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].boxplot(r2_scores)
axs[0].set_title('R2 Scores')
axs[1].boxplot(mses)
axs[1].set_title('MSEs')
fig.tight_layout()
plt.show()
```



Hyperparameter tweaking Alternating depth values

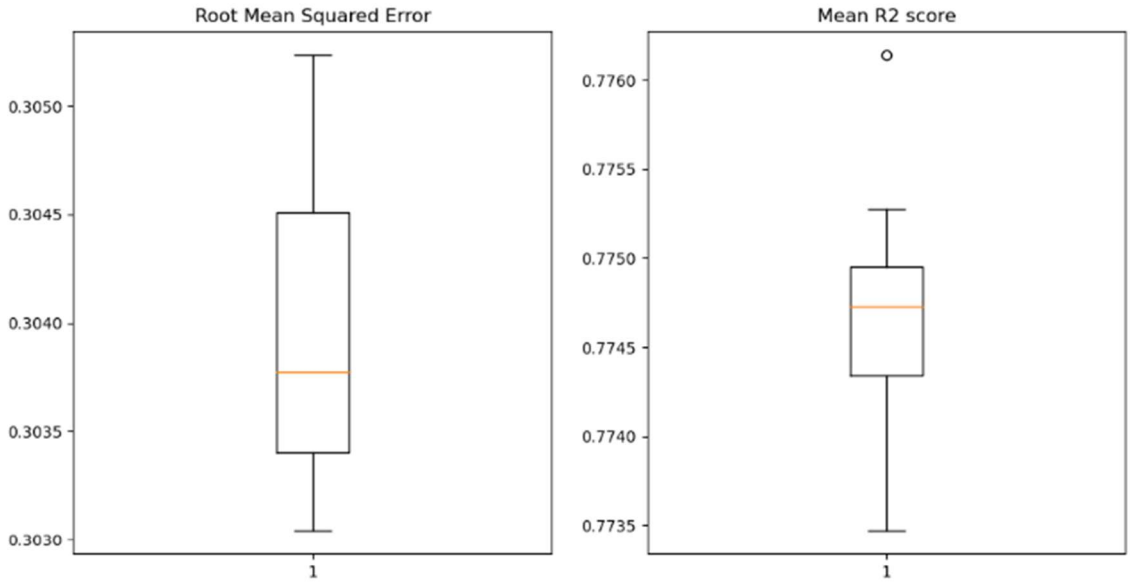
k fold with best hyperparameters k =10 randomforest n_estimators = 100, depth =10

In [104]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score, mean_squared_log_error
import numpy as np
import matplotlib.pyplot as plt

num_folds = 10
X = taxi_data.drop(['trip_duration'], axis=1)
y = taxi_data['trip_duration']

kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
mse_scores = []
rmse_scores = []
rmsle_scores = []
r2_scores = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X)):
    print(f'Fold {fold + 1}')
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    rf = RandomForestRegressor(n_estimators=100,max_depth=10, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    rmsle = np.sqrt(mean_squared_log_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    print(f'MSE: {mse:.2f}')
    print(f'RMSE: {rmse:.2f}')
    print(f'RMSLE: {rmsle:.2f}')
    print(f'R-squared: {r2:.2f}')
    mse_scores.append(mse)
    rmse_scores.append(rmse)
    rmsle_scores.append(rmsle)
    r2_scores.append(r2)

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
axes[0].boxplot(rmse_scores)
axes[0].set_title('Root Mean Squared Error')
axes[1].boxplot(r2_scores)
axes[1].set_title('Mean R2 score')
plt.show()
```

```
Fold 1
MSE: 0.09
```

```
Fold 1
MSE: 0.09
RMSE: 0.31
RMSLE: 0.04
R-squared: 0.77
Fold 2
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 3
MSE: 0.09
RMSE: 0.31
RMSLE: 0.04
R-squared: 0.78
Fold 4
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 5
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.78
Fold 6
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 7
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 8
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 9
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
Fold 10
MSE: 0.09
RMSE: 0.30
RMSLE: 0.04
R-squared: 0.77
```

```
RMSLE: 0.04
R-squared: 0.77
```



In [106]:
```
mean_r2 = np.mean(r2_scores)
mean_mse = np.mean(mse_scores)
print('Mean R2 score:', mean_r2)
print('Mean MSE:', mean_mse)
```

```
Mean R2 score: 0.7746809624274605
Mean MSE: 0.09240789871753803
```

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)

_mse = []
_r2 = []
estimator = 10
for x in range(1, 20):
    estimator += 10
    rf = RandomForestRegressor(n_estimators=estimator, max_depth=10, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    _mse.append(mse)
    _r2.append(r2)

fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(8, 8))
print(len(_mse))
print(len(_r2))
ax1.plot(range(10, 200, 10), _mse, label='Mean Squared Error')

ax1.set_xlabel('Number of Estimators')
ax1.set_ylabel('Score')
ax1.set_title('Random Forest Regression Mean Squared Error')
max_mse_index = np.argmin(_mse)
ax1.axvline(max_mse_index*10+10, color='red', linestyle='--')
ax1.text(max_mse_index*10+10, max(_mse), f'Max MSE: {max(_mse):.4f}', ha='center', va='bottom')
ax2.plot(range(10, 200, 10), _r2, label='R-squared score')
ax2.set_xlabel('Number of Estimators')
ax2.set_ylabel('Score')
ax2.set_title('Random Forest Regression R-squared score')
max_r2_index = np.argmax(_r2)
ax2.axvline(max_r2_index*10+10, color='red', linestyle='--')
ax2.text(max_r2_index*10+10, max(_r2), f'Max R2: {max(_r2):.4f}', ha='center', va='bottom')
plt.tight_layout()
plt.show()
```

In [64]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

num_folds = 10

X = df.drop(['trip_duration'], axis=1)
y = df['trip_duration']
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
mse_scores = []
r2_scores = []
y_pred_list = []
y_test_list = []

for fold, (train_idx, test_idx) in enumerate(kf.split(X)):
    print(f'Fold {fold + 1}')
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    rf = DecisionTreeRegressor(max_depth=10, random_state=42)
    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f'MSE: {mse:.2f}')
    print(f'R-squared: {r2:.2f}')
    mse_scores.append(mse)
    r2_scores.append(r2)
    y_pred_list.append(y_pred)
    y_test_list.append(y_test)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))
axes[0].boxplot(mse_scores)
axes[0].set_title('Mean Squared Error')
axes[1].boxplot(r2_scores)
axes[1].set_title('R-squared')
plt.show()
```

```
Fold 1
MSE: 90067.46
R-squared: 0.77
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(taxi_data.drop(['trip_duration'], axis=1), taxi_data['trip_duration'], test_size=0.3, random_state=42)

# Define a function to create and train the deep learning model
def create_model(batch_size, epochs):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size)
    return model

# Define lists to store the results
mse_list = []
r2_list = []
rmse_list = []

# Train and evaluate the model for each batch size and epochs
batch_sizes = [2,4,8, 16, 32, 64, 128, 256]
epochs_list = [2, 4, 8, 10, 16, 20, 25, 30]
for batch_size in batch_sizes:
    for epochs in epochs_list:
        # Create and train the model
        model = create_model(batch_size, epochs)

        # Make predictions on the testing set
        y_pred = model.predict(X_test)

        # Calculate the metrics
        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        rmse = np.sqrt(mse)

        # Append the results to the lists
        mse_list.append(mse)
        r2_list.append(r2)
        rmse_list.append(rmse)

# Plot the results in subplots
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

axs[0].plot(epochs_list, mse_list, 'r')
axs[0].set_title('MSE')
axs[0].set_xlabel('Number of Epochs')
axs[0].set_ylabel('Metric Score')

axs[1].plot(epochs_list, r2_list, 'b')
```

Definitions and equations

latitude and longitude, coordinate system by means of which the position or location of any place on Earth's surface can be determined and described. [19]

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles. [6]

$$
\begin{aligned}
d &= 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + (1 - \operatorname{hav}(\varphi_1 - \varphi_2) - \operatorname{hav}(\varphi_1 + \varphi_2)) \cdot \operatorname{hav}(\lambda_2 - \lambda_1)}\right) \\
&= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \left(1 - \sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) - \sin^2\left(\frac{\varphi_2 + \varphi_1}{2}\right)\right) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad [9] \\
&= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right).
\end{aligned}
$$

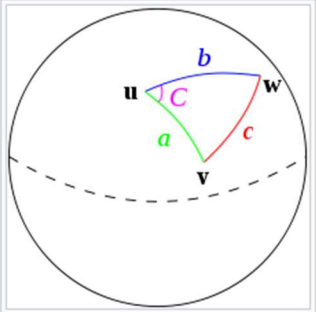
The law of haversines [edit]

Given a unit sphere, a "triangle" on the surface of the sphere is defined by the great circles connecting three points u, v, and w on the sphere. If the lengths of these three sides are a (from u to v), b (from u to w), and c (from v to w), and the angle of the corner opposite c is C, then the law of haversines states:[10]

$$\operatorname{hav}(c) = \operatorname{hav}(a - b) + \sin(a)\sin(b)\operatorname{hav}(C).$$

Since this is a unit sphere, the lengths a, b, and c are simply equal to the angles (in radians) subtended by those sides from the center of the sphere (for a non-unit sphere, each of these arc lengths is equal to its central angle multiplied by the radius R of the sphere).

In order to obtain the haversine formula of the previous section from this law, one simply considers the special case where u is the north pole, while v and w are the two points whose separation d is to be determined. In that case, a and b are $\frac{\pi}{2} - \varphi_{1,2}$ (that is, the, co-latitudes), C is the longitude separation $\lambda_2 - \lambda_1$, and c is the desired $\frac{d}{R}$. Noting that $\sin(\frac{\pi}{2} - \varphi) = \cos(\varphi)$, the haversine formula immediately follows.



Spherical triangle solved by the law of haversines

To derive the law of haversines, one starts with the spherical law of cosines:

$$\cos(c) = \cos(a)\cos(b) + \sin(a)\sin(b)\cos(C).$$

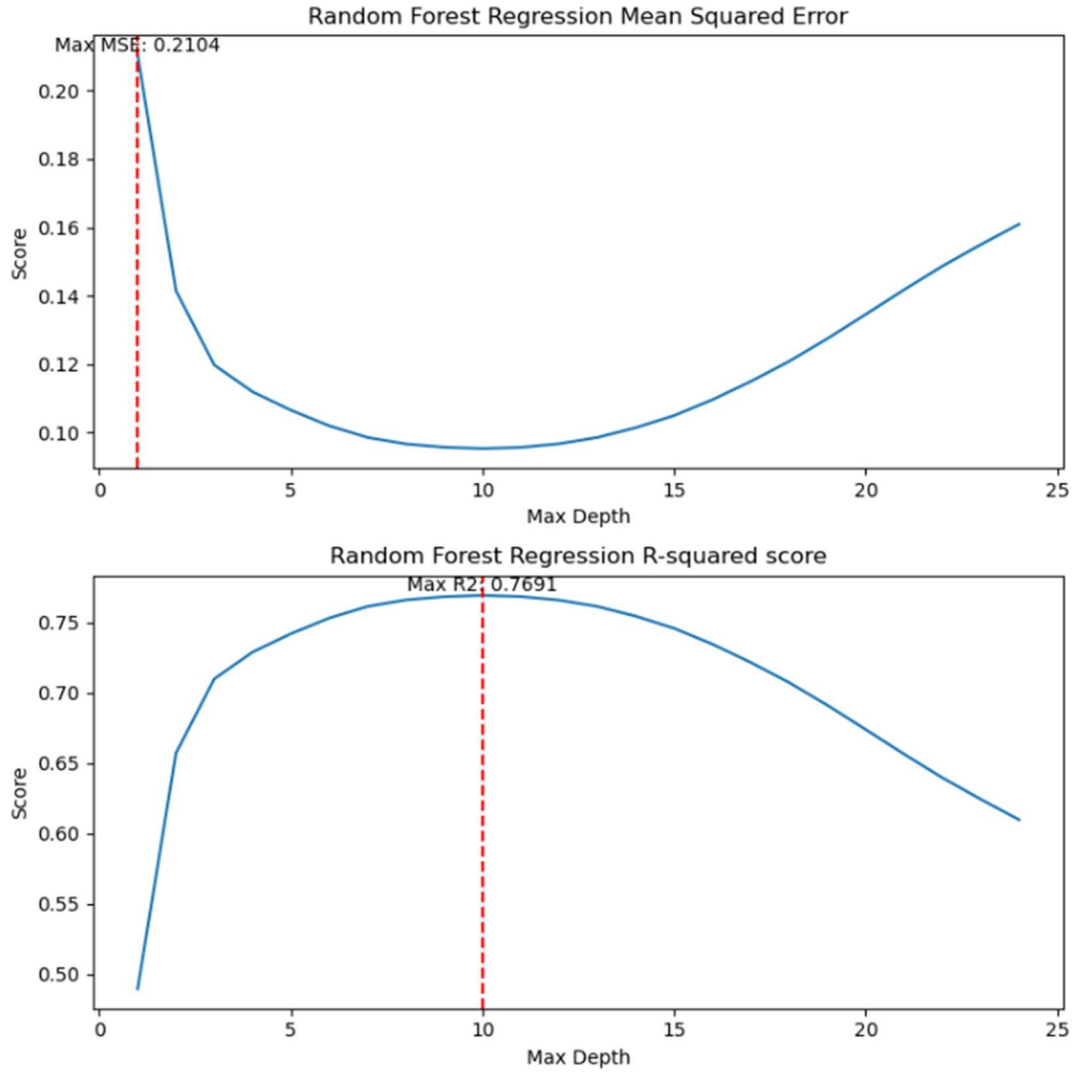
As mentioned above, this formula is an ill-conditioned way of solving for c when c is small. Instead, we substitute the identity that $\cos(\theta) = 1 - 2\operatorname{hav}(\theta)$, and also employ the addition identity $\cos(a - b) = \cos(a)\cos(b) + \sin(a)\sin(b)$, to obtain the law of haversines, above.

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. [19]

Model	Number of layers	Layer	R2 score (3.sf)	MSE (3.sf)	RMSE
Sequential	6	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(4, activation='relu')) (Dense(1, activation='linear'))	0.775	0.0923	0.304
Sequential	2	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(1, activation='linear')	0.766	0.0965	0.312
Sequential	3	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(1, activation='linear'))	0.7738138207655503	0.09327222772179597	
Sequential	4	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(1, activation='linear'))	0.7746210813716057	0.09293933826173402	
Sequential	5	(Dense(64, activation='relu', input_dim=X_train.shape[1]))	0.7726110395953949	0.09376821770483529	

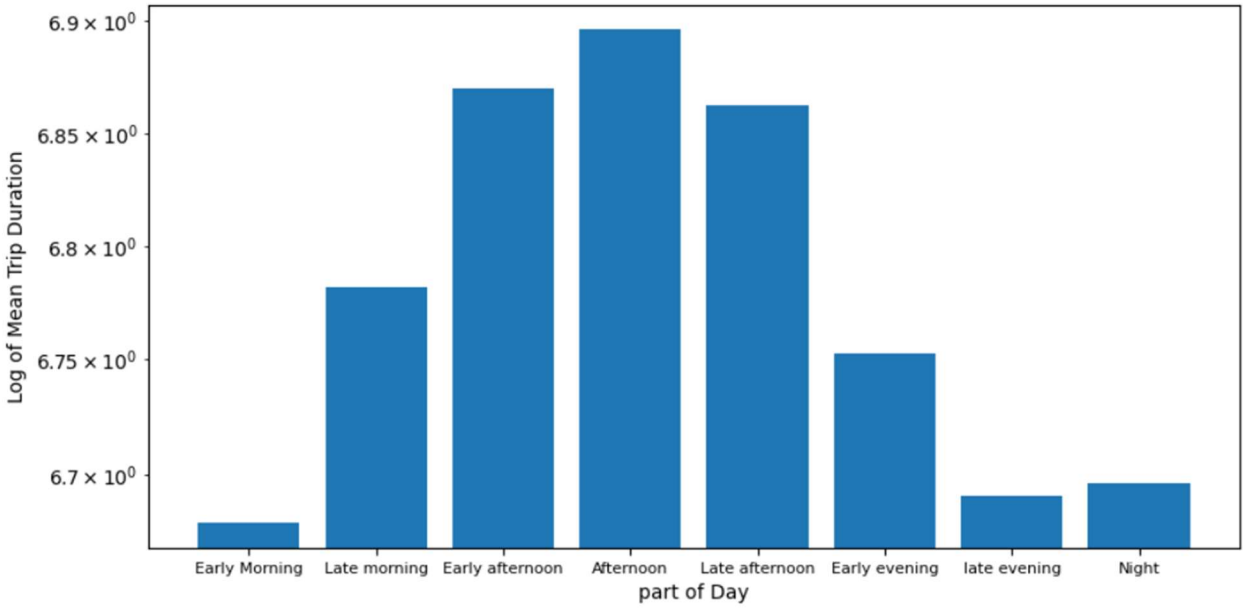
		(Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(1, activation='linear'))			
Sequential	7	(Dense(64, activation='relu', input_dim=X_train.shape[1])) (Dense(32, activation='relu')) (Dense(16, activation='relu')) (Dense(8, activation='relu')) (Dense(4, activation='relu')) (Dense(2, activation='relu')) (Dense(1, activation='linear'))	-7.873200537300562e-06	0.4123725082957821	

Model	R2 score	mse
Random Forest Regression	0.7748166735174706	0.09285868207296308
Decision tree regression	0.7728617024124562	0.09366485206404565
N nearest neighbour regression	0.7334185864182337	0.10992998068296528
Ada booster regression	0.7529829809524184	0.10186222575466765
Gradient booster regression	0.7758346869884016	0.09243888460962599



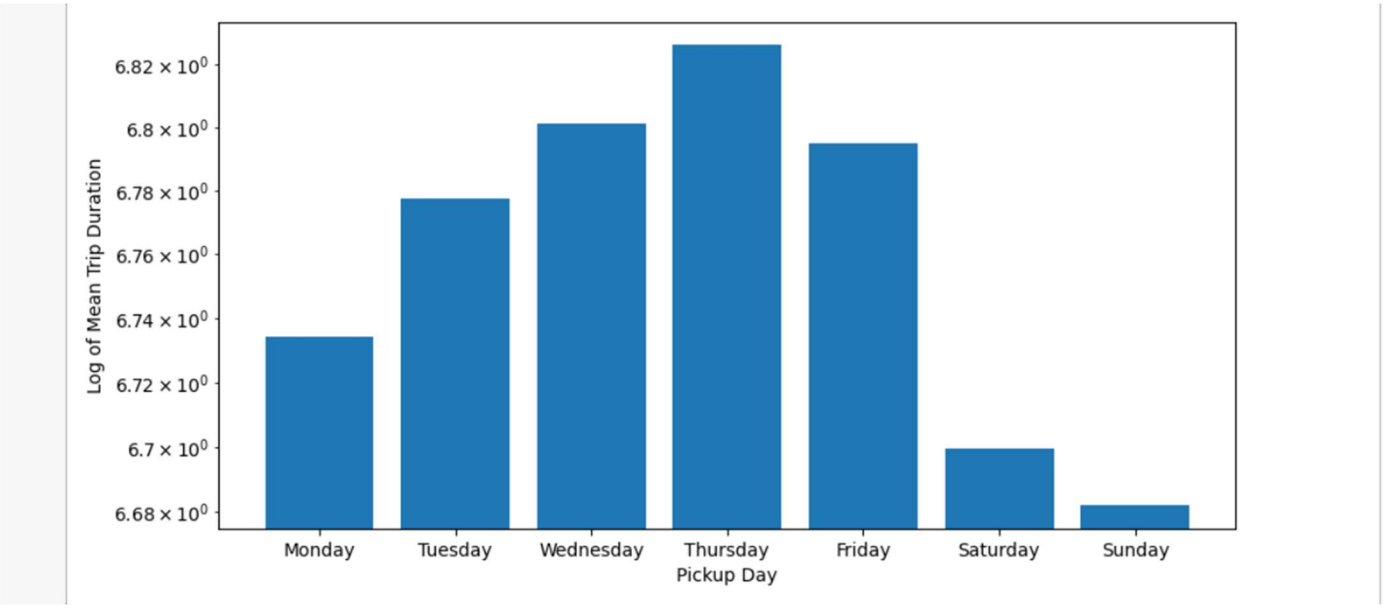
Random forest max depth variations, peak is max depth =10

Log mean trip duration against different parts of the day



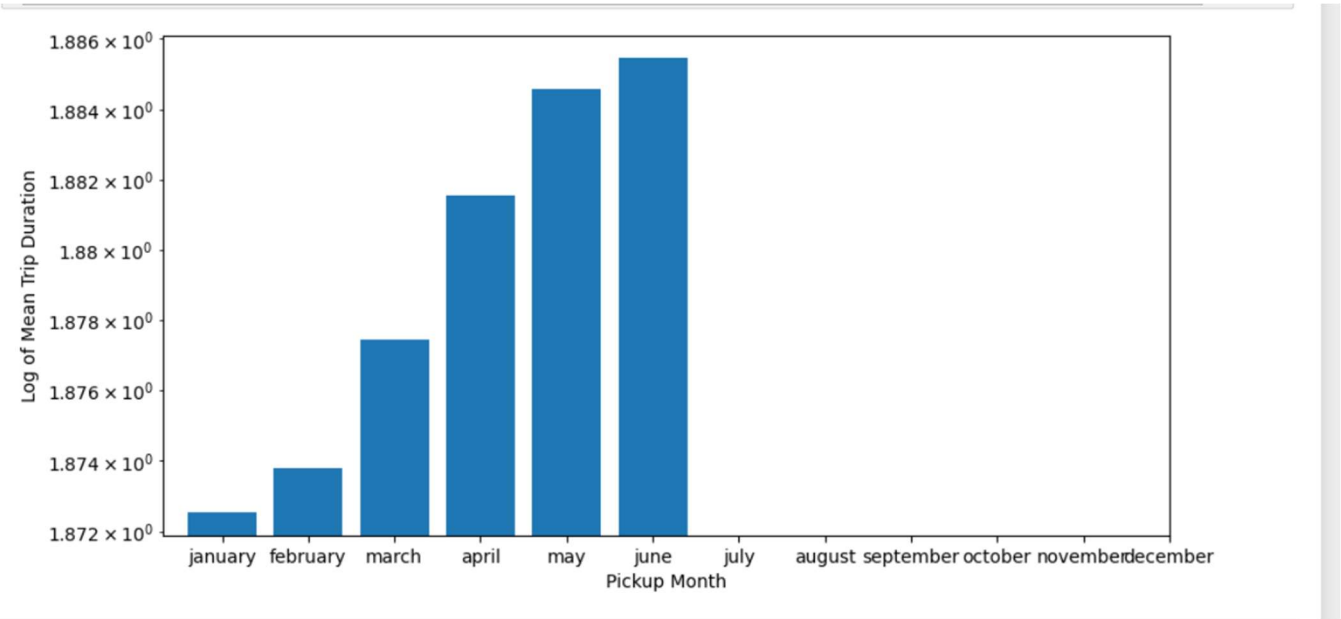
Afternoon is where the most is

Log mean trip duration against days of the week

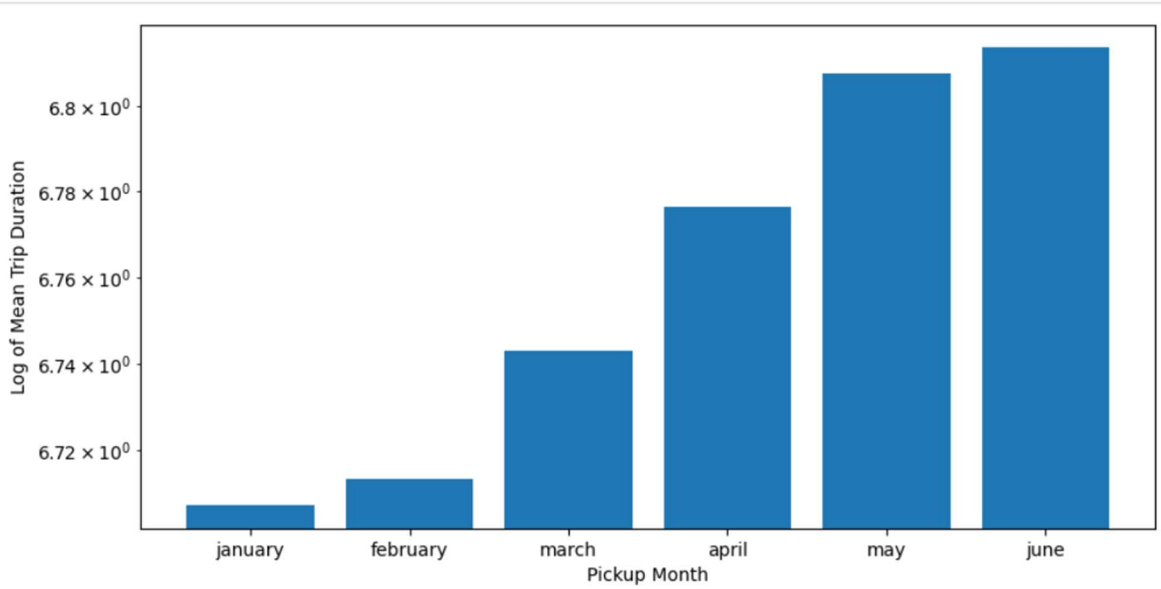


Thursday's seem to have the most trip durations and Sunday seem to have the least

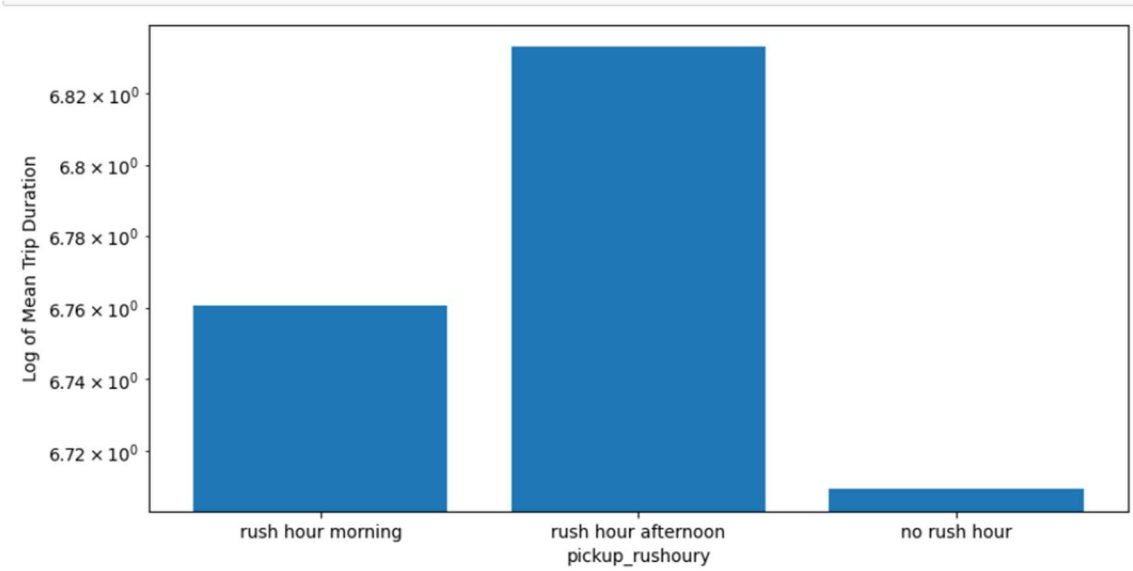
Log mean trip duration against month of pickup



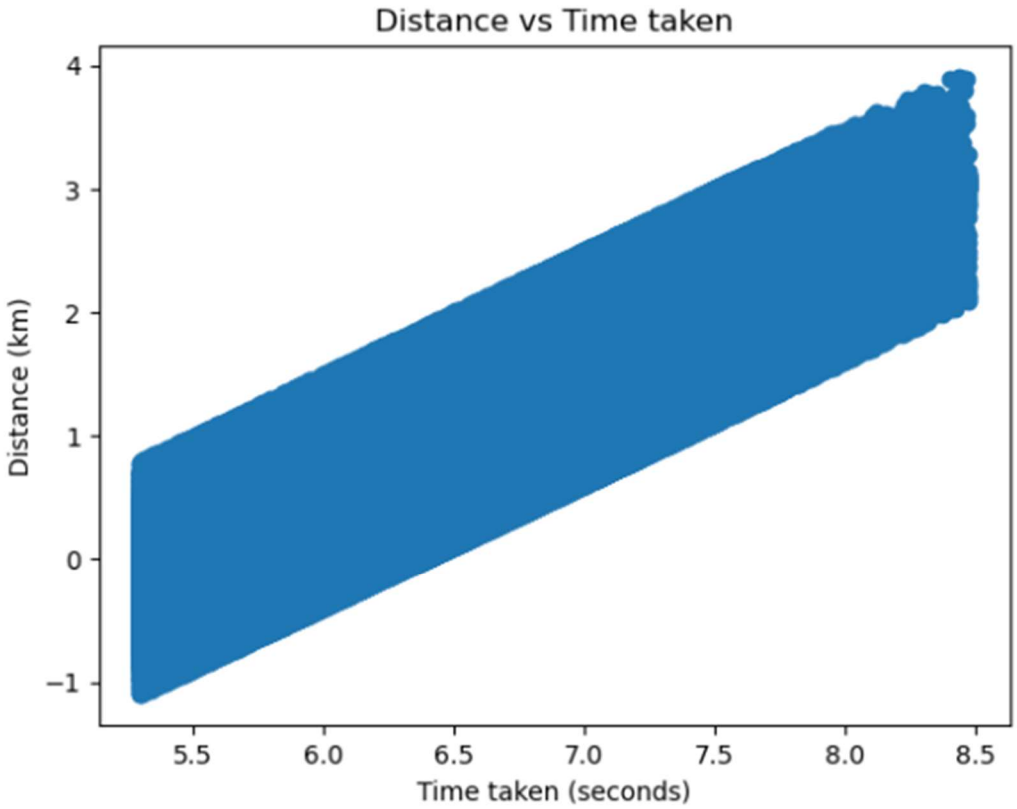
June and may seem to have significantly longer trip durations whereas January have significantly lower trip durations. Unfortunately, data after June does not exist, therefore this feature can only be useable if the date is within January to June

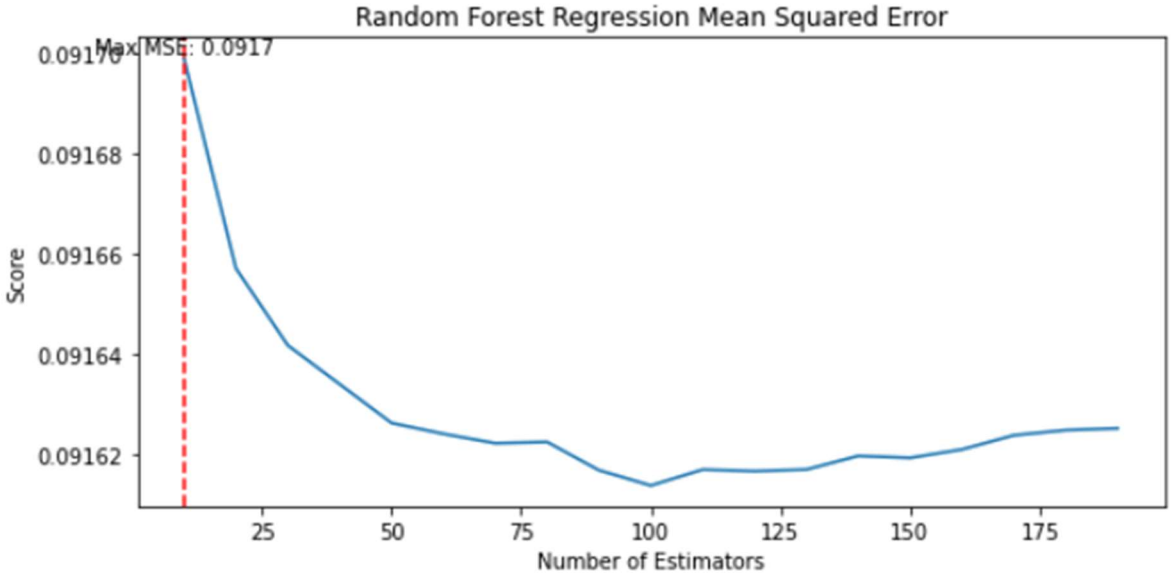


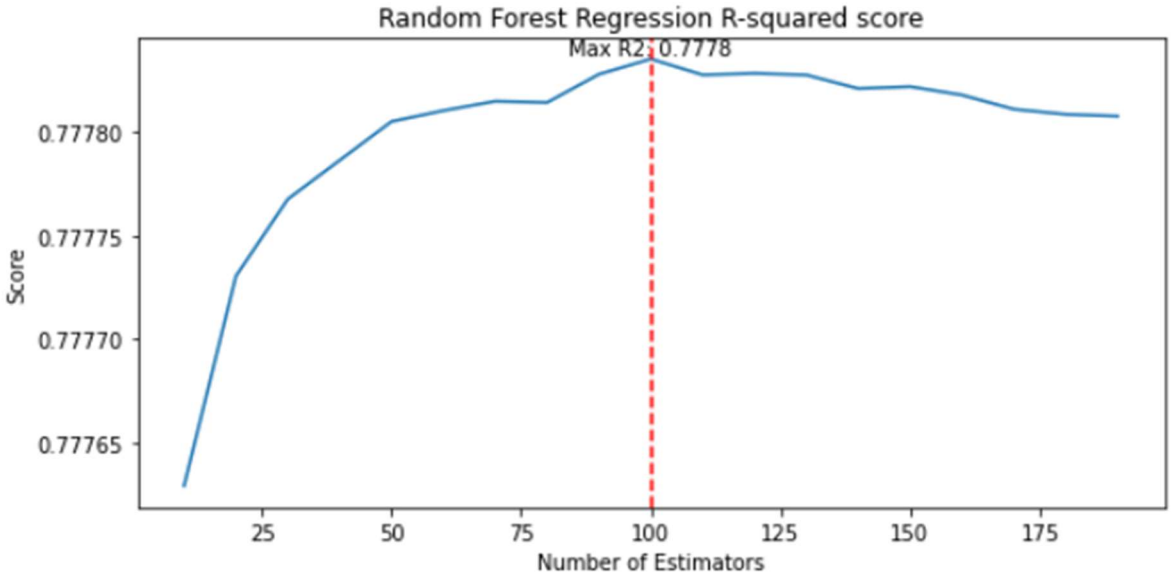
Log mean trip duration against rush hour



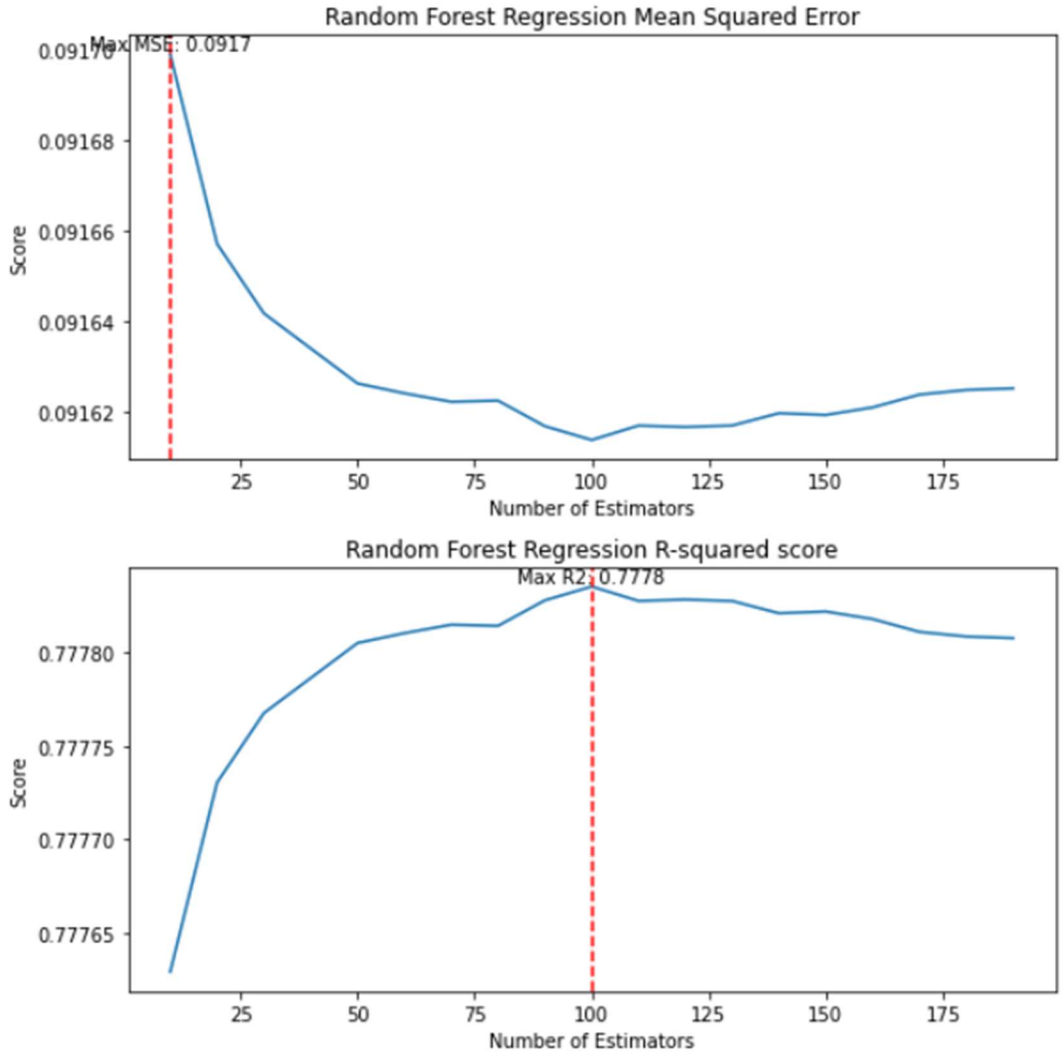
Clearly, rush hour affects the trip duration… rush hour in the afternoon has much higher trip duration than rush hour in the morning.


Distance vs Time taken
Distance (km)
Time taken (seconds)
4
3
2
1
0
−1
5.5
6.0
6.5
7.0
7.5
8.0
8.5


Random Forest Regression Mean Squared Error
Max MSE: 0.0917
0.09170
0.09168
0.09166
0.09164
0.09162
Score
25
50
75
100
125
150
175
Number of Estimators

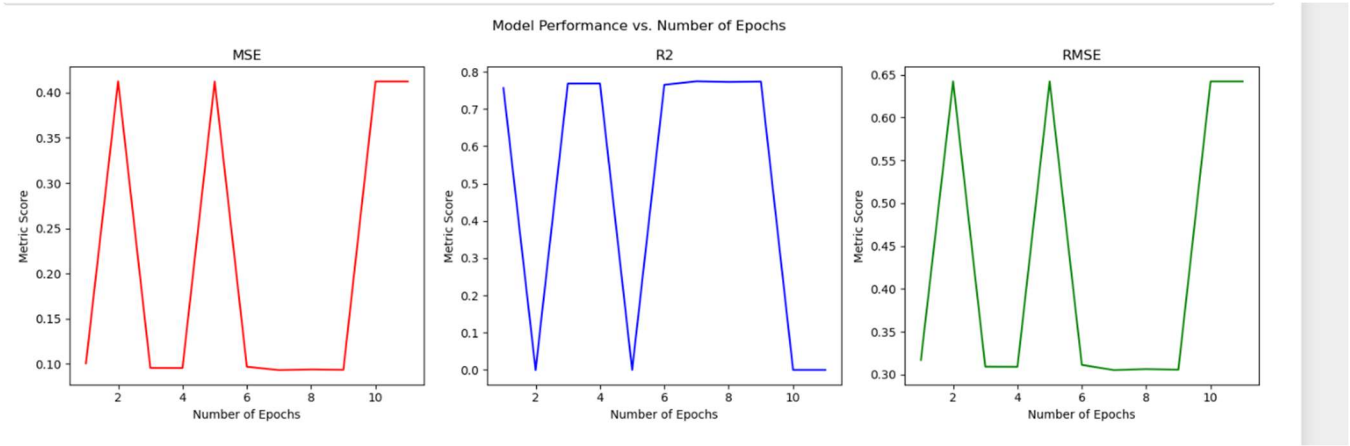

Random Forest Regression R-squared score
Max R2: 0.7778
0.77780
0.77775
0.77770
0.77765
Score
25
50
75
100
125
150
175
Number of Estimators

19



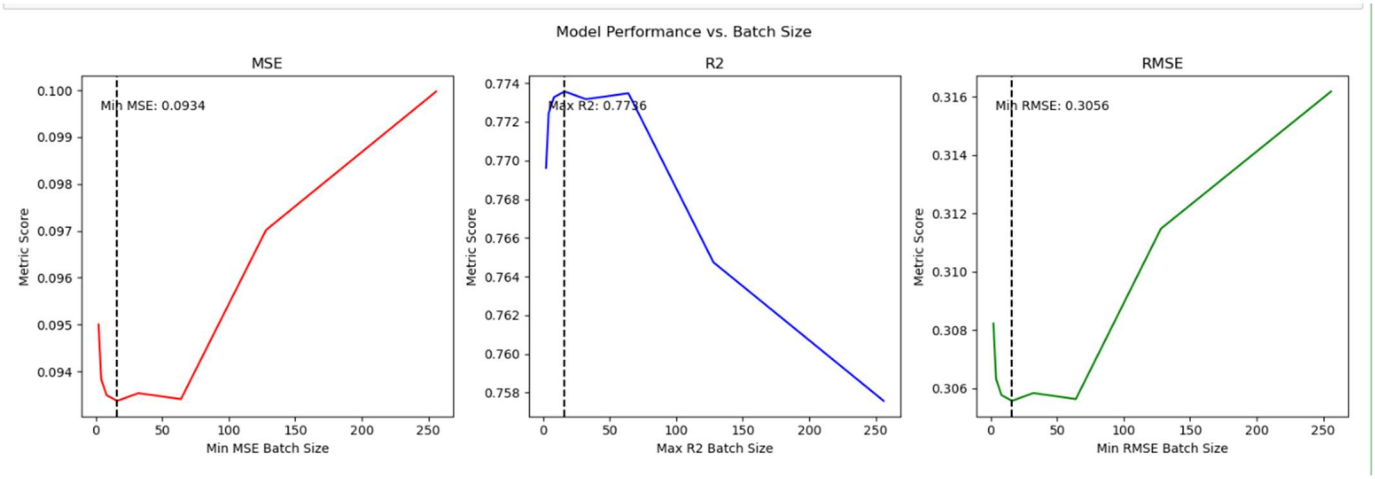
ANN

For epochs



Best epochs = 7

For batch size

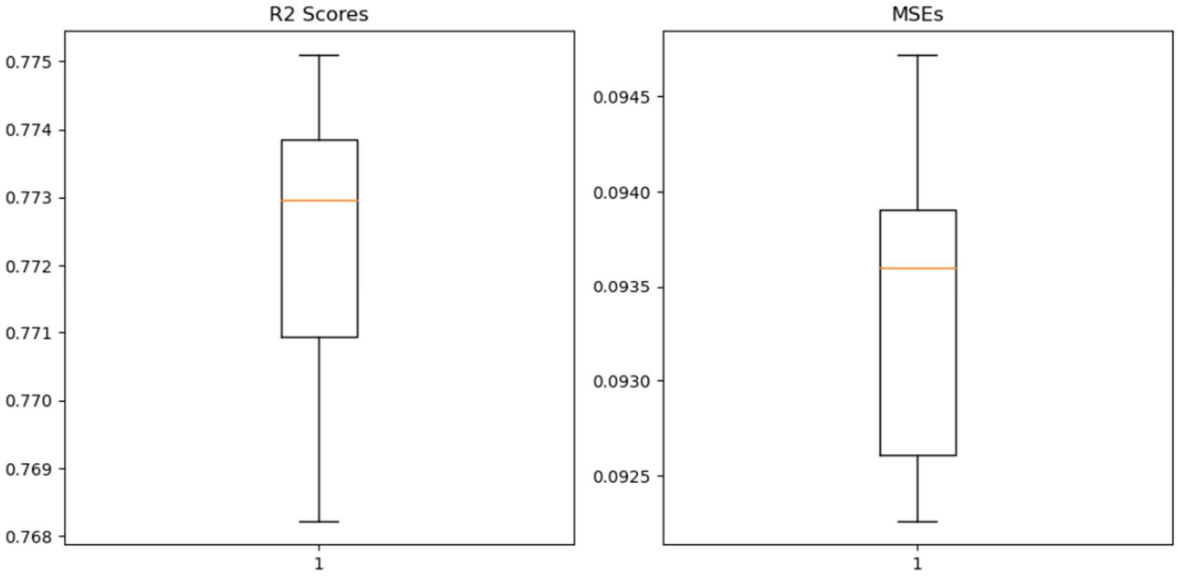


Best Batch size = 16

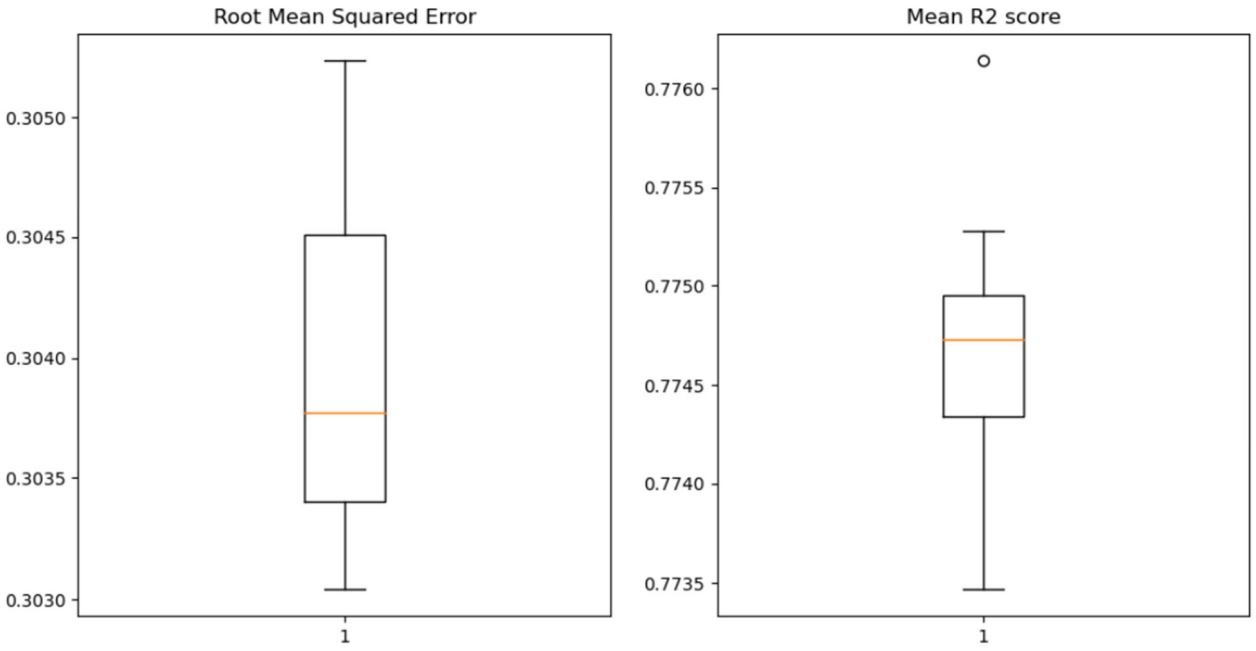
K fold for cnn with 64 32 16 1

4 layers

```
Mean R2 score: 0.7723111281968238
Mean MSE: 0.09337855344673832
```



K fold for random forest 100 n estimators 10 depth




```
Mean R2 score: 0.7746809624274605
Mean MSE: 0.09240789871753803
```