# CSE 454 DATA MINING FINAL PROJECT

In the project, it is aimed to predict the distance by looking at the taxi data in New York City. While doing this, preprocessing, postprocessing, random forest, neural network and linear regression methods were used.

## Dataset

The idea is to measure recommended route distance and duration(average based on historical data) between two co-ordinates using Google's Distance Matrix API. At first the dataset had 14 columns, but when the dataset was encoded, the number of columns increased to 25. the number of lines is 39397.

## Reading file

```
data = pd.read_csv("train_distance_matrix.csv")
```

## Columns in dataset

```
data.columns
```

```
Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'gc_distance', 'trip_duration', 'google_distance', 'google_duration'],
      dtype='object')
```

## Data types

```
data.dtypes
```

```
id                    object
vendor_id              int64
pickup_datetime       object
dropoff_datetime      object
passenger_count        int64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
store_and_fwd_flag    object
gc_distance          float64
trip_duration          int64
google_distance      float64
google_duration      float64
dtype: object
```

Selected the target as duration of the trip

```
data.isnull().sum()
```

```
id                    0
vendor_id             0
pickup_datetime       0
dropoff_datetime      0
passenger_count       0
pickup_longitude      0
pickup_latitude       0
dropoff_longitude     0
dropoff_latitude      0
store_and_fwd_flag    0
gc_distance           0
trip_duration         0
google_distance     559
google_duration     559
dtype: int64
```

Splitted date by using datetime library and added to data columns so increased the column number.

```
data['pickup_datetime'] = pd.to_datetime(data['pickup_datetime'])
data['dropoff_datetime'] = pd.to_datetime(data['dropoff_datetime'])
```

```
data.shape
```

```
(39396, 14)
```

```
data['pickup_datetime'].dtypes
```

```
dtype('<M8[ns]')
```

```
data['pickup_day'] = data['pickup_datetime'].dt.day_name()
data['dropoff_day'] = data['dropoff_datetime'].dt.day_name()
```

```
data.shape
```

```
(39396, 16)
```

```
data['pickup_day_no'] = data['pickup_datetime'].dt.weekday
data['droff_day_no'] = data['dropoff_datetime'].dt.weekday
```

```
data.shape
```

```
(39396, 18)
```

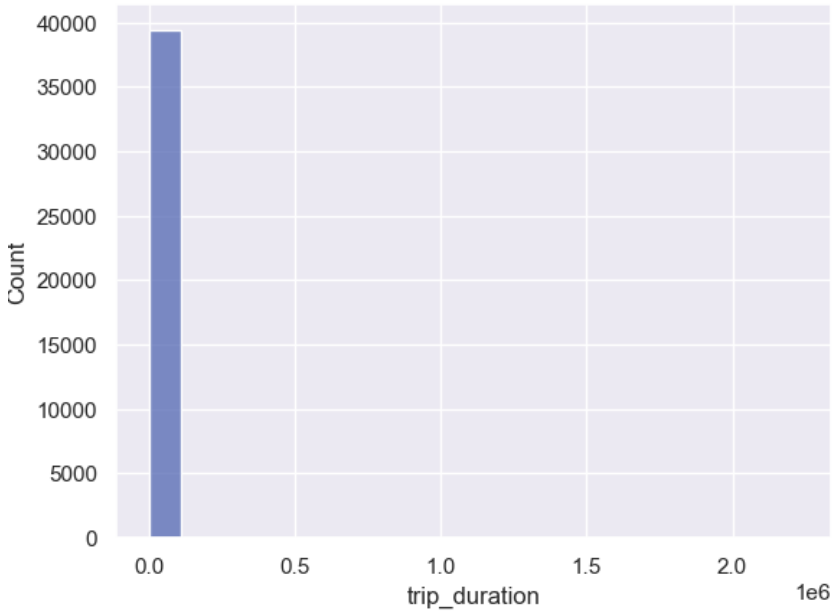
```
data.sample(5)
```

```
data['pickup_hour'] = data['pickup_datetime'].dt.hour
data['dropoff_hour'] = data['dropoff_datetime'].dt.hour

data['pickup_month'] = data['pickup_datetime'].dt.month
data['dropoff_month'] = data['pickup_datetime'].dt.month
```

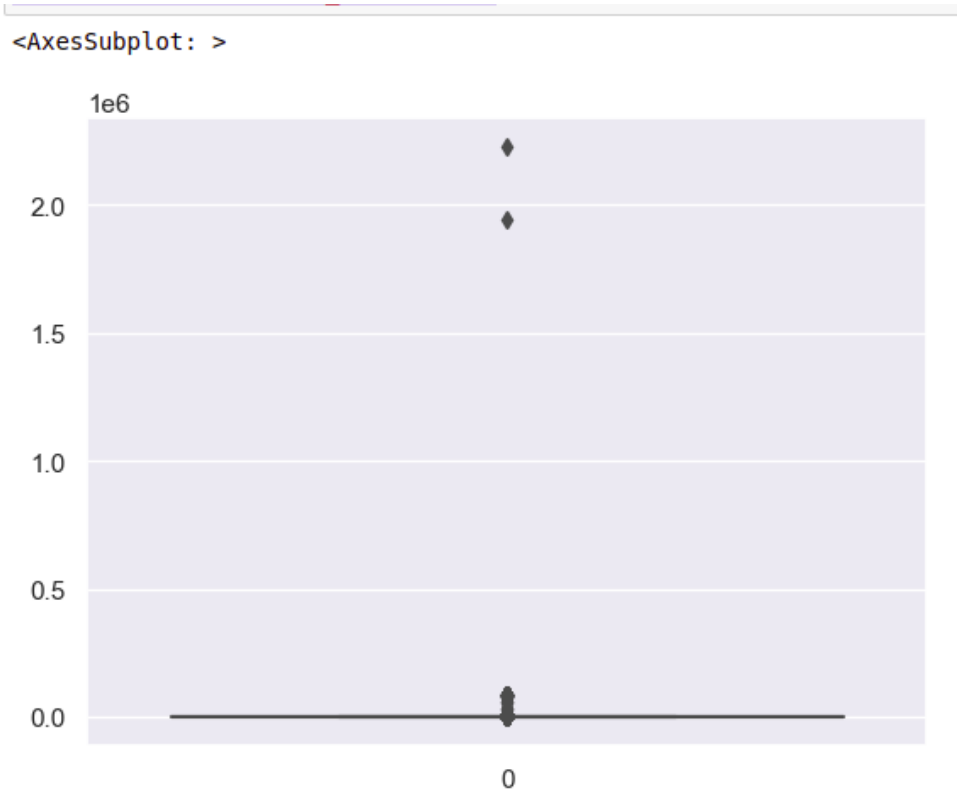
The function that is being applied is a lambda function that calculates the distance between two geographic coordinates using the great_circle() function from the geopy.distance library. The great_circle() function takes two sets of coordinates (latitude and longitude) as input and returns the distance between them in kilometers.

```
: from geopy.distance import great_circle

: data['distance'] = data.apply(lambda x : great_circle((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_lat:

: data['distance'] = data.apply(lambda x : great_circle((x['pickup_latitude'],x['pickup_longitude']), (x['dropoff_lat:
```

This code will create a histogram of the 'trip_duration' column, which shows the distribution of the trip duration values. The histogram will have 20 bins, and it will display the frequency of trip durations within each bin. The x-axis represents the trip duration, and the y-axis represents the frequency of trip durations.



A box plot is a standardized way of displaying the distribution of data based on five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum"). It is also known as a box-and-whisker plot. It is a good way to visualize the distribution of a numerical variable and identify any outliers or patterns in the data. The box in the plot represents the interquartile range (IQR = Q3 - Q1), which contains the middle 50% of the data. The line inside the box represents the median of the data. The whiskers extend from the box to show the range of the data, and the dots outside the whiskers represent the outliers.



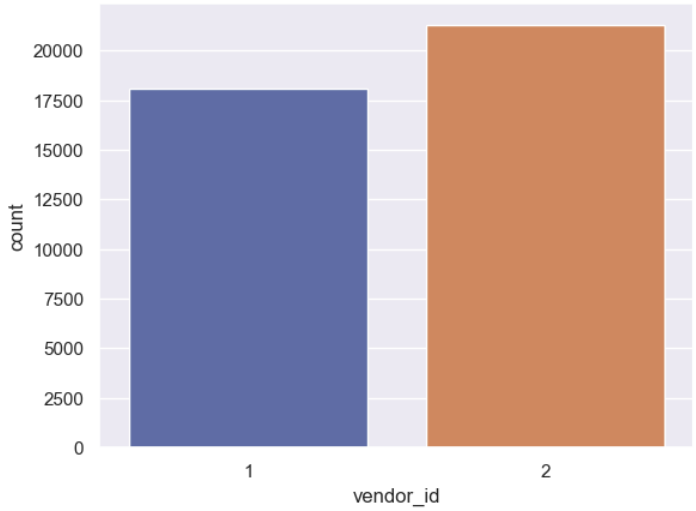
Remove outliers in the data, and it is likely that the trip_duration of 1939736 is an outlier, which is an unusual value that is significantly different from the other values in the dataset.

```
: data['trip_duration'].sort_values(ascending = False)
```

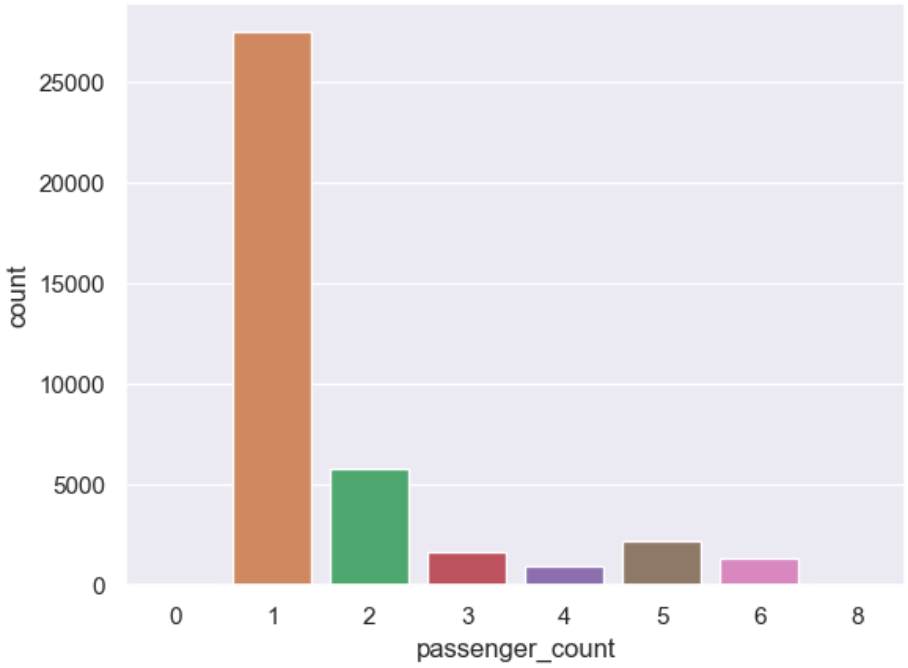
```
: 19563    2227612
  19297    1939736
  17686      86354
  2578       86353
  20824      86346
             ...
  7406           3
  19101          3
  32884          3
  32995          3
  32060          2
  Name: trip_duration, Length: 39396, dtype: int64
```

```
: data.drop(data[data['trip_duration'] == 1939736].index, inplace=True)
```

Created a bar chart showing the number of occurrences of each unique value of the 'vendor_id' column. The x-axis represents the unique values of the 'vendor_id' column, and the y-axis represents the count of occurrences of each unique value. It is a good way to visualize the distribution of a categorical variable and to identify patterns or trends in the data.



Create a bar chart showing the number of occurrences of each unique value of the 'passenger_count' column. It shows how many times each number of passenger_count appears in the dataset. It shows how many people usually travel together in cabs.



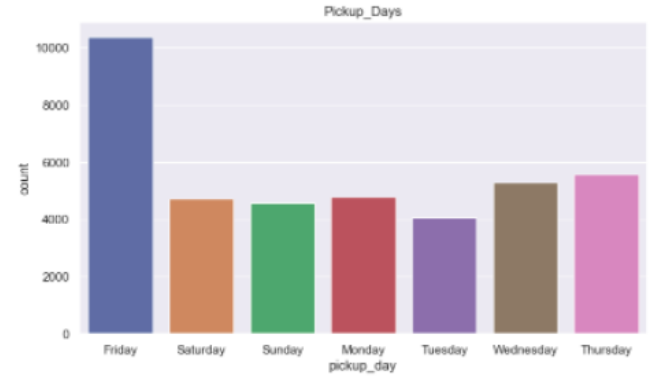

The output of this code will be a series of objects showing the frequency of each unique value of the 'store_and_fwd_flag' column. The index of the series will be the unique values of the column and the values will be the normalized frequency of each unique value.

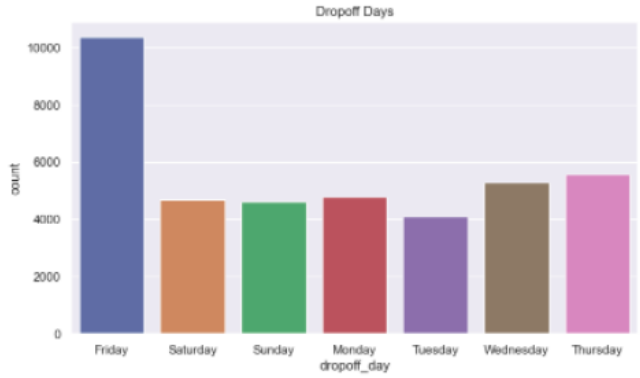
This code is useful when you want to get a quick and easy to understand representation of the data. It gives you the idea of how many times each value of store_and_fwd_flag appears in the dataset, and it normalizes it so that it gives you the percentage of each value in the dataset.

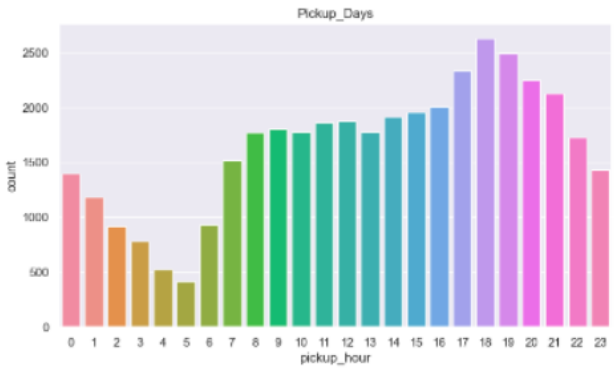
```
In [37]: data['store_and_fwd_flag'].value_counts(normalize = True)

Out[37]: N    0.994161
         Y    0.005839
         Name: store_and_fwd_flag, dtype: float64
```

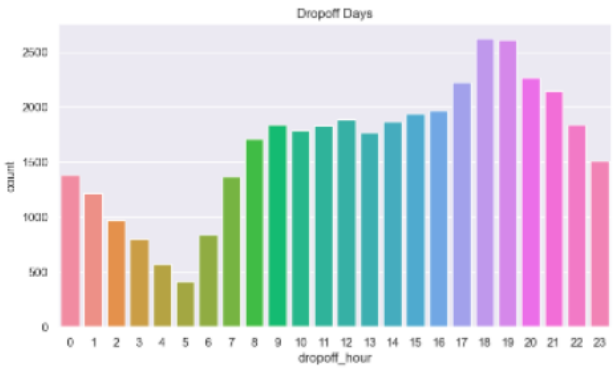
Count plot graphics according to different parameters

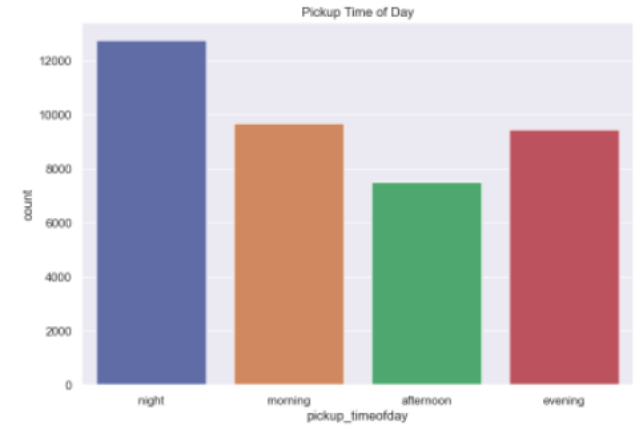
```
ax = sns.countplot(x ='dropoff_day', data = data, ax =ax2)
```

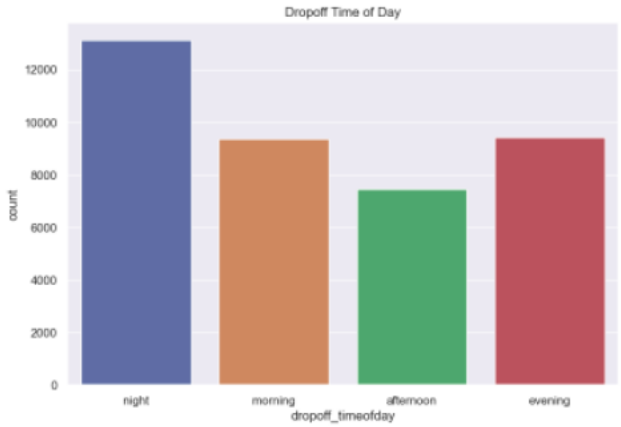


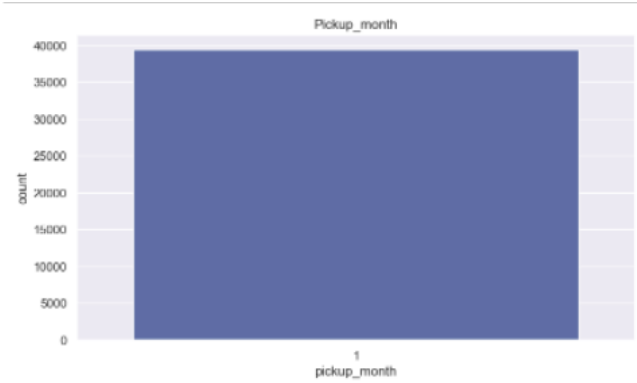



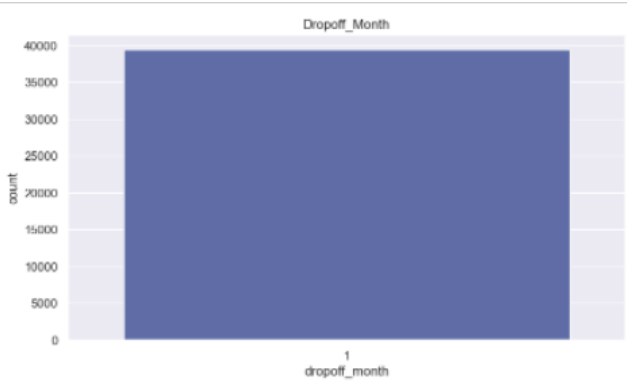



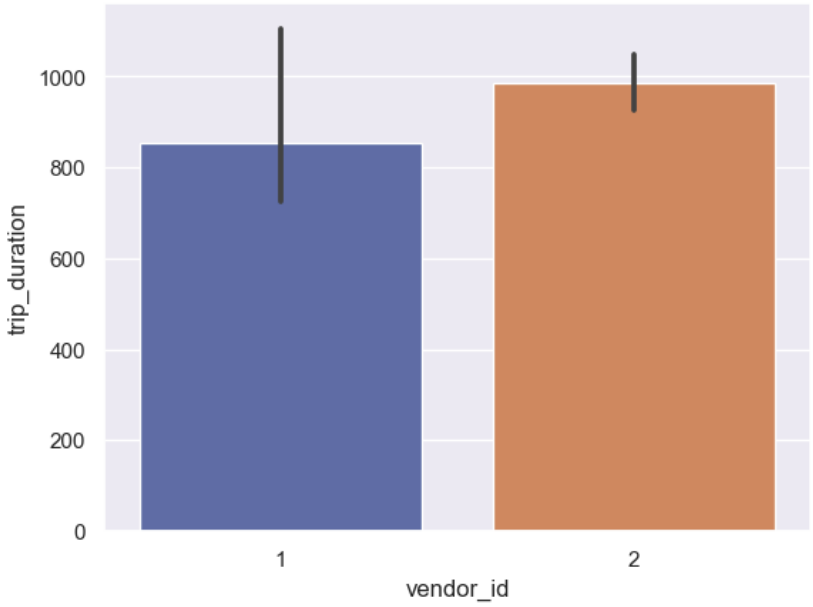


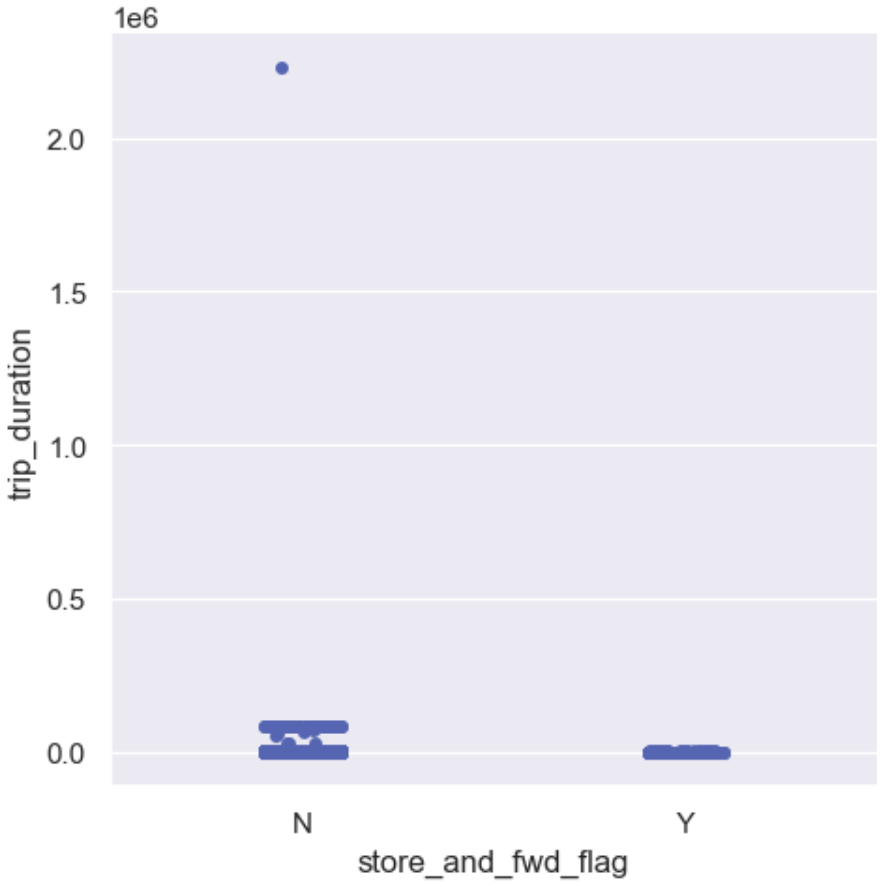




Create a bar chart showing the mean of the 'trip_duration' column grouped by the unique values of the 'vendor_id' column. The x-axis represents the unique values of the 'vendor_id' column and the y-axis represents the mean of the 'trip_duration' column for each unique value of the 'vendor_id' column. It is a good way to visualize the relationship between a numerical variable and a categorical variable and to identify patterns or trends in the data.

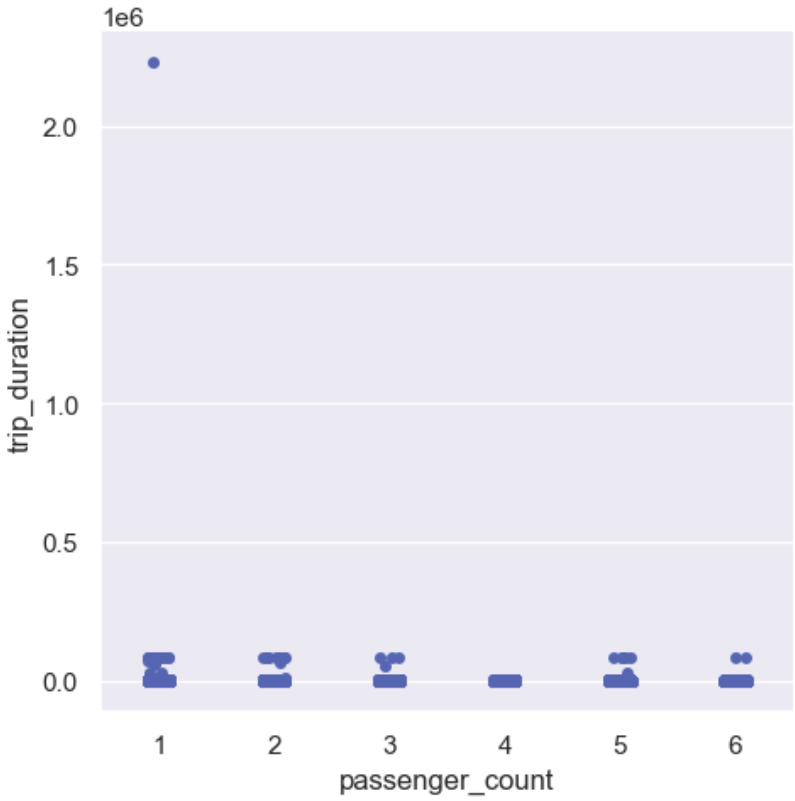



Create a strip plot of the 'trip_duration' column grouped by the unique values of the 'store_and_fwd_flag' column. The x-axis represents the unique values of the 'store_and_fwd_flag' column and the y-axis represents the values of the 'trip_duration' column. Each point on the graph represents a single observation of the 'trip_duration' column for a specific value of the 'store_and_fwd_flag' column. It is a good way to visualize the distribution of a numerical variable within a categorical variable and to identify patterns or trends in the data. It gives you the distribution of trip_duration for each store_and_fwd_flag value, it helps in identifying how trip_duration is distributed for each value of store_and_fwd_flag.

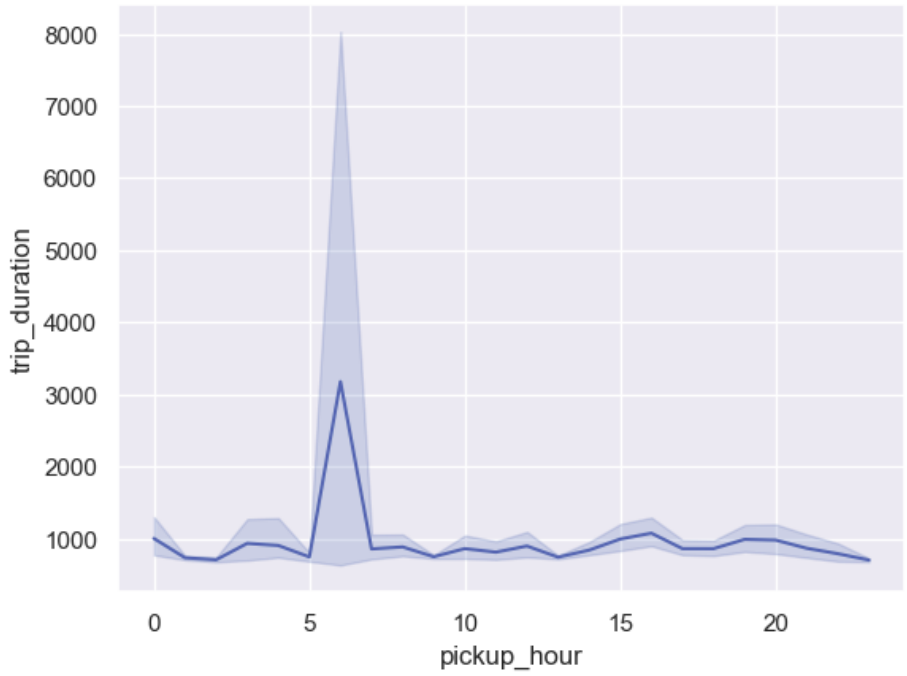
```
<seaborn.axisgrid.FacetGrid at 0x7f1ae1a42e80>
```



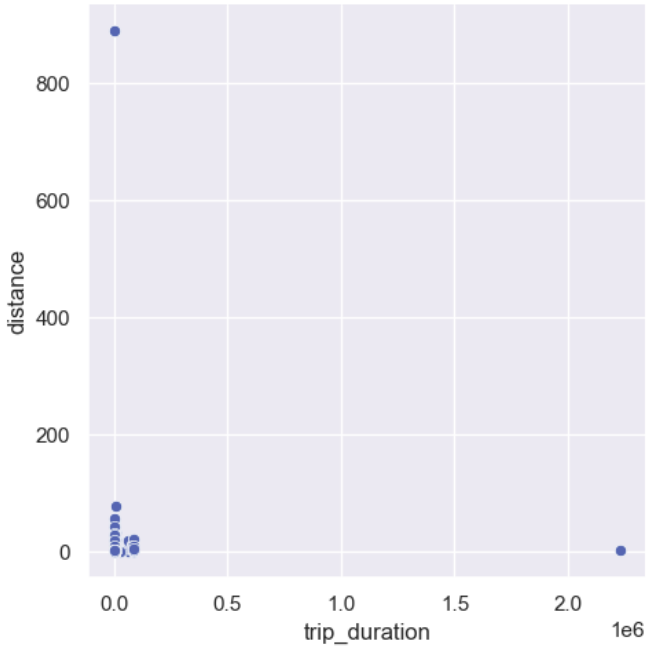

Created a strip plot of the 'trip_duration' column grouped by the unique values of the 'passenger_count' column. The x-axis represents the unique values of the 'passenger_count' column and the y-axis represents the values of the 'trip_duration' column. Each point on the graph represents a single observation of the 'trip_duration' column for a specific value of the 'passenger_count' column. It is a good way to visualize the distribution of a numerical variable within a categorical variable and to identify patterns or trends in the data.




Created a line chart of the 'trip_duration' column against the 'pickup_hour' column. The x-axis represents the unique values of the 'pickup_hour' column and the y-axis represents the values of the 'trip_duration' column. It will show the relation between the pickup hour and the trip duration. It will help to identify whether the trip duration is affected by the pickup hour, whether the trip duration is more at a specific pickup hour or not.




This chart identifies whether the distance traveled is affected by the trip duration, whether the distance traveled is more for a specific trip duration or not.



The output shows that the number of non-null values in each column is 38012, which means that there are no missing values in the dataframe for the rows where the distance is greater than 0.5.

```
data[data.distance>0.5].count()
```

```
id                    38012
vendor_id             38012
pickup_datetime       38012
dropoff_datetime      38012
passenger_count       38012
pickup_longitude      38012
pickup_latitude       38012
dropoff_longitude     38012
dropoff_latitude      38012
store_and_fwd_flag    38012
gc_distance           38012
trip_duration         38012
google_distance       37472
google_duration       37472
pickup_day            38012
dropoff_day           38012
pickup_day_no         38012
droff_day_no          38012
pickup_hour           38012
dropoff_hour          38012
pickup_month          38012
dropoff_month         38012
pickup_timeofday      38012
dropoff_timeofday     38012
distance              38012
dtype: int64
```

The above code is looping through each column in the dataframe and printing the unique values for each column. It is useful to get a sense of the unique values present in each column and understand if there are any outliers or unusual values.

```
for col in data:
    print(f"{col}:{data[col].unique()}")
```

```
id:['id0190469' 'id1665586' 'id1078247' ... 'id3324326' 'id3410388'
 'id0044708']
vendor_id:[2 1]
pickup_datetime:['2016-01-01T00:00:00.000000000' '2016-01-01T00:01:00.000000000'
 '2016-01-01T00:02:00.000000000' ... '2016-01-08T21:54:00.000000000'
 '2016-01-08T21:55:00.000000000' '2016-01-08T21:56:00.000000000']
dropoff_datetime:['2016-01-01T00:14:00.000000000' '2016-01-01T00:22:00.000000000'
 '2016-01-01T00:03:00.000000000' ... '2016-01-08T22:04:00.000000000'
 '2016-01-08T22:12:00.000000000' '2016-01-08T22:14:00.000000000']
passenger_count:[5 1 2 3 4 6]
pickup_longitude:[-73.98174286 -73.98508453 -73.97333527 ... -73.99565125 -74.02311707
 -73.95082092]
pickup_latitude:[40.71915817 40.74716568 40.76407242 ... 40.75350189 40.73135757
 40.7230835 ]
dropoff_longitude:[-73.93882751 -73.95803833 -73.97485352 ... -74.0306015  -73.93953705
 -73.94979095]
dropoff_latitude:[40.82918167 40.71749115 40.76173401 ... 40.75825882 40.73993301
 40.72644424]
store_and_fwd_flag:['N' 'Y']
gc_distance:[7.92882961 2.49248213 0.18009874 ... 1.46001777 0.30859276 1.59083972]
trip_duration:[  849  1294   114 ...  4505 84456  2857]
google_distance:[15309.  9970.   289. ...  9974.  9548.  8335.]
google_duration:[1109. 1277.   79. ... 2615. 2592. 3164.]
pickup_day:['Friday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday' 'Thursday']
dropoff_day:['Friday' 'Saturday' 'Sunday' 'Monday' 'Tuesday' 'Wednesday' 'Thursday']
pickup_day_no:[4 5 6 0 1 2 3]
droff_day_no:[4 5 6 0 1 2 3]
pickup_hour:[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
dropoff_hour:[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
pickup_month:[1]
dropoff_month:[1]
pickup_timeofday:['night' 'morning' 'afternoon' 'evening']
dropoff_timeofday:['night' 'morning' 'afternoon' 'evening']
distance:[12.75663856  4.01013682  0.28975967 ...  2.34901199  0.49649281
  2.55949137]
```

Data=data[['vendor_id','passenger_count','pickup_latitude','dropoff_latitude','trip_duration','pickup_day','dropoff_day','distance']]

This line of code is selecting only the specified columns from the dataframe 'data' and creating a new dataframe with only those columns. The columns that are being selected are: vendor_id, passenger_count, pickup_latitude, dropoff_latitude, trip_duration, pickup_day, dropoff_day, and distance. This is useful for reducing the number of features in the dataset and only keeping the relevant columns for the analysis or modeling that will be done.

## Sample data

```
data.sample(5)
```

| | vendor_id | passenger_count | pickup_latitude | dropoff_latitude | trip_duration | pickup_day | dropoff_day | distance |
|---|---|---|---|---|---|---|---|---|
| **30106** | 1 | 1 | 40.754608 | 40.759148 | 472 | Thursday | Thursday | 0.983214 |
| **27706** | 2 | 1 | 40.756699 | 40.744629 | 461 | Wednesday | Wednesday | 1.405601 |
| **21451** | 1 | 2 | 40.751335 | 40.739449 | 442 | Tuesday | Tuesday | 1.364441 |
| **31273** | 2 | 1 | 40.762127 | 40.760464 | 1074 | Thursday | Thursday | 1.093981 |
| **25313** | 1 | 1 | 40.764271 | 40.778057 | 716 | Wednesday | Wednesday | 2.312575 |

Now, the values of the pickup_day and dropoff_day columns are numerical values, with Monday being represented as 1, Tuesday as 2, and so on.

## Sample data

```
data.sample(5)
```

| | vendor_id | passenger_count | pickup_latitude | dropoff_latitude | trip_duration | pickup_day | dropoff_day | distance |
|---|---|---|---|---|---|---|---|---|
| **26217** | 1 | 2 | 40.753864 | 40.745892 | 688 | 3 | 3 | 1.067398 |
| **21921** | 2 | 1 | 40.740921 | 40.760399 | 1014 | 2 | 2 | 2.420328 |
| **12259** | 1 | 1 | 40.775028 | 40.799732 | 539 | 7 | 7 | 2.799696 |
| **16034** | 2 | 5 | 40.774342 | 40.786366 | 721 | 1 | 1 | 1.699813 |
| **9902** | 1 | 1 | 40.740410 | 40.750355 | 664 | 7 | 7 | 1.565454 |

Splitted the data into training and testing sets. The 'X' variable contains all the columns except for 'pickup_day', 'dropoff_day', and 'distance' which are the target variables and are stored in the 'y' variable. The train_test_split function is used to split the data into 80% training and 20% testing sets. The 'random_state' parameter is set to 5, which means that the same random split of data will be used every time the code is run.

## Trained data

```
X_train[:4]
```

| | vendor_id | passenger_count | pickup_latitude | dropoff_latitude | trip_duration |
|---|---|---|---|---|---|
| **9245** | 1 | 4 | 40.744553 | 40.618622 | 2501 |
| **18381** | 1 | 1 | 40.752098 | 40.746296 | 198 |
| **22800** | 1 | 1 | 40.645512 | 40.746887 | 1844 |
| **6009** | 2 | 2 | 40.787540 | 40.766529 | 417 |

Created a sequential neural network model using the Keras API in TensorFlow. The model has four layers, with the first layer having 5 neurons and an input shape of 5 (matching the number of input features in the dataset), and all layers using the ReLU activation function except for the last layer which uses the sigmoid activation function. The model is then compiled using the Adam optimizer, binary crossentropy loss function, and accuracy metric. The model is then fit to the training data for 100 epochs, and the model's performance on the test data is evaluated using the evaluate() function.

## Result of neural network method is 30%.

```
model.evaluate(X_test,y_test)
```

```
247/247 [==============================] - 1s 2ms/step - loss: -4869062131712.0000 - accuracy: 0.3064
[-4869062131712.0, 0.3064229488372803]
```

Used the RandomForestRegressor class from the sklearn.ensemble library to create a random forest regression model. It then fits the model to the training data (X_train and y_train) and evaluates the model's performance on the test data (X_test and y_test) using the .score() method. The output of this code will be a single number indicating the model's performance on the test data, where a higher number represents better performance.

## Accuracy with Random Forest

```
Out[84]: 0.24358657122113295
```

Linear regression is a statistical method for modeling the relationship between a dependent variable (also known as the outcome or response variable) and one or more independent variables (also known as predictor or explanatory variables). The basic idea behind linear regression is to find the line of best fit that minimizes the difference between the predicted values (based on the line) and the actual values of the dependent variable. The line of best fit is represented by the equation of a straight line (y = mx + b) where m is the slope of the line and b is the y-intercept. The slope of the line represents the relationship between the independent and dependent variables, while the y-intercept represents the value of the dependent variable when the independent variable is zero. Linear regression can be used for both simple and multiple regression analysis, depending on the number of independent variables.

Linear regression is implemented.

Demo link

Data Mining-20230121_110847-Toplantı Kaydı.mp4