Project – Predict the City Taxi Trip Duration

Name: Appana Jaya Ratna Uttam
Email: ajruttam@gmail.com

Introduction

- The taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, vendor types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP).
- This dataset has geolocational data which makes it more interesting to work on.
- Business Goal: To improve the efficiency of electronic taxi dispatching systems it is important to be able to predict how long a driver will have his taxi occupied. If a dispatcher knew approximately when a taxi driver would be ending their current ride, they would be better able to identify which driver to assign to each pickup request.
- ML Goal: To build a model that predicts the total ride duration of taxi trips in New York City
- Data Source: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Data Exploration

- Reading the dataset

```
df = pd.read_csv(path+'/train.csv')
```

- It has 1458644 rows and 11 columns

```
df.shape
```

```
(1458644, 11)
```

- Data has these columns

```
df.columns
```

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

About Data

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_count	the number of passengers in the vehicle (driver entered value)
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

Data Exploration

- First 5 rows of the data looks like this

```
df.head()
```

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435

Data Exploration

Datatypes of each column

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1458644 entries, 0 to 1458643
Data columns (total 11 columns):
 #   Column              Non-Null Count    Dtype
---  ------              --------------    -----
 0   id                  1458644 non-null  object
 1   vendor_id           1458644 non-null  int64
 2   pickup_datetime     1458644 non-null  object
 3   dropoff_datetime    1458644 non-null  object
 4   passenger_count     1458644 non-null  int64
 5   pickup_longitude    1458644 non-null  float64
 6   pickup_latitude     1458644 non-null  float64
 7   dropoff_longitude   1458644 non-null  float64
 8   dropoff_latitude    1458644 non-null  float64
 9   store_and_fwd_flag  1458644 non-null  object
 10  trip_duration       1458644 non-null  int64
dtypes: float64(4), int64(3), object(4)
memory usage: 122.4+ MB
```

Null values in each column

```
df.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
trip_duration         0
dtype: int64
```

There are no null values in any column

Data Exploration

- Unique values in each column
  - 2 kind of vendors for taxi
  - Pickup_datetime and drop_off datetime are in object datatype
  - Store and forward flags are 2, N – no and Y – yes

```
df.nunique()
```

```
id                    1458644
vendor_id                   2
pickup_datetime       1380222
dropoff_datetime      1380377
passenger_count              10
pickup_longitude          23047
pickup_latitude           45245
dropoff_longitude         33821
dropoff_latitude          62519
store_and_fwd_flag            2
trip_duration              7417
dtype: int64
```

Data Exploration

- Target variable is trip_duration – Predicts the total ride duration of taxi trips

```
df['trip_duration'].describe().apply(lambda x: format(x, 'f'))
```

```
count    1458644.000000
mean         959.492273
std         5237.431724
min            1.000000
25%          397.000000
50%          662.000000
75%         1075.000000
max      3526282.000000
Name: trip_duration, dtype: object
```

Data Exploration

- Passenger_count feature

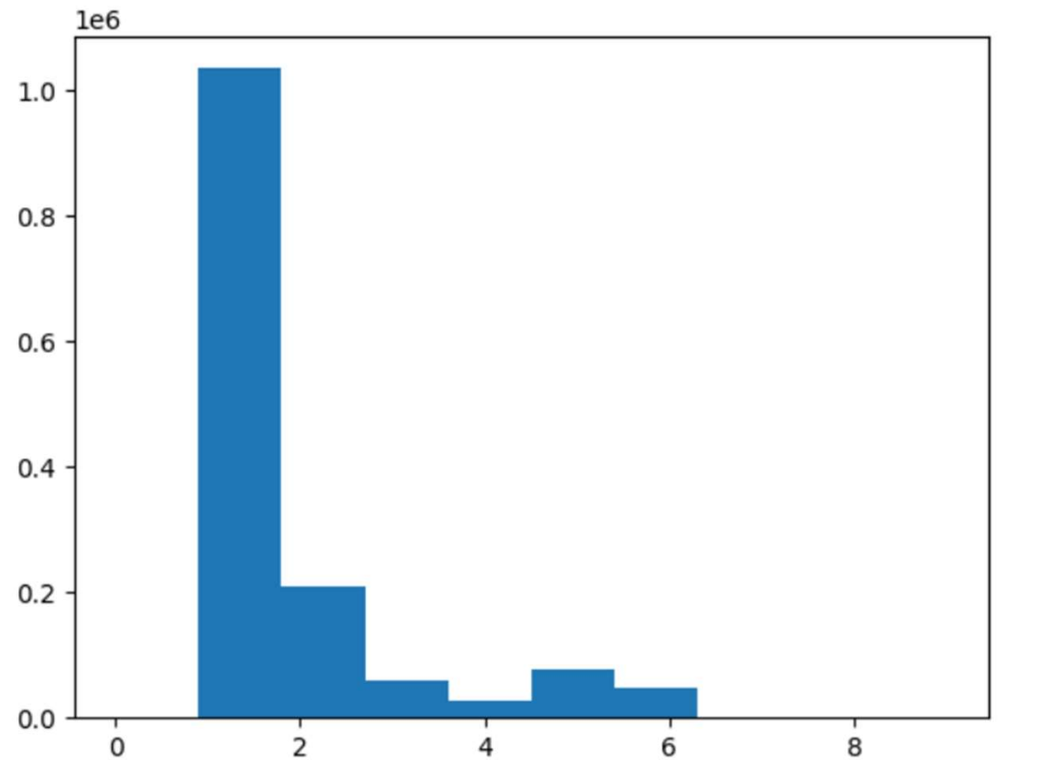
```
df['passenger_count'].value_counts()
```

```
1    1033540
2     210318
5      78088
3      59896
6      48333
4      28404
0         60
7          3
9          1
8          1
Name: passenger_count, dtype: int64
```

Most of the passengers prefer to travel solo

```
plt.hist(df['passenger_count'])
```

```
(array([6.00000e+01, 1.03354e+06, 2.10318e+05, 5.98960e+04, 2.84040e+04,
        7.80880e+04, 4.83330e+04, 3.00000e+00, 1.00000e+00, 1.00000e+00]),
 array([0. , 0.9, 1.8, 2.7, 3.6, 4.5, 5.4, 6.3, 7.2, 8.1, 9. ]),
 <BarContainer object of 10 artists>)
```

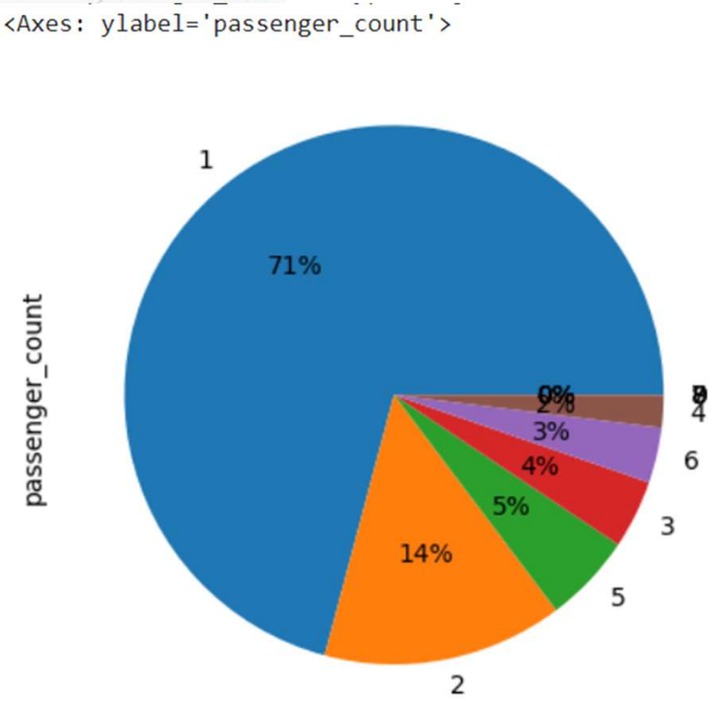


Data Exploration

- Passenger_count normalised

```
print(df['passenger_count'].value_counts(normalize=True).apply(lambda x: format(x, 'f')))
df['passenger_count'].value_counts().plot(kind="pie", autopct='%1.0f%%')
```

```
1    0.708562
2    0.144187
5    0.053535
3    0.041063
6    0.033136
4    0.019473
0    0.000041
7    0.000002
9    0.000001
8    0.000001
Name: passenger_count, dtype: object
```



Almost 71% of people prefer to travel solo

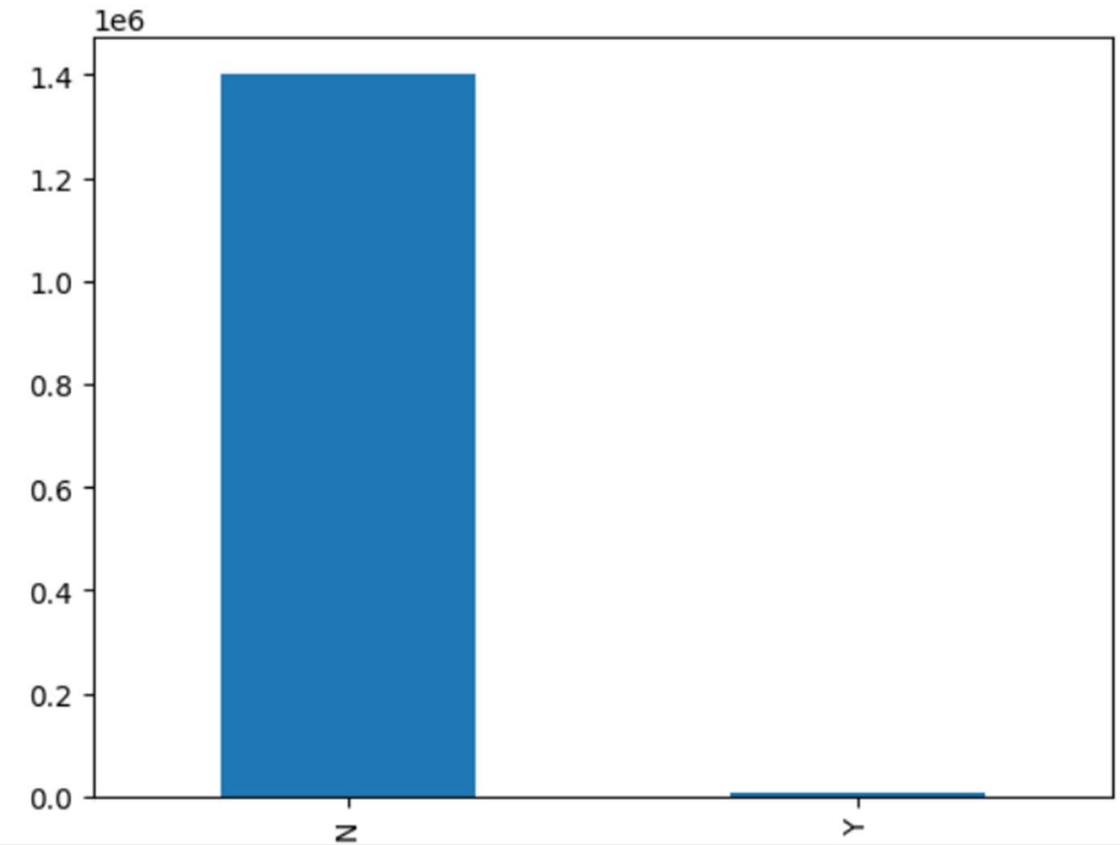
Data Exploration

- Store_and_fwd_flag

  Almost all trips are not store and Forward trip.

```
df['store_and_fwd_flag'].value_counts().plot(kind="bar")
print(df['store_and_fwd_flag'].value_counts())
```

```
N    1402204
Y       8042
Name: store_and_fwd_flag, dtype: int64
```

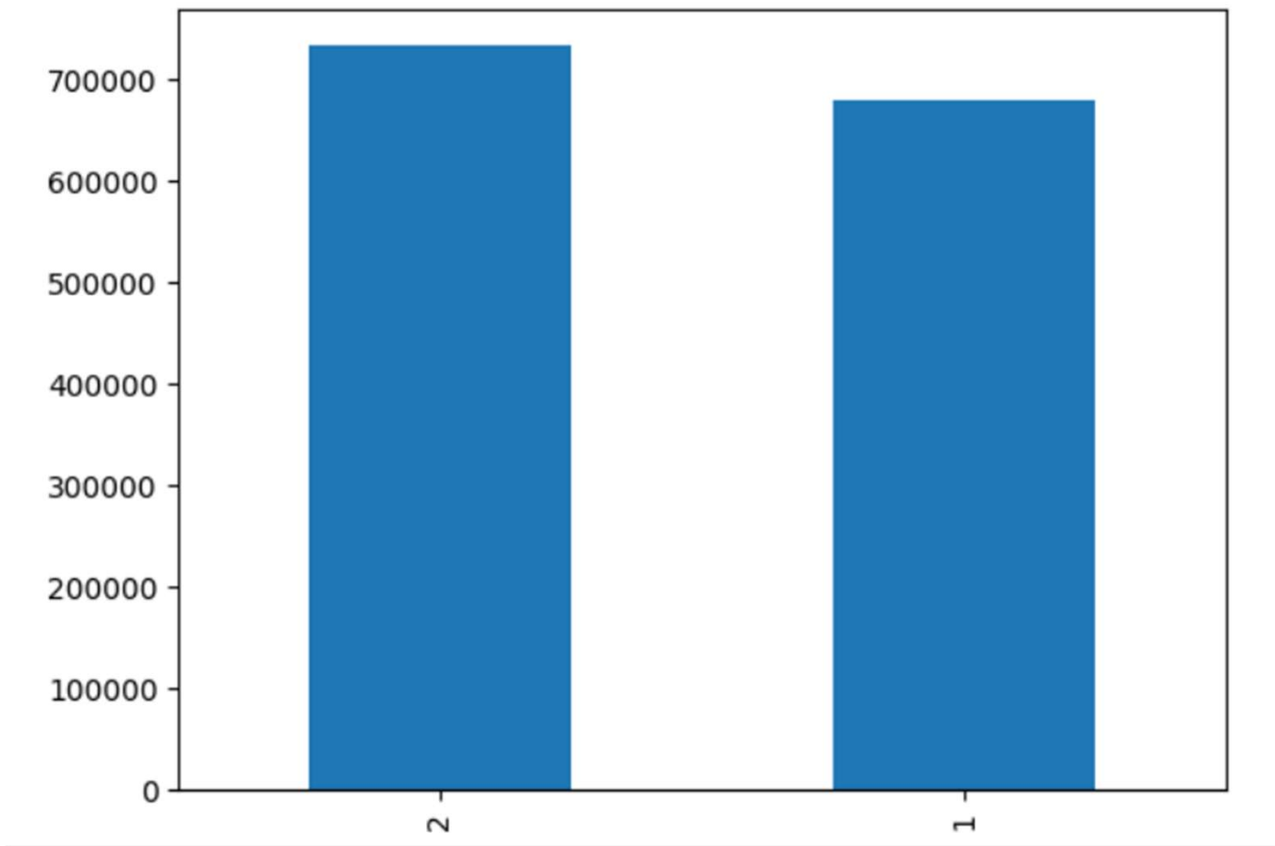


Data Exploration

- Vendor_id feature
  - There are 2 vendors
    - Vendor 1
    - Vendor 2
  - Most of them prefer using vendor 2

```
df['vendor_id'].value_counts().plot(kind="bar")
print(df['vendor_id'].value_counts())
```

```
2    732087
1    678159
Name: vendor_id, dtype: int64
```



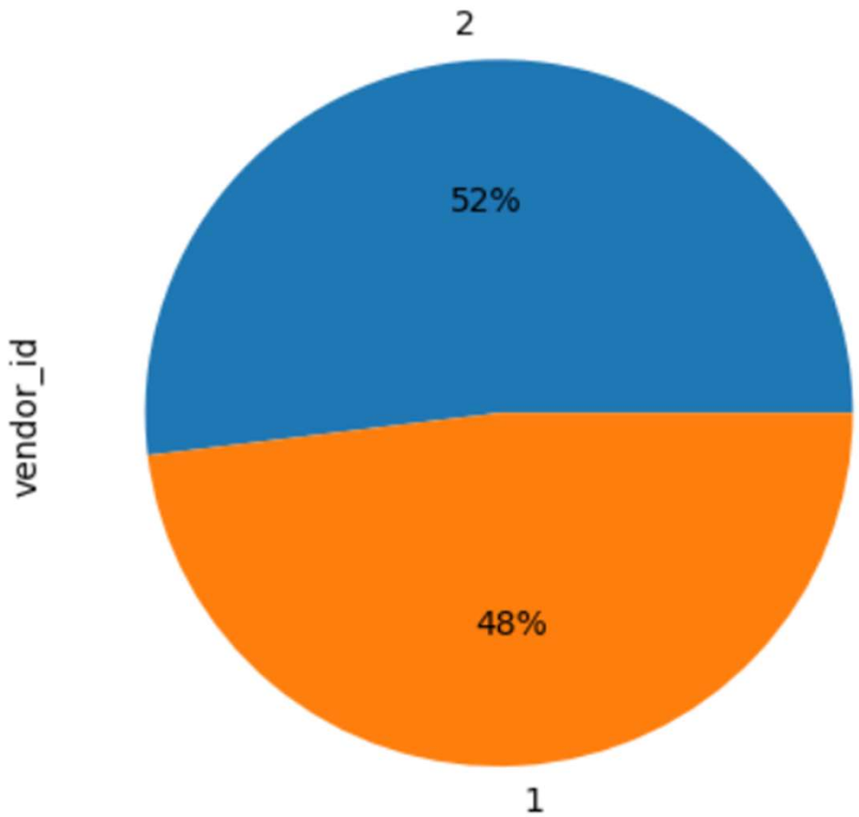
Data Exploration

- Vendor_id feature

52% of the total passengers chose vendor 2
And the rest 48% passengers chose vendor 1

```
df['vendor_id'].value_counts().plot(kind="pie", autopct='%1.0f%%')
```

```
<Axes: ylabel='vendor_id'>
```



Data Preprocessing

- We can see that there are very less percentage of passengers for 7, 8 and 9 and taxi can't be for 0 passengers. So we have to remove these.

```
print(df['passenger_count'].value_counts(normalize=True).apply(lambda x: format(x, 'f')))
df['passenger_count'].value_counts().plot(kind="pie", autopct='%1.0f%%')
```

```
1    0.708562
2    0.144187
5    0.053535
3    0.041063
6    0.033136
4    0.019473
0    0.000041
7    0.000002
9    0.000001
8    0.000001
Name: passenger_count, dtype: object
```

Data Preprocessing

- After removing them

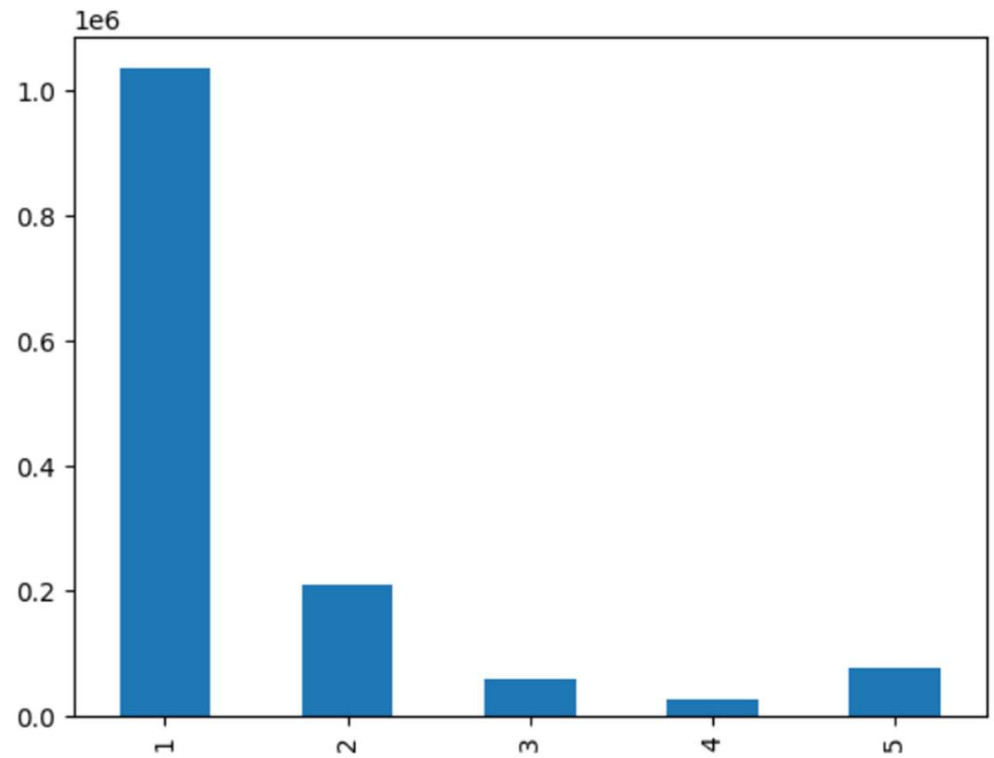
```
df = df[~((df['passenger_count'] == 0) | (df['passenger_count'] > 5))]
```

```
df['passenger_count'].value_counts().sort_index()
```

```
1    1033540
2     210318
3      59896
4      28404
5      78088
Name: passenger_count, dtype: int64
```

```
df['passenger_count'].value_counts().sort_index().plot(kind="bar")
```

```
<Axes: >
```



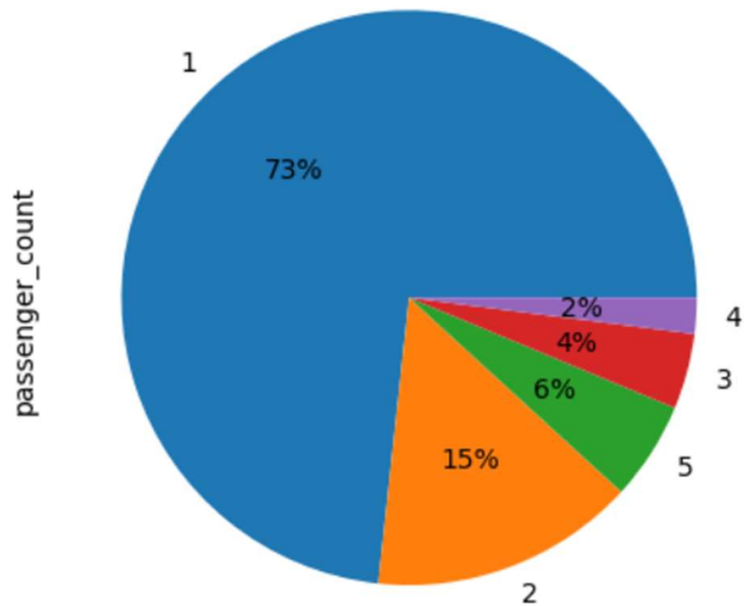
Data Preprocessing

- Now for passenger count

  1-5 passengers travel in the taxis

```
print(df['passenger_count'].value_counts(normalize=True).apply(lambda x: format(x, 'f')))
df['passenger_count'].value_counts().plot(kind="pie", autopct='%1.0f%%')
```

```
1    0.732879
2    0.149136
5    0.055372
3    0.042472
4    0.020141
Name: passenger_count, dtype: object
<Axes: ylabel='passenger_count'>
```

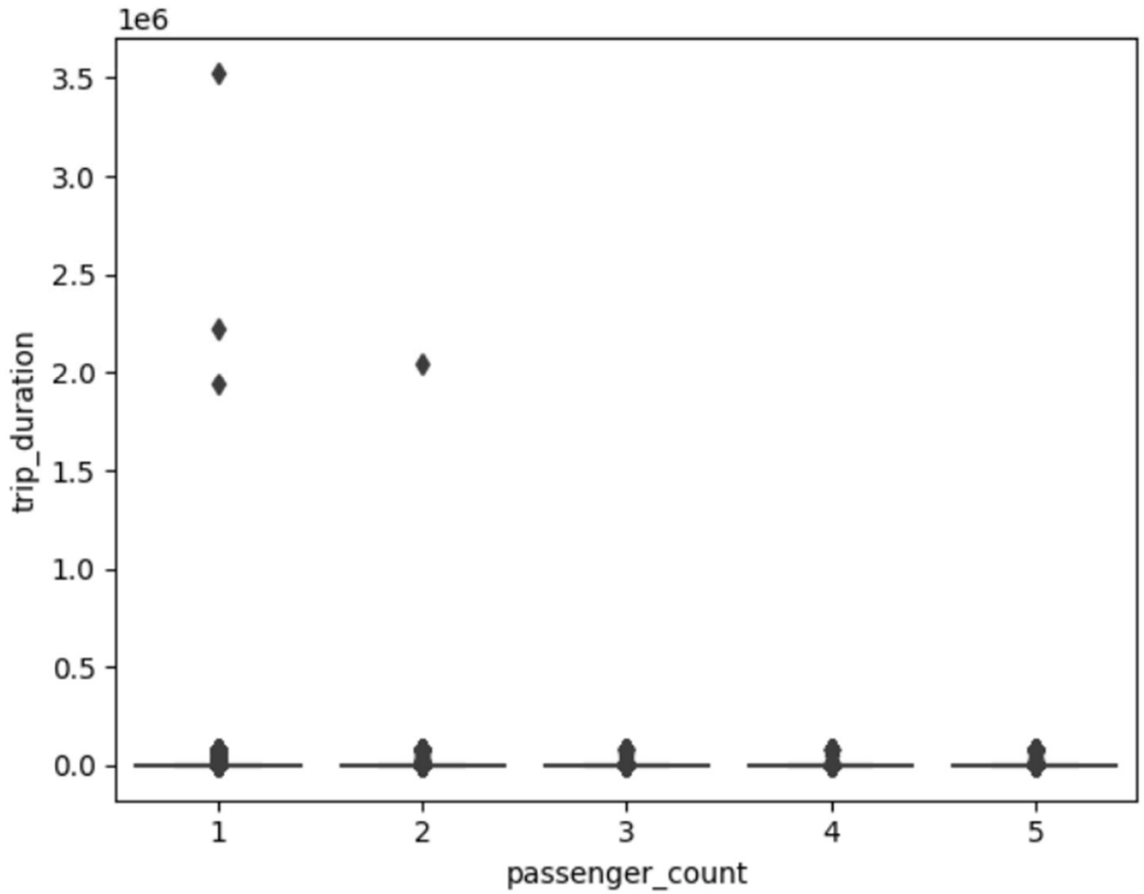


Data Preprocessing

- When we observe the relation between passenger_count and trip_duration, for some trips the duration is very high. These are the outliers and are to be removed

```
sns.boxplot(x=df['passenger_count'],y=df['trip_duration'])
```

```
<Axes: xlabel='passenger_count', ylabel='trip_duration'>
```



Data Preprocessing

- So using IQR method, we can remove the outliers
- IQR = Inter Quantile Range

```
Q1 = df['trip_duration'].quantile(0.25)
Q3 = df['trip_duration'].quantile(0.75)
IQR = Q3-Q1
lower = Q1 - 1.5 * IQR
higher = Q3 + 1.5 * IQR
df = df[~((df['trip_duration'] < lower) | (df['trip_duration'] > higher))]
```

Data Preprocessing

- Previously the pickup_datetime and dropoff_datetime is object. Changing it into the desired datatype i.e. datetime datatype

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

- And when we type df.info() their datatype will be changed to this

```
 2   pickup_datetime     1338558 non-null  datetime64[ns]
 3   dropoff_datetime    1338558 non-null  datetime64[ns]
```

Data Preprocessing

- Now using the new datatime columns, we can make new columns on it
- Day – Sunday, Monday, Tuesday…, Saturday
- Month – 1,2,3, … , 11, 12
- Weekday_num – 0,1…,6
- Pickup_hour – 0,1,2…23

```
df['day'] = df['pickup_datetime'].dt.day_name()
df['month'] = df['pickup_datetime'].dt.month
df['weekday_num'] = df['pickup_datetime'].dt.weekday
df['pickup_hour'] = df['pickup_datetime'].dt.hour
```

Data Preprocessing

- Creating new column “distance” (in km) with the latitudes and longitudes given
- Using geodesic - shortest path between the vertices

```
from geopy.distance import geodesic
```

```
distance = []
for i in range(len(df['pickup_latitude'])):
    distance.append(geodesic((df['pickup_latitude'].iloc[i],df['pickup_longitude'].iloc[i]),(df['dropoff_latitude'].iloc[i],df['dropoff_longitude'].iloc[i])).km)
df['distance'] = distance
```

Data Preprocessing

- Creating another column to see which year data exists in the dataset

```
df['year'] = df['pickup_datetime'].dt.year
```

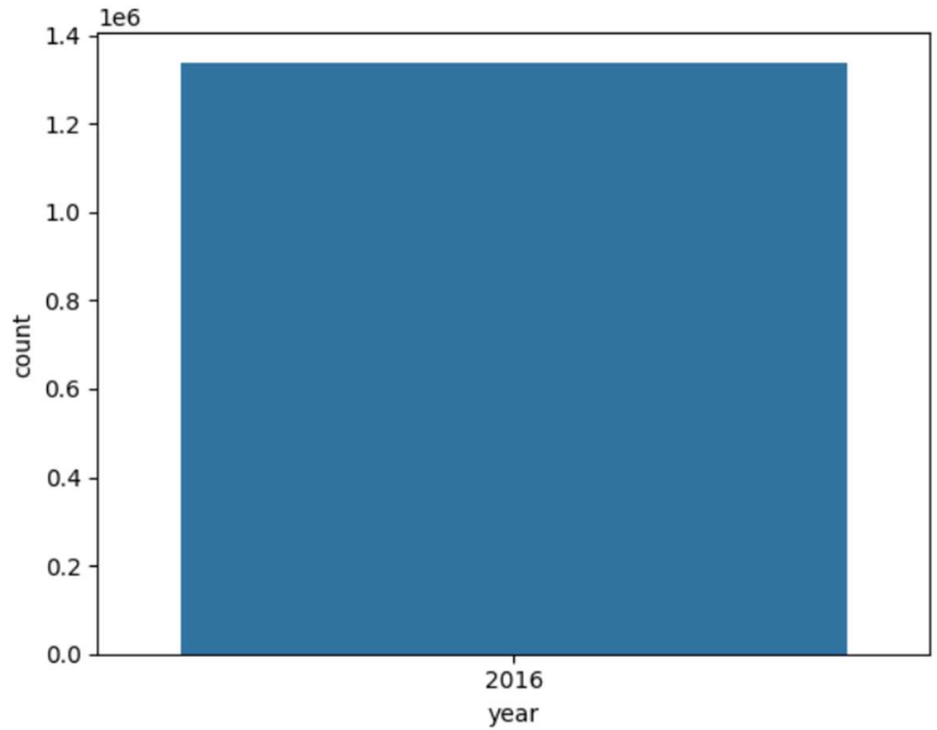
```
df['year'].value_counts()
```

```
2016    1338558
Name: year, dtype: int64
```

- So data is from the year 2016

```
sns.countplot(x='year',data=df)
```

```
<Axes: xlabel='year', ylabel='count'>
```



Data Preprocessing

- In the NYC taxi trip dataset, vendor_id and store_and_fwd_flag are the 2 categorical features.
- So applying One hot encoding on them – as there is no specific order of importance for them

```
df = pd.get_dummies(df,columns=['vendor_id','store_and_fwd_flag'])
```

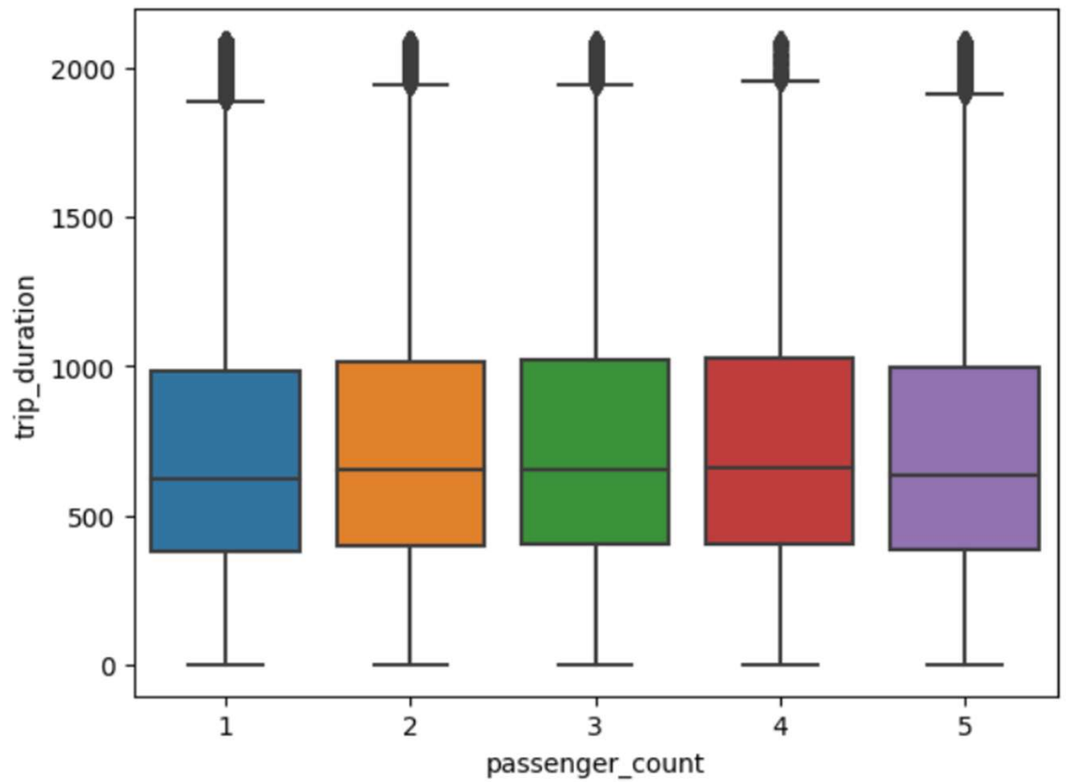
```
vendor_id_1            1338558 non-null  uint8
vendor_id_2            1338558 non-null  uint8
store_and_fwd_flag_N   1338558 non-null  uint8
store_and_fwd_flag_Y   1338558 non-null  uint8
```

Data Visualization

- Relation between passenger_count and trip_duration

```
sns.boxplot(x=df['passenger_count'],y=df['trip_duration'])
```

```
<Axes: xlabel='passenger_count', ylabel='trip_duration'>
```

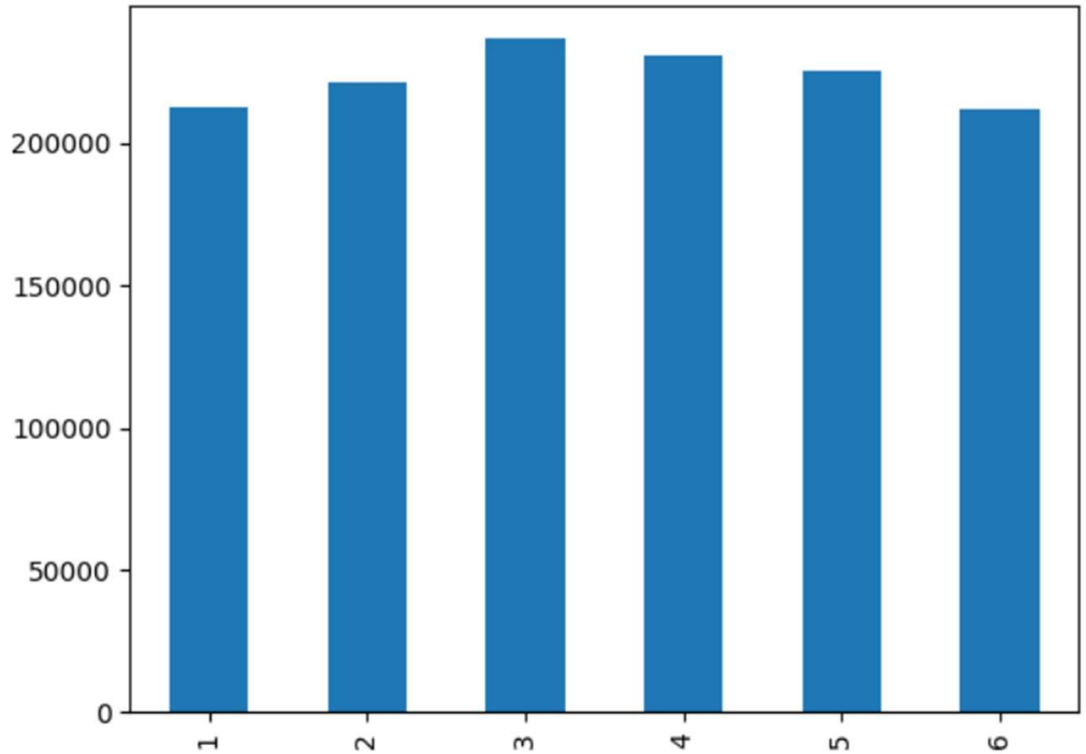


Data Visualization

- Visualizing month feature

```
print("Trip per month")
print(df['month'].value_counts().sort_index())
df['month'].value_counts().sort_index().plot(kind="bar")
```

```
Trip per month
1    212770
2    221312
3    236459
4    230674
5    225317
6    212026
```



Data of only starting 6 months are given in the dataset and mostly trips travelled are in 3 (March)

Data Visualization

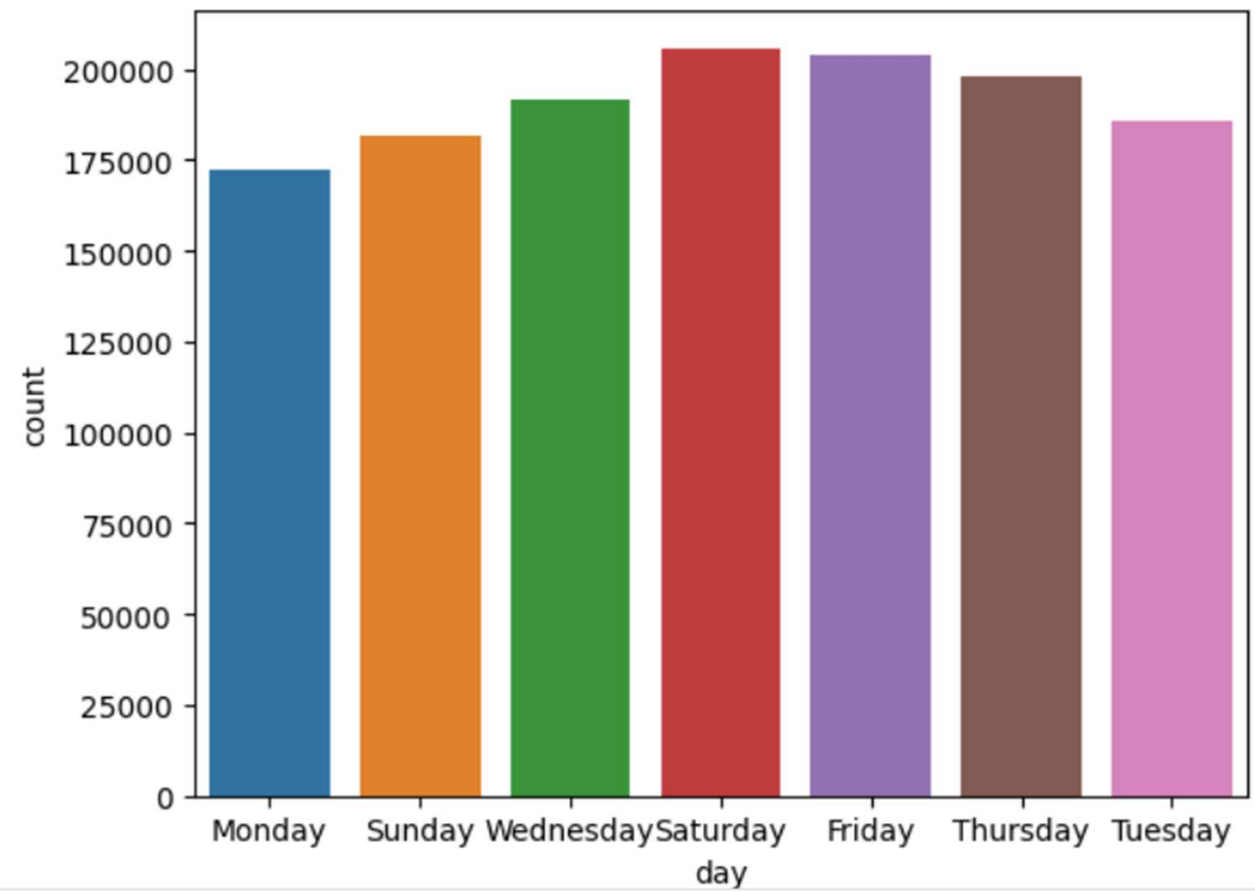
- Trips per week

More travelled on Saturday

Less travelled on Monday

```
print("Trip per week")
sns.countplot(x='day', data=df)
```

```
Trip per week
<Axes: xlabel='day', ylabel='count'>
```

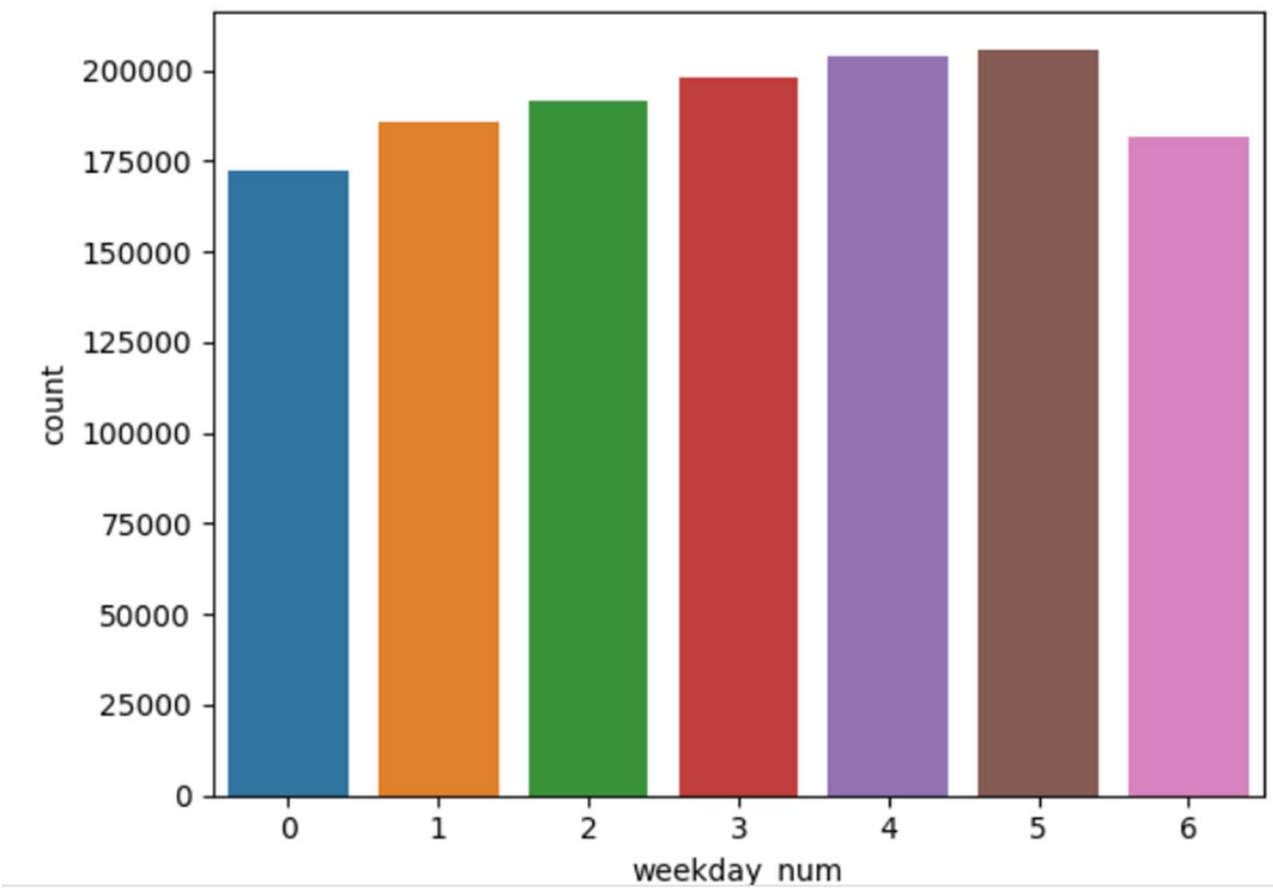


Data Visualization

- Trips per week based on weekday_num

```
print("Trip per week")
sns.countplot(x='weekday_num', data=df)
```

```
Trip per week
<Axes: xlabel='weekday_num', ylabel='count'>
```

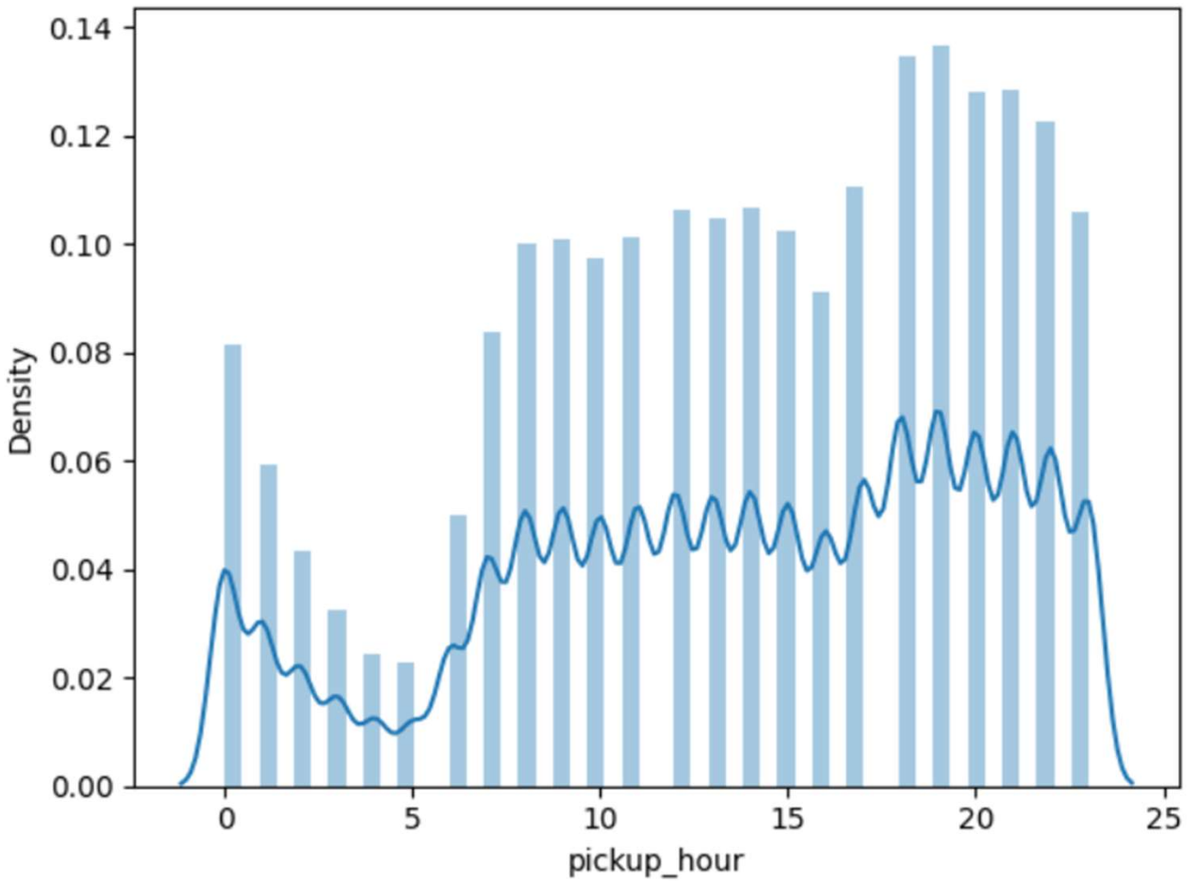


Data Visualization

- Based on Hour

```
sns.distplot(df['pickup_hour'])
```

- Density is more at nighttime than daytime.
- In between 7-8pm, the trips are more



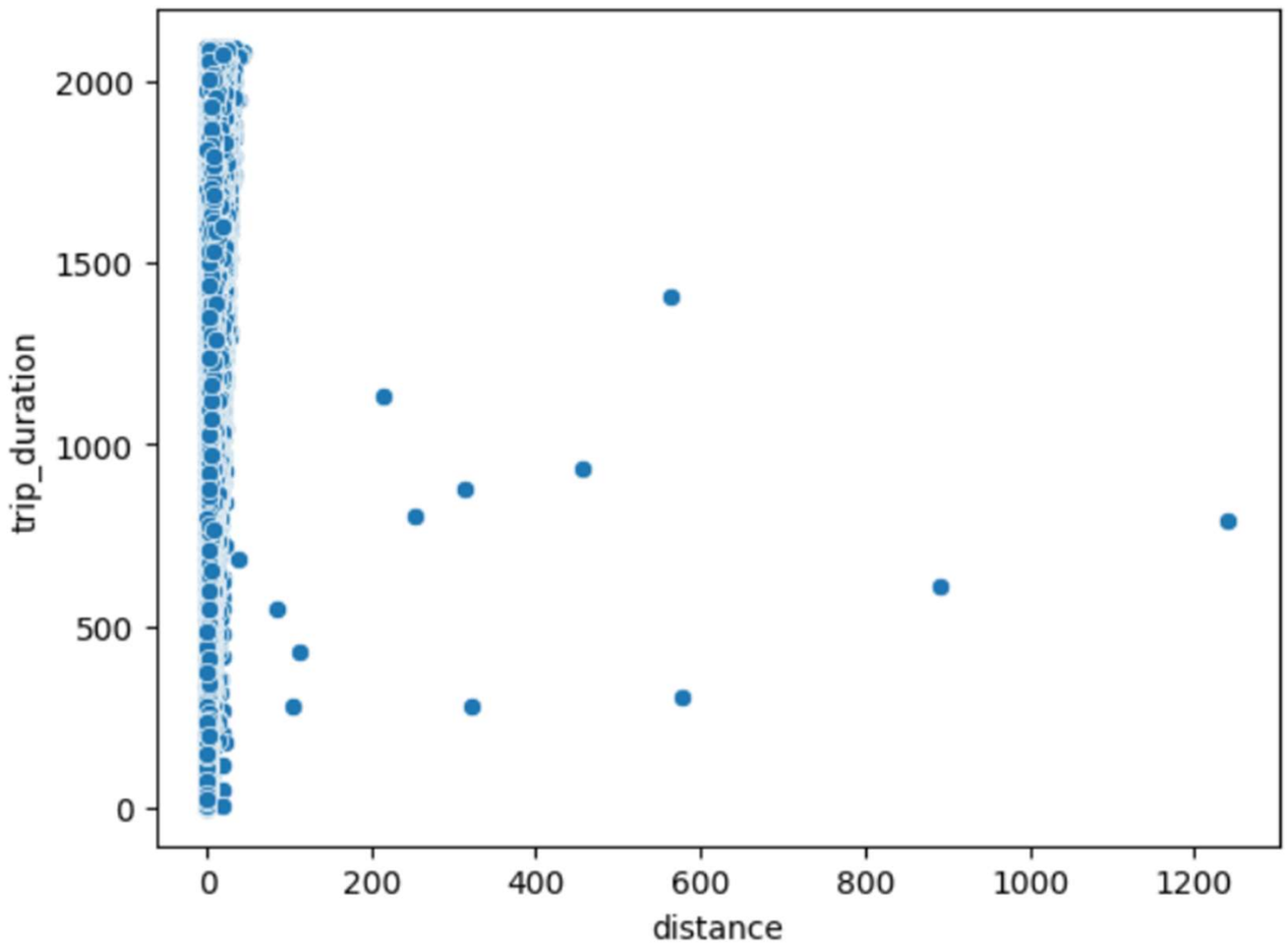
Data Visualization

- Distance

  Most of the trips are short trips

```
sns.scatterplot(x=df['distance'],y=df['trip_duration'])
```

```
<Axes: xlabel='distance', ylabel='trip_duration'>
```

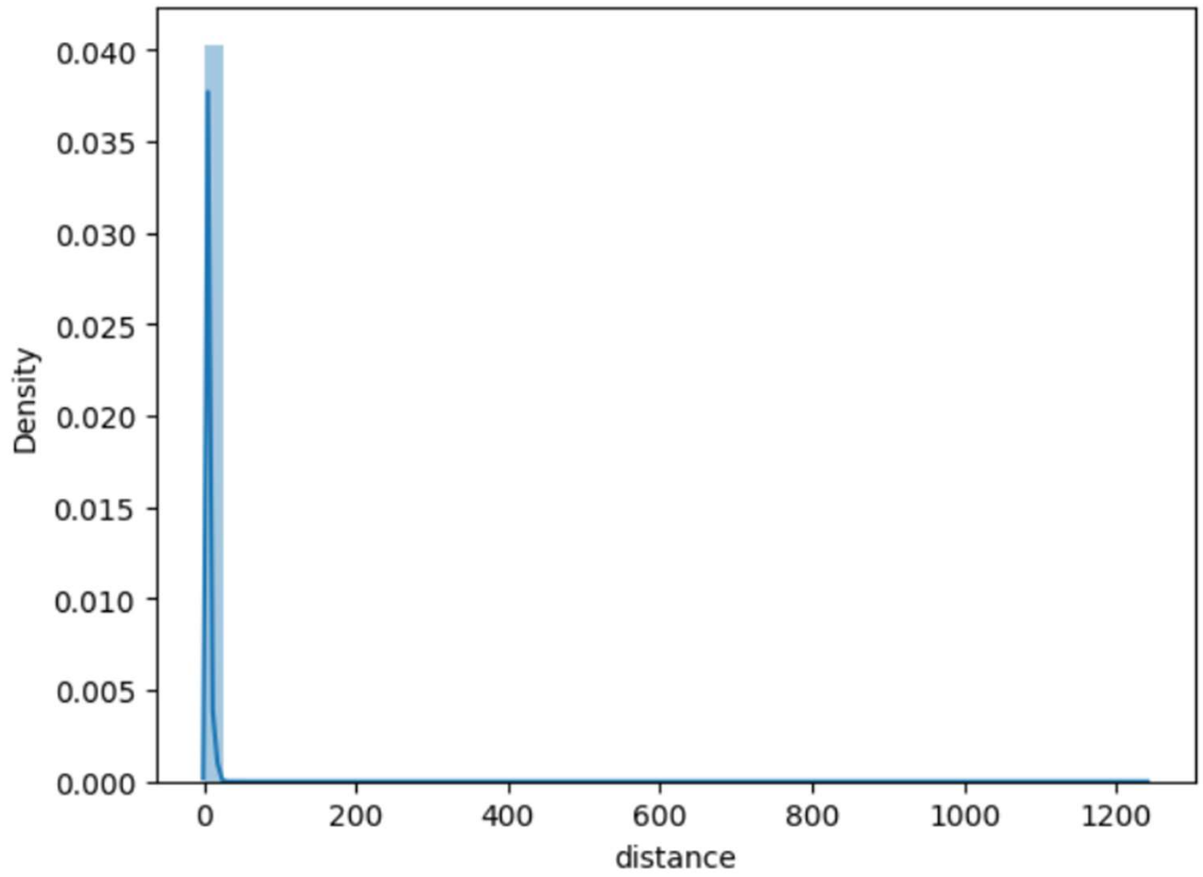


Data Visualization

- Distance

```
sns.distplot(df['distance'])
```

- Density is high for small trips



Data Visualization

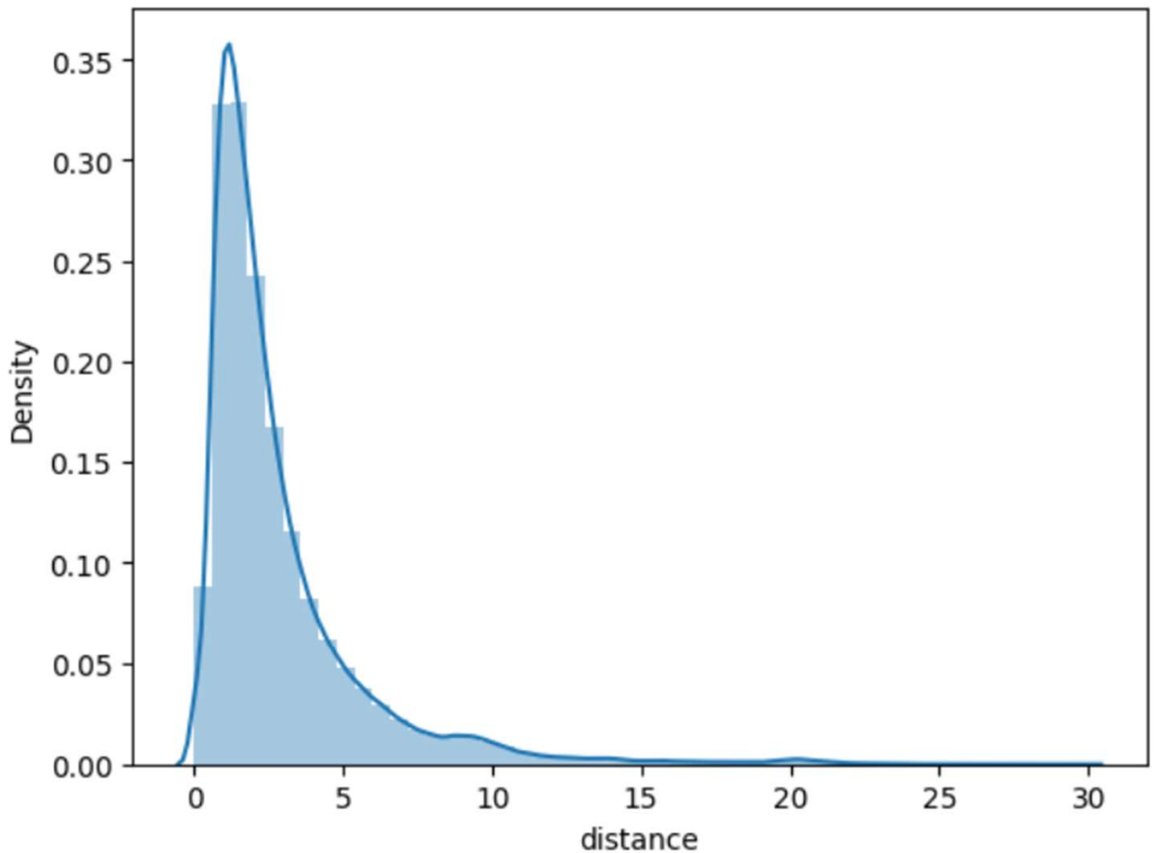
- Distance

  Analyzing the distance for

  Less than 30kms

```
sns.distplot(df['distance'][df['distance'] < 30])
```

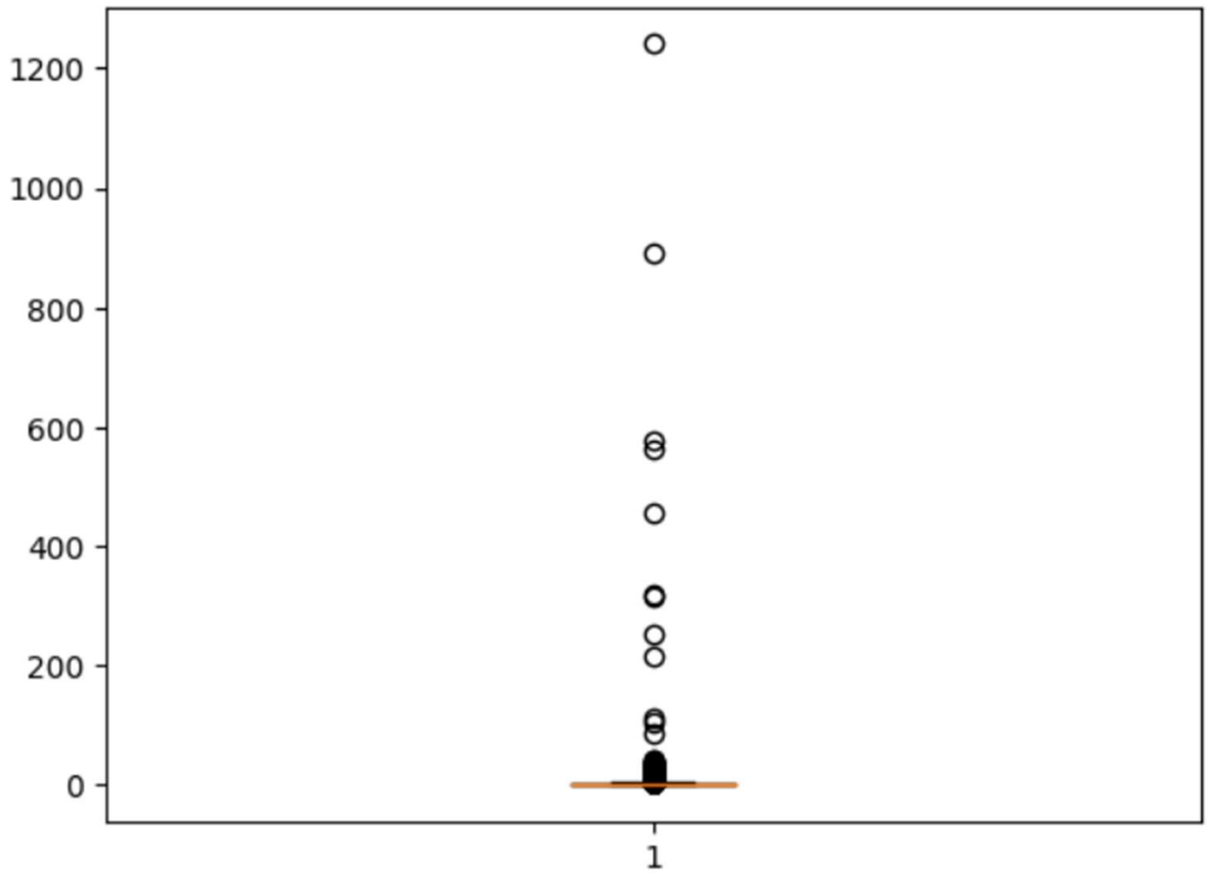
- We can observe that more trips are booked for less than 3 kms



Data Visualization

- Boxplot of Distance

```
plt.boxplot(df['distance'])
```



Data Visualization

- Trips with distances above 100 kms

After preprocessing and removing outliers in the data, we have 11 trips more than 100 km

```
df['distance'][df['distance'] > 100].value_counts()
```

```
1240.510256    1
314.260625     1
112.829898     1
320.445202     1
105.023359     1
253.987879     1
578.579744     1
563.028024     1
891.663777     1
456.364941     1
215.511294     1
Name: distance, dtype: int64
```

Data Visualization

- Now to draw heatmap, there should not be any categorical variables. Here we have day. Dropping it

```
df.drop('day', axis=1, inplace=True)
```

- As the year column have only 2016 and has no relation with the trip_duration value, we can drop that column too

```
df.drop('year', axis=1, inplace=True)
```

Data Visualization

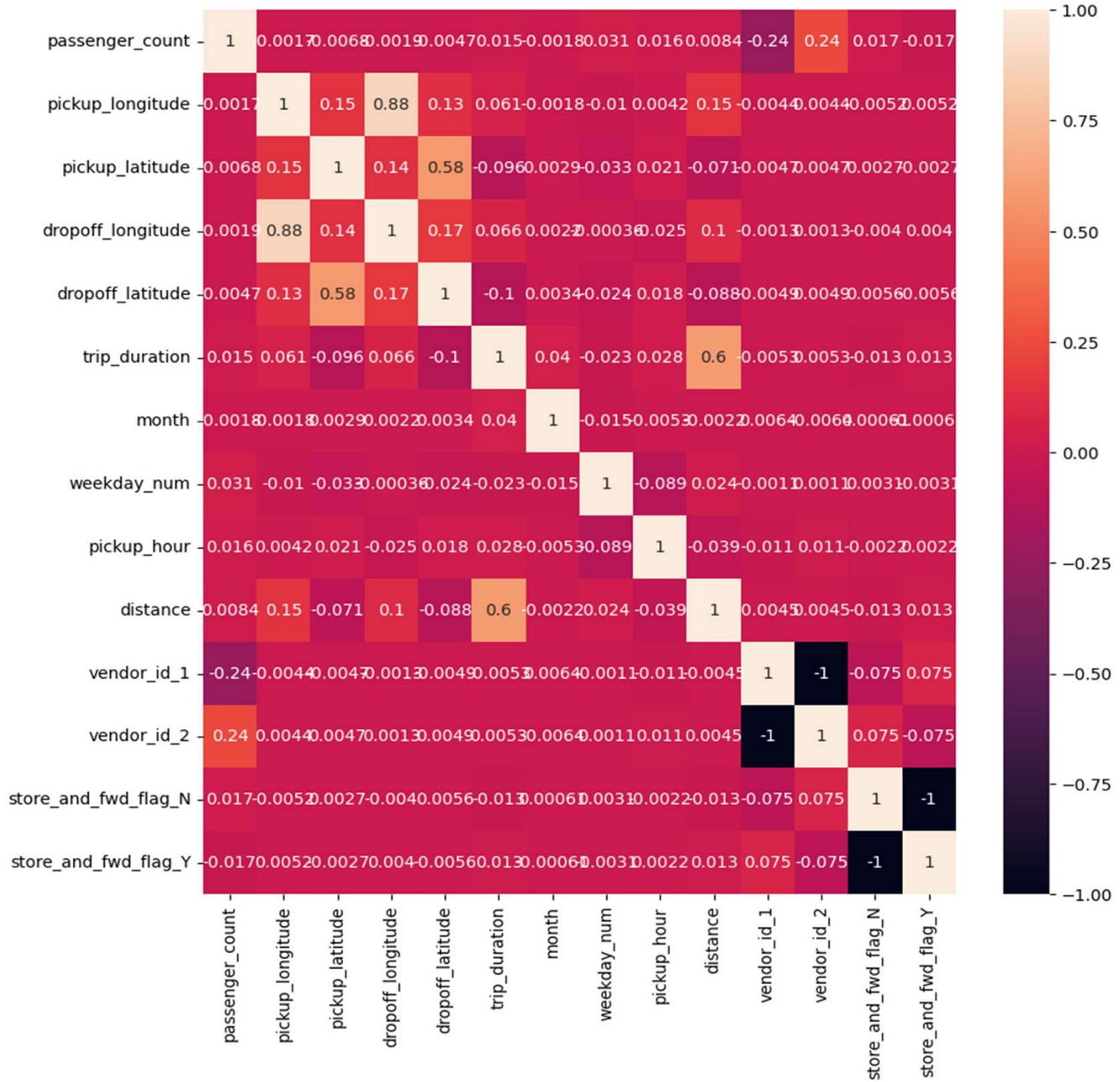
- Now the columns are

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338558 entries, 0 to 1458643
Data columns (total 16 columns):
 #   Column                Non-Null Count    Dtype
---  ------                --------------    -----
 0   id                    1338558 non-null  object
 1   pickup_datetime       1338558 non-null  datetime64[ns]
 2   passenger_count       1338558 non-null  int64
 3   pickup_longitude      1338558 non-null  float64
 4   pickup_latitude       1338558 non-null  float64
 5   dropoff_longitude     1338558 non-null  float64
 6   dropoff_latitude      1338558 non-null  float64
 7   trip_duration         1338558 non-null  int64
 8   month                 1338558 non-null  int64
 9   weekday_num           1338558 non-null  int64
 10  pickup_hour           1338558 non-null  int64
 11  distance              1338558 non-null  float64
 12  vendor_id_1           1338558 non-null  uint8
 13  vendor_id_2           1338558 non-null  uint8
 14  store_and_fwd_flag_N  1338558 non-null  uint8
 15  store_and_fwd_flag_Y  1338558 non-null  uint8
dtypes: datetime64[ns](1), float64(5), int64(5), object(1), uint8(4)
memory usage: 137.9+ MB
```

Data Visualization

- Correlation between different features in the dataframe



Data Modelling

- For modelling we should have only the columns with numbers. All other columns with other datatypes to be dropped.
- Thus, columns to drop are:

```
columns_to_drop = ['id', 'pickup_datetime','pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude']
```

- Only columns which are not created from one hot encoding are to be scaled. Here, scaled and unscaled columns are

```
scale_column = ['passenger_count','month', 'weekday_num', 'pickup_hour', 'distance']
```

```
not_scale_columns = ['vendor_id_1', 'vendor_id_2', 'store_and_fwd_flag_N','store_and_fwd_flag_Y']
```

Data Modelling

- Target column is stored in Y and the features in X

```
Y = df['trip_duration']
X = df.drop('trip_duration',axis=1)
```

- Dropping the columns in X

```
X = X.drop(columns_to_drop,axis=1)
```

```
X.columns
```

```
Index(['passenger_count', 'month', 'weekday_num', 'pickup_hour', 'distance',
       'vendor_id_1', 'vendor_id_2', 'store_and_fwd_flag_N',
       'store_and_fwd_flag_Y'],
      dtype='object')
```

Data Modelling

- Scaling the columns in X

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X[scale_column] = scaler.fit_transform(X[scale_column])
```

Thus, for modelling we have 9 features.

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338558 entries, 0 to 1458643
Data columns (total 9 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   passenger_count        1338558 non-null  float64
 1   month                  1338558 non-null  float64
 2   weekday_num            1338558 non-null  float64
 3   pickup_hour            1338558 non-null  float64
 4   distance               1338558 non-null  float64
 5   vendor_id_1            1338558 non-null  uint8
 6   vendor_id_2            1338558 non-null  uint8
 7   store_and_fwd_flag_N   1338558 non-null  uint8
 8   store_and_fwd_flag_Y   1338558 non-null  uint8
dtypes: float64(5), uint8(4)
memory usage: 66.4 MB
```

Data Modelling

- Splitting the data into 80:20 ratio (80 – train, 20 – test)

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state=31)
```

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

```
(1070846, 9) (1070846,)
(267712, 9) (267712,)
```

Data Modelling

- Linear Regression

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
```

```
lr = LinearRegression()
lr.fit(X_train,y_train)
lr_train_score = lr.score(X_train, y_train)
lr_train_score
```

```
0.3483352016438638
```

```
lr_test_score = lr.score(X_test, y_test)
lr_test_score
```

```
0.4249357782699188
```

Linear regression r2 score: 0.4249

Data Modelling

- Lasso Regression

```
lasso_0001 = Lasso(alpha=0.0001,max_iter=int(1e5))
lasso_0001.fit(X_train,y_train)
lasso_0001_train_score = lasso_0001.score(X_train, y_train)
print(lasso_0001_train_score)
print("Selected coeff:", np.sum(lasso_0001.coef_!= 0))
```

```
0.3483356761745229
Selected coeff: 9
```

```
lasso_001 = Lasso(alpha=0.001,max_iter=int(1e5))
lasso_001.fit(X_train,y_train)
lasso_001_train_score = lasso_001.score(X_train, y_train)
print(lasso_001_train_score)
print("Selected coeff:", np.sum(lasso_001.coef_!= 0))
```

```
0.3483356751665444
Selected coeff: 7
```

Data Modelling

- Lasso Regression

```
lasso_1 = Lasso(alpha=1,max_iter=int(1e5))
lasso_1.fit(X_train,y_train)
lasso_1_train_score = lasso_1.score(X_train, y_train)
print(lasso_1_train_score)
print("Selected coeff:", np.sum(lasso_1.coef_!= 0))
```

```
0.348280443964607
Selected coeff: 5
```

```
lasso_10 = Lasso(alpha=10,max_iter=int(1e5))
lasso_10.fit(X_train,y_train)
lasso_10_train_score = lasso_10.score(X_train, y_train)
print(lasso_10_train_score)
print("Selected coeff:", np.sum(lasso_10.coef_!= 0))
```

```
0.346259181681110l4
Selected coeff: 4
```

Data Modelling

- Decision tree Regressor

```
from sklearn.tree import DecisionTreeRegressor
```

```
dtr = DecisionTreeRegressor(random_state=31)
dtr.fit(X_train,y_train)
```

```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=31)
```

```
dtr_train_score = dtr.score(X_train, y_train)
dtr_train_score
```

```
0.9983894058603949
```

```
dtr_test_score = dtr.score(X_test,y_test)
dtr_test_score
```

```
0.30441053362438797
```

Decision Tree Regressor with random state 31 - r2 score: 0.3044

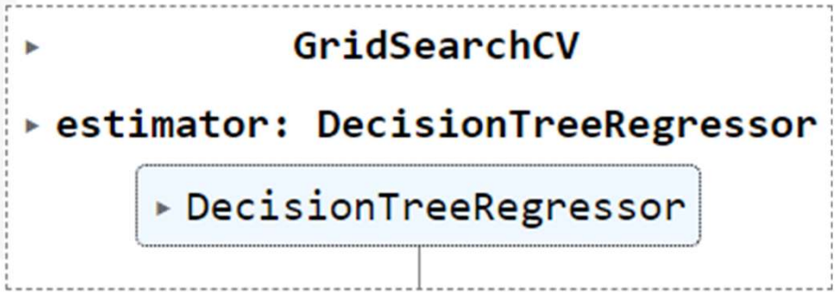
Data Modelling

- Decision tree regressor using GridSearchCV

```
max_depth = [4,6,8,10]
min_samples_split = [10,20,30]
min_samples_leaf = [10,16,20]
param_dt = {
                'max_depth' : max_depth,
                'min_samples_split' : min_samples_split,
                'min_samples_leaf' : min_samples_leaf}
```

```
dtmodel = DecisionTreeRegressor()
dt_grid = GridSearchCV(estimator=dtmodel,
                       param_grid = param_dt,
                       cv = 5, verbose=2, scoring='r2')

dt_grid.fit(X_train,y_train)
```




```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total time=   2.1s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total time=   2.0s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total time=   2.2s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total time=   1.8s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=10; total time=   1.8s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total time=   2.3s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total time=   2.6s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total time=   1.7s
[CV] END max_depth=4, min_samples_leaf=10, min_samples_split=20; total time=   1.6s
```

Training Score:

```
dt_grid.best_score_
```

```
0.6582805816569899
```

Data Modelling

- Decision Tree Regressor using GridSearchCV

```
dt_optimal_model = dt_grid.best_estimator_
```

```
dt_optimal_model.score(X_test,y_test)
```

```
0.6587428312675909
```

Decision Tree Regressor best estimator with given hyperparameters r2 score: 0.6587

- Mean Squared Error

```
y_pred_dto_test=dt_optimal_model.predict(X_test)
```

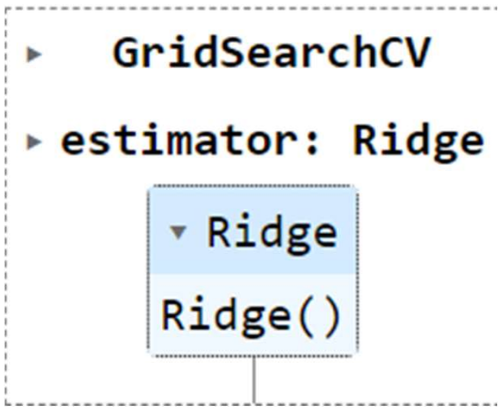
```
mean_squared_error(y_test,y_pred_dto_test,squared=False)
```

```
261.88374322926035
```

Data Modelling

- Ridge Regression using GridSearchCV

```
ridge = Ridge()
parameters = {'alpha': [1e-15,1e-13,1e-10,1e-8,1e-5,1e-4,1e-3,1e-2,1e-1,1,5,10,20,30,40,45,50,55,60,100]}
ridge_regressor = GridSearchCV(ridge, parameters, scoring='r2', cv=5)
ridge_regressor.fit(X_train, y_train)
```




```
ridge_regressor.score(X_test,y_test)
```

```
0.42504924278625533
```

```
ridge_regressor.best_params_
```

```
{'alpha': 100}
```

Ridge regression using GridSearchCV with different Paremeters r2 score: 0.4250
Best parameter – alpha = 100

Data Modelling

- XGBoost Regression

```
n_estimators = [50,100,120]
max_depth = [5,7,9]
min_samples_split = [40,50]
param_xgb = {'n_estimators' : n_estimators,
             'max_depth' : max_depth,
            'min_samples_split':min_samples_split
             }
```

```
xgb_model = XGBRegressor()
xgb_grid = GridSearchCV(estimator=xgb_model,param_grid = param_xgb,cv = 3, verbose=2,scoring="r2")
xgb_grid.fit(X_train,y_train)
```

```
xgb_optimal_model = xgb_grid.best_estimator_
xgb_optimal_model.score(X_test,y_test)
```

```
0.6662572042216606
```

XGBoost regression algorithm with GridSearchCV best model-r2 score: 0.6662

Data Modelling

- Gradient Boost Regression

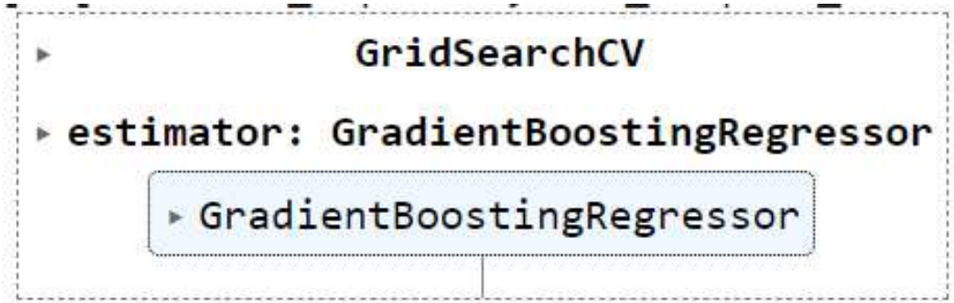
```
n_estimators = [100,120]
max_depth = [5,8,10]
min_samples_split = [50,80]
min_samples_leaf = [40,50]
param_gb = {'n_estimators' : n_estimators,
            'max_depth' : max_depth,
            'min_samples_split' : min_samples_split,
            'min_samples_leaf' : min_samples_leaf}
```

```
gb_model=GradientBoostingRegressor()
gb_grid = GridSearchCV(estimator=gb_model,
                       param_grid = param_gb,
                       cv = 3, verbose=2, scoring='r2')
gb_grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=100; total time= 2.2min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=100; total time= 2.3min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=100; total time= 2.2min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=120; total time= 2.6min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=120; total time= 2.6min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=50, n_estimators=120; total time= 2.7min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=100; total time= 2.2min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=100; total time= 2.2min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=100; total time= 2.3min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=120; total time= 2.6min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=120; total time= 2.7min
[CV] END max_depth=5, min_samples_leaf=40, min_samples_split=80, n_estimators=120; total time= 2.6min
```

Data Modelling

- Gradient Boost Regression




```
gb_optimal_model = gb_grid.best_estimator_
gb_optimal_model.score(X_test,y_test)
```

```
0.666812240699227
```

```
gb_grid.best_estimator_
```

```
GradientBoostingRegressor(max_depth=8, min_samples_leaf=50,
                          min_samples_split=50)
```

Gradient Boost Algorithm using GridSearchCV
R2 score: 0.6669

Data Modelling

- Random Forest

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf = RandomForestRegressor()
rf.fit(X_train, y_train)
```

RandomForestRegressor

RandomForestRegressor()

```
rf.score(X_test,y_test)
```

```
0.6152319629687303
```

Random Forest Regression
R2 score: 0.6152

Conclusion

- The most important features for the model are: distance, month, vendor_id, store_and_fwd_flag, passenger_count, weekday_num and pickup_hour
- The target variable is trip_duration
- This is a regression problem
- Thus, from the algorithms used and models trained we can see that highest score is for Gradient boost regression which is 0.6669
- That means, model explains or predicts 66.69% of the relationship between dependent and independent variables

Thank You!