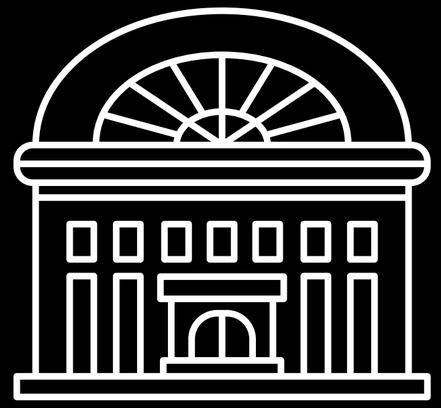
NYC Taxi trip prediction Project

Submitted By : Ashmika Dwivedi,Vellore Institute of Technologyhab,ilitation
ReSport



OBJECTIVE

Creating a model that predicts the complete ride duration of New York City taxi trips.
The objective is to preprocess, visualize, and design the optimal model for the task.
The dataset used is one released by the New York City Taxi and Limousine Commission, and it includes factors is such as geo-coordinates, passenger count, date and time, and so on.

INTRODUCTION

This project predicts the trip duration of a NYC taxi by using the dataset provided for training and testing as mentioned.
Then we fit out dataset into the Linear Regression, Lasso, and Ridge Models.
Before doing so we must preprocess, clean the data. Data exploration and visualization is also done.

Data Exploration

Data exploration for a New York City taxi trip duration prediction model involves closely examining the dataset's characteristics. We start by understanding the data's structure, checking for missing values, and examining the distributions of key variables like trip duration and fares

```
In [2]: df.columns

Out[2]: 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')

In [3]: df.shape

Out[3]: (1458644, 11)

In [4]: df.describe

Out[4]: <bound method NDFrame.describe of                 id  vendor_id      pickup_datetime     dropoff_datetime
        0        id2875421          2  2016-03-14 17:24:55  2016-03-14 17:32:30
        1        id2377394          1  2016-06-12 00:43:35  2016-06-12 00:54:38
        2        id3858529          2  2016-01-19 11:35:24  2016-01-19 12:10:48
        3        id3504673          2  2016-04-06 19:32:31  2016-04-06 19:39:40
        4        id2181028          2  2016-03-26 13:30:55  2016-03-26 13:38:10
        ...            ...        ...                  ...                  ...
        1458639  id2376096          2  2016-04-08 13:31:04  2016-04-08 13:44:02
        1458640  id1049543          1  2016-01-10 07:35:15  2016-01-10 07:46:10
        1458641  id2304944          2  2016-04-22 06:57:41  2016-04-22 07:10:25
        1458642  id2714485          1  2016-01-05 15:56:26  2016-01-05 16:02:39
        1458643  id1209952          1  2016-04-05 14:44:25  2016-04-05 14:47:43
```

Here we are checking if there are any null values present in our dataset.

- We are also using functions like df.info and df.describe to gather more information from our dataset.(output screenshot on previous slide)
- df.shape() function tells us how many rows and columns are present in our dataset.

snull()

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
639	False	False	False	False	False	False	False	False	False
640	False	False	False	False	False	False	False	False	False
641	False	False	False	False	False	False	False	False	False
642	False	False	False	False	False	False	False	False	False
643	False	False	False	False	False	False	False	False	False

644 rows × 11 columns

```
In [5]: df.info

Out[5]: <bound method DataFrame.info of                 id  vendor_id      pickup_datetime     dropoff_datetime  \
0        id2875421          2  2016-03-14 17:24:55  2016-03-14 17:32:30
1        id2377394          1  2016-06-12 00:43:35  2016-06-12 00:54:38
2        id3858529          2  2016-01-19 11:35:24  2016-01-19 12:10:48
3        id3504673          2  2016-04-06 19:32:31  2016-04-06 19:39:40
4        id2181028          2  2016-03-26 13:30:55  2016-03-26 13:38:10
...            ...        ...                  ...                  ...
1458639  id2376096          2  2016-04-08 13:31:04  2016-04-08 13:44:02
1458640  id1049543          1  2016-01-10 07:35:15  2016-01-10 07:46:10
1458641  id2304944          2  2016-04-22 06:57:41  2016-04-22 07:10:25
1458642  id2714485          1  2016-01-05 15:56:26  2016-01-05 16:02:39
1458643  id1209952          1  2016-04-05 14:44:25  2016-04-05 14:47:43

         passenger_count  pickup_longitude  pickup_latitude  \
0                      1        -73.982155        40.767937
1                      1        -73.980415        40.738564
2                      1        -73.979027        40.763939
3                      1        -74.010040        40.719971
4                      1        -73.973053        40.793209
...                  ...               ...              ...
1458639                4        -73.982201        40.745522
1458640                1        -74.000946        40.747379
1458641                1        -73.959129        40.768799
1458642                1        -73.982079        40.749062
```

```
3                      1        -74.010040        40.719971
4                      1        -73.973053        40.793209
...                  ...               ...              ...
1458639                4        -73.982201        40.745522
1458640                1        -74.000946        40.747379
1458641                1        -73.959129        40.768799
1458642                1        -73.982079        40.749062
1458643                1        -73.979538        40.781750

         dropoff_longitude  dropoff_latitude store_and_fwd_flag  trip_duration
0                -73.964630         40.765602                  N            455
1                -73.999481         40.731152                  N            663
2                -74.005333         40.710087                  N           2124
3                -74.012268         40.706718                  N            429
4                -73.972923         40.782520                  N            435
...                     ...               ...                ...            ...
1458639          -73.994911         40.740170                  N            778
1458640          -73.970184         40.796547                  N            655
1458641          -74.004433         40.707371                  N            764
1458642          -73.974632         40.757107                  N            373
1458643          -73.972809         40.790585                  N            198

[1458644 rows x 11 columns]>
```

DATA PREPROCESSING

I In datapreprocessing, we do the following steps

1. **Remove any Null values present.**

2.**Feature Engineering:** Here we create new features that could enhance the model's predictive power. For example, calculate distances between pickup and drop-off coordinates (as trip_distance)extract day of the week, time of day,extract day of the week, time of day. etc.

3.**Label-Encoding-** here label encoding is done for the following variables (categorical ones) which are vendor_id, and store_and_fwd_flag

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

```
In [8]:
df = pd.get_dummies(df,columns=['vendor_id','store_and_fwd_flag'])
```

```
from geopy.distance import geodesic
import pandas as pd

# Assuming you already have a DataFrame 'df' with columns 'pickup_latitude', 'pickup_longitude', 'dropoff_latitude', 'dropoff

def calculate_trip_distance(row):
    pickup_coords = (row['pickup_latitude'], row['pickup_longitude'])
    dropoff_coords = (row['dropoff_latitude'], row['dropoff_longitude'])
    return geodesic(pickup_coords, dropoff_coords).km

df['trip_distance'] = df.apply(calculate_trip_distance, axis=1)
```

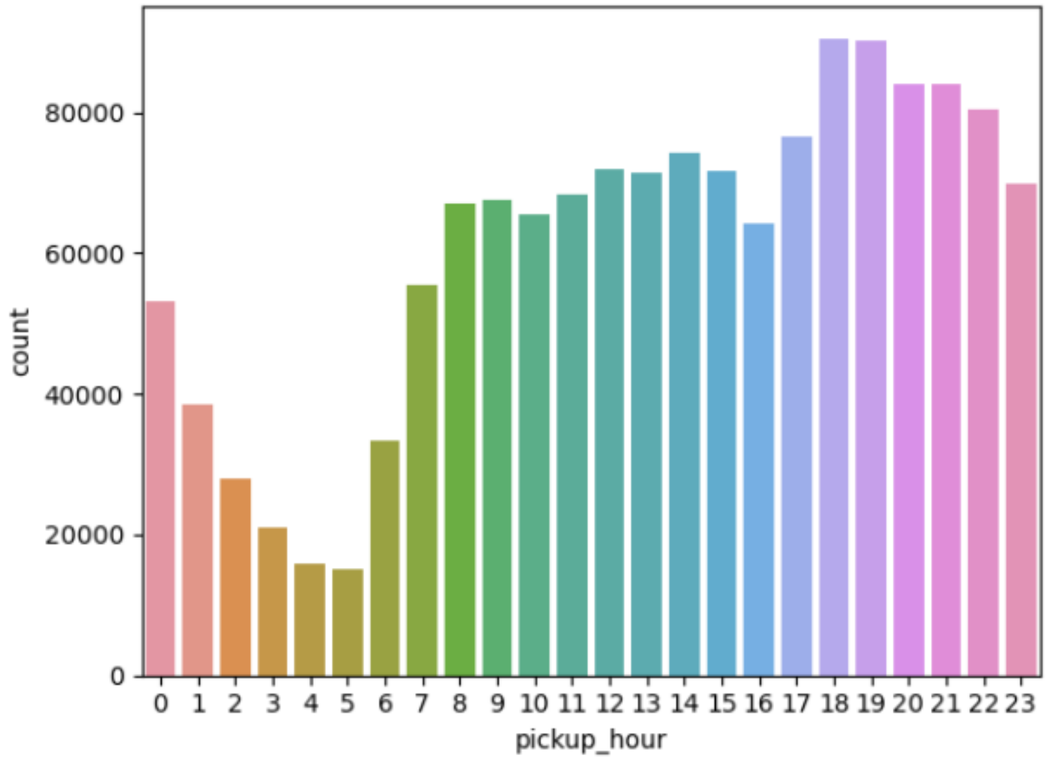
Data Visualization

Visualizing the NYC taxi trip duration data provides valuable insights into patterns and relationships within the dataset. Through interactive maps, scatter plots & charts, we can uncover geographic clusters of trips, understand peak travel times, and explore how variables like passenger_count/vendor ID impact trip durations.
Here , we are Vvsualizing at which hour of the day there was most pickups are done
i.e. busiest hour of the day with more taxi requests.

```
In [98]: sns.countplot(x='pickup_hour', data=df)
```

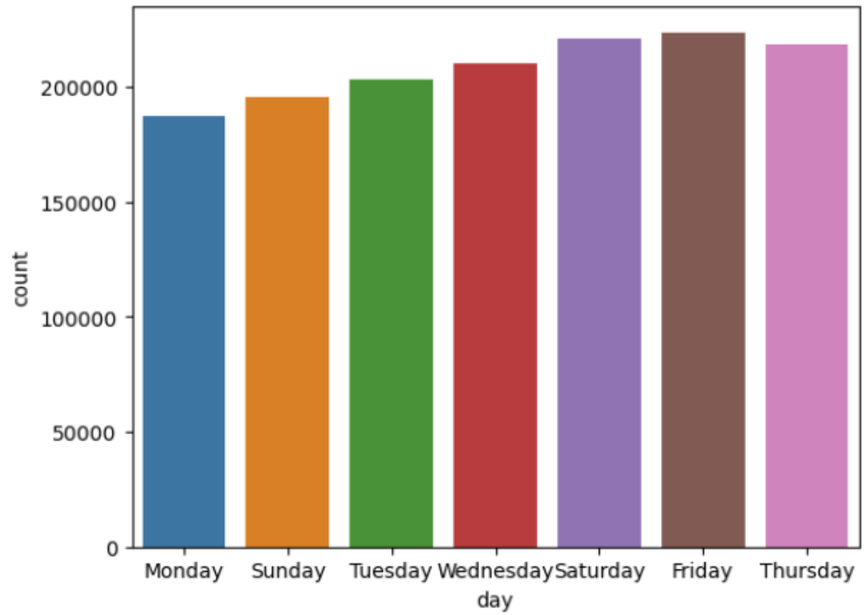
Trip per week

```
Out[98]: <Axes: xlabel='pickup_hour', ylabel='count'>
```

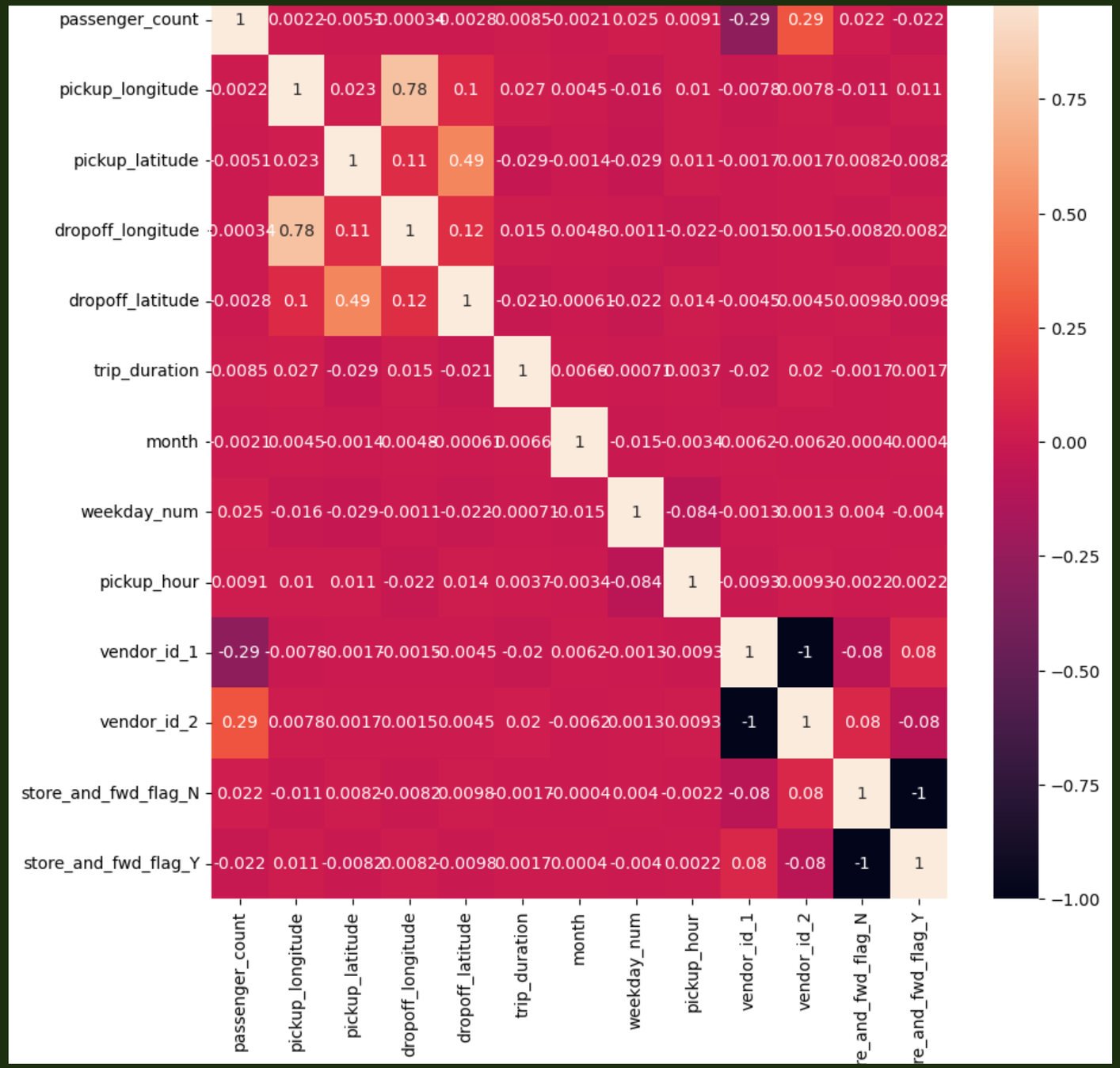


DATA VISUALIZATION

```
In [95]: print("Trip per week")
         sns.countplot(x='day', data=df)

         Trip per week

Out[95]: <Axes: xlabel='day', ylabel='count'>
```



this countplot helps us visualize the trips made per week. I have used the seaborn library for this implementation.

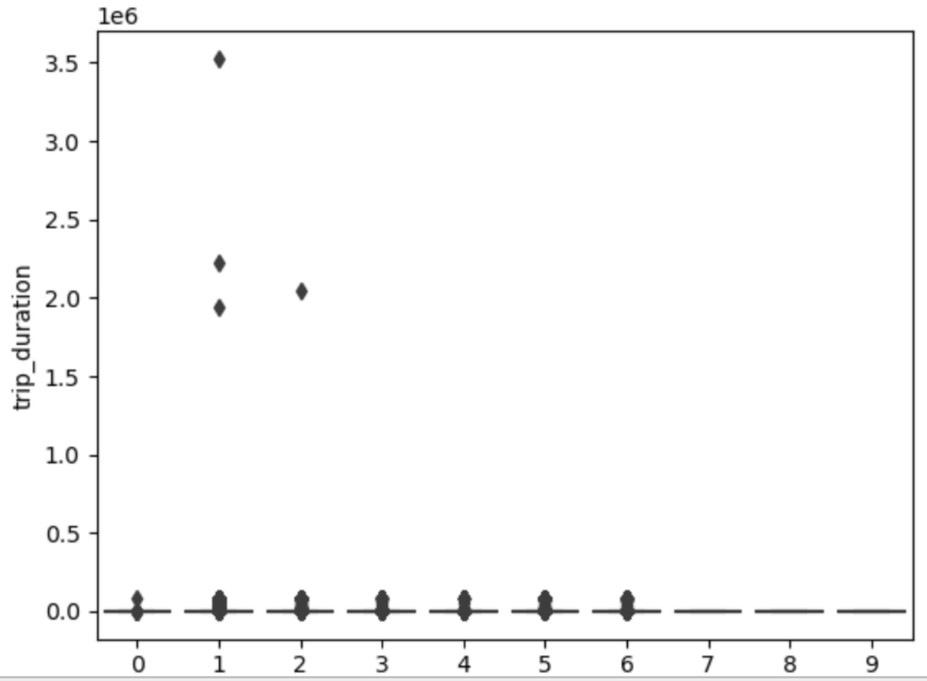


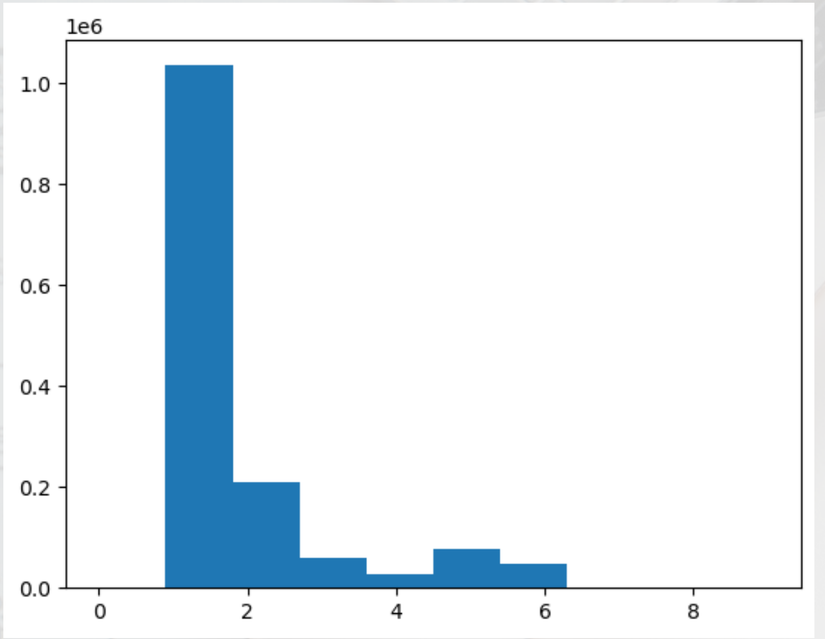
Analyzing data through heatmaps is a powerful way to visualize relationships and patterns within a dataset, especially when dealing with numerical data or correlations between variables.

DATA VISUALIZATION CONTD.

Plotting a Histogram for the passenger_count

```
In [97]: sns.boxplot(x=df['passenger_count'],y=df['trip_duration'])

Out[97]: <Axes: xlabel='passenger_count', ylabel='trip_duration'>
```



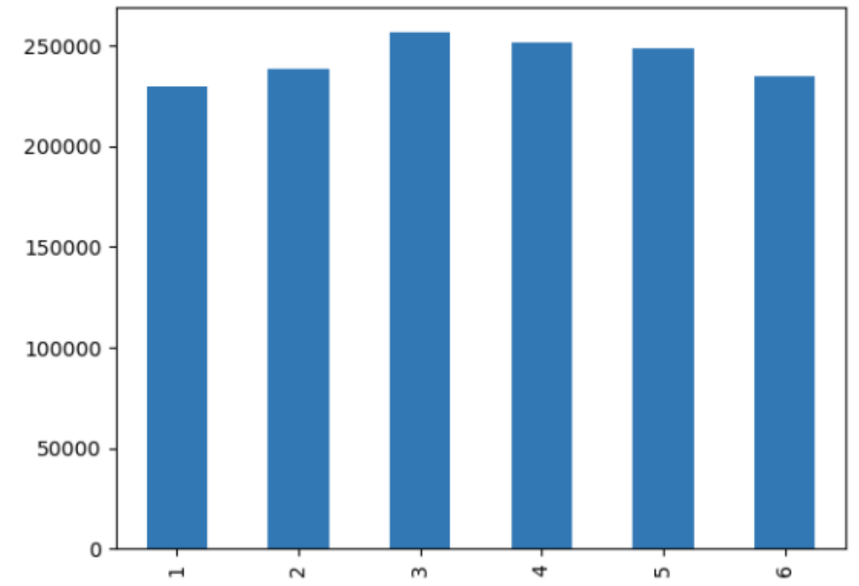


DATA VISUALIZATION CONTD.

This graph helps depict the trips per month

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

```
Trip per month
1    229707
2    238300
3    256189
4    251645
5    248487
6    234316
Name: month, dtype: int64

Out[96]: <Axes: >
```



Data Modelling

Data Scaling: Data scaling is crucial to ensure that features with different scales do not dominate the model's performance. In the context of taxi trip duration prediction, features like distance in miles, time of day, and traffic conditions can vary significantly

HERE WE ARE USING LINEAR REGRESSION MODEL FIRST

Train-Test Split: To evaluate the model's performance and avoid overfitting, it's essential to split the dataset into two parts: a training set and a testing set.

- The training set is used to train the model, allowing it to learn patterns and relationships within the data.
- The testing set is used to assess the model's generalization performance. It contains data the model has never seen before.

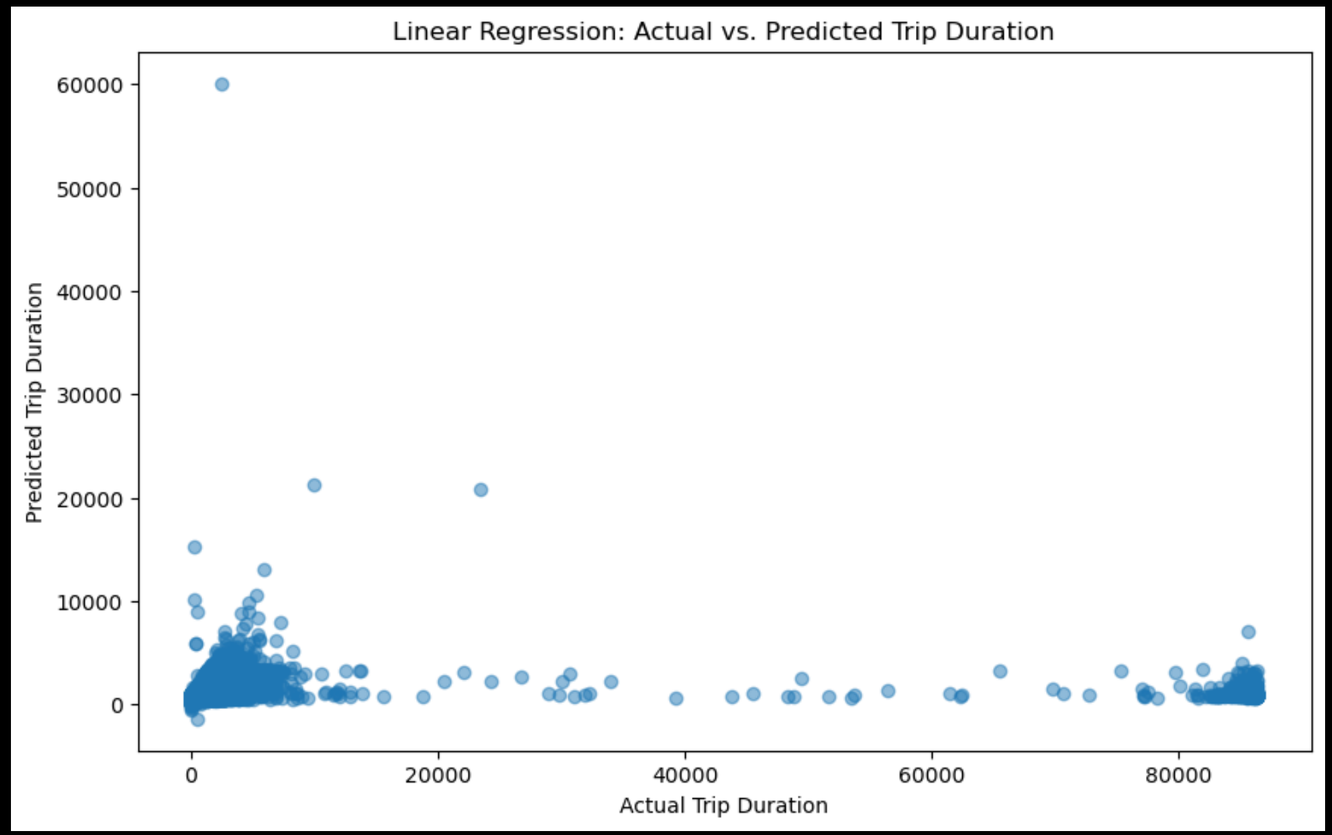
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
# Define the features (X) and target variable (y)
features = ['vendor_id_1','vendor_id_2', 'passenger_count', 'pickup_lo
X = df[features]
y = df['trip_duration']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

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

# Plot the line graph to visualize predictions vs. actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Trip Duration")
plt.ylabel("Predicted Trip Duration")
plt.title("Linear Regression: Actual vs. Predicted Trip Duration")
plt.show()
```

Here is the graph of the Actual vs the Predicted Trip Duration

RMSE (Root Mean Squared Error):
RMSE is a measure of the average error between predicted and actual values in a regression model.
It calculates the square root of the average of squared differences between predicted and observed values.

R^2 Score (Coefficient of Determination):

- R^2 score measures the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (features) in a regression model.




```
[31]: rmse = mean_squared_error(y_test,y_pred,squared=False)
      rmse

Out[31]: 3212.514657485366

[32]: from sklearn.metrics import r2_score
      r2 = r2_score(y_test,y_pred)
      r2

Out[32]: 0.025641547027708045
```

LASSO MODEL

Lasso (Least Absolute Shrinkage and Selection Operator) emphasizes feature selection by driving some feature coefficients to zero, simplifying the model.

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso

# Assuming you have already trained the Lasso model and have X_scaled

#features = ['vendor_id_1','vendor_id_2', 'passenger_count', 'pickup_l
# Make predictions on the test data for alpha=0.01
lasso_model = Lasso(alpha=0.01)  # You can adjust the alpha value

# Fit the model on the training data
lasso_model.fit(X_scaled, y)

# Make predictions on the scaled testing data
y_test_pred = lasso_model.predict(X_test)

# Print the predictions
print("Predictions for Test Data:")
print(y_test_pred)
```

```
Predictions for Test Data:
[-1713.11058743 -1963.71819456 -2457.48571824 ... -1749.83946863
 -1061.46012407   -99.16704128]
```

In [46]:

```
print(y_test_pred)
```

```
[-1713.11058743 -1963.71819456 -2457.48571824 ... -1749.83946863
 -1061.46012407   -99.16704128]
```

RIDGE MODEL

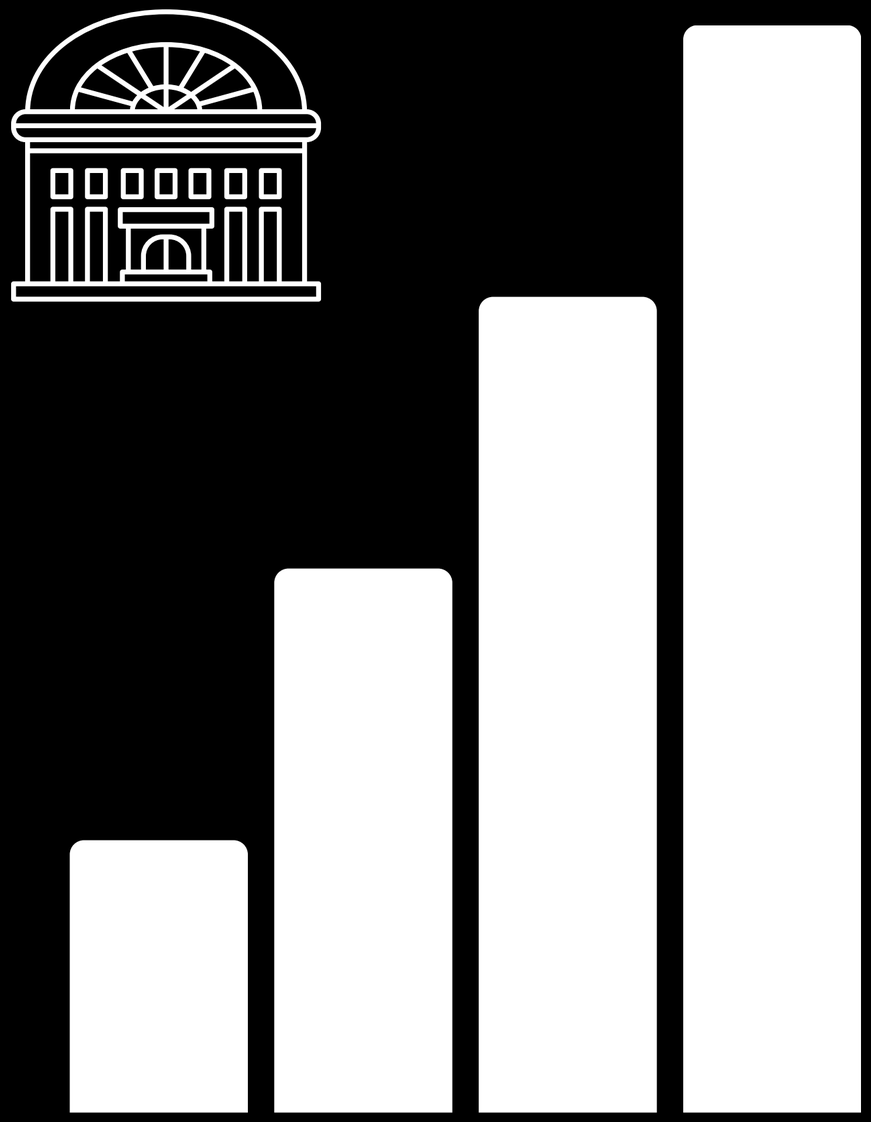
Ridge regression focuses on preventing overfitting by penalizing large coefficient magnitudes with less emphasis on feature selection.
Both methods offer a way to control model complexity and enhance predictive performance based on your dataset's characteristics and modeling goals.

```
# for 0.01 as alpha
ridge_model = Ridge(alpha=0.01) # You can adjust the alpha value
ridge_model.fit(X_scaled, y)
y_pred_ridge = ridge_model.predict(X_scaled)
# Calculate RMSE for Ridge Regression
rmse_ridge = np.sqrt(mean_squared_error(y, y_pred_ridge))
print("RMSE for rigit data with 0.01 alpha", rmse_ridge)
```

```
RMSE for rigit data with 0.01 alpha 5212.459595787802
```

```
# for 0.01 as alpha
ridge_model1= Ridge(alpha=1) # You can adjust the alpha value
ridge_model1.fit(X_scaled, y)
y_pred_ridge1 = ridge_model1.predict(X_scaled)
# Calculate RMSE for Ridge Regression
rmse_ridge1 = np.sqrt(mean_squared_error(y, y_pred_ridge1))
print("RMSE for rigit data with 1 alpha", rmse_ridge1)
```

```
RMSE for rigit data with 1 alpha 5212.459595787815
```



OBSEVERATION:

Here we see that the lasso and linear regression models and ridge models are 3212 and 5212 respecitively. Hence we can conclude that the Linear Regression model works better for this dataset