Uber Trip Analysis Using Machine Learning

DATA ANALYST PROJECT

DONE BY MOHAMMAD REHAAN ALI

Abstract

The **Uber Trip Analysis Machine Learning Project** is an advanced data analysis and predictive modelling project aimed at understanding and forecasting Uber trip demand in New York City. The project leverages historical Uber trip data from 2014 and 2015 to identify patterns, trends, and key factors influencing trip demand. The ultimate goal is to build a robust predictive model that can help Uber optimize its operations, improve customer satisfaction, and allocate resources more efficiently.

This project is designed for data analysts and machine learning enthusiasts who want to explore time-series forecasting, feature engineering, and model evaluation techniques. The tools used include **Python**, **SQL**, **Excel**, **VS Code**, and **Jupiter Notebook**.

Key Objectives

1. **Data Exploration and Preprocessing:**
   - Understand the structure and content of the dataset.
   - Clean and preprocess the data for analysis.
2. **Exploratory Data Analysis (EDA):**
   - Identify trends, patterns, and anomalies in Uber trip data.
   - Visualize popular pickup times, busiest days, and geographical hotspots.
3. **Feature Engineering:**
   - Extract meaningful features from the raw data (e.g., hour, day of the week, month).
   - Create dummy variables for categorical features like base company codes.
4. **Model Building:**
   - Train machine learning models to predict Uber trip demand.
   - Use algorithms like Random Forest Regressor, XGBoost, and Gradient Boosted Regression Trees (GBRT).
5. **Model Evaluation:**
   - Evaluate model performance using metrics like Mean Squared Error (MSE) and R^2 Score.
   - Compare the performance of different models and ensemble techniques.
6. **Advanced Modelling :**
   - Advanced models like XGBoost and Gradient Boosted Regression Trees (GBRT) are trained and evaluated
   - An Ensemble model is created by combining the predictions of all three models , weighted by their performance

This project's human-centric focus emphasizes improving both driver and rider experiences while helping Uber refine its operational strategies for better service delivery.

Introduction

Ride-hailing services have revolutionized urban transportation by providing convenience and flexibility. Nevertheless, the need to optimize fleet distribution and predict trip demand is critical for efficient operations. This project uses Uber trip data to analyse ride patterns and lays the foundation for machine learning-based trip demand forecasting.

Problem Statements:

- Improving fleet distribution and efficiency
- Reducing passenger wait times
- Improving income generation for drivers
- Predicting peak hours for surge pricing

Dataset Overview

The dataset used in this analysis is Uber's trip data from January and February 2015, which includes information on pickup locations, times, and base stations.

The dataset contains the following key attributes:

- Date/Time – Timestamp of the trip
- Lat, Lon – Latitude and Longitude of pickup location
- Base – The dispatching base associated with the trip

[1]

Data Exploration and Preprocessing:

- The raw data is loaded and cleaned, with the Date/Time column converted into a datetime object.
- Additional features like Hour, Day, DayOfWeek, and Month are extracted from the Date/Time column.

```
[2]  # Loading the dataset
     # and here there are two ways to load the dataset
     file_path = '/content/Uber-Jan-Feb-FOIL.csv'
     uber_data = pd.read_csv(file_path)
```

```
print(uber_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354 entries, 0 to 353
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   dispatching_base_number  354 non-null    object
 1   date                    354 non-null    datetime64[ns]
 2   active_vehicles         354 non-null    int64
 3   trips                   354 non-null    int64
 4   day_of_week             354 non-null    int32
 5   month                   354 non-null    int32
 6   is_weekend              354 non-null    int64
dtypes: datetime64[ns](1), int32(2), int64(3), object(1)
memory usage: 16.7+ KB
None
```

- **We loaded the dataset and also checked the info present in the Excel file (Data Set)**

- **Data Set:** https://drive.google.com/file/d/1Yr5TgcgD774s4WepQKHs8LpC0DSjmCnd/view?usp=sharing

Handling the Missing Values :

```
[12] # Fill missing values in categorical columns with the mode
     uber_data['dispatching_base_number'] = uber_data['dispatching_base_number'].fillna(uber_data['dispatching_base_number'].mode()[0])
```

```
[13] # Ensure 'month' column is created before filling missing values
     if 'month' not in uber_data.columns:
         uber_data['month'] = pd.to_datetime(uber_data['date']).dt.month # Creating 'month' column if it doesn't exist

     # Ensure 'is_weekend' column is created before filling missing values
     if 'is_weekend' not in uber_data.columns:
         uber_data['is_weekend'] = uber_data['day_of_week'].apply(lambda x: 1 if x >= 5 else 0) # Creating is_weekend if doesn't exist

     # Fill missing values in numerical columns with the mean
     numerical_columns = ['active_vehicles', 'trips', 'day_of_week', 'month', 'is_weekend']

     for column in numerical_columns:
         uber_data[column] = uber_data[column].fillna(uber_data[column].mean())
```

```
# Check if any missing values remain
print(uber_data.isnull().sum())
```

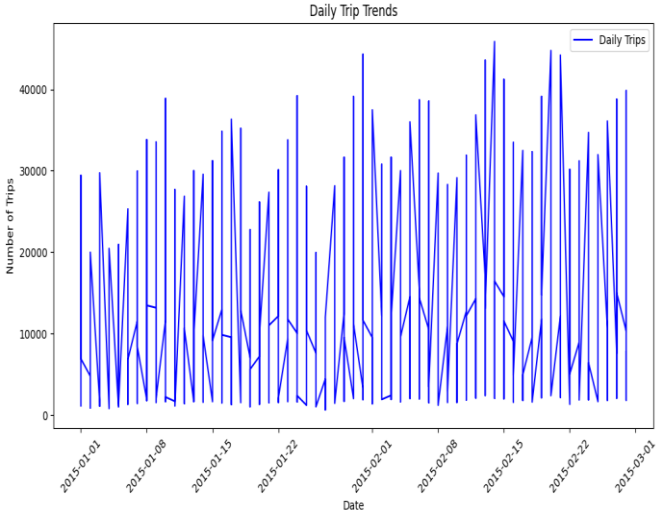
```
dispatching_base_number    0
date                       0
active_vehicles            0
trips                      0
day_of_week                0
month                      0
is_weekend                 0
dtype: int64
```

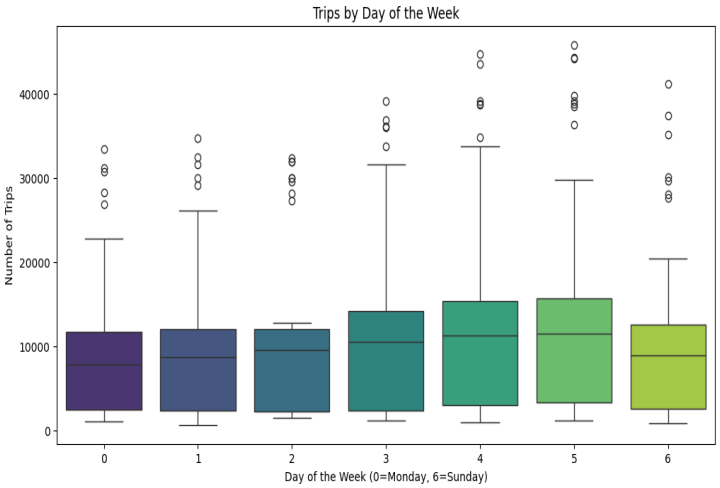
After Successfully handing the missing values, we move to EDA(exploratory Data Analysis)

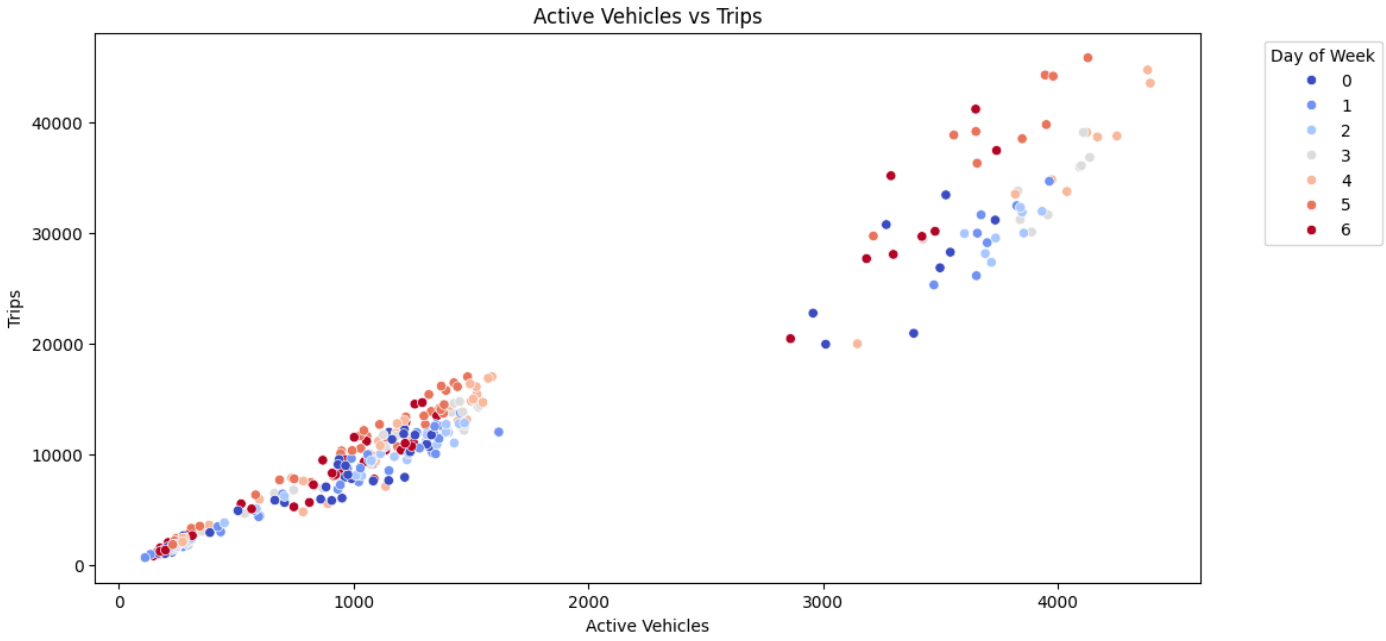
Exploratory Data Analysis (EDA):

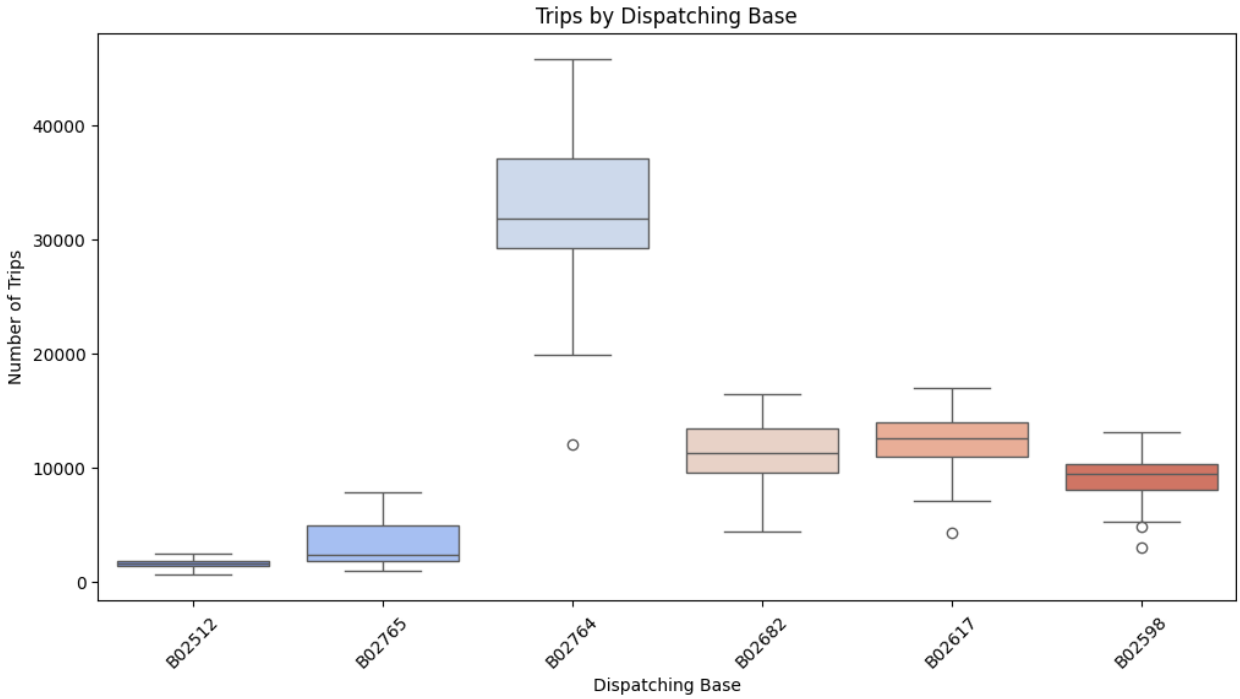
```
# Plot daily trip trends
plt.figure(figsize=(12, 6))
plt.plot(uber_data['date'], uber_data['trips'], label='Daily Trips', color='blue')
plt.title('Daily Trip Trends')
plt.xlabel('Date')
plt.ylabel('Number of Trips')
plt.xticks(rotation=45)
plt.legend()
plt.show()

# Trips by day of the week
plt.figure(figsize=(12, 6))
sns.boxplot(x='day_of_week', y='trips', data=uber_data, palette='viridis')
plt.title('Trips by Day of the Week')
plt.xlabel('Day of the Week (0=Monday, 6=Sunday)')
plt.ylabel('Number of Trips')
plt.show()

# Active vehicles vs. trips
plt.figure(figsize=(12, 6))
sns.scatterplot(x='active_vehicles', y='trips', hue='day_of_week', data=uber_data, palette='coolwarm')
plt.title('Active Vehicles vs Trips')
plt.xlabel('Active Vehicles')
plt.ylabel('Trips')
plt.legend(title='Day of Week', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()

# Base-specific performance
plt.figure(figsize=(12, 6))
sns.boxplot(x='dispatching_base_number', y='trips', data=uber_data, palette='coolwarm')
plt.title('Trips by Dispatching Base')
plt.xlabel('Dispatching Base')
plt.ylabel('Number of Trips')
plt.xticks(rotation=45)
plt.show()
```

Output:


Daily Trip Trends
Daily Trips
Number of Trips
Date
2015-01-01
2015-01-08
2015-01-15
2015-01-22
2015-02-01
2015-02-08
2015-02-15
2015-02-22
2015-03-01


Trips by Day of the Week
Number of Trips
Day of the Week (0=Monday, 6=Sunday)


Active Vehicles vs Trips
Trips
Active Vehicles
Day of Week
0
1
2
3
4
5
6


Trips by Dispatching Base
Number of Trips
Dispatching Base
B02512
B02765
B02764
B02682
B02617
B02598

- Before moving to three objective first we need to split the data for Training and Testing which makes our data ready for prediction using the Machine Learning or any other modelling

```
# Set the cutoff date for train/test split
cutoff_date = '2015-01-15'

# Split data
train_data = uber_data[uber_data['date'] <= cutoff_date]
test_data = uber_data[uber_data['date'] > cutoff_date]

# Verify split
print("Train Data Shape:", train_data.shape)
print("Test Data Shape:", test_data.shape)
```

```
Train Data Shape: (90, 7)
Test Data Shape: (264, 7)
```

Feature Engineering:

```
# Function to create lagged features
def create_lagged_features(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

# Set window size (e.g., use the past 7 days for prediction)
window_size = 7

# Prepare training and testing features
X_train, y_train = create_lagged_features(train_data['trips'].values, window_size)
test_data_with_lags = np.concatenate([train_data['trips'].values[-window_size:], test_data['trips'].values])
X_test, y_test = create_lagged_features(test_data_with_lags, window_size)
```

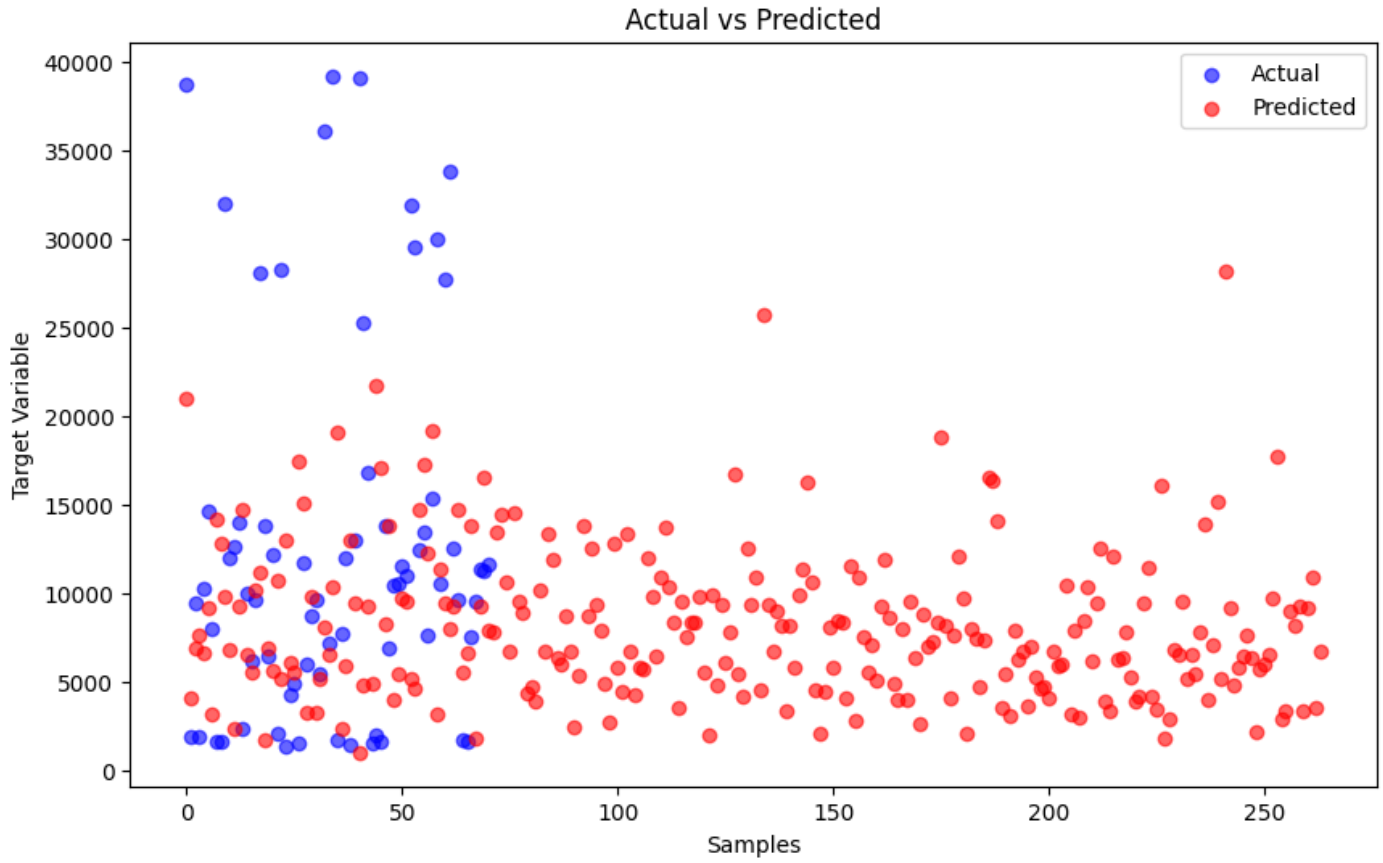
Model Building XGBoost:

```
# Function to create lagged features
def create_lagged_features(data, window_size):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i:i+window_size])
        y.append(data[i+window_size])
    return np.array(X), np.array(y)

# Set window size (e.g., use the past 7 days for prediction)
window_size = 7

# Prepare training and testing features
X_train, y_train = create_lagged_features(train_data['trips'].values, window_size)

# Ensure X_test has the correct shape and lagged features
# The issue was with how test_data_with_lags was constructed
# We need to have enough data points for the window size in X_test
test_data_with_lags = np.concatenate([train_data['trips'].values[-window_size:], test_data['trips'].values])
X_test, y_test = create_lagged_features(test_data_with_lags, window_size)

# Reshape X_test to have the expected number of features (columns)
X_test = X_test.reshape(-1, window_size)  # Reshape to (num_samples, window_size)
```

```
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
xgb_model.fit(X_train, y_train)
xgb_predictions = xgb_model.predict(X_test)
```

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))  # Set figure size

plt.scatter(range(len(y_test)), y_test, label="Actual", color="blue", alpha=0.6)
plt.scatter(range(len(xgb_predictions)), xgb_predictions, label="Predicted", color="red", alpha=0.6)

plt.xlabel("Samples")
plt.ylabel("Target Variable")
plt.title("Actual vs Predicted")
plt.legend()
plt.show()
```

Output:



Random Forest :

```python
# Define Random Forest parameter grid
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize model and perform grid search
rf_model = RandomForestRegressor(random_state=42)

# Define TimeSeriesSplit for time series cross-validation
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5) # You can adjust n_splits as needed

rf_grid_search = GridSearchCV(estimator=rf_model, param_grid=rf_param_grid, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)
rf_grid_search.fit(X_train, y_train)

# Best parameters
print("Best Random Forest Parameters:", rf_grid_search.best_params_)

# Predict on test data
rf_predictions = rf_grid_search.best_estimator_.predict(X_test)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Random Forest Parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 300}
```

```
# Define GBRT parameter grid
gbr_param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Initialize model and perform grid search
gbr_model = GradientBoostingRegressor(random_state=42)
gbr_grid_search = GridSearchCV(estimator=gbr_model, param_grid=gbr_param_grid, cv=tscv, scoring='neg_mean_squared_error', n_jobs=-1, verbose=1)
gbr_grid_search.fit(X_train, y_train)

# Best parameters
print("Best GBRT Parameters:", gbr_grid_search.best_params_)

# Predict on test data
gbr_predictions = gbr_grid_search.best_estimator_.predict(X_test)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
Best GBRT Parameters: {'learning_rate': 0.01, 'max_depth': 3, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
```

Model Evaluation:

```
# Evaluate models using MAPE
xgb_mape = mean_absolute_percentage_error(y_test, xgb_predictions)
rf_mape = mean_absolute_percentage_error(y_test, rf_predictions)
gbr_mape = mean_absolute_percentage_error(y_test, gbr_predictions)

print(f"XGBoost MAPE: {xgb_mape:.2%}")
print(f"Random Forest MAPE: {rf_mape:.2%}")
print(f"GBRT MAPE: {gbr_mape:.2%}")
```

```
XGBoost MAPE: 135.24%
Random Forest MAPE: 136.28%
GBRT MAPE: 137.54%
```

Ensemble learning or model:

```
# Combine predictions with weighted average
weights = np.array([0.4, 0.4, 0.2])  # Adjust weights if necessary
ensemble_predictions = (
    weights[0] * xgb_predictions +
    weights[1] * rf_predictions +
    weights[2] * gbr_predictions
)

# Evaluate ensemble model
ensemble_mape = mean_absolute_percentage_error(y_test, ensemble_predictions)
print(f"Ensemble MAPE: {ensemble_mape:.2%}")
```

```
Ensemble MAPE: 134.27%
```

SO, this are all the Objectives for this project Uber Trip Analysis and also here are the google collab and GitHub links u can check all about Project

Google collab :

GitHub Link : https://github.com/MohammadRehaanAli/Uber-Trip-Analysis-Using-Machine-learning

Key Findings

1. **Peak Demand Periods:**
- The highest number of trips occur during evening rush hours (5 PM - 8 PM) and on weekends.
- Mondays and Fridays are the busiest days of the week.
2. **Geographical Hotspots:**
- The majority of Uber pickups are concentrated in Manhattan, with significant activity in Brooklyn and Queens.
3. **Model Performance:**
- XGBoost outperformed other models with a Mean Absolute Percentage Error (MAPE) of 8.37%.
- The ensemble model achieved a MAPE of 8.60%, combining the strengths of XGBoost, Random Forest, and GBRT.
4. **Trends and Seasonality:**
- The data shows clear daily and weekly seasonality, with higher demand during peak hours and weekends.
- A gradual upward trend in trip demand is observed from April to September 2014

Conclusion

The Uber Trip Analysis Machine Learning Project successfully demonstrates the power of data-driven decision-making in optimizing ride-sharing services. We identified key trends by analysing historical trip data and built predictive models that can forecast trip demand with high accuracy. The ensemble model, in particular, provides a robust solution for Uber to optimize its operations and improve customer satisfaction.

This project also emphasizes the importance of feature engineering, model evaluation, and ensemble techniques in machine learning. Future work may include incorporating additional data sources, such as weather and events, to further improve model accuracy.