UNIFIED MENTOR
YOUR SKILL, SUCCESS & JOURNEY

Project Title	**Uber Trip Analysis**
language	Machine learning, python, SQL, Excel
Tools	VS code, Jupyter notebook
Domain	Data Analyst
Project Difficulties level	Advance

Dataset : Dataset is available in the given link. You can download it at your convenience.

Click here to download data set

About Dataset

Uber TLC FOIL Response

This directory contains data on over 4.5 million Uber pickups in New York City from April to September 2014, and 14.3 million more Uber pickups from January to June 2015. Trip-level data on 10 other for-hire vehicle (FHV) companies, as well as aggregated data for 329 FHV companies, is also included. All the files are as they were received on August 3, Sept. 15 and Sept. 22, 2015.

FiveThirtyEight obtained the data from the NYC Taxi & Limousine Commission (TLC) by submitting a Freedom of Information Law request on July 20, 2015. The TLC has sent us the data in batches as it continues to review trip data Uber and other HFV companies have submitted to it. The TLC's correspondence with FiveThirtyEight is included in the files `TLC_letter.pdf`, `TLC_letter2.pdf` and `TLC_letter3.pdf`. TLC records requests can be made here.

This data was used for four FiveThirtyEight stories: Uber Is Serving New York’s Outer Boroughs More Than Taxis Are, Public Transit Should Be Uber’s New Best Friend, Uber Is Taking Millions Of Manhattan Rides Away From Taxis, and Is Uber Making NYC Rush-Hour Traffic Worse?.

The Data

The dataset contains, roughly, four groups of files:

- Uber trip data from 2014 (April - September), separated by month, with detailed location information
- Uber trip data from 2015 (January - June), with less fine-grained location information
- non-Uber FHV (For-Hire Vehicle) trips. The trip information varies by company, but can include day of trip, time of trip, pickup location, driver's for-hire license number, and vehicle's for-hire license number.
- aggregate ride and vehicle statistics for all FHV companies (and, occasionally, for taxi companies)

Uber trip data from 2014

There are six files of raw data on Uber pickups in New York City from April to September 2014. The files are separated by month and each has the following columns:

- `Date/Time` : The date and time of the Uber pickup
- `Lat` : The latitude of the Uber pickup
- `Lon` : The longitude of the Uber pickup
- `Base` : The TLC base company code affiliated with the Uber pickup

These files are named:

- `uber-raw-data-apr14.csv`
- `uber-raw-data-aug14.csv`
- `uber-raw-data-jul14.csv`
- `uber-raw-data-jun14.csv`
- `uber-raw-data-may14.csv`
- `uber-raw-data-sep14.csv`

Uber trip data from 2015

Also included is the file `uber-raw-data-janjune-15.csv` This file has the following columns:

- `Dispatching_base_num` : The TLC base company code of the base that dispatched the Uber
- `Pickup_date` : The date and time of the Uber pickup
- `Affiliated_base_num` : The TLC base company code affiliated with the Uber pickup
- `locationID` : The pickup location ID affiliated with the Uber pickup

The `Base` codes are for the following Uber bases:

B02512 : Unter

B02598 : Hinter

B02617 : Weiter

B02682 : Schmecken

B02764 : Danach-NY

B02765 : Grun

B02835 : Dreist

B02836 : Drinnen

For coarse-grained location information from these pickups, the file `taxi-zone-lookup.csv` shows the taxi `Zone` (essentially, neighborhood) and `Borough` for each `locationID`.

Non-Uber FLV trips

The dataset also contains 10 files of raw data on pickups from 10 for-hire vehicle (FHV) companies. The trip information varies by company, but can include day of trip, time of trip, pickup location, driver's for-hire license number, and vehicle's for-hire license number.

These files are named:

- `American_B01362.csv`
- `Diplo_B01196.csv`
- `Highclass_B01717.csv`
- `Skyline_B00111.csv`
- `Carmel_B00256.csv`
- `Federal_02216.csv`
- `Lyft_B02510.csv`
- `Dial7_B00887.csv`

- `Firstclass_B01536.csv`
- `Prestige_B01338.csv`

Aggregate Statistics

There is also a file `other-FHV-data-jan-aug-2015.csv` containing daily pickup data for 329 FHV companies from January 2015 through August 2015.

The file `Uber-Jan-Feb-FOIL.csv` contains aggregated daily Uber trip statistics in January and February 2015.

Uber Trip Analysis Machine Learning Project

Project Overview

The goal of this project is to analyze Uber trip data to identify patterns and build a predictive model for trip demand. The analysis will cover various aspects such as popular pickup times, busiest days, and fare prediction.

Dataset

The dataset used for this project is typically Uber's trip data, which includes details such as:

- **Date/Time:** When the trip started.
- **Lat:** Latitude of the pickup.
- **Lon:** Longitude of the pickup.
- **Base:** TLC base company code affiliated with the Uber pickup.

Uber provides various datasets on platforms like Kaggle, which you can download for analysis.

Steps and Implementation

1. **Data Preprocessing**
2. **Exploratory Data Analysis (EDA)**
3. **Feature Engineering**
4. **Model Building**
5. **Model Evaluation**

6. **Visualization**

Implementation Code

Here is a sample implementation in Python:

```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load the dataset
data = pd.read_csv('uber-raw-data-apr14.csv')

# Display basic info about the dataset
print(data.info())

# Data Preprocessing
# Convert Date/Time to datetime object
data['Date/Time'] = pd.to_datetime(data['Date/Time'])

# Extracting useful information from Date/Time
data['Hour'] = data['Date/Time'].dt.hour
data['Day'] = data['Date/Time'].dt.day
data['DayOfWeek'] = data['Date/Time'].dt.dayofweek
data['Month'] = data['Date/Time'].dt.month

# Exploratory Data Analysis
# Plotting the number of trips per hour
plt.figure(figsize=(10,6))
sns.countplot(data['Hour'])
plt.title('Trips per Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
plt.show()

# Plotting the number of trips per day of the week
plt.figure(figsize=(10,6))
```

```
sns.countplot(data['DayOfWeek'])
plt.title('Trips per Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Trips')
plt.show()

# Feature Engineering
# Create dummy variables for categorical features
data = pd.get_dummies(data, columns=['Base'], drop_first=True)

# Define features and target variable
X = data[['Hour', 'Day', 'DayOfWeek', 'Month', 'Lat', 'Lon']]
y = data['Trips'] # Assume we have a 'Trips' column indicating the number of trips

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Model Building
# Train a Random Forest Regressor
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train, y_train)

# Predict on the test set
y_pred = rfr.predict(X_test)

# Model Evaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R^2 Score:", r2_score(y_test, y_pred))

# Visualization of Predictions
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.xlabel('Actual Trips')
plt.ylabel('Predicted Trips')
plt.title('Actual vs Predicted Trips')
plt.show()
```

Explanation of Code

1. **Data Preprocessing:**
    - Load the dataset and convert the 'Date/Time' column to a datetime object.

    - Extract useful information like hour, day, day of the week, and month from the 'Date/Time' column.
2. **Exploratory Data Analysis (EDA):**
    - Visualize the number of trips per hour and per day of the week using count plots.
3. **Feature Engineering:**
    - Create dummy variables for the categorical feature 'Base'.
    - Define the feature set `X` and the target variable `y`.
4. **Model Building:**
    - Split the data into training and testing sets.
    - Train a Random Forest Regressor on the training data.
    - Predict the number of trips on the test data.
5. **Model Evaluation:**
    - Evaluate the model using Mean Squared Error (MSE) and R² score.
    - Visualize the actual vs predicted trips to assess model performance.

Additional Resources

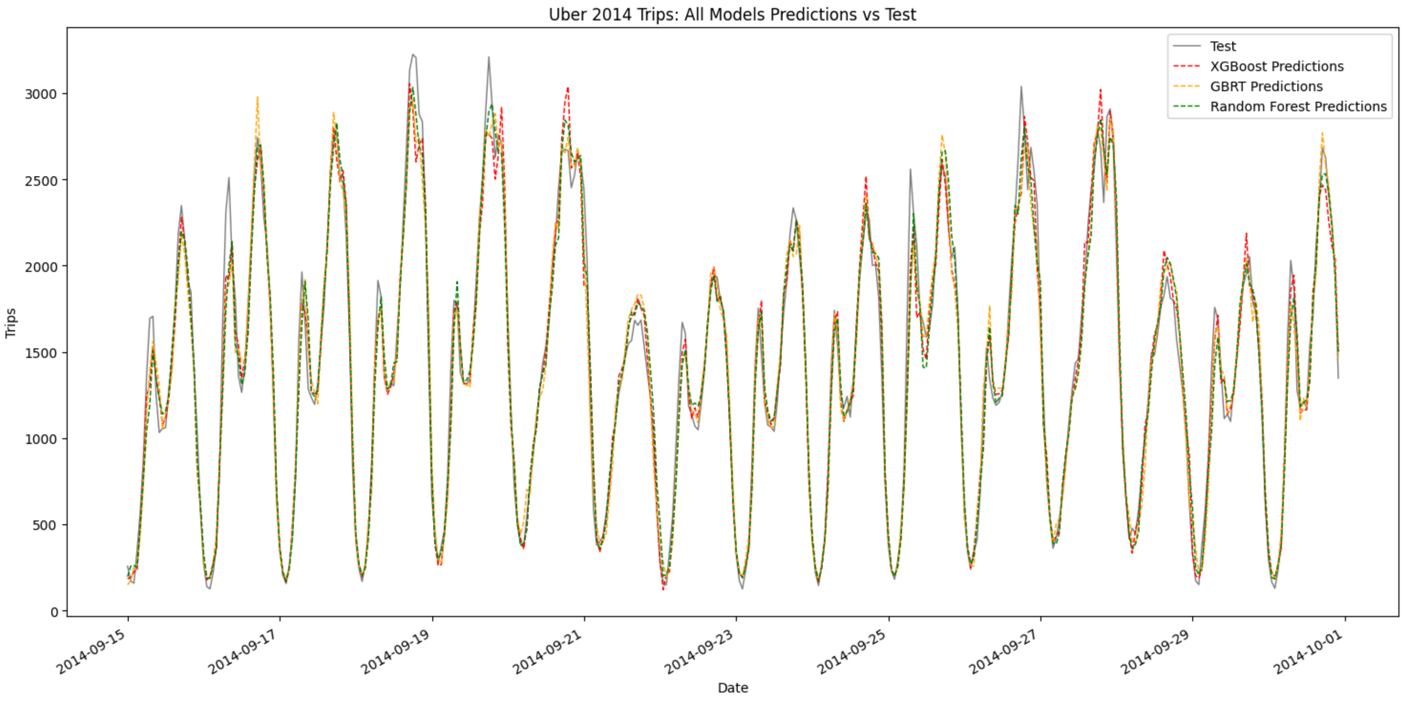
- Uber Trip Data on Kaggle
- Random Forest Regressor Documentation
- Handling DateTime in Pandas

This implementation provides a framework for analyzing and predicting Uber trips. You can extend it by adding more features, trying different models, and improving feature engineering techniques.

Sample code

Uber Trips Forecasting with XGBoost, Random Forests and

Gradient Boosted Tree Regressors + Ensemble¶



The proliferation of ride-hailing services like Uber has led to an immense accumulation of trip data, providing a rich resource for predictive analytics. Accurate forecasting of Uber trips is critical for optimizing operations, improving customer satisfaction, and streamlining resource allocation.

This notebook aims to delve into the predictive power of XGBoost, Random Forest and Gradient Boosted Tree Regressor in forecasting Uber trips using historical data from 2014.

While there are currently other state-of-the-art strategies to predict univariate time series, this Notebook intends to provide a machine learning oriented approach to time series forecasting as an alternative tool.

Objectives

The primary objectives of this notebook are:

- Data Exploration and Preprocessing: Understand and prepare the 2014 Uber trip data for model training.
- Model Training: Train three distinct types of models—XGBoost, GBTR and Random Forests networks—using the 2014 data.
- Model Evaluation: Assess the performance of each model using Mean Average Percentage Error as the main evaluation metric.
- Ensemble Techniques: Explore ensemble methods to combine the strengths of the individual models and enhance forecasting accuracy.

- Comparative Analysis: Provide a comparative analysis of the forecasting capabilities of the models and the ensemble approach.

1. Importing the necessary libraries + useful functions

The first step is to import all necessary libraries and include useful functions to make the code below more readable.

In [1]:

```
import warnings
warnings.filterwarnings("ignore")

import os
import numpy as np
import pandas as pd
import seaborn as sns
import xgboost as xgb
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from xgboost import plot_importance, plot_tree
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV,
TimeSeriesSplit
```

In [2]:

```
def PlotDecomposition(result):
    plt.figure(figsize=(22,18))
    plt.subplot(4,1,1)
    plt.plot(result.observed,label='Observed',lw=1)
    plt.legend(loc='upper left')
    plt.subplot(4,1,2)
    plt.plot(result.trend,label='Trend',lw=1)
    plt.legend(loc='upper left')
    plt.subplot(4, 1, 3)
    plt.plot(result.seasonal, label='Seasonality',lw=1)
    plt.legend(loc='upper left')
    plt.subplot(4, 1, 4)
    plt.plot(result.resid, label='Residuals',lw=1)
    plt.legend(loc='upper left')
    plt.show()
def CalculateError(pred,sales):
    percentual_errors = []
    for A_i, B_i in zip(sales, pred):
        percentual_error = abs((A_i - B_i) / B_i)
        percentual_errors.append(percentual_error)
    return sum(percentual_errors) / len(percentual_errors)
def PlotPredictions(plots,title):
    plt.figure(figsize=(18, 8))
    for plot in plots:
        plt.plot(plot[0], plot[1], label=plot[2], linestyle=plot[3],
color=plot[4],lw=1)
    plt.xlabel('Date')
    plt.ylabel("Trips")
    plt.title(title)
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.show()
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)
```

2. Reading the Uber Trips Dataset and preparing the data

As you will see, one of the key aspects of this (particluarly important) step is resampling it on an hourly basis. Originally, the data isn't time series prediction ready. Once we finish preparing the data, we will be able to begin training models.

In [3]:

```
files = []

# Get all uber rides raw data
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        #print(os.path.join(dirname, filename))
        files.append(os.path.join(dirname, filename)) if "raw" in filename else None

# Keep the jun - sep 2014 data on a separate list
files = files[:-1]
```

In [4]:

```
# Read and concatenate all CSV files
dataframes = [pd.read_csv(file) for file in files]
```

```
uber2014 = pd.concat(dataframes, ignore_index=True)
# Now make sure the date column is set to datetime, sorted and with an adequate name
uber2014['Date/Time'] = pd.to_datetime(uber2014['Date/Time'], format='%m/%d/%Y %H:%M:%S')
uber2014 = uber2014.sort_values(by='Date/Time')
uber2014 = uber2014.rename(columns={'Date/Time':'Date'})
uber2014.set_index('Date',inplace=True)
```

In [5]:

```
# Group by hour and count occurrences of 'Base'
hourly_counts = uber2014['Base'].resample('h').count()
# Convert the series to a dataframe
uber2014 = hourly_counts.reset_index()
# Rename columns for clarity
uber2014.columns = ['Date', 'Count']
uber2014.set_index('Date',inplace=True)
```

In [6]:

```
uber2014.head()
```

Out[6]:

	Count
Date	
2014-04-01 00:00:00	138
2014-04-01 01:00:00	66

2014-04-01 02:00:00	53
2014-04-01 03:00:00	93
2014-04-01 04:00:00	166

3. Choosing the optimal train / test sets

In order to choose the correct train / test sets, we need to first visualize the series, then do a seasonal decompose if the trend can inform us of a suggested approach to that split

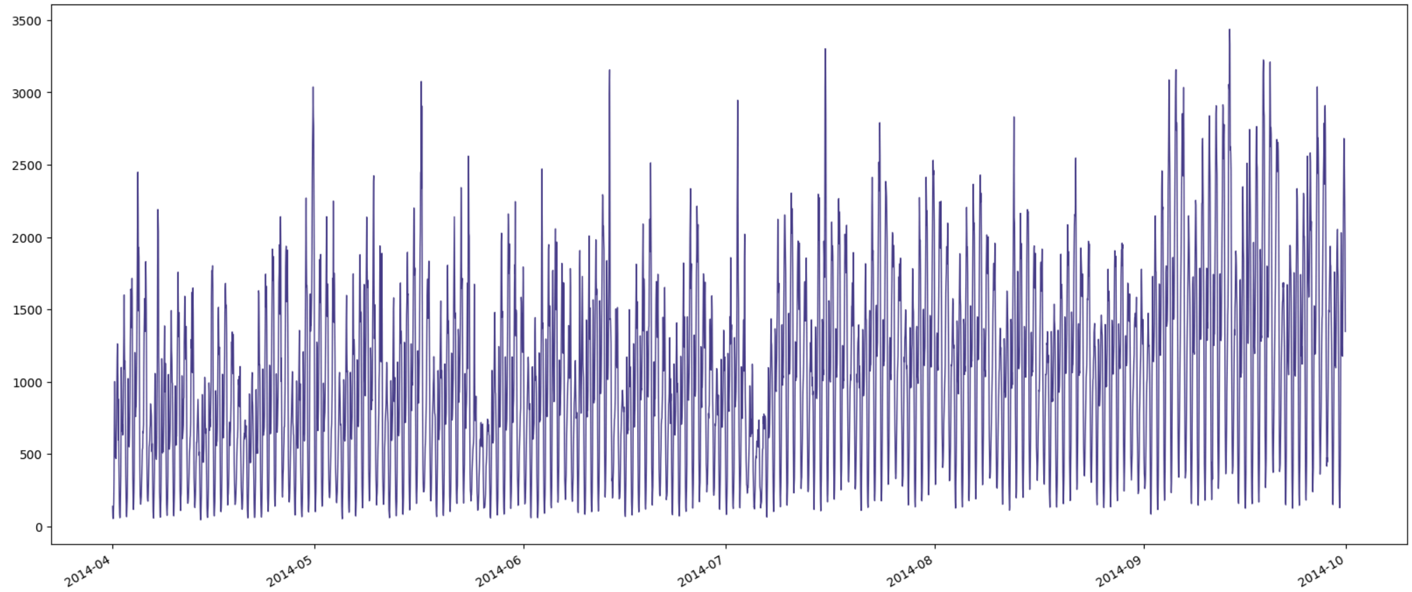
In [7]:

```
print(uber2014.index.min())
print(uber2014.index.max())
```

```
2014-04-01 00:00:00
2014-09-30 22:00:00
```

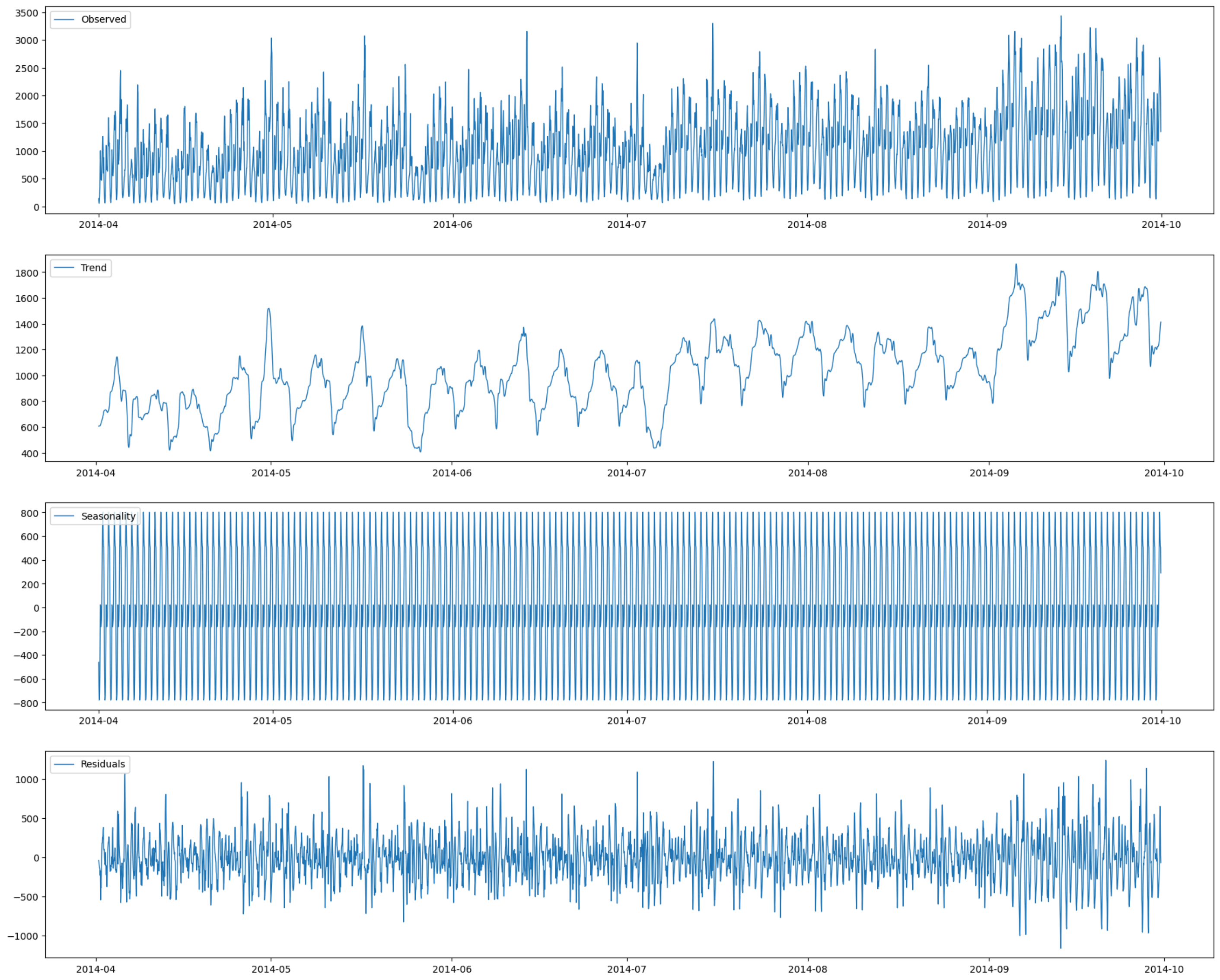
In [8]:

```
# Let's plot the series
plt.figure(figsize=(20, 8))
plt.plot(uber2014['Count'],linewidth = 1, color='darkslateblue')
plt.xticks(rotation=30,ha='right')
plt.show()
```



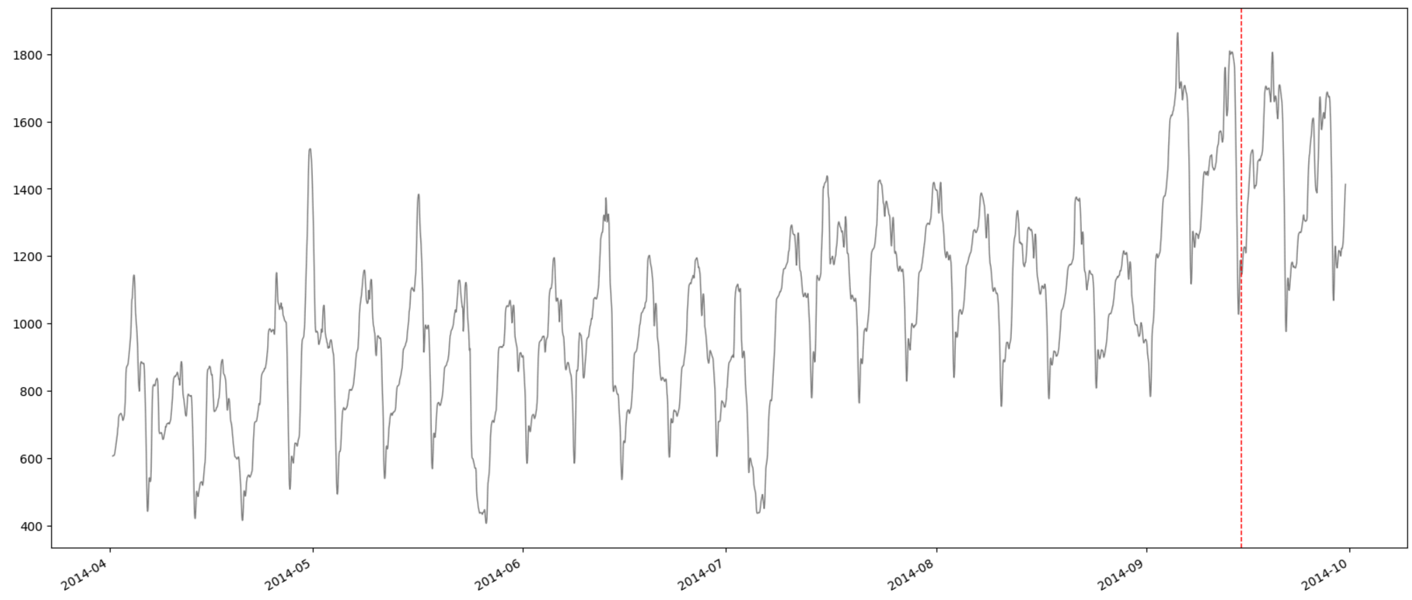
In [9]:

```
result=seasonal_decompose(uber2014['Count'],model='add', period=24*1)
PlotDecomposition(result)
```



In [10]:

```
cutoff_date = '2014-09-15 00:00:00'
plt.figure(figsize=(20, 8))
plt.plot(result.trend,linewidth = 1, color='gray')
plt.axvline(x=pd.Timestamp(cutoff_date), color='red', linestyle='--', linewidth=1)
plt.xticks(rotation=30,ha='right')
plt.show()
```



As seen above, the trend stays relatively stable until around September 2014, and then increases to 4 more peaks. Leaving up to the first 2 peaks as train data and the remaining 2 as test would be sufficient. This is particularly important, because if we did the usual 80/20 split, we would likely encounter errors due to the said trend increase.

In [11]:

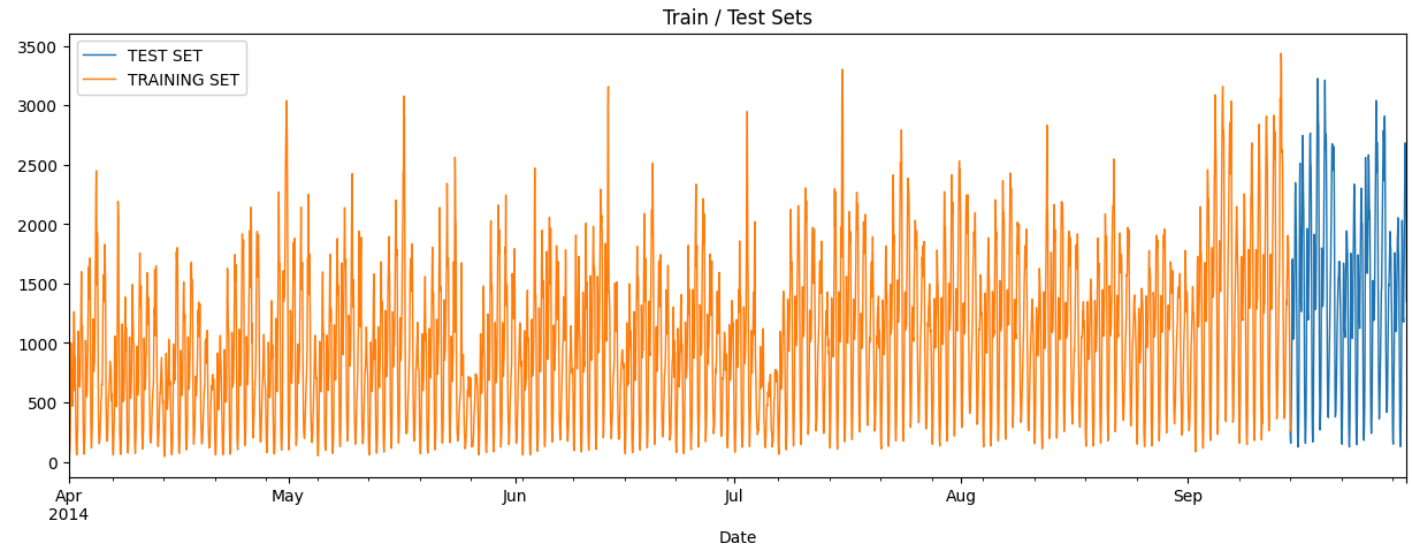
```
uber2014_train = uber2014.loc[:cutoff_date]
uber2014_test = uber2014.loc[cutoff_date:]
```

In [12]:

```
uber2014_test.rename(columns={'Count':'TEST
SET'}).join(uber2014_train.rename(columns={'Count':'TRAINING SET'}),
how='outer').plot(figsize=(15,5),title='Train / Test Sets', style='-',lw=1)
```

Out[12]:

```
<Axes: title={'center': 'Train / Test Sets'}, xlabel='Date'>
```



In [13]:

```
# Set the window size
window_size = 24

# Split data into training and test sets
X_train, y_train = create_lagged_features(uber2014_train['Count'].values,
window_size)
```

In [14]:

```
test_data = np.concatenate([uber2014_train['Count'].values[-window_size:],
uber2014_test['Count'].values])
X_test, y_test = create_lagged_features(test_data, window_size)
```

In [15]:

```
seed = 12345
```

4. XGBoost Model model

XGBoost is one of the strongest ML algorithms available. However, it is usually prone to overfitting. We avoid it by doing Cross Validation and fine tuning the training process.

In [16]:

```
tscv = TimeSeriesSplit(n_splits=5)
```

In [17]:

```
xgb_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 6, 9],
    'learning_rate': [0.01, 0.1, 0.3],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0]
}
```

In [18]:

```
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=seed)
```

In [19]:

```
xgb_grid_search = GridSearchCV(estimator=xgb_model, param_grid=xgb_param_grid,
cv=tscv, scoring='neg_mean_absolute_percentage_error', n_jobs=-1,verbose=1)
xgb_grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
```

Out[19]:

GridSearchCV

estimator: XGBRegressor

XGBRegressor

In [20]:

```
print("Best XGBoost parameters:", xgb_grid_search.best_params_)
```

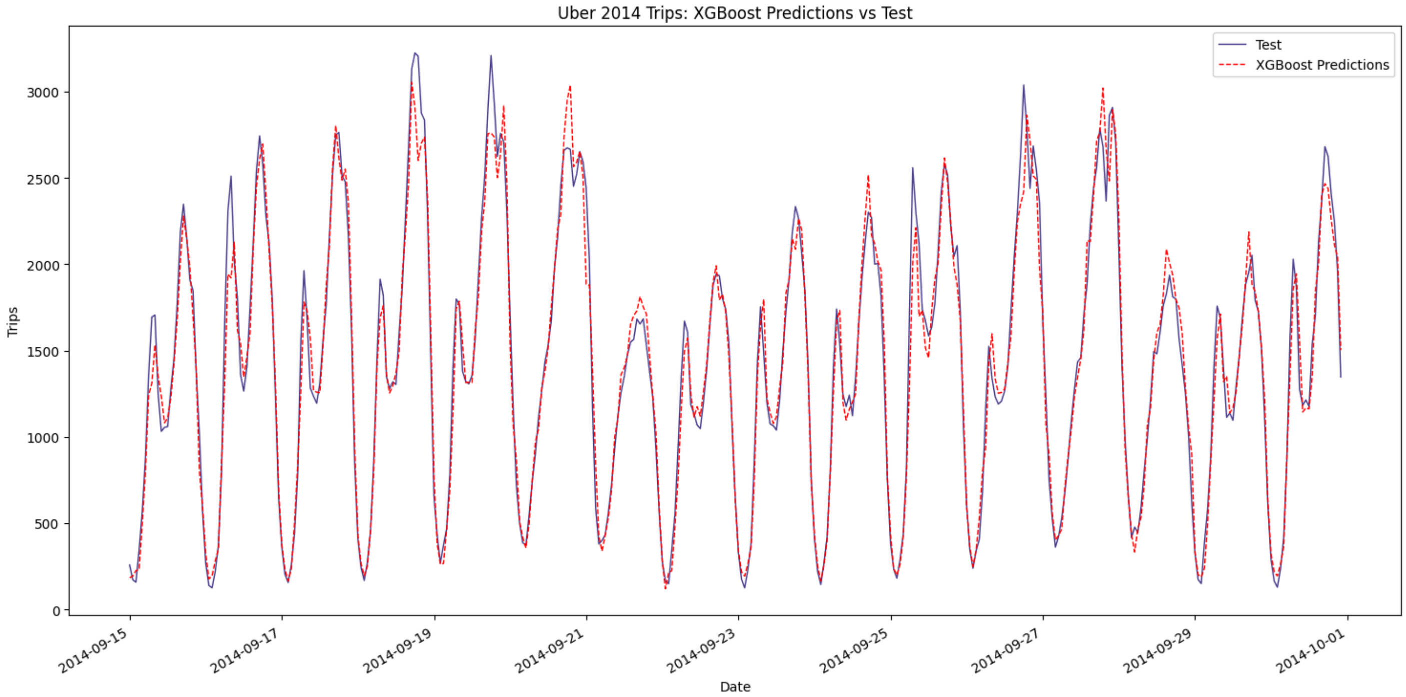
```
Best XGBoost parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1,
'max_depth': 6, 'n_estimators': 300, 'subsample': 0.6}
```

In [21]:

```python
xgb_predictions = xgb_grid_search.best_estimator_.predict(X_test)
```

In [22]:

```python
PlotPredictions([
    (uber2014_test.index,uber2014_test['Count'],'Test','-','darkslateblue'),
    (uber2014_test.index,xgb_predictions,'XGBoost Predictions','--','red')],
    'Uber 2014 Trips: XGBoost Predictions vs Test')
```



In [23]:

```python
xgb_mape = mean_absolute_percentage_error(uber2014_test['Count'], xgb_predictions)
print(f'XGBoost MAPE:\t\t{xgb_mape:.2%}')
```

```
XGBoost MAPE:		8.37%
```

5. Random Forest model

Finally, Random Forests are less susceptible to overfitting, however (again, in my experience) don't usually perform better than XGB

In [24]:

```
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],
    'max_features': [None, 'sqrt', 'log2']
}
```

In [25]:

```
rf_model = RandomForestRegressor(random_state=seed)
```

In [26]:

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

```
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
```

Out[26]:

```
GridSearchCV

estimator: RandomForestRegressor

RandomForestRegressor
```

In [27]:

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

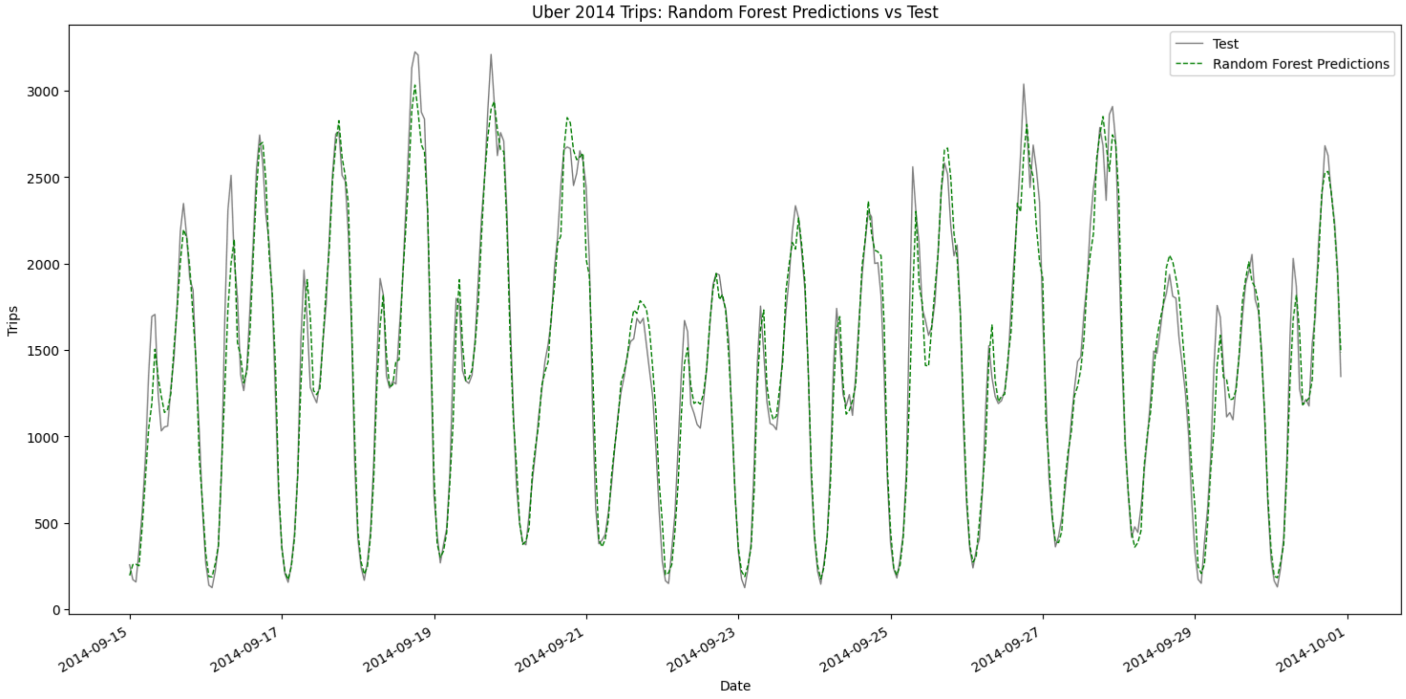
```
Best Random Forest parameters: {'max_depth': 30, 'max_features': None,
'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
```

In [28]:

```python
rf_predictions = rf_grid_search.best_estimator_.predict(X_test)
```

In [29]:

```python
PlotPredictions([
    (uber2014_test.index,uber2014_test['Count'],'Test','-','gray'),
    (uber2014_test.index,rf_predictions,'Random Forest Predictions','--','green')],
    'Uber 2014 Trips: Random Forest Predictions vs Test')
```



In [30]:

```python
rf_mape = mean_absolute_percentage_error(uber2014_test['Count'], rf_predictions)
print(f'Random Forest Mean Percentage Error:\t{rf_mape:.2%}')
```

```
Random Forest Mean Percentage Error:    9.61%
```

6. Gradient Boosted Regression Tree model

In [31]:

```
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']
}
```

In [32]:

```
gbr_model = GradientBoostingRegressor(random_state=seed)
```

In [33]:

```
gbr_grid_search = GridSearchCV(estimator=gbr_model, param_grid=gbr_param_grid,
cv=tscv, n_jobs=-1, scoring='neg_mean_absolute_percentage_error',verbose = 1)
gbr_grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
```

Out[33]:

GridSearchCV

estimator: GradientBoostingRegressor

GradientBoostingRegressor

In [34]:

```
print("Best Random Forest parameters:", gbr_grid_search.best_params_)
```

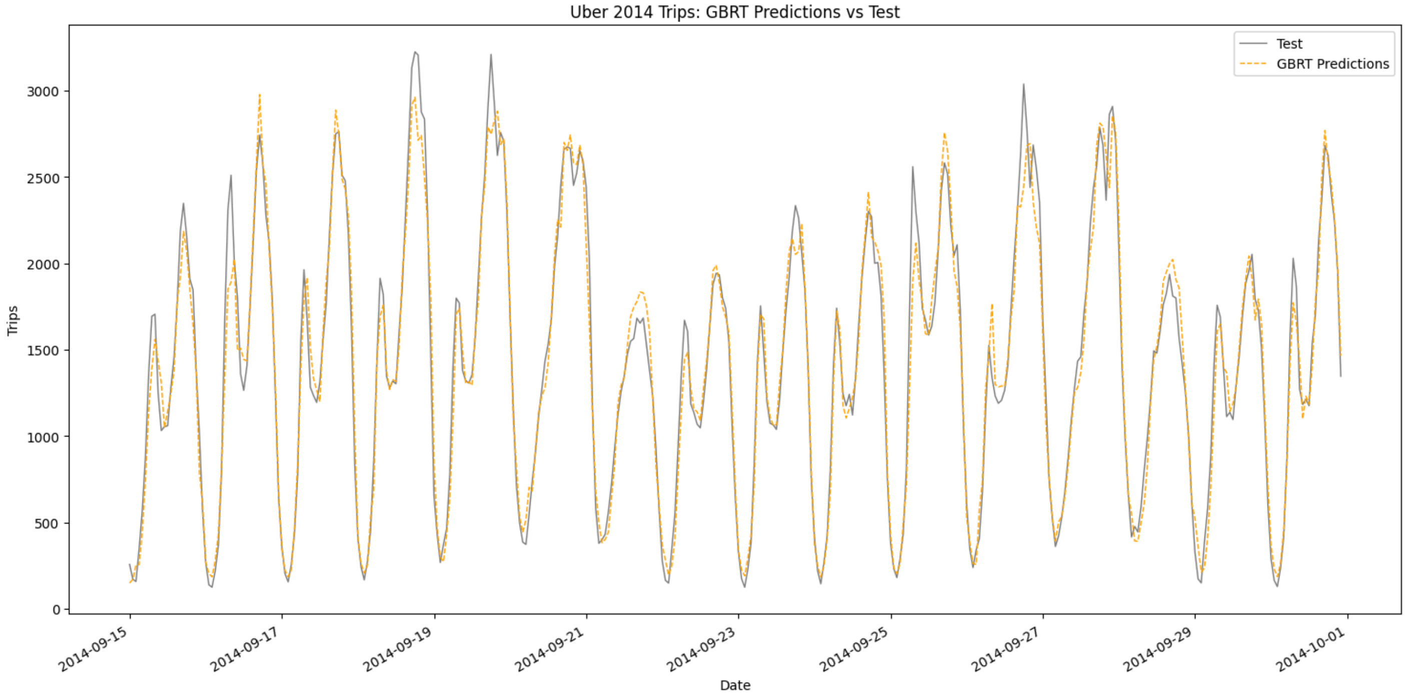
```
Best Random Forest parameters: {'learning_rate': 0.1, 'max_depth': 5,
'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5,
'n_estimators': 300}
```

In [35]:

```
gbr_predictions = gbr_grid_search.best_estimator_.predict(X_test)
```

In [36]:

```
PlotPredictions([
    (uber2014_test.index,uber2014_test['Count'],'Test','-','gray'),
    (uber2014_test.index,gbr_predictions,'GBRT Predictions','--','orange')],
    'Uber 2014 Trips: GBRT Predictions vs Test')
```



In [37]:

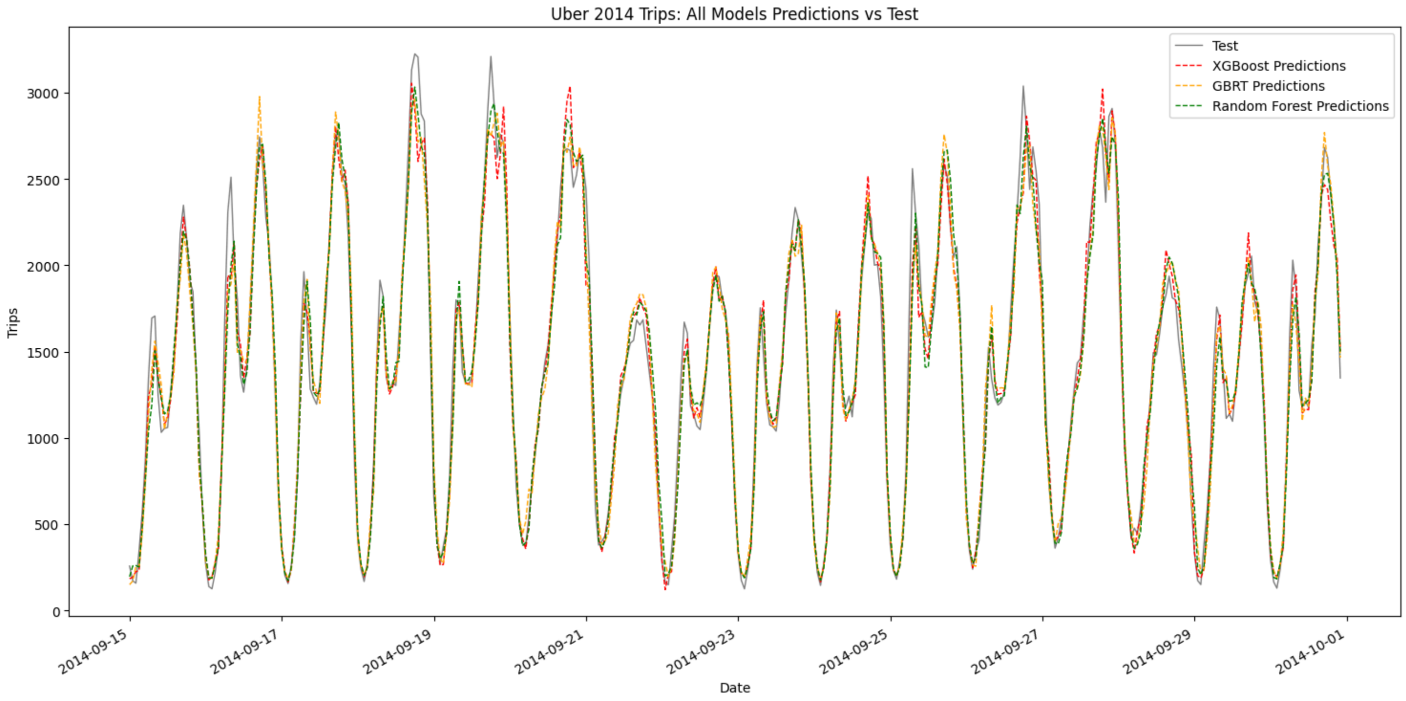
```
gbr_mape = mean_absolute_percentage_error(y_test, gbr_predictions)
print(f'GBTR Percentage Error:\t{gbr_mape:.2%}')
```

```
GBTR Percentage Error: 10.02%
```

7. Visualizing all models at once

In [38]:

```
PlotPredictions([
    (uber2014_test.index,uber2014_test['Count'],'Test','-','gray'),
    (uber2014_test.index,xgb_predictions,'XGBoost Predictions','--','red'),
    (uber2014_test.index,gbr_predictions,'GBRT Predictions','--','orange'),
    (uber2014_test.index,rf_predictions,'Random Forest Predictions','--','green')],
    'Uber 2014 Trips: All Models Predictions vs Test')
```



The above plot shows how all algorithms have actually being very close to predicting the test set. Visually, we can safely assume that using either algorithm could be a safe bet. The last step is to try an ensemble to

8. Ensemble

Building the ensemble requires to understand how each algorithm has performed individually first. Then, decide how we can leverage each one's strenghts to our advantage.

In [39]:

```
print(f'XGBoost MAPE:\t\t\t{xgb_mape:.2%}')
print(f'Random Forest MAPE:\t\t{rf_mape:.2%}')
print(f'GBTR Percentage Error:\t\t{gbr_mape:.2%}')
```

```
XGBoost MAPE:                8.37%
Random Forest MAPE:          9.61%
GBTR Percentage Error:       10.02%
```

Convert MAPE scores to weights: Since MAPE is inversely related to model performance, we can use the reciprocal of MAPE as a starting point for determining the weights. Normalize these reciprocals to get the weights. The ensemble prediction formula can be expressed as follows:

Reciprocal of XGBoost MAPE = 1/8.37 ≈ 0.119

Reciprocal of Random Forest MAPE = 1/9.61 ≈ 0.104

Reciprocal of GBTR MAPE = 1/10.02 ≈ 0.1

After doing the sum of all of them and applying each one's weight, we come up with the following formula:

Ensemble Prediction = 0.368 *XGBoost Prediction + 0.322* Random Forest Prediction + 0.310 * GBTR Prediction

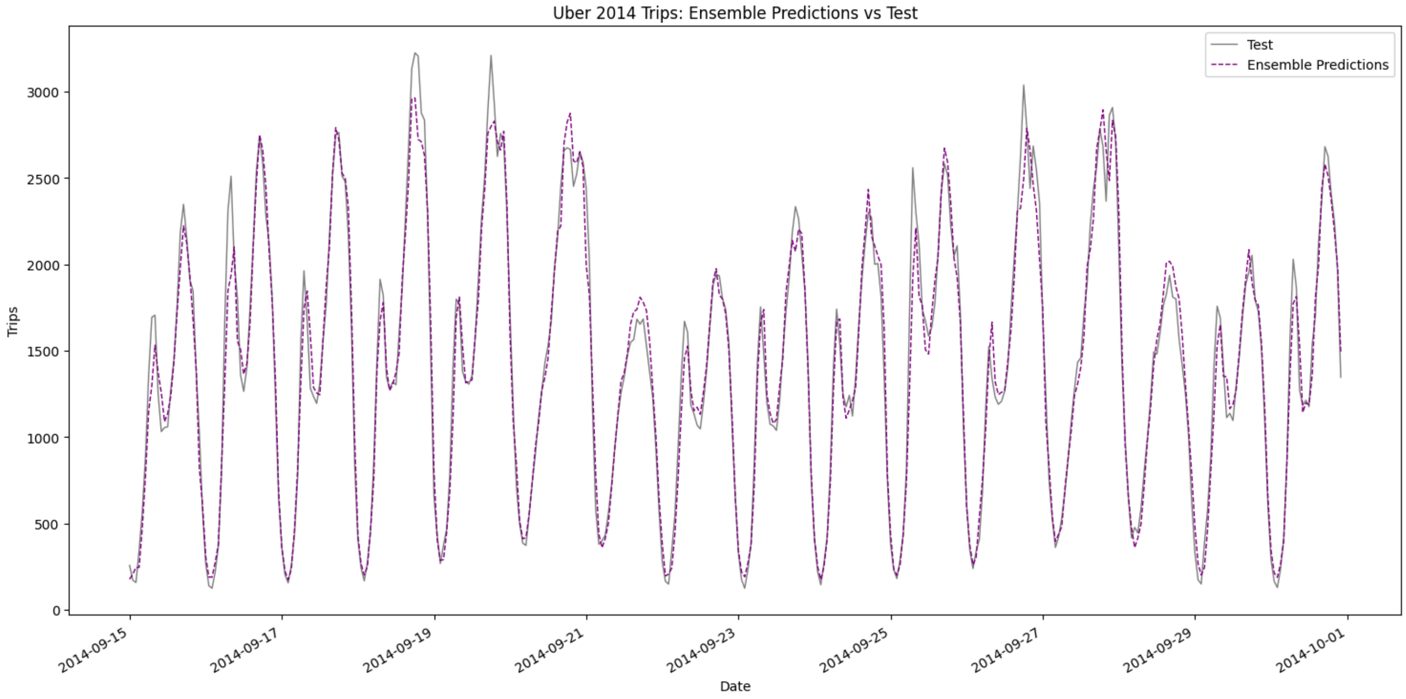
In [40]:

```
# Weights
weights = np.array([0.368, 0.322, 0.310])

# Combine predictions using weighted average
ensemble_predictions = (weights[0] * xgb_predictions + weights[1] * rf_predictions +
weights[2] * gbr_predictions)
```

In [41]:

```
PlotPredictions([
    (uber2014_test.index,uber2014_test['Count'],'Test','-','gray'),
    (uber2014_test.index,ensemble_predictions,'Ensemble
Predictions','--','purple')],
    'Uber 2014 Trips: Ensemble Predictions vs Test')
```



In [42]:

```
# Calculate MAPE for ensemble predictions on test set
ensemble_mape = mean_absolute_percentage_error(uber2014_test['Count'],
ensemble_predictions)
print(f'Ensemble MAPE:\t{ensemble_mape:.2%}')
```

```
Ensemble MAPE:	8.60%
```

In [43]:

```
print(f'XGBoost MAPE:\t\t{xgb_mape:.2%}')
print(f'Random Forest MAPE:\t{rf_mape:.2%}')
print(f'GBTR MAPE:\t\t{gbr_mape:.2%}')
print(f'Ensemble MAPE:\t\t{ensemble_mape:.2%}')
```

```
XGBoost MAPE:          8.37%
Random Forest MAPE:    9.61%
GBTR MAPE:       10.02%
Ensemble MAPE:         8.60%
```

linkcode

9. Insights and Conclusions from Training and Evaluation

Model Performance Overview:

- XGBoost: With a MAPE of 8.37%, XGBoost remains the top-performing model, effectively capturing patterns in the Uber Trip 2014 data. Its strong performance highlights its ability to manage complex interactions and temporal dependencies.
- Random Forest: Recorded a MAPE of 9.61%, showing good performance. This model effectively utilizes the window-based logic to capture time-dependent variations in the data.
- Gradient Boosted Tree Regressor (GBTR): Achieved a MAPE of 10.02%, indicating reasonable performance, although it does not match the effectiveness of XGBoost or Random Forest.

Ensemble Model:

- The ensemble model achieved a MAPE of 8.60%, which is an improvement over both Random Forest and GBTR. This performance showcases the ensemble's ability to integrate the strengths of the individual models while providing robust and stable predictions.
- The ensemble combines predictions from XGBoost, Random Forest, and GBTR, capitalizing on the complementary strengths of each model.

Impact of Window-Based Logic:

- Applying window-based logic to model training has effectively captured temporal dependencies in the data, resulting in enhanced predictive accuracy across all models.
- This approach ensures that the models can better handle seasonality and trends, which is crucial for accurate time series forecasting, particularly in dynamic contexts like ride-sharing demand.

Cross-Validation and Parameter Tuning:

- Cross-validation has provided a reliable assessment of model performance in temporal contexts, ensuring robustness and reducing the risk of overfitting.
- Parameter tuning, particularly for XGBoost and GBTR, has likely contributed to their strong performances, reflecting effective optimization efforts.

Practical Implications:

- For practical applications, XGBoost is recommended for scenarios where achieving the lowest error is critical due to its superior MAPE.
- The ensemble model serves as a strong alternative, providing improved predictive performance over the individual models, particularly useful for scenarios requiring stability and reliability.

Final Conclusion

The training and evaluation of these models underscore the effectiveness of XGBoost, with its best-in-class MAPE of 8.37%. The ensemble model, achieving a MAPE of 8.60%, effectively combines the strengths of the individual models, resulting in robust and reliable predictions. These findings highlight the importance of considering temporal structures in time series data and lay a strong foundation for future predictive modeling efforts in similar applications.

Reference link