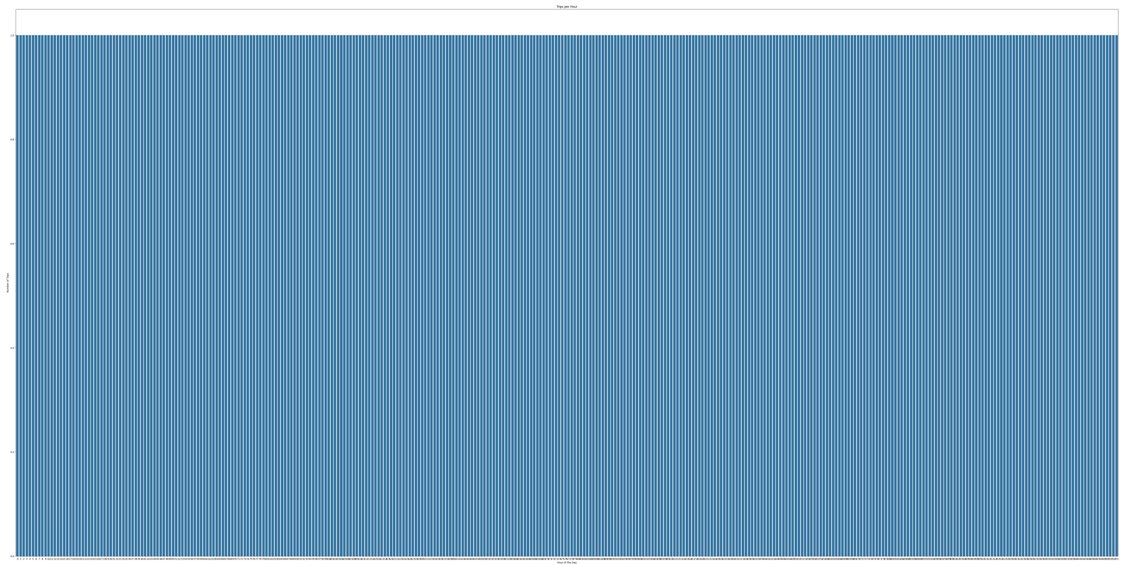
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score


# Load the dataset
data = pd.read_csv('/content/Uber-Jan-Feb-FOIL.csv')
# Display basic info about the dataset
print(data.info())
```

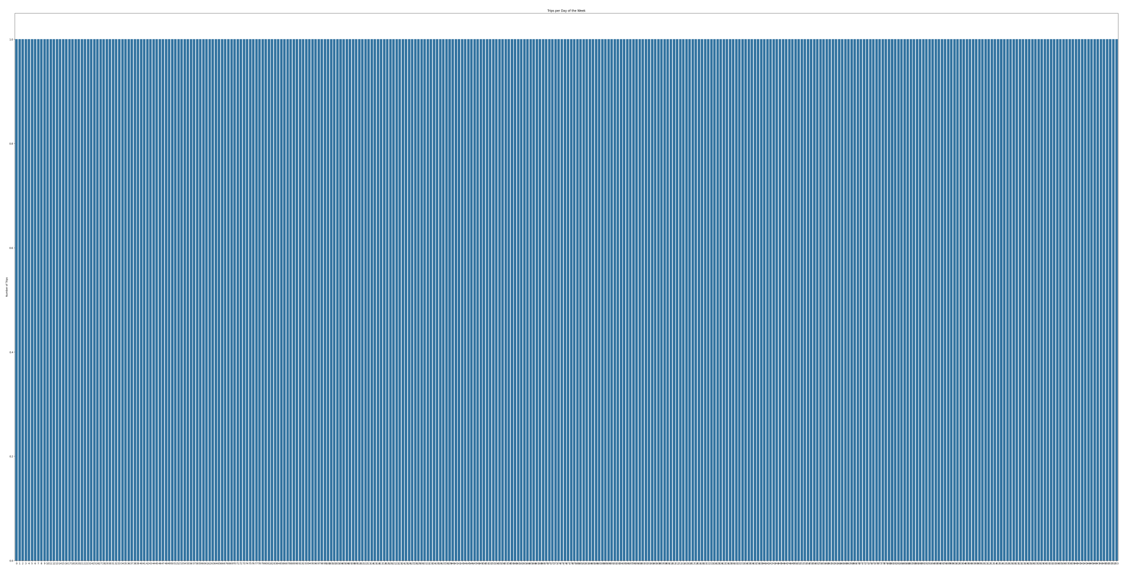
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 354 entries, 0 to 353
Data columns (total 4 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   dispatching_base_number  354 non-null    object
 1   date                     354 non-null    object
 2   active_vehicles          354 non-null    int64 
 3   trips                    354 non-null    int64 
dtypes: int64(2), object(2)
memory usage: 11.2+ KB
None
```

```
# Data Preprocessing
# Convert Date/Time to datetime object
data['date'] = pd.to_datetime(data['date'])
# Extracting useful information from Date/Time
data['Hour'] = data['date'].dt.hour
data['Day'] = data['date'].dt.day
data['DayOfWeek'] = data['date'].dt.dayofweek
data['Month'] = data['date'].dt.month


# Exploratory Data Analysis
# Plotting the number of trips per hour
plt.figure(figsize=(80,40))
sns.countplot(data['Hour'])
plt.title('Trips per Hour')
plt.xlabel('Hour of the Day')
```

```
plt.ylabel('Number of Trips')
plt.show()
```



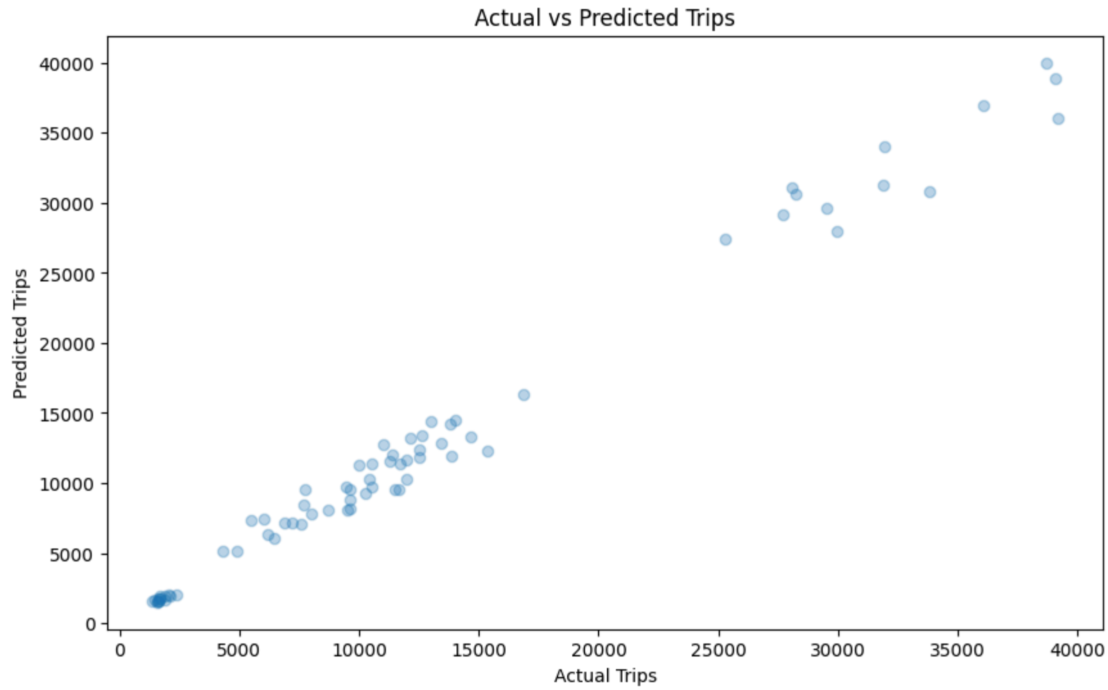
```
# Plotting the number of trips per day of the week
plt.figure(figsize=(80,40))
sns.countplot(data['DayOfWeek'])
plt.title('Trips per Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Trips')
plt.show()
```



```
# Feature Engineering
# Create dummy variables for categorical features
data = pd.get_dummies(data, columns=['dispatching_base_number'], drop_first=Tr
# Define features and target variable
X = data.drop(['trips', 'date'], axis=1) # Drop the target variable and the or
y = data['trips']

#Split the data into training and testing sets
X_train , X_test , y_train , y_test = train_test_split(X, y, test_size=0.2, ra

#Model Building
#Train a Random Forest Regressor
rfr = RandomForestRegressor(random_state=42)
rfr.fit(X_train, y_train)
#Make predictions on the test set
y_pred = rfr.predict(X_test)
#model Exaluation
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R2 Score:", r2_score(y_test, y_pred))
```

```
Mean Squared Error: 1510657.4039422534
R2 Score: 0.9859903305628406
```

```
# 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()
```



** Uber Trips Forecasting with XGBoost, Random Forests and Gradient Boosted Tree Regressors + Ensemble**

```
#Import the Necessary libraries + usefull functions
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,TimeSerie


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
 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)
```

```
#Reading the Uber Trips Dataset and preparing the data
# Use the existing dataframe 'data' loaded from /content/Uber-Jan-Feb-FOIL.csv
uber2014 = data.copy()
#Now make sure the date column is set to datetime, sorted and with an adequate
uber2014['date'] = pd.to_datetime(uber2014['date'])
uber2014 = uber2014.sort_values(by='date')
uber2014 = uber2014.rename(columns={'date':'Date'})
uber2014.set_index("Date" , inplace=True)
```

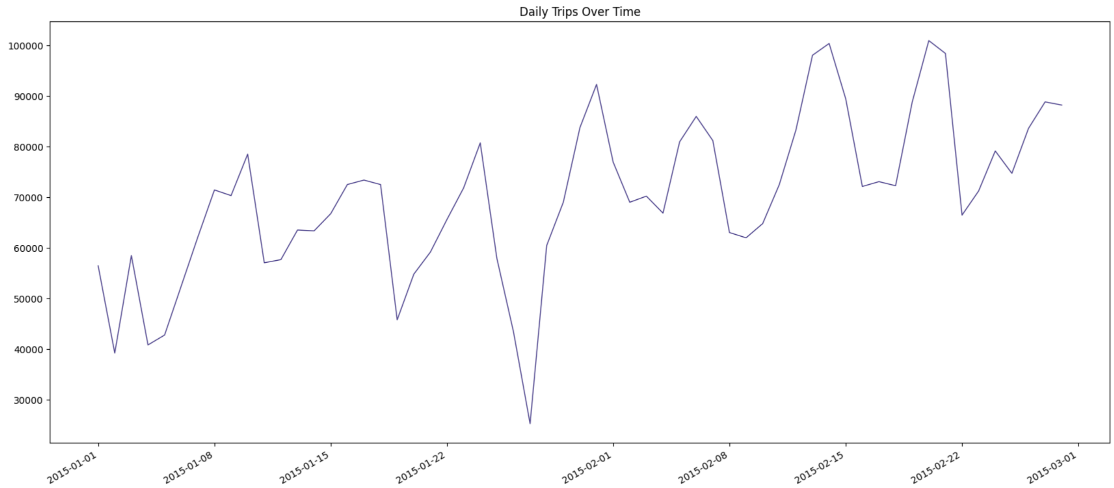
```
# Reload the dataset to get the original dispatching_base_number column
original_data = pd.read_csv('/content/Uber-Jan-Feb-FOIL.csv')

# Group by dispatching_base_number and count occurrences
hourly_counts = original_data.groupby('dispatching_base_number').size().reset_

# Convert the series to a DataFrame
uber2014 = hourly_counts.copy()

#Rename columns for clarity
uber2014.columns = ['Dispatching_Base_Number' , 'Count' ]

uber2014.head()
```

	Dispatching_Base_Number	Count
0	B02512	59
1	B02598	59
2	B02617	59
3	B02682	59
4	B02764	59

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

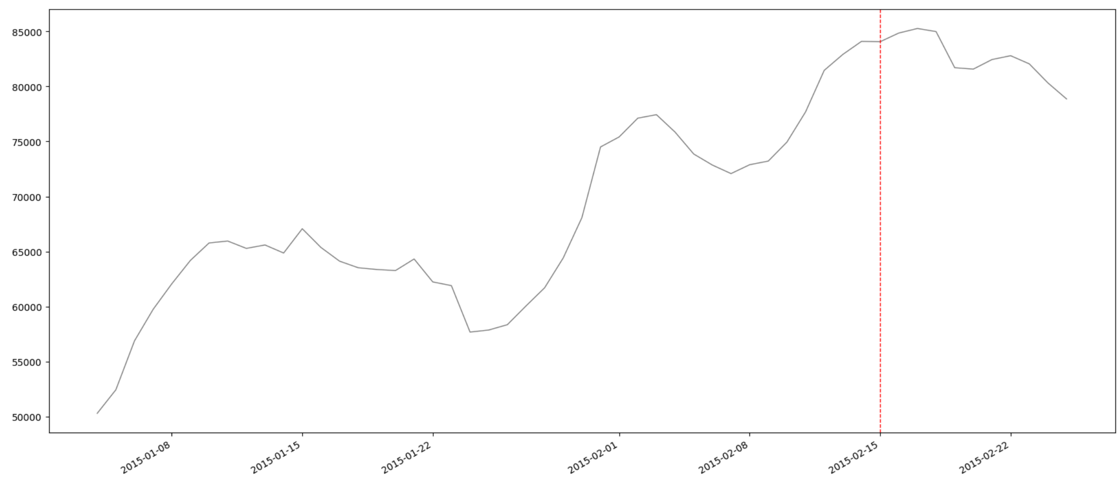
```
0
5
```

```
# Let's plot the series
# Assuming you want to perform seasonal decomposition on the 'trips' over time
# Use the original data dataframe and set 'date' as index
data['date'] = pd.to_datetime(data['date'])
data_time_series = data.set_index('date')['trips']

# Resample the data to a daily frequency and sum the trips for each day
data_time_series_daily = data_time_series.resample('D').sum()

plt.figure(figsize=(20, 8))
plt.plot(data_time_series_daily, linewidth=1, color='darkslateblue')
plt.xticks(rotation=30, ha='right')
plt.title('Daily Trips Over Time')
plt.show()

# Perform seasonal decomposition on the daily trips data
# You might need to adjust the period based on the seasonality you expect (e.g
# Given the data is for Jan and Feb, a weekly seasonality might be more approp
result = seasonal_decompose(data_time_series_daily, model='add', period=7)
PlotDecomposition(result)
```


Daily Trips Over Time
100000
90000
80000
70000
60000
50000
40000
30000
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


Observed
Trend
Seasonality
Residuals
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

```
cutoff_date = '2015-02-15 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='--', linewidt
plt.xticks(rotation=30,ha='right')
plt.show()
```



```
# Split the time series data into training and testing sets
uber2014_train = data_time_series_daily.loc[:cutoff_date]
uber2014_test = data_time_series_daily.loc[cutoff_date:]

# Plot the training and testing sets
plt.figure(figsize=(15,5))
uber2014_train.plot(label='TRAINING SET', style='-', lw=1)
uber2014_test.plot(label='TEST SET', style='-', lw=1)
plt.title('Train / Test Sets')
plt.legend()
plt.show()
```

```
#Set the window size
window_size = 7
#split data into training and test sets
X_train, y_train = create_lagged_features(uber2014_train.values, window_size)
X_test, y_test = create_lagged_features(uber2014_test.values, window_size)
seed =12345
```

XGBoost Model

```
tscv = TimeSeriesSplit(n_splits=5)
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],
}


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


xgb_grid_search = GridSearchCV(estimator=xgb_model, param_grid=xgb_param_grid,
xgb_grid_search.fit(X_train, y_train)
```

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

GridSearchCV

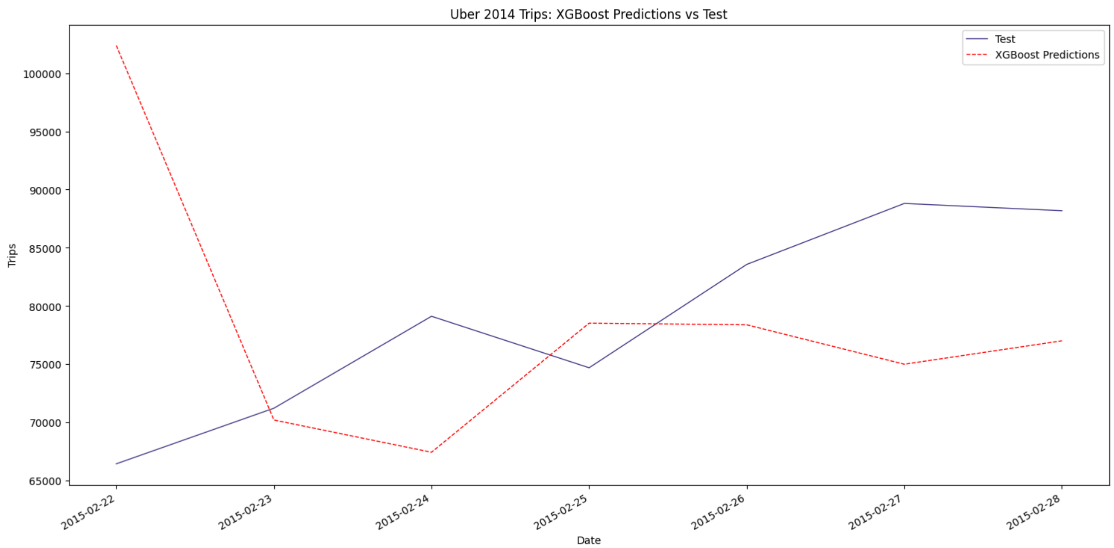
best_estimator_: XGBRegressor

XGBRegressor

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

```
Best XGBoost parameters: {'colsample_bytree': 0.6, 'learning_rate': 0.3,
```

```
xgb_predictions = xgb_grid_search.best_estimator_.predict(X_test)
PlotPredictions([
    (uber2014_test.index[window_size:],y_test , 'Test','-','darkslateblue'), #
    (uber2014_test.index[window_size:],xgb_predictions , 'XGBoost Predictions'
                'Uber 2014 Trips: XGBoost Predictions vs Test')
```



```
xgb_mape = mean_absolute_percentage_error(y_test, xgb_predictions)
print(f'XGBoost MAPE:\t\t{xgb_mape:.2%}')
```

```
XGBoost MAPE:           15.69%
```

Random Forest Model

```
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']
 }
rf_model =RandomForestRegressor(random_state=seed)
```

```
rf_grid_search = GridSearchCV(estimator=rf_model, param_grid=rf_param_grid, cv
rf_grid_search.fit(X_train, y_train)
```

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

▸ **GridSearchCV**

▸ **best_estimator_: RandomForestRegressor**

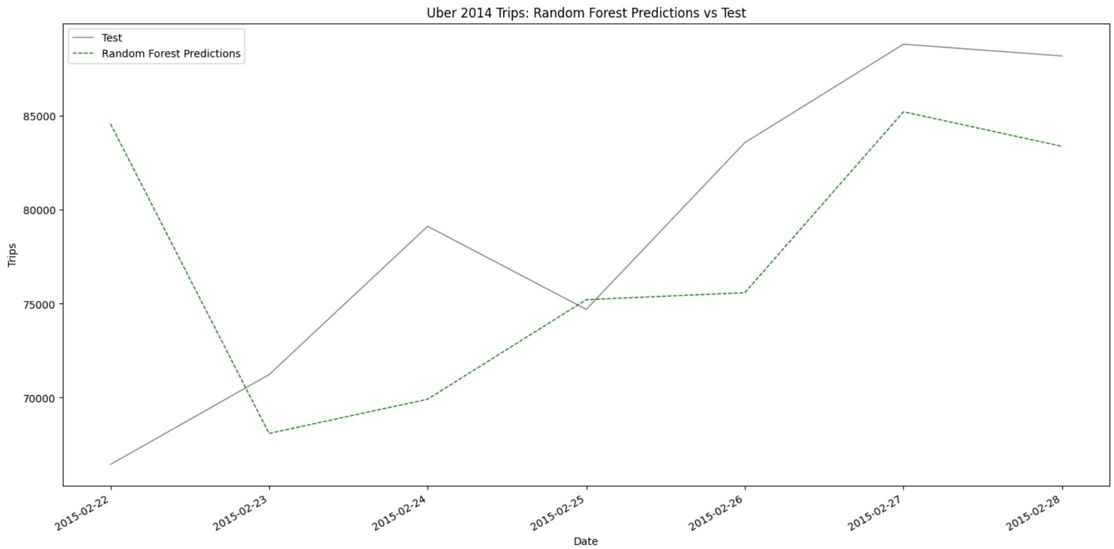
▸ RandomForestRegressor

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

```
Best Random Forest Parameters: {'max_depth': 10, 'max_features': None, 'mi
```

```
# Best Random Forest parameters: {'max_depth': 30, 'max_features': None,'min_s
rf_predictions = rf_grid_search.best_estimator_.predict(X_test)
```

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




```
rf_mape = mean_absolute_percentage_error(y_test, rf_predictions) # This line w
print(f'Random Forest Mean Percentage Error:\t{rf_mape:.2%}')
```

```
Random Forest Mean Percentage Error:    9.01%
```

GradientBoosted Regression Tree Model

```
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']
 }
gbr_model = GradientBoostingRegressor(random_state=seed)


gbr_grid_search = GridSearchCV(estimator=gbr_model, param_grid=gbr_param_grid,
gbr_grid_search.fit(X_train, y_train)
```

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

GridSearchCV

best_estimator_: GradientBoostingRegressor

GradientBoostingRegressor

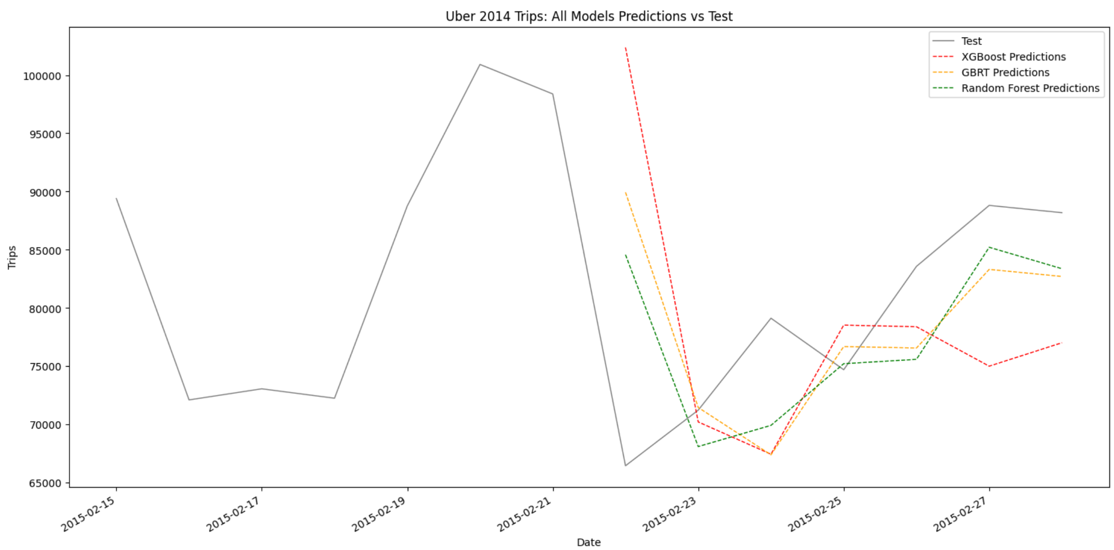
```
gbr_mape = mean_absolute_percentage_error(y_test, gbr_predictions)
print(f'Gradient Boosted Tree Mean Percentage Error:\t{gbr_mape:.2%}')
print("Best Random Forest parameters:", gbr_grid_search.best_params_)
```

```
Gradient Boosted Tree Mean Percentage Error:    10.57%
Best Random Forest parameters: {'learning_rate': 0.01, 'max_depth': 3, 'ma
```

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

Visualizing all Models at once

```
PlotPredictions([
 (uber2014_test.index,uber2014_test,'Test','-','gray'),
 (uber2014_test.index[window_size:],xgb_predictions,'XGBoost Predictions','--'
 (uber2014_test.index[window_size:],gbr_predictions,'GBRT Predictions','--','c
 (uber2014_test.index[window_size:],rf_predictions,'Random Forest Predictions'
```



Ensemble

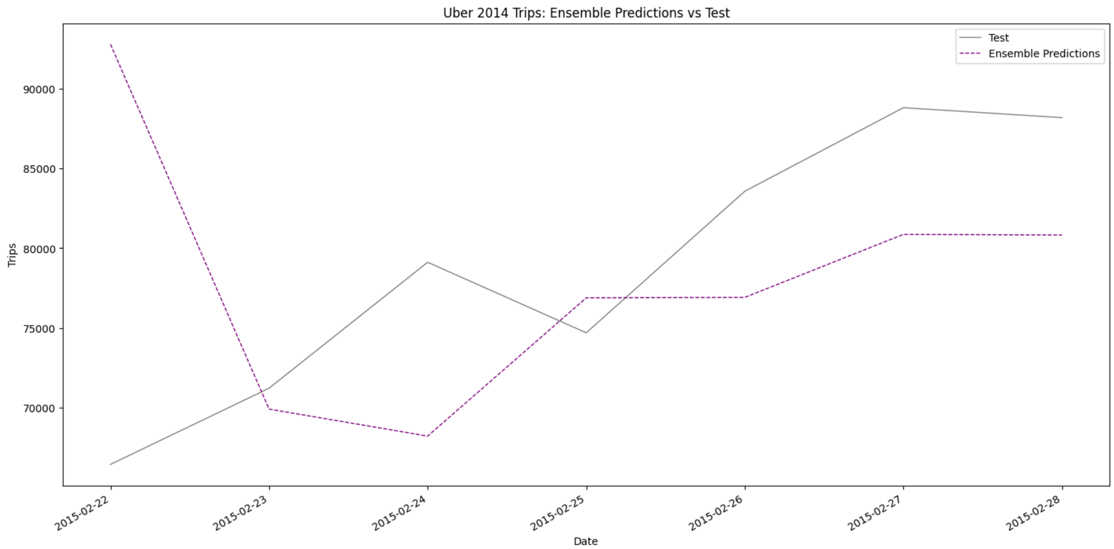
```
rf_mape = mean_absolute_percentage_error(y_test, rf_predictions)


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

```
XGBoost MAPE:                   15.69%
Random Forest MAPE:             9.01%
GBTR Percentage MAPE:           10.57%
```

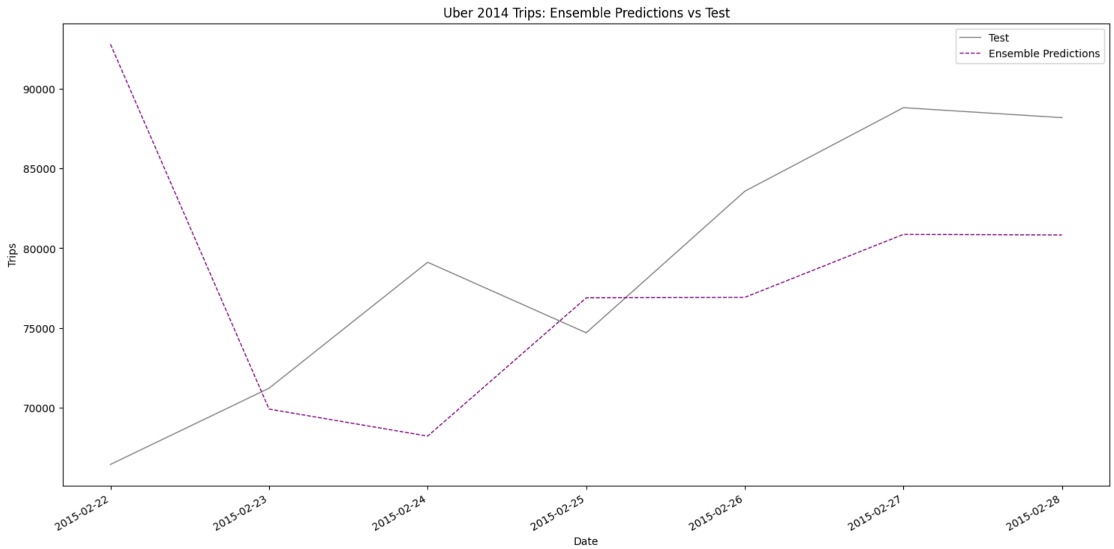
Convert MAPE scores to weights: Since MAPE is inversely related to model performance, we can use the reciprocal of MAPE as a starting point for determining the weights. Normalize these reciprocals to get the weights. The ensemble prediction formula can be expressed as follows: Reciprocal of XGBoost MAPE = 1/8.37 ≈ 0.119 Reciprocal of Random Forest MAPE = 1/9.61 ≈ 0.104 Reciprocal of GBTR MAPE = 1/10.02 ≈ 0.1

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

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


# Combine predictions using weighted average
ensemble_predictions = (weights[0] * xgb_predictions + weights[1] * rf_predict
PlotPredictions([(uber2014_test.index[window_size:],y_test,'Test','-','gray'),
```




```
PlotPredictions([
  (uber2014_test.index[window_size:],y_test,'Test','-','gray'),
  (uber2014_test.index[window_size:],ensemble_predictions,'Ensemble Predictior
```




```
# Calculate MAPE for ensemble predictions on test set
ensemble_mape = mean_absolute_percentage_error(y_test,
```