CSCI 6401-01
DATA MINING

Phase 5: Modeling data
Team: TSquad

1) **Team Members:**
    1. Kotha Priyatham Prem Kumar – pkoth10@unh.newhaven.edu
    2. Yamana Venkata Sai Sushmanth - vyama2@unh.newhaven.edu

Github Link: https://github.com/Premkumar5225/DataMining_Phase6

2) **Research Question:**

    This research question aims to investigate the feasibility of developing a predictive model that leverages insights from electric vehicle charging behavior to estimate the average charging time for a given zip code. By analyzing charging patterns and behavior data, this study seeks to provide a practical tool for users and stakeholders in the electric vehicle ecosystem to better plan and optimize charging experiences based on geographical location.

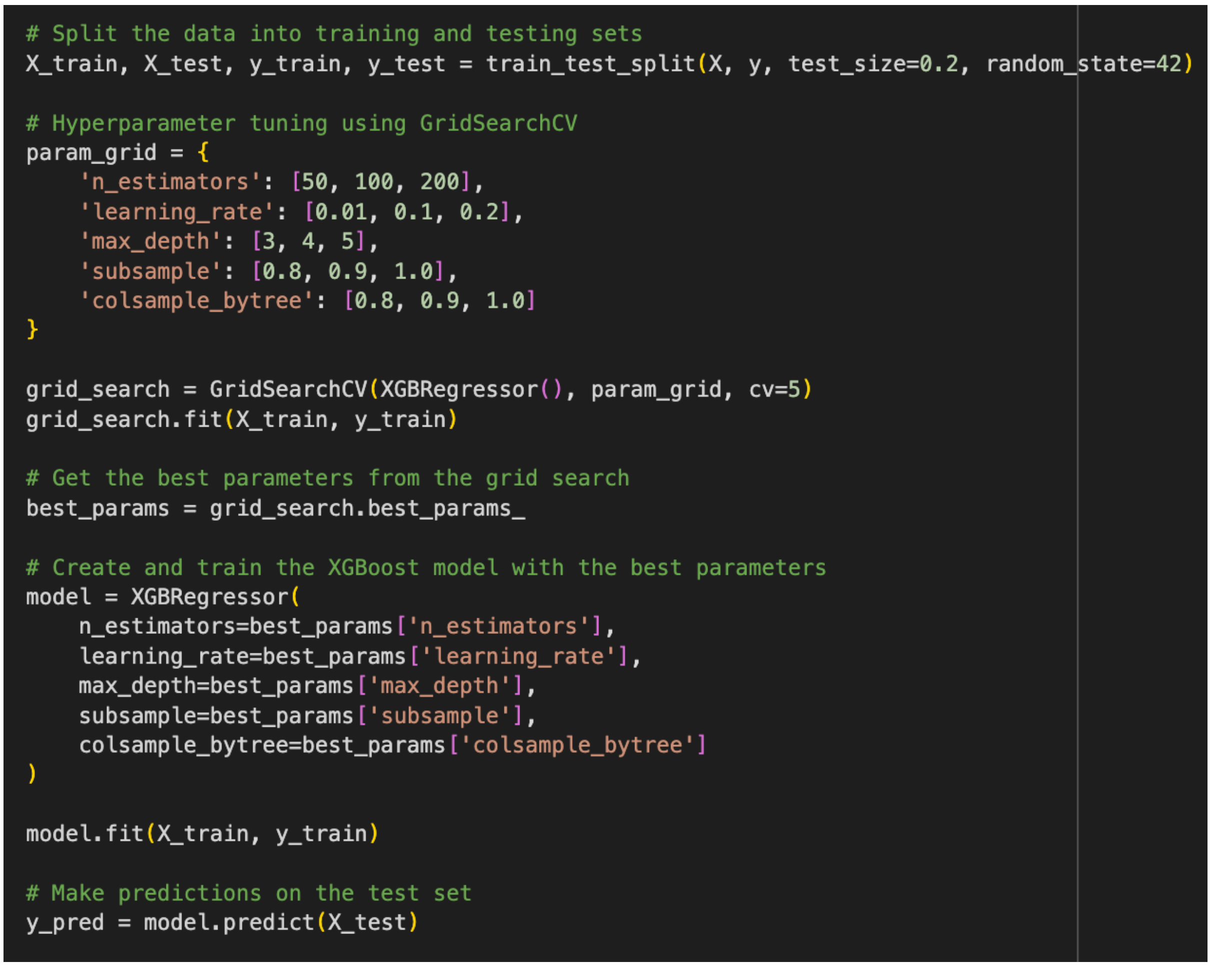
    Dataset link:
    https://dataverse.harvard.edu/file.xhtml?persistentId=doi:10.7910/DVN/NFPQLW/EQRRQH&version=1.0

    List of data mining techniques used:

    - **Neural Networks**
    - **Gradient Boosting**
    - **Naïve Bayes**
    - **Linear Regression**
    - **Decision Trees**
    - **Random Forest**
    - **Support Vector Machine**

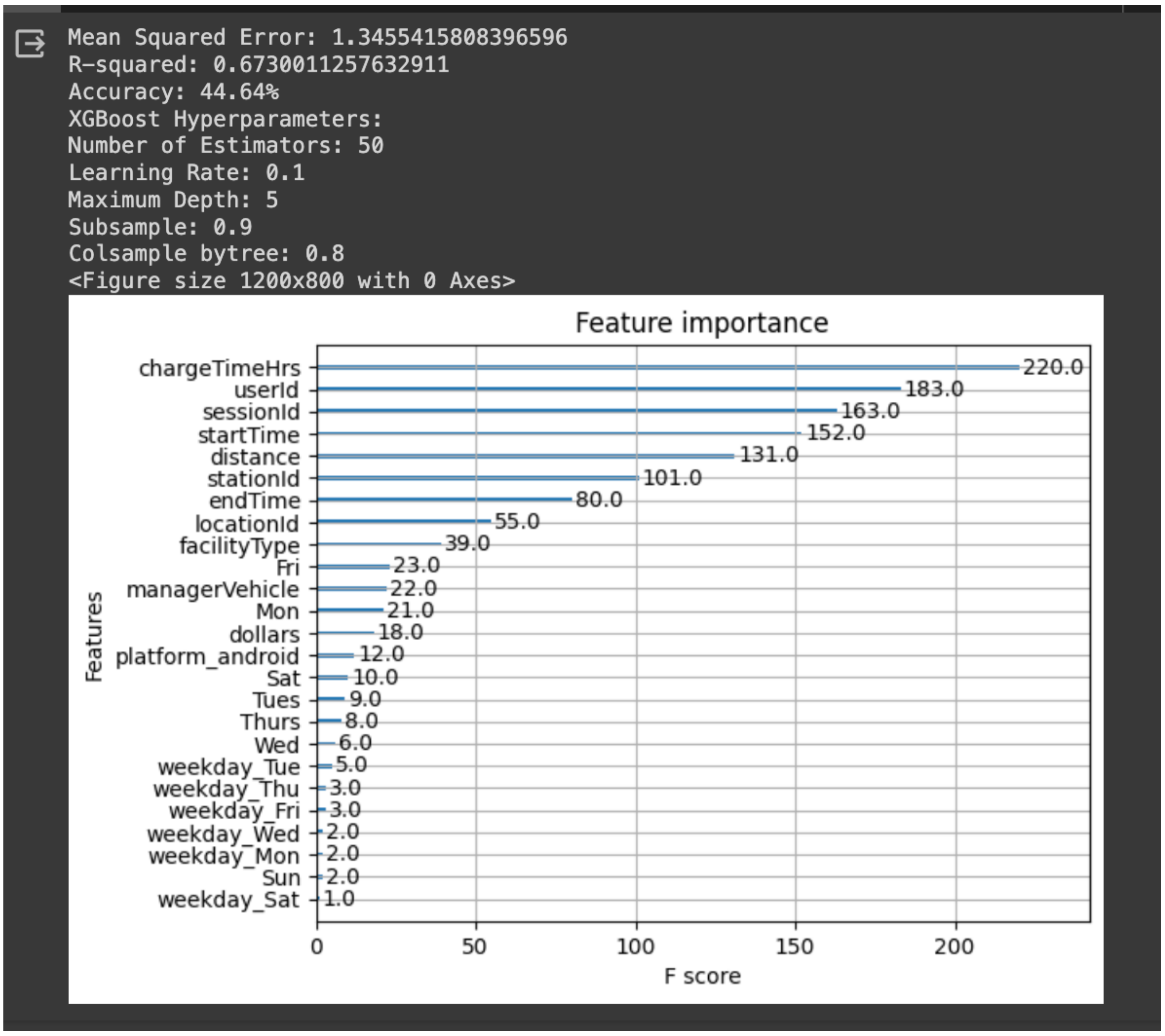
List Of Optimization Techniques used:
1. XGB Booster
2. ADA Boost
3. Gradient Boost
4. Neural Networks
5. Linear Regression
6. Random Forest

XGB Booster:



```
# 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, random_state=42)

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

grid_search = GridSearchCV(XGBRegressor(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_

# Create and train the XGBoost model with the best parameters
model = XGBRegressor(
    n_estimators=best_params['n_estimators'],
    learning_rate=best_params['learning_rate'],
    max_depth=best_params['max_depth'],
    subsample=best_params['subsample'],
    colsample_bytree=best_params['colsample_bytree']
)

model.fit(X_train, y_train)

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

he Mean Squared Error (MSE) of 1.35 indicates the average squared difference between predicted and actual values in a regression model, with lower values suggesting better performance. An R-squared of 0.67 represents the proportion of variance in the dependent variable explained by the model, with higher values indicating better fit. The 44.64% accuracy pertains to a classification model, revealing the percentage of correctly predicted instances. The XGBoost hyperparameters include 50 estimators, a learning rate of 0.1, a maximum depth of 5, a subsample of 0.9, and a colsample bytree of 0.8, providing details about the model's configuration.

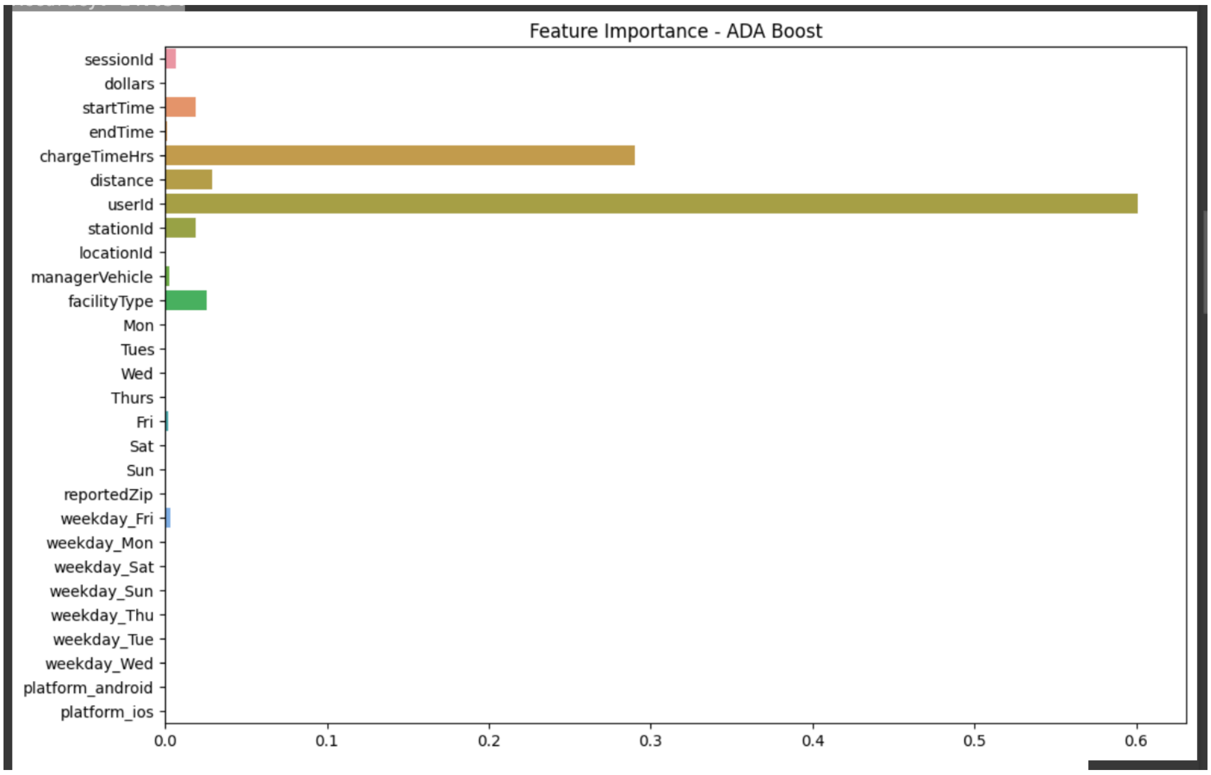


```
Mean Squared Error: 1.3455415808396596
R-squared: 0.6730011257632911
Accuracy: 44.64%
XGBoost Hyperparameters:
Number of Estimators: 50
Learning Rate: 0.1
Maximum Depth: 5
Subsample: 0.9
Colsample bytree: 0.8
<Figure size 1200x800 with 0 Axes>
```

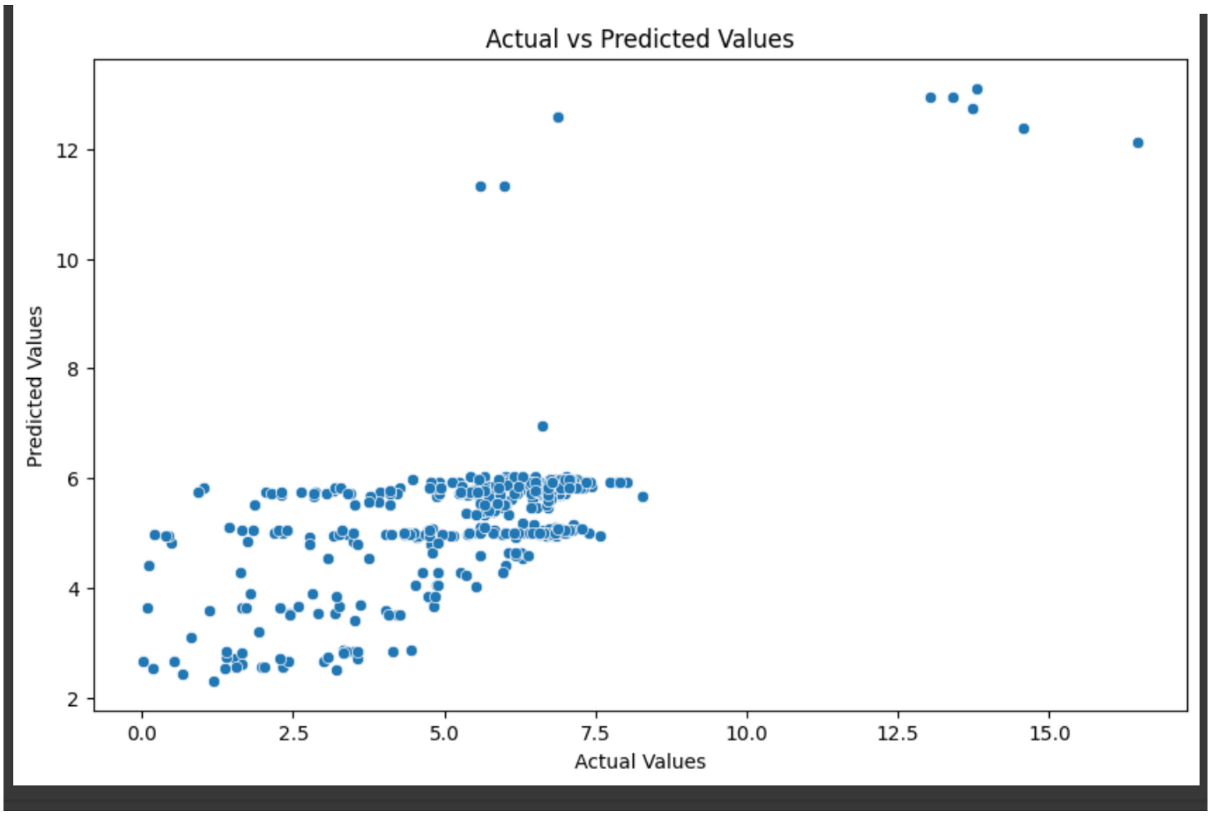
ADA:

```
# 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, random_state=42)

# Create and train the AdaBoost model
base_model = DecisionTreeRegressor(max_depth=3)
model = AdaBoostRegressor(base_model, n_estimators=50, learning_rate=0.1, random_state=42)
model.fit(X_train, y_train)

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

The Mean Squared Error (MSE) is a measure of the average squared difference between predicted and actual values, indicating a value of 2.17 in this context. The R-squared value of 0.47 suggests that approximately 47.3% of the variability in the data is explained by the model. The 24.03% accuracy indicates the proportion of correctly predicted outcomes in the model's performance.



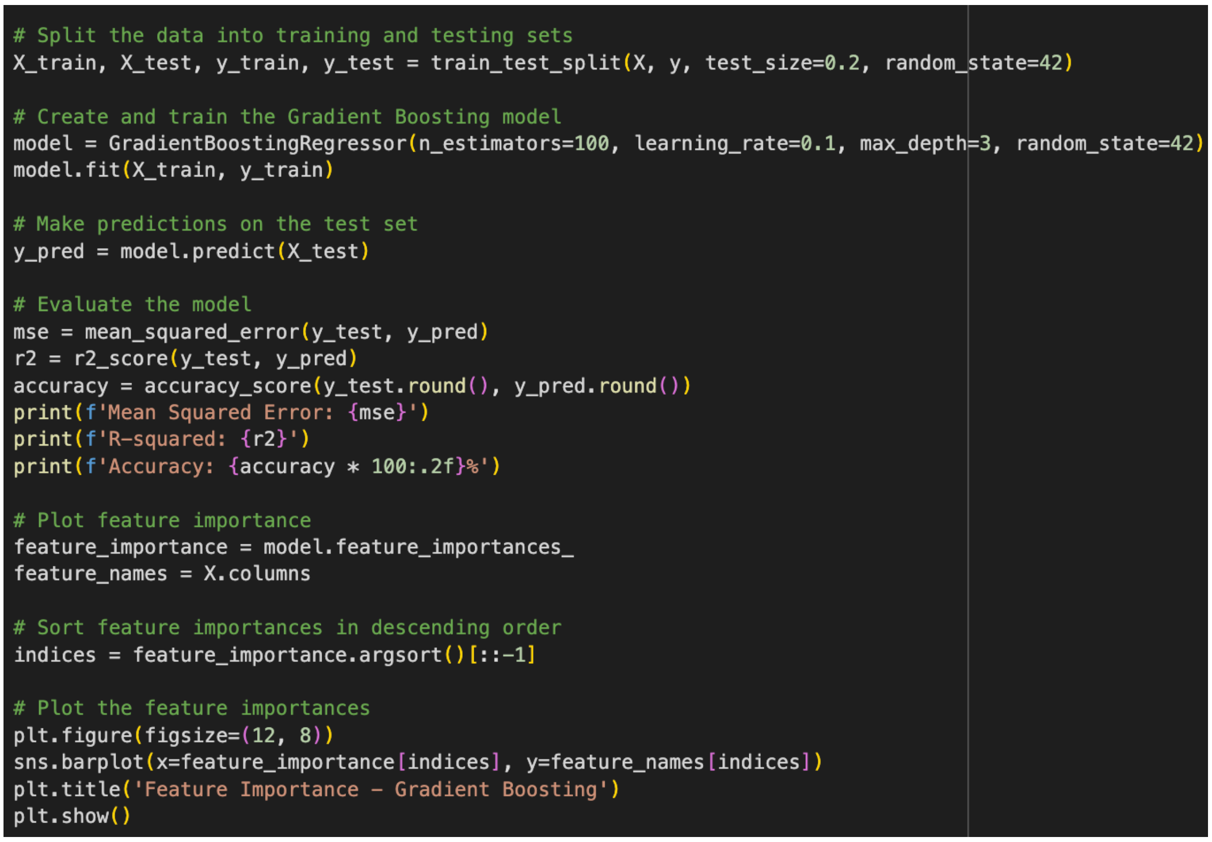
Feature Importance - ADA Boost
sessionId
dollars
startTime
endTime
chargeTimeHrs
distance
userId
stationId
locationId
managerVehicle
facilityType
Mon
Tues
Wed
Thurs
Fri
Sat
Sun
reportedZip
weekday_Fri
weekday_Mon
weekday_Sat
weekday_Sun
weekday_Thu
weekday_Tue
weekday_Wed
platform_android
platform_ios
0.0
0.1
0.2
0.3
0.4
0.5
0.6



Actual vs Predicted Values
Predicted Values
Actual Values
2
4
6
8
10
12
0.0
2.5
5.0
7.5
10.0
12.5
15.0

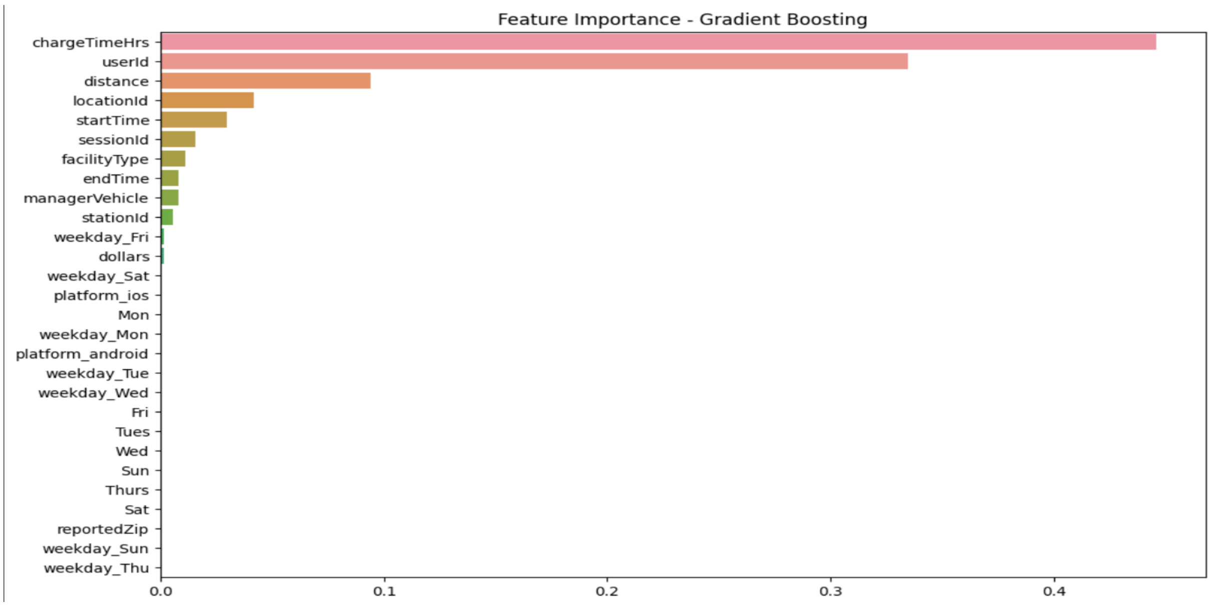
Gradient Boost:

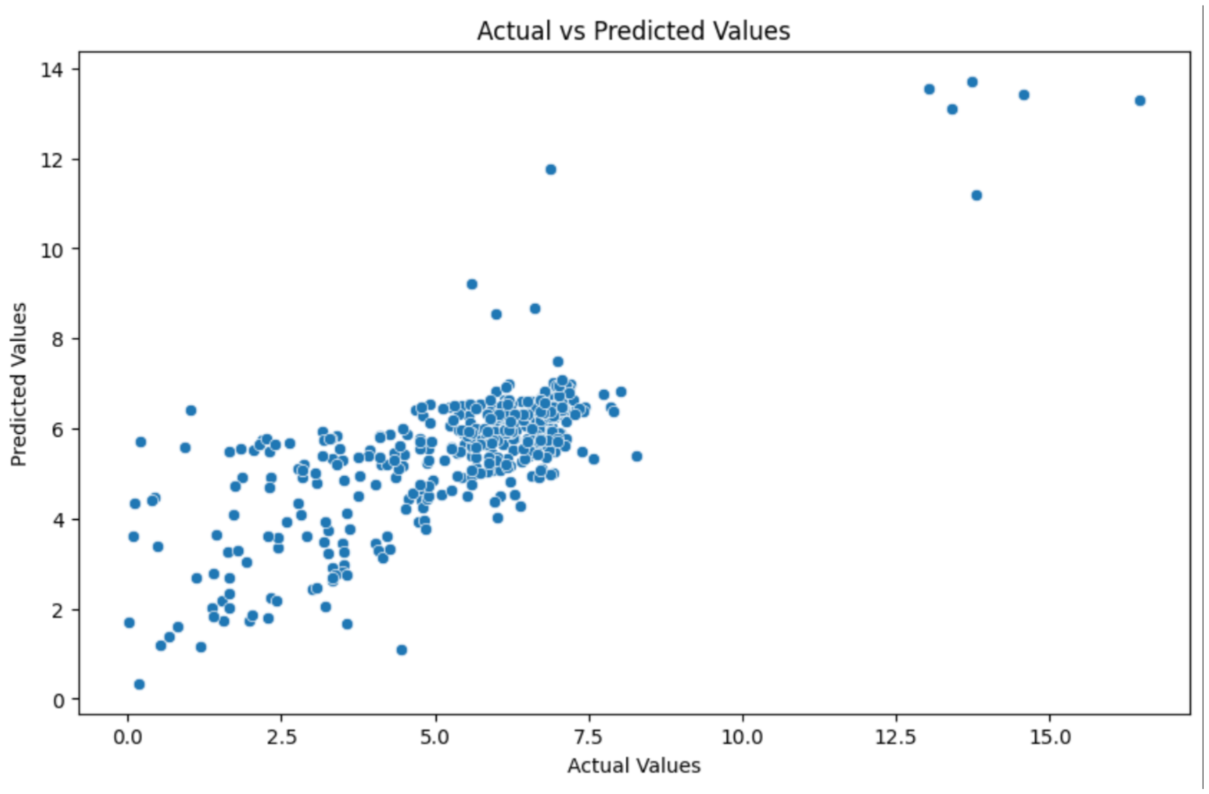
Gradient Boost is an ensemble learning technique where weak models, usually decision trees, are sequentially trained to correct the errors of their predecessors.



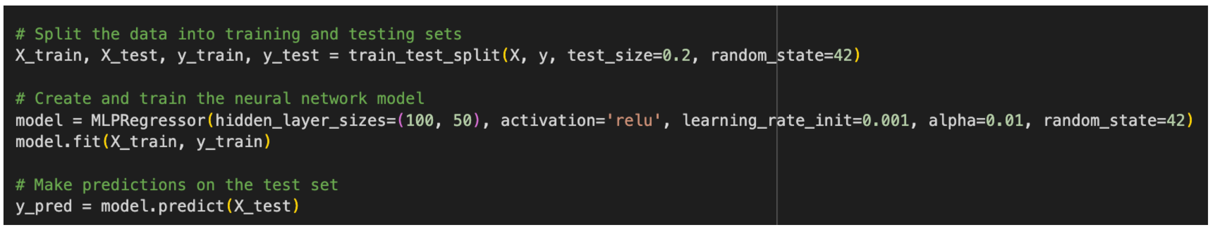
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Gradient Boosting model
model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
model.fit(X_train, y_train)

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

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = accuracy_score(y_test.round(), y_pred.round())
print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')
print(f'Accuracy: {accuracy * 100:.2f}%')

# Plot feature importance
feature_importance = model.feature_importances_
feature_names = X.columns

# Sort feature importances in descending order
indices = feature_importance.argsort()[::-1]

# Plot the feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x=feature_importance[indices], y=feature_names[indices])
plt.title('Feature Importance - Gradient Boosting')
plt.show()
```

Mean Squared Error: 1.5696655300939433
R-squared: 0.6185336309349991
Accuracy: 37.34%



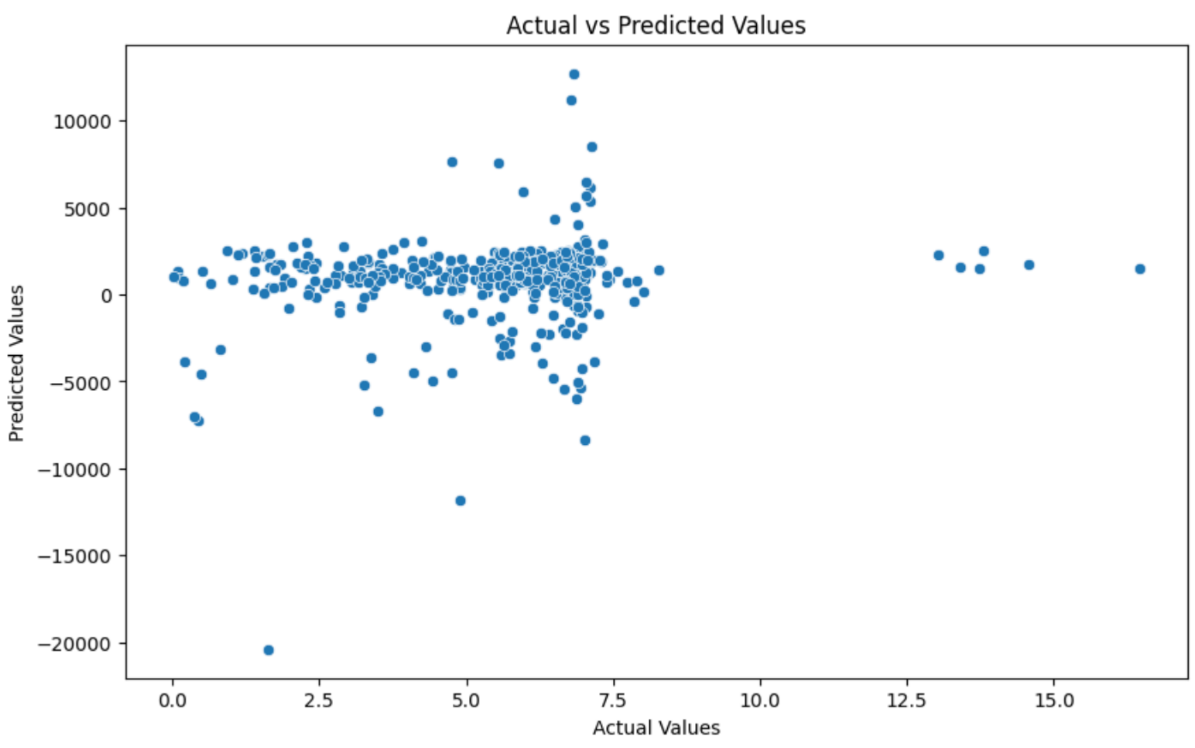


Neural Networks:

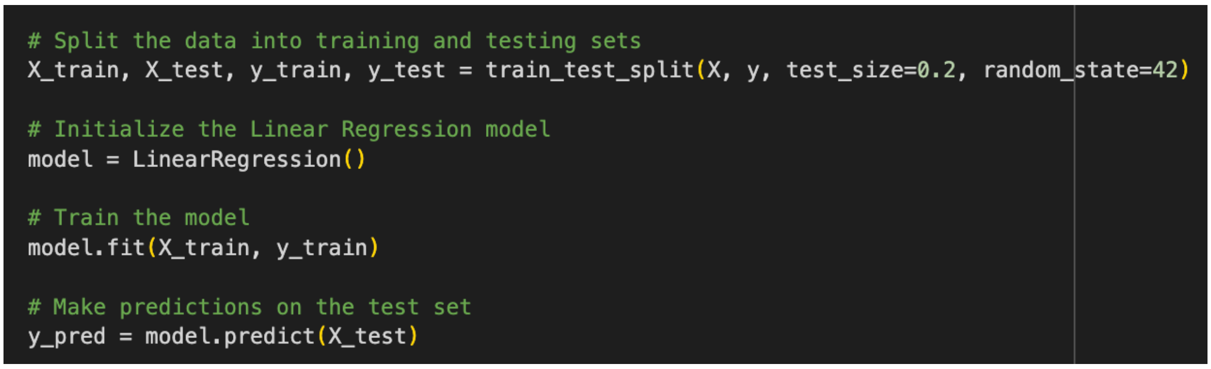


```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the neural network model
model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', learning_rate_init=0.001, alpha=0.01, random_state=42)
model.fit(X_train, y_train)

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

Mean Squared Error: 6159965.055536111
R-squared: -1497018.2427949996
Neural Network Hyperparameters:
Hidden Layer Sizes: (100, 50)
Activation Function: relu
Initial Learning Rate: 0.001
L2 Regularization Term (Alpha): 0.01

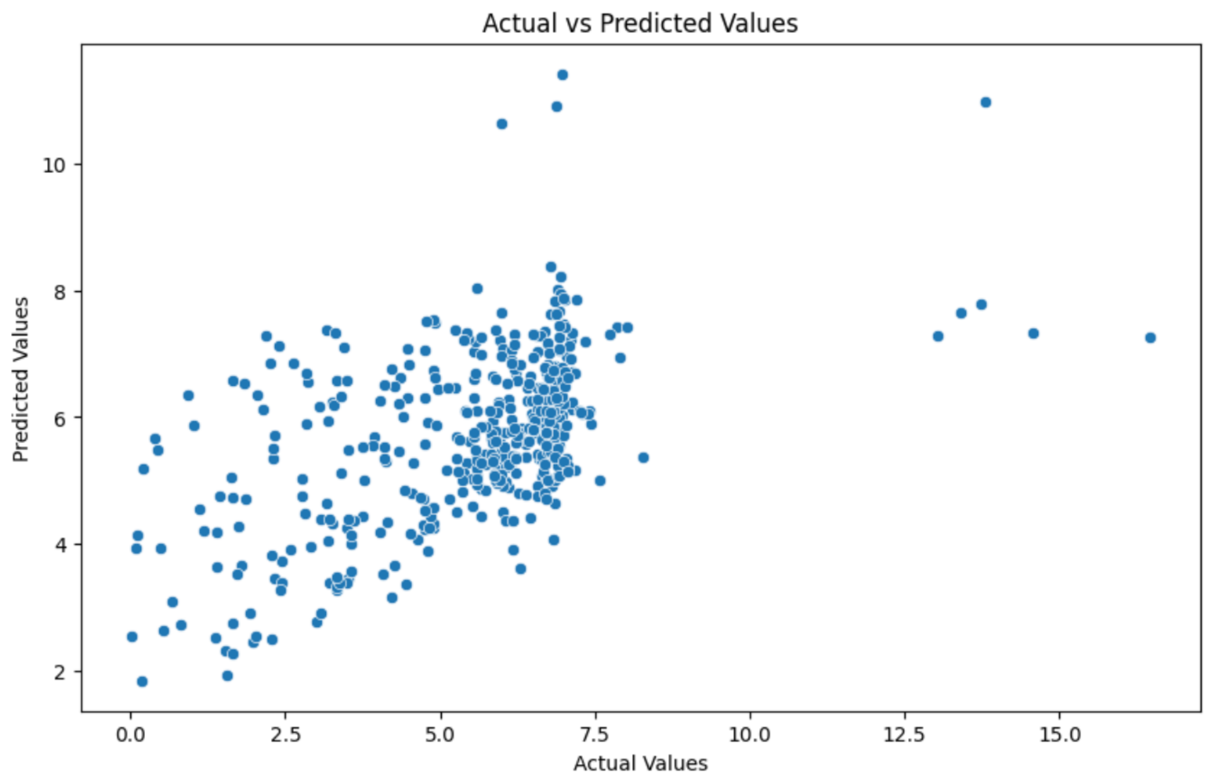


Linear Regression:

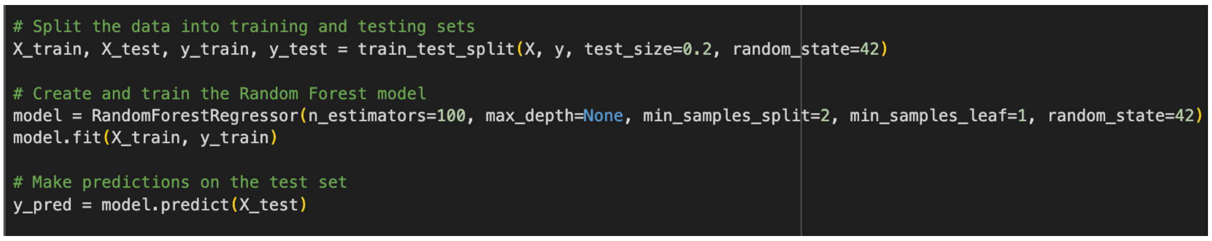


```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

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

Mean Squared Error (MSE): 3.0085285336910723
R-squared (R2): 0.2688554128426974

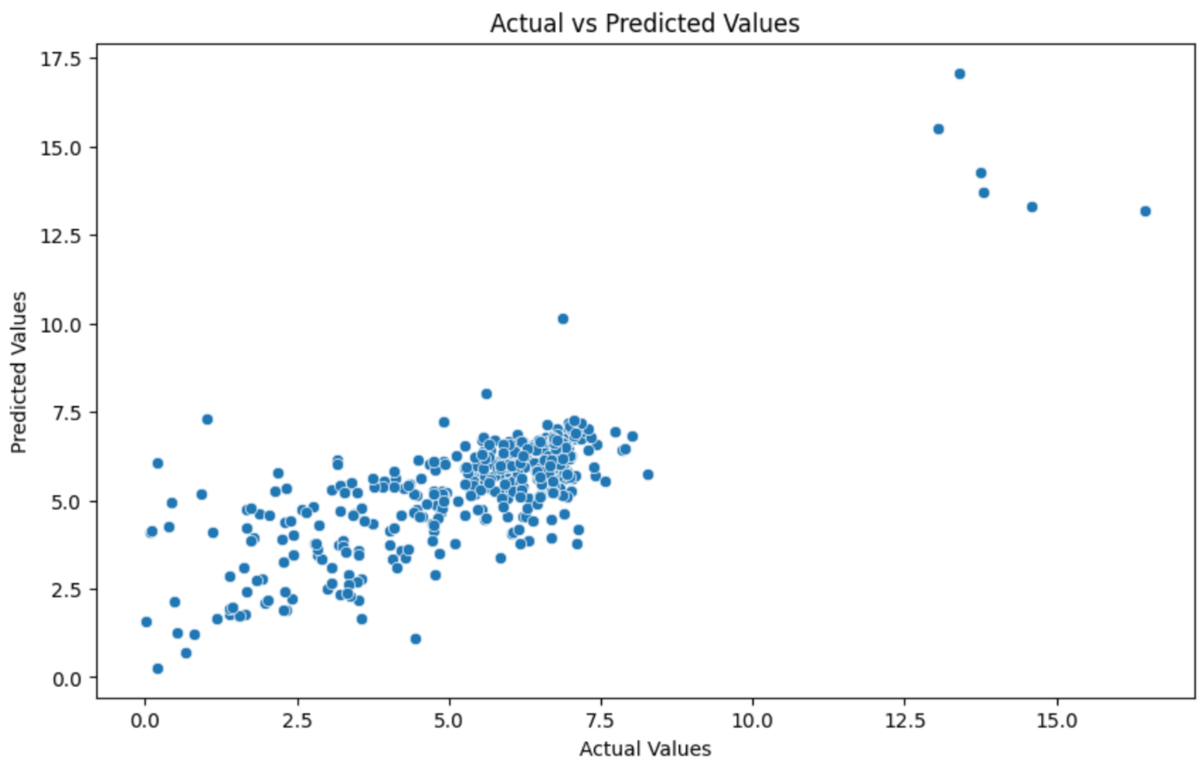


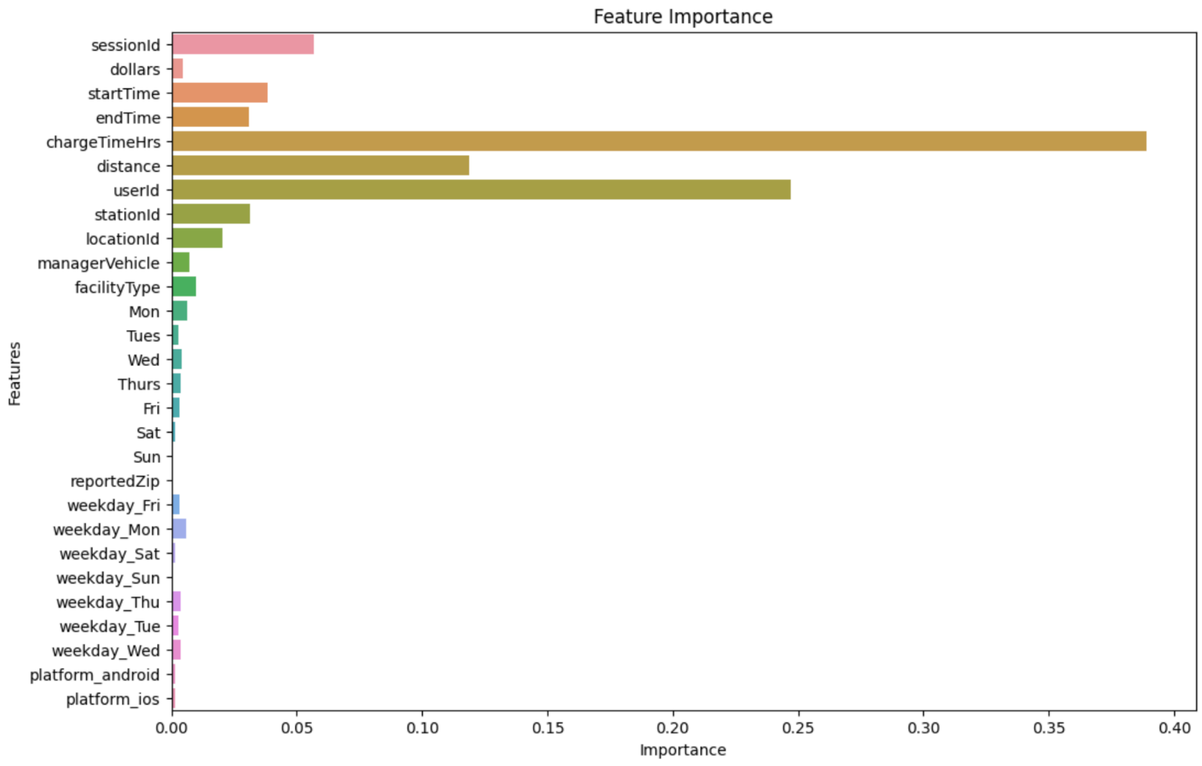
Random Forest



```
# 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, random_state=42)

# Create and train the Random Forest model
model = RandomForestRegressor(n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1, random_state=42)
model.fit(X_train, y_train)

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

Mean Squared Error: 1.4636601238841205
R-squared: 0.6442954869691961
Random Forest Hyperparameters:
Number of Trees (n_estimators): 100
Maximum Depth of Trees (max_depth): None
Minimum Samples Split (min_samples_split): 2
Minimum Samples Leaf (min_samples_leaf): 1





In conclusion, among the models evaluated, XGBoost demonstrated the best performance with a Mean Squared Error of 1.35, an R-squared value of 0.67, and an accuracy of 44.64%. The XGBoost model was optimized with hyperparameters including 50 estimators, a learning rate of 0.1, maximum depth of 5, subsample of 0.9, and colsample bytree of 0.8. Other models, such as ADA, Gradient Boost, Neural Networks, Linear Regression, and Random Forest, exhibited varying degrees of performance, with XGBoost showcasing superior predictive capabilities in this analysis.