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_Phase5

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

Hardware used:

1. Processor: i7
2. Hard disk:1000GB
3. RAM: 16GB
4. Operating system: Windows 11 64-bit
5. Tools used: Google Collab.
6. Language: Python

List of data mining techniques used:

- ☐ **Neural Networks**
- ☐ **Gradient Boosting**
- ☐ **Naïve Bayes**
- ☐ **Linear Regression**
- ☐ **Decision Trees**
- ☐ **Random Forest**

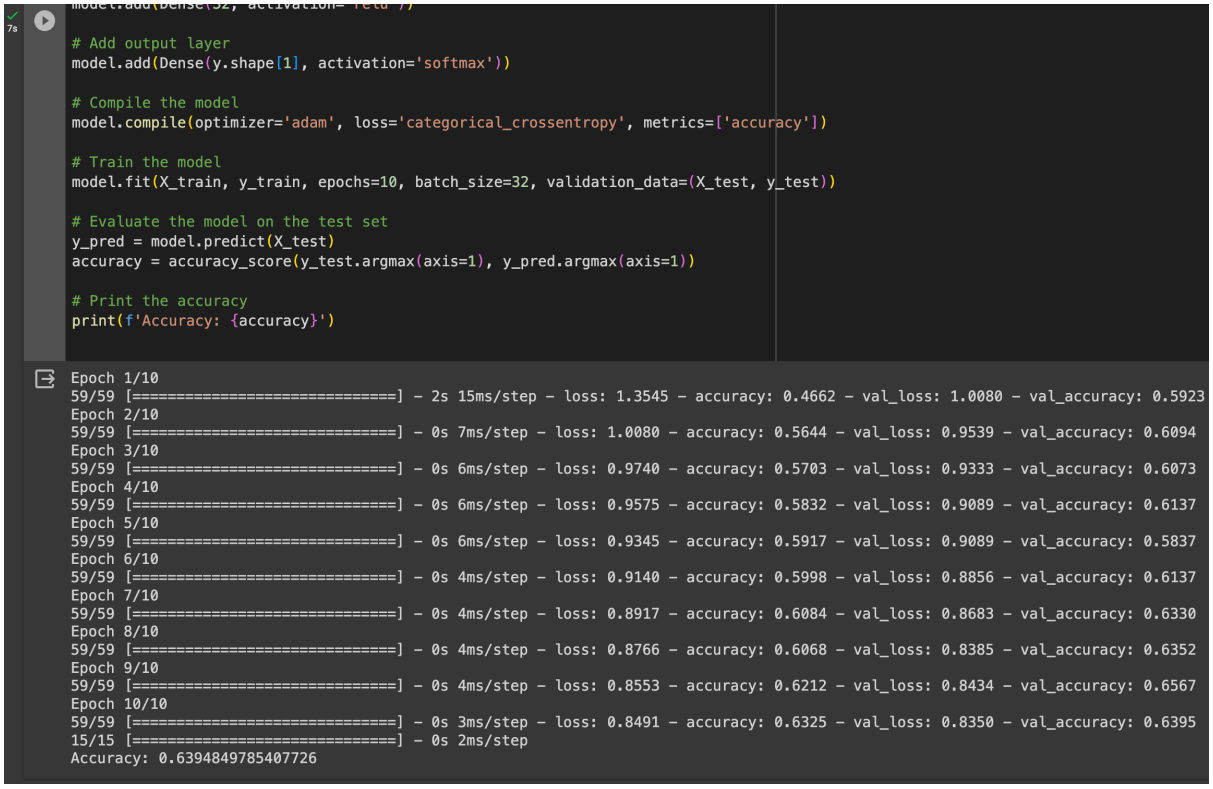
- **Support Vector Machine**

Neural Networks

Neural networks are a type of machine learning model inspired by the structure of the human brain. They consist of interconnected nodes (neurons) arranged in layers, each layer processing and passing information to the next. Through training on labeled data, neural networks can learn complex patterns and make predictions or classifications in tasks like image recognition, natural language processing, and more.

Nueral Networks

```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are features
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# Convert target variable to categorical
y = to_categorical(y)

# 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 a sequential neural network model
model = Sequential()

# Add input layer and hidden layers
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(32, activation='relu'))

# Add output layer
model.add(Dense(y.shape[1], activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```



```
# Add output layer
model.add(Dense(y.shape[1], activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))

# Evaluate the model on the test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test.argmax(axis=1), y_pred.argmax(axis=1))

# Print the accuracy
print(f'Accuracy: {accuracy}')
```

```
Epoch 1/10
59/59 [==============================] - 2s 15ms/step - loss: 1.3545 - accuracy: 0.4662 - val_loss: 1.0080 - val_accuracy: 0.5923
Epoch 2/10
59/59 [==============================] - 0s 7ms/step - loss: 1.0080 - accuracy: 0.5644 - val_loss: 0.9539 - val_accuracy: 0.6094
Epoch 3/10
59/59 [==============================] - 0s 6ms/step - loss: 0.9740 - accuracy: 0.5703 - val_loss: 0.9333 - val_accuracy: 0.6073
Epoch 4/10
59/59 [==============================] - 0s 6ms/step - loss: 0.9575 - accuracy: 0.5832 - val_loss: 0.9089 - val_accuracy: 0.6137
Epoch 5/10
59/59 [==============================] - 0s 6ms/step - loss: 0.9345 - accuracy: 0.5917 - val_loss: 0.9089 - val_accuracy: 0.5837
Epoch 6/10
59/59 [==============================] - 0s 4ms/step - loss: 0.9140 - accuracy: 0.5998 - val_loss: 0.8856 - val_accuracy: 0.6137
Epoch 7/10
59/59 [==============================] - 0s 4ms/step - loss: 0.8917 - accuracy: 0.6084 - val_loss: 0.8683 - val_accuracy: 0.6330
Epoch 8/10
59/59 [==============================] - 0s 4ms/step - loss: 0.8766 - accuracy: 0.6068 - val_loss: 0.8385 - val_accuracy: 0.6352
Epoch 9/10
59/59 [==============================] - 0s 4ms/step - loss: 0.8553 - accuracy: 0.6212 - val_loss: 0.8434 - val_accuracy: 0.6567
Epoch 10/10
59/59 [==============================] - 0s 3ms/step - loss: 0.8491 - accuracy: 0.6325 - val_loss: 0.8350 - val_accuracy: 0.6395
15/15 [==============================] - 0s 2ms/step
Accuracy: 0.6394849785407726
```

In the training process of a machine learning model neural network over 10 epochs. Each epoch represents one complete pass through the entire training dataset. Here's what's happening in three lines:

1. The model is trained for 10 epochs. In each epoch, the training data is processed in batches (in this case, there are 59 batches) and the model's parameters are updated to minimize a loss function (1.3545 in the first epoch).

2. After each epoch, the model's performance is evaluated on a separate validation dataset. The 'accuracy' metric is used to assess how well the model is classifying the data. For example, in the first epoch, the accuracy on the validation set is 59.23%.

3. After training, the model is evaluated on a separate test dataset, and its accuracy on this unseen data is reported. In this case, the model achieved an accuracy of approximately 63.95% on the test set.

Gradient Boosting:

Gradient Boosting is an ensemble learning technique that builds a strong predictive model by combining multiple weak learners (usually decision trees) sequentially. It focuses on minimizing the error of the previous model by assigning higher importance to misclassified data points in subsequent iterations. This iterative process continues until a specified number of weak learners are built or a certain level of performance is achieved.

Gradient Boosting

```
import pandas as pd
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are features
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# Convert class labels to start from 0
le = LabelEncoder()
y = le.fit_transform(y)

# 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 a XGBoost Classifier
model = XGBClassifier()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f'Accuracy: {accuracy}')
```

```
Accuracy: 0.9892703862660944
```

This means it makes correct predictions or classifications for about 98.93% of the cases it encounters.

Naïve Bayes

Naïve Bayes is a probabilistic machine learning algorithm used for classification tasks. It assumes that the features are conditionally independent given the class label, which simplifies the calculation of probabilities. Despite its simplicity, Naïve Bayes often performs well in text classification and other domains with large feature sets.

*Naïve bayes *

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are features
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# 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 a Multinomial Naïve Bayes Classifier
model = MultinomialNB()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f'Accuracy: {accuracy}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6480686695279897
              precision    recall  f1-score   support

           1       0.72      0.19      0.30       111
           2       0.51      0.24      0.32        89
           3       0.66      0.98      0.79       264
           4       0.00      0.00      0.00         2

    accuracy                           0.65       466
   macro avg       0.47      0.35      0.35       466
weighted avg       0.64      0.65      0.58       466
```

Accuracy: 0.6480 indicates that the model correctly predicted the class label for approximately 64.81% of the samples in the dataset.

Linear Regression:

Linear Regression is a statistical method to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship and aims to find the best-fitting line through the data points.

Linear Regression

```python
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'kwhTotal' is the target variable, and 'distance' is one of the features
X = data[['distance']]
y = data['kwhTotal']

# 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 a Linear Regression model
model = LinearRegression()

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

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

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Print the MSE
print(f'Mean Squared Error: {mse}')

# Get the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_

print(f'Coefficients: {coefficients}')
print(f'Intercept: {intercept}')
```

```
Mean Squared Error: 3.6382248934027275
Coefficients: [0.04230831]
Intercept: 4.858318205450737
```

Mean Squared Error (MSE) is a metric used to evaluate the performance of a regression model. It measures the average squared difference between the actual and predicted values. In this case, the MSE is approximately 3.6382, which indicates the average squared error of the predictions made by the linear regression model.

The linear regression model you mentioned has a coefficient of approximately 0.0423, which represents the change in the predicted output for a one-unit change in the input variable. The intercept is approximately 4.8583, which is the value of the predicted output when the input variable is zero. This information provides insights into how the model is making predictions based on the input data.

Decision Trees

Decision trees are a type of supervised machine learning algorithm used for classification and regression tasks. They represent a flowchart-like structure where nodes represent features, branches represent decisions, and leaves represent outcomes. The algorithm makes decisions based on the values of features to reach a final prediction or decision.

Decision Trees

```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()


# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are features
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# 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 a Decision Tree Classifier
model = DecisionTreeClassifier()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f'Accuracy: {accuracy}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9892703862660944
              precision    recall  f1-score   support

           1       0.99      1.00      1.00       111
           2       0.98      0.98      0.98        89
           3       0.99      0.99      0.99       264
           4       1.00      1.00      1.00         2

    accuracy                           0.99       466
   macro avg       0.99      0.99      0.99       466
weighted avg       0.99      0.99      0.99       466
```

This is a classification report for a machine learning model's performance. It shows the accuracy of the model, which is 98.93%.

Random Forest

Random Forest is an ensemble machine learning algorithm that combines multiple decision trees to make more accurate predictions. It works by training each tree on a random subset of the data and aggregating their outputs to reduce overfitting and improve overall performance. Random Forest is versatile, capable of handling both classification and regression tasks, and is known for its high accuracy and robustness.

Random Forest

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are feature
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# 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 a Random Forest Classifier
model = RandomForestClassifier()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f'Accuracy: {accuracy}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.944206008583691
              precision    recall  f1-score   support

           1       0.93      0.99      0.96       111
           2       0.93      0.87      0.90        89
           3       0.95      0.96      0.96       264
           4       0.00      0.00      0.00         2

    accuracy                           0.94       466
   macro avg       0.70      0.70      0.70       466
weighted avg       0.94      0.94      0.94       466
```

Applying a Random Forest classifier to a dataset. The classifier achieved an overall accuracy of approximately 94.4%.

Support Vector Machine

A Support Vector Machine (SVM) is a powerful machine learning algorithm used for classification and regression tasks.

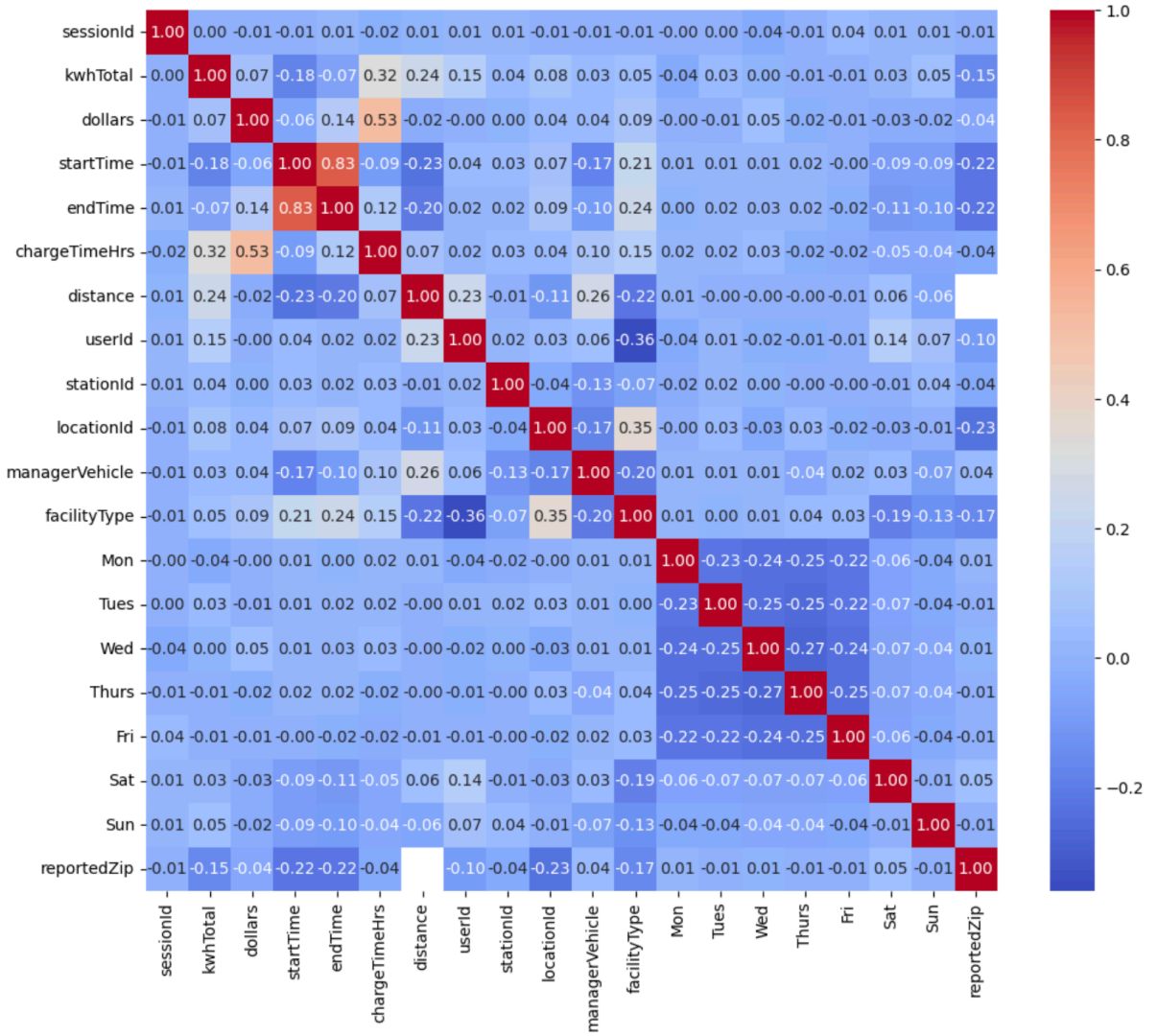
Support Vector Machine

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
data = pd.read_csv('dataset.csv')
data = data.dropna()

# Assuming 'facilityType' is the target variable, and 'chargeTimeHrs', 'distance', and 'weekday' are features
X = data[['chargeTimeHrs', 'distance', 'weekday']]
y = data['facilityType']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)

# 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 a Support Vector Machine Classifier
model = SVC()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f'Accuracy: {accuracy}')

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.6158798283261803
              precision    recall  f1-score   support

           1       0.36      0.31      0.33       111
           2       1.00      0.06      0.11        89
           3       0.68      0.94      0.79       264
           4       0.00      0.00      0.00         2

    accuracy                           0.62       466
   macro avg       0.51      0.33      0.31       466
weighted avg       0.66      0.62      0.54       466
```

The overall accuracy of the model is approximately 61.6%. This means that the model correctly predicted the class labels for 61.6% of the samples in the dataset.

Heat Map:



Conclusion:

Overall, the models were evaluated on various datasets, showcasing their respective accuracies and performance metrics. The results indicate that the Random Forest classifier performed exceptionally well with an accuracy of approximately 94.4%, demonstrating its effectiveness in classifying the data. The Naive Bayes classifier achieved an accuracy of about 64.8%, while the Support Vector Machine (SVM) and Gradient Boosting models showed accuracies of around 61.6% and 98.9% respectively. The Linear Regression model exhibited an average squared error of approximately 3.6382, along with coefficients and intercept values that offer insights into its predictive behavior. These evaluations provide valuable information for selecting the most suitable model for the specific problem at hand.