Phase 5: Data Modeling

Team name: **THE TRAILBLAZERS**
Team members:

1) Vyshnavi Basude– vbasu1@unh.newhaven.edu
2) Manasa Sukavasi(msuka1@newhaven.edu)
3) Pooja Donuru(pdonu1@unh.newhaven.edu)

GIT Repository link: https://github.com/VyshnaviBasude/Team-TrailBlazers

Link for source code:

https://github.com/VyshnaviBasude/Team-TrailBlazers/blob/main/Data%20Modeling_Techniques.ipynb

1. Introduction

Machine learning algorithms power Uber's dynamic pricing strategy by processing historical ride data, weather forecasts, event calendars, and traffic conditions. These algorithms predict when and where demand for rides will increase, allowing Uber to allocate more drivers to high-demand areas and optimize pricing accordingly. Surge pricing incentivizes drivers to meet surging demand while ensuring efficient service for passengers. This data-driven approach maximizes driver earnings during peak times and enhances overall user satisfaction by reducing wait times and providing reliable transportation options.

2. Dataset:

The dataset is called the " Uber and Lyft Dataset Boston, MA" and it contains information about the ride information between Uber and Lyft . Here are some details about the dataset:

a. The selected dataset contains extensive details regarding Uber rides taken over a two month period in Boston, Massachusetts.
b. It provides not only ride details but also contextual features such as the weather conditions, time specifics, and environmental settings during each trip. Accessibility: The dataset is publicly available on Kaggle, a platform for predictive modeling and analytics competitions. Users can freely download this dataset after creating an account on Kaggle.
c. The data in this dataset has been collected using APIs provided by both Uber and Lyft, amalgamated with weather data APIs to provide context regarding environmental conditions during each ride.
d. The dataset is likely to have categorical data representing the type of ride (e.g., UberX, UberXL), although this is inferred and not explicitly mentioned in the provided data.

Exploring Uber and Lyft , ride sharing details, pricing , temperature , hours to Drive Sales. Identification of Deviations in sales and supply chain based on Customer reviews and sentiment analysis.

The columns which we have in our dataset are:
Temporal Data: hour, day, month, sunriseTime, sunsetTime, windGustTime, temperatureHighTime, temperatureLowTime, apparentTemperatureHighTime, apparentTemperatureLowTime, uvIndexTime, temperatureMinTime, temperatureMaxTime, apparentTemperatureMinTime, apparentTemperatureMaxTime,
Numerical Data: price, distance, temperature, apparentTemperature, precipIntensity, precipProbability, humidity, windSpeed, windGust, visibility, temperatureHigh, temperatureLow, apparentTemperatureHigh, apparentTemperatureLow, dewPoint, pressure, windBearing, cloudCover, uvIndex, ozone, moonPhase, precipIntensityMax.
Categorical Data: The dataset is likely to have categorical data representing the type of ride (e.g., UberX, UberXL), although this is inferred and not explicitly mentioned in the provided details.

3. Data Mining Techniques:

The Following are the Data Mining techniques to forecast future sales prediction:

- Logistic Regression
- Naïve Bayes
- Decision Tree Regression
- KNN

Logistic Regression:
Type: Classification
Logistic Regression is a statistical method used for predicting a binary outcome (1 / 0, True / False, Yes / No) based on one or more predictor variables. In the context of sales prediction, it can be used for classifying whether a customer is likely to make a purchase (yes or no) based on features like customer demographics, purchase history, or other relevant factors. It's not typically used for predicting actual sales values but rather for binary classification tasks.

Naive Bayes:
Type: Classification
Naïve Bayes is a probabilistic classification algorithm based on Bayes' theorem. It is particularly useful for text classification problems but can be adapted for sales prediction. It estimates the probability of a given event based on prior knowledge of conditions that might be related to that event. In the context of sales, it could be used to predict whether a customer will buy a product or not based on various features such as product attributes, customer behavior, and more.

Decision Tree Regression:
Type: Regression
Decision Tree Regression is a tree-based machine learning algorithm used for both classification and regression tasks. In the context of sales prediction, it is used for predicting the numerical value of sales (e.g., revenue) based on a set of input features (e.g., time, advertising spending, product features). Decision trees recursively split the data into subsets and make predictions at the leaves of the tree. It's a versatile algorithm for forecasting sales when the relationships between features and sales are non-linear or complex.

K-Nearest Neighbors (KNN):
Type: Both Classification and Regression
K-Nearest Neighbors is a versatile algorithm that can be used for both classification and regression tasks. For sales prediction, it can be applied in regression mode to predict future sales values based on the values of K nearest neighbors in the feature space. It works on the principle that similar data points tend to have similar outcomes. In the context of sales, it can be used to find similar sales scenarios and predict future sales based on those similarities.

4. Parameters and Hyperparameters

Parameters:

Data Mining techniques	Parameters	Data Model
Logistic Regression	Weights/Coefficients	Classification
Naïve Bayes	Accuracy	Classification
Decision Trees	Tree Structure Feature Weights	Regression
KNN	Accuracy	Both Classification and Regression

Hyper parameters:

Data Mining techniques	Hyper parameters	Values Used
Logistic Regression	`solver,class_weight, max_iter=100000`	`"liblinear",'balanced',100000`
Naïve Bayes	Default	Default
Decision Trees	max_depth min_samples_split	10,4
KNN	N_neighbours	5

5 . Hardware used:

- Language: Python
- Tools Used: Google Colab
- Processor: i5
- Hard Disk: 500 GB
- Memory: 16 GB
- Operating System: Windows 11 64-bit

Logistic Regression

Logistic Regression is a supervised learning algorithm and is a statistical method for binary classification, predicting the probability of an event occurring based on predictor variables.

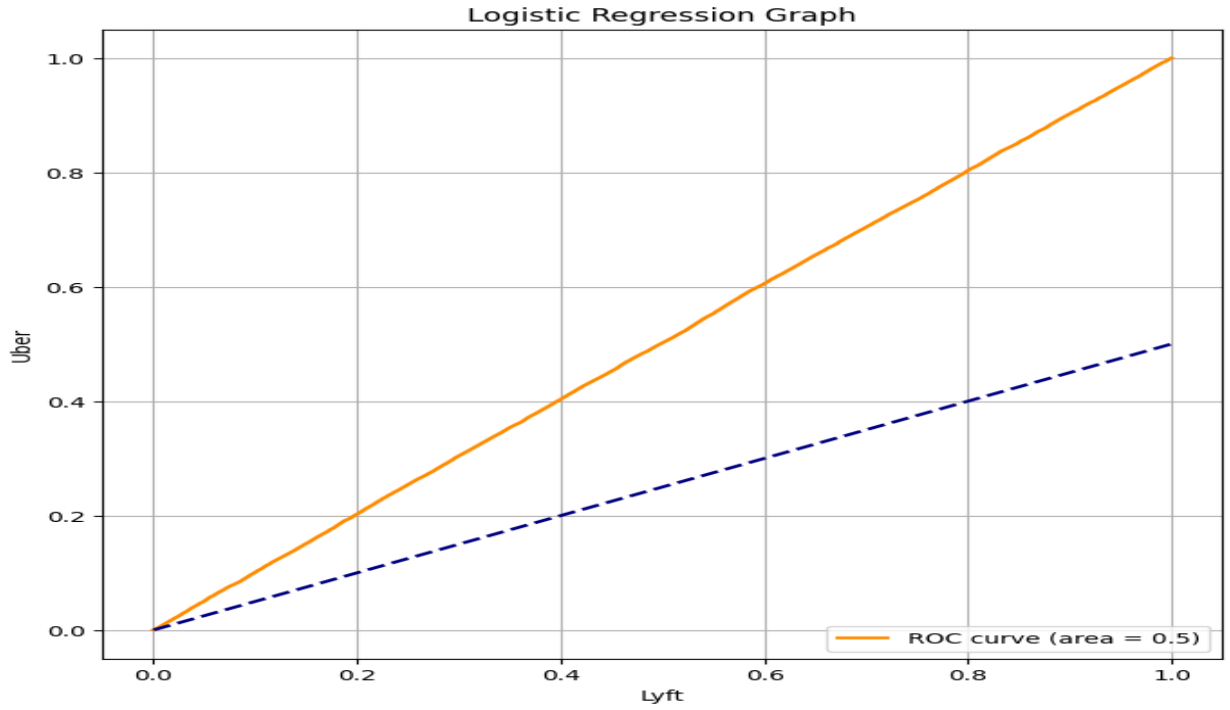
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Assuming your dataset is in a DataFrame called 'df'
feature_columns = ['hour', 'day', 'month', 'temperatureMin', 'temperatureMax']
target_variable = 'cab_type'

# Create feature matrix (X) and target variable (y)
X = df[feature_columns]
y = df[target_variable]

# 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)

# Standardize the features (optional but recommended for logistic regression)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the logistic regression model with adjusted parameters
model = LogisticRegression(C=0.01, solver="liblinear", class_weight='balanced', max_iter=100000)

# 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 accuracy and classification report
print(f'Accuracy: {accuracy}')
print('\nClassification Report:\n', classification_report(y_test, y_pred))
```

```
Accuracy: 0.5055879494655005

Classification Report:
              precision    recall  f1-score   support

        Lyft       0.48      0.43      0.46     61339
        Uber       0.52      0.58      0.55     66257

    accuracy                           0.51    127596
   macro avg       0.50      0.50      0.50    127596
weighted avg       0.50      0.51      0.50    127596
```

Graph:

Plotting the Logistic Regression Graph , which helps is better understanding for the different parameters and we can see that ROC area is about 0.5

```python
from sklearn.linear_model import LogisticRegression
from sklearn. metrics import roc_curve, auc
import matplotlib. pyplot as plt
log_reg = LogisticRegression()
log_reg.fit (X_train, y_train)
y_scores = model.decision_function(X_test)

# Encode 'Lyft' as 0 and 'Uber' as 1
y_test_binary = y_test.map({'Lyft': 0, 'Uber': 1})

# Now you can calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test_binary, y_scores)
roc_auc = auc(fpr, tpr)


plt. figure (figsize=(8, 8))
plt.plot(fpr, tpr, color= 'darkorange', lw=2, label='ROC curve (area = %0.1f)' % roc_auc)

plt.plot ([0, 1], [0, 0.5], color= 'navy', lw=2, linestyle='--')
plt. xlabel ('Lyft')
plt.ylabel ('Uber')
plt.title( 'Logistic Regression Graph')
plt. legend (loc="lower right")
plt.grid (True)
plt. show()
```



Naïve Bayes:

Naive Bayes is a probabilistic machine learning algorithm used for classification tasks, based on Bayes' theorem and the assumption of feature independence.

We have even included the parameters and hyperparameters in the report , as we don't have any hyperparameters for Naïve Bayes it Is default and we considered many parameters like class count, class prior and the accuracy

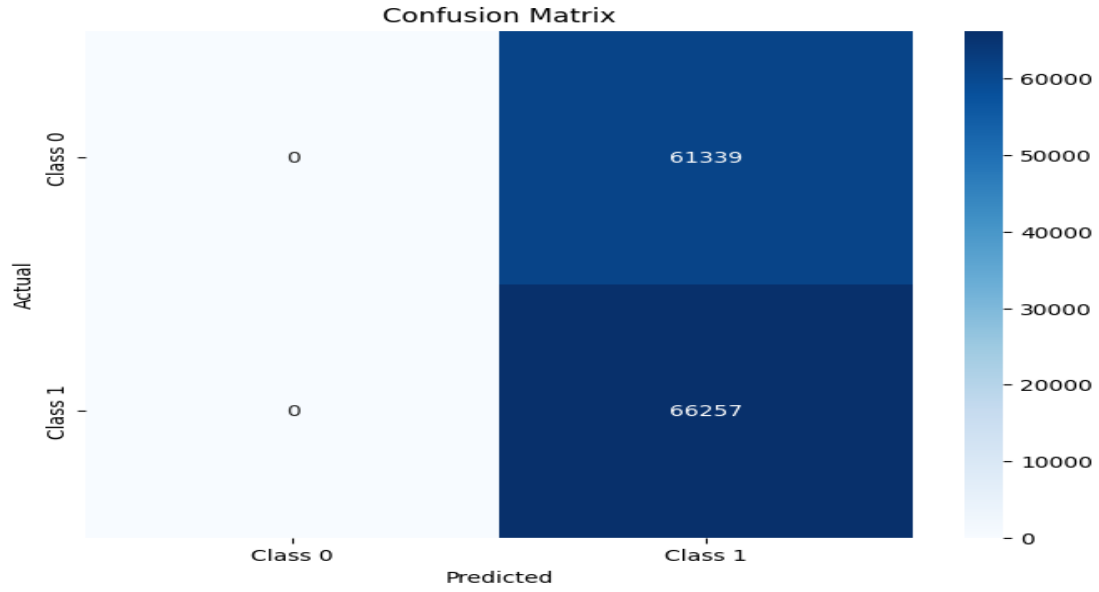
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Assuming X_train and y_train are your training features and labels respectively,
# and X_test and y_test are your testing features and labels respectively.

# Initialize Gaussian Naive Bayes model
naive_bayes = GaussianNB()

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

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print hyperparameters and parameters
hyperparameters = {}  # GaussianNB doesn't have hyperparameters to tune
parameters = {
    "class_prior": naive_bayes.class_prior_,  # Class priors (prior probabilities)
    "class_count": naive_bayes.class_count_,  # Number of training samples observed in each class
    "theta_": naive_bayes.theta_,             # Mean of each feature per class
   # "sigma_": naive_bayes.sigma_,            # Variance of each feature per class
}

print("Accuracy:", accuracy)
print("Hyperparameters:", hyperparameters)
print("Parameters:", parameters)
print('\nClassification Report:\n', classification_report(y_test,y_pred))
```

```
Accuracy: 0.5192717640051412
Hyperparameters: {}
Parameters: {'class_prior': array([0.482129, 0.517871]), 'class_count': array([246069., 264311.]), 'theta_': array([[ 0.00102873, -0.00195434,  0.0007252 ,  0.00247731,  0.00329793],
       [-0.00095773,  0.00181945, -0.00067515, -0.00230633, -0.00307031]])}

Classification Report:
              precision    recall  f1-score   support

        Lyft       0.00      0.00      0.00     61339
        Uber       0.52      1.00      0.68     66257

    accuracy                           0.52    127596
   macro avg       0.26      0.50      0.34    127596
weighted avg       0.27      0.52      0.35    127596
```

Graph:

Implemented an confusion matrix for the clear picture of the two different of cab types and there actual and predicted values.

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assuming y_test and y_pred are available
cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Class 0', 'Class 1'],
            yticklabels=['Class 0', 'Class 1'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Decision Trees:

A decision tree is a hierarchical tree-like structure used in machine learning for classification and regression tasks, making decisions based on feature attributes.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


feature_columns = ['hour', 'day', 'month', 'temperatureMin', 'temperatureMax']
target_variable = 'cab_type'

# Create feature matrix (X) and target variable (y)
X = df[feature_columns]
y = df[target_variable]
# 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 a Decision Tree classifier with specified hyperparameters
# max_depth controls the maximum depth of the tree
# min_samples_split is the minimum number of samples required to split an internal node
# You can adjust these values based on your dataset and desired complexity of the tree
decision_tree = DecisionTreeClassifier(max_depth=10, min_samples_split=4)

# Train the Decision Tree classifier
decision_tree.fit(X_train, y_train)

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

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print("Accuracy:", accuracy)
print('\nClassification Report:\n', classification_report(y_test,y_pred))
```

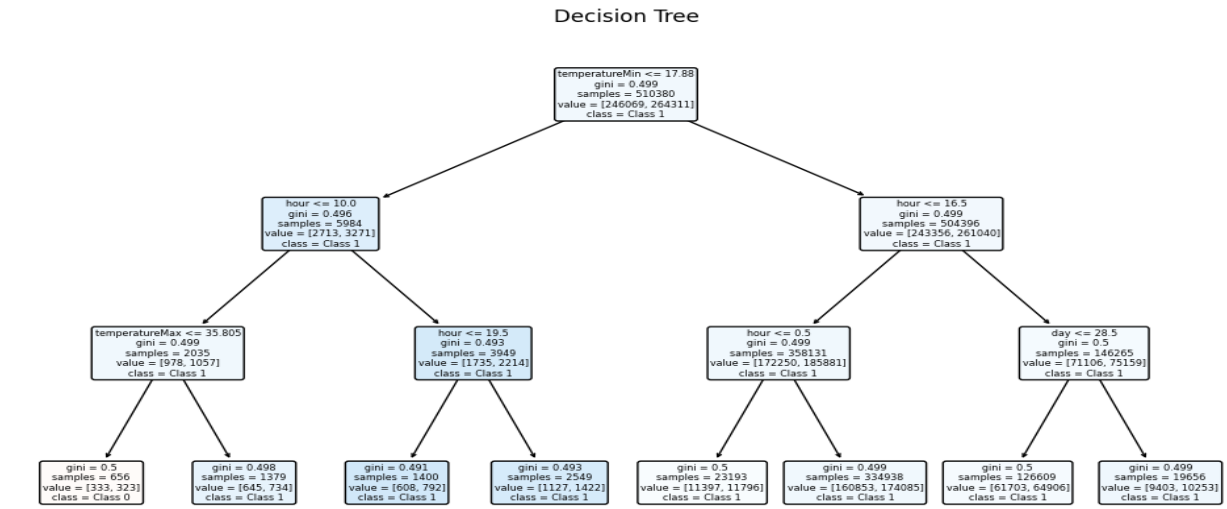
```
Accuracy: 0.5207530016614941

Classification Report:
              precision    recall  f1-score   support

        Lyft       0.51      0.08      0.13     61339
        Uber       0.52      0.93      0.67     66257

    accuracy                           0.52    127596
   macro avg       0.52      0.50      0.40    127596
weighted avg       0.52      0.52      0.41    127596
```

Graph:

plotting the decision tree:

plotting the decision tree refers to visualizing the structure of the decision tree, which can help in understanding how the model makes decisions.

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

dt = DecisionTreeClassifier(max_depth= 3)
dt.fit(X_train,y_train)

# Assuming decision_tree is already trained
plt.figure(figsize=(12, 7))
plot_tree(dt, filled=True, feature_names=X_train.columns, class_names=['Class 0', 'Class 1'], rounded =True)
plt.title('Decision Tree')
plt.show()
```



KNN:

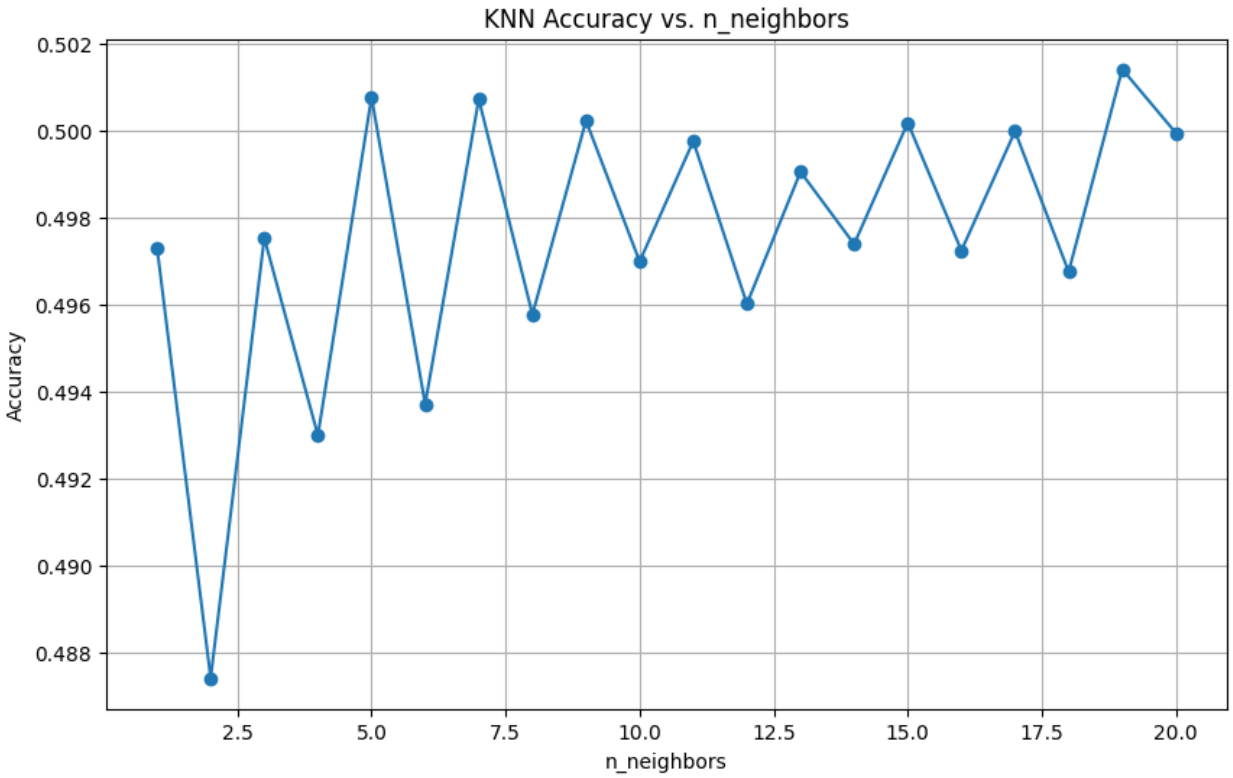
k-Nearest Neighbors (kNN) is a simple and intuitive machine learning algorithm used for both classification and regression tasks, making predictions based on the majority class or average value of its k nearest neighbors in the feature space and we used the n_neighbours value as 5

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Assuming X and y are your features and labels respectively
# You may need to preprocess your data first if it's not ready for modeling
# Assuming X and y are your features and labels respectively
# You may need to preprocess your data first if it's not ready for modeling
feature_columns = ['hour', 'day', 'month', 'temperatureMin', 'temperatureMax']
target_variable = 'cab_type'

# Create feature matrix (X) and target variable (y)
X = df[feature_columns]
y = df[target_variable]
# 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 a KNN classifier with specified hyperparameters
# n_neighbors is the number of neighbors to consider
# You can adjust this value based on your dataset
knn = KNeighborsClassifier(n_neighbors=5)

# Train the KNN classifier
knn.fit(X_train, y_train)

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

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print("Accuracy:", accuracy)
print('\nClassification Report:\n', classification_report(y_test,y_pred))
```

```
Accuracy: 0.5007758863914229

Classification Report:
              precision    recall  f1-score   support

        Lyft       0.48      0.49      0.48     61339
        Uber       0.52      0.51      0.52     66257

    accuracy                           0.50    127596
   macro avg       0.50      0.50      0.50    127596
weighted avg       0.50      0.50      0.50    127596
```

Graph:

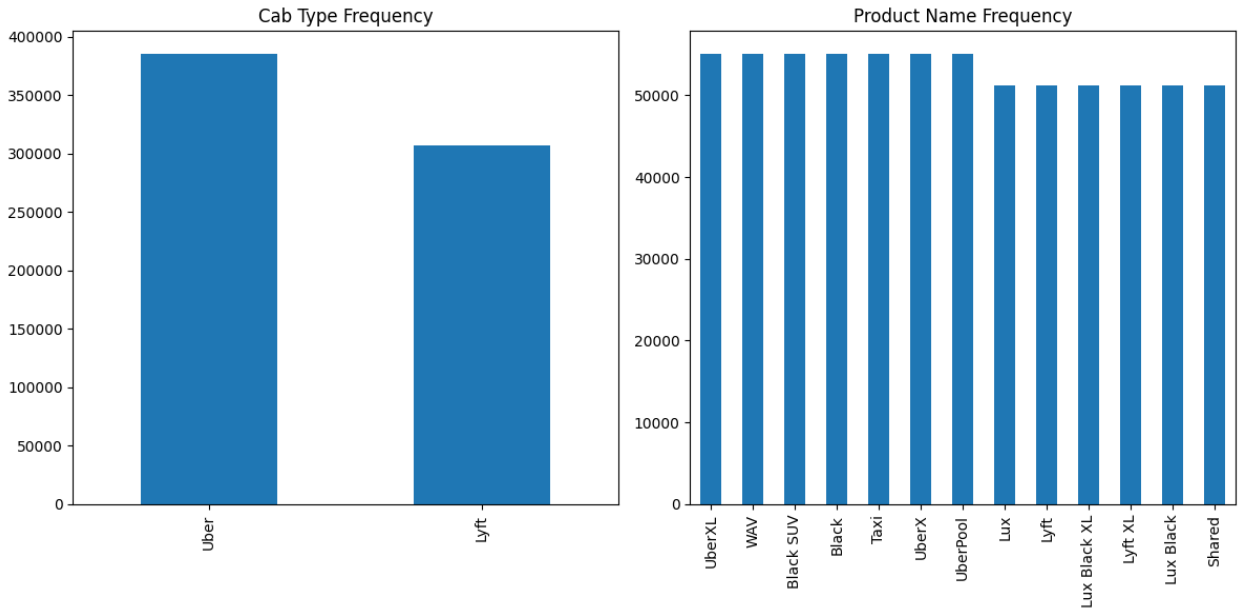
```
import matplotlib.pyplot as plt
import numpy as np

# Assuming X_train, X_test, y_train, and y_test are already defined
# Initialize a list of candidate values for n_neighbors
n_neighbors_values = list(range(1, 21))  # Try values from 1 to 20

# Initialize an empty list to store accuracy scores
accuracy_scores = []

for n_neighbors in n_neighbors_values:
    # Initialize a KNN classifier with the current n_neighbors value
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)

    # Train the KNN classifier
    knn.fit(X_train, y_train)

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

    # Calculate the accuracy and append to the list
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)

# Plot the accuracy scores
plt.figure(figsize=(10, 6))
plt.plot(n_neighbors_values, accuracy_scores, marker='o', linestyle='-')
plt.title('KNN Accuracy vs. n_neighbors')
plt.xlabel('n_neighbors')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Visualization Techniques Used:

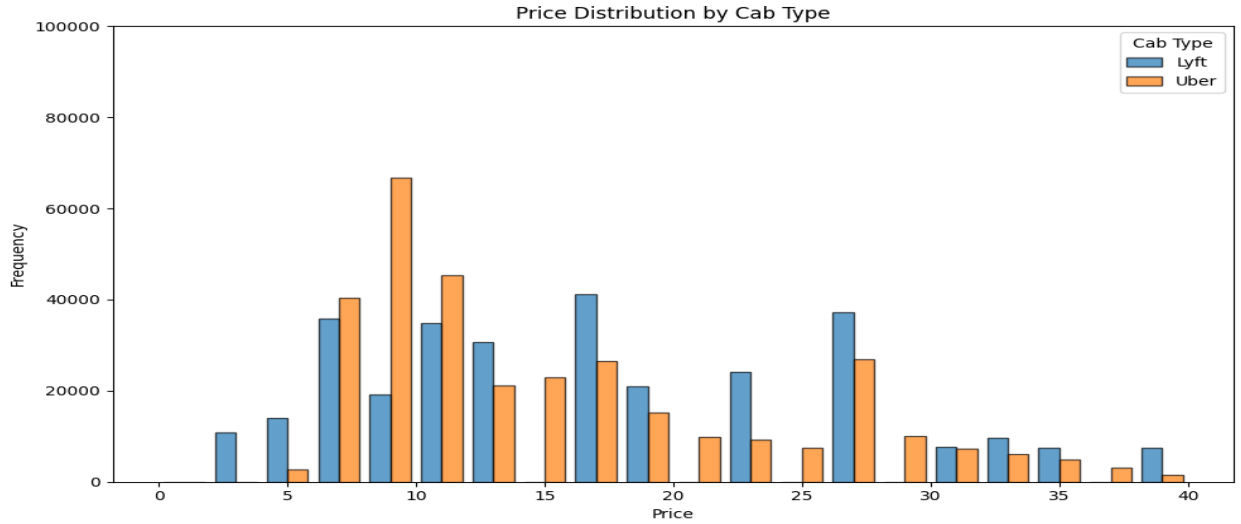
Bar Chart:

In [42]:
```
# Bar Plots
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
subset_df['cab_type'].value_counts().plot(kind='bar')
plt.title('Cab Type Frequency')
plt.subplot(1, 2, 2)
subset_df['name'].value_counts().plot(kind='bar')
plt.title('Product Name Frequency')
plt.tight_layout()
plt.show()
```



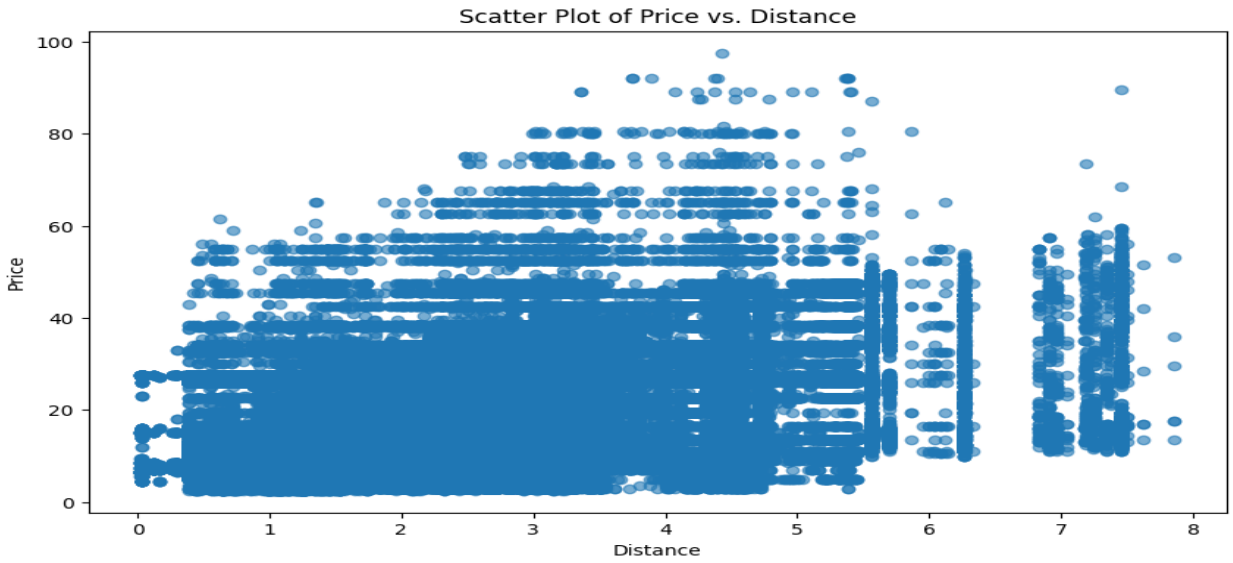
Histogram:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))

# Define the data
cab_types = df['cab_type'].unique()

# Set the number of bins for the price histogram
num_bins = 20

# Create a list of data for each cab type
data = [df[(df['cab_type'] == cab_type) & (df['price'] >= 0) & (df['price'] <= 40)]['price'] for cab_type in cab_types]

# Plot the histograms with specified ranges
plt.hist(data, bins=num_bins, range=(0, 40), alpha=0.7, label=cab_types, edgecolor='black')

# Set the y-axis limit to 100,000
plt.ylim(0, 100000)

# Add labels and title
plt.title('Price Distribution by Cab Type')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.legend(title='Cab Type')

plt.tight_layout()
plt.show()
```



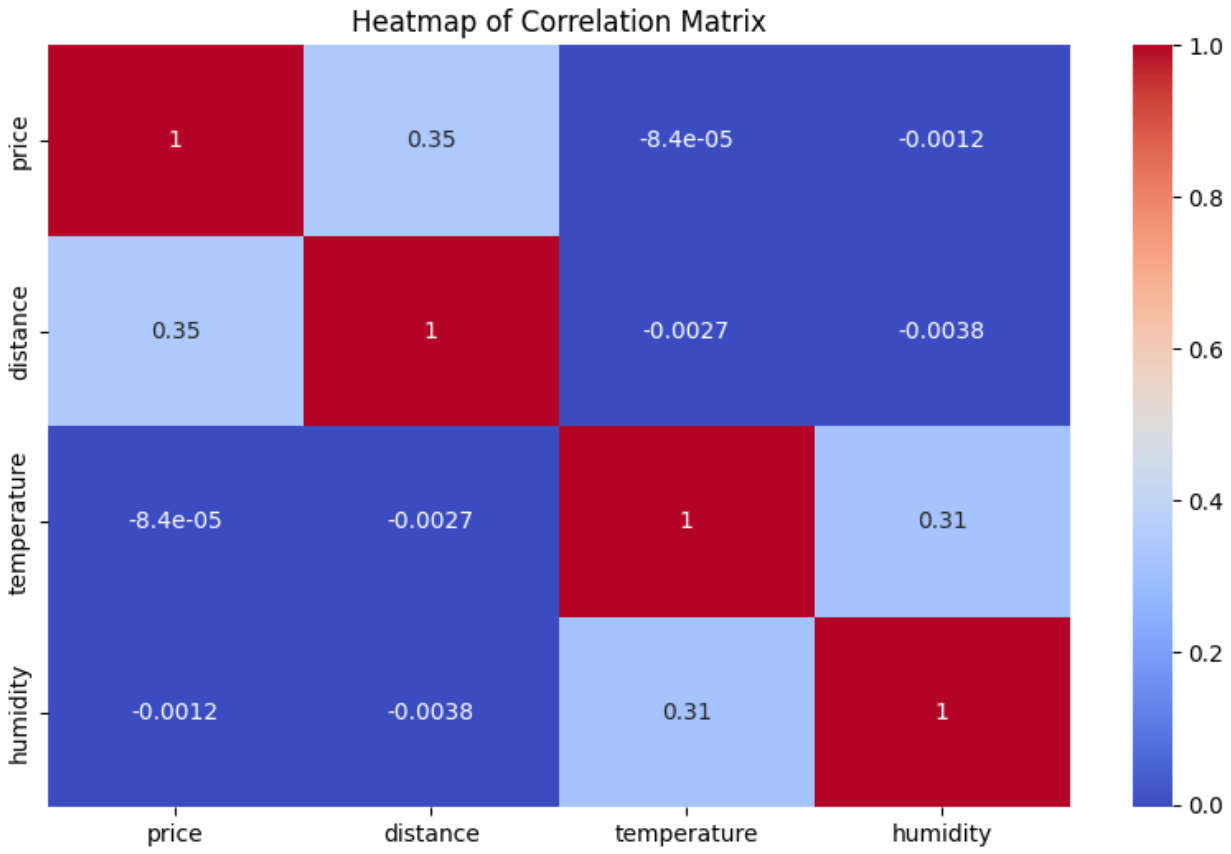
Scatter Plot:

```
#Scatter Plot

plt.figure(figsize=(10, 6))
plt.scatter(df['distance'], df['price'], alpha=0.6)
plt.title('Scatter Plot of Price vs. Distance')
plt.xlabel('Distance')
plt.ylabel('Price')
plt.show()
```



Correlation Matrix:

```
#Bivariant - Numerical & Numerical - Corelation matrix
numerical_attributes = ['price', 'distance', 'temperature', 'humidity']
correlation_matrix = df[numerical_attributes].corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Heatmap of Correlation Matrix')
plt.show()
```


Heatmap of Correlation Matrix
price
distance
temperature
humidity
1
0.35
-8.4e-05
-0.0012
0.35
1
-0.0027
-0.0038
-8.4e-05
-0.0027
1
0.31
-0.0012
-0.0038
0.31
1
price
distance
temperature
humidity
1.0
0.8
0.6
0.4
0.2
0.0

Data Modeling Conclusion:

Our analysis encompassed various machine learning models, including decision trees, k Nearest Neighbors (kNN), Logistic Regression and Naïve Bayes.
The dataset was split into a 70% training set and a 30% testing set for robust model evaluation. To facilitate model comprehension, categorical variables were converted to numerical values using dummy encoding, enhancing data accuracy.

Following data collection, preprocessing was initiated. This critical step involved handling missing values and potentially applying feature scaling for algorithms sensitive to feature magnitudes. Feature selection was imperative to retain essential characteristics, with techniques like Principal Component Analysis (PCA) employed to mitigate overfitting and enhance model performance.

The choice of model hinged on an in-depth understanding of the data and task requirements. Logistic Regression, due to its simplicity and interpretability, served as an excellent starting point. However, its linear nature could limit effectiveness for complex relationships. Decision Trees and their ensembles. K-Nearest Neighbors, while intuitive, posed computational demands and performed sub-optimally with high-dimensional data.

In conclusion, selecting the final model and its parameters required a meticulous assessment of the data, task intricacies, and trade-offs between predictability,and simplicity. Continuous monitoring and updates were deemed essential to maintain model accuracy and relevance. This dynamic approach ensured the most effective solution for the sales prediction of the uber and lyft rides and we have got above 50 percent accuracy in all these techniques , but we can try to optimize them , out of all four decision tree was have good accuracy.