Team

Name	ID
NAWAF ALTHUNAYYAN	201820500
Hamza alhelal	201865160

Table of Contents

Part 1: Data Understanding and Exploration

1. Dataset Overview

Source and Context:
Uber, a global ride-hailing business established in 2009, provides historical trip data in this dataset. It contains anonymized travel logs with an emphasis on trip lengths, classifications (personal vs. business), and objectives..
Description:

- Trip Behavior: The pattern in the lengths and reasons of travel.
- Operational Insights: Useful for urban planning, transportation logistics, and travel analysis.

Problem Domain:

- Rider Behavior: Determining patterns of peak travel.
- Expense management: classifying travel as either personal or professional.
- Service Optimization: Using route and purpose analysis to improve operational efficiency.

2. Feature Description

```
[6]: uber_data.dtypes
```

```
[6]: START_DATE     object
     END_DATE       object
     CATEGORY       object
     START          object
     STOP           object
     MILES         float64
     PURPOSE        object
     dtype: object
```

breakdown of the features in the dataset:

Feature	Data Type	Description	Significance
START_DATE	Object (Datetime)	The date and time when the trip started.	Helps analyze the temporal distribution of trips, such as peak travel times or seasonal trends.
END_DATE	Object (Datetime)	The date and time when the trip ended.	Useful for calculating trip duration and understanding time spent on trips.
CATEGORY	Categorical	Indicates whether the trip was for Business or Personal purposes.	Helps in expense categorization and understanding the context of trips.
START	Categorical	The starting location of the trip.	Useful for identifying common trip origins and travel patterns.

STOP	Categorical	The destination of the trip.	Helps in analyzing common destinations and travel routes.
MILES	Numerical	The distance traveled in miles.	A key metric for understanding trip lengths, which can be used for operational insights like fuel consumption.
PURPOSE	Categorical	The reason for the trip (e.g., Meeting, Errand, Customer Visit).	Provides context for why the trip was taken, aiding in behavior analysis and classification of travel purposes.

Is there a target variable?
the target variable is CATEGORY. It classifies trips as either "Business" or "Personal."

3. Dataset Structure

Size and Structure:

- Number of Rows: **1,156**
- Number of Columns: **7**

```
uber_data = pd.read_csv('UberDataset.csv')
uber_data
```

	START_DATE	END_DATE	CATEGORY	START	STOP	MILES	PURPOSE
0	01-01-2016 21:11	01-01-2016 21:17	Business	Fort Pierce	Fort Pierce	5.1	Meal/Entertain
1	01-02-2016 01:25	01-02-2016 01:37	Business	Fort Pierce	Fort Pierce	5.0	NaN
2	01-02-2016 20:25	01-02-2016 20:38	Business	Fort Pierce	Fort Pierce	4.8	Errand/Supplies
3	01-05-2016 17:31	01-05-2016 17:45	Business	Fort Pierce	Fort Pierce	4.7	Meeting
4	01-06-2016 14:42	01-06-2016 15:49	Business	Fort Pierce	West Palm Beach	63.7	Customer Visit
...	...	...	...	...	...	...	...
1151	12/31/2016 13:24	12/31/2016 13:42	Business	Kar?chi	Unknown Location	3.9	Temporary Site
1152	12/31/2016 15:03	12/31/2016 15:38	Business	Unknown Location	Unknown Location	16.2	Meeting
1153	12/31/2016 21:32	12/31/2016 21:50	Business	Katunayake	Gampaha	6.4	Temporary Site
1154	12/31/2016 22:08	12/31/2016 23:51	Business	Gampaha	Ilukwatta	48.2	Temporary Site
1155	Totals	NaN	NaN	NaN	NaN	12204.7	NaN

1156 rows × 7 columns

4. Missing Values and Duplicates

- Missing value: In the PURPOSE column there are 504 missing values. And for END_DATE, CATEGORY, START, STOP, each one has one missing value.

- Duplicates: And there is one duplicated row

```
#Q4
#Check for missing values in each column
missing_values = uber_data.isnull().sum()
print("Missing Values in Each Column:")
print(missing_values)
# Check for duplicate rows
duplicate_rows = uber_data.duplicated().sum()
print("\nNumber of Duplicate Rows:")
print(duplicate_rows)
```

```
Missing Values in Each Column:
START_DATE      0
END_DATE        1
CATEGORY        1
START           1
STOP            1
MILES           0
PURPOSE       503
dtype: int64

Number of Duplicate Rows:
1
```

This missing values can reduce the effectiveness of analyses.
Duplicates row may effect the accurate of analysis.

5. Statistical Summary

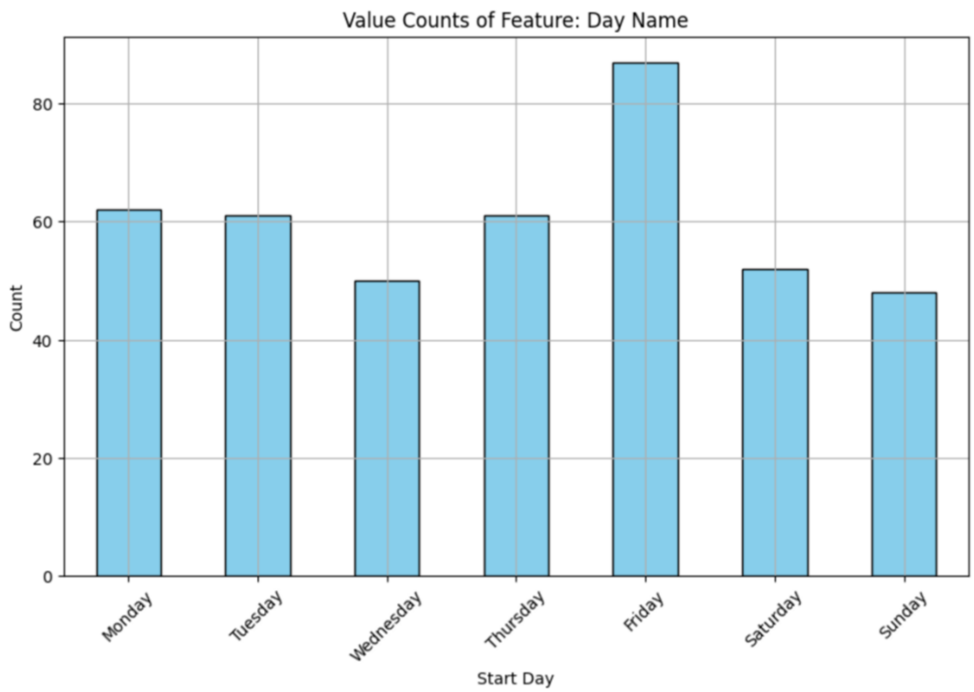
```
#Q5
# Compute summary statistics for numerical features
numerical_summary = uber_data.describe()
# Compute summary statistics for categorical features
categorical_summary = uber_data.describe(include=['object'])
# Print summaries
print("Numerical Summary:")
print(numerical_summary)

print("\nCategorical Summary:")
print(categorical_summary)
```

```
Numerical Summary:
              MILES
count   1156.000000
mean      21.115398
std      359.299007
min        0.500000
25%        2.900000
50%        6.000000
75%       10.400000
max    12204.700000

Categorical Summary:
             START_DATE         END_DATE CATEGORY START  STOP PURPOSE
count              1156             1155     1155  1155  1155     653
unique             1155             1154        2   177   188      10
top     6/28/2016 23:34  6/28/2016 23:59 Business  Cary  Cary Meeting
freq                  2                2     1078   201   203     187
```

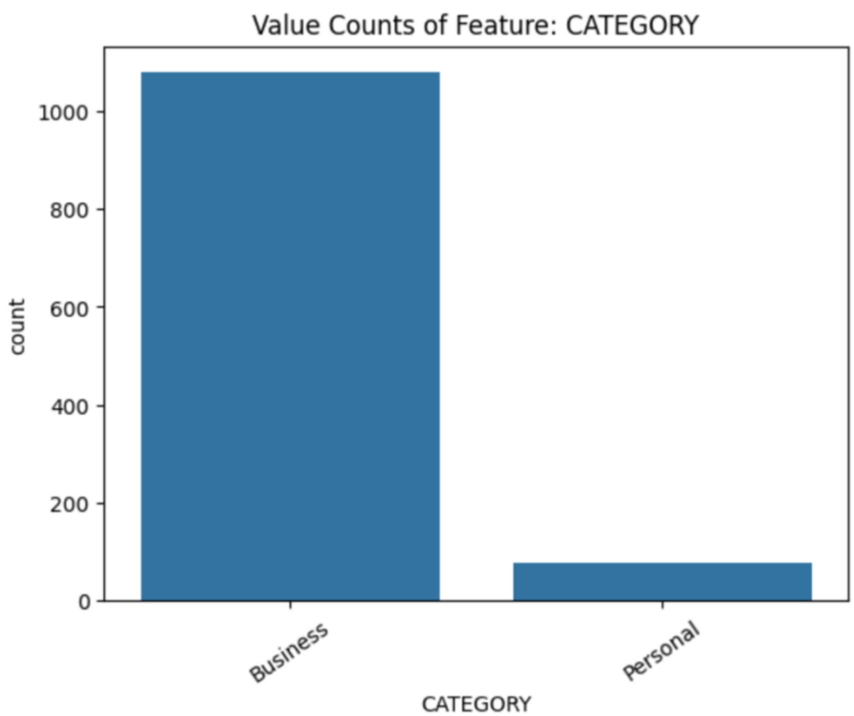
Note: The 50% value in the statistical summary represents the median.

6. Data Distribution

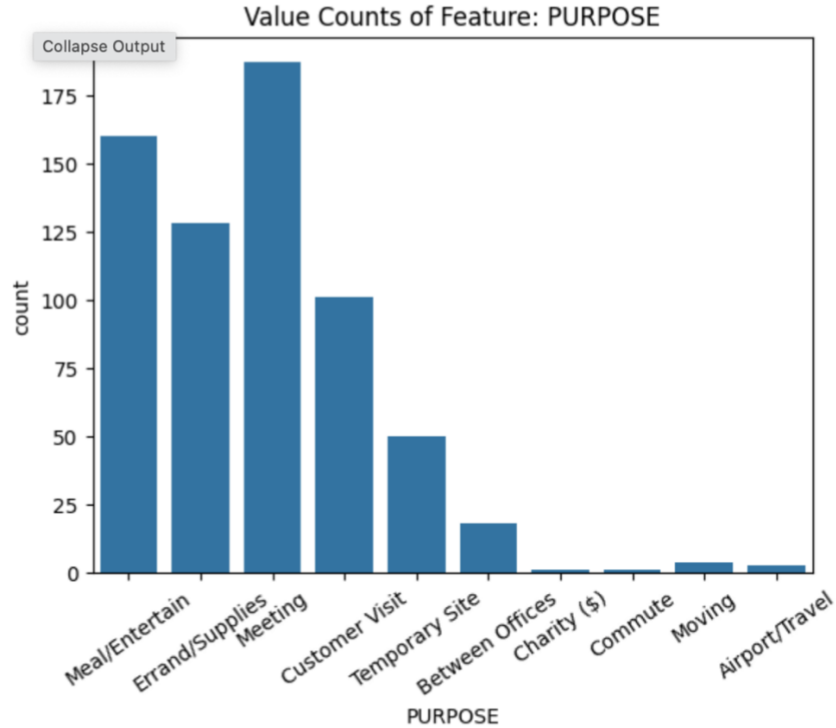
the number of trips per each day:



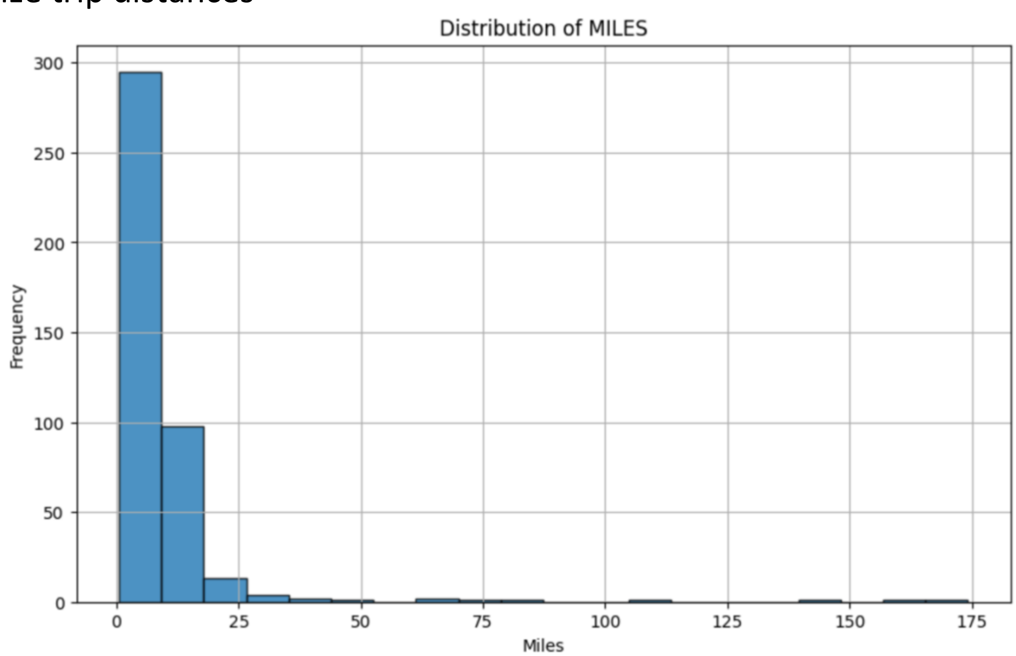
Visualize trip category



Visualize trip purposes



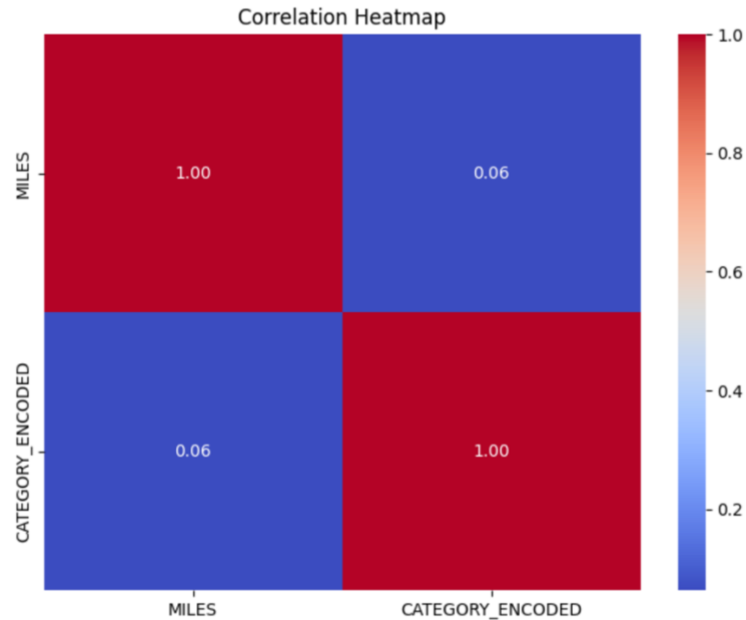
Visualize trip distances



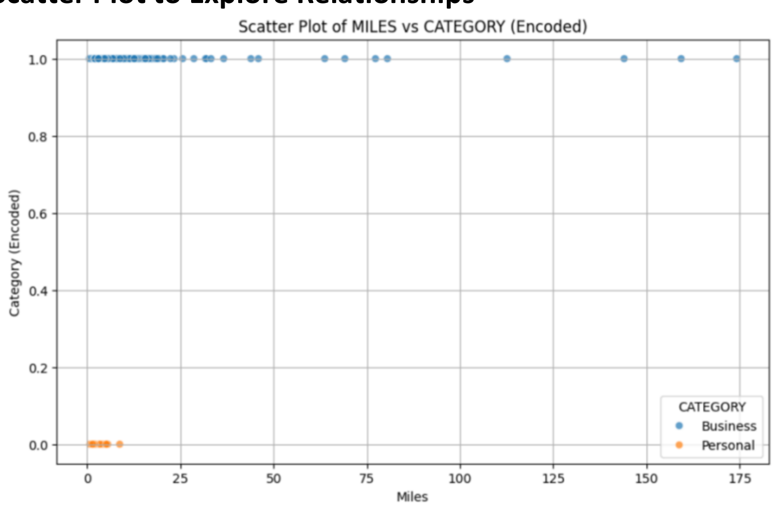
7. Correlation Analysis

The correlation between MILES and CATEGORY is negligible (correlation coefficient 0.02), indicating that miles traveled have little impact on whether a trip is classified as "Business" or "Personal."

Visualize Relationships with a Heatmap



Scatter Plot to Explore Relationships



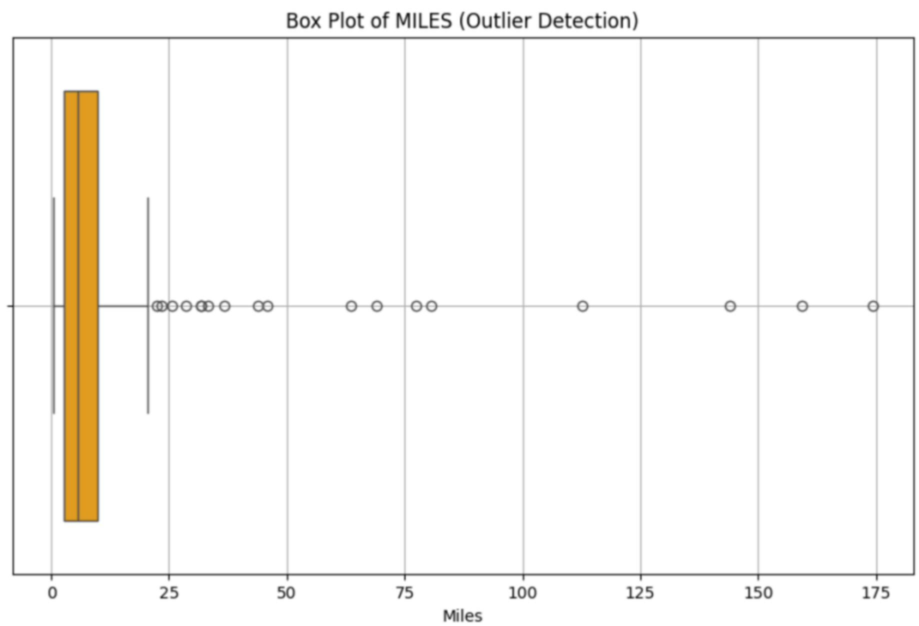
The scatter plot shows that trips categorized as "Business" (encoded as 1) and "Personal" (encoded as 0) mostly overlap in terms of miles traveled, with no clear pattern.

8. Outlier Detection

the analysis confirms that there are **outliers** in the MILES (trip distance) data.

```python
#Q8
# Box plot for MILES
plt.figure(figsize=(10, 6))
sns.boxplot(x=uber_data['MILES'], color='orange')
plt.title('Box Plot of MILES (Outlier Detection)')
plt.xlabel('Miles')
plt.grid(True)
plt.show()


# IQR method for detecting outliers
Q1 = uber_data['MILES'].quantile(0.25)
Q3 = uber_data['MILES'].quantile(0.75)
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = uber_data[(uber_data['MILES'] < lower_bound) | (uber_data['MILES'] > upper_bound)]

# Print results
print(f"Lower Bound: {lower_bound}")
print(f"Upper Bound: {upper_bound}")
print(f"Number of Outliers Detected: {outliers.shape[0]}")
```

Visualize Outliers Using Box Plot




```
Lower Bound: -7.8500000000000005
Upper Bound: 20.55
Number of Outliers Detected: 18
```

Impact of Outliers:

- Outliers can skew the mean and standard deviation, leading to misleading conclusions.
- if building predictive models, outliers may reduce the accuracy and robustness of the model.

- If your analysis involves total trip distances, outliers may distort the results.

Part 2: Data Preprocessing

9. Handling Missing Data

handle missing data:

- Critical columns (CATEGORY, START, STOP): Drop rows with missing values since these are essential for analysis.
- Non-critical columns (PURPOSE): Impute missing values with the most frequent value or leave them as-is depending on their relevance.

```
#Q9
# Drop rows with missing values in critical columns
critical_columns = ['CATEGORY', 'START', 'STOP']
uber_data_cleaned = uber_data.dropna(subset=critical_columns)

# Impute missing values in PURPOSE with the most frequent value
most_frequent_purpose = uber_data_cleaned['PURPOSE'].mode()[0]
uber_data_cleaned['PURPOSE'].fillna(most_frequent_purpose, inplace=True)

# Show the result
print(f"Remaining missing values:\n{uber_data_cleaned.isnull().sum()}")
```

```
Remaining missing values:
START_DATE          0
END_DATE            0
CATEGORY            0
START               0
STOP                0
MILES               0
PURPOSE             0
Day Name            0
CATEGORY_ENCODED    0
dtype: int64
```

For START and STOP variables if we have the value of one of them as Unknown Location and the other is known we replaced the missing value with it and if both are Unknown Location we kept the values as is.

```
def replace_unknown_location(row):
    if row['START'] == "Unknown Location" and row['STOP'] != "Unknown Location":
        row['START'] = row['STOP']
    elif row['STOP'] == "Unknown Location" and row['START'] != "Unknown Location":
        row['STOP'] = row['START']
    return row

# Apply the function to each row in the DataFrame
data = data.apply(replace_unknown_location, axis=1)
```

10. Encoding Categorical Variables

the categorical variables CATEGORY, START, STOP, and PURPOSE need to be encoded.

- CATEGORY: Use label encoding since it's binary (Business/Personal).
- START, STOP, PURPOSE: Use one-hot encoding since they have multiple unique values.

```
#Q10
# Label encode the CATEGORY column
label_encoder = LabelEncoder()
uber_data['CATEGORY_ENCODED'] = label_encoder.fit_transform(uber_data['CATEGORY'])

# One-hot encode the START, STOP, and PURPOSE columns
uber_data_encoded = pd.get_dummies(uber_data, columns=['START', 'STOP', 'PURPOSE'], drop_first=True)

# Show the result
print(uber_data_encoded)
```

```
            START_DATE          END_DATE  CATEGORY  MILES  CATEGORY_ENCODED  \
0     01-01-2016 21:11  01-01-2016 21:17  Business    5.1                 0
1     01-02-2016 01:25  01-02-2016 01:37  Business    5.0                 0
2     01-02-2016 20:25  01-02-2016 20:38  Business    4.8                 0
3     01-05-2016 17:31  01-05-2016 17:45  Business    4.7                 0
4     01-06-2016 14:42  01-06-2016 15:49  Business   63.7                 0
...                ...               ...       ...    ...               ...
1151  12/31/2016 13:24  12/31/2016 13:42  Business    3.9                 0
1152  12/31/2016 15:03  12/31/2016 15:38  Business   16.2                 0
1153  12/31/2016 21:32  12/31/2016 21:50  Business    6.4                 0
1154  12/31/2016 22:08  12/31/2016 23:51  Business   48.2                 0
```

11. Feature Scaling

For Logistic Regression, feature scaling is required to optimize the model's convergence. We'll scale the numerical feature MILES using Standard Scaling. Feature scaling ensures features contribute equally, improves convergence during gradient descent, and leads to better performance with interpretable coefficients.

12. Feature Selection

For predicting CATEGORY (Business vs. Personal), we'll include the following features:

1. MILES: Distance traveled is a key indicator of trip type (e.g., longer trips may correlate with business).
2. PURPOSE: Provides context for the trip's objective, which can strongly indicate its type.
3. START and STOP: Location-based features can provide patterns for personal vs. business trips.

Feature Selection Method:

1. **Manual Selection:** Based on domain knowledge.

Part 3: Modeling

13.Algorithm Selection

The problem is a binary classification task (predicting CATEGORY as Business or Personal). Logistic Regression is chosen because it is well-suited for classification tasks, provides interpretable coefficients that show the relationship between features and the target variable, and is computationally efficient, performing well when the data is linearly separable.

14.Data Splitting

We will use the hold-out method to split the data into training and testing sets.

- Training Set (80%): Used to train the model.
- Testing Set (20%): Used to evaluate the model's performance on unseen data.

The hold-out method is simple and effective for ensuring the model generalizes well.

```
#Q14
# Ensure CATEGORY is scaled
uber_data['CATEGORY_ENCODED'] = uber_data['CATEGORY'].map({'Business': 1, 'Personal': 0})

# Ensure MILES is scaled
scaler = StandardScaler()
uber_data['MILES_STANDARDIZED'] = scaler.fit_transform(uber_data[['MILES']])

# Define features and target
X = uber_data[['MILES_STANDARDIZED']]  # Standardized MILES
y = uber_data['CATEGORY_ENCODED']  # Encoded target

# Split 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)

print(f"Training set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
```

```
Training set size: 924 samples
Testing set size: 232 samples
```

15.Model Training

We will train the Logistic Regression model using the training set. The model will be optimized using the default hyperparameters initially.

Training Process:

1. Use the training set to fit the model.
2. Evaluate the model on the testing set to validate its performance.

```python
#Q15
# Load and preprocess data
uber_data['CATEGORY_ENCODED'] = uber_data['CATEGORY'].map({'Business': 1, 'Personal': 0})

# Drop rows with missing values in features or target
uber_data_cleaned = uber_data.dropna(subset=['MILES', 'CATEGORY_ENCODED'])

# Scale the MILES feature
scaler = StandardScaler()
uber_data_cleaned['MILES_STANDARDIZED'] = scaler.fit_transform(uber_data_cleaned[['MILES']])

# Define features and target
X = uber_data_cleaned[['MILES_STANDARDIZED']]
y = uber_data_cleaned['CATEGORY_ENCODED']

# Split 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)

# Train Logistic Regression model
logistic_model = LogisticRegression(random_state=42)
logistic_model.fit(X_train, y_train)

# Evaluate training performance
y_train_pred = logistic_model.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f"Training Accuracy: {train_accuracy:.2f}")
```

```
Training Accuracy: 0.93
```

16.Model Evaluation

We will use Accuracy, Precision, Recall, and F1-Score to evaluate the model:

- Accuracy: Measures overall correctness.
- Precision: Focuses on correct positive predictions (Business trips).
- Recall: Identifies all actual positives.
- F1-Score: Balances precision and recall.

```python
#Q16
# Predict on the test set
y_test_pred = logistic_model.predict(X_test)

# Calculate and print metrics
print(f"Test Accuracy: {accuracy_score(y_test, y_test_pred):.2f}")
print(f"Precision: {precision_score(y_test, y_test_pred):.2f}")
print(f"Recall: {recall_score(y_test, y_test_pred):.2f}")
print(f"F1-Score: {f1_score(y_test, y_test_pred):.2f}")
```

```
Test Accuracy: 0.94
Precision: 0.94
Recall: 1.00
F1-Score: 0.97
```

17.Performance Analysis

The model's performance on the testing set is summarized as follows:

- Test Accuracy: 94% – The model correctly predicts the trip type (Business vs. Personal) for 94% of the test data.
- Precision: 94% – Of all trips predicted as "Business," 94% are actually business trips.
- Recall: 100% – The model successfully identifies all actual "Business" trips.
- F1-Score: 97% – Indicates a strong balance between precision and recall.

The model achieves complete recall and great accuracy in differentiating between business and personal excursions. This is especially helpful if keeping track of every business travel is essential (for example, for cost reporting). But a little less accuracy indicates that some personal travels might be mistakenly categorized as business, which might be fixed.

18.Model Improvement

To improve performance:

1. Hyperparameter Tuning: Optimize the regularization strength (C) in Logistic Regression.
2. Feature Engineering: Create new features like interactions between MILES and PURPOSE.
3. Algorithm Comparison: Test models like Random Forest or Gradient Boosting.

```
#Q18
# Define parameter grid for Logistic Regression
param_grid = {'C': [0.01, 0.1, 1, 10, 100]}

# Grid search for best hyperparameter
grid_search = GridSearchCV(LogisticRegression(random_state=42), param_grid, cv=5, scoring='f1')
grid_search.fit(X_train, y_train)

# Best parameter and model
best_model = grid_search.best_estimator_
print(f"Best C value: {grid_search.best_params_['C']}")

# Evaluate on test set
y_test_pred = best_model.predict(X_test)
print(f"Improved Test F1-Score: {f1_score(y_test, y_test_pred):.2f}")
```

```
Best C value: 0.01
Improved Test F1-Score: 0.97
```

19.Validation

We use K-Fold Cross-Validation (5-fold) to validate the model's performance. This ensures the model generalizes well by evaluating it on multiple data splits, reducing the risk of overfitting.

```
#Q19
# Perform 5-fold cross-validation
logistic_model = LogisticRegression(C=0.01, random_state=42)
cv_scores = cross_val_score(logistic_model, X, y, cv=5, scoring='f1')

# Print cross-validation results
print(f"Cross-Validation F1-Scores: {cv_scores}")
print(f"Mean F1-Score: {cv_scores.mean():.2f}")
```

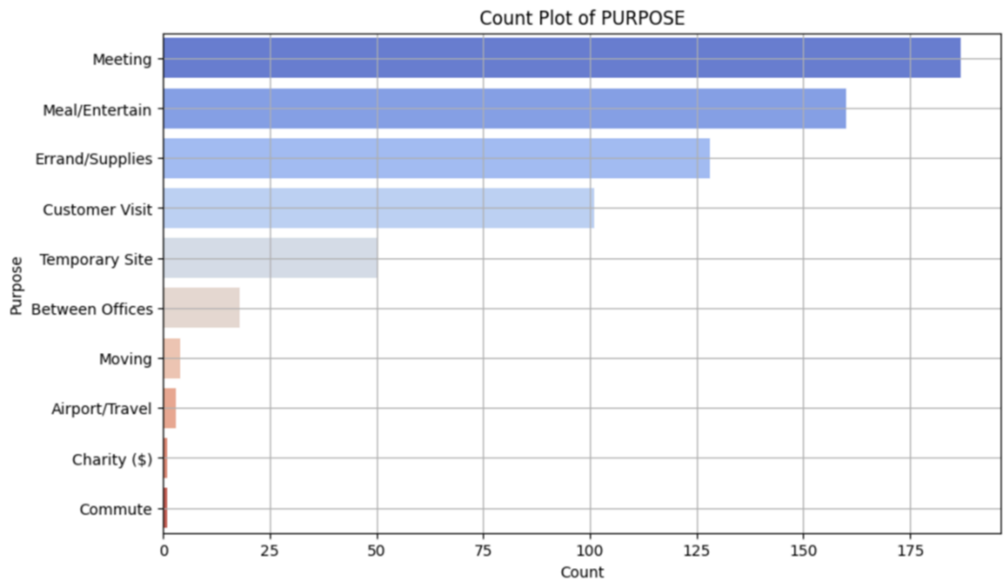
```
Cross-Validation F1-Scores: [0.96644295 0.96644295 0.96644295 0.96412556 0.96412556]
Mean F1-Score: 0.97
```

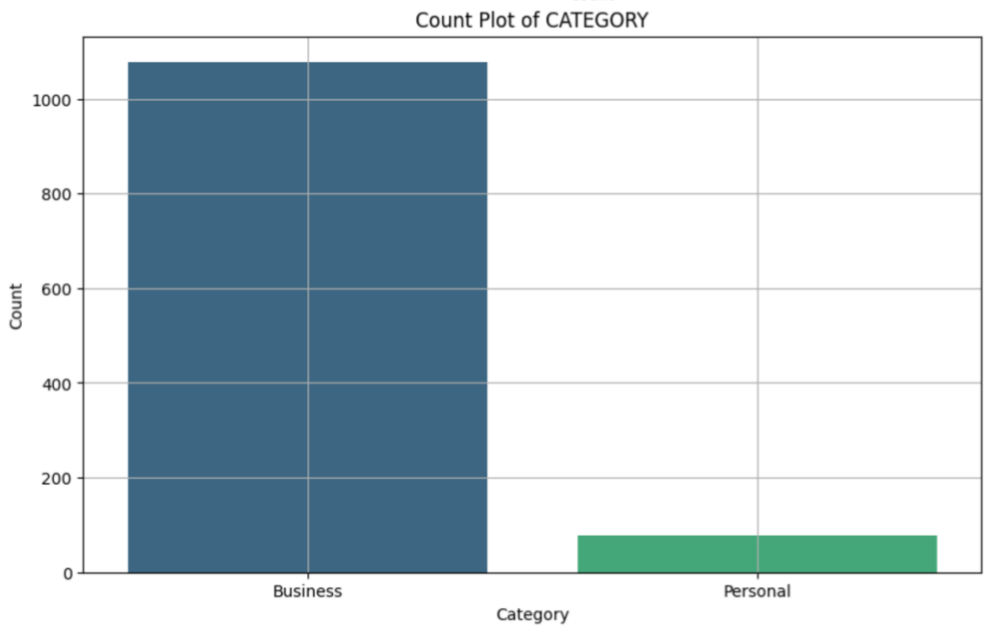
20.Final Model Selection

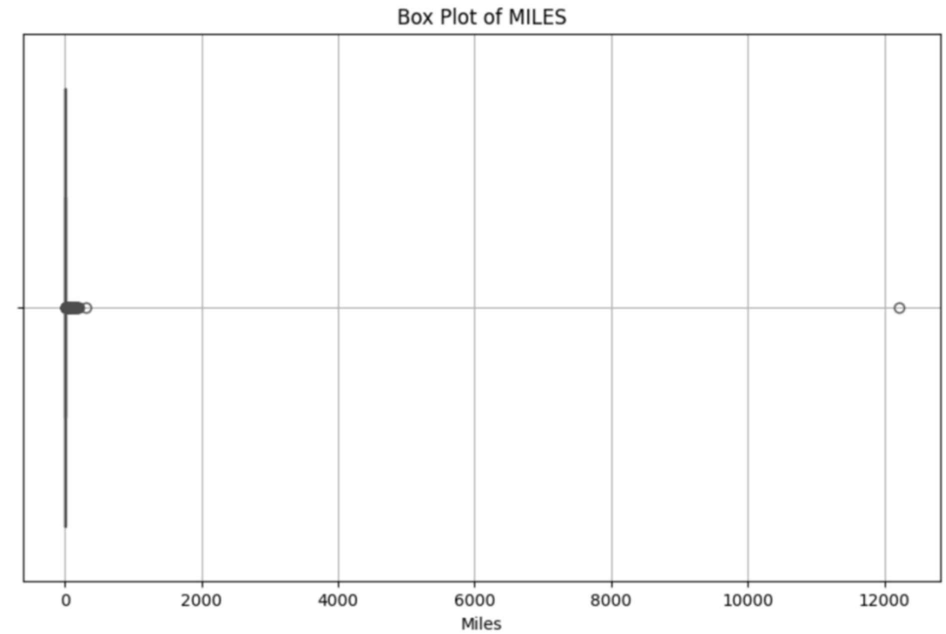
The final model is Logistic Regression due to its high F1-Score (0.97), simplicity, and interpretability. It balances performance and efficiency better than more complex models like Random Forest or SVM.

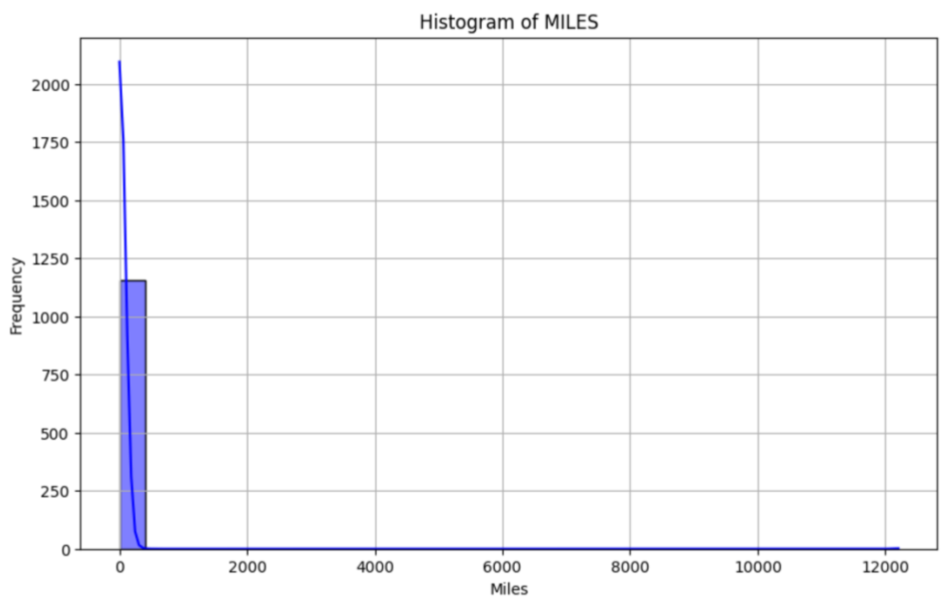
Part 4: Visualization

21.Data Distribution



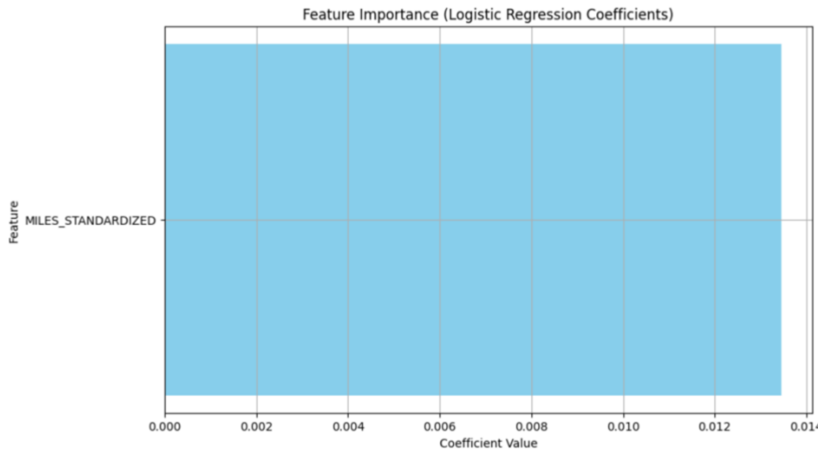



Box Plot of MILES
0
2000
4000
6000
8000
10000
12000
Miles


Histogram of MILES
2000
1750
1500
1250
1000
750
500
250
0
Frequency
0
2000
4000
6000
8000
10000
12000
Miles

22.Feature Importance

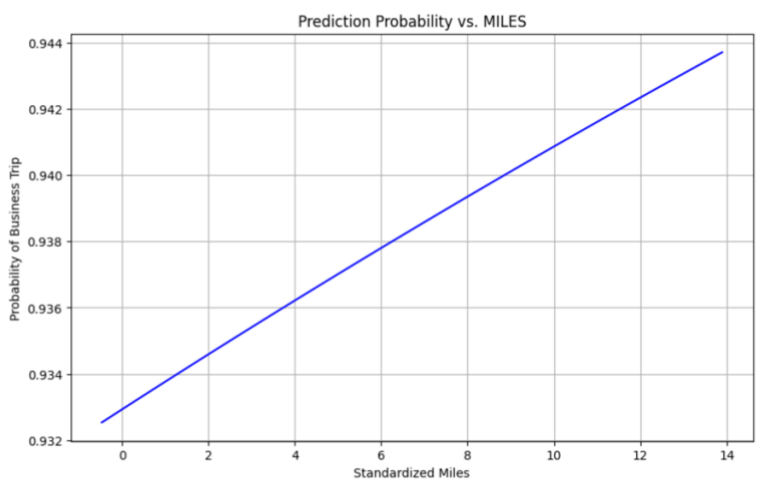
The feature MILES_STANDARDIZED has a positive coefficient, indicating that longer trips are more likely to be classified as Business. As the only feature, it plays a significant role in predicting CATEGORY.

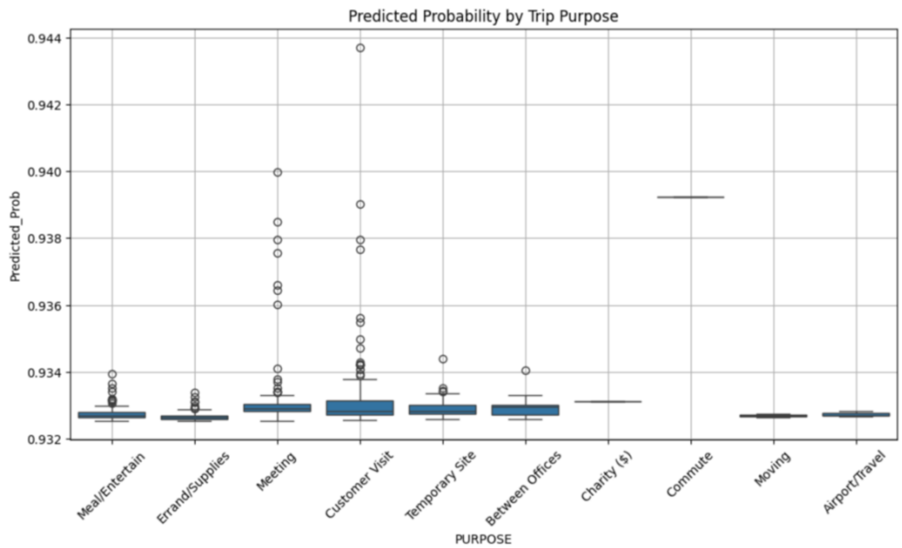


23.Model Performance Across Features

evaluate model performance across features:

1. MILES: Plot the probability of predicting Business as MILES increases.
2. PURPOSE: Use a box plot to show how predictions vary across different trip purposes.





Predicted Probability by Trip Purpose
Predicted_Prob
0.944
0.942
0.940
0.938
0.936
0.934
0.932
Meal/Entertain
Errand/Supplies
Meeting
Customer Visit
Temporary Site
Between Offices
Charity ($)
Commute
Moving
Airport/Travel
PURPOSE