UBER Analysis

The project submitted to the

SRM University – AP, Andhra Pradesh

For the partial fulfillment of the requirements to award the degree of

Bachelor of Technology/Master of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Candidate Name

U. Revanth AP21110010834

V.Sreeroop AP21110010831

I.Vinay Datta AP21110010924

K.Hemanth AP21110010949

B.Rohith kumar AP21110010795

Under the guidance of

Mr. Murali Krishna Enduri

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2024]

Dataset: Uber data set

Introduction: The purpose of this analysis is to explore and understand the characteristics of a uber dataset. The dataset contains various features related to individuals' trip in uber.

ABOUT THE DATASET:

Dataset with taxi trip data that includes a variety of attributes, including the number of passengers, the distance travelled, the time of pickup and drop-off, and the cost of the trip. A single trip is represented by each row. Dataset includes 100000 rows and 19 columns.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# import warnings
# warnings.flterwarnings("ignore")
from sklearn.preprocessing import LabelEncoder
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.metrics import *


data=pd.read_csv("uber_data.csv")
data


# Convert categorical variables to numerical format
label_encoder = LabelEncoder()
data['VendorID'] = label_encoder.fit_transform(data['VendorID'])
data['RatecodeID'] = label_encoder.fit_transform(data['RatecodeID'])
data['store_and_fwd_flag'] =
label_encoder.fit_transform(data['store_and_fwd_flag'])
data['payment_type'] =
label_encoder.fit_transform(data['payment_type'])
data['tpep_pickup_datetime'] =
pd.to_datetime(data['tpep_pickup_datetime'])
data['tpep_dropoff_datetime'] =
pd.to_datetime(data['tpep_dropoff_datetime'])

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29999 entries, 0 to 29998
Data columns (total 19 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               29999 non-null  int64
 1   tpep_pickup_datetime   29999 non-null  datetime64[ns]
 2   tpep_dropoff_datetime  29999 non-null  datetime64[ns]
 3   passenger_count        29999 non-null  int64
 4   trip_distance          29999 non-null  float64
 5   pickup_longitude       29999 non-null  float64
 6   pickup_latitude        29999 non-null  float64
 7   RatecodeID             29999 non-null  int64
 8   store_and_fwd_flag     29999 non-null  int32
 9   dropoff_longitude      29999 non-null  float64
 12  fare_amount            29999 non-null  float64
 13  extra                  29999 non-null  float64
 14  mta_tax                29999 non-null  float64
 15  tip_amount             29999 non-null  float64
 16  tolls_amount           29999 non-null  float64
 17  improvement_surcharge  29999 non-null  float64
 18  total_amount           29999 non-null  float64
dtypes: datetime64[ns](2), float64(12), int32(2), int64(3)
memory usage: 4.1 MB

# Visualizing the distribution of numerical variables
sns.pairplot(data)
plt.show()
```

1. Justify how the problem fits to the data science application.

Predictive Modelling: The data scientists can therefore build models that are predictive by using this dataset. For example, predicting how much to charge for a trip by using features such as trip distance, pickup/drop-off address, etc. It may help taxi companies to improve their pricing strategies and to provide the customers with price estimates from this.

Route Optimization: Through analysing historical trip data, data scientists shall have a capacity of identifying the roads with high traffic frequently and congested spots. This may be utilized to build-the-best taxi routes, management the travel time and eventually achieve the best result through the successful execution of optimization strategies.

Customer Segmentation: Monitoring of boarding percentage, trip duration, and other demographic specifications can be used to differentiate passengers according to their changeable preferences and behaviour. It can give the taxi companies tools to identify clientele and apply decision-making features to their marketing campaigns.

Anomaly Detection: Abnormalities in taxi trip data like unusually longer trips, peculiar patterns of pickup/drop off locations or fare errors could be spotted to detect fraudulent activities like criminal acts or some operational issues.

Demand Prediction: Historicity and external influence of factors like weather, events and holidays can be taken care of by data scientists who can forecast the demands in different intervals and localities accordingly. This can help to take taxes companies to the point of allocation and planning.

Geospatial Analysis: Such local coordinates which mark the pickup/drop-off locations can be employed in the context of geospatial analysis to understand the spatial patterns, pinpoint the hubs, and evaluate the correlation of the geography with taxi trips.

Operational Efficiency: Factors like the duration of the trips, the time when drivers are idling, and driver behaviour can be analysed to help take the right decisions regarding the fleet management and so improve the overall operational efficiency.

Basically, the whole set of data results in an incredible resource which can be used to instrument many data science applications to help improve taxi services, to optimize operations and to boost a customer experience.

2. Perform Exploratory Data Analysis

Import all the libraries

pandas (import pandas as pd): Pandas is a powerful data manipulation and analysis library in Python. It provides data structures like DataFrames and Series that are particularly useful for handling structured data. Common operations include data cleaning, transformation, filtering, and aggregation.

NumPy (import numpy as np): NumPy is a fundamental library for numerical computations in Python. It provides support for arrays, matrices, and mathematical functions that are essential for tasks like numerical simulations, linear algebra operations, and statistical analysis.

Matplotlib (import matplotlib.pyplot as plt): Matplotlib is a plotting library in Python used to create static, animated, and interactive visualizations. It offers a wide range of plotting functions to generate line plots, bar charts, histograms, scatter plots, and more.

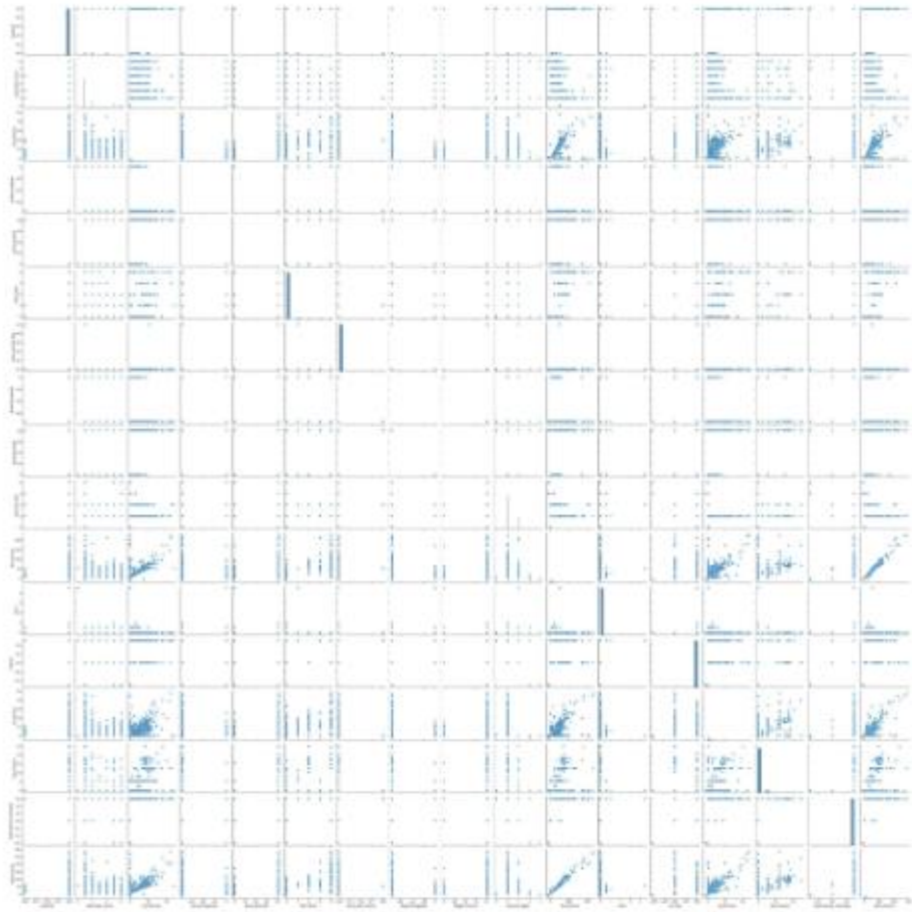
Seaborn (import seaborn as sns): Seaborn is built on top of Matplotlib and provides a higherlevel interface for creating attractive statistical graphics. It simplifies the process of generating complex visualizations such as heatmaps, pair plots, violin plots, and more, with added customization options.

EDA (Exploratory Data Analysis):

"EDA" stands for Exploratory Data Analysis. It's an approach to analyzing datasets to summarize their main characteristics, often employing visual methods. EDA is a crucial step in the data analysis process as it helps to understand the data, find patterns, detect anomalies, and formulate hypotheses for further investigation.

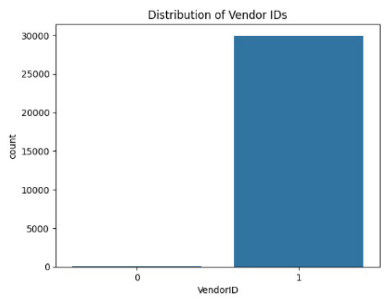
Following are some EDA processes:

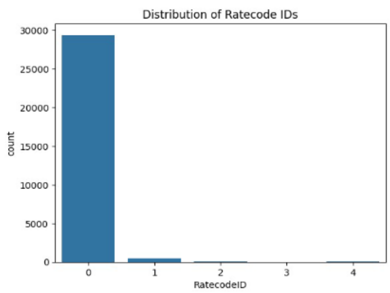
```python
# Visualizing the distribution of numerical variables
sns.pairplot(data)
plt.show()
```

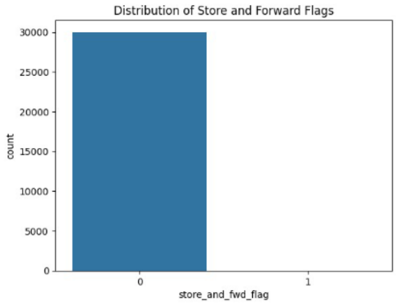


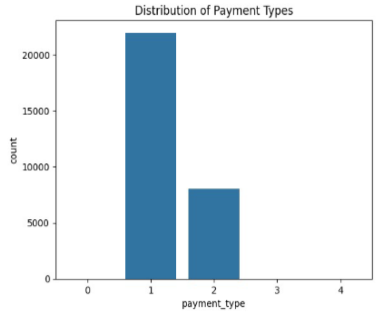
```python
# Visualizing the distribution of categorical variables
sns.countplot(x='VendorID', data=data)
plt.title('Distribution of Vendor IDs')
plt.show()

sns.countplot(x='RatecodeID', data=data)
plt.title('Distribution of Ratecode IDs')
plt.show()

sns.countplot(x='store_and_fwd_flag', data=data)
plt.title('Distribution of Store and Forward Flags')
plt.show()
```

```python
sns.countplot(x='payment_type', data=data)
plt.title('Distribution of Payment Types')
plt.show()
```

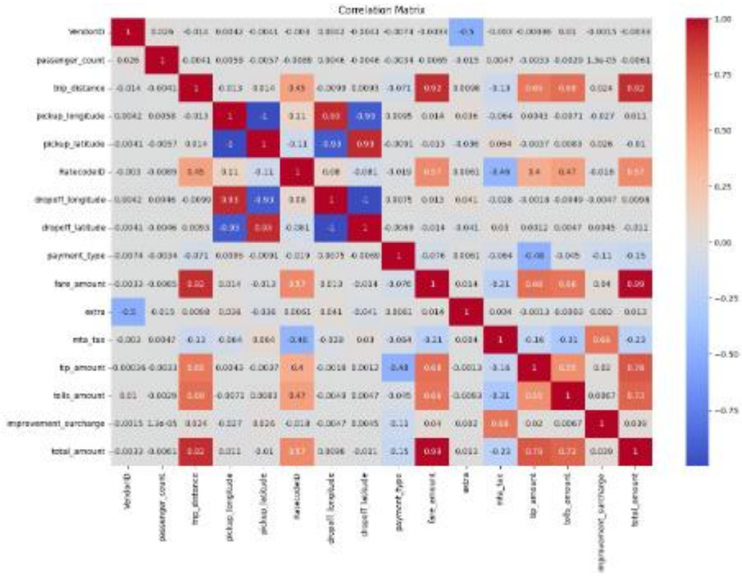









```python
# Correlation matrix
correlation_matrix = data.corr()
fig, ax =plt.subplots(figsize=(15,10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



3. Perform Pre-processing

- handling null values is a fundamental step in data preprocessing. Missing data can have a significant impact on the results of any analysis or modeling tasks. Therefore, it's crucial to address null values before proceeding with further analysis.
- Handling duplicates is another important step in data preprocessing. Duplicates can arise in datasets for various reasons, such as data entry errors, merging datasets, or collecting multiple observations of the same entity.

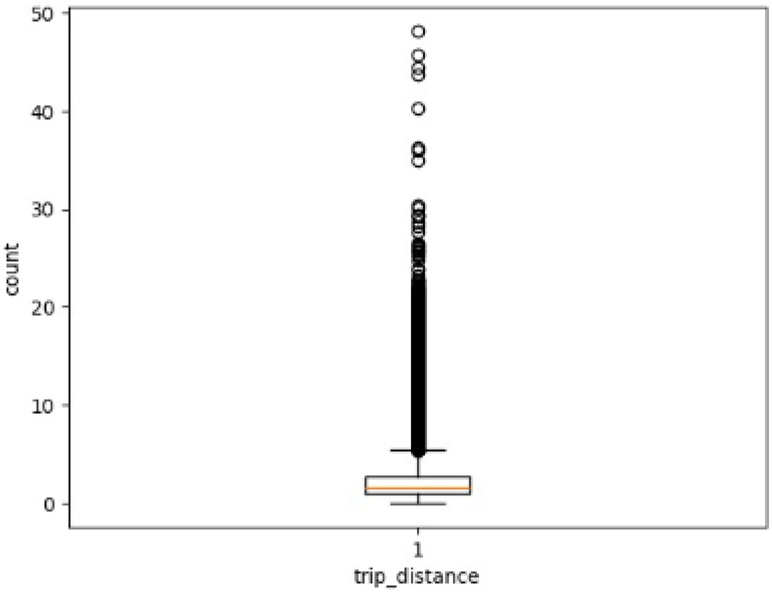
```
data=data.drop(columns=["tpep_pickup_datetime","tpep_dropoff_datetime"
,"store_and_fwd_flag"])

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29999 entries, 0 to 29998
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               29999 non-null  int64
 1   passenger_count        29999 non-null  int64
 2   trip_distance          29999 non-null  float64
 3   pickup_longitude       29999 non-null  float64
 4   pickup_latitude        29999 non-null  float64
 5   RatecodeID             29999 non-null  int64
 6   dropoff_longitude      29999 non-null  float64
 7   dropoff_latitude       29999 non-null  float64
 8   payment_type           29999 non-null  int32
 9   fare_amount            29999 non-null  float64
 10  extra                  29999 non-null  float64
 11  mta_tax                29999 non-null  float64
 12  tip_amount             29999 non-null  float64
```

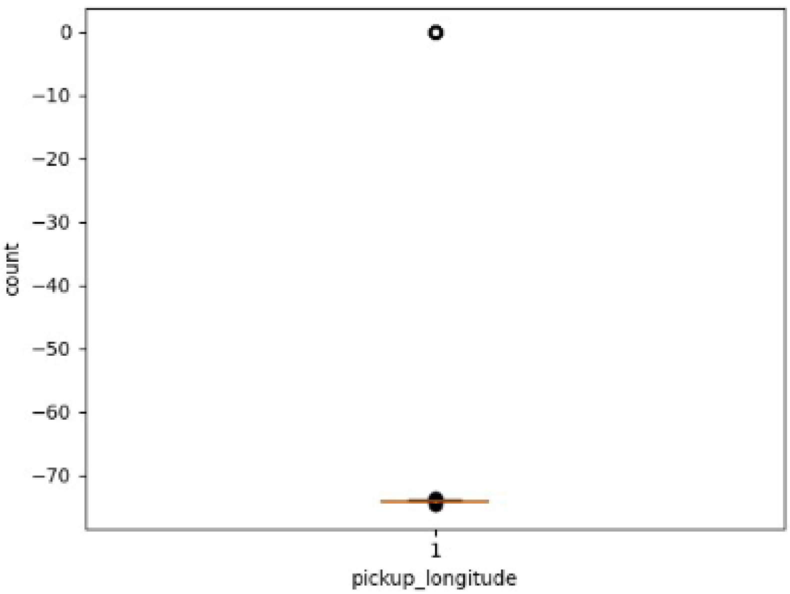
```
 13  tolls_amount           29999 non-null  float64
 14  improvement_surcharge  29999 non-null  float64
 15  total_amount           29999 non-null  float64
dtypes: float64(12), int32(1), int64(3)
memory usage: 3.5 MB

# count of nulls
data.isnull().sum()

VendorID                 0
passenger_count          0
trip_distance            0
pickup_longitude         0
pickup_latitude          0
RatecodeID               0
dropoff_longitude        0
dropoff_latitude         0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64

# count of duplicates
data.duplicated().sum()

6

# removing duplicates
data.drop_duplicates(inplace=True)

# count of duplicates
data.duplicated().sum()

0

# shape of dataframe after removing duplicates
data.shape

(29993, 16)

# Statiscal summary
data.describe().T

                       count      mean       std       min
25%  \
VendorID             29993.0  0.997733  0.047562  0.000000
1.000000
```

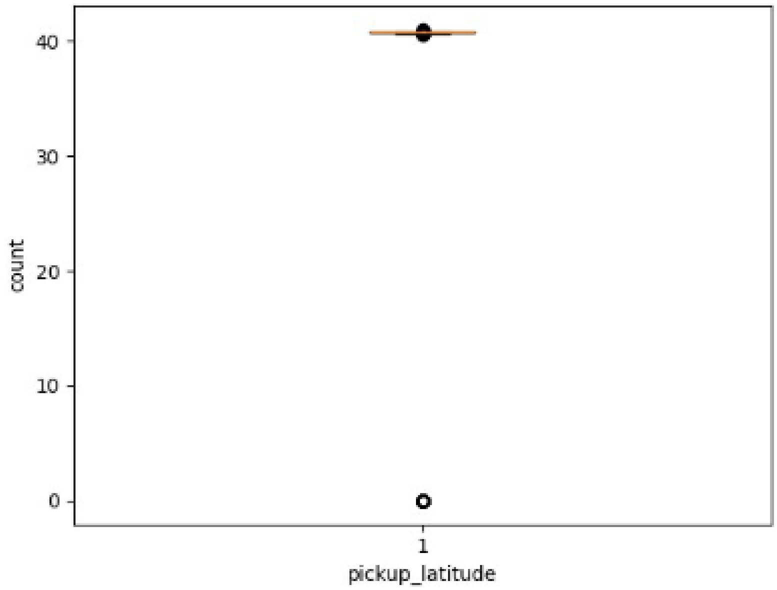
```
passenger_count        29993.0   2.021105   1.654525   0.000000
1.000000
trip_distance          29993.0   2.655719   3.272309   0.000000
0.980000
pickup_longitude       29993.0 -73.396366   6.508575 -74.651306 -
73.990837
pickup_latitude        29993.0  40.437290   3.585912   0.000000
40.741543
RatecodeID             29993.0   0.028973   0.239603   0.000000
0.000000
dropoff_longitude      29993.0 -73.403348   6.481114 -74.651306 -
73.990677
dropoff_latitude       29993.0  40.437886   3.570496   0.000000
40.741112
payment_type           29993.0   1.269196   0.446023   0.000000
1.000000
fare_amount            29993.0  12.304743  10.138337  -7.500000
6.500000
extra                  29993.0   0.002884   0.045283   0.000000
0.000000
mta_tax                29993.0   0.497583   0.038335  -0.500000
0.500000
tip_amount             29993.0   1.860362   2.347918  -2.340000
0.000000
tolls_amount           29993.0   0.299876   1.400675   0.000000
0.000000
improvement_surcharge  29993.0   0.299510   0.015583  -0.300000
0.300000
total_amount           29993.0  15.264930  12.838917 -10.140000
8.300000

                             50%        75%         max
VendorID                1.000000   1.000000    1.000000
passenger_count         1.000000   2.000000    6.000000
trip_distance           1.600000   2.730000   48.180000
pickup_longitude      -73.978836 -73.962112    0.000000
pickup_latitude        40.756470  40.772385   41.051239
RatecodeID              0.000000   0.000000    4.000000
dropoff_longitude     -73.978828 -73.968384    0.000000
dropoff_latitude       40.755589  40.766026   41.051239
payment_type            1.000000   2.000000    4.000000
fare_amount             9.500000  14.000000  225.000000
extra                   0.000000   0.000000    4.500000
mta_tax                 0.500000   0.500000    0.500000
tip_amount              1.460000   2.390000   47.560000
tolls_amount            0.000000   0.000000   25.000000
improvement_surcharge   0.300000   0.300000    0.300000
total_amount           11.620000  16.800000  285.360000
```

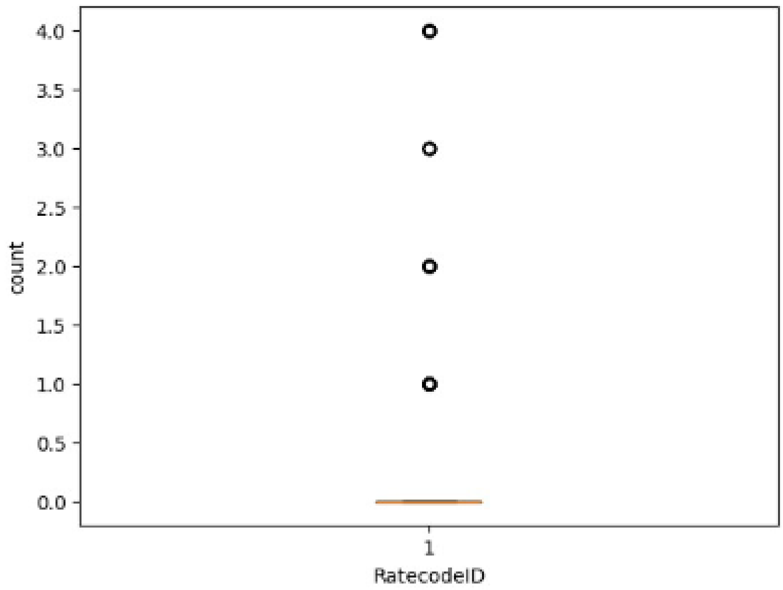
- Detect and handle outliers in the data. Outliers can skew statistical analysis and modelling results. Depending on the context, outliers can be removed, transformed, or treated separately.

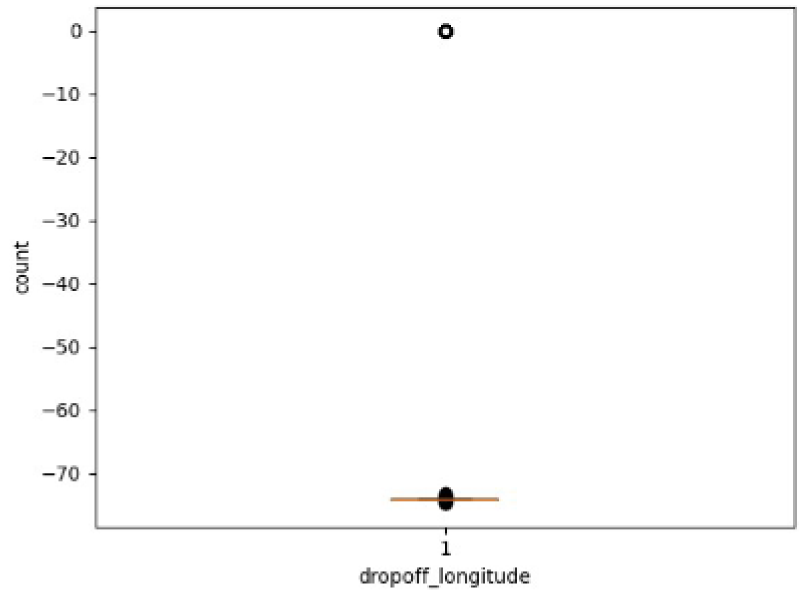
```
# outliers::
for i in data.columns:
    if i!="total_amount":
     plt.boxplot(data[i])
     plt.xlabel(i)
     plt.ylabel("count")
     plt.show()
```

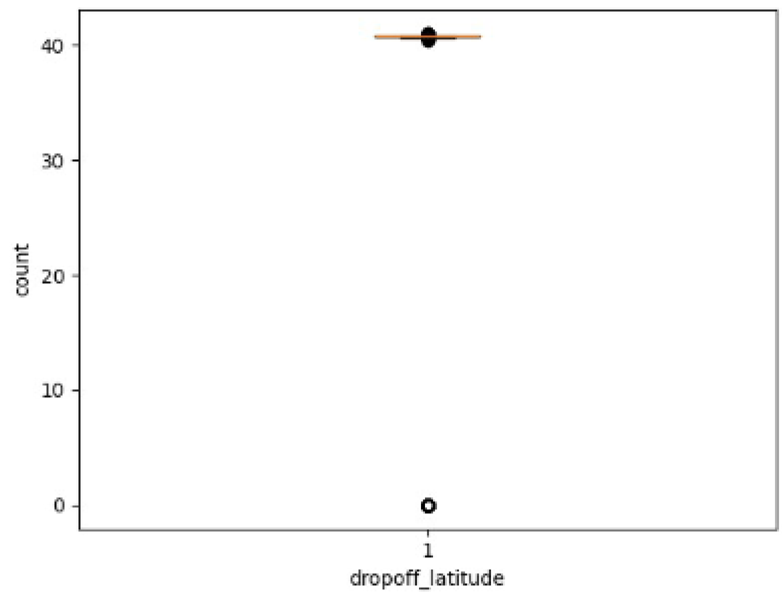


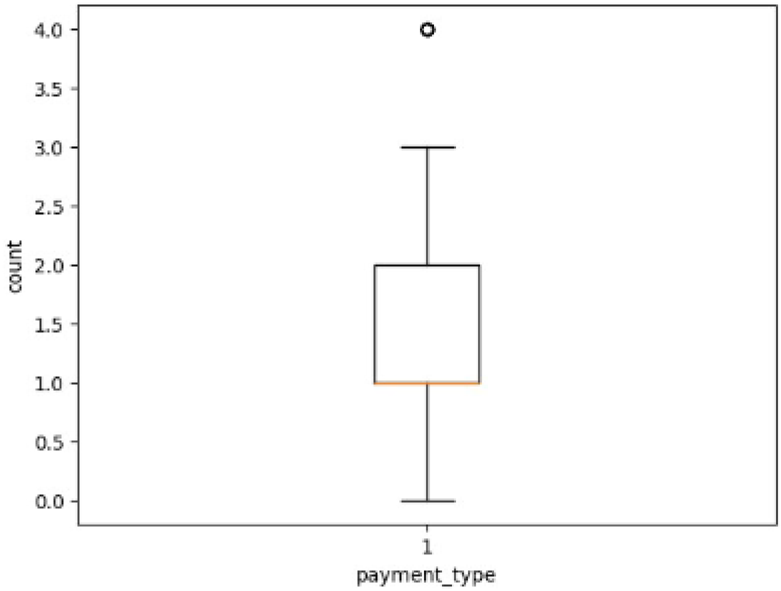


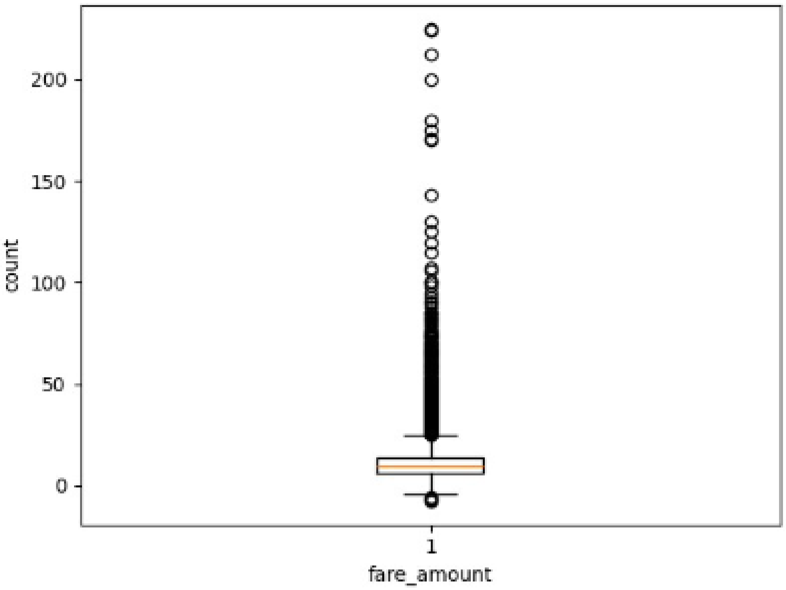

40
30
20
10
0
count
1
pickup_latitude

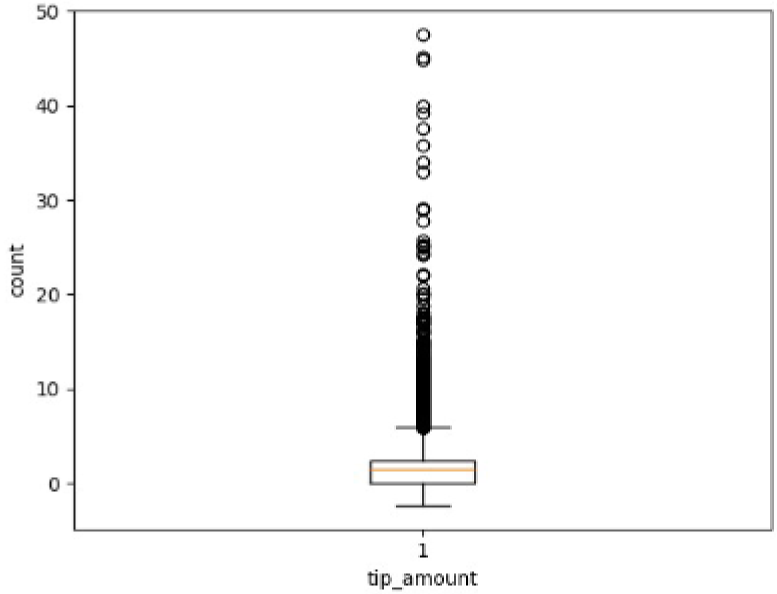

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
count
1
RatecodeID

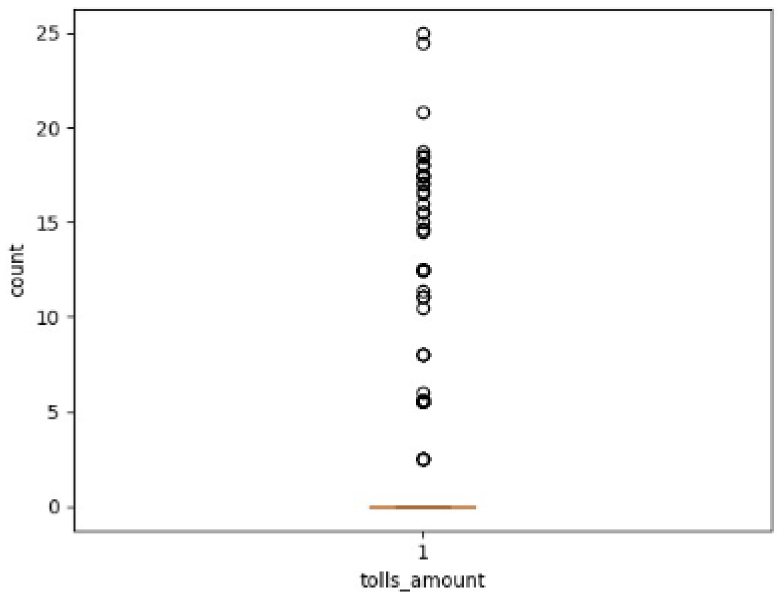

0
−10
−20
−30
−40
−50
−60
−70
count
1
dropoff_longitude

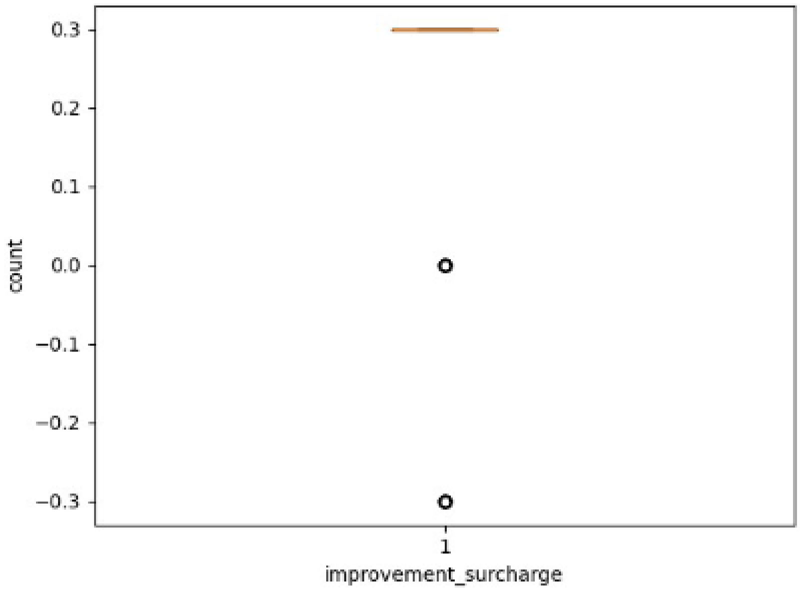

40
30
20
10
0
count
1
dropoff_latitude


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
count
1
payment_type


200
150
100
50
0
count
1
fare_amount


50
40
30
20
10
0
count
1
tip_amount


25
20
15
10
5
0
count
1
tolls_amount




```
data.isnull().sum()

VendorID                 0
passenger_count          0
trip_distance            0
pickup_longitude         0
pickup_latitude          0
RatecodeID               0
dropoff_longitude        0
dropoff_latitude         0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
dtype: int64

data.dropna(inplace=True)

# shape of DataFrame after removing outliers::
data.shape

(21112, 16)
```

- The VIF (Variance Inflation Factor) measures how much the variance of an estimated regression coefficient is inflated due to multicollinearity. Specifically, the VIF for each predictor variable is calculated as the ratio of the variance of the estimated coefficient when that variable is included in the model to the variance of the estimated coefficient when that variable is not included in the model.

```
# VIF Analysis::
l1=[]
for i in data.columns:
    if i!="total_amount":
        l1.append(i)
l1
x=data[l1]
```

```
vif_data=pd.DataFrame()
vif_data["Features"]=x.columns
vif_data["VIF"]=[variance_inflation_factor(x.values,i) for i in
range(len(x.columns))]
vif_data
```

```
c:\Users\revanth\AppData\Local\Programs\Python\Python310\lib\site-
packages\statsmodels\regression\linear_model.py:1784: RuntimeWarning:
invalid value encountered in scalar divide
  return 1 - self.ssr/self.uncentered_tss
```

	Features	VIF
0	VendorID	5.267265e+02
1	passenger_count	6.326495e+00
2	trip_distance	1.987328e+01
3	pickup_longitude	3.330774e+06
4	pickup_latitude	3.329665e+06
5	RatecodeID	2.595756e+00
6	dropoff_longitude	3.503569e+06
7	dropoff_latitude	3.501762e+06
8	payment_type	2.306407e+01
9	fare_amount	3.923792e+01
10	extra	1.616636e+00
11	mta_tax	6.073374e+03
12	tip_amount	7.591342e+00
13	tolls_amount	NaN
14	improvement_surcharge	6.747313e+03

```
data.drop(columns=["dropoff_longitude"],inplace=True)

data.drop(columns=["pickup_latitude"],inplace=True)

data.drop(columns=["improvement_surcharge"],inplace=True)

data.drop(columns=["dropoff_latitude"],inplace=True)

data.drop(columns=["mta_tax"],inplace=True)

data.drop(columns=["pickup_longitude"],inplace=True)

data.drop(columns=["RatecodeID"],inplace=True)

data.drop(columns=["fare_amount"],inplace=True)

data.drop(columns=["VendorID"],inplace=True)
```

```python
# DataFrame after doing VIF analysis::
data
```

```
       passenger_count  trip_distance  payment_type  extra  tip_amount  \
0                    1           2.50             1    0.5        2.05

1                    1           2.90             1    0.5        3.05
8                    1           0.70             1    0.5        2.00
10                   2           0.54             2    0.5        0.00
11                   1           1.70             2    0.5        0.00
...                ...            ...           ...    ...         ...
29990                1           3.72             1    0.0        3.06
29991                1           0.77             1    0.0        0.80
29992                2           1.38             1    0.0        2.32
29995                1           0.76             1    0.0        1.46
29997                1           2.37             1    0.0        3.56

       tolls_amount  total_amount
0               0.0         12.35
1               0.0         15.35
8               0.0          8.80
10              0.0          5.30
11              0.0          9.30
...             ...           ...
29990           0.0         18.36
29991           0.0          6.10
29992           0.0         11.62
29995           0.0          8.76
29997           0.0         21.36

[21112 rows x 7 columns]
```

```python
x=data.iloc[:,:-1]
y=data.iloc[:,-1]
print(y)
```

```
0        12.35
1        15.35
8         8.80
10        5.30
11        9.30
         ...
29990    18.36
29991     6.10
29992    11.62
29995     8.76
```

4. Perform feature selection and feature generation

Feature Selection:

- **Definition**: Feature selection involves choosing a subset of relevant features from the original set of features based on certain criteria.
- **Importance**: Selecting only the most relevant features can reduce overfitting, improve model performance, and reduce computational complexity.
- Using SelectKBest or Recursive Feature Elimination (RFE) in scikit-learn.

Feature Generation:

- **Definition**: Feature generation involves creating new features from the existing ones in the dataset.
- **Importance**: Generating new features can help capture additional information, improve model performance, and better represent the underlying patterns in the data.
- Creating interaction terms between different numerical features, deriving new features

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Handle non-numeric values in the 'payment_type' column
data['payment_type'] = pd.to_numeric(data['payment_type'],
errors='coerce')

# Feature Generation
data['distance_per_passenger'] = data['trip_distance'] /
data['passenger_count']
data['tip_percentage'] = (data['tip_amount'] / data['total_amount']) *
100

# Drop rows with missing values
data.dropna(inplace=True)

# Feature Selection
X = data.drop(columns=['total_amount'])  # Features
y = data['total_amount']  # Target variable
selector = SelectKBest(score_func=f_regression, k=5)  # Example:
Select top 5 features
X_selected = selector.fit_transform(X, y)

# Get indices of top 5 selected features
selected_indices = selector.get_support(indices=True)

# Get corresponding feature names
selected_features = X.columns[selected_indices]

# Print top 5 selected features
print("Top 5 selected features:")
for feature in selected_features:
    print(feature)

Top 5 selected features:
trip_distance
payment_type
tip_amount
distance_per_passenger
tip_percentage
```

5. Apply any of the machine learning algorithms discussed in the class for your selected problem.

Linear Regression Model:

- Definition: Linear regression is a linear approach to modeling the relationship between a dependent variable and one or more independent variables. It assumes that there is a linear relationship between the independent variables and the dependent variable.
- Model: The linear regression model can be represented as: $y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots+\beta_nx_n+\epsilon$, where yy is the dependent variable, $x_1,x_2,\ldots,x_nx_1,x_2,\ldots,x_n$ are the independent variables, $\beta_0,\beta_1,\ldots,\beta_n$ are the coefficients, and $\epsilon\epsilon$ is the error term.

```python
# linear regression model:
from sklearn.linear_model import LinearRegression
le=LinearRegression()
le.fit(X_train,y_train)
pred1=le.predict(X_test)
pred1
```

```
array([15.42009468, 22.6444754 , 21.60741047, ...,  9.20472615,
       10.78973588, 22.52593077])
```

```python
r2=r2_score(y_test,pred1)
print("r2_score=",r2)
mae=mean_absolute_error(y_test,pred1)
print("mean_absolute_error=",mae)
mse=mean_squared_error(y_test,pred1)
print("mean_squared_error=",mse)
rmse=np.sqrt(mean_squared_error(y_test,pred1))
print("rmse=",rmse)
```

```
r2_score= 0.8888097089853652
mean_absolute_error= 1.0449874909944374
mean_squared_error= 2.145629777135399
rmse= 1.4647968381777041
```

- **Accuracy: 88.88%**

Decision Tree Regression Model:

- Definition: Decision tree regression is a non-parametric supervised learning method used for regression tasks. It works by recursively partitioning the feature space into regions and fitting a simple model (e.g., mean or median) to each region.
- Model: A decision tree consists of nodes representing decision points, branches representing possible outcomes of decisions, and leaf nodes representing the final prediction.

```
# Decision tree regression model
from sklearn.tree import DecisionTreeRegressor
dt=DecisionTreeRegressor()
dt.fit(X_train,y_train)
pred2=dt.predict(X_test)
pred2
```

```
array([14.76      , 19.3       , 18.59      , ...,  8.50833333,
        9.3       , 20.3       ])
```

```
r2=r2_score(y_test,pred2)
print("r2_score=",r2)
mae=mean_absolute_error(y_test,pred2)
print("mean_absolute_error=",mae)
mse=mean_squared_error(y_test,pred2)
print("mean_squared_error=",mse)
rmse=np.sqrt(mean_squared_error(y_test,pred2))
print("rmse=",rmse)
```

```
r2_score= 0.9364082046738831
mean_absolute_error= 0.4796521667069244
mean_squared_error= 1.2271255735382274
rmse= 1.107757001123544
```

- **Accuracy : 93.64%**

Random Forest Regression Model:

- Definition: Random forest regression is an ensemble learning method that combines multiple decision trees to improve predictive performance and reduce overfitting.
- Model: A random forest consists of a collection of decision trees trained on random subsets of the data (bootstrap samples) and random subsets of the features (feature bagging). The final prediction is obtained by averaging or taking the majority vote of the predictions from individual trees.

```
# Random Forest regression model
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor()
rf.fit(X_train,y_train)
pred3=rf.predict(X_test)
pred3

array([14.76      , 19.314     , 18.6719    , ...,  8.58664503,
       10.55533333, 20.31      ])

r2=r2_score(y_test,pred2)
print("r2_score=",r2)
mae=mean_absolute_error(y_test,pred2)
print("mean absolute error=",mae)
mse=mean_squared_error(y_test,pred2)
print("mean_squared_error=",mse)
rmse=np.sqrt(mean_squared_error(y_test,pred2))
print("rmse=",rmse)

r2_score= 0.9364082046738831
mean_absolute_error= 0.4796521667069244
mean_squared_error= 1.2271255735382274
rmse= 1.107757001123544
```

- **Accuracy : 93.64%**

KNN Regression Model:

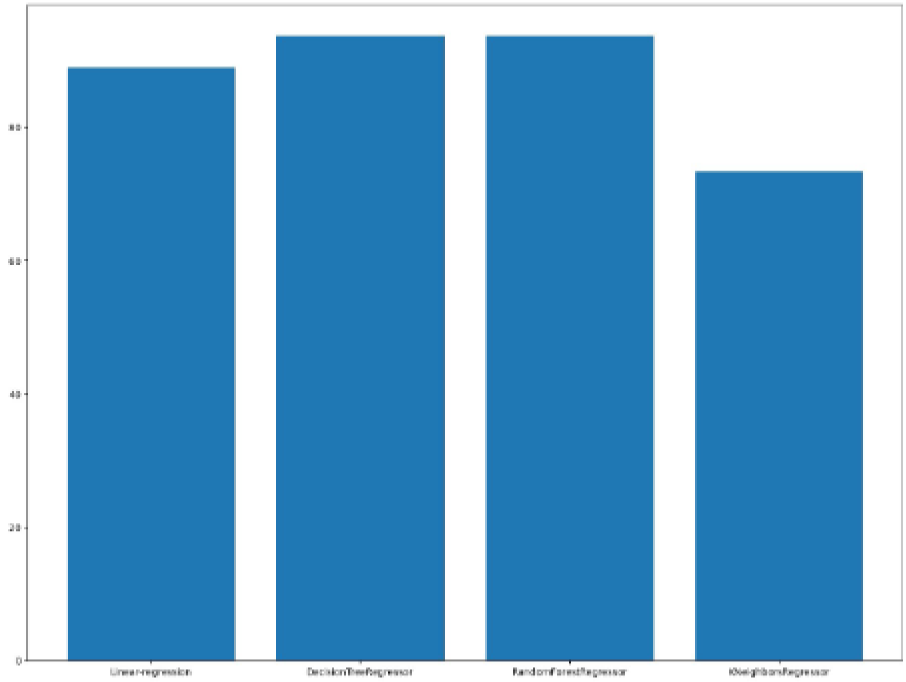
- Definition: KNN (K-Nearest Neighbors) regression is a non-parametric method used for regression tasks. It predicts the value of a target variable by averaging the values of its kk nearest neighbors in the feature space.
- Model: Given a new data point, KNN regression identifies the kk nearest neighbors based on a distance metric (e.g., Euclidean distance) and computes the average (or weighted average) of their target variable values as the prediction.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error,r2_score

# Select features and target variable
X = data[['passenger_count', 'trip_distance']]  # Adjust features as needed
y = data['total_amount']  # Target variable

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

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the KNN model
knn = KNeighborsRegressor(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# Make predictions
y_pred = knn.predict(X_test_scaled)

# Evaluate the model
r2=r2_score(y_test, y_pred)
print("r2_score=",r2)
mae=mean_absolute_error(y_test, y_pred)
print("mean_absolute_error=",mae)
mse=mean_squared_error(y_test, y_pred)
print("mean_squared_error=",mse)
rmse=np.sqrt(mean_squared_error(y_test, y_pred))
print("rmse=",rmse)

r2_score= 0.7331492799630404
mean_absolute_error= 1.711903317535545
```

- **Accuracy : 73.31%**

CONCLUSION:

```
fig = plt.figure()
ax = fig.add_axes([0,0,2,2])
models = ['Linear-regression','DecisionTreeRegressor','RandomForestRegressor','KNeighborsRegressor']
accuracy = [88.88,93.64,93.64,73.31]
ax.bar(models,accuracy)
plt.show()
```



1. The Linear Regression model has an accuracy of 88.88%.
2. The Decision Tree Regression and Random Forest Regression model has an accuracy of 93.64%.
3. The K-Nearest Neighbours Regression model has an accuracy of 73.31%.

From this, we can conclude that the Decision Tree Regression and Random Forest Regression model has the highest accuracy among the four models presented. This suggests that the Decision Tree Regression and Random Forest Regression may be the most suitable models for the given dataset.