Source Code

Reading the data:

```
pip install pandas
```

```
pip install sodapy
```

```
import pandas as pd
from sodapy import Socrata
client = Socrata("data.cityofnewyork.us", None)
results = client.get("qp3b-zxtp", limit=250000)
# Convert to pandas DataFrame
Data_df = pd.DataFrame.from_records(results)
Data_df
```

Data Preparation and Wrangling:

```
# Remove rows with NaN values
Data_df = Data_df.dropna()

# Display the updated DataFrame
Data_df.head()
```

```
# List of columns to exclude from conversion
exclude_column = ['tpep_pickup_datetime', 'tpep_dropoff_datetime',
'store_and_fwd_flag']

# Convert all variables except exclude_columns to float
for column in Data_df.columns:
    if column not in exclude_column:
        Data_df[column] = Data_df[column].astype(float)

# Check the data types of the DataFrame
Data_df.dtypes
```

Eliminating Rows with Negative Fare Amounts:

```
# Remove rows with negative fare amounts
Data_df = Data_df[Data_df['fare_amount'] > 0]
Data_df
```

Statistical Analysis for payment type predictor:

Evaluating the proportion of individuals within each payment type category:

```
# Initialize a dictionary to store the percentage of each payment type
payment_type_percent = {}

# Iterate over the payment types
for payment_type in [1.0, 2.0, 3.0, 4.0]:
    # Count the number of rows with the current payment type
    num_payment_type = len(Data_df[Data_df['payment_type'] ==
payment_type])

    # Calculate the percentage of the current payment type
    percentage = (num_payment_type / len(Data_df)) * 100

    # Store the percentage
    payment_type_percent[payment_type] = percentage

print("Percentage of each payment type:")
print(payment_type_percent)
```

```
import matplotlib.pyplot as plt

# Initialize a list to store the payment types
labels = ['Credit Card', 'Cash ', 'No charge ', 'Dispute']

# Get the percentages from the dictionary
percentages = [payment_type_percent[1.0], payment_type_percent[2.0],
               payment_type_percent[3.0], payment_type_percent[4.0]]

# Create the pie chart
plt.figure(figsize=(4, 4))
plt.pie(percentages, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Percentage of each payment type')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.show()
```

Evaluating the proportion of individuals within each payment type category who provide a tip:

```
# Filter tip amounts without zero
non_zero_tips = Data_df[Data_df['tip_amount'] > 0]

# Initialize a dictionary to store the percentage of each payment type
payment_type_percentage = {}

# Iterate over the payment types
for payment_type in [1.0, 2.0, 3.0, 4.0]:
    # Count the number of rows with the current payment type
    payment_type_num = len(non_zero_tips[non_zero_tips['payment_type']
== payment_type])

    # Calculate the percentage of the current payment type
    percentage = (payment_type_num / len(non_zero_tips)) * 100

    # Store the percentage
    payment_type_percentage[payment_type] = percentage

print("Percentage of each payment type:")
print(payment_type_percentage)
```

Calculating the mean tip amount for each payment type:

```
# Group the data by 'payment_type'
average_tip_by_payment_type =
non_zero_tips.groupby('payment_type')['tip_amount'].mean()

print("Average tip provided by each payment_type passenger:")
print(average_tip_by_payment_type)
```

```
import matplotlib.pyplot as plt

# Define colors for each payment type
colors = ['green', 'lightsalmon', 'skyblue', 'coral']

# Get the payment types and corresponding average tips
payment_types = average_tip_by_payment_type.index
average_tips = average_tip_by_payment_type.values

# Create the histogram with custom colors
plt.figure(figsize=(8, 6))
plt.bar(payment_types, average_tips, color=colors)
plt.xlabel('Payment Type')
plt.ylabel('Average Tip Amount')
```

```
plt.title('Average Tip Amount by Payment Type')
plt.xticks(payment_types)  # Ensure all payment types are shown on x-axis
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Statistical Analysis for fare_amount predictor:

Calculating Minimum, Average and Maximun fare amounts in the data:

```
# Calculate minimum fare_amount
min_fare_amount = Data_df['fare_amount'].min()

# Calculate average fare_amount
avg_fare_amount = Data_df['fare_amount'].mean()

# Calculate maximum fare_amount
max_fare_amount = Data_df['fare_amount'].max()

# Print the results
print("Minimum fare amount charged from passengers:", min_fare_amount)
print("Average fare amount charged from passengers:", avg_fare_amount)
print("Maximum fare amount charged from passengers:", max_fare_amount)
```

Calculating the average tip amounts in different ranges of fare amonut:

```
# Define the fare amount ranges
ranges = [(0, 200), (200, 400), (400, 600)]

# Initialize dictionaries to store tip amounts for each range
tip_amounts = {range_: [] for range_ in ranges}

# Categorize fare amounts into ranges and calculate tip amounts
for range_ in ranges:
    min_fare, max_fare = range_
    filtered_data = Data_df[(Data_df['fare_amount'] >= min_fare) & (Data_df['fare_amount'] < max_fare)]
    tip_amounts[range_] = filtered_data['tip_amount']

# Calculate average tip amount for each range
average_tip_amounts = {range_: tip_amount.mean() for range_, tip_amount in tip_amounts.items()}

# Print average tip amounts for each range
for range_, avg_tip_amount in average_tip_amounts.items():
    print(f"Average tip amount for fare amount range {range_}: {avg_tip_amount:.2f}")
```

```
import matplotlib.pyplot as plt

# Extract the ranges and average tip amounts
ranges = [f"{range_[0]} - {range_[1]}" for range_ in
average_tip_amounts.keys()]
average_tips = list(average_tip_amounts.values())

# Create the bar plot
plt.figure(figsize=(8, 6))
plt.bar(ranges, average_tips, color='coral', edgecolor='black')

# Add labels and title
plt.xlabel('Fare Amount Range')
plt.ylabel('Average Tip Amount')
plt.title('Average Tip Amounts for Fare Amount Ranges')

# Add grid
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show the plot
plt.show()
```

Correlation Analysis:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Convert all variables except exclude_columns to float
numeric_data= Data_df.drop(columns=exclude_column).astype(float)

# Calculate correlation matrix
correlation_matrix1 = numeric_data.corr()

# Plot heatmap
plt.figure(figsize=(16, 13))
sns.heatmap(correlation_matrix1, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```

Data Visualization:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))

# Create the box plot
sns.boxplot(x='payment_type', y='tip_amount', data=Data_df)

# Add labels and title
plt.title('Box Plot of Tip Amount by Payment Type')
plt.xlabel('Payment Type')
plt.ylabel('Tip Amount')

# Show the plot
plt.show()
```

```
import matplotlib.pyplot as plt

# Scatter plot between 'tip_amount' and 'fare_amount'
plt.figure(figsize=(8, 6))
plt.scatter(Data_df['fare_amount'], Data_df['tip_amount'], alpha=0.5)
plt.title('Scatter Plot of Tip Amount by Fare amount')
plt.xlabel('Fare Amount')
plt.ylabel('Tip Amount')
plt.grid(True)
plt.show()
```

Linear Regression:

```
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Create interaction term
Data_df['interaction']=Data_df['payment_type'] * Data_df['fare_amount']

# Define predictor variables (X) and target variable (y) including the
interaction term
X = Data_df[['payment_type', 'fare_amount', 'interaction']]
y = Data_df['tip_amount']

# Add constant term for intercept
X = sm.add_constant(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)

# Fit linear regression model
model = sm.OLS(y_train, X_train).fit()

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

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

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

# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

Visualizing the Model Performance:

1.Fitted plot between fare_amount and tip_amount:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Create a scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='fare_amount', y='tip_amount', data=Data_df,
alpha=0.5)

# Add a regression line
sns.regplot(x='fare_amount', y='tip_amount', data=Data_df,
scatter=False, color='red')

# Add labels and title
plt.xlabel('Fare Amount ($)')
plt.ylabel('Tip Amount ($)')
plt.title('Scatter Plot of Fare Amount vs. Tip Amount with Regression
Line')

# Show plot
plt.grid(True)
plt.show()
```

2. Residual Plot:

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Calculate residuals
residuals = model.resid

# Plot residuals against predicted tip amounts
plt.figure(figsize=(10, 6))
plt.scatter(model.fittedvalues, residuals, alpha=0.5)
plt.xlabel('Predicted Tip Amount ($)')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.grid(True)
plt.axhline(y=0, color='red', linestyle='--')  # Add horizontal line at
y=0
plt.show()
```

3.Actual Vs Predicted Plot:

```
import statsmodels.api as sm
import matplotlib.pyplot as plt

# Get predicted tip amounts
predicted_tip_amounts = model.predict(X)

# Plot predicted vs. actual tip amounts
plt.figure(figsize=(10, 6))
plt.scatter(y, predicted_tip_amounts, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()],
          color='red', linestyle='--')  # Add diagonal line (y = x)

plt.xlabel('Actual Tip Amount ($)')
plt.ylabel('Predicted Tip Amount ($)')
plt.title('Predicted vs. Actual Tip Amounts')
plt.grid(True)
plt.show()
```

Predicting tip amounts in test data:

```
X_test
```

```
y_test
```

```
# Define the new data
X_new = pd.DataFrame({
    'const': 1,  # Adding a constant term
    'payment_type': [1.0],
    'fare_amount': [10],
    'interaction': [1.0 * 10]  # Calculate interaction term
})

# Predict tip amount for new data
tip_amount_prediction = model.predict(X_new)

print("Predicted Tip Amount:", tip_amount_prediction.iloc[0])
```

Random Forest Model:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score

# Create interaction term
Data_df['interaction'] = Data_df['payment_type'] *
Data_df['fare_amount']

# Define predictor variables (X) and target variable (y) including the
interaction term
X = Data_df[['payment_type', 'fare_amount', 'interaction']]
y = Data_df['tip_amount']

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

# Initialize the Random Forest regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Predict on the test data
y_pred = rf_model.predict(X_test)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)
```

```
# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
print("R-squared score:", r2)
```

10 Fold-Cross Validation:

```
import numpy as np
np.random.seed(123)
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

# Perform 10-fold cross-validation
mae_scores = -cross_val_score(rf_model, X, y, cv=10,
scoring='neg_mean_absolute_error')
mse_scores = -cross_val_score(rf_model, X, y, cv=10,
scoring='neg_mean_squared_error')
r2_scores = cross_val_score(rf_model, X, y, cv=10, scoring='r2')

# Calculate mean scores
mean_mae = np.mean(mae_scores)
mean_mse = np.mean(mse_scores)
mean_r2 = np.mean(r2_scores)

# Print the results
print("Mean Absolute Error (MAE):", mean_mae)
print("Mean Squared Error (MSE):", mean_mse)
print("Mean R-squared value:", mean_r2)
```

Google Colab link of our analysis:

https://colab.research.google.com/drive/1E6uqlW4mazOb8xSrgJmLpkVVqMUYanwG?usp=sharing