YHILLS EDUTECH – INTERNSHIP

Name : Gurrala Madhu Mohan Vamsi

Domain of Project : Data Science

Duration of Project : 2 Months

Title of Project : Capstone

Problem Statement:

Prediction of amount using Regression.

Solution:

We'll first load the data and take a look at the first few rows.

```
import pandas as pd
# Load the dataset
taxi_data = pd.read_csv(r'E:\Python\New folder\TaxiFare.csv')
# Display the first few rows of the dataset
taxi_data.head()
```

	unique_id	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dropoff	no_of_passenger
0	26:21.0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1
1	52:16.0	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1
2	35:00.0	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2
3	30:42.0	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1
4	51:00.0	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1

The dataset contains the following columns:

1. unique_id: An identifier for each trip.
2. amount: The fare amount for the taxi trip (our dependent variable).
3. date_time_of_pickup: The date and time when the trip started.
4. longitude_of_pickup: The longitude where the passenger was picked up.
5. latitude_of_pickup: The latitude where the passenger was picked up.
6. longitude_of_dropoff: The longitude where the passenger was dropped off.
7. latitude_of_dropoff: The latitude where the passenger was dropped off.
8. no_of_passenger: The number of passengers for the trip.

Before we proceed with regression, we should check for any missing values, explore the distribution of the dependent variable (amount), and potentially engineer features from the given data.

Let's start by checking for any missing values in the dataset.

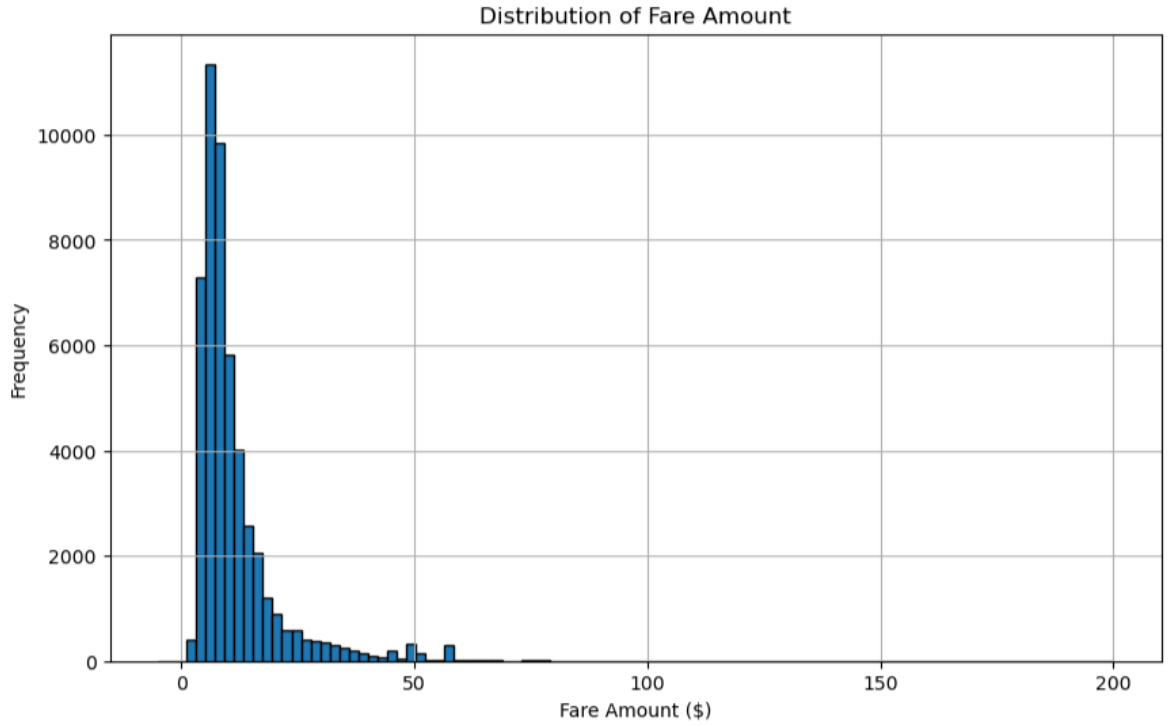
```
# Check for missing values in the dataset
missing_values = taxi_data.isnull().sum()
missing_values
```

```
unique_id            0
amount               0
date_time_of_pickup  0
longitude_of_pickup  0
latitude_of_pickup   0
longitude_of_dropoff 0
latitude_of_dropoff  0
no_of_passenger      0
dtype: int64
```

Great, there are no missing values in the dataset.

Next, let's explore the distribution of the dependent variable (amount). This will help us understand its distribution and identify any outliers.

```
import matplotlib.pyplot as plt
# Plot the distribution of the 'amount' column
plt.figure(figsize=(10, 6))
plt.hist(taxi_data['amount'], bins=100, edgecolor='k')
plt.title('Distribution of Fare Amount')
plt.xlabel('Fare Amount ($)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



The distribution of the fare amount shows a positive skew, with most fares being on the lower side. However, there seem to be some extreme values on the higher end, suggesting potential outliers.

To better handle the extreme values, we might consider:

- Investigating and possibly removing extreme outliers.
- Transforming the dependent variable (e.g., a log transformation) to make it more normally distributed for modeling.

But before making any decisions on the outliers, let's further investigate the high fare amounts to determine if they are genuine values or errors. We'll examine the rows with fare amounts greater than a certain threshold (e.g., $100) to get a sense of their characteristics.

```
# Filter the dataset for rows where the fare amount is greater than $100
high_fares = taxi_data[taxi_data['amount'] > 100]
high_fares.describe()
```

	amount	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dropoff	no_of_passenger
count	16.000000	16.000000	16.000000	16.000000	16.000000	16.00000
mean	139.756875	-69.225953	38.267570	-69.228812	38.320751	1.50000
std	28.490658	18.462011	10.206673	18.462786	10.221068	1.21106
min	103.000000	-74.429332	0.000000	-74.429332	0.000000	1.00000
25%	117.500000	-73.988409	40.723894	-73.989742	40.737210	1.00000
50%	135.500000	-73.860855	40.755950	-73.857810	40.775776	1.00000
75%	161.250000	-73.641737	40.823976	-73.639324	41.023610	1.00000
max	200.000000	0.000000	41.391042	0.000000	41.391042	5.00000

From the summary statistics of the high fare amounts:

- There are 16 trips with fare amounts greater than $100.
- The mean fare for these trips is approximately $139.76.
- The maximum fare is $200.
- Some of the longitude and latitude values are 0, which might indicate missing or erroneous data since a value of 0 for both longitude and latitude is highly unlikely for taxi trips.

Given these observations, we might consider:

- Removing or imputing rows with latitude or longitude values of 0, as these are likely errors.
- Retaining other high fare values if they seem legitimate (e.g., long-distance trips).

```
# Filter the high_fares dataset for rows where either latitude or longitude values are 0
erroneous_data = high_fares[(high_fares['latitude_of_pickup'] == 0) |
                            (high_fares['longitude_of_pickup'] == 0) |
                            (high_fares['latitude_of_dropoff'] == 0) |
                            (high_fares['longitude_of_dropoff'] == 0)]
erroneous_data
```

	unique_id	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dropoff	no_of_passenger
6630	51:00.0	128.83	2014-05-16 01:51:00 UTC	0.0	0.0	0.0	0.0	1

It looks like there's one row where both the pickup and dropoff latitude and longitude values are 0. Given a fare amount of $128.83, these values are very likely to be erroneous or missing.

```
# Remove the erroneous data from the main dataset
taxi_data_cleaned = taxi_data.drop(erroneous_data.index)
# Confirm the removal
len(taxi_data) - len(taxi_data_cleaned)
```

```
1
```

The erroneous row has been successfully removed.

Next, let's consider feature engineering. One valuable feature we could derive is the distance between the pickup and dropoff points, as the fare would likely be correlated with the distance traveled. We can compute this using the Haversine formula, which calculates the distance between two points on the Earth's surface given their latitude and longitude.

```
import numpy as np
def haversine_distance(lon1, lat1, lon2, lat2):
#Calculate the Haversine distance between two points on the earth specified by latitude and longitude.
    # Convert degrees to radians
    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    r = 6371 # Radius of Earth in kilometers
    distance = r * c
    return distance
#Compute the distance for each row in the dataset
taxi_data_cleaned['distance'] = haversine_distance(taxi_data_cleaned['longitude_of_pickup'],
                                                   taxi_data_cleaned['latitude_of_pickup'],
                                                   taxi_data_cleaned['longitude_of_dropoff'],
                                                   taxi_data_cleaned['latitude_of_dropoff'])
# Display the first few rows with the new distance feature
taxi_data_cleaned.head()
```

	unique_id	amount	date_time_of_pickup	longitude_of_pickup	latitude_of_pickup	longitude_of_dropoff	latitude_of_dropoff	no_of_passenger	distance
0	26:21.0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.712278	1	1.030764
1	52:16.0	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.782004	1	8.450134
2	35:00.0	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.750562	2	1.389525
3	30:42.0	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.758092	1	2.799270
4	51:00.0	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.783762	1	1.999157

The distance feature has been successfully computed and added to the dataset. It represents the distance (in kilometers) between the pickup and dropoff locations.

Before moving on to regression modeling, we can also extract some features from the date_time_of_pickup column, such as:

- Hour of the day: could capture the effect of rush hours or late-night fares.
- Day of the week: might capture weekly patterns like increased fares on weekends.
- Month: to account for monthly or seasonal effects.

```
# Convert the 'date_time_of_pickup' column to datetime format
taxi_data_cleaned['date_time_of_pickup'] = pd.to_datetime(taxi_data_cleaned['date_time_of_pickup'])
# Extract features from the 'date_time_of_pickup' column
taxi_data_cleaned['hour_of_pickup'] = taxi_data_cleaned['date_time_of_pickup'].dt.hour
taxi_data_cleaned['day_of_week'] = taxi_data_cleaned['date_time_of_pickup'].dt.dayofweek
taxi_data_cleaned['month'] = taxi_data_cleaned['date_time_of_pickup'].dt.month
# Display the first few rows with the new features
taxi_data_cleaned[['date_time_of_pickup', 'hour_of_pickup', 'day_of_week', 'month']].head()
```

	date_time_of_pickup	hour_of_pickup	day_of_week	month
0	2009-06-15 17:26:21+00:00	17	0	6
1	2010-01-05 16:52:16+00:00	16	1	1
2	2011-08-18 00:35:00+00:00	0	3	8
3	2012-04-21 04:30:42+00:00	4	5	4
4	2010-03-09 07:51:00+00:00	7	1	3

The features have been successfully extracted:

- hour_of_pickup: Represents the hour during which the trip started.
- day_of_week: Day of the week when the trip started (0 = Monday, 6 = Sunday).
- month: Month during which the trip occurred.

Now, we're ready to proceed with regression modeling. The next steps would be:

- Splitting the data into training and test sets.
- Training a regression model on the training set.
- Evaluating the model's performance on the test set.

```
from sklearn.model_selection import train_test_split
# Define the features and target variable
X = taxi_data_cleaned[['longitude_of_pickup', 'latitude_of_pickup',
                       'longitude_of_dropoff', 'latitude_of_dropoff',
                       'no_of_passenger', 'distance', 'hour_of_pickup',
                       'day_of_week', 'month']]
y = taxi_data_cleaned['amount']
# Split the data into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train.shape, X_test.shape
```

```
((39999, 9), (10000, 9))
```

The data has been split into training (39,999 samples) and test (10,000 samples) sets.

We train different regression models on the same dataset and compare their performances. Let's consider the following regression models:

- **Linear Regression:** A simple linear approach to modeling the relationship between the dependent variable and one or more independent variables.
- **Ridge Regression**: A type of linear regression that includes a regularization term to prevent overfitting.
- **Lasso Regression:** Another regularized version of linear regression, but it can lead to some feature coefficients being exactly zero.
- **Decision Tree Regressor:** A tree-based model that makes decisions based on asking a series of questions.
- **Random Forest Regressor:** An ensemble learning method that combines the predictions of multiple decision trees to produce a more accurate and robust prediction.

We'll train each of these models on the training data and evaluate their performances on the test set using RMSE and MAE. Let's proceed.

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error
# Dictionary to store results
results = {}

# List of regression models to evaluate
regressors = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'Decision Tree Regressor': DecisionTreeRegressor(),
    'Random Forest Regressor': RandomForestRegressor(n_estimators=100, random_state=42)
}

# Train each model and evaluate on test set
for name, regressor in regressors.items():
    regressor.fit(X_train, y_train)
    y_pred = regressor.predict(X_test)

    rmse = mean_squared_error(y_test, y_pred) ** 0.5
    mae = mean_absolute_error(y_test, y_pred)

    results[name] = (rmse, mae)

results
```

```
{'Linear Regression': (9.513757998297976, 5.9840622526362175),
 'Ridge Regression': (9.513757929237439, 5.984062183385498),
 'Lasso Regression': (9.52038232943459, 5.9888835768814745),
 'Decision Tree Regressor': (6.389799615706966, 3.1119606666666666),
 'Random Forest Regressor': (4.701174366989718, 2.2678751442460316)}
```

Let's analyze the results. The results table is below:

Model	RMSE	MAE
Linear Regression	9.51	5.98
Ridge Regression	9.51	5.98
Lasso Regression	9.52	5.99
Decision Tree Regressor	6.39	3.11
Random Forest Regressor	4.70	2.27

Analysis:

1. **Linear, Ridge, and Lasso Regression:**
   - These models have very similar performance. The RMSE and MAE values for these models are relatively high, indicating that their predictions are, on average, off by around $6 to $10.
   - The slight difference between Ridge and Linear Regression is due to the regularization term in Ridge, but it doesn't seem to have made a significant impact on this dataset.

- Lasso Regression, another regularized linear regression variant, has a slightly higher error than Ridge and simple Linear Regression, suggesting that the penalty it applies to certain coefficients might not be ideal for this data.

2. **Decision Tree Regressor:**
   - The decision tree has a lower RMSE and MAE compared to the linear models, indicating a better fit to the data. This suggests that the relationship between the features and the target might be non-linear, which decision trees can capture more effectively than linear models.
   - However, individual decision trees can sometimes overfit to the training data, leading to less optimal generalization to new data.
3. **Random Forest Regressor:**
   - The Random Forest, an ensemble of decision trees, performed the best among all the models. With the lowest RMSE and MAE values, it provides the most accurate predictions on the test data.
   - Random forests tend to generalize better than individual decision trees because they average the results of many trees, which reduces over fitting and variance.

Conclusion:

The Random Forest Regressor is the best model among the ones evaluated for this dataset, given its lowest RMSE and MAE values. It captures the complexities and potential non-linearities in the data better than the other models. If you were to deploy a model or use one for further predictions on similar data, the Random Forest Regressor would be the recommended choice.