

AI and DS-1

Experiment 5

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

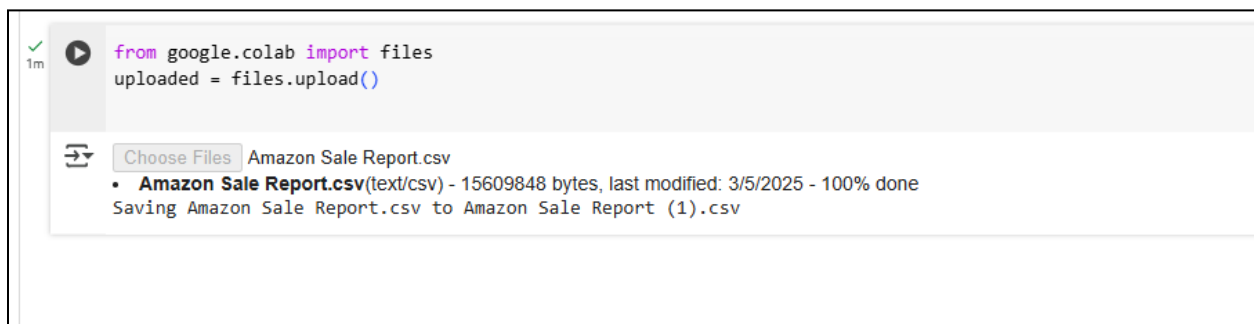
Objectives:

1. To perform Logistic regression to find out relation between variables
2. To apply regression model technique to predict the data on the selected dataset.

For this experiment we have selected the “Amazon Sales Report” dataset which consists of 24 columns and 130000 rows.

Steps performed:

1. Loading the dataset:



```
import pandas as pd

df = pd.read_csv("Amazon Sale Report.csv")

df.head()
```

	Order ID	Date	Status	Fulfilment	ship-service-level	Category	Size	Courier Status	Qty	Amount	ship-city	ship-state	ship-postal-code	B2B	fulfilled-by
0	405-8078784-5731545	04-30-22	Cancelled	Merchant	Standard	Set	S	NaN	0	647.62	MUMBAI	MAHARASHTRA	400081.0	False	Easy Ship
1	171-9198151-1101146	04-30-22	Shipped - Delivered to Buyer	Merchant	Standard	kurta	3XL	Shipped	1	406.00	BENGALURU	KARNATAKA	560085.0	False	Easy Ship
2	404-0687676-7273146	04-30-22	Shipped	Amazon	Expedited	kurta	XL	Shipped	1	329.00	NAVI MUMBAI	MAHARASHTRA	410210.0	True	NaN
3	403-9615377-8133951	04-30-22	Cancelled	Merchant	Standard	Western Dress	L	NaN	0	753.33	PUDUCHERRY	PUDUCHERRY	605008.0	False	Easy Ship
4	407-1069790-7240320	04-30-22	Shipped	Amazon	Expedited	Top	3XL	Shipped	1	574.00	CHENNAI	TAMIL NADU	600073.0	False	NaN

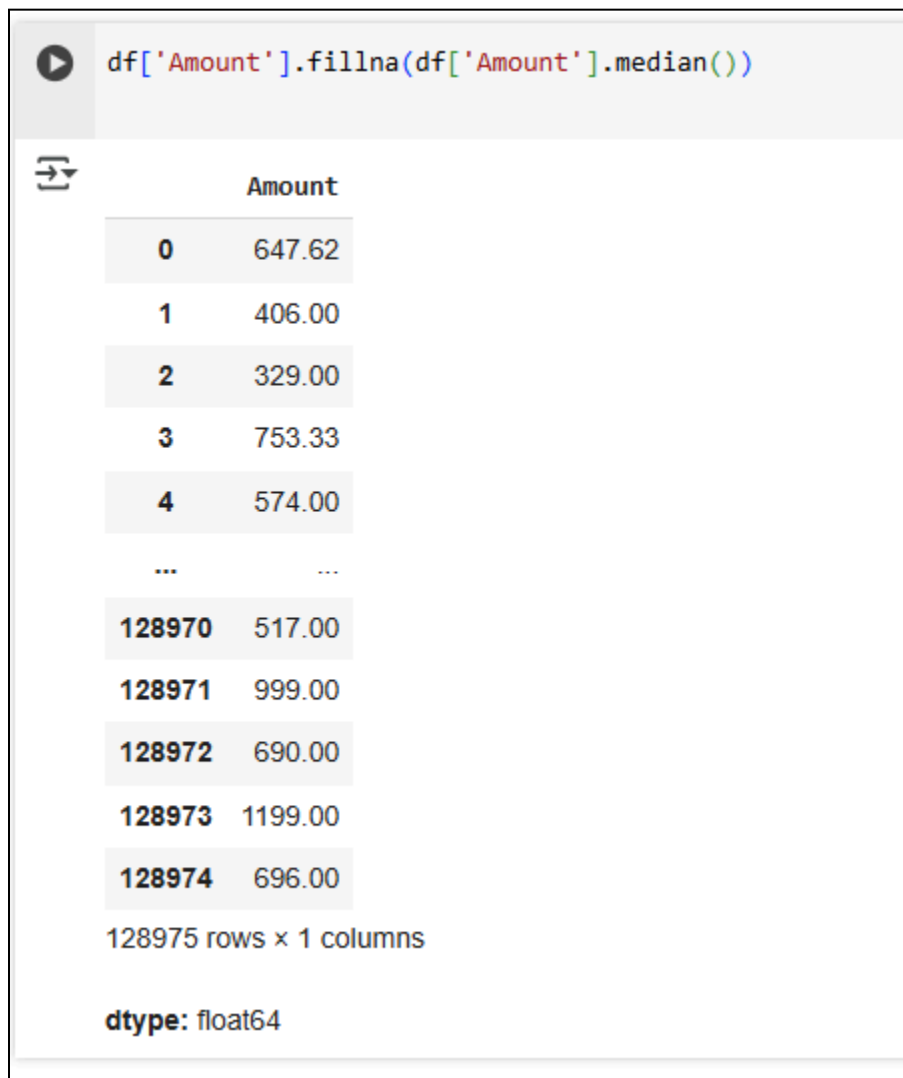
2. Preprocessing:

Since the dataset is not clean, performing preprocessing is necessary before applying any models.

a. Dropping columns that are not required for training the model

```
drop_cols = [
    'index', 'Order ID', 'SKU', 'ASIN', 'Date', 'Status', 'Fulfilment',
    'Sales Channel', 'ship-service-level', 'Courier Status', 'promotion-ids',
    'fulfilled-by', 'ship-city', 'ship-state', 'ship-postal-code', 'ship-country',
    'Unnamed: 22'
]
df = df.drop(columns=drop_cols, errors='ignore')
```

b. Replacing missing values in amount column with median



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
df['Amount'].fillna(df['Amount'].median())
```

The output displays a series of values for the 'Amount' column. The first five rows are indexed 0 to 4, followed by an ellipsis, and then rows 128970 to 128974. The values are: 647.62, 406.00, 329.00, 753.33, 574.00, ..., 517.00, 999.00, 690.00, 1199.00, 696.00. Below the values, it indicates '128975 rows x 1 columns' and 'dtype: float64'.

	Amount
0	647.62
1	406.00
2	329.00
3	753.33
4	574.00
...	...
128970	517.00
128971	999.00
128972	690.00
128973	1199.00
128974	696.00

128975 rows x 1 columns

dtype: float64

c. Handling categorical values

```
from sklearn.preprocessing import LabelEncoder

df["B2B"] = df["B2B"].astype(int)

category_new = LabelEncoder()
df["Category"] = category_new.fit_transform(df["Category"])

size_new = LabelEncoder()
df["Size"] = size_new.fit_transform(df["Size"])

print(df.head())
```

	Category	Size	Qty	Amount	B2B
0	5	7	0	647.62	0
1	8	0	1	406.00	0
2	8	8	1	329.00	1
3	7	5	0	753.33	0
4	6	0	1	574.00	0

3. Importing necessary libraries

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import mean_squared_error, r2_score
```

Linear regression model:

1. Splitting and training the model

```
X = df.drop(columns=["Amount"])  
y = df["Amount"]
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)  
  
model = LinearRegression()  
model.fit(X_train, y_train)  
  
y_pred = model.predict(X_test)
```

We have used linear regression model to predict amount based on other features in the dataset.

Y is the target variable (amount)

X contains the features that will be used for predicting amount (Category, Size, Qty, B2B)

2. Calculating MSE and R²

We have computed two key metrics: Mean Squared Error (MSE) and R² Score to evaluate the performance of the Linear Regression Model.

```
[ ] mse = mean_squared_error(y_test, y_pred)
```

```
[ ] print(f"Linear Regression MSE: {mse}")
```

```
Linear Regression MSE: 60078.54486639962
```

```
[ ] r2 = r2_score(y_test, y_pred)  
print(f"R2 Score: {r2}")
```

```
R2 Score: 0.20655238579347845
```

- MSE measures how far the predicted values are from the actual values.
 - A lower MSE is better because it means that the model is making smaller errors.
 - In our case the MSE came out to be 60078 which is quite high, indicating the model's predictions are not very close to the actual values.
-
- R^2 (R-squared) is a statistical measure that tells us how well our independent variables (features) explain the variation in the dependent variable (Amount).
 - It ranges from 0 to 1:
 - 0 → The model explains none of the variability (bad model).
 - 1 → The model explains 100% of the variability (perfect model).
 - 0.206 (our case) → The model explains only 20.6% of the variability in Amount, which is low.

High MSE and Low R^2 suggest that:

1. Some important features might be missing.
2. There might be non-linear relationships that Linear Regression can't capture.

Using a different set of features to perform linear regression:

Here we have computed the average transaction amount per city and state using the ship-city and ship-state columns.

```
# Compute mean Amount per city & state
city_avg_price = df.groupby('ship-city')['Amount'].mean().to_dict()
state_avg_price = df.groupby('ship-state')['Amount'].mean().to_dict()

# Map the computed values to new columns
df['city_avg_amount'] = df['ship-city'].map(city_avg_price)
df['state_avg_amount'] = df['ship-state'].map(state_avg_price)
```

Here we have calculated the Qty_size by multiplying values from quantity and size columns. This represents the total volume or weight of the order depending on the size.

```
df['Qty_Size'] = df['Qty'] * df['Size']
```

Based on the average transaction amount, we have calculated the Qty_CityImpact

```
df['Qty_CityImpact'] = df['Qty'] * df['city_avg_amount']  
df['Qty_StateImpact'] = df['Qty'] * df['state_avg_amount']
```

Splitting and testing the model again:

```
df.drop(columns=['ship-city', 'ship-state'], inplace=True)  
  
X = df[['Qty', 'Size', 'city_avg_amount', 'state_avg_amount', 'Qty_Size', 'Qty_CityImpact', 'Qty_StateImpact']  
      + list(df.filter(like='Category_').columns)] # Keeping one-hot encoded Category  
  
y = df['Amount']
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Split Data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train Model
model = LinearRegression()
model.fit(X_train, y_train)

# Make Predictions
y_pred = model.predict(X_test)

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

Mean Squared Error: 67979.03941884013

The MSE is still very high which means even after feature engineering the model's predictions did not improve.

We can conclude that,

- The relationship between Amount and other columns is not a simple straight-line relationship.
- Linear Regression assumes $y = mX + b$, but Amount might depend on complex interactions or non-linear patterns.
- If external factors (e.g., demand, seasonality, discounts) impact Amount, but those are not present in our dataset.

Logistic regression:

We are using logistic regression to classify the customer as high spender or low spender

1. Creating target variable

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
import numpy as np

df["High_Spend"] = (df["Amount"] > df["Amount"].median()).astype(int)
```

The target variable (High_Spend) is defined as whether the order's Amount is greater than the median Amount.

If Amount is greater than the median, it assigns 1 (High Spend); otherwise, 0 (Low Spend).

2. Handling categorical values

```
categorical_cols = ["Category", "Size"] # Specifying the categorical columns
X_categorical = df[categorical_cols]

# One-hot encode categorical features
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
X_categorical_encoded = encoder.fit_transform(X_categorical)
X_categorical_encoded = pd.DataFrame(X_categorical_encoded, columns=encoder.get_feature_names_out(categorical_cols))
```

OneHotEncoder converts categorical features (Category and Size) into numerical format for the model.

3. Selecting numerical features

```
# Select only relevant numeric columns (excluding Amount & High_Spend)
X_numeric = df.select_dtypes(include=['number']).drop(columns=["Amount", "High_Spend"])
```

Here the features other than amount and high spend have been selected. We have not included high spend since it is the target variable and amount because it directly determines target variable.

4. Train test split

```
X = pd.concat([X_numeric, X_categorical_encoded], axis=1)

y = df["High_Spend"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Now we have the numeric + encoded categorical values in X and Y contains target variable High Spend.

Next, we have splitted the dataset - 80% training and 20% testing.

5. Standardizing numerical features

```
# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Standardizing helps the model perform better

6. Training and predictions

```
# Train Logistic Regression Model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Make Predictions
y_pred = model.predict(X_test_scaled)
```

We have fit the scaled data to the regression model and the model can now classify the data as high spend or low spend

7. Evaluating model performance

```
# Evaluate Model Performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Accuracy: 0.8443

Classification Report:					
	precision	recall	f1-score	support	
0	0.87	0.83	0.85	13686	
1	0.82	0.86	0.84	12109	
accuracy			0.84	25795	
macro avg	0.84	0.85	0.84	25795	
weighted avg	0.85	0.84	0.84	25795	

Here we have calculated the accuracy of the model which is 84.43%

Classification report summary:

- Class 0 (Low Spend Customers)
 - Precision = 0.87 → When the model predicts "Low Spend," it is correct 87% of the time.
 - Recall = 0.83 → The model correctly identifies 83% of actual "Low Spend" customers.
 - F1-score = 0.85 → A good balance between precision and recall.

- Class 1 (High Spend Customers)
 - Precision = 0.82 → When the model predicts "High Spend," it is correct 82% of the time.
 - Recall = 0.86 → The model correctly identifies 86% of actual "High Spend" customers.
 - F1-score = 0.84 → A strong score, showing that both precision and recall are balanced.

Conclusion:

In this experiment we used linear regression to predict amount based on other features but the MSE turned out to be very high and R^2 value very low which indicates that amount may not have a linear relationship with other attributes. We used logistic regression to classify spenders as high spender and low spender based on the amount attribute. The accuracy of the model came out to be 84.43% which tells us that the model is accurately classifying the data.