**E D A (Exploratory Data Analysis)**

CASE Study : **E-commerce Sales Analytics Dashboard**

* Load the data
* Clean and analyze it (EDA)
* Prepare it for Machine Learning
* Train a basic ML model (e.g., Linear Regression or Classifier based on data)

1. **Load and Explore the Dataset**

import pandas as pd

# Load CSV

df= pd.read\_csv('/content/Amazon Sale Report.csv')

# Show top rows

df.head()

1. **Basic Info & Data Overview**

# Basic shape

print("Shape:", df.shape)

# Data types, nulls

df.info()

# Quick stats

df.describe(include='all')

1. **Check for Missing Values**

# Missing values count

df.isnull().sum()

# Visual heatmap

import seaborn as sns

import matplotlib.pyplot as plt

----------------------------------------------------------------------

Three ways for visualization of Missing values

1

sns.heatmap(df.isnull(), cbar=False, cmap='YlOrBr')

plt.title("Missing Value Heatmap")

plt.show()

2

!pip install missingno

import pandas as pd

import plotly.express as px

# Load the uploaded dataset

df= pd.read\_csv('/content/Amazon Sale Report.csv')

# Step 1: Calculate missing values

missing\_df = df.isnull().sum().reset\_index()

missing\_df.columns = ['Column', 'Missing Values']

missing\_df = missing\_df[missing\_df['Missing Values'] > 0]

# Step 2: Create a Plotly bar chart with labels

fig = px.bar(

missing\_df,

x='Column',

y='Missing Values',

text='Missing Values',

color='Missing Values',

color\_continuous\_scale='Agsunset',

title='Missing Values per Column (with Data Labels)',

height=500

)

fig.update\_traces(texttemplate='%{text}', textposition='outside')

fig.update\_layout(xaxis\_title='Column Name', yaxis\_title='Missing Count')

fig.show()

1. **Handle Missing Values**

# Drop columns with too many missing values (if any)

# df.drop(['ColumnName'], axis=1, inplace=True)

# Fill missing numerical values with mean

num\_cols = df.select\_dtypes(include=['float64', 'int64']).columns

for col in num\_cols:

    df[col].fillna(df[col].mean(), inplace=True)

# Fill missing categorical values with mode

cat\_cols = df.select\_dtypes(include=['object']).columns

for col in cat\_cols:

    df[col].fillna(df[col].mode()[0], inplace=True)

1. **Exploratory Data Analysis (EDA)**

**--> Distribution of numerical features**

df[num\_cols].hist(figsize=(10, 8), bins=20)

plt.tight\_layout()

plt.show()

**--> Correlation Matrix**

sns.heatmap(df[num\_cols].corr(), annot=True, cmap='coolwarm')

plt.title("Correlation Matrix")

plt.show()

**--> Value counts for categorical columns**

for col in cat\_cols:

    print(f"\n{col} value counts:\n", df[col].value\_counts())

1. **Feature Engineering**

**Depending on your dataset, extract useful features. For example, if there’s a "Date" column:**

# Convert to datetime

df['Date'] = pd.to\_datetime(df['Date'], errors='coerce')

# Extract parts of date

df['Day'] = df['Date'].dt.day

df['Month'] = df['Date'].dt.month

df['Year'] = df['Date'].dt.year

# Show updated columns

df[['Date', 'Day', 'Month', 'Year']].head()

**Also drop columns not needed:**

# Drop columns

df.drop(['Order ID', 'Customer Name'], axis=1, inplace=True, errors='ignore')

# ✅ Check column list

print("📦 Columns after dropping:", df.columns.tolist())

print()

df.head()

1. **Convert Categorical Columns to Numeric**

df = pd.get\_dummies(df, drop\_first=True)

1. **Feature Scaling**

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[num\_cols] = scaler.fit\_transform(df[num\_cols])

1. **Prepare Features and Target**

X = df.drop('Amount', axis=1)

y = df['Amount']

**10. Train-Test Split**

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**11. Train a ML Model**

Linear Regression (if target is numeric):

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print("R² Score:", r2\_score(y\_test, y\_pred))

print("MSE:", mean\_squared\_error(y\_test, y\_pred))

Random Forest (can handle both regression and classification):

from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor()

rf.fit(X\_train, y\_train)

preds = rf.predict(X\_test)

print("R² Score:", r2\_score(y\_test, preds))

print("MSE:", mean\_squared\_error(y\_test, preds))

**12. Plot Predictions vs Actual**

plt.scatter(y\_test, preds, alpha=0.5)

plt.xlabel("Actual")

plt.ylabel("Predicted")

plt.title("Actual vs Predicted")

plt.show()

**Step 1: Install and Import Required Libraries**

!pip install plotly --quiet

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import plotly.graph\_objects as go

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, auc, ConfusionMatrixDisplay

# -----------------------------------------------------------------------------

**Step 2: Load the Data**

df = pd.read\_csv('/mnt/data/Amazon Sale Report.csv')

df.head()

#------------------------------------------------------------------

Step 3: Initial Exploration

df.info()

df.describe(include='all')

#  ------------------------------------------------------

Step 4: Handle Missing Values

# Visualize missing

sns.heatmap(df.isnull(), cbar=False, cmap='magma')

plt.title("Missing Value Heatmap")

plt.show()

# Fill missing numerical with mean

num\_cols = df.select\_dtypes(include=['float64', 'int64']).columns

for col in num\_cols:

    df[col].fillna(df[col].mean(), inplace=True)

# Fill missing categorical with mode

cat\_cols = df.select\_dtypes(include=['object']).columns

for col in cat\_cols:

    df[col].fillna(df[col].mode()[0], inplace=True)

# ------------------------------------------------------

**Step 5: Attractive EDA (Exploratory Data Analysis)**

📌 Distribution of Categorical Features

for col in cat\_cols:

    fig = px.histogram(df, x=col, color=col, title=f"Distribution of {col}")

    fig.show()

**Correlation Heatmap**

corr = df[num\_cols].corr()

fig = px.imshow(corr, text\_auto=True, color\_continuous\_scale='Bluered\_r', title='Correlation Heatmap')

fig.show()

Interactive Box Plot (e.g., Amount by Category)

if 'Amount' in df.columns and 'Category' in df.columns:

    fig = px.box(df, x='Category', y='Amount', color='Category', title='Amount by Category')

    fig.show()

# ------------------------------------------------------------------

**Step 6: Encoding Categorical Variables**

df\_encoded = df.copy()

le = LabelEncoder()

for col in cat\_cols:

    df\_encoded[col] = le.fit\_transform(df\_encoded[col])

# -------------------------------------------

**Step 7: Define Features and Target**

 target = 'Status'  # <- Change if another column is your label

X = df\_encoded.drop(target, axis=1)

y = df\_encoded[target]

# ------------------------------------------------------

**Step 8: Train-Test Split + Scaling**

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# -------------------------------------------------------------

**Step 9: Train a Classifier**

model = RandomForestClassifier(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# ----------------------------------

**Step 10: Evaluation Report**

print(classification\_report(y\_test, y\_pred))

# ---------------------------------------------

**Step 11: Confusion Matrix (Fancy)**

cm = confusion\_matrix(y\_test, y\_pred)

labels = sorted(df[target].unique())

fig = px.imshow(cm,

                x=labels, y=labels,

                text\_auto=True,

                color\_continuous\_scale='Teal',

                title='Confusion Matrix')

fig.update\_layout(xaxis\_title='Predicted', yaxis\_title='Actual')

fig.show()

# ----------------------------------------------------

**Step 12: ROC Curve (if binary classification)**

if len(np.unique(y)) == 2:

    y\_prob = model.predict\_proba(X\_test)[:, 1]

    fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)

    roc\_auc = auc(fpr, tpr)

    fig = go.Figure()

    fig.add\_trace(go.Scatter(x=fpr, y=tpr, name='ROC Curve', mode='lines'))

    fig.add\_trace(go.Scatter(x=[0, 1], y=[0, 1], line=dict(dash='dash'), name='Random'))

    fig.update\_layout(title=f'ROC Curve (AUC = {roc\_auc:.2f})', xaxis\_title='False Positive Rate', yaxis\_title='True Positive Rate')

    fig.show()