#EDA USING PYTHON

**## IMPORT THE NECESSARY LIBRARIES**

import numpy as np

import pandas as pd #for database manipulation

import matplotlib.pyplot as plt

import seaborn as sns # For Visual representation

get\_ipython().run\_line\_magic('matplotlib', 'inline')

**import os**

# Get the current working directory

current\_directory = os.getcwd()

# Print the current working directory

print("Current Working Directory:", current\_directory)

# step 1 Import the dataset

df = pd.read\_excel("C:/Users/pc/Desktop/pizzaproject.xlsx")

**# step 2 Understanding the data**

df.head

df.describe()

df.tail()

ax = df['Contribution to turnover%'].plot(kind='hist',

title='contribution by department')

ax.set\_xlabel('Department')

**# check to know the mean, std, count and max**

df.describe()

**# step 3 Check the columns available in the dataset**

df.columns

**# step 4 Check for the unique values**

df.nunique()

**# Step 5 data** **preprocessing**

# 5.1 checking for null values

df.isnull().sum()

**# 5.2 Encoding categorical variables Description and Department. first we check the data type**

df.dtypes

# **We apply the label encoder on the Description. We look at the counts of this category**

df["Description"].value\_counts()

df["Department"].value\_counts()

**# Visualizing the relationship between the features using scatter plot**

fig,axs = plt.subplots(1,3,sharey=True)

df.plot(kind='scatter',x='Sold Cost',y='Contribution to turnover%',ax=axs[0],figsize=(16,8))

df.plot(kind='scatter',x='Sales Incl VAT',y='Contribution to turnover%',ax=axs[1])

df.plot(kind='scatter',x='QTY Sold',y='Contribution to turnover%',ax=axs[2])

**# Distribution plot of each features**

sns.distplot(df['Contribution to turnover%'], bins=5)

# Distribution of the cost

sns.distplot(df['Sold Cost'], bins=5)

**# a BAr plot**

plt.figure(figsize=(5,10))

sns.barplot(x= 'Department',y='Contribution to turnover%', data=df)

**# A Regression plot plot**

sns.regplot(y= 'Sold Cost',x='Contribution to turnover%', data=df)

**# Box plot**

sns.boxplot(df)

**#Import label encoder from sklearn**

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

print(label\_encoder)

#Apply the label encoder on the categorical variable

df["Description"] = label\_encoder.fit\_transform(df["Description"]

#View the new dataframe

df.head()

# to check the data type of the new data frame

df.dtypes

# to look at the value in Description now

df["Description"].unique()

# To check the new counts of Description

df["Description"].value\_counts()

**## Remove the columns that we wont need in our dataset, these are redundant variables**.

pizzaproject = df.drop(["Department", "Sales Excl VAT", "Profit", "Gross Profit%"], axis=1)

pizzaproject.head()

# plotting the Histogram of the target variable

plt.hist(pizzaproject["Contribution to turnover%"])

plt.xlabel("Contribution to turnover%")

plt.ylabel("Frequency")

plt.title("Histogram of Contribution")

**# FEATURE SCALING**

# Checking through the Statistics in Line 62, there is a need to scale the values and because we need to use linear regression

# First we do a pairplot

sns.pairplot(pizzaproject)

##Feature Scaling using MINMAX( each feature ranges between 0 and 1)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

pizzaproject = scaler.fit\_transform(pizzaproject)

pizzaproject = pd.DataFrame(pizzaproject, columns =['Description', 'QTY Sold', 'Sold Cost', 'Sales Incl VAT', 'Contribution to turnover%'])

sns.pairplot(pizzaproject)

# check the new statistics

pizzaproject.describe().round(2)

**# Feature Engineering**

# (a).( Outlier Analysis)

sns.scatterplot(pizzaproject)

# Box plot to show outlier

sns.boxplot(pizzaproject)

# Rename the column names

new\_column\_names = {

'Description': 'Description',

'Qty Sold': 'Qty\_Sold',

'Sold Cost': 'Sold\_Cost',

'Sales Incl VAT': 'Sales\_Incl',

'Contribution to turnover%': 'Contribution'

}

pizzaproject.rename(columns=new\_column\_names, inplace=True)

print(pizzaproject)

#The Encoded and scaled features dataset is now as below

**# Feature Engineering**

**# (b) Feature Importance**

CorrMat = pizzaproject.corr()

plt.figure(figsize = (10,8))

sns.heatmap(CorrMat, annot=True)

# TRAIN AND TEST SPLIT

#split the data into X and y

X = pizzaproject.drop('Contribution', axis=1)

y = pizzaproject['Contribution']

X.shape, y.shape

X

y

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

#split the data into training and testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 0)

# Data Validation- to confirm no overfitting using the HOLD -OUT VALIDATION

#print out the shape of the training and testing sets

print("Training set shape: ", X\_train.shape, y\_train.shape)

print("Testing set shape: ", X\_test.shape, y\_test.shape)

X\_train.head()

y\_train.head()

**# ML REGRESSION TECHNIQUES**

#1. To Create our Linear regression Model

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

**# To Train our Model, we use**:

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

**# print metric to get performance**

print("Accuracy: ",lr.score(X\_test, y\_test)\*100)

**#predicting the test**

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

y\_pred\_test

**# MODEL VALIDATION**

import scipy.stats as stats

# We already defined 'y\_observed' as the actual target values and 'y\_predicted' as the predicted values from the linear regression model.

**# Calculate the residuals (observed - predicted**)

resid = y\_test - y\_pred\_test

# Standardize the residuals to have mean 0 and standard deviation 1

standardized\_resid = (resid - np.mean(resid)) / np.std(resid)

**# Generate theoretical quantiles for a normal distribution based on the number of residuals**

theoretical\_quantiles = stats.norm.ppf(np.linspace(0.01, 0.99, len(standardized\_resid)))

**# Sort the standardized residuals and theoretical quantiles**

sorted\_resid = np.sort(standardized\_resid)

sorted\_theoretical\_quantiles = np.sort(theoretical\_quantiles)

**# Create the P-P plot**

plt.figure(figsize=(8, 6))

plt.plot(sorted\_theoretical\_quantiles, sorted\_resid, 'o', markersize=5)

plt.plot([-3, 3], [-3, 3], '--', color='red') # Adding a diagonal reference line

plt.xlabel('Theoretical Quantiles')

plt.ylabel('Standardized Resid')

plt.title('Probability-Probability (P-P) Plot')

plt.grid(True)

plt.show()

# ## In the P-P plot, if the model's residuals (standardized) align closely with the reference diagonal line (red dashed line), it suggests that the linear regression model's predictions conform to the assumptions of a normally distributed error term. Deviations from the diagonal line indicate potential departures from normality and may require further investigation or model refinement.

**# plotting the scatter plot to check Homocesticity.**

# Calculate the residuals

residuals = y\_test - y\_pred\_test

# Calculate mean and standard deviation of residuals

residuals\_mean = np.mean(residuals)

residuals\_std = np.std(residual

# Standardize the residuals

standardized\_residuals = (residuals - residuals\_mean) / residuals\_std

**# Create a scatter plot between predicted values and standardized residuals**

plt.scatter(y\_pred\_test, standardized\_residuals)

plt.axhline(y=0, color='red', linestyle='--')

plt.xlabel('Predicted Values')

plt.ylabel('Standardized Residuals')

plt.title('Scatter Plot of Predicted Values vs. Standardized Residuals')

plt.show()

# **Plot between the Actual and predicted value using scatter plot. Green line is the Actua**l

# and the red is the best fit for the predicted

# plot scatter plot between the test and predicted value()

plt.scatter(y\_test, y\_pred\_test, c='green')

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'k--',c='red',lw=3)

plt.xlabel('y\_test')

plt.ylabel('y\_pred\_test')

plt.title('Actuals Vs Predicted values')

**# Create a Distribution plot between the test and pred. values**

# Create a distribution plot of the predicted values using Seaborn

sns.histplot(y\_pred\_test, bins=30, kde=True)

plt.xlabel('Predicted Values')

plt.ylabel('Frequency')

plt.title('Distribution Plot of Linear Regression Model Predictions')

plt.show()

**# check the mean squared error**

from sklearn.metrics import mean\_squared\_error

mean\_squared\_error(y\_test, y\_pred\_test)

**# check the root mean square error**

np.sqrt(mean\_squared\_error(y\_test, y\_pred\_test))

**# check the Mean Absolute Error**

from sklearn.metrics import mean\_absolute\_error as mae

mean\_absolute\_error(y\_test, y\_pred\_test).round(6)

**# Using R2 Score for exact accuracy**

from sklearn.metrics import r2\_score

**# check accuracy of prediction**

#1. using visual

import matplotlib.pyplot as plt

plt.scatter(y\_train, y\_pred\_train)

plt.xlabel("Actual Contribution")

plt.ylabel("Predicted Contribution")

plt.show()

r2\_score(y\_test, y\_pred\_test)

**# ## FITTING THE DECISION TREE REGRESSION**

##criterion="squared\_error",

max\_depth=10,

min\_samples\_split=10,

random\_state=5)

**# Import the Decision Tree Model**

from sklearn.tree import DecisionTreeRegressor

reg = DecisionTreeRegressor()

**# Fitting the Model**

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

**# Print performance of model**

print("Accuracy: ",reg.score(X\_test,y\_test)\*100)

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

**# predictions for X\_test**

y\_pred

**# Mean Squared Error check**

from sklearn.metrics import mean\_squared\_error

mean\_squared\_error(y\_test, y\_pred)

**# Checking the Root mean square error**

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

**# Mean Absolute Error check**

from sklearn.metrics import mean\_absolute\_error

mean\_absolute\_error(y\_test, y\_pred)

**# r^2 SCORE CHECK**

r2\_score(y\_test, y\_pred)

# # Random Forest Regreesion

# Import the Random froest regressor

from sklearn.ensemble import RandomForestRegressor

**# Set the object for the model**

rf = RandomForestRegressor(n\_estimators=100)

**# Fit the model**

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

**# Print Accuracy of model**

print("Accuracy: ",rf.score(X\_train, y\_train)\*100)

**# # Feature Importance**

# Get the feature importances from the trained model

feature\_importances = rf.feature\_importances\_

**# Create a pandas Series with feature importances, indexed by feature names**

model\_ranks = pd.Series(feature\_importances, index=X\_train.columns, name="Importance")

**# Sort the feature importances in descending order**

model\_ranks\_sorted = model\_ranks.sort\_values(ascending=False)

**# Visualize the feature importances using a bar plot**

plt.figure(figsize=(10, 6))

ax = model\_ranks\_sorted.plot(kind='barh')

ax.set\_title("Feature Importance")

ax.set\_xlabel("Importance Score")

plt.show()

**# Make prediction of the variables**

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

#we now print out our Y\_pred(Preditted values for y\_test)

y\_pred

**# MODEL VALIDATION : ploting the p-plot graph**

import scipy.stats as stats

# We already defined 'y\_observed' as the actual target values and 'y\_predicted' as the predicted values from your linear regression model.

**# Calculate the residuals (observed - predicted)**

resid = y\_test - y\_pred

**# Standardize the residuals to have mean 0 and standard deviation 1**

standardized\_resid = (resid - np.mean(resid)) / np.std(resid)

**# Generate theoretical quantiles for a normal distribution based on the number of residuals**

theoretical\_quantiles = stats.norm.ppf(np.linspace(0.01, 0.99, len(standardized\_resid)))

**# Sort the standardized residuals and theoretical quantiles**

sorted\_resid = np.sort(standardized\_resid)

sorted\_theoretical\_quantiles = np.sort(theoretical\_quantiles)

**# Create the P-P plot**

plt.figure(figsize=(8, 6))

plt.plot(sorted\_theoretical\_quantiles, sorted\_resid, 'o', markersize=5)

plt.plot([-3, 3], [-3, 3], '--', color='red') # Adding a diagonal reference line

plt.xlabel('Theoretical Quantiles')

plt.ylabel('Standardized Resid')

plt.title('Probability-Probability (P-P) Plot')

plt.grid(True)

plt.show()

**# plotting the scatter plot to check Homocedencity.**

# Calculate the residuals

residuals = y\_test - y\_pred

**# Calculate mean and standard deviation of residuals**

residuals\_mean = np.mean(residuals)

residuals\_std = np.std(residuals)

**# Standardize the residuals**

standardized\_residuals = (residuals - residuals\_mean) / residuals\_std

**# Create a scatter plot between predicted values and standardized residuals**

plt.scatter(y\_pred, standardized\_residuals)

plt.axhline(y=0, color='red', linestyle='--')

plt.xlabel('Predicted Values')

plt.ylabel('Standardized Residuals')

plt.title('Scatter Plot of Predicted Values vs. Standardized Residuals')

plt.show()

# ## Now, the code calculates the standardized residuals and then creates a scatter plot between the predicted values and standardized residuals. The red dashed line at y=0 represents the horizontal reference line, indicating the center (mean) of the standardized residuals. Since there is no pattern# In[

# Evaluation of predited

# Get the mean squared error for the y\_test

from sklearn.metrics import mean\_squared\_error

mean\_squared\_error(y\_test, y\_pred)

**#Get the mean Absolute error for the y\_test**

from sklearn.metrics import mean\_absolute\_error

mean\_absolute\_error(y\_test, y\_pred)

# Get the root mean squared error

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

r2\_score(y\_test, y\_pred)

r2\_score(y\_test, y\_test)

**# # Deploying Selected Model**

# #R^2( Coefficient of Determination)

# What R-squared does: Compares the models predictions to the mean of the targets. Values can range from negative infinity(a very poor model) to 1. If all all the model does is predict the mean of the targets, it's R^2 value will be 0. And if the model perfectly predicts a range of numbers it's R^2 VALUE WOULD BE 1.(wikipedia)