```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug  9 12:57:27 2024

@author: vksku
"""
#======================================== IMPORT DATA ======================================
import pandas as pd
df = pd.read_csv('D:/DATA ANALYTICS/Python_ML/Wisdom_meghamart_project/Meghamart_data_raw.csv')
df.head()
df.shape
df.info()

#==================================== dropped irrelavant columns ========================
df.drop(columns = ['Row ID','Order ID', 'Customer Name'], inplace = True)

# Null operations ==================================
df.isnull().sum()

df['Segment'].value_counts()
rep = df['Segment'].mode()        # Replaced df['Segment'] nulls with mode value
rep.values
rep1 = rep.values[0]
df['Segment'].fillna(rep1, inplace = True)
df.isnull().sum()
                                  # Converted df['Sales'] to number data type
df['Sales'] = pd.to_numeric(df['Sales'], errors = 'coerce')
df['Sales'].iloc[39]
df.isnull().sum()

rep2 = df['Sales'].median()         # Replaced df['Sales'] nulls with median value
df['Sales'].fillna(rep2, inplace = True)
df.isnull().sum()
                                # converted df['Order Date'] and df['Ship Date'] into datetime form

df['Order Date'] = pd.to_datetime(df['Order Date'], errors = 'coerce', dayfirst=True)
df['Ship Date'] = pd.to_datetime(df['Ship Date'], errors = 'coerce', dayfirst=True)
df.isnull().sum()
df.info()
df.dropna(inplace = True)       # Dropped all null values from both date columns
df.shape

#==================================== DUPLICATES ========================================

df[df['Ship Mode'] == 'FC']['Ship Mode']
                                            #Replaced FC with First Class
df['Ship Mode'] = df['Ship Mode'].replace('FC', 'First Class')

df.apply(lambda x: x.duplicated().any())

#====================================== Feature Engineering ================================
def status(x):                  # Coverted df['Profit/Loss'] to Profit and Loss
    if(x<0):
        return "Loss"
    else:
        return "Profit"
```

```
df['Profit/Loss']=df['Profit/Loss'].apply(lambda x:status(x))
df['Profit/Loss']
                                # Standardized decimals to the round of 2

df['Sales'] = df['Sales'].round(2)
df['Discount'] = df['Discount'].round(2)

df.head(10)


# ==================================EDA Exploratory Data Analysis ============================
import seaborn as sns
import matplotlib.pyplot as plt

# 1. Yearly Sales Analysis
   # 2022 Showed highest sales while 2023 had lowest sales by small margin. All year showed equival
yearsales = df.groupby(df['Order Date'].dt.year)['Sales'].sum().reset_index()
yearsales = yearsales.sort_values(by = 'Sales', ascending = False )
plt.figure(figsize=(15,15))
var = sns.barplot(x=yearsales['Order Date'], y=yearsales['Sales'], palette='viridis', order=yearsal
for bars in var.containers:
    var.bar_label(bars)


# 2. Monthly Sales Analysis
    # MAY month shows the highest sales while FEB showed the lowest sales
monthlysales = df.groupby(df['Order Date'].dt.month_name())['Sales'].sum().reset_index().sort_value

plt.figure(figsize=(15,10))
var1 = sns.barplot(x=monthlysales['Order Date'], y=monthlysales['Sales'], palette='viridis', order=
for bars in var1.containers:
    var1.bar_label(bars)


# 3. Sales by Shipmode
    # Equivalent sales by each shipmode
shipmode = df.groupby(df['Ship Mode'])['Sales'].sum().reset_index().sort_values(by = 'Sales', ascer
shipmode

# 4. Sales by State
   #KA turned up the highest sales while MH was at the bottom
statesales = df.groupby('State')['Sales'].sum().reset_index().sort_values(by = 'Sales', ascending =
plt.figure(figsize=(15,10))
var2 = sns.barplot(x=statesales['State'], y=statesales['Sales'], palette='Dark2', order=statesales[
for bars in var2.containers:
    var2.bar_label(bars)

    # Breaking KA sales
dfk = df[df['State']=='Karnataka']
dfk = dfk.groupby('Segment')['Sales'].sum().reset_index()
plt.pie(dfk['Sales'], labels = dfk['Segment'], autopct = '%1.1f%%', startangle= 140)
plt.title('KARNATAKA')

    # Breaking MH sales
dfm = df[df['State']=='Maharashtra']
dfm = dfm.groupby('Segment')['Sales'].sum().reset_index()
plt.pie(dfm['Sales'], labels = dfm['Segment'], autopct = '%1.1f%%', startangle= 140)
```

```
plt.title('MAHARASTRA')
  # MH has to improve in Consumer and Corporate segements to top the charts

# 5. Sales by Discount
    #Sales are not dependent on discount
plt.figure(figsize=(20,10))
sns.relplot(x = df['Discount'], y =df['Sales'], kind = 'line')

        # Discount by Segement
df.groupby('Segment')['Discount'].mean()
df.groupby('Segment')['Sales'].sum()
 # DISCOUNT NEED TO BE INCREASED AND ADVERTISED FOR EACH SECTION


# 6. Customer sales analysis
cxqty = df['Customer ID'].value_counts().reset_index().sort_values(by = 'count', ascending = False)
cxqty.nlargest(50, 'count')

cxsales = df.groupby('Customer ID')['Sales'].sum().reset_index().sort_values(by = 'Sales', ascendir
cxsales.nlargest(50, 'Sales')
 # This give the Top 50 puchasing customer IDs whom we can send curated offer to increase sales


# 7. WEEKDAY Wise Sales
weeksales = df.groupby(df['Order Date'].dt.day_name())['Sales'].sum().reset_index().sort_values(by
plt.figure(figsize=(10,6))
sns.lineplot(x = weeksales['Order Date'], y = weeksales['Sales'], marker = 'o' )
# this shows the trend of online orders day wise and wednesday tops the chart

#====> INPUT AND GET THE ANSWER <=========
ask = input(str("Enter the Ship Mode: "))
if ask in df['Ship Mode'].values:
    a = df[df['Ship Mode'] == ask]['Sales']
    t = a.sum()
    print("SALES FOR THE PROVIDED SEGEMENT IS:", t)
else:
    print("Enter correct Shipmode & Try again")

#=====> INPUT AND GET THE ANSWER <=========
ask2 = input(str('Enter the state: '))
if ask2 in df['State'].values:
    x = df[df['State'] == ask2]
    tt = x.groupby('City')['Sales'].sum()
    print('Sales for the respective State & Cities are: ', tt)
else:
    print('Wrong input, enter correct state and try again')


#======================== BOX PLOT (OUTLIERS) ================================================

df.info()
df.rename(columns ={'Profit/Loss':'PLstatus'}, inplace = True)

outlier = df.select_dtypes(include = ['int64','float64']).columns
outlier
plt.figure(figsize=(20,15))
sns.boxplot(data = df[outlier])
```

```
                                    # BOXPLOT for sales column
sns.boxplot(df['Sales'])
q1 = df['Sales'].quantile(0.25)
q3 = df['Sales'].quantile(0.75)
IQR = q3-q1
ll = q1-(1.5*IQR)
ul = q3+(1.5*IQR)
ll , ul
df.shape

df = df[(df['Sales'] >= ll)&(df['Sales']<=ul)]
df.shape

df.columns

df.drop(columns = ['Order Date', 'Ship Date', 'Customer ID'],inplace = True)

# ============================== CORRELATIONS ============================================

correlations = df.select_dtypes(include = ['int64','float64']).columns
df[correlations].corr()
sns.heatmap(df[correlations].corr(), annot = True)

# ============================== SKEWNESS ================================================
sk = df.select_dtypes(include = ['int64','float64']).columns
df[sk].skew()

sns.displot(df['Sales'])
import numpy as np
sns.displot(np.log(df['Sales']))

# ============================== Transformation ==========================================
df.info()
                        #Label Encoding
from sklearn.preprocessing import LabelEncoder
labelc = df.select_dtypes(include = ['object']).columns
for c in labelc:
    le = LabelEncoder()
    df[c]=le.fit_transform(df[c])

df.head()

# ============================== Standardization =========================================
X = df.iloc[:,:7]
Y = df['PLstatus']
                        # STANDARD SCALER
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_S = ss.fit_transform(X)
x = pd.DataFrame(X_S)
y = Y
# ============================ 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)
x_train.shape,y_train.shape , x_test.shape, y_test.shape
```

```
# ============================== DATA BALANCING ============================================
y_train.value_counts()

from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state= 42)
x_trainb,y_trainb = sm.fit_resample(x_train, y_train)
y_trainb.value_counts()

# ========================== MODEL TRAINING + PREDICTIONS + ACCURACY ============================

# M O D E L --> 1 [Logistic Regression + balanced data]      ==> Accuracy: 73%
from sklearn.linear_model import LogisticRegression
lr1 = LogisticRegression()
lr1.fit(x_trainb,y_trainb)
#PREDICTION
y_test_pred1 = lr1.predict(x_test)
#ACCURACY
from sklearn.metrics import accuracy_score
accuracy_lr1 = accuracy_score(y_test,y_test_pred1)
accuracy_lr1

# M O D E L --> 2 [Logistic Regression]                      ==> Accuracy: 82%
from sklearn.linear_model import LogisticRegression
lr2 = LogisticRegression()
lr2.fit(x_train,y_train)
#PREDICTION
y_test_pred2 = lr2.predict(x_test)
y_train_pred2 = lr2.predict(x_train)
#ACCURACY
from sklearn.metrics import accuracy_score
accuracy_lrtest2 = accuracy_score(y_test,y_test_pred2)
accuracy_lrtrain2 = accuracy_score(y_train,y_train_pred2)
accuracy_lrtest2 , accuracy_lrtrain2



# M O D E L --> 3 [DT Classifier]                            ==> Accuracy: 77%
from sklearn.tree import DecisionTreeClassifier
dtc3 = DecisionTreeClassifier(criterion='gini')
dtc3.fit(x_train, y_train)
#PREDICTION
y_test_pred3 = dtc3.predict(x_test)
#ACCURACY
from sklearn.metrics import accuracy_score
accuracy_dtctest3 = accuracy_score(y_test,y_test_pred3)
accuracy_dtctest3

# M O D E L --> 4 [DT Classifier + balanced data]            ==> Accuracy: 76%
from sklearn.tree import DecisionTreeClassifier
dtc4 = DecisionTreeClassifier(criterion='entropy')
dtc4.fit(x_trainb, y_trainb)
#PREDICTION
y_test_pred4 = dtc4.predict(x_test)
#ACCURACY
from sklearn.metrics import accuracy_score
accuracy_dtctest4 = accuracy_score(y_test,y_test_pred4)
accuracy_dtctest4
```

```
# M O D E L --> 5 [Random Forest Classifier + balanced]      ==> Accuracy: 80%
from sklearn.ensemble import RandomForestClassifier
acc = []
ne = []
for j in range(20,100):
    rfc5 = RandomForestClassifier(n_estimators=j,
        criterion='entropy',
        random_state=42)
    rfc5.fit(x_trainb, y_trainb)
    #PREDICTION
    y_test_pred5 = rfc5.predict(x_test)
    #ACCURACY
    accuracy_rfctest5 = accuracy_score(y_test,y_test_pred5)
    ne.append(j)
    acc.append(accuracy_rfctest5)

acc
ne

pd.DataFrame(acc)
```