```
# -*- coding: utf-8 -*-
"""Task_Sale.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1VAliODEgDag-sSxFkoMLr38MGDjPJwgJ

### Understand Dataset
"""

import pandas as pd
df = pd.read_csv("Sales.csv",encoding="ISO-8859-1")
df.head()

df.info()

df.describe()

# Check unique ship modes

unique_ship_modes_count = df['Ship Mode'].nunique()
print(f"Number of unique Ship Modes: {unique_ship_modes_count}")
print(df['Ship Mode'].unique())

# Check unique Customers

unique_customer_name_count = df['Customer Name'].nunique()
print(f"Number of unique Scustomer Name: {unique_customer_name_count}")

# Check unique Segment

unique_Segment_count = df['Segment'].nunique()
print(f"Number of unique Segment: {unique_Segment_count}")
print(df['Segment'].unique())

# Check unique Country

unique_Country_count = df['Country'].nunique()
print(f"Number of unique Country: {unique_Country_count}")
print(df['Country'].unique())

# Check unique State

unique_State_count = df['State'].nunique()
print(f"Number of unique State: {unique_State_count}")
print(df['State'].unique())

# Check unique City

unique_City_count = df['City'].nunique()
print(f"Number of unique City: {unique_City_count}")
print(df['City'].unique())

# Check unique Region

unique_Region_count = df['Region'].nunique()
print(f"Number of unique City: {unique_Region_count}")
print(df['Region'].unique())

# Check unique Category

unique_Category_count = df['Category'].nunique()
print(f"Number of unique Category: {unique_Category_count}")
print(df['Category'].unique())

# Check unique Sub-Category

unique_Sub_Category_count = df['Sub-Category'].nunique()
print(f"Number of unique Sub-Category: {unique_Sub_Category_count}")
print(df['Sub-Category'].unique())

# Find numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns.tolist()

print("Numerical Columns:", numerical_columns)

# Find non-numerical columns
non_numerical_columns = df.select_dtypes(exclude=['number']).columns.tolist()

print("Non-Numerical Columns:", non_numerical_columns)

"""### Data preprocessing"""

# Check for missing values in each column
missing_values = df.isnull().sum()
print("Missing Values:")
```

```
print(missing_values)

# Select numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns

# Calculate IQR for each numerical column
for column in numerical_columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify outliers
    outliers = df[(df[column] < lower_bound) | (df[column] > upper_bound)]
    print(f"Outliers in {column}: {len(outliers)}")

# Check for duplicate rows
duplicates = df.duplicated()
print("Number of duplicates:", duplicates.sum())

df['Order Date'] = pd.to_datetime(df['Order Date'])
df['Ship Date'] = pd.to_datetime(df['Ship Date'])
df.info()

df.duplicated(subset=["Order ID","Product ID","Order Date","Ship Date"]).sum()



df[df.duplicated(subset=["Order ID", "Order Date"], keep=False)]

df = df.drop_duplicates(subset=["Order ID","Product ID","Order Date"],keep=False)

print(df.shape)

df['Lead Time'] = (df['Ship Date'] - df['Order Date']).dt.days

df[['Order Date', 'Ship Date', 'Lead Time']].head()

df["Order_year"]=df.loc[:,"Order Date"].dt.year
df['Order_year_month'] = df['Order Date'].dt.strftime('%Y-%m')

df.head()

df = df.drop(columns=["Product ID","Customer Name","Row ID","Postal Code","Country","Ship Date"])

df.head()

print(df.shape)

"""### Data Vizualization"""

import seaborn as sns
import matplotlib.pyplot as plt

sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title("Missing Values Heatmap")
plt.show()

# Find numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns.tolist()

print("Numerical Columns:", numerical_columns)

# ... (rest of the code)

# Plot boxplots for numerical columns
for column in numerical_columns:
    sns.boxplot(x=df[column])
    plt.title(f"Boxplot of {column}")
    plt.show()

top_selling_products = df.groupby('Product Name')['Sales'].sum().reset_index()

top_selling_products = top_selling_products.sort_values(by='Sales', ascending=False)

top_selling_products.head(10)

top_10_products = top_selling_products.head(10)

# Plot the top 10 selling products
plt.figure(figsize=(10, 6))
sns.barplot(x='Sales', y='Product Name', data=top_10_products, palette='viridis')
plt.title('Top 10 Selling Products')
plt.xlabel('Total Sales')
plt.ylabel('Product Name')
plt.show()
```

```
yearly_sales = df.groupby('Order_year')['Sales'].sum().reset_index()

yearly_sales

df.groupby('Category')['Sales'].sum().sort_values(ascending=True).plot.barh()

category_revenue_profit = df.groupby('Category').agg({'Sales': 'sum', 'Profit': 'sum'}).reset_index()

highest_revenue_profit = category_revenue_profit.sort_values(by='Profit', ascending=False).iloc[0]

print("Category generating the highest revenue and profit:")
print(highest_revenue_profit)

sales_by_region = df.groupby('Region')['Sales'].sum().sort_values(ascending = True).plot.barh()

df.groupby('Sub-Category')['Sales'].sum().sort_values(ascending = True).plot.barh()

ship_type_profit=df[["Ship Mode","Profit"]]
ship_type_profit.groupby('Ship Mode')['Profit'].mean().sort_values(ascending=False).plot(kind="bar",title="profit vs kind of ship")
plt.xlabel("Ship Mode")
plt.ylabel("avg Profit");

df.groupby('Region')['Profit'].mean().sort_values(ascending=False).plot(kind="bar",title="profit vs Region")
plt.xlabel("Region")
plt.ylabel("avg Profit");

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(15,5))
product_info=df[['Product Name','Sales',"Profit"]]
sales_info=product_info.groupby('Product Name')['Sales'].sum().sort_values().tail().plot(kind="bar",ax=ax1,color="green")
profit_info=product_info.groupby('Product Name')['Profit'].sum().sort_values().tail().plot(kind="bar",ax=ax2,color="orange")
ax1.set_title("top 5 sales products")
ax2.set_title("top 5 profit products")
plt.show();

montly_sales=df.groupby("Order Date")
montly_sales=montly_sales['Sales'].mean()
montly_sales=montly_sales.resample("M").sum()
plt.figure(figsize=(25,8))
plt.plot(montly_sales)
plt.xlabel("order date")
plt.ylabel("sales")
plt.title("monthly sales trend")
plt.show();

montly_sales=df.groupby("Order Date")
montly_sales=montly_sales['Profit'].mean()
montly_sales=montly_sales.resample("M").sum()
plt.figure(figsize=(25,8))
plt.plot(montly_sales)
plt.xlabel("order date")
plt.ylabel("Profit")
plt.title("monthly sales trend")
plt.show();

discount_group=df.groupby(['Quantity'])
discount_group=discount_group[['Sales','Profit']].mean()
ax=discount_group.plot(kind="bar")
ax.set_ylabel("Sales and profit")
plt.show();

import plotly.express as px
categories = df['Category'].value_counts().reset_index(name='Orders')
px.pie(data_frame=categories,values='Orders',names='Category',title='Number of Orders for each Category')

"""### Feature Engineering"""

import numpy as np
# Assuming df is your original DataFrame
df["Profit_Margin"] = df["Profit"] / df["Sales"]
df["Discount_Impact"] = df["Discount"] * df["Sales"]
df["Log_Sales"] = np.log1p(df["Sales"])
df["Log_Profit"] = np.log1p(df["Profit"])

from scipy.stats.mstats import winsorize
import pandas as pd

# Assuming df is already defined and contains a "Profit" column
df["Winsorized_Profit"] = winsorize(df["Profit"], limits=[0.05, 0.05])

# Encode categorical variables
df_encoded = pd.get_dummies(df, columns=["Category", "Segment", "Ship Mode"], drop_first=True)

# Define features (excluding Profit and Sales)
features = ["Quantity", "Discount", "Profit_Margin", "Discount_Impact", "Log_Sales"] + list(df_encoded.columns[-6:])

# Define targets
```

```
target_profit = "Winsorized_Profit"
target_sales = "Sales"

# Split data into features and targets
X = df_encoded[features]
y_profit = df_encoded[target_profit]
y_sales = df_encoded[target_sales]

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Split data into train and test sets
X_train, X_test, y_train_profit, y_test_profit = train_test_split(X, y_profit, test_size=0.2, random_state=42)
X_train, X_test, y_train_sales, y_test_sales = train_test_split(X, y_sales, test_size=0.2, random_state=42)

# Train Random Forest model for Profit
rf_profit = RandomForestRegressor(random_state=42)
rf_profit.fit(X_train, y_train_profit)

# Train XGBoost model for Sales
xgb_sales = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_sales.fit(X_train, y_train_sales)

# Evaluate Profit model
y_pred_profit = rf_profit.predict(X_test)
print("Profit Model Evaluation:")
print(f"MAE: {mean_absolute_error(y_test_profit, y_pred_profit)}")
print(f"MSE: {mean_squared_error(y_test_profit, y_pred_profit)}")
print(f"R²: {r2_score(y_test_profit, y_pred_profit)}")

# Evaluate Sales model
y_pred_sales = xgb_sales.predict(X_test)
print("\nSales Model Evaluation:")
print(f"MAE: {mean_absolute_error(y_test_sales, y_pred_sales)}")
print(f"MSE: {mean_squared_error(y_test_sales, y_pred_sales)}")
print(f"R²: {r2_score(y_test_sales, y_pred_sales)}")

# Create a DataFrame for comparison
comparison_df = pd.DataFrame({
    "Actual_Sales": y_test_sales,
    "Predicted_Sales": y_pred_sales,
    "Actual_Profit": y_test_profit,
    "Predicted_Profit": y_pred_profit
})

# Display the first few rows of the comparison DataFrame
print(comparison_df.head())

import matplotlib.pyplot as plt
import seaborn as sns

# Plot Actual vs. Predicted Sales
plt.figure(figsize=(10, 6))
sns.scatterplot(x="Actual_Sales", y="Predicted_Sales", data=comparison_df)
plt.plot([comparison_df["Actual_Sales"].min(), comparison_df["Actual_Sales"].max()],
         [comparison_df["Actual_Sales"].min(), comparison_df["Actual_Sales"].max()],
         color='red', linestyle='--')  # Diagonal line for perfect predictions
plt.title("Actual vs. Predicted Sales")
plt.xlabel("Actual Sales")
plt.ylabel("Predicted Sales")
plt.show()

# Plot Actual vs. Predicted Profit
plt.figure(figsize=(10, 6))
sns.scatterplot(x="Actual_Profit", y="Predicted_Profit", data=comparison_df)
plt.plot([comparison_df["Actual_Profit"].min(), comparison_df["Actual_Profit"].max()],
         [comparison_df["Actual_Profit"].min(), comparison_df["Actual_Profit"].max()],
         color='red', linestyle='--')  # Diagonal line for perfect predictions
plt.title("Actual vs. Predicted Profit")
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.show()

# Calculate absolute and percentage errors
comparison_df["Sales_Absolute_Error"] = abs(comparison_df["Actual_Sales"] - comparison_df["Predicted_Sales"])
comparison_df["Sales_Percentage_Error"] = (comparison_df["Sales_Absolute_Error"] / comparison_df["Actual_Sales"]) * 100

comparison_df["Profit_Absolute_Error"] = abs(comparison_df["Actual_Profit"] - comparison_df["Predicted_Profit"])
comparison_df["Profit_Percentage_Error"] = (comparison_df["Profit_Absolute_Error"] / comparison_df["Actual_Profit"]) * 100

# Display the first few rows with errors
print(comparison_df.head())

# Plot distribution of Sales Absolute Errors
plt.figure(figsize=(10, 6))
```

```
sns.histplot(comparison_df["Sales_Absolute_Error"], kde=True)
plt.title("Distribution of Sales Absolute Errors")
plt.xlabel("Absolute Error")
plt.ylabel("Frequency")
plt.show()

# Plot distribution of Profit Absolute Errors
plt.figure(figsize=(10, 6))
sns.histplot(comparison_df["Profit_Absolute_Error"], kde=True)
plt.title("Distribution of Profit Absolute Errors")
plt.xlabel("Absolute Error")
plt.ylabel("Frequency")
plt.show()
```

```
import pandas as pd

# Function to collect user input
def get_user_input():
    print("Please enter the following details:")
    quantity = float(input("Quantity: "))
    discount = float(input("Discount (e.g., 0.1 for 10%): "))
    category = input("Category (e.g., Furniture, Office Supplies, Technology): ")
    segment = input("Segment (e.g., Consumer, Corporate, Home Office): ")
    ship_mode = input("Ship Mode (e.g., Standard Class, Second Class, First Class, Same Day): ")

    # Return input as a dictionary
    return {
        "Quantity": quantity,
        "Discount": discount,
        "Category": category,
        "Segment": segment,
        "Ship Mode": ship_mode
    }

# Get user input
input_data = get_user_input()

# Convert to DataFrame
input_df = pd.DataFrame([input_data])

# Apply feature engineering (same as training data)
input_df["Profit_Margin"] = 0  # Placeholder, since Profit is not available
input_df["Discount_Impact"] = input_df["Discount"] * 0  # Placeholder, since Sales is not available
input_df["Log_Sales"] = 0  # Placeholder, since Sales is not available

# Encode categorical variables
input_df_encoded = pd.get_dummies(input_df, columns=["Category", "Segment", "Ship Mode"], drop_first=True)

# Ensure all columns are present (in case some categories were not in the input)
missing_cols = set(features) - set(input_df_encoded.columns)
for col in missing_cols:
    input_df_encoded[col] = 0

# Reorder columns to match the training data
input_df_encoded = input_df_encoded[features]

# Predict Profit and Sales
predicted_profit = rf_profit.predict(input_df_encoded)
predicted_sales = xgb_sales.predict(input_df_encoded)

print(f"Predicted Profit: {predicted_profit[0]}")
print(f"Predicted Sales: {predicted_sales[0]}")
```

```
import joblib

# Save the Random Forest model for profit prediction
joblib.dump(rf_profit, "rf_profit_model.pkl")

# Save the XGBoost model for sales prediction
joblib.dump(xgb_sales, "xgb_sales_model.pkl")

print("Models saved successfully!")
```