Sales Analysis

1. Prepare Data for Analysis

- Begin by importing necessary libraries
- Read the csv file and store it as a DataFrame named df
- Print the DataFrame df and determine its dimensions
- Check for missing values in the DataFrame
- Verify that there are no non-missing values in the DataFrame

```
# Import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns


# Read CSV file

df = pd.read_csv("Sales.csv")


# Print DataFrame

print(df)


# Check dimensions

print("Shape of data:", df.shape)


# Check missing values

print(df.isnull().sum())
```

2. Normalize Data for Analysis

- Create a new DataFrame called df_dataonly from the existing df object
- Create a normalize object
- Invoke the fit_transform() method, and pass this newly created object called df_dataonly and name the resulting object as normalize_data.
- The normalize_data object is an ndarray of 2 columns and 7560 rows. The first column is the **Unit**, and the second column is the **Sales** data. Normalization will scale the data in each column to a range between 0 and 1.
- Check the min and max values of each column. Min should be 0.0 and max 1.0, for Unit and Sales column.

```
from sklearn.preprocessing import MinMaxScaler
```

> [sklesrn = This is the name of the library,
preprocessing = This is a module for preparing data. Used for Scaling, Normalization, etc.
MinMaxScaler = This is a class. Scales data between 0 and 1.]

```
# Create new DataFrame with only numeric columns

df_dataonly = df[['Unit', 'Sales']]
```

> [Only two columns Unit and Sales were taken from the original DataFrame.
Double brackets → to select multiple columns.
df_dataonly= New DataFrame. This will be scaled instead of the original df.]

```
# Create normalize object

scaler = MinMaxScaler()
```

> [Call the class and create an object. Now the scaler is ready to scale the data.]

```
# Fit and transform

normalize_data = scaler.fit_transform(df_dataonly)
```

> [fit= Finds the minimum and maximum value in data.
transform= Converts values from 0 to 1 by applying formula

fit_transform = First fits then transforms, does both these works together

df_dataonly = Input data

normalize_data = Output variable

This is a NumPy array, not a DataFrame.]

# Convert to DataFrame

normalize_df = pd.DataFrame(normalize_data, columns=['Unit','Sales'])

> [pd.DataFrame() = Convert NumPy array back to DataFrame.

normalize_data = Input array

columns=['Unit','Sales'] => Column names assigned.

normalize_df = Final normalized DataFrame]

# Check min and max

print(normalize_df.min())

print(normalize_df.max())

> [Shows the minimum and maximum value of each column.]

3. Visualize Overall Trends

- Plot the **Date** versus **Unit** and **Date** versus **Sales** line plot for the entire season. Note the values of **Unit** and **Sales** are summed up for each day.

# Convert Date column to datetime

df['Date'] = pd.to_datetime(df['Date'])

> [df['Date'] = Accessing the Date column in your DataFrame

pd = pandas library

to_datetime = Converts a date in string format to actual datetime format.]

# Group by Date

daily_data = df.groupby('Date').sum()

> [groupby('Date') = All records with the same date are grouped together.

.sum() = Totals Units and Sales for each date.

daily_data = New DataFrame, groupby के बाद Date index बन गया हे]

# Plot

plt.figure(figsize=(12,5))

plt.plot(daily_data.index, daily_data['Unit'], label='Units Sold')

plt.legend()   # Shows label in graph. (Show Legend)

plt.title("Daily Units")   # add title

plt.show()   # display graph

> [plt = matplotlib.pyplot library.

figure() = Creates a new graph area.

figsize=(12,5) = Width = 12, Height = 5, Makes the graph larger and clearer

plot() = Creates a line graph.

daily_data.index = X-axis values.   Here is Date

daily_data['Unit'] = Y-axis values. Total Units for each Date
label='Units Sold' = To show name in legend]

plt.figure(figsize=(12,5))

plt.plot(daily_data.index, daily_data['Sales'], label='Sales Revenue')

plt.legend()

plt.title("Sales Trend")

plt.show()

4. Analyze Monthly Data

- Chunk this quarterly data into monthly data and perform the analysis. For each month, get the sub-DataFrame, using the **loc** feature of DataFrame, and pass the range by date.

```
df = df.set_index('Date')    # Created an index on the 'Date' column

> [set_index() = This function sets a column from a DataFrame as an index.]

df_oct = df.loc['2020-10']

> [loc means: Label-based indexing, here label means Date.
['2020-10'] = Select all records from October 2023]


df_nov = df.loc['2020-11']

df_dec = df.loc['2020-12']
```

5. Describe Data

- Describing the data will give you basic information such as count, mean, std (standard deviation), min, max and the quartiles. You will use the describe() command on the DataFrame to get it. All the values are for the entire three-month period.
- For each of the three months, you can invoke describe() on df_oct, df_nov, and df_dec.

```
print("Overall Data Description")

print(df.describe())



print("October")

print(df_oct.describe())



print("November")

print(df_nov.describe())
print("December")

print(df_dec.describe())
```

6. Analyze Unit Data

- Unit Analysis

  You will **use** boxplots to **visualize** the distribution of units sold for each month.

- Sales Analysis

  You will **explore** the distribution of sales revenue for each month.

```
# add month column

df_oct['Month'] = 'October'

df_nov['Month'] = 'November'

df_dec['Month'] = 'December'

# combine all months

final_df = pd.concat([df_oct, df_nov, df_dec])

> [pd.concat() = Vertically joins multiple DataFrames.

[df_oct, df_nov, df_dec] = Three DataFrames are given in the List.]


# boxplot

sns.boxplot(x='Month', y='Unit', data=final_df)

plt.title("Units Sold Distribution")

> [sns.boxplot() = Seaborn's statistical plot function

x='Month' = Month column used on the X-axis.

y='Unit' = Unit column will be plotted on the Y-axis

data=final_df = Complete DataFrame given to seaborn]
```

```
sns.boxplot(x='Month', y= 'Sales', data=final_df)
plt.title("Sales Revenue Distribution")
```

7. Explore Monthly Plots and Analysis

- Overall **Unit** and **Sales** figures
- Units sold in **October**, **November,** and **December**
- Sales numbers for **October**, **November,** and **December**
- Consolidated 3-month sales plot

```
# Overall Totals
total_units = df['Unit'].sum()
total_sales = df['Sales'].sum()
print("Total Units (Quarter):", total_units)
print("Total Sales (Quarter):", total_sales)


# Units Sold in October, November, December
oct_units = df_oct['Unit'].sum()
nov_units = df_nov['Unit'].sum()
dec_units = df_dec['Unit'].sum()
print(oct_units, nov_units, dec_units)


# plot
plt.figure(figsize=(8,5))
plt.bar(['October','November','December'],
    [oct_units, nov_units, dec_units])
plt.title("Monthly Units Sold")
plt.ylabel("Units")
plt.show()


# Sales Numbers for October, November, December

oct_sales = df_oct['Sales'].sum()

nov_sales = df_nov['Sales'].sum()

dec_sales = df_dec['Sales'].sum()

print(oct_sales, nov_sales, dec_sales)


# plot

plt.figure(figsize=(8,5))

plt.bar(['October','November','December'],

      [oct_sales, nov_sales, dec_sales])

plt.title("Monthly Sales Revenue")

plt.ylabel("Sales")

plt.show()


# Consolidated 3-Month Sales Plot

# Daily Sales Trend (Full Quarter)

daily_sales = df.resample('D').sum()
```

> [resample() = Used to group time-series data at a new frequency.
'D' = Daily frequency
Aggregation is required after resample. इसी लिए sum()]

```
plt.figure(figsize=(12,5))

plt.plot(daily_sales.index, daily_sales['Sales'])

plt.title("Consolidated 3-Month Sales Trend")

plt.xlabel("Date")

plt.ylabel("Sales")

plt.show()
```

8. Analyze Statewise Sales in the United States

```
state_sales = df.groupby('State').sum()

state_sales['Sales'].sort_values().plot(kind='barh', figsize=(8,6))

> [['Sales'] = Selecting only the Sales column

.sort_values() = Sorts sales values in ascending order

.plot() = Pandas built-in plotting function

kind='barh' = Here we create a horizontal bar chart

figsize=(8,6) = Sets the graph size.]


plt.title("Statewise Sales")

plt.show()
```

9. Conduct Groupwise Analysis

```
group_analysis = df.groupby('Group').sum()

group_analysis['Sales'].sort_values().plot(kind='barh', figsize=(8,6))

plt.title("Groupwise Sales")

plt.show()
```

10. Explore Timewise Analysis

```
df['Month'] = df.index.month
```

```
df.groupby('Month').sum()['Sales'].plot(kind='line')

plt.title("Month-wise Sales Trend")

plt.show()
```