IMPORTING ALL LIBRARIES

```python
import plotly.graph_objects as go
import pandas as pd
import numpy as np
from plotly.subplots import make_subplots
from sklearn.linear_model import LinearRegression
from datetime import datetime, timedelta
```

QUESTION 1

```python
# Dataset
data = {
    'Company': ['Apple', 'Microsoft', 'Amazon', 'Google', 'Facebook'],
    'Revenue_2022': [394, 198, 513, 280, 117],
    'Revenue_2023': [420, 215, 540, 310, 130]
}
df = pd.DataFrame(data)
# Creating the grouped bar chart
fig = go.Figure()
fig.add_trace(go.Bar(
    x=df['Company'],
    y=df['Revenue_2022'],
    name='Revenue 2022',
    marker_color='pink'
))
fig.add_trace(go.Bar(
    x=df['Company'],
    y=df['Revenue_2023'],
    name='Revenue 2023',
    marker_color='red'
))
# Layout customization
fig.update_layout(
    title='Annual Revenue Comparison (2022 vs 2023)',
    xaxis_title='Company',
    yaxis_title='Revenue (in billion dollars)',
    barmode='group',
    template='plotly_white'
)
# Show the figure
fig.show()
```

QUESTION 2

```python
# Dataset
data = {
    'Advertising Budget': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    'Sales Revenue': [15, 25, 40, 50, 65, 80, 85, 100, 120, 140]
}
df = pd.DataFrame(data)
# Creating the scatter plot
fig = px.scatter(
    df,
    x='Advertising Budget',
    y='Sales Revenue',
    title='Advertising Budget vs Sales Revenue',
    labels={'Advertising Budget': 'Advertising Budget (in $1000)',
'Sales Revenue': 'Sales Revenue (in $1000)'},
    color=df['Sales Revenue'],  # Color by revenue
    size=df['Sales Revenue'],    # Size by revenue
    color_continuous_scale='plasma' # Color scale
)
# Show the figure
fig.show()
```

QUESTION 3

```
# Dataset
data = {
    'Brand': ['Apple', 'Samsung', 'Xiaomi', 'Oppo', 'Vivo'],
    'Market Share': [30, 28, 17, 12, 13]
}
df = pd.DataFrame(data)
# Creating the pie chart
fig = px.pie(
    df,
    names='Brand',
    values='Market Share',
    title='Smartphone Market Share',
    hover_data=['Market Share'],
    labels={'Market Share': 'Percentage'},
    color_discrete_sequence=px.colors.qualitative.Set2
)
# Show percentage values on hover
fig.update_traces(textinfo='percent+label')
# Show the figure
fig.show()
```

QUESTION 4

```
# Dataset
data = {
    'Job Sector': ['IT', 'Finance', 'Healthcare', 'Education',
'Retail'] * 5,
    'Salary': [75000, 85000, 62000, 48000, 40000, 77000, 90000, 65000,
50000, 42000,
               78000, 87000, 67000, 52000, 43000, 80000, 89000, 69000,
54000, 45000,
               82000, 91000, 71000, 56000, 47000]
}
df = pd.DataFrame(data)
# Creating the box plot
fig = px.box(
    df,
    x='Job Sector',
    y='Salary',
    title='Salary Distribution Across Job Sectors',
    labels={'Salary': 'Salary (in USD)', 'Job Sector': 'Sector'},
    color='Job Sector',
    points='outliers'  # Highlight outliers
)
# Show the figure
fig.show()
```

QUESTION 5

```
# Dataset
data = {
    'GDP': [19352.46582, 47585.00101, 36867.70315, 30334.26573,
8644.913382, 8643.731496, 3846.096996, 43442.63114, 30454.63558,
35695.55631],
    'Inflation': [0.592630224, 4.864594335, 4.245991884, 1.455525998,
1.318212352, 1.325320294, 1.869090093, 2.861403942, 2.443752584,
1.810531131],
    'Unemployment': [10.34223474, 4.673926328, 6.505735782,
7.39634212, 8.472839811, 12.42211154, 5.396085386, 9.170813261,
10.10897483, 3.557404953],
    'Interest Rate': [6.467903667, 2.534717113, 1.585464337,
9.539969835, 9.690688298, 8.275576133, 3.741523923, 1.879049026,
7.158097239, 4.961372444]
}
df = pd.DataFrame(data)
# Compute correlation matrix
corr_matrix = df.corr()
# Generate heatmap
fig = ff.create_annotated_heatmap(
    z=corr_matrix.values,
    x=list(corr_matrix.columns),
    y=list(corr_matrix.index),
    annotation_text=np.round(corr_matrix.values, 2),
    colorscale='plasma',
    showscale=True
)
# Update layout
fig.update_layout(
    title='Correlation Matrix Heatmap',
    xaxis_title='Financial Indicators',
    yaxis_title='Financial Indicators'
)
# Show the figure
fig.show()
```

QUESTION 6

```
# Dataset
data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'],
    'Product A Sales': [500, 600, 700, 800, 750, 780],
    'Product B Sales': [400, 450, 470, 490, 520, 550],
    'Profit': [50, 80, 100, 120, 110, 130]  # Profit in thousand
dollars
}
df = pd.DataFrame(data)

# Creating subplots
fig = make_subplots(rows=2, cols=2, subplot_titles=("Product Sales",
"Sales Comparison", "Profit Trend"))

# Line chart for Product A Sales
fig.add_trace(go.Scatter(x=df['Month'], y=df['Product A Sales'],
mode='lines+markers', name='Product A Sales', marker_color='blue'),
row=1, col=1)

# Bar chart for Product A vs Product B Sales
fig.add_trace(go.Bar(x=df['Month'], y=df['Product A Sales'],
name='Product A', marker_color='green'), row=1, col=2)
fig.add_trace(go.Bar(x=df['Month'], y=df['Product B Sales'],
name='Product B', marker_color='yellow'), row=1, col=2)

# Scatter plot for Profit
fig.add_trace(go.Scatter(x=df['Month'], y=df['Profit'],
mode='markers', name='Profit', marker=dict(size=10, color='green')),
row=2, col=1)

# Update layout
fig.update_layout(
    title='Sales and Profit Analysis',
    showlegend=True,
    template='plotly_white'
)

# Show the figure
fig.show()
```

QUESTION 7

```
# Dataset
data = {
    'Country': ['USA', 'China', 'India', 'Germany', 'Brazil'] * 3,
    'Year': [2000, 2000, 2000, 2000, 2000, 2010, 2010, 2010, 2010,
2010, 2020, 2020, 2020, 2020, 2020],
    'GDP': [10, 5, 2, 3, 1, 15, 9, 5, 4, 2, 22, 14, 7, 5, 3],  # GDP
in trillion dollars
    'Population': [280, 1260, 1000, 83, 175, 310, 1350, 1200, 82, 190,
331, 1440, 1380, 80, 210],  # in million
    'Life Expectancy': [77, 71, 65, 80, 68, 79, 74, 69, 82, 72, 81,
76, 72, 83, 75]  # in years
}
df = pd.DataFrame(data)
# Creating animated scatter plot
fig = px.scatter(
    df,
    x='Population',
    y='Life Expectancy',
    size='GDP',
    color='Country',
    animation_frame='Year',
    hover_name='Country',
    title='Economic Growth Visualization (2000-2020)',
    labels={'Population': 'Population (in millions)', 'Life
Expectancy': 'Life Expectancy (in years)'},
    size_max=50,
    template='plotly_white'
)
# Show the figure
fig.show()
```

QUESTION 8

```
# Dataset
data = {
    'User A': ['Alice', 'Alice', 'Bob', 'Charlie', 'David', 'Eve',
'Frank', 'Grace', 'Hannah', 'Ivan'],
    'User B': ['Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Grace',
'Hannah', 'Ivan', 'Alice', 'Bob'],
    'Connection Strength': [1, 2, 3, 1, 4, 5, 2, 3, 4, 5]  #
Represents the strength of the relationship
}
df = pd.DataFrame(data)
# Create a graph
G = nx.Graph()

# Add edges with weights
for i in range(len(df)):
    G.add_edge(df.loc[i, 'User A'], df.loc[i, 'User B'],
weight=df.loc[i, 'Connection Strength'])
# Get positions using spring layout
pos = nx.spring_layout(G)
# Create edges
edge_x = []
edge_y = []
for edge in G.edges(data=True):
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.extend([x0, x1, None])
    edge_y.extend([y0, y1, None])

edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=0.5, color='#888'),
    hoverinfo='none',
    mode='lines'
)
# Create nodes
node_x = []
node_y = []
nodes = list(G.nodes())
for node in nodes:
    x, y = pos[node]
    node_x.append(x)
    node_y.append(y)

node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers+text',
    marker=dict(size=10, color='lightblue', line=dict(width=2)),
    text=nodes,
    textposition="top center"
)
# Create figure
fig = go.Figure(data=[edge_trace, node_trace],
                layout=go.Layout(
                    title='Network Graph Visualization',
                    showlegend=False,
                    hovermode='closest',
                    margin=dict(b=0, l=0, r=0, t=40)))
# Show the figure
fig.show()
```

QUESTION 9

```
# Define the raw data as a multiline string
s = """Date Stock_Price
01-01-2024 102.4836
```

```
02-01-2024 100.3188
03-01-2024 105.2586
04-01-2024 110.6455
05-01-2024 102.8696
06-01-2024 103.8798
07-01-2024 113.9567
08-01-2024 110.9079
09-01-2024 105.7334
10-01-2024 111.8037
11-01-2024 107.7839
12-01-2024 108.7825
13-01-2024 113.331
14-01-2024 103.5649
15-01-2024 105.5168
16-01-2024 112.3401
17-01-2024 111.0975
18-01-2024 118.743
19-01-2024 113.6417
20-01-2024 112.1304
21-01-2024 127.5303
22-01-2024 120.0832
23-01-2024 122.5599
24-01-2024 116.1086
25-01-2024 121.5205
26-01-2024 125.8071
27-01-2024 120.5077
28-01-2024 129.1512
29-01-2024 125.2796
30-01-2024 127.8345
31-01-2024 127.2945
01-02-2024 140.5745
02-02-2024 132.2557
03-02-2024 128.0448
04-02-2024 138.4562
05-02-2024 129.2493
06-02-2024 137.408
07-02-2024 127.5754
08-02-2024 131.7429
09-02-2024 140.3782
10-02-2024 144.0964
11-02-2024 142.271
12-02-2024 141.846
13-02-2024 141.9288
14-02-2024 137.0518
15-02-2024 141.8553
16-02-2024 144.1615
17-02-2024 152.7604
18-02-2024 150.2029
19-02-2024 140.6797
```

```
20-02-2024 152.1255
21-02-2024 149.5897
22-02-2024 149.1406
23-02-2024 156.5937
24-02-2024 159.7005
25-02-2024 160.212
26-02-2024 152.3696
27-02-2024 156.0297
28-02-2024 160.2422
29-02-2024 164.4737
01-03-2024 158.2102
02-03-2024 160.6879
03-03-2024 157.0946
04-03-2024 157.6553
05-03-2024 168.7091
06-03-2024 172.4378
07-03-2024 166.3066
08-03-2024 172.6944
09-03-2024 170.495
10-03-2024 166.4714
11-03-2024 172.514
12-03-2024 179.4074
13-03-2024 172.5481
14-03-2024 181.5606
15-03-2024 161.6487
16-03-2024 179.8671
17-03-2024 177.2029
18-03-2024 176.2827
19-03-2024 179.2467
20-03-2024 169.8601
21-03-2024 179.7097
22-03-2024 183.6037
23-03-2024 190.2178
24-03-2024 181.247
25-03-2024 180.806
26-03-2024 183.3498
27-03-2024 191.4457
28-03-2024 189.5225
29-03-2024 186.2401
30-03-2024 192.4653
31-03-2024 191.3945
01-04-2024 196.7624
02-04-2024 189.419
03-04-2024 192.3011
04-04-2024 192.989
05-04-2024 188.642
06-04-2024 198.4503
07-04-2024 199.2851
08-04-2024 199.0155
```

```python
09-04-2024 198.8271"""

# Convert the string data into a DataFrame
data_lines = [line.split(" ") for line in s.split('\n')]
df = pd.DataFrame(data_lines[1:], columns=data_lines[0])

# Convert columns to appropriate data types
df['Date'] = pd.to_datetime(df['Date'], format='%d-%m-%Y')
df['Stock_Price'] = pd.to_numeric(df['Stock_Price'])

# Calculate the number of days from the start date
df['Days'] = (df['Date'] - df['Date'].min()).dt.days

# Prepare data for linear regression
X = df['Days'].values.reshape(-1, 1)  # Independent variable (days)
y = df['Stock_Price'].values  # Dependent variable (stock price)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X, y)

# Generate future predictions
last_date = df['Date'].max()
future_days = 30
future_dates = [last_date + timedelta(days=i+1) for i in
range(future_days)]
future_days_numeric = [df['Days'].max() + i + 1 for i in
range(future_days)]

# Predict stock prices
predicted_stock_prices = model.predict(X)  # Predictions for existing
data
future_predictions =
model.predict(np.array(future_days_numeric).reshape(-1, 1))  # Future
predictions

# Assign predicted values to a new column in the DataFrame
df['Predicted'] = predicted_stock_prices  # Assign predictions to
'Predicted' column


# Create a figure object
fig = go.Figure()

# Add historical stock prices (actual values) as a line plot with
markers
fig.add_trace(go.Scatter(
    x=df['Date'],
    y=df['Stock_Price'],
    mode='lines+markers',  # Both lines and markers for better
visibility
    name='Historical Stock Price',
    line=dict(color='blue'),  # Set line color to blue
    marker=dict(size=4)  # Adjust marker size
))

# Add the regression line (predicted stock prices based on the model)
fig.add_trace(go.Scatter(
    x=df['Date'],
    y=df['Predicted'],  # Now using the 'Predicted' column
    mode='lines',  # Line only
    name='Regression Line',
    line=dict(color='red', dash='dash')  # Red dashed line for
regression
))

# Add future stock price predictions based on the model
fig.add_trace(go.Scatter(
    x=future_dates,
    y=future_predictions,
    mode='lines',  # Line only
    name='Future Prediction',
    line=dict(color='green', dash='dash')  # Green dashed line for
future predictions
))

# Update layout with title, axis labels, legend, and hover settings
fig.update_layout(
    title='Stock Price History with Linear Regression Prediction',
    xaxis_title='Date',
    yaxis_title='Stock Price ($)',
    legend_title='Data Type',
    hovermode='x unified',  # Ensures hover data is unified across
traces
    width=1000,
    height=600
)

# Highlight the future prediction region with a shaded rectangle
fig.add_shape(
    type="rect",
    xref="x",
    yref="paper",  # Span the entire height
    x0=last_date,  # Start from the last known date
    y0=0,
    x1=future_dates[-1],  # Extend to the last future date
    y1=1,
    fillcolor="lightblue",  # Light green for prediction zone
    opacity=0.2,  # Make it slightly transparent
    layer="below",
    line_width=0  # No border for the rectangle
)

# Get model parameters: slope (coefficient), intercept, and R-squared
value
slope = model.coef_[0]
intercept = model.intercept_
r_squared = model.score(X, y)  # Goodness-of-fit measure

# Add annotation displaying the regression equation and R-squared
value
fig.add_annotation(
    xref="paper",
    yref="paper",
    x=0.02,
    y=0.98,
    text=f"Model: y = {slope:.2f}x + {intercept:.2f}<br>R² =
{r_squared:.4f}",
    showarrow=False,
    bgcolor="white",
    bordercolor="orange",
    borderwidth=1
)
# Display the figure
fig.show()

# Load the Iris dataset
df = pd.read_csv("https://raw.githubusercontent.com/mwaskom/seaborn-
data/master/iris.csv")
# Scatter Plot
fig1 = px.scatter(df, x="petal_length", y="petal_width",
color="species",
                  title="Petal Length vs Petal Width",
                  labels={"petal_length": "Petal Length (cm)",
"petal_width": "Petal Width (cm)"})
fig1.show()
# 3D Scatter Plot
fig2 = px.scatter_3d(df, x="sepal_length", y="sepal_width",
z="petal_length",
                     color="species", title="3D Scatter Plot of Iris
Dataset",
                     labels={"sepal_length": "Sepal Length (cm)",
"sepal_width": "Sepal Width (cm)", "petal_length": "Petal Length
(cm)"})
fig2.show()
# Pair Plot
fig3 = px.scatter_matrix(df, dimensions=["sepal_length",
"sepal_width", "petal_length", "petal_width"],
                         color="species", title="Pairwise
```

```
Relationships in Iris Dataset")
fig3.show()
```