1) Acquiring and plotting data

```
import pandas as pd
import matplotlib.pyplot as plt

# Acquiring data
# Assuming you have a CSV file named 'data.csv' with columns 'x' and 'y'
data = pd.read_csv('data.csv')

# Plotting data
plt.plot(data['x'], data['y'])
plt.xlabel('X-axis Label')  # Set X-axis label
plt.ylabel('Y-axis Label')  # Set Y-axis label
plt.title('Data Plot')  # Set title
plt.grid(True)  # Add grid
plt.show()  # Show the plot
```

2) Statistical Analysis – such as Multivariate Analysis, PCA, LDA, Correlation regression and analysis of variance

```
# Import the required libraries
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.feature_selection import f_classif
from scipy.stats import pearsonr
from statsmodels.formula.api import ols
import statsmodels.api as sm

# Load the dataset
# Assuming you have a dataset in a pandas DataFrame called 'df'
```

```
# Perform Multivariate Analysis
# Example of PCA
pca = PCA(n_components=3)
X_pca = pca.fit_transform(df) # Assuming 'df' contains your data
# 'X_pca' now contains the transformed data with reduced dimensionality using PCA

# Perform Linear Discriminant Analysis (LDA)
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]  # Target variable
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)  # 'X_lda' now contains the transformed data using LDA

# Perform Correlation Analysis
# Example of Pearson's correlation coefficient
corr_coefficient, p_value = pearsonr(df['feature1'], df['feature2'])
print("Correlation coefficient: ", corr_coefficient)
print("p-value: ", p_value)

# Perform Regression
# Example of linear regression
X = df[['feature1', 'feature2']]  # Features
y = df['target']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
y_pred = regression_model.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R2 score: ", r2)

# Perform Analysis of Variance (ANOVA)
# Example of one-way ANOVA
model = ols('target ~ C(group)', data=df).fit()  # Assuming 'group' is the categorical variable
anova_table = sm.stats.anova_lm(model, typ=2)
```

```
print(anova_table)
```

3) Financial analysis using Clustering, Histogram and HeatMap.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

# Load financial data
financial_data = pd.read_csv('financial_data.csv')

# Perform clustering using KMeans
X = financial_data[['Income', 'Savings']] # Extract relevant features
kmeans = KMeans(n_clusters=3, random_state=0).fit(X) # Perform KMeans clustering
labels = kmeans.labels_ # Get cluster labels

# Add cluster labels to the financial data
financial_data['Cluster'] = labels

# Plot clusters on a scatter plot
sns.scatterplot(x='Income', y='Savings', hue='Cluster', data=financial_data)
plt.xlabel('Income')
plt.ylabel('Savings')
plt.title('Financial Data Clustering')
plt.show()

# Generate histograms for Income and Savings
sns.histplot(financial_data['Income'], bins=10)
plt.xlabel('Income')
plt.ylabel('Count')
plt.title('Income Distribution')
plt.show()
```

```
sns.histplot(financial_data['Savings'], bins=10)
plt.xlabel('Savings')
plt.ylabel('Count')
plt.title('Savings Distribution')
plt.show()

# Generate a heatmap to visualize correlation between Income and Savings
sns.heatmap(financial_data[['Income', 'Savings']].corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap (Income vs. Savings)')
plt.show()
```

4) Time-series analysis – stock market.

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Load stock market data
df = pd.read_csv('stock_data.csv', index_col='Date', parse_dates=True)

# Inspect data
print(df.head())

# Visualize stock prices
plt.figure(figsize=(10, 6))
plt.plot(df['Close'])
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Stock Price over Time')
plt.show()

# Perform ARIMA modeling
model = ARIMA(df['Close'], order=(1, 0, 0))  # ARIMA(1, 0, 0) model
```

```
results = model.fit()

# Forecast stock prices
forecast = results.forecast(steps=30)  # Forecast 30 steps ahead

# Visualize forecasted stock prices
plt.figure(figsize=(10, 6))
plt.plot(df['Close'], label='Observed')
plt.plot(forecast, label='Forecasted')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.title('Forecasted Stock Price')
plt.legend()
plt.show()
```

5) Visualization of various massive dataset - Finance - Healthcare - Census – Geospatial

(i) Finance Dataset Visualization with Pandas and Matplotlib:

```
import pandas as pd
import matplotlib.pyplot as plt

# Load finance dataset into a Pandas DataFrame
finance_df = pd.read_csv('finance_dataset.csv')

# Group by date and calculate total revenue
revenue_by_date = finance_df.groupby('date')['revenue'].sum()

# Line plot of total revenue by date
plt.figure(figsize=(10, 6))
revenue_by_date.plot()
plt.title('Total Revenue by Date')
plt.xlabel('Date')
```

```
plt.ylabel('Total Revenue')
plt.show()
```

(ii) Healthcare Dataset Visualization with Seaborn:

```
import seaborn as sns
# Load healthcare dataset into a Pandas DataFrame
healthcare_df = pd.read_csv('healthcare_dataset.csv')

# Box plot of BMI by gender
plt.figure(figsize=(8, 6))
sns.boxplot(x='gender', y='bmi', data=healthcare_df)
plt.title('BMI by Gender')
plt.xlabel('Gender')
plt.ylabel('BMI')
plt.show()
```

(iii) Census Dataset Visualization with Pandas and Plotly:

```
import pandas as pd
import plotly.express as px

# Load census dataset into a Pandas DataFrame
census_df = pd.read_csv('census_dataset.csv')

# Group by state and calculate total population
population_by_state = census_df.groupby('state')['population'].sum().reset_index()

# Choropleth map of total population by state
fig = px.choropleth(population_by_state,
                    locations='state',
                    locationmode='USA-states',
                    color='population',
                    color_continuous_scale='Viridis',
                    title='Total Population by State')
fig.update_geos(projection_type='albers usa')
```

```
fig.show()
```

(iv) Geospatial Dataset Visualization with Geopandas and Matplotlib:

```
import geopandas as gpd
import matplotlib.pyplot as plt

# Load geospatial dataset into a Geopandas DataFrame
geospatial_df = gpd.read_file('geospatial_dataset.shp')

# Plot geospatial data
fig, ax = plt.subplots(figsize=(10, 10))
geospatial_df.plot(ax=ax, column='category', legend=True)
ax.set_title('Geospatial Data Visualization')
plt.show()
```

6) Visualization on Streaming dataset (Stock market dataset, weather forecasting)

```
import matplotlib.pyplot as plt
import pandas as pd

# Example stock market streaming dataset
# Replace with your actual streaming dataset
stock_df = pd.read_csv('https://streamsource.com/stockdata')

# Example weather forecasting streaming dataset
# Replace with your actual streaming dataset
weather_df = pd.read_csv('https://streamsource.com/weatherdata')

# Set up the plot
fig, ax = plt.subplots(2, 1, figsize=(10, 8))

# Plot the stock market data
ax[0].plot(stock_df['timestamp'], stock_df['price'])
```

```
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Price')
ax[0].set_title('Stock Market Data')

# Plot the weather forecasting data
ax[1].plot(weather_df['timestamp'], weather_df['temperature'])
ax[1].set_xlabel('Time')
ax[1].set_ylabel('Temperature')
ax[1].set_title('Weather Forecasting Data')

# Show the plot
plt.show()
```

7) Market-Basket Data analysis-visualization

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import matplotlib.pyplot as plt

# Load market-basket data
data = pd.read_csv('market_basket_data.csv', header=None)
# Pre-process the data (e.g., converting to a list of lists)
data_list = data.values.tolist()

# Perform Apriori algorithm to mine frequent itemsets
frequent_itemsets = apriori(data_list, min_support=0.05, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)
# Filter rules based on desired metrics (e.g., confidence, lift)
filtered_rules = rules[(rules['confidence'] >= 0.7) & (rules['lift'] > 1)]
```

```
# Visualize association rules
plt.figure(figsize=(10, 5))
plt.scatter(filtered_rules['confidence'], filtered_rules['lift'], alpha=0.5)
plt.xlabel('Confidence')
plt.ylabel('Lift')
plt.title('Association Rules')
plt.show()
```

8) Text visualization using web analytics

Step 1: Install Required Libraries

```
!pip install pandas matplotlib
```

Step 2: Load Web Analytics Data

```
import pandas as pd

# Load web analytics data from CSV file
web_data = pd.read_csv('web_analytics_data.csv')

# Preview the data
print(web_data.head())
```

Step 3: Analyze Text Data

```
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt

# Load web analytics data from CSV file
web_data = pd.read_csv('web_analytics_data.csv')

# Extract text data
text_data = ' '.join(web_data['text_column'])  # Replace 'text_column' with the actual column name containing text data

# Tokenize words
tokens = text_data.split()
```

```
# Count word frequencies
word_freqs = Counter(tokens)

# Get top N words by frequency
top_n = 10
top_words = word_freqs.most_common(top_n)

# Extract word and frequency data
words = [word for word, freq in top_words]
freqs = [freq for word, freq in top_words]

# Create bar chart of word frequencies
plt.figure(figsize=(10, 6))
plt.bar(words, freqs)
plt.title('Top {} Words by Frequency'.format(top_n))
plt.xlabel('Words')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```

Step 4: Visualize Text Data

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Load web analytics data from CSV file
web_data = pd.read_csv('web_analytics_data.csv')

# Extract text data
text_data = ' '.join(web_data['text_column'])  # Replace 'text_column' with the actual column name containing text data

# Create word cloud
wordcloud = WordCloud(width=800, height=400, max_words=100, background_color='white').generate(text_data)
```

```
# Plot word cloud
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Web Analytics Data')
plt.show()
```

9) Google API with maps

```
import googlemaps

# Set up the Google Maps API client
gmaps = googlemaps.Client(key='YOUR_API_KEY')

# Example 1: Geocoding - convert an address to latitute and longitude
address = '1600 Amphitheatre Parkway, Mountain View, CA'
geocode_result = gmaps.geocode(address)
latitude = geocode_result[0]['geometry']['location']['lat']
longitude = geocode_result[0]['geometry']['location']['lng']
print(f'Latitude: {latitude}, Longitude: {longitude}')

# Example 2: Reverse geocoding - convert latitude and longitude to an address
latitude = 37.4219999
longitude = -122.0840575
reverse_geocode_result = gmaps.reverse_geocode((latitude, longitude))
formatted_address = reverse_geocode_result[0]['formatted_address']
print(f'Formatted Address: {formatted_address}')

# Example 3: Places search - search for nearby places
latitude = 37.4219999
longitude = -122.0840575
places_result = gmaps.places_nearby(location=(latitude, longitude), radius=5000, type='restaurant')
for place in places_result['results']:
```

```
name = place['name']
address = place['vicinity']
print(f'Name: {name}, Address: {address}')
```

10) Network Visualization using Gephi

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
import random
from gephistreamer import graph
from gephistreamer import streamer

# Create a sample network data
# You can replace this with your own data
edges = [(1, 2), (1, 3), (2, 3), (3, 4), (4, 5), (5, 6), (5, 7), (6, 7), (7, 8), (8, 9), (9, 10), (10, 11), (11, 12),
(11, 13)]

# Create a NetworkX graph
G = nx.DiGraph()
G.add_edges_from(edges)

# Generate node labels
node_labels = {node: f"Node {node}" for node in G.nodes()}

# Generate random node attributes for demonstration
node_attributes = {node: random.randint(1, 100) for node in G.nodes()}

# Convert NetworkX graph to Gephi graph
gephi_graph = graph.Graph("my_graph")
for edge in G.edges():
    gephi_graph.add_edge(graph.Edge(node_labels[edge[0]], node_labels[edge[1]]))

# Set node attributes
for node in G.nodes():
    gephi_graph.add_node(graph.Node(node_labels[node], label=node_labels[node],
size=node_attributes[node]))

# Create a Gephi streaming session
stream = streamer.Streamer(streamer.GephiWebSocket())

# Stream the graph to Gephi
stream.force_start()
stream.send_graph(gephi_graph)
stream.stop()

# Open Gephi and visualize the graph
# You can manually open Gephi, go to the "Streaming" tab, and click "Start" to see the visualization
```

11) Visualization of reconstruction network using Qlickview

```
# Import necessary Python libraries for visualization
import matplotlib.pyplot as plt
import numpy as np

# Generate sample data for reconstruction network
# Replace this with your actual data or data loading logic
x = np.linspace(-5, 5, 100)
y = x**2 + np.random.normal(0, 2, 100)

# Create reconstruction network plot using Matplotlib
fig, ax = plt.subplots()
ax.plot(x, y, 'b', label='Original Data')
ax.plot(x, x**2, 'r', label='Reconstructed Data')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Reconstruction Network')
ax.legend()

# Save the plot as an image
plt.savefig('reconstruction_network.png')

# Optional: Show the plot in Python
plt.show()
```

12) Dash Board Creation using Tableau

To create a dashboard in Tableau using Python, you can use the Tableau Python Server (TabPy) API. Here are the steps:

1. Install TabPy: You can install TabPy using pip by running the command **pip install tabpy** in your command prompt or terminal.
2. Start TabPy: You can start TabPy by running the command **tabpy** in your command prompt or terminal.
3. Connect to TabPy: In Python, you can connect to TabPy using the **tabpy_tools** module. You can install it using pip by running the command **pip install tabpy_tools**.
4. Load your data: Load your data into Python using your preferred data analysis library, such as Pandas.
5. Create a function: Create a Python function that performs the data analysis and returns the result in a format that can be used in Tableau. This function should take in parameters that can be used to filter or customize the data.
6. Publish the function: Use the **tabpy_tools** module to publish the function to TabPy.
7. Create a dashboard in Tableau: Use Tableau to create a new dashboard or modify an existing one.
8. Add a Python script: Add a new Python script to your Tableau dashboard by selecting "Script" from the "Objects" menu and pasting in your Python code.
9. Use the Python script in your dashboard: Use the Python script to create a new visualization or modify an existing one.

Step 1: Install Tableau Server Client (TSC) library

```
!pip install tableauserverclient
```

Step 2: Connect to Tableau Server

```
from tableauserverclient import Server

# Connect to Tableau Server
server = Server('https://your-tableau-server-url', use_server_version=True)
server.auth.sign_in('username', 'password')
```

Step 3: Create a Dashboard

```
from tableauserverclient import WorkbookItem, DatasourceItem, ProjectItem
from tableauserverclient.server.endpoint.enums import Permission

# Create a new project
project = ProjectItem(name='My Project', content_permissions={Permission.View: ['All Users']})
project = server.projects.create(project)

# Publish a workbook to the project
workbook_path = '/path/to/your/workbook.twbx'  # Replace with the path to your Tableau workbook
workbook_item = WorkbookItem(project_id=project.id, file_path=workbook_path)
workbook_item = server.workbooks.publish(workbook_item)
```

Step 4: Create a Dashboard using TSC

```
from tableauserverclient import Sheet, ZoneItem, DashboardItem

# Create a sheet object
sheet = Sheet(workbook_item, workbook_item.worksheets[0].id)

# Add a sheet to the dashboard
zone_item = ZoneItem(sheet)
zone_item.height = 500
zone_item.width = 800
dashboard_item = DashboardItem(workbook_item, is_hidden=True)
dashboard_item.zones.append(zone_item)
dashboard_item = server.dashboards.publish(dashboard_item)
```

Step 5: Retrieve the Dashboard Output

```
# Retrieve the URL of the dashboard
dashboard_url = server.dashboards.get_by_id(dashboard_item.id).url

# You can use the dashboard_url to open the dashboard in a web browser or extract data from it using a web scraping library like Beautiful Soup or Selenium
```

```
# For example, you can use requests and Beautiful Soup to retrieve the HTML content of the dashboard page
import requests
from bs4 import Beautiful Soup

response = requests.  Get (dashboard url)
soup = Beautiful Soup (response
```