1.

a)
```python
import pandas as pd

# Step 1: Create a DataFrame from a dictionary
data = {
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
}
df = pd.DataFrame(data)

# Display the DataFrame
print("Original DataFrame:")
print(df)

# Step 2: Select a column
print("\nSelect column 'A':")
print(df['A'])

# Step 3: Select multiple columns
print("\nSelect columns 'A' and 'B':")
print(df[['A', 'B']])

# Step 4: Select a row by index
print("\nSelect row with index 1:")
print(df.iloc[1])

# Step 5: Delete a column
df = df.drop(columns=['C'])
print("\nDataFrame after deleting column 'C':")
print(df)

# Step 6: Delete a row
```

```python
df = df.drop(index=0)
print("\nDataFrame after deleting row with index 0:")
print(df)

# Step 7: Add a new column
df['D'] = [10, 11]  # Adding a new column
print("\nDataFrame after adding column 'D':")
print(df)

# Step 8: Add a new row
new_row = pd.Series({'A': 4, 'B': 7, 'D': 12})
df = df.append(new_row, ignore_index=True)
print("\nDataFrame after adding a new row:")
print(df)


# Step 9: Rename a column
df = df.rename(columns={'A': 'Alpha', 'B': 'Beta'})
print("\nDataFrame after renaming columns:")
print(df)
```

b)

```
import pandas as pd

# Sample data
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve', 'Frank', 'Alice'],
    'score': [85, 95, 85, 70, 95, 60, 90]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Sort the DataFrame first by 'name' in descending order, then by 'score' in ascending order
a=df.sort_values(by=['name','score'],ascending=[ False,True])

# Display the sorted DataFrame
print("\nSorted DataFrame:")
print(a)
```

2

a)

```
import pandas as pd

# Sample data
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'attempts': [1, 3, 2, 4, 1],
    'score': [85, 95, 75, 80, 90]
}

# Create a DataFrame
df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Select rows where the number of attempts is greater than 2
filtered_df = df[df['attempts'] > 2]

# Display the filtered DataFrame
print("\nFiltered DataFrame (attempts > 2):")
print(filtered_df)
```

b)
```
import pandas as pd

# Sample data
```

```
data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'attempts': [1, 3, 2, 4, 1],
    'score': [85, 95, 75, 80, 90]
}

# Create the original DataFrame
df = pd.DataFrame(data)

# Display the original DataFrame
print("Original DataFrame:")
print(df)

# Step 1: Append a new row 'k' with given values
new_row = {'name': 'K', 'attempts': 2, 'score': 88}
df = df.append(new_row, ignore_index=True)

# Display the DataFrame after appending the new row
print("\nDataFrame after appending new row 'k':")
print(df)

# Step 2: Delete the new row
df = df.drop(index=len(df) - 1)  # Drop the last row (the new row 'k')

# Display the DataFrame after deleting the new row
print("\nDataFrame after deleting the new row 'k':")
print(df)
```

3

A_) import matplotlib.pyplot as plt

```
import numpy as np
```

```python
# Sample data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Create a plot
plt.plot(x, y1, label='Sine Wave')
plt.plot(x, y2, label='Cosine Wave')

# Add a legend with custom vertical spacing
plt.legend(handletextpad=2, labelspacing=1.5)  # Adjust these values as needed

# Add titles and labels
plt.title('Sine and Cosine Waves')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

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

b) import pandas as pd

```python
import matplotlib.pyplot as plt

# Load the dataset from a CSV file
# Make sure to replace 'data.csv' with the path to your actual CSV file
```

```
# The CSV file should have columns named 'x' and 'y' for this example
df = pd.read_csv('data.csv')

# Display the first few rows of the DataFrame (optional)
print("Data from CSV:")
print(df.head())

# Create a plot
plt.figure(figsize=(10, 6))
plt.plot(df['x'], df['y'], color='blue', label='Data Line')  # Change 'blue' to your desired color

# Add titles and labels
plt.title('Graph Plot from CSV Data')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')

# Add a legend
plt.legend()

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

5) import pandas as pd

```
import matplotlib.pyplot as plt

# Given data
data = [2, 3, 1, 4, 2, 2, 3, 1, 4, 4, 4, 2, 2, 2]
```

```
# Create a frequency distribution table
frequency_distribution = pd.Series(data).value_counts().sort_index()

# Display the frequency distribution table
print("Frequency Distribution Table:")
print(frequency_distribution)

# Plotting the frequency distribution curve (histogram)
plt.figure(figsize=(8, 5))
plt.hist(data)

# Adding titles and labels
plt.title('Frequency Distribution Curve')
plt.xlabel('Values')
plt.ylabel('Frequency')


# Show the plot # Set x-ticks to match the data values
plt.show()
```

7) import scipy.stats as stats

```
# Input mean, standard deviation, and the value (x) for which you want to find the probability
mean = float(input("Enter the mean: "))
std_dev = float(input("Enter the standard deviation: "))
x = float(input("Enter the value of x: "))
```

```
# Calculate the cumulative probability (area under the normal curve to the left of x)
probability = stats.norm.cdf(x, mean, std_dev)

print(f"the probability of getting value lesser than or equal to {x} is {probability:.4f}")
```

9)import numpy as np

```
import matplotlib.pyplot as plt

# Example data (you can replace with your own data)
x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 5, 4, 5])

# Calculate the Pearson correlation coefficient
```

```
correlation_matrix = np.corrcoef(x, y)
pearson_corr = correlation_matrix[0, 1]

print(f"Pearson correlation coefficient: {pearson_corr:.4f}")

# Plotting the data points
plt.scatter(x, y, color='blue', label='Data Points')
plt.title('Scatter plot of data points')
plt.xlabel('X values')
plt.ylabel('Y values')

# Add the line of best fit for visualizing correlation
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b, color='red', label='Line of Best Fit')

plt.legend()
plt.grid(True)
plt.show()
```

10) import numpy as np

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Given data
x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# Create the linear regression model
```

```
model = LinearRegression()

# Fit the model to the data
model.fit(x, y)

# Get the coefficient (slope) and intercept
slope = model.coef_[0]
intercept = model.intercept_

print(f"Slope (coefficient): {slope:.4f}")
print(f"Intercept: {intercept:.4f}")

# Predict y values using the model
y_pred = model.predict(x)

# Plot the data and the regression line
plt.scatter(x, y, color='blue', label='Data Points')
plt.plot(x, y_pred, color='red', label=f'Best Fit Line: y = {slope:.2f}x + {intercept:.2f}')

plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```

11)
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Example dataset with two independent variables (features)
# Replace with your own data if needed!
X = np.array([
    [1, 2],
    [2, 3],
    [4, 5],
```

```
    [3, 2],
    [5, 4],
    [6, 7],
    [7, 8],
    [8, 7],
    [9, 10],
    [10, 9]
])
y = np.array([10, 12, 20, 15, 25, 30, 35, 38, 40, 45])

# Create the multiple linear regression model
model = LinearRegression()

# Fit the model to the data
model.fit(X, y)

# Get the coefficients and intercept
coefficients = model.coef_
intercept = model.intercept_
variance_score = model.score(X, y)

print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept:.4f}")
print(f"Variance score (R^2): {variance_score:.4f}")

# Predict the y values
y_pred = model.predict(X)

# Calculate residual errors
residuals = y - y_pred

# Plot the residuals
plt.scatter(y_pred, residuals, color='blue', label='Residuals')
plt.hlines(y=0, xmin=min(y_pred), xmax=max(y_pred), color='red', linestyle='--', label='Zero Error Line')
plt.xlabel('Predicted values')
```

```
plt.ylabel('Residuals')
plt.title('Residual Plot for Multiple Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

12)

a) import pandas as pd

```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt

# Example dataset
data = {
    'values': [65, 70, 68, 72, 66, 74, 69, 71, 67, 73]
}
df = pd.DataFrame(data)
```

```
# Evaluate data distribution
mean_sample = df['values'].mean()
std_sample = df['values'].std(ddof=1)
print("Sample Mean:", mean_sample)
print("Sample Standard Deviation:", std_sample)

# Plot histogram to check data distribution
plt.hist(df['values'], bins=5, color='skyblue', edgecolor='black')
plt.title('Data Distribution')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.grid(axis='y')
plt.show()

# Population mean (e.g., known value)
pop_mean = 70

# Calculate Z-test statistic
n = len(df)
standard_error = std_sample / np.sqrt(n)
z_score = (mean_sample - pop_mean) / standard_error
p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))

print("Z-test statistic:", z_score)
print("P-value:", p_value)

# Hypothesis Testing
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference.")
else:
    print("Fail to reject the null hypothesis: No significant difference.")
```

b) import numpy as np

```python
# Two independent samples (replace these with your actual data)
sample1 = np.array([2, 4, 7, 5, 9, 6, 8, 10])
sample2 = np.array([3, 5, 6, 7, 5, 4, 6, 7])

# Sample sizes
n1 = len(sample1)
n2 = len(sample2)

# Sample means
mean1 = np.mean(sample1)
mean2 = np.mean(sample2)

# Sample standard deviations
std1 = np.std(sample1, ddof=1)
std2 = np.std(sample2, ddof=1)

# Calculate the t-statistic for independent samples
pooled_se = np.sqrt(std1**2 / n1 + std2**2 / n2)
t_statistic = (mean1 - mean2) / pooled_se

# Degrees of freedom (Welch-Satterthwaite approximation)
df_num = (std1**2 / n1 + std2**2 / n2)**2
df_den = ((std1**2 / n1)**2 / (n1 - 1)) + ((std2**2 / n2)**2 / (n2 - 1))
df = df_num / df_den

print(f"Sample 1 Mean: {mean1}")
print(f"Sample 2 Mean: {mean2}")
print(f"t-statistic: {t_statistic:.4f}")
print(f"Degrees of freedom: {df:.4f}")
```

13

a) import numpy as np

from scipy import stats

Sample data

ages = np.array([45, 89, 23, 46, 12, 69, 45, 24, 34, 67])

Hypothesized population mean (for example, let's assume 50)

pop_mean = 50

Calculate sample mean and standard deviation

mean_sample = np.mean(ages)

std_sample = np.std(ages, ddof=1)

n = len(ages)

```
# Calculate t-statistic
t_statistic = (mean_sample - pop_mean) / (std_sample / np.sqrt(n))

# Calculate p-value (two-tailed)
p_value = 2 * (1 - stats.t.cdf(abs(t_statistic), df=n-1))

print(f"Sample mean: {mean_sample:.2f}")
print(f"Sample standard deviation: {std_sample:.2f}")
print(f"T-statistic: {t_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Determine if result is significant at alpha = 0.05
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The sample mean is significantly different from the population mean.")
else:
    print("Fail to reject the null hypothesis: No significant difference.")
```

b) import pandas as pd

```
import numpy as np
from scipy import stats

# Sample data
data = {
    'sample1': [45, 89, 23, 46, 12, 69, 45, 24, 34, 67],
    'sample2': [50, 60, 55, 65, 62, 58, 53, 61, 59, 57]
}
df = pd.DataFrame(data)

# Calculate means, stds, and sizes
mean1, mean2 = df['sample1'].mean(), df['sample2'].mean()
std1, std2 = df['sample1'].std(ddof=1), df['sample2'].std(ddof=1)
n1, n2 = len(df['sample1']), len(df['sample2'])
```

```
# Compute Z-test statistic
se = np.sqrt(std1**2/n1 + std2**2/n2)
z = (mean1 - mean2) / se

# Two-tailed p-value
p = 2 * (1 - stats.norm.cdf(abs(z)))

print(f"Z-statistic: {z:.4f}")
print(f"P-value: {p:.4f}")

if p < 0.05:
    print("Reject null hypothesis: means differ.")
else:
    print("Fail to reject null hypothesis: no difference.")
```

14

a) import numpy as np

```
from scipy import stats

# Sample data
data = np.array([45, 89, 23, 46, 12, 69, 45, 24, 34, 67])

# Hypothesized population mean
pop_mean = 50

# Perform one-sample t-test
t_statistic, p_value = stats.ttest_1samp(data, pop_mean)

print(f"T-statistic: {t_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret result
```

```
alpha = 0.05
if p_value < alpha:
    print("Reject null hypothesis: Sample mean is significantly different from population mean.")
else:
    print("Fail to reject null hypothesis: No significant difference from population mean.")
```

b) import numpy as np

```
from scipy import stats

# Generate dummy voter ages for Minnesota (e.g., 100 voters, ages 18 to 90)
np.random.seed(42)  # For reproducibility
minnesota_ages = np.random.randint(18, 90, size=100)

# Hypothesized population mean age (e.g., 45 years)
population_mean = 45

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(minnesota_ages, population_mean)

print(f"Sample mean age: {minnesota_ages.mean():.2f}")
print(f"T-statistic: {t_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpret the result
```

```
alpha = 0.05
if p_value < alpha:
    print("Reject null hypothesis: The average age of Minnesota voters differs from the population mean.")
else:
    print("Fail to reject null hypothesis: No significant difference in average age.")
```

15)

a)
```
import numpy as np
from scipy import stats

# Sample data: Three groups
group1 = [23, 45, 56, 67, 45]
group2 = [34, 56, 67, 78, 56]
group3 = [23, 45, 56, 67, 50]

# Perform one-way ANOVA
f_statistic, p_value = stats.f_oneway(group1, group2, group3)

print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

alpha = 0.05
if p_value < alpha:
    print("Reject null hypothesis: At least one group mean is different.")
else:
```

```
    print("Fail to reject null hypothesis: No significant difference between group means.")
```

b) import pandas as pd

```
import statsmodels.api as sm
from statsmodels.formula.api import ols

# Sample data: Dependent variable, and two factors (categorical)
data = {
    'score': [23, 45, 56, 67, 45, 34, 56, 67, 78, 56, 25, 40, 55, 60, 50, 30, 47, 52, 65, 59],
    'factor_A': ['Low', 'Low', 'Low', 'Low', 'Low', 'High', 'High', 'High', 'High', 'High',
                 'Low', 'Low', 'Low', 'Low', 'Low', 'High', 'High', 'High', 'High', 'High'],
    'factor_B': ['Type1', 'Type1', 'Type2', 'Type2', 'Type3', 'Type1', 'Type1', 'Type2', 'Type2', 'Type3',
                 'Type1', 'Type1', 'Type2', 'Type2', 'Type3', 'Type1', 'Type1', 'Type2', 'Type2', 'Type3']
}

df = pd.DataFrame(data)

# Define the model with interaction term
model = ols('score ~ C(factor_A) + C(factor_B) + C(factor_A):C(factor_B)', data=df).fit()

# Perform two-way ANOVA
anova_table = sm.stats.anova_lm(model, typ=2)
```

```
print(anova_table)
```

16

a) import numpy as np

```
from scipy import stats

# Sample data: three groups
group1 = [23, 45, 56, 67, 45]
group2 = [34, 56, 67, 78, 56]
group3 = [23, 45, 56, 67, 50]

# Perform One-way ANOVA
f_statistic, p_value = stats.f_oneway(group1, group2, group3)

print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

alpha = 0.05
if p_value < alpha:
    print("Reject null hypothesis: At least one group mean is different.")
else:
    print("Fail to reject null hypothesis: No significant difference between group means.")
```

b) from scipy import stats

```
# Sample average marks for 3 colleges (each list = marks of students in that college)
college1 = [75, 80, 85, 90, 95]
college2 = [65, 70, 75, 80, 85]
college3 = [78, 82, 88, 91, 94]

# Perform One-way ANOVA
f_statistic, p_value = stats.f_oneway(college1, college2, college3)

print(f"F-Score: {f_statistic:.4f}")
print(f"P-Value: {p_value:.4f}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Reject null hypothesis: At least one college's average mark differs significantly.")
else:
    print("Fail to reject null hypothesis: No significant difference between the colleges' average marks.")
```

17) import numpy as np

```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score


# Sample data: feature (X) and response (y)

X = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]).reshape(-1, 1)

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])


# Split data into training and testing sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Create and train linear regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Predict on test set

y_pred = model.predict(X_test)


# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)
```

```
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")

# Plotting
plt.scatter(X, y, color='blue', label='Data points')
plt.plot(X_test, y_pred, color='red', label='Regression line (Test set)')
plt.xlabel('Feature (X)')
plt.ylabel('Response (y)')
plt.title('Linear Regression: Data & Regression Line')
plt.legend()
plt.show()
```

18) import numpy as np

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Sample data (X = feature, y = binary class)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Split data into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Plot data points
plt.scatter(X, y, color='blue', label='Data points')
```

```
# Plot logistic regression curve
X_fit = np.linspace(0, 11, 300).reshape(-1,1)
y_prob = model.predict_proba(X_fit)[:, 1]
plt.plot(X_fit, y_prob, color='red', label='Logistic regression curve')

plt.xlabel('Feature X')
plt.ylabel('Probability of class 1')
plt.title('Logistic Regression Model')
plt.legend()
plt.show()
```

19) import pandas as pd

```
import matplotlib.pyplot as plt
import seaborn as sns

# For demonstration, let's load airline passengers data from seaborn's built-in dataset
# If you have your own CSV, replace this part with: pd.read_csv('your_file.csv')
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/airline-passengers.csv"
df = pd.read_csv(url)

# Check the first few rows
print(df.head())

# Convert 'Month' column to datetime
df['Month'] = pd.to_datetime(df['Month'])

# Set 'Month' as index (optional but good for time series)
df.set_index('Month', inplace=True)

# Plotting with Seaborn and Matplotlib
plt.figure(figsize=(12,6))
sns.lineplot(data=df, x=df.index, y='Passengers', marker='o')
plt.title('Monthly Airline Passengers Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Passengers')
plt.grid(True)
```

```
plt.show()
```