

Assignment 1- Setting Up Python for Data Science & Install and configure Anaconda and Jupiter Notebook

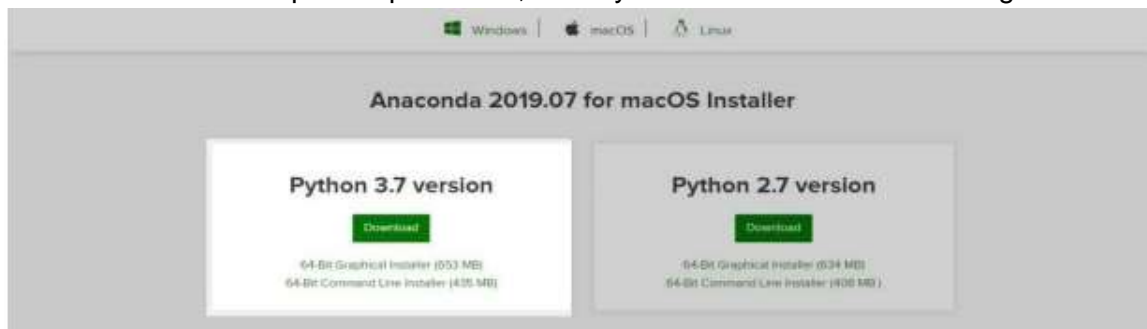
Download and Install Anaconda on Windows

Step #1: Go To Anaconda.com

Go to Anaconda.com, and download the Anaconda version for Windows.

Step #2: Download the Python 3 version for Windows.

Version 2 will not be updated past 2020, so do yourself a favor and start using V3.



Step #3: Double-click on the executable file.

To get the installation of Anaconda started on your operating system open the executable file in your Download folder.

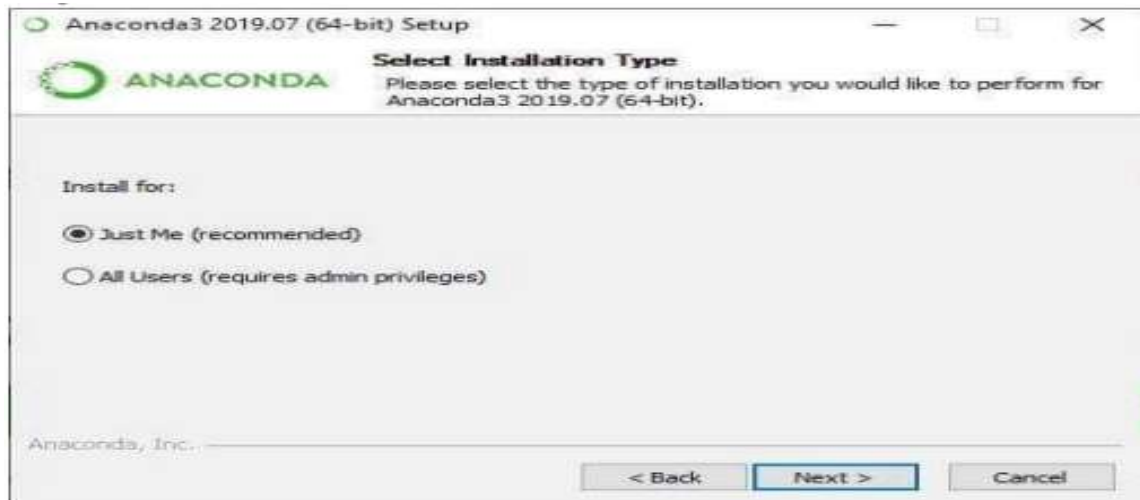


Step #4: Click Next



Step #5: Click I agree to the terms and conditions

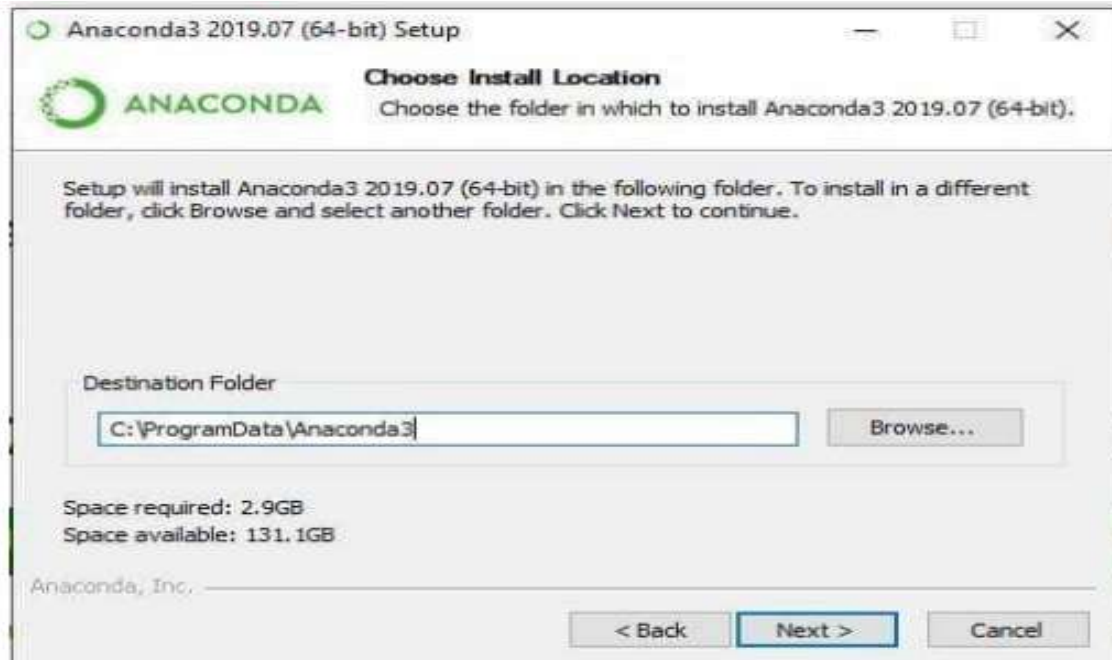
Step #6: Select Who You Want To Give Anaconda To



This step will ask you if you want to install Anaconda just for you or for all the users using this PC. Click “Just-Me”, or “All users”, depending on your preference. Both options will do but to select “all users” you will need admin privileges.

Step #7: Select the installation location

If you have selected “All users”, by default, Anaconda will get installed in the `C:\ProgramData\Anaconda3` folder. So make sure that you have at least the right amount of space available to install the subdirectory comparing it the the space required.

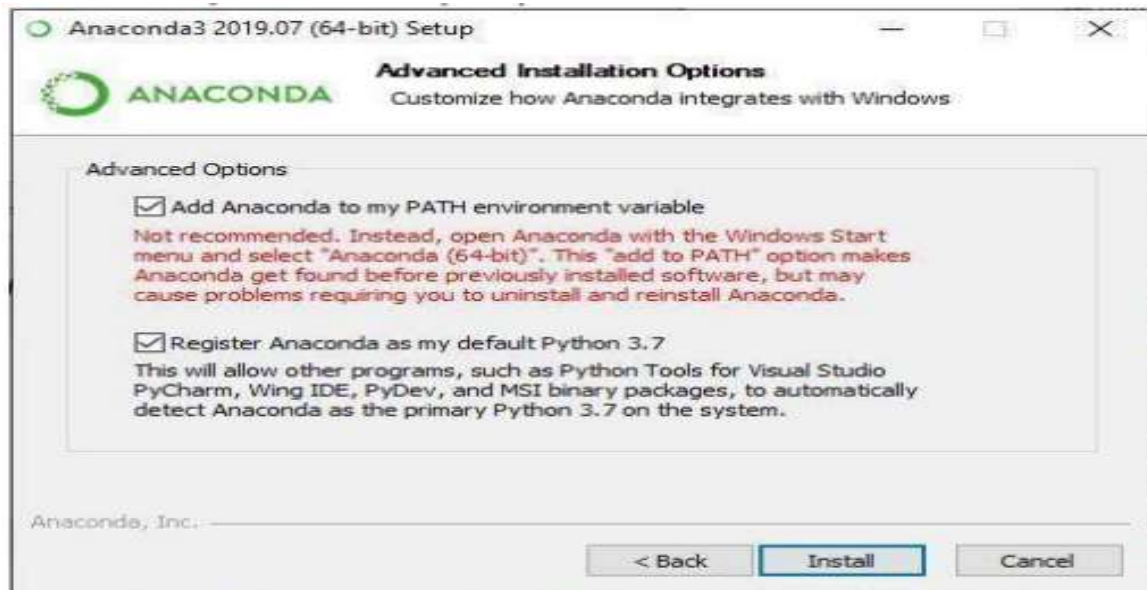


Step #8: Select the environment variables

Depending on if you have any version of Python already installed on your operating system, or not, to do different set-up.

If You Are Installing Python For The First Time

Check the Add Anaconda to my PATH environment variable. This will let you use Anaconda in your command prompt.

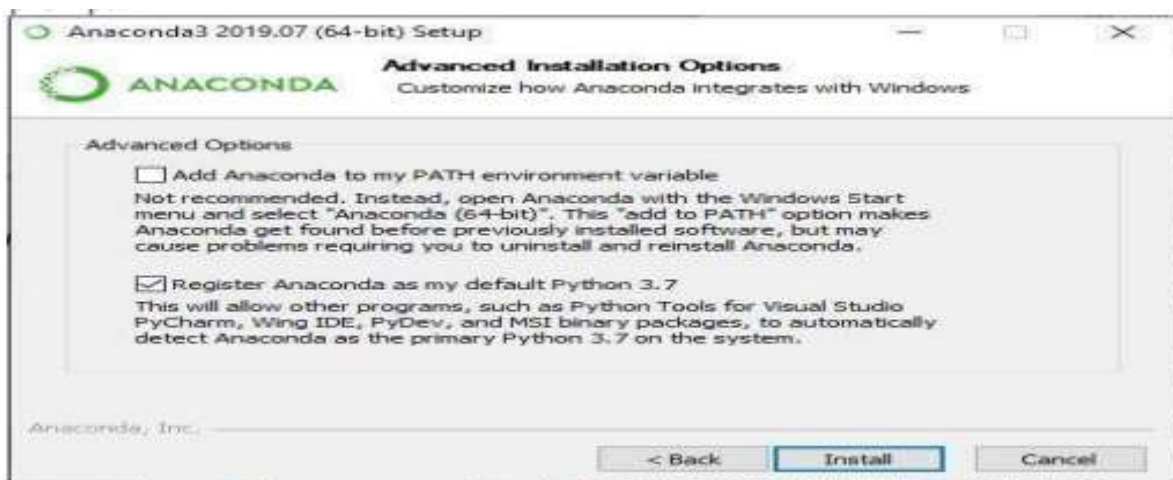


If You Already Have Python Installed

Leave Add Anaconda to my PATH environment variable unchecked.

Leaving it unchecked means that you will have to use Anaconda Command Prompt in order to use Anaconda.

So, unless you add the PATH later, you will not be able to use Python from your command prompt.

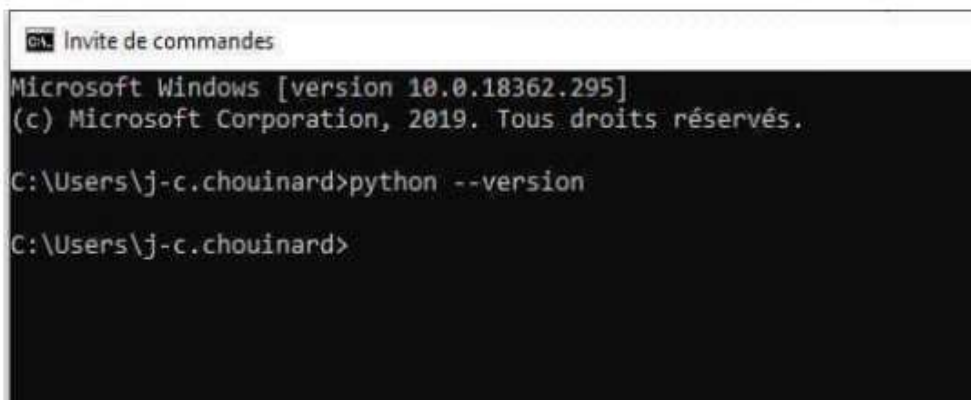


Python is not usually included by default on Windows, however we can check if any version exists on the system.

To know if you have Python Installed.

1. Go to Start Menu and type "Command Prompt" to open it.
2. Type the following command and hit the Enter key "python --version"
3. If nothing happens, you don't have Python installed. Otherwise, you will get this Result.

```
$ python --version  
Python 3.7.0
```



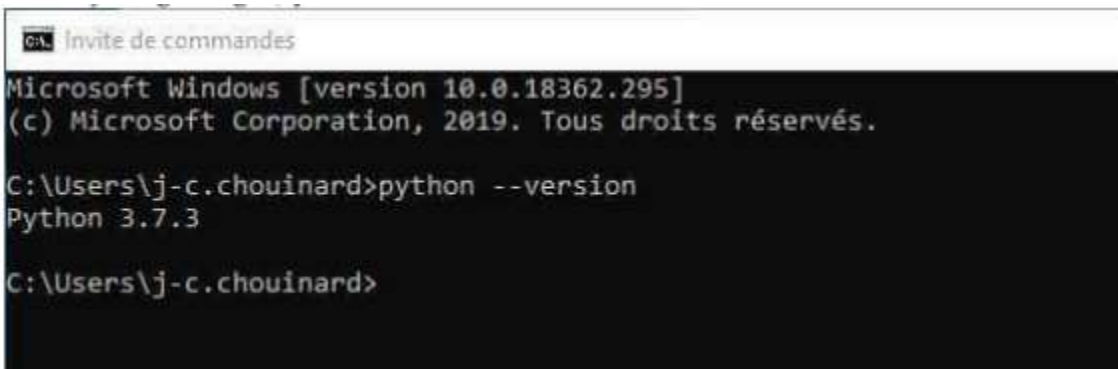
```
Git - Invite de commandes  
Microsoft Windows [version 10.0.18362.295]  
(c) Microsoft Corporation, 2019. Tous droits réservés.  
  
C:\Users\j-c.chouinard>python --version  
  
C:\Users\j-c.chouinard>
```

Step #9: Click Next and then "Finish".

Step #10: See if Python Is Installed

If everything went right you can repeat the step 7 by opening your command prompt and enter "python --version".

If everything is right, you'll see this result.



```
Git - Invite de commandes  
Microsoft Windows [version 10.0.18362.295]  
(c) Microsoft Corporation, 2019. Tous droits réservés.  
  
C:\Users\j-c.chouinard>python --version  
Python 3.7.3  
  
C:\Users\j-c.chouinard>
```

Assignment 2- Create a Jupiter Notebook and perform basic Python operations & Calculate statistics (mean, median, variance) using NumPy functions

```
import numpy as np

# Basic operations

a = 10

b = 5

print("Addition:", a + b)

print("Subtraction:", a - b)

print("Multiplication:", a * b)

print("Division:", a / b)


# Calculating statistics

data = np.array([1,2,3,4,4])

mean = np.mean(data)

median = np.median(data)

population_variance = np.var(data)

sample_variance = np.var(data, ddof=1)


print("Mean:", mean)

print("Median:", median)

print("Population Variance",population_variance)

print("Sample Variance",sample_variance)
```

Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Mean: 2.8
Median: 3.0
Population Variance 1.3599999999999999
Sample Variance 1.7
```

Assignment 3- Load datasets into Pandas Data Frames from CSV files & Perform basic Data Frame operations (e.g., filtering, sorting).

```
import pandas as pd

import numpy as np

data = pd.read_csv('student.csv')

print(data)

df_filter = data[data['percentage'] > 65]

print("\nFilter: roll no > 5")

print(df_filter)

sorted_by_name = data.sort_values(by='student name')

print("\nSorted by roll no (Ascending):")

print(sorted_by_name)

sorted_by_rno = data.sort_values(by='roll no', ascending=False)
```

```
print("\nSorted by roll no (Descending):")
```

```
print(sorted_by_rno)
```

Output:

| | roll no | student name | city | contact no | percentage |
|---|---------|--------------|----------|----------------|------------|
| 0 | 2 | Roy | mumbai | (469) 555-0146 | 80.5 |
| 1 | 4 | Rima | pune | (218) 555-0121 | 75.6 |
| 2 | 5 | Aasha | jalgaon | (605) 555-0160 | 70.1 |
| 3 | 7 | Kiran | dhule | (865) 555-0186 | 70.6 |
| 4 | 1 | Mina | nashik | (281) 555-0128 | 92.1 |
| 5 | 3 | Ashwin | kolhapur | (410) 555-0140 | 68.4 |
| 6 | 8 | Harsh | satara | (711) 555-0171 | 60.4 |
| 7 | 9 | Lokesh | nashik | (648) 555-0164 | 65.7 |
| 8 | 6 | Nisha | mumbai | (997) 555-0199 | 64.1 |
| 9 | 12 | Bhavana | thane | (288) 555-0128 | 62.1 |

Filter: roll no > 5

| | roll no | student name | city | contact no | percentage |
|---|---------|--------------|----------|----------------|------------|
| 0 | 2 | Roy | mumbai | (469) 555-0146 | 80.5 |
| 1 | 4 | Rima | pune | (218) 555-0121 | 75.6 |
| 2 | 5 | Aasha | jalgaon | (605) 555-0160 | 70.1 |
| 3 | 7 | Kiran | dhule | (865) 555-0186 | 70.6 |
| 4 | 1 | Mina | nashik | (281) 555-0128 | 92.1 |
| 5 | 3 | Ashwin | kolhapur | (410) 555-0140 | 68.4 |
| 7 | 9 | Lokesh | nashik | (648) 555-0164 | 65.7 |

Sorted by roll no (Ascending):

| | roll no | student name | city | contact no | percentage |
|---|---------|--------------|----------|----------------|------------|
| 2 | 5 | Aasha | jalgaon | (605) 555-0160 | 70.1 |
| 5 | 3 | Ashwin | kolhapur | (410) 555-0140 | 68.4 |
| 9 | 12 | Bhavana | thane | (288) 555-0128 | 62.1 |
| 6 | 8 | Harsh | satara | (711) 555-0171 | 60.4 |
| 3 | 7 | Kiran | dhule | (865) 555-0186 | 70.6 |
| 7 | 9 | Lokesh | nashik | (648) 555-0164 | 65.7 |
| 4 | 1 | Mina | nashik | (281) 555-0128 | 92.1 |
| 8 | 6 | Nisha | mumbai | (997) 555-0199 | 64.1 |
| 1 | 4 | Rima | pune | (218) 555-0121 | 75.6 |
| 0 | 2 | Roy | mumbai | (469) 555-0146 | 80.5 |

Sorted by roll no (Descending):

| | roll no | student name | city | contact no | percentage |
|---|---------|--------------|----------|----------------|------------|
| 9 | 12 | Bhavana | thane | (288) 555-0128 | 62.1 |
| 7 | 9 | Lokesh | nashik | (648) 555-0164 | 65.7 |
| 6 | 8 | Harsh | satara | (711) 555-0171 | 60.4 |
| 3 | 7 | Kiran | dhule | (865) 555-0186 | 70.6 |
| 8 | 6 | Nisha | mumbai | (997) 555-0199 | 64.1 |
| 2 | 5 | Aasha | jalgaon | (605) 555-0160 | 70.1 |
| 1 | 4 | Rima | pune | (218) 555-0121 | 75.6 |
| 5 | 3 | Ashwin | kolhapur | (410) 555-0140 | 68.4 |
| 0 | 2 | Roy | mumbai | (469) 555-0146 | 80.5 |
| 4 | 1 | Mina | nashik | (281) 555-0128 | 92.1 |

Assignment 4- Write a program to implement Normalize and standardize numerical data using Scikit-learn

```
import numpy as np
```

```
import pandas as pd
```

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
data = {  
    'A': [1, 2, 3, 4, 5],  
    'B': [11, 25, 32, 44, 58],  
    'C': [100, 200, 300, 400, 500]  
}
```

```
df = pd.DataFrame(data)
```

```
print("Original Data:")
```

```
print(df)
```

```
min_max_scaler = MinMaxScaler()
```



```

normalized_data = min_max_scaler.fit_transform(df)

df_normalized = pd.DataFrame(normalized_data, columns=df.columns)

print("\nNormalized Data (Min-Max Scaling):")

print(df_normalized)


standard_scaler = StandardScaler()

standardized_data = standard_scaler.fit_transform(df)

df_standardized = pd.DataFrame(standardized_data, columns=df.columns)

print("\nStandardized Data (Z-score Scaling):")

print(df_standardized)

```

Output:

Original Data:

| | A | B | C |
|---|---|----|-----|
| 0 | 1 | 11 | 100 |
| 1 | 2 | 25 | 200 |
| 2 | 3 | 32 | 300 |
| 3 | 4 | 44 | 400 |
| 4 | 5 | 58 | 500 |

Normalized Data (Min-Max Scaling):

| | A | B | C |
|---|------|----------|------|
| 0 | 0.00 | 0.000000 | 0.00 |
| 1 | 0.25 | 0.297872 | 0.25 |
| 2 | 0.50 | 0.446809 | 0.50 |
| 3 | 0.75 | 0.702128 | 0.75 |
| 4 | 1.00 | 1.000000 | 1.00 |

Standardized Data (Z-score Scaling):

| | A | B | C |
|---|-----------|-----------|-----------|
| 0 | -1.414214 | -1.431917 | -1.414214 |
| 1 | -0.707107 | -0.560316 | -0.707107 |
| 2 | 0.000000 | -0.124515 | 0.000000 |
| 3 | 0.707107 | 0.622573 | 0.707107 |
| 4 | 1.414214 | 1.494175 | 1.414214 |

Assignment-5 Write a program to implement Encode categorical variables using One-Hot Encoding and Label Encoding with Pandas and Scikit-learn.

```
import pandas as pd

from sklearn.preprocessing import OneHotEncoder,LabelEncoder

data={

    'Category':['A','B','A','C','B','C'],

    'Value':[10,20,10,30,20,30]

}

df=pd.DataFrame(data)

print("Original DataFrame")

print(df)

one_hot_encoder_df=pd.get_dummies(df,columns=['Category'])

print(one_hot_encoder_df)

label_encoder=LabelEncoder()

df["Category_label"]=label_encoder.fit_transform(df['Category'])

print(df)
```

Output:

```
Original DataFrame
   Category  Value
0         A     10
1         B     20
2         A     10
3         C     30
4         B     20
5         C     30
```

One_Hot_encoding:

| | Value | Category_A | Category_B | Category_C |
|---|-------|------------|------------|------------|
| 0 | 10 | 1 | 0 | 0 |
| 1 | 20 | 0 | 1 | 0 |
| 2 | 10 | 1 | 0 | 0 |
| 3 | 30 | 0 | 0 | 1 |
| 4 | 20 | 0 | 1 | 0 |
| 5 | 30 | 0 | 0 | 1 |

Label_Encoding:

| | Category | Value | Category_label |
|---|----------|-------|----------------|
| 0 | A | 10 | 0 |
| 1 | B | 20 | 1 |
| 2 | A | 10 | 0 |
| 3 | C | 30 | 2 |
| 4 | B | 20 | 1 |
| 5 | C | 30 | 2 |

Assignment-6 Write a program to Plot histograms to visualize data distribution and calculate skewness and kurtosis of given dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew,kurtosis

data=[12,15,14,10,8,15,14,12,10,9,12,13,14,10,9,11,13,15,14,12]
data_series=pd.Series(data)
```

```

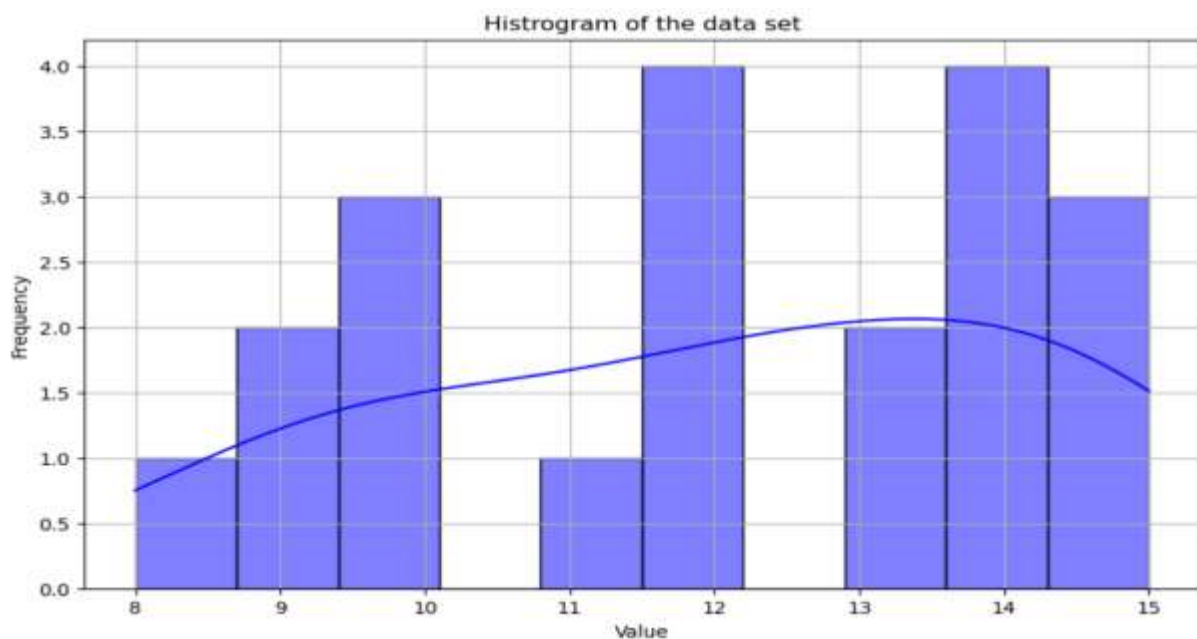
data_skewness=skew(data_series)
data_kurtosis=kurtosis(data_series)
print(f"skewness:{data_skewness}")
print(f"Kurtosis:{data_kurtosis}")

plt.figure(figsize=(10,6))
sns.histplot(data_series,bins=10,kde=True,color='blue')
plt.title('Histogram of the data set')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

```

Output:

skewness : -0.2824965921154543
Kurtosis : -1.1186865450610162



Assignment-7 write a program to perform a t-test(one-sample,independent) on sample data and interpret the result

```
import numpy as np

import pandas as pd

from scipy import stats

# Sample data for one-sample t-test

sample_data_one = [12, 15, 14, 10, 8, 15, 14, 12, 10, 9]

population_mean = 12 # Known population mean to test against


# Perform one-sample t-test

t_stat_one, p_value_one = stats.ttest_1samp(sample_data_one, population_mean)


# Interpretation for one-sample t-test

print("One-Sample T-Test:")

print(f"T-statistic: {t_stat_one}, P-value: {p_value_one}")

if p_value_one < 0.05:

    print("Reject the null hypothesis: The sample mean is significantly different from the population mean.")

else:

    print("Fail to reject the null hypothesis: The sample mean is not significantly different from the population mean.")
```

```

# Sample data for independent two-sample t-test
sample_data_a = [12, 15, 14, 10, 8]
sample_data_b = [14, 15, 13, 17, 19]

# Perform independent two-sample t-test
t_stat_two, p_value_two = stats.ttest_ind(sample_data_a, sample_data_b)

# Interpretation for independent two-sample t-test
print("\nIndependent Two-Sample T-Test:")
print(f"T-statistic: {t_stat_two}, P-value: {p_value_two}")

if p_value_two < 0.05:
    print("Reject the null hypothesis: The means of the two samples are significantly different.")
else:
    print("Fail to reject the null hypothesis: The means of the two samples are not significantly different.")

```

Output:

```

One-Sample T-Test:
T-statistic: -0.12361284651454894, P-value: 0.9043384285084789
Fail to reject the null hypothesis: The sample mean is not significantly different from the population mean.

Independent Two-Sample T-Test:
T-statistic: -2.270934357735347, P-value: 0.05281335911209561
Fail to reject the null hypothesis: The means of the two samples are not significantly different.

```

Assignment-8 Write a program to Calculate and interpret the Pearson and Spearman correlation coefficient

```
import numpy as np

import scipy.stats as stats

# Function to calculate and interpret Pearson correlation

def pearson_correlation(x, y):

    # Calculate Pearson correlation coefficient

    pearson_corr, p_value = stats.pearsonr(x, y)

    print(f"Pearson Correlation Coefficient: {pearson_corr:.4f}")

# Interpretation based on the value of Pearson's correlation

if pearson_corr > 0.8:

    print("Interpretation: Strong positive linear relationship")

elif pearson_corr > 0.5:

    print("Interpretation: Moderate positive linear relationship")

elif pearson_corr > 0:

    print("Interpretation: Weak positive linear relationship")

elif pearson_corr < -0.8:

    print("Interpretation: Strong negative linear relationship")

elif pearson_corr < -0.5:

    print("Interpretation: Moderate negative linear relationship")
```

```

elif pearson_corr < 0:

    print("Interpretation: Weak negative linear relationship")

else:

    print("Interpretation: No linear relationship")


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

if p_value < 0.05:

    print("Conclusion: The correlation is statistically significant.")

else:

    print("Conclusion: The correlation is not statistically significant.")


# Function to calculate and interpret Spearman correlation

def spearman_correlation(x, y):

    # Calculate Spearman correlation coefficient

    spearman_corr, p_value = stats.spearmanr(x, y)


    print(f"Spearman Correlation Coefficient: {spearman_corr:.4f}")


    # Interpretation based on the value of Spearman's correlation

    if spearman_corr > 0.8:

        print("Interpretation: Strong positive monotonic relationship")

    elif spearman_corr > 0.5:

        print("Interpretation: Moderate positive monotonic relationship")

    elif spearman_corr > 0:

```



```

    print("Interpretation: Weak positive monotonic relationship")

elif spearman_corr < -0.8:

    print("Interpretation: Strong negative monotonic relationship")

elif spearman_corr < -0.5:

    print("Interpretation: Moderate negative monotonic relationship")

elif spearman_corr < 0:

    print("Interpretation: Weak negative monotonic relationship")

else:

    print("Interpretation: No monotonic relationship")


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

if p_value < 0.05:

    print("Conclusion: The correlation is statistically significant.")

else:

    print("Conclusion: The correlation is not statistically significant.")


# Example data (you can change these lists)

x = [10, 20, 30, 40, 50]

y = [12, 22, 31, 43, 48]


# Calculate and interpret Pearson correlation

print("Pearson Correlation:")

pearson_correlation(x, y)

```

```
# Calculate and interpret Spearman correlation

print("\nSpearman Correlation:")

spearman_correlation(x, y)
```

Output:

```
Pearson Correlation:
Pearson Correlation Coefficient: 0.9943
Interpretation: Strong positive linear relationship
P-value: 0.0005
Conclusion: The correlation is statistically significant.

Spearman Correlation:
Spearman Correlation Coefficient: 1.0000
Interpretation: Strong positive monotonic relationship
P-value: 0.0000
Conclusion: The correlation is statistically significant.
```

Assignment-9 Write a program to implement Conduct ANOVA (One-Way and Two-Way) test

```
import numpy as np

import scipy.stats as stats

import pandas as pd

import statsmodels.api as sm

from statsmodels.formula.api import ols

from statsmodels.stats.anova import anova_lm
```

```

# ----- One-Way ANOVA -----

# Sample data: Replace these with your actual data

group_A = [89, 89, 88, 78, 79]
group_B = [93, 92, 94, 89, 88]
group_C = [89, 88, 89, 93, 90]


# Perform the One-Way ANOVA test

f_statistic, p_value = stats.f_oneway(group_A, group_B, group_C)


print("\n----- One-Way ANOVA -----")
print(f"F-statistic: {f_statistic}")
print(f"P-value: {p_value}")


# Interpretation of the result

alpha = 0.05 # significance level (5%)

if p_value < alpha:

    print("Reject the null hypothesis: At least one of the group means is significantly
different.")

else:

    print("Fail to reject the null hypothesis: The group means are not significantly
different.")

```

```

# ----- Two-Way ANOVA -----

# Example Data: Two factors and a dependent variable

# 'FactorA' and 'FactorB' are the two independent variables (factors)

# 'Value' is the dependent variable

data = {

    'FactorA': ['A1', 'A1', 'A1', 'A2', 'A2', 'A2', 'A1', 'A1', 'A1', 'A2', 'A2', 'A2'],

    'FactorB': ['B1', 'B2', 'B1', 'B2', 'B1', 'B2', 'B1', 'B2', 'B1', 'B2', 'B1', 'B2'],

    'Value': [23, 21, 25, 30, 28, 31, 22, 23, 21, 35, 34, 32]

}

df = pd.DataFrame(data)

print(df)

# Perform Two-Way ANOVA

model = ols('Value ~ C(FactorA) + C(FactorB) + C(FactorA):C(FactorB)', data=df).fit()

anova_result = anova_lm(model)

print("Two-Way ANOVA Results:")

print(anova_result)

# Interpretation

# Look at the p-values to see if there is a significant effect for FactorA, FactorB, or their
interaction

alpha = 0.05

if anova_result['PR(>F)'][0] < alpha:

```

```

    print("FactorA has a significant effect on the dependent variable.")
else:
    print("FactorA does not have a significant effect on the dependent variable.")

if anova_result['PR(>F)'][1] < alpha:
    print("FactorB has a significant effect on the dependent variable.")
else:
    print("FactorB does not have a significant effect on the dependent variable.")

if anova_result['PR(>F)'][2] < alpha:
    print("There is a significant interaction effect between FactorA and FactorB.")
else:
    print("There is no significant interaction effect between FactorA and FactorB.")

```

Output:

```

----- One-Way ANOVA -----
F-statistic: 4.35011990407674
P-value: 0.03795204795237708
Reject the null hypothesis: At least one of the group means is significantly different.

```

Two-Way ANOVA Results:

| | FactorA | FactorB | Value |
|----|---------|---------|-------|
| 0 | A1 | B1 | 23 |
| 1 | A1 | B2 | 21 |
| 2 | A1 | B1 | 25 |
| 3 | A2 | B2 | 30 |
| 4 | A2 | B1 | 28 |
| 5 | A2 | B2 | 31 |
| 6 | A1 | B1 | 22 |
| 7 | A1 | B2 | 23 |
| 8 | A1 | B1 | 21 |
| 9 | A2 | B2 | 35 |
| 10 | A2 | B1 | 34 |
| 11 | A2 | B2 | 32 |

| | df | sum_sq | mean_sq | F | PR(>F) |
|-----------------------|-----|------------|------------|-----------|----------|
| C(FactorA) | 1.0 | 252.083333 | 252.083333 | 47.173489 | 0.000129 |
| C(FactorB) | 1.0 | 0.041667 | 0.041667 | 0.007797 | 0.931807 |
| C(FactorA):C(FactorB) | 1.0 | 2.041667 | 2.041667 | 0.382066 | 0.553685 |
| Residual | 8.0 | 42.750000 | 5.343750 | NaN | NaN |

FactorA has a significant effect on the dependent variable.

FactorB does not have a significant effect on the dependent variable.

There is no significant interaction effect between FactorA and FactorB.

Assignment-10 Write a program to implement simple linear regression, Multiple linear regression on given data set.

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```

import seaborn as sns

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, r2_score


# Sample dataset for Simple and Multiple Linear Regression

data = {

    'X1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], # Independent variable 1 (e.g., years of experience)

    'X2': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11], # Independent variable 2 (e.g., education level)

    'y': [10, 15, 20, 25, 30, 35, 40, 45, 50, 55] # Dependent variable (e.g., salary)

}


# Convert the data dictionary into a pandas DataFrame

df = pd.DataFrame(data)


# --- SIMPLE LINEAR REGRESSION ---

# Let's assume 'X1' as the independent variable and 'y' as the dependent variable for
simple linear regression.


# Prepare the data for Simple Linear Regression

X_simple = df[['X1']] # Independent variable (e.g., years of experience)

y_simple = df['y'] # Dependent variable (e.g., salary)


# Split the data into training and testing sets

```

```
X_train_simple, X_test_simple, y_train_simple, y_test_simple =  
train_test_split(X_simple, y_simple, test_size=0.2, random_state=42)
```

```
# Create and train the model
```

```
model_simple = LinearRegression()
```

```
model_simple.fit(X_train_simple, y_train_simple)
```

```
# Predict the results
```

```
y_pred_simple = model_simple.predict(X_test_simple)
```

```
# Calculate Mean Squared Error and R-squared
```

```
mse_simple = mean_squared_error(y_test_simple, y_pred_simple)
```

```
r2_simple = r2_score(y_test_simple, y_pred_simple)
```

```
# Plotting the Simple Linear Regression results
```

```
plt.figure(figsize=(8, 6))
```

```
plt.scatter(X_test_simple, y_test_simple, color='blue', label='Actual values')
```

```
plt.plot(X_test_simple, y_pred_simple, color='red', label='Regression Line')
```

```
plt.xlabel('X1 (Years of Experience)')
```

```
plt.ylabel('y (Salary)')
```

```
plt.title('Simple Linear Regression: Salary vs Experience')
```

```
plt.legend()
```

```
plt.show()
```



```
print("\n----- Simple Linear Regression -----")  
print(f"Intercept: {model_simple.intercept_}")  
print(f"Coefficient: {model_simple.coef_}")  
print(f"Mean Squared Error: {mse_simple}")  
print(f"R-squared: {r2_simple}")
```

```
# --- MULTIPLE LINEAR REGRESSION ---
```

```
# Now, let's use both 'X1' (experience) and 'X2' (education level) as independent  
variables.
```

```
# Prepare the data for Multiple Linear Regression
```

```
X_multiple = df[['X1', 'X2']] # Independent variables
```

```
y_multiple = df['y'] # Dependent variable
```

```
# Split the data into training and testing sets
```

```
X_train_multiple, X_test_multiple, y_train_multiple, y_test_multiple =  
train_test_split(X_multiple, y_multiple, test_size=0.2, random_state=42)
```

```
# Create and train the model
```

```
model_multiple = LinearRegression()
```

```
model_multiple.fit(X_train_multiple, y_train_multiple)
```

```
# Predict the results
```

```
y_pred_multiple = model_multiple.predict(X_test_multiple)
```

```
# Calculate Mean Squared Error and R-squared for Multiple Linear Regression
```

```
mse_multiple = mean_squared_error(y_test_multiple, y_pred_multiple)
```

```
r2_multiple = r2_score(y_test_multiple, y_pred_multiple)
```

```
# Print Multiple Linear Regression results
```

```
print("\n----- Multiple Linear Regression -----")
```

```
print(f"Intercept: {model_multiple.intercept_}")
```

```
print(f"Coefficients: {model_multiple.coef_}")
```

```
print(f"Mean Squared Error: {mse_multiple}")
```

```
print(f"R-squared: {r2_multiple}")
```

```
# Note: Visualization for Multiple Linear Regression is complex with more than 1  
independent variable, but you can plot the residuals
```

```
# Plotting Residuals for Multiple Linear Regression
```

```
residuals_multiple = y_test_multiple - y_pred_multiple
```

```
plt.figure(figsize=(8, 6))
```

```
sns.residplot(y_pred_multiple, residuals_multiple, lowess=True, color='blue',  
line_kws={'color': 'red', 'lw': 1})
```

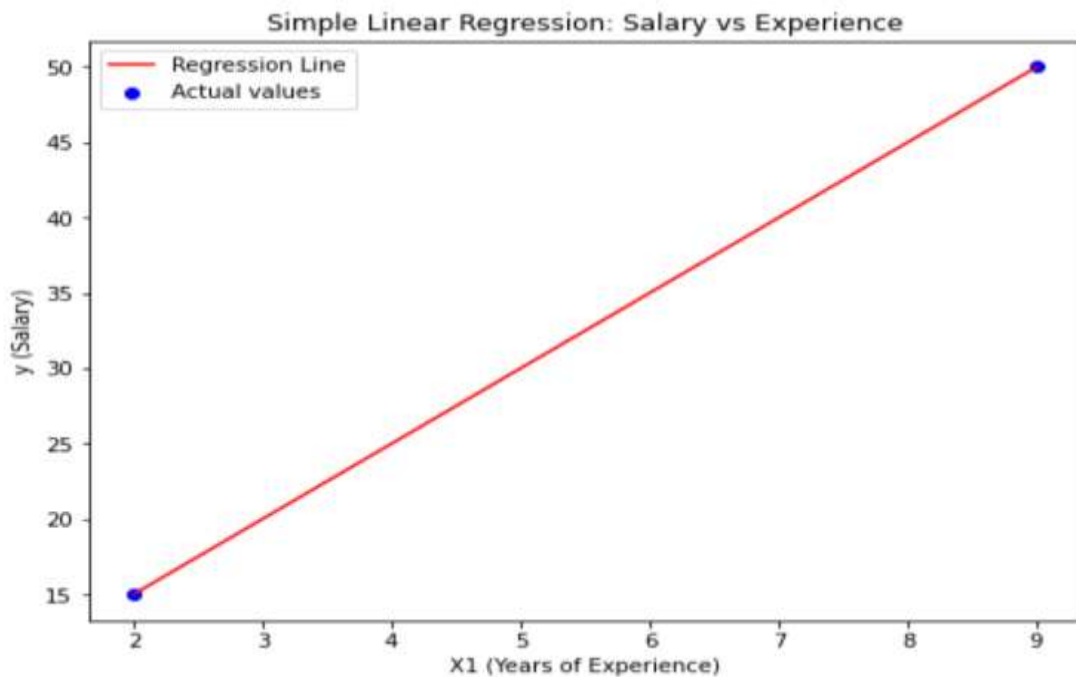
```
plt.xlabel('Predicted Values')
```

```
plt.ylabel('Residuals')
```

```
plt.title('Residuals Plot: Multiple Linear Regression')
```

```
plt.show()
```

Output:



----- Simple Linear Regression -----

Intercept: 4.9999999999999964

Coefficient: [5.]

Mean Squared Error: 1.5777218104420236e-30

R-squared: 1.0

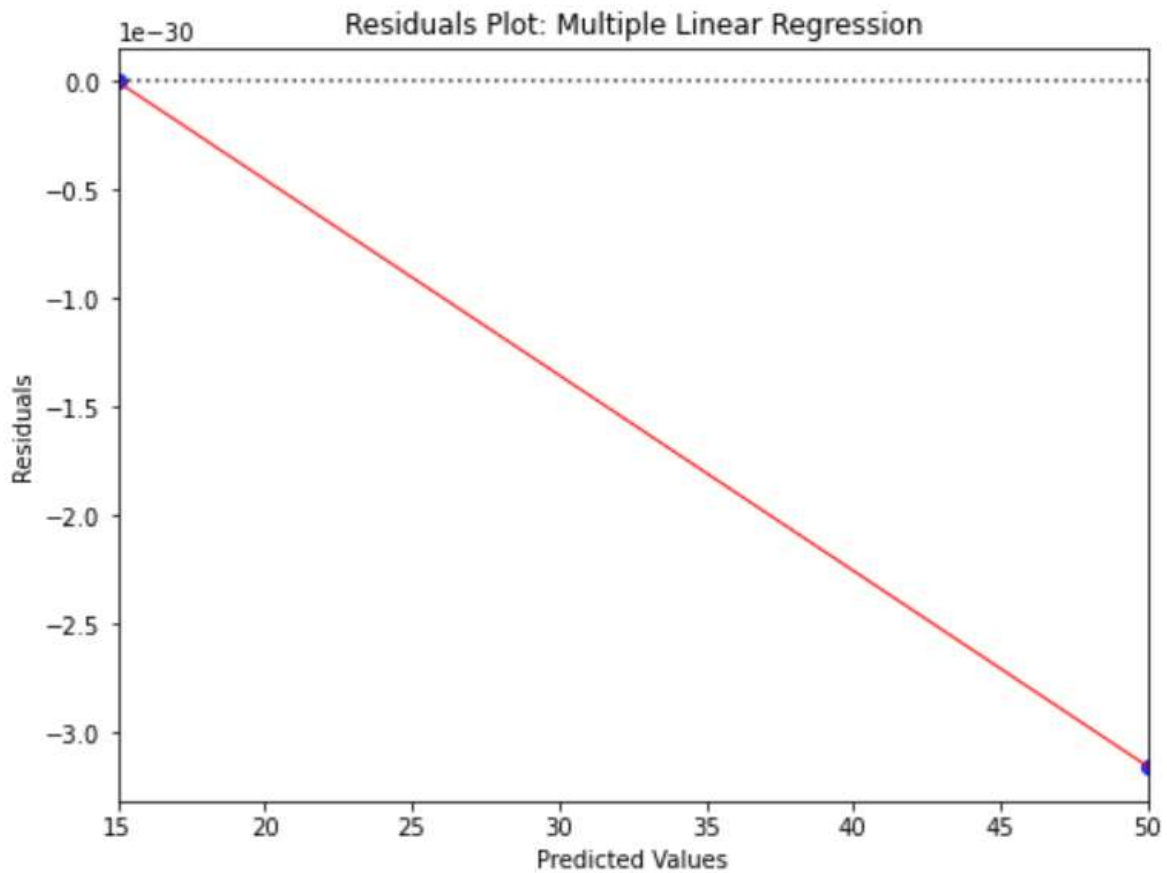
----- Multiple Linear Regression -----

Intercept: 2.4999999999999993

Coefficients: [2.5 2.5]

Mean Squared Error: 3.155443620884047e-29

R-squared: 1.0



Assignment-11 Write a program to implement simple logistic regression on given data set.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
```

```
# Sample data (Years of experience and binary outcome: 1 means purchased, 0 means not purchased)
```

```
data = {  
    'X': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], # Independent variable (e.g., years of experience)  
    'y': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1] # Dependent variable (0 = No purchase, 1 = Purchased)  
}
```

```
# Convert the data into a pandas DataFrame
```

```
df = pd.DataFrame(data)
```

```
# Prepare the data for Logistic Regression
```

```
X = df[['X']] # Independent variable (years of experience)
```

```
y = df['y'] # Dependent variable (purchase or not)
```

```
# Split the data into training and testing sets
```

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

```
# Create and train the Logistic Regression model
```

```
model = LogisticRegression()
```

```
model.fit(X_train, y_train)
```

```
# Predict the results
```

```
y_pred = model.predict(X_test)
```

```

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

# Confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

# Print the results

print("\n----- Logistic Regression -----")

print(f"Accuracy: {accuracy}")

print(f"Confusion Matrix:\n{conf_matrix}")

# Plotting the decision boundary for logistic regression

plt.figure(figsize=(8, 6))

plt.scatter(X_train, y_train, color='blue', label='Train data')

plt.scatter(X_test, y_test, color='red', label='Test data')

plt.plot(X_test, model.predict_proba(X_test)[:,-1], color='green', label='Decision
Boundary', linestyle='--')

plt.xlabel('Years of Experience')

plt.ylabel('Purchase (0 or 1)')

plt.title('Logistic Regression: Purchase vs Experience')

plt.legend()

plt.show()

```

Output:

----- Logistic Regression -----

Accuracy: 1.0

Confusion Matrix:

```
[[1 0]
 [0 1]]
```

