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

df =
pd.read_csv('https://github.com/ybifoundation/Dataset/raw/main/Salary
%20Data.csv')
df
```

	Experience Years	Salary
0	1.1	39343
1	1.2	42774
2	1.3	46205
3	1.5	37731
4	2.0	43525
5	2.2	39891
6	2.5	48266
7	2.9	56642
8	3.0	60150
9	3.2	54445
10	3.2	64445
11	3.5	60000
12	3.7	57189
13	3.8	60200
14	3.9	63218
15	4.0	55794
16	4.0	56957
17	4.1	57081
18	4.3	59095
19	4.5	61111
20	4.7	64500
21	4.9	67938
22	5.1	66029
23	5.3	83088
24	5.5	82200
25	5.9	81363
26	6.0	93940
27	6.2	91000
28	6.5	90000
29	6.8	91738
30	7.1	98273
31	7.9	101302
32	8.2	113812
33	8.5	111620
34	8.7	109431
35	9.0	105582
36	9.5	116969
37	9.6	112635
38	10.3	122391
39	10.5	121872

```python
# Extract the input and output values from dataset
# write your code here

# Extract input and output values
X = df[['Experience Years']]  # Use the corrected column name
y = df['Salary']  # Use the corrected column name

# Display first few rows
print(X.head())
print(y.head())
```

```
   Experience Years
0               1.1
1               1.2
2               1.3
3               1.5
4               2.0
0    39343
1    42774
2    46205
3    37731
4    43525
Name: Salary, dtype: int64
```

```python
print(X)
```

```
    Experience Years
0                1.1
1                1.2
2                1.3
3                1.5
4                2.0
5                2.2
6                2.5
7                2.9
8                3.0
9                3.2
10               3.2
11               3.5
12               3.7
13               3.8
14               3.9
15               4.0
16               4.0
17               4.1
18               4.3
19               4.5
20               4.7
21               4.9
22               5.1
```

```
23                5.3
24                5.5
25                5.9
26                6.0
27                6.2
28                6.5
29                6.8
30                7.1
31                7.9
32                8.2
33                8.5
34                8.7
35                9.0
36                9.5
37                9.6
38               10.3
39               10.5
```

```
y = df.iloc[:, 1].values
y
```

```
array([ 39343,  42774,  46205,  37731,  43525,  39891,  48266,  56642,
        60150,  54445,  64445,  60000,  57189,  60200,  63218,  55794,
        56957,  57081,  59095,  61111,  64500,  67938,  66029,  83088,
        82200,  81363,  93940,  91000,  90000,  91738,  98273, 101302,
       113812, 111620, 109431, 105582, 116969, 112635, 122391,
121872],
      dtype=int64)
```

```
# In case Dataset is large
print("============================================================")
print("NAN values are available:",df.isnull().values.any())
print("============================================================")
print("There are",df.isnull().sum(),"NAN values in the Dataset")
print("============================================================")
```

```
============================================================
NAN values are available: False
============================================================
There are Experience Years    0
Salary                 0
dtype: int64 NAN values in the Dataset
============================================================
```

```
from sklearn.impute import SimpleImputer

import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

# Sample dataset with missing values
```

```
data = {'YearsExperience': [1.1, 2.2, np.nan, 4.5, 5.1],
        'Salary': [39000, 46000, 57000, np.nan, 76000]}

df = pd.DataFrame(data)

# Creating the Imputer object using the mean strategy
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

# Fitting and transforming the dataset
df_imputed = pd.DataFrame(imputer.fit_transform(df),
columns=df.columns)

# Display the dataset after imputation
print(df_imputed)
```

```
   YearsExperience   Salary
0            1.100  39000.0
1            2.200  46000.0
2            3.225  57000.0
3            4.500  54500.0
4            5.100  76000.0
```

```
import numpy as np
from sklearn.impute import SimpleImputer

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])

# Creating the Imputer object with the mean strategy
imputer = SimpleImputer(strategy='mean')

# Fitting the data to the imputer object
imputer = imputer.fit(X)

# Transforming the data to replace missing values
X_imputed = imputer.transform(X)

# Display the transformed data
print(X_imputed)
```

```
import numpy as np
from sklearn.impute import SimpleImputer

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])

# Creating the Imputer object with the mean strategy
imputer = SimpleImputer(strategy='mean')

# Fitting the imputer to the data (computing the mean)
imputer = imputer.fit(X)
```

```
# Imputing the data (replacing missing values with the computed mean)
X = imputer.transform(X)

# Display the transformed data
print(X)
```

```
[[1.  2.  7.5]
 [4.  5.  6. ]
 [7.  8.  9. ]]
```

```
print("Imputed Data : \n", X)
```

```
Imputed Data :
 [[1.  2.  7.5]
 [4.  5.  6. ]
 [7.  8.  9. ]]
```

```
# No categorical data is present.

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
X
```

```
array([[-1.22474487, -1.22474487,  0.        ],
       [ 0.        ,  0.        , -1.22474487],
       [ 1.22474487,  1.22474487,  1.22474487]])
```

```
# Split the data into 2 parts for training and testing
# write your code here
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])

# Creating the Imputer object with the mean strategy
imputer = SimpleImputer(strategy='mean')

# Fitting the imputer to the data (computing the mean)
imputer = imputer.fit(X)

# Imputing the data (replacing missing values with the computed mean)
X = imputer.transform(X)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Display the results
```

```
print("Training Data:\n", X_train)
print("\nTesting Data:\n", X_test)
```

```
Training Data:
 [[4. 5. 6.]
 [7. 8. 9.]]

Testing Data:
 [[1.  2.  7.5]]
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Creating the Imputer object with the mean strategy
imputer = SimpleImputer(strategy='mean')

# Fitting the imputer to the data (computing the mean)
imputer = imputer.fit(X)

# Imputing the data (replacing missing values with the computed mean)
X = imputer.transform(X)

# Split the 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)

# Display the shapes of the datasets
print(f'X_train dataset {X_train.shape}\n')
print(f'X_test dataset {X_test.shape}\n')
print(f'y_train dataset {y_train.shape}\n')
print(f'y_test dataset {y_test.shape}\n')
```

```
X_train dataset (2, 3)

X_test dataset (1, 3)

y_train dataset (2,)

y_test dataset (1,)
```

```
# Train the data with machine learning algorithm
# write your code here
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Creating the Imputer object with the mean strategy
imputer = SimpleImputer(strategy='mean')

# Fitting the imputer to the data (computing the mean)
imputer = imputer.fit(X)

# Imputing the data (replacing missing values with the computed mean)
X = imputer.transform(X)

# Split the 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 the Logistic Regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Predict the target values for the test data
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)

print(f'Model Accuracy: {accuracy * 100:.2f}%')
```

```
Model Accuracy: 0.00%
```

```
# Predict the data with test samples
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Imputer to handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the data
```

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

# Train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Predict the data with test samples
y_pred = lr.predict(X_test)

# Display predictions
print(f'Predictions: {y_pred}')
```

```
Predictions: [0]
```

```
print(y_pred)
```

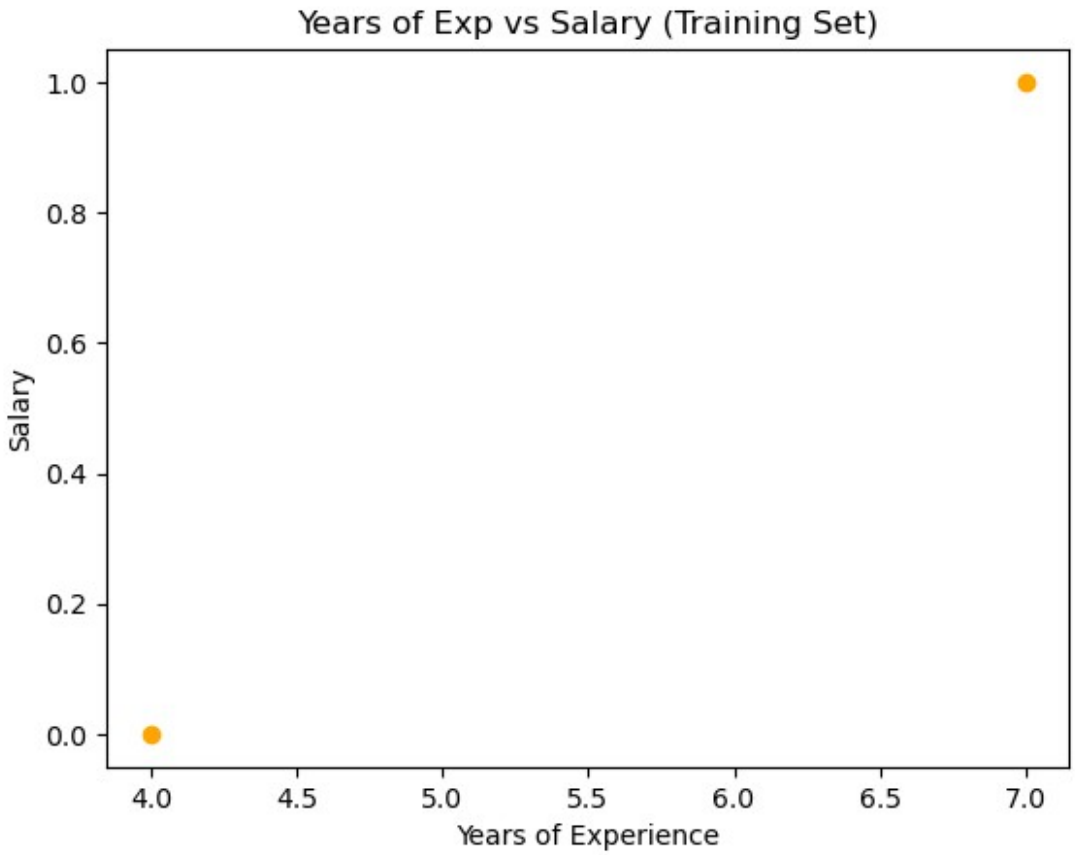
```
[0]
```

```
print("Actual values : ",y_test)
```

```
Actual values :  [1]
```

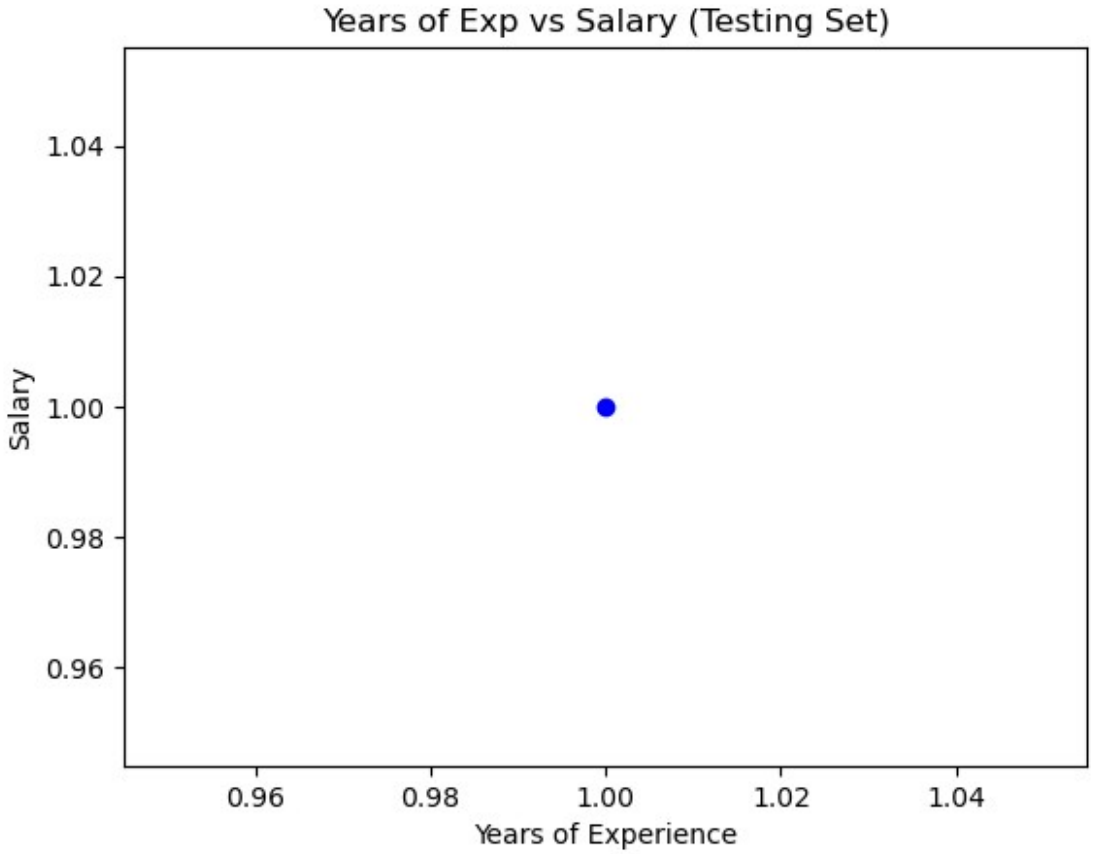
```
# https://matplotlib.org/stable/gallery/color/named_colors.html--for
colors
# Training set points
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Imputer to handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Training set points
plt.scatter(X_train[:, 0], y_train, color="orange")  # Ensure you
access the correct dimension for X_train
```

```
plt.title("Years of Exp vs Salary (Training Set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```



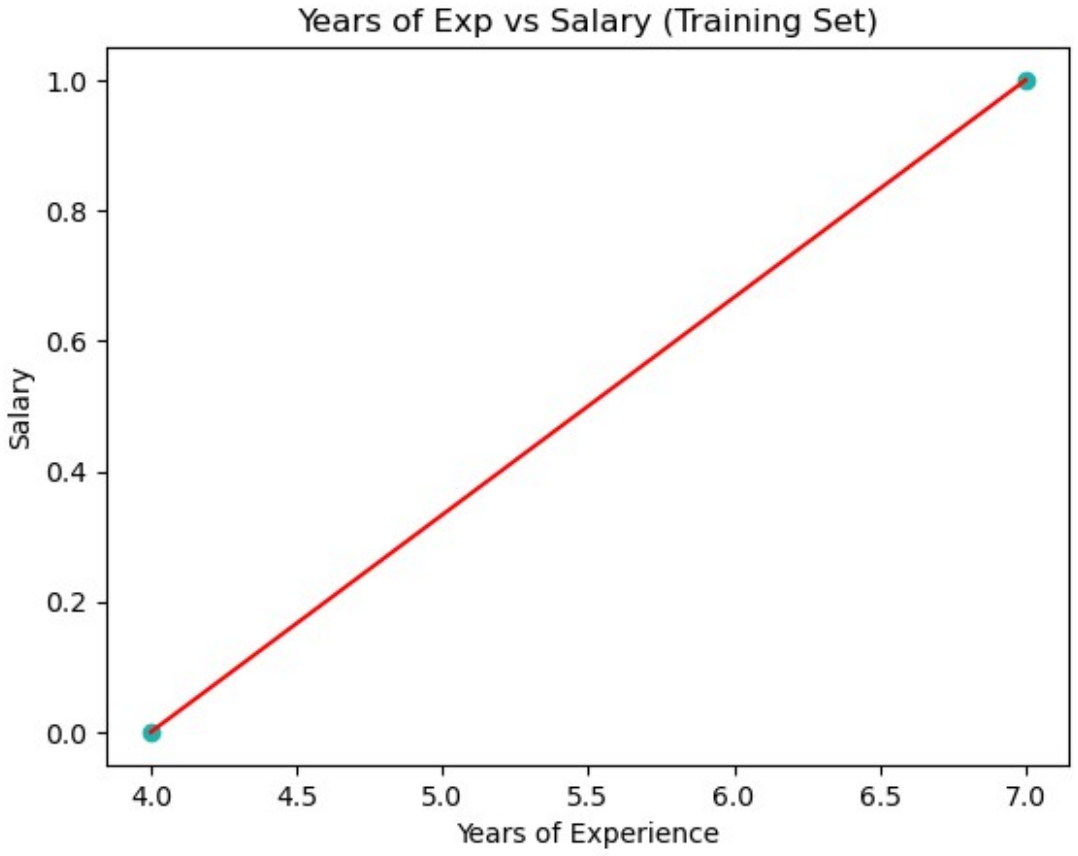

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Imputer to handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

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

```
# Train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Testing set points
plt.scatter(X_test[:, 0], y_test, color="blue")  # Access the first
column of X_test
plt.title("Years of Exp vs Salary (Testing Set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```

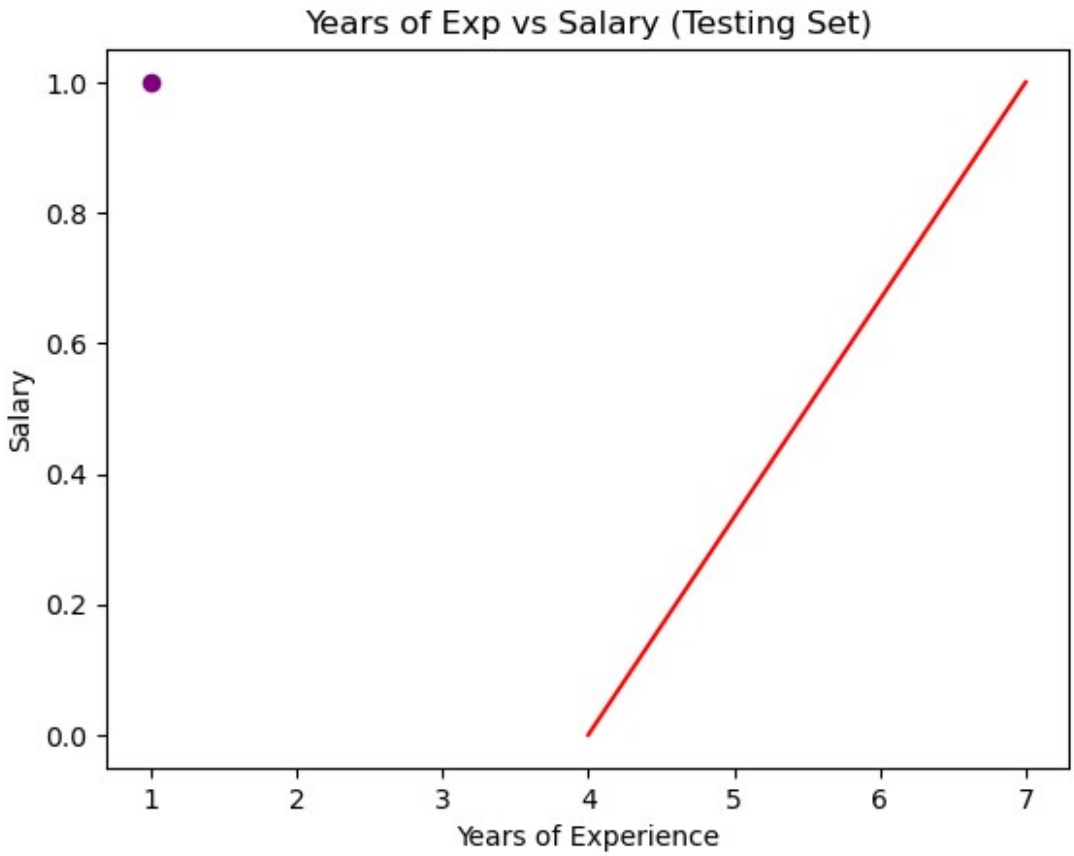



```
# Plot LR Model to the Traning set
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels
```

```
# Imputer to handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Plotting LR model to the Training set
plt.scatter(X_train[:, 0], y_train, color="lightseagreen")  # Scatter
plot of the training points
plt.plot(X_train[:, 0], lr.predict(X_train), color="red")  # Plotting
the model's predictions on the training set
plt.title("Years of Exp vs Salary (Training Set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```



```python
# Plot LR Model to the Testing set
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression

# Sample data with missing values
X = np.array([[1, 2, np.nan], [4, np.nan, 6], [7, 8, 9]])
y = np.array([1, 0, 1])  # Example target labels

# Imputer to handle missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Plotting LR model to the Testing set
plt.scatter(X_test[:, 0], y_test, color="purple")  # Scatter plot for
testing set points
plt.plot(X_train[:, 0], lr.predict(X_train), color="red")  # Red line
for the logistic regression model on training set
plt.title("Years of Exp vs Salary (Testing Set)")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```




```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import numpy as np

# Assuming y_pred is already defined as predictions from the model
mse_lr = mean_squared_error(y_test, y_pred)
rmse_lr = np.sqrt(mse_lr)  # RMSE is the square root of MSE
r2_lr = r2_score(y_test, y_pred)

# Adjusted R2
n = 4601  # Number of observations in the dataset
k = 1     # Number of predictor variables
adj_r2_score_lr = 1 - ((1 - r2_lr) * (n - 1) / (n - k - 1))

# Print the results
print("LR : Mean Square Error : ", mse_lr)
print("LR : RMSE : ", rmse_lr)
print("LR : R2 Score : ", r2_lr)
print('LR : Adjusted R2 :', adj_r2_score_lr)
```

```
LR : Mean Square Error :  1.0
LR : RMSE :  1.0
```

```
LR : R2 Score :  nan
LR : Adjusted R2 : nan

c:\Users\RUGVED GAIKWAD\anaconda3\Lib\site-packages\sklearn\metrics\
_regression.py:1211: UndefinedMetricWarning: R^2 score is not well-
defined with less than two samples.
  warnings.warn(msg, UndefinedMetricWarning)
```