Practice Machine Learning Algorithm: Linear Regression

In [1]:
```python
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
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
```

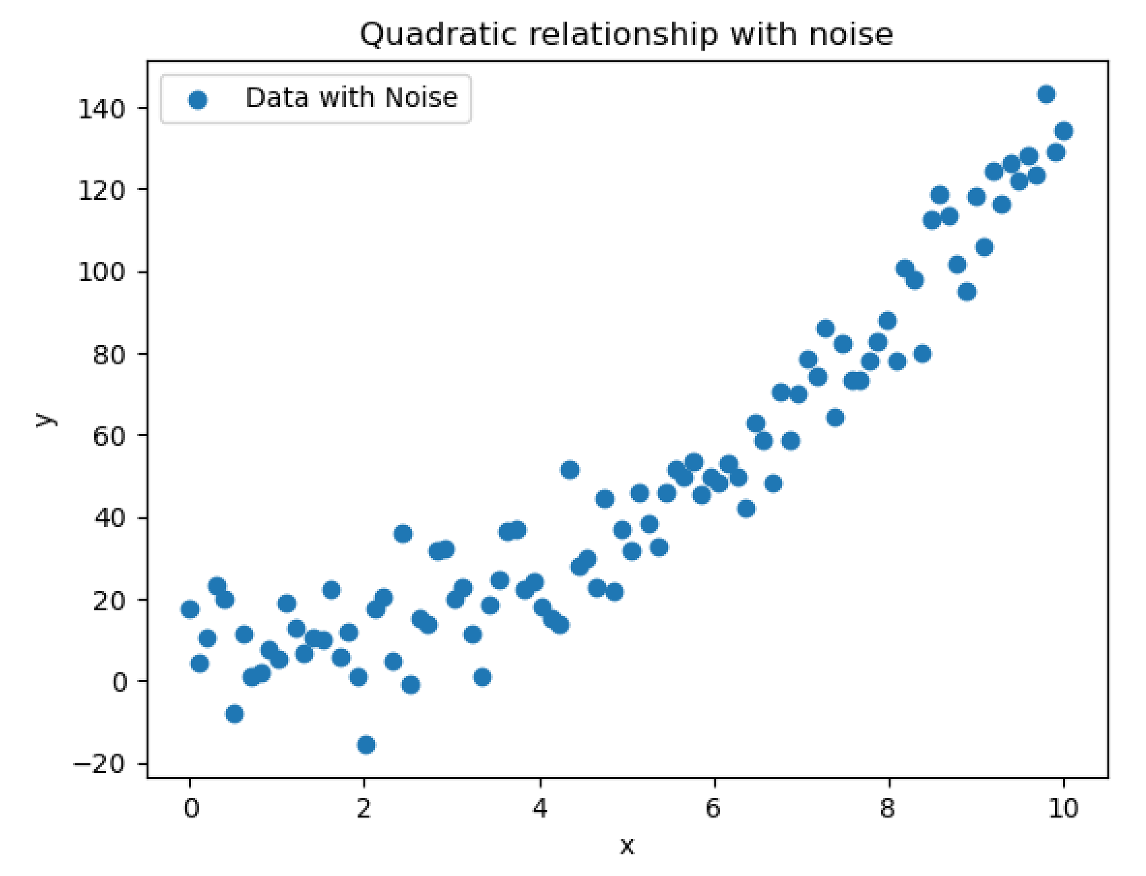
1. Generate independent and dependent variables

In [2]:
```python
np.random.seed(0)
# Using np.random.seed(0) ensures that the random numbers generated using NumPy will be the same each time we r
```

In [3]:
```python
x= np.linspace(0,10,100) # Independent variable
# The np.linspace function in NumPy is used to create an array of evenly spaced values over a specified range
```

In [4]:
```python
y= 3*x+x**2+np.random.normal(0,10,100) # Dependent variable
# Generates a set of y values based on a quadratic equation with some added noise.
```

In [5]:
```python
plt.scatter(x, y, label='Data with Noise')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Quadratic relationship with noise')
plt.legend()
plt.show()
```



2. Splitting the data into training and test sets

In [6]:
```python
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_state = 42)
# test_size=0.2: Specifies that 20% of the data should be used for testing and the remaining 80% for training
# random_state=42: Ensures reproducibility. The data is shuffled in a specific way to produce the same
# train-test split each time the code is run
```

In [7]:
```python
plt.scatter(x_train, y_train, color = 'blue', label ='Training data')
plt.scatter(x_test, y_test, color = 'red', label ='Test data')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Training and Testing Data split')
plt.legend()
plt.show()
```

In [8]:
```python
x_train # 1D array
```

Out[8]:
```
array([ 5.55555556,  8.88888889,  2.62626263,  4.24242424,  6.96969697,
        1.51515152,  4.04040404,  9.6969697 ,  0.90909091,  7.27272727,
        1.11111111,  4.74747475,  8.58585859,  2.82828283,  9.39393939,
        0.50505051,  6.66666667,  6.56565657,  3.53535354,  1.61616162,
        4.94949495,  3.43434343,  0.70707071,  9.5959596 ,  2.72727273,
        1.91919192,  8.18181818,  2.52525253,  6.26262626,  1.31313131,
        2.42424242,  0.3030303 ,  1.71717172,  3.83838384,  0.80808081,
        7.87878788,  0.60606061,  6.46464646,  3.63636364,  8.98989899,
        5.65656566, 10.        ,  5.45454545,  4.34343434,  5.05050505,
        6.76767677,  4.64646465,  6.86868687,  6.16161616,  9.7979798 ,
        7.97979798,  4.14141414,  5.85858586,  4.84848485,  9.8989899 ,
        5.75757576,  7.57575758,  3.23232323,  9.49494949,  5.95959596,
        6.36363636,  8.48484848,  3.73737374,  2.92929293,  0.1010101 ,
        5.25252525,  2.12121212,  0.2020202 ,  2.32323232,  8.78787879,
        9.19191919,  7.47474747,  8.68686869,  8.28282828,  2.02020202,
        6.06060606,  7.17171717,  1.41414141,  9.29292929,  5.15151515])
```

3. Training the model

In [9]:
```python
x_train= x_train.reshape(-1,1) # 1D converted to 2D array
# Machine learning models typically expect the input feature data (x_train) to be in a 2D array format where ea
# a sample and each column is a feature. This means if you have 80 samples and 1 feature, the shape should be (
# On the other hand, the target data (y_train) is expected to be a 1D array
# where each element corresponds to the target value of a sample, so its shape is (80,).
```

In [10]:
```python
model= LinearRegression()
```

In [11]:
```python
model.fit(x_train, y_train)
```

Out[11]:
```
▾ LinearRegression
LinearRegression()
```

In [12]:
```python
x_train.shape # 2d array
```

Out[12]:
```
(80, 1)
```

In [13]:
```python
y_train.shape # 1d array
```

Out[13]:
```
(80,)
```

In [14]:
```python
y_train
```

Out[14]:
```
array([ 51.8141829 ,  94.97148613,  15.23362843,  13.66273427,
        70.00522072,  10.17788193,  17.96054728, 123.2271306 ,
         7.65970403,  86.10475065,  19.1106363 ,  44.55584428,
       118.43343517,  31.81182439, 126.19430588,  -8.00255127,
        48.14246097,  58.78700647,  24.66827492,  22.40125395,
        37.2185823 ,  18.61862363,   1.10758902, 127.93605104,
        13.74799621,   0.89991599, 100.49586818,  -0.59099885,
        49.87690367,   6.88045795,  35.84722484,  23.40985027,
         6.04861122,  22.37507383,   2.0450485 ,  82.59613675,
        11.68637545,  62.95985473,  36.43513821, 118.33249809,
        49.63160422, 134.01989363,  45.8338802 ,  51.40347888,
        31.7044508 ,  70.73230169,  23.00107405,  58.71193626,
        52.85483059, 143.25305245,  88.17822316,  15.37537414,
        45.5555649 ,  21.9142814 , 128.95609164,  53.44712485,
        73.27127466,  11.2670257 , 122.20257837,  49.76816022,
        42.32395084, 112.3297212 ,  37.20388215,  32.06222355,
         4.31480543,  38.23854591,  17.39936318,  10.43425261,
         4.94545519, 101.7912016 , 124.29158671,  82.31950865,
       113.31008936,  98.11035361, -15.3880759 ,  48.18815953,
        74.23850781,  10.68085251, 116.3200725 ,  45.86167879])
```

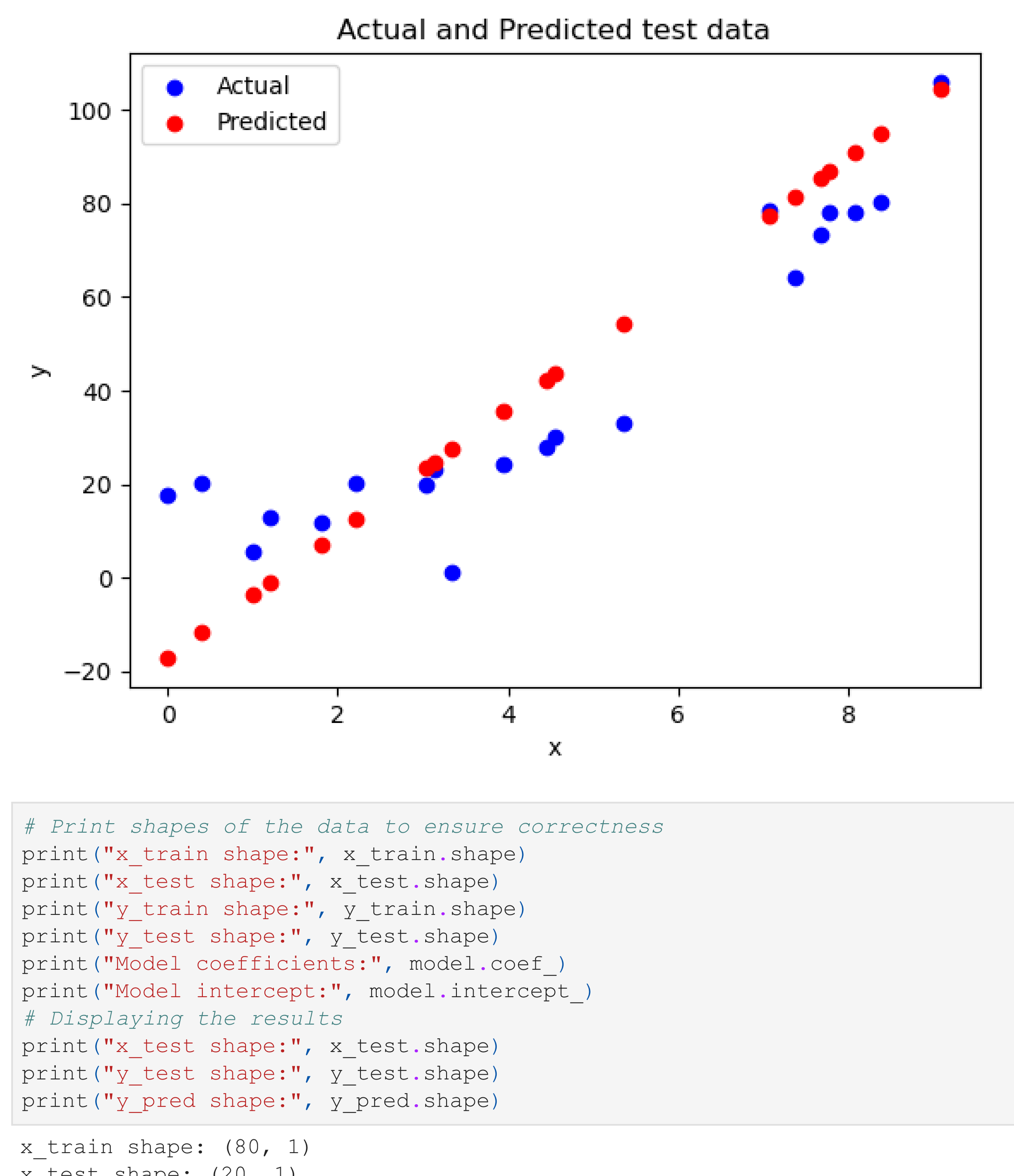
4. Model used to make predictions on the test data

In [15]:
```python
x_test= x_test.reshape(-1,1)
x_test
```

Out[15]:
```
array([[8.38383838],
       [5.35353535],
       [7.07070707],
       [4.54545455],
       [4.44444444],
       [3.93939394],
       [2.22222222],
       [8.08080808],
       [1.01010101],
       [0.        ],
       [1.81818182],
       [3.03030303],
       [7.37373737],
       [3.33333333],
       [9.09090909],
       [0.4040404 ],
       [7.67676768],
       [7.77777778],
       [1.21212121],
       [3.13131313]])
```

In [16]:
```python
y_pred= model.predict(x_test)
```

In [17]:
```python
plt.scatter(x_test, y_test, color='blue', label='Actual')
plt.scatter(x_test, y_pred, color='red', label='Predicted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Actual and Predicted test data')
plt.legend()
plt.show()
```

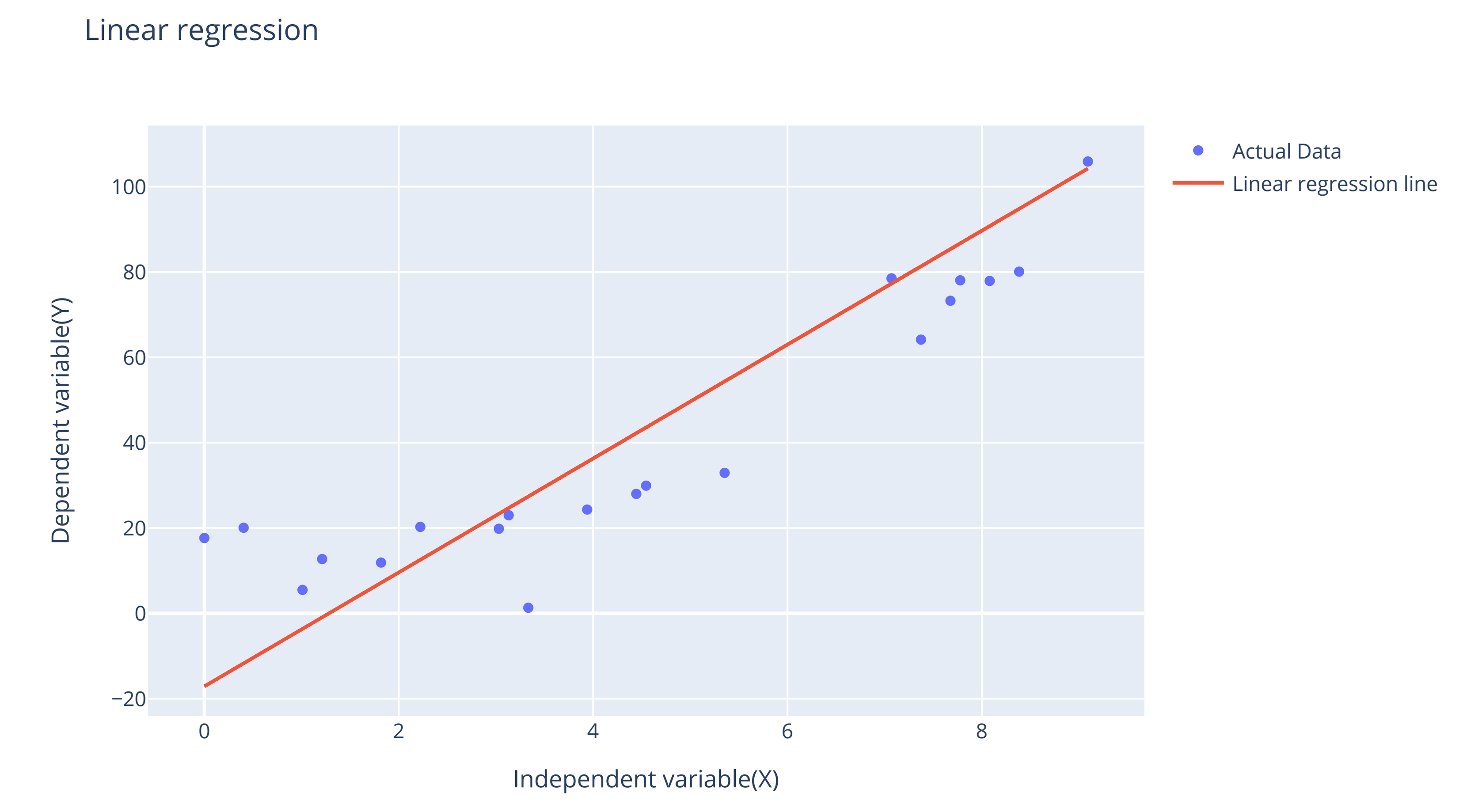


In [18]:
```python
# Print shapes of the data to ensure correctness
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
print("Model coefficients:", model.coef_)
print("Model intercept:", model.intercept_)
# Displaying the results
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)
print("y_pred shape:", y_pred.shape)
```

```
x_train shape: (80, 1)
x_test shape: (20, 1)
y_train shape: (80,)
y_test shape: (20,)
Model coefficients: [13.34962728]
Model intercept: -17.12238419162958
x_test shape: (20, 1)
y_test shape: (20,)
y_pred shape: (20,)
```

5. Results visualized using Plotly

In [19]:
```python
fig=go.Figure()
fig.add_trace(go.Scatter(x= x_test.flatten(), y= y_test, mode= 'markers', name= 'Actual Data'))
fig.add_trace(go.Scatter(x= x_test.flatten(), y= y_pred, mode= 'lines', name= 'Linear regression line'))
fig.update_layout(title ='Linear regression', xaxis_title ='Independent variable(X)',yaxis_title ='Dependent va
fig.show()
```



6. Model Evaluation

In [20]:
```python
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

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

```
Mean Squared Error: 256.9949119622545
R-squared: 0.7258608090887344
```

Observation:

- Mean squared error gives you an idea of how well your model is performing in terms of prediction accuracy. Lower values of MSE indicate better model performance.
- An R-squared value of 0.7258608090887344 indicates that approximately 72.59% of the variance in the actual values is explained by the model's predictions. This suggests that the model has a good fit, explaining a significant portion of the variance, but there is still about 27.41% of the variance that is not explained by the model.

In [ ]: