CoxAssignment04

Univariate Linear Regression

Exploratory Data Analysis

In [1]:
```
import pandas as pd
```

In [2]:
```
path_to_file = 'student_scores.csv'
df = pd.read_csv(path_to_file)
```

In [3]:
```
df.head()
```

Out[3]:

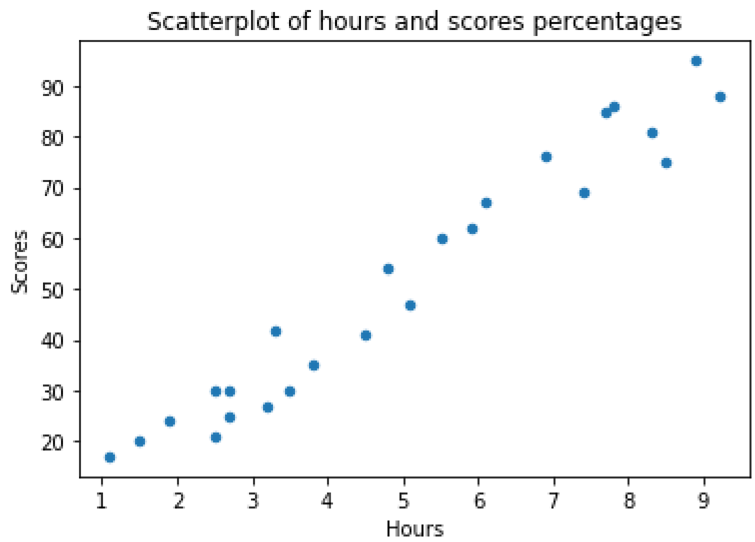
	Hours	Scores
0	2.5	21
1	5.1	47
2	3.2	27
3	8.5	75
4	3.5	30

```
- The first three steps were used to import the necessary library and
to read the file to display the head to show the file was loaded
correctly
```

In [4]:
```
df.shape
```

Out[4]:
```
(25, 2)
```

In [5]:
```
df.plot.scatter(x='Hours', y='Scores', title='Scatterplot of hours and scores percenta
```



- This scatter plot shows that as the hours increase, so does the scores showing that there is a high correlation since the shape of the data points appear straight

In [6]:
```
print(df.corr())
```

```
           Hours    Scores
Hours   1.000000  0.976191
Scores  0.976191  1.000000
```

In [7]:
```
print(df.describe())
```

```
           Hours     Scores
count  25.000000  25.000000
mean    5.012000  51.480000
std     2.525094  25.286887
min     1.100000  17.000000
25%     2.700000  30.000000
50%     4.800000  47.000000
75%     7.400000  75.000000
max     9.200000  95.000000
```

Data Preprocessing

In [8]:
```
y = df['Scores'].values.reshape(-1, 1)
X = df['Hours'].values.reshape(-1, 1)
```

In [9]:
```
print(df['Hours'].values)
print(df['Hours'].values.shape)
```

```
[2.5 5.1 3.2 8.5 3.5 1.5 9.2 5.5 8.3 2.7 7.7 5.9 4.5 3.3 1.1 8.9 2.5 1.9
 6.1 7.4 2.7 4.8 3.8 6.9 7.8]
(25,)
```

- By extracting the values above we have a 1D array compared to the code one line below this which shows it in a 2D array where hour is a 1-element array

In [10]:
```
print(X.shape)
print(X)
```

```
(25, 1)
[[2.5]
 [5.1]
 [3.2]
 [8.5]
 [3.5]
 [1.5]
 [9.2]
 [5.5]
 [8.3]
 [2.7]
 [7.7]
 [5.9]
 [4.5]
 [3.3]
 [1.1]
 [8.9]
 [2.5]
 [1.9]
 [6.1]
 [7.4]
 [2.7]
 [4.8]
 [3.8]
 [6.9]
 [7.8]]
```

In [11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

- This is used to train our model so that we can test it to determine how accurate it is

In [12]:
```
SEED = 42
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

- The defining SEED allows us to be able to have the same results when we run the test then we print the X_train and y_train below to obtain the study hours and score percentages

In [14]:
```
print(X_train)
print(y_train)
```

```
[[2.7]
 [3.3]
 [5.1]
 [3.8]
 [1.5]
 [3.2]
 [4.5]
 [8.9]
 [8.5]
 [3.5]
 [2.7]
 [1.9]
 [4.8]
 [6.1]
 [7.8]
 [5.5]
 [7.7]
 [1.1]
 [7.4]
 [9.2]]
[[25]
 [42]
 [47]
 [35]
 [20]
 [27]
 [41]
 [95]
 [75]
 [30]
 [30]
 [24]
 [54]
 [67]
 [86]
 [60]
 [85]
 [17]
 [69]
 [88]]
```

Training a Linear Regression Model

In [15]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
```

In [16]:
```
regressor.fit(X_train, y_train)
```

Out[16]:
```
LinearRegression()
```

In [17]:
```
print(regressor.intercept_)
```

```
[2.82689235]
```

- The last three statements of code are used to train the test sets using the LinearRegression model and fitting our line to the data finally showing that our regressor has found the best fitting line for our data

In [18]:
```
print(regressor.coef_)
```

```
[[9.68207815]]
```

- This statement finds the slope which is also the coefficient of x

Making Predictions

In [19]:
```
def calc(slope, intercept, hours):
    return slope*hours+intercept

score = calc(regressor.coef_, regressor.intercept_, 9.5)
print(score)
```

```
[[94.80663482]]
```

- This was used to avoid running calculations ourselves possibly elimating any human error by using a calculator assuming our equation above is correct

In [20]:
```
score = regressor.predict([[9.5]])
print(score)
```

```
[[94.80663482]]
```

- This method does the same as above with less code and decreasing our chance of human error since there is less code.

In [21]:
```
y_pred = regressor.predict(X_test)
```

In [23]:
```
df_preds = pd.DataFrame({'Actual': y_test.squeeze(), 'Predicted': y_pred.squeeze()})
print(df_preds)
```

```
   Actual  Predicted
0      81  83.188141
1      30  27.032088
2      21  27.032088
3      76  69.633232
4      62  59.951153
```

- This was used to predict our test data to ensure that our results from the start of this section were accurate, as the site stated: "the ground truth results"

Evaluating the Model

In [24]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

In [25]:
```
import numpy as np
```

In [26]:
```
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

In [27]:
```
print(f'Mean absolute error: {mae:.2f}')
print(f'Mean squared error: {mse:.2f}')
print(f'Root mean squared error: {rmse:.2f}')
```

```
Mean absolute error: 3.92
Mean squared error: 18.94
Root mean squared error: 4.35
```

- Thankfully sklearn has ways of calculating the MAE, MSE, and RMSE which helps us determine how well our model predicts by using these metrics. We can use any of these metrics to compare different models. We can also use hyperparameter tuning to compare the same regression model with different values or data then decide which metrics to use. From the results we can see that all of our errors are low and are only missing the true value by +-4.35 which is a small range for the data we have.

In [ ]: