In [3]:
```
import pandas as pd
```

In [5]:
```
df = pd.read_csv('Student_Marks.csv')
```

In [108]:
```
df.head()
```

Out[108]:

	number_courses	time_study	Marks
0	3	4.508	19.202
1	4	0.096	7.734
2	4	3.133	13.811
3	6	7.909	53.018
4	8	7.811	55.299

Data Preprocessing Tasks

1> Checking Null Values

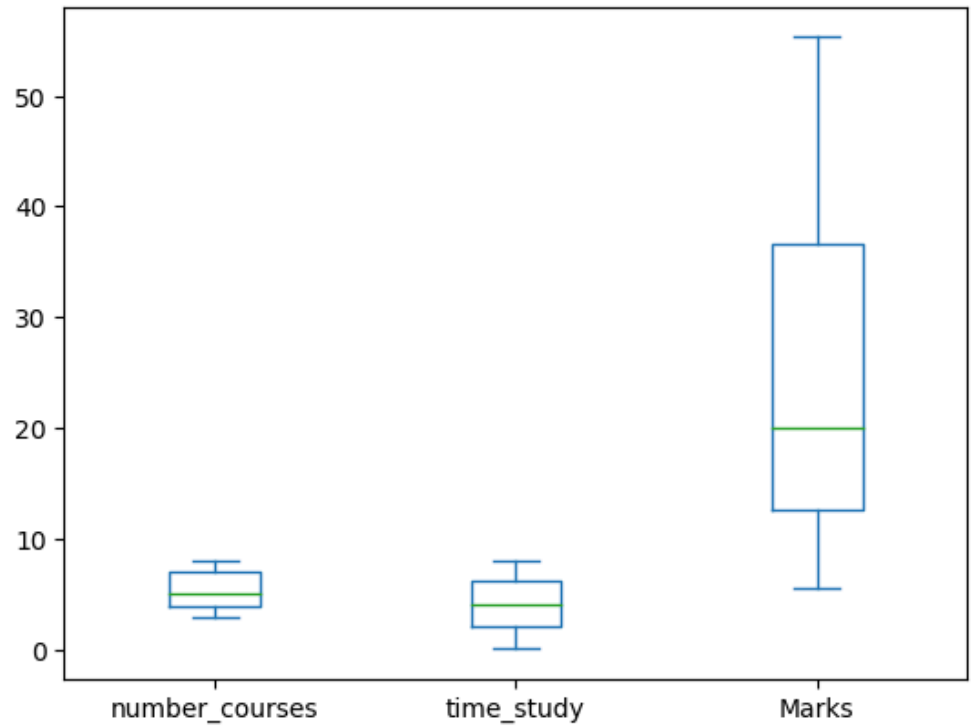
In [9]:
```
df.isnull().sum()
```

Out[9]:
```
number_courses    0
time_study        0
Marks             0
dtype: int64
```

2> Detecting Outliers

In [10]:
```
df.plot(kind='box')
```

Out[10]: <Axes: >



Min-Max Normalization

In [12]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [13]:
```
min_max_scaler = MinMaxScaler()
```

In [16]:
```
df_normalized = min_max_scaler.fit_transform(df)
```

In [17]:
```
df_normalized  = pd.DataFrame(df_normalized,columns = df.columns)
print(df_normalized)
```

```
    number_courses  time_study     Marks
0              0.0    0.561252  0.273556
1              0.2    0.000000  0.042765
2              0.2    0.386338  0.165063
3              0.6    0.993894  0.954095
4              1.0    0.981427  1.000000
..             ...         ...       ...
95             0.6    0.440784  0.272067
96             0.0    0.026078  0.000000
97             0.2    0.898995  0.721171
98             0.8    0.027096  0.129161
99             0.0    0.793665  0.538297

[100 rows x 3 columns]
```

Unit-Vector (L2) Normalization

In [23]:
```
from sklearn.preprocessing import Normalizer
```

In [26]:
```
normalizer = Normalizer(norm='l2')
```

In [27]:
```
df_norm = normalizer.fit_transform(df)
```

In [28]:
```
df_norm = pd.DataFrame(df_norm,columns = df.columns)
print(df_norm)
```

```
    number_courses  time_study     Marks
0         0.150369    0.225955  0.962462
1         0.459364    0.011025  0.888180
2         0.271814    0.212898  0.938505
3         0.111236    0.146627  0.982918
4         0.141799    0.138449  0.980166
..             ...         ...       ...
95        0.294684    0.174895  0.939453
96        0.471105    0.047268  0.880810
97        0.094679    0.169546  0.980964
98        0.502902    0.022200  0.864058
99        0.090614    0.191346  0.977331

[100 rows x 3 columns]
```

Z-Score Scaling

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

In [30]:
```
standard_scaler = StandardScaler()
```

In [31]:
```
df_standardized = standard_scaler.fit_transform(df)
```

In [32]:
```
df_standardized = pd.DataFrame(df_standardized,columns=df.columns)
print(df_standardized)
```

```
    number_courses  time_study     Marks
0        -1.278970    0.182489 -0.365901
1        -0.720468   -1.686195 -1.170425
2        -0.720468   -0.399887 -0.744100
3         0.396537    1.622968  2.006422
4         1.513541    1.581461  2.166442
..             ...         ...       ...
95        0.396537   -0.218609 -0.371092
96       -1.278970   -1.599368 -1.319502
97       -0.720468    1.307003  1.194461
98        0.955039   -1.595980 -0.869254
99       -1.278970    0.956307  0.556973

[100 rows x 3 columns]
```

Robust Scaling

In [33]:
```
from sklearn.preprocessing import RobustScaler
```

In [34]:
```
robust_scaler = RobustScaler()
```

In [35]:
```
df_robust = robust_scaler.fit_transform(df)
df_robust = pd.DataFrame(df_robust,columns=df.columns)
print(df_robust)
```

```
    number_courses  time_study     Marks
0        -0.666667    0.117940 -0.035665
1        -0.333333   -0.952739 -0.512639
2        -0.333333   -0.215737 -0.259886
3         0.333333    0.943275  1.370801
4         1.000000    0.919493  1.465671
..             ...         ...       ...
95        0.333333   -0.111873 -0.038743
96       -0.666667   -0.902991 -0.601021
97       -0.333333    0.762240  0.889418
98        0.666667   -0.901050 -0.334085
99       -0.666667    0.561306  0.511474

[100 rows x 3 columns]
```

Comparative Analysis using 5 different models

In [57]:
```
from sklearn.model_selection import train_test_split
```

In [58]:
```
X=df.drop(columns='Marks')
y=df['Marks']
```

In [59]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 0
```

In [60]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [73]:
```
print(X_train_scaled.shape,X_test_scaled.shape)
print(y_train.shape,y_test.shape)
```

```
(80, 2) (20, 2)
(80,) (20,)
```

1> Random Forest Regressor

In [61]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train_scaled, y_train)
```

Out[61]:
```
RandomForestRegressor
RandomForestRegressor(n_estimators=10, random_state=0)
```

In [62]:
```
y_predict = regressor.predict(X_test_scaled)
print(y_predict)
```

```
[12.1674 23.9882 14.9112 14.6821  6.2797 41.358  30.3518  7.2138 49.6929
 19.2053 36.1274 23.929  47.273  18.9088 17.5028 22.478  11.8461 16.6351
 41.2824 20.7415]
```

In [63]:
```
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_predict)
```

Out[63]: 1.9975615395000055

2> SVRegression

In [64]:
```
from sklearn.svm import SVR
regressor = SVR(kernel = 'linear')
regressor.fit(X_train_scaled,y_train)
```

Out[64]:
```
SVR
SVR(kernel='linear')
```

In [65]:
```
y_predict = regressor.predict(X_test_scaled)
print(y_predict)
```

```
[ 8.49727673 26.65805466 16.90994468 20.76945176  3.35977192 40.58583281
 31.76642796  7.57466446 46.30094135 22.41534438 36.77175566 26.57460144
 43.92584688 21.07150561 19.99942771 27.28289714 13.04944751 19.47153797
 37.8920624  23.42618897]
```

In [103]:
```
mean_squared_error(y_test, y_predict)
```

Out[103]: 6.5437269

3> Decision Tree

In [86]:
```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train_scaled, y_train)
```

Out[86]:
```
DecisionTreeRegressor
DecisionTreeRegressor(random_state=0)
```

In [87]:
```
y_predict = regressor.predict(X_test_scaled)
print(y_predict)
```

```
[12.591 24.451 10.844 13.416  5.609 41.358 35.939  6.053 55.299 17.822
 40.024 28.043 54.321 19.202 16.106 24.451 10.429 16.606 40.602 19.466]
```

In [104]:
```
mean_squared_error(y_test, y_predict)
```

Out[104]: 6.5437269

4> Linear Regression

In [92]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train_scaled, y_train)
```

Out[92]:
```
LinearRegression
LinearRegression()
```

In [93]:
```
y_pred = regressor.predict(X_test_scaled)
```

In [105]:
```
mean_squared_error(y_test, y_pred)
```

Out[105]: 13.703258318207736

5> Polynomial Regression

In [95]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree=5)
X_poly = poly_reg.fit_transform(X_train)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y_train)
```

Out[95]:
```
LinearRegression
LinearRegression()
```

In [96]:
```
y_pred_1 = lin_reg_2.predict(poly_reg.fit_transform(X_test))
print(y_pred_1)
```

```
[12.31564747 23.59712272 13.85497489 18.07749707  6.0767785  42.15047746
 31.17334154  7.5011399  52.80569752 19.32259038 36.98810795 24.12586526
 49.52729993 17.96498151 16.96923365 24.15707949 11.83566108 16.12084093
 40.05701877 20.26043507]
```

In [106]:
```
mean_squared_error(y_test, y_pred_1)
```

Out[106]: 0.09331223177810442

Comparing the 5 models

In [107]:
```
mse_results = {
    'Random Forest Regressor': 1.9975615395000055,
    'Support Vector Regressor (SVR)': 6.5437269,
    'Decision Tree Regressor': 6.5437269,
    'Linear Regression': 13.703258318207736,
    'Polynomial Regression': 0.09331223177810442
}

# Print the MSE values for each model
print("Mean Squared Error (MSE) for each model:")
for model, mse in mse_results.items():
    print(f"{model}: {mse}")

# Find the model with the lowest MSE
best_model = min(mse_results, key=mse_results.get)
print("\nThe best model based on Mean Squared Error (MSE):", best_model)
```

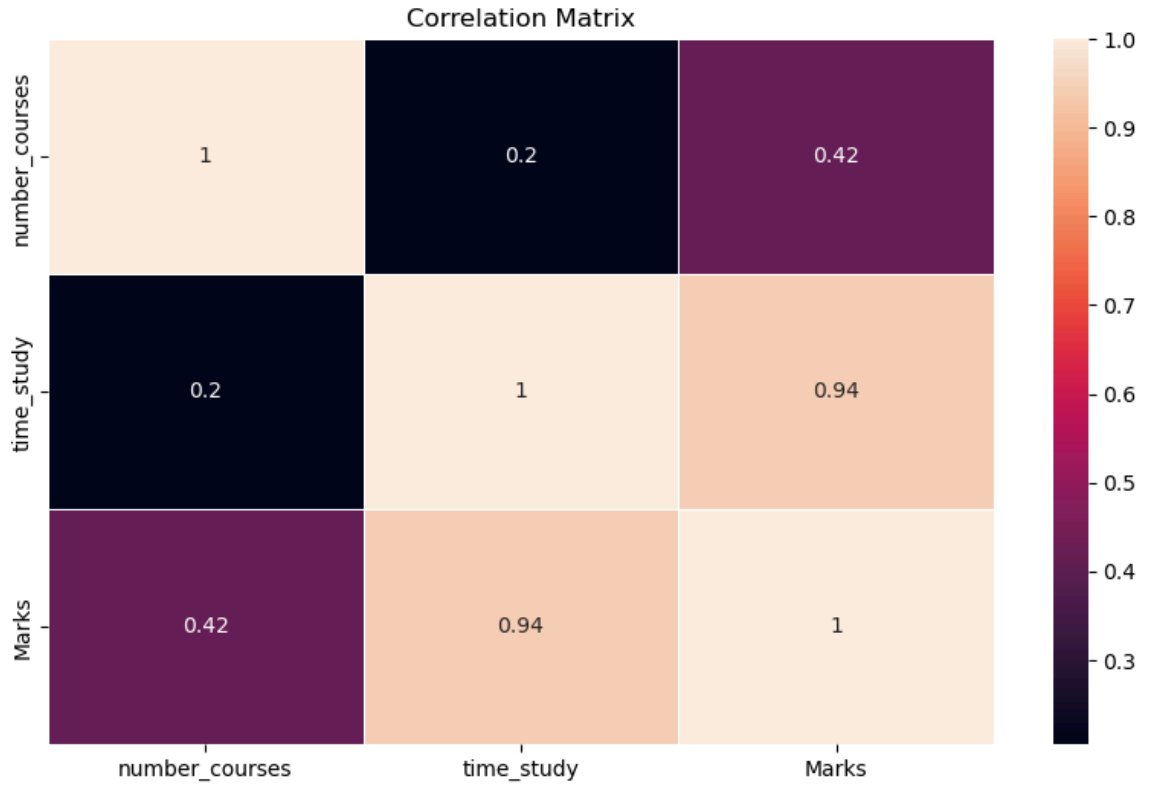
```
Mean Squared Error (MSE) for each model:
Random Forest Regressor: 1.9975615395000055
Support Vector Regressor (SVR): 6.5437269
Decision Tree Regressor: 6.5437269
Linear Regression: 13.703258318207736
Polynomial Regression: 0.09331223177810442

The best model based on Mean Squared Error (MSE): Polynomial Regression
```

Correlation Analysis

In [101]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



From the heatmap it is evident that Time Studied is directly proportional to the Marks obtained by the students.

We can drop Number_of_courses and see whether it will improve the accuracy of the model

In [110]:
```
df_reduced = df.drop(columns=['number_courses'])
```

In [111]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

We are using the Decision Tree Model to check for improvement

In [112]:
```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(X_train_scaled, y_train)
```

Out[112]:
```
DecisionTreeRegressor

DecisionTreeRegressor(random_state=0)
```

In [113]:
```
y_predict = regressor.predict(X_test_scaled)
print(y_predict)
```

```
[12.132 25.133 13.119 15.725  6.349 41.358 26.882  6.623 53.359 19.771
 38.49  28.043 51.583 19.202 19.106 24.451 13.562 16.606 40.602 19.466]
```

In [114]:
```
mean_squared_error(y_test, y_predict)
```

Out[114]:
```
2.9798997999999997
```

Earlier the Decision Tree Model was producing an error of 6.543 whereas after removing 'Number_courses' feature, it is producing error of 2.979 thereby indicating Improvement in Performance.

In [ ]:
```

```