Creating dataset of Student Performance

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

np.random.seed(42)

num_instances = 30


math_scores = np.random.randint(60, 81, num_instances)
reading_scores = np.random.randint(75, 96, num_instances).astype(float)
writing_scores = np.random.randint(60, 81, num_instances)
placement_scores = np.random.randint(75, 101, num_instances).astype(float)
club_join_dates = np.random.randint(2018, 2022, num_instances)
placement_offer_count = (placement_scores / 10).astype(int)

# Introduce exactly NaN values into each column
num_impurities = int(0.2 * num_instances)
nan_indices_reading = np.random.choice(num_instances, size=num_impurities, replace=False)
nan_indices_placement = np.random.choice(num_instances, size=num_impurities, replace=False)


reading_scores[nan_indices_reading] = np.nan
placement_scores[nan_indices_placement] = np.nan


df = pd.DataFrame({
    "Math_Score": math_scores,
    "Reading_Score": reading_scores,
    "Writing_Score": writing_scores,
    "Placement_Score": placement_scores,
    "Club_Join_Date": club_join_dates,
    "Placement_Offer_Count": placement_offer_count
})

df.to_csv("StudentsPerformance.csv", index=False)

print("Dataset saved as 'StudentsPerformance.csv'")
```

```
Dataset saved as 'StudentsPerformance.csv'
```

In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv('StudentsPerformance.csv')
data
```

Out[18]:

	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
0	66	94.0	69	93.0	2020	9
1	79	77.0	63	NaN	2019	9
2	74	NaN	77	82.0	2020	8
3	70	93.0	71	77.0	2018	7
4	67	NaN	61	77.0	2018	7
5	80	95.0	69	75.0	2019	7
6	66	83.0	63	79.0	2020	7
7	78	NaN	73	84.0	2020	8
8	70	92.0	75	81.0	2019	8
9	70	78.0	74	100.0	2020	10
10	80	88.0	67	83.0	2020	8
11	63	92.0	73	81.0	2018	8
12	67	83.0	67	83.0	2020	8
13	62	95.0	80	NaN	2020	8
14	80	76.0	75	86.0	2019	8
15	61	NaN	72	76.0	2019	7
16	71	89.0	77	75.0	2021	7
17	65	81.0	74	90.0	2018	9
18	61	86.0	80	97.0	2020	9
19	80	82.0	72	97.0	2020	9
20	60	NaN	68	98.0	2021	9
21	71	77.0	74	NaN	2020	7
22	71	88.0	72	77.0	2018	7
23	76	91.0	60	86.0	2021	8
24	69	78.0	66	82.0	2018	8
25	75	92.0	68	NaN	2021	9
26	74	82.0	60	NaN	2021	7
27	74	NaN	71	75.0	2019	7
28	78	76.0	67	77.0	2018	7
29	71	80.0	70	NaN	2020	7

In [19]:

```
data1 = data.copy()
data2 = data.copy()
```

In [20]:

```
#describing the data
data.describe()
```

Out[20]:

	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
count	30.000000	24.000000	30.000000	24.000000	30.000000	30.000000
mean	70.966667	85.333333	70.266667	83.791667	2019.500000	7.900000
std	6.283439	6.644165	5.438898	7.994450	1.042213	0.884736
min	60.000000	76.000000	60.000000	75.000000	2018.000000	7.000000
25%	66.250000	79.500000	67.000000	77.000000	2019.000000	7.000000
50%	71.000000	84.500000	71.000000	82.000000	2020.000000	8.000000
75%	75.750000	92.000000	74.000000	87.000000	2020.000000	8.750000
max	80.000000	95.000000	80.000000	100.000000	2021.000000	10.000000

Handling Null Values

In [21]:

```
data.isna().sum()
```

```
Out[21]: Math_Score            0
         Reading_Score         6
         Writing_Score         0
         Placement_Score       6
         Club_Join_Date        0
         Placement_Offer_Count 0
         dtype: int64
```

```
In [22]: #filling null values for first data for minmax scaling
         data['Reading_Score'] = data['Reading_Score'].fillna(data['Reading_Score'].mean())
         data['Placement_Score'] = data['Placement_Score'].fillna(data['Placement_Score'].median())
         data.isna().sum()
```

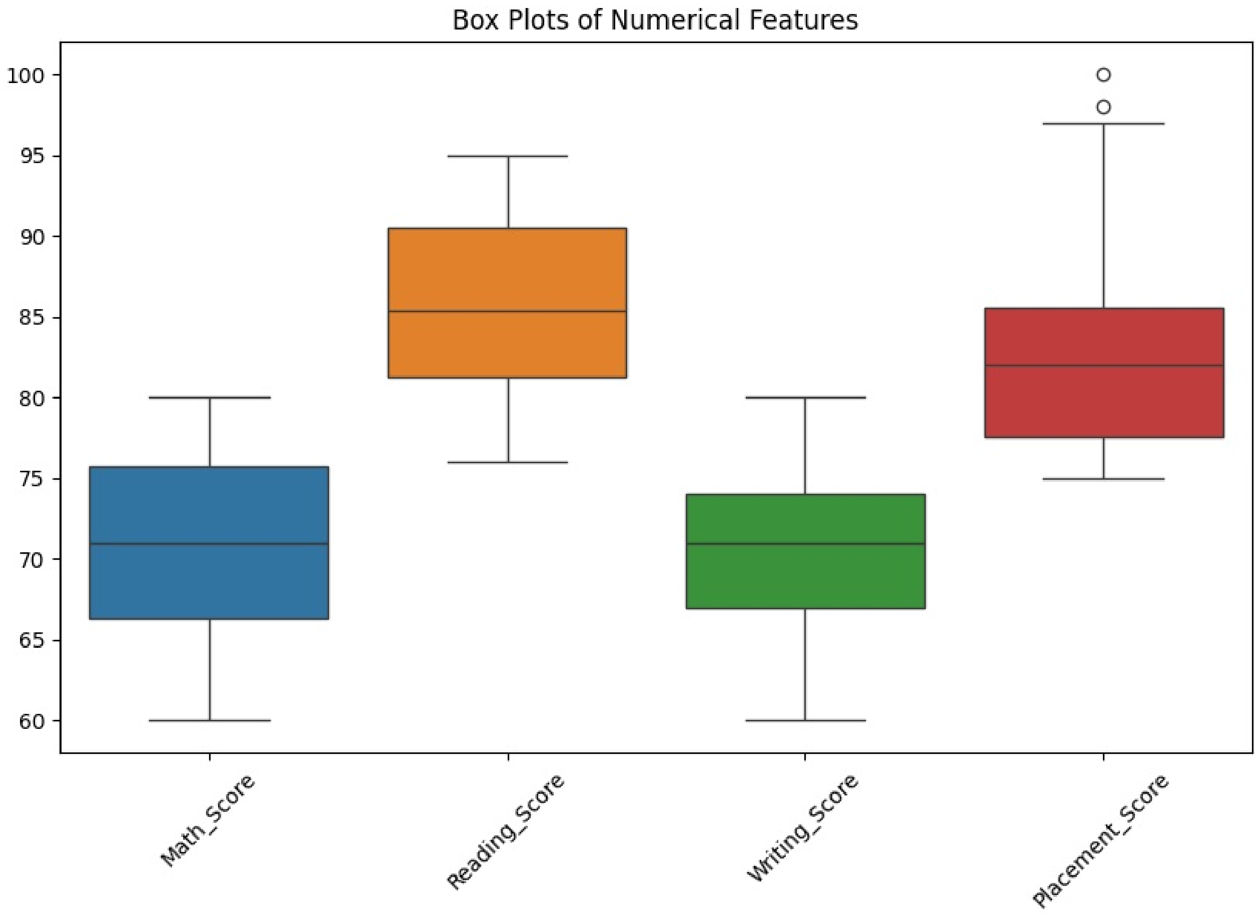
```
Out[22]: Math_Score            0
         Reading_Score         0
         Writing_Score         0
         Placement_Score       0
         Club_Join_Date        0
         Placement_Offer_Count 0
         dtype: int64
```

```
In [23]: #filling null values for second data for normlization
         data1['Reading_Score'] = data1['Reading_Score'].fillna(data1['Reading_Score'].mean())
         data1['Placement_Score'] = data1['Placement_Score'].fillna(data1['Placement_Score'].median())
         data1.isna().sum()
```

```
Out[23]: Math_Score            0
         Reading_Score         0
         Writing_Score         0
         Placement_Score       0
         Club_Join_Date        0
         Placement_Offer_Count 0
         dtype: int64
```

Detecting and handling outliers

```
In [24]: # Box plot for all numerical columns
         plt.figure(figsize=(10, 6))
         sns.boxplot(data=data[['Math_Score', 'Reading_Score', 'Writing_Score', 'Placement_Score']])
         plt.title('Box Plots of Numerical Features')
         plt.xticks(rotation=45)
         plt.show()
```




```
In [25]: #handling outlier using iqr
         def handle_outlier(data,column):
             Q1 = data[column].quantile(0.25)
```

```
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 -1.5* IQR
    upper_bound = Q3 + 1.5 * IQR
    data[column] = np.where(data[column]> upper_bound,upper_bound,data[column])
    data[column] = np.where(data[column] < lower_bound,lower_bound,data[column])
    return data
data = handle_outlier(data, 'Math_Score')
data = handle_outlier(data, 'Reading_Score')
data = handle_outlier(data, 'Writing_Score')
data = handle_outlier(data, 'Placement_Score')
```

In [26]:
```
data
```

Out[26]:

	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
0	66.0	94.000000	69.0	93.0	2020	9
1	79.0	77.000000	63.0	82.0	2019	9
2	74.0	85.333333	77.0	82.0	2020	8
3	70.0	93.000000	71.0	77.0	2018	7
4	67.0	85.333333	61.0	77.0	2018	7
5	80.0	95.000000	69.0	75.0	2019	7
6	66.0	83.000000	63.0	79.0	2020	7
7	78.0	85.333333	73.0	84.0	2020	8
8	70.0	92.000000	75.0	81.0	2019	8
9	70.0	78.000000	74.0	97.5	2020	10
10	80.0	88.000000	67.0	83.0	2020	8
11	63.0	92.000000	73.0	81.0	2018	8
12	67.0	83.000000	67.0	83.0	2020	8
13	62.0	95.000000	80.0	82.0	2020	8
14	80.0	76.000000	75.0	86.0	2019	8
15	61.0	85.333333	72.0	76.0	2019	7
16	71.0	89.000000	77.0	75.0	2021	7
17	65.0	81.000000	74.0	90.0	2018	9
18	61.0	86.000000	80.0	97.0	2020	9
19	80.0	82.000000	72.0	97.0	2020	9
20	60.0	85.333333	68.0	97.5	2021	9
21	71.0	77.000000	74.0	82.0	2020	7
22	71.0	88.000000	72.0	77.0	2018	7
23	76.0	91.000000	60.0	86.0	2021	8
24	69.0	78.000000	66.0	82.0	2018	8
25	75.0	92.000000	68.0	82.0	2021	9
26	74.0	82.000000	60.0	82.0	2021	7
27	74.0	85.333333	71.0	75.0	2019	7
28	78.0	76.000000	67.0	77.0	2018	7
29	71.0	80.000000	70.0	82.0	2020	7

In [27]:
```
#handling the outlier using zscore
from scipy import stats
import pandas as pd
import numpy as np

def handle_outliers_zscore(data, column, threshold=3):
    z_scores = np.abs(stats.zscore(data[column]))
    data_filtered = data[(z_scores < threshold)]
    return data_filtered
data1 = handle_outliers_zscore(data1, 'Math_Score')
data1 = handle_outliers_zscore(data1, 'Reading_Score')
data1 = handle_outliers_zscore(data1, 'Writing_Score')
data1 = handle_outliers_zscore(data1, 'Placement_Score')
data1
```

Out[27]:

	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
0	66	94.000000	69	93.0	2020	9
1	79	77.000000	63	82.0	2019	9
2	74	85.333333	77	82.0	2020	8
3	70	93.000000	71	77.0	2018	7
4	67	85.333333	61	77.0	2018	7
5	80	95.000000	69	75.0	2019	7
6	66	83.000000	63	79.0	2020	7
7	78	85.333333	73	84.0	2020	8
8	70	92.000000	75	81.0	2019	8
9	70	78.000000	74	100.0	2020	10
10	80	88.000000	67	83.0	2020	8
11	63	92.000000	73	81.0	2018	8
12	67	83.000000	67	83.0	2020	8
13	62	95.000000	80	82.0	2020	8
14	80	76.000000	75	86.0	2019	8
15	61	85.333333	72	76.0	2019	7
16	71	89.000000	77	75.0	2021	7
17	65	81.000000	74	90.0	2018	9
18	61	86.000000	80	97.0	2020	9
19	80	82.000000	72	97.0	2020	9
20	60	85.333333	68	98.0	2021	9
21	71	77.000000	74	82.0	2020	7
22	71	88.000000	72	77.0	2018	7
23	76	91.000000	60	86.0	2021	8
24	69	78.000000	66	82.0	2018	8
25	75	92.000000	68	82.0	2021	9
26	74	82.000000	60	82.0	2021	7
27	74	85.333333	71	75.0	2019	7
28	78	76.000000	67	77.0	2018	7
29	71	80.000000	70	82.0	2020	7

Data Normalization

In [28]:

```
#using minmax Normalization
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
data_scaled = pd.DataFrame(min_max_scaler.fit_transform(data[data.columns]), columns=data.columns, index=data.i
data = data_scaled
data
```

Out[28]:

	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
0	0.30	0.947368	0.45	0.800000	0.666667	0.666667
1	0.95	0.052632	0.15	0.311111	0.333333	0.666667
2	0.70	0.491228	0.85	0.311111	0.666667	0.333333
3	0.50	0.894737	0.55	0.088889	0.000000	0.000000
4	0.35	0.491228	0.05	0.088889	0.000000	0.000000
5	1.00	1.000000	0.45	0.000000	0.333333	0.000000
6	0.30	0.368421	0.15	0.177778	0.666667	0.000000
7	0.90	0.491228	0.65	0.400000	0.666667	0.333333
8	0.50	0.842105	0.75	0.266667	0.333333	0.333333
9	0.50	0.105263	0.70	1.000000	0.666667	1.000000
10	1.00	0.631579	0.35	0.355556	0.666667	0.333333
11	0.15	0.842105	0.65	0.266667	0.000000	0.333333
12	0.35	0.368421	0.35	0.355556	0.666667	0.333333
13	0.10	1.000000	1.00	0.311111	0.666667	0.333333
14	1.00	0.000000	0.75	0.488889	0.333333	0.333333
15	0.05	0.491228	0.60	0.044444	0.333333	0.000000
16	0.55	0.684211	0.85	0.000000	1.000000	0.000000
17	0.25	0.263158	0.70	0.666667	0.000000	0.666667
18	0.05	0.526316	1.00	0.977778	0.666667	0.666667
19	1.00	0.315789	0.60	0.977778	0.666667	0.666667
20	0.00	0.491228	0.40	1.000000	1.000000	0.666667
21	0.55	0.052632	0.70	0.311111	0.666667	0.000000
22	0.55	0.631579	0.60	0.088889	0.000000	0.000000
23	0.80	0.789474	0.00	0.488889	1.000000	0.333333
24	0.45	0.105263	0.30	0.311111	0.000000	0.333333
25	0.75	0.842105	0.40	0.311111	1.000000	0.666667
26	0.70	0.315789	0.00	0.311111	1.000000	0.000000
27	0.70	0.491228	0.55	0.000000	0.333333	0.000000
28	0.90	0.000000	0.35	0.088889	0.000000	0.000000
29	0.55	0.210526	0.50	0.311111	0.666667	0.000000

In [29]:

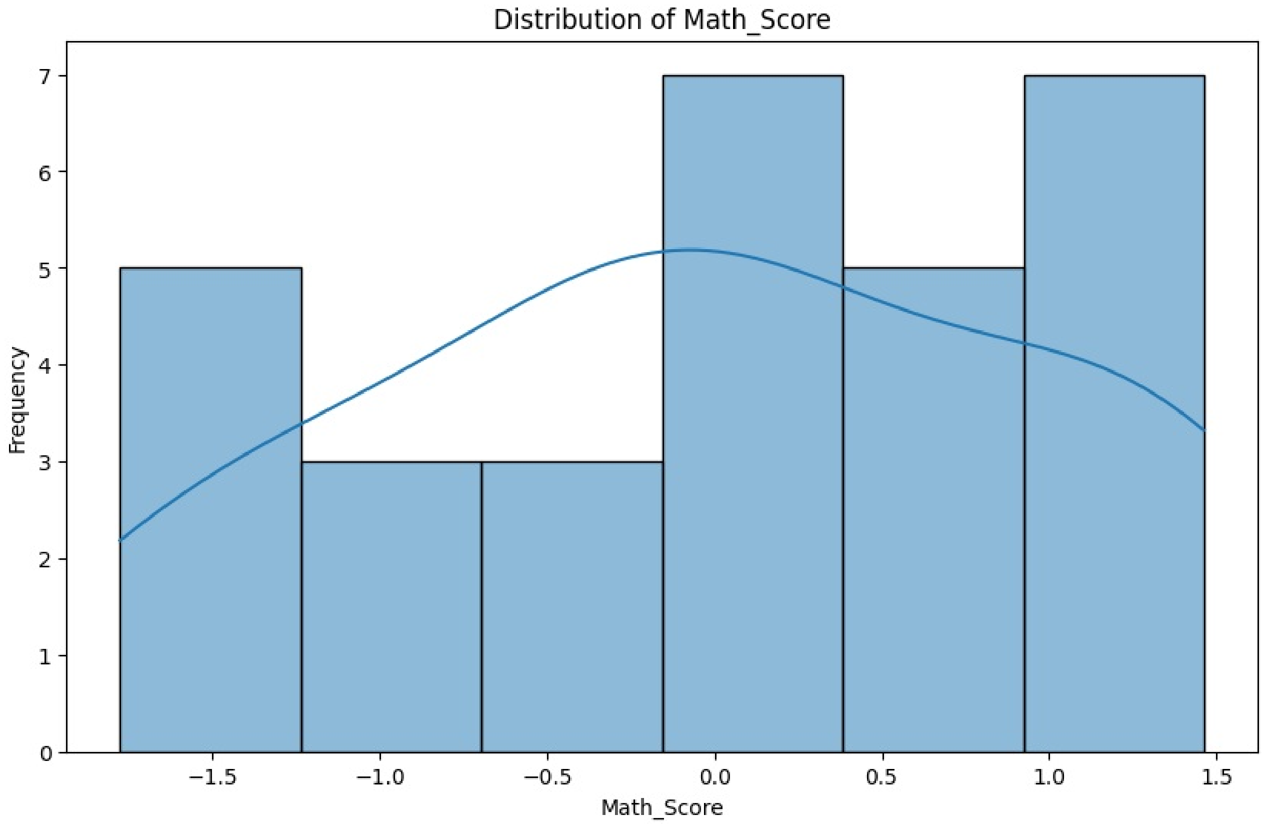
```
#using zscore for normalization
from scipy.stats import zscore

def normalize_data(data):
    numerical_cols = data1.columns

    # Apply z-score normalization
    for col in numerical_cols:
        data[col] = zscore(data[col])

    return data

data1 = normalize_data(data)
data1
```

Out[29]:

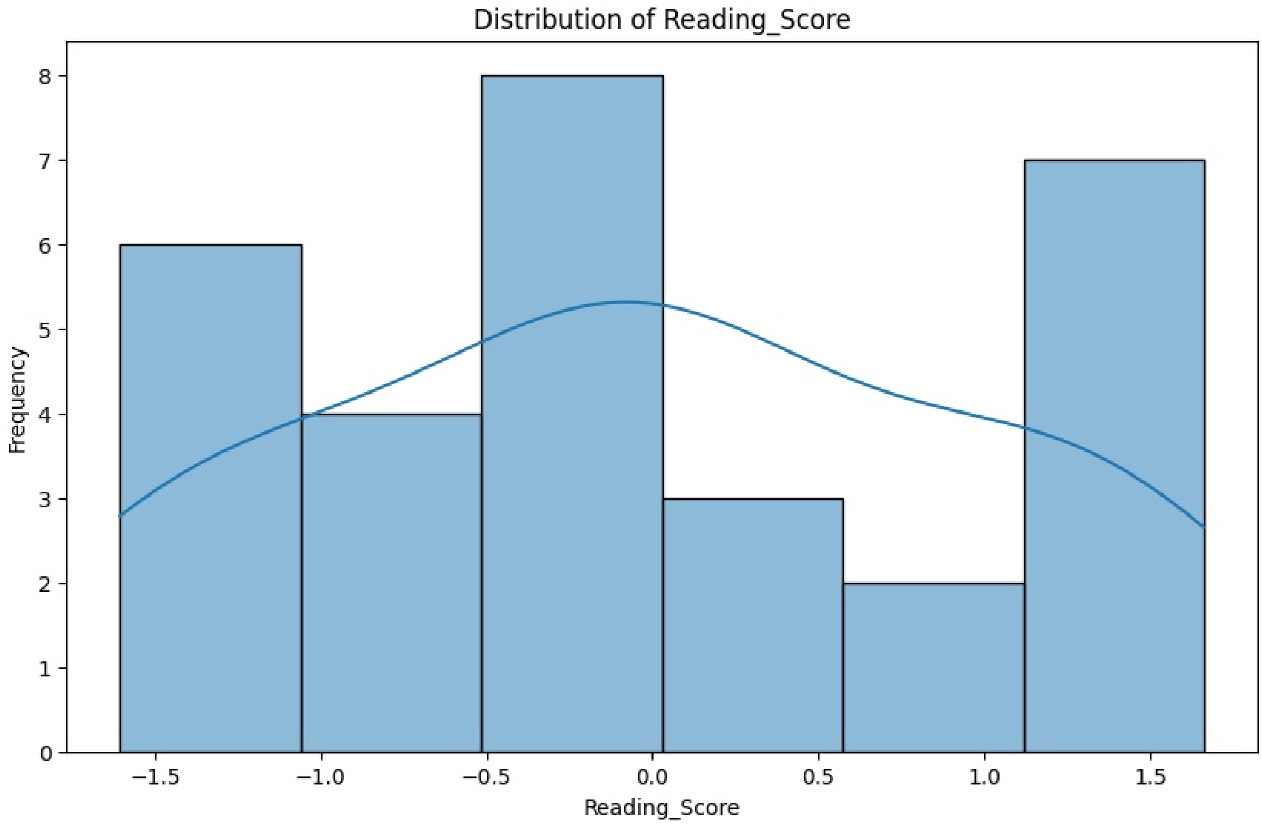
	Math_Score	Reading_Score	Writing_Score	Placement_Score	Club_Join_Date	Placement_Offer_Count
0	-0.803950	1.489733e+00	-0.236872	1.418012	0.48795	1.264563
1	1.300349	-1.432435e+00	-1.358895	-0.195588	-0.48795	1.264563
2	0.491003	-3.625936e-16	1.259160	-0.195588	0.48795	0.114960
3	-0.156474	1.317841e+00	0.137136	-0.929043	-1.46385	-1.034642
4	-0.642081	-3.625936e-16	-1.732903	-0.929043	-1.46385	-1.034642
5	1.462218	1.661625e+00	-0.236872	-1.222424	-0.48795	-1.034642
6	-0.803950	-4.010819e-01	-1.358895	-0.635661	0.48795	-1.034642
7	1.138480	-3.625936e-16	0.511144	0.097794	0.48795	0.114960
8	-0.156474	1.145948e+00	0.885152	-0.342279	-0.48795	0.114960
9	-0.156474	-1.260543e+00	0.698148	2.078122	0.48795	2.414165
10	1.462218	4.583793e-01	-0.610880	-0.048897	0.48795	0.114960
11	-1.289558	1.145948e+00	0.511144	-0.342279	-1.46385	0.114960
12	-0.642081	-4.010819e-01	-0.610880	-0.048897	0.48795	0.114960
13	-1.451427	1.661625e+00	1.820172	-0.195588	0.48795	0.114960
14	1.462218	-1.604328e+00	0.885152	0.391176	-0.48795	0.114960
15	-1.613296	-3.625936e-16	0.324140	-1.075734	-0.48795	-1.034642
16	0.005396	6.302716e-01	1.259160	-1.222424	1.46385	-1.034642
17	-0.965820	-7.448664e-01	0.698148	0.977940	-1.46385	1.264563
18	-1.613296	1.145948e-01	1.820172	2.004776	0.48795	1.264563
19	1.462218	-5.729742e-01	0.324140	2.004776	0.48795	1.264563
20	-1.775166	-3.625936e-16	-0.423876	2.078122	1.46385	1.264563
21	0.005396	-1.432435e+00	0.698148	-0.195588	0.48795	-1.034642
22	0.005396	4.583793e-01	0.324140	-0.929043	-1.46385	-1.034642
23	0.814742	9.740561e-01	-1.919907	0.391176	1.46385	0.114960
24	-0.318343	-1.260543e+00	-0.797883	-0.195588	-1.46385	0.114960
25	0.652872	1.145948e+00	-0.423876	-0.195588	1.46385	1.264563
26	0.491003	-5.729742e-01	-1.919907	-0.195588	1.46385	-1.034642
27	0.491003	-3.625936e-16	0.137136	-1.222424	-0.48795	-1.034642
28	1.138480	-1.604328e+00	-0.610880	-0.929043	-1.46385	-1.034642
29	0.005396	-9.167587e-01	-0.049868	-0.195588	0.48795	-1.034642

In [30]:

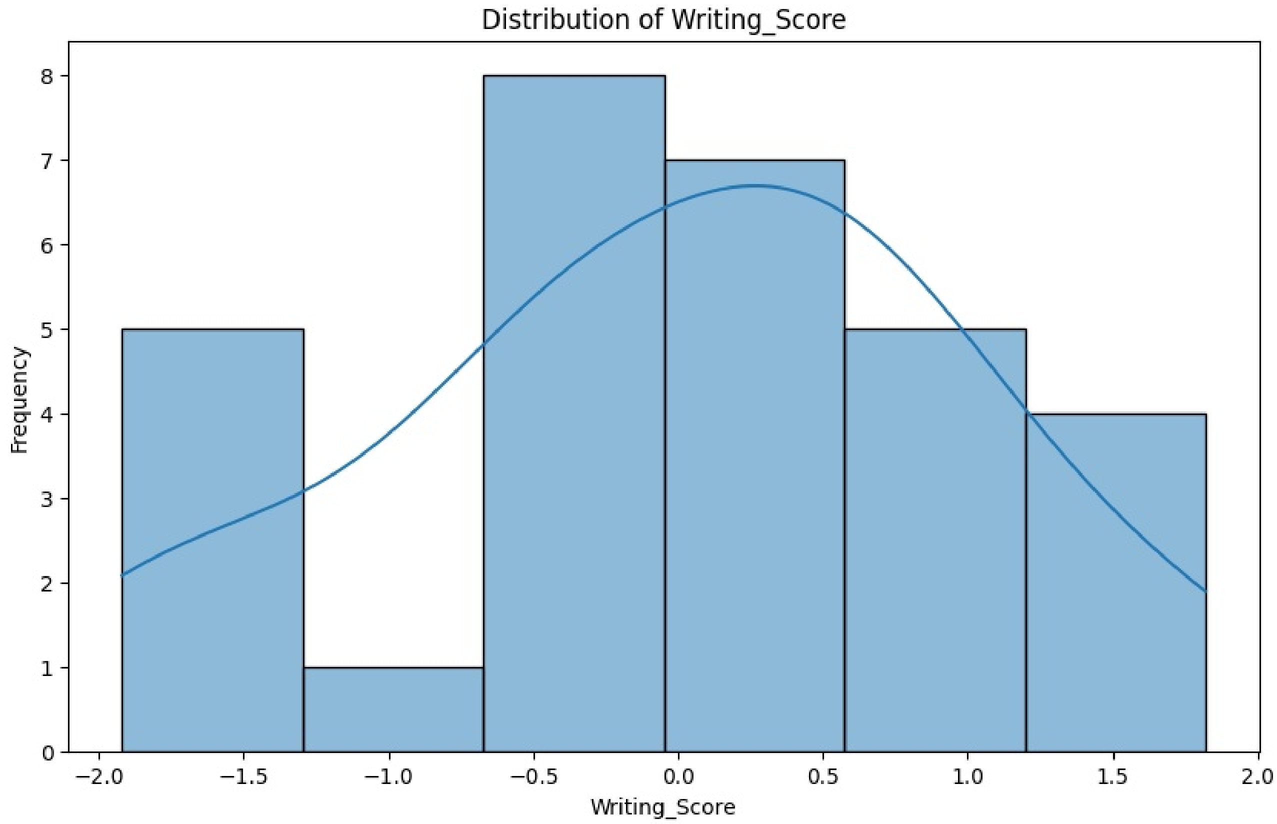
```python
#plotting histogram for checking the skweness of attributes
def plot_skewness(data,column):
    plt.figure(figsize=(10,6))
    sns.histplot(data[column],kde=True)
    plt.title(f'Distribution of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
    skewness = data[column].skew()
    print(f"Skewness of {column}: {skewness}")
    if skewness > 0.5 or skewness < -0.5 :
      print(f'{column} is skewed')
    else:
      print(f'{column} is fairly symmetrical')
plot_skewness(data, 'Math_Score')
plot_skewness(data, 'Reading_Score')
plot_skewness(data, 'Writing_Score')
plot_skewness(data, 'Placement_Score')
```



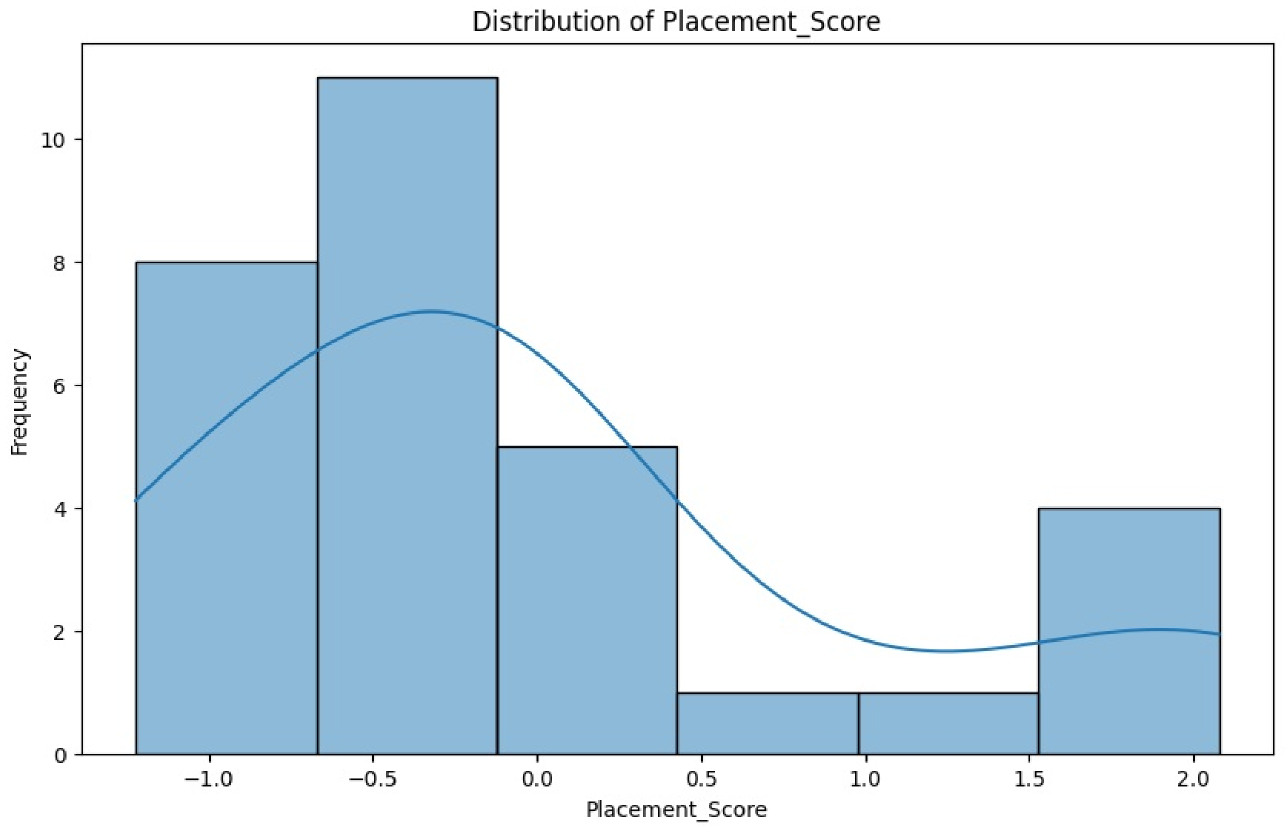

```
Skewness of Math_Score: -0.10104718790415089
Math_Score is fairly symmetrical
```




```
Skewness of Reading_Score: 0.028851481680230343
Reading_Score is fairly symmetrical
```




```
Skewness of Writing_Score: -0.24268945522328003
Writing_Score is fairly symmetrical
```




```
Skewness of Placement_Score: 0.9706086380422002
Placement_Score is skewed
```

In [ ]: