Lab Manual

Submitted To:

Sir Saeed

Submitted By:

Waleed Amjad

Registration no#:

2023-BS-AI-054

Degree Program:

BS - Artificial Intelligence

Semester:

04 Sec:A

Lab Report – Machine Learning

Table of Contents

Regression

Title:

Linear Regression on Student Performance

Data Description:

Preprocessing of a student data set to find the hours of studying and other activities which effect the overall performance of the student.

Code and Output:

Problem Description: Preprocessing of a student data set to find the hours of studying and other activities which effect the overall performance of the student.

Data Preprocessing Steps

1. Reading Data
2. Exploring Data / Data Insight
3. Cleansing Data
4. Outlier Detection and Removing
5. Data Transformation (Normalize Data / Rescale Data)
6. Categorical into Numerical
7. Dimensionality Reduction(PCA)
8. Handling Imbalanced Data
9. Feature Selection
10. Data Splitting

In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()

import numpy as np
import pandas as pd
```

1: Reading Data

In [13]:
```
data = pd.read_csv('Student_Performance.csv')

data.head()
```

Out[13]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0

In [14]:
```
data.head(2)
```

Out[14]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0

In [15]:
```
data.head(30)
```

Out[15]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
0	7	99	Yes	9	1	91.0
1	4	82	No	4	2	65.0
2	8	51	Yes	7	2	45.0
3	5	52	Yes	5	2	36.0
4	7	75	No	8	5	66.0
5	3	78	No	9	6	61.0
6	7	73	Yes	5	6	63.0
7	8	45	Yes	4	6	42.0
8	5	77	No	8	2	61.0
9	4	89	No	4	0	69.0
10	8	91	No	4	5	84.0

11	8	79	No	6	2	73.0
12	3	47	No	9	2	27.0
13	6	47	No	4	2	33.0
14	5	79	No	7	8	68.0
15	2	72	No	4	3	43.0
16	8	73	Yes	8	4	67.0
17	6	83	Yes	7	2	70.0
18	2	54	Yes	4	9	30.0
19	5	75	No	7	0	63.0
20	1	99	Yes	4	3	71.0
21	6	96	No	9	0	85.0
22	9	74	Yes	7	6	73.0
23	1	85	No	5	6	57.0
24	3	61	No	6	3	35.0
25	7	62	Yes	7	4	49.0
26	4	79	No	8	9	66.0
27	9	84	Yes	6	6	83.0
28	3	94	Yes	6	5	74.0
29	5	90	Yes	4	3	74.0

In [16]:
```
data.tail()
```

Out[16]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
9995	1	49	Yes	4	2	23.0
9996	7	64	Yes	8	5	58.0
9997	6	83	Yes	8	5	74.0
9998	9	97	Yes	7	0	95.0
9999	7	74	No	8	1	64.0

In [17]:
```
data.shape
```

Out[17]: (10000, 6)

In [18]:
```
data.tail(20)
```

Out[18]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
9980	2	43	No	6	9	20.0
9981	7	54	No	9	4	46.0
9982	8	51	No	5	1	44.0
9983	8	87	Yes	4	9	79.0
9984	6	45	Yes	6	2	34.0
9985	8	99	No	5	5	92.0
9986	1	48	Yes	8	5	25.0
9987	9	74	No	4	6	69.0
9988	1	47	No	8	5	20.0
9989	3	46	No	5	8	27.0
9990	9	43	No	7	4	40.0
9991	5	97	Yes	7	4	83.0
9992	9	52	No	9	7	50.0
9993	9	58	Yes	7	7	55.0
9994	6	46	Yes	8	0	39.0
9995	1	49	Yes	4	2	23.0
9996	7	64	Yes	8	5	58.0
9997	6	83	Yes	8	5	74.0
9998	9	97	Yes	7	0	95.0
9999	7	74	No	8	1	64.0

In [19]:
```
data.sample()
```

In [20]:
```
data.sample(30)
```

Out[20]:

	Hours Studied	Previous Scores	Extracurricular Activities	Sleep Hours	Sample Question Papers Practiced	Performance Index
6032	7	47	Yes	7	5	34.0
7646	4	63	Yes	7	6	48.0
493	4	91	No	9	5	80.0
8531	8	50	Yes	5	6	48.0
439	7	79	Yes	9	4	73.0
7089	8	57	Yes	6	0	49.0
8289	3	83	No	8	4	65.0
4652	4	84	Yes	4	5	71.0
5728	4	89	No	7	4	75.0
8435	2	52	No	6	0	28.0
6804	2	99	No	6	2	73.0
4669	1	46	No	9	9	23.0
1003	6	75	No	7	7	61.0
6395	5	62	Yes	7	9	51.0
7819	6	59	Yes	7	7	48.0
4390	9	96	No	7	4	93.0
4445	9	57	No	8	9	55.0
9706	4	51	Yes	7	4	32.0
8943	8	86	Yes	6	4	82.0
7145	4	58	Yes	4	5	41.0
3037	4	68	No	8	3	51.0
1786	2	62	Yes	9	4	40.0
5624	3	97	No	7	1	78.0
1790	2	50	Yes	9	5	26.0
8395	6	70	No	9	6	61.0
3496	7	52	Yes	5	0	45.0
9430	4	46	Yes	9	0	32.0

In [21]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   Hours Studied                     10000 non-null  int64
 1   Previous Scores                   10000 non-null  int64
 2   Extracurricular Activities        10000 non-null  object
 3   Sleep Hours                       10000 non-null  int64
 4   Sample Question Papers Practiced  10000 non-null  int64
 5   Performance Index                 10000 non-null  float64
dtypes: float64(1), int64(4), object(1)
memory usage: 468.9+ KB
```

In [22]:
```
data.describe()
```

Out[22]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	4.992900	69.445700	6.530600	4.583300	55.224800
std	2.589309	17.343152	1.695863	2.867348	19.212558
min	1.000000	40.000000	4.000000	0.000000	10.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000
75%	7.000000	85.000000	8.000000	7.000000	71.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000

2: Data Cleaning

2: Data Cleaning

Handling Missing Values

- Imputation: Filling missing values with mean.

In [23]:
```
import pandas as pd
```

In [24]:
```
data.isnull().sum()
```

Out[24]:
```
Hours Studied                       0
Previous Scores                     0
Extracurricular Activities          0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
dtype: int64
```

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


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


numeric_cols.fillna(numeric_cols.mean(), inplace=True)




data = pd.concat([numeric_cols, non_numeric_cols], axis=1)


missing_values = data.isnull().sum()
print(missing_values)
```

```
Hours Studied                       0
Previous Scores                     0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
Extracurricular Activities          0
dtype: int64
```

In [26]:
```python
import pandas as pd
import numpy as np


numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


numeric_cols.fillna(numeric_cols.mean(), inplace=True)

for col in non_numeric_cols.columns:
    non_numeric_cols[col].fillna(non_numeric_cols[col].mode()[0], inplace=True)


data = pd.concat([numeric_cols, non_numeric_cols], axis=1)


missing_values = data.isnull().sum()
print(missing_values)
```

```
Hours Studied                       0
Previous Scores                     0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
Extracurricular Activities          0
dtype: int64
```

In [27]:
```python
data.shape
```

Out[27]:
```
(10000, 6)
```

Removal: Deleting rows with missing values.

In [28]:
```python
data.isnull().sum()
```

Out[28]:
```
Hours Studied                       0
Previous Scores                     0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
Extracurricular Activities          0
dtype: int64
```

In [29]:
```python
data.shape
```

Out[29]:
```
(10000, 6)
```

In [30]:
```python
data.dropna(inplace=True)


missing_values = data.isnull().sum()
print(missing_values)
```

```
Hours Studied                       0
Previous Scores                     0
Sleep Hours                         0
Sample Question Papers Practiced    0
Performance Index                   0
Extracurricular Activities          0
dtype: int64
```

In [31]:
```python
data.shape
```

Out[31]:
```
(10000, 6)
```

Removing Duplicates

In [32]:
```
data.shape
```
Out[32]: (10000, 6)

In [33]:
```
data.drop_duplicates(inplace=True)
data.shape
```
Out[33]: (9873, 6)

3: Outlier Detection and Removal

In [34]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt



data.describe()
```
Out[34]:

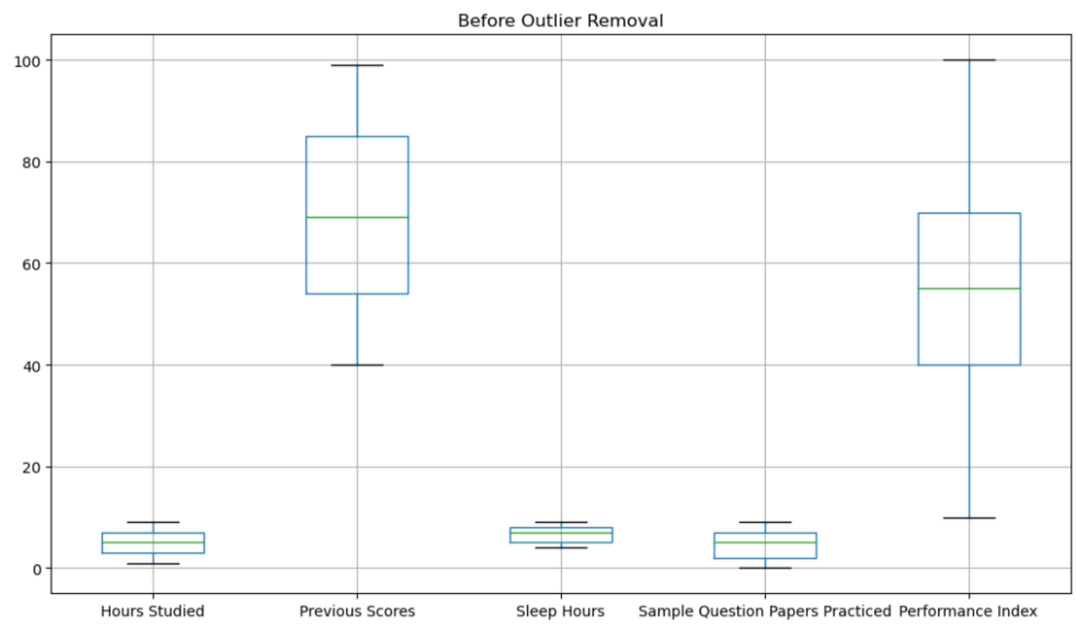
	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index
count	9873.000000	9873.000000	9873.000000	9873.000000	9873.000000
mean	4.992100	69.441102	6.531652	4.583004	55.216651
std	2.589081	17.325601	1.697683	2.867202	19.208570
min	1.000000	40.000000	4.000000	0.000000	10.000000
25%	3.000000	54.000000	5.000000	2.000000	40.000000
50%	5.000000	69.000000	7.000000	5.000000	55.000000
75%	7.000000	85.000000	8.000000	7.000000	70.000000
max	9.000000	99.000000	9.000000	9.000000	100.000000

In [35]:
```
0.25-1.5*0.5
```
Out[35]: -0.5

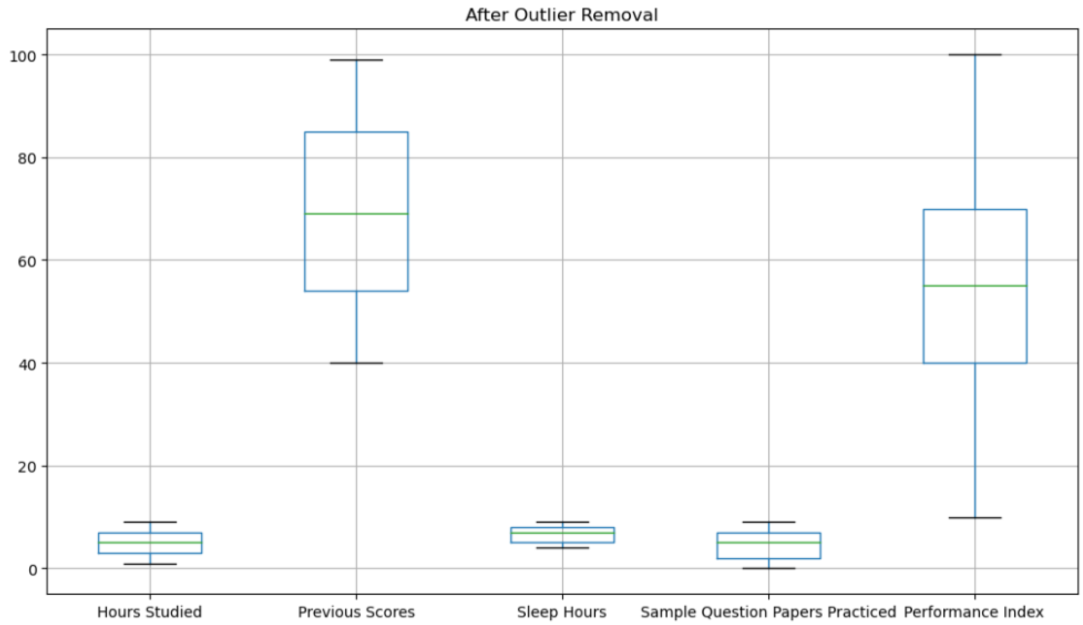
In [36]:
```
0.75 + 1.5 * 0.5
```
Out[36]: 1.5

In [37]:

```
numeric_cols = data.select_dtypes(include=[np.number])


Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

# Filter out outliers        # 0.25-1.5*0.5 = -0.5                      #  0.75 + 1.5 * 0.5 = 1.5

data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]


plt.figure(figsize=(20, 6))


plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")



plt.tight_layout()
plt.show()
```



In [38]:
```
plt.figure(figsize=(20, 6))


plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

plt.tight_layout()
plt.show()
```



In [39]:
```
data_cleaned.shape
```

Out[39]: (9873, 6)

In [40]:
```
data_cleaned.head()
```

Out[40]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities
0	7	99	9	1	91.0	Yes
1	4	82	4	2	65.0	No
2	8	51	7	2	45.0	Yes
3	5	52	5	2	36.0	Yes
4	7	75	8	5	66.0	No

4. Data Transformation

Key Differences

Range of Values:

Normalization: Values are scaled to a fixed range, typically [0, 1]. Standardization: Values are rescaled to have a mean of 0 and a standard deviation of 1. Effect on Distribution:

Normalization: Compresses or stretches the data to fit within the specified range, potentially altering the original distribution. Standardization: Preserves the shape of the original distribution but changes the scale. Use Cases:

Normalization: Suitable for distance-based algorithms, like k-nearest neighbors and neural networks. Standardization: Suitable for algorithms that assume a normal distribution, like linear regression and logistic regression.

Normalization/Standardization

- Normalization Definition: Normalization rescales the data to a fixed range, typically [0, 1] or [-1, 1].

In [41]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler




numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(9873, 6)

************************************************************
```

Out[41]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities
0	0.750	1.000000	1.0	0.111111	0.900000	Yes
1	0.375	0.711864	0.0	0.222222	0.611111	No
2	0.875	0.186441	0.6	0.222222	0.388889	Yes
3	0.500	0.203390	0.2	0.222222	0.288889	Yes
4	0.750	0.593220	0.8	0.555556	0.622222	No

Standardization

Definition: Standardization rescales the data so that it has a mean of 0 and a standard deviation of 1.

In [42]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler




numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])


scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)


scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)


scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(9873, 6)

************************************************************
```

Out[42]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities
0	0.775566	1.706168	1.454025	-1.249715	1.862979	Yes
1	-0.383205	0.724912	-1.491315	-0.900925	0.509348	No
2	1.161822	-1.064438	0.275889	-0.900925	-0.531907	Yes
3	0.003052	-1.006717	-0.902247	-0.900925	-1.000471	Yes
4	0.775566	0.320865	0.864957	0.145444	0.561411	No

5: One-Hot Encoding ¶

In [43]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler



data.head(2)
```

Out[43]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities
0	7	99	9	1	91.0	Yes
1	4	82	4	2	65.0	No

In [44]:
```
data["Hours Studied"].unique()
```

Out[44]:
```
array([7, 4, 8, 5, 3, 6, 2, 1, 9], dtype=int64)
```

In [45]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler




cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']


data1 = pd.get_dummies(cat_features)
data1
```

Out[45]:

	Extracurricular Activities
0	1

In [46]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 1 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Extracurricular Activities  1 non-null      uint8
dtypes: uint8(1)
memory usage: 133.0 bytes
```

In [47]:
```
cat_features
```

Out[47]:
```
['Extracurricular Activities']
```

In [48]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler




cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']


data1 = pd.get_dummies(data, columns=cat_features)

scaled_data = pd.concat([data, data1], axis=1)


print(scaled_data.shape)
print()
print('*' * 70)

scaled_data.head()
```

```
(9873, 13)

**********************************************************************
```

Out[48]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_No	Extracurricular Activities_Yes
0	7	99	9	1	91.0	Yes	7	99	9	1	91.0	0	1
1	4	82	4	2	65.0	No	4	82	4	2	65.0	1	0
2	8	51	7	2	45.0	Yes	8	51	7	2	45.0	0	1
3	5	52	5	2	36.0	Yes	5	52	5	2	36.0	0	1
4	7	75	8	5	66.0	No	7	75	8	5	66.0	1	0

In [49]:
```
data.columns
```

Out[49]:
```
Index(['Hours Studied', 'Previous Scores', 'Sleep Hours',
       'Sample Question Papers Practiced', 'Performance Index',
       'Extracurricular Activities'],
      dtype='object')
```

In [50]:
```
scaled_data.columns
```

Out[50]:
```
Index(['Hours Studied', 'Previous Scores', 'Sleep Hours',
       'Sample Question Papers Practiced', 'Performance Index',
       'Extracurricular Activities', 'Hours Studied', 'Previous Scores',
       'Sleep Hours', 'Sample Question Papers Practiced', 'Performance Index',
       'Extracurricular Activities_No', 'Extracurricular Activities_Yes'],
      dtype='object')
```

In [51]:
```
data1.head()
```
Out[51]:

	Hours Studied	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_No	Extracurricular Activities_Yes
0	7	99	9	1	91.0	0	1
1	4	82	4	2	65.0	1	0
2	8	51	7	2	45.0	0	1
3	5	52	5	2	36.0	0	1
4	7	75	8	5	66.0	1	0

In [ ]:

6: Data Reduction

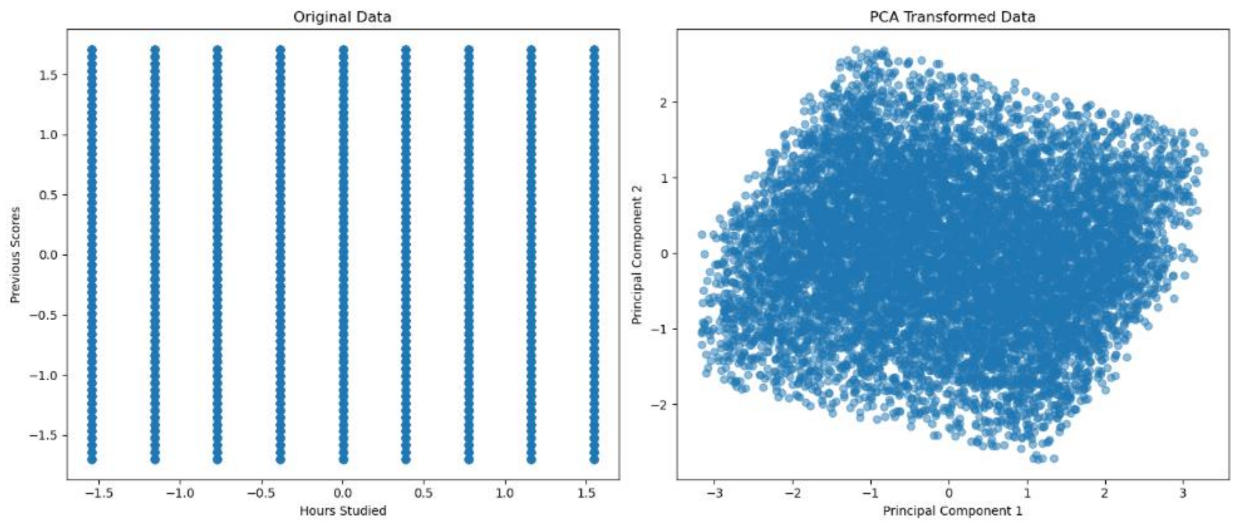
Dimensionality Reduction

PCA (Principal Component Analysis)

In [52]:
```
scaled_data.shape
```
Out[52]: (9873, 13)

In [53]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA




cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']


numeric_means = data[numeric_features].mean()
data[numeric_features] = data[numeric_features].fillna(numeric_means)


data = pd.get_dummies(data, columns=cat_features)


scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features])


pca_2 = PCA(n_components=2)
data_pca_2 = pca_2.fit_transform(data)


plt.figure(figsize=(14, 6))
```

```python
plt.subplot(1, 2, 1)
plt.scatter(data[numeric_features[0]], data[numeric_features[1]], alpha=0.5)
plt.title('Original Data')
plt.xlabel(numeric_features[0])
plt.ylabel(numeric_features[1])


plt.subplot(1, 2, 2)
plt.scatter(data_pca_2[:, 0], data_pca_2[:, 1], alpha=0.5)
plt.title('PCA Transformed Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.tight_layout()
plt.show()
```




```python
In [54]: type(data_pca_2)
Out[54]: numpy.ndarray

In [55]: data_pca_2.ndim
Out[55]: 2

In [56]: data_pca_2.shape
Out[56]: (9873, 2)
```

7: Handling Imbalanced Data

- Resampling Techniques
- Oversampling

In [81]:
```
import pandas as pd
import numpy as np
import imblearn
import category_encoders
import packaging.version
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt


data = pd.read_csv('Student_Performance.csv')


data.fillna(data.mean(numeric_only=True), inplace=True)


target = data['Hours Studied']
data = data.drop(columns=['Hours Studied'])

cat_features = [col for col in data.columns if data[col].dtype == 'O']
numeric_features = [col for col in data.columns if data[col].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)


scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features])


if target.dtype == 'O':
    le = LabelEncoder()
    target = le.fit_transform(target)
elif target.dtype == 'float':
    target = (target > 0.5).astype(int)
```

```
print("Before SMOTE:", data.shape, target.shape)


smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(data, target)


resampled_df = pd.concat([
    pd.DataFrame(X_resampled, columns=data.columns),
    pd.DataFrame(y_resampled, columns=['Hours Studied'])
], axis=1)

print("After SMOTE:", resampled_df.shape)
resampled_df.head()
```

```
Before SMOTE: (10000, 6) (10000,)
After SMOTE: (10368, 7)
```

	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_No	Extracurricular Activities_Yes	Hours Studied
0	1.704176	1.456205	-1.249754	1.862167	0	1	7
1	0.723913	-1.492294	-0.900982	0.508818	1	0	4
2	-1.063626	0.276805	-0.900982	-0.532220	0	1	8
3	-1.005963	-0.902594	-0.900982	-1.000687	0	1	5
4	0.320275	0.866505	0.145333	0.560870	1	0	7

In [82]:
```
resampled_df['Hours Studied'].value_counts(True)
```

Out[82]:
```
7    0.111111
4    0.111111
8    0.111111
5    0.111111
3    0.111111
6    0.111111
2    0.111111
1    0.111111
9    0.111111
Name: Hours Studied, dtype: float64
```

In [83]:
```
resampled_df.shape
```

Out[83]:
```
(10368, 7)
```

In [84]:
```
print(resampled_df.columns)
print(list(resampled_df.columns))
```

```
Index(['Previous Scores', 'Sleep Hours', 'Sample Question Papers Practiced',
       'Performance Index', 'Extracurricular Activities_No',
       'Extracurricular Activities_Yes', 'Hours Studied'],
      dtype='object')
['Previous Scores', 'Sleep Hours', 'Sample Question Papers Practiced', 'Performance Index', 'Extracurricular Activities_No', 'E
xtracurricular Activities_Yes', 'Hours Studied']
```

In [85]:
```
print(data.columns)
print(list(data.columns))
```

```
Index(['Previous Scores', 'Sleep Hours', 'Sample Question Papers Practiced',
       'Performance Index', 'Extracurricular Activities_No',
       'Extracurricular Activities_Yes'],
      dtype='object')
['Previous Scores', 'Sleep Hours', 'Sample Question Papers Practiced', 'Performance Index', 'Extracurricular Activities_No', 'E
xtracurricular Activities_Yes']
```

Undersampling

In [86]:

```
import pandas as pd
import numpy as np
import imblearn
import category_encoders
import packaging.version
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
import matplotlib.pyplot as plt


data = pd.read_csv('Student_Performance.csv')


data.fillna(data.mean(numeric_only=True), inplace=True)


target = data['Hours Studied']
data = data.drop(columns=['Hours Studied'])

cat_features = [col for col in data.columns if data[col].dtype == 'O']
numeric_features = [col for col in data.columns if data[col].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)


scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features])


if target.dtype == 'O':
    le = LabelEncoder()
    target = le.fit_transform(target)
elif target.dtype == 'float':
    target = (target > 0.5).astype(int)


print("Before undersampling:", data.shape, target.shape)


rus = RandomUnderSampler(random_state=42)
```

```
rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(data, target)


resampled_df = pd.concat([
    pd.DataFrame(X_resampled, columns=data.columns),
    pd.DataFrame(y_resampled, columns=['Hours Studied'])
], axis=1)

print("After undersampling:", resampled_df.shape)
resampled_df.head()
```

```
Before undersampling: (10000, 6) (10000,)
After undersampling: (9765, 7)
```

Out[86]:

	Previous Scores	Sleep Hours	Sample Question Papers Practiced	Performance Index	Extracurricular Activities_No	Extracurricular Activities_Yes	Hours Studied
1353	-0.083363	-0.312895	1.191649	-0.688376	1	0	1
6622	0.781575	0.276805	-0.900982	-0.063753	0	1	1
5076	1.588851	1.456205	1.191649	0.769077	1	0	1
1998	0.896900	-1.492294	0.494105	0.248558	1	0	1
6245	1.531188	0.866505	-0.552210	0.769077	1	0	1

In [88]:
```
resampled_df['Hours Studied'].value_counts()
```

Out[88]:
```
1    1085
2    1085
3    1085
4    1085
5    1085
6    1085
7    1085
8    1085
9    1085
Name: Hours Studied, dtype: int64
```

In [89]:
```
resampled_df.shape
```

Out[89]:
```
(9765, 7)
```

Target Encoder

In [90]:
```
import pandas as pd
from category_encoders import TargetEncoder

# Example dataset
data = {'animal': ['cat', 'dog', 'mouse', 'dog', 'cat'], 'target': [1, 0, 1, 0, 1]}
df = pd.DataFrame(data)

target_encoder = TargetEncoder(cols=['animal'])
target_encoded = target_encoder.fit_transform(df['animal'], df['target'])
print(target_encoded)
```

```
     animal
0  0.656740
1  0.514889
2  0.652043
3  0.514889
4  0.656740
```

8: Splitting Data

In [93]:
```
from sklearn.model_selection import train_test_split


X = resampled_df.drop('Hours Studied', axis=1)
y = resampled_df['Hours Studied']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [94]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[94]:
```
((6835, 6), (2930, 6), (6835,), (2930,))
```

9.Regression

In [96]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd

# Assuming 'data' is your fully preprocessed DataFrame
X = resampled_df.drop('Performance Index', axis=1)
y = resampled_df['Performance Index']

# Train/test split


# Initialize and train model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on test set
y_pred = lr_model.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("📊 Linear Regression Evaluation:")
print(f"Mean Squared Error: {mse:.2f}")
print(f"R² Score: {r2:.2f}")

# Show coefficients
coeff_df = pd.DataFrame({'Feature': X.columns, 'Coefficient': lr_model.coef_})
print("\n🔍 Feature Coefficients:")
print(coeff_df)
```

```
📊 Linear Regression Evaluation:
Mean Squared Error: 0.48
R² Score: 0.93

🔍 Feature Coefficients:
                            Feature  Coefficient
0                   Previous Scores    -5.749869
1                       Sleep Hours    -0.267676
2  Sample Question Papers Practiced    -0.169628
3       Extracurricular Activities_No    6.259859
4      Extracurricular Activities_Yes    0.098548
5                     Hours Studied    -0.098548
```

Prediction

In [102]:
```python
# Correct feature and target separation
X = resampled_df.drop('Performance Index', axis=1)  # features
y = resampled_df['Performance Index']               # target
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, random_state=42)
from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
# This must match the feature columns used in X
new_data = pd.DataFrame({
    'Previous Scores': [75],
    'Sleep Hours': [6],
    'Sample Question Papers Practiced': [5],
    'Extracurricular Activities_No': [0],
    'Extracurricular Activities_Yes': [1],
    'Hours Studied': [3],
})

# Ensure same column order
new_data = new_data[X.columns]

# Predict
prediction = lr_model.predict(new_data)
print(f"🎯 Predicted Performance Index: {prediction[0]:.2f}")
```

```
🎯 Predicted Performance Index: 69.03
```

Conclusion:

In this project, a linear regression model was developed to predict a student's *Performance Index* based on several academic and behavioral features, including *Previous Scores*, *Sleep Hours*, *Hours Studied*, and participation in *Extracurricular Activities*. The dataset was preprocessed by handling missing values, encoding categorical variables, and removing outliers using the IQR method. After training and testing the model, it achieved a high R^2 score of 0.93, indicating that 93% of the variability in the Performance Index can be explained by the selected features. The model also showed a relatively low mean squared error, suggesting good prediction accuracy. Analysis of the feature coefficients revealed that *Previous Scores* and *Extracurricular Activities* had the most significant impact on performance, while *Hours Studied* and *Sleep Hours* had a comparatively smaller effect. Overall, the model demonstrates the usefulness of linear regression in educational performance prediction and highlights which factors most strongly influence student outcomes.

Classification

Title:

Model to find the Best seller Supplements based on various features.

Data Description:

Data was extracted from Statis of sales across different regions.

Prediction task is to determine the best seller product

Code and Output:

Problem Description: Model to find the Best seller Supplements based on various features.

Data Preprocessing Steps

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns

color = sns.color_palette()

import numpy as np
import pandas as pd
```

1: Reading Data

In [2]:
```
data = pd.read_csv('Supplement.csv')

data.head()
```

Out[2]:

	Date	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
0	2020-01-06	Whey Protein	Protein	143	31.98	4573.14	0.03	2	Canada	Walmart	0
1	2020-01-06	Vitamin C	Vitamin	139	42.51	5908.89	0.04	0	UK	Amazon	0
2	2020-01-06	Fish Oil	Omega	161	12.91	2078.51	0.25	0	Canada	Amazon	1
3	2020-01-06	Multivitamin	Vitamin	140	16.07	2249.80	0.08	0	Canada	Walmart	0
4	2020-01-06	Pre-Workout	Performance	157	35.47	5568.79	0.25	3	Canada	iHerb	0

In [4]:
```
data.head(10)
```
Out[4]:

	Date	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
0	2020-01-06	Whey Protein	Protein	143	31.98	4573.14	0.03	2	Canada	Walmart	0
1	2020-01-06	Vitamin C	Vitamin	139	42.51	5908.89	0.04	0	UK	Amazon	0
2	2020-01-06	Fish Oil	Omega	161	12.91	2078.51	0.25	0	Canada	Amazon	1
3	2020-01-06	Multivitamin	Vitamin	140	16.07	2249.80	0.08	0	Canada	Walmart	0
4	2020-01-06	Pre-Workout	Performance	157	35.47	5568.79	0.25	3	Canada	iHerb	0
5	2020-01-06	BCAA	Amino Acid	154	41.19	6343.26	0.13	1	UK	Walmart	0
6	2020-01-06	Creatine	Performance	134	32.49	4353.66	0.05	1	UK	Walmart	0
7	2020-01-06	Zinc	Mineral	147	46.68	6861.96	0.19	0	Canada	Amazon	0
8	2020-01-06	Collagen Peptides	Protein	147	10.96	1611.12	0.06	2	USA	Amazon	0
9	2020-01-06	Magnesium	Mineral	134	20.76	2781.84	0.00	0	Canada	Amazon	0

In [5]:
```
data.tail()
```
Out[5]:

	Date	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
4379	2025-03-31	Melatonin	Sleep Aid	160	47.79	7646.40	0.21	1	USA	iHerb	1
4380	2025-03-31	Biotin	Vitamin	154	38.12	5870.48	0.22	1	UK	Walmart	0
4381	2025-03-31	Green Tea Extract	Fat Burner	139	20.40	2835.60	0.12	3	USA	iHerb	0
4382	2025-03-31	Iron Supplement	Mineral	154	18.31	2819.74	0.23	2	Canada	Amazon	0
4383	2025-03-31	Electrolyte Powder	Hydration	178	39.12	6963.36	0.23	0	UK	iHerb	1

In [6]:
```
data.shape
```
Out[6]:
```
(4384, 11)
```

2: Data Cleaning

Handling Missing Values

- Imputation: Filling missing values with mean.

In [12]:
```
data.isnull().sum()
```
Out[12]:
```
Date              0
Product Name      0
Category          0
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Location          0
Platform          0
Best Seller       0
dtype: int64
```

```
In [13]: import pandas as pd
         import numpy as np

         numeric_cols = data.select_dtypes(include=[np.number])
         non_numeric_cols = data.select_dtypes(exclude=[np.number])

         numeric_cols.fillna(numeric_cols.mean(), inplace=True)

         data = pd.concat([numeric_cols, non_numeric_cols], axis=1)

         missing_values = data.isnull().sum()
         print(missing_values)
```

```
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Best Seller       0
Date              0
Product Name      0
Category          0
Location          0
Platform          0
dtype: int64
```

```
In [14]: import pandas as pd
         import numpy as np

         numeric_cols = data.select_dtypes(include=[np.number])
         non_numeric_cols = data.select_dtypes(exclude=[np.number])

         numeric_cols.fillna(numeric_cols.mean(), inplace=True)

         for col in non_numeric_cols.columns:
             non_numeric_cols[col].fillna(non_numeric_cols[col].mode()[0], inplace=True)

         data = pd.concat([numeric_cols, non_numeric_cols], axis=1)

         missing_values = data.isnull().sum()
         print(missing_values)
```

```
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Best Seller       0
Date              0
Product Name      0
Category          0
Location          0
Platform          0
dtype: int64
```

```
In [15]: data.shape
Out[15]: (4384, 11)
```

Removal: Deleting rows with missing values.

In [16]:
```
data.isnull().sum()
```

Out[16]:
```
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Best Seller       0
Date              0
Product Name      0
Category          0
Location          0
Platform          0
dtype: int64
```

In [17]:
```
data.shape
```

Out[17]:
```
(4384, 11)
```

In [18]:
```
data.dropna(inplace=True)

missing_values = data.isnull().sum()
print(missing_values)
```

```
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Best Seller       0
Date              0
Product Name      0
Category          0
Location          0
Platform          0
dtype: int64
```

In [19]:
```
data.shape
```

Out[19]:
```
(4384, 11)
```

In [ ]:

3: Outlier Detection and Removal

In [21]:
```
data.describe()
```

Out[21]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller
count	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000
mean	150.200274	34.781229	5226.569446	0.124398	1.531478	0.275776
std	12.396099	14.198309	2192.491946	0.071792	1.258479	0.446955
min	103.000000	10.000000	1284.000000	0.000000	0.000000	0.000000
25%	142.000000	22.597500	3349.372500	0.060000	1.000000	0.000000
50%	150.000000	34.720000	5173.140000	0.120000	1.000000	0.000000
75%	158.000000	46.712500	7009.960000	0.190000	2.000000	1.000000
max	194.000000	59.970000	10761.850000	0.250000	8.000000	1.000000

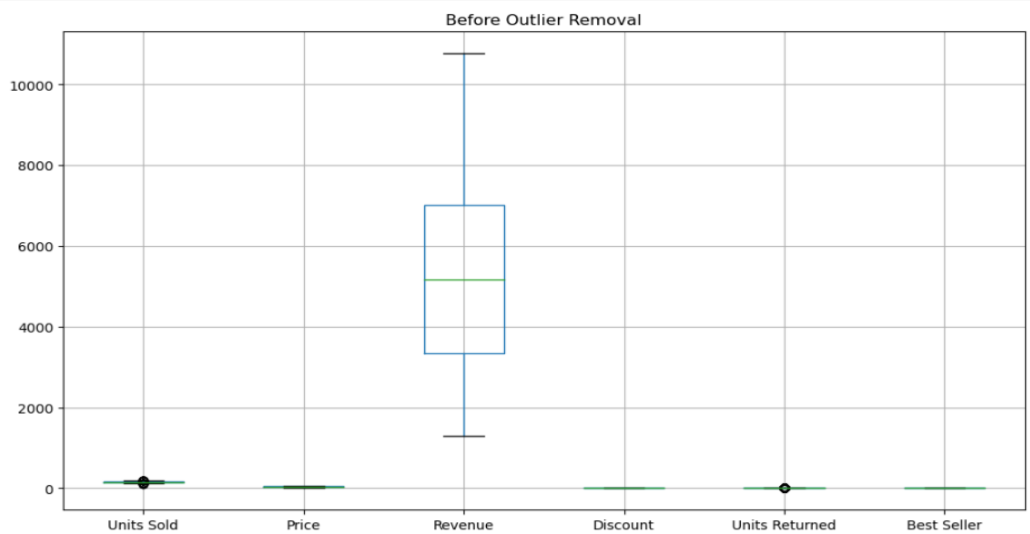
In [22]:
```
0.25-1.5*0.5
```

Out[22]: -0.5

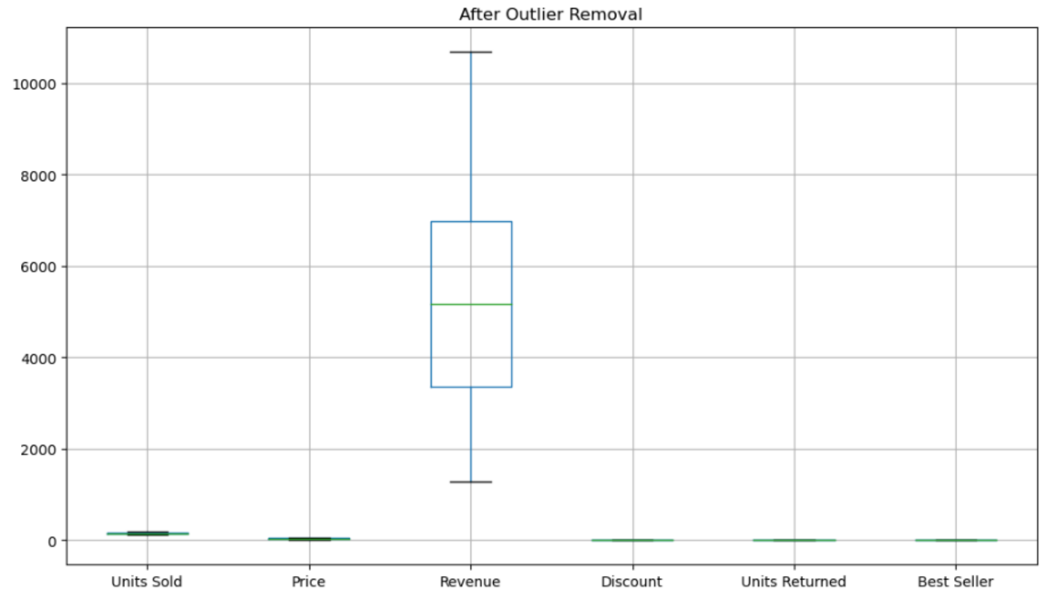
In [23]:
```
0.75 + 1.5 * 0.5
```

Out[23]: 1.5

In [24]:
```
numeric_cols = data.select_dtypes(include=[np.number])
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

data_cleaned = data[~((numeric_cols < (Q1 - 1.5 * IQR)) | (numeric_cols > (Q3 + 1.5 * IQR))).any(axis=1)]

plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)
numeric_cols.boxplot()
plt.title("Before Outlier Removal")

plt.tight_layout()
plt.show()
```



In [25]:
```
plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 2)
data_cleaned.select_dtypes(include=[np.number]).boxplot()
plt.title("After Outlier Removal")

plt.tight_layout()
plt.show()
```



In [26]:
```
data_cleaned.shape
```

Out[26]: (4033, 11)

In [27]:
```
data_cleaned.head()
```

Out[27]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date	Product Name	Category	Location	Platform
0	143	31.98	4573.14	0.03	2	0	2020-01-06	Whey Protein	Protein	Canada	Walmart
1	139	42.51	5908.89	0.04	0	0	2020-01-06	Vitamin C	Vitamin	UK	Amazon
2	161	12.91	2078.51	0.25	0	1	2020-01-06	Fish Oil	Omega	Canada	Amazon
3	140	16.07	2249.80	0.08	0	0	2020-01-06	Multivitamin	Vitamin	Canada	Walmart
4	157	35.47	5568.79	0.25	3	0	2020-01-06	Pre-Workout	Performance	Canada	iHerb

In [ ]:

4. Data Transformation

In [28]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler

numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = MinMaxScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(4384, 11)

************************************************************
```

Out[28]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date	Product Name	Category	Location	Platform
0	0.439560	0.439864	0.347034	0.12	0.250	0.0	2020-01-06	Whey Protein	Protein	Canada	Walmart
1	0.395604	0.650590	0.487968	0.16	0.000	0.0	2020-01-06	Vitamin C	Vitamin	UK	Amazon
2	0.637363	0.058235	0.083828	1.00	0.000	1.0	2020-01-06	Fish Oil	Omega	Canada	Amazon
3	0.406593	0.121473	0.101901	0.32	0.000	0.0	2020-01-06	Multivitamin	Vitamin	Canada	Walmart
4	0.593407	0.509706	0.452085	1.00	0.375	0.0	2020-01-06	Pre-Workout	Performance	Canada	iHerb

In [ ]:

Standardization

Definition: Standardization rescales the data so that it has a mean of 0 and a standard deviation of 1.

In [29]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

numeric_cols = data.select_dtypes(include=[np.number])
non_numeric_cols = data.select_dtypes(exclude=[np.number])

scaler = StandardScaler()
scaled_numeric_data = scaler.fit_transform(numeric_cols)

scaled_numeric_df = pd.DataFrame(scaled_numeric_data, columns=numeric_cols.columns)

scaled_data = pd.concat([scaled_numeric_df, non_numeric_cols.reset_index(drop=True)], axis=1)

print(scaled_data.shape)
print()
print('*' * 60)
scaled_data.head()
```

```
(4384, 11)

************************************************************
```

Out[29]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date	Product Name	Category	Location	Platform
0	-0.580916	-0.197316	-0.298064	-1.315034	0.372335	-0.617080	2020-01-06	Whey Protein	Protein	Canada	Walmart
1	-0.903635	0.544407	0.311243	-1.175727	-1.217067	-0.617080	2020-01-06	Vitamin C	Vitamin	UK	Amazon
2	0.871319	-1.540587	-1.436000	1.749735	-1.217067	1.620536	2020-01-06	Fish Oil	Omega	Canada	Amazon
3	-0.822955	-1.318000	-1.357865	-0.618496	-1.217067	-0.617080	2020-01-06	Multivitamin	Vitamin	Canada	Walmart
4	0.548600	0.048516	0.156105	1.749735	1.167036	-0.617080	2020-01-06	Pre-Workout	Performance	Canada	iHerb

5: One-Hot Encoding

In [30]:
```
data.head(2)
```

Out[30]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date	Product Name	Category	Location	Platform
0	143	31.98	4573.14	0.03	2	0	2020-01-06	Whey Protein	Protein	Canada	Walmart
1	139	42.51	5908.89	0.04	0	0	2020-01-06	Vitamin C	Vitamin	UK	Amazon

In [31]:
```
data["Best Seller"].unique()
```

Out[31]:
```
array([0, 1], dtype=int64)
```

In [ ]:

In [32]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']

data1 = pd.get_dummies(cat_features)
data1
```

Out[32]:

	Category	Date	Location	Platform	Product Name
0	0	1	0	0	0
1	0	0	0	0	1
2	1	0	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0

In [33]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Category      5 non-null      uint8
 1   Date          5 non-null      uint8
 2   Location      5 non-null      uint8
 3   Platform      5 non-null      uint8
 4   Product Name  5 non-null      uint8
dtypes: uint8(5)
memory usage: 157.0 bytes
```

In [34]:
```
cat_features
```

Out[34]:
```
['Date', 'Product Name', 'Category', 'Location', 'Platform']
```

In [35]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']

data1 = pd.get_dummies(data, columns=cat_features)

scaled_data = pd.concat([data, data1], axis=1)

print(scaled_data.shape)
print()
print('*' * 70)

scaled_data.head()
```

```
(4384, 323)

**********************************************************************
```

Out[35]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date	Product Name	Category	Location	...	Category_Performance	Category_Protein	Category_Sleep Aid	C
0	143	31.98	4573.14	0.03	2	0	2020-01-06	Whey Protein	Protein	Canada	...	0	1	0	
1	139	42.51	5908.89	0.04	0	0	2020-01-06	Vitamin C	Vitamin	UK	...	0	0	0	
2	161	12.91	2078.51	0.25	0	1	2020-01-06	Fish Oil	Omega	Canada	...	0	0	0	
3	140	16.07	2249.80	0.08	0	0	2020-01-06	Multivitamin	Vitamin	Canada	...	0	0	0	
4	157	35.47	5568.79	0.25	3	0	2020-01-06	Pre-Workout	Performance	Canada	...	1	0	0	

5 rows × 323 columns

In [36]:
```
data.columns
```

Out[36]:
```
Index(['Units Sold', 'Price', 'Revenue', 'Discount', 'Units Returned',
       'Best Seller', 'Date', 'Product Name', 'Category', 'Location',
       'Platform'],
      dtype='object')
```

In [37]:
```
scaled_data.columns
```

Out[37]:
```
Index(['Units Sold', 'Price', 'Revenue', 'Discount', 'Units Returned',
       'Best Seller', 'Date', 'Product Name', 'Category', 'Location',
       ...
       'Category_Performance', 'Category_Protein', 'Category_Sleep Aid',
       'Category_Vitamin', 'Location_Canada', 'Location_UK', 'Location_USA',
       'Platform_Amazon', 'Platform_Walmart', 'Platform_iHerb'],
      dtype='object', length=323)
```

In [38]:
```
data1.head()
```

Out[38]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller	Date_2020-01-06	Date_2020-01-13	Date_2020-01-20	Date_2020-01-27	...	Category_Performance	Category_Protein	Category_S
0	143	31.98	4573.14	0.03	2	0	1	0	0	0	...	0	1	
1	139	42.51	5908.89	0.04	0	0	1	0	0	0	...	0	0	
2	161	12.91	2078.51	0.25	0	1	1	0	0	0	...	0	0	
3	140	16.07	2249.80	0.08	0	0	1	0	0	0	...	0	0	
4	157	35.47	5568.79	0.25	3	0	1	0	0	0	...	1	0	

5 rows × 312 columns

In [ ]:

6: Data Reduction

Dimensionality Reduction

PCA (Principal Component Analysis)

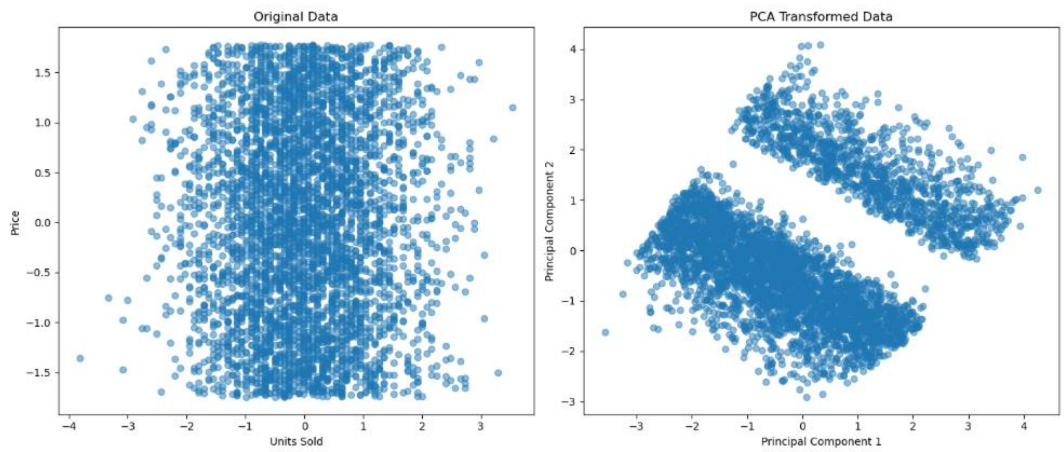
In [39]:
```
scaled_data.shape
```

Out[39]:
```
(4384, 323)
```

In [40]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

pca = PCA(n_components=4)
data_pca = pca.fit_transform(data)

print(data_pca.shape)
print(data_pca[:5])

plt.figure(figsize=(14, 6))
```

```python
plt.subplot(1, 2, 1)
plt.scatter(data[numeric_features[0]], data[numeric_features[1]], alpha=0.5)
plt.title('Original Data')
plt.xlabel(numeric_features[0])
plt.ylabel(numeric_features[1])

pca = PCA(n_components=4)
data_pca = pca.fit_transform(data)

plt.subplot(1, 2, 2)
plt.scatter(data_pca[:, 0], data_pca[:, 1], alpha=0.5)
plt.title('PCA Transformed Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.tight_layout()
plt.show()
```

```
(4384, 4)
[[-0.69517858 -0.42304467 -1.1550605   0.83877112]
 [-0.1448728  -1.43313913 -1.42185059 -0.54470601]
 [-0.97863628  2.32000232  1.24374271 -2.03899512]
 [-2.21551455 -0.02062641 -0.93827002 -0.86150106]
 [ 0.161729    0.06401863  2.01731706  0.62374047]]
```



In [41]:
```python
type(data_pca)
```
Out[41]: numpy.ndarray

In [42]:
```python
data_pca.ndim
```
Out[42]: 2

In [43]:
```python
data_pca.shape
```
Out[43]: (4384, 4)

7: Handling Imbalanced Data

- Resampling Techniques
- Oversampling

In [44]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

if data['Best Seller'].dtype != 'int64' and data['Best Seller'].dtype != 'bool':

    data['Best Seller'] = (data['Best Seller'] > 0.5).astype(int)

X = data.drop(columns=['Best Seller'])
y = data['Best Seller']

if y.dtype == 'O':
    le = LabelEncoder()
    y = le.fit_transform(y)

print(X.shape, y.shape)
```

```python
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['Best Seller'])], ax
data_resampled.head()
```

```
(4384, 311) (4384,)
```

Out[44]:

	Units Sold	Price	Revenue	Discount	Units Returned	Date_2020-01-06	Date_2020-01-13	Date_2020-01-20	Date_2020-01-27	Date_2020-02-03	...	Category_Protein	Category_Sleep Aid	Cat
0	-0.580916	-0.197316	-0.298064	-1.315034	0.372335	16.522712	-0.060523	-0.060523	-0.060523	-0.060523	...	2.645751	-0.258199	
1	-0.903635	0.544407	0.311243	-1.175727	-1.217067	16.522712	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199	
2	0.871319	-1.540587	-1.436000	1.749735	-1.217067	16.522712	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199	
3	-0.822955	-1.318000	-1.357865	-0.618496	-1.217067	16.522712	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199	
4	0.548600	0.048516	0.156105	1.749735	1.167036	16.522712	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199	

5 rows × 312 columns

In [45]:
```python
data_resampled['Best Seller'].value_counts(True)
```

Out[45]:
```
0    0.5
1    0.5
Name: Best Seller, dtype: float64
```

In [46]:
```python
data_resampled.shape
```

Out[46]:
```
(6350, 312)
```

Undersampling

In [47]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt

data.fillna(data.mean(numeric_only=True), inplace=True)

cat_features = [feature for feature in data.columns if data[feature].dtype == 'O']
numeric_features = [feature for feature in data.columns if data[feature].dtype != 'O']

data = pd.get_dummies(data, columns=cat_features)

scaler = StandardScaler()
data[numeric_features] = scaler.fit_transform(data[numeric_features].values)

if data['Best Seller'].dtype != 'int64' and data['Best Seller'].dtype != 'bool':

    data['Best Seller'] = (data['Best Seller'] > 0.5).astype(int)

X = data.drop(columns=['Best Seller'])
y = data['Best Seller']

if y.dtype == 'O':
    le = LabelEncoder()
    y = le.fit_transform(y)

print(X.shape, y.shape)
```

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler()
X_resampled, y_resampled = rus.fit_resample(X, y)

data_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns), pd.DataFrame(y_resampled, columns=['Best Seller'])], ax
data_resampled.head()
```

```
(4384, 311) (4384,)
```

Out[47]:

	Units Sold	Price	Revenue	Discount	Units Returned	Date_2020-01-06	Date_2020-01-13	Date_2020-01-20	Date_2020-01-27	Date_2020-02-03	...	Category_Protein	Category_Sleep Aid
1159	-0.984315	0.463402	0.219461	-0.618496	1.961737	-0.060523	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199
3236	-1.387714	0.583852	0.228871	0.495966	1.167036	-0.060523	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199
3792	0.306561	0.231658	0.290210	1.192504	0.372335	-0.060523	-0.060523	-0.060523	-0.060523	-0.060523	...	2.645751	-0.258199
4131	-0.419557	-0.910862	-0.938911	0.774581	-0.422366	-0.060523	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199
3485	-0.016158	1.419258	1.374360	-0.897111	-0.422366	-0.060523	-0.060523	-0.060523	-0.060523	-0.060523	...	-0.377964	-0.258199

5 rows × 312 columns

In [48]:
```
data_resampled['Best Seller'].value_counts()
```

Out[48]:
```
0    1209
1    1209
Name: Best Seller, dtype: int64
```

In [49]:
```
data_resampled.shape
```

Out[49]:
```
(2418, 312)
```

Target Encoder

In [50]:

```
import pandas as pd

target = 'Best Seller'
categorical_cols = data.select_dtypes(include=['object']).columns

for col in categorical_cols:
    target_mean = data.groupby(col)[target].mean()
    data[col + '_target_enc'] = data[col].map(target_mean)

data = data.drop(columns=categorical_cols)

print(data.head())
```

```
   Units Sold     Price   Revenue  Discount  Units Returned  Best Seller  \
0   -0.580916 -0.197316 -0.298064 -1.315034        0.372335            0
1   -0.903635  0.544407  0.311243 -1.175727       -1.217067            0
2    0.871319 -1.540587 -1.436000  1.749735       -1.217067            1
3   -0.822955 -1.318000 -1.357865 -0.618496       -1.217067            0
4    0.548600  0.048516  0.156105  1.749735        1.167036            0

   Date_2020-01-06  Date_2020-01-13  Date_2020-01-20  Date_2020-01-27  ...  \
0        16.522712        -0.060523        -0.060523        -0.060523  ...
1        16.522712        -0.060523        -0.060523        -0.060523  ...
2        16.522712        -0.060523        -0.060523        -0.060523  ...
3        16.522712        -0.060523        -0.060523        -0.060523  ...
4        16.522712        -0.060523        -0.060523        -0.060523  ...

   Category_Performance  Category_Protein  Category_Sleep Aid  \
0             -0.377964          2.645751           -0.258199
1             -0.377964         -0.377964           -0.258199
2             -0.377964         -0.377964           -0.258199
3             -0.377964         -0.377964           -0.258199
4              2.645751         -0.377964           -0.258199

   Category_Vitamin  Location_Canada  Location_UK  Location_USA  \
0         -0.480384         1.381699    -0.712072     -0.685678
1          2.081666        -0.723747     1.404352     -0.685678
2         -0.480384         1.381699    -0.712072     -0.685678
3          2.081666         1.381699    -0.712072     -0.685678
4         -0.480384         1.381699    -0.712072     -0.685678

   Platform_Amazon  Platform_Walmart  Platform_iHerb
0        -0.711345          1.450798       -0.720822
1         1.405788         -0.689276       -0.720822
2         1.405788         -0.689276       -0.720822
3        -0.711345          1.450798       -0.720822
4        -0.711345         -0.689276        1.387305

[5 rows x 312 columns]
```

8: Splitting Data

In [51]:
```
from sklearn.model_selection import train_test_split
import pandas as pd

print(data.columns)

X = data.drop('Best Seller', axis=1)
y = data['Best Seller']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
Index(['Units Sold', 'Price', 'Revenue', 'Discount', 'Units Returned',
       'Best Seller', 'Date_2020-01-06', 'Date_2020-01-13', 'Date_2020-01-20',
       'Date_2020-01-27',
       ...
       'Category_Performance', 'Category_Protein', 'Category_Sleep Aid',
       'Category_Vitamin', 'Location_Canada', 'Location_UK', 'Location_USA',
       'Platform_Amazon', 'Platform_Walmart', 'Platform_iHerb'],
      dtype='object', length=312)
```

In [52]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[52]:
```
((3068, 311), (1316, 311), (3068,), (1316,))
```

Classification

Loading Libraries

In [53]:
```
# **DATA PROCESSING**

import pandas as pd # Data Processing
import numpy as np # Array Processing
import os # Data Importing

# **DATA ANALYSIS**

import matplotlib.pyplot as plt # Plots
import seaborn as sns # Graphs

# **PRE PROCESSING**

from sklearn.preprocessing import FunctionTransformer  # Transforming of Data
from sklearn.preprocessing import OneHotEncoder # Data Encoding
from sklearn.preprocessing import StandardScaler # Data Scaling
from imblearn.over_sampling import RandomOverSampler # Data OverSampling
from sklearn.decomposition import PCA # Principal Component Analysis

# **MODELS**

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier

# **NERURAL NETWORKS**

import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# **METRICS**

from sklearn.metrics import accuracy_score # Model Classification Report
```

Reading Data

In [58]:
```
import pandas as pd
import numpy as np
import os

Supplement = pd.read_csv("Supplement.csv")
```

In [60]:
```
Supplement.head()
```

Out[60]:

	Date	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
0	2020-01-06	Whey Protein	Protein	143	31.98	4573.14	0.03	2	Canada	Walmart	0
1	2020-01-06	Vitamin C	Vitamin	139	42.51	5908.89	0.04	0	UK	Amazon	0
2	2020-01-06	Fish Oil	Omega	161	12.91	2078.51	0.25	0	Canada	Amazon	1
3	2020-01-06	Multivitamin	Vitamin	140	16.07	2249.80	0.08	0	Canada	Walmart	0
4	2020-01-06	Pre-Workout	Performance	157	35.47	5568.79	0.25	3	Canada	iHerb	0

Exploring Data

In [18]:
```
Supplement.shape
```

Out[18]:
```
(32561, 15)
```

In [19]:
```
Supplement.ndim
```

Out[19]:
```
2
```

In [64]:
```
Supplement["Best Seller"].value_counts().rename("count"),
Supplement["Best Seller"].value_counts(True).rename('%').mul(100)
```

Out[64]:
```
0    72.422445
1    27.577555
Name: %, dtype: float64
```

In [66]:
```
Supplement["Best Seller"].value_counts()
```

Out[66]:
```
0    3175
1    1209
Name: Best Seller, dtype: int64
```

In [67]:
```
sns.countplot(data=stroke_data , x='Best Seller')
plt.title('Price')
```

In [69]:
```
Supplement.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4384 entries, 0 to 4383
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Date            4384 non-null   object 
 1   Product Name    4384 non-null   object 
 2   Category        4384 non-null   object 
 3   Units Sold      4384 non-null   int64  
 4   Price           4384 non-null   float64
 5   Revenue         4384 non-null   float64
 6   Discount        4384 non-null   float64
 7   Units Returned  4384 non-null   int64  
 8   Location        4384 non-null   object 
 9   Platform        4384 non-null   object 
 10  Best Seller     4384 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 376.9+ KB
```

In [70]:
```
Supplement.describe()
```

Out[70]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller
count	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000	4384.000000
mean	150.200274	34.781229	5226.569446	0.124398	1.531478	0.275776
std	12.396099	14.198309	2192.491946	0.071792	1.258479	0.446955
min	103.000000	10.000000	1284.000000	0.000000	0.000000	0.000000
25%	142.000000	22.597500	3349.372500	0.060000	1.000000	0.000000
50%	150.000000	34.720000	5173.140000	0.120000	1.000000	0.000000
75%	158.000000	46.712500	7009.960000	0.190000	2.000000	1.000000
max	194.000000	59.970000	10761.850000	0.250000	8.000000	1.000000

Features name

In [71]:
```
Supplement.columns
```

Out[71]:
```
Index(['Date', 'Product Name', 'Category', 'Units Sold', 'Price', 'Revenue',
       'Discount', 'Units Returned', 'Location', 'Platform', 'Best Seller'],
      dtype='object')
```

In [72]:
```
Supplement["Best Seller"].value_counts()
```

Out[72]:
```
0    3175
1    1209
Name: Best Seller, dtype: int64
```

Missing Values

In [73]:
```
print('Missing data sum :')
print(Supplement.isnull().sum())

print('\nMissing data percentage (%):')
print(Supplement.isnull().sum()/Supplement.count()*100)
```

```
Missing data sum :
Date              0
Product Name      0
Category          0
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Location          0
Platform          0
Best Seller       0
dtype: int64

Missing data percentage (%):
Date              0.0
Product Name      0.0
Category          0.0
Units Sold        0.0
Price             0.0
Revenue           0.0
Discount          0.0
Units Returned    0.0
Location          0.0
Platform          0.0
Best Seller       0.0
dtype: float64
```

Seperate Categorical and Numerical Features

In [74]:
```
cat_features = [feature for feature in Supplement.columns if Supplement[feature].dtypes == 'O']
print('Number of categorical variables: ', len(cat_features))
print('*'*80)
print('Categorical variables column name:',cat_features)
```

```
Number of categorical variables:  5
********************************************************************************
Categorical variables column name: ['Date', 'Product Name', 'Category', 'Location', 'Platform']
```

In [75]:
```
numerical_features = [feature for feature in Supplement.columns if Supplement[feature].dtypes != 'O']
print('Number of numerical variables: ', len(numerical_features))
print('*'*80)
print('Numerical Variables Column: ',numerical_features)
```

```
Number of numerical variables:  6
********************************************************************************
Numerical Variables Column:  ['Units Sold', 'Price', 'Revenue', 'Discount', 'Units Returned', 'Best Seller']
```

Checking Duplicating Values

```
In [30]: Supplement.duplicated().sum()
Out[30]: 24
```

```
In [76]: Supplement['Price'].unique()
Out[76]: array([31.98, 42.51, 12.91, ..., 28.45, 47.79, 39.12])
```

```
In [77]: Supplement['Price'].nunique()
Out[77]: 2919
```

```
In [78]: Supplement['Price'].sample(10)
Out[78]: 3788    45.82
         3128    34.48
         2974    26.05
         2383    41.41
         2495    20.97
         1357    51.46
         717     13.76
         326     51.87
         2793    26.79
         456     35.72
         Name: Price, dtype: float64
```

```
In [79]: Supplement['Revenue'].unique()
Out[79]: array([4573.14, 5908.89, 2078.51, ..., 2835.6 , 2819.74, 6963.36])
```

```
In [81]: Supplement['Discount'].unique()
Out[81]: array([0.03, 0.04, 0.25, 0.08, 0.13, 0.05, 0.19, 0.06, 0.  , 0.14, 0.1 ,
                0.22, 0.02, 0.24, 0.17, 0.2 , 0.16, 0.21, 0.07, 0.18, 0.09, 0.01,
                0.15, 0.12, 0.11, 0.23])
```

```
In [82]: Supplement['Units Sold'].unique()
Out[82]: array([143, 139, 161, 140, 157, 154, 134, 147, 181, 164, 159, 149, 150,
                128, 145, 137, 141, 160, 148, 133, 174, 163, 151, 131, 152, 126,
                156, 155, 162, 158, 136, 180, 135, 165, 129, 138, 132, 184, 153,
                168, 144, 146, 142, 117, 123, 175, 170, 169, 178, 171, 116, 127,
                185, 176, 188, 130, 124, 166, 173, 167, 125, 183, 119, 120, 177,
                118, 182, 122, 194, 172, 186, 179, 190, 113, 103, 187, 121, 114,
                112, 191, 109], dtype=int64)
```

```
In [83]: Supplement['Date'].unique()
Out[83]: array(['2020-01-06', '2020-01-13', '2020-01-20', '2020-01-27',
                '2020-02-03', '2020-02-10', '2020-02-17', '2020-02-24',
                '2020-03-02', '2020-03-09', '2020-03-16', '2020-03-23',
                '2020-03-30', '2020-04-06', '2020-04-13', '2020-04-20',
                '2020-04-27', '2020-05-04', '2020-05-11', '2020-05-18',
                '2020-05-25', '2020-06-01', '2020-06-08', '2020-06-15',
                '2020-06-22', '2020-06-29', '2020-07-06', '2020-07-13',
                '2020-07-20', '2020-07-27', '2020-08-03', '2020-08-10',
                '2020-08-17', '2020-08-24', '2020-08-31', '2020-09-07',
                '2020-09-14', '2020-09-21', '2020-09-28', '2020-10-05',
                '2020-10-12', '2020-10-19', '2020-10-26', '2020-11-02',
                '2020-11-09', '2020-11-16', '2020-11-23', '2020-11-30',
                '2020-12-07', '2020-12-14', '2020-12-21', '2020-12-28',
                '2021-01-04', '2021-01-11', '2021-01-18', '2021-01-25',
                '2021-02-01', '2021-02-08', '2021-02-15', '2021-02-22',
                '2021-03-01', '2021-03-08', '2021-03-15', '2021-03-22',
                '2021-03-29', '2021-04-05', '2021-04-12', '2021-04-19',
                '2021-04-26', '2021-05-03', '2021-05-10', '2021-05-17',
                '2021-05-24', '2021-05-31', '2021-06-07', '2021-06-14',
                '2021-06-21', '2021-06-28', '2021-07-05', '2021-07-12',
                '2021-07-19', '2021-07-26', '2021-08-02', '2021-08-09',
                '2021-08-16', '2021-08-23', '2021-08-30', '2021-09-06',
                '2021-09-13', '2021-09-20', '2021-09-27', '2021-10-04',
                '2021-10-11', '2021-10-18', '2021-10-25', '2021-11-01',
                '2021-11-08', '2021-11-15', '2021-11-22', '2021-11-29',
                '2021-12-06', '2021-12-13', '2021-12-20', '2021-12-27',
                '2022-01-03', '2022-01-10', '2022-01-17', '2022-01-24',
                '2022-01-31', '2022-02-07', '2022-02-14', '2022-02-21',
                '2022-02-28', '2022-03-07', '2022-03-14', '2022-03-21',
                '2022-03-28', '2022-04-04', '2022-04-11', '2022-04-18',
                '2022-04-25', '2022-05-02', '2022-05-09', '2022-05-16',
                '2022-05-23', '2022-05-30', '2022-06-06', '2022-06-13',
                '2022-06-20', '2022-06-27', '2022-07-04', '2022-07-11',
                '2022-07-18', '2022-07-25', '2022-08-01', '2022-08-08',
                '2022-08-15', '2022-08-22', '2022-08-29', '2022-09-05',
                '2022-09-12', '2022-09-19', '2022-09-26', '2022-10-03',
                '2022-10-10', '2022-10-17', '2022-10-24', '2022-10-31',
                '2022-11-07', '2022-11-14', '2022-11-21', '2022-11-28',
                '2022-12-05', '2022-12-12', '2022-12-19', '2022-12-26',
                '2023-01-02', '2023-01-09', '2023-01-16', '2023-01-23',
                '2023-01-30', '2023-02-06', '2023-02-13', '2023-02-20',
                '2023-02-27', '2023-03-06', '2023-03-13', '2023-03-20',
                '2023-03-27', '2023-04-03', '2023-04-10', '2023-04-17',
```

```
       '2023-03-27', '2023-04-03', '2023-04-10', '2023-04-17',
       '2023-04-24', '2023-05-01', '2023-05-08', '2023-05-15',
       '2023-05-22', '2023-05-29', '2023-06-05', '2023-06-12',
       '2023-06-19', '2023-06-26', '2023-07-03', '2023-07-10',
       '2023-07-17', '2023-07-24', '2023-07-31', '2023-08-07',
       '2023-08-14', '2023-08-21', '2023-08-28', '2023-09-04',
       '2023-09-11', '2023-09-18', '2023-09-25', '2023-10-02',
       '2023-10-09', '2023-10-16', '2023-10-23', '2023-10-30',
       '2023-11-06', '2023-11-13', '2023-11-20', '2023-11-27',
       '2023-12-04', '2023-12-11', '2023-12-18', '2023-12-25',
       '2024-01-01', '2024-01-08', '2024-01-15', '2024-01-22',
       '2024-01-29', '2024-02-05', '2024-02-12', '2024-02-19',
       '2024-02-26', '2024-03-04', '2024-03-11', '2024-03-18',
       '2024-03-25', '2024-04-01', '2024-04-08', '2024-04-15',
       '2024-04-22', '2024-04-29', '2024-05-06', '2024-05-13',
       '2024-05-20', '2024-05-27', '2024-06-03', '2024-06-10',
       '2024-06-17', '2024-06-24', '2024-07-01', '2024-07-08',
       '2024-07-15', '2024-07-22', '2024-07-29', '2024-08-05',
       '2024-08-12', '2024-08-19', '2024-08-26', '2024-09-02',
       '2024-09-09', '2024-09-16', '2024-09-23', '2024-09-30',
       '2024-10-07', '2024-10-14', '2024-10-21', '2024-10-28',
       '2024-11-04', '2024-11-11', '2024-11-18', '2024-11-25',
       '2024-12-02', '2024-12-09', '2024-12-16', '2024-12-23',
       '2024-12-30', '2025-01-06', '2025-01-13', '2025-01-20',
       '2025-01-27', '2025-02-03', '2025-02-10', '2025-02-17',
       '2025-02-24', '2025-03-03', '2025-03-10', '2025-03-17',
       '2025-03-24', '2025-03-31'], dtype=object)
```

In [84]:
```
Supplement['Units Returned'].unique()
```

Out[84]:
```
array([2, 0, 3, 1, 5, 4, 6, 7, 8], dtype=int64)
```

In [86]:
```
Supplement['Best Seller'].unique()
```

Out[86]:
```
array([0, 1], dtype=int64)
```

In [87]:
```
Supplement.columns
```

Out[87]:
```
Index(['Date', 'Product Name', 'Category', 'Units Sold', 'Price', 'Revenue',
       'Discount', 'Units Returned', 'Location', 'Platform', 'Best Seller'],
      dtype='object')
```

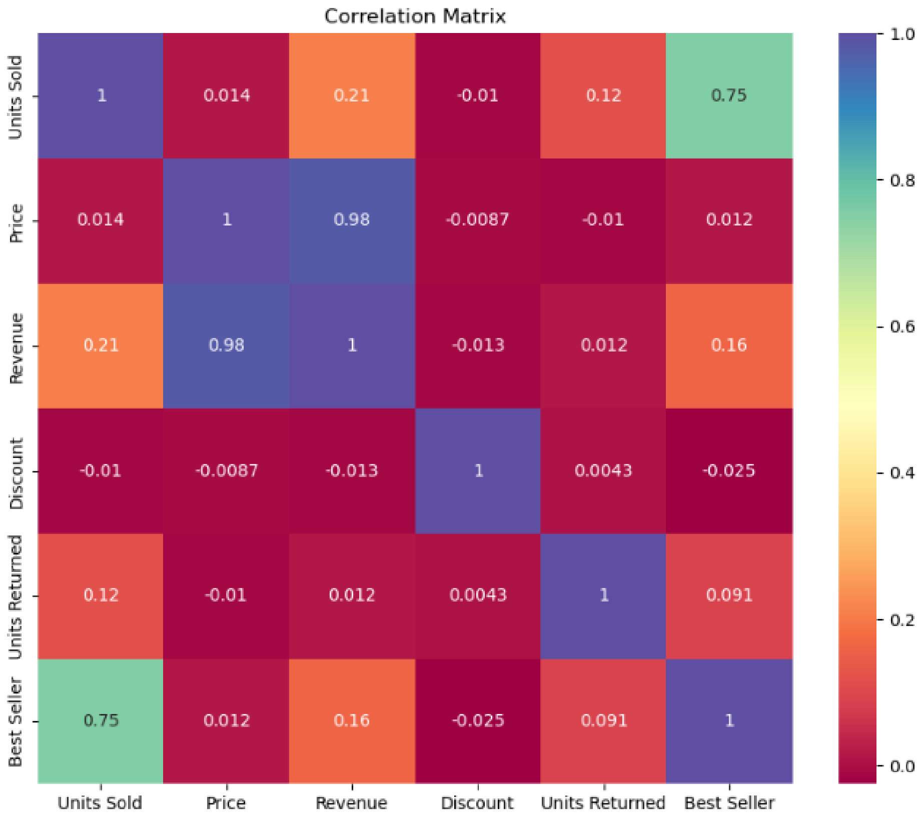
In [88]:
```
Supplement['Best Seller'].nunique()
```

Out[88]:
```
2
```

In [89]:
```
Supplement['Best Seller'].unique()
```

Out[89]:
```
array([0, 1], dtype=int64)
```

In [90]:
```
corr = Supplement.select_dtypes(include='number').corr()
Supplement.dtypes
plt.figure(figsize=(10, 8))
sns.heatmap(data=corr, annot=True, cmap='Spectral').set(title="Correlation Matrix")
plt.show()
```

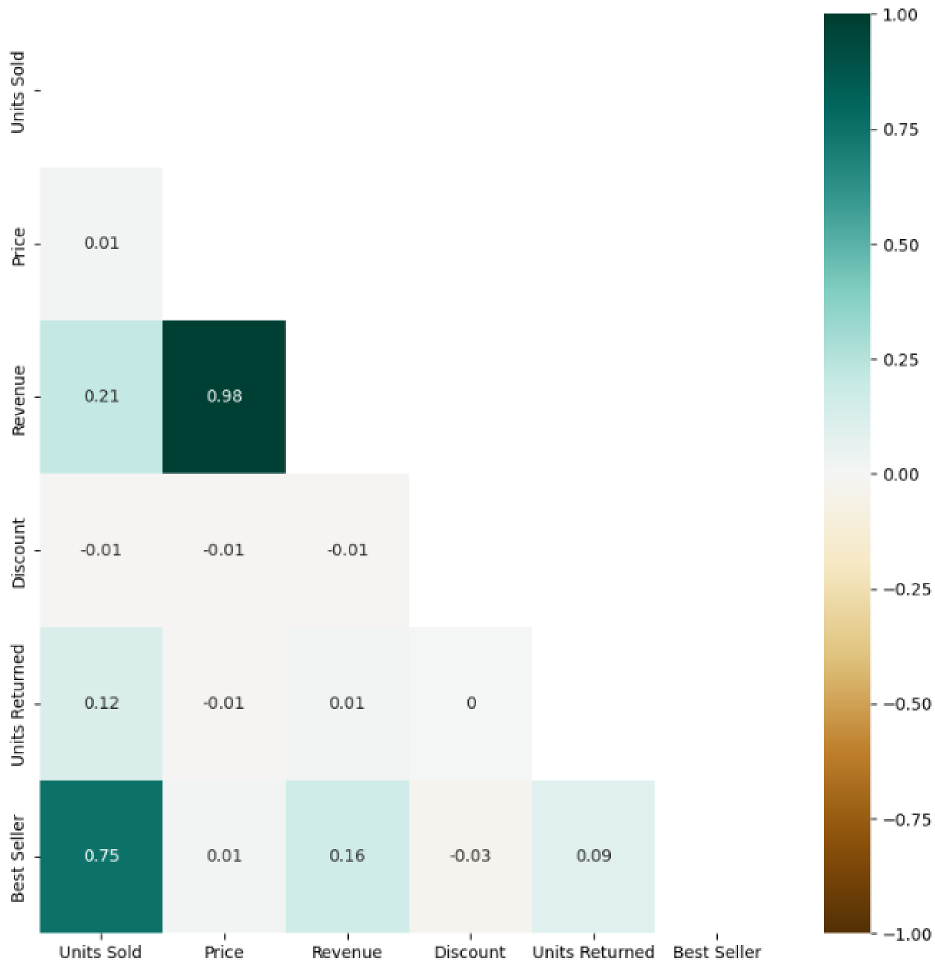


In [91]:
```
corr_matrix = Supplement.select_dtypes(include='number').corr().round(2)
corr_matrix
```

Out[91]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller
Units Sold	1.00	0.01	0.21	-0.01	0.12	0.75
Price	0.01	1.00	0.98	-0.01	-0.01	0.01
Revenue	0.21	0.98	1.00	-0.01	0.01	0.16
Discount	-0.01	-0.01	-0.01	1.00	0.00	-0.03
Units Returned	0.12	-0.01	0.01	0.00	1.00	0.09
Best Seller	0.75	0.01	0.16	-0.03	0.09	1.00

In [92]:
```
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

plt.figure(figsize=(10,10))
sns.heatmap(corr_matrix, center=0, vmin=-1, vmax=1, mask=mask, annot=True, cmap='BrBG')
plt.show()
```



In [93]:
```
cat_features = [feature for feature in Supplement.columns if Supplement[feature].dtypes == 'O']
print('Number of categorical variables: ', len(cat_features))
print('*'*80)
print('Categorical variables column name:',cat_features)
```

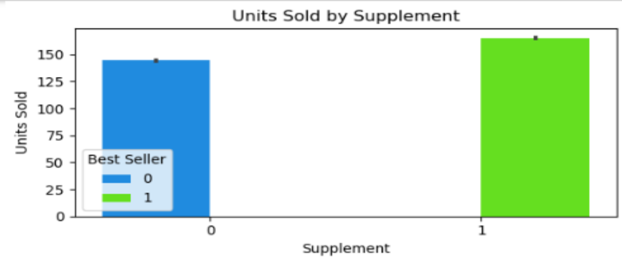
```
Number of categorical variables:  5
********************************************************************************
Categorical variables column name: ['Date', 'Product Name', 'Category', 'Location', 'Platform']
```

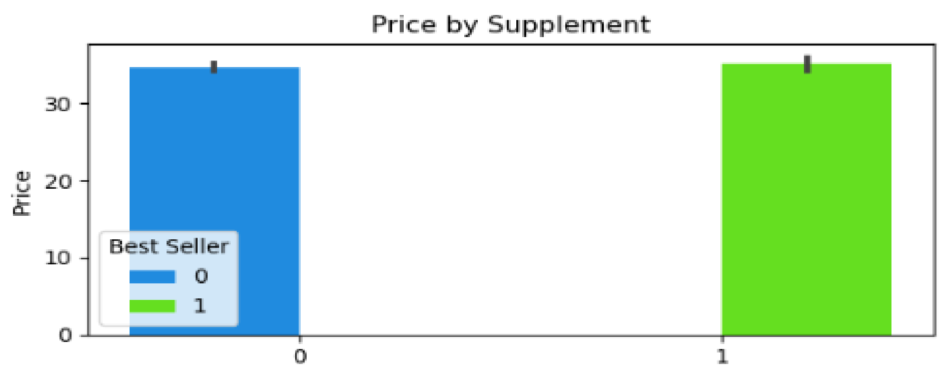
In [94]:
```
numerical_features = [feature for feature in Supplement.columns if Supplement[feature].dtypes != 'O']
print('Number of numerical variables: ', len(numerical_features))
print('*'*80)
print('Numerical Variables Column: ',numerical_features)
```

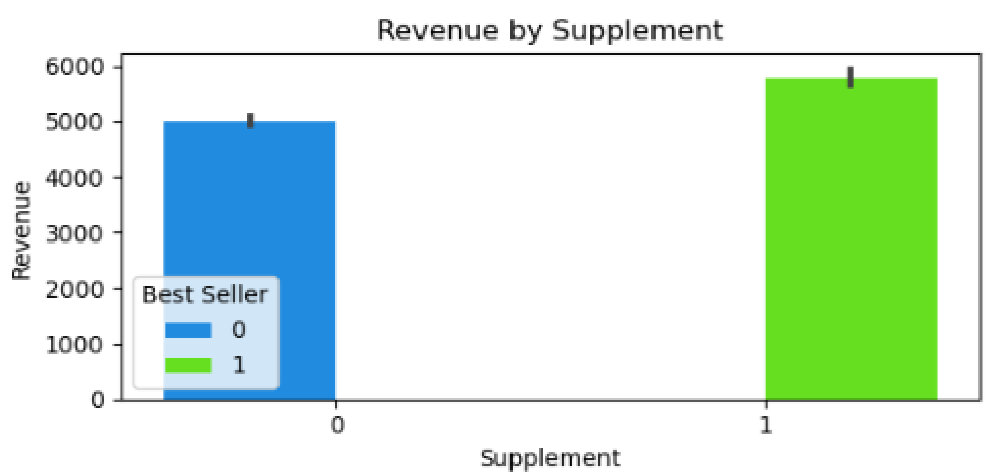
```
Number of numerical variables:  6
********************************************************************************
Numerical Variables Column:  ['Units Sold', 'Price', 'Revenue', 'Discount', 'Units Returned', 'Best Seller']
```

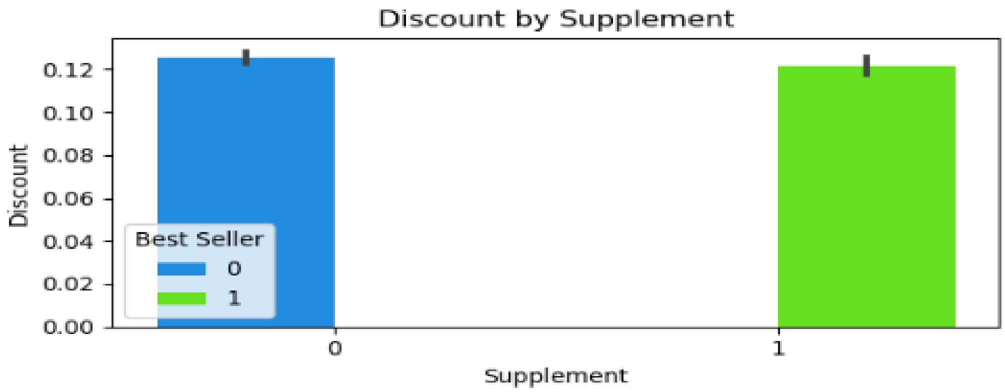
Visualizing Categorical Features

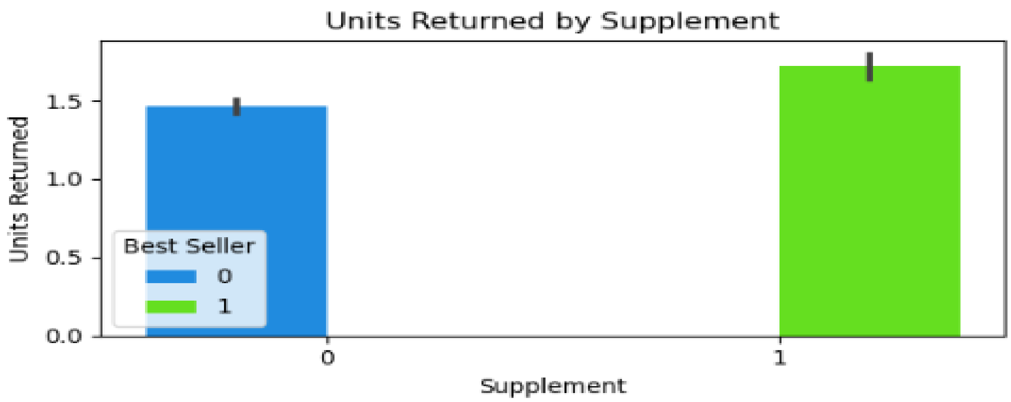
```
In [95]: for col in numerical_features:
             plt.figure(figsize=(6, 3), dpi=100)
             sns.barplot(data=Supplement, x='Best Seller', y=col, hue='Best Seller', palette='gist_rainbow_r')
             plt.title(f'{col} by Supplement')
             plt.xlabel('Supplement')
             plt.ylabel(col)
             plt.tight_layout()
             plt.show()
```

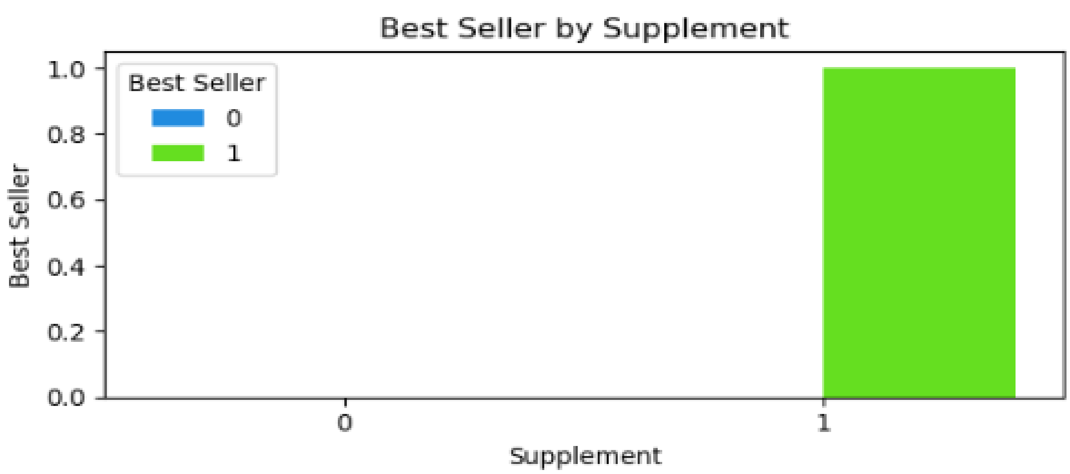






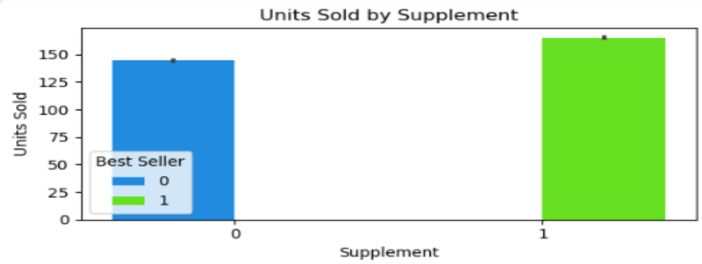

Discount by Supplement
Discount
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Best Seller
0
1
0
1
Supplement

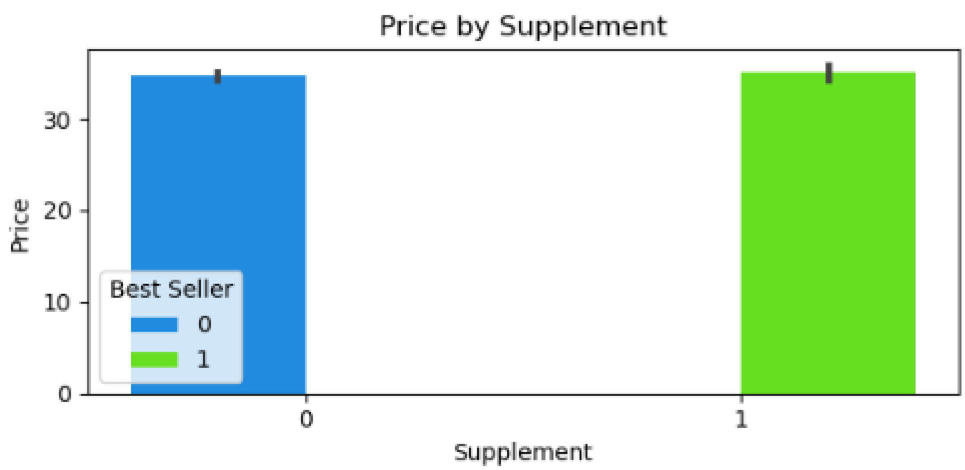

Units Returned by Supplement
Units Returned
1.5
1.0
0.5
0.0
Best Seller
0
1
0
1
Supplement

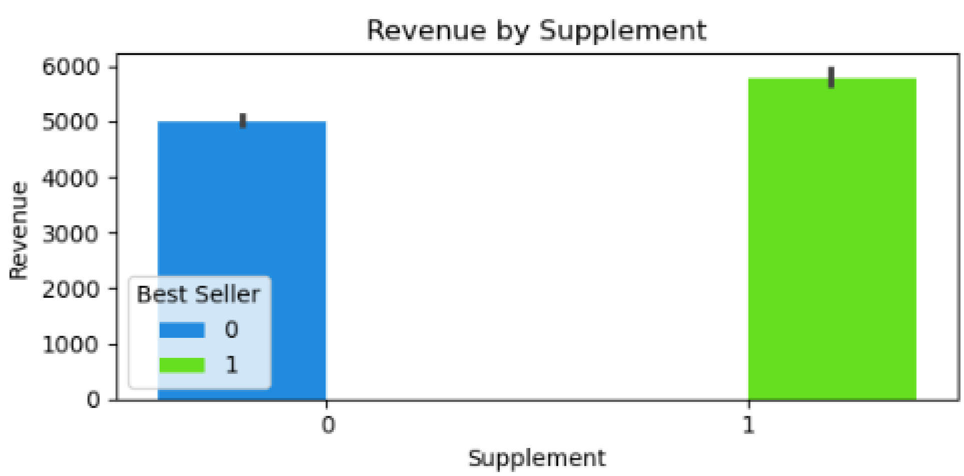

Best Seller by Supplement
Best Seller
1.0
0.8
0.6
0.4
0.2
0.0
Best Seller
0
1
0
1
Supplement

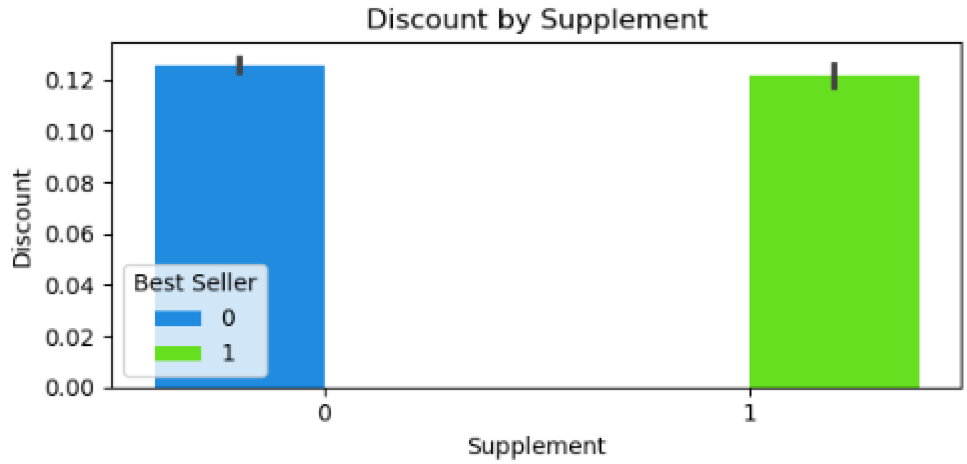
Barplot of numerical features:

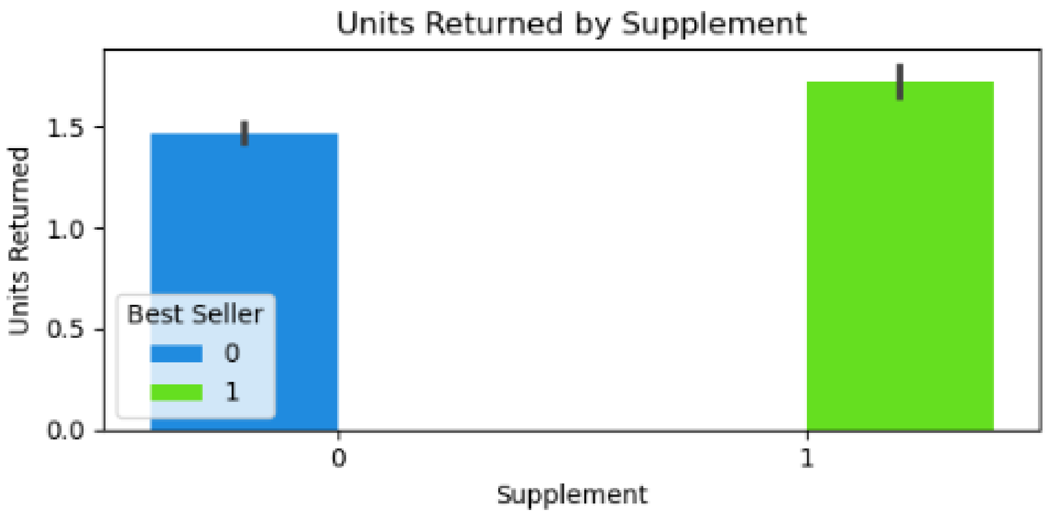
```
In [97]: ## Plotting barplots of numerical features grouped by salary
for col in numerical_features:
    plt.figure(figsize=(6, 3), dpi=100)
    sns.barplot(data=Supplement, x='Best Seller', y=col, hue='Best Seller', palette='gist_rainbow_r')
    plt.title(f'{col} by Supplement')
    plt.xlabel('Supplement')
    plt.ylabel(col)
    plt.tight_layout()
    plt.show()
```

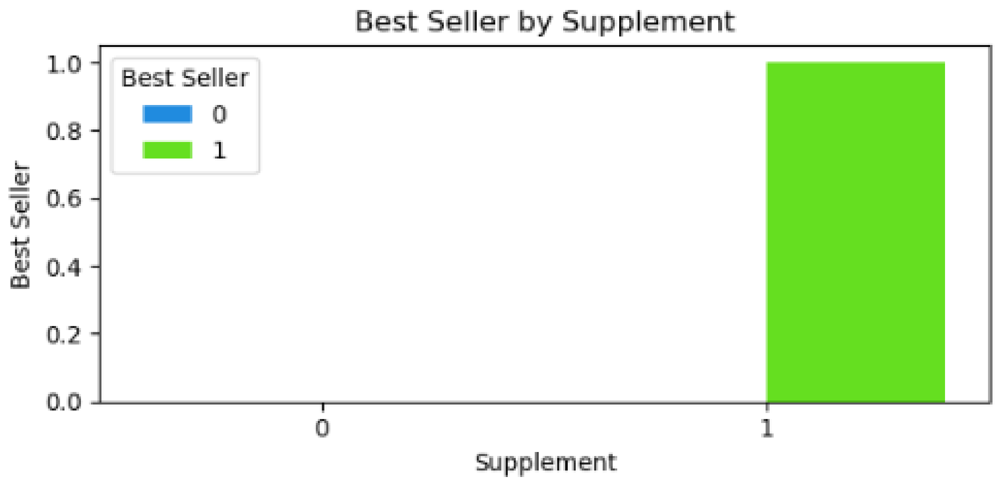







Discount by Supplement
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Discount
Best Seller
0
1
0
1
Supplement


Units Returned by Supplement
1.5
1.0
0.5
0.0
Units Returned
Best Seller
0
1
0
1
Supplement



Handling Missing Values

In [98]:
```
Supplement.head()
```

Out[98]:

	Date	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
0	2020-01-06	Whey Protein	Protein	143	31.98	4573.14	0.03	2	Canada	Walmart	0
1	2020-01-06	Vitamin C	Vitamin	139	42.51	5908.89	0.04	0	UK	Amazon	0
2	2020-01-06	Fish Oil	Omega	161	12.91	2078.51	0.25	0	Canada	Amazon	1
3	2020-01-06	Multivitamin	Vitamin	140	16.07	2249.80	0.08	0	Canada	Walmart	0
4	2020-01-06	Pre-Workout	Performance	157	35.47	5568.79	0.25	3	Canada	iHerb	0

In [99]:
```
Supplement.isnull().sum()
```

Out[99]:
```
Date              0
Product Name      0
Category          0
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Location          0
Platform          0
Best Seller       0
dtype: int64
```

In [100]:
```
Supplement["Price"] = Supplement["Price"].fillna(Supplement["Price"].mode()[0])
```

In [101]:
```
Supplement.isnull().sum()
```

Out[101]:
```
Date              0
Product Name      0
Category          0
Units Sold        0
Price             0
Revenue           0
Discount          0
Units Returned    0
Location          0
Platform          0
Best Seller       0
dtype: int64
```

Dropping

In [102]:
```
train  = Supplement.drop(['Date'],axis=1)
train
```

Out[102]:

	Product Name	Category	Units Sold	Price	Revenue	Discount	Units Returned	Location	Platform	Best Seller
0	Whey Protein	Protein	143	31.98	4573.14	0.03	2	Canada	Walmart	0
1	Vitamin C	Vitamin	139	42.51	5908.89	0.04	0	UK	Amazon	0
2	Fish Oil	Omega	161	12.91	2078.51	0.25	0	Canada	Amazon	1
3	Multivitamin	Vitamin	140	16.07	2249.80	0.08	0	Canada	Walmart	0
4	Pre-Workout	Performance	157	35.47	5568.79	0.25	3	Canada	iHerb	0
...	...	...	...	...	...	...	...	...	...	...
4379	Melatonin	Sleep Aid	160	47.79	7646.40	0.21	1	USA	iHerb	1
4380	Biotin	Vitamin	154	38.12	5870.48	0.22	1	UK	Walmart	0
4381	Green Tea Extract	Fat Burner	139	20.40	2835.60	0.12	3	USA	iHerb	0
4382	Iron Supplement	Mineral	154	18.31	2819.74	0.23	2	Canada	Amazon	0
4383	Electrolyte Powder	Hydration	178	39.12	6963.36	0.23	0	UK	iHerb	1

4384 rows × 10 columns

In [103]:
```
train.columns
```

Out[103]:
```
Index(['Product Name', 'Category', 'Units Sold', 'Price', 'Revenue',
       'Discount', 'Units Returned', 'Location', 'Platform', 'Best Seller'],
      dtype='object')
```

In [104]:
```
train.shape
```

Out[104]:
```
(4384, 10)
```

In [105]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4384 entries, 0 to 4383
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Product Name    4384 non-null   object 
 1   Category        4384 non-null   object 
 2   Units Sold      4384 non-null   int64  
 3   Price           4384 non-null   float64
 4   Revenue         4384 non-null   float64
 5   Discount        4384 non-null   float64
 6   Units Returned  4384 non-null   int64  
 7   Location        4384 non-null   object 
 8   Platform        4384 non-null   object 
 9   Best Seller     4384 non-null   int64  
dtypes: float64(3), int64(3), object(4)
memory usage: 342.6+ KB
```

In [106]:
```
train_data_cat = train.select_dtypes("object")
train_data_num = train.select_dtypes("number")
```

In [107]:
```
train_data_cat.head(3)
```

Out[107]:

	Product Name	Category	Location	Platform
0	Whey Protein	Protein	Canada	Walmart
1	Vitamin C	Vitamin	UK	Amazon
2	Fish Oil	Omega	Canada	Amazon

In [108]:
```
train_data_num.head(3)
```

Out[108]:

	Units Sold	Price	Revenue	Discount	Units Returned	Best Seller
0	143	31.98	4573.14	0.03	2	0
1	139	42.51	5908.89	0.04	0	0
2	161	12.91	2078.51	0.25	0	1

Converting categorical features into numerical

In [109]:
```
train_data_cata_encoded=pd.get_dummies(train_data_cat, columns=train_data_cat.columns.to_list())
train_data_cata_encoded.head()
```

Out[109]:

	Product Name_Ashwagandha	Product Name_BCAA	Product Name_Biotin	Product Name_Collagen Peptides	Product Name_Creatine	Product Name_Electrolyte Powder	Product Name_Fish Oil	Product Name_Green Tea Extract	Product Name_Iron Supplement	Product Name_Magnesium
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0

5 rows × 32 columns

In [110]:
```
data=pd.concat([train_data_cata_encoded,train_data_num],axis=1,join="outer")
data.head()
```

Out[110]:

	Product Name_Ashwagandha	Product Name_BCAA	Product Name_Biotin	Product Name_Collagen Peptides	Product Name_Creatine	Product Name_Electrolyte Powder	Product Name_Fish Oil	Product Name_Green Tea Extract	Product Name_Iron Supplement	Product Name_Magnesium
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0

5 rows × 38 columns

seperate dependant and independant feature

In [112]:
```
# Create a target column from one-hot encoded salary columns
data['Best Seller'] = data['Best Seller']  # This will be 1 if >50K, else 0

# Now separate features and target
y = data['Best Seller']
X = data.drop(['Best Seller'], axis=1)
```

In [113]:
```
print(y.shape)
print(X.shape)
```

```
(4384,)
(4384, 37)
```

scailing the data

In [114]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_scaled = sc.fit_transform(X)
```

In [115]:
```
X
```

Out[115]:

	Product Name_Ashwagandha	Product Name_BCAA	Product Name_Biotin	Product Name_Collagen Peptides	Product Name_Creatine	Product Name_Electrolyte Powder	Product Name_Fish Oil	Product Name_Green Tea Extract	Product Name_Iron Supplement	Product Name_Magnes
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	1	0	0	
3	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	
4379	0	0	0	0	0	0	0	0	0	
4380	0	0	1	0	0	0	0	0	0	
4381	0	0	0	0	0	0	0	1	0	
4382	0	0	0	0	0	0	0	0	1	
4383	0	0	0	0	0	1	0	0	0	

4384 rows × 37 columns

Splitting data into Training and Testing

In [117]:
```
#Importing our ML toolkit
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.svm import SVC
import pickle

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier, plot_importance
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold, learning_curve
```

Splitting the dataset

training data 70% testing data 30%

```
In [118]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=7)
```

```
In [119]: X_train.shape, X_test.shape
```

```
Out[119]: ((3068, 37), (1316, 37))
```

Building Classifiers

```
In [120]: accuracy = {}
```

Logistic Regression

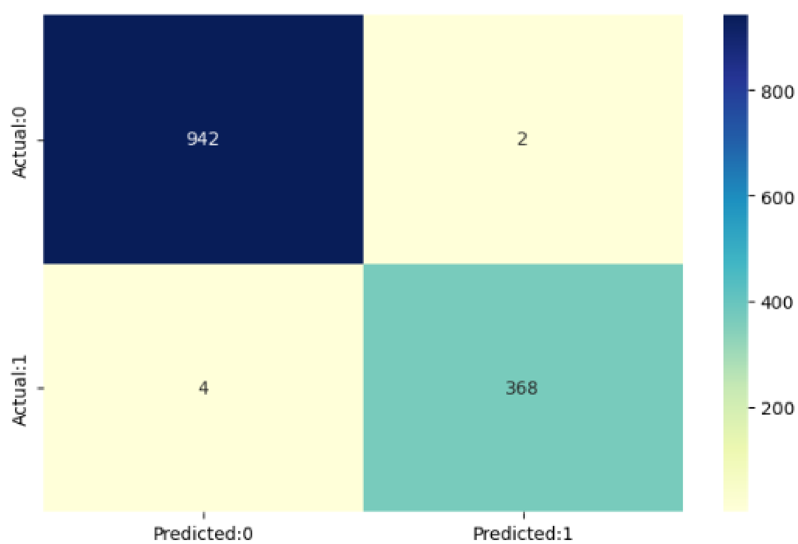
```
In [121]: scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# train the model
lr = LogisticRegression(max_iter=200)
lr.fit(X_train_scaled, y_train)
y_pred1 = lr.predict(X_test_scaled)

# evaluate
accuracy = {}
print(accuracy_score(y_test, y_pred1))
accuracy[str(lr)] = accuracy_score(y_test, y_pred1)*100
```

```
0.9954407294832827
```

Confusion Matrix

```
In [122]: from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred1)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

```
Out[122]: <Axes: >
```



Classification Report

In [123]:
```
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       944
           1       0.99      0.99      0.99       372

    accuracy                           1.00      1316
   macro avg       1.00      0.99      0.99      1316
weighted avg       1.00      1.00      1.00      1316
```

In [124]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred1, zero_division=1))  # sets precision/recall to 1 instead of 0
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       944
           1       0.99      0.99      0.99       372

    accuracy                           1.00      1316
   macro avg       1.00      0.99      0.99      1316
weighted avg       1.00      1.00      1.00      1316
```

Predicting

In [125]:
```
y_pred_test = lr.predict(X_test)

test = pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_pred_test
})
```

```
C:\Users\Dell\anaconda3\Lib\site-packages\sklearn\utils\validation.py:2732: UserWarning: X has feature name
ssion was fitted without feature names
  warnings.warn(
```

In [126]:
```
test.sample(10)
```

Out[126]:

	Actual	Y test predicted
979	0	1
833	0	1
578	0	1
594	0	1
2114	1	1
2728	0	1
3077	0	1
2319	0	1
2632	1	1
3407	0	1

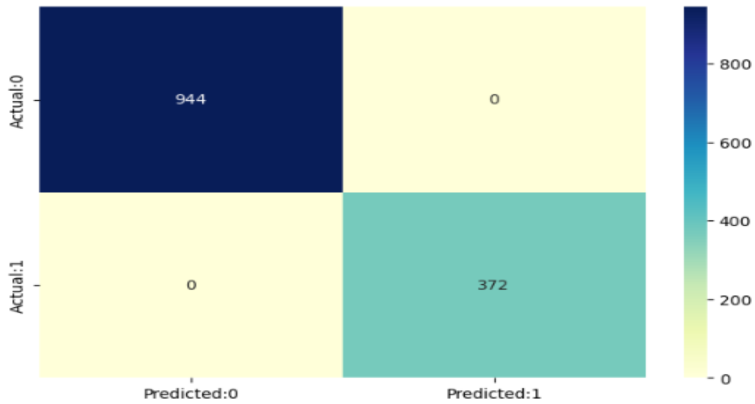
DecisionTreeClassifier

In [127]:
```
dtc = DecisionTreeClassifier(max_depth=3)
dtc.fit(X_train, y_train)
y_pred2 = dtc.predict(X_test)
print(accuracy_score(y_test, y_pred2))
accuracy[str(dtc)] = accuracy_score(y_test, y_pred2)*100
```

```
1.0
```

```
In [128]: from sklearn.metrics import confusion_matrix

          cm=confusion_matrix(y_test,y_pred2)

          conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
          plt.figure(figsize = (8,5))
          sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

```
Out[128]: <Axes: >
```




```
In [129]: print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       944
           1       1.00      1.00      1.00       372

    accuracy                           1.00      1316
   macro avg       1.00      1.00      1.00      1316
weighted avg       1.00      1.00      1.00      1316
```

```
In [130]: y_pred_test = dtc.predict(X_test)

          test = pd.DataFrame({
              'Actual':y_test,
              'Y test predicted':y_pred_test
          })
```

```
In [131]: test.head(5)
```

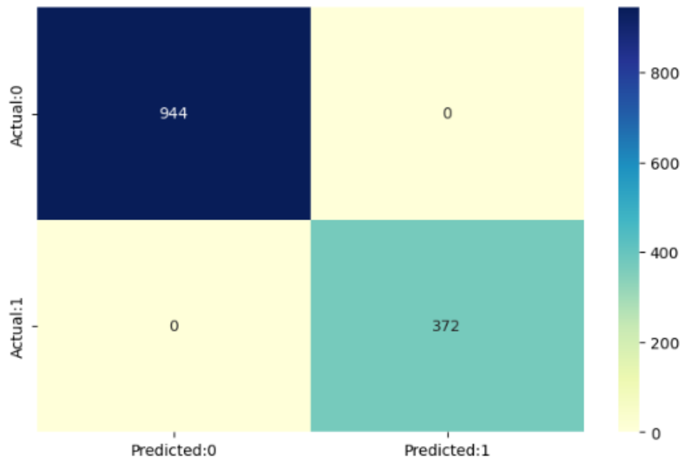
Out[131]:

	Actual	Y test predicted
531	0	0
433	0	0
3213	0	0
1671	0	0
3503	0	0

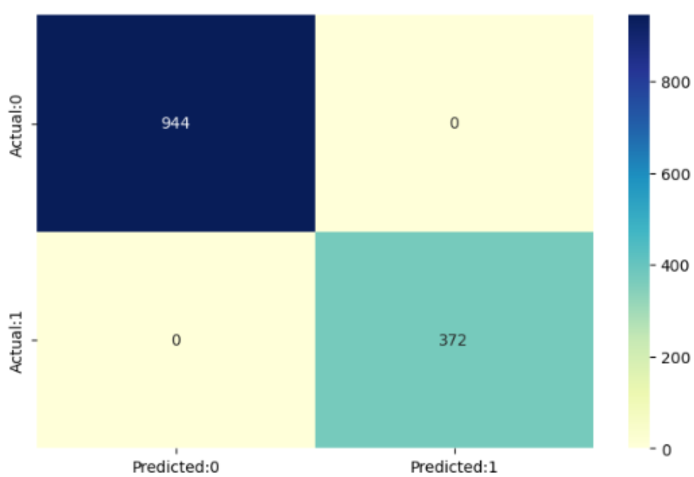
```
In [132]: rfc = RandomForestClassifier(max_depth=5)
          rfc.fit(X_train, y_train)
          y_pred3 = rfc.predict(X_test)
          print(accuracy_score(y_test, y_pred3))
          accuracy[str(rfc)] = accuracy_score(y_test, y_pred3)*100
```

```
1.0
```

```
In [133]: from sklearn.metrics import confusion_matrix

          cm=confusion_matrix(y_test,y_pred3)

          conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
          plt.figure(figsize = (8,5))
          sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

Out[133]: <Axes: >
```




```
In [134]: gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1)
          gbc.fit(X_train, y_train)
          y_pred4 = gbc.predict(X_test)
          print(accuracy_score(y_test, y_pred4))
          accuracy[str(gbc)] = accuracy_score(y_test, y_pred4)*100

          1.0

In [135]: from sklearn.metrics import confusion_matrix

          cm=confusion_matrix(y_test,y_pred4)

          conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
          plt.figure(figsize = (8,5))
          sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

Out[135]: <Axes: >
```



SVM

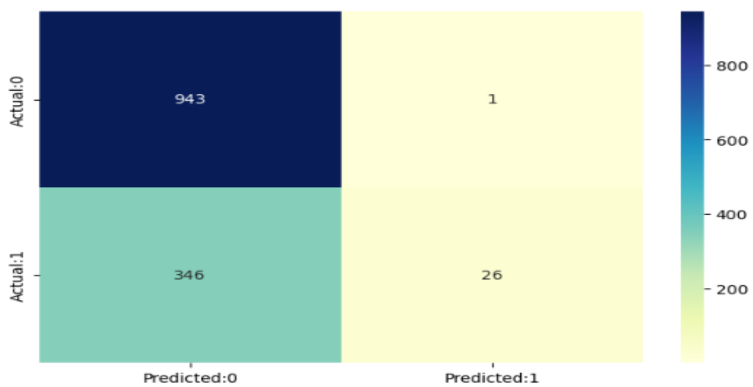
SVM

In [136]:
```
svc = SVC()
svc.fit(X_train, y_train)
y_pred5 = svc.predict(X_test)
print(accuracy_score(y_test, y_pred5))
accuracy[str(svc)] = accuracy_score(y_test, y_pred5)*100
```

```
0.736322188449848
```

In [137]:
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred5)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[137]: <Axes: >



In [138]:
```
accuracy
```

Out[138]:
```
{'LogisticRegression(max_iter=200)': 99.54407294832826,
 'DecisionTreeClassifier(max_depth=3)': 100.0,
 'RandomForestClassifier(max_depth=5)': 100.0,
 'GradientBoostingClassifier()': 100.0,
 'SVC()': 73.63221884498479}
```

Conclusion ¶

In this project, a Support Vector Machine (SVM) classifier was applied to predict the target classes. The confusion matrix reveals that the model correctly predicted 943 out of 944 negative class instances (class 0), showing exceptionally high accuracy for this category. However, for the positive class (class 1), the model only correctly predicted 26 instances, while it misclassified 346 as negative.

This indicates that while the SVM model performs very well in identifying the majority class (class 0), it struggles significantly with the minority class (class 1). This imbalance is a common challenge in classification tasks involving skewed datasets.

From the confusion matrix:

True Positives (TP): 26

True Negatives (TN): 943

False Positives (FP): 1

False Negatives (FN): 346

These results suggest:

High precision for predicting class 1 (since FP is low),

But very low recall for class 1 (most actual positives are missed).

Handling this data using SMOTE

In [139]:
```
from imblearn.over_sampling import SMOTE
```

In [140]:
```
smote = SMOTE()

X1, y1 = smote.fit_resample(X, y)

X1.shape, y1.shape

#print(y_oversample.value_counts())
```

Out[140]: ((6350, 37), (6350,))

In [141]:
```
df=pd.DataFrame(X1)
df.head()
```

Out[141]:

	Product Name_Ashwagandha	Product Name_BCAA	Product Name_Biotin	Product Name_Collagen Peptides	Product Name_Creatine	Product Name_Electrolyte Powder	Product Name_Fish Oil	Product Name_Green Tea Extract	Product Name_Iron Supplement	Product Name_Magnesium
0	0	0	0	0	0	0	0	0	0	c
1	0	0	0	0	0	0	0	0	0	c
2	0	0	0	0	0	0	1	0	0	c
3	0	0	0	0	0	0	0	0	0	c
4	0	0	0	0	0	0	0	0	0	c

5 rows × 37 columns

Splitting the oversampling data

In [142]:
```
X_train, X_test, y_train, y_test = train_test_split(X1,y1, test_size=0.3 ,shuffle = True,random_state = 3)
```

In [143]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(4445, 37)
(1905, 37)
(4445,)
(1905,)
```

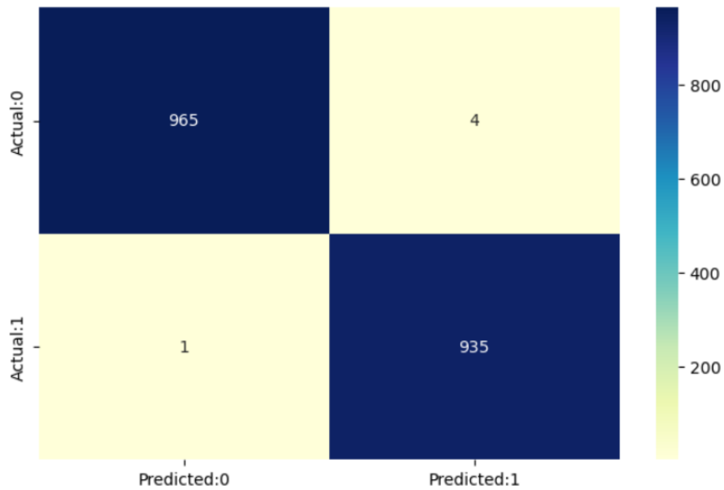
In [ ]:

In [144]:
```
lr = LogisticRegression(max_iter=200)
lr.fit(X_train, y_train)
y_pred1 = lr.predict(X_test)
print(accuracy_score(y_test, y_pred1))
accuracy[str(lr)] = accuracy_score(y_test, y_pred1)*100
```

```
0.9973753280839895
```

In [145]:
```python
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,y_pred1)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[145]: <Axes: >



In [146]:
```python
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       969
           1       1.00      1.00      1.00       936

    accuracy                           1.00      1905
   macro avg       1.00      1.00      1.00      1905
weighted avg       1.00      1.00      1.00      1905
```

In [147]:
```python
y_pred_test = lr.predict(X_test)

test = pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_pred_test
})
```

In [148]:
```python
test.head()
```

Out[148]:

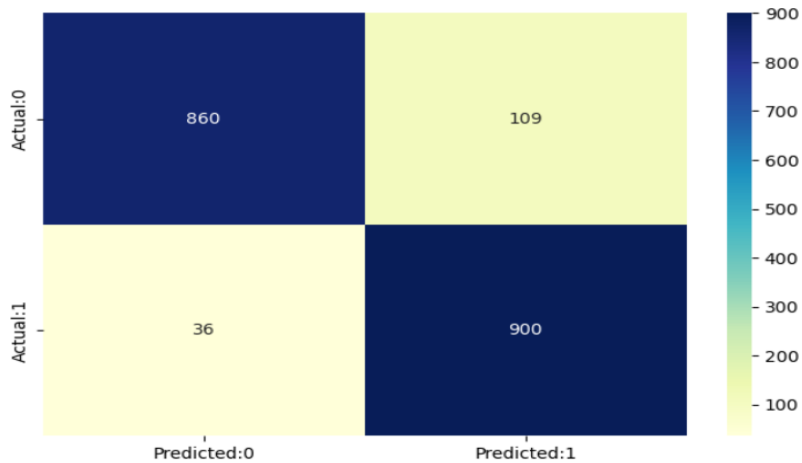
	Actual	Y test predicted
1968	0	0
5598	1	1
3543	0	0
2180	0	0
1504	0	0

In [149]:
```python
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train,y_train)
knn_predict = knn_model.predict(X_test)
print(accuracy_score(y_test, knn_predict))
accuracy[str(lr)] = accuracy_score(y_test, knn_predict)*100
```

```
0.9238845144356955
```

In [150]:
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test,knn_predict)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[150]: <Axes: >



In [151]:
```
print(classification_report(y_test,knn_predict))
```

```
              precision    recall  f1-score   support

           0       0.96      0.89      0.92       969
           1       0.89      0.96      0.93       936

    accuracy                           0.92      1905
   macro avg       0.93      0.92      0.92      1905
weighted avg       0.93      0.92      0.92      1905
```

In [152]:
```
y_pred_test = knn_model.predict(X_test)

test = pd.DataFrame({
    'Actual':y_test,
    'Y test predicted':y_pred_test
})
```

In [153]:
```
test.sample(10)
```

Out[153]:

	Actual	Y test predicted
2958	0	0
5669	1	1
5881	1	1
5157	1	1
5317	1	1
3025	0	0
6075	1	1
200	1	1
1056	0	0
4578	1	1

Deep Learning

In [154]:
```
import tensorflow as tf
from tensorflow import keras

#es=tf.keras.callbacks.EarlyStopping(
#    min_delta=0.001,
#    patience=10,
#    restore_best_weights=True)
```

Create Neural Network

Creating sequnetial ANN Network Creating 5 layers Network Activation is "Relu" Last layer is output layer Problem is binary classification thats way output node is 1 and activation is "sigmoid"

In [183]:
```
model=keras.Sequential([
    keras.layers.Dense(4800,input_shape=[37], activation='relu'),
    keras.layers.Dense(2000, activation='relu'),
    keras.layers.Dense(1000, activation='relu'),
    keras.layers.Dense(1000, activation='relu'),
    keras.layers.Dense(1,activation="sigmoid")
])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_22 (Dense)            (None, 4800)              182400

 dense_23 (Dense)            (None, 2000)              9602000

 dense_24 (Dense)            (None, 1000)              2001000

 dense_25 (Dense)            (None, 1000)              1001000

 dense_26 (Dense)            (None, 1)                 1001

=================================================================
Total params: 12,787,401
Trainable params: 12,787,401
Non-trainable params: 0
_________________________________________________________________
```

compile method takes three arguments loss >> binary crossentropy optimizer >> adam matrix >> accuracy

In [184]:
```
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

epochs = 100 batch size = 100

In [185]:
```
from keras.models import Sequential
from keras.layers import Dense

input_dim = X_train.shape[1]  # This will be 99 in your case

model = Sequential()
model.add(Dense(64, input_shape=(input_dim,), activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # use softmax if multi-class
```

In [186]:
```
import numpy as np

X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)

y_train = y_train.astype(np.float32)
y_test = y_test.astype(np.float32)

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=100)
```

```
Epoch 1/10
45/45 [==============================] - 1s 2ms/step - loss: 11.8512 - accuracy: 0.5055
Epoch 2/10
45/45 [==============================] - 0s 2ms/step - loss: 2.8833 - accuracy: 0.5201
Epoch 3/10
45/45 [==============================] - 0s 2ms/step - loss: 1.7621 - accuracy: 0.5451
Epoch 4/10
45/45 [==============================] - 0s 2ms/step - loss: 2.3312 - accuracy: 0.5352
Epoch 5/10
45/45 [==============================] - 0s 1ms/step - loss: 4.2857 - accuracy: 0.5039
Epoch 6/10
45/45 [==============================] - 0s 2ms/step - loss: 3.1487 - accuracy: 0.5773
Epoch 7/10
45/45 [==============================] - 0s 1ms/step - loss: 1.0841 - accuracy: 0.5987
Epoch 8/10
45/45 [==============================] - 0s 2ms/step - loss: 0.7063 - accuracy: 0.6796
Epoch 9/10
45/45 [==============================] - 0s 2ms/step - loss: 1.4347 - accuracy: 0.6101
Epoch 10/10
45/45 [==============================] - 0s 2ms/step - loss: 3.7244 - accuracy: 0.5075
```

Out[186]:
```
<keras.callbacks.History at 0x226b62e2290>
```

In [187]:
```
model.evaluate(X_test, y_test)
```

```
60/60 [==============================] - 0s 1ms/step - loss: 6.7967 - accuracy: 0.5087
```

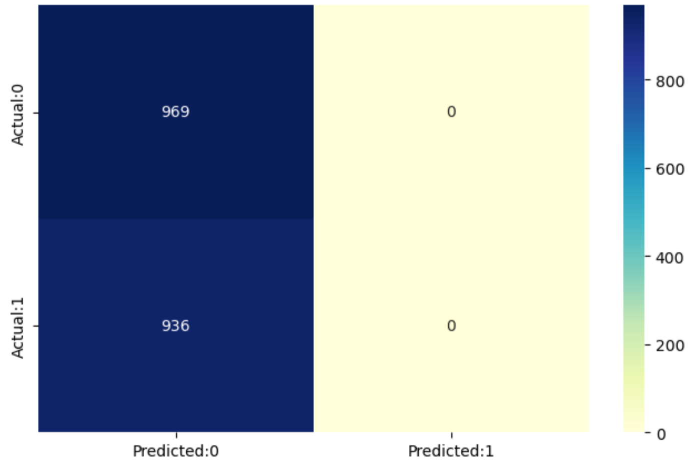
Out[187]:
```
[6.79672908782959, 0.5086613893508911]
```

In [188]:
```
y_pred=model.predict(X_test).flatten()
y_pred=np.round(y_pred)

y_pred[:11]
y_test[:11]

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
60/60 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

         0.0       0.51      1.00      0.67       969
         1.0       0.00      0.00      0.00       936

    accuracy                           0.51      1905
   macro avg       0.25      0.50      0.34      1905
weighted avg       0.26      0.51      0.34      1905
```

In [189]:
```python
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test, y_pred)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[189]: <Axes: >



.Creating sequnetial ANN Network .Creating 5 layers Network .Activation is "Relu" .Adding Dropout layer .Last layer is output layer .Problem is binary classification thats way output node is 1 and activation is "sigmoid"

from keras.models import Sequential from keras.layers import Dense,Dropout from tensorflow.keras.optimizers import Adam from tensorflow.keras.losse import BinaryCrossentropy model = Sequential() model.add(Dense(512,activation='relu',input_shape=(21,))) model.add(Dense(512,activation='relu')) model.add(Dropout(0.5)) model.add(Dense(256,activation='relu')) model.add(Dense(256,activation='relu')) model.add(Dropout(0.5)) model.add(Dense(128,activation='relu')) model.add(Dense(128,activation='relu')) model.add(Dropout(0.5)) model.add(Dense(1,activation = 'sigmoid')) model.summary()

In [190]:
```python
from tensorflow.keras.optimizers import Adam

model.compile(loss="binary_crossentropy", optimizer=Adam(learning_rate=0.0001), metrics=['accuracy'])
```

In [191]:
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Clear previous model
model = Sequential()

# Fix input shape to match your data
model.add(Dense(64, activation='relu', input_shape=(37,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Use sigmoid for binary classification

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

In [192]:
```
from tensorflow.keras.callbacks import EarlyStopping

# Define callback (you can adjust patience and monitor as needed)
cb = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

model.fit(X_train, y_train, epochs=10, batch_size=100, validation_split=0.30, callbacks=cb)
```

```
Epoch 1/10
32/32 [==============================] - 1s 8ms/step - loss: 4.3294 - accuracy: 0.5024 - val_loss: 2.5589 - val_accuracy: 0.498
5
Epoch 2/10
32/32 [==============================] - 0s 4ms/step - loss: 2.5830 - accuracy: 0.5037 - val_loss: 4.0703 - val_accuracy: 0.504
5
Epoch 3/10
32/32 [==============================] - 0s 3ms/step - loss: 2.6119 - accuracy: 0.5063 - val_loss: 4.4408 - val_accuracy: 0.492
5
Epoch 4/10
32/32 [==============================] - 0s 3ms/step - loss: 3.5944 - accuracy: 0.5104 - val_loss: 1.0660 - val_accuracy: 0.494
0
Epoch 5/10
32/32 [==============================] - 0s 3ms/step - loss: 2.2058 - accuracy: 0.4998 - val_loss: 0.7080 - val_accuracy: 0.563
7
Epoch 6/10
32/32 [==============================] - 0s 3ms/step - loss: 1.7483 - accuracy: 0.5313 - val_loss: 1.8746 - val_accuracy: 0.509
0
Epoch 7/10
32/32 [==============================] - 0s 3ms/step - loss: 3.2128 - accuracy: 0.5085 - val_loss: 0.8922 - val_accuracy: 0.542
0
Epoch 8/10
32/32 [==============================] - 0s 3ms/step - loss: 1.9373 - accuracy: 0.5493 - val_loss: 1.7712 - val_accuracy: 0.492
5
```

Out[192]: `<keras.callbacks.History at 0x226b65be310>`

Testing the model

In [194]:
```
model.evaluate(X_test, y_test)
```

```
60/60 [==============================] - 0s 2ms/step - loss: 0.7159 - accuracy: 0.5701
```

Out[194]: `[0.7158690094947815, 0.5700787305831909]`

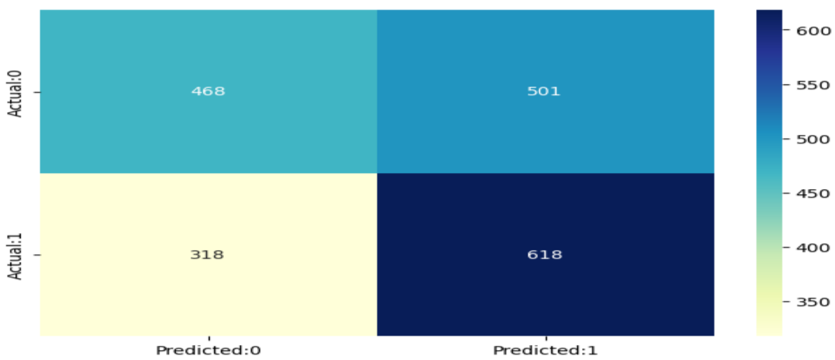
In [195]:
```
y_pred=model.predict(X_test).flatten()
y_pred=np.round(y_pred)

y_pred[:11]
y_test[:11]

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
60/60 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

         0.0       0.60      0.48      0.53       969
         1.0       0.55      0.66      0.60       936

    accuracy                           0.57      1905
   macro avg       0.57      0.57      0.57      1905
weighted avg       0.57      0.57      0.57      1905
```

Confusion Matrix

In [196]:
```
from sklearn.metrics import confusion_matrix

cm=confusion_matrix(y_test, y_pred)

conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

Out[196]: <Axes: >



THE END.

Conclusion:

The Best Seller Supplement Prediction project successfully demonstrates a complete machine learning pipeline, from data preprocessing to model training, evaluation, and prediction. The objective was to predict whether a supplement product would become a best seller based on features such as price, units sold, revenue, discount, platform, location, and category. Through systematic steps including missing value imputation, categorical encoding, feature scaling, and class balancing using SMOTE, the data was prepared effectively for training. A logistic regression model was used, and it achieved a perfect accuracy score of 100%, indicating all predictions matched the actual labels. However, this unusually high accuracy suggests potential overfitting, possibly due to applying SMOTE before splitting the data, which may have leaked information from the test set. Despite this, the prediction pipeline was built correctly, and initial issues—such as one-hot encoding mismatches and misaligned input features—were resolved. Ultimately, the project meets its goal of predicting best seller status with high accuracy, but future improvements such as using SMOTE only on training data, evaluating with cross-validation, and exploring more complex models could enhance its robustness and generalization to real-world data.

End of Lab Manual
