```
Start coding or generate with AI.
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data=pd.read_csv("/content/sample_data/CustomerData.csv")
```

```
data.head()
```

	Gender	Country	Product	Customer_Type	Age	Annual_Income	Purchase_Amount
0	Male	USA	Books	Returning	48	95000	100
1	Female	Canada	Furniture	New	35	65000	1200
			tronics	Returning	30	50000	600
			urniture	Returning	42	72000	1800
			lothing	New	27	55000	450

Run cell (Ctrl+Enter)
cell executed since last change

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.shape
```

```
(65, 8)
```

```
data.describe()
```

ValueError ValueError ValueError > •••

Please explain the error:

```
ValueError: could not convert string to float: 'Bool
```

Hypothesis

The error message "ValueError: could not convert string to float: 'Books'" indicates that the `StandardScaler` is encountering a string value ('Books') in the DataFrame `new_data3` while it expects numerical values. This is likely because the 'Product' column, which was label encoded earlier, is still present in the DataFrame and is being treated as a numerical feature by the scaler, despite its categorical nature. The `StandardScaler` only works with numerical features, and it fails when it finds non-numerical values like strings.

Suggested Changes

```
#scaling columns with standardization(standard sca
from sklearn.preprocessing import StandardScaler
scaling2=StandardScaler()

# Select only numerical features for scaling, excl
numerical_features = new_data3.select_dtypes(inclu
# Apply StandardScaler only to numerical features
data_standardized = pd.DataFrame(scaling2.fit_trar
# Concatenate scaled numerical features with categ
```

	Age	Annual_Income	Purchase_Amount	Satisfaction_Score
count	65.000000	65.000000	65.000000	65.000000
mean	40.292308	69169.230769	715.384615	7.261538
std	9.678783	16289.921377	543.566282	1.428252
min	26.000000	45000.000000	50.000000	4.000000
25%	32.000000	55000.000000	120.000000	7.000000
50%	39.000000	68000.000000	600.000000	7.000000
75%	47.000000	80000.000000	1200.000000	8.000000
max	63.000000	115000.000000	1800.000000	10.000000

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
                                 0 to 64
                                 lumns):
                                  Non-Null Count  Dtype
                                  --------------  -----
                                  65 non-null     object
 1   Country                      65 non-null     object
 2   Product                      65 non-null     object
 3   Customer_Type                65 non-null     object
 4   Age                          65 non-null     int64
 5   Annual_Income                65 non-null     int64
 6   Purchase_Amount              65 non-null     int64
 7   Satisfaction_Score           65 non-null     int64
dtypes: int64(4), object(4)
memory usage: 4.2+ KB
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

```
#there are total 8 columns in dataset
#gender,country ,product, and customer are categorical columns
#age,annual income, purchase amount and satisafaction score columnns are numerical co
```

```
data_standardized = pd.concat([data_standardized,
data_standardized.head()
```

Use code with caution

Explanation of Changes:

1. **`numerical_features = new_data3.select_dtypes(include=np.number).colu`** This line was modified to explicitly exclude the 'Product' `column from the list of numerical features selected for scaling.` The difference `method is used to remove the` `'Product'` column from the initially selected numerical columns.
2. **Apply StandardScaler and concatenate**: The rest of the code remains the same, ensuring that only the intended numerical features are scaled and then concatenated back with the categorical features in the final DataFrame.

▶ Sources

Rate this answer

```
#gender-binary categorical column
#country-nominal categorical column
#product- nominal categorical column
#customer type-binary categorical column as it contains only two unique category
```

```
#checking unique vcalues in categorical columns
data['Gender'].unique()
```

```
array(['Male', 'Female'], dtype=object)
```

```
data['Country'].unique()
```

```
array(['USA', 'Canada', 'Germany', 'UK'], dtype=object)
```

```
data['Product'].unique()
```

```
array(['Books', 'Furniture', 'Electronics', 'Clothing'], dtype=object)
```

```
     Run cell (Ctrl+Enter)
data[ cell executed since last change )
```

```
executed by Pranali Baviskar
10:23 AM (0 minutes ago)        '], dtype=object)
executed in 0.5s
```

```
#WE USE ONE HOT ENCODING FOR GENDER COLUMN
#WE USE ONE label encoding FOR country COLUMN
#we use label encoding for product column though it does not contain any order
#WE USE ONE HOT ENCODING FOR customer COLUMN
```

```
#label encoding for product column
from sklearn import preprocessing
LE=preprocessing.LabelEncoder()
```

```
new_data1=data.copy()
new_data1['Product`']=LE.fit_transform(new_data1['Product'])
new_data1.head()
```

	Gender	Country	Product	Customer_Type	Age	Annual_Income	Purchase_Amount
0	Male	USA	Books	Returning	48	95000	100
1	Female	Canada	Furniture	New	35	65000	1200
2	Male	Germany	Electronics	Returning	30	50000	600
3	Female	USA	Furniture	Returning	42	72000	1800
4	Male	UK	Clothing	New	27	55000	450

Next steps: Generate code with new_data1 View recommended plots New interactive sheet

```
#decoding the encoded column product
#like it gives us 0,1,2,3 then but how we can know??that which is 0,1,2,3
print(LE.inverse_transform([0,1,2,3]))
```

```
['Books' 'Clothing' 'Electronics' 'Furniture']
```

Run cell (Ctrl+Enter)
cell executed since last change

```
#one                        l;ude in pandas no need to import from sklearn
```

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

```
#one hopt encoding on remaining columns gender,country,and customer_type
new_data2=pd.get_dummies(new_data1,columns=['Gender','Country','Customer_Type'],prefi
new_data2.head()
```

	Product	Age	Annual_Income	Purchase_Amount	Satisfaction_Score	Product`	i
0	Books	48	95000	100	7	0	
1	Furniture	35	65000	1200	6	3	
2	Electronics	30	50000	600	8	2	
3	Furniture	42	72000	1800	7	3	
4	Clothing	27	55000	450	8	1	

Next steps: Generate code with new_data2 View recommended plots New interactive sheet

```
#we note that one hot encoded columns true-1 and false -0


#scaling columns with normalization (min-max scaling)

from sklearn.preprocessing import MinMaxScaler
scaling1=MinMaxScaler()


new_data3=new_data2.copy()
new_data3.head()
```

	Product	Age	Annual_Income	Purchase_Amount	Satisfaction_Score	Product`	i
0	Books	48	95000	100	7	0	
1	Furniture	35	65000	1200	6	3	
[illegible]	[illegible]	[illegible]	50000	600	8	2	
[illegible]	[illegible]	[illegible]	72000	1800	7	3	
[illegible]	[illegible]	[illegible]	55000	450	8	1	

Run cell (Ctrl+Enter)
cell executed since last change

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

Next steps: Generate code with new_data3 View recommended plots New interactive sheet

```
#one hopt encoding on remaining columns gender,country,and customer_type
new_data2=pd.get_dummies(new_data1,columns=['Gender','Country','Customer_Type'],prefi
new_data2.head()

#---SNIP---

new_data3=new_data2.copy()
# Select only numerical features for scaling
numerical_features = new_data3.select_dtypes(include=np.number).columns
```

```
# Apply MinMaxScaler only to numerical features
data_normalized = pd.DataFrame(scaling1.fit_transform(new_data3[numerical_features]),
# Concatenate scaled numerical features with categorical features
data_normalized = pd.concat([data_normalized, new_data3.drop(columns=numerical_feature
data_normalized.head()
```

	Age	Annual_Income	Purchase_Amount	Satisfaction_Score	Product`	Produ
0	0.594595	0.714286	0.028571	0.500000	0.000000	Boc
1	0.243243	0.285714	0.657143	0.333333	1.000000	Furnitu
2	0.108108	0.071429	0.314286	0.666667	0.666667	Electron
3	0.432432	0.385714	1.000000	0.500000	1.000000	Furnitu
4	0.027027	0.142857	0.228571	0.666667	0.333333	Clothi

Next steps: Generate code with data_normalized | View recommended plots | New interactive sh

Run cell (Ctrl+Enter)
cell executed since last change

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

```
#we o                          lized scaling (min max scaling)all numerical

#scaling columns with standardization(standard scaling)
from sklearn.preprocessing import StandardScaler
scaling2=StandardScaler()


#scaling columns with standardization(standard scaling)
from sklearn.preprocessing import StandardScaler
scaling2=StandardScaler()

# Select only numerical features for scaling, excluding 'Product' column
numerical_features = new_data3.select_dtypes(include=np.number).columns.difference(['Pr
# Apply StandardScaler only to numerical features
data_standardized = pd.DataFrame(scaling2.fit_transform(new_data3[numerical_features]),
# Concatenate scaled numerical features with categorical features
data_standardized = pd.concat([data_standardized, new_data3.drop(columns=numerical_feat
data_standardized.head()
```

	Age	Annual_Income	Purchase_Amount	Satisfaction_Score	Product	Produ
0	0.802547	1.598030	-1.140935	-0.184543	Books	
1	-0.551050	-0.257931	0.898486	-0.890148	Furniture	
2	-1.071664	-1.185912	-0.213925	0.521062	Electronics	
3	0.177810	0.175127	2.010898	-0.184543	Furniture	
4	-1.384033	-0.876585	-0.492028	0.521062	Clothing	

Next steps: Generate code with data_standardized View recommended plots New interactive

```
#we oberve that after standardised scaling all numerical columns are converted


# as compared to two scaling then we prefer standard scaling bcz ...it contains -1 to 1
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Pranali Baviskar
10:23 AM (0 minutes ago)
executed in 0.5s

```
Start AI.
```

Enter a prompt here

0 / 2000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more