```python
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

Importing the requiste libraries

```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt  ## library for data visualization
import seaborn as sns  ## library for data visualization

df = pd.read_csv("/kaggle/input/aerofit/aerofit_treadmill.txt") ##
Load the Dataset

df.head() # first 5 rows
```

	Product	Age	Gender	Education	MaritalStatus	Usage	Fitness	Income	Miles
0	KP281	18	Male	14	Single	3	4	29562	112
1	KP281	19	Male	15	Single	2	3	31836	75
2	KP281	19	Female	14	Partnered	4	3	30699	66
3	KP281	19	Male	12	Single	3	3	32973	85

```
4   KP281   20    Male         13     Partnered      4        2
35247     47
```

```
df.shape
```

```
(180, 9)
```

```
df.info() ## There were no null valuyes in any of the rows or columns
and the info gives the datatypes of each columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Product        180 non-null    object
 1   Age            180 non-null    int64
 2   Gender         180 non-null    object
 3   Education      180 non-null    int64
 4   MaritalStatus  180 non-null    object
 5   Usage          180 non-null    int64
 6   Fitness        180 non-null    int64
 7   Income         180 non-null    int64
 8   Miles          180 non-null    int64
dtypes: int64(6), object(3)
memory usage: 12.8+ KB
```

```
df.describe().T
```

```
           count          mean           std      min       25%
50%  \
Age        180.0     28.788889      6.943498     18.0     24.00
26.0
Education  180.0     15.572222      1.617055     12.0     14.00
16.0
Usage      180.0      3.455556      1.084797      2.0      3.00
3.0
Fitness    180.0      3.311111      0.958869      1.0      3.00
3.0
Income     180.0  53719.577778  16506.684226  29562.0  44058.75
50596.5
Miles      180.0    103.194444     51.863605     21.0     66.00
94.0

                75%       max
Age           33.00      50.0
Education     16.00      21.0
Usage          4.00       7.0
Fitness        4.00       5.0
Income     58668.00  104581.0
Miles        114.75     360.0
```

```python
df.isna().sum() # There are no null values in any of the columns
```

```
Product          0
Age              0
Gender           0
Education        0
MaritalStatus    0
Usage            0
Fitness          0
Income           0
Miles            0
dtype: int64
```

```python
#to find the unique values in the dataset columns
for i in df.columns:
    print(i," : ", df[i].unique()," ", df[i].nunique)
```

```
Product  :  ['KP281' 'KP481' 'KP781']   <bound method
IndexOpsMixin.nunique of 0      KP281
1      KP281
2      KP281
3      KP281
4      KP281
       ...
175    KP781
176    KP781
177    KP781
178    KP781
179    KP781
Name: Product, Length: 180, dtype: object>
Age  :  [18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41
 43 44 46 47 50 45 48 42]   <bound method IndexOpsMixin.nunique of 0
18
1      19
2      19
3      19
4      20
       ..
175    40
176    42
177    45
178    47
179    48
Name: Age, Length: 180, dtype: int64>
Gender  :  ['Male' 'Female']   <bound method IndexOpsMixin.nunique of
0        Male
1        Male
2      Female
3        Male
```

```
4        Male
        ...
175      Male
176      Male
177      Male
178      Male
179      Male
Name: Gender, Length: 180, dtype: object>
Education  :  [14 15 12 13 16 18 20 21]   <bound method
IndexOpsMixin.nunique of 0      14
1      15
2      14
3      12
4      13
       ..
175    21
176    18
177    16
178    18
179    18
Name: Education, Length: 180, dtype: int64>
MaritalStatus  :  ['Single' 'Partnered']   <bound method
IndexOpsMixin.nunique of 0         Single
1         Single
2      Partnered
3         Single
4      Partnered
         ...
175       Single
176       Single
177       Single
178    Partnered
179    Partnered
Name: MaritalStatus, Length: 180, dtype: object>
Usage  :  [3 2 4 5 6 7]   <bound method IndexOpsMixin.nunique of 0
3
1      2
2      4
3      3
4      4
      ..
175    6
176    5
177    5
178    4
179    4
Name: Usage, Length: 180, dtype: int64>
Fitness  :  [4 3 2 1 5]   <bound method IndexOpsMixin.nunique of 0
4
```

```
1      3
2      3
3      3
4      2
      ..
175    5
176    4
177    5
178    5
179    5
Name: Fitness, Length: 180, dtype: int64>
Income  :  [ 29562  31836  30699  32973  35247  37521  36384  38658
40932  34110
  39795  42069  44343  45480  46617  48891  53439  43206  52302  51165
  50028  54576  68220  55713  60261  67083  56850  59124  61398  57987
  64809  47754  65220  62535  48658  54781  48556  58516  53536  61006
  57271  52291  49801  62251  64741  70966  75946  74701  69721  83416
  88396  90886  92131  77191  52290  85906 103336  99601  89641  95866
 104581  95508]   <bound method IndexOpsMixin.nunique of 0       29562
1       31836
2       30699
3       32973
4       35247
        ...
175     83416
176     89641
177     90886
178    104581
179     95508
Name: Income, Length: 180, dtype: int64>
Miles  :  [112  75  66  85  47 141 103  94 113  38 188  56 132 169  64
53 106  95
 212  42 127  74 170  21 120 200 140 100  80 160 180 240 150 300 280
260
 360]   <bound method IndexOpsMixin.nunique of 0      112
1       75
2       66
3       85
4       47
      ...
175    200
176    200
177    160
178    120
179    180
Name: Miles, Length: 180, dtype: int64>

## From the above we can differentiate the continuos and discrete
variables
```

```
## Age, Miles, Education and Income are continous variables and Marital Status, Gender and Fitness are categoriacal variables
```

Checking value counts for categorical columns

```
df["Product"].value_counts()

Product
KP281    80
KP481    60
KP781    40
Name: count, dtype: int64

df["MaritalStatus"].value_counts()

MaritalStatus
Partnered    107
Single        73
Name: count, dtype: int64

df["Gender"].value_counts

<bound method IndexOpsMixin.value_counts of 0        Male
1        Male
2      Female
3        Male
4        Male
        ...
175      Male
176      Male
177      Male
178      Male
179      Male
Name: Gender, Length: 180, dtype: object>

df["Fitness"].value_counts()

Fitness
3    97
5    31
2    26
4    24
1     2
Name: count, dtype: int64

## else we could run value_counts on every column
for i in df.columns:
    print(i)
    print(df[i].value_counts())
```

```
Product
Product
KP281    80
KP481    60
KP781    40
Name: count, dtype: int64
Age
Age
25    25
23    18
24    12
26    12
28     9
35     8
33     8
30     7
38     7
21     7
22     7
27     7
31     6
34     6
29     6
20     5
40     5
32     4
19     4
48     2
37     2
45     2
47     2
46     1
50     1
18     1
44     1
43     1
41     1
39     1
36     1
42     1
Name: count, dtype: int64
Gender
Gender
Male      104
Female     76
Name: count, dtype: int64
Education
Education
16    85
14    55
```

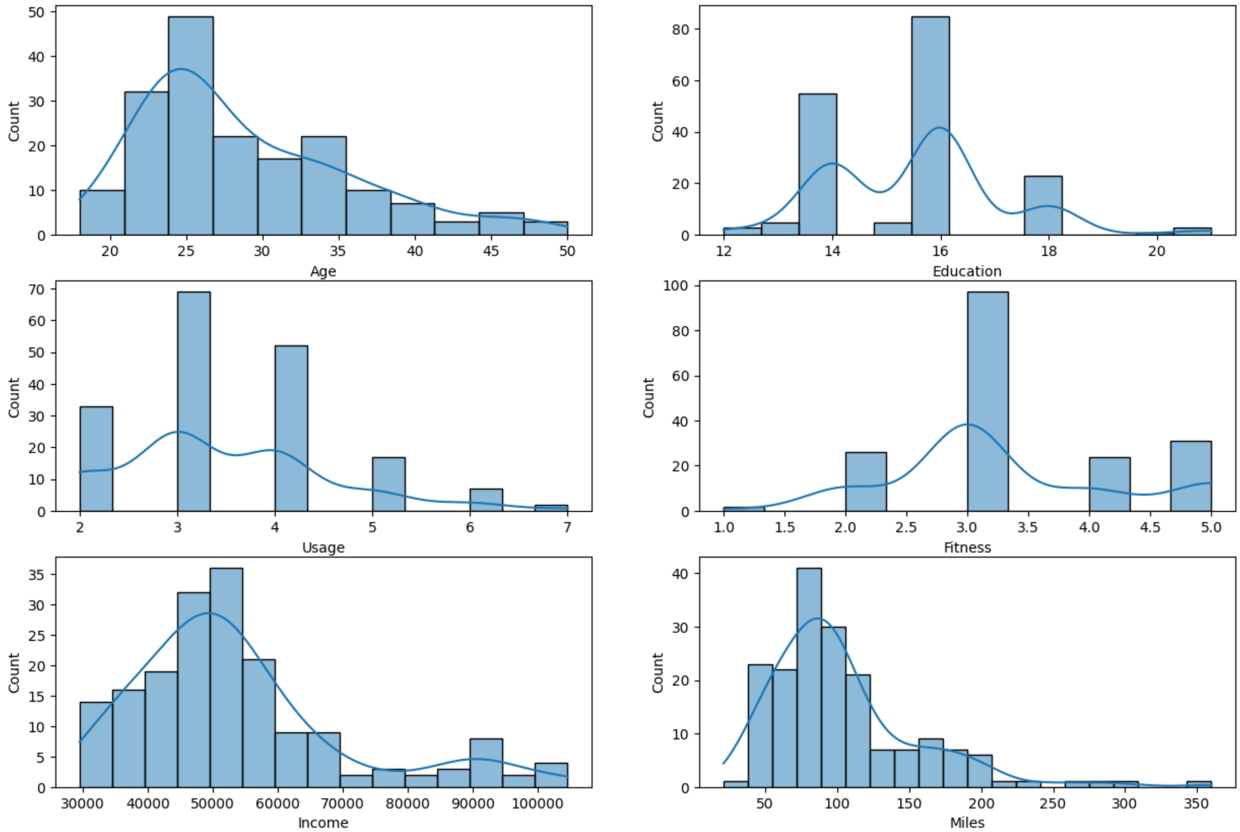
```
18    23
15     5
13     5
12     3
21     3
20     1
Name: count, dtype: int64
MaritalStatus
MaritalStatus
Partnered    107
Single        73
Name: count, dtype: int64
Usage
Usage
3    69
4    52
2    33
5    17
6     7
7     2
Name: count, dtype: int64
Fitness
Fitness
3    97
5    31
2    26
4    24
1     2
Name: count, dtype: int64
Income
Income
45480    14
52302     9
46617     8
54576     8
53439     8
         ..
65220     1
55713     1
68220     1
30699     1
95508     1
Name: count, Length: 62, dtype: int64
Miles
Miles
85     27
95     12
66     10
75     10
47      9
```

```
106    9
94     8
113    8
53     7
100    7
180    6
200    6
56     6
64     6
127    5
160    5
42     4
150    4
38     3
74     3
170    3
120    3
103    3
132    2
141    2
280    1
260    1
300    1
240    1
112    1
212    1
80     1
140    1
21     1
169    1
188    1
360    1
Name: count, dtype: int64
```

Univariate Analysis

```
# for Continuous variables

fig,axis=plt.subplots(nrows=3,ncols=2,figsize=(15,10))
sns.histplot(df["Age"],kde=True,ax=axis[0,0])
sns.histplot(df["Education"], kde=True, ax=axis[0,1])
sns.histplot(df["Usage"], kde=True, ax=axis[1,0])
sns.histplot(df["Fitness"], kde=True, ax=axis[1,1])
sns.histplot(df["Income"], kde=True, ax=axis[2,0])
sns.histplot(df["Miles"], kde=True, ax=axis[2,1])
plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

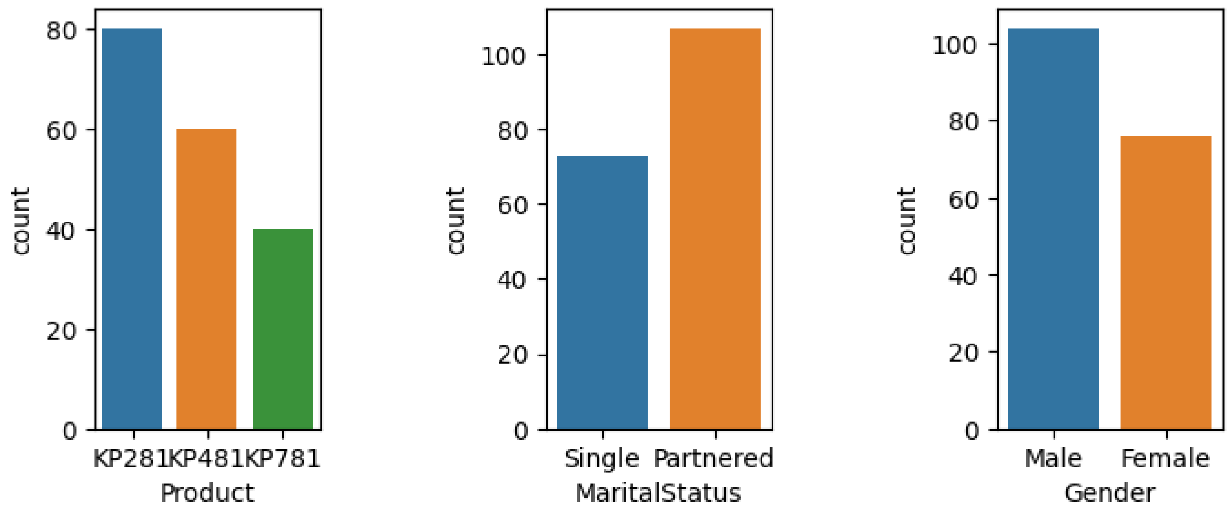


Miles and Income have significant right skewness in the graph which means are outliers

The profile of majority Treadmill buyer is that of Age=25, Fitness= level 3, Education = 16 years, Income = 45000 - 600000

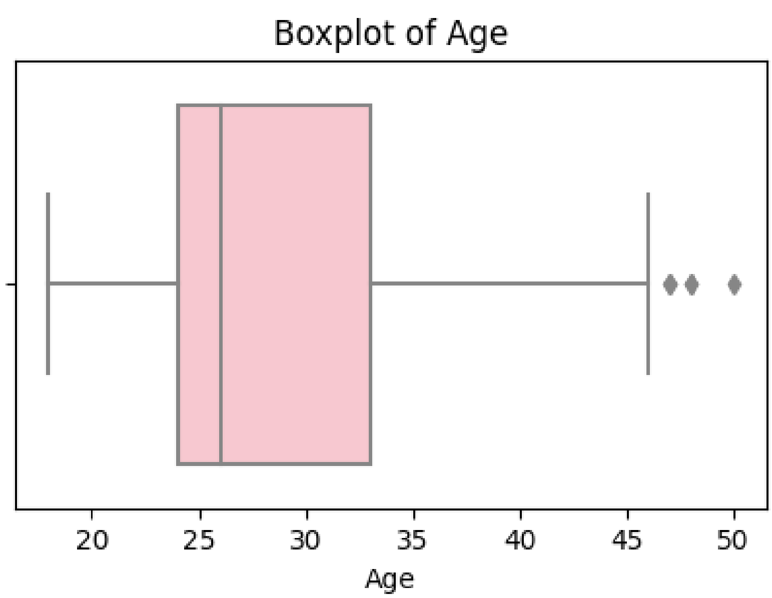
```
# For Categorical variables

fig, axis= plt.subplots(nrows=1, ncols=3,  figsize=(8,3))
fig.subplots_adjust(wspace=1)
sns.countplot(x=df["Product"], ax=axis[0])
sns.countplot(x=df["MaritalStatus"], ax=axis[1])
sns.countplot(x=df["Gender"], ax=axis[2])
```

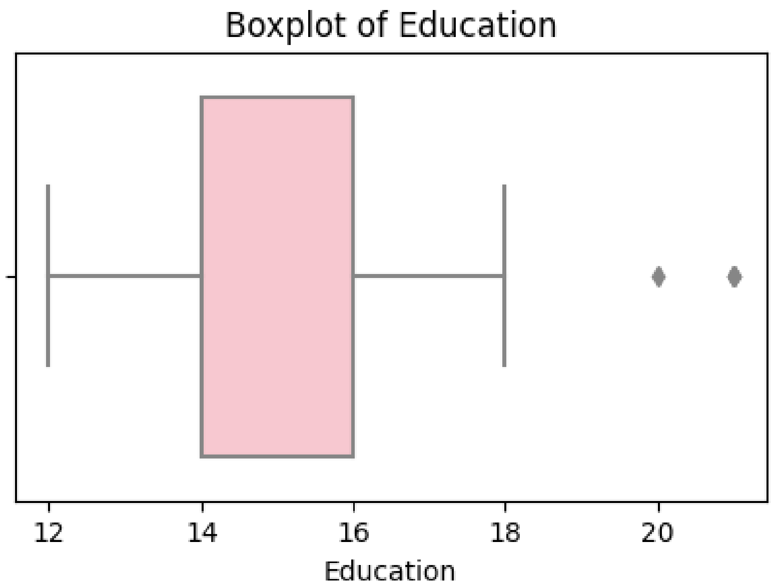
```
<Axes: xlabel='Gender', ylabel='count'>
```

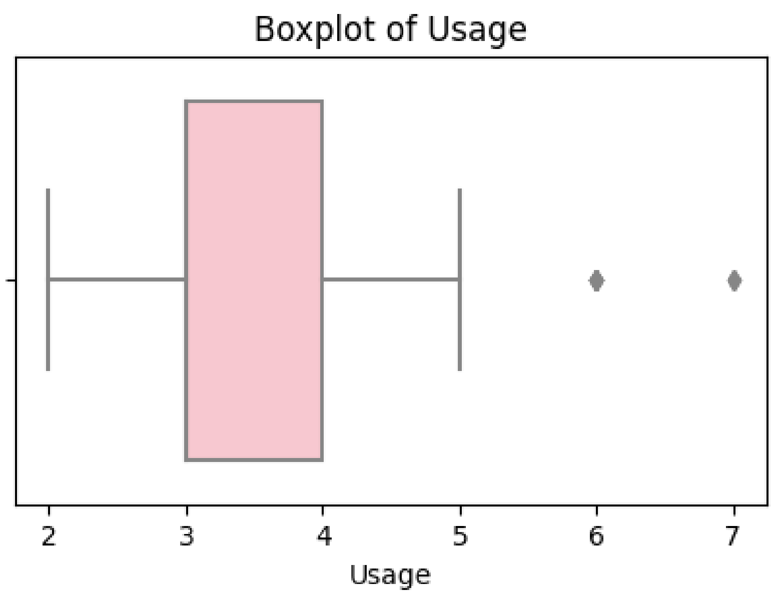


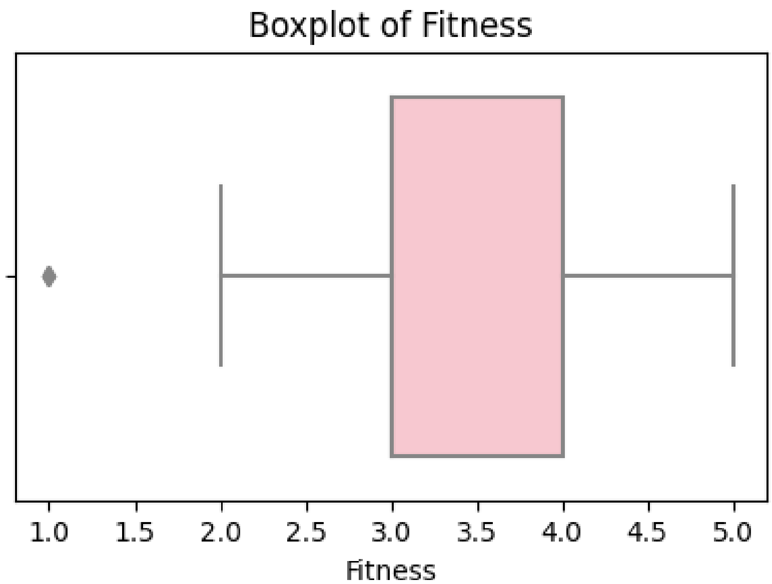
Boxplots

```
for i in ["Age", "Education", "Usage", "Fitness", "Income", "Miles"]:
    plt.figure(figsize=(5, 3))
    sns.boxplot(x=df[i], orient="h", color="pink")
    plt.title(f"Boxplot of {i}")
    plt.show()
```

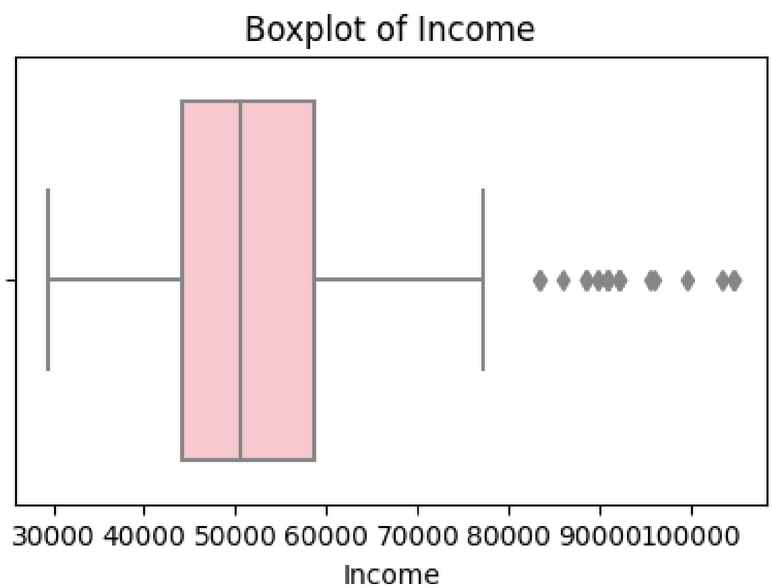



Boxplot of Education
12
14
16
18
20
Education

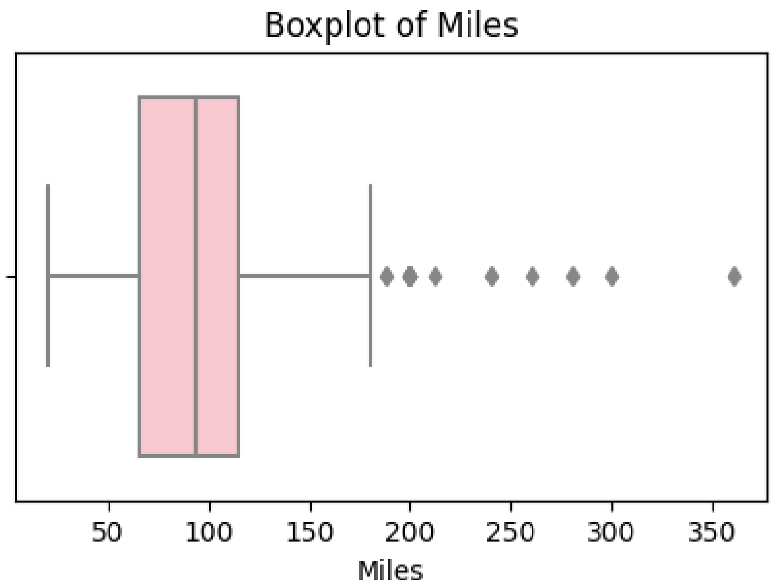

Boxplot of Usage
2
3
4
5
6
7
Usage



Boxplot of Fitness
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Fitness



Boxplot of Income
30000
40000
50000
60000
70000
80000
90000100000
Income



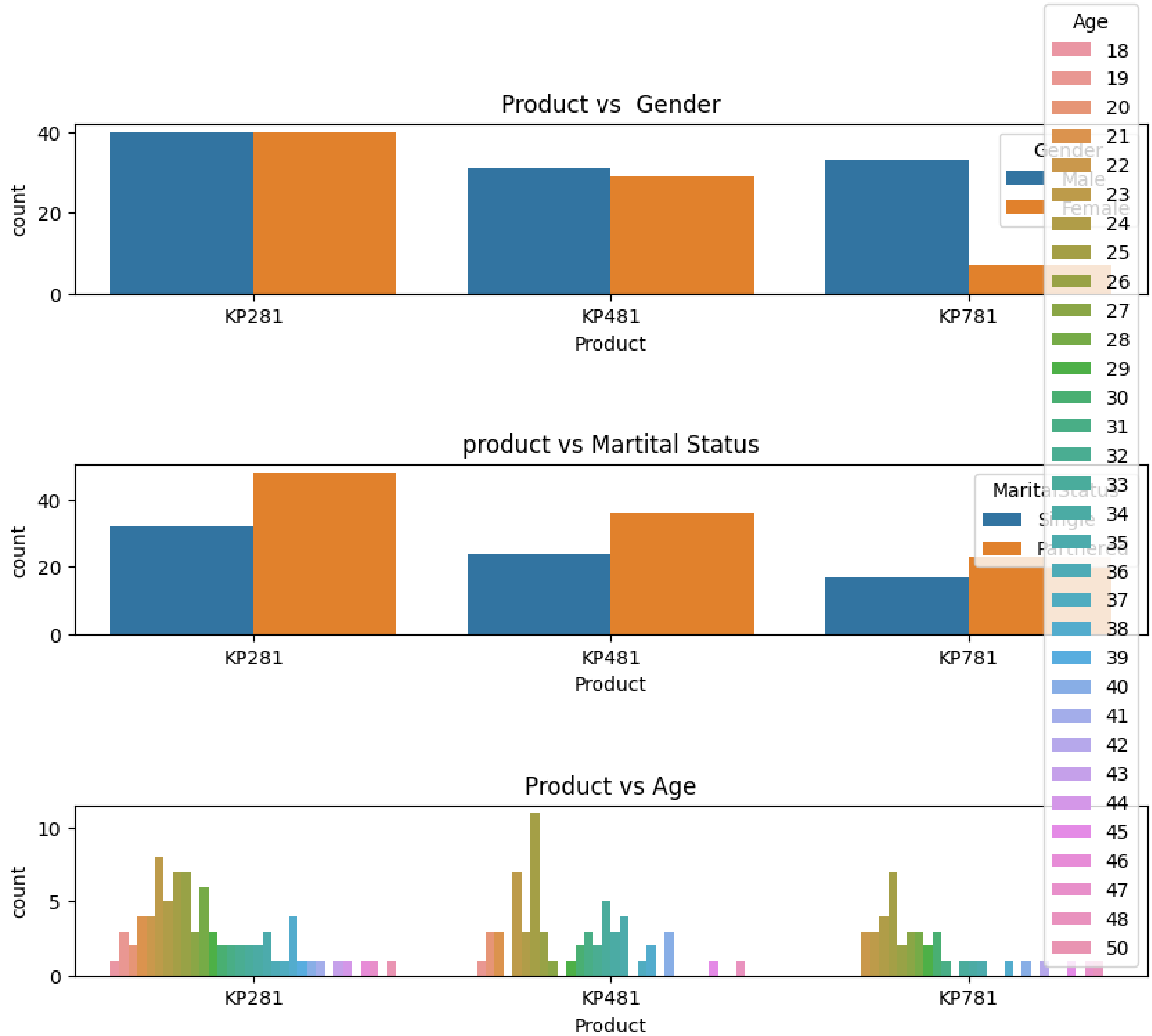
The boxplots are also iterating the same as the histplots -

Many outliers in Income and Miles group

The profile of majority buyers is Age between 25-35 Education between 14 to 16 years Fitness levels between 3 and 4 Income ranging from 45000 to 60000

Bivariate Analysis

```
fig, axis=plt.subplots(nrows=3,ncols=1, figsize=(10,8))
fig.subplots_adjust(hspace=1)
sns.countplot(data=df, x="Product", hue="Gender", ax=axis[0])
sns.countplot(data=df, x="Product", hue="MaritalStatus", ax=axis[1])
sns.countplot(data=df, x="Product", hue="Age", ax=axis[2])
axis[0].set_title("Product vs  Gender")
axis[1].set_title("product vs Martital Status")
axis[2].set_title("Product vs Age")
plt.show()
```



Parterned people are buying more of the 3 models of the treadmills

Females are purchasing less treadmills than males for models KP781 and KP481 while there are on the same level with KP281

THe Age group 24 to 28 are the highest purchasers for any of the treadmill model

Multivariate Analysis

```
cols=["Age", "Education", "Usage", "Fitness", "Income", "Miles"]
df[cols].corr()
```

	Age	Education	Usage	Fitness	Income	Miles
Age	1.000000	0.280496	0.015064	0.061105	0.513414	0.036618
Education	0.280496	1.000000	0.395155	0.410581	0.625827	0.307284

```
Usage      0.015064   0.395155  1.000000  0.668606  0.519537  0.759130
Fitness    0.061105   0.410581  0.668606  1.000000  0.535005  0.785702
Income     0.513414   0.625827  0.519537  0.535005  1.000000  0.543473
Miles      0.036618   0.307284  0.759130  0.785702  0.543473  1.000000

fig,axis=plt.subplots(figsize=(10,8))
sns.heatmap(df[cols].corr(), annot=True,ax=axis)

<Axes: >
```

(Miles & Fitness) and (Miles & Usage) attributes are highly correlated, which means if a customer's fitness level is high they use more treadmills.

Income and Education shows a strong correlation. High-income and highly educated people prefer the KP781 treadmill which is having advanced features.

There is no correlation between (Usage & Age) or (Fitness & Age) attributes, which mean Age should not be a barrier to using treadmills or specific model of treadmills.

Conditional Probability

```
## Impact of model of Treadmill with respect to Gender

pd.crosstab(index=df["Product"], columns=df["Gender"], margins=True,
normalize="index")
```

```
Gender      Female      Male
Product
KP281     0.500000  0.500000
KP481     0.483333  0.516667
KP781     0.175000  0.825000
All       0.422222  0.577778
```

Marginal Probabilities

P(KP281)= 0.444

p(KP481)= 0.333

P(KP781)= 0.222

P(Males)= 0.58

P(Females)=0.42

With all the above steps you can answer questions like: What is the probability of a male customer buying a KP781 treadmill?

P(KP781/Male)=0.18

```
## impact of Marital Status on model purchased

pd.crosstab(index=df["Product"],
columns=df["MaritalStatus"],normalize="index", margins=True)
```

```
MaritalStatus  Partnered    Single
Product
KP281           0.600000  0.400000
KP481           0.600000  0.400000
KP781           0.575000  0.425000
All             0.594444  0.405556
```

P(Parterned) =0.59444

p(Single) = 0.40

Customer Profiling:

Gender: Male and Female Marital Status: partnered and single Age: 24-28 Income:29000-60000 Education: 14-16 years Fitness: 3 and 4 Usage: 3 times per week Miles: Customers run 60-100 miles per week