DATA EXTRACTION

In [1]:
```
#importing libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#loading dataset

url=pd.read_csv(r"E:\cloth_size _predicition.csv",header=0)
df=pd.DataFrame(url)
df
```

Out[1]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
...	...	...	...	...
119729	63	42.0	175.26	M
119730	45	29.0	154.94	S
119731	61	31.0	172.72	M
119732	74	31.0	167.64	XL
119733	70	30.0	167.64	XL

119734 rows × 4 columns

SUMMARIZATION

In [2]:
```
#describing the data

df.describe()
```

Out[2]:

	weight	age	height
count	119734.000000	119477.000000	119404.000000
mean	61.756811	34.027311	165.805794
std	9.944863	8.149447	6.737651
min	22.000000	0.000000	137.160000
25%	55.000000	29.000000	160.020000
50%	61.000000	32.000000	165.100000
75%	67.000000	37.000000	170.180000
max	136.000000	117.000000	193.040000

In [3]:
```
df.shape
```

Out[3]:
```
(119734, 4)
```

In [4]:
```
df.dtypes
```

Out[4]:
```
weight      int64
age       float64
height    float64
size       object
dtype: object
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['weight', 'age', 'height', 'size'], dtype='object')
```

In [6]:
```
df.head(20)
```

Out[6]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
5	50	27.0	160.02	S
6	53	65.0	160.02	M
7	51	33.0	160.02	XXS
8	54	26.0	167.64	M
9	53	32.0	165.10	S
10	63	30.0	170.18	XXXL
11	77	35.0	172.72	XXXL
12	64	26.0	165.10	L
13	52	28.0	160.02	M
14	65	33.0	165.10	L
15	63	30.0	167.64	L
16	54	21.0	167.64	XXS
17	63	27.0	172.72	M
18	63	30.0	167.64	M
19	54	20.0	167.64	S

In [7]:
```
df.tail(1)
```

Out[7]:

	weight	age	height	size
119733	70	30.0	167.64	XL

DATA CLEANING

In [8]:
```
#Cells with data of wrong format can make it difficult, or even impossible, to ar

#To fix it, you have two options: remove the rows, or convert all cells in the co
```

In [9]:
```
#It's important to understand these different types of missing data from a statis
#The type of missing data will influence how you deal with filling in the missing
```

In [10]:
```
#checking that missing values are present in every column?

df[["weight","height","age","size"]][0:1000:8].isnull()

# it looks like no missing values are present. but we have to check entire data.
```

Out[10]:

	weight	height	age	size
0	False	False	False	False
8	False	False	False	False
16	False	False	False	False
24	False	False	False	False
32	False	False	False	False
...	...	...	...	...
960	False	False	False	False
968	False	False	False	False
976	False	False	False	False
984	False	False	False	False
992	False	False	False	False

125 rows × 4 columns

In [11]:
```
#looking into the which type of bad data is present?

#checking for standard missing values: Blank cells, NaN, n/a → These will be trea

df.isnull().sum()
```

Out[11]:
```
weight      0
age       257
height    330
size        0
dtype: int64
```

In [12]:
```
#checking for non standard missing values

#Sometimes missing values will be entered like .., __, - , missing, na, @,??,***,

#To find these

df["weight"].unique()
```

Out[12]:
```
array([ 62,  59,  61,  65,  50,  53,  51,  54,  63,  77,  64,  52,  55,
        74,  58,  47,  86,  68,  78,  49,  56,  81,  72,  60,  70,  79,
       113,  88, 104,  57,  95,  99,  80,  71,  83,  90,  66,  46,  73,
        48, 112,  69,  84,  45,  97,  75,  43,  67,  92,  76,  44, 102,
        40,  96, 108,  89,  82, 115, 122,  87,  91,  93,  85, 111,  98,
        94, 101, 125, 126, 131, 106, 107,  42,  22, 117,  31, 136, 120,
       123,  41, 118, 116, 100, 105, 124, 103,  35, 129, 119, 121, 109,
        26,  38,  36, 114,  39], dtype=int64)
```

In [13]:
```
df["height"].unique()
```

Out[13]:
```
array([172.72, 167.64, 165.1 , 175.26, 160.02, 170.18, 154.94, 177.8 ,
       157.48, 162.56, 147.32, 185.42, 180.34, 152.4 , 142.24, 182.88,
       187.96,    nan, 149.86, 190.5 , 144.78, 137.16, 193.04, 139.7 ])
```

In [14]:
```
df["age"].unique()
```

Out[14]:
```
array([ 28.,  36.,  34.,  27.,  45.,  65.,  33.,  26.,  32.,  30.,  35.,
        21.,  20.,  37.,  50.,  43.,  29.,  47.,  31.,  48.,  40.,  52.,
        24.,  25.,  49.,  22.,  42.,  53.,  69.,  23.,  17.,  44.,  41.,
        59.,  39.,  38.,  51.,  46.,  54.,  nan,  62.,  58.,  19.,  55.,
        60.,  61.,  57.,  56.,  70.,  16.,   0.,  64.,  63.,  77.,  68.,
        66.,  18.,  67.,  75.,   3.,  85.,  87., 116.,  72.,   2.,  91.,
       117.,  71.,  92.,   9.,  99.,  15.,   4.,  76.,  14., 100.,   1.,
         5.,  73., 113.,  81.,  88., 112.])
```

In [15]:
```
df["size"].unique()
```

Out[15]:
```
array(['XL', 'L', 'M', 'S', 'XXS', 'XXXL', 'XXL'], dtype=object)
```

In [16]:
```
#filling missing values

#Different ways to fill the missing values

#Mean/Median, Mode

#bfill,ffill

#interpolate

#replace
```

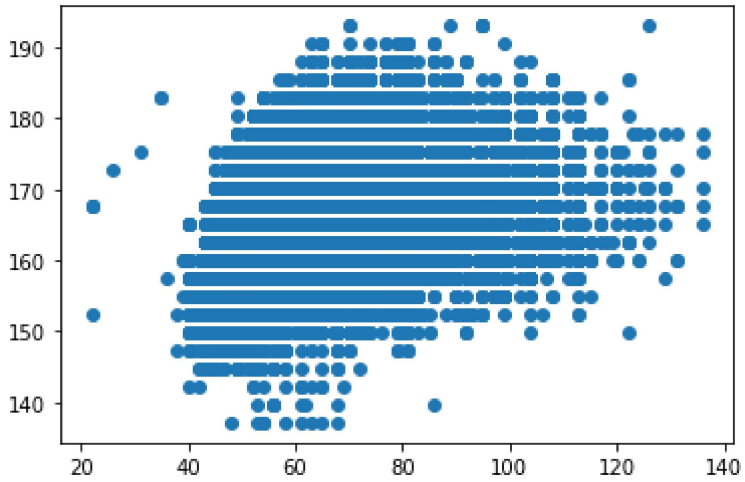
In [17]:
```
# In columns having numerical data, we can fill the missing values by mean/median
# Mean – When the data has no outliers. Mean is the average value. Mean will be a
# [Example. If we are calculating, mean salary of the employees in a room and if
# Median – When the data has more outliers, it's best to replace them with the me
# In columns having categorical data, we can fill the missing values by mode
# Mode – Most common value.
```

In [18]:
```
#method-1

df.describe()[["weight","age"]]
```

Out[18]:

	weight	age
count	119734.000000	119477.000000
mean	61.756811	34.027311
std	9.944863	8.149447
min	22.000000	0.000000
25%	55.000000	29.000000
50%	61.000000	32.000000
75%	67.000000	37.000000
max	136.000000	117.000000

In [19]:
```
#method-2 - using explaratory analysis

fig = plt.scatter(x=df["weight"], y=df["height"])
```



In [20]:
```
#method-3

def find_outliers_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   return outliers
```

In [21]:
```
outliers = find_outliers_IQR(df["height"])

print("number of outliers: "+ str(len(outliers)))

print("max outlier value: "+ str(outliers.max()))

print("min outlier value: "+ str(outliers.min()))
```

```
number of outliers: 185
max outlier value: 193.04
min outlier value: 137.16
```

In [22]:
```
outliers = find_outliers_IQR(df["weight"])

print("number of outliers: "+ str(len(outliers)))

print("max outlier value: "+ str(outliers.max()))

print("min outlier value: "+ str(outliers.min()))
```

```
number of outliers: 3510
max outlier value: 136
min outlier value: 22
```

In [23]:
```
def drop_outliers_IQR(df):

   q1=df.quantile(0.25)

   q3=df.quantile(0.75)

   IQR=q3-q1

   not_outliers = df[~((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

   outliers_dropped = outliers.dropna().reset_index()

   return outliers_dropped
```

In [24]:
```
df
```

Out[24]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
...	...	...	...	...
119729	63	42.0	175.26	M
119730	45	29.0	154.94	S
119731	61	31.0	172.72	M
119732	74	31.0	167.64	XL
119733	70	30.0	167.64	XL

119734 rows × 4 columns

In [25]:
```
#In the above process we find that there are huge number of outliers. for having
#But here i findout the outliers and delete the outliers. so there no outliers ir

# Step 1: Arrange the scores in numerical order.## not compulsory
# Step 2: add every value present in the every column and
# Step 3: Divide the sum by its total score count
# Step 4: Now use mean value to replace in standard missing values


mean=df.mean()
print(mean)
```

```
weight     61.756811
age        34.027311
height    165.805794
dtype: float64
```

In [26]:
```
# now replace with those values in their columns at where the missing values are

df["weight"].fillna(61.756811,inplace=True)
df
```

Out[26]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
...	...	...	...	...
119729	63	42.0	175.26	M
119730	45	29.0	154.94	S
119731	61	31.0	172.72	M
119732	74	31.0	167.64	XL
119733	70	30.0	167.64	XL

119734 rows × 4 columns

In [27]:
```
df["height"].fillna(165.805794,inplace=True)
df
```

Out[27]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
...	...	...	...	...
119729	63	42.0	175.26	M
119730	45	29.0	154.94	S
119731	61	31.0	172.72	M
119732	74	31.0	167.64	XL
119733	70	30.0	167.64	XL

119734 rows × 4 columns

In [28]:
```
df["age"].fillna(34.027311,inplace=True)
df
```

Out[28]:

	weight	age	height	size
0	62	28.0	172.72	XL
1	59	36.0	167.64	L
2	61	34.0	165.10	M
3	65	27.0	175.26	L
4	62	45.0	172.72	M
...	...	...	...	...
119729	63	42.0	175.26	M
119730	45	29.0	154.94	S
119731	61	31.0	172.72	M
119732	74	31.0	167.64	XL
119733	70	30.0	167.64	XL

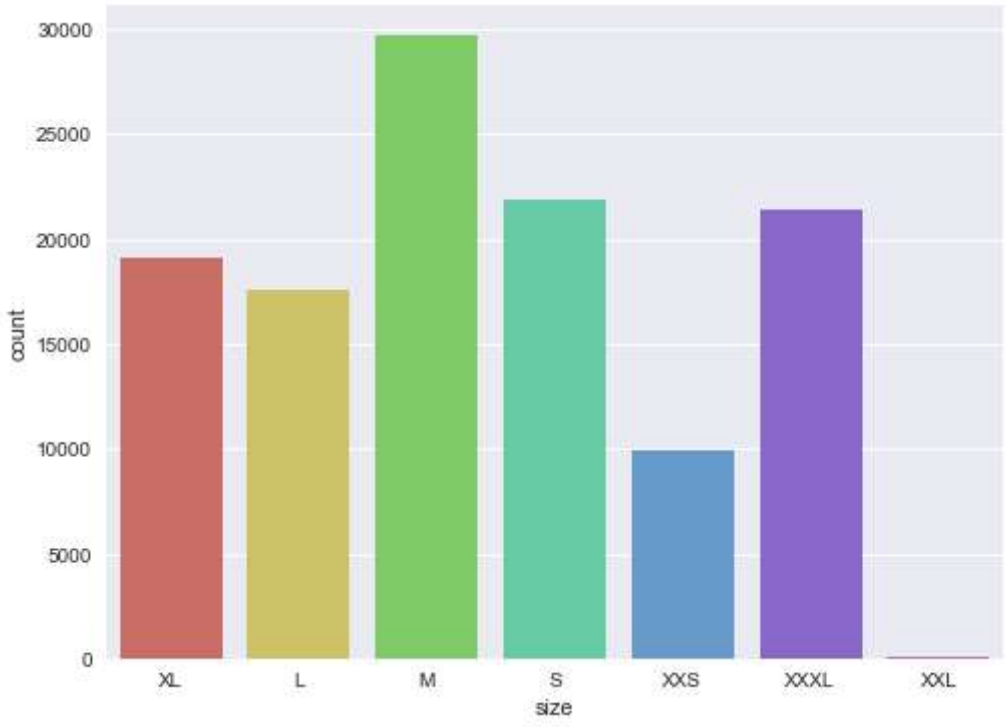
119734 rows × 4 columns

In [29]:
```
# categorial data

#If we want to replace missing values in categorical data, we can replace them wi

# but for my case/senerio it doesn't suits. so i replace them with bfill-backward

# or you can give as df.fillna(method='pad',inplace=True)

df["size"].fillna(method='pad',inplace=True)
df["size"][10:1000:5]
```

Out[29]:
```
10     XXXL
15        L
20        S
25       XL
30        S
       ...
975       M
980       M
985      XL
990       L
995      XL
Name: size, Length: 198, dtype: object
```

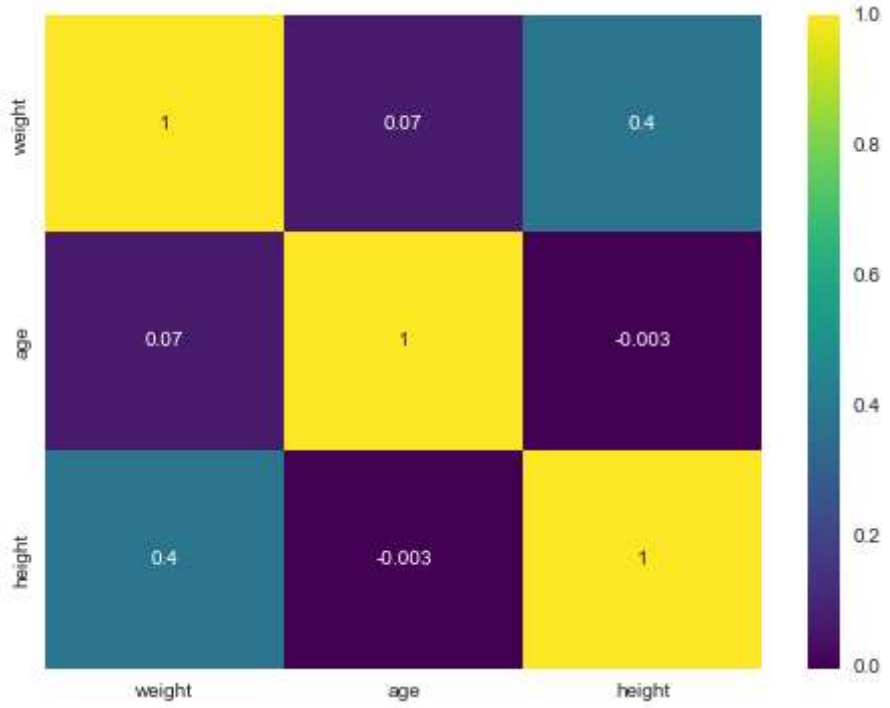
EXPLORATORY DATA ANALYSIS

In [30]:
```
import seaborn as sns
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(8,6))
sns.countplot(x=df["size"], palette="hls");
```



In [31]:
```
# Most size of the cloth used is M and minimum is XXL
```

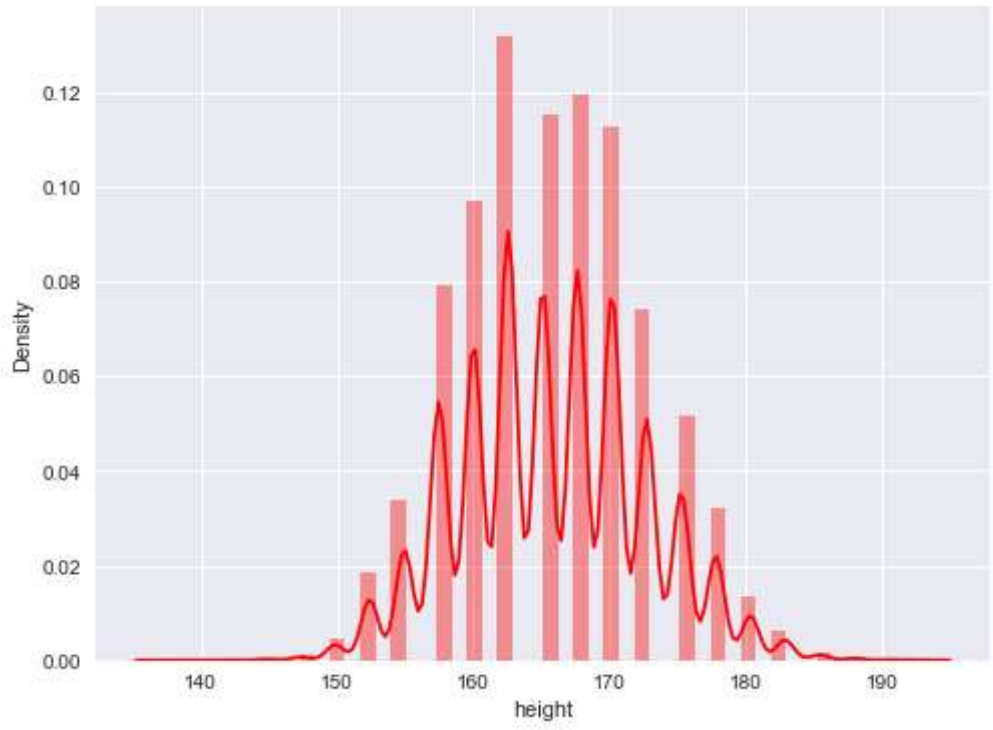
In [32]:
```
fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, fmt='.1g', cmap="viridis",);
```



In [33]:
```
#The clothes size is highly dependent on the weight comapared to height and age.
```

In [34]:
```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(df["height"], color="r");
```

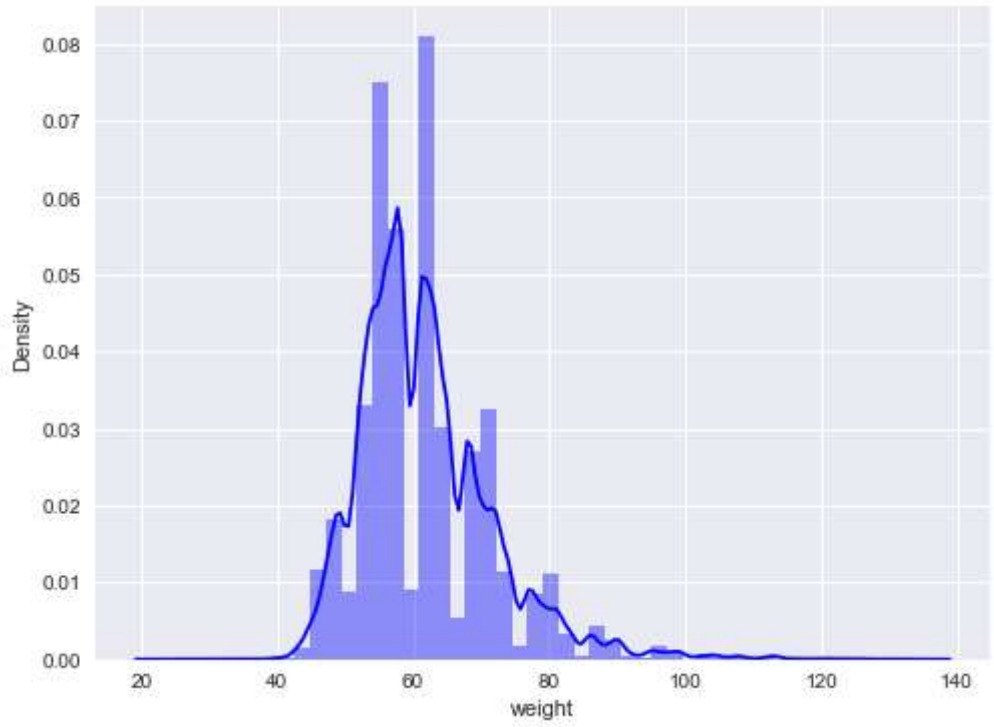
```
D:\dsp-anaconda\envs\tensor\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level funct
ion with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\cbook\__init__.py:140
2: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`)
is deprecated and will be removed in a future version.  Convert to a numpy arra
y before indexing instead.
  ndim = x[:, None].ndim
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:276: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  x = x[:, np.newaxis]
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:278: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  y = y[:, np.newaxis]
```



In [35]:
```
# Most of people have their heights between 150 cm to 180 cm
```

In [36]:
```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(df["weight"], color="b");
```

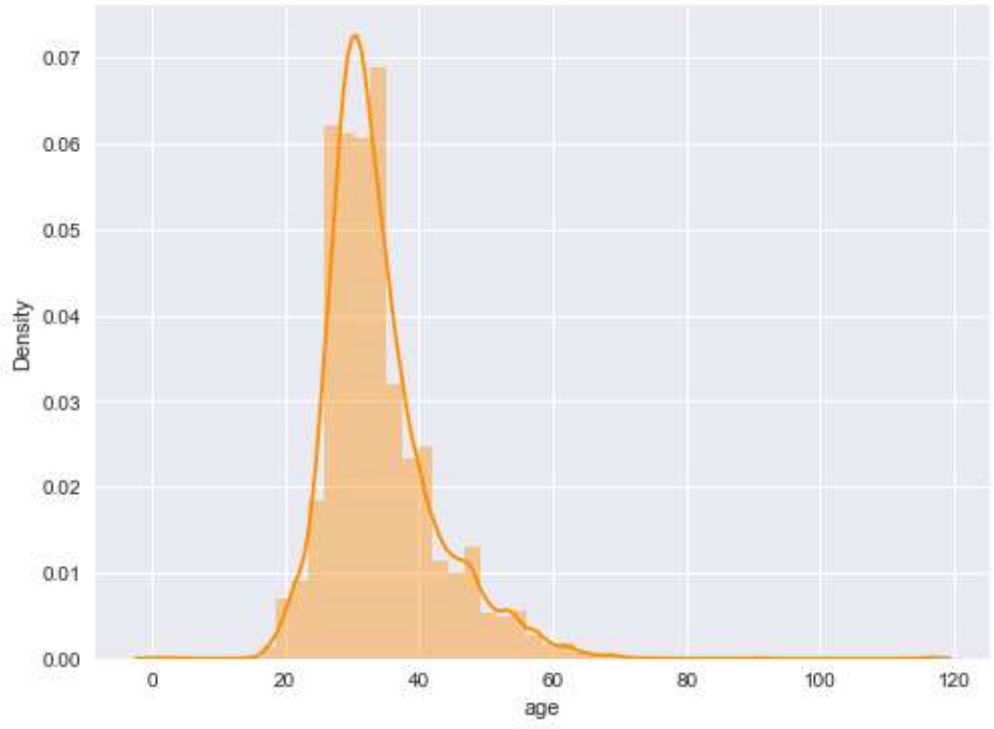
```
D:\dsp-anaconda\envs\tensor\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level funct
ion with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\cbook\__init__.py:140
2: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`)
is deprecated and will be removed in a future version.  Convert to a numpy arra
y before indexing instead.
  ndim = x[:, None].ndim
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:276: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  x = x[:, np.newaxis]
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:278: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  y = y[:, np.newaxis]
```



In [37]:
```
# The average weight appears to be between 40 to 85 kilos
```

In [38]:
```
plt.style.use("seaborn")
fig, ax = plt.subplots(figsize=(8,6))
sns.distplot(df["age"], color="darkorange");
```

```
D:\dsp-anaconda\envs\tensor\lib\site-packages\seaborn\distributions.py:2619: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a futur
e version. Please adapt your code to use either `displot` (a figure-level funct
ion with similar flexibility) or `histplot` (an axes-level function for histogr
ams).
  warnings.warn(msg, FutureWarning)
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\cbook\__init__.py:140
2: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`)
is deprecated and will be removed in a future version.  Convert to a numpy arra
y before indexing instead.
  ndim = x[:, None].ndim
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:276: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  x = x[:, np.newaxis]
D:\dsp-anaconda\envs\tensor\lib\site-packages\matplotlib\axes\_base.py:278: Fut
ureWarning: Support for multi-dimensional indexing (e.g. `obj[:, None]`) is dep
recated and will be removed in a future version.  Convert to a numpy array befo
re indexing instead.
  y = y[:, np.newaxis]
```



In [39]:
```
# An average age of the people seems to be around 20 to 65 years
```

In [40]:
```
df["size"].value_counts()
```

Out[40]:
```
M       29712
S       21924
XXXL    21359
XL      19119
L       17587
XXS      9964
XXL        69
Name: size, dtype: int64
```

FEATURE SELECTION

In [41]:
```
df["BMI"]=0
df
```

Out[41]:

	weight	age	height	size	BMI
0	62	28.0	172.72	XL	0
1	59	36.0	167.64	L	0
2	61	34.0	165.10	M	0
3	65	27.0	175.26	L	0
4	62	45.0	172.72	M	0
...	...	...	...	...	...
119729	63	42.0	175.26	M	0
119730	45	29.0	154.94	S	0
119731	61	31.0	172.72	M	0
119732	74	31.0	167.64	XL	0
119733	70	30.0	167.64	XL	0

119734 rows × 5 columns

In [42]:
```
new=df.head(10000)
a=pd.DataFrame(new)
```

In [43]:
```
a
```

Out[43]:

	weight	age	height	size	BMI
0	62	28.0	172.72	XL	0
1	59	36.0	167.64	L	0
2	61	34.0	165.10	M	0
3	65	27.0	175.26	L	0
4	62	45.0	172.72	M	0
...	...	...	...	...	...
9995	68	38.0	167.64	L	0
9996	70	28.0	172.72	XL	0
9997	61	46.0	165.10	XXXL	0
9998	61	28.0	162.56	M	0
9999	63	21.0	167.64	XL	0

10000 rows × 5 columns

In [44]:
```
for ind , row in a.iterrows():
    a.loc[ind,"BMI"]=row['weight']/(row['height']/100)**2
```

In [45]:
```
a.head(20)
```

Out[45]:

	weight	age	height	size	BMI
0	62	28.0	172.72	XL	20.782914
1	59	36.0	167.64	L	20.994073
2	61	34.0	165.10	M	22.378743
3	65	27.0	175.26	L	21.161563
4	62	45.0	172.72	M	20.782914
5	50	27.0	160.02	S	19.526368
6	53	65.0	160.02	M	20.697950
7	51	33.0	160.02	XXS	19.916895
8	54	26.0	167.64	M	19.214914
9	53	32.0	165.10	S	19.443826
10	63	30.0	170.18	XXXL	21.753218
11	77	35.0	172.72	XXXL	25.811038
12	64	26.0	165.10	L	23.479337
13	52	28.0	160.02	M	20.307423
14	65	33.0	165.10	L	23.846202
15	63	30.0	167.64	L	22.417400
16	54	21.0	167.64	XXS	19.214914
17	63	27.0	172.72	M	21.118122
18	63	30.0	167.64	M	22.417400
19	54	20.0	167.64	S	19.214914

TRAINING THE DATASET

INCORPORATING MACHINE LEARNING ALGORITHM

KNN-ALGORITHM

In [46]:
```
# To use KNN algorithm we need to find euclidean distance. for that create a colu

#formula:- square root of the sum of the difference between a new point x & excis
```

In [47]:
```
# here new points are (50,163)
d=pd.read_csv(r"C:\Users\PUJA BEHARA\Downloads\final_test.csv",header=0)
new=pd.DataFrame(d)
new.head()
```

Out[47]:

	weight	age	height	size	euclidean_distance
0	62	28.0	172.72	XL	15.442746
1	59	36.0	167.64	L	10.125690
2	61	34.0	165.10	M	11.198661
3	65	27.0	175.26	L	19.372857
4	62	45.0	172.72	M	15.442746

In [48]:
```
new=new.head(10000)
```

In [49]:
```
new["euclidean_distance"].fillna(method='pad',inplace=True)
```

In [50]:
```
#by doing this we got nearest value to our new point and they are in ascending or
new=new.sort_values("euclidean_distance",ascending=True)
new
```

Out[50]:

	weight	age	height	size	euclidean_distance
299	50	32.0	162.56	XXS	0.440000
250	50	48.0	162.56	XXS	0.440000
488	49	31.0	162.56	S	1.092520
55	49	30.0	162.56	M	1.092520
351	52	28.0	162.56	S	2.047828
...	...	...	...	...	...
418	97	43.0	162.56	XXXL	47.002060
98	99	44.0	172.72	XXXL	49.954764
77	104	34.0	154.94	XXXL	54.598201
73	113	27.0	167.64	XXXL	63.170639
233	112	38.0	175.26	XXXL	63.200535

10000 rows × 5 columns

In [51]:
```
# k value = 11
new=new.head(11)
new
```

Out[51]:

	weight	age	height	size	euclidean_distance
299	50	32.0	162.56	XXS	0.440000
250	50	48.0	162.56	XXS	0.440000
488	49	31.0	162.56	S	1.092520
55	49	30.0	162.56	M	1.092520
351	52	28.0	162.56	S	2.047828
502	52	50.0	162.56	S	2.047828
209	50	28.0	165.10	S	2.100000
280	52	30.0	165.10	S	2.900000
156	52	42.0	165.10	S	2.900000
192	50	52.0	160.02	L	2.980000
23	50	43.0	160.02	M	2.980000

In [52]:
```
# Test
# now we have to find which size is occured most of time
new['size'].value_counts()
```

Out[52]:
```
S      6
XXS    2
M      2
L      1
Name: size, dtype: int64
```

In [53]:
```
# here it says that our customer size is S
#if we increase k value for a range it gives same result. but by increasing the k
```

In [54]:
```
SIZE = new["size"]
```

In [55]:
```
SIZE
```

Out[55]:
```
299    XXS
250    XXS
488      S
55       M
351      S
502      S
209      S
280      S
156      S
192      L
23       M
Name: size, dtype: object
```

In [56]:
```
Scount=0
Mcount=0
Lcount=0
XLcount=0
XXLcount=0
XXXLcount=0
XXScount=0

for i in SIZE:
    if(i=='S'):
        Scount=Scount+1
    elif(i=='M'):
        Mcount=Mcount+1
    elif(i=='L'):
        Lcount=Lcount+1
    elif(i=='XL'):
        XLcount=XLcount+1
    elif(i=='XXL'):
        XXLcount=XXLcount+1
    elif(i=='XXXL'):
        XXXLcount=XXXLcount+1
    elif(i=='XXS'):
        XXScount=XXScount+1
    else:
        print("we didn't sale kidsware clothes")
```

In [57]:
```
if Scount > Mcount and Scount > Lcount and Scount > XLcount and Scount > XXLcount
    print("your size is S")
elif Mcount > Scount and Mcount > Lcount and Mcount > XLcount and Mcount > XXLcou
    print("your size is M")
elif Lcount > Scount and Lcount > Mcount and Lcount > XLcount and Lcount > XXLcou
    print("your size is L")
elif XLcount > Scount and XLcount > Mcount and XLcount > Lcount and XLcount > XXL
    print("your size is XL")
if XXLcount > Scount and XXLcount > Mcount and XXLcount > Lcount and XXLcount > X
    print("your size is XXL")
elif XXXLcount > Scount and XXXLcount > Mcount and XXXLcount > Lcount and XXXLcou
    print("your size is XXXL")
elif XXScount > Scount and XXScount > Mcount and XXScount > Lcount and XXScount >
    print("your size is XXS")
```

```
your size is S
```

In [58]:
```
# here it says that our customer size is S
#if we increase k value for a range it gives same result. but by increasing the k
```

In [59]:
```
# here i will do some calculations regarding health by using height and weight of
```

In [61]:
```
height = float(input("Enter your height in cm: "))
weight = float(input("Enter your weight in kg: "))
```

```
Enter your height in cm: 165
Enter your weight in kg: 60
```

In [62]:
```
BMI = weight / (height/100)**2
```

In [63]:
```
print(f"Your BMI is {BMI}")
```

```
Your BMI is 22.03856749311295
```

In [64]:
```
if BMI <= 18.4:
    print("You are underweight.")
elif BMI <= 24.9:
    print("You are healthy.")
elif BMI <= 29.9:
    print("You are over weight.")
elif BMI <= 34.9:
    print("You are severely over weight.")
elif BMI <= 39.9:
    print("You are obese.")
else:
    print("You are severely obese.")
```

```
You are healthy.
```

In [65]:
```
# K Nearest Neighbor Classifier
```

In [66]:
```
a=pd.DataFrame(df)
```

In [67]:
```
a
```

Out[67]:

	weight	age	height	size	BMI
0	62	28.0	172.72	XL	0
1	59	36.0	167.64	L	0
2	61	34.0	165.10	M	0
3	65	27.0	175.26	L	0
4	62	45.0	172.72	M	0
...	...	...	...	...	...
119729	63	42.0	175.26	M	0
119730	45	29.0	154.94	S	0
119731	61	31.0	172.72	M	0
119732	74	31.0	167.64	XL	0
119733	70	30.0	167.64	XL	0

119734 rows × 5 columns

In [68]:
```
xi=float(input("enter your weight: "))
yi=float(input("enter your height: "))
```

```
enter your weight: 55
enter your height: 165
```

In [69]:
```
import math
sum=0
row=0
a['Euclidean_Distance']=[0 for i in range(len(a))]
for x,y in zip(a['weight'],a['height']):
    sum=(x-xi)**2+(y-yi)**2
    s=math.sqrt(sum)
    a['Euclidean_Distance'][row]=s
    row=row+1
```

```
D:\dsp-anaconda\envs\tensor\lib\site-packages\ipykernel_launcher.py:8: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
```

In [70]:
```
a=a.sort_values(by=['Euclidean_Distance'],ascending=True)
a
```

Out[70]:

	weight	age	height	size	BMI	Euclidean_Distance
48040	55	37.0	165.10	M	0	0
69886	55	26.0	165.10	M	0	0
117691	55	29.0	165.10	S	0	0
74576	55	25.0	165.10	L	0	0
5830	55	33.0	165.10	S	0	0
...	...	...	...	...	...	...
46591	136	36.0	167.64	XXXL	0	81
88415	136	33.0	170.18	XXXL	0	81
19669	136	31.0	175.26	XXXL	0	81
86319	136	37.0	165.10	XXXL	0	81
12994	136	29.0	177.80	XXXL	0	82

119734 rows × 6 columns

In [71]:
```
a=a.head(10)
a
```

Out[71]:

	weight	age	height	size	BMI	Euclidean_Distance
48040	55	37.0	165.1	M	0	0
69886	55	26.0	165.1	M	0	0
117691	55	29.0	165.1	S	0	0
74576	55	25.0	165.1	L	0	0
5830	55	33.0	165.1	S	0	0
86545	55	55.0	165.1	L	0	0
93078	55	28.0	165.1	M	0	0
11583	55	39.0	165.1	M	0	0
60526	55	30.0	165.1	M	0	0
113672	55	26.0	165.1	XXS	0	0

In [72]:
```
# Test part
```

In [73]:
```
a['size'].value_counts()
```

Out[73]:
```
M      5
S      2
L      2
XXS    1
Name: size, dtype: int64
```

In [75]:
```
# here we got highest value for M. so the unknown customer cloth size is 'M'
```

DECISION TREE - ALGORITHM

In [76]:
```
df
```

Out[76]:

	weight	age	height	size	BMI	Euclidean_Distance
0	62	28.0	172.72	XL	0	10
1	59	36.0	167.64	L	0	4
2	61	34.0	165.10	M	0	6
3	65	27.0	175.26	L	0	14
4	62	45.0	172.72	M	0	10
...	...	...	...	...	...	...
119729	63	42.0	175.26	M	0	13
119730	45	29.0	154.94	S	0	14
119731	61	31.0	172.72	M	0	9
119732	74	31.0	167.64	XL	0	19
119733	70	30.0	167.64	XL	0	15

119734 rows × 6 columns

In [77]:
```
df=df.head(10000)
df
```

Out[77]:

	weight	age	height	size	BMI	Euclidean_Distance
0	62	28.0	172.72	XL	0	10
1	59	36.0	167.64	L	0	4
2	61	34.0	165.10	M	0	6
3	65	27.0	175.26	L	0	14
4	62	45.0	172.72	M	0	10
...	...	...	...	...	...	...
9995	68	38.0	167.64	L	0	13
9996	70	28.0	172.72	XL	0	16
9997	61	46.0	165.10	XXXL	0	6
9998	61	28.0	162.56	M	0	6
9999	63	21.0	167.64	XL	0	8

10000 rows × 6 columns

In [78]:
```
print("This model is suitable for above 16 ys peoples. So by giving the inputs  o
print("enter the values for weight above 39 and below 114")
print("enter the values for age between 16-70")
print("enter the values for height between 143-188 ")
print("otherwise you will get a incorrect results")
```

```
This model is suitable for above 16 ys peoples. So by giving the inputs  once s
ee this chart
enter the values for weight above 39 and below 114
enter the values for age between 16-70
enter the values for height between 143-188
otherwise you will get a incorrect results
```

In [79]:
```
inputs=df[["weight","age","height"]]
target=df["size"]
```

MODEL EVALUTION

In [80]:
```
from sklearn import tree
model=tree.DecisionTreeClassifier()
model.fit(inputs,target)
```

Out[80]:
```
DecisionTreeClassifier()
```

In [81]:
```
model.score(inputs,target)
```

Out[81]:
```
0.7316
```

MODEL PREDICTION

In [82]:
```
model.predict([[70,24,155]])
```

```
D:\dsp-anaconda\envs\tensor\lib\site-packages\sklearn\base.py:451: UserWarning:
X does not have valid feature names, but DecisionTreeClassifier was fitted with
feature names
  "X does not have valid feature names, but"
```

Out[82]:
```
array(['L'], dtype=object)
```

In [83]:
```
# here we got cloth size is 'L' for for those input sizes of a person
```

In [84]:
```
model.predict([[80,25,185]])
```

```
D:\dsp-anaconda\envs\tensor\lib\site-packages\sklearn\base.py:451: UserWarning:
X does not have valid feature names, but DecisionTreeClassifier was fitted with
feature names
  "X does not have valid feature names, but"
```

Out[84]:
```
array(['XXXL'], dtype=object)
```

In [85]:
```
w=float(input("enter a value of weight: "))
a=float(input("enter a value of age: "))
h=float(input("enter a value of height: "))
```

```
enter a value of weight: 55
enter a value of age: 20
enter a value of height: 155
```

In [88]:
```
if w < 39 or w > 114:
    print("It is developed based above 39 yrs and below 114 yrs peoples. so enter
elif a < 16 or a > 70:
    print("It is developed based above 16 yrs and below 70 yrs peoples. so enter
elif h < 143 or h > 188:
    print("It is developed based on above 16 yrs old and below 70 yrs peoples. so
else:
    p= model.predict([[w,a,h]])
```

```
D:\dsp-anaconda\envs\tensor\lib\site-packages\sklearn\base.py:451: UserWarning:
X does not have valid feature names, but DecisionTreeClassifier was fitted with
feature names
  "X does not have valid feature names, but"
```

In [89]:
```
print(p)
```

```
['S']
```

In [ ]:
```
# like this we can predict the cloth size of a person.
```

In [ ]:
```

```

In [ ]:
```

```