

Exploratory Data Analysis

September 28, 2022

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
[4]: data=pd.read_csv('Travel.csv')
```

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

```
[14]: CustomerID  ProdTaken  Age  TypeofContact  CityTier  DurationOfPitch  \
0      200000      1  41.0    Self Enquiry      3          6.0
1      200001      0  49.0  Company Invited      1         14.0
2      200002      1  37.0    Self Enquiry      1          8.0
3      200003      0  33.0  Company Invited      1          9.0
4      200004      0   NaN    Self Enquiry      1          8.0
```

```
Occupation  Gender  NumberOfPersonVisiting  NumberOfFollowups  \
0    Salaried  Female              3              3.0
1    Salaried   Male              3              4.0
2  Free Lancer   Male              3              4.0
3    Salaried  Female              2              3.0
4 Small Business   Male              2              3.0
```

```
ProductPitched  PreferredPropertyStar  MaritalStatus  NumberOfTrips  \
0      Deluxe              3.0      Single          1.0
1      Deluxe              4.0    Divorced          2.0
2      Basic              3.0      Single          7.0
3      Basic              3.0    Divorced          2.0
4      Basic              4.0    Divorced          1.0
```

```
Passport  PitchSatisfactionScore  OwnCar  NumberOfChildrenVisiting  \
0      1              2      1          0.0
1      0              3      1          2.0
2      1              3      0          0.0
```

3	1	5	1	1.0
4	0	5	1	0.0

	Designation	MonthlyIncome
0	Manager	20993.0
1	Manager	20130.0
2	Executive	17090.0
3	Executive	17909.0
4	Executive	18468.0

```
[15]: data.columns
```

```
[15]: Index(['CustomerID', 'ProdTaken', 'Age', 'TypeofContact', 'CityTier',
          'DurationOfPitch', 'Occupation', 'Gender', 'NumberOfPersonVisiting',
          'NumberOfFollowups', 'ProductPitched', 'PreferredPropertyStar',
          'MaritalStatus', 'NumberOfTrips', 'Passport', 'PitchSatisfactionScore',
          'OwnCar', 'NumberOfChildrenVisiting', 'Designation', 'MonthlyIncome'],
          dtype='object')
```

```
[25]: data.shape # indicates the number of rows,columns
```

```
[25]: (4888, 20)
```

```
[26]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   CustomerID                           4888 non-null   int64
1   ProdTaken                            4888 non-null   int64
2   Age                                   4662 non-null   float64
3   TypeofContact                        4863 non-null   object
4   CityTier                             4888 non-null   int64
5   DurationOfPitch                      4637 non-null   float64
6   Occupation                           4888 non-null   object
7   Gender                               4888 non-null   object
8   NumberOfPersonVisiting               4888 non-null   int64
9   NumberOfFollowups                    4843 non-null   float64
10  ProductPitched                       4888 non-null   object
11  PreferredPropertyStar                 4862 non-null   float64
12  MaritalStatus                        4888 non-null   object
13  NumberOfTrips                        4748 non-null   float64
14  Passport                             4888 non-null   int64
15  PitchSatisfactionScore                4888 non-null   int64
16  OwnCar                               4888 non-null   int64
```

```

17 NumberOfChildrenVisiting 4822 non-null float64
18 Designation               4888 non-null object
19 MonthlyIncome             4655 non-null float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB

```

```
[36]: data.describe().T # Completely based on Integer or numerical features
```

```
[36]:
```

	count	mean	std	min \
CustomerID	4888.0	202443.500000	1411.188388	200000.0
ProdTaken	4888.0	0.188216	0.390925	0.0
Age	4662.0	37.622265	9.316387	18.0
CityTier	4888.0	1.654255	0.916583	1.0
DurationOfPitch	4637.0	15.490835	8.519643	5.0
NumberOfPersonVisiting	4888.0	2.905074	0.724891	1.0
NumberOfFollowups	4843.0	3.708445	1.002509	1.0
PreferredPropertyStar	4862.0	3.581037	0.798009	3.0
NumberOfTrips	4748.0	3.236521	1.849019	1.0
Passport	4888.0	0.290917	0.454232	0.0
PitchSatisfactionScore	4888.0	3.078151	1.365792	1.0
OwnCar	4888.0	0.620295	0.485363	0.0
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0

	25%	50%	75%	max
CustomerID	201221.75	202443.5	203665.25	204887.0
ProdTaken	0.00	0.0	0.00	1.0
Age	31.00	36.0	44.00	61.0
CityTier	1.00	1.0	3.00	3.0
DurationOfPitch	9.00	13.0	20.00	127.0
NumberOfPersonVisiting	2.00	3.0	3.00	5.0
NumberOfFollowups	3.00	4.0	4.00	6.0
PreferredPropertyStar	3.00	3.0	4.00	5.0
NumberOfTrips	2.00	3.0	4.00	22.0
Passport	0.00	0.0	1.00	1.0
PitchSatisfactionScore	2.00	3.0	4.00	5.0
OwnCar	0.00	1.0	1.00	1.0
NumberOfChildrenVisiting	1.00	1.0	2.00	3.0
MonthlyIncome	20346.00	22347.0	25571.00	98678.0

```
[20]: # Missing Values
data.isnull().sum()
```

```
[20]: CustomerID      0
ProdTaken           0
Age                226
TypeofContact      25
```

```

CityTier                0
DurationOfPitch          251
Occupation               0
Gender                  0
NumberOfPersonVisiting  0
NumberOfFollowups        45
ProductPitched           0
PreferredPropertyStar    26
MaritalStatus            0
NumberOfTrips            140
Passport                 0
PitchSatisfactionScore   0
OwnCar                   0
NumberOfChildrenVisiting 66
Designation              0
MonthlyIncome            233
dtype: int64

```

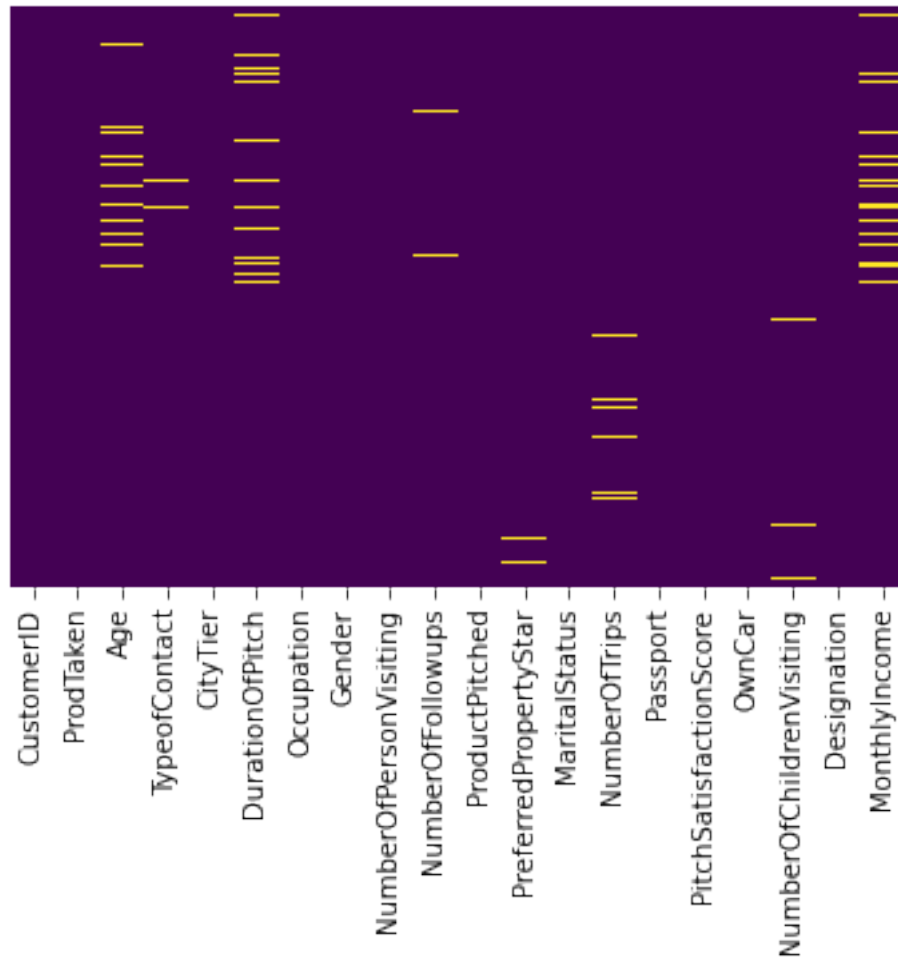
```
[22]: [features for features in data.columns if data[features].isnull().sum()>0]
```

```
[22]: ['Age',
      'TypeofContact',
      'DurationOfPitch',
      'NumberOfFollowups',
      'PreferredPropertyStar',
      'NumberOfTrips',
      'NumberOfChildrenVisiting',
      'MonthlyIncome']

```

```
[28]: sns.heatmap(data.isnull(),yticklabels=False,cbar=False,cmap="viridis") #
      ↪Graphically indicates the missing values
```

```
[28]: <AxesSubplot:>
```



```
[31]: cat_col=[fea for fea in data.columns if data[fea].dtype == 'O'] #segregating_
      ↪ categorical and numerical variables
      cat_col
```

```
[31]: ['TypeofContact',
      'Occupation',
      'Gender',
      'ProductPitched',
      'MaritalStatus',
      'Designation']
```

```
[32]: num_col=[fea for fea in data.columns if data[fea].dtype != 'O'] #segregating_
      ↪ categorical and numerical variables
      num_col
```

```
[32]: ['CustomerID',
      'ProdTaken',
      'Age',
      'CityTier',
      'DurationOfPitch',
      'NumberOfPersonVisiting',
      'NumberOfFollowups',
      'PreferredPropertyStar',
      'NumberOfTrips',
      'Passport',
      'PitchSatisfactionScore',
      'OwnCar',
      'NumberOfChildrenVisiting',
      'MonthlyIncome']
```

```
[33]: data[cat_col]
```

```
[33]:      TypeofContact      Occupation  Gender  ProductPitched  MaritalStatus  \
0      Self Enquiry      Salaried  Female      Deluxe      Single
1      Company Invited      Salaried   Male      Deluxe      Divorced
2      Self Enquiry      Free Lancer   Male      Basic      Single
3      Company Invited      Salaried  Female      Basic      Divorced
4      Self Enquiry  Small Business   Male      Basic      Divorced
...      ...      ...      ...      ...      ...
4883     Self Enquiry  Small Business   Male      Deluxe      Unmarried
4884  Company Invited      Salaried   Male      Basic      Single
4885     Self Enquiry      Salaried  Female      Standard      Married
4886     Self Enquiry  Small Business   Male      Basic      Single
4887     Self Enquiry      Salaried   Male      Basic      Unmarried

      Designation
0      Manager
1      Manager
2      Executive
3      Executive
4      Executive
...      ...
4883      Manager
4884      Executive
4885  Senior Manager
4886      Executive
4887      Executive

[4888 rows x 6 columns]
```

```
[34]: data[num_col]
```

[34]:

	CustomerID	ProdTaken	Age	CityTier	DurationOfPitch	\
0	200000	1	41.0	3	6.0	
1	200001	0	49.0	1	14.0	
2	200002	1	37.0	1	8.0	
3	200003	0	33.0	1	9.0	
4	200004	0	NaN	1	8.0	
...	
4883	204883	1	49.0	3	9.0	
4884	204884	1	28.0	1	31.0	
4885	204885	1	52.0	3	17.0	
4886	204886	1	19.0	3	16.0	
4887	204887	1	36.0	1	14.0	

	NumberOfPersonVisiting	NumberOfFollowups	PreferredPropertyStar	\
0	3	3.0	3.0	
1	3	4.0	4.0	
2	3	4.0	3.0	
3	2	3.0	3.0	
4	2	3.0	4.0	
...	
4883	3	5.0	4.0	
4884	4	5.0	3.0	
4885	4	4.0	4.0	
4886	3	4.0	3.0	
4887	4	4.0	4.0	

	NumberOfTrips	Passport	PitchSatisfactionScore	OwnCar	\
0	1.0	1	2	1	
1	2.0	0	3	1	
2	7.0	1	3	0	
3	2.0	1	5	1	
4	1.0	0	5	1	
...	
4883	2.0	1	1	1	
4884	3.0	1	3	1	
4885	7.0	0	1	1	
4886	3.0	0	5	0	
4887	3.0	1	3	1	

	NumberOfChildrenVisiting	MonthlyIncome
0	0.0	20993.0
1	2.0	20130.0
2	0.0	17090.0
3	1.0	17909.0
4	0.0	18468.0
...
4883	1.0	26576.0

4884	2.0	21212.0
4885	3.0	31820.0
4886	2.0	20289.0
4887	2.0	24041.0

[4888 rows x 14 columns]

```
[47]: numeric_data_1=data[num_col]
      numeric_data_1
```

```
[47]:
```

	CustomerID	ProdTaken	Age	CityTier	DurationOfPitch	\
0	200000	1	41.0	3	6.0	
1	200001	0	49.0	1	14.0	
2	200002	1	37.0	1	8.0	
3	200003	0	33.0	1	9.0	
4	200004	0	NaN	1	8.0	
...	
4883	204883	1	49.0	3	9.0	
4884	204884	1	28.0	1	31.0	
4885	204885	1	52.0	3	17.0	
4886	204886	1	19.0	3	16.0	
4887	204887	1	36.0	1	14.0	

	NumberOfPersonVisiting	NumberOfFollowups	PreferredPropertyStar	\
0	3	3.0	3.0	
1	3	4.0	4.0	
2	3	4.0	3.0	
3	2	3.0	3.0	
4	2	3.0	4.0	
...	
4883	3	5.0	4.0	
4884	4	5.0	3.0	
4885	4	4.0	4.0	
4886	3	4.0	3.0	
4887	4	4.0	4.0	

	NumberOfTrips	Passport	PitchSatisfactionScore	OwnCar	\
0	1.0	1	2	1	
1	2.0	0	3	1	
2	7.0	1	3	0	
3	2.0	1	5	1	
4	1.0	0	5	1	
...	
4883	2.0	1	1	1	
4884	3.0	1	3	1	
4885	7.0	0	1	1	
4886	3.0	0	5	0	


```
4887          3.0          1          3          1
```

```

      NumberOfChildrenVisiting  MonthlyIncome
0                             0.0         20993.0
1                             2.0         20130.0
2                             0.0         17090.0
3                             1.0         17909.0
4                             0.0         18468.0
...
4883                          1.0         26576.0
4884                          2.0         21212.0
4885                          3.0         31820.0
4886                          2.0         20289.0
4887                          2.0         24041.0

```

```
[4888 rows x 14 columns]
```

```
[48]: numeric_data_1 = numeric_data_1.drop('CustomerID',axis=1) # dropping Customer ID
```

```
[49]: numeric_data_1.describe().T
```

```
[49]:
```

	count	mean	std	min	25%	\
ProdTaken	4888.0	0.188216	0.390925	0.0	0.0	
Age	4662.0	37.622265	9.316387	18.0	31.0	
CityTier	4888.0	1.654255	0.916583	1.0	1.0	
DurationOfPitch	4637.0	15.490835	8.519643	5.0	9.0	
NumberOfPersonVisiting	4888.0	2.905074	0.724891	1.0	2.0	
NumberOfFollowups	4843.0	3.708445	1.002509	1.0	3.0	
PreferredPropertyStar	4862.0	3.581037	0.798009	3.0	3.0	
NumberOfTrips	4748.0	3.236521	1.849019	1.0	2.0	
Passport	4888.0	0.290917	0.454232	0.0	0.0	
PitchSatisfactionScore	4888.0	3.078151	1.365792	1.0	2.0	
OwnCar	4888.0	0.620295	0.485363	0.0	0.0	
NumberOfChildrenVisiting	4822.0	1.187267	0.857861	0.0	1.0	
MonthlyIncome	4655.0	23619.853491	5380.698361	1000.0	20346.0	

	50%	75%	max
ProdTaken	0.0	0.0	1.0
Age	36.0	44.0	61.0
CityTier	1.0	3.0	3.0
DurationOfPitch	13.0	20.0	127.0
NumberOfPersonVisiting	3.0	3.0	5.0
NumberOfFollowups	4.0	4.0	6.0
PreferredPropertyStar	3.0	4.0	5.0
NumberOfTrips	3.0	4.0	22.0
Passport	0.0	1.0	1.0
PitchSatisfactionScore	3.0	4.0	5.0

OwnCar	1.0	1.0	1.0
NumberOfChildrenVisiting	1.0	2.0	3.0
MonthlyIncome	22347.0	25571.0	98678.0

```
[55]: numeric_data_1.corr() # checking correlation matrix
```

```
[55]:
```

	ProdTaken	Age	CityTier	DurationOfPitch	\
ProdTaken	1.000000	-0.147254	0.086852	0.078257	
Age	-0.147254	1.000000	-0.015625	-0.012063	
CityTier	0.086852	-0.015625	1.000000	0.022703	
DurationOfPitch	0.078257	-0.012063	0.022703	1.000000	
NumberOfPersonVisiting	0.009627	0.011621	-0.001671	0.065141	
NumberOfFollowups	0.112171	-0.002577	0.023652	0.009434	
PreferredPropertyStar	0.099577	-0.010474	-0.009164	-0.006637	
NumberOfTrips	0.018898	0.184905	-0.029709	0.009715	
Passport	0.260844	0.033399	0.001793	0.033034	
PitchSatisfactionScore	0.051394	0.018510	-0.042160	-0.002880	
OwnCar	-0.011508	0.048654	0.003817	-0.001626	
NumberOfChildrenVisiting	0.007421	0.007370	0.000672	0.031408	
MonthlyIncome	-0.130585	0.464869	0.051817	-0.006252	

	NumberOfPersonVisiting	NumberOfFollowups	\
ProdTaken	0.009627	0.112171	
Age	0.011621	-0.002577	
CityTier	-0.001671	0.023652	
DurationOfPitch	0.065141	0.009434	
NumberOfPersonVisiting	1.000000	0.328569	
NumberOfFollowups	0.328569	1.000000	
PreferredPropertyStar	0.033867	-0.024176	
NumberOfTrips	0.195223	0.139517	
Passport	0.011177	0.004970	
PitchSatisfactionScore	-0.019581	0.004054	
OwnCar	0.010362	0.012112	
NumberOfChildrenVisiting	0.610621	0.286425	
MonthlyIncome	0.195134	0.176503	

	PreferredPropertyStar	NumberOfTrips	Passport	\
ProdTaken	0.099577	0.018898	0.260844	
Age	-0.010474	0.184905	0.033399	
CityTier	-0.009164	-0.029709	0.001793	
DurationOfPitch	-0.006637	0.009715	0.033034	
NumberOfPersonVisiting	0.033867	0.195223	0.011177	
NumberOfFollowups	-0.024176	0.139517	0.004970	
PreferredPropertyStar	1.000000	0.012115	0.001040	
NumberOfTrips	0.012115	1.000000	0.012949	
Passport	0.001040	0.012949	1.000000	
PitchSatisfactionScore	-0.022701	-0.004378	0.002926	

OwnCar	0.015742	-0.011825	-0.022330
NumberOfChildrenVisiting	0.035798	0.168795	0.020264
MonthlyIncome	0.014289	0.139105	0.002545

	PitchSatisfactionScore	OwnCar	\
ProdTaken	0.051394	-0.011508	
Age	0.018510	0.048654	
CityTier	-0.042160	0.003817	
DurationOfPitch	-0.002880	-0.001626	
NumberOfPersonVisiting	-0.019581	0.010362	
NumberOfFollowups	0.004054	0.012112	
PreferredPropertyStar	-0.022701	0.015742	
NumberOfTrips	-0.004378	-0.011825	
Passport	0.002926	-0.022330	
PitchSatisfactionScore	1.000000	0.068850	
OwnCar	0.068850	1.000000	
NumberOfChildrenVisiting	0.000878	0.026572	
MonthlyIncome	0.030421	0.080262	

	NumberOfChildrenVisiting	MonthlyIncome
ProdTaken	0.007421	-0.130585
Age	0.007370	0.464869
CityTier	0.000672	0.051817
DurationOfPitch	0.031408	-0.006252
NumberOfPersonVisiting	0.610621	0.195134
NumberOfFollowups	0.286425	0.176503
PreferredPropertyStar	0.035798	0.014289
NumberOfTrips	0.168795	0.139105
Passport	0.020264	0.002545
PitchSatisfactionScore	0.000878	0.030421
OwnCar	0.026572	0.080262
NumberOfChildrenVisiting	1.000000	0.201643
MonthlyIncome	0.201643	1.000000

```
[56]: numeric_data_1.cov() # checking covariance matrix
```

```
[56]:
```

	ProdTaken	Age	CityTier	\
ProdTaken	0.152822	-0.535959	0.031120	
Age	-0.535959	86.795067	-0.134168	
CityTier	0.031120	-0.134168	0.840125	
DurationOfPitch	0.260897	-0.944440	0.176544	
NumberOfPersonVisiting	0.002728	0.078126	-0.001110	
NumberOfFollowups	0.043969	-0.024058	0.021716	
PreferredPropertyStar	0.031050	-0.077447	-0.006707	
NumberOfTrips	0.013743	3.158372	-0.050474	
Passport	0.046318	0.141269	0.000747	
PitchSatisfactionScore	0.027440	0.235235	-0.052778	

OwnCar	-0.002184	0.220050	0.001698
NumberOfChildrenVisiting	0.002494	0.058470	0.000529
MonthlyIncome	-276.097744	23365.183349	255.945958

	DurationOfPitch	NumberOfPersonVisiting	\
ProdTaken	0.260897	0.002728	
Age	-0.944440	0.078126	
CityTier	0.176544	-0.001110	
DurationOfPitch	72.584310	0.401434	
NumberOfPersonVisiting	0.401434	0.525466	
NumberOfFollowups	0.080699	0.238479	
PreferredPropertyStar	-0.045149	0.019575	
NumberOfTrips	0.154174	0.261809	
Passport	0.128118	0.003680	
PitchSatisfactionScore	-0.033569	-0.019386	
OwnCar	-0.006731	0.003646	
NumberOfChildrenVisiting	0.229346	0.380176	
MonthlyIncome	-289.182796	756.518793	

	NumberOfFollowups	PreferredPropertyStar	\
ProdTaken	0.043969	0.031050	
Age	-0.024058	-0.077447	
CityTier	0.021716	-0.006707	
DurationOfPitch	0.080699	-0.045149	
NumberOfPersonVisiting	0.238479	0.019575	
NumberOfFollowups	1.005024	-0.019309	
PreferredPropertyStar	-0.019309	0.636818	
NumberOfTrips	0.259783	0.017840	
Passport	0.002265	0.000377	
PitchSatisfactionScore	0.005550	-0.024736	
OwnCar	0.005894	0.006102	
NumberOfChildrenVisiting	0.246696	0.024441	
MonthlyIncome	958.027680	58.848919	

	NumberOfTrips	Passport	PitchSatisfactionScore	\
ProdTaken	0.013743	0.046318	0.027440	
Age	3.158372	0.141269	0.235235	
CityTier	-0.050474	0.000747	-0.052778	
DurationOfPitch	0.154174	0.128118	-0.033569	
NumberOfPersonVisiting	0.261809	0.003680	-0.019386	
NumberOfFollowups	0.259783	0.002265	0.005550	
PreferredPropertyStar	0.017840	0.000377	-0.024736	
NumberOfTrips	3.418872	0.010882	-0.011035	
Passport	0.010882	0.206326	0.001815	
PitchSatisfactionScore	-0.011035	0.001815	1.865387	
OwnCar	-0.010622	-0.004923	0.045641	
NumberOfChildrenVisiting	0.267269	0.007898	0.001029	

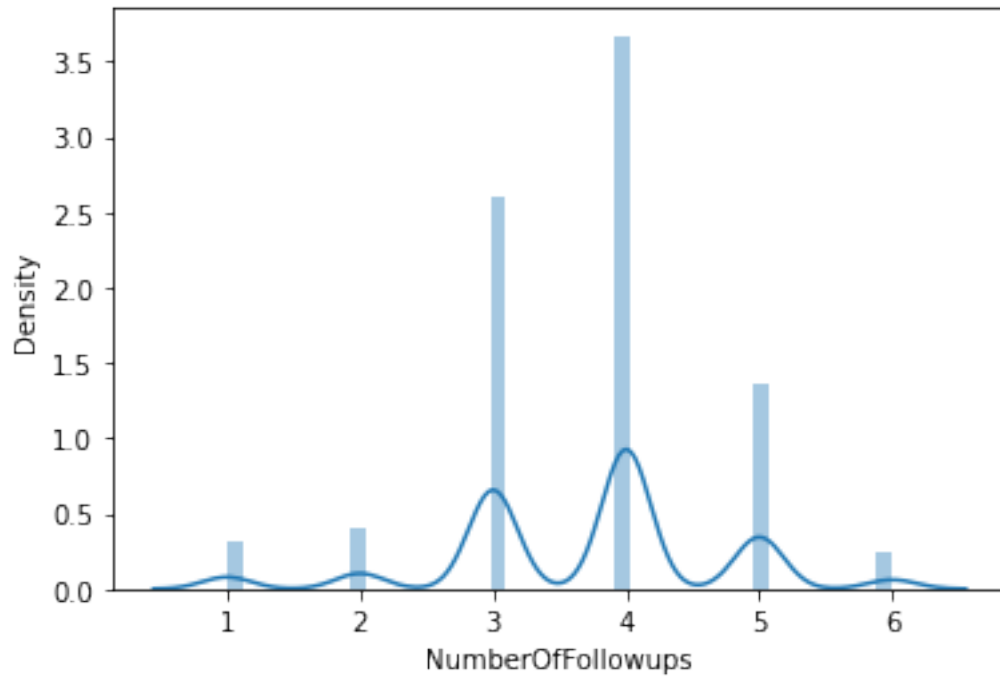
MonthlyIncome	1265.226486	6.230027	223.667867
	OwnCar	NumberOfChildrenVisiting	MonthlyIncome
ProdTaken	-0.002184	0.002494	-2.760977e+02
Age	0.220050	0.058470	2.336518e+04
CityTier	0.001698	0.000529	2.559460e+02
DurationOfPitch	-0.006731	0.229346	-2.891828e+02
NumberOfPersonVisiting	0.003646	0.380176	7.565188e+02
NumberOfFollowups	0.005894	0.246696	9.580277e+02
PreferredPropertyStar	0.006102	0.024441	5.884892e+01
NumberOfTrips	-0.010622	0.267269	1.265226e+03
Passport	-0.004923	0.007898	6.230027e+00
PitchSatisfactionScore	0.045641	0.001029	2.236679e+02
OwnCar	0.235577	0.011073	2.095549e+02
NumberOfChildrenVisiting	0.011073	0.735926	9.048496e+02
MonthlyIncome	209.554880	904.849579	2.895191e+07

```
[57]: numeric_data_1.skew()
```

```
[57]: ProdTaken      1.595763
      Age           0.382989
      CityTier      0.736531
      DurationOfPitch 1.752037
      NumberOfPersonVisiting 0.029817
      NumberOfFollowups -0.372719
      PreferredPropertyStar 0.895545
      NumberOfTrips  1.453884
      Passport       0.920980
      PitchSatisfactionScore -0.127726
      OwnCar         -0.495892
      NumberOfChildrenVisiting 0.272199
      MonthlyIncome  1.949160
      dtype: float64
```

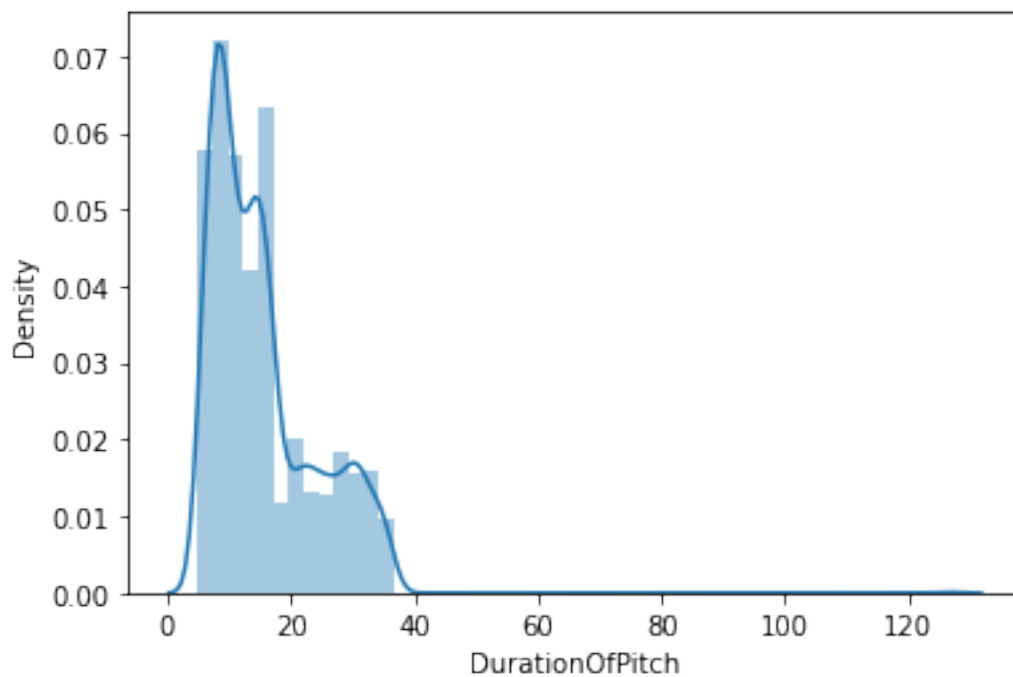
```
[62]: sns.distplot(numeric_data_1['NumberOfFollowups']) # Plotting distribution of
      ↪NumberOfFollowups feature
```

```
[62]: <AxesSubplot:xlabel='NumberOfFollowups', ylabel='Density'>
```



```
[69]: sns.distplot(numeric_data_1['DurationOfPitch']) # Plotting distribution of
      ↪DurationOfPitch feature
      # Inference is that this feature is not normaly distributed.
```

```
[69]: <AxesSubplot:xlabel='DurationOfPitch', ylabel='Density'>
```



```
[73]: cat_df=data[cat_col].groupby('Gender').mean()  
cat_df
```

```
[73]: Empty DataFrame  
Columns: []  
Index: [Fe Male, Female, Male]
```

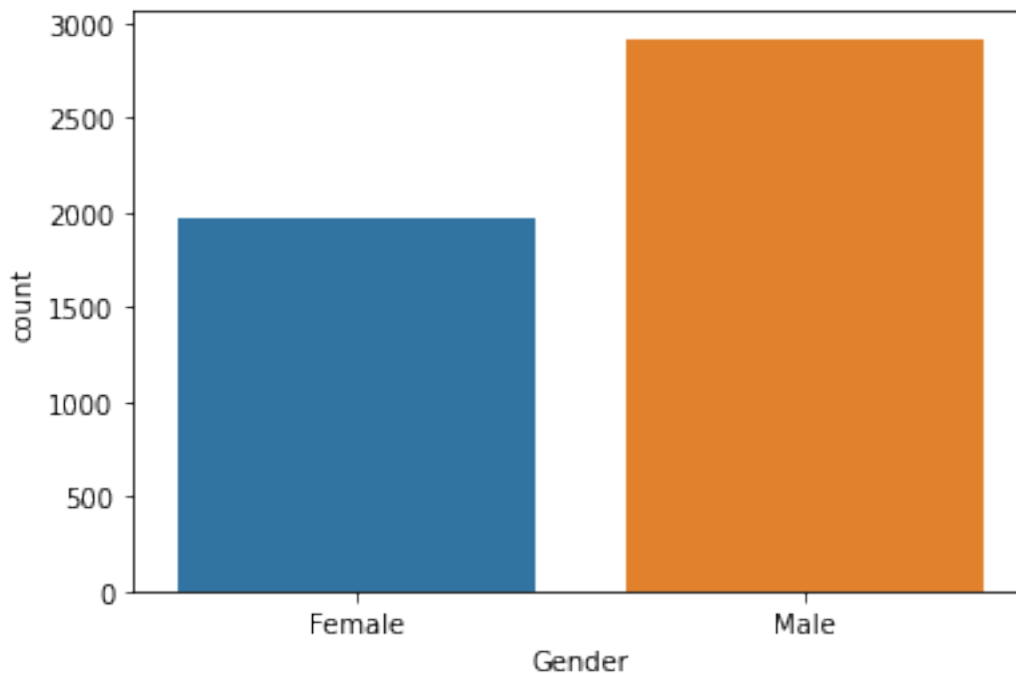
```
[75]: cat_df=data[cat_col].groupby('Gender').mean()  
cat_df
```

```
[75]: Empty DataFrame  
Columns: []  
Index: [Fe Male, Female, Male]
```

```
[85]: a=data[cat_col]['Gender'] = data[cat_col]['Gender'].replace({'Fe Male':  
↪ 'Female'}) # Replacing "Fe Male by Female"
```

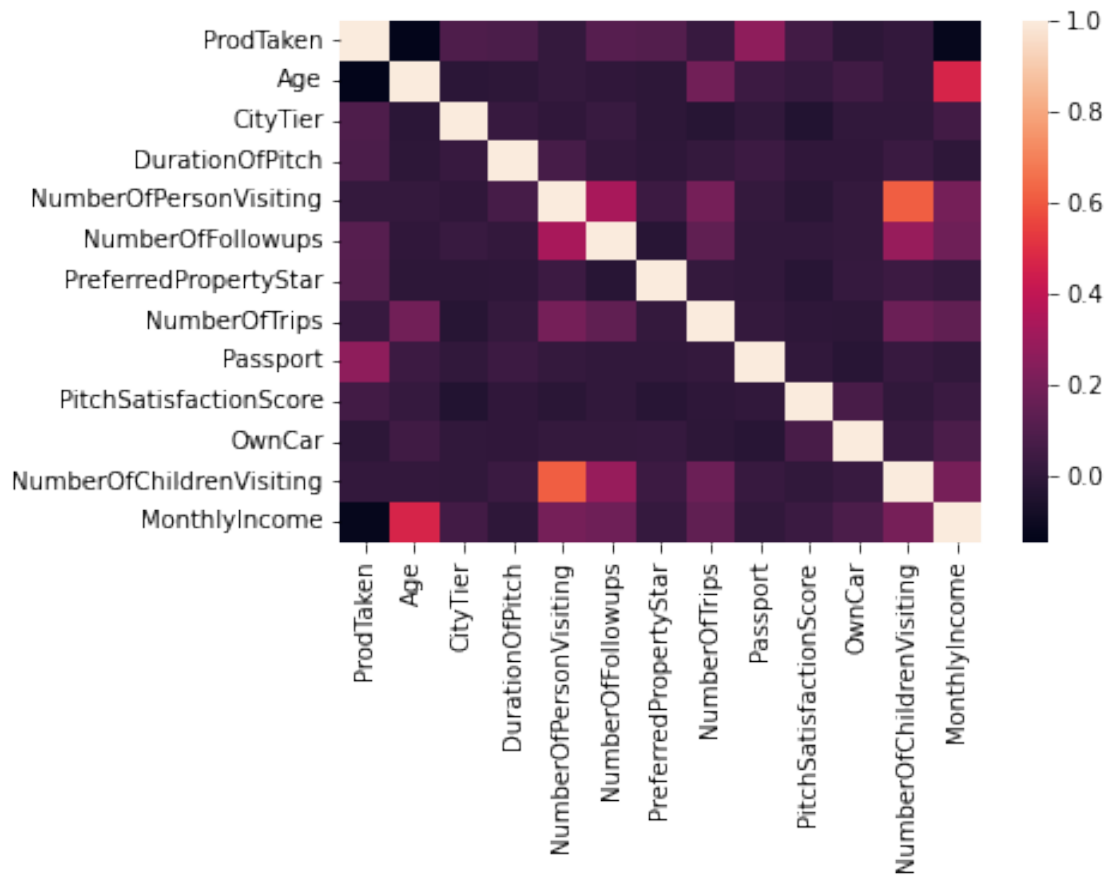
```
[87]: sns.countplot(x=a)
```

```
[87]: <AxesSubplot:xlabel='Gender', ylabel='count'>
```



```
[88]: sns.heatmap(numeric_data_1.corr()) # drawing the correlation heatmap for
      ↪ numerical features
```

```
[88]: <AxesSubplot:>
```



Encoding Categorical Variables

September 28, 2022

```
[53]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
[54]: data=pd.read_csv('Travel.csv')
```

```
[55]: cat_col=[fea for fea in data.columns if data[fea].dtype == 'O'] #segregating
↪ categorical and numerical variables
cat_col
```

```
[55]: ['TypeofContact',
'Occupation',
'Gender',
'ProductPitched',
'MaritalStatus',
'Designation']
```

```
[56]: X=data[cat_col]
X
```

```
[56]:
```

	TypeofContact	Occupation	Gender	ProductPitched	MaritalStatus	\
0	Self Enquiry	Salaried	Female	Deluxe	Single	
1	Company Invited	Salaried	Male	Deluxe	Divorced	
2	Self Enquiry	Free Lancer	Male	Basic	Single	
3	Company Invited	Salaried	Female	Basic	Divorced	
4	Self Enquiry	Small Business	Male	Basic	Divorced	
...	
4883	Self Enquiry	Small Business	Male	Deluxe	Unmarried	
4884	Company Invited	Salaried	Male	Basic	Single	
4885	Self Enquiry	Salaried	Female	Standard	Married	
4886	Self Enquiry	Small Business	Male	Basic	Single	
4887	Self Enquiry	Salaried	Male	Basic	Unmarried	

```

      Designation
0      Manager
1      Manager
2      Executive
3      Executive
4      Executive
...
4883      Manager
4884      Executive
4885  Senior Manager
4886      Executive
4887      Executive

[4888 rows x 6 columns]

```

```
[57]: X.isnull().sum()
```

```

[57]: TypeofContact      25
      Occupation         0
      Gender             0
      ProductPitched     0
      MaritalStatus      0
      Designation        0
      dtype: int64

```

```

[58]: # Replacing Null values in "TypeofContact" feature by Mode
      X=X.fillna(X.mode().iloc[0])

```

```
[59]: X.isnull().sum()
```

```

[59]: TypeofContact      0
      Occupation         0
      Gender             0
      ProductPitched     0
      MaritalStatus      0
      Designation        0
      dtype: int64

```

Method 1: Creating Binary variables through One Hot Encoding

```

[67]: # Using Pandas
      X_encoded=pd.get_dummies(X,drop_first=True)# drop_first =True implies that
      ↪dropping the first binary variable

```

```

[61]: # Using Sklearn
      from sklearn.preprocessing import OneHotEncoder
      encoder=OneHotEncoder(categories='auto',drop='first',sparse=False)

```

```
X_encoded=encoder.fit(X)
X_encoded
```

```
[61]: OneHotEncoder(drop='first', sparse=False)
```

```
[62]: X_transformed=encoder.transform(X)
X_transformed
```

```
[62]: array([[1., 0., 1., ..., 1., 0., 0.],
           [0., 0., 1., ..., 1., 0., 0.],
           [1., 0., 0., ..., 0., 0., 0.],
           ...,
           [1., 0., 1., ..., 0., 1., 0.],
           [1., 0., 0., ..., 0., 0., 0.],
           [1., 0., 1., ..., 0., 0., 0.]])
```

Method 2: Replacing Categories with Ordinal Numbers

```
[63]: # Using sklearn
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder()
enc.fit(X)
X_trans=enc.transform(X)
X_trans
```

```
[63]: array([[1., 2., 1., 1., 2., 2.],
           [0., 2., 2., 1., 0., 2.],
           [1., 0., 2., 0., 2., 1.],
           ...,
           [1., 2., 1., 3., 1., 3.],
           [1., 3., 2., 0., 2., 1.],
           [1., 2., 2., 0., 3., 1.]])
```

Method 3: Label Encoding

```
[64]: from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(X['TypeofContact'])
```

```
[64]: LabelEncoder()
```

```
[65]: list(le.classes_)
```

```
[65]: ['Company Invited', 'Self Enquiry']
```

```
[66]: le.transform(X['TypeofContact'])
```

```
[66]: array([1, 0, 1, ..., 1, 1, 1])
```

Handling_Missing_Values

September 28, 2022

```
[130]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

```
[131]: data=pd.read_csv('Travel.csv')
```

```
[132]: num_col=[fea for fea in data.columns if (data[fea].dtype !='0' and data[fea].
↳dtype !=int)]#segregating categorical and numerical variables
num_col
```

```
[132]: ['Age',
'DurationOfPitch',
'NumberOfFollowups',
'PreferredPropertyStar',
'NumberOfTrips',
'NumberOfChildrenVisiting',
'MonthlyIncome']
```

```
[133]: data[num_col].isnull().mean()
```

```
[133]: Age                                0.046236
DurationOfPitch                        0.051350
NumberOfFollowups                      0.009206
PreferredPropertyStar                  0.005319
NumberOfTrips                         0.028642
NumberOfChildrenVisiting               0.013502
MonthlyIncome                         0.047668
dtype: float64
```

Method 1: Performing Mean Imputation

```
[134]: # Method 1: Performing Mean Imputation

for var in num_col:
```

```
value=data[var].mean()
data[var]=data[var].fillna(value)
```

```
data[num_col].isnull().mean() # Checking after imputation
```

Age	0.0
DurationOfPitch	0.0
NumberOfFollowups	0.0
PreferredPropertyStar	0.0
NumberOfTrips	0.0
NumberOfChildrenVisiting	0.0
MonthlyIncome	0.0
dtype:	float64

Method 2: Imputing Missing Values by mean using scikit-learn

```
# Method 2: Imputing Missing Values by mean using scikit-learn
import sklearn
from sklearn.impute import SimpleImputer
imputer= SimpleImputer(strategy='mean')
```

```
data=pd.read_csv('Travel.csv')
num_col=[fea for fea in data.columns if (data[fea].dtype !='O' and data[fea].
↳dtype !=int)]
#segregating categorical and numerical variables
X=data[num_col]
```

```
imputer.fit(X) # fitting SimpleImputer to the dataframe X
```

```
SimpleImputer()
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	41.0	6.0	3.0	3.0	
1	49.0	14.0	4.0	4.0	
2	37.0	8.0	4.0	3.0	
3	33.0	9.0	3.0	3.0	
4	NaN	8.0	3.0	4.0	
...	
4883	49.0	9.0	5.0	4.0	
4884	28.0	31.0	5.0	3.0	
4885	52.0	17.0	4.0	4.0	
4886	19.0	16.0	4.0	3.0	
4887	36.0	14.0	4.0	4.0	

NumberOfTrips NumberOfChildrenVisiting MonthlyIncome

0	1.0	0.0	20993.0
1	2.0	2.0	20130.0
2	7.0	0.0	17090.0
3	2.0	1.0	17909.0
4	1.0	0.0	18468.0
...
4883	2.0	1.0	26576.0
4884	3.0	2.0	21212.0
4885	7.0	3.0	31820.0
4886	3.0	2.0	20289.0
4887	3.0	2.0	24041.0

[4888 rows x 7 columns]

```
[140]: X=imputer.transform(X)
X
```

```
[140]: array([[4.1000e+01, 6.0000e+00, 3.0000e+00, ..., 1.0000e+00, 0.0000e+00,
2.0993e+04],
[4.9000e+01, 1.4000e+01, 4.0000e+00, ..., 2.0000e+00, 2.0000e+00,
2.0130e+04],
[3.7000e+01, 8.0000e+00, 4.0000e+00, ..., 7.0000e+00, 0.0000e+00,
1.7090e+04],
...,
[5.2000e+01, 1.7000e+01, 4.0000e+00, ..., 7.0000e+00, 3.0000e+00,
3.1820e+04],
[1.9000e+01, 1.6000e+01, 4.0000e+00, ..., 3.0000e+00, 2.0000e+00,
2.0289e+04],
[3.6000e+01, 1.4000e+01, 4.0000e+00, ..., 3.0000e+00, 2.0000e+00,
2.4041e+04]])
```

```
[143]: result=pd.DataFrame(X,columns=['Age',
'DurationOfPitch',
'NumberOfFollowups',
'PreferredPropertyStar',
'NumberOfTrips',
'NumberOfChildrenVisiting',
'MonthlyIncome'])
result # dataframe after mean imputation by sklearn's simple Imputer
```

```
[143]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	41.000000	6.0	3.0	3.0	
1	49.000000	14.0	4.0	4.0	
2	37.000000	8.0	4.0	3.0	
3	33.000000	9.0	3.0	3.0	
4	37.622265	8.0	3.0	4.0	
...	

4883	49.000000	9.0	5.0	4.0
4884	28.000000	31.0	5.0	3.0
4885	52.000000	17.0	4.0	4.0
4886	19.000000	16.0	4.0	3.0
4887	36.000000	14.0	4.0	4.0

	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
0	1.0	0.0	20993.0
1	2.0	2.0	20130.0
2	7.0	0.0	17090.0
3	2.0	1.0	17909.0
4	1.0	0.0	18468.0
...
4883	2.0	1.0	26576.0
4884	3.0	2.0	21212.0
4885	7.0	3.0	31820.0
4886	3.0	2.0	20289.0
4887	3.0	2.0	24041.0

[4888 rows x 7 columns]

```
[144]: result.isnull().mean() # Checking after imputation
```

```
[144]: Age                                0.0
DurationOfPitch                        0.0
NumberOfFollowups                      0.0
PreferredPropertyStar                  0.0
NumberOfTrips                          0.0
NumberOfChildrenVisiting                0.0
MonthlyIncome                          0.0
dtype: float64
```

```
[145]: # The above imputations are applicable for median imputation also.
```

Method 3: Replacing missing values in categorical features by the feature's mode.

```
[146]: data=pd.read_csv('Travel.csv')
```

```
[147]: cat_col=[fea for fea in data.columns if data[fea].dtype == 'O'] #segregating
        ↪categorical and numerical variables
cat_col
```

```
[147]: ['TypeofContact',
        'Occupation',
        'Gender',
        'ProductPitched',
        'MaritalStatus',
```

```
'Designation']
```

```
[149]: X=data[cat_col]  
X
```

```
[149]:
```

	TypeofContact	Occupation	Gender	ProductPitched	MaritalStatus	\
0	Self Enquiry	Salaried	Female	Deluxe	Single	
1	Company Invited	Salaried	Male	Deluxe	Divorced	
2	Self Enquiry	Free Lancer	Male	Basic	Single	
3	Company Invited	Salaried	Female	Basic	Divorced	
4	Self Enquiry	Small Business	Male	Basic	Divorced	
...	
4883	Self Enquiry	Small Business	Male	Deluxe	Unmarried	
4884	Company Invited	Salaried	Male	Basic	Single	
4885	Self Enquiry	Salaried	Female	Standard	Married	
4886	Self Enquiry	Small Business	Male	Basic	Single	
4887	Self Enquiry	Salaried	Male	Basic	Unmarried	

	Designation
0	Manager
1	Manager
2	Executive
3	Executive
4	Executive
...	...
4883	Manager
4884	Executive
4885	Senior Manager
4886	Executive
4887	Executive


```
[4888 rows x 6 columns]
```

```
[150]: X.isnull().mean()
```

```
[150]: TypeofContact    0.005115  
Occupation           0.000000  
Gender               0.000000  
ProductPitched       0.000000  
MaritalStatus        0.000000  
Designation          0.000000  
dtype: float64
```

```
[151]: # Only the first feature that is TypeofContact has missing values
```

```
[152]: value=X['TypeofContact'].mode()[0]  
X['TypeofContact']=X['TypeofContact'].fillna(value)
```



```
X
```

```
[152]:
```

	TypeofContact	Occupation	Gender	ProductPitched	MaritalStatus	\
0	Self Enquiry	Salaried	Female	Deluxe	Single	
1	Company Invited	Salaried	Male	Deluxe	Divorced	
2	Self Enquiry	Free Lancer	Male	Basic	Single	
3	Company Invited	Salaried	Female	Basic	Divorced	
4	Self Enquiry	Small Business	Male	Basic	Divorced	
...	
4883	Self Enquiry	Small Business	Male	Deluxe	Unmarried	
4884	Company Invited	Salaried	Male	Basic	Single	
4885	Self Enquiry	Salaried	Female	Standard	Married	
4886	Self Enquiry	Small Business	Male	Basic	Single	
4887	Self Enquiry	Salaried	Male	Basic	Unmarried	

	Designation
0	Manager
1	Manager
2	Executive
3	Executive
4	Executive
...	...
4883	Manager
4884	Executive
4885	Senior Manager
4886	Executive
4887	Executive

```
[4888 rows x 6 columns]
```

```
[153]: X.isnull().mean() # Checking after imputation, null values of 'TypeofContact'
↳ feature is zero.
```

```
[153]: TypeofContact    0.0
Occupation           0.0
Gender               0.0
ProductPitched       0.0
MaritalStatus        0.0
Designation          0.0
dtype: float64
```

Imputing using sklearn

```
[154]: #Imputing using sklearn
X_1=data[cat_col]

imputer_1=SimpleImputer(strategy='most_frequent')
```

```
X_1.isnull().mean()
```

```
[154]: TypeofContact      0.005115
      Occupation          0.000000
      Gender              0.000000
      ProductPitched      0.000000
      MaritalStatus       0.000000
      Designation         0.000000
      dtype: float64
```

```
[155]: imputer_1.fit(X_1)
      X_1=imputer_1.transform(X_1)
      X_1
```

```
[155]: array([[ 'Self Enquiry', 'Salaried', 'Female', 'Deluxe', 'Single',
                'Manager'],
       [ 'Company Invited', 'Salaried', 'Male', 'Deluxe', 'Divorced',
                'Manager'],
       [ 'Self Enquiry', 'Free Lancer', 'Male', 'Basic', 'Single',
                'Executive'],
       ...,
       [ 'Self Enquiry', 'Salaried', 'Female', 'Standard', 'Married',
                'Senior Manager'],
       [ 'Self Enquiry', 'Small Business', 'Male', 'Basic', 'Single',
                'Executive'],
       [ 'Self Enquiry', 'Salaried', 'Male', 'Basic', 'Unmarried',
                'Executive']], dtype=object)
```

```
[156]: result_1=pd.DataFrame(X_1,columns=[ 'TypeofContact',# converting back to
      ↪Dataframe from Numpy Array
      'Occupation',
      'Gender',
      'ProductPitched',
      'MaritalStatus',
      'Designation'])
      result_1
```

```
[156]:
```

	TypeofContact	Occupation	Gender	ProductPitched	MaritalStatus	\
0	Self Enquiry	Salaried	Female	Deluxe	Single	
1	Company Invited	Salaried	Male	Deluxe	Divorced	
2	Self Enquiry	Free Lancer	Male	Basic	Single	
3	Company Invited	Salaried	Female	Basic	Divorced	
4	Self Enquiry	Small Business	Male	Basic	Divorced	
...	
4883	Self Enquiry	Small Business	Male	Deluxe	Unmarried	
4884	Company Invited	Salaried	Male	Basic	Single	
4885	Self Enquiry	Salaried	Female	Standard	Married	

4886	Self Enquiry	Small Business	Male	Basic	Single
4887	Self Enquiry	Salaried	Male	Basic	Unmarried

	Designation
0	Manager
1	Manager
2	Executive
3	Executive
4	Executive
...	...
4883	Manager
4884	Executive
4885	Senior Manager
4886	Executive
4887	Executive

[4888 rows x 6 columns]

```
[157]: result_1.isnull().mean() # Checking after imputation
```

```
[157]: TypeofContact    0.0
Occupation            0.0
Gender                0.0
ProductPitched        0.0
MaritalStatus         0.0
Designation           0.0
dtype: float64
```

Method 4: Replacing Missing Values with an arbitrary number

```
[158]: data=pd.read_csv('Travel.csv')
num_col=[fea for fea in data.columns if (data[fea].dtype != 'O' and data[fea].
dtype !=int)]
#segregating categorical and numerical variables
X=data[num_col]
```

```
[159]: X.max()
```

```
[159]: Age                61.0
DurationOfPitch         127.0
NumberOfFollowups        6.0
PreferredPropertyStar    5.0
NumberOfTrips           22.0
NumberOfChildrenVisiting  3.0
MonthlyIncome          98678.0
dtype: float64
```

```
[160]: X.isnull().sum()
```

```
[160]: Age                226
      DurationOfPitch      251
      NumberOfFollowups    45
      PreferredPropertyStar 26
      NumberOfTrips        140
      NumberOfChildrenVisiting 66
      MonthlyIncome        233
      dtype: int64
```

```
[161]: X['Age'].fillna(70,inplace=True) # Replacing by Null values by an arbitrary
      ↪ number 70 for Age feature because it is greater than
      # the max value of Age
```

```
[162]: X.isnull().sum() # Checking after imputation
```

```
[162]: Age                0
      DurationOfPitch      251
      NumberOfFollowups    45
      PreferredPropertyStar 26
      NumberOfTrips        140
      NumberOfChildrenVisiting 66
      MonthlyIncome        233
      dtype: int64
```

```
[163]: # Using sklearn
      imputer=SimpleImputer(strategy='constant',fill_value=70)
      data=pd.read_csv('Travel.csv')
      X_1=data['Age'].to_numpy() # Converting to Numpy array
      X_1=X_1.reshape(-1, 1)
```

```
[164]: imputer.fit(X_1)
```

```
[164]: SimpleImputer(fill_value=70, strategy='constant')
```

```
[165]: X_1=imputer.transform(X_1)
```

```
[166]: result_1=pd.DataFrame(X_1,columns=['Age'])# converting back to DataFrame from
      ↪ Numpy Array

      result_1.isnull().sum() # checking after Imputation
```

```
[166]: Age    0
      dtype: int64
```

```
[168]: result_1
```

```
[168]:      Age
0      41.0
1      49.0
2      37.0
3      33.0
4      70.0
...
4883  49.0
4884  28.0
4885  52.0
4886  19.0
4887  36.0
```

[4888 rows x 1 columns]

Method 5: Replacing Missing Values in Categorical Variables

```
[169]: data=pd.read_csv('Travel.csv')
X=data[cat_col] # We would replace missing values with a string "Missing"
X.isnull().sum()
```

```
[169]: TypeofContact      25
Occupation              0
Gender                 0
ProductPitched         0
MaritalStatus          0
Designation            0
dtype: int64
```

```
[170]: for var in cat_col:
        X[var].fillna('Missing',inplace=True)
```

```
[171]: X.isnull().sum() # Checking after Imputation
```

```
[171]: TypeofContact      0
Occupation              0
Gender                 0
ProductPitched         0
MaritalStatus          0
Designation            0
dtype: int64
```

```
[172]: # Using sklearn
imputer=SimpleImputer(strategy='constant',fill_value='Missing')
data=pd.read_csv('Travel.csv')
X=data[cat_col]
imputer.fit(X)
```

```
[172]: SimpleImputer(fill_value='Missing', strategy='constant')
```

```
[173]: X=imputer.transform(X)
```

```
[174]: X
```

```
[174]: array([[ 'Self Enquiry', 'Salaried', 'Female', 'Deluxe', 'Single',  
          'Manager'],  
        [ 'Company Invited', 'Salaried', 'Male', 'Deluxe', 'Divorced',  
          'Manager'],  
        [ 'Self Enquiry', 'Free Lancer', 'Male', 'Basic', 'Single',  
          'Executive'],  
        ...,  
        [ 'Self Enquiry', 'Salaried', 'Female', 'Standard', 'Married',  
          'Senior Manager'],  
        [ 'Self Enquiry', 'Small Business', 'Male', 'Basic', 'Single',  
          'Executive'],  
        [ 'Self Enquiry', 'Salaried', 'Male', 'Basic', 'Unmarried',  
          'Executive']], dtype=object)
```

```
[175]: result_1=pd.DataFrame(X)# converting back to DataFrame from Numpy Array  
result_1.isnull().sum() # Checking after Imputation
```

```
[175]: 0    0  
1    0  
2    0  
3    0  
4    0  
5    0  
dtype: int64
```

Method 6:

Replacing missing values with a value at the end of the distribution

Replacing missing values with a value at the end of the distribution, is equivalent to replacing with an arbitrary value manually. But in this case we do it by automatically selecting as those at the very end of the variable distribution. As per the IQR proximity rule, missing values are replaced with $q3+1.5(IQR)$ at the right tail or by $q1-1.5(IQR)$ at the left tail.

```
[176]: X=data[num_col]  
X.isnull().sum()
```

```
[176]: Age                226  
DurationOfPitch        251  
NumberOfFollowups       45  
PreferredPropertyStar   26
```

```

NumberOfTrips          140
NumberOfChildrenVisiting 66
MonthlyIncome          233
dtype: int64

```

```

[177]: for var in num_col:
        IQR =X[var].quantile(0.75)-X[var].quantile(0.25)
        value=X[var].quantile(0.75)+(1.5*IQR)
        X[var]=X[var].fillna(value)

```

```

[178]: X

```

```

[178]:      Age  DurationOfPitch  NumberOfFollowups  PreferredPropertyStar \
0    41.0             6.0             3.0             3.0
1    49.0            14.0             4.0             4.0
2    37.0             8.0             4.0             3.0
3    33.0             9.0             3.0             3.0
4    63.5             8.0             3.0             4.0
...    ...              ...              ...              ...
4883  49.0             9.0             5.0             4.0
4884  28.0            31.0             5.0             3.0
4885  52.0            17.0             4.0             4.0
4886  19.0            16.0             4.0             3.0
4887  36.0            14.0             4.0             4.0

      NumberOfTrips  NumberOfChildrenVisiting  MonthlyIncome
0                1.0                0.0        20993.0
1                2.0                2.0        20130.0
2                7.0                0.0        17090.0
3                2.0                1.0        17909.0
4                1.0                0.0        18468.0
...              ...              ...              ...
4883              2.0                1.0        26576.0
4884              3.0                2.0        21212.0
4885              7.0                3.0        31820.0
4886              3.0                2.0        20289.0
4887              3.0                2.0        24041.0

```

```

[4888 rows x 7 columns]

```

```

[179]: X.isnull().sum()# Checking after imputation

```

```

[179]: Age                0
        DurationOfPitch    0
        NumberOfFollowups  0
        PreferredPropertyStar 0
        NumberOfTrips      0

```

```
NumberOfChildrenVisiting    0
MonthlyIncome               0
dtype: int64
```

```
[180]: # Doing the same with feature Engine
X=data[num_col]

import feature_engine
from feature_engine.imputation import EndTailImputer
```

```
[181]: # set up the imputer
tail_imputer = EndTailImputer(imputation_method='gaussian',
                               tail='right',
                               fold=3,
                               variables=num_col)

# fit the imputer
tail_imputer.fit(X)
```

```
[181]: EndTailImputer(variables=['Age', 'DurationOfPitch', 'NumberOfFollowups',
                                'PreferredPropertyStar', 'NumberOfTrips',
                                'NumberOfChildrenVisiting', 'MonthlyIncome'])
```

```
[183]: X=tail_imputer.transform(X)
X
```

```
[183]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	41.000000	6.0	3.0	3.0	
1	49.000000	14.0	4.0	4.0	
2	37.000000	8.0	4.0	3.0	
3	33.000000	9.0	3.0	3.0	
4	65.571426	8.0	3.0	4.0	
...	
4883	49.000000	9.0	5.0	4.0	
4884	28.000000	31.0	5.0	3.0	
4885	52.000000	17.0	4.0	4.0	
4886	19.000000	16.0	4.0	3.0	
4887	36.000000	14.0	4.0	4.0	
	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome		
0	1.0	0.0	20993.0		
1	2.0	2.0	20130.0		
2	7.0	0.0	17090.0		
3	2.0	1.0	17909.0		
4	1.0	0.0	18468.0		
...		
4883	2.0	1.0	26576.0		
4884	3.0	2.0	21212.0		

4885	7.0	3.0	31820.0
4886	3.0	2.0	20289.0
4887	3.0	2.0	24041.0

[4888 rows x 7 columns]

Method 7: Multivariate Imputation by Chained Equations

Multivariate Imputation by Chained Equations is a multiple Imputation Technique that models each variable with missing values as a function of the remaining variables and uses that estimate for Imputation

A more sophisticated approach is to use the IterativeImputer class, which models each feature with missing values as a function of other features, and uses that estimate for imputation. It does so in an iterated round-robin fashion: at each step, a feature column is designated as output y and the other feature columns are treated as inputs X . A regressor is fit on (X, y) for known y . Then, the regressor is used to predict the missing values of y . This is done for each feature in an iterative fashion, and then is repeated for `max_iter` imputation rounds. The results of the final imputation round are returned.

```
[184]: import numpy as np
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
```

```
[185]: X=data[num_col]
X
```

```
[185]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	41.0	6.0	3.0	3.0	
1	49.0	14.0	4.0	4.0	
2	37.0	8.0	4.0	3.0	
3	33.0	9.0	3.0	3.0	
4	NaN	8.0	3.0	4.0	
...	
4883	49.0	9.0	5.0	4.0	
4884	28.0	31.0	5.0	3.0	
4885	52.0	17.0	4.0	4.0	
4886	19.0	16.0	4.0	3.0	
4887	36.0	14.0	4.0	4.0	
	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome		
0	1.0	0.0	20993.0		
1	2.0	2.0	20130.0		
2	7.0	0.0	17090.0		
3	2.0	1.0	17909.0		
4	1.0	0.0	18468.0		
...		
4883	2.0	1.0	26576.0		

4884	3.0	2.0	21212.0
4885	7.0	3.0	31820.0
4886	3.0	2.0	20289.0
4887	3.0	2.0	24041.0

[4888 rows x 7 columns]

```
[186]: X.isnull().sum()
```

```
[186]: Age                226
      DurationOfPitch    251
      NumberOfFollowups   45
      PreferredPropertyStar  26
      NumberOfTrips      140
      NumberOfChildrenVisiting  66
      MonthlyIncome      233
      dtype: int64
```

```
[187]: imp = IterativeImputer(max_iter=10,random_state=0)
```

```
[188]: imp.fit(X)
```

```
[188]: IterativeImputer(random_state=0)
```

```
[189]: X=imp.transform(X)
      X
```

```
[189]: array([[4.1000e+01, 6.0000e+00, 3.0000e+00, ..., 1.0000e+00, 0.0000e+00,
          2.0993e+04],
          [4.9000e+01, 1.4000e+01, 4.0000e+00, ..., 2.0000e+00, 2.0000e+00,
          2.0130e+04],
          [3.7000e+01, 8.0000e+00, 4.0000e+00, ..., 7.0000e+00, 0.0000e+00,
          1.7090e+04],
          ...,
          [5.2000e+01, 1.7000e+01, 4.0000e+00, ..., 7.0000e+00, 3.0000e+00,
          3.1820e+04],
          [1.9000e+01, 1.6000e+01, 4.0000e+00, ..., 3.0000e+00, 2.0000e+00,
          2.0289e+04],
          [3.6000e+01, 1.4000e+01, 4.0000e+00, ..., 3.0000e+00, 2.0000e+00,
          2.4041e+04]])
```

```
[190]: result_1=pd.DataFrame(X)# Converting back to a DataFrame
      result_1.isnull().sum()
```

```
[190]: 0    0
      1    0
      2    0
```

```
3    0
4    0
5    0
6    0
dtype: int64
```

```
[ ]:
```

Handling_Outliers

September 28, 2022

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

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

```
[3]: num_col=[fea for fea in data.columns if (data[fea].dtype !='0' and data[fea].
↳dtype !=int)]
#segregating categorical and numerical variables
num_col
```

```
[3]: ['Age',
      'DurationOfPitch',
      'NumberOfFollowups',
      'PreferredPropertyStar',
      'NumberOfTrips',
      'NumberOfChildrenVisiting',
      'MonthlyIncome']
```

```
[4]: X=data[num_col]
X['MonthlyIncome'].sum()
```

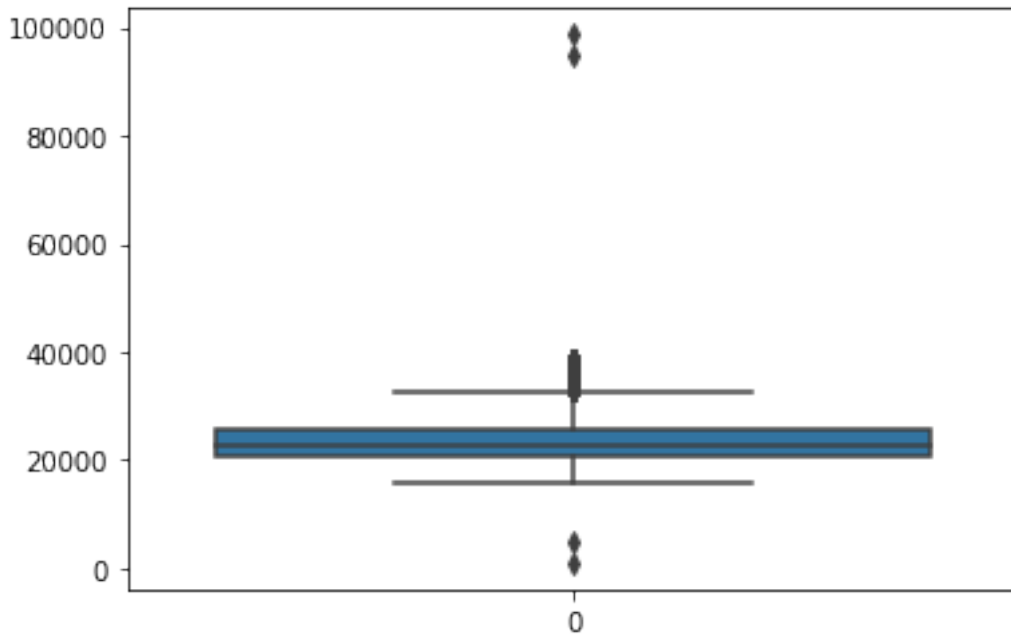
```
[4]: 109950418.0
```

```
[5]: # Performing Mean Imputation

for var in num_col:
    value=X[var].mean()
    X[var]=X[var].fillna(value)
```

```
[6]: sns.boxplot(X['MonthlyIncome']) # Taking Monthly Income as the variable
```

```
[6]: <AxesSubplot:>
```



Method 1: Replacing Outliers in the dataset with threshold values or Winorization

```
[7]: def find_boundaries(df,variable):
```

```
    IQR=df[variable].quantile(0.75)-df[variable].quantile(0.25)
    lower_boundary=df[variable].quantile(0.25)-(IQR*1.5)
    upper_boundary=df[variable].quantile(0.75)+(IQR*1.5)
    return upper_boundary,lower_boundary
```

```
[8]: def replace_with_threshold(df,var):
```

```
    for variable in num_col:
        upper_boundary,lower_boundary=find_boundaries(df,variable)
        data.loc[data[variable]<lower_boundary,variable]=lower_boundary
        data.loc[data[variable]>upper_boundary,variable]=upper_boundary
```

```
[9]: MonthlyIncome_upper_limit,MonthlyIncome_lower_limit=find_boundaries(X,'MonthlyIncome')
```

```
[10]: MonthlyIncome_upper_limit,MonthlyIncome_lower_limit
```

```
[10]: (32834.375, 13075.375)
```

```
[11]: replace_with_threshold(X,'MonthlyIncome')
```

```
[12]: X['MonthlyIncome'].sum()
```

```
[12]: 115453843.86337271
```

Method 2: Trimming outliers from data set

Here we remove outliers completely

```
[14]: outliers_MonthlyIncome = np.where(X['MonthlyIncome'] > MonthlyIncome_upper_limit, True,
    np.where(X['MonthlyIncome'] < MonthlyIncome_lower_limit, True, False))
```

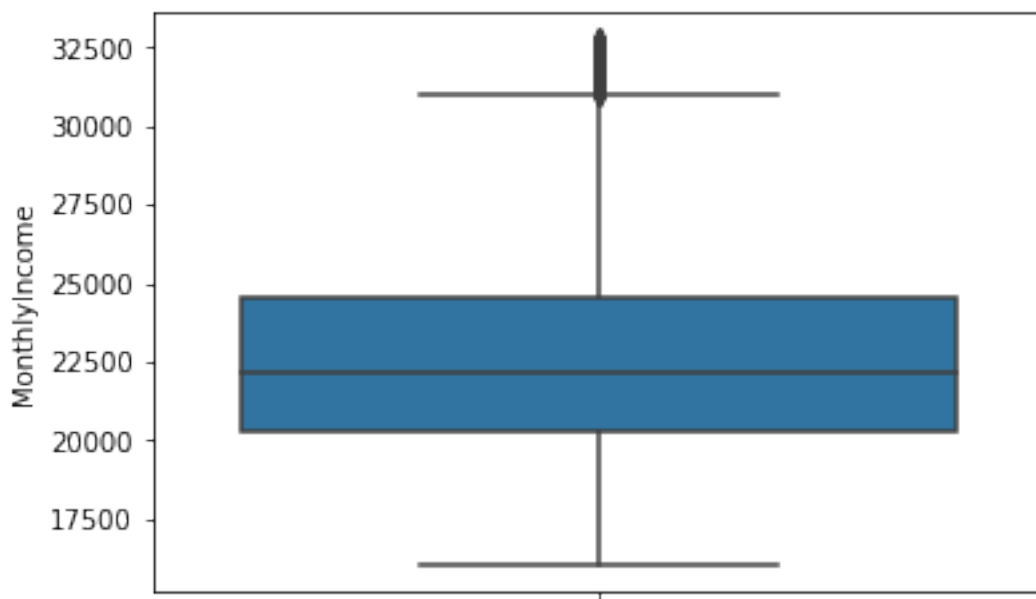
```
[27]: X_trimmed = X.loc[~outliers_MonthlyIncome] # deleting Outliers

X.shape, X_trimmed.shape
```

```
[27]: ((4888, 7), (4513, 7))
```

```
[16]: sns.boxplot(y=X_trimmed['MonthlyIncome'])
```

```
[16]: <AxesSubplot:ylabel='MonthlyIncome'>
```



Scaling

September 28, 2022

```
[4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
import sklearn
```

```
[5]: data=pd.read_csv('Travel.csv')
```

```
[6]: num_col=[fea for fea in data.columns if (data[fea].dtype !='0' and data[fea].
↳dtype !=int)]
#segregating categorical and numerical variables
num_col
```

```
[6]: ['Age',
'DurationOfPitch',
'NumberOfFollowups',
'PreferredPropertyStar',
'NumberOfTrips',
'NumberOfChildrenVisiting',
'MonthlyIncome']
```

```
[12]: X=data[num_col]
```

Method 1: Standardizing Features

Standardization is the process of centering the variable at zero and standardizing the variance to 1. To standardize the features we subtract the mean from each observation and then divide the result by standard deviation:

$$z=(x-\text{mean}(x))/\text{std}(x)$$

```
[11]: # Performing Mean Imputation

for var in num_col:
    value=X[var].mean()
    X[var]=X[var].fillna(value)
```

```
[18]: # using sklearn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X_scaled=scaler.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

```
[18]:
```

	0	1	2	3	4	5	6
0	3.712822e-01	-1.143871	-0.710021	-0.730127	-1.227404	-1.393568	-0.500322
1	1.250646e+00	-0.179681	0.292203	0.526467	-0.678603	0.953955	-0.664693
2	-6.839967e-02	-0.902823	0.292203	-0.730127	2.065400	-1.393568	-1.243704
3	-5.080815e-01	-0.782299	-0.710021	-0.730127	-0.678603	-0.219807	-1.087714
4	-7.810318e-16	-0.902823	-0.710021	0.526467	-1.227404	-1.393568	-0.981245
...
4883	1.250646e+00	-0.782299	1.294428	0.526467	-0.678603	-0.219807	0.563041
4884	-1.057684e+00	1.869222	1.294428	-0.730127	-0.129803	0.953955	-0.458610
4885	1.580407e+00	0.181890	0.292203	0.526467	2.065400	2.127717	1.561836
4886	-2.046968e+00	0.061367	0.292203	-0.730127	-0.129803	0.953955	-0.634409
4887	-1.783201e-01	-0.179681	0.292203	0.526467	-0.129803	0.953955	0.080213

[4888 rows x 7 columns]

Method 2: Performing mean normalization

In mean normalization, we center the variable at zero and rescale the distribution to the value range. This procedure involves subtracting from the mean from each observation and then dividing the result by the difference between the minimum and maximum values.

$$x_scaled = (x - \text{mean}(x)) / (\text{max}(x) - \text{min}(x))$$

The transformation results in a distribution centered around 0, with min and max values within the range of -1 to 1

```
[24]: from sklearn.preprocessing import StandardScaler, RobustScaler
scaler_mean=StandardScaler(with_mean=True, with_std=False) # No division by
↳ standard deviation
scaler_minmax=RobustScaler(with_centering=False, with_scaling=True, quantile_range=(0,100))
scaler_mean.fit(X)
scaler_minmax.fit(X)
X_scaled=scaler_minmax.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

```
[24]:
```

	0	1	2	3	4	5	6
0	0.953488	0.049180	0.6	1.5	0.047619	0.000000	0.214920
1	1.139535	0.114754	0.8	2.0	0.095238	0.666667	0.206085
2	0.860465	0.065574	0.8	1.5	0.333333	0.000000	0.174963
3	0.767442	0.073770	0.6	1.5	0.095238	0.333333	0.183347

4	0.874936	0.065574	0.6	2.0	0.047619	0.000000	0.189070
...
4883	1.139535	0.073770	1.0	2.0	0.095238	0.333333	0.272078
4884	0.651163	0.254098	1.0	1.5	0.142857	0.666667	0.217163
4885	1.209302	0.139344	0.8	2.0	0.333333	1.000000	0.325764
4886	0.441860	0.131148	0.8	1.5	0.142857	0.666667	0.207713
4887	0.837209	0.114754	0.8	2.0	0.142857	0.666667	0.246125

[4888 rows x 7 columns]

Method 3: Scaling to the maximum and minimum values

Scaling to the minimum and minimum values squeezes the values of the variables between 0 and 1.

$x_{scaled} = (x - \min(x)) / (\max(x) - \min(x))$

```
[26]: from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(X)
X_scaled=scaler.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

[26]:	0	1	2	3	4	5	6
0	0.534884	0.008197	0.4	0.0	0.000000	0.000000	0.204683
1	0.720930	0.073770	0.6	0.5	0.047619	0.666667	0.195848
2	0.441860	0.024590	0.6	0.0	0.285714	0.000000	0.164725
3	0.348837	0.032787	0.4	0.0	0.047619	0.333333	0.173110
4	0.456332	0.024590	0.4	0.5	0.000000	0.000000	0.178832
...
4883	0.720930	0.032787	0.8	0.5	0.047619	0.333333	0.261840
4884	0.232558	0.213115	0.8	0.0	0.095238	0.666667	0.206925
4885	0.790698	0.098361	0.6	0.5	0.285714	1.000000	0.315527
4886	0.023256	0.090164	0.6	0.0	0.095238	0.666667	0.197475
4887	0.418605	0.073770	0.6	0.5	0.095238	0.666667	0.235887

[4888 rows x 7 columns]

Method 4: Implementing Maximum absolute scaling

$x_{scaled} = x / \max(x)$

```
[ ]: from sklearn.preprocessing import MaxAbsScaler
scaler=MaxAbsScaler()
scaler.fit(X)
X_scaled=scaler.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

Method 5: Scaling with median and Quantiles

$$x_{scaled} = (x - \text{median}(x)) / (q_3(x) - q_1(x))$$

```
[28]: scaler=RobustScaler()
scaler.fit(X)
X_scaled=scaler.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

```
[28]:
```

	0	1	2	3	4	5	6
0	0.333333	-0.8	-1.0	0.0	-1.0	-1.0	-0.336454
1	1.000000	0.0	0.0	1.0	-0.5	1.0	-0.511159
2	0.000000	-0.6	0.0	0.0	2.0	-1.0	-1.126575
3	-0.333333	-0.5	-1.0	0.0	-0.5	0.0	-0.960777
4	0.051855	-0.6	-1.0	1.0	-1.0	-1.0	-0.847614
...
4883	1.000000	-0.5	1.0	1.0	-0.5	0.0	0.793765
4884	-0.750000	1.7	1.0	0.0	0.0	1.0	-0.292120
4885	1.250000	0.3	0.0	1.0	2.0	2.0	1.855357
4886	-1.500000	0.2	0.0	0.0	0.0	1.0	-0.478972
4887	-0.083333	0.0	0.0	1.0	0.0	1.0	0.280581

[4888 rows x 7 columns]

Method 6: Scaling to vector unit length

$$x_{scaled} = x / \text{norm}$$

where norm may be either Manhattan distance or Euclidean Distance

```
[29]: from sklearn.preprocessing import Normalizer
scaler=Normalizer(norm='l2')# l2 for Euclidean Distance
scaler.fit(X)
X_scaled=scaler.transform(X)
X_scaled_final=pd.DataFrame(X_scaled)
X_scaled_final
```

```
[29]:
```

	0	1	2	3	4	5	6
0	0.001953	0.000286	0.000143	0.000143	0.000048	0.000000	0.999998
1	0.002434	0.000695	0.000199	0.000199	0.000099	0.000099	0.999997
2	0.002165	0.000468	0.000234	0.000176	0.000410	0.000000	0.999997
3	0.001843	0.000503	0.000168	0.000168	0.000112	0.000056	0.999998
4	0.002037	0.000433	0.000162	0.000217	0.000054	0.000000	0.999998
...
4883	0.001844	0.000339	0.000188	0.000151	0.000075	0.000038	0.999998
4884	0.001320	0.001461	0.000236	0.000141	0.000141	0.000094	0.999998
4885	0.001634	0.000534	0.000126	0.000126	0.000220	0.000094	0.999998
4886	0.000936	0.000789	0.000197	0.000148	0.000148	0.000099	0.999999

```
4887  0.001497  0.000582  0.000166  0.000166  0.000125  0.000083  0.999999
```

```
[4888 rows x 7 columns]
```

Transforming Numerical Variables

September 28, 2022

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

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

```
[7]: num_col=[fea for fea in data.columns if (data[fea].dtype !='0' and data[fea].
↳dtype !=int)]
#segregating categorical and numerical variables
num_col
```

```
[7]: ['Age',
'DurationOfPitch',
'NumberOfFollowups',
'PreferredPropertyStar',
'NumberOfTrips',
'NumberOfChildrenVisiting',
'MonthlyIncome']
```

```
[8]: cat_col=[fea for fea in data.columns if data[fea].dtype =='0'] #segregating_
↳categorical and numerical variables
cat_col
```

```
[8]: ['TypeofContact',
'Occupation',
'Gender',
'ProductPitched',
'MaritalStatus',
'Designation']
```

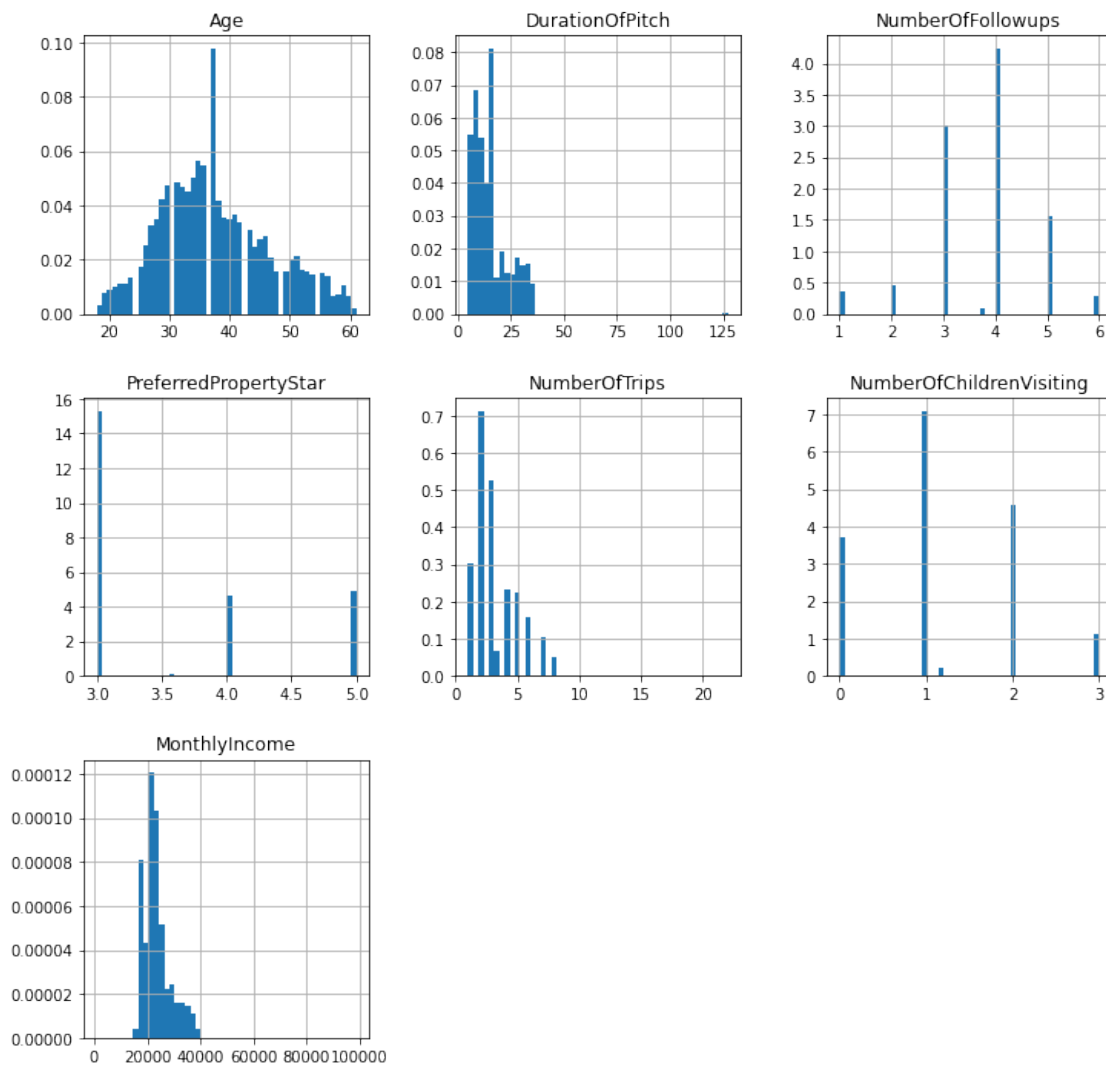
Method 1: Transforming Variables with the Logarithm:

Transforming Variables may improve the performance of Linear and Logistic regression machine learning models

```
[51]: #Using sklearn
import scipy.stats as stats
from sklearn.preprocessing import FunctionTransformer
X=data[num_col]
```

```
[52]: # Performing Mean Imputation
for var in num_col:
    value=X[var].mean()
    X[var]=X[var].fillna(value)
```

```
[53]: X.hist(bins=50,figsize=(12,12),density=True)
plt.show()
```



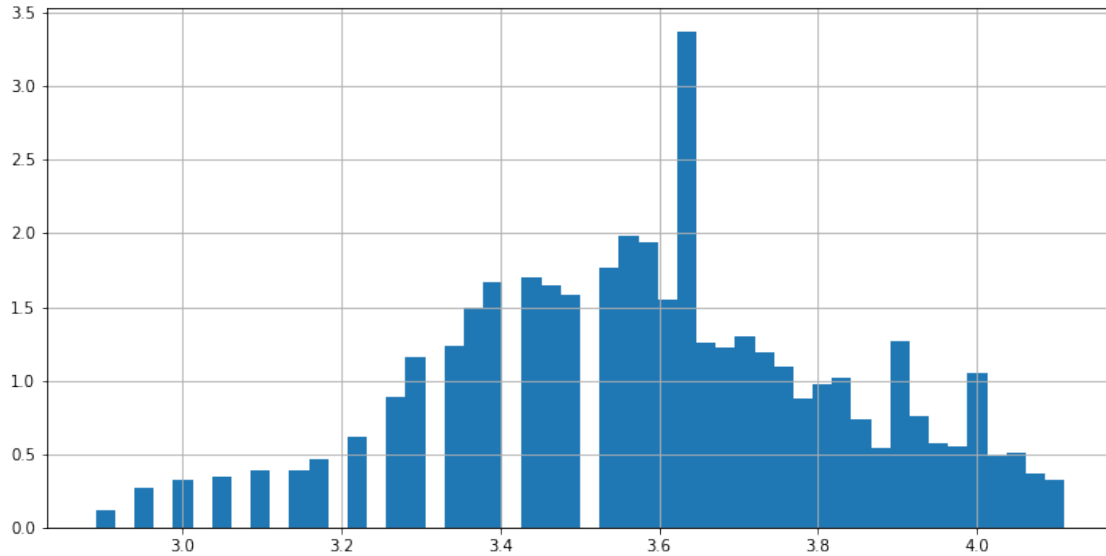
```
[54]: # creating a copy of the original dataframe using pandas copy()
X_tf=X.copy()
# Applying log transformation
X_tf=np.log(X_tf)
X_tf
```

```
[54]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	3.713572	1.791759	1.098612	1.098612	
1	3.891820	2.639057	1.386294	1.386294	
2	3.610918	2.079442	1.386294	1.098612	
3	3.496508	2.197225	1.098612	1.098612	
4	3.627596	2.079442	1.098612	1.386294	
...	
4883	3.891820	2.197225	1.609438	1.386294	
4884	3.332205	3.433987	1.609438	1.098612	
4885	3.951244	2.833213	1.386294	1.386294	
4886	2.944439	2.772589	1.386294	1.098612	
4887	3.583519	2.639057	1.386294	1.386294	
	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome		
0	0.000000	-inf	9.951944		
1	0.693147	0.693147	9.909967		
2	1.945910	-inf	9.746249		
3	0.693147	0.000000	9.793059		
4	0.000000	-inf	9.823795		
...		
4883	0.693147	0.000000	10.187764		
4884	1.098612	0.693147	9.962322		
4885	1.945910	1.098612	10.367850		
4886	1.098612	0.693147	9.917834		
4887	1.098612	0.693147	10.087516		

[4888 rows x 7 columns]

```
[55]: X_tf_1=X_tf['Age']
X_tf_1.hist(bins=50,figsize=(12,6),density=True)
plt.show()
```



```
[56]: #Using sklearn
transformer=FunctionTransformer(np.log)
```

```
[57]: data_tf=transformer.transform(X_tf)
```

```
[58]: data_tf=pd.DataFrame(data_tf)
```

```
[59]: data_tf
```

```
[59]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar \
0	1.311994	0.583198	0.094048	0.094048
1	1.358877	0.970422	0.326634	0.326634
2	1.283962	0.732099	0.326634	0.094048
3	1.251765	0.787195	0.094048	0.094048
4	1.288570	0.732099	0.094048	0.326634
...
4883	1.358877	0.787195	0.475885	0.326634
4884	1.203634	1.233722	0.475885	0.094048
4885	1.374030	1.041412	0.326634	0.326634
4886	1.079918	1.019781	0.326634	0.094048
4887	1.276345	0.970422	0.326634	0.326634

	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
0	-inf	NaN	2.297768
1	-0.366513	-0.366513	2.293541
2	0.665730	NaN	2.276882
3	-0.366513	-inf	2.281674
4	-inf	NaN	2.284807

...
4883	-0.366513	-inf	2.321187
4884	0.094048	-0.366513	2.298810
4885	0.665730	0.094048	2.338710
4886	0.094048	-0.366513	2.294335
4887	0.094048	-0.366513	2.311299

[4888 rows x 7 columns]

Method 2: Transforming variables with reciprocal function

```
[63]: # Using Numpy
X_input=X['Age']
np.reciprocal(X_input)
```

```
[63]: 0      0.024390
      1      0.020408
      2      0.027027
      3      0.030303
      4      0.026580
      ...
      4883    0.020408
      4884    0.035714
      4885    0.019231
      4886    0.052632
      4887    0.027778
      Name: Age, Length: 4888, dtype: float64
```

```
[67]: X_input=X['Age']
X_input
#Using sklearn
transformer=FunctionTransformer(np.reciprocal)
X_input=transformer.transform(X_input)
X_input
```

```
[67]: 0      0.024390
      1      0.020408
      2      0.027027
      3      0.030303
      4      0.026580
      ...
      4883    0.020408
      4884    0.035714
      4885    0.019231
      4886    0.052632
      4887    0.027778
      Name: Age, Length: 4888, dtype: float64
```


Method 3: Transforming variables with square root and cube root

```
[73]: #Using numpy
data=pd.read_csv('Travel.csv')
X=data[num_col]
X_tf=np.sqrt(X)
X_tf
```

```
[73]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	6.403124	2.449490	1.732051	1.732051	
1	7.000000	3.741657	2.000000	2.000000	
2	6.082763	2.828427	2.000000	1.732051	
3	5.744563	3.000000	1.732051	1.732051	
4	NaN	2.828427	1.732051	2.000000	
...	
4883	7.000000	3.000000	2.236068	2.000000	
4884	5.291503	5.567764	2.236068	1.732051	
4885	7.211103	4.123106	2.000000	2.000000	
4886	4.358899	4.000000	2.000000	1.732051	
4887	6.000000	3.741657	2.000000	2.000000	

	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
0	1.000000	0.000000	144.889613
1	1.414214	1.414214	141.880231
2	2.645751	0.000000	130.728727
3	1.414214	1.000000	133.824512
4	1.000000	0.000000	135.897020
...
4883	1.414214	1.000000	163.021471
4884	1.732051	1.414214	145.643400
4885	2.645751	1.732051	178.381613
4886	1.732051	1.414214	142.439461
4887	1.732051	1.414214	155.051604

[4888 rows x 7 columns]

```
[74]: #using sklearn
data=pd.read_csv('Travel.csv')
X=data[num_col]
transformer=FunctionTransformer(np.sqrt)
X=transformer.transform(X)
X
```

```
[74]:
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	6.403124	2.449490	1.732051	1.732051	
1	7.000000	3.741657	2.000000	2.000000	
2	6.082763	2.828427	2.000000	1.732051	

3	5.744563	3.000000	1.732051	1.732051
4	NaN	2.828427	1.732051	2.000000
...
4883	7.000000	3.000000	2.236068	2.000000
4884	5.291503	5.567764	2.236068	1.732051
4885	7.211103	4.123106	2.000000	2.000000
4886	4.358899	4.000000	2.000000	1.732051
4887	6.000000	3.741657	2.000000	2.000000

	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
0	1.000000	0.000000	144.889613
1	1.414214	1.414214	141.880231
2	2.645751	0.000000	130.728727
3	1.414214	1.000000	133.824512
4	1.000000	0.000000	135.897020
...
4883	1.414214	1.000000	163.021471
4884	1.732051	1.414214	145.643400
4885	2.645751	1.732051	178.381613
4886	1.732051	1.414214	142.439461
4887	1.732051	1.414214	155.051604

[4888 rows x 7 columns]

Method 4: Transforming variables with power transformations

```
[76]: #using sklearn
data=pd.read_csv('Travel.csv')
X=data[num_col]
transformer=FunctionTransformer(lambda x: np.power(x,0.3))
X=transformer.transform(X)
X
```

	Age	DurationOfPitch	NumberOfFollowups	PreferredPropertyStar	\
0	3.046738	1.711770	1.390389	1.390389	
1	3.214096	2.207183	1.515717	1.515717	
2	2.954340	1.866066	1.515717	1.390389	
3	2.854659	1.933182	1.390389	1.390389	
4	NaN	1.866066	1.390389	1.515717	
...	
4883	3.214096	1.933182	1.620657	1.515717	
4884	2.717361	2.801615	1.620657	1.390389	
4885	3.271907	2.339563	1.515717	1.515717	
4886	2.418945	2.297397	1.515717	1.390389	
4887	2.930156	2.207183	1.515717	1.515717	

	NumberOfTrips	NumberOfChildrenVisiting	MonthlyIncome
--	---------------	--------------------------	---------------

0	1.000000	0.000000	19.798047
1	1.231144	1.231144	19.550287
2	1.792790	0.000000	18.613267
3	1.231144	1.000000	18.876497
4	1.000000	0.000000	19.051359
...
4883	1.231144	1.000000	21.249410
4884	1.390389	1.231144	19.859782
4885	1.792790	1.390389	22.429007
4886	1.390389	1.231144	19.596486
4887	1.390389	1.231144	20.619862

[4888 rows x 7 columns]

Method 5: Transforming Numerical variables with Box-Cox transformations

```
[99]: #defined by (X**(lambda)-1)/X
#lambda is transformation parameter and X is the variable
data=pd.read_csv('Travel.csv')
X=data[num_col]
# Performing Mean Imputation
for var in num_col:
    value=X[var].mean()
    X[var]=X[var].fillna(value)
X_in=X['Age'].array.reshape(-1, 1)
from sklearn.preprocessing import PowerTransformer
#using sklearn
transformer=PowerTransformer(method='box-cox',standardize=False)
transformer.fit(X_in)
X_1=transformer.transform(X_in)
#X_in
```

Method 6: Transforming Numerical variables with Yeo-Johnson transformation

```
[102]: data=pd.read_csv('Travel.csv')
X=data[num_col]
# Performing Mean Imputation
for var in num_col:
    value=X[var].mean()
    X[var]=X[var].fillna(value)
X_in=X['Age'].array.reshape(-1, 1)
from sklearn.preprocessing import PowerTransformer
#using sklearn
transformer=PowerTransformer(method='yeo-johnson',standardize=False)
transformer.fit(X_in)
X_1=transformer.transform(X_in)

#X_in
```