CAPSTONE JDS

(GROUP 2)

Amirul Faris bin Jaafre

Shahrul Nizam bin Abu Bakar

Nurul Nadhirah binti Mohd Ikmal Hisham

Rafizatul Nabila binti Abdul Razak

Nur Farah Nabilah binti Ahmed Zhaini

OBJECTIVE:

Design a predictive model to determine the potential customers. The target variable is '*Potential_Customer*'

A) Data Preprocessing:	Integrity Check, Exploratory Data Analysis, Cleaning
B) Feature Engineering:	Make new features or change the current features
C) Feature Selection:	Choose the best features
D) Predictive Models:	Create several predictive models and tune the hyperparameters
E) Model Evaluation:	Compare the performance of the models
F) Choose the Best Model:	Choose the model that has the best performance

A) DATA PREPROCESSING (no. 1-5)

DATA CLEANING

Import packages → load data → no. of rows → remove $ and , sign

1. Import necessary Packages

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

     from sklearn.linear_model import LogisticRegression
     from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

2. Load the Data into Pandas Dataframe

```
data = pd.read_csv('data.csv')
data.head().T
```

	0	1	2	3	4
Potential_Customer	1	1	0	1	1
C_ID	9946	87939	88003	188721	88056
Cust_Last_Purchase	$5.00	$30.00	NaN	$20.00	$5.00
Pur_3_years	2	1	5	1	3
Pur_5_years	17	7	12	11	15
Pur_3_years_Indirect	2	0	3	0	2
Pur_5_years_Indirect	4	3	5	3	7
Pur_latest	$0.00	$25.00	$15.00	$20.00	$3.00
Pur_3_years_Avg	$7.50	$25.00	$15.00	$20.00	$4.33
Cust_Ann_Income	$65,957.00	$0.00	$0.00	$76,293.00	$113,663.00

3.1. How big is the data? (number of rows, features and their types)

```
[ ]  #number of rows and columns

     print("Data on rows: {}".format(len(data)))
     print("Data on columns: {}".format(len(data.columns)))
```

```
Data on rows: 4469
Data on columns: 25
```

3.1.1.1 Remove the dollar sign and comma from data

```
#Removed '$' and ',' signs

data = data.replace({'\$':''}, regex = True)
data = data.replace({'\,':''}, regex = True)

data.head().T
```

	0	1	2	3	4
Potential_Customer	1	1	0	1	1
C_ID	9946	87939	88003	188721	88056
Cust_Last_Purchase	5.00	30.00	NaN	20.00	5.00
Pur_3_years	2	1	5	1	3
Pur_5_years	17	7	12	11	15
Pur_3_years_Indirect	2	0	3	0	2
Pur_5_years_Indirect	4	3	5	3	7
Pur_latest	0.00	25.00	15.00	20.00	3.00
Pur_3_years_Avg	7.50	25.00	15.00	20.00	4.33
Cust_Ann_Income	65957.00	0.00	0.00	76293.00	113663.00

DATA CLEANING

Drop duplicates → drop 'C_ID' → change data types

3.2. Check data for duplicate rows

```
def check_duplicate(d,c):
    if len(d[c])>len(set(d[c])):
        print('Data has duplicate')
    else:
        print('Data does not have duplicate')

check_duplicate(data, 'C_ID')
```

```
Data has duplicate
```

3.2.1 Remove duplicate rows

Hint: Use `df=df.drop_duplicates()`

```
#drop all duplicates

data = data.drop_duplicates()
```

3.3. Do we need `C_ID` in our analysis?

```
#No, because C_ID does not contribute to predict potential

data.drop('C_ID', axis=1, inplace=True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3618 entries, 0 to 3617
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Potential_Customer       3618 non-null   int64
 1   Cust_Last_Purchase       1736 non-null   object
 2   Pur_3_years              3618 non-null   int64
 3   Pur_5_years              3618 non-null   int64
```

3.4. Check if the column types are accurate? if not fix them

```
#check data types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3618 entries, 0 to 3617
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Potential_Customer       3618 non-null   int64
 1   Cust_Last_Purchase       1736 non-null   object
 2   Pur_3_years              3618 non-null   int64
 3   Pur_5_years              3618 non-null   int64
 4   Pur_3_years_Indirect     3618 non-null   int64
 5   Pur_5_years_Indirect     3618 non-null   int64
 6   Pur_latest               3618 non-null   object
 7   Pur_3_years_Avg          3618 non-null   object
 8   Pur_5_years_Avg          3618 non-null   object
 9   Pur_3_years_Avg_Indirect 2956 non-null   object
 10  InAct_Last               3618 non-null   int64
 11  InAct_First              3618 non-null   int64
 12  Ad_Res_1_year            3618 non-null   int64
 13  Ad_Res_3_Year            3618 non-null   int64
 14  Ad_Res_5_Year            3618 non-null   int64
 15  Ad_Res_Ind_1_Year        3618 non-null   int64
 16  Ad_Res_Ind_3_Year        3618 non-null   int64
 17  Ad_Res_Ind_5_Year        3618 non-null   int64
 18  Status_Cust              3618 non-null   object
 19  Status_Latest_Ad         3618 non-null   int64
 20  Age                      2825 non-null   float64
 21  Gender                   3618 non-null   object
 22  Cust_Prop                3618 non-null   object
 23  Cust_Ann_Income          3618 non-null   object
dtypes: float64(1), int64(14), object(9)
memory usage: 706.6+ KB
```

```
#To distinguish data type

CatCols = ['Potential_Customer','Status_Cust', 'Status_Latest_Ad',
           'Gender', 'Cust_Prop']
NumCols = list(set(data.columns)-set(CatCols))
```

```
#Change data type

data[CatCols] = data[CatCols].apply(lambda x: x.astype('category'))
data[NumCols] = data[NumCols].apply(lambda x: x.astype('float64'))

data.info()

# Only 'Potential_Customer', 'Status_Cust', 'Status_Latest_Ad', 'Gender', 'Cust_Prop'
# changed into category variables, all other columns changed to float 64
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3618 entries, 0 to 3617
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Potential_Customer       3618 non-null   category
 1   Cust_Last_Purchase       1736 non-null   float64
 2   Pur_3_years              3618 non-null   float64
 3   Pur_5_years              3618 non-null   float64
 4   Pur_3_years_Indirect     3618 non-null   float64
 5   Pur_5_years_Indirect     3618 non-null   float64
 6   Pur_latest               3618 non-null   float64
 7   Pur_3_years_Avg          3618 non-null   float64
 8   Pur_5_years_Avg          3618 non-null   float64
 9   Pur_3_years_Avg_Indirect 2956 non-null   float64
 10  InAct_Last               3618 non-null   float64
 11  InAct_First              3618 non-null   float64
 12  Ad_Res_1_year            3618 non-null   float64
 13  Ad_Res_3_Year            3618 non-null   float64
 14  Ad_Res_5_Year            3618 non-null   float64
 15  Ad_Res_Ind_1_Year        3618 non-null   float64
 16  Ad_Res_Ind_3_Year        3618 non-null   float64
 17  Ad_Res_Ind_5_Year        3618 non-null   float64
 18  Status_Cust              3618 non-null   category
 19  Status_Latest_Ad         3618 non-null   category
 20  Age                      2825 non-null   float64
 21  Gender                   3618 non-null   category
 22  Cust_Prop                3618 non-null   category
 23  Cust_Ann_Income          3618 non-null   float64
```

EXPLORATORY DATA ANALYSIS

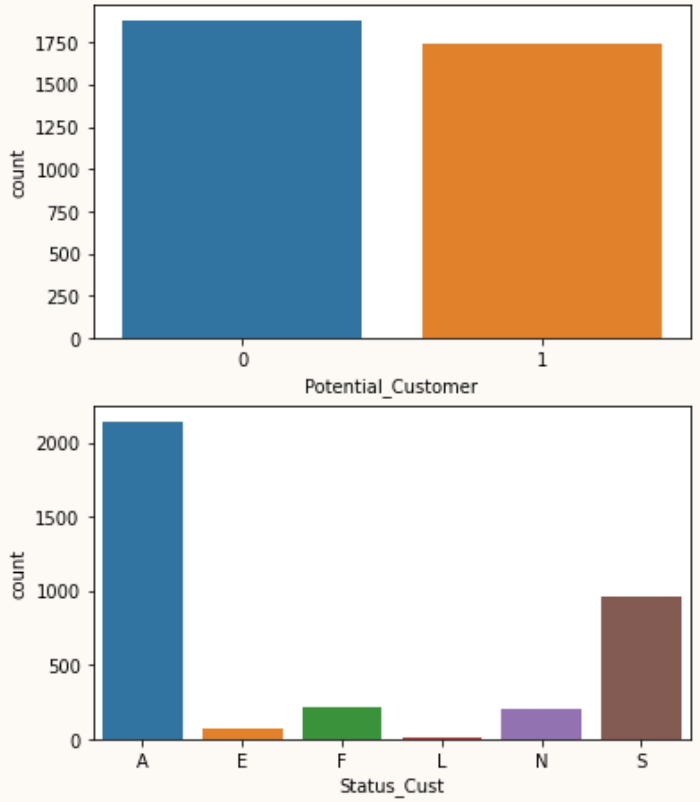
1) Explore Categorical Variables

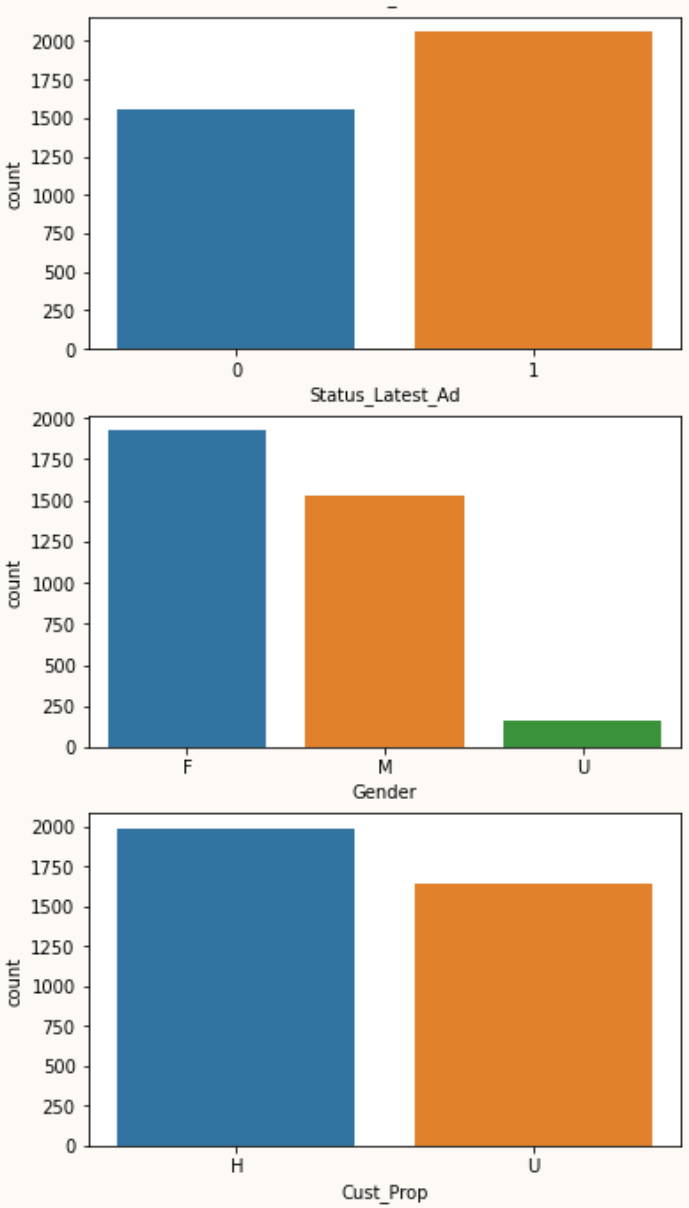
```
# describe all 5 categorical data columns

data.describe(include = ('category'))
```

	Potential_Customer	Status_Cust	Status_Latest_Ad	Gender	Cust_Prop
count	3618	3618	3618	3618	3618
unique	2	6	2	3	2
top	0	A	1	F	H
freq	1882	2146	2057	1922	1981

EXPLORATORY DATA ANALYSIS (CONT.)


```
# countplot for all 5 categorical data

fig, ax=plt.subplots(nrows=len(CatCols), figsize=(6,20))
for a in np.arange(len(CatCols)):
    sns.countplot(x=data[CatCols[a]],ax=ax[a]);
```

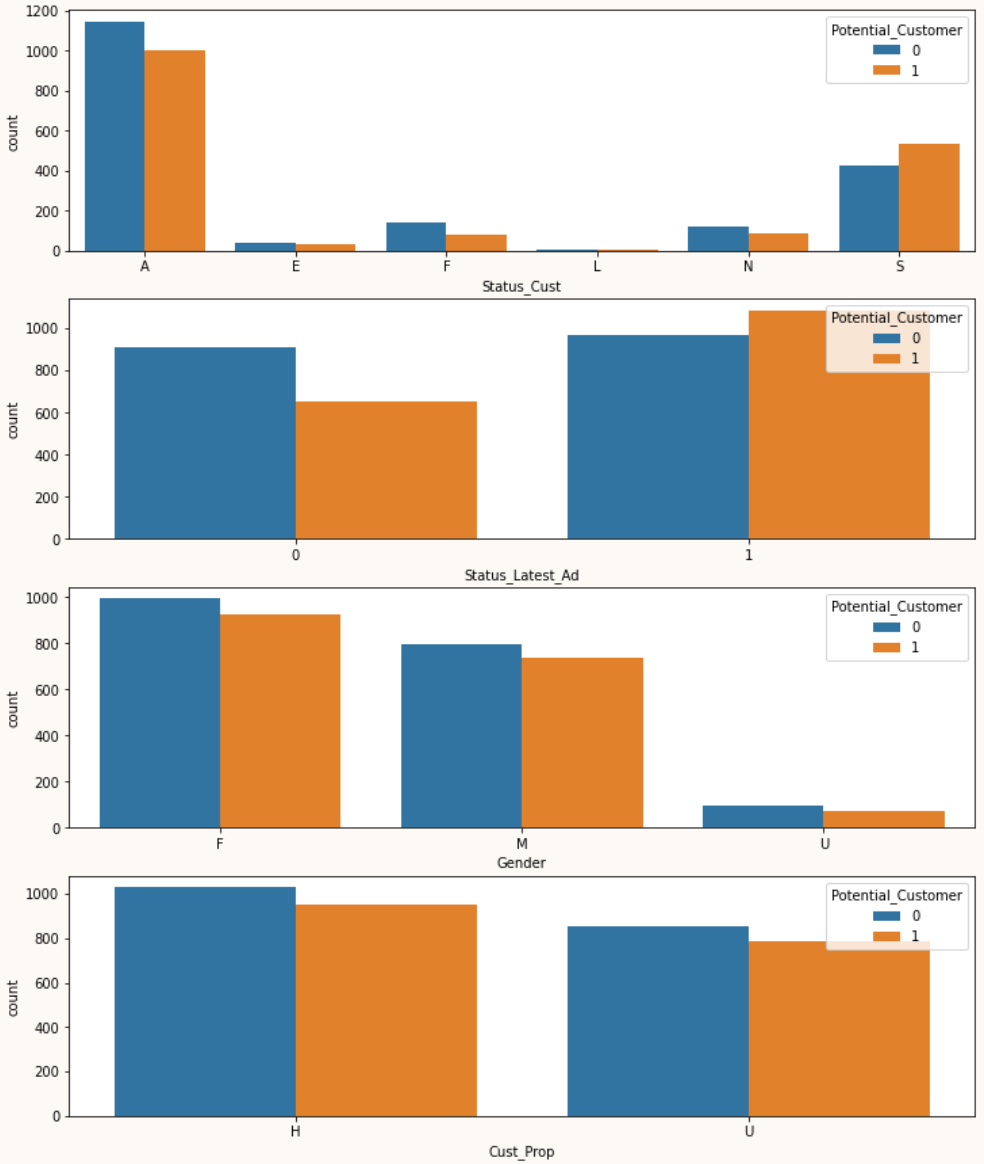




EXPLORATORY DATA ANALYSIS (CONT.)

2) Explore Relationship Between Categorical & Target Variable

```
# countplot for all 5 categorical data - based on Potential_Customer hue

Cats = ['Status_Cust', 'Status_Latest_Ad',
        'Gender', 'Cust_Prop']

fig, ax=plt.subplots(nrows=len(Cats), figsize=(12,15))
for a in np.arange(len(Cats)):
    sns.countplot(x=data[Cats[a]], hue=data.Potential_Customer,ax=ax[a]);
```



Insight:

- The distribution for Status_Cust and Status_Latest_Ad are significant to Potential Customer as we can observe the obvious difference
- While as for gender and Cust_Prop, they show slight difference which may have only little contribution towards Potential_Customer

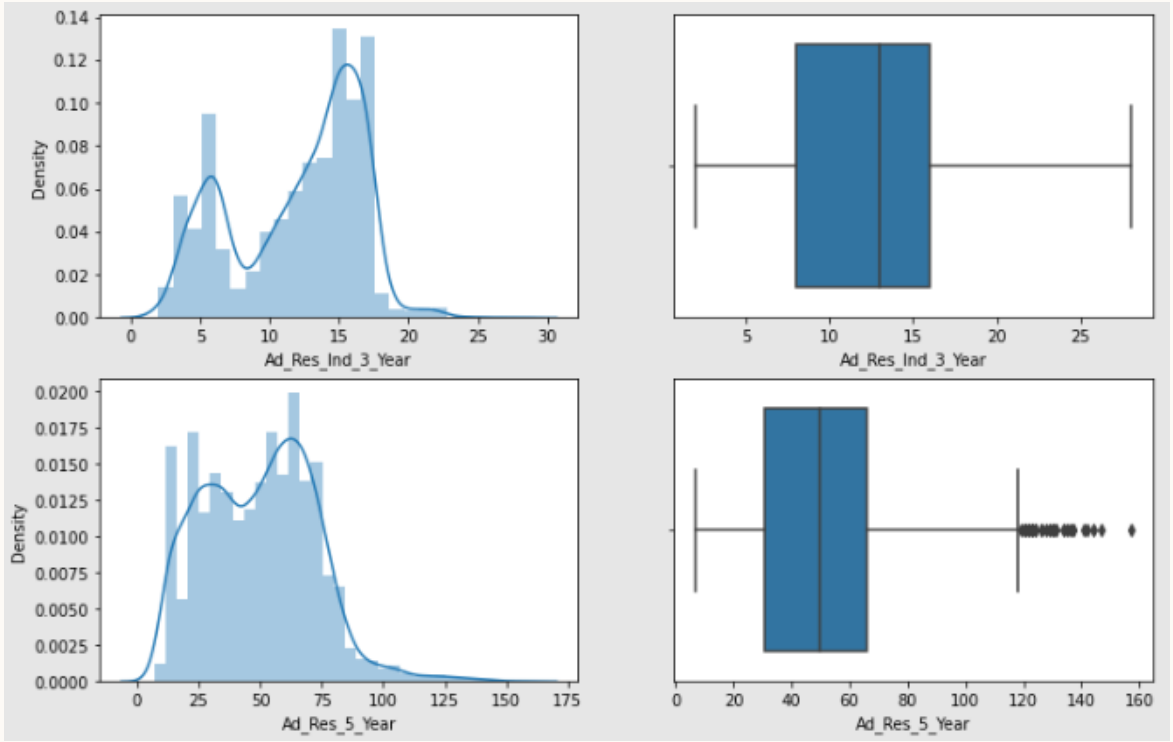
Solution:

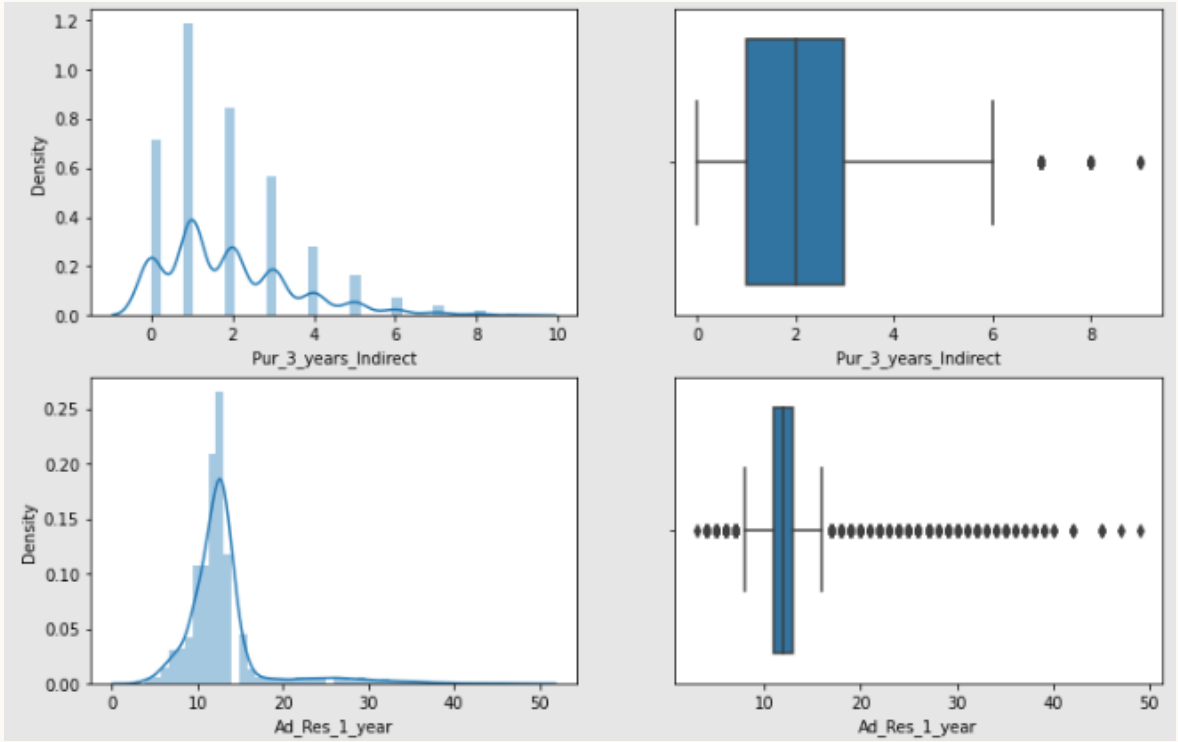
- Categorical variable can be converted into numerical for easier interpretation (This is done in part 8.1)

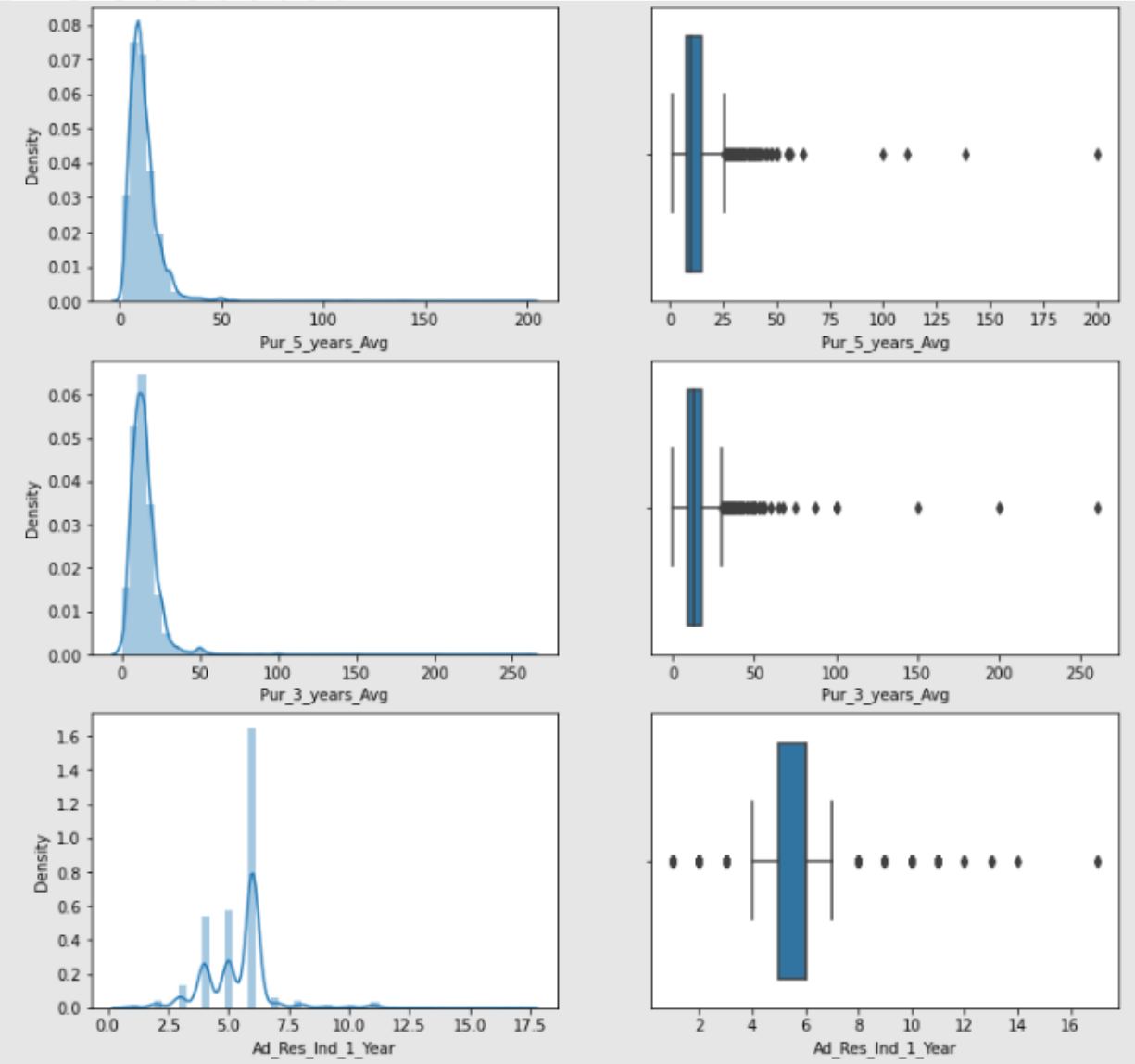
EXPLORATORY DATA ANALYSIS (CONT.)

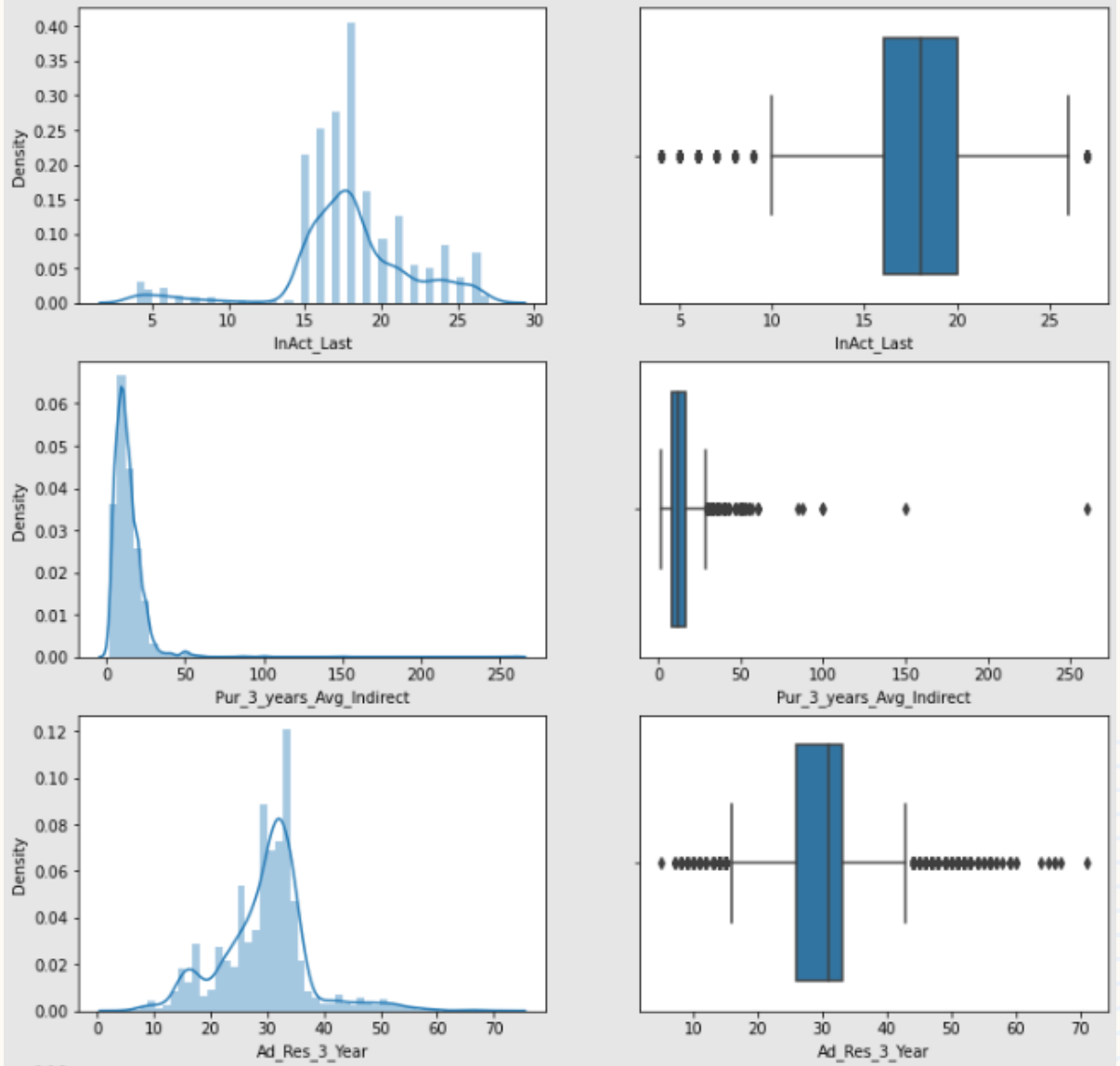
3) Explore Numerical Variables

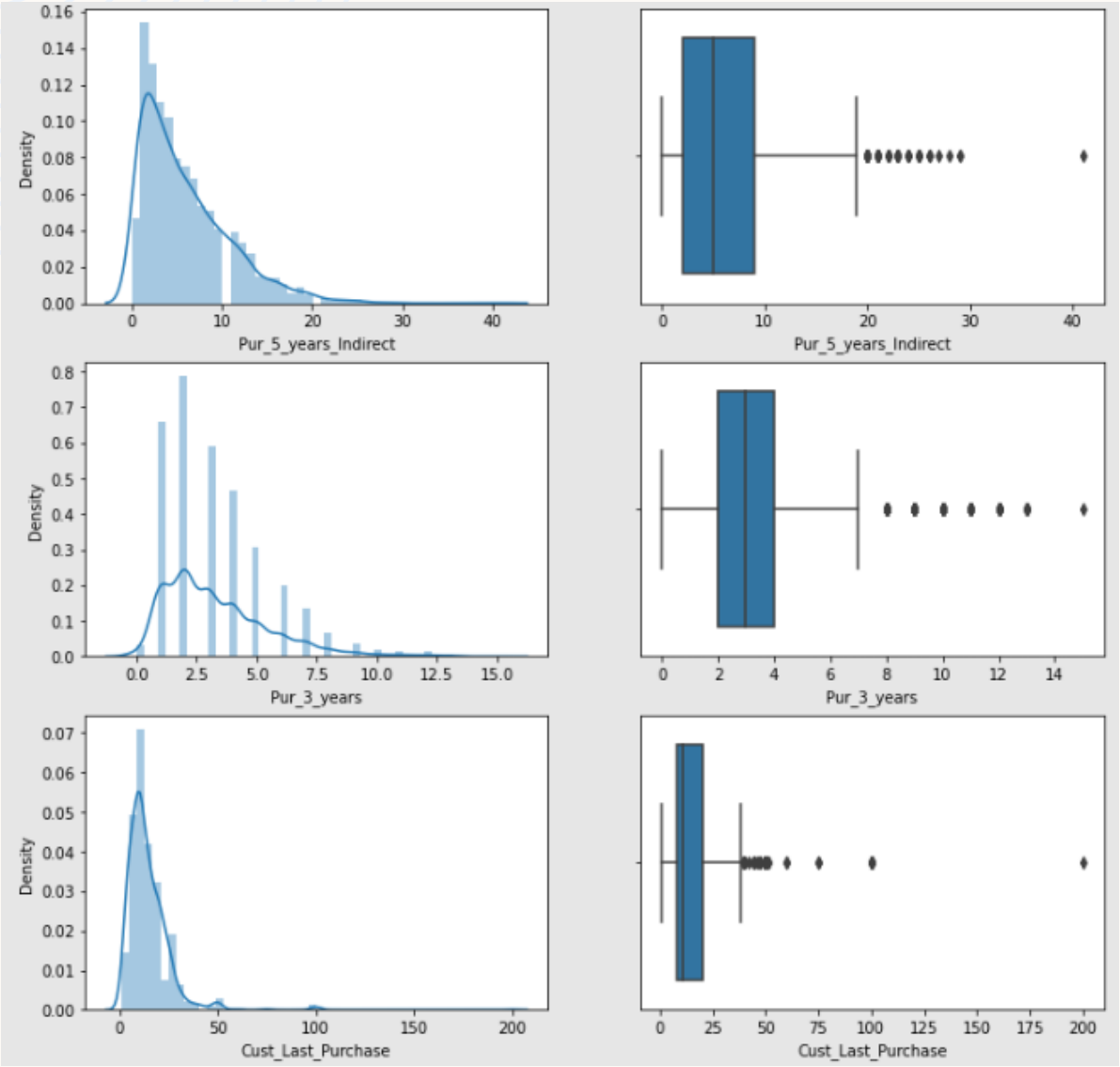
```
fig, ax=plt.subplots(nrows=len(NumCols), ncols=2, figsize=(12,80))
for a in range(len(NumCols)):
    sns.distplot(data[NumCols[a]],ax=ax[a,0])
    sns.boxplot(data[NumCols[a]],ax=ax[a,1]);
```

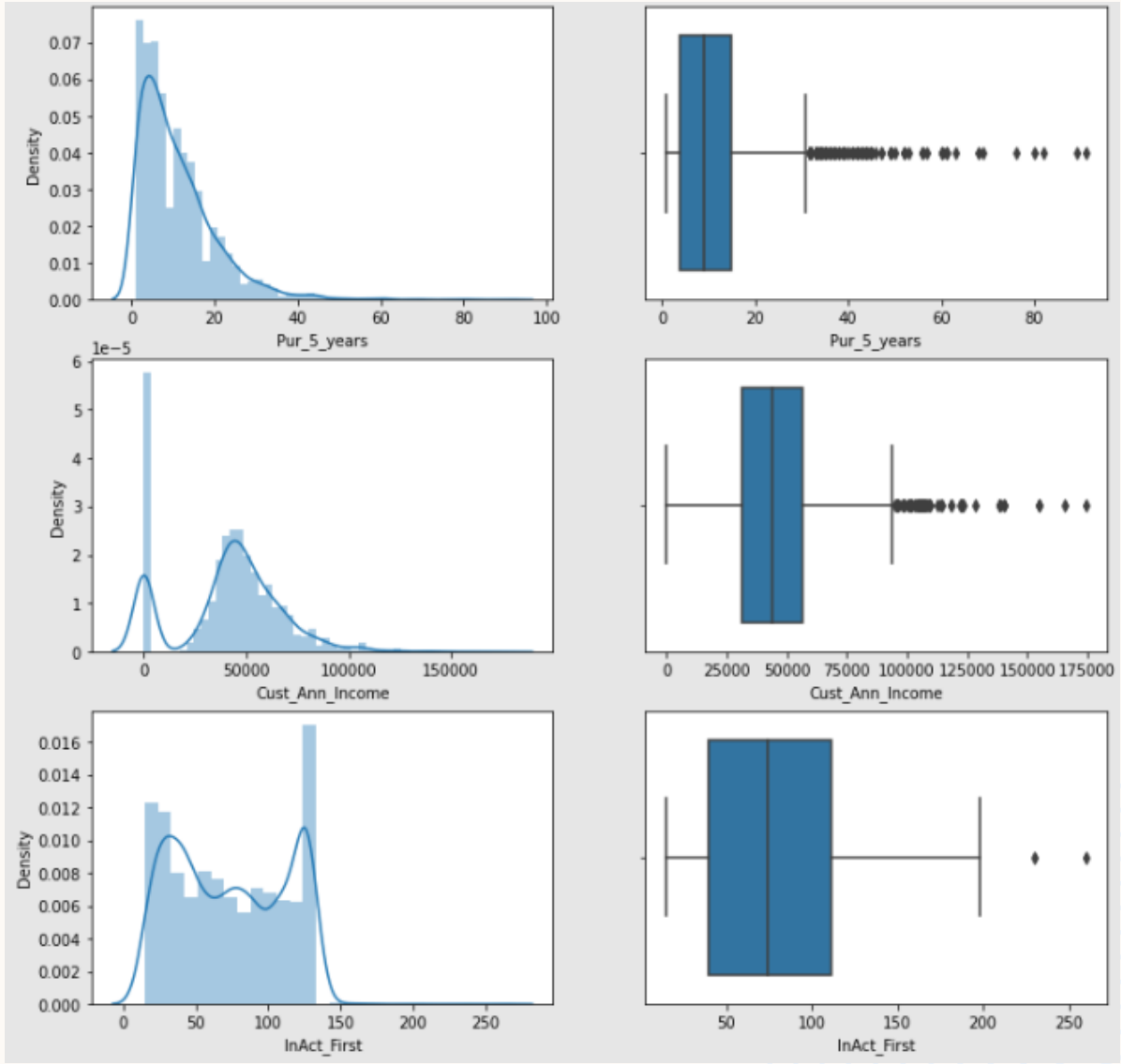


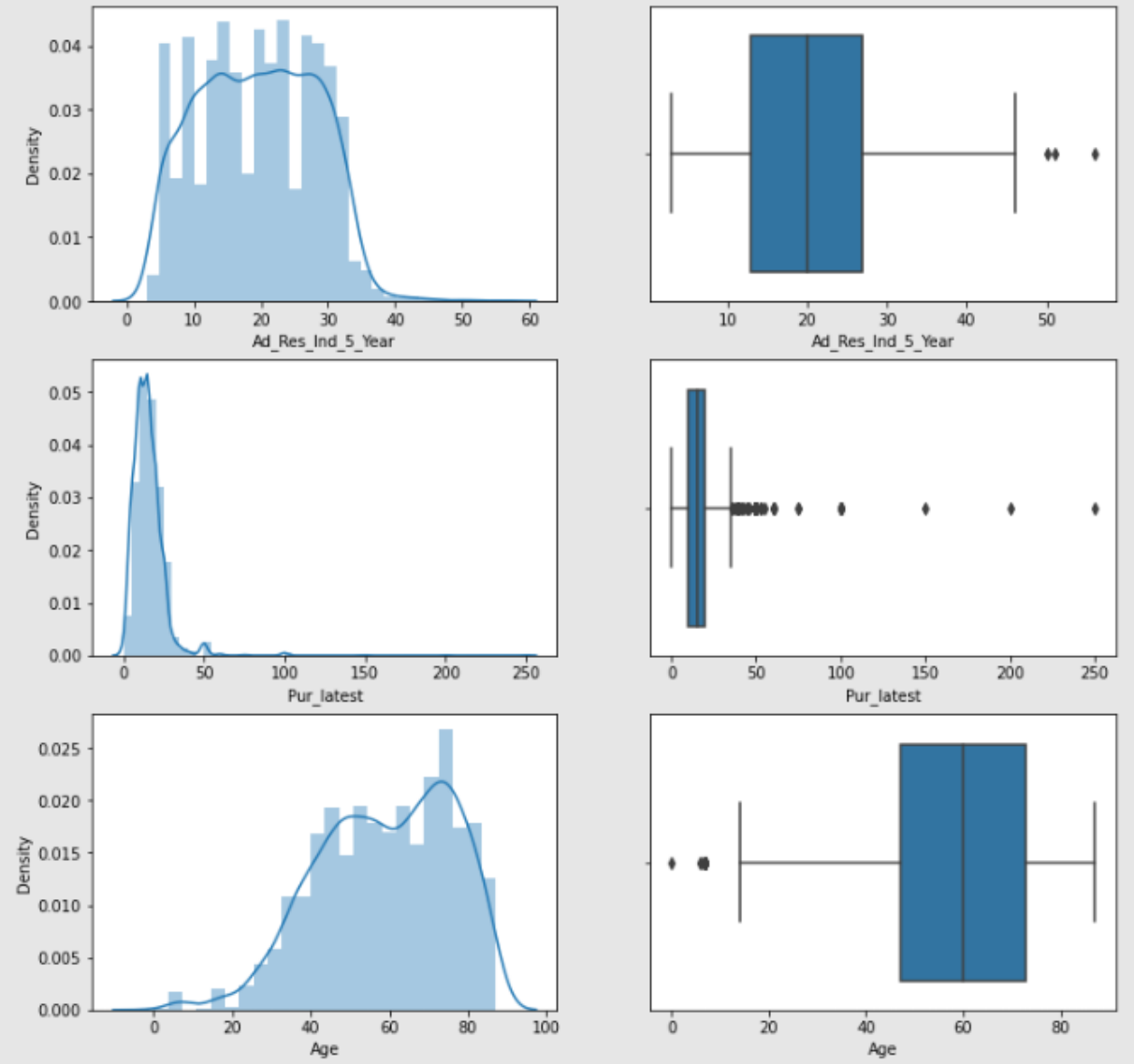



Density
Pur_5_years_Avg
Pur_5_years_Avg
Density
Pur_3_years_Avg
Pur_3_years_Avg
Density
Ad_Res_Ind_1_Year
Ad_Res_Ind_1_Year


Density
InAct_Last
InAct_Last
Density
Pur_3_years_Avg_Indirect
Pur_3_years_Avg_Indirect
Density
Ad_Res_3_Year
Ad_Res_3_Year


Density
Pur_5_years_Indirect
Pur_5_years_Indirect
Density
Pur_3_years
Pur_3_years
Density
Cust_Last_Purchase
Cust_Last_Purchase


Density
Pur_5_years
Pur_5_years
1e−5
Density
Cust_Ann_Income
Cust_Ann_Income
Density
InAct_First
InAct_First




```
from scipy.stats import skew
```

```
#skewness for all data
data.skew()
```

```
Cust_Last_Purchase          4.966810
Pur_3_years                 1.291322
Pur_5_years                 2.079296
Pur_3_years_Indirect        1.124769
Pur_5_years_Indirect        1.287071
Pur_latest                  6.660116
Pur_3_years_Avg             7.913836
Pur_5_years_Avg             6.756579
Pur_3_years_Avg_Indirect    8.043963
InAct_Last                 -0.808274
InAct_First                 0.136455
Ad_Res_1_year               2.788479
Ad_Res_3_Year               0.321259
Ad_Res_5_Year               0.401333
Ad_Res_Ind_1_Year           0.943712
Ad_Res_Ind_3_Year          -0.448007
Ad_Res_Ind_5_Year           0.068045
Age                        -0.406354
Cust_Ann_Income             0.039826
dtype: float64
```

```
# the skewness are mostly right-skewed (positive skewness)
```

Insight:

- Most of the variables show positive skew as they are skewed to the right except for InAct_Last, Age
- Some of the variables have outliers, hence we need to remove them

EXPLORATORY DATA ANALYSIS (CONT.)

Handling Missing Values

```
#Identify missing values
data.isna().sum()
```

```
Potential_Customer            0
Cust_Last_Purchase         1882
Pur_3_years                   0
Pur_5_years                   0
Pur_3_years_Indirect          0
Pur_5_years_Indirect          0
Pur_latest                    0
Pur_3_years_Avg               0
Pur_5_years_Avg               0
Pur_3_years_Avg_Indirect    662
InAct_Last                    0
InAct_First                   0
Ad_Res_1_year                 0
Ad_Res_3_Year                 0
Ad_Res_5_Year                 0
Ad_Res_Ind_1_Year             0
Ad_Res_Ind_3_Year             0
Ad_Res_Ind_5_Year             0
Status_Cust                   0
Status_Latest_Ad              0
Age                         793
Gender                        0
Cust_Prop                     0
Cust_Ann_Income               0
dtype: int64
```

```
#Display columns with missing values
data.columns[data.isnull().any()]
```

```
Index(['Cust_Last_Purchase', 'Pur_3_years_Avg_Indirect', 'Age'], dtype='object')
```

EXPLORATORY DATA ANALYSIS (CONT.)

Handling Missing Values:

1) Cust_Last_Purchase

```
#fill missing values of 'Cust_Last_Purchase' with 0

X_train.Cust_Last_Purchase = X_train.Cust_Last_Purchase.fillna(0)
X_test.Cust_Last_Purchase = X_test.Cust_Last_Purchase.fillna(0)
```

2) Pur_3_years_Avg_Indirect

```
#fill missing values of 'Pur_3_years_Avg_Indirect' with 0

X_train.Pur_3_years_Avg_Indirect = X_train.Pur_3_years_Avg_Indirect.fillna(0)
X_test.Pur_3_years_Avg_Indirect = X_test.Pur_3_years_Avg_Indirect.fillna(0)
```

3) Age

```
#fill missing values of 'Age' with median age (because cannot be 0)

X_train.Age = X_train.Age.fillna(int(X_train.Age.median())).astype(int)
X_test.Age = X_test.Age.fillna(int(X_train.Age.median())).astype(int)
```

EXPLORATORY DATA ANALYSIS (CONT.)

Handling Missing Values

```
X_train.isna().sum()
```

```
Cust_Last_Purchase          0
Pur_3_years                 0
Pur_5_years                 0
Pur_3_years_Indirect        0
Pur_5_years_Indirect        0
Pur_latest                  0
Pur_3_years_Avg             0
Pur_5_years_Avg             0
Pur_3_years_Avg_Indirect    0
InAct_Last                  0
InAct_First                 0
Ad_Res_1_year               0
Ad_Res_3_Year               0
Ad_Res_5_Year               0
Ad_Res_Ind_1_Year           0
Ad_Res_Ind_3_Year           0
Ad_Res_Ind_5_Year           0
Status_Cust                 0
Status_Latest_Ad            0
Age                         0
Gender                      0
Cust_Prop                   0
Cust_Ann_Income             0
dtype: int64
```

```
X_test.isna().sum()
```

```
Cust_Last_Purchase          0
Pur_3_years                 0
Pur_5_years                 0
Pur_3_years_Indirect        0
Pur_5_years_Indirect        0
Pur_latest                  0
Pur_3_years_Avg             0
Pur_5_years_Avg             0
Pur_3_years_Avg_Indirect    0
InAct_Last                  0
InAct_First                 0
Ad_Res_1_year               0
Ad_Res_3_Year               0
Ad_Res_5_Year               0
Ad_Res_Ind_1_Year           0
Ad_Res_Ind_3_Year           0
Ad_Res_Ind_5_Year           0
Status_Cust                 0
Status_Latest_Ad            0
Age                         0
Gender                      0
Cust_Prop                   0
Cust_Ann_Income             0
dtype: int64
```

B) FEATURE ENGINEERING (no. 6)

ADDING HIGH LEVEL FEATURES

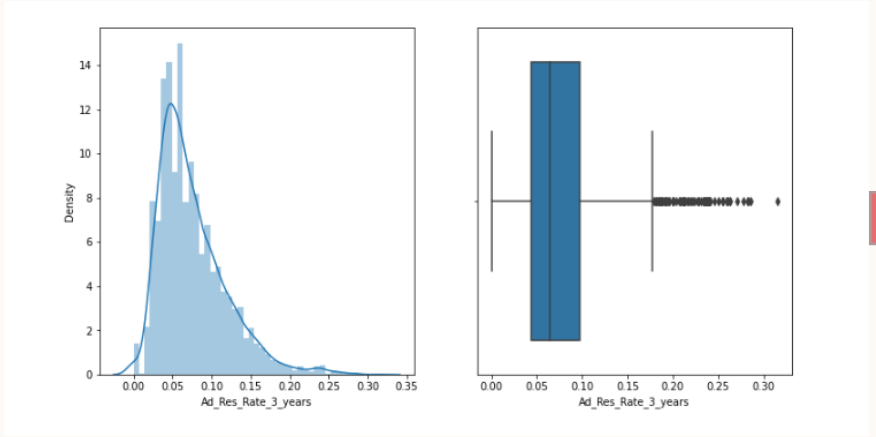
```
data['Pur_5_years_Total']=data['Pur_5_years']*data['Pur_5_years_Avg']
data['Pur_3_years_Total']=data['Pur_3_years']*data['Pur_3_years_Avg']
data['Ad_Res_Rate_3_years']=data['Pur_3_years']/(data['Ad_Res_3_Year']+data['Ad_Res_Ind_3_Year']+1)

data['sqrt_Pur_3_years_Indirect']=np.sqrt(data['Pur_3_years_Indirect'])
data['sqrt_Pur_latest']=np.sqrt(data['Pur_latest'])
data['sqrt_Pur_3_years']=np.sqrt(data['Pur_3_years'])
data['sqrt_Pur_3_years_Avg_Indirect']=np.sqrt(data['Pur_3_years_Avg_Indirect'])
data['sqrt_Pur_5_years_Indirect']=np.sqrt(data['Pur_5_years_Indirect'])
data['sqrt_Pur_3_years_Avg']=np.sqrt(data['Pur_3_years_Avg'])
data['sqrt_Pur_5_years']=np.sqrt(data['Pur_5_years'])
data['sqrt_Ad_Res_Rate_3_years']=np.sqrt(data['Ad_Res_Rate_3_years'])
data['sqrt_Pur_5_years_Total']=np.sqrt(data['Pur_5_years_Total'])
data['sqrt_Pur_3_years_Total']=np.sqrt(data['Pur_3_years_Total'])


newCols = ['Pur_5_years_Total','Ad_Res_Rate_3_years','sqrt_Pur_3_years_Indirect','sqrt_Pur_latest', 'sqrt_Pur_3_years','sqrt_Pur_3_years_Avg_Indirect',
           'sqrt_Pur_3_years_Avg','sqrt_Pur_5_years','sqrt_Ad_Res_Rate_3_years','sqrt_Pur_5_years_Total','sqrt_Pur_3_years_Total']

SumCols = newCols + NumCols
```

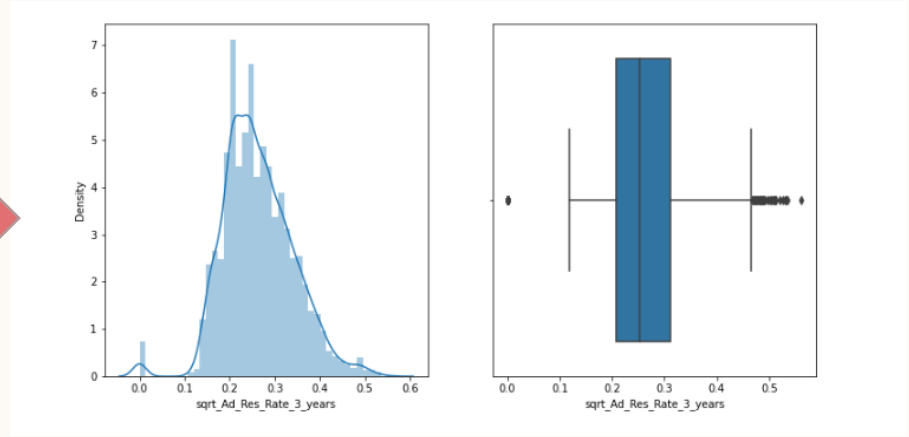
np.sqrt is applied to transform **right skewed features**

COMPARING SKEWNESS

Cust_Last_Purchase	4.966810
Pur_3_years	1.291322
Pur_5_years	2.079296
Pur_3_years_Indirect	1.124769
Pur_5_years_Indirect	1.287071
Pur_latest	6.660116
Pur_3_years_Avg	7.913836
Pur_5_years_Avg	6.756579
Pur_3_years_Avg_Indirect	8.043963
InAct_Last	-0.808274
InAct_First	0.136455
Ad_Res_1_year	2.788479
Ad_Res_3_Year	0.321259
Ad_Res_5_Year	0.401333
Ad_Res_Ind_1_Year	0.943712
Ad_Res_Ind_3_Year	-0.448007
Ad_Res_Ind_5_Year	0.068045
Age	-0.406354
Cust_Ann_Income	0.039826
dtype: float64	

sqrt_Pur_3_years_Indirect	-0.307275
sqrt_Pur_latest	1.307960
sqrt_Pur_3_years	0.316438
sqrt_Pur_3_years_Avg_Indirect	1.636768
sqrt_Pur_5_years_Indirect	0.157325
sqrt_Pur_3_years_Avg	1.210127
sqrt_Pur_5_years	0.601166
sqrt_Ad_Res_Rate_3_years	0.241969
sqrt_Pur_5_years_Total	1.663245
sqrt_Pur_3_years_Total	1.114593





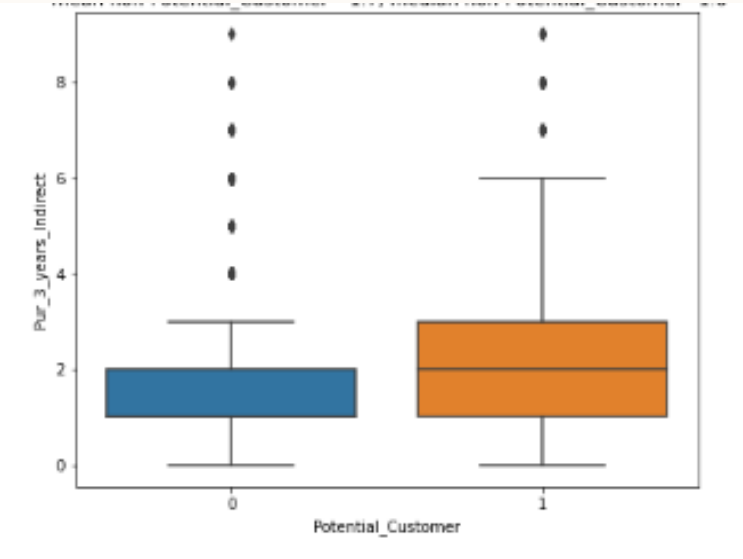
Before np.sqrt applied

After np.sqrt applied

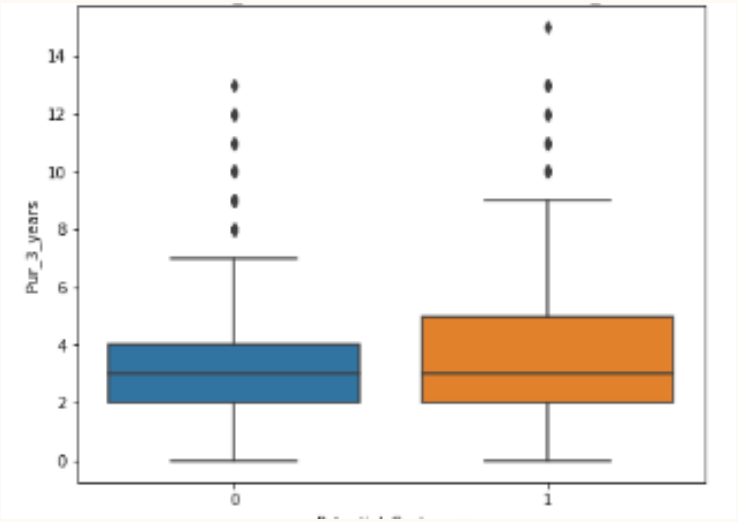
Value of skewness changes towards normal distribution after square root transformation

EXPLORE RELATIONSHIP BETWEEN NUMERICAL VARIABLES & TARGET VARIABLE

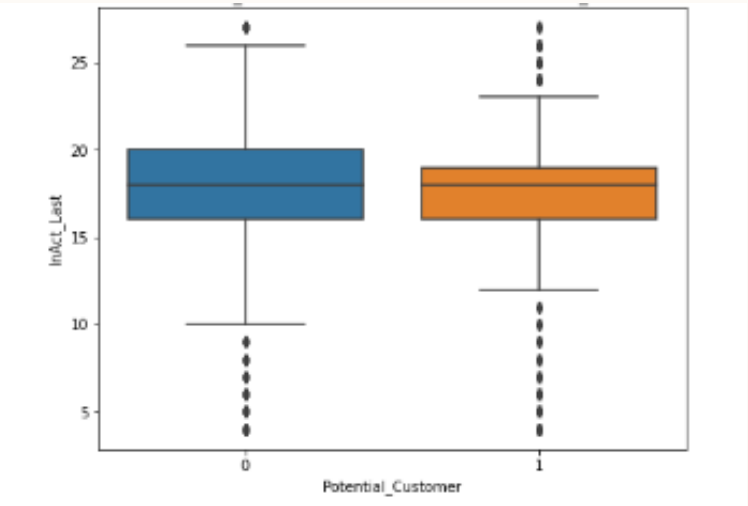
```
fig, ax=plt.subplots(nrows=len(NumCols), figsize=(12,80))
for a in np.arange(len(NumCols)):
    sns.boxplot(x=data.Potential_Customer, y=data[NumCols[a]], hue=data.Potential_Customer,ax=ax[a]);
```



Relationship on Potential_Customer and Pur_3_years_Indirect



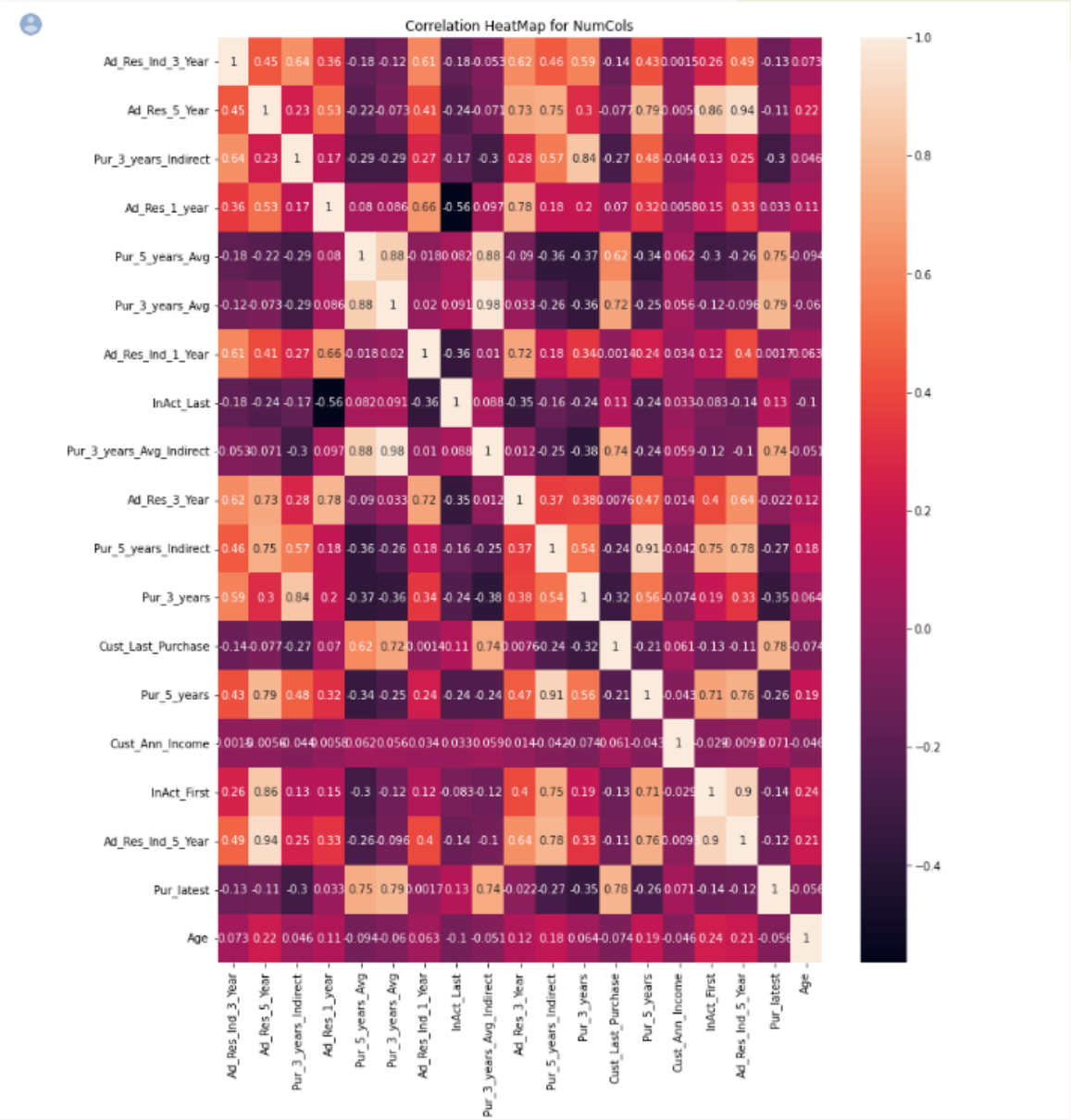
Relationship on Potential_Customer and Pur_3_years



Relationship on Potential_Customer and InAct_Last

CORRELATION BETWEEN NUMERICAL VARIABLE

```
# HeatMap of Numerical Variables - NumCols

plt.figure(figsize=(12,15))
sns.heatmap(data[NumCols].corr(), annot=True, square=False)
plt.title('Correlation HeatMap for NumCols');
```



C) FEATURE SELECTION (no. 7-8)

FEATURE SELECTION

7. Feature Selection

```
# Based on heatmap, some of the numerical data have high correlation thus, it is better to exclude numerical columns with high correlations
# Therefore, use automatic feature selection techniques which is Principle Component Analysis (PCA) for data reduction
# The feature selection is performed as a pre-processing step before doing the actual learning like in part 8.2 Data Scaling
```

4. Split data to train/test

Split the data into 75/25 train/test set. Use random_state=42 and stratify=y

```
[ ] # define X and Y
    X = data.drop('Potential_Customer', axis=1)
    y = data['Potential_Customer']
```

```
[ ] # split data into 75/25 train/test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42, stratify=y)
```

```
[ ] #check if X already exclude 'Potential Customer'
    X_train.head()
```

	Cust_Last_Purchase	Pur_3_years	Pur_5_years	Pur_3_years_Indirect	Pur_5_years_Indirect	Pur_latest
3151	5.0	4.0	29.0	2.0	9.0	6.0
1698	16.0	5.0	22.0	4.0	13.0	8.0
2327	15.0	5.0	8.0	2.0	3.0	10.0
721	2.0	6.0	7.0	3.0	3.0	5.0
811	16.0	3.0	15.0	2.0	9.0	10.0

5 rows × 23 columns

8.1 Dummy Variables

```
[ ] # Change categorical variables to numerical variables (0 or 1) using dummy variables

    X_train_dum = pd.get_dummies(X_train, drop_first=True).reset_index(drop=True)
    X_test_dum = pd.get_dummies(X_test, drop_first=True).reset_index(drop=True)
```

```
[ ] X_train_dum
    X_test_dum
```

Cust_Ann_Income	Status_Cust_E	Status_Cust_F	Status_Cust_L	Status_Cust_N	Status_Cust_S
44789.0	0	0	0	0	0
46901.0	0	0	0	0	1
49219.0	0	0	0	0	1
60483.0	0	0	0	0	0
0.0	0	0	0	0	1
...	...	...	...	...	...
38328.0	0	0	0	0	0
28646.0	0	0	0	1	0
0.0	0	0	0	0	0
80946.0	0	0	0	0	0
81514.0	0	0	0	0	0

FEATURE SELECTION

8.2 Data Scaling

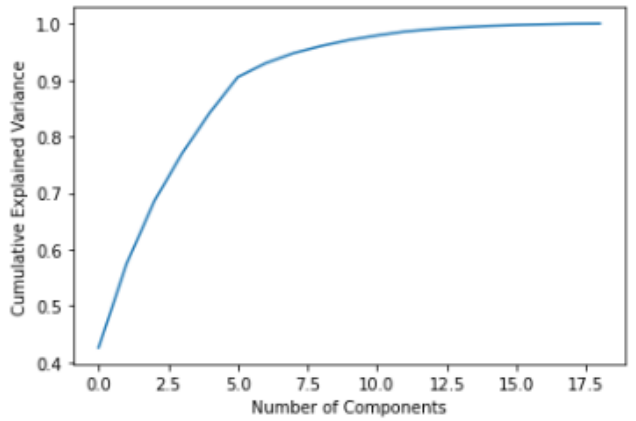
```
[ ] from sklearn.preprocessing import MinMaxScaler,PowerTransformer
    pw_scale = PowerTransformer().fit(X_train_dum[NumCols])

    X_train_pw=pd.DataFrame(pw_scale.transform(X_train_dum[NumCols]), columns=NumCols)
    X_test_pw=pd.DataFrame(pw_scale.transform(X_test_dum[NumCols]), columns=NumCols)

    X_train_pw.index=X_train_dum.index
    X_test_pw.index=X_test_dum.index

    train_pw=pd.concat([X_train_pw, X_train_dum[['Cust_Prop_U', 'Status_Cust_S',
           'Status_Cust_N', 'Gender_M', 'Gender_U','Status_Latest_Ad_1']]],axis=1)

    test_pw=pd.concat([X_test_pw, X_test_dum[['Cust_Prop_U', 'Status_Cust_S',
           'Status_Cust_N', 'Gender_M', 'Gender_U','Status_Latest_Ad_1']]],axis=1)
```

```
    # Standardize data – scaler
[ ] scaler=MinMaxScaler().fit(X_train_dum)
    X_train_Scaled=pd.DataFrame(scaler.transform(X_train_dum), columns=X_train_dum.columns)
    X_test_Scaled=pd.DataFrame(scaler.transform(X_test_dum), columns=X_test_dum.columns)
```

```
[ ] from sklearn.decomposition import PCA
    train_PCA=X_train_Scaled[NumCols]
    test_PCA=X_test_Scaled[NumCols]
```

```
[ ] # PCA is used to reduce the dimension # Reduce to 2-dimension
    pca = PCA(2)
    projected = pca.fit_transform(train_PCA)
```

```
[ ] pca = PCA().fit(train_PCA)
    plt.plot(np.cumsum(pca.explained_variance_ratio_))
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance');
```




```
[ ] pca=PCA(n_components=7).fit(train_PCA)

    PCA_train=pd.DataFrame(pca.transform(train_PCA))
    PCA_train.index=X_train.index

    PCA_test=pd.DataFrame(pca.transform(test_PCA))
    PCA_test.index=X_test.index

    X_train_pca=pd.concat([PCA_train, X_train_dum[['Cust_Prop_U', 'Status_Cust_S',
           'Status_Cust_N', 'Gender_M', 'Gender_U','Status_Latest_Ad_1']]],axis=1)

    X_test_pca=pd.concat([PCA_test, X_test_dum[['Cust_Prop_U', 'Status_Cust_S',
           'Status_Cust_N', 'Gender_M', 'Gender_U','Status_Latest_Ad_1']]],axis=1)
```

D) PREDICTIVE MODELS: (no. 9)

MODEL CREATION

```
[ ] # KNN
    knn_param_grid = {'n_neighbors': np.arange(1, 20),
                  'p': [1,2],
                  'weights': ['uniform','distance']}

    grid = GridSearchCV(KNeighborsClassifier(),
                        knn_param_grid, scoring='roc_auc',
                        cv=3)

    grid.fit(X_train_dum, y_train)

    print("KNN best parameters : ", grid.best_params_)
    knn_model = grid.best_estimator_
    knn_model.fit(X_train_dum,y_train)

    print("KNN best score : ", grid.best_score_)

    knn_pred = knn_model.predict(X_test_dum)
    print("KNN best confustion matrix on the test data \n ", confusion_matrix(y_test, knn_pred))
    print("*******************************************************")
    print("KNN best model precision score on the test data = {:.2f}".format(precision_score(y_test, knn_pred)))
    print("KNN best model recall score on the test data = {:.2f}".format(recall_score(y_test, knn_pred)))
    print("KNN best model F1 score on the test data = {:.2f}".format(f1_score(y_test, knn_pred)))
    print("KNN best model accuracy score on the test data = {:.2f}".format(accuracy_score(y_test, knn_pred)))
```

```
KNN best parameters :  {'n_neighbors': 6, 'p': 1, 'weights': 'distance'}
KNN best score :  0.6656763682963797
KNN best confustion matrix on the test data
  [[321 150]
 [173 261]]
*******************************************************
KNN best model precision score on the test data = 0.64
KNN best model recall score on the test data = 0.60
KNN best model F1 score on the test data = 0.62
KNN best model accuracy score on the test data = 0.64
```

```
] # DecisionTree
  dt_param_grid = {"max_depth" : np.arange(1,10)}

  dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=1),dt_param_grid, scoring='roc_auc',cv=3)

  dt_grid.fit(X_train_dum, y_train)
  print("Decision Tree best parameters: ",dt_grid.best_params_)
  dt_model = dt_grid.best_estimator_
  print("Decision Tree best scores: ",dt_grid.best_score_)
```

```
Decision Tree best parameters:  {'max_depth': 1}
Decision Tree best scores:  0.9996159754224271
```

```
] # Logistic Regression
  lr_param_grid = {'fit_intercept': [True,False],'C': [0.001,0.01,1,10,100],
                'penalty' : ['l1','l2']}

  logr_grid = GridSearchCV(LogisticRegression(solver='liblinear'),
                      lr_param_grid, scoring='roc_auc', cv=3)

  logr_grid.fit(X_train_dum, y_train)
  print("Logistic Regression best parameters: ", logr_grid.best_params_)
  logr_model = logr_grid.best_estimator_
  print("Logistic Regression best scores: ", logr_grid.best_score_)
```

```
Logistic Regression best parameters:  {'C': 1, 'fit_intercept': True, 'penalty': 'l1'}
Logistic Regression best scores:  1.0
```

E) MODEL EVALUATION (no. 10)

Creating a function to run models on dataset

GridSearchCV allows to search for the best parameters for each model in order to achieve the best scores

```
# define new function 'best_model'

def best_model(X_train, X_test, y_train, y_test, model, params, CV = 3, **kwargs):
    param_grid = params
    grid = GridSearchCV(model,param_grid,cv = CV,
                        scoring = 'roc_auc', return_train_score = True)
    grid.fit(X_train, y_train)
    best_par = grid.best_params_
    best_model = grid.best_estimator_.fit(X_train, y_train)
    y_pred = best_model.predict(X_test)
    f1 = f1_score(y_test, y_pred)
    best_sc = grid.best_score_
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)
    return best_par, best_sc, f1, conf_matrix, class_report
```

Creating the parameters for each model to run in the function

Three training & test sets are selected for each model to determine which values yield the best scores

```
train_list = [X_train_dum, X_train_pw, PCA_train]
test_list = [X_test_dum, X_test_pw, PCA_test ]
data_names = ['Original','Power_transform', 'PCA_transform']
model_names = ['KNN','DecisionTree','Logistic Regression','SVM']
models=[KNeighborsClassifier(), DecisionTreeClassifier(), LogisticRegression(solver='saga', max_iter=800), SVC()]

knn_param_grid = {'n_neighbors' :np.arange(10, 40), 'weights':['uniform','distance'],'p': [1,2]}

dt_param_grid = {'max_depth': np.arange(1,10)}

lr_param_grid = {'C':[0.0001, 0.001, 0.004, 0.01, 0.04, 1, 10, 100], 'penalty':['l1', 'l2']}

svc_param_grid = {'C':[0.001, 0.004, 0.01, 0.04, 0.1, 1, 10], 'kernel':['liner', 'rbf'], 'gamma':[0.01, 0.1, 1]}

params=[knn_param_grid, dt_param_grid, lr_param_grid, svc_param_grid ]
```

Creating another function to illustrate the result obtained from executing each model based on the type of train & test set

```
# define new function 'Report'

def Report(data_name,model_name, model):
    print("""
Data : {}
Model : {}
Best_parameters :
{}

Best_Score :
{}

F1 Score :
{}

Confusion_Matrix :
{}

Classification report :
{}
-----------------------------------------------
""".format(data_name, model_name, model[0], model[1],model[2],model[3],model[4]))
```

Results obtained from executing both functions created (Type of train & test set refers to Power Transform and model used was KNN)

```
knn_model_1 = best_model(train_list[1], test_list[1], y_train, y_test, models[0], params = params[0])
Report(data_names[1], model_names[0], knn_model_1)
```

```
Data : Power_transform
Model : KNN
Best parameters :
{'n_neighbors': 38, 'p': 2, 'weights': 'distance'}

Best_Score :
0.9995068865597262

F1 Score :
0.9862700228832951

Confusion_Matrix :
[[462   9]
 [  3 431]]

Classification report :
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       471
           1       0.98      0.99      0.99       434

    accuracy                           0.99       905
   macro avg       0.99      0.99      0.99       905
weighted avg       0.99      0.99      0.99       905
```

Best parameters for KNN to obtain 0.99 score

Results obtained from executing both functions created (Type of train & test set refers to Original and model used was Decision Tree)

```
dt_model_0 = best_model(train_list[0], test_list[0], y_train, y_test, models[1], params = params[1])
Report(data_names[0], model_names[1], dt_model_0)
```

```
Data : Original
Model : DecisionTree
Best parameters :
{'max_depth': 1}

Best_Score :
0.9996159754224271

F1 Score :
1.0

Confusion_Matrix :
[[471   0]
 [  0 434]]

Classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       471
           1       1.00      1.00      1.00       434

    accuracy                           1.00       905
   macro avg       1.00      1.00      1.00       905
weighted avg       1.00      1.00      1.00       905
```

Best parameter for DT to obtain 1.0 score

Results obtained from executing both functions created (Type of train & test set refers to PCA transform and model used was Logistic Regression)

```
lr_model_2 = best_model(train_list[2], test_list[2], y_train, y_test, models[2], params = params[2])
Report(data_names[2], model_names[2], lr_model_2)
```

```
Data : PCA_transform
Model : Logistic Regression
Best parameters :
{'C': 0.04, 'penalty': 'l2'}

Best_Score :
0.5795298645420395

F1 Score :
0.45682451253481887

Confusion_Matrix :
[[351 120]
 [270 164]]

Classification report :
              precision    recall  f1-score   support

           0       0.57      0.75      0.64       471
           1       0.58      0.38      0.46       434

    accuracy                           0.57       905
   macro avg       0.57      0.56      0.55       905
weighted avg       0.57      0.57      0.55       905
```

Best parameter for LR to obtain 0.57 score

Results obtained from executing both functions created (Type of train & test set refers to Power Transform and model used was SVM)

```
[75] svm_model_1 = best_model(train_list[1], test_list[1], y_train, y_test, models[3], params = params[3])
     Report(data_names[1], model_names[3], svm_model_1)
```

```
Data : Power_transform
Model : SVM
Best_parameters :
{'C': 0.04, 'gamma': 0.01, 'kernel': 'rbf'}

Best_Score :
1.0

F1 Score :
1.0

Confusion_Matrix :
[[471   0]
 [  0 434]]

Classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       471
           1       1.00      1.00      1.00       434

    accuracy                           1.00       905
   macro avg       1.00      1.00      1.00       905
weighted avg       1.00      1.00      1.00       905
```

Best parameter for SVM to obtain 1.0 score

F) CHOOSE THE BEST MODEL

ML MODELS:

KNN

DECISION TREE

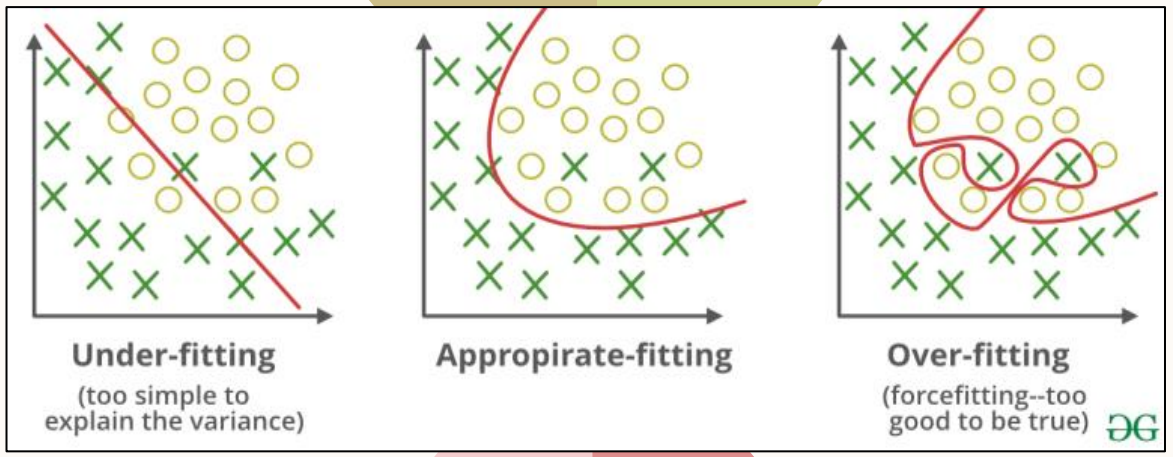
LOGISTIC REGRESSION

SVM

Best Model chosen was using KNN, whereby the type of data would be transformed using PowerTransformer()

```
knn_model_1 = best_model(train_list[1], test_list[1], y_train, y_test, models[0], params = params[0])
Report(data_names[1], model_names[0], knn_model_1)
```

```
Data : Power_transform
Model : KNN
Best_parameters :
{'n_neighbors': 38, 'p': 2, 'weights': 'distance'}

Best_Score :
0.9995068865597262

F1 Score :
0.9862700228832951

Confusion_Matrix :
[[462   9]
 [  3 431]]

Classification report :
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       471
           1       0.98      0.99      0.99       434

    accuracy                           0.99       905
   macro avg       0.99      0.99      0.99       905
weighted avg       0.99      0.99      0.99       905
```



- Despite other models trained with the Power transformed data resulting in scores of 1.0, the probable cause would due to overfitting of data - No training error
- Hence, selecting the KNN model which yielded 0.99 accuracy score would be the best even if overfitting still occurs but not nearly severe as other models
- Occurrence of overfitting maybe due to noise & inaccurate values in dataset

When a model fits the training data perfectly, it probably means it is "overfit" and will not perform well with new data set

THANK YOU :)