EDA, FE & Visualization, on Training Dataset

```
import numpy as np
np.random.seed(123)
import pandas as pd
import pandas.testing as tm
import pandas.util.testing as tm
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from matplotlib.pyplot import figure
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFECV
from sklearn.feature_selection import RFE
from statsmodels.stats.outliers_influence import variance_inflation_factor
from tqdm import tqdm_notebook
from sklearn.decomposition import TruncatedSVD
from pandas.plotting import scatter_matrix
!pip install mglearn
import mglearn
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import FastICA
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD
```

1: loading the dataset

```
# loading the train and test dataset
df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/22 Case Study 1/mercedes-benz

```

```
print(df_1.shape)
df_1=df_1.sort_values(by=['y'])
df_1.head()
```

```
(4209, 378)
```

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16
1594	3185	72.11	az	s	as	f	d	ad	j	n	0	0	0	0	1	0	0
1871	3747	72.50	az	l	n	f	d	ah	g	s	0	0	0	0	0	0	0

- This dataset has 8 categorical features
- and 369 numerical/ Binary features
- That means those contain either 0 or 1

```
1 df_1.describe()
```

	ID	y	X10	X11	X12	X13	
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.00
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.42
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.49
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.00
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.00
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.00
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.00
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.00

8 rows × 370 columns

2: % Null/Missing values in the dataset

- Due to improper handling of missing values, the results obtained will differ from ones where missing values are present
- That missing data can be left or do data imputation to replace them
- In case of multivariate analysis, if there is a larger no of missing values, then it can be better to drop those cases(rather than to do imputation) and replace them
- On other hand in univariate analysis, imputation can decrease the amount of bias in the data, if the values are missing at random
- https://web.stanford.edu/class/stats202/content/lec25-cond.pdf

```
1 df_1.isnull().sum().sum()
```

```
0
```

- From the above output we got to know that there are no Null/Missing values in this dataset

- Percentage of Missing values in this dataset is Zero => 0%

```
x_1=df_1.drop(['y'], axis=1)
y=df_1['y']
print(x_1.shape, y.shape)
```

```
(4209, 377) (4209,)
```

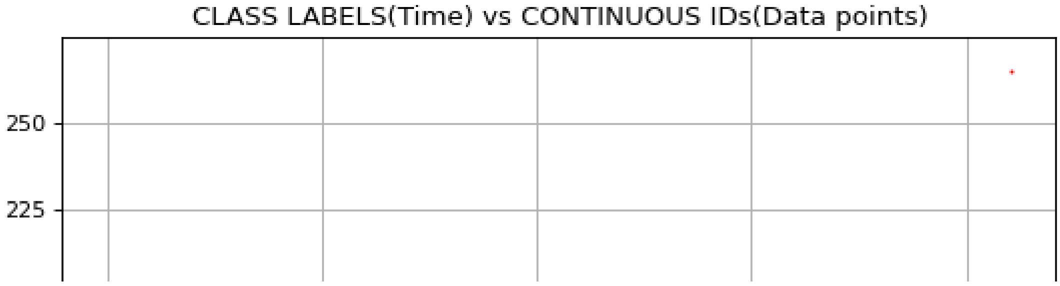
3: Data Visualization

- First let's take only Class variable and plot it on y_axis and it's resettled indices on the x_axis (array of numbers upto len(y))
- Because id's are not continuous unit's
- Over plotting is one of the most common problem in dataviz. When your dataset is big, dots of your scatterplot tend overlap, hence we reduced the size of dots to accomodate more number of dots in a unit area

```
print(len(y))
p=np.arange(len(y))
```

```
4209
```

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')#https://stackove
# plot in seaborn is not so effective so we will plot in matplotlib
# plot using the Matplotlib
plt.plot(p,y, 'ro', markersize=0.7 )# https://python-graph-gallery.com/134-how-to-avoid
plt.ylabel('TOTAL TIME')
plt.xlabel('Data points')
plt.grid()
plt.title('CLASS LABELS(Time) vs CONTINUOUS IDs(Data points)')
plt.show()
```



- From the above diagram we can see that the class label(y-time) looks like a line
- But a small portion of points at the ends are not on the line
- and also there's only one point whose time is above 250 which is an outlier
- Because of all the class labels not lying on a line, the Metric R^2 square won't have large values, it's very sensitive to outliers, as SSres increases
- The best possible R^2 Square value is 1.0

4: Plotting the PDF of Class variable

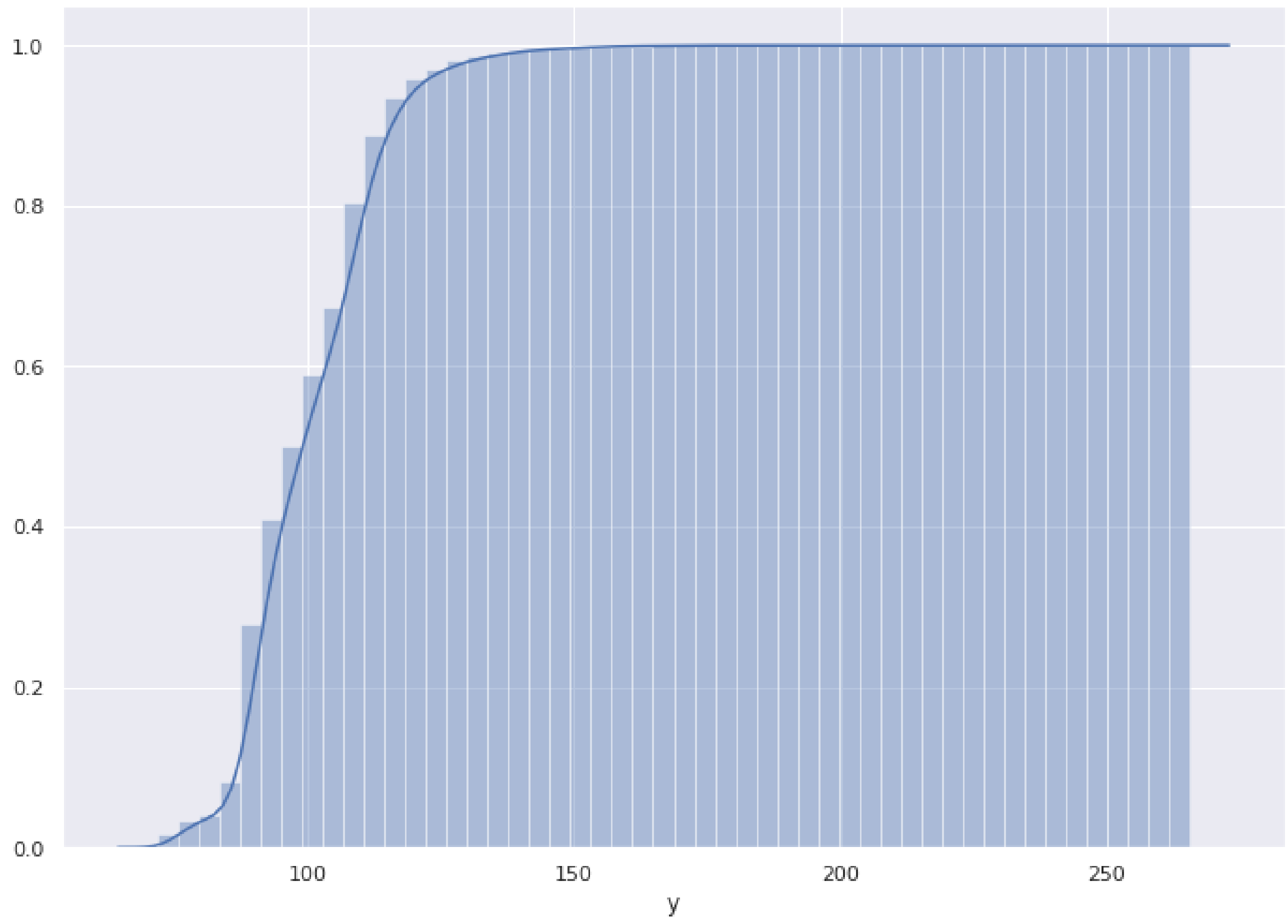
```
sns.set(rc={'figure.figsize':(11.7,8.27)})#https://stackoverflow.com/questions/31594549
sns.distplot(y, rug=True, hist=True)

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4fafff1160>
```

5: Plotting the CDF of Class variable


```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.distplot(y, hist_kws=dict(cumulative=True), kde_kws=dict(cumulative=True)) #https:/
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4fb00eb8d0>
```



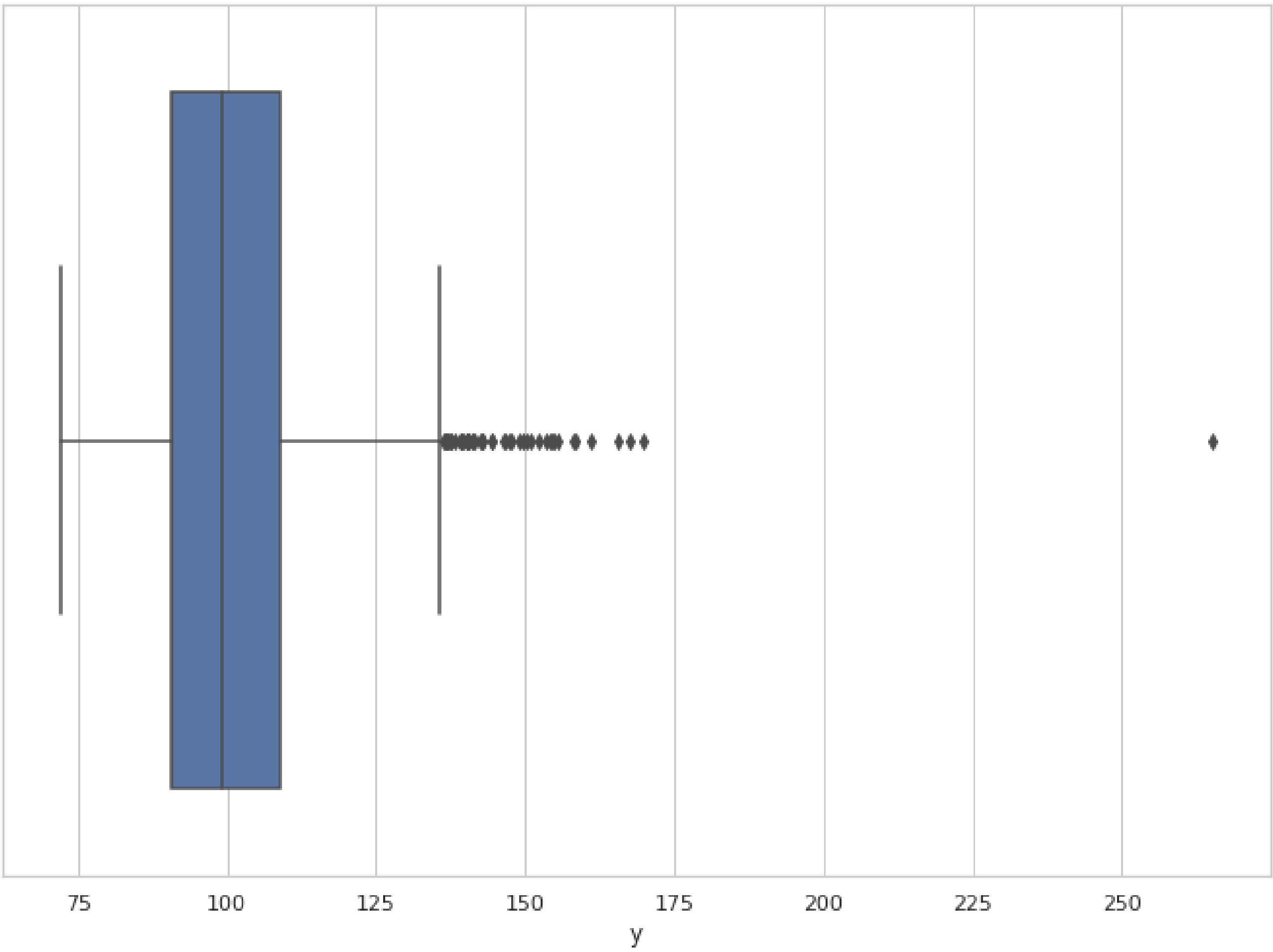
- From the PDF and CDF we can see that,

1. Almost all datapoints have Class variable below 140
2. so the points having class label more than 140 can be considered as outliers
3. Outliers must be discarded, because the metric R^2 is sensitive to outliers

6: BoxPlot Univariate analysis, Y

```
import seaborn as sns
sns.set(style="whitegrid")
ax = sns.boxplot(x=df_1["y"])
```

```
ax = sns.boxplot(x=df_1['y'])
```



- BoxPlot drawn with respect to class label, very beautifully shows the distribution of data based on a five numbered summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”) and here we can consider the values which are larger that max value as outliers

7: Converting Categorical features to numerical features

```
# converting categorical features to numerical features
x_dummies=pd.get_dummies(df_1, prefix_sep='_', drop_first=True)
x_dummies.head()
#https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.get_dummies.html
#https://towardsdatascience.com/encoding-categorical-features-21a2651a065c
```

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X
1594	3185	72.11	0	0	0	0	1	0	0	0	0	0	0	0	0	
1871	3747	72.50	0	0	0	0	0	0	0	0	0	0	0	0	0	

```
x_dummies.corr().abs()
```

	ID	y	X10	X11	X12	X13	X14	X15	X
ID	1.000000	0.055108	0.001602	NaN	0.058988	0.031917	0.025438	0.002237	0.036
y	0.055108	1.000000	0.026985	NaN	0.089792	0.048276	0.193643	0.023116	0.048
X10	0.001602	0.026985	1.000000	NaN	0.033084	0.028806	0.100474	0.002532	0.005
X11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
X12	0.058988	0.089792	0.033084	NaN	1.000000	0.214825	0.246513	0.006212	0.014
...	...	...	...	...	...	...	...	...	
X8_u	0.012768	0.006558	0.019807	NaN	0.038434	0.025157	0.026224	0.003719	0.008
X8_v	0.000879	0.022090	0.015636	NaN	0.006170	0.110324	0.031928	0.004793	0.010
X8_w	0.012303	0.026756	0.013701	NaN	0.037291	0.035523	0.084117	0.004819	0.099
X8_x	0.010427	0.026395	0.005278	NaN	0.006457	0.013604	0.052195	0.003488	0.008
X8_y	0.011479	0.010169	0.019549	NaN	0.012618	0.014131	0.015653	0.003671	0.008

557 rows × 557 columns

```
print(x_dummies.corr().abs()['y'])
```

```
ID      0.055108
y       1.000000
X10     0.026985
X11          NaN
X12     0.089792
          ...
X8_u    0.006558
X8_v    0.022090
X8_w    0.026756
X8_x    0.026395
X8_y    0.010169
Name: y, Length: 557, dtype: float64
```

8: Checking for columns with Unique values

```
# here correlation of x11 with class label y is NaN because that column contains
# only zeros
# hence we need to check for columns with only zeros and delete them
print(x_dummies.any())#https://pandas.pydata.org/pandas-docs/stable/reference/api/panda
```

```
ID        True
y         True
X10       True
X11      False
X12       True
         ...
X8_u      True
X8_v      True
X8_w      True
X8_x      True
X8_y      True
```

```
# creating a list of columns which have only zeros
zeros=[]
for i,j in x_dummies.any().items():#https://pandas.pydata.org/pandas-docs/stable/refere
    if j==False:
        zeros.append(i)
```

```
zeros # 'X339' is missing in my code
# we need to drop these columns
```

```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

- Dropping columns with only zeros

```
x_dummies = x_dummies.drop(zeros, axis=1)
```

9: Dropping outliers => From the conclusions drawn out of BOXPLOT, PDF, CDF

```
x_filtered= x_dummies[x_dummies['y']>70]#https://www.geeksforgeeks.org/drop-rows-from-t
x_filtered= x_filtered[x_filtered['y']<150]
x_filtered.describe()
```

11: Bivariate analysis for the top 20 features

```
! pip install mglearn
```

```
from pandas.plotting import scatter_matrix
import mglearn
sns_df= x_filtered[top_20_features]
sns_df=sns_df.drop('ID',axis=1)
sns_df['y']=y
grr = scatter_matrix(sns_df,c =y,figsize = (100,100),marker = 'o',
                        hist_kwds={'bins':20},s=60,alpha=.8,cmap = mglearn.cm3)
plt.show()
```

	ID	y	X10	X12	X13	X14	
count	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	41
mean	4209.773724	100.439938	0.013352	0.074392	0.057940	0.428231	
std	2437.897217	11.994753	0.114792	0.262439	0.233658	0.494881	
min	0.000000	72.110000	0.000000	0.000000	0.000000	0.000000	
25%	2096.250000	90.800000	0.000000	0.000000	0.000000	0.000000	
50%	4224.000000	99.090000	0.000000	0.000000	0.000000	0.000000	

```
print(x_filtered.shape)
y=x_filtered['y']

```

```
(4194, 545)
```

```
x_filtered=x_filtered.drop(['y'], axis=1)
print(x_filtered.shape, y.shape)
print(type(x_filtered), type(y))
```

```
(4194, 544) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

- Splitting data into Train and Test

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_filtered, y, test_size=0.2, rando
```

10: Feature selection using recursive feature elimination

```
from sklearn.feature_selection import RFECV
from sklearn.feature_selection import RFE
```

RFECV using RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(bootstrap=True,
 max_depth=None, max_features='auto', max_leaf_nodes=None,
 min_impurity_decrease=0.0, min_impurity_split=None,
 min_samples_leaf=28, min_samples_split=111,
 min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
 oob_score=False, random_state=25, verbose=0, warm_start=False)
```

```
# RFECV using RandomForestRegressor
selector = RFECV(clf)
selector = selector.fit(x_filtered, y)
important_features= x_filtered.columns[selector.get_support()]
important_features
```

```
Index(['X314'], dtype='object')
```

RFECV using XGBRegressor

```
# RFECV using XGBRegressor
import xgboost as xgb
clf_1=xgb.XGBRegressor()
selector = RFECV(clf_1)
selector = selector.fit(x_filtered, y)
important_features= x_filtered.columns[selector.get_support()]
important_features
```

```
important_features
```

```
Index(['X29', 'X314', 'X315'], dtype='object')
```

RFECV using DecisionTreeRegressor

```
# RFECV using DecisionTreeRegressor
from sklearn.tree import DecisionTreeRegressor
clf_2=DecisionTreeRegressor(max_depth=5 )
selector = RFECV(clf_2)
selector = selector.fit(x_filtered, y)
important_features= x_filtered.columns[selector.get_support()]
important_features
```

```
Index(['X314'], dtype='object')
```

- From the output of above three cell we have learn't that X314, X315 and X29 are the most important features and X314 is more important that X315 and X29

RFE using RandomForestRegressor

- Using Recursive feature elimination we will find the top 20 important features and perform bivariate analysis on them

```
from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor(bootstrap=True,
```

```
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=28, min_samples_split=111,
min_weight_fraction_leaf=0.0, n_estimators=121, n_jobs=-1,
oob_score=False, random_state=25, verbose=0, warm_start=False)
```

```
selector = RFE(clf, n_features_to_select=20)
selector = selector.fit(x_filtered, y)
top_20_features = x_filtered.columns[selector.get_support()]
top_20_features
```

```
Index(['ID', 'X29', 'X48', 'X54', 'X64', 'X76', 'X118', 'X119', 'X127', 'X136',
       'X189', 'X232', 'X263', 'X279', 'X311', 'X314', 'X315', 'X1_aa', 'X6_g',
       'X6_j'],
      dtype='object')
```

```
# the above are the features when automatic feature selection is alloted
# these features were selected as top 20 important features when number of features is
top_20_features= ['ID', 'X29', 'X54', 'X58', 'X64', 'X118', 'X127', 'X132', 'X189',
       'X218', 'X224', 'X273', 'X311', 'X314', 'X315', 'X0_z', 'X1_aa', 'X2_s',
       'X6_g', 'X6_j']
type(top_20_features)ho
```

```
list
```

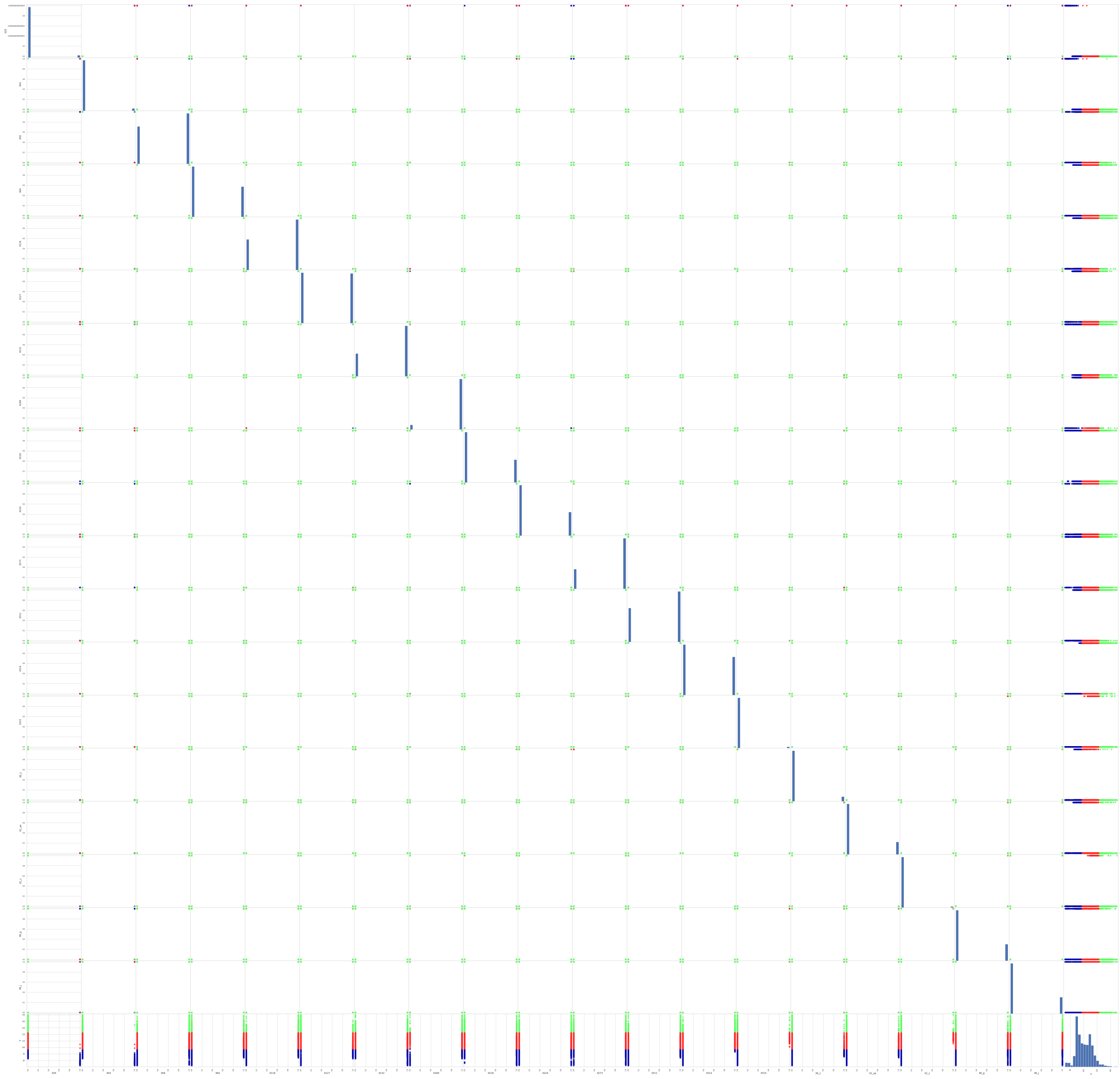
```
type(x_filtered)
```

```
pandas.core.frame.DataFrame
```

```
x_filtered[top_20_features]
```

	ID	X29	X54	X58	X64	X118	X127	X132	X189	X218	X224	X273	X311	X314
1594	3185	1	1	0	0	0	0	1	0	1	1	0	0	0
1871	3747	1	1	0	0	0	0	1	0	1	0	1	0	0
1803	3616	1	1	0	0	0	0	1	0	1	0	1	0	0
2926	5865	1	1	1	0	0	0	1	0	1	1	1	0	0
1458	2902	1	1	0	0	0	0	1	0	1	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2905	5820	0	0	0	1	0	0	1	1	0	1	1	0	1
681	1322	0	0	1	0	1	0	0	1	1	0	1	1	1
2376	4762	0	0	1	1	1	0	1	1	1	1	0	1	1
4176	8344	0	0	1	0	1	0	0	1	0	1	0	1	1
1141	2264	0	0	1	0	1	0	0	0	0	0	1	0	1

4194 rows × 20 columns



- a plot similar to this plot is plotted below and is explained

```
sns_df.columns
```

```
Index(['X29', 'X54', 'X58', 'X64', 'X118', 'X127', 'X132', 'X189', 'X218',
       'X224', 'X273', 'X311', 'X314', 'X315', 'X0_z', 'X1_aa', 'X2_s', 'X6_g',
       'X6_j', 'y'],
      dtype='object')
```

```
print(type(top_20_features))
top_20_features
```

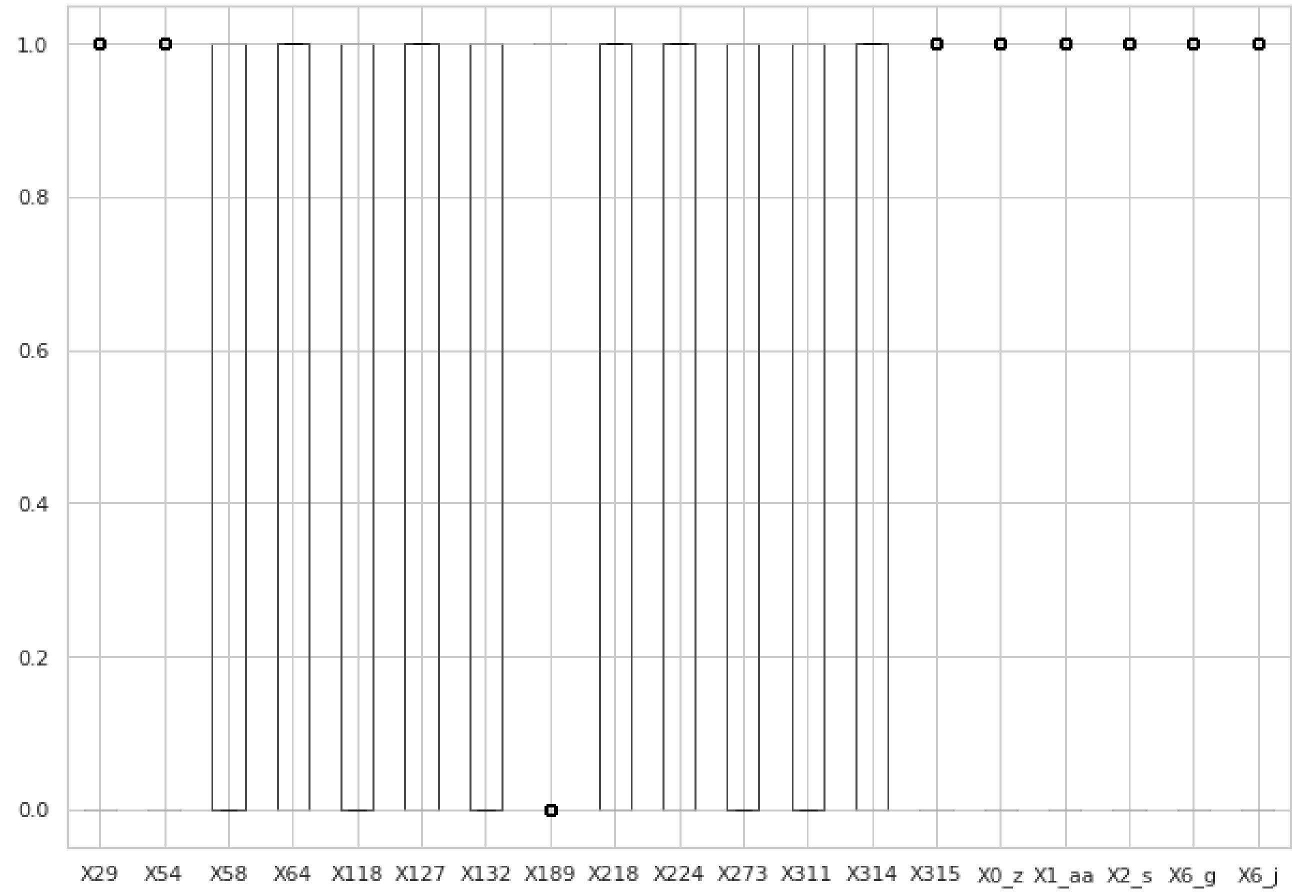
```
<class 'list'>
['ID',
 'X29',
 'X54',
 'X58',
 'X64',
 'X118',
 'X127',
 'X132',
 'X189',
 'X218',
 'X224',
 'X273',
```

- Boxplot of top 20 features

```
 'X315',
```

```
sns_df=sns_df.drop('y',axis=1)
sns_df.boxplot(column=top_20_features.remove('ID'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe544227dd8>
```




```
sns_df.X54.value_counts()
```

```
0    4012
1     182
Name: X54, dtype: int64
```

- From the above plot we can see that out of these 20 important features only X58, X64, X118, X127, X132, X218, X224, X273, X311, X314 have number of zeros and ones

balanced, other have either almost all ones or almost all zeros
- hence those other 10 features are not that important, because they don't provide useful information
- they are similar to columns which have all zero's and all one's

---

12: Detecting Multicollinearity using VIF (Variable Inflation Factor)

https://www.sigmamagic.com/blogs/what-is-variance-inflation-factor/#:~:text=If%20there%20is%20perfect%20correlation,to%20the%20presence%20of%20multicollinearity.

https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/

- VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable
- VIF - Conclusion

  1 = No Multicollinearity

  4-5 = Moderate

  10 or greater = Severe

- Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model
- This means that one independent variable can be predicted from another independent variable in a regression model
- This can be a problem in a regression model because we would not be able to distinguish between the individf_1.isnull().sum().sum()dual effects of the independent variables on the dependent variable.
- Multicollinearity may not affect the accuracy of the model as much. But we might lose reliability in determining the effects of individual features on the model. and that can be a problem when it comes to interpretability

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(X):

    # Calculationg VIF
    vif=pd.DataFrame()
    vif['variables']=X.columns
    vif['VIF']= [variance_inflation_factor(X.values, i) for i in tqdm_notebook(range(X.

```

```
    return (vif)

```

```
vif_values=calc_vif(x_filtered)
```

```
HBox(children=(FloatProgress(value=0.0, max=544.0), HTML(value='')))
```

```
vif_values.head()
```

	variables	VIF
0	ID	487.143315
1	X10	inf
2	X12	inf
3	X13	inf
4	X14	inf

```
vif_values=vif_values.sort_values(by='VIF', ascending=True)
```

```
vif_values.head()
```

	variables	VIF
173	X190	1.077204
305	X332	1.101729
31	X42	1.125967
267	X288	1.128819
91	X104	1.170182

```
lst_variables= list(vif_values['variables'])
lst_vif= list(vif_values['VIF'])
vif_dict = dict(zip(lst_variables, lst_vif))
inf_indices=[]
count=0
for k,v in vif_dict.items():
    if np.isinf(v):
        inf_indices.append(k)
        count+=1
```

```
# removing top_20_features from the features that need to be dropped from VIF
inf_indices=list(set(inf_indices)-set(top_20_features))
```

```
import json
with open('/content/drive/My Drive/Colab Notebooks/22 Case Study 1/inf_indices.txt', 'w
    f.write(json.dumps(inf_indices))
```

```
with open('/content/drive/My Drive/Colab Notebooks/22 Case Study 1/inf_indices.txt', 'r
    checc=json.loads(f.read())

```

```
inf_indices=checc
```

```
print(x_filtered.shape)
x_filtered = x_filtered.drop(inf_indices, axis=1)
print(x_filtered.shape)
```

```
(4194, 544)
(4194, 171)
```

13: Finding the "k" in Truncated SVD, Gavish-Donoho method

```
#http://www.pyrunner.com/weblog/2016/08/01/optimal-svht/
# with reference to the research paper
U, s, V = np.linalg.svd(x_filtered)
```

```
print(U.shape, s.shape, V.shape)
```

```
(4194, 4194) (171,) (171, 171)
```

```
def omega_approx(beta):
    """Return an approximate omega value for given beta. Equation (5) from Gavish 2014.
    return 0.56 * beta**3 - 0.95 * beta**2 + 1.82 * beta + 1.43


```

```
beta = min(x_filtered.shape) / max(x_filtered.shape)
tau = np.median(s) * omega_approx(beta)

```

```
print(tau)
print(np.log(tau))
```

```
15.625431208108179
2.74889979256059
```

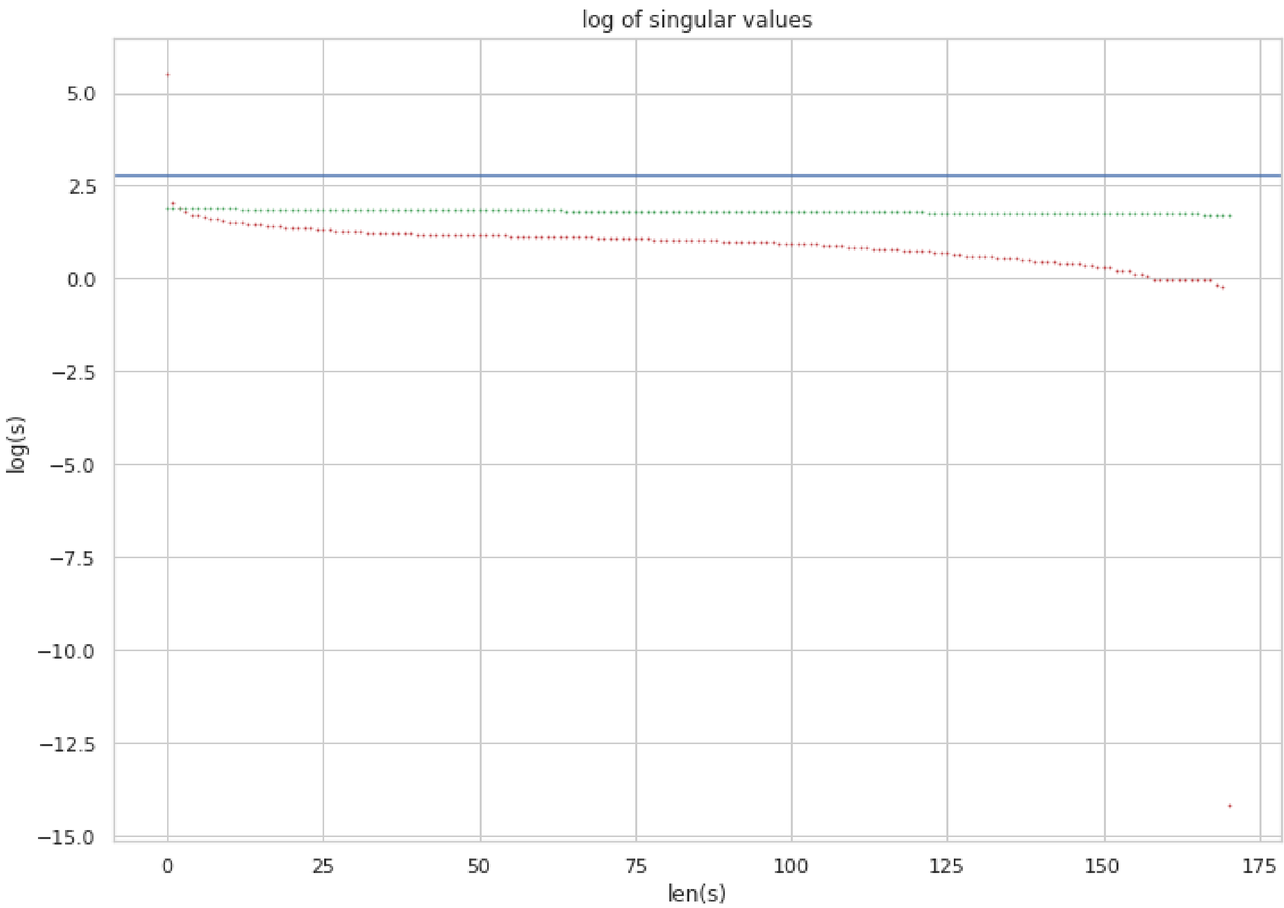
```
# initializing normally distributed matrix of size equal to to size of x_filtered, mean
noise = np.random.normal(0, 1, (4194, 171))
noise.shape
```

```
(4194, 171)
```

```
# getting the singular values of the noise  matrix
U_n, s_n, V_n = np.linalg.svd(noise)
print(U_n.shape, s_n.shape, V_n.shape)
```

```
(4194, 4194) (171,) (171, 171)
```

```
plt.plot(np.arange(s.shape[0]),np.log10(s), 'ro', markersize=0.7 )# https://python-grap
plt.plot(np.arange(s_n.shape[0]), np.log10(s_n), 'go', markersize=0.7)
plt.axhline(y=np.log(tau))
plt.grid()
plt.ylabel('log(s)')
plt.xlabel('len(s)')
plt.grid()
plt.title('log of singular values')
plt.show()
```



- From the graph and the values of tau retrived with reference to the research paper, the number of components of SVD that contain maximum variance is either 1 or 2
- Because 2 points of the singular values of original matrix got deviated from the noise matrix's singlar values, of which one has a very high deviation above the tau value calculated

14: Adding new features to the dataframe using dimensionality reduction techniques

```
from sklearn.decomposition import TruncatedSVD
tsvd= TruncatedSVD(n_components=2, random_state=42)
tsvd_train= tsvd.fit_transform(x_filtered)
#tsvd_test=  tsvd.transform(x_dummies_2)
```

```
tsvd_train.shape
```

```
(4194, 2)
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, random_state=42)
pca_train= pca.fit_transform(x_filtered)
#pca_test= pca.transform(x_dummies_2)
```

```
pca_train.shape
```

```
(4194, 2)
```

```
from sklearn.decomposition import FastICA
ica=FastICA(n_components=2, random_state=42)
ica_train= ica.fit_transform(x_filtered)
#ica_test= ica.transform(x_dummies_2)
```

```
ica_train.shape
```

```
(4194, 2)
```

```
for i in range(0, tsvd_train.shape[1]):
    x_filtered['tsvd_'+str(i)]= tsvd_train[:, i]
    #x_dummies_2['tsvd_'+str(i)]= tsvd_test[:, i]
    x_filtered['pca_'+str(i)]= pca_train[:, i]
    #x_dummies_2['pca_'+str(i)]= pca_test[:, i]
    x_filtered['ica_'+str(i)]= ica_train[:, i]
    #x_dummies_2['ica_'+str(i)]= ica_test[:, i]
```

```
print(x_filtered.shape)#, x_dummies_2.shape)
```

```
(4194, 180)
```

15: Adding new features using the top important features

```
# out of all the top_20_important features, after all the processing ['X315', 'X54', 'X
# the above features are dropped and X64, X218, X224, X273 these four features are pres

```

```
x_filtered['X64 + X218']=x_filtered['X64']+x_filtered['X218']

x_filtered['X218 + X224 + X273']=x_filtered['X218']+x_filtered['X224'] + x_filtered['X2

x_filtered['X64 + X224 + X273']=x_filtered['X64']+x_filtered['X224'] + x_filtered['X273

x_filtered['X314 + X315']=x_filtered['X314']+x_filtered['X315']

x_filtered['X314 + X315 + X29']=x_filtered['X314']+x_filtered['X315'] + x_filtered['X29

x_filtered['X314 + X315 + X118']=x_filtered['X314']+x_filtered['X315'] + x_filtered['X1

x_filtered['X127 + X189']=x_filtered['X127']+x_filtered['X189']

x_filtered['X118 + X54']=x_filtered['X118']+x_filtered['X54']
```

16: Summary of FE and EDA

1. First thing after loading the data, the results of null value analysis said that, data contains 0% null values.
2. From Data visualization, PDF, CDF and Box plot it's confirmed that data contains outliers and were dropped.
3. Using the Recursive Feature Elimination (using RandomForestRegressor from sklearn), top 20 features are calculated and bivariate analysis is done, from which 10 features had balanced 0's and 1's while other 10 have either almost all 0's or 1's which are similar to features with single value, hence can be considered as not important
4. The results of finding Correlation between features, arised the doubt of multicollinearity. For which we have done Multicollinearity detection using the VIF(Variable Inflation Factor) and features which have a VIF value of inf are dropped.
5. To add new features using SVD, calculation of "k" in Truncated SVD is done in 2 ways:

- one is just by plotting the "Explained variance(by the features)" VS "no of Features", and
- the other method is using the Gavish-Donoho method, by plotting the graph of "log of singular values" vs "number of components"

6. Now after dropping the features using VIF excluding the top_20_features
7. using 2 way and 3 way interaction between them new features are generated