

EDA, Visualization & FE.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Settings User

+ Code + Text Reconnect Editing

EDA, FE & Visualization, on Training Dataset

```
[ ] 1 import numpy as np
2 np.random.seed(123)
3 import pandas as pd
4 import pandas.testing as tm
5 import pandas.util.testing as tm
6 import seaborn as sns
7 import matplotlib.pyplot as plt
8 import warnings
9 warnings.filterwarnings("ignore")
10 from matplotlib.pyplot import figure
11 import seaborn as sns
12 from sklearn.model_selection import train_test_split
13 from sklearn.feature_selection import RFEcv
14 from sklearn.feature_selection import RFE
15 from statsmodels.stats.outliers_influence import variance_inflation_factor
16 from tqdm import tqdm_notebook
17 from sklearn.decomposition import TruncatedSVD
18 from pandas.plotting import scatter_matrix
19 !pip install mglearn
20 import mglearn
21 from sklearn.tree import DecisionTreeRegressor
22 import xgboost as xgb
23 from sklearn.ensemble import RandomForestRegressor
24 from sklearn.decomposition import FastICA
25 from sklearn.decomposition import PCA
26 from sklearn.decomposition import TruncatedSVD
```

1: loading the dataset

```
[ ] 1 # loading the train and test dataset
2 df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/22 Case Study 1/mercedes-benz-greener-manufacturing/train.csv/train.csv')
3
```

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

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	X28	X29	X30	X31	X32	X33	X34	X35	X36	X37	X38			
1594	3185	72.11	az	s	as	f	d	ad	j	n	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	0
1871	3747	72.50	az	l	n	f	d	ah	g	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0		
1803	3616	72.94	az	l	n	f	d	ae	a	b	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0			
2926	5865	73.02	az	s	n	c	d	p	d	f	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0			
1458	2902	73.15	az	l	n	f	d	ac	j	s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0			

5 rows × 378 columns

- 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	X14	X15	X16	X17	X18	X19	X20	X21		
count	4209.000000	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000		
mean	4205.960798	100.669318	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603	0.007840	0.099549	0.142789	0.002613	0.086
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872	0.088208	0.299433	0.349899	0.051061	0.281
min	0.000000	72.110000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2095.000000	90.820000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4220.000000	99.150000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	6314.000000	109.010000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

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%

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

```
↳ (4209, 377) (4209,
```

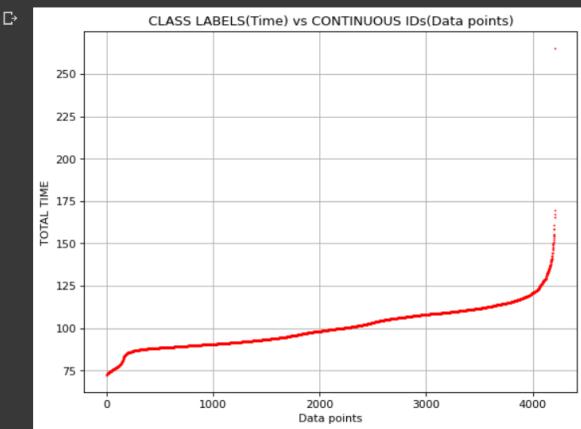
▼ 3: Data Visualization

- First let's take only Class variable and plot it on y_axis and its 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

```
[ ] 1 print(len(y))
2 p=np.arange(len(y))
```

```
↳ 4209
```

```
[ ] 1 from matplotlib.pyplot import figure
2 figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')#https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib
3 # plot in seaborn is not so effective so we will plot in matplotlib
4 # plot using the Matplotlib
5 plt.plot(p,y, 'ro', markersize=0.7)# https://python-graph-gallery.com/134-how-to-avoid-overplotting-with-python/
6 plt.ylabel('TOTAL TIME')
7 plt.xlabel('Data points')
8 plt.grid()
9 plt.title('CLASS LABELS(Time) vs CONTINUOUS IDs(Data points)')
10 plt.show()
```

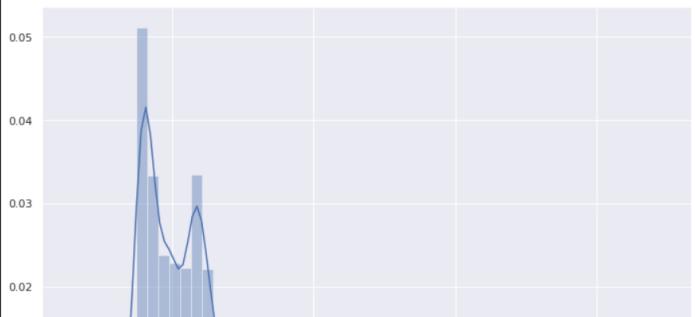


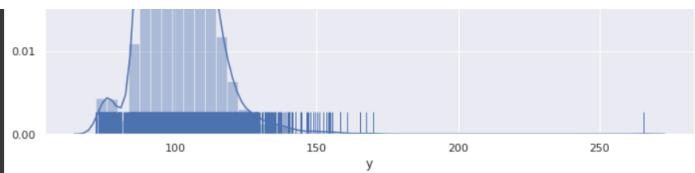
- From the above diagram we can see that the class label(y-time) looks like a line
- But a small portion of points at the ends are not on the line
- and also there's only one point whose time is above 250 which is an outlier
- Because of all the class labels not lying on a line, the Metric R^2 square 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

```
[ ] 1 sns.set(rc={'figure.figsize':(11.7,8.27)})#https://stackoverflow.com/questions/31594549/how-do-i-change-the-figure-size-for-a-seaborn-plot
2 sns.distplot(y, rug=True, hist=True)
3
```

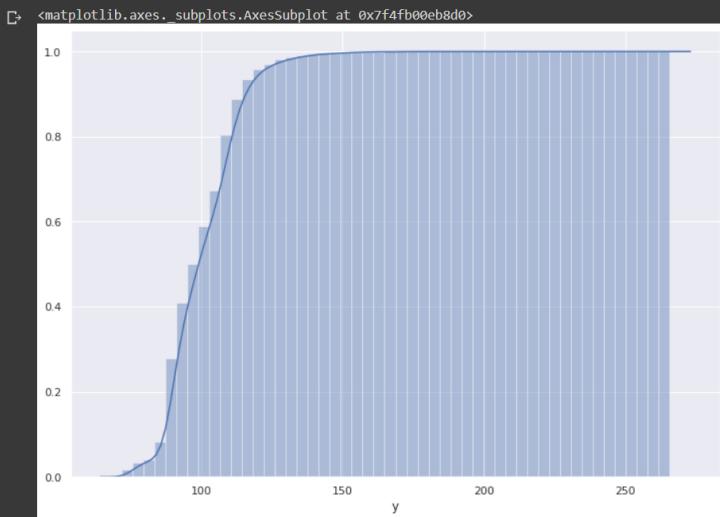
```
↳ <matplotlib.axes._subplots.AxesSubplot at 0x7f4faf1160>
```





5: Plotting the CDF of Class variable

```
[ ] 1 sns.set(rc={'figure.figsize':(11.7,8.27)})
2 sns.distplot(y, hist_kws=dict(cumulative=True), kde_kws=dict(cumulative=True)) #https://geek-answers.github.io/articles/841259/index.html
```

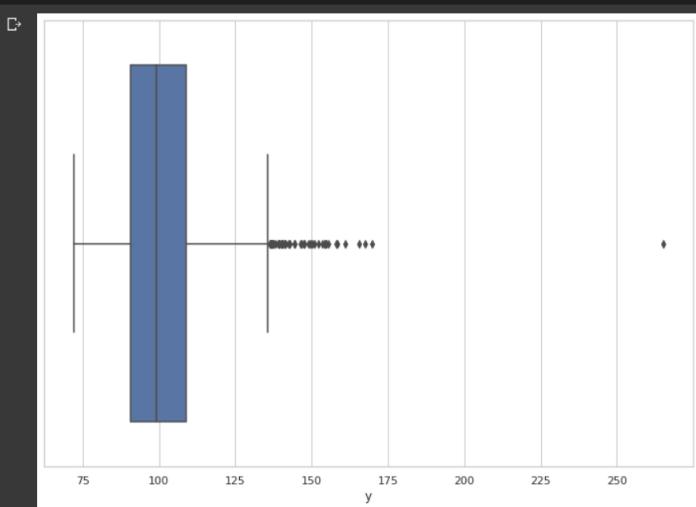


- From the PDF and CDF we can see that,

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

6: BoxPlot Univariate analysis, Y

```
[ ] 1 import seaborn as sns
2 sns.set(style="whitegrid")
3 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 than max value as outliers

7: Converting Categorical features to numerical features

```
[ ] 1 # converting categorical features to numerical features
2 x_dummies=pd.get_dummies(df_1, prefix_sep='_', drop_first=True)
3 x_dummies.head()
4 #https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.get_dummies.html
```

```
5 #https://towardsdatascience.com/encoding-categorical-features-21a2651a065c
```

ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	X28	X29	X30	X31	X32	X33	X34	X35	X36	X37	X38	X39	X40	X41	X42	X43	X44
1594	3185	72.11	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0		
1871	3747	72.50	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0		
1803	3616	72.94	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0		
2926	5665	73.02	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	0	1	0		
1458	2902	73.15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0		

5 rows × 557 columns

ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	
ID	1.000000	0.055108	0.001602	NaN	0.058988	0.031917	0.025438	0.002237	0.036480	0.038171	0.027063	0.012762	0.063216	0.029946	0.096610	0.010612	0.008972	0.048634	0.004214
y	0.055108	1.000000	0.026985	NaN	0.089792	0.048276	0.193643	0.023116	0.048946	0.159815	0.001789	0.168895	0.095148	0.031111	0.094320	0.078676	0.003526	0.021173	0.053483
X10	0.001602	0.026985	1.000000	NaN	0.033084	0.028806	0.100474	0.002532	0.005944	0.010164	0.010323	0.038610	0.047393	0.005944	0.035836	0.016870	0.005067	0.008223	0.070276
X11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
X12	0.058988	0.089792	0.033084	NaN	1.000000	0.214825	0.246513	0.006212	0.014584	0.024937	0.025327	0.094730	0.116280	0.014584	0.087924	0.041391	0.012433	0.020175	0.184599
...	
X8_u	0.012768	0.006558	0.019807	NaN	0.038434	0.025157	0.026224	0.003719	0.008731	0.014930	0.066097	0.005524	0.004064	0.103570	0.022118	0.024781	0.007444	0.012079	0.011617
X8_v	0.000879	0.022090	0.015636	NaN	0.006170	0.110324	0.031928	0.004793	0.010941	0.006195	0.019540	0.016320	0.047612	0.011252	0.064864	0.008043	0.042436	0.000516	0.103647
X8_w	0.012303	0.026756	0.013701	NaN	0.037291	0.035523	0.084117	0.004819	0.099113	0.006363	0.019646	0.009372	0.096730	0.011313	0.011831	0.007518	0.016244	0.015649	0.011595
X8_x	0.010427	0.026395	0.005278	NaN	0.006457	0.013604	0.052195	0.003488	0.008188	0.003538	0.003053	0.023138	0.008739	0.008188	0.015514	0.158781	0.006980	0.011327	0.067652
X8_y	0.011479	0.010169	0.019549	NaN	0.012618	0.014131	0.015653	0.003671	0.008618	0.014735	0.050854	0.007040	0.014782	0.019808	0.000448	0.024458	0.007346	0.011921	0.000561

557 rows × 557 columns

ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	
ID	0.055108	1.000000	0.026985	NaN	0.089792	
y	0.055108	1.000000	0.026985	NaN	0.089792	0.048276	0.193643	0.023116	0.048946	0.159815	0.001789	0.168895	0.095148	0.031111	0.094320	0.078676	0.003526	0.021173	0.053483
X10	0.001602	0.026985	1.000000	NaN	0.033084	0.028806	0.100474	0.002532	0.005944	0.010164	0.010323	0.038610	0.047393	0.005944	0.035836	0.016870	0.005067	0.008223	0.070276
X11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
X12	0.058988	0.089792	0.033084	NaN	1.000000	0.214825	0.246513	0.006212	0.014584	0.024937	0.025327	0.094730	0.116280	0.014584	0.087924	0.041391	0.012433	0.020175	0.184599
...	
X8_u	0.012768	0.006558	0.019807	NaN	0.038434	0.025157	0.026224	0.003719	0.008731	0.014930	0.066097	0.005524	0.004064	0.103570	0.022118	0.024781	0.007444	0.012079	0.011617
X8_v	0.000879	0.022090	0.015636	NaN	0.006170	0.110324	0.031928	0.004793	0.010941	0.006195	0.019540	0.016320	0.047612	0.011252	0.064864	0.008043	0.042436	0.000516	0.103647
X8_w	0.012303	0.026756	0.013701	NaN	0.037291	0.035523	0.084117	0.004819	0.099113	0.006363	0.019646	0.009372	0.096730	0.011313	0.011831	0.007518	0.016244	0.015649	0.011595
X8_x	0.010427	0.026395	0.005278	NaN	0.006457	0.013604	0.052195	0.003488	0.008188	0.003538	0.003053	0.023138	0.008739	0.008188	0.015514	0.158781	0.006980	0.011327	0.067652
X8_y	0.011479	0.010169	0.019549	NaN	0.012618	0.014131	0.015653	0.003671	0.008618	0.014735	0.050854	0.007040	0.014782	0.019808	0.000448	0.024458	0.007346	0.011921	0.000561

8: Checking for columns with Unique values

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

ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27
ID	True	True	True	True	False	True												
y	True	True	True	True	False	True												
X10	True	True	True	True	False	True												
X11	False	True	True	True	False	True												
X12	True	True	True	True	False	True												
...

```
[ ] 1 # creating a list of columns which have only zeros  
2 zeros=[]  
3 for i,j in x_dummies.any().items():#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.iteritems.html  
4     if j==False:  
5         zeros.append(i)
```

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

- Dropping columns with only zeros

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

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

```
[ ] 1 x_filtered= x_dummies[x_dummies['y']>70]https://www.geeksforgeeks.org/drop-rows-from-the-dataframe-based-on-certain-condition-applied-on-a-column/
2 x_filtered= x_filtered[x_filtered['y']<150]
3 x_filtered.describe()
```

	ID	y	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22
count	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000
mean	4209.773724	100.439938	0.013352	0.074392	0.057940	0.428231	0.000477	0.002623	0.007630	0.007868	0.099666	0.142823	0.002623	0.086791
std	2437.897217	11.994753	0.114792	0.262439	0.233658	0.494881	0.021835	0.051152	0.087026	0.088365	0.299590	0.349934	0.051152	0.281562
min	0.000000	72.110000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2096.250000	90.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	4224.000000	99.090000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	6316.750000	108.967500	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	8417.000000	149.630000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 545 columns

```
[ ] 1 print(x_filtered.shape)
2 y=x_filtered['y']
3
```

4 (4194, 545)

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

4 (4194, 544) (4194,)

<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>

- Splitting data into Train and Test

```
[ ] 1 from sklearn.model_selection import train_test_split
2 x_train, x_test, y_train, y_test = train_test_split(x_filtered, y, test_size=0.2, random_state=42)
```

10: Feature selection using recursive feature elimination

```
[ ] 1 from sklearn.feature_selection import RFECV
2 from sklearn.feature_selection import RFE
```

RFECV using RandomForestRegressor

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

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

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

RFECV using XGBRegressor

```
[ ] 1 # RFECV using XGBRegressor
2 import xgboost as xgb
3 clf_1=xgb.XGBRegressor()
4 selector = RFECV(clf_1)
5 selector = selector.fit(x_filtered, y)
6 important_features= x_filtered.columns[selector.get_support()]
7 important_features
```

8 important_features

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

RFECV using DecisionTreeRegressor

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

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

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

- From the output of above three cell we have learnt that X314, X315 and X29 are the most important features and X314 is more important than X315 and X29

▼ RFE using RandomForestRegressor

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

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

[ ] 1 selector = RFE(clf, n_features_to_select=20)
2 selector = selector.fit(x_filtered, y)
3 top_20_features = x_filtered.columns[selector.get_support()]
4 top_20_features

[2]: 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')

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

[3]: list

[ ] 1 type(x_filtered)
[4]: pandas.core.frame.DataFrame

[ ] 1 x_filtered[top_20_features]
```

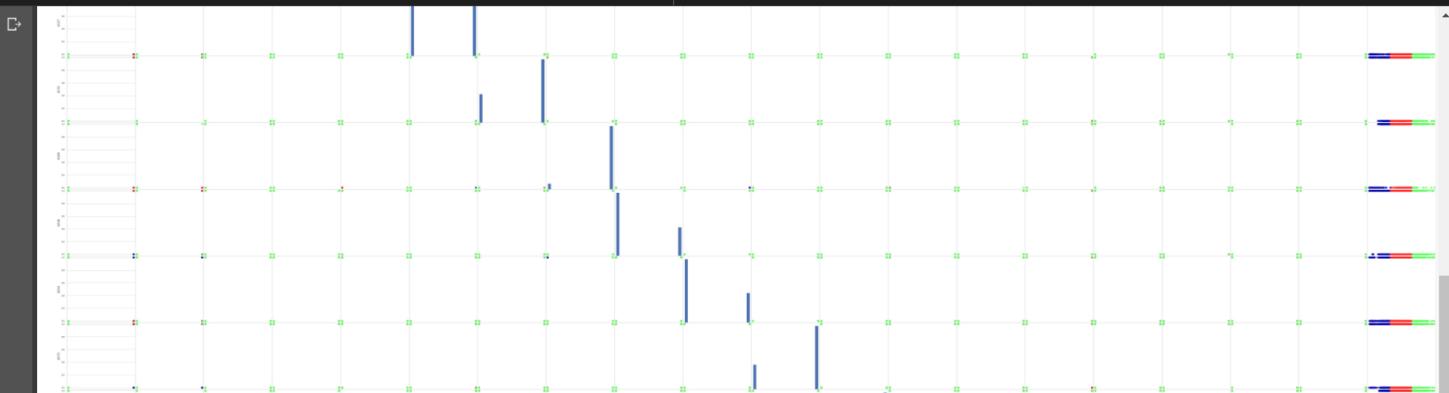
	ID	X29	X54	X58	X64	X118	X127	X132	X189	X218	X224	X273	X311	X314	X315	X0_z	X1_aa	X2_s	X6_g	X6_j	
1594	3185	1	1	0	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	1	
1871	3747	1	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0
1803	3616	1	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
2926	5865	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0
1458	2902	1	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1
...
2905	5820	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0	0	1
681	1322	0	0	1	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1	0	0
2376	4762	0	0	1	1	1	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0
4176	8344	0	0	1	0	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0	1
1141	2264	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1

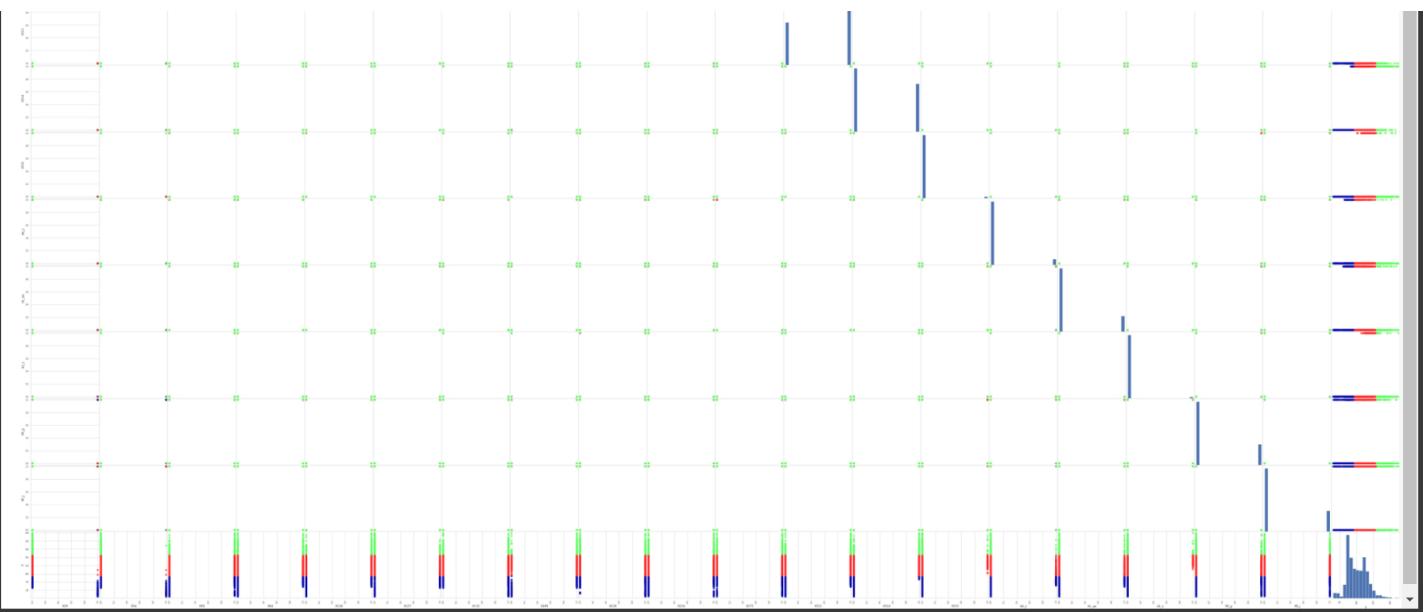
4194 rows × 20 columns

▼ 11: Bivariate analysis for the top 20 features

```
[ ] 1 ! pip install mglearn

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



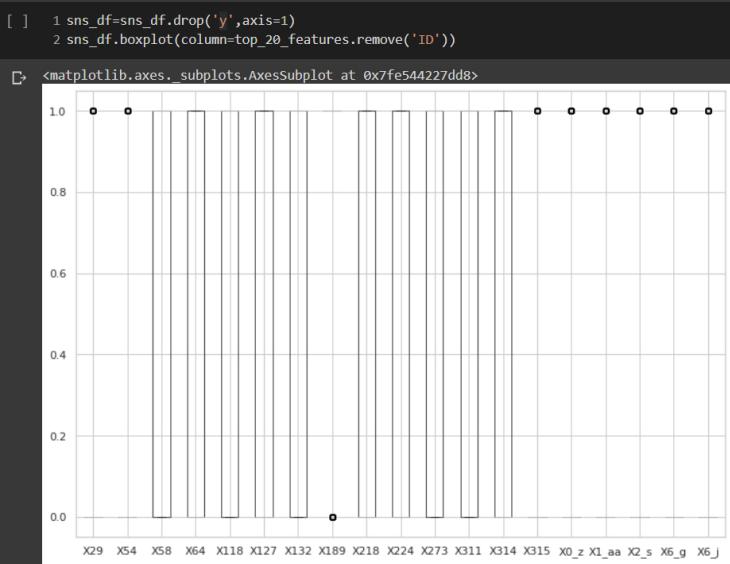


- a plot similar to this plot is plotted below and is explained

```
[ ] 1 sns_df.columns
[ ] 2 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')
```

```
[ ] 1 print(type(top_20_features))
2 top_20_features
[ ] <class 'list'>
['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']
```

- Boxplot of top 20 features



```
[ ] 1 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.sigmaprime.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 individual 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

```
[ ] 1 from statsmodels.stats.outliers_influence import variance_inflation_factor
2 def calc_vif(X):
3
4     # Calculating VIF
5     vif=pd.DataFrame()
6     vif['variables']=X.columns
7     vif['VIF']= [variance_inflation_factor(X.values, i) for i in tqdm_notebook(range(X.shape[1]))]
8
9     return (vif)
10
```

```
[ ] 1 vif_values=calc_vif(x_filtered)
2+ HBox(children=(FloatProgress(value=0.0, max=544.0), HTML(value='')))
```

```
[ ] 1 vif_values.head()
```

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

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

```
[ ] 1 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

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

```
[ ] 1 # removing top_20 features from the features that need to be dropped from VIF
2 inf_indices=list(set(inf_indices)-set(top_20_features))
```

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

```
[ ] 1 with open('/content/drive/My Drive/Colab Notebooks/22 Case Study 1/inf_indices.txt', 'r') as f:
2     check=json.loads(f.read())
3
```

```
[ ] 1 inf_indices=checc
[ ] 1 print(x_filtered.shape)
2 x_filtered = x_filtered.drop(inf_indices, axis=1)
3 print(x_filtered.shape)

⇒ (4194, 544)
(4194, 171)
```

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

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

[ ] 1 print(U.shape, s.shape, V.shape)

⇒ (4194, 4194) (171,) (171, 171)

[ ] 1 def omega_approx(beta):
2     """Return an approximate omega value for given beta. Equation (5) from Gavish 2014."""
3     return 0.56 * beta**3 - 0.95 * beta**2 + 1.82 * beta + 1.43
4
5

[ ] 1 beta = min(x_filtered.shape) / max(x_filtered.shape)
2 tau = np.median(s) * omega_approx(beta)
3

[ ] 1 print(tau)
2 print(np.log(tau))

⇒ 15.625431208108179
2.74889979256059

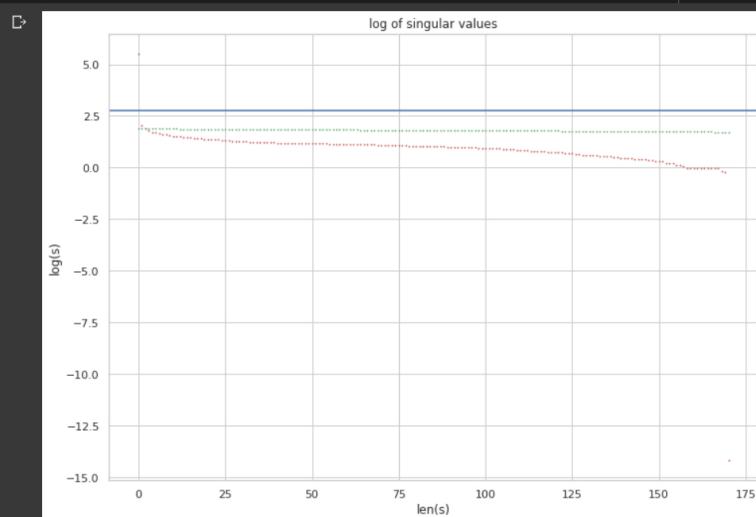
[ ] 1 # initializing normally distributed matrix of size equal to to size of x_filtered, mean =0, variance=1
2 noise = np.random.normal(0, 1, (4194, 171))
3 noise.shape

⇒ (4194, 171)
```

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

⇒ (4194, 4194) (171,) (171, 171)
```

```
[ ] 1 plt.plot(np.arange(s.shape[0]),np.log10(s), 'ro', markersize=0.7 )# https://python-graph-gallery.com/134-how-to-avoid-overplotting-with-python/
2 plt.plot(np.arange(s_n.shape[0]), np.log10(s_n), 'go', markersize=0.7)
3 plt.axhline(y=np.log(tau))
4 plt.grid()
5 plt.ylabel('log(s)')
6 plt.xlabel('len(s)')
7 plt.grid()
8 plt.title('log of singular values')
9 plt.show()
```



- From the graph and the values of tau retrieved 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 singular 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

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

```
[ ] 1 tsvd_train.shape
```

```
⇒ (4194, 2)
```

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

```
[ ] 1 pca_train.shape
```

```
⇒ (4194, 2)
```

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

```
[ ] 1 ica_train.shape
```

```
⇒ (4194, 2)
```

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

```
[ ] 1 print(x_filtered.shape#, x_dummies_2.shape)
```

```
⇒ (4194, 180)
```

▼ 15: Adding new features using the top important features

```
[ ] 1 # out of all the top_20_important features, after all the processing ['X315', 'X54', 'X1_aa', 'X58', 'X314', 'X2_s', 'X189', 'X118', 'X311', 'X29', 'X127', 'X132', 'X0_z']  
2 # the above features are dropped and X64, X218, X224, X273 these four features are present, so let's generate new features from these 4 features  
3
```

```
▶ 1 x_filtered['X64 + X218']=x_filtered['X64']+x_filtered['X218']  
2  
3 x_filtered['X218 + X224 + X273']=x_filtered['X218']+x_filtered['X224'] + x_filtered['X273']  
4  
5 x_filtered['X64 + X224 + X273']=x_filtered['X64']+x_filtered['X224'] + x_filtered['X273']  
6  
7 x_filtered['X314 + X315']=x_filtered['X314']+x_filtered['X315']  
8  
9 x_filtered['X314 + X315 + X29']=x_filtered['X314']+x_filtered['X315'] + x_filtered['X29']  
10  
11 x_filtered['X314 + X315 + X118']=x_filtered['X314']+x_filtered['X315'] + x_filtered['X118']  
12  
13 x_filtered['X127 + X189']=x_filtered['X127']+x_filtered['X189']  
14  
15 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

