Project - Mercedes-Benz Greener Manufacturing

Description - Reduce the time a Mercedes-Benz spends on the test bench.

Name- Arka Prava Panda

Programme - Data Scientist Masters Programme

We have imported the necessary modules/libraries in order to process the dataset

In [59]:
```
import numpy as np
import pandas as pd

data = pd.read_csv('Datasets/Project Dataset/train.csv')
data_test = pd.read_csv('Datasets/Project Dataset/test.csv')
```

i) We have read the train and test datasets and saved it as a pandas dataframe in data and data_test respectively

ii) We have saved the y column which is the dependent variable as target and the rest of data which are the independent variables as data_train

In [300…
```
data_train = data.drop('y',axis=1)
target = data['y']
```

Checking the first five rows of the training and test data

In [302…
```
data_train.head()
```

Out[302]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	k	v	at	a	d	u	j	o	0	...	0	0	1	0	0	0	0	0	0	0
1	6	k	t	av	e	d	y	l	o	0	...	1	0	0	0	0	0	0	0	0	0
2	7	az	w	n	c	d	x	j	x	0	...	0	0	0	0	0	0	1	0	0	0
3	9	az	t	n	f	d	x	l	e	0	...	0	0	0	0	0	0	0	0	0	0
4	13	az	v	n	f	d	h	d	n	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

In [47]:
```
data_test.head()
```

Out[47]:

	ID	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	1	az	v	n	f	d	t	a	w	0	...	0	0	0	1	0	0	0	0	0	0
1	2	t	b	ai	a	d	b	g	y	0	...	0	0	1	0	0	0	0	0	0	0
2	3	az	v	as	f	d	a	j	j	0	...	0	0	0	1	0	0	0	0	0	0
3	4	az	l	n	f	d	z	l	n	0	...	0	0	0	1	0	0	0	0	0	0
4	5	w	s	as	c	d	y	i	m	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 377 columns

Observing the statistical insights from the dataframe.describe() function

In [62]:
```
data_train.describe()
```

Out[62]:

	ID	X10	X11	X12	X13	X14	X15	X16	X17	X18	...	X3
count	4209.000000	4209.000000	4209.0	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	4209.000
mean	4205.960798	0.013305	0.0	0.075077	0.057971	0.428130	0.000475	0.002613	0.007603	0.007840	...	0.318
std	2437.608688	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872	0.088208	...	0.466
min	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000
25%	2095.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000
50%	4220.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000
75%	6314.000000	0.000000	0.0	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	...	1.000
max	8417.000000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000

8 rows × 369 columns

In [63]:

```
data_test.describe()
```

Out[63]:

	ID	X10	X11	X12	X13	X14	X15	X16	X17	X18	...	
count	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	4209.000000	...	420
mean	4211.039202	0.019007	0.000238	0.074364	0.061060	0.427893	0.000713	0.002613	0.008791	0.010216	...	
std	2423.078926	0.136565	0.015414	0.262394	0.239468	0.494832	0.026691	0.051061	0.093357	0.100570	...	
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	2115.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
50%	4202.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	
75%	6310.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	...	
max	8416.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	

8 rows × 369 columns

Since it has been clearly mentioned that the columns with variance of 0 to be dropped we will look for the columns with variance 0

To do this we can get help from the describe function but before that we need to check the columns that have categorical values

In [98]:
```
#As the describe doesnt capture the categorical variables
#we need to check the number of categorical variables in the dataset

set(data_train.columns) - set(data_train._get_numeric_data().columns)
```

Out[98]:
```
{'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'}
```

In [101…
```
#Hence we can see that the above are the categorical variables
#Those variables were ignored in the describe function

categorical_features = np.array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'])
categorical_features
```

Out[101]:
```
array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8'], dtype='<U2')
```

Using the T method we can transpose a dataframe

Here we have transposed the dataset in order to get the features as rows, thus we can easily eliminate the rows which have 0 variance as we all know that $\sqrt(variance) = stdev$

In [64]:
```
desc_train = data_train.describe().T
desc_test = data_test.describe().T
```

In [65]:
```
desc_train.head()
```

Out[65]:

	count	mean	std	min	25%	50%	75%	max
ID	4209.0	4205.960798	2437.608688	0.0	2095.0	4220.0	6314.0	8417.0
X10	4209.0	0.013305	0.114590	0.0	0.0	0.0	0.0	1.0
X11	4209.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
X12	4209.0	0.075077	0.263547	0.0	0.0	0.0	0.0	1.0
X13	4209.0	0.057971	0.233716	0.0	0.0	0.0	0.0	1.0

In [66]:
```
desc_test.head()
```

Out[66]:

	count	mean	std	min	25%	50%	75%	max
ID	4209.0	4211.039202	2423.078926	1.0	2115.0	4202.0	6310.0	8416.0
X10	4209.0	0.019007	0.136565	0.0	0.0	0.0	0.0	1.0
X11	4209.0	0.000238	0.015414	0.0	0.0	0.0	0.0	1.0
X12	4209.0	0.074364	0.262394	0.0	0.0	0.0	0.0	1.0
X13	4209.0	0.061060	0.239468	0.0	0.0	0.0	0.0	1.0

Now we will only take the column-names where the Standard deviation is not 0

We will perform the same operation for the training and the test dataset

In [68]:

```
desc_train[desc_train['std']!=0]
```

Out[68]:

	count	mean	std	min	25%	50%	75%	max
ID	4209.0	4205.960798	2437.608688	0.0	2095.0	4220.0	6314.0	8417.0
X10	4209.0	0.013305	0.114590	0.0	0.0	0.0	0.0	1.0
X12	4209.0	0.075077	0.263547	0.0	0.0	0.0	0.0	1.0
X13	4209.0	0.057971	0.233716	0.0	0.0	0.0	0.0	1.0
X14	4209.0	0.428130	0.494867	0.0	0.0	0.0	1.0	1.0
...	...	...	...	...	...	...	...	...
X380	4209.0	0.008078	0.089524	0.0	0.0	0.0	0.0	1.0
X382	4209.0	0.007603	0.086872	0.0	0.0	0.0	0.0	1.0
X383	4209.0	0.001663	0.040752	0.0	0.0	0.0	0.0	1.0
X384	4209.0	0.000475	0.021796	0.0	0.0	0.0	0.0	1.0
X385	4209.0	0.001426	0.037734	0.0	0.0	0.0	0.0	1.0

357 rows × 8 columns

In [69]:

```
desc_test[desc_test['std']!=0]
```

Out[69]:

	count	mean	std	min	25%	50%	75%	max
ID	4209.0	4211.039202	2423.078926	1.0	2115.0	4202.0	6310.0	8416.0
X10	4209.0	0.019007	0.136565	0.0	0.0	0.0	0.0	1.0
X11	4209.0	0.000238	0.015414	0.0	0.0	0.0	0.0	1.0
X12	4209.0	0.074364	0.262394	0.0	0.0	0.0	0.0	1.0
X13	4209.0	0.061060	0.239468	0.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...
X380	4209.0	0.008078	0.089524	0.0	0.0	0.0	0.0	1.0
X382	4209.0	0.008791	0.093357	0.0	0.0	0.0	0.0	1.0
X383	4209.0	0.000475	0.021796	0.0	0.0	0.0	0.0	1.0
X384	4209.0	0.000713	0.026691	0.0	0.0	0.0	0.0	1.0
X385	4209.0	0.001663	0.040752	0.0	0.0	0.0	0.0	1.0

364 rows × 8 columns

We will save the features whose variance is not zero in num_features

In [102…

```
num_features = desc_train[desc_train['std']!=0].index.values
num_features
```

Out[102]:
```
array(['ID', 'X10', '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', 'X45', 'X46',
       'X47', 'X48', 'X49', 'X50', 'X51', 'X52', 'X53', 'X54', 'X55',
       'X56', 'X57', 'X58', 'X59', 'X60', 'X61', 'X62', 'X63', 'X64',
       'X65', 'X66', 'X67', 'X68', 'X69', 'X70', 'X71', 'X73', 'X74',
       'X75', 'X76', 'X77', 'X78', 'X79', 'X80', 'X81', 'X82', 'X83',
       'X84', 'X85', 'X86', 'X87', 'X88', 'X89', 'X90', 'X91', 'X92',
       'X94', 'X95', 'X96', 'X97', 'X98', 'X99', 'X100', 'X101', 'X102',
       'X103', 'X104', 'X105', 'X106', 'X108', 'X109', 'X110', 'X111',
       'X112', 'X113', 'X114', 'X115', 'X116', 'X117', 'X118', 'X119',
       'X120', 'X122', 'X123', 'X124', 'X125', 'X126', 'X127', 'X128',
       'X129', 'X130', 'X131', 'X132', 'X133', 'X134', 'X135', 'X136',
       'X137', 'X138', 'X139', 'X140', 'X141', 'X142', 'X143', 'X144',
       'X145', 'X146', 'X147', 'X148', 'X150', 'X151', 'X152', 'X153',
       'X154', 'X155', 'X156', 'X157', 'X158', 'X159', 'X160', 'X161',
       'X162', 'X163', 'X164', 'X165', 'X166', 'X167', 'X168', 'X169',
       'X170', 'X171', 'X172', 'X173', 'X174', 'X175', 'X176', 'X177',
       'X178', 'X179', 'X180', 'X181', 'X182', 'X183', 'X184', 'X185',
       'X186', 'X187', 'X189', 'X190', 'X191', 'X192', 'X194', 'X195',
       'X196', 'X197', 'X198', 'X199', 'X200', 'X201', 'X202', 'X203',
       'X204', 'X205', 'X206', 'X207', 'X208', 'X209', 'X210', 'X211',
       'X212', 'X213', 'X214', 'X215', 'X216', 'X217', 'X218', 'X219',
       'X220', 'X221', 'X222', 'X223', 'X224', 'X225', 'X226', 'X227',
       'X228', 'X229', 'X230', 'X231', 'X232', 'X234', 'X236', 'X237',
       'X238', 'X239', 'X240', 'X241', 'X242', 'X243', 'X244', 'X245',
       'X246', 'X247', 'X248', 'X249', 'X250', 'X251', 'X252', 'X253',
       'X254', 'X255', 'X256', 'X257', 'X258', 'X259', 'X260', 'X261',
       'X262', 'X263', 'X264', 'X265', 'X266', 'X267', 'X269', 'X270',
       'X271', 'X272', 'X273', 'X274', 'X275', 'X276', 'X277', 'X278',
       'X279', 'X280', 'X281', 'X282', 'X283', 'X284', 'X285', 'X286',
       'X287', 'X288', 'X291', 'X292', 'X294', 'X295', 'X296', 'X298',
       'X299', 'X300', 'X301', 'X302', 'X304', 'X305', 'X306', 'X307',
       'X308', 'X309', 'X310', 'X311', 'X312', 'X313', 'X314', 'X315',
       'X316', 'X317', 'X318', 'X319', 'X320', 'X321', 'X322', 'X323',
       'X324', 'X325', 'X326', 'X327', 'X328', 'X329', 'X331', 'X332',
       'X333', 'X334', 'X335', 'X336', 'X337', 'X338', 'X339', 'X340',
       'X341', 'X342', 'X343', 'X344', 'X345', 'X346', 'X348', 'X349',
       'X350', 'X351', 'X352', 'X353', 'X354', 'X355', 'X356', 'X357',
       'X358', 'X359', 'X360', 'X361', 'X362', 'X363', 'X364', 'X365',
```

```
       'X366', 'X367', 'X368', 'X369', 'X370', 'X371', 'X372', 'X373',
       'X374', 'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382',
       'X383', 'X384', 'X385'], dtype=object)
```

We have concatenated the categorical feature names and numerical feature names in features

In [112…

```
features = np.append(categorical_features, num_features)
features
```

```
Out[112]: array(['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8', 'ID', 'X10', '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', 'X45', 'X46', 'X47', 'X48', 'X49',
                 'X50', 'X51', 'X52', 'X53', 'X54', 'X55', 'X56', 'X57', 'X58',
                 'X59', 'X60', 'X61', 'X62', 'X63', 'X64', 'X65', 'X66', 'X67',
                 'X68', 'X69', 'X70', 'X71', 'X73', 'X74', 'X75', 'X76', 'X77',
                 'X78', 'X79', 'X80', 'X81', 'X82', 'X83', 'X84', 'X85', 'X86',
                 'X87', 'X88', 'X89', 'X90', 'X91', 'X92', 'X94', 'X95', 'X96',
                 'X97', 'X98', 'X99', 'X100', 'X101', 'X102', 'X103', 'X104',
                 'X105', 'X106', 'X108', 'X109', 'X110', 'X111', 'X112', 'X113',
                 'X114', 'X115', 'X116', 'X117', 'X118', 'X119', 'X120', 'X122',
                 'X123', 'X124', 'X125', 'X126', 'X127', 'X128', 'X129', 'X130',
                 'X131', 'X132', 'X133', 'X134', 'X135', 'X136', 'X137', 'X138',
                 'X139', 'X140', 'X141', 'X142', 'X143', 'X144', 'X145', 'X146',
                 'X147', 'X148', 'X150', 'X151', 'X152', 'X153', 'X154', 'X155',
                 'X156', 'X157', 'X158', 'X159', 'X160', 'X161', 'X162', 'X163',
                 'X164', 'X165', 'X166', 'X167', 'X168', 'X169', 'X170', 'X171',
                 'X172', 'X173', 'X174', 'X175', 'X176', 'X177', 'X178', 'X179',
                 'X180', 'X181', 'X182', 'X183', 'X184', 'X185', 'X186', 'X187',
                 'X189', 'X190', 'X191', 'X192', 'X194', 'X195', 'X196', 'X197',
                 'X198', 'X199', 'X200', 'X201', 'X202', 'X203', 'X204', 'X205',
                 'X206', 'X207', 'X208', 'X209', 'X210', 'X211', 'X212', 'X213',
                 'X214', 'X215', 'X216', 'X217', 'X218', 'X219', 'X220', 'X221',
                 'X222', 'X223', 'X224', 'X225', 'X226', 'X227', 'X228', 'X229',
                 'X230', 'X231', 'X232', 'X234', 'X236', 'X237', 'X238', 'X239',
                 'X240', 'X241', 'X242', 'X243', 'X244', 'X245', 'X246', 'X247',
                 'X248', 'X249', 'X250', 'X251', 'X252', 'X253', 'X254', 'X255',
                 'X256', 'X257', 'X258', 'X259', 'X260', 'X261', 'X262', 'X263',
                 'X264', 'X265', 'X266', 'X267', 'X269', 'X270', 'X271', 'X272',
                 'X273', 'X274', 'X275', 'X276', 'X277', 'X278', 'X279', 'X280',
                 'X281', 'X282', 'X283', 'X284', 'X285', 'X286', 'X287', 'X288',
                 'X291', 'X292', 'X294', 'X295', 'X296', 'X298', 'X299', 'X300',
                 'X301', 'X302', 'X304', 'X305', 'X306', 'X307', 'X308', 'X309',
                 'X310', 'X311', 'X312', 'X313', 'X314', 'X315', 'X316', 'X317',
                 'X318', 'X319', 'X320', 'X321', 'X322', 'X323', 'X324', 'X325',
                 'X326', 'X327', 'X328', 'X329', 'X331', 'X332', 'X333', 'X334',
                 'X335', 'X336', 'X337', 'X338', 'X339', 'X340', 'X341', 'X342',
                 'X343', 'X344', 'X345', 'X346', 'X348', 'X349', 'X350', 'X351',
                 'X352', 'X353', 'X354', 'X355', 'X356', 'X357', 'X358', 'X359',
```

```
       'X360', 'X361', 'X362', 'X363', 'X364', 'X365', 'X366', 'X367',
       'X368', 'X369', 'X370', 'X371', 'X372', 'X373', 'X374', 'X375',
       'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'], dtype=object)
```

Filtering both the training and test dataset which contains only the above feature names

In [113…
```
#Filtering out the dataset with certain features

data_train_1 = data_train[features]
data_test_1 = data_test[features]
```

In [114…
```
#checking the dataset if everything is in order
data_train_1.head()
```

Out[114]:

	X0	X1	X2	X3	X4	X5	X6	X8	ID	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	k	v	at	a	d	u	j	o	0	0	...	0	0	1	0	0	0	0	0	0	0
1	k	t	av	e	d	y	l	o	6	0	...	1	0	0	0	0	0	0	0	0	0
2	az	w	n	c	d	x	j	x	7	0	...	0	0	0	0	0	0	1	0	0	0
3	az	t	n	f	d	x	l	e	9	0	...	0	0	0	0	0	0	0	0	0	0
4	az	v	n	f	d	h	d	n	13	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

In [115…
```
data_test_1.head()
```

Out[115]:

	X0	X1	X2	X3	X4	X5	X6	X8	ID	X10	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	az	v	n	f	d	t	a	w	1	0	...	0	0	0	1	0	0	0	0	0	0
1	t	b	ai	a	d	b	g	y	2	0	...	0	0	1	0	0	0	0	0	0	0
2	az	v	as	f	d	a	j	j	3	0	...	0	0	0	1	0	0	0	0	0	0
3	az	l	n	f	d	z	l	n	4	0	...	0	0	0	1	0	0	0	0	0	0
4	w	s	as	c	d	y	i	m	5	0	...	1	0	0	0	0	0	0	0	0	0

5 rows × 365 columns

The dataset(training) has no null/missing values

In [116…
```
#Checking for null values
data_train_1.isna().sum()
```

Out[116]:
```
X0      0
X1      0
X2      0
X3      0
X4      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 365, dtype: int64
```

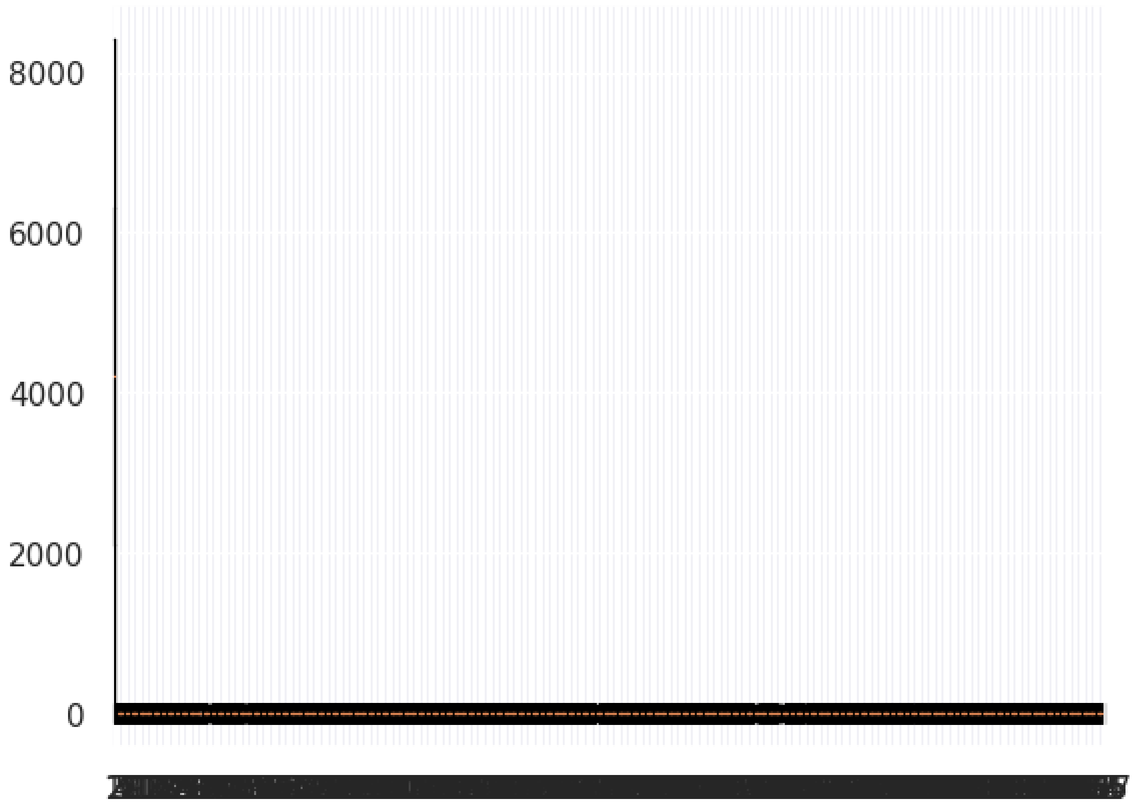
In [117…
```
data_test_1.isna().sum()
```

Out[117]:
```
X0      0
X1      0
X2      0
X3      0
X4      0
       ..
X380    0
X382    0
X383    0
X384    0
X385    0
Length: 365, dtype: int64
```

Plotting a boxplot to check if the features have any significant outliers

In [125…
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set()
```

```
plt.boxplot(data_train_1[num_features])
plt.show()
```



Performing some EDA to check the categorical variables

In [137…
```
#We will further check the categorical values

data_train_1[categorical_features].iloc[:,0].value_counts()
#From the first column we can see that there are several unique values for the categorical columns
```

Out[137]:
```
z     360
ak    349
y     324
ay    313
t     306
x     300
o     269
f     227
n     195
w     182
j     181
az    175
aj    151
s     106
ap    103
h      75
d      73
al     67
v      36
af     35
m      34
ai     34
e      32
ba     27
at     25
a      21
ax     19
aq     18
am     18
i      18
u      17
aw     16
l      16
ad     14
au     11
k      11
b      11
r      10
as     10
bc      6
ao      4
```

```
c       3
aa      2
q       2
ac      1
g       1
ab      1
Name: X0, dtype: int64
```

Since the instruction clearly says to apply label encoder, we will do so

In [303…

```
#Now we will label encode the categorical columns
#Since in label encoder we have to pass a 1-d array, it will take time to encode the whole columns
#But instead we can use the Ordinal Encoder which works the same
#yet processess all the categorical features to return an encoded array

from sklearn.preprocessing import LabelEncoder, OrdinalEncoder

le = LabelEncoder()

encode = OrdinalEncoder()

data_train_1_encoded = encode.fit_transform(data_train_1)
```

In [188…

```
data_train_1_encoded
```

Out[188]:

```
array([[32., 23., 17., ...,  0.,  0.,  0.],
       [32., 21., 19., ...,  0.,  0.,  0.],
       [20., 24., 34., ...,  0.,  0.,  0.],
       ...,
       [ 8., 23., 38., ...,  0.,  0.,  0.],
       [ 9., 19., 25., ...,  0.,  0.,  0.],
       [46., 19.,  3., ...,  0.,  0.,  0.]])
```

Dimensionality Reduction

In [271…

```
#We can use the principal component analysis to sort out the most important features
#We have selected 30 principal components in order to ensure the model doesnt overfit the training dataset


from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=30)
data_train_1_encoded_reduced  = pca.fit_transform(data_train_1_encoded,target)
pca.explained_variance_ratio_
```

Out[271]:
```
array([9.99659608e-01, 1.38083753e-04, 7.69770342e-05, 4.40107808e-05,
       3.31849481e-05, 2.66009200e-05, 5.73061934e-06, 2.67845282e-06,
       1.56203752e-06, 1.05802559e-06, 8.69673184e-07, 8.51175070e-07,
       7.32194816e-07, 6.01103751e-07, 5.33242231e-07, 4.63034402e-07,
       3.72740073e-07, 3.44879436e-07, 3.21997035e-07, 2.83068830e-07,
       2.55585324e-07, 2.31969262e-07, 2.24009901e-07, 2.10430764e-07,
       1.84605062e-07, 1.73950883e-07, 1.55694846e-07, 1.49111645e-07,
       1.35987698e-07, 1.32567075e-07])
```

Modelling

In [253…
```
#Creating the training and validation data out of the train dataset
#Although we have a test dataset separately, Still we will chalk out a validation dataset...
#...out of training dataset in order to check for overfitting and underfitting

from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(data_train_1_encoded_reduced,target)

X_train.shape, X_val.shape
```

Out[253]:
```
((3156, 30), (1053, 30))
```

Project instruction asks to use XGBoost, Hence we will do so

In [304…
```
#We have imported the XGBRFRegressor
#We imported the XGB Random Forest Regressor as we have to predict values and not probability

from xgboost import XGBRFRegressor

xgb = XGBRFRegressor(n_jobs=-1,max_depth=10)

xgb.fit(X_train,y_train)
```

Out[304]:

XGBRFRegressor

```
XGBRFRegressor(base_score=0.5, booster='gbtree', callbacks=None,
               colsample_bylevel=1, colsample_bytree=1,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
               grow_policy='depthwise', importance_type=None,
               interaction_constraints='', max_bin=256, max_cat_threshold=64,
               max_cat_to_onehot=4, max_delta_step=0, max_depth=10,
               max_leaves=0, min_child_weight=1, missing=nan,
               monotone_constraints='()', n_estimators=100, n_jobs=-1,
```

In [289…

```python
#The xgb.score shows the mean accuracy of self.predict(X) wrt. y

xgb.score(X_train,y_train) #training xgb.score
```

Out[289]: 0.7867312762329959

In [290…

```python
#Predicting using the validation dataset

y_pred = xgb.predict(X_val)  #test xgb.score
```

In [291…

```python
#Calculating the score using the validation dataset
#Although the score in the validation set is lower than that of the score of the training set...
#...still the training set is not overfitting the model

xgb.score(X_val,y_val)
```

Out[291]: 0.5055256466285112

The RMSE shows that on average the predicted values differ from the original values by 8.716

In [292…

```python
#Now we will evaluate the model

from sklearn.metrics import mean_squared_error,r2_score

rmse = np.sqrt(mean_squared_error(list(y_val),y_pred))
```

```
rmse
```

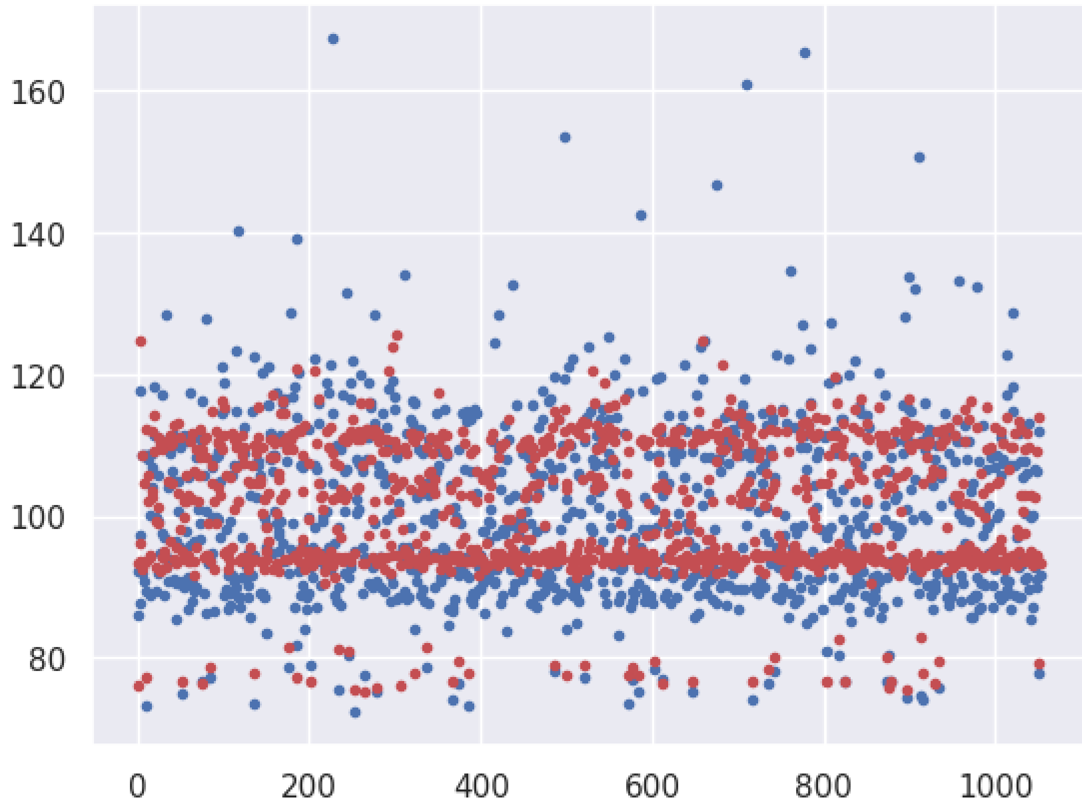
Out[292]: 8.71620625839888

We will plot the predicted values over the real values of y

In [293…
```
#plotting the predicted vs real gives a visual idea or the differences between the real and predicted


plt.plot(list(y_val),'b.')
plt.plot(y_pred,'r.')
```

Out[293]: [<matplotlib.lines.Line2D at 0x7f0bd30cc890>]



In [305…
```
#As we can see that our prediction was able to capture the main trend of the data
```

Using the test dataset

In [295…
```
#We can use the pipeline to process the data

from sklearn.pipeline import Pipeline

pipeline = Pipeline([('encode',OrdinalEncoder()),('pca', PCA(n_components=30))])

data_test_1_encoded_reduced = pipeline.fit_transform(data_test_1)
```

In [297…
```
y_pred = xgb.predict(data_test_1_encoded_reduced)
```

In [298…
```
y_pred
```

Out[298]:
```
array([ 80.51625 , 101.39458 ,  79.300545, ..., 109.5979  , 110.05873 ,
        98.30962 ], dtype=float32)
```

Conclusion

In [306…
```
#We have performed/followed all the instructions prescribed in the project description

#which are--
        #If for any column(s), the variance is equal to zero, then you need to remove those variable(s).
        #Check for null and unique values for test and train sets.
        #Apply label encoder.
        #Perform dimensionality reduction.
        #Predict your test_df values using XGBoost.
```

In [ ]: