Business Problem

To ensure the safety and reliability of each and every unique car configuration before they hit the road, Daimler's engineers have developed a robust testing system. But, optimizing the speed of their testing system for so many possible feature combinations is complex and time-consuming. Hence, Daimler has challenged to reduce the time that cars spend on the test bench. The Objective of the Case Study is to optimize the testing process in a greener way i.e. reducing the testing time with lower carbon dioxide emissions without reducing Daimler's standards on safety and efficiency.

ML Formulation

We can pose this problem as a regression problem to predict the testing time by selecting some important features from the dataset to tackle the curse of dimensionality.

In order to know how our ML model is performing better, We will develop a baseline or random model and we will compare our models with the baseline model, to get a knowledge where our model stands.

Performance Metric

The metric we will use to evaluate our models is - R^2

Why R^2 as metric?

What is R^2

It is the amount of the variation in the output dependent attribute which is predictable from the input independent variable. It is used to check how well-observed results are reproduced by the model, depending on the ratio of total deviation of results described by the model.

Interpretation -

Assume R2 = 0.68 It can be referred that 68% of the changeability of the dependent output attribute can be explained by the model while the remaining 32 % of the variability is still unaccounted for.

R-squared = 1 - (SSres / SStot)

SSres is the sum of squares of the residual errors.

SStot is the total sum of the errors.

which means we scale our simple MSE based on the difference of actual values from their mean.

R^2 is a convenient rescaling of MSE that is unit invariant.

It is also very interpretable as -

The best possible score is 1 which is obtained when the predicted values are the same as the actual values.

R^2 with value 0 means the model is same as simple mean model.

Negative value of R^2 mean that the model is worse than simple mean model.

Why?

MSE or MAE penalizes the large prediction errors hence the sum of errors can become very large and interpreting it won't be trivial.

Whereas, R^2 is a scale-free score i.e. irrespective of the values being small or large, the value of R square will be less than one or in worst cases just greater than 1.

EDA

In [1]:

```
'''importing dependencies'''
import pandas as pd
import warnings
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os

from xgboost import XGBRegressor
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
sns.set()
```

Reading the data

In [2]:

```
data = pd.read_csv('downloads/trainwa.csv')
data.head()
```

Out[2]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	1	0	0	0	0	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	1	0	0	0	0	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	1	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	0	0	0	0	0	0	0	0	0
4	13	78.02	az	v	n	f	d	h	d	n	...	0	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

Knowing the data

Data Shape

In [3]:

```
print("Train Data Shape : = ", data.shape)
```

```
Train Data Shape : =  (4209, 378)
```

Detail View of Train Data

In [4]:

```
data.describe()
```

Out[4]:

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17
count	4209.000000	4209.000000	4209.000000	4209.0	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
std	2437.608688	12.679381	0.114590	0.0	0.263547	0.233716	0.494867	0.021796	0.051061	0.086872
min	0.000000	72.110000	0.000000	0.0	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

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17
50%	4220.000000	99.150000	0.000000	0.0	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
max	8417.000000	265.320000	1.000000	0.0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 370 columns

Checking for null values in the Data

In [5]:

```
print("Number of missing values in Train data: ",data.isnull().sum().sum())
```

```
Number of missing values in Train data:  0
```

Checking for duplicate values in the Data

In [6]:

```
print(len(data[data.duplicated()]))
```

```
0
```

Analyzing the Prediction Column 'y'

In [7]:

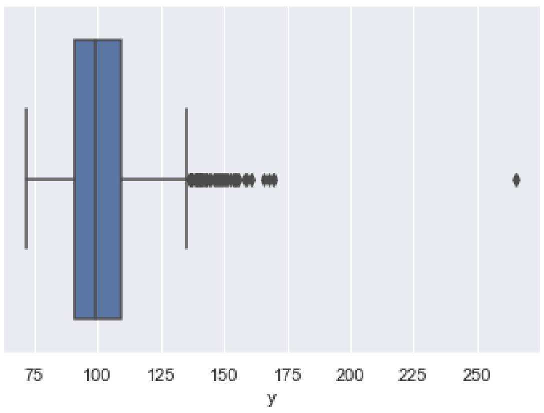
```
y = data['y']
y.describe()
```

Out[7]:

```
count    4209.000000
mean      100.669318
std        12.679381
min        72.110000
25%        90.820000
50%        99.150000
75%       109.010000
max       265.320000
Name: y, dtype: float64
```

In [8]:

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



Observations

1. The dataset does not have null values.
2. The dataset does not have duplicate values.
3. The Prediction column y has some outlier points

We will first remove these outlier points then we will know the features

Knowing the percentiles to decide the threshold to remove the outliers

In [9]:

```
print("Listing all percentiles for training time: ")
print("100: ", np.percentile(data.y,100))
print("99.9: ", np.percentile(data.y,99.9))
print("99.8: ", np.percentile(data.y,99.8))
print("99.7: ", np.percentile(data.y,99.7))
print("99.6: ", np.percentile(data.y,99.6))
print("99.5: ", np.percentile(data.y,99.5))
print("99 : ", np.percentile(data.y,99))
```

```
Listing all percentiles for training time:
100:  265.32
99.9:  160.38328000000087
99.8:  154.68695999999994
99.7:  151.4276800000003
99.6:  149.0374399999998
99.5:  146.23040000000006
99 :  137.4304
```

Observations -

We can see from 99.7 percentile onwards the y value is increasing drastically. We decide the threshold as 99.7

Removing Noise

In [10]:

```
threshold = np.percentile(data.y,99.7)
outlliers = data[data['y']>=threshold]
data.drop(data[data['y']>=threshold].index, inplace = True)
```

Data Shape after removing noise

In [11]:

```
data.shape
```

Out[11]:

```
(4196, 378)
```

In [12]:

```
outlliers.head()
```

Out[12]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
342	681	169.91	aa	l	ak	f	d	i	c	d	...	0	0	0	0	0	0	0	0	0	0
429	836	154.87	ak	l	ae	f	d	d	g	w	...	0	0	0	0	0	0	0	0	0	0
883	1770	265.32	y	r	ai	f	d	ag	l	t	...	0	0	0	0	0	0	0	0	0	0
889	1784	158.53	aj	l	as	f	d	ag	k	e	...	0	0	0	0	0	0	0	0	0	0

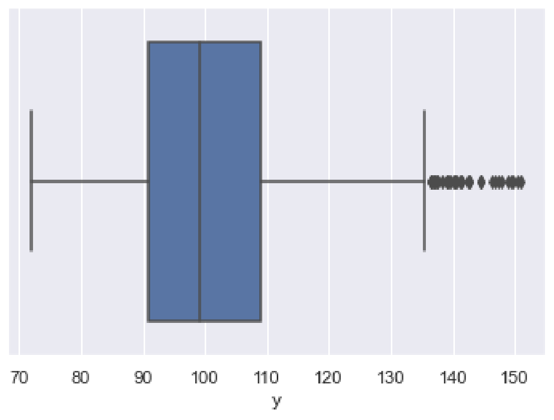
	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
1060	211[illegible]	154.43	w	v	f	c	d	ag	d	q	...	1	0	0	0	0	0	0	0	0	0

5 rows × 378 columns

Box-Plot of y after removing noise

In [13]:

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



Observations

1. The dataset looks more cleaner now.

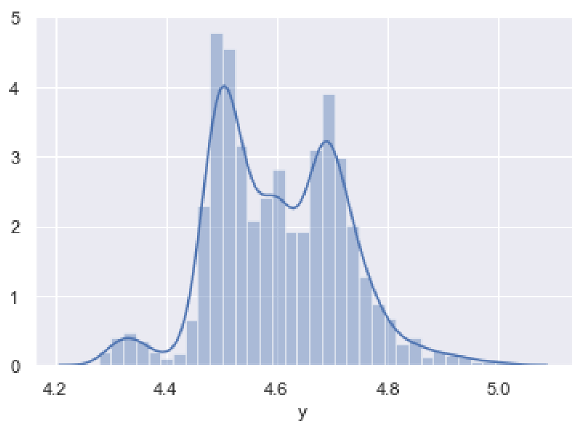
Log Transformation Distribution of target

In [59]:

```
sns.distplot(np.log(data['y']))
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5ef2cc310>
```



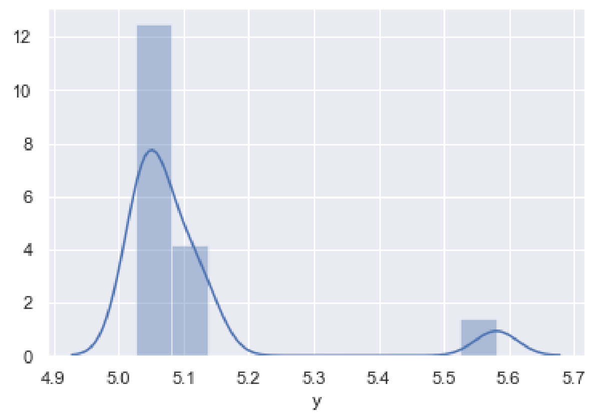
Log Transformation Distribution of target variable in outlier data

In [61]:

```
sns.distplot(np.log(outlliers['y']))
```

Out[61]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5ef3c11c0>
```



Observations

1. Most of the y values lie between 80 and 140.
2. Only a bunch of values are greater than 150.
3. Few values are less than 80.

Selecting important features for EDA

Building a simple XGBoost Model to get Feature Importances

In [16]:

```
y = data['y']
x = data.drop(columns = ['ID','y'], axis = 1)
#x = x.drop('id')
x.shape, y.shape
cols = x.columns
```

In [17]:

```
x_cat = data.loc[:,'X0':'X8']
x_num = data.loc[:,'X10':]
```

In [18]:

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
for i in x_cat.columns:
    x_cat[i] = enc.fit_transform(x_cat[i])
```

In [19]:

```
x = pd.DataFrame(np.hstack((x_cat,x_num)), columns = cols)
```

In [20]:

```
x.head()
```

Out[20]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	32	23	17	0	3	24	9	14	0	0	...	0	0	1	0	0	0	0	0	0	0
1	32	21	19	4	3	28	11	14	0	0	...	1	0	0	0	0	0	0	0	0	0
2	20	24	34	2	3	27	9	23	0	0	...	0	0	0	0	0	0	1	0	0	0
3	20	21	34	5	3	27	11	4	0	0	...	0	0	0	0	0	0	0	0	0	0

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
3	20	21	34	5	3	27	11	4	0	0	...	0	0	0	0	0	0	0	0	0	0
4	20	23	34	5	3	12	3	13	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 376 columns

In [21]:

```
model = XGBRegressor(n_estimators=100, learning_rate = 0.1,n_jobs = -1)
model.fit(x,y)
```

Out[21]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [22]:

```
imp = pd.DataFrame()
imp['columns'] = x.columns
imp['importances'] = model.feature_importances_[0]
result = imp.sort_values(by = 'importances')[:10]
```

Top 10 important features with importances

In [23]:

```
result
```

Out[23]:

	columns	importances
0	X0	0.000737
255	X263	0.000737
254	X262	0.000737
253	X261	0.000737
252	X260	0.000737
251	X259	0.000737
250	X258	0.000737
249	X257	0.000737
248	X256	0.000737
247	X255	0.000737

In [24]:

```
print("The Top 10 important features are : ", list(result['columns']))
```

```
The Top 10 important features are :  ['X0', 'X263', 'X262', 'X261', 'X260', 'X259', 'X258',
'X257', 'X256', 'X255']
```

EDA for important Features

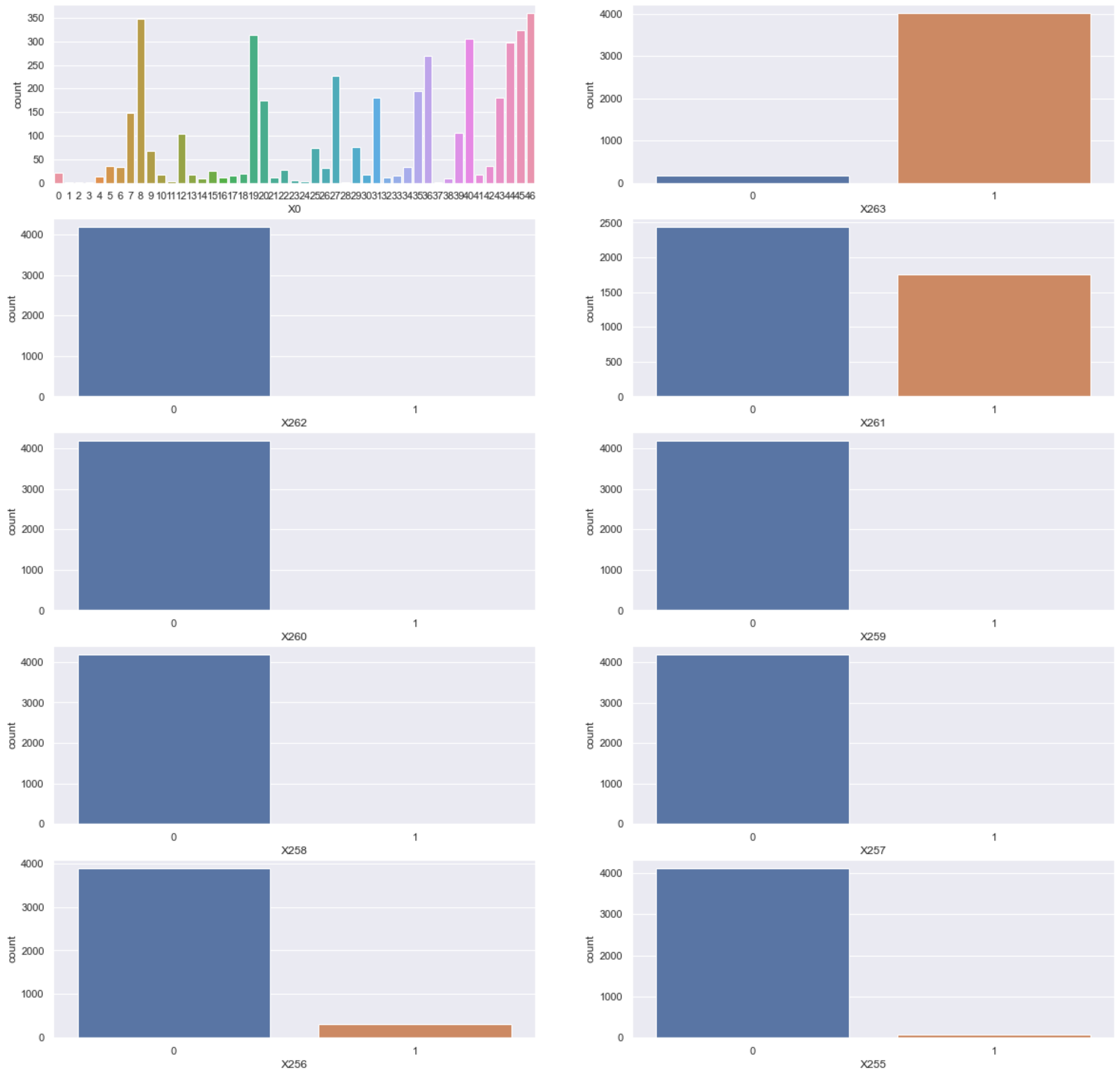
Knowing important features

In [25]:

```
imp_features = list(result['columns'])
imp_data = pd.DataFrame()
for i in imp_features:
    imp_data[i] = x[i]
```

In [26]:

```
#plotting categorical columns
fig, ax = plt.subplots(5, 2, figsize=(20, 20))
for variable, subplot in zip(imp_data.columns, ax.flatten()):
    sns.countplot(imp_data[variable], ax=subplot)
```



Observations -

1. The Categorical Feature X0 has well distributed Categories.
2. For the binary features X261 and X263 have significant values of 1 while the rest have mostly 0 values, which means tey are sparse.

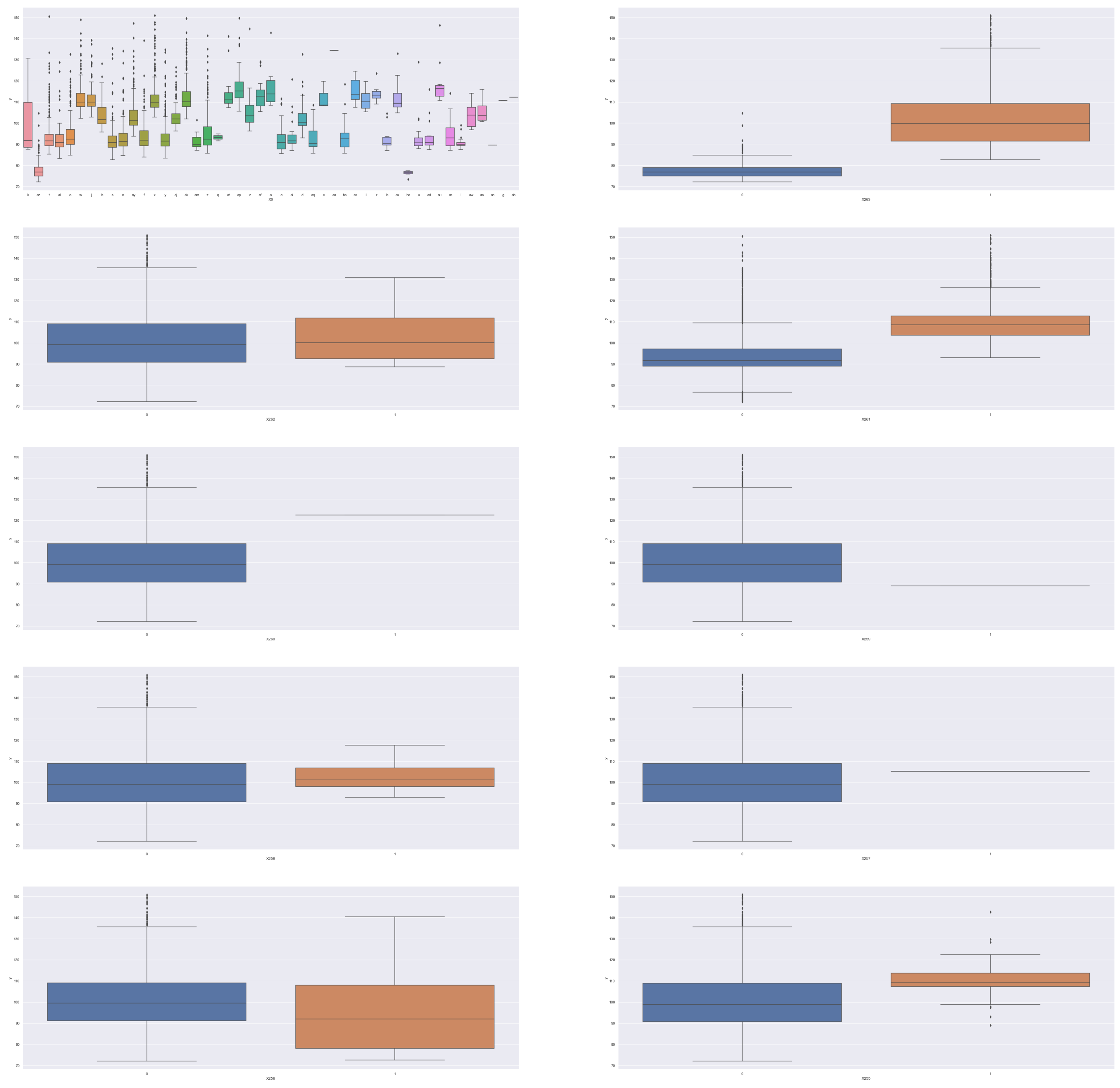
Univariate Analysis for Important features

Box - Plot

Box - Plot

In [27]:

```
fig, ax = plt.subplots(5, 2, figsize=(60, 60))
for variable, subplot in zip(imp_data.columns, ax.flatten()):
    sns.boxplot(x = data[variable],y = data['y'], ax=subplot)
```



Observations -

1. For binary features X263, X261, X255, X260, X259 the testing time is distinguishable according to the 0 and 1 value.

1. For X0 the percentiles for all the categories is distinguishable so we can interpret the testing time acoording to category.

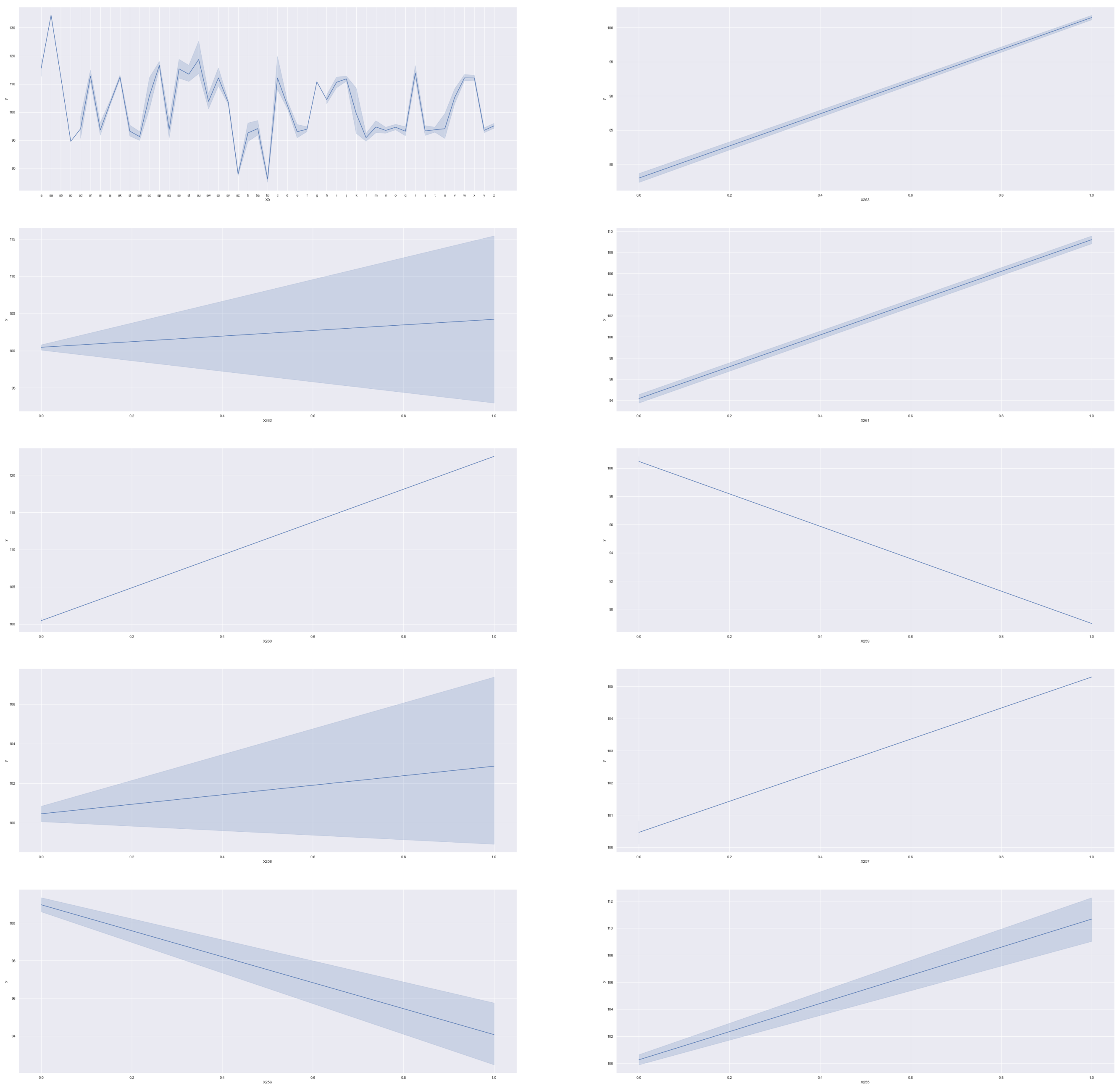
1. For X263, 0 value indicates testing time less than 85 and 1 value indicates testing time between 80 and 130.

1. For X261, 0 value indicates testing time between 75 and 110 and 1 value indicates testing time between 100 and 130.

Line Plot

In [28]:

```
fig, ax = plt.subplots(5, 2, figsize=(60, 60))
for variable, subplot in zip(imp_data.columns, ax.flatten()):
    sns.lineplot(x = data[variable],y = data['y'], ax=subplot)
```



Observations -

1. Line Plot is very interpretable for X0 which has many categories, while for binary features this does not give better information than Box-Plot.

1. For X0 we can conclude that, category 'aa' results in testing time greater than 130 while categories 'az' and 'bc' result in less than 80. The rest lie between 80 and 130.

In [ ]:

```

```

In [ ]:

```

```

Bi - Variate Analysis of Important Features

Co - relation of features

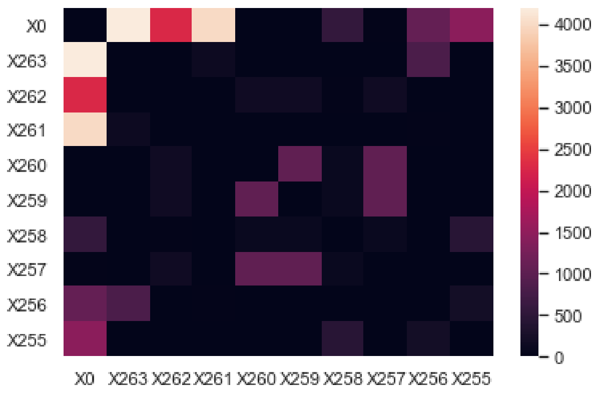
Chi Squared Test

In [30]:

```
import scipy.stats as stats
rows = imp_data.columns
col =  imp_data.columns
chi2_matrix = pd.DataFrame(columns = col, index = rows)
p_matrix = pd.DataFrame(columns = col, index = rows)
lesser_correlated_cols = []
for i in imp_features:
    for j in imp_features:
        if i != j:
            table = pd.crosstab(imp_data[i],imp_data[j])
            #Observed value
            obs_val = table.values
            chi2,p,dof,exp = stats.chi2_contingency(table)
            chi2_matrix[i][j] = chi2
            p_matrix[i][j] = p
            if p>=0.05:
                if (j,i) not in lesser_correlated_cols:
                    lesser_correlated_cols.append((i,j))

chi2_matrix = chi2_matrix.fillna(0)
print("The less realated column pairs are : ")
print(lesser_correlated_cols)
print("The heatmap for chi square values : ")
print(sns.heatmap(chi2_matrix))
```

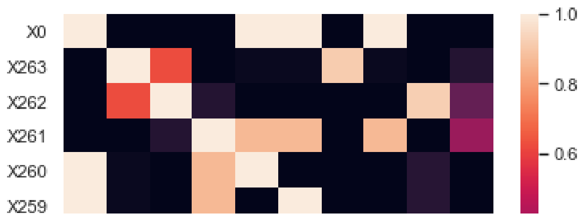
```
The less realated column pairs are :
[('X0', 'X260'), ('X0', 'X259'), ('X0', 'X257'), ('X263', 'X262'), ('X263', 'X258'), ('X263', 'X25
5'), ('X262', 'X261'), ('X262', 'X256'), ('X262', 'X255'), ('X261', 'X260'), ('X261', 'X259'), ('X
261', 'X257'), ('X261', 'X255'), ('X260', 'X256'), ('X259', 'X256'), ('X258', 'X256'), ('X257', 'X
256')]
The heatmap for chi square values :
AxesSubplot(0.125,0.125;0.62x0.755)
```

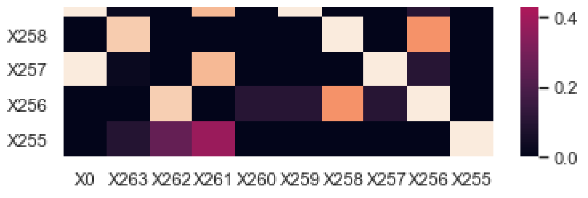


In [31]:

```
p_matrix = p_matrix.fillna(1)
print("The P-value matrix is :")
print(sns.heatmap(p_matrix))
```

```
The P-value matrix is :
AxesSubplot(0.125,0.125;0.62x0.755)
```





Observations -

1. The pair of less related features are :

   ('X0', 'X260'), ('X0', 'X259'), ('X0', 'X257'), ('X263', 'X262'), ('X263', 'X258'), ('X263', 'X255'), ('X262', 'X261'), ('X262', 'X256'), ('X262', 'X255'), ('X261', 'X260'), ('X261', 'X259'), ('X261', 'X257'), ('X261', 'X255'), ('X260', 'X256'), ('X259', 'X256'), ('X258', 'X256'), ('X257', 'X256')

1. The above pairs are decided on the basis of p-value, the null hypothesis that the features are not related is accepted.

Now lets see how these lesser corelated column pairs impact the target together

Feature Pairs with XO

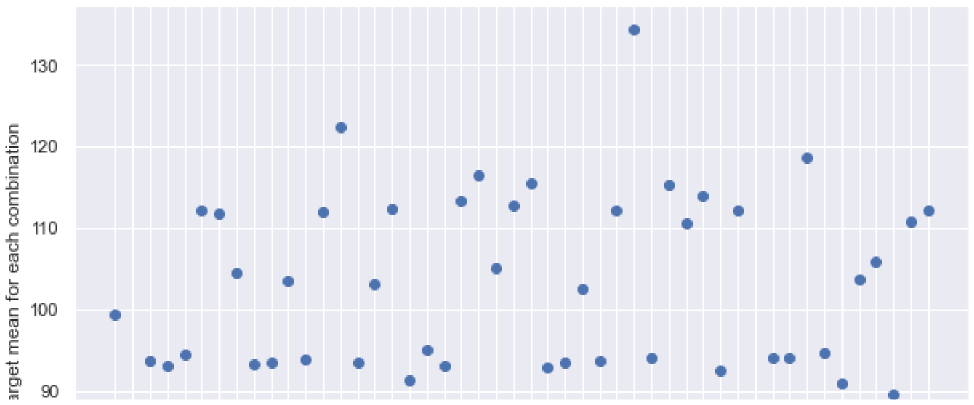
In [33]:

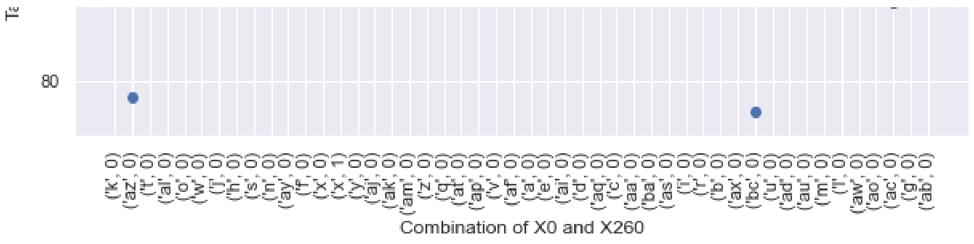
```
'''function to calculate mean of testing time for each pair of categories in the less co related f
eatures '''
def mean_with_category(i):
    col1 = i[0]
    col2 = i[1]
    means = {}
    for j in data[col1].unique():
        for k in data[col2].unique():
            temp = data[data[col1] == j]
            temp = temp[temp[col2] == k]
            if not temp.empty:
                target_mean = temp['y'].mean()
                means[(j,k)] = target_mean
    return means
```

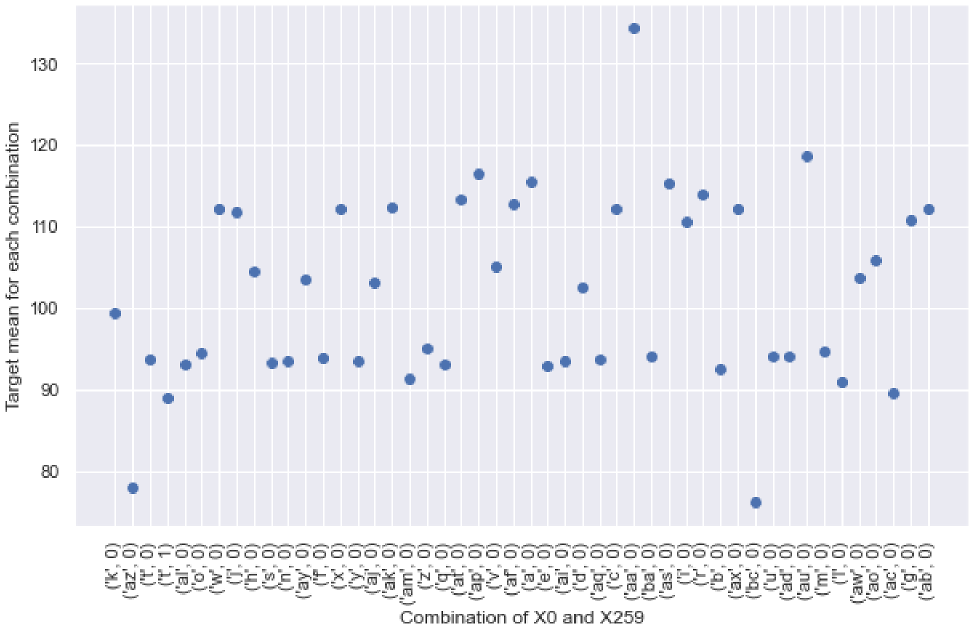
In [34]:

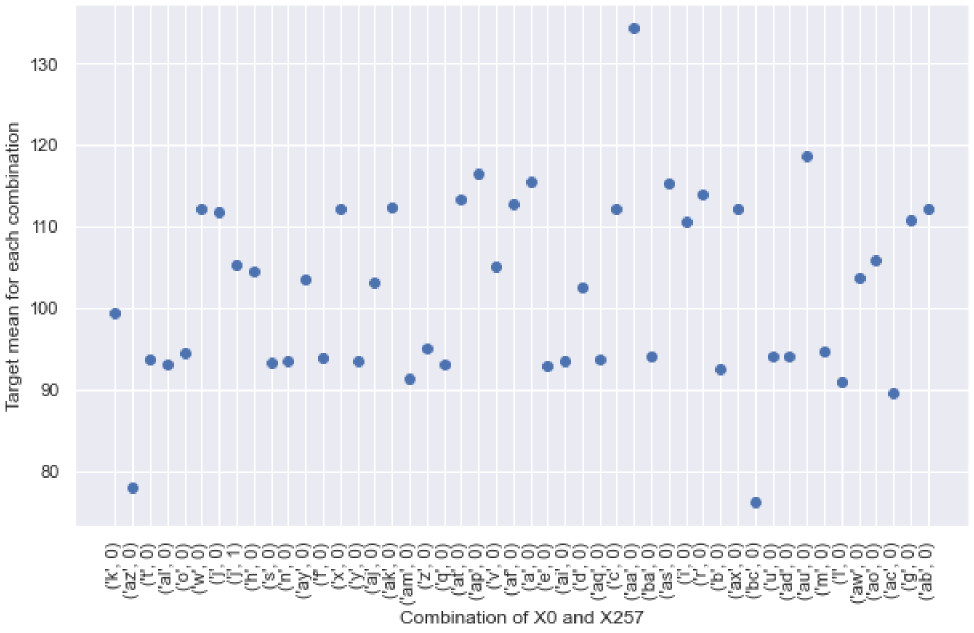
```
for i in lesser_correlated_cols[:3]:
    ans = mean_with_category(i)
    x_plot = ans.keys()
    y_plot = ans.values()

    plt.figure(figsize=(10, 6))
    plt.scatter([str(a) for a in x_plot], y_plot)
    plt.xticks(rotation='vertical')
    plt.xlabel("Combination of "+i[0]+ " and "+i[1])
    plt.ylabel("Target mean for each combination")
```









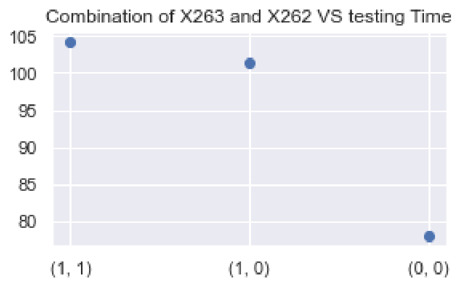
Observations -

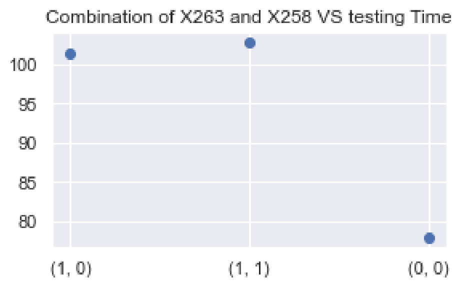
1. For the combination X0 with X260, X259, X257, the category 'aa' and value 0 has highest testing time mean and 'bc' and 0 has lowest.

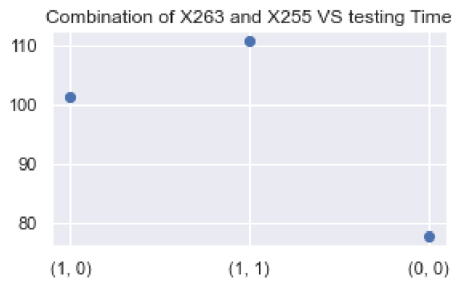
Binary-Binary features

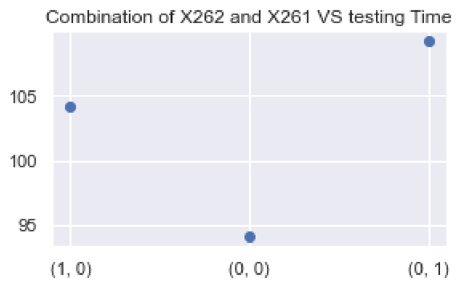
In [35]:

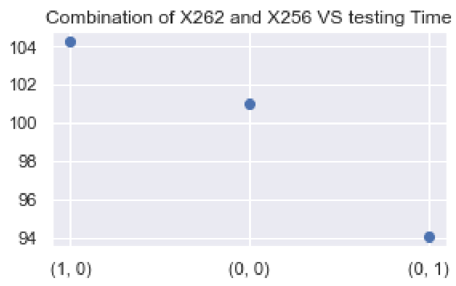
```
fig, ax = plt.subplots(14, 1, figsize=(5, 40))
for i ,subplot in zip(lesser_correlated_cols[3:], ax.flatten()):
    ans = mean_with_category(i)
    ans = mean_with_category(i)
    x_plot = ans.keys()
    y_plot = ans.values()
    subplot.scatter([str(a) for a in x_plot], y_plot)
    subplot.set_title("Combination of "+i[0]+ " and "+i[1]+" VS testing Time")
    fig.tight_layout(pad=3.0)
```

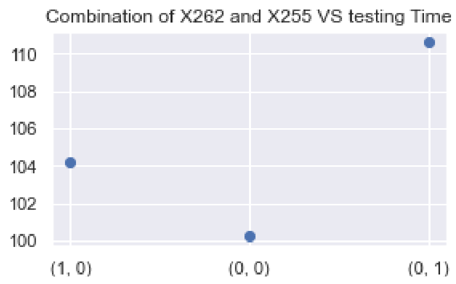




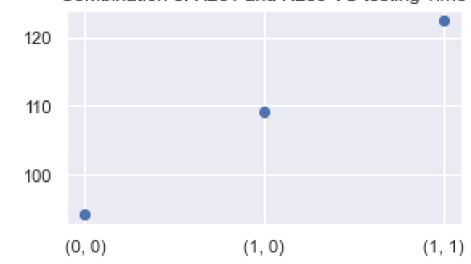




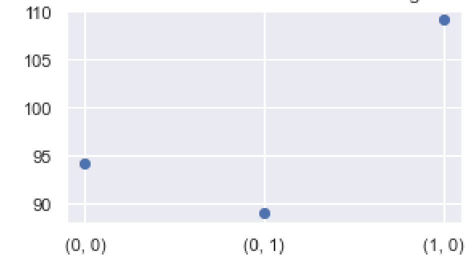




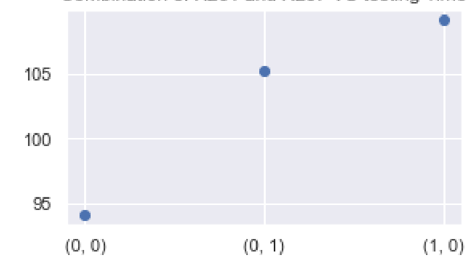
Combination of X261 and X260 VS testing Time



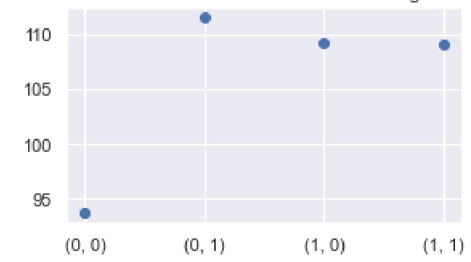
Combination of X261 and X259 VS testing Time



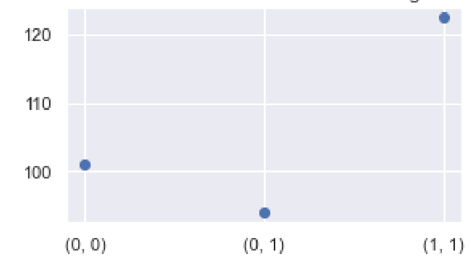
Combination of X261 and X257 VS testing Time



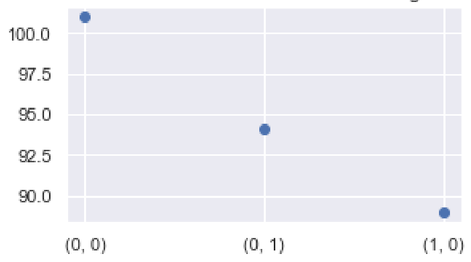
Combination of X261 and X255 VS testing Time



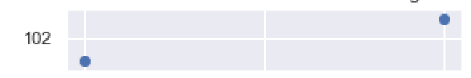
Combination of X260 and X256 VS testing Time

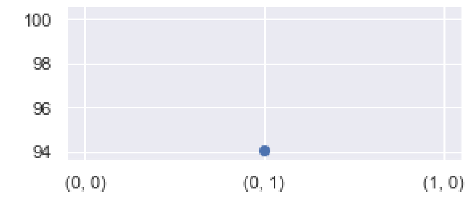


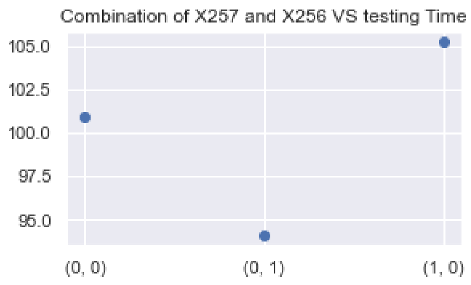
Combination of X259 and X256 VS testing Time



Combination of X258 and X256 VS testing Time







Observations -

1. For X263 and X262,

   if both have 0 values the average testing time is less than 80. if X263 has 1 and X262 has 0, the average testing time is more than 100. if both have value 1, the average testing time is more than 105.

1. For X263 and X258,

   if both have 0 values the average testing time is less than 80. if X263 has 1 and X258 has 0, the average testing time is close to 100. if both have value 1, the average testing time is more than 100.

1. For X263 and X255,

   if both have 0 values the average testing time is less than 80. if X263 has 1 and X255 has 0, the average testing time is close to 100. if both have value 1, the average testing time is more than 110.

1. For X262 and X261,

   if both have 0 values the average testing time is less than 95. if X262 has 1 and X261 has 0, the average testing time is close to 105. if X262 has 0 and X261 has 1, the average testing time is more than 105.

1. For X262 and X256,

   if both have 0 values the average testing time is close to 101. if X262 has 1 and X256 has 0, the average testing time is close to 104. if X262 has 0 and X261 has 1, the average testing time is more than 94.

1. For X262 and X255,

   if both have 0 values the average testing time is close to 100. if X262 has 1 and X255 has 0, the average testing time is close to 104. if X262 has 0 and X255 has 1, the average testing time is more than 110.

1. For X261 and X260,

   if both have 0 values the average testing time is less than 100. if X261 has 1 and X260 has 0, the average testing time is close to 110. if both have value 1, the average testing time is more than 120.

1. For X261 and X259,

   if both have 0 values the average testing time is close to 95. if X261 has 0 and X259 has 1, the average testing time is less than 90. if X261 has 1 and X259 has 0, the average testing time is close to 110.

1. For X261 and X257,

   if both have 0 values the average testing time is close to 95. if X261 has 0 and X257 has 1, the average testing time is less than 105. if X261 has 1 and X257 has 0, the average testing time is close to 110.

1. For X261 and X255,

   if both have 0 values the average testing time is close to 95. if X261 has 0 and X255 has 1, the average testing time is more than 110. if X261 has 1 and X257 has 0, the average testing time is close to 110. if both have value 1, the average testing time is close to 110.

1. For X260 and X256,

   if both have 0 values the average testing time is close to 100. if X260 has 0 and X256 has 1, the average testing time is less than 100. if both have value 1, the average testing time is more than 120.

1. For X259 and X256,

   if both have 0 values the average testing time is close to 100. if X259 has 0 and X256 has 1, the average testing time is close to 94. if X259 has 1 and X256 has 0, the average testing time is more than 120.

1. For X258 and X256,

   if both have 0 values the average testing time is close to 101. if X258 has 0 and X256 has 1, the average testing time is close to 94. if X258 has 1 and X256 has 0, the average testing time is more than 102.

1. For X257 and X256,

   if both have 0 values the average testing time is close to 101. if X257 has 0 and X256 has 1, the average testing time is close to 94. if X257 has 1 and X256 has 0, the average testing time is more than 105.

EDA - Conclusion

1. There are no null and duplicate values in the data.

2. Categorical features seem to hold more information as they are more widely present and also account for testing time geater than 130 and less than 80. Also they give information about for testing time between 80 and 130

3. The numerical features are either 0 or 1 with most of the values being 0.

4. The most important features are 'X0', 'X263', 'X262', 'X261', 'X260', 'X259', 'X258', 'X257', 'X256', 'X255'

5. The prediction column y has most values between 80 and 150.

Feature Engineering

In [37]:

```
# split into train test sets
from sklearn.model_selection import train_test_split
y = data['y']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1)
print("Done")
```

```
Done
```

Baseline model

model which outputs mean

In [38]:

```
y_pred_value  = y_train.mean()
y_pred = []
for i in range(0,len(y_test)):
    y_pred.append(y_pred_value)
```

In [39]:

```
from sklearn.metrics import r2_score
score = r2_score(y_test, y_pred)
print(score)
```

```
-0.002042687819177935
```

In [ ]:

```

```

In [ ]:

```

```

In [40]:

```
x.head()
```

Out[40]:

	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	...	X375	X376	X377	X378	X379	X380	X382	X383	X384	X385
0	32	23	17	0	3	24	9	14	0	0	...	0	0	1	0	0	0	0	0	0	0
1	32	21	19	4	3	28	11	14	0	0	...	1	0	0	0	0	0	0	0	0	0
2	20	24	34	2	3	27	9	23	0	0	...	0	0	0	0	0	0	1	0	0	0
3	20	21	34	5	3	27	11	4	0	0	...	0	0	0	0	0	0	0	0	0	0
4	20	23	34	5	3	12	3	13	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 376 columns

In [41]:

```
# split into train test sets
y = data['y']
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=1)
print("Done")
```

```
Done
```

In [43]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [44]:

```
 #train autoencoder for regression with no compression in the bottleneck layer
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.utils import plot_model
from matplotlib import pyplot
from keras import backend as K
```

In [45]:

```
n_inputs = x.shape[1]
```

In [47]:

```
# define encoder
input_data = Input(shape=(n_inputs,))
#encoder level 1
encoder = Dense(n_inputs*2)(input_data)
encoder = BatchNormalization()(encoder)
encoder = LeakyReLU()(encoder)

# define bottleneck
n_bottleneck = n_inputs
bottleneck = Dense(n_bottleneck)(encoder)

# decoder level 2
decoder = Dense(n_inputs*2)(bottleneck)
decoder = BatchNormalization()(decoder)
decoder = LeakyReLU()(decoder)

# output layer
output = Dense(n_inputs, activation='linear')(decoder)
# define autoencoder model
model = Model(inputs=input_data, outputs=output)
# compile autoencoder model
model.compile(optimizer='adam', loss='mse')
```

In [48]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 376)]             0         
_________________________________________________________________
dense_3 (Dense)              (None, 752)               283504    
_________________________________________________________________
batch_normalization_1 (Batch (None, 752)               3008      
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 752)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 376)               283128    
_________________________________________________________________
dense_5 (Dense)              (None, 752)               283504    
_________________________________________________________________
batch_normalization_2 (Batch (None, 752)               3008      
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 752)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 376)               283128    
=================================================================
Total params: 1,139,280
Trainable params: 1,136,272
Non-trainable params: 3,008
_________________________________________________________________
```

In [49]:

```
# plot the autoencoder
plot_model(model, 'autoencoder.png', show_shapes=True)

# fit the autoencoder model to reconstruct input
history = model.fit(X_train, y_train, epochs=400, batch_size=16, verbose=2, validation_data=(X_test
,y_test))
# plot loss
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
# define an encoder model (without the decoder)
encoder = Model(inputs=input_data, outputs=bottleneck)
plot_model(encoder, 'encoder.png', show_shapes=True)
# save the encoder to file
encoder.save('encoder.h5')
```

```
('Failed to import pydot. You must `pip install pydot` and install graphviz
(https://graphviz.gitlab.io/download/), ', 'for `pydotprint` to work.')
Epoch 1/400
176/176 - 3s - loss: 5556.4712 - val_loss: 1782.9253
Epoch 2/400
Epoch 2/400
176/176 - 1s - loss: 512.0928 - val_loss: 502.6837
Epoch 3/400
176/176 - 1s - loss: 157.7329 - val_loss: 185.0869
Epoch 4/400
176/176 - 1s - loss: 118.4300 - val_loss: 173.9447
Epoch 5/400
176/176 - 1s - loss: 103.6975 - val_loss: 139.9839
Epoch 6/400
176/176 - 1s - loss: 103.8155 - val_loss: 123.9464
Epoch 7/400
176/176 - 1s - loss: 96.5262 - val_loss: 150.4998
Epoch 8/400
176/176 - 1s - loss: 90.9939 - val_loss: 95.8988
Epoch 9/400
176/176 - 1s - loss: 81.2611 - val_loss: 100.8224
Epoch 10/400
176/176 - 2s - loss: 81.6729 - val_loss: 89.8388
Epoch 11/400
176/176 - 2s - loss: 83.3211 - val_loss: 96.6176
Epoch 12/400
176/176 - 2s - loss: 81.8898 - val_loss: 107.0758
Epoch 13/400
176/176 - 2s - loss: 77.1092 - val_loss: 87.3325
Epoch 14/400
176/176 - 1s - loss: 71.3312 - val_loss: 96.0054
Epoch 15/400
176/176 - 1s - loss: 72.7723 - val_loss: 84.5512
Epoch 16/400
176/176 - 1s - loss: 70.3689 - val_loss: 77.4287
Epoch 17/400
176/176 - 1s - loss: 67.2233 - val_loss: 95.3801
Epoch 18/400
176/176 - 1s - loss: 70.7376 - val_loss: 68.7528
Epoch 19/400
176/176 - 1s - loss: 69.4516 - val_loss: 79.6886
Epoch 20/400
176/176 - 2s - loss: 67.8337 - val_loss: 89.5401
Epoch 21/400
176/176 - 1s - loss: 67.4541 - val_loss: 72.2064
Epoch 22/400
176/176 - 1s - loss: 64.2605 - val_loss: 75.5713
Epoch 23/400
176/176 - 1s - loss: 66.7956 - val_loss: 72.4099
Epoch 24/400
176/176 - 1s - loss: 62.0917 - val_loss: 70.9035
Epoch 25/400
176/176 - 1s - loss: 64.2485 - val_loss: 83.0614
Epoch 26/400
176/176 - 1s - loss: 63.6541 - val_loss: 78.8501
Epoch 27/400
176/176 - 1s - loss: 62.4848 - val_loss: 79.0196
Epoch 28/400
176/176 - 1s - loss: 62.4412 - val_loss: 69.9420
Epoch 29/400
176/176 - 1s - loss: 61.7483 - val_loss: 89.3455
Epoch 30/400
176/176 - 1s - loss: 63.3623 - val_loss: 90.2701
Epoch 31/400
176/176 - 1s - loss: 57.7759 - val_loss: 129.1003
Epoch 32/400
176/176 - 1s - loss: 59.6784 - val_loss: 78.8426
Epoch 33/400
176/176 - 1s - loss: 58.9369 - val_loss: 69.1881
Epoch 34/400
176/176 - 2s - loss: 59.3417 - val_loss: 80.6803
Epoch 35/400
176/176 - 1s - loss: 58.2560 - val_loss: 102.1302
Epoch 36/400
176/176 - 1s - loss: 55.7797 - val_loss: 69.9326
Epoch 37/400
176/176 - 1s - loss: 57.5114 - val_loss: 75.7685
Epoch 38/400
176/176 - 1s - loss: 55.5150 - val_loss: 69.6757
Epoch 39/400
176/176 - 1s - loss: 56.9763 - val_loss: 72.7902
Epoch 40/400
176/176 - 1s - loss: 55.7231 - val_loss: 80.6351
```

```
176/176 - 1s - loss: 55.7251 - val_loss: 80.6551
Epoch 41/400
176/176 - 1s - loss: 55.7523 - val_loss: 71.9818
Epoch 42/400
176/176 - 1s - loss: 55.4282 - val_loss: 82.2222
Epoch 43/400
176/176 - 1s - loss: 58.0019 - val_loss: 68.8024
Epoch 44/400
176/176 - 2s - loss: 56.5689 - val_loss: 68.1868
Epoch 45/400
176/176 - 1s - loss: 55.3645 - val_loss: 71.9460
Epoch 46/400
176/176 - 1s - loss: 54.2749 - val_loss: 75.8361
Epoch 47/400
176/176 - 1s - loss: 54.9893 - val_loss: 74.0074
Epoch 48/400
176/176 - 1s - loss: 52.1161 - val_loss: 97.4380
Epoch 49/400
176/176 - 1s - loss: 54.3762 - val_loss: 80.5588
Epoch 50/400
176/176 - 1s - loss: 53.9672 - val_loss: 68.4819
Epoch 51/400
176/176 - 1s - loss: 51.8396 - val_loss: 84.6558
Epoch 52/400
176/176 - 1s - loss: 55.9465 - val_loss: 82.1754
Epoch 53/400
176/176 - 1s - loss: 52.2347 - val_loss: 70.3601
Epoch 54/400
176/176 - 1s - loss: 50.3280 - val_loss: 68.2681
Epoch 55/400
176/176 - 2s - loss: 48.3705 - val_loss: 73.2899
Epoch 56/400
176/176 - 2s - loss: 50.8121 - val_loss: 77.5329
Epoch 57/400
176/176 - 1s - loss: 50.8950 - val_loss: 76.2002
Epoch 58/400
176/176 - 1s - loss: 50.3799 - val_loss: 70.7951
Epoch 59/400
176/176 - 1s - loss: 50.9064 - val_loss: 69.5939
Epoch 60/400
176/176 - 1s - loss: 49.6096 - val_loss: 116.3060
Epoch 61/400
176/176 - 1s - loss: 49.0486 - val_loss: 82.4908
Epoch 62/400
176/176 - 2s - loss: 50.2605 - val_loss: 75.3310
Epoch 63/400
176/176 - 2s - loss: 49.5939 - val_loss: 78.6892
Epoch 64/400
176/176 - 2s - loss: 47.2187 - val_loss: 74.6893
Epoch 65/400
176/176 - 2s - loss: 49.2484 - val_loss: 70.3628
Epoch 66/400
176/176 - 2s - loss: 49.2426 - val_loss: 75.8183
Epoch 67/400
176/176 - 2s - loss: 49.5528 - val_loss: 74.9444
Epoch 68/400
176/176 - 2s - loss: 48.8073 - val_loss: 75.5855
Epoch 69/400
176/176 - 2s - loss: 48.3479 - val_loss: 81.4009
Epoch 70/400
176/176 - 2s - loss: 46.2783 - val_loss: 71.9666
Epoch 71/400
176/176 - 2s - loss: 47.9662 - val_loss: 78.7030
Epoch 72/400
176/176 - 2s - loss: 46.1597 - val_loss: 76.6311
Epoch 73/400
176/176 - 1s - loss: 47.4255 - val_loss: 71.9791
Epoch 74/400
176/176 - 1s - loss: 47.1181 - val_loss: 68.8333
Epoch 75/400
176/176 - 2s - loss: 45.9568 - val_loss: 71.2506
Epoch 76/400
176/176 - 1s - loss: 46.0177 - val_loss: 72.0966
Epoch 77/400
176/176 - 2s - loss: 48.3610 - val_loss: 68.0258
Epoch 78/400
176/176 - 2s - loss: 47.7975 - val_loss: 83.1112
Epoch 79/400
Epoch 79/400
176/176 - 2s - loss: 46.1358 - val_loss: 74.5534
Epoch 80/400
176/176 - 1s - loss: 45.3106 - val_loss: 68.8068
Epoch 81/400
176/176 - 1s - loss: 44.6888 - val_loss: 80.0350
Epoch 82/400
176/176 - 2s - loss: 46.3971 - val_loss: 88.4229
Epoch 83/400
176/176 - 1s - loss: 45.4874 - val_loss: 72.0487
Epoch 84/400
176/176 - 2s - loss: 45.3525 - val_loss: 77.1295
Epoch 85/400
176/176 - 2s - loss: 44.0092 - val_loss: 74.9170
Epoch 86/400
176/176 - 1s - loss: 44.1021 - val_loss: 77.3125
Epoch 87/400
176/176 - 2s - loss: 44.8831 - val_loss: 88.4788
Epoch 88/400
176/176 - 2s - loss: 46.6648 - val_loss: 79.0569
Epoch 89/400
176/176 - 1s - loss: 45.0049 - val_loss: 72.1549
Epoch 90/400
176/176 - 1s - loss: 44.5933 - val_loss: 71.3396
Epoch 91/400
176/176 - 1s - loss: 43.2731 - val_loss: 77.6449
Epoch 92/400
176/176 - 1s - loss: 45.2739 - val_loss: 75.3588
Epoch 93/400
176/176 - 1s - loss: 44.1981 - val_loss: 79.9160
Epoch 94/400
176/176 - 1s - loss: 44.9881 - val_loss: 73.5743
Epoch 95/400
176/176 - 1s - loss: 43.6127 - val_loss: 77.8092
Epoch 96/400
176/176 - 1s - loss: 41.8999 - val_loss: 75.6418
Epoch 97/400
176/176 - 1s - loss: 42.2667 - val_loss: 81.7789
Epoch 98/400
176/176 - 1s - loss: 43.3752 - val_loss: 71.4186
Epoch 99/400
176/176 - 1s - loss: 41.2697 - val_loss: 72.7897
Epoch 100/400
176/176 - 1s - loss: 42.3867 - val_loss: 72.5481
Epoch 101/400
176/176 - 1s - loss: 42.0227 - val_loss: 83.8698
Epoch 102/400
176/176 - 1s - loss: 43.4074 - val_loss: 90.4554
Epoch 103/400
176/176 - 1s - loss: 41.8256 - val_loss: 72.8874
Epoch 104/400
176/176 - 1s - loss: 42.1047 - val_loss: 72.4228
Epoch 105/400
176/176 - 1s - loss: 41.3475 - val_loss: 82.7436
Epoch 106/400
176/176 - 1s - loss: 41.0212 - val_loss: 91.1728
Epoch 107/400
176/176 - 1s - loss: 41.4090 - val_loss: 69.4805
Epoch 108/400
176/176 - 1s - loss: 40.6836 - val_loss: 84.1482
Epoch 109/400
176/176 - 1s - loss: 42.0050 - val_loss: 76.3280
Epoch 110/400
176/176 - 1s - loss: 43.3149 - val_loss: 84.8022
Epoch 111/400
176/176 - 1s - loss: 40.4112 - val_loss: 70.7411
Epoch 112/400
176/176 - 1s - loss: 40.4491 - val_loss: 92.5038
Epoch 113/400
176/176 - 1s - loss: 42.2251 - val_loss: 80.5907
Epoch 114/400
176/176 - 1s - loss: 39.8923 - val_loss: 72.4437
Epoch 115/400
176/176 - 1s - loss: 40.6871 - val_loss: 75.0150
Epoch 116/400
176/176 - 1s - loss: 40.0313 - val_loss: 70.8639
Epoch 117/400
176/176 - 1s - loss: 39.4257 - val_loss: 76.6933
```

```
176/176 - 1s - loss: 39.4257 - val_loss: 76.6933
Epoch 118/400
176/176 - 1s - loss: 39.9244 - val_loss: 85.4239
Epoch 119/400
176/176 - 1s - loss: 41.9549 - val_loss: 70.8209
Epoch 120/400
176/176 - 2s - loss: 39.7994 - val_loss: 79.2041
Epoch 121/400
176/176 - 1s - loss: 39.5110 - val_loss: 90.4877
Epoch 122/400
176/176 - 2s - loss: 39.6735 - val_loss: 69.8902
Epoch 123/400
176/176 - 1s - loss: 39.8406 - val_loss: 75.8592
Epoch 124/400
176/176 - 1s - loss: 40.7664 - val_loss: 74.2802
Epoch 125/400
176/176 - 1s - loss: 38.3389 - val_loss: 80.3906
Epoch 126/400
176/176 - 1s - loss: 38.8220 - val_loss: 84.4167
Epoch 127/400
176/176 - 1s - loss: 39.5303 - val_loss: 72.9624
Epoch 128/400
176/176 - 1s - loss: 38.9721 - val_loss: 85.5988
Epoch 129/400
176/176 - 1s - loss: 38.6948 - val_loss: 76.2838
Epoch 130/400
176/176 - 1s - loss: 40.1375 - val_loss: 83.0488
Epoch 131/400
176/176 - 1s - loss: 38.7235 - val_loss: 84.3676
Epoch 132/400
176/176 - 1s - loss: 39.0681 - val_loss: 87.5428
Epoch 133/400
176/176 - 1s - loss: 37.2937 - val_loss: 77.0291
Epoch 134/400
176/176 - 1s - loss: 38.5966 - val_loss: 78.6316
Epoch 135/400
176/176 - 1s - loss: 37.5467 - val_loss: 82.6833
Epoch 136/400
176/176 - 1s - loss: 39.1766 - val_loss: 101.6048
Epoch 137/400
176/176 - 1s - loss: 38.8729 - val_loss: 74.1701
Epoch 138/400
176/176 - 1s - loss: 38.0273 - val_loss: 79.3542
Epoch 139/400
176/176 - 1s - loss: 40.5920 - val_loss: 75.8452
Epoch 140/400
176/176 - 1s - loss: 38.2868 - val_loss: 74.4754
Epoch 141/400
176/176 - 1s - loss: 38.3757 - val_loss: 71.6737
Epoch 142/400
176/176 - 1s - loss: 36.8867 - val_loss: 79.7674
Epoch 143/400
176/176 - 1s - loss: 37.5809 - val_loss: 75.8729
Epoch 144/400
176/176 - 1s - loss: 37.6637 - val_loss: 77.0694
Epoch 145/400
176/176 - 1s - loss: 36.7569 - val_loss: 85.1389
Epoch 146/400
176/176 - 2s - loss: 36.6986 - val_loss: 77.8608
Epoch 147/400
176/176 - 1s - loss: 37.9202 - val_loss: 77.6600
Epoch 148/400
176/176 - 1s - loss: 35.6639 - val_loss: 71.4464
Epoch 149/400
176/176 - 1s - loss: 38.6539 - val_loss: 77.4919
Epoch 150/400
176/176 - 1s - loss: 36.5671 - val_loss: 71.8748
Epoch 151/400
176/176 - 1s - loss: 37.2146 - val_loss: 69.9226
Epoch 152/400
176/176 - 1s - loss: 36.1470 - val_loss: 76.3923
Epoch 153/400
176/176 - 1s - loss: 38.3597 - val_loss: 73.9579
Epoch 154/400
176/176 - 1s - loss: 36.6978 - val_loss: 75.9008
Epoch 155/400
176/176 - 1s - loss: 35.7935 - val_loss: 79.5125
Epoch 156/400
Epoch 156/400
176/176 - 1s - loss: 37.2873 - val_loss: 81.0484
Epoch 157/400
176/176 - 1s - loss: 35.5469 - val_loss: 73.7766
Epoch 158/400
176/176 - 1s - loss: 36.1095 - val_loss: 79.8868
Epoch 159/400
176/176 - 1s - loss: 36.7857 - val_loss: 82.2789
Epoch 160/400
176/176 - 1s - loss: 36.6842 - val_loss: 79.5393
Epoch 161/400
176/176 - 1s - loss: 35.7019 - val_loss: 72.8999
Epoch 162/400
176/176 - 1s - loss: 36.6630 - val_loss: 72.7392
Epoch 163/400
176/176 - 1s - loss: 36.9806 - val_loss: 72.3361
Epoch 164/400
176/176 - 1s - loss: 36.7301 - val_loss: 83.9760
Epoch 165/400
176/176 - 1s - loss: 34.5785 - val_loss: 75.7390
Epoch 166/400
176/176 - 1s - loss: 35.9035 - val_loss: 71.9462
Epoch 167/400
176/176 - 1s - loss: 36.2283 - val_loss: 78.9382
Epoch 168/400
176/176 - 1s - loss: 35.7518 - val_loss: 76.2076
Epoch 169/400
176/176 - 1s - loss: 34.8027 - val_loss: 76.7373
Epoch 170/400
176/176 - 2s - loss: 35.6562 - val_loss: 75.1471
Epoch 171/400
176/176 - 2s - loss: 35.7009 - val_loss: 73.0736
Epoch 172/400
176/176 - 2s - loss: 35.3250 - val_loss: 83.5773
Epoch 173/400
176/176 - 2s - loss: 35.2278 - val_loss: 80.2568
Epoch 174/400
176/176 - 2s - loss: 37.5342 - val_loss: 74.3951
Epoch 175/400
176/176 - 2s - loss: 36.2436 - val_loss: 73.1516
Epoch 176/400
176/176 - 1s - loss: 37.3877 - val_loss: 71.5289
Epoch 177/400
176/176 - 2s - loss: 33.7517 - val_loss: 71.1902
Epoch 178/400
176/176 - 2s - loss: 34.2610 - val_loss: 76.1464
Epoch 179/400
176/176 - 1s - loss: 35.7776 - val_loss: 75.9002
Epoch 180/400
176/176 - 1s - loss: 37.3090 - val_loss: 74.5098
Epoch 181/400
176/176 - 2s - loss: 34.2154 - val_loss: 80.2562
Epoch 182/400
176/176 - 2s - loss: 35.1068 - val_loss: 75.5300
Epoch 183/400
176/176 - 1s - loss: 34.4095 - val_loss: 88.4230
Epoch 184/400
176/176 - 2s - loss: 35.8921 - val_loss: 70.9561
Epoch 185/400
176/176 - 1s - loss: 34.8029 - val_loss: 73.4762
Epoch 186/400
176/176 - 1s - loss: 33.2377 - val_loss: 69.6971
Epoch 187/400
176/176 - 1s - loss: 34.5201 - val_loss: 94.0209
Epoch 188/400
176/176 - 1s - loss: 33.9559 - val_loss: 71.6596
Epoch 189/400
176/176 - 1s - loss: 33.5308 - val_loss: 74.7829
Epoch 190/400
176/176 - 1s - loss: 35.6291 - val_loss: 77.6493
Epoch 191/400
176/176 - 1s - loss: 34.1195 - val_loss: 74.9060
Epoch 192/400
176/176 - 1s - loss: 33.7269 - val_loss: 73.9093
Epoch 193/400
176/176 - 1s - loss: 33.5741 - val_loss: 69.6250
Epoch 194/400
```

```
176/176 - 1s - loss: 34.7942 - val_loss: 76.7422
Epoch 195/400
176/176 - 1s - loss: 34.8592 - val_loss: 77.1596
Epoch 196/400
176/176 - 1s - loss: 34.1864 - val_loss: 78.6716
Epoch 197/400
176/176 - 1s - loss: 33.5562 - val_loss: 77.1417
Epoch 198/400
176/176 - 1s - loss: 34.5864 - val_loss: 70.1996
Epoch 199/400
176/176 - 1s - loss: 32.8751 - val_loss: 73.8105
Epoch 200/400
176/176 - 1s - loss: 33.0441 - val_loss: 73.5270
Epoch 201/400
176/176 - 1s - loss: 34.0390 - val_loss: 80.0576
Epoch 202/400
176/176 - 2s - loss: 32.2648 - val_loss: 81.1023
Epoch 203/400
176/176 - 2s - loss: 32.6343 - val_loss: 88.7597
Epoch 204/400
176/176 - 1s - loss: 33.5313 - val_loss: 74.1847
Epoch 205/400
176/176 - 1s - loss: 32.4264 - val_loss: 73.0720
Epoch 206/400
176/176 - 1s - loss: 33.6329 - val_loss: 75.8854
Epoch 207/400
176/176 - 1s - loss: 33.1059 - val_loss: 81.6454
Epoch 208/400
176/176 - 1s - loss: 34.8373 - val_loss: 80.9128
Epoch 209/400
176/176 - 1s - loss: 32.5761 - val_loss: 73.4668
Epoch 210/400
176/176 - 1s - loss: 34.1607 - val_loss: 78.2958
Epoch 211/400
176/176 - 1s - loss: 33.4020 - val_loss: 88.0258
Epoch 212/400
176/176 - 1s - loss: 33.2084 - val_loss: 78.0507
Epoch 213/400
176/176 - 1s - loss: 33.3372 - val_loss: 80.1686
Epoch 214/400
176/176 - 1s - loss: 32.3672 - val_loss: 74.1035
Epoch 215/400
176/176 - 2s - loss: 33.5847 - val_loss: 76.4576
Epoch 216/400
176/176 - 1s - loss: 33.3927 - val_loss: 75.0087
Epoch 217/400
176/176 - 1s - loss: 32.1470 - val_loss: 70.1487
Epoch 218/400
176/176 - 1s - loss: 33.3173 - val_loss: 77.5680
Epoch 219/400
176/176 - 1s - loss: 31.5694 - val_loss: 78.8978
Epoch 220/400
176/176 - 1s - loss: 31.8963 - val_loss: 79.4470
Epoch 221/400
176/176 - 1s - loss: 32.6587 - val_loss: 73.2973
Epoch 222/400
176/176 - 2s - loss: 31.6694 - val_loss: 74.5059
Epoch 223/400
176/176 - 1s - loss: 33.2405 - val_loss: 78.3112
Epoch 224/400
176/176 - 1s - loss: 33.4026 - val_loss: 86.2959
Epoch 225/400
176/176 - 2s - loss: 31.6469 - val_loss: 70.7184
Epoch 226/400
176/176 - 1s - loss: 33.3436 - val_loss: 80.4201
Epoch 227/400
176/176 - 2s - loss: 32.5376 - val_loss: 77.0959
Epoch 228/400
176/176 - 2s - loss: 32.4026 - val_loss: 82.0631
Epoch 229/400
176/176 - 2s - loss: 32.9145 - val_loss: 78.0140
Epoch 230/400
176/176 - 1s - loss: 31.9820 - val_loss: 86.1144
Epoch 231/400
176/176 - 1s - loss: 32.4404 - val_loss: 79.7252
Epoch 232/400
176/176 - 2s - loss: 32.8413 - val_loss: 76.6440
```

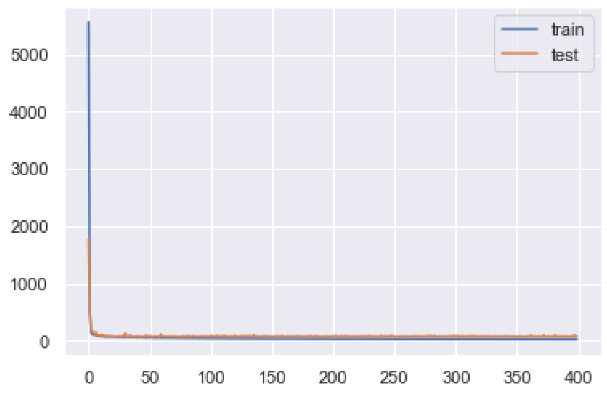
```
Epoch 233/400
176/176 - 1s - loss: 32.7205 - val_loss: 75.1156
Epoch 234/400
176/176 - 1s - loss: 31.9463 - val_loss: 73.6684
Epoch 235/400
176/176 - 1s - loss: 31.1170 - val_loss: 78.9801
Epoch 236/400
176/176 - 2s - loss: 31.2139 - val_loss: 74.3570
Epoch 237/400
176/176 - 2s - loss: 32.6895 - val_loss: 75.6303
Epoch 238/400
176/176 - 2s - loss: 31.3195 - val_loss: 77.1245
Epoch 239/400
176/176 - 1s - loss: 31.6636 - val_loss: 73.5255
Epoch 240/400
176/176 - 1s - loss: 32.8576 - val_loss: 93.7406
Epoch 241/400
176/176 - 2s - loss: 32.0798 - val_loss: 75.8067
Epoch 242/400
176/176 - 2s - loss: 31.0412 - val_loss: 76.6846
Epoch 243/400
176/176 - 2s - loss: 33.1256 - val_loss: 78.1815
Epoch 244/400
176/176 - 2s - loss: 30.9050 - val_loss: 84.0156
Epoch 245/400
176/176 - 2s - loss: 31.0625 - val_loss: 75.4825
Epoch 246/400
176/176 - 2s - loss: 31.1791 - val_loss: 77.4769
Epoch 247/400
176/176 - 2s - loss: 31.3237 - val_loss: 75.6801
Epoch 248/400
176/176 - 1s - loss: 30.7428 - val_loss: 93.4244
Epoch 249/400
176/176 - 1s - loss: 31.8361 - val_loss: 80.6102
Epoch 250/400
176/176 - 1s - loss: 31.7206 - val_loss: 79.6452
Epoch 251/400
176/176 - 1s - loss: 32.0795 - val_loss: 72.8468
Epoch 252/400
176/176 - 2s - loss: 30.5012 - val_loss: 74.7136
Epoch 253/400
176/176 - 2s - loss: 31.7176 - val_loss: 69.1903
Epoch 254/400
176/176 - 2s - loss: 30.0034 - val_loss: 74.7954
Epoch 255/400
176/176 - 1s - loss: 30.8005 - val_loss: 69.9461
Epoch 256/400
176/176 - 2s - loss: 31.2940 - val_loss: 75.2249
Epoch 257/400
176/176 - 2s - loss: 31.5190 - val_loss: 78.4646
Epoch 258/400
176/176 - 2s - loss: 29.9474 - val_loss: 75.6548
Epoch 259/400
176/176 - 2s - loss: 32.8314 - val_loss: 77.7458
Epoch 260/400
176/176 - 2s - loss: 30.5251 - val_loss: 76.5253
Epoch 261/400
176/176 - 2s - loss: 32.1527 - val_loss: 71.4029
Epoch 262/400
176/176 - 2s - loss: 31.3454 - val_loss: 76.3403
Epoch 263/400
176/176 - 2s - loss: 31.1141 - val_loss: 74.5195
Epoch 264/400
176/176 - 1s - loss: 30.2028 - val_loss: 73.1168
Epoch 265/400
176/176 - 2s - loss: 30.5038 - val_loss: 72.3241
Epoch 266/400
176/176 - 2s - loss: 29.9661 - val_loss: 77.0572
Epoch 267/400
176/176 - 2s - loss: 31.1436 - val_loss: 76.7154
Epoch 268/400
176/176 - 2s - loss: 30.1253 - val_loss: 78.9260
Epoch 269/400
176/176 - 2s - loss: 29.6429 - val_loss: 81.1323
Epoch 270/400
176/176 - 1s - loss: 29.3655 - val_loss: 81.5487
Epoch 271/400
```

```
176/176 - 2s - loss: 29.9119 - val_loss: 73.0393
Epoch 272/400
176/176 - 2s - loss: 29.5002 - val_loss: 81.1808
Epoch 273/400
176/176 - 1s - loss: 30.1194 - val_loss: 73.3678
Epoch 274/400
176/176 - 2s - loss: 30.2415 - val_loss: 77.3867
Epoch 275/400
176/176 - 1s - loss: 29.9945 - val_loss: 77.6692
Epoch 276/400
176/176 - 1s - loss: 30.2368 - val_loss: 79.0955
Epoch 277/400
176/176 - 1s - loss: 28.7475 - val_loss: 76.2302
Epoch 278/400
176/176 - 1s - loss: 30.0776 - val_loss: 85.2428
Epoch 279/400
176/176 - 2s - loss: 29.0585 - val_loss: 83.8499
Epoch 280/400
176/176 - 2s - loss: 30.5519 - val_loss: 86.4913
Epoch 281/400
176/176 - 2s - loss: 30.2695 - val_loss: 87.6279
Epoch 282/400
176/176 - 2s - loss: 30.0004 - val_loss: 74.3120
Epoch 283/400
176/176 - 1s - loss: 28.8607 - val_loss: 78.8665
Epoch 284/400
176/176 - 2s - loss: 28.9155 - val_loss: 86.9294
Epoch 285/400
176/176 - 2s - loss: 29.2397 - val_loss: 73.1386
Epoch 286/400
176/176 - 2s - loss: 30.5101 - val_loss: 71.4213
Epoch 287/400
176/176 - 2s - loss: 29.7629 - val_loss: 73.3077
Epoch 288/400
176/176 - 2s - loss: 29.0188 - val_loss: 78.4830
Epoch 289/400
176/176 - 2s - loss: 29.9700 - val_loss: 74.9051
Epoch 290/400
176/176 - 1s - loss: 30.6656 - val_loss: 74.3258
Epoch 291/400
176/176 - 1s - loss: 28.8422 - val_loss: 75.3579
Epoch 292/400
176/176 - 2s - loss: 29.5466 - val_loss: 74.6152
Epoch 293/400
176/176 - 2s - loss: 29.2327 - val_loss: 74.1301
Epoch 294/400
176/176 - 2s - loss: 29.3853 - val_loss: 74.6813
Epoch 295/400
176/176 - 1s - loss: 28.3953 - val_loss: 77.9268
Epoch 296/400
176/176 - 1s - loss: 29.4573 - val_loss: 86.5220
Epoch 297/400
176/176 - 2s - loss: 29.6561 - val_loss: 77.7417
Epoch 298/400
176/176 - 2s - loss: 29.2926 - val_loss: 75.2371
Epoch 299/400
176/176 - 1s - loss: 28.9144 - val_loss: 85.9978
Epoch 300/400
176/176 - 2s - loss: 29.9135 - val_loss: 79.4807
Epoch 301/400
176/176 - 2s - loss: 29.8883 - val_loss: 77.0879
Epoch 302/400
176/176 - 2s - loss: 29.0304 - val_loss: 92.9905
Epoch 303/400
176/176 - 2s - loss: 29.3857 - val_loss: 79.7454
Epoch 304/400
176/176 - 1s - loss: 30.1818 - val_loss: 74.3419
Epoch 305/400
176/176 - 2s - loss: 28.3700 - val_loss: 80.7052
Epoch 306/400
176/176 - 2s - loss: 29.5029 - val_loss: 78.6330
Epoch 307/400
176/176 - 2s - loss: 28.5035 - val_loss: 74.6445
Epoch 308/400
176/176 - 2s - loss: 27.5927 - val_loss: 76.1905
Epoch 309/400
176/176 - 2s - loss: 29.3570 - val_loss: 79.7515
```

```
Epoch 310/400
176/176 - 2s - loss: 29.3134 - val_loss: 72.8558
Epoch 311/400
176/176 - 2s - loss: 29.7900 - val_loss: 79.1726
Epoch 312/400
176/176 - 1s - loss: 28.7743 - val_loss: 81.3378
Epoch 313/400
176/176 - 1s - loss: 27.8303 - val_loss: 81.9241
Epoch 314/400
176/176 - 2s - loss: 28.1440 - val_loss: 75.4078
Epoch 315/400
176/176 - 2s - loss: 27.4664 - val_loss: 89.7734
Epoch 316/400
176/176 - 1s - loss: 28.2456 - val_loss: 77.6505
Epoch 317/400
176/176 - 1s - loss: 29.5400 - val_loss: 84.6505
Epoch 318/400
176/176 - 1s - loss: 30.0165 - val_loss: 78.4435
Epoch 319/400
176/176 - 1s - loss: 28.2976 - val_loss: 71.4600
Epoch 320/400
176/176 - 2s - loss: 29.0055 - val_loss: 73.8231
Epoch 321/400
176/176 - 2s - loss: 28.4978 - val_loss: 76.7961
Epoch 322/400
176/176 - 2s - loss: 28.1603 - val_loss: 81.2641
Epoch 323/400
176/176 - 2s - loss: 29.0493 - val_loss: 73.6902
Epoch 324/400
176/176 - 2s - loss: 28.9681 - val_loss: 79.7523
Epoch 325/400
176/176 - 2s - loss: 29.7955 - val_loss: 74.4852
Epoch 326/400
176/176 - 2s - loss: 29.2592 - val_loss: 77.8949
Epoch 327/400
176/176 - 2s - loss: 28.5973 - val_loss: 80.7866
Epoch 328/400
176/176 - 2s - loss: 27.6404 - val_loss: 80.3360
Epoch 329/400
176/176 - 1s - loss: 28.3976 - val_loss: 73.5325
Epoch 330/400
176/176 - 2s - loss: 28.7027 - val_loss: 73.5539
Epoch 331/400
176/176 - 2s - loss: 27.9522 - val_loss: 76.1667
Epoch 332/400
176/176 - 1s - loss: 28.2185 - val_loss: 70.3990
Epoch 333/400
176/176 - 1s - loss: 28.8799 - val_loss: 80.1630
Epoch 334/400
176/176 - 1s - loss: 27.3085 - val_loss: 75.3989
Epoch 335/400
176/176 - 2s - loss: 28.2997 - val_loss: 72.2734
Epoch 336/400
176/176 - 2s - loss: 27.7024 - val_loss: 77.4501
Epoch 337/400
176/176 - 1s - loss: 28.6147 - val_loss: 72.9833
Epoch 338/400
176/176 - 2s - loss: 27.6337 - val_loss: 74.2442
Epoch 339/400
176/176 - 1s - loss: 29.0741 - val_loss: 69.4119
Epoch 340/400
176/176 - 1s - loss: 27.6802 - val_loss: 80.4398
Epoch 341/400
176/176 - 1s - loss: 27.1109 - val_loss: 79.7493
Epoch 342/400
176/176 - 1s - loss: 27.1231 - val_loss: 77.3531
Epoch 343/400
176/176 - 2s - loss: 26.9284 - val_loss: 76.3838
Epoch 344/400
176/176 - 2s - loss: 27.4961 - val_loss: 80.0111
Epoch 345/400
176/176 - 2s - loss: 26.8155 - val_loss: 77.2113
Epoch 346/400
176/176 - 1s - loss: 27.0372 - val_loss: 79.3195
Epoch 347/400
176/176 - 1s - loss: 28.3604 - val_loss: 76.1731
Epoch 348/400
```

```
176/176 - 1s - loss: 28.8727 - val_loss: 76.6587
Epoch 349/400
176/176 - 1s - loss: 28.9531 - val_loss: 75.6275
Epoch 350/400
176/176 - 1s - loss: 28.1907 - val_loss: 72.4509
Epoch 351/400
176/176 - 1s - loss: 26.9136 - val_loss: 73.8234
Epoch 352/400
176/176 - 1s - loss: 27.3013 - val_loss: 71.9258
Epoch 353/400
176/176 - 1s - loss: 27.2092 - val_loss: 77.1226
Epoch 354/400
176/176 - 1s - loss: 26.5516 - val_loss: 81.7296
Epoch 355/400
176/176 - 1s - loss: 28.4366 - val_loss: 72.4548
Epoch 356/400
176/176 - 2s - loss: 26.4347 - val_loss: 73.4862
Epoch 357/400
176/176 - 2s - loss: 27.6992 - val_loss: 72.6641
Epoch 358/400
176/176 - 2s - loss: 27.7462 - val_loss: 71.4602
Epoch 359/400
176/176 - 1s - loss: 26.5838 - val_loss: 73.5862
Epoch 360/400
176/176 - 1s - loss: 27.1730 - val_loss: 76.2618
Epoch 361/400
176/176 - 1s - loss: 26.4260 - val_loss: 73.2395
Epoch 362/400
176/176 - 1s - loss: 27.5620 - val_loss: 95.1904
Epoch 363/400
176/176 - 1s - loss: 26.6812 - val_loss: 81.2416
Epoch 364/400
176/176 - 1s - loss: 28.4071 - val_loss: 80.0672
Epoch 365/400
176/176 - 1s - loss: 28.1897 - val_loss: 70.8201
Epoch 366/400
176/176 - 1s - loss: 27.7077 - val_loss: 74.4439
Epoch 367/400
176/176 - 1s - loss: 27.6775 - val_loss: 74.3842
Epoch 368/400
176/176 - 1s - loss: 27.4811 - val_loss: 78.5466
Epoch 369/400
176/176 - 1s - loss: 27.8199 - val_loss: 73.9816
Epoch 370/400
176/176 - 2s - loss: 27.3817 - val_loss: 75.2754
Epoch 371/400
176/176 - 1s - loss: 28.0411 - val_loss: 76.7792
Epoch 372/400
176/176 - 2s - loss: 26.7670 - val_loss: 80.5833
Epoch 373/400
176/176 - 2s - loss: 26.7673 - val_loss: 100.8869
Epoch 374/400
176/176 - 2s - loss: 26.4046 - val_loss: 70.5768
Epoch 375/400
176/176 - 1s - loss: 28.5220 - val_loss: 73.6651
Epoch 376/400
176/176 - 1s - loss: 27.7823 - val_loss: 80.3450
Epoch 377/400
176/176 - 1s - loss: 29.3983 - val_loss: 71.6695
Epoch 378/400
176/176 - 1s - loss: 27.0561 - val_loss: 77.4852
Epoch 379/400
176/176 - 1s - loss: 28.5193 - val_loss: 72.5746
Epoch 380/400
176/176 - 1s - loss: 27.3356 - val_loss: 72.5576
Epoch 381/400
176/176 - 1s - loss: 28.4910 - val_loss: 73.1514
Epoch 382/400
176/176 - 1s - loss: 25.2965 - val_loss: 100.4141
Epoch 383/400
176/176 - 1s - loss: 27.0661 - val_loss: 83.8512
Epoch 384/400
176/176 - 1s - loss: 27.7197 - val_loss: 72.4956
Epoch 385/400
176/176 - 1s - loss: 26.1169 - val_loss: 81.5255
Epoch 386/400
176/176 - 2s - loss: 26.8439 - val_loss: 70.0337
```

```
Epoch 387/400
176/176 - 2s - loss: 26.9906 - val_loss: 73.9399
Epoch 388/400
176/176 - 2s - loss: 28.2546 - val_loss: 80.6196
Epoch 389/400
176/176 - 2s - loss: 26.9869 - val_loss: 71.9728
Epoch 390/400
176/176 - 2s - loss: 27.2919 - val_loss: 79.6705
Epoch 391/400
176/176 - 1s - loss: 26.9564 - val_loss: 74.6817
Epoch 392/400
176/176 - 1s - loss: 25.2662 - val_loss: 75.7077
Epoch 393/400
176/176 - 1s - loss: 27.8869 - val_loss: 73.6118
Epoch 394/400
176/176 - 1s - loss: 27.1275 - val_loss: 71.6015
Epoch 395/400
176/176 - 1s - loss: 26.2813 - val_loss: 77.7321
Epoch 396/400
176/176 - 1s - loss: 25.7307 - val_loss: 78.2033
Epoch 397/400
176/176 - 1s - loss: 25.9058 - val_loss: 70.2237
Epoch 398/400
176/176 - 1s - loss: 27.6556 - val_loss: 105.7201
Epoch 399/400
176/176 - 1s - loss: 27.7573 - val_loss: 79.2037
Epoch 400/400
176/176 - 1s - loss: 26.5484 - val_loss: 79.5344
```



```
('Failed to import pydot. You must `pip install pydot` and install graphviz
(https://graphviz.gitlab.io/download/), ', 'for `pydotprint` to work.')
```

In [50]:

```
from tensorflow.keras.models import load_model
# load the model from file
encoder = load_model('encoder.h5')
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not*
compiled. Compile it manually.
```

In [51]:

```
X_train_encode = encoder.predict(X_train)
# encode the test data
X_test_encode = encoder.predict(X_test)
```

In [52]:

```
X_train_encode.shape
```

Out[52]:

```
(2811, 376)
```

Random XGBoost Model with encoded Features

In [53]:

```
reg = XGBRegressor(n_estimators=100, learning_rate = 0.1)
reg.fit(X_train_encode,y_train)
y_pred = reg.predict(X_test_encode)
score = r2_score(y_test, y_pred)
print(score)
```

```
0.4418955976553791
```

Observation -

The Random XGBoost Model with encoded features produces better r2 than Simple mean model

Some Other Feature Engg Techniques

PCA Features

In [54]:

```
from sklearn.decomposition import PCA
#taking top 10 components
components = 10
pca = PCA(n_components=components, random_state=420)

x_pca = pd.DataFrame(pca.fit_transform(x_num))

print(x_pca.shape)
print(x_pca.head())
```

```
(4196, 10)
          0         1         2         3         4         5         6  \
0  0.749079  2.255215  1.016952  0.929835  1.388466  0.044720  0.608854
1 -0.216276  1.103237 -0.832779 -0.670249  0.243599  0.037451  1.199850
2 -0.888967  2.978889  0.269069  2.570786 -0.994356  3.279242 -0.877726
3 -0.509223  2.445044 -0.645017  2.985013 -1.728502  3.132381  0.108157
4 -0.488367  2.236544 -0.787629  3.193679 -2.052340  3.167224 -0.091210

          7         8         9
0 -0.929240  0.200321 -0.730352
1 -0.563451 -0.083035  0.459006
2  0.533689 -0.939597 -0.102745
3  0.017575 -1.028170  0.260647
4  0.125130 -1.729599 -0.344936
```

SVD

In [55]:

```
# get the matrix factors
U, S, VT = np.linalg.svd(x_num,full_matrices=1)
# calculating the aspect ratio b
m = x_num.shape[1]
n = x_num.shape[0]
b = m/n

#taking w_b from table correspondng to b
w_b = 1.6089

# getting the median singular value
ymed = np.median(S)

# finding the  Hard threshold
cutoff = w_b * ymed
```

```
print("The Hard Threshold for Truncation = ",cutoff)
# get the number of components
r = np.max(np.where(S > cutoff))
print("Number of total components to be selected = ",r)
```

```
The Hard Threshold for Truncation =  3.8431396016312185
Number of total components to be selected =  152
```

In [56]:

```
from sklearn.decomposition import TruncatedSVD
n_comp = r

tsvd = TruncatedSVD(n_components=r, random_state=420)

x_svd= tsvd.fit_transform(x_num)


print(x_svd.shape)
```

```
(4196, 152)
```

In [ ]: