USE CASE

DESCRIPTION

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

Problem Statement Scenario:
Since the first automobile, the Benz Patent Motor Car in 1886, Mercedes-Benz has stood for important automotive innovations. These include the passenger safety cell with a crumple zone, the airbag, and intelligent assistance systems. Mercedes-Benz applies for nearly 2000 patents per year, making the brand the European leader among premium carmakers. Mercedes-Benz is the leader in the premium car industry. With a huge selection of features and options, customers can choose the customized Mercedes-Benz of their dreams.

To ensure the safety and reliability of every unique car configuration before they hit the road, the company's engineers have developed a robust testing system. As one of the world's biggest manufacturers of premium cars, safety and efficiency are paramount on Mercedes-Benz's production lines. However, optimizing the speed of their testing system for many possible feature combinations is complex and time-consuming without a powerful algorithmic approach.

You are required to reduce the time that cars spend on the test bench. Others will work with a dataset representing different permutations of features in a Mercedes-Benz car to predict the time it takes to pass testing. Optimal algorithms will contribute to faster testing, resulting in lower carbon dioxide emissions without reducing Mercedes-Benz's standards.

TASKS:

Following actions should be performed:

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

IMPORT LIBRARIES

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data_train = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Machine Learning/Assessments/Mercedes-Benz/train.csv")
data_train.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

In [3]:
```
 data_test = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Machine Learning/Assessments/Mercedes-Benz/test.csv")
data_test.head()
```

Out[3]:

	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

DATA ANALYSIS

Verify total no of records and features in train data set

In [4]:
```
print("Total records present in train data: ", data_train.shape, "\n")
print("Total records present in test data: ", data_test.shape, "\n")
```

```
Total records present in train data:  (4209, 378)

Total records present in test data:  (4209, 377)
```

Verify the Label feature

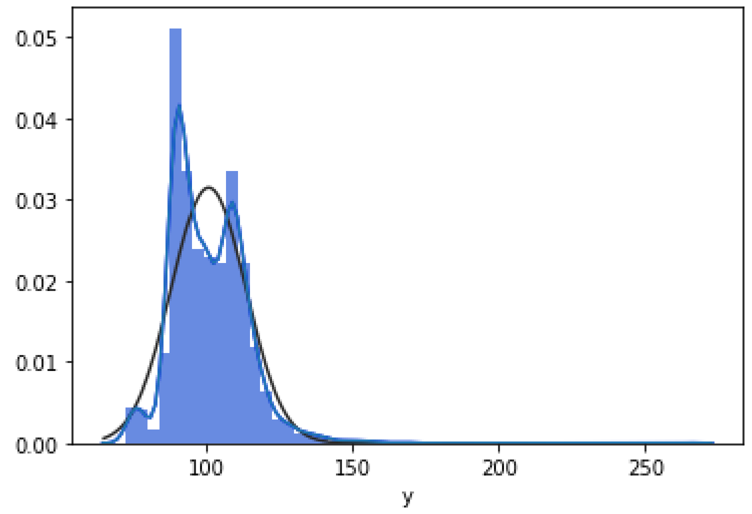
Observation: We can analyze that the train data has **378 records**, however test data has only **377 records**. There is a missing column in the Test data. Lets verify the missing data in the Test Data

In [5]:
```
for i in data_train.columns:
    if i in data_test.columns:
        continue
    else:
        print("The column not present in Test Data is: ", i)
```

```
The column not present in Test Data is:  y
```

Analysis: Since **y** is not present in the Test Data, Hence it is the **Output Variable**

In [6]:
```
# To detect any outliers, plot the y values.
sns.distplot(data_train.y, fit = norm, color='blue')
sns.distplot(data_train.y)
plt.show()
```



Verify the information about the data

In [7]:
```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4209 entries, 0 to 4208
Columns: 378 entries, ID to X385
dtypes: float64(1), int64(369), object(8)
memory usage: 12.1+ MB
```

Train data has 3 different data type features:
Float : 1 Feature
Int : 369 features
Object: 8 features

In [8]:
```
numericfeatures = [col for col in data_train.columns if (data_train[col].dtype == 'float64')
                   | (data_train[col].dtype == 'int64') & (col != 'Target')]
```

In [9]:
```
objectfeatures = [obj for obj in data_train.columns if data_train[obj].dtype == 'object']
objectfeatures
```

Out[9]:
```
['X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8']
```

TASK 1: If for any column(s), the variance is equal to zero, then you need to remove those variable(s).

In [10]:
```
dataVariance = data_train.var()
toBeDroppedRows = dataVariance[dataVariance == 0].index.to_list()
toBeDroppedRows
```

Out[10]:
```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

In [11]:
```
data_train.drop(toBeDroppedRows,axis=1,inplace=True)
```

In [12]:
```
data_train.shape
```

Out[12]:
```
(4209, 366)
```

TASK 2: Verify Null Values in Train Data

Dropping off the column names from **numericfeatures** which has **variance=0**

In [13]:
```
toBeDroppedRows = set(toBeDroppedRows)
numericfeatures = [items for items in numericfeatures if items not in toBeDroppedRows]
```

In [14]:
```
nullValues = data_train[numericfeatures].isnull().sum()
nullValues[nullValues > 0]
```
Out[14]:
```
Series([], dtype: int64)
```
In [15]:
```
data_train[objectfeatures].isnull().any()
```
Out[15]:
```
X0    False
X1    False
X2    False
X3    False
X4    False
X5    False
X6    False
X8    False
dtype: bool
```
In [16]:
```
data_train['y'].isnull().any()
```
Out[16]:
```
False
```
There is **No Null** value present in the **train data**

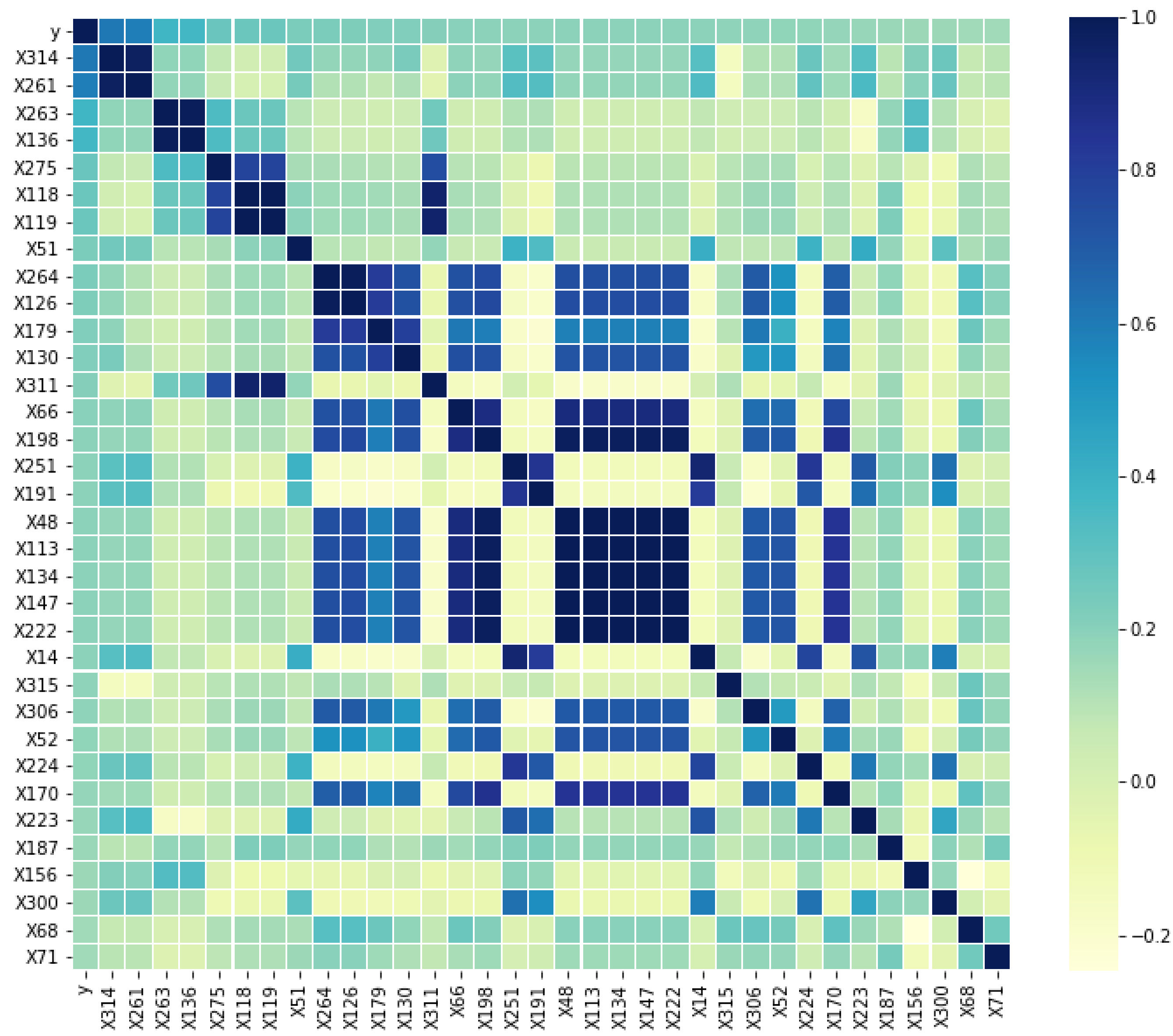
Verify the **unique values** in the **Categorical features>/b>**

In [17]:
```
for col in data_train[objectfeatures]:
    print("Unique values for: ", col)
    print(data_train[col].unique())
    print("\n")
```
```
Unique values for:  X0
['k' 'az' 't' 'al' 'o' 'w' 'j' 'h' 's' 'n' 'ay' 'f' 'x' 'y' 'aj' 'ak' 'am'
 'z' 'q' 'at' 'ap' 'v' 'af' 'a' 'e' 'ai' 'd' 'aq' 'c' 'aa' 'ba' 'as' 'i'
 'r' 'b' 'ax' 'bc' 'u' 'ad' 'au' 'm' 'l' 'aw' 'ao' 'ac' 'g' 'ab']


Unique values for:  X1
['v' 't' 'w' 'b' 'r' 'l' 's' 'aa' 'c' 'a' 'e' 'h' 'z' 'j' 'o' 'u' 'p' 'n'
 'i' 'y' 'd' 'f' 'm' 'k' 'g' 'q' 'ab']


Unique values for:  X2
['at' 'av' 'n' 'e' 'as' 'aq' 'r' 'ai' 'ak' 'm' 'a' 'k' 'ae' 's' 'f' 'd'
 'ag' 'ay' 'ac' 'ap' 'g' 'i' 'aw' 'y' 'b' 'ao' 'al' 'h' 'x' 'au' 't' 'an'
 'z' 'ah' 'p' 'am' 'j' 'q' 'af' 'l' 'aa' 'c' 'o' 'ar']


Unique values for:  X3
['a' 'e' 'c' 'f' 'd' 'b' 'g']


Unique values for:  X4
['d' 'b' 'c' 'a']


Unique values for:  X5
['u' 'y' 'x' 'h' 'g' 'f' 'j' 'i' 'd' 'c' 'af' 'ag' 'ab' 'ac' 'ad' 'ae'
 'ah' 'l' 'k' 'n' 'm' 'p' 'q' 's' 'r' 'v' 'w' 'o' 'aa']


Unique values for:  X6
['j' 'l' 'd' 'h' 'i' 'a' 'g' 'c' 'k' 'e' 'f' 'b']


Unique values for:  X8
['o' 'x' 'e' 'n' 's' 'a' 'h' 'p' 'm' 'k' 'd' 'i' 'v' 'j' 'b' 'q' 'w' 'g'
 'y' 'l' 'f' 'u' 'r' 't' 'c']
```

Check Corelation between the variables

In [18]:
```python
# y correlation matrix
# k : number of variables for heatmap
corrmat = data_train.corr()
k = 35

cols = corrmat.nlargest(k, 'y')['y'].index

cm = np.corrcoef(data_train[cols].values.T)
f, ax = plt.subplots(figsize =(12, 10))

sns.heatmap(cm, ax = ax, cmap ="YlGnBu",
            linewidths = 0.1, yticklabels = cols.values,
                              xticklabels = cols.values)
```

Out[18]: `<matplotlib.axes._subplots.AxesSubplot at 0x1ae8bb4c288>`



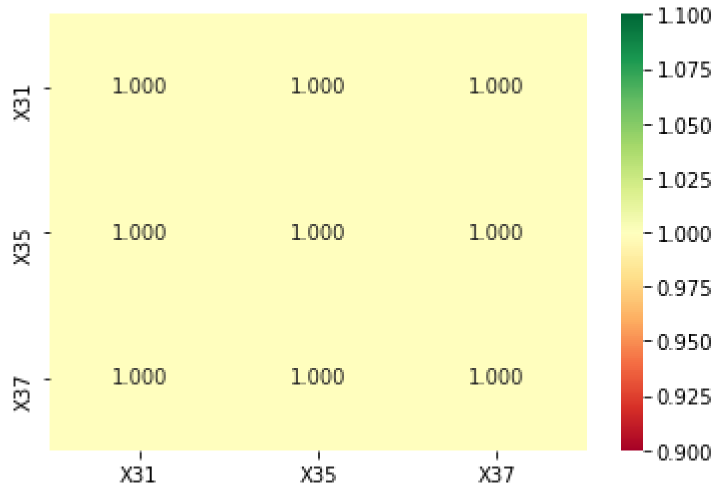
In [19]:
```python
# See the correlations in descending order
correlation = data_train.corr()
c1 = correlation.abs().unstack()
c1.sort_values(ascending = False)
c2 = c1[(c1 > 0.6) & (c1 < 1.0)]
c2
```

Out[19]:
```
y     X314    0.606005
X14   X178    0.832712
      X191    0.819047
      X223    0.727050
      X224    0.779874
                ...
X379  X63     0.911995
      X78     0.773115
      X174    0.737298
X382  X325    0.865197
X384  X366    0.632230
Length: 1296, dtype: float64
```

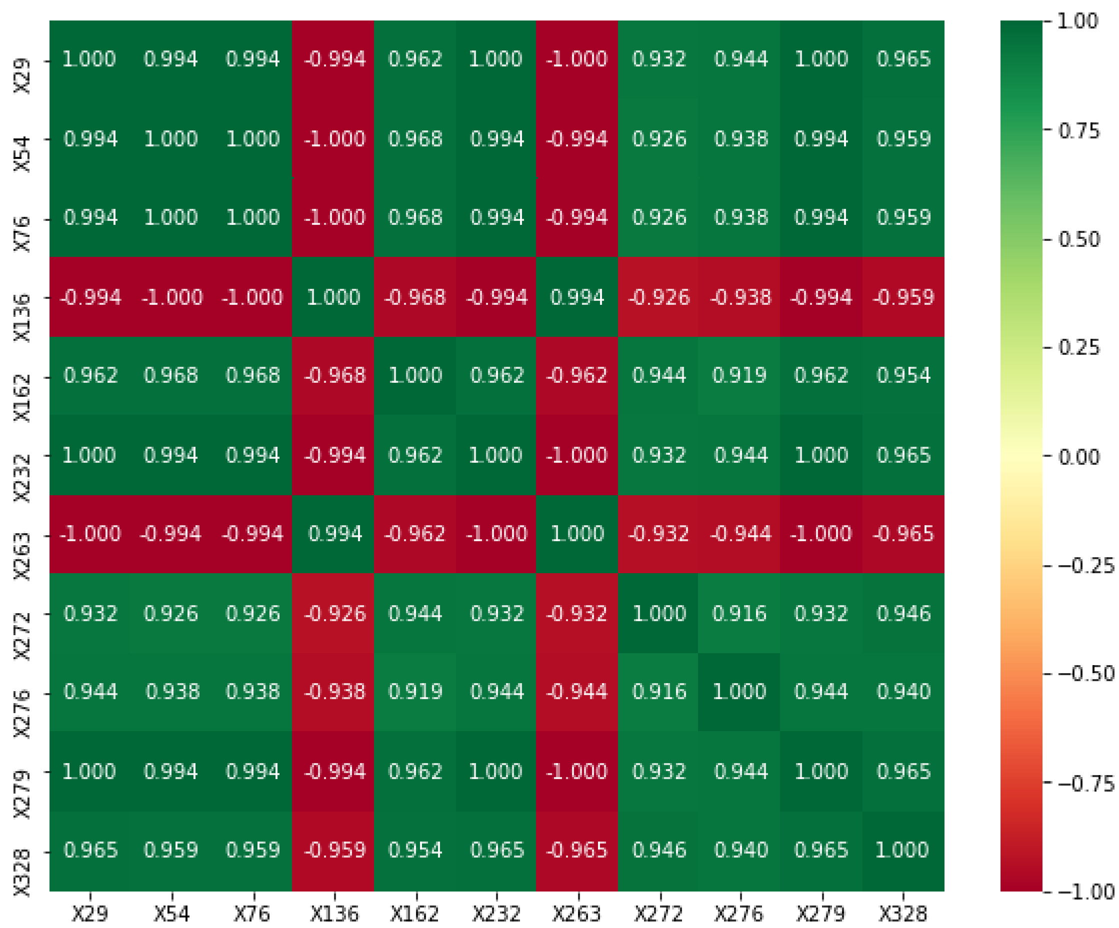
In [20]:
```python
correlation = data_train.corr()

# Select upper triangle of correlation matrix
upper = correlation.where(np.triu(np.ones(correlation.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.90
highVarianceColumns = [column for column in upper.columns if any(abs(upper[column]) > 0.9)]

print("Total correlated columns are: ", len(highVarianceColumns), "\n")
print(highVarianceColumns)
```

```
Total correlated columns are:  96

['X35', 'X37', 'X39', 'X54', 'X61', 'X66', 'X76', 'X84', 'X90', 'X94', 'X99', 'X101', 'X102', 'X111', 'X113', 'X119',
'X120', 'X122', 'X129', 'X130', 'X134', 'X136', 'X137', 'X140', 'X146', 'X147', 'X150', 'X157', 'X158', 'X162', 'X17
2', 'X179', 'X187', 'X194', 'X198', 'X199', 'X205', 'X213', 'X214', 'X215', 'X216', 'X217', 'X219', 'X222', 'X226',
'X227', 'X229', 'X232', 'X238', 'X239', 'X242', 'X243', 'X244', 'X245', 'X247', 'X248', 'X249', 'X250', 'X251', 'X25
3', 'X254', 'X262', 'X263', 'X264', 'X265', 'X266', 'X272', 'X276', 'X279', 'X296', 'X299', 'X302', 'X311', 'X314',
'X320', 'X324', 'X326', 'X328', 'X337', 'X346', 'X348', 'X352', 'X358', 'X360', 'X362', 'X363', 'X364', 'X365', 'X36
7', 'X368', 'X370', 'X371', 'X378', 'X379', 'X382', 'X385']
```

In [21]:
```python
sns.heatmap(correlation.loc[correlation['X35'].abs() > 0.9, correlation['X35'].abs() > 0.9],
            annot=True, cmap = plt.cm.RdYlGn, fmt='.3f');
```



In [22]:
```python
plt.figure(figsize=(10,8))
sns.heatmap(correlation.loc[correlation['X328'].abs() > 0.9, correlation['X328'].abs() > 0.9],
            annot=True, cmap = plt.cm.RdYlGn, fmt='.3f');
```



In [23]:
```
data_train.drop(columns=highVarianceColumns, inplace=True)
data_train.head()
```

Out[23]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X369	X372	X373	X374	X375	X376	X377	X380	X383	X384
0	0	130.81	k	v	at	a	d	u	j	o	...	0	0	0	0	0	0	1	0	0	0
1	6	88.53	k	t	av	e	d	y	l	o	...	0	0	0	0	1	0	0	0	0	0
2	7	76.26	az	w	n	c	d	x	j	x	...	0	0	0	0	0	0	0	0	0	0
3	9	80.62	az	t	n	f	d	x	l	e	...	0	1	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 × 270 columns

TASK 3: PERFORM LABEL ENCODING

In [24]:
```
labelencoder = LabelEncoder()
for column in objectfeatures:
    data_train[column] = labelencoder.fit_transform(data_train[column])
data_train
```

Out[24]:

	ID	y	X0	X1	X2	X3	X4	X5	X6	X8	...	X369	X372	X373	X374	X375	X376	X377	X380	X383	X384
0	0	130.81	32	23	17	0	3	24	9	14	...	0	0	0	0	0	0	1	0	0	0
1	6	88.53	32	21	19	4	3	28	11	14	...	0	0	0	0	1	0	0	0	0	0
2	7	76.26	20	24	34	2	3	27	9	23	...	0	0	0	0	0	0	0	0	0	0
3	9	80.62	20	21	34	5	3	27	11	4	...	0	1	0	0	0	0	0	0	0	0
4	13	78.02	20	23	34	5	3	12	3	13	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	8405	107.39	8	20	16	2	3	0	3	16	...	0	0	0	0	1	0	0	0	0	0
4205	8406	108.77	31	16	40	3	3	0	7	7	...	0	0	0	0	0	1	0	0	0	0
4206	8412	109.22	8	23	38	0	3	0	6	4	...	0	0	0	0	0	0	1	0	0	0
4207	8415	87.48	9	19	25	5	3	0	11	20	...	0	0	0	1	0	0	0	0	0	0
4208	8417	110.85	46	19	3	2	3	0	6	22	...	0	0	0	0	1	0	0	0	0	0

4209 rows × 270 columns

Drop column ID as it doesn't add much value for the predcition

In [25]:
```
data_train.drop(columns=['ID'], inplace=True)
data_train
```

Out[25]:

	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	...	X369	X372	X373	X374	X375	X376	X377	X380	X383	X384
0	130.81	32	23	17	0	3	24	9	14	0	...	0	0	0	0	0	0	1	0	0	0
1	88.53	32	21	19	4	3	28	11	14	0	...	0	0	0	0	1	0	0	0	0	0
2	76.26	20	24	34	2	3	27	9	23	0	...	0	0	0	0	0	0	0	0	0	0
3	80.62	20	21	34	5	3	27	11	4	0	...	0	1	0	0	0	0	0	0	0	0
4	78.02	20	23	34	5	3	12	3	13	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4204	107.39	8	20	16	2	3	0	3	16	0	...	0	0	0	0	1	0	0	0	0	0
4205	108.77	31	16	40	3	3	0	7	7	0	...	0	0	0	0	0	1	0	0	0	0
4206	109.22	8	23	38	0	3	0	6	4	0	...	0	0	0	0	0	0	1	0	0	0
4207	87.48	9	19	25	5	3	0	11	20	0	...	0	0	0	1	0	0	0	0	0	0
4208	110.85	46	19	3	2	3	0	6	22	0	...	0	0	0	0	1	0	0	0	0	0

4209 rows × 269 columns

Separating out feature and label

In [26]:
```
features = data_train.iloc[:,1:]
label = data_train.iloc[:,[0]]
```

In [27]:
```
print("features: ", features.shape)
print("label: ", label.shape)
```

```
features:  (4209, 268)
label:  (4209, 1)
```

In [28]:
```
features = features.values
label = label.values
```

In [29]:
```
print("Dimension of fatures: ", features.ndim, "\t", "Dimension of label: ", label.ndim)
```

```
Dimension of fatures:  2         Dimension of label:  2
```

SPLIT DATA INTO TRAIN AND TEST DATA

In [30]:
```
import warnings
warnings.filterwarnings('ignore')
for i in range(1,50):

    X_train,X_test,y_train,y_test = train_test_split(features, label, test_size=0.2, random_state = i)
    model = XGBRegressor(silent=True)
    model.fit(X_train,y_train)

    train_score = model.score(X_train,y_train)
    test_score = model.score(X_test,y_test)

    if (test_score > train_score) & (test_score>0.50):
        print("Test: {} , Train: {} , RS : {}".format(test_score,train_score,i))
```

```
Test: 0.6459891028573748 , Train: 0.5971145731490303 , RS : 6
Test: 0.6172083823582479 , Train: 0.5994172681434732 , RS : 11
Test: 0.6421808121279696 , Train: 0.593563813696471 , RS : 15
Test: 0.6228808620516164 , Train: 0.6039224828663703 , RS : 17
Test: 0.610518103296064 , Train: 0.6003978844381486 , RS : 18
Test: 0.6033010821397202 , Train: 0.6021664916658018 , RS : 20
Test: 0.6272503200259458 , Train: 0.597747084460431 , RS : 27
Test: 0.6449199314332152 , Train: 0.595721721824291 , RS : 28
Test: 0.619734399307938 , Train: 0.6014066454301915 , RS : 35
Test: 0.6355018264062284 , Train: 0.6010171619916194 , RS : 47
Test: 0.6354498922841102 , Train: 0.5959495435759852 , RS : 48
```

Depending on the above scores, we would select RS = 6

In [31]:
```
X_train,X_test,y_train,y_test = train_test_split(features, label, test_size=0.2, random_state = 6)
```

Standardizing the data

In [32]:
```
#Standardization(StandardScaler)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

TASK 4: Perform dimensionality reduction

In [33]:
```
#Initialize the PCA and detect the number of principal components
principalComponents = PCA(n_components=268) #Here n_components = n_features
principalComponents.fit(X_train,y_train)
```

Out[33]:
```
PCA(copy=True, iterated_power='auto', n_components=268, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [34]:
```
principalComponents.explained_variance_ratio_
```

Out[34]:
```
array([4.89615151e-02, 4.57868018e-02, 3.56616412e-02, 3.12036887e-02,
       2.78255002e-02, 2.61717302e-02, 2.07595728e-02, 1.78622864e-02,
       1.68429854e-02, 1.63170071e-02, 1.61033332e-02, 1.52165984e-02,
       1.46317052e-02, 1.35577059e-02, 1.26429582e-02, 1.24525268e-02,
       1.19293205e-02, 1.09978005e-02, 1.06951154e-02, 1.01577821e-02,
       9.92984483e-03, 9.82025953e-03, 9.57458834e-03, 9.50041098e-03,
       8.77059338e-03, 8.75700357e-03, 8.43127755e-03, 8.29569537e-03,
       7.96725238e-03, 7.79848046e-03, 7.64226708e-03, 7.47803986e-03,
       7.18401858e-03, 7.12563310e-03, 6.88742220e-03, 6.63202469e-03,
       6.49068853e-03, 6.45223533e-03, 6.39249242e-03, 6.20751736e-03,
       6.02435420e-03, 5.95435241e-03, 5.82587785e-03, 5.80441367e-03,
       5.70610459e-03, 5.62592989e-03, 5.56433557e-03, 5.46481271e-03,
       5.30888979e-03, 5.25542208e-03, 5.19037909e-03, 5.12424003e-03,
       5.03653909e-03, 4.91031866e-03, 4.86332891e-03, 4.75632537e-03,
       4.73384302e-03, 4.69255194e-03, 4.66738091e-03, 4.65032546e-03,
       4.54335766e-03, 4.51749107e-03, 4.51254541e-03, 4.42766711e-03,
       4.37940293e-03, 4.32979914e-03, 4.29464747e-03, 4.27761275e-03,
       4.19743032e-03, 4.18826034e-03, 4.15667926e-03, 4.14464563e-03,
       4.06167141e-03, 4.04308940e-03, 3.99737599e-03, 3.96694941e-03,
       3.94763545e-03, 3.92529389e-03, 3.90972746e-03, 3.89704965e-03,
       3.87947729e-03, 3.83074163e-03, 3.81102450e-03, 3.79407534e-03,
       3.75606917e-03, 3.72311340e-03, 3.71065955e-03, 3.69997502e-03,
       3.64111731e-03, 3.62990019e-03, 3.59836594e-03, 3.57107998e-03,
       3.53843929e-03, 3.48691907e-03, 3.45853782e-03, 3.44658958e-03,
       3.38668303e-03, 3.34157822e-03, 3.32013008e-03, 3.27410525e-03,
       3.24934346e-03, 3.22285991e-03, 3.18847566e-03, 3.13697932e-03,
       3.12060684e-03, 3.07749405e-03, 3.00309444e-03, 2.98004520e-03,
       2.93330895e-03, 2.90621993e-03, 2.88102127e-03, 2.84976330e-03,
       2.81597843e-03, 2.77023391e-03, 2.71471534e-03, 2.69515136e-03,
       2.67261527e-03, 2.65238329e-03, 2.57412373e-03, 2.55527292e-03,
       2.54709680e-03, 2.50950321e-03, 2.49370654e-03, 2.41459783e-03,
       2.38591384e-03, 2.36886715e-03, 2.32512115e-03, 2.31412288e-03,
       2.21893365e-03, 2.17725224e-03, 2.16057063e-03, 2.11732071e-03,
       2.09535528e-03, 2.04989905e-03, 2.02506324e-03, 1.99643221e-03,
       1.94162547e-03, 1.92067198e-03, 1.87552653e-03, 1.86941595e-03,
       1.78457801e-03, 1.76180772e-03, 1.73897082e-03, 1.68199644e-03,
       1.63949272e-03, 1.61052403e-03, 1.57721826e-03, 1.54483083e-03,
       1.51853193e-03, 1.46747328e-03, 1.39777159e-03, 1.35584038e-03,
       1.35041778e-03, 1.32572339e-03, 1.27282234e-03, 1.25084100e-03,
       1.20573395e-03, 1.19600114e-03, 1.19039108e-03, 1.14136936e-03,
       1.13426530e-03, 1.11160115e-03, 1.08977049e-03, 1.06643665e-03,
       1.02917895e-03, 1.00399749e-03, 9.95444207e-04, 9.59116758e-04,
       9.49676315e-04, 9.06830691e-04, 8.75520378e-04, 8.47926539e-04,
       8.43542584e-04, 8.11498327e-04, 7.99292675e-04, 7.76239936e-04,
       7.63149611e-04, 7.48563399e-04, 7.26767426e-04, 7.16727871e-04,
       6.93735778e-04, 6.69331226e-04, 6.43137216e-04, 6.21076404e-04,
       6.16747298e-04, 6.01260002e-04, 5.76851791e-04, 5.69844205e-04,
       5.55729140e-04, 5.40843762e-04, 5.35302205e-04, 5.27871607e-04,
       5.08373677e-04, 5.02180659e-04, 4.87612526e-04, 4.47416201e-04,
       4.39044619e-04, 4.35389967e-04, 4.30233115e-04, 4.13789911e-04,
       3.96910699e-04, 3.89384816e-04, 3.77058120e-04, 3.56671192e-04,
       3.32485553e-04, 3.29929004e-04, 3.15109137e-04, 2.96485121e-04,
       2.80310894e-04, 2.76639426e-04, 2.43360082e-04, 2.28365172e-04,
       2.16255213e-04, 2.01642863e-04, 1.94590458e-04, 1.90632796e-04,
       1.69669110e-04, 1.58097741e-04, 1.49771648e-04, 1.43822594e-04,
       1.23386133e-04, 1.16398440e-04, 1.01250206e-04, 9.22115759e-05,
       8.07573583e-05, 7.41311848e-05, 6.53851153e-05, 6.29631356e-05,
       5.63542946e-05, 4.95476823e-05, 4.68702575e-05, 4.38457637e-05,
       4.04999168e-05, 3.41132928e-05, 2.46288924e-05, 2.39218800e-05,
       1.90458448e-05, 1.13600941e-05, 3.84099827e-06, 1.62057809e-32,
       9.38780923e-33, 4.63531116e-33, 2.98025355e-33, 2.90388127e-33,
       2.61194170e-33, 2.28663578e-33, 1.99887396e-33, 1.52042114e-33,
       1.28002888e-33, 1.09630007e-33, 9.17582271e-34, 7.28886062e-34,
       7.24067871e-34, 7.19338301e-34, 5.92900690e-34, 5.60969265e-34,
       4.35505280e-34, 3.88961442e-34, 3.61588180e-34, 2.86586972e-34,
       2.68060027e-34, 2.41204078e-34, 2.22809094e-34, 2.22809094e-34,
       1.54387428e-34, 8.37305354e-35, 4.42795545e-35, 4.09786401e-36])
```

In [35]:
```
#Step3: Apply PCA with correct n_components
principalComponentsFinal = PCA(n_components=20) #Here n_components = n_features
principalComponentsFinal.fit(X_train,y_train)
principalComponentsFinal.explained_variance_ratio_
X_train = principalComponentsFinal.transform(X_train)
X_test = principalComponentsFinal.transform(X_test)
```

In [36]:
```
finalModel = XGBRegressor(silent=True)
finalModel.fit(X_train,y_train)
```

Out[36]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=True, subsample=1, verbosity=1)
```

In [37]:
```
print("Train Score: ", finalModel.score(X_train,y_train))
print("Test Score: ", finalModel.score(X_test,y_test))
```

```
Train Score:  0.5720484987438172
Test Score:  0.5216426125483891
```

In [38]:
```
predictionValue = finalModel.predict(X_test)
list(predictionValue)[:8]
```

Out[38]:
```
[92.75285,
 96.49849,
 114.23427,
 99.22041,
 95.33731,
 103.54232,
 92.9209,
 99.147835]
```

In [39]:
```
# Converting Dimensions of y_test from 2 to 1 dimension
y_test = y_test.ravel()
```

In [40]:
```
predictedFrame = pd.DataFrame(predictionValue, columns=['Predicted'])
targetFrame = pd.DataFrame(y_test, columns=['Target'])
Observed_Predicted_Table =pd.concat([targetFrame.reset_index(drop=True),predictedFrame],axis=1)
Observed_Predicted_Table.head()
```

Out[40]:

	Target	Predicted
0	89.66	92.752853
1	88.62	96.498489
2	112.32	114.234268
3	106.88	99.220413
4	96.93	95.337311

In [41]:
```
Observed_Predicted_Table_Split = Observed_Predicted_Table.head(20)
Observed_Predicted_Table_Split.plot(kind='bar',figsize=(15,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.xlabel("Records",  fontdict = {'fontsize' : 20})
plt.ylabel("Time Spent",  fontdict = {'fontsize' : 20})
plt.title("Time that cars spend on the test bench",  fontdict = {'fontsize' : 20})
plt.show()
```