```python
#Using sklearn.datasets.load_diabetes apply Variance method for
removing the constant column also after applying
#the Variance method apply multi linear regression on that data
/aleternate dataset given by ajay sharma and told to apply svm on it
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

from sklearn.feature_selection import VarianceThreshold

df=pd.read_csv("E:\diabetesnew.csv")
var_thres=VarianceThreshold(threshold=0.2)
var_thres.fit(df)
var_thres.get_support()
df.columns[var_thres.get_support() == True]
columns_having_var_more_than_50 = df.columns[var_thres.get_support()
== True]

columns_having_var_less_than_50 = df.columns[var_thres.get_support()
== False]
df.drop(columns_having_var_less_than_50,inplace = True,axis= 1)
df.isnull().values.any()
```

```
False
```

```python
from sklearn import svm
from sklearn.model_selection import train_test_split

x = df.iloc[:, :-2]
y = df.iloc[:, -1]
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state
= 0, test_size = 0.2)

clf = svm.SVC(kernel='rbf')
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)

from sklearn.metrics import accuracy_score
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7987012987012987
```

```python
#Using sklearn.datasets.load_wine Apply Correlation and make a heat
map using seaborn and remove the highly
#correlated columns if exist and the apply SVM and get the best
accuracy by changing the Hyperparameters
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
%matplotlib inline

from sklearn.datasets import load_wine
wine=load_wine()
type(wine)
print(wine.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names'])
```

```
colums_name=wine.feature_names
```

```
df = pd.DataFrame(wine['data'],columns=wine['feature_names'])
```

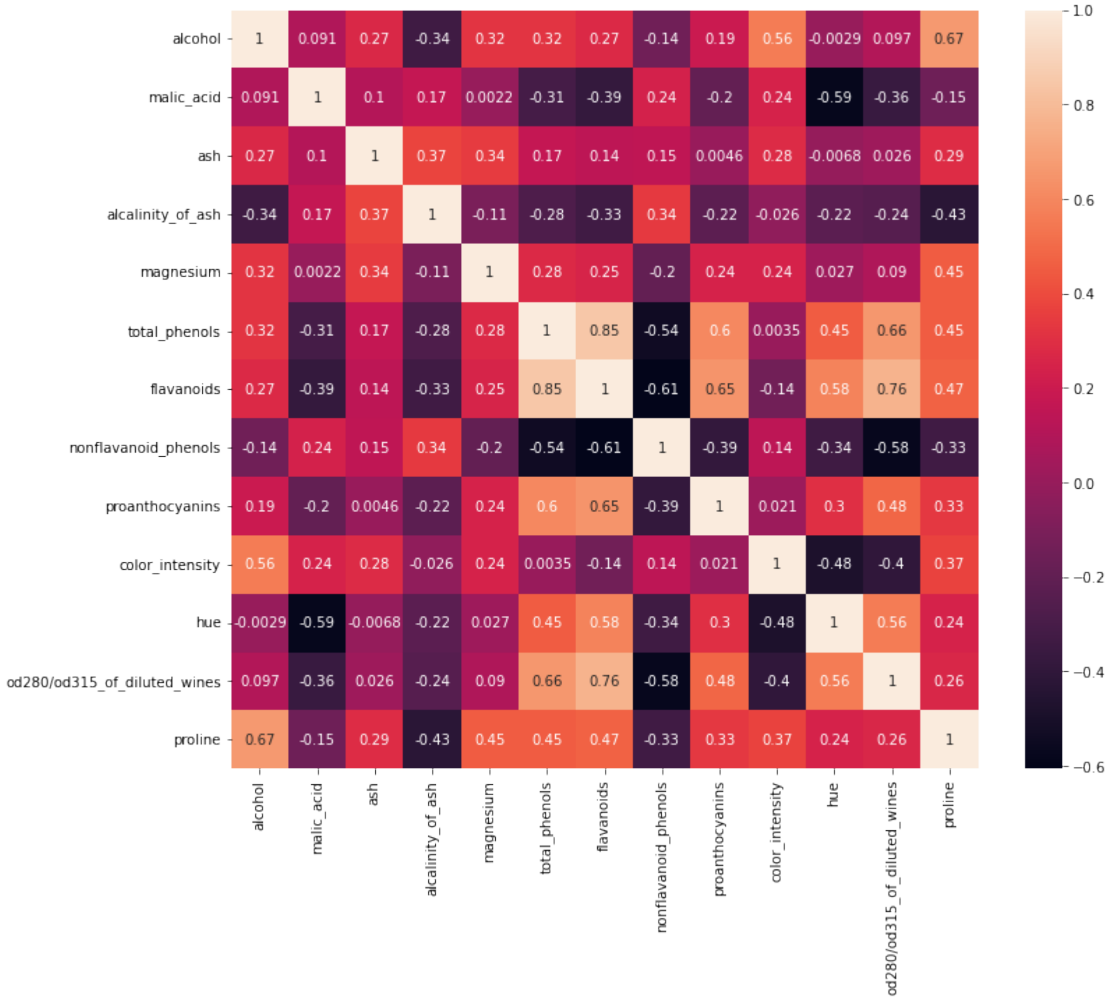
```
df["medv"]=wine.target
```

```
x=df.drop("medv",axis=1)
y=df["medv"]

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,rando
m_state=40)
x_train.shape,x_test.shape
```

```
((124, 13), (54, 13))
```

```
import seaborn as sns
plt.figure(figsize=(12,10))
cor=x_train.corr()
sns.heatmap(cor,annot=True)
plt.show()
```



```
def correlation(dataset, threshold):#X_train,
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix)): #traverse through the rows
        for j in range(i): #traverse through column
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are
interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name
of column
                col_corr.add(colname)
    return col_corr

corr_features = correlation(x_train, 0.3)
len((corr_features))

10

df.head()

   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols  \
```

```
0    14.23        1.71  2.43               15.6      127.0
2.80
1    13.20        1.78  2.14               11.2      100.0
2.65
2    13.16        2.36  2.67               18.6      101.0
2.80
3    14.37        1.95  2.50               16.8      113.0
3.85
4    13.24        2.59  2.87               21.0      118.0
2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity
hue  \
0        3.06                  0.28             2.29             5.64
1.04
1        2.76                  0.26             1.28             4.38
1.05
2        3.24                  0.30             2.81             5.68
1.03
3        3.49                  0.24             2.18             7.80
0.86
4        2.69                  0.39             1.82             4.32
1.04

   od280/od315_of_diluted_wines  proline  medv
0                          3.92   1065.0     0
1                          3.40   1050.0     0
2                          3.17   1185.0     0
3                          3.45   1480.0     0
4                          2.93    735.0     0
```

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

feature_df=df[["alcohol","malic_acid","ash"]]
x=np.asarray(feature_df)
x[0:5]
```

```
array([[14.23,  1.71,  2.43],
       [13.2 ,  1.78,  2.14],
       [13.16,  2.36,  2.67],
       [14.37,  1.95,  2.5 ],
       [13.24,  2.59,  2.87]])
```

```
wine['medv']=df['medv'].astype('int')
y=np.asarray(wine['medv'])
print("y values:",y[0:5])
```

```
y values: [0 0 0 0 0]
```

```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.2,random
_state=50)
print("Train set:",x_train.shape,y_train.shape)
print("Test set:",x_test.shape,y_test.shape)
```

```
Train set: (142, 3) (142,)
Test set: (36, 3) (36,)
```

```python
from sklearn import svm
clf=svm.SVC(kernel='poly')
clf.fit(x_train,y_train)

yhat=clf.predict(x_test)
yhat[0:5]
```

```
array([1, 1, 1, 2, 2])
```

```python
from sklearn.metrics import f1_score
f1_score(y_test,yhat,average='weighted')
```

```
0.75
```

```python
clf2=svm.SVC(kernel='rbf')
clf2.fit(x_train,y_train)
yhat2=clf2.predict(x_test)
print("avg f1-score:%.4f"% f1_score(y_test,yhat2,average='weighted'))
```

```
avg f1-score:0.7467
```

```python
clf2=svm.SVC(kernel='linear')
clf2.fit(x_train,y_train)
yhat2=clf2.predict(x_test)
print("avg f1-score:%.4f"% f1_score(y_test,yhat2,average='weighted'))
```

```
avg f1-score:0.7235
```

```python
#Using sklearn.datasets.load_diabetes apply Mutual info Classification
and check which are the best columns
#according to the target column.
#Then Apply decision tree on that data and try to get best accuracy by
changing the hyperparameters
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
%matplotlib inline

from sklearn.datasets import load_diabetes
data =load_diabetes ()
type(data)
```

```
sklearn.utils.Bunch
```

```
data.keys
```

```
<function Bunch.keys>
```

```
data.feature_names
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
columns_name = data.feature_names
```

```
columns_name
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
df = pd.DataFrame(data.data, columns = columns_name)
```

```
df.head()
```

```
        age       sex       bmi        bp        s1        s2
s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -
0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163
0.074412
2  0.085299  0.050680  0.044451 -0.005671 -0.045599 -0.034194 -
0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -
0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596
0.008142

         s4        s5        s6
0 -0.002592  0.019908 -0.017646
1 -0.039493 -0.068330 -0.092204
2 -0.002592  0.002864 -0.025930
3  0.034309  0.022692 -0.009362
4 -0.002592 -0.031991 -0.046641
```

```
df["daibetes"] = data.target #dependent colummn CONTENT
```

```
X = df.drop("daibetes",axis=1)   #independent variable : all column
except Target Dv colun
y = df["daibetes"] #dependent variables only target column will be in
Y
```

```
X_train,X_test,y_train,y_test=train_test_split(X, #INDEPENDENDENT
VARIABLE
    y, # as DEPENDENT VARIABLE
    test_size=0.3, #70% TRAINING DS AND 30% TEST DATA
    random_state=0)
```

```
X_train.shape
```

```
(309, 10)
```

```
from sklearn.feature_selection import mutual_info_regression
# determine the mutual information
mutual_info = mutual_info_regression(X_train, y_train)
mutual_info #impactful variable will get high value and less
impactfull will get low values
```

```
array([0.01410304, 0.03507104, 0.19056747, 0.10522933, 0.08487774,
       0.00482829, 0.05602829, 0.12500372, 0.15324809, 0.1194692 ])
```

```
mutual_info = pd.Series(mutual_info)
```

```
mutual_info.index = X_train.columns
```

```
mutual_info.sort_values(ascending=False)
```

```
bmi    0.190567
s5     0.153248
s4     0.125004
s6     0.119469
bp     0.105229
s1     0.084878
s3     0.056028
sex    0.035071
age    0.014103
s2     0.004828
dtype: float64
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression

sel_best_cols = SelectKBest(mutual_info_regression, k=5)
```

```
sel_best_cols.fit(X_train, y_train)
```

```
SelectKBest(k=5,
            score_func=<function mutual_info_regression at
0x00000247DDAC9D30>)
```

```
X_train.columns[sel_best_cols.get_support()==True]
```

```
Index(['bmi', 'bp', 's4', 's5', 's6'], dtype='object')
```

```
type(X_train)
```

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

```
X_train = X_train[['sex', 'bmi', 's3', 's4', 's5']]
```

```
X_test = X_test[['sex', 'bmi', 's3', 's4', 's5']]
```

```
from sklearn.tree import DecisionTreeRegressor
tree=DecisionTreeRegressor()#(criterion="entropy",max_depth=4)
tree
```

```
DecisionTreeRegressor()
```

```
tree.fit(X_train,y_train)
```

```
DecisionTreeRegressor()
```

```
y_pred=tree.predict(X_test)
#y_pred=tree.predict(X_test)
```

```
print(y_pred[0:5])
print(y_test[0:5])
```

```
[261. 310. 225. 214. 191.]
362    321.0
249    215.0
271    127.0
435     64.0
400    175.0
Name: daibetes, dtype: float64
```

```
from sklearn import metrics
print("DecisionTrees's Accuracy: ",metrics.r2_score(y_pred,y_test))
```

```
DecisionTrees's Accuracy:  -0.149140648377031
```

```
from sklearn.metrics import mean_squared_error


rmse = (np.sqrt(mean_squared_error(y_pred,y_test)))
rmse
```

```
83.59538246772253
```

```
#Using sklearn.datasets.load_boston apply Mutual info Regression and
check which are the best columns according
#to the target column.
#Then Apply MultiLinear Regression on that data and try to get best
accuracy by changing the hyperparameters
from sklearn.datasets import load_boston
data =load_boston ()
type(data )
```

```
C:\Users\Renuka\anaconda3\lib\site-packages\sklearn\utils\
deprecation.py:87: FutureWarning: Function load_boston is deprecated;
`load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can
refer to
```

```
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use
of this
    dataset unless the purpose of the code is to study and educate
about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22,
header=None)
        data = np.hstack([raw_df.values[::2, :],
raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames
housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

```
sklearn.utils.Bunch
```

```
data.feature_names
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

```
import pandas as pd
columns_name = data.feature_names
```

```
df = pd.DataFrame(data.data, columns = columns_name)
df.head()
```

```
      CRIM    ZN  INDUS  CHAS    NOX     RM   AGE     DIS  RAD    TAX
\
0  0.00632  18.0   2.31   0.0  0.538  6.575  65.2  4.0900  1.0  296.0

1  0.02731   0.0   7.07   0.0  0.469  6.421  78.9  4.9671  2.0  242.0

2  0.02729   0.0   7.07   0.0  0.469  7.185  61.1  4.9671  2.0  242.0

3  0.03237   0.0   2.18   0.0  0.458  6.998  45.8  6.0622  3.0  222.0

4  0.06905   0.0   2.18   0.0  0.458  7.147  54.2  6.0622  3.0  222.0


   PTRATIO       B  LSTAT
0     15.3  396.90   4.98
1     17.8  396.90   9.14
2     17.8  392.83   4.03
3     18.7  394.63   2.94
4     18.7  396.90   5.33
```

```
df["dv"] = data.target #dependent colummn CONTENT

X = df.drop("dv",axis=1)   #independent variable : all column except
Target Dv colun
y = df["dv"] #dependent variables only target column will be in Y

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X, #INDEPENDENDENT
VARIABLE
    y, # as DEPENDENT VARIABLE
    test_size=0.3, #70% TRAINING DS AND 30% TEST DATA
    random_state=0)

X_train.shape
```

```
(354, 13)
```

```
y_train.shape
```

```
(354,)
```

```
from sklearn.feature_selection import mutual_info_regression
# determine the mutual information
mutual_info = mutual_info_regression(X_train, y_train)
mutual_info #impactful variable will get high value and less
impactfull will get low values
```

```
array([0.32559106, 0.18773109, 0.53440947, 0.03137659, 0.43582696,
       0.59739155, 0.33866851, 0.30723383, 0.22056715, 0.36521941,
       0.51343429, 0.16564181, 0.65114809])
```

```
X_train.shape
```

```
(354, 13)
```

```
mutual_info = pd.Series(mutual_info)
mutual_info.index = X_train.columns
mutual_info.sort_values(ascending=False)
```

```
LSTAT      0.651148
RM         0.597392
INDUS      0.534409
PTRATIO    0.513434
NOX        0.435827
TAX        0.365219
AGE        0.338669
CRIM       0.325591
DIS        0.307234
RAD        0.220567
ZN         0.187731
B          0.165642
CHAS       0.031377
dtype: float64
```

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_regression
```

```
sel_best_cols = SelectKBest(mutual_info_regression, k=5)
```

```
sel_best_cols.fit(X_train, y_train)
```

```
SelectKBest(k=5,
            score_func=<function mutual_info_regression at
0x00000247DDAC9D30>)
```

```
X_train.columns[sel_best_cols.get_support()==True]
```

```
Index(['INDUS', 'NOX', 'RM', 'PTRATIO', 'LSTAT'], dtype='object')
```

```
X_train = X_train[['INDUS', 'NOX', 'RM', 'PTRATIO', 'LSTAT']]
```

```
X_train.shape
```

```
(354, 5)
```

```
X_test = X_test[['INDUS', 'NOX', 'RM', 'PTRATIO', 'LSTAT']]
```

```
y_train.shape
```

```
(354,)
```

```
X_train.shape
```

```
(354, 5)
```

```
X_train = X_train.values
```

```
import numpy as np
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X_train,y_train)#training func question + answers
# The coefficients
print ('Intercept: ',regr.intercept_)
print ('Coefficient : ',regr.coef_)
```

```
Intercept:  24.661233547043977
Coefficient :  [ 0.0533226  -6.52551265  4.57081317 -1.15326995 -
0.51504091]
```

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

```
y_hat = regr.predict(X_train)
#rmse = (np.sqrt(mean_squared_error(y_train, y_hat)))


r2 = r2_score(y_train, y_hat)
```

```
r2
```

```
0.7123290285129122
```

```
y_test_predict = regr.predict(X_test)
#rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))
r2 = r2_score(y_test,y_test_predict)
```

```
C:\Users\Renuka\anaconda3\lib\site-packages\sklearn\base.py:443:
UserWarning: X has feature names, but LinearRegression was fitted
without feature names
  warnings.warn(
```

```
r2
```

```
0.5948238037827576
```