Exam : 1

Subject : INFX598

ULID : C00535934

Name : Rajesh Awal

Description of Scientific Paper

My scientific paper's title is "Recognizing and classifying MNIST data set from 0 to 9 and selection of optimized model". It implements Classification Machine learning approaches. The objective is to study of the basic theoretical and mathematical concepts of classification machine learning (ML) models. The motivation of the research paper is classify MNIST dataset with higher accuracy. Along with that, the secondary motivation is to get familiar with different machine learning models, feature engineering, performance metrics, and loss functions. It reads the handwritten image data and studies the shapes of the handwritten numbers in image to predict number from 0 10 9.

Mnist Dataset

MNIST data set is a large collection of handwritten digits that is commonly used for training various image processing systems. The data is downloaded from http://yann.lecun.com/exdb/mnist/. The dataset is very good for learning techniques and pattern recognition methods on real-world data.

Wine Classification Dataset

The exam project is the study Machine learning classification which is similar to experimental paper. Wine Classification dataset is used to classify the quality based on numeric value 0, 1 and 2 which can be referred as low, medium, and high quality. MNIST training models uses handwritten image to classify. However, Wine Classification uses numerical values of different quality determining attributes to predict the quality of wine.

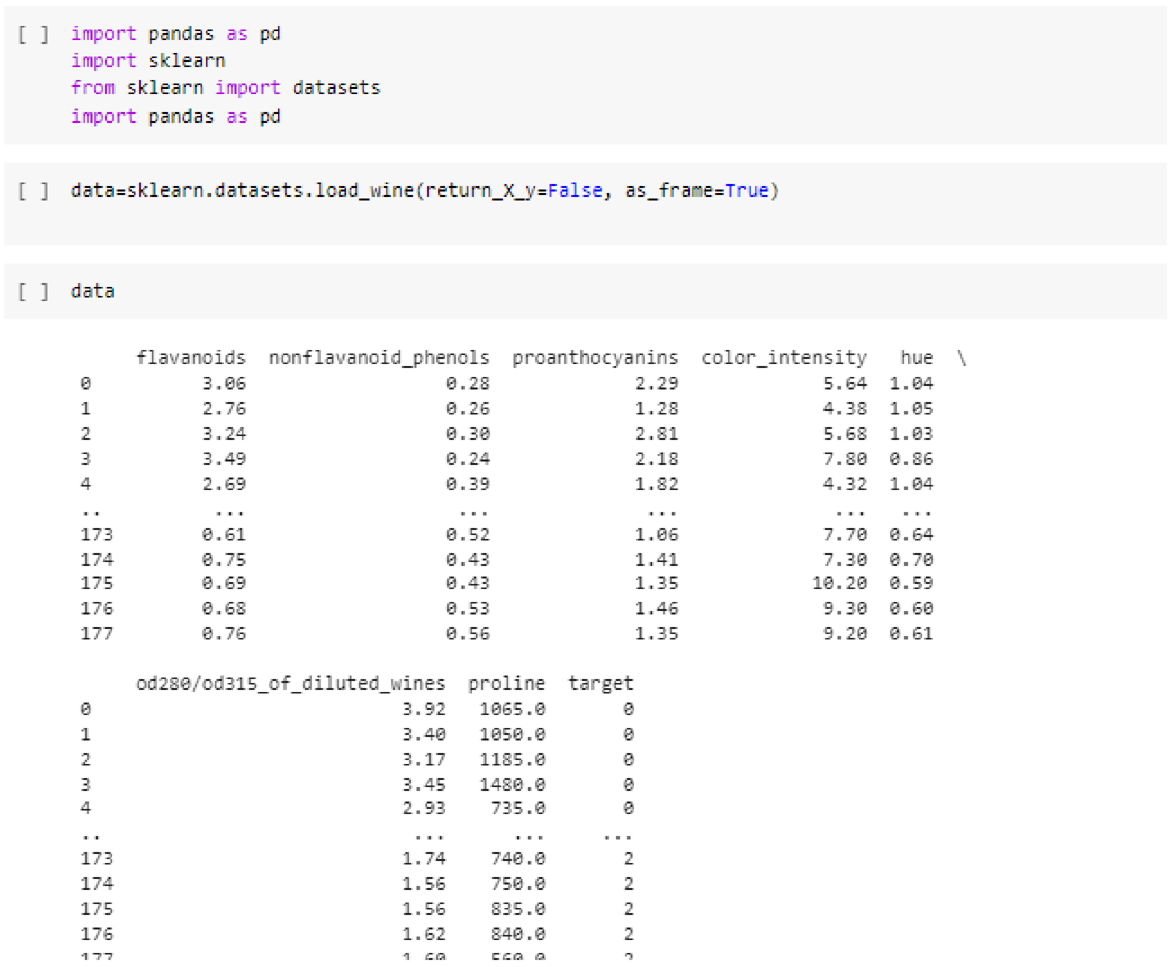
Machine learning steps for Wine Classification

a. Frame the problem and look at the big picture.

   The project involves study of chemical substance concentration which is responsible for determining the quality of wine. This data has concentration metrics such as the malic acid, ash, ash alkalinity and so on for wine. Price of the product depends on quality. It helps the production team to analyze the class of wine. It gives the idea to control combination of different chemical composition for producing desired class of wine. Further, it can be quality management tool for wine producing industries.

b. Get the data

   SK learn library data set is used for training and testing purposes. Using this command "sklearn.datasets.**load_wine**(*, *return_X_y=False*, *as_frame=False*)", it load and return the wine dataset. The Features data and target data are combined to form pandas data frame which will be used for visualizing, cleaning, splitting, training and testing.



```
[ ] import pandas as pd
    import sklearn
    from sklearn import datasets
    import pandas as pd
```

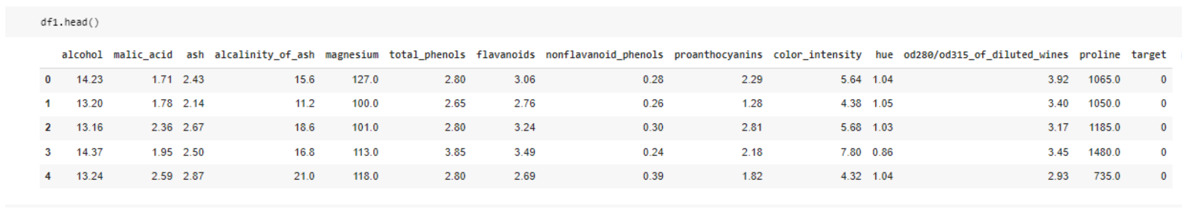
```
[ ] data=sklearn.datasets.load_wine(return_X_y=False, as_frame=True)
```

```
[ ] data
```

```
     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
..          ...                   ...              ...              ...   ...
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     od280/od315_of_diluted_wines  proline  target
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
..                            ...      ...     ...
173                          1.74    740.0       2
174                          1.56    750.0       2
175                          1.56    835.0       2
176                          1.62    840.0       2
177                          1.60    560.0       2
```

c. Explore the data to gain insight

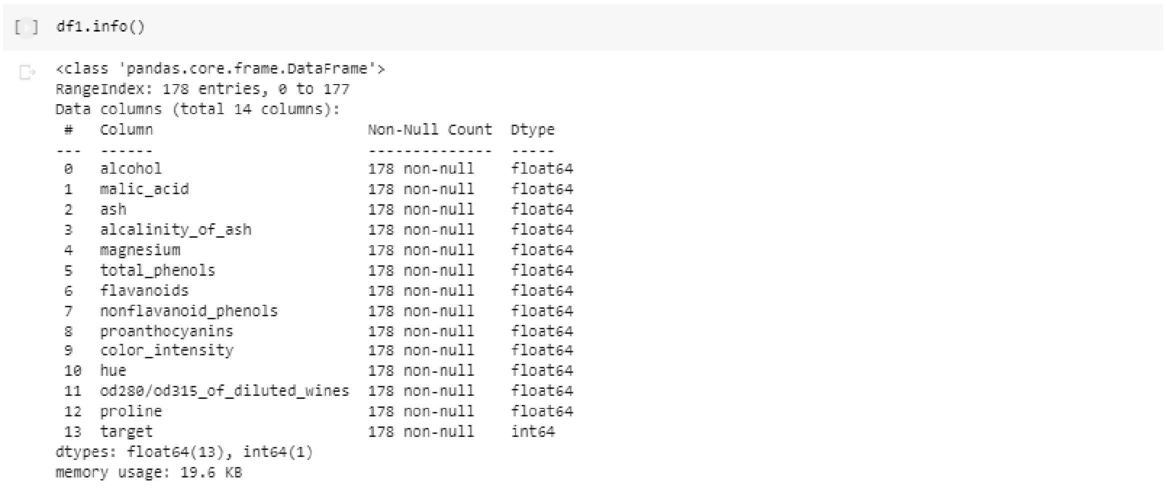
The data frame has 13 features responsible for determining class of data. The feature represents concentrations of different chemicals in different samples. Let's take a look at the top five rows using the head() method.



```
df1.head()
```

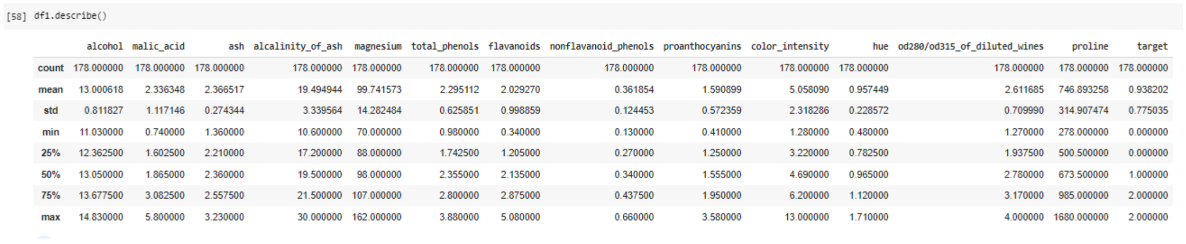
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

The target value 0, 1 and 2 represents class of Wine as Class A, B and C. Using Info() function we can see shape and type of each feature variable.



```
[ ]  df1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
```

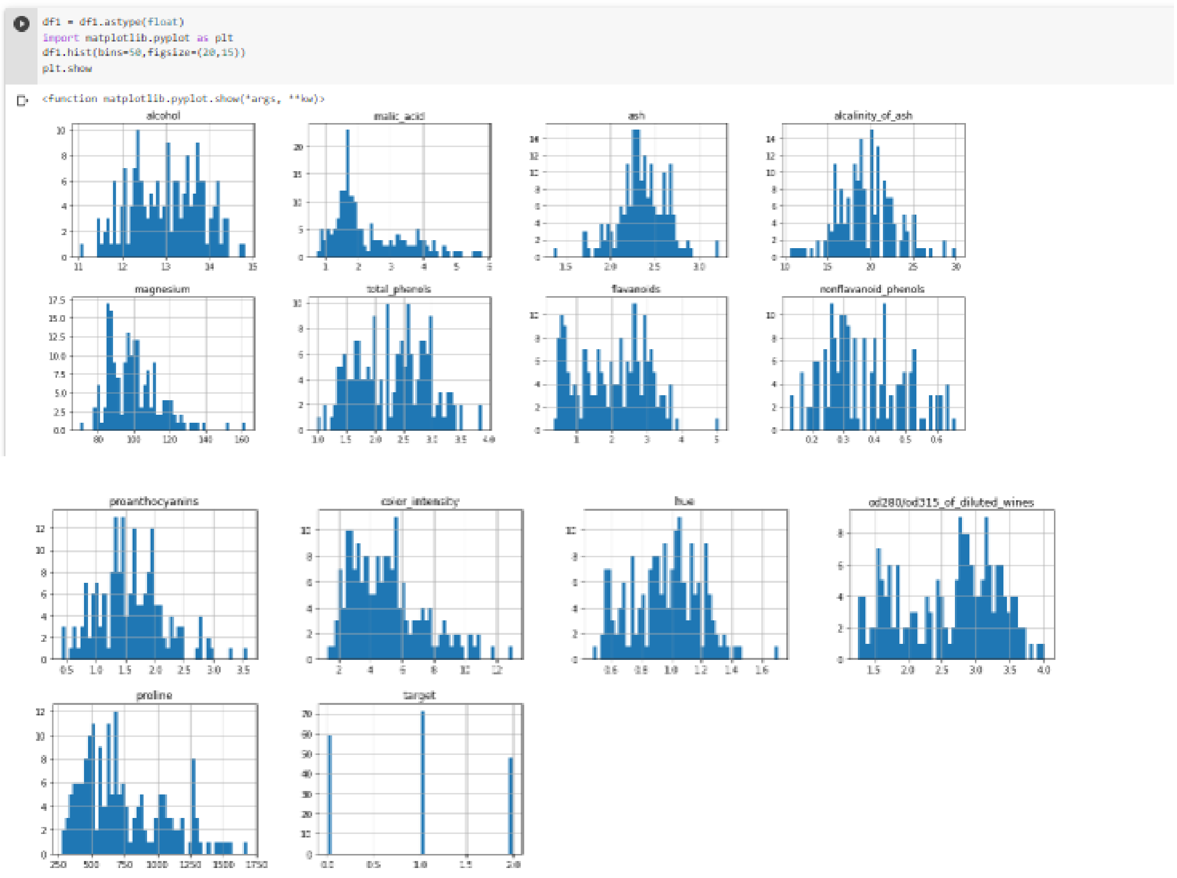
For further analyzing the information, statistical analysis give strong intuition for further processing of data.



```
[58] df1.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	0.938202
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	0.775035
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	0.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	0.000000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	1.000000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	2.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	2.000000

Distribution of data gives an intuition to recognize weight of each variable contributing for prediction of target. Let's visualize each feature with histogram.



```
df1 = df1.astype(float)
import matplotlib.pyplot as plt
df1.hist(bins=50,figsize=(20,15))
plt.show
```

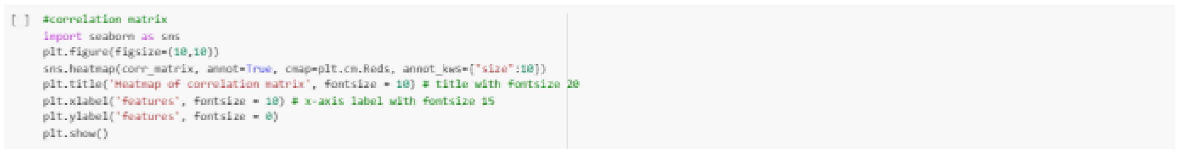
```
<function matplotlib.pyplot.show(*args, **kw)>
```

Each graph visualize it`s data distribution along it’s range. This is very helpful to realize probability density of data.

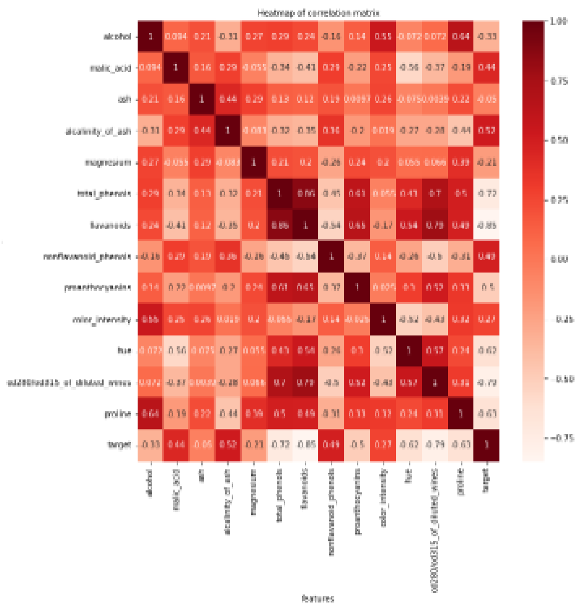
d. Prepare the data

Cleaning

We now have understand the type and nature of data. To predict target variable lets work on a feature data cleaning. To do so lets find out correlation matrix.

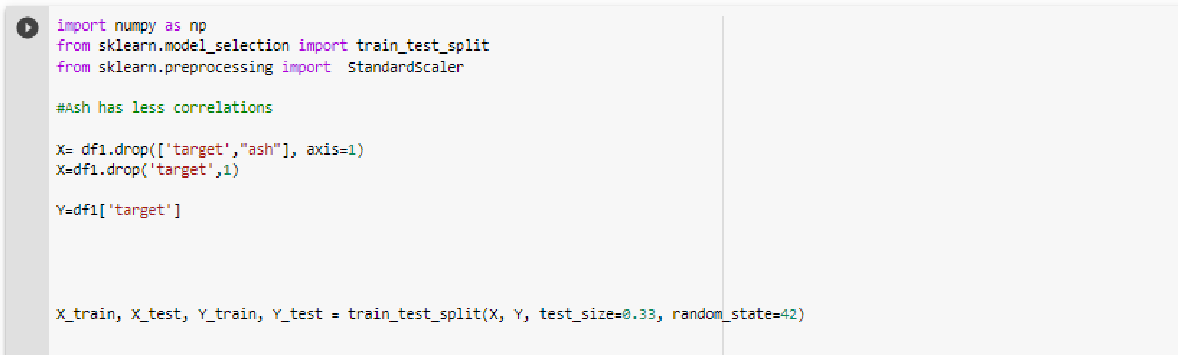


```
[ ] #correlation matrix
    import seaborn as sns
    plt.figure(figsize=(10,10))
    sns.heatmap(corr_matrix, annot=True, cmap=plt.cm.Reds, annot_kws={"size":10})
    plt.title('Heatmap of correlation matrix', fontsize = 10) # title with fontsize 20
    plt.xlabel('features', fontsize = 10) # x-axis label with fontsize 15
    plt.ylabel('features', fontsize = 0)
    plt.show()
```



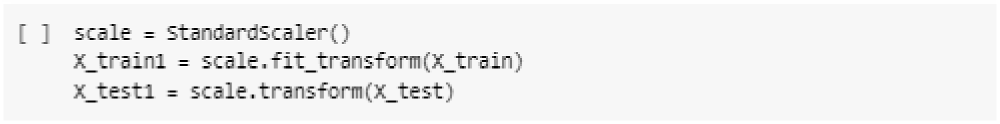
e. As we can see the variable ash has very low correlation with other variables. It is least dependent and possess least significance for predicting class of wine. Since we have 13 features to determine target with only 3 class, we can drop ash feature.

Now lets split data set into train and test. Selecting randomly will give better results.



```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import  StandardScaler

#Ash has less correlations

X= df1.drop(['target',"ash"], axis=1)
X=df1.drop('target',1)

Y=df1['target']



X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=42)
```

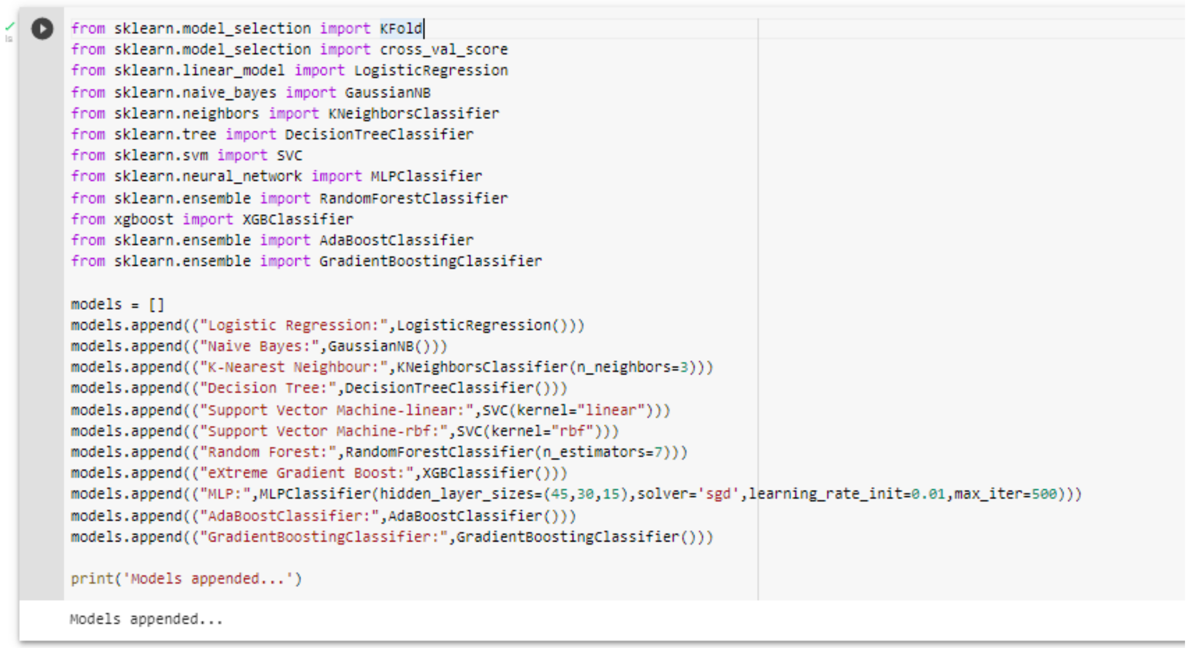
f. Different columns have different range of data. Column with bigger data might more significance than small one. To solve this, let's use standard scaler. It scales data relative to its standard deviation. Following codes is used to scale train and test dataset.



```
[ ]  scale = StandardScaler()
     X_train1 = scale.fit_transform(X_train)
     X_test1 = scale.transform(X_test)
```

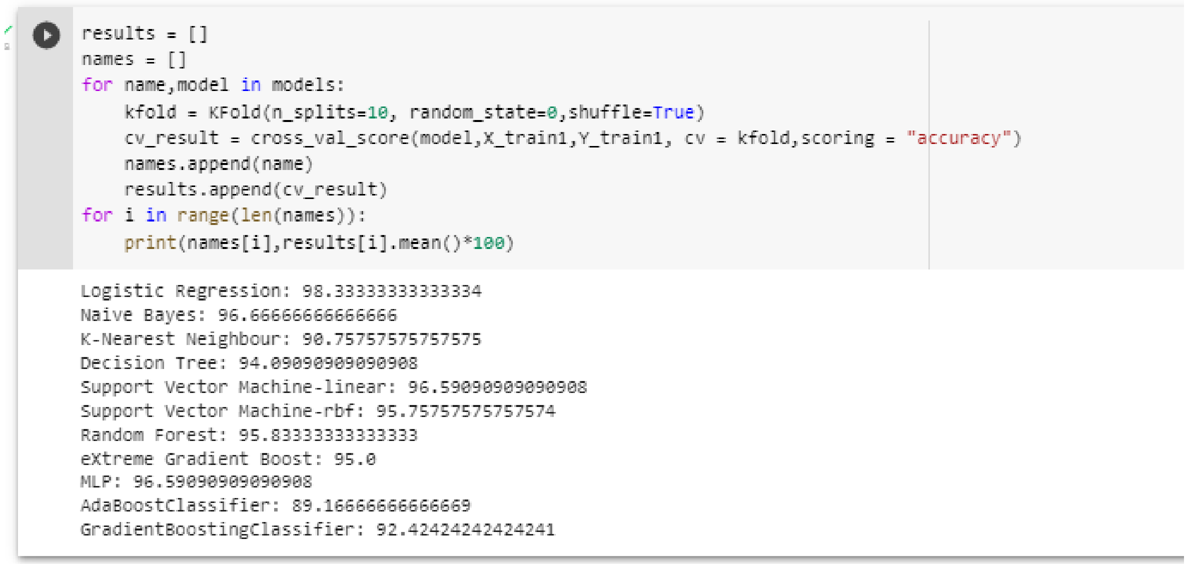
g. Explore Different Models and short-list best ones

For the project I have tried 5 models. Each model work in different algorithms, so lets try each of them. Let's check different models by running via loop and predict each modal accuracy.



```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

models = []
models.append(("Logistic Regression:",LogisticRegression()))
models.append(("Naive Bayes:",GaussianNB()))
models.append(("K-Nearest Neighbour:",KNeighborsClassifier(n_neighbors=3)))
models.append(("Decision Tree:",DecisionTreeClassifier()))
models.append(("Support Vector Machine-linear:",SVC(kernel="linear")))
models.append(("Support Vector Machine-rbf:",SVC(kernel="rbf")))
models.append(("Random Forest:",RandomForestClassifier(n_estimators=7)))
models.append(("eXtreme Gradient Boost:",XGBClassifier()))
models.append(("MLP:",MLPClassifier(hidden_layer_sizes=(45,30,15),solver='sgd',learning_rate_init=0.01,max_iter=500)))
models.append(("AdaBoostClassifier:",AdaBoostClassifier()))
models.append(("GradientBoostingClassifier:",GradientBoostingClassifier()))

print('Models appended...')

Models appended...
```

Let's use for loop to check accuracy of each model using our train datasets.



```
results = []
names = []
for name,model in models:
    kfold = KFold(n_splits=10, random_state=0,shuffle=True)
    cv_result = cross_val_score(model,X_train1,Y_train1, cv = kfold,scoring = "accuracy")
    names.append(name)
    results.append(cv_result)
for i in range(len(names)):
    print(names[i],results[i].mean()*100)
```

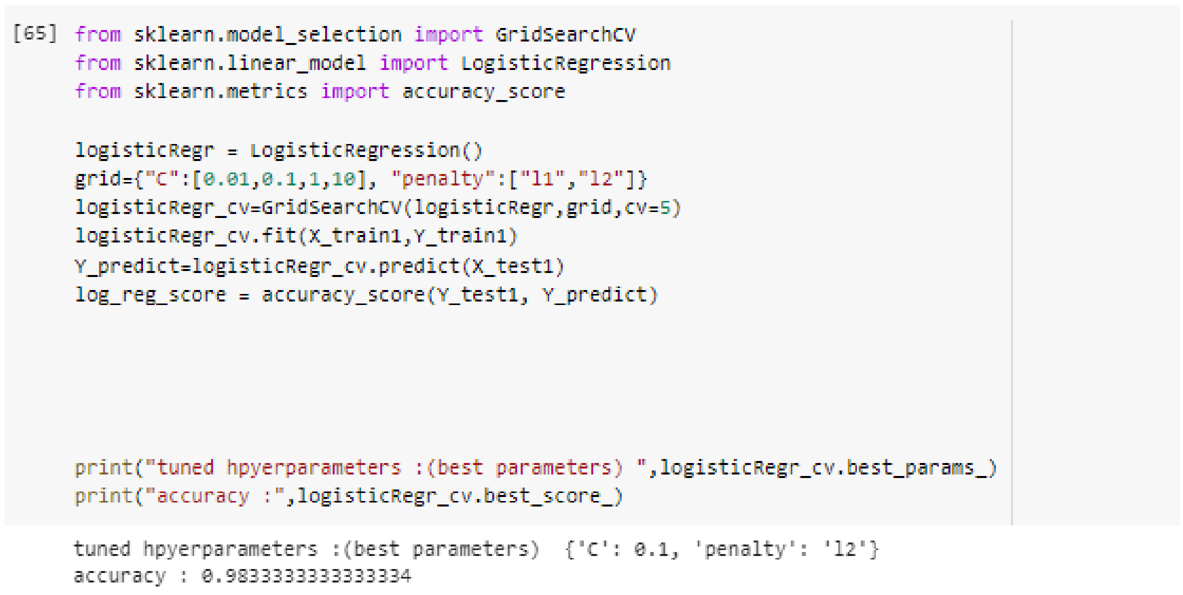
```
Logistic Regression: 98.33333333333334
Naive Bayes: 96.66666666666666
K-Nearest Neighbour: 90.75757575757575
Decision Tree: 94.09090909090908
Support Vector Machine-linear: 96.59090909090908
Support Vector Machine-rbf: 95.75757575757574
Random Forest: 95.83333333333333
eXtreme Gradient Boost: 95.0
MLP: 96.59090909090908
AdaBoostClassifier: 89.16666666666669
GradientBoostingClassifier: 92.42424242424241
```

Here we can see that all these models have high accuracy. Now let's try some of models with hyper parameter tuning so that we can get higher accuracy.

h. Fine tuning models

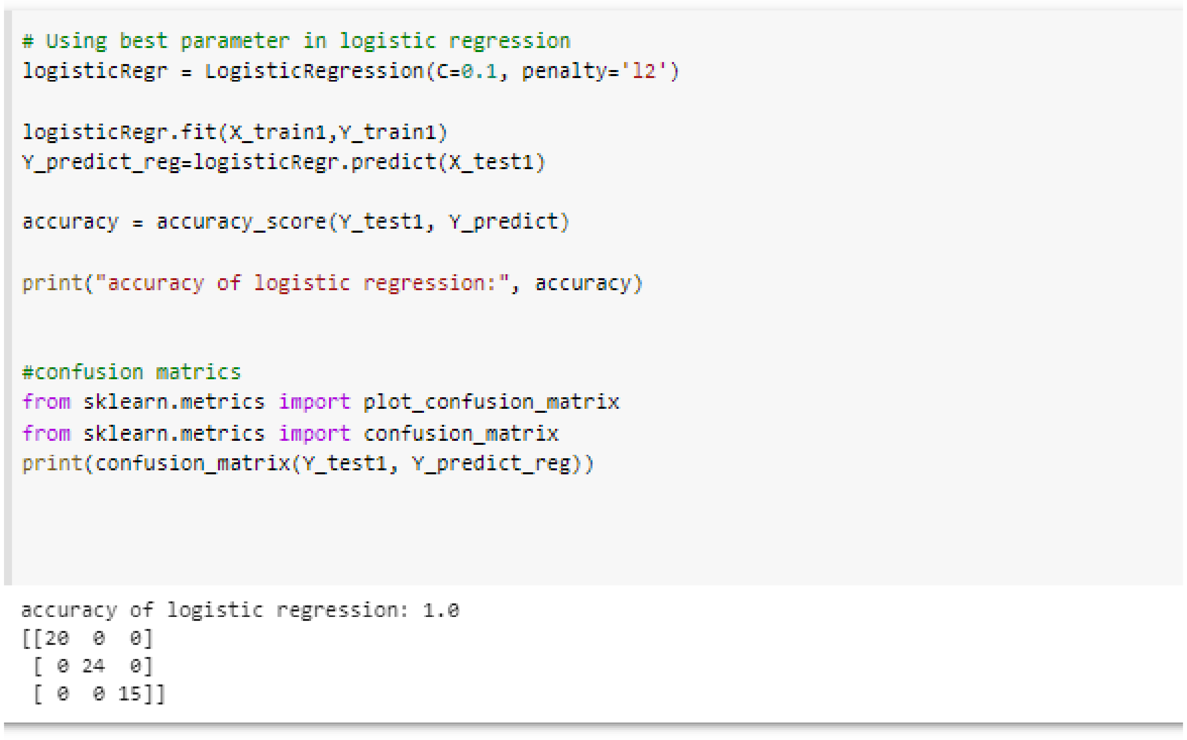
The best hyper parameter is searched with GridSearchCV(). Let's try with different models.

Let's check with Logistic Regression.



```
[65] from sklearn.model_selection import GridSearchCV
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score

     logisticRegr = LogisticRegression()
     grid={"C":[0.01,0.1,1,10], "penalty":["l1","l2"]}
     logisticRegr_cv=GridSearchCV(logisticRegr,grid,cv=5)
     logisticRegr_cv.fit(X_train1,Y_train1)
     Y_predict=logisticRegr_cv.predict(X_test1)
     log_reg_score = accuracy_score(Y_test1, Y_predict)




     print("tuned hpyerparameters :(best parameters) ",logisticRegr_cv.best_params_)
     print("accuracy :",logisticRegr_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2'}
accuracy : 0.9833333333333334
```

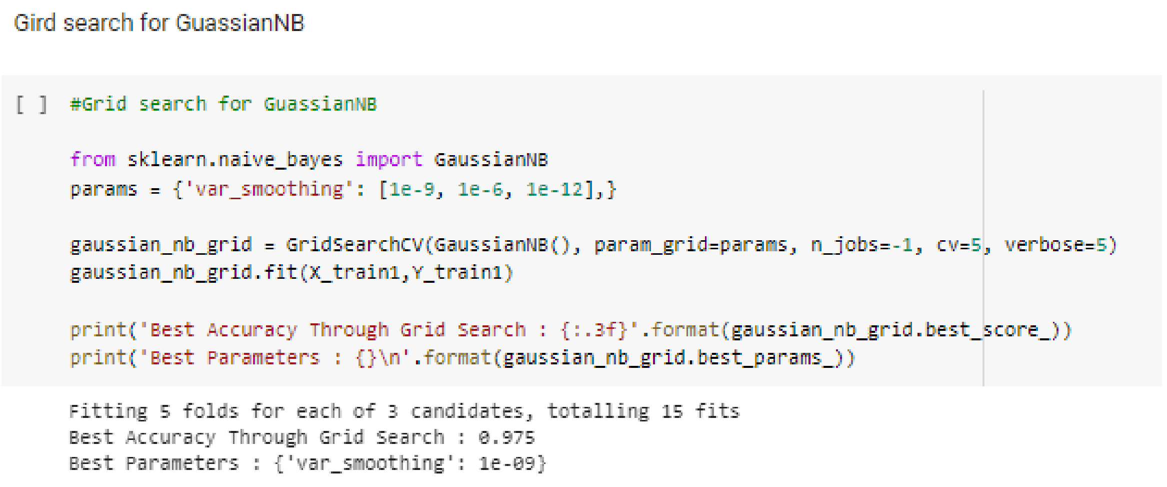


```
# Using best parameter in logistic regression
logisticRegr = LogisticRegression(C=0.1, penalty='l2')

logisticRegr.fit(X_train1,Y_train1)
Y_predict_reg=logisticRegr.predict(X_test1)

accuracy = accuracy_score(Y_test1, Y_predict)

print("accuracy of logistic regression:", accuracy)


#confusion matrics
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
print(confusion_matrix(Y_test1, Y_predict_reg))
```

```
accuracy of logistic regression: 1.0
[[20  0  0]
 [ 0 24  0]
 [ 0  0 15]]
```

The modal has predicted with 100% accuracy. The parameters of a logistic regression are most commonly estimated by maximum-likelihood estimation (MLE). This does not have a closed-form expression, unlike linear least squares. Logistic regression is used in various fields, including machine learning, most medical fields, and social sciences.

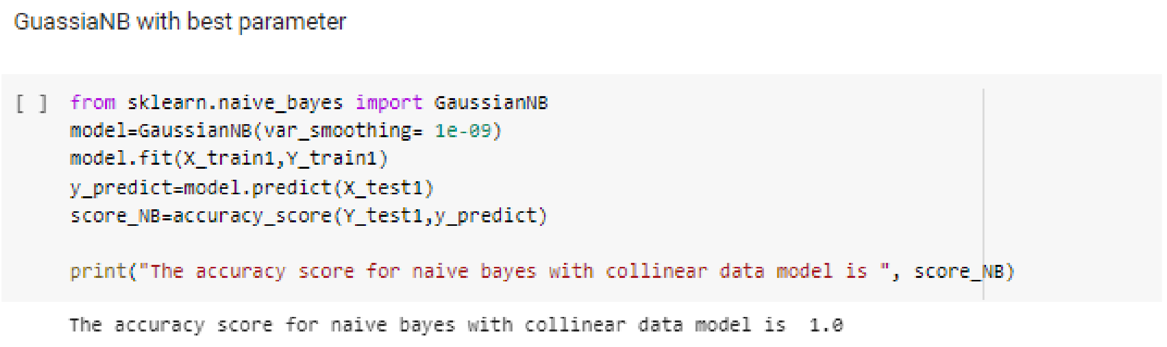
Let’s look with Gaussian Naive Bayes (GaussianNB)



Gird search for GuassianNB

```
[ ]  #Grid search for GuassianNB

     from sklearn.naive_bayes import GaussianNB
     params = {'var_smoothing': [1e-9, 1e-6, 1e-12],}

     gaussian_nb_grid = GridSearchCV(GaussianNB(), param_grid=params, n_jobs=-1, cv=5, verbose=5)
     gaussian_nb_grid.fit(X_train1,Y_train1)

     print('Best Accuracy Through Grid Search : {:.3f}'.format(gaussian_nb_grid.best_score_))
     print('Best Parameters : {}\n'.format(gaussian_nb_grid.best_params_))
```

```
     Fitting 5 folds for each of 3 candidates, totalling 15 fits
     Best Accuracy Through Grid Search : 0.975
     Best Parameters : {'var_smoothing': 1e-09}
```



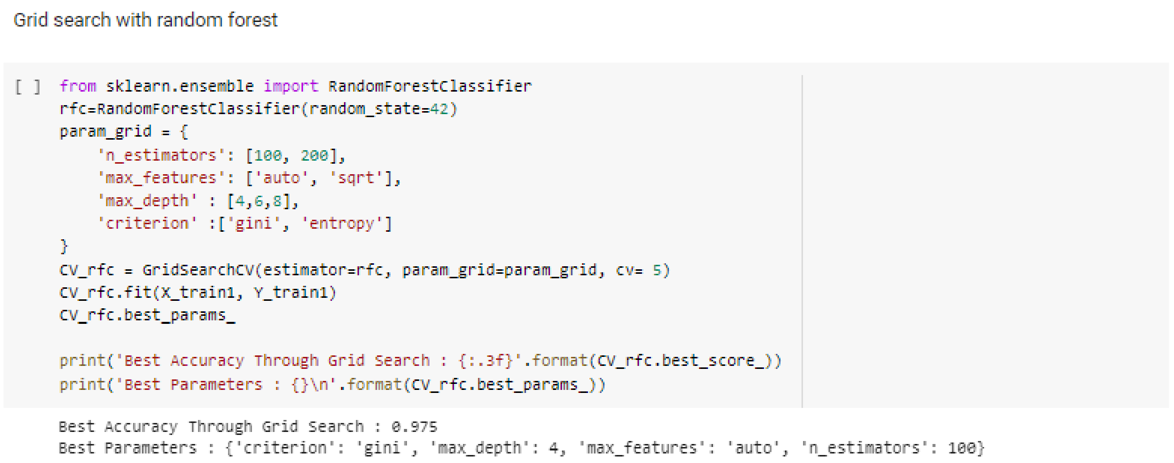
GuassiaNB with best parameter

```
[ ]  from sklearn.naive_bayes import GaussianNB
     model=GaussianNB(var_smoothing= 1e-09)
     model.fit(X_train1,Y_train1)
     y_predict=model.predict(X_test1)
     score_NB=accuracy_score(Y_test1,y_predict)

     print("The accuracy score for naive bayes with collinear data model is ", score_NB)
```

```
     The accuracy score for naive bayes with collinear data model is  1.0
```

GaussianNB also works well with test data. This model can perform online updates to model parameters via partial fit. This method is expected to be called several times consecutively on different chunks of a dataset to implement out-of-core or online learning.

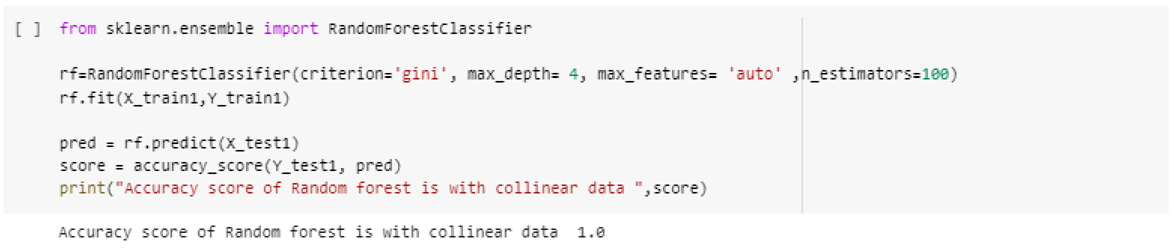
Let’s look Random Forest Classifier.



Grid search with random forest

```
[ ] from sklearn.ensemble import RandomForestClassifier
    rfc=RandomForestClassifier(random_state=42)
    param_grid = {
        'n_estimators': [100, 200],
        'max_features': ['auto', 'sqrt'],
        'max_depth' : [4,6,8],
        'criterion' :['gini', 'entropy']
    }
    CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
    CV_rfc.fit(X_train1, Y_train1)
    CV_rfc.best_params_

    print('Best Accuracy Through Grid Search : {:.3f}'.format(CV_rfc.best_score_))
    print('Best Parameters : {}\n'.format(CV_rfc.best_params_))
```

```
Best Accuracy Through Grid Search : 0.975
Best Parameters : {'criterion': 'gini', 'max_depth': 4, 'max_features': 'auto', 'n_estimators': 100}
```

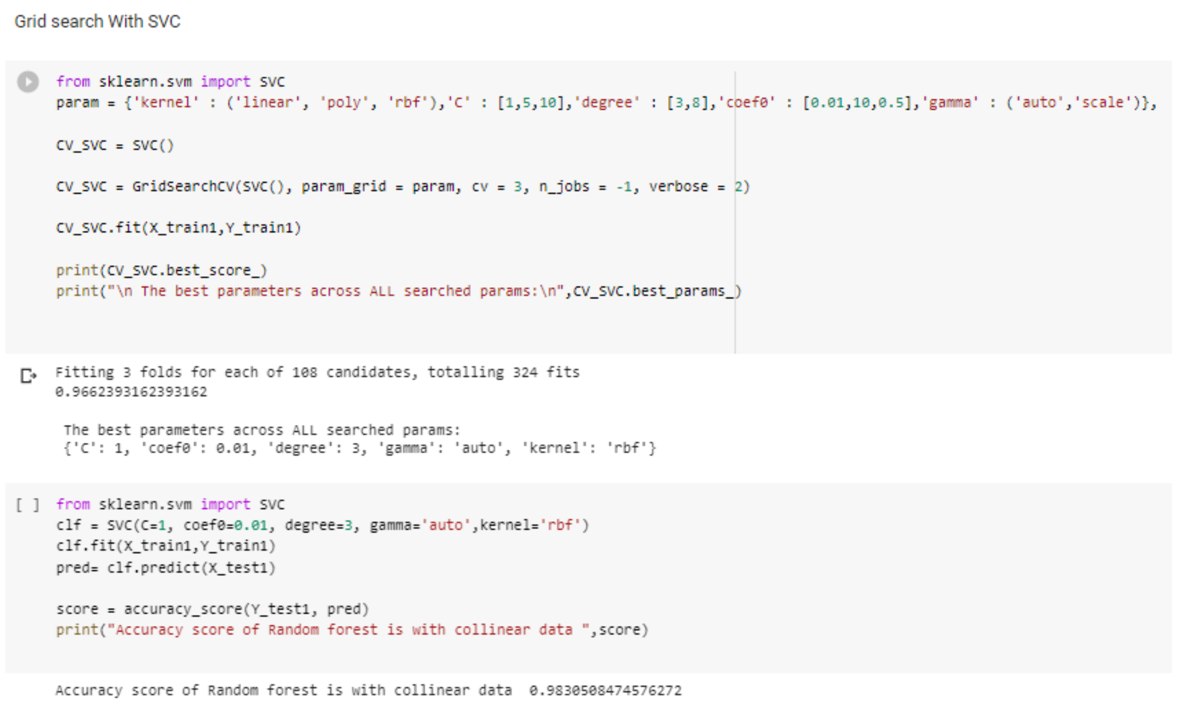


```
[ ] from sklearn.ensemble import RandomForestClassifier

    rf=RandomForestClassifier(criterion='gini', max_depth= 4, max_features= 'auto' ,n_estimators=100)
    rf.fit(X_train1,Y_train1)

    pred = rf.predict(X_test1)
    score = accuracy_score(Y_test1, pred)
    print("Accuracy score of Random forest is with collinear data ",score)
```

```
Accuracy score of Random forest is with collinear data  1.0
```

Here again accuracy is perfect for the test dataset. Random forest regressor fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Let’s check with C-Support Vector Classification


Grid search With SVC

```
from sklearn.svm import SVC
param = {'kernel' : ('linear', 'poly', 'rbf'),'C' : [1,5,10],'degree' : [3,8],'coef0' : [0.01,10,0.5],'gamma' : ('auto','scale')},

CV_SVC = SVC()

CV_SVC = GridSearchCV(SVC(), param_grid = param, cv = 3, n_jobs = -1, verbose = 2)

CV_SVC.fit(X_train1,Y_train1)

print(CV_SVC.best_score_)
print("\n The best parameters across ALL searched params:\n",CV_SVC.best_params_)
```

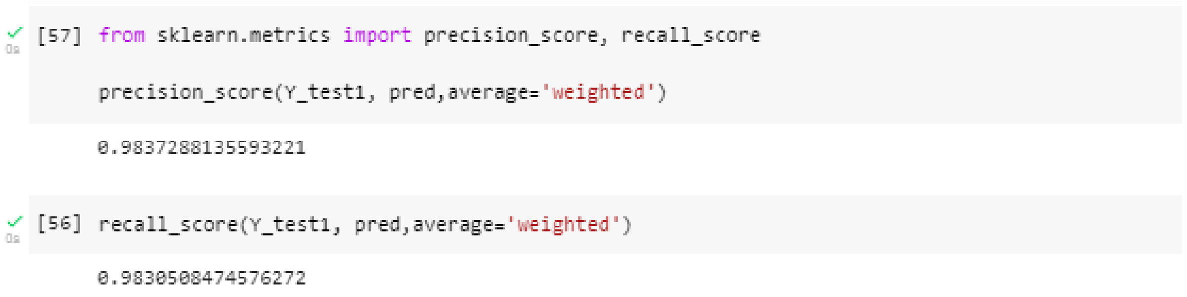
```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
0.9662393162393162

 The best parameters across ALL searched params:
 {'C': 1, 'coef0': 0.01, 'degree': 3, 'gamma': 'auto', 'kernel': 'rbf'}
```

```
[ ] from sklearn.svm import SVC
    clf = SVC(C=1, coef0=0.01, degree=3, gamma='auto',kernel='rbf')
    clf.fit(X_train1,Y_train1)
    pred= clf.predict(X_test1)

    score = accuracy_score(Y_test1, pred)
    print("Accuracy score of Random forest is with collinear data ",score)
```

```
Accuracy score of Random forest is with collinear data  0.9830508474576272
```

Here, the accuracy is 98.3%, lets check precisions and recalls.

Precisions and recalls



```
[57] from sklearn.metrics import precision_score, recall_score

     precision_score(Y_test1, pred,average='weighted')
```

```
0.9837288135593221
```

```
[56] recall_score(Y_test1, pred,average='weighted')
```

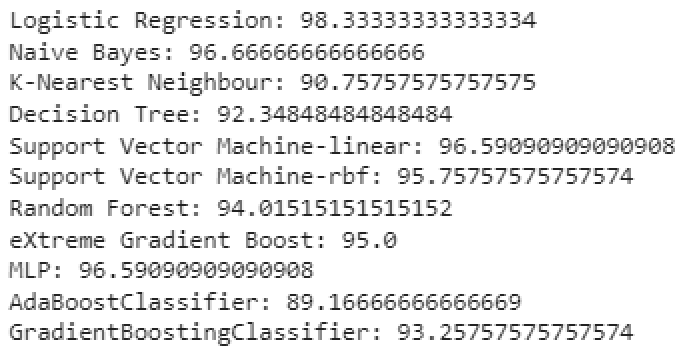
```
0.9830508474576272
```

The Precisions and recalls are quite high, so trade off is significantly low. Therefore, models behaves well for these data sets.

The SVC classifiers fits least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. The method works on simple estimators as well as on nested objects (such as Pipeline).

i. Present Your Solution

When we look for train data for different modals, we get following accuracy.



```
Logistic Regression: 98.33333333333334
Naive Bayes: 96.66666666666666
K-Nearest Neighbour: 90.75757575757575
Decision Tree: 92.34848484848484
Support Vector Machine-linear: 96.59090909090908
Support Vector Machine-rbf: 95.75757575757574
Random Forest: 94.01515151515152
eXtreme Gradient Boost: 95.0
MLP: 96.59090909090908
AdaBoostClassifier: 89.16666666666669
GradientBoostingClassifier: 93.25757575757574
```

And we have tried best parameter for test data in Logistic Regression, GaussianNB, Random Forest Classifier and Support Vector machine. The first three models predict accurately with 100% accuracy. However, the result of support vector machine is 98.3% accuracy with weighted precisions 98.4% and recall 98.3%, which is also a good modal.

j. Launch Monitor and Maintain your System

For simplicity, random forest regressor model is selected for the project. This is because sub-sample size can be controlled with the “max_samples” parameter. Otherwise, the whole dataset can also be used to build each tree. It is promising to perform well for large amount of data with similar features. This is because it uses averaging to improve the predictive accuracy and control over-fitting.

As production goes on periodic hyper parameter tuning and accuracy check should be performed to maintain the system.