Importing Dataset

```
import pandas as pd
```

```
datasets=pd.read_csv("/content/Indian Liver Patient Dataset (ILPD).csv",names=['Age','Gender','Total_Bilirubin','Direct_Bilirubin','Alkaline
```

```
datasets
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Amir
0	65	Female	0.7	0.1	187	
1	62	Male	10.9	5.5	699	
2	62	Male	7.3	4.1	490	
3	58	Male	1.0	0.4	182	
4	72	Male	3.9	2.0	195	
...	...	...	...	...	...	
578	60	Male	0.5	0.1	500	
579	40	Male	0.6	0.1	98	
580	52	Male	0.8	0.2	245	
581	31	Male	1.3	0.5	184	
582	38	Male	1.0	0.3	216	

583 rows × 11 columns

```
datasets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphatase        583 non-null    int64
 5   Alkaline_Aminotransferase   583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Proteins              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

```
datasets.describe()
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspartate_Aminotransferase	Total_
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	5
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	

Preprocessing

```
datasets.isnull().sum()
```

```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphatase          0
Alkaline_Aminotransferase     0
Aspartate_Aminotransferase    0
Total_Proteins                0
Albumin                       0
Albumin_and_Globulin_Ratio    4
Dataset                       0
dtype: int64
```

```
datasets['Albumin_and_Globulin_Ratio'].fillna(value=datasets['Albumin_and_Globulin_Ratio'].mean(), inplace=True)
```

```
datasets.isnull().sum()
```

```
Age                           0
Gender                        0
Total_Bilirubin               0
Direct_Bilirubin              0
Alkaline_Phosphatase          0
Alkaline_Aminotransferase     0
Aspartate_Aminotransferase    0
Total_Proteins                0
Albumin                       0
Albumin_and_Globulin_Ratio    0
Dataset                       0
dtype: int64
```

```
datasets.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphatase        583 non-null    int64
 5   Alkaline_Aminotransferase   583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Proteins              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  583 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

```
datasets.describe()
```

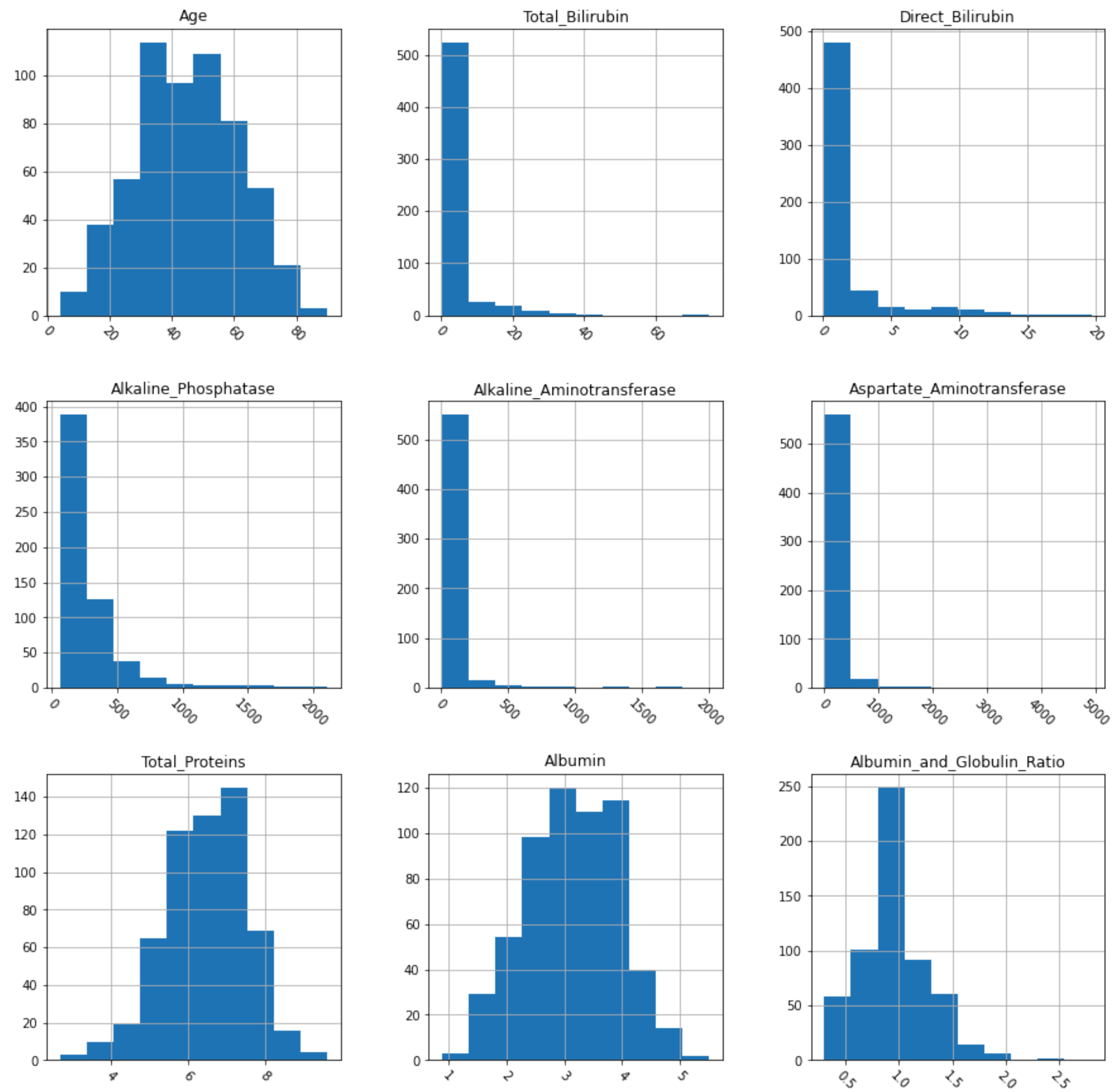
	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspartate_Aminotransferase	Total_
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	5
mean	44.746141	3.298799	1.486106	290.576329	80.713551	109.910806	
std	16.189833	6.209522	2.808498	242.937989	182.620356	288.918529	
min	4.000000	0.400000	0.100000	63.000000	10.000000	10.000000	
25%	33.000000	0.800000	0.200000	175.500000	23.000000	25.000000	
50%	45.000000	1.000000	0.300000	208.000000	35.000000	42.000000	
75%	58.000000	2.600000	1.300000	298.000000	60.500000	87.000000	
max	90.000000	75.000000	19.700000	2110.000000	2000.000000	4929.000000	

```
print(datasets['Dataset'].value_counts())
```

```
1    416
2    167
Name: Dataset, dtype: int64
```

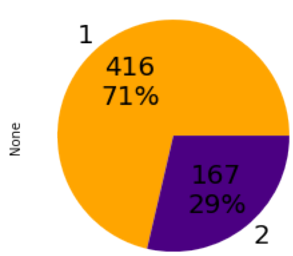
Visualizing each Attribute

```
import matplotlib.pyplot as plt
plot=datasets.drop(['Dataset'],axis=1)
plot.hist(figsize=(15,15), xrot=-45, bins=10)
plt.show()
```



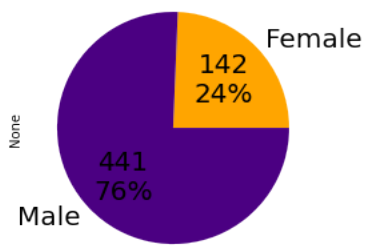

```
import matplotlib.pyplot as plt
def label_function(val):
    return f'{val / 100 * len(datasets):.0f}\n{val:.0f}%'
datasets.groupby('Dataset').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},
                                 colors=['orange','indigo'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5ef35ee50>
```



```
import matplotlib.pyplot as plt
def label_function(val):
    return f'{val / 100 * len(datasets):.0f}\n{val:.0f}%'
datasets.groupby('Gender').size().plot(kind='pie', autopct=label_function, textprops={'fontsize': 20},
                                  colors=['orange','indigo'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5f271f450>
```




```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
datasets['Gender']= label_encoder.fit_transform(datasets['Gender'])
datasets['Gender'].unique()
```

```
array([0, 1])
```

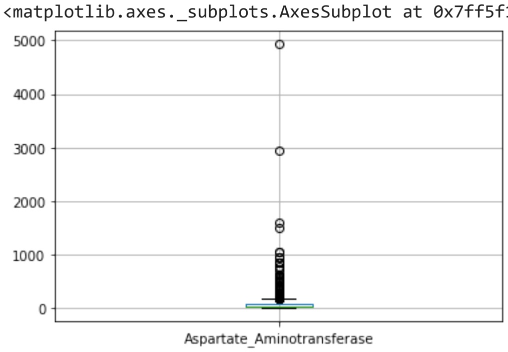
```
datasets.describe()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspartate_Aminotransfe
count	583.000000	583.000000	583.000000	583.000000	583.000000	583.000000	583.00
mean	44.746141	0.756432	3.298799	1.486106	290.576329	80.713551	109.91
std	16.189833	0.429603	6.209522	2.808498	242.937989	182.620356	288.91
min	4.000000	0.000000	0.400000	0.100000	63.000000	10.000000	10.00
25%	33.000000	1.000000	0.800000	0.200000	175.500000	23.000000	25.00
50%	45.000000	1.000000	1.000000	0.300000	208.000000	35.000000	42.00
75%	58.000000	1.000000	2.600000	1.300000	298.000000	60.500000	87.00
max	90.000000	1.000000	75.000000	19.700000	2110.000000	2000.000000	4929.00

Outliers

```
datasets[['Aspartate_Aminotransferase']].boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5f12346d0>
```




```
datasets = datasets[datasets.Aspartate_Aminotransferase <1000 ]
datasets.shape
```

```
(577, 11)
```

Removing Duplicates

```
datasets = datasets.drop_duplicates()
print( datasets.shape )
```

```
(564, 11)
```

```
x=datasets.drop(['Dataset'],axis=1)
```

Correlation Matrix

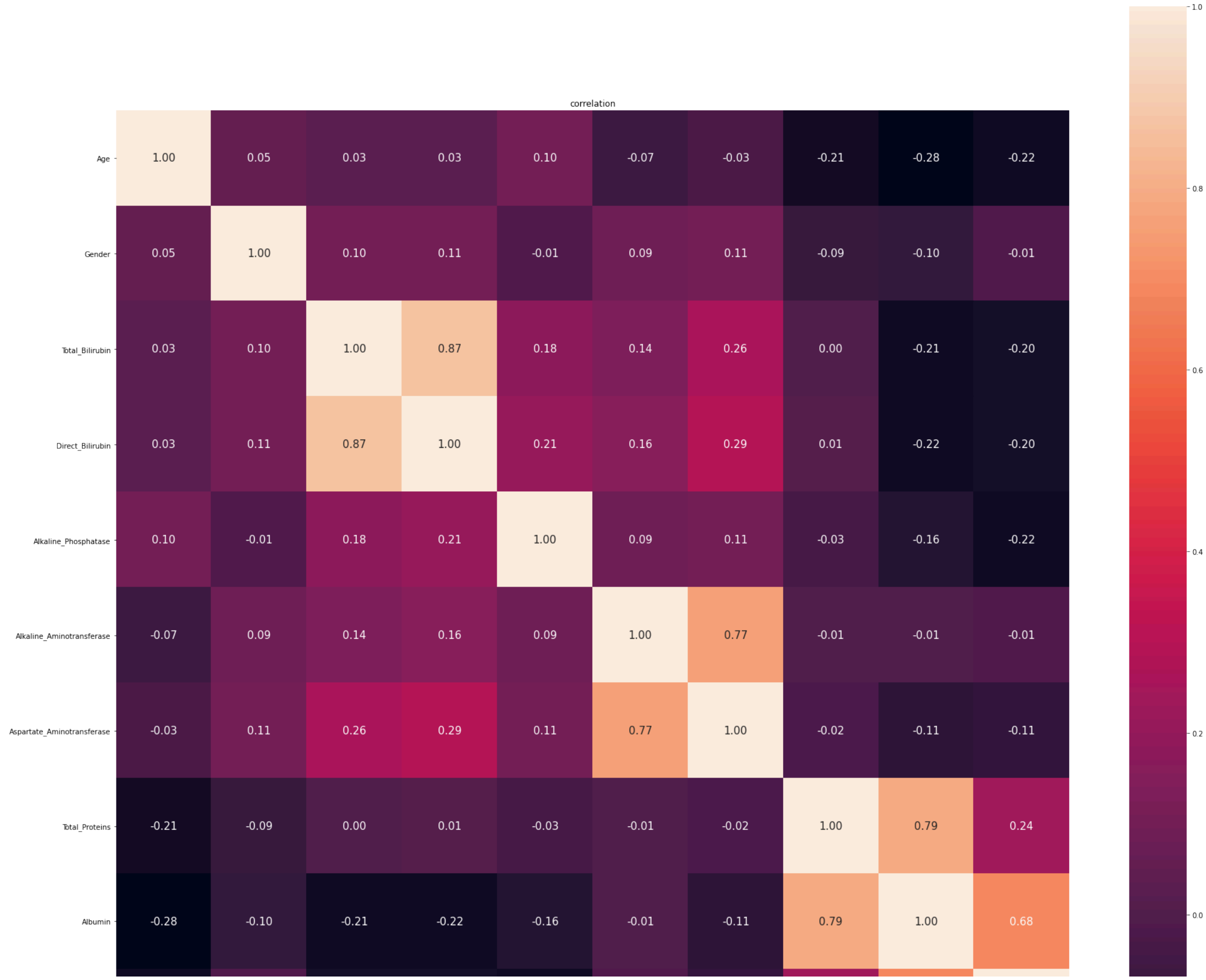
```
x.corr()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspa
Age	1.000000	0.053095	0.033594	0.027999	0.096047	-0.065282	
Gender	0.053095	1.000000	0.097594	0.108489	-0.012457	0.086565	
Total_Bilirubin	0.033594	0.097594	1.000000	0.867714	0.178303	0.138926	
Direct_Bilirubin	0.027999	0.108489	0.867714	1.000000	0.208704	0.163306	
Alkaline_Phosphatase	0.096047	-0.012457	0.178303	0.208704	1.000000	0.090631	
Alkaline_Aminotransferase	-0.065282	0.086565	0.138926	0.163306	0.090631	1.000000	
Aspartate_Aminotransferase	-0.028375	0.110416	0.262954	0.292252	0.105105	0.767775	
Total_Proteins	-0.205213	-0.085642	0.001889	0.009725	-0.031360	-0.005786	
Albumin	-0.279131	-0.095494	-0.214703	-0.221637	-0.156434	-0.006130	
Albumin_and_Globulin_Ratio	-0.220544	-0.011801	-0.202318	-0.196155	-0.221363	-0.012539	

Correlation Heatmap

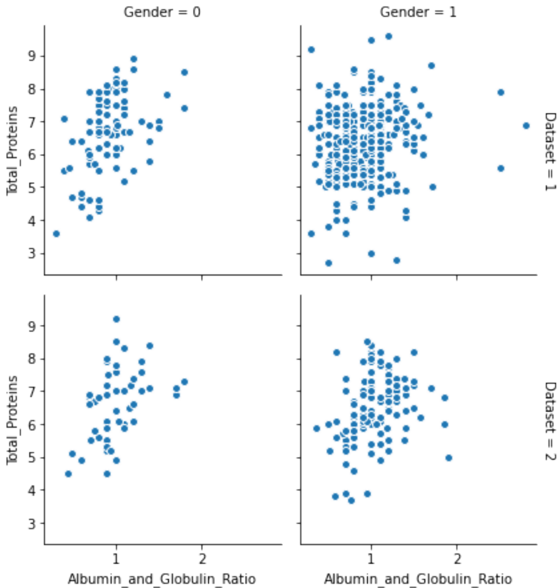
```
import seaborn as sns
plt.figure(figsize=(30, 30))
sns.heatmap(x.corr(), cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 15},)
plt.title("correlation")
```

```
Text(0.5, 1.0, 'correlation')
```

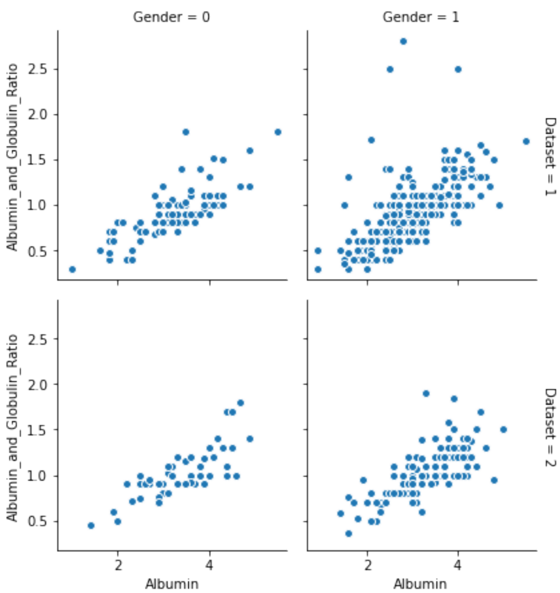


Scatterplot and Jointplot between different attributes

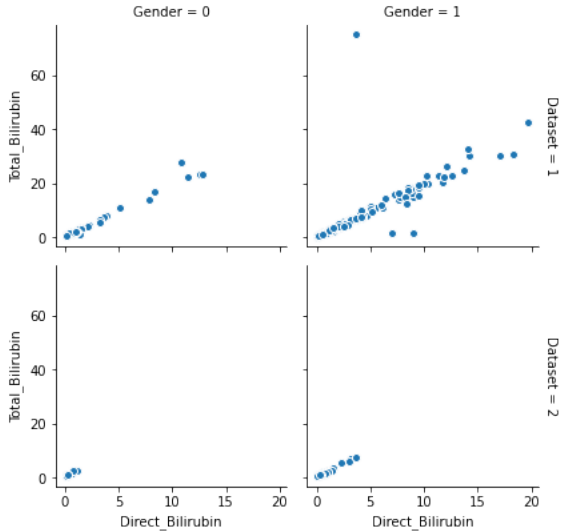
```
import seaborn as sns
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin_and_Globulin_Ratio", "Total_Proteins",  edgecolor="w")
plt.subplots_adjust(top=1)
```




```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin", "Albumin_and_Globulin_Ratio",  edgecolor="w")
plt.subplots_adjust(top=1)
```

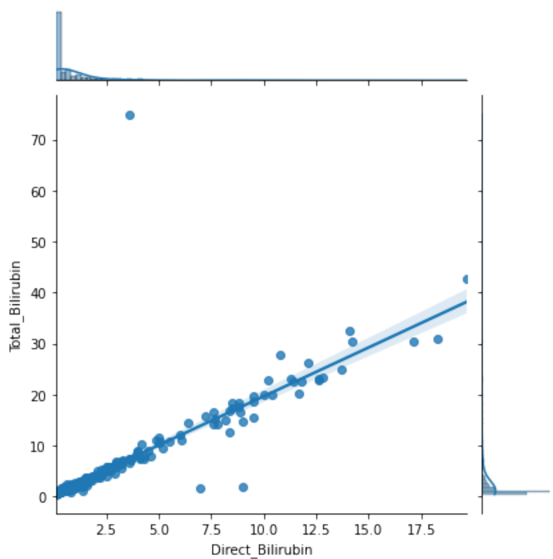



```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Direct_Bilirubin", "Total_Bilirubin", edgecolor="w")
plt.subplots_adjust(top=0.9)
```

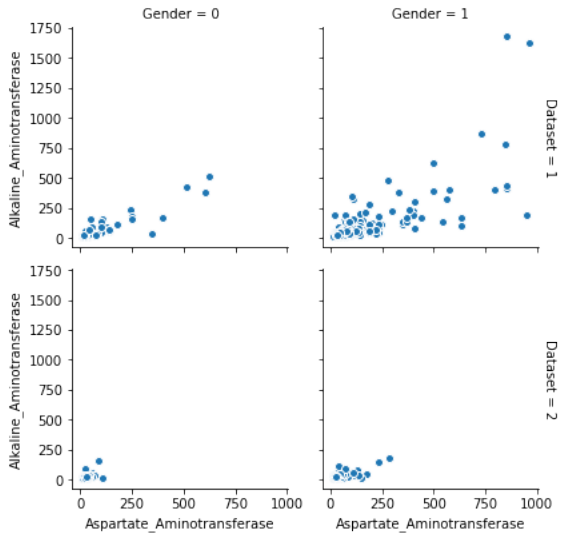



```
sns.jointplot("Direct_Bilirubin", "Total_Bilirubin", data=datasets, kind="reg")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. Frc
  FutureWarning
<seaborn.axisgrid.JointGrid at 0x7ff5f1a526d0>
```

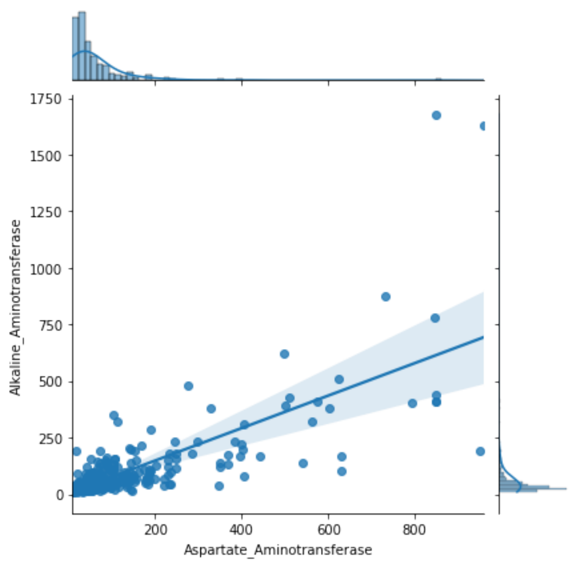



```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,'Aspartate_Aminotransferase','Alkaline_Aminotransferase',  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

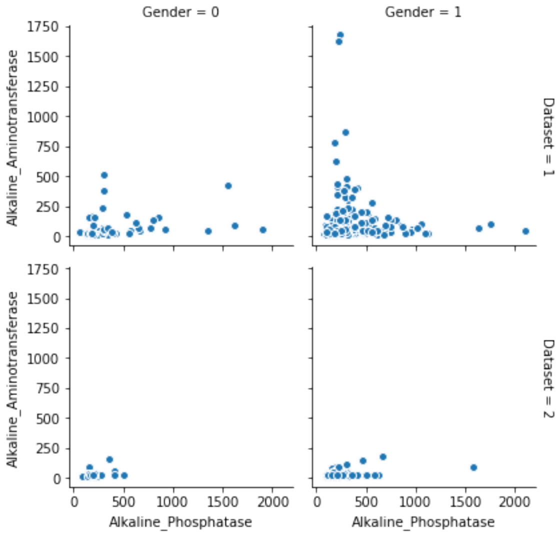



```
sns.jointplot("Aspartate_Aminotransferase","Alkaline_Aminotransferase", data=datasets, kind="reg")
```

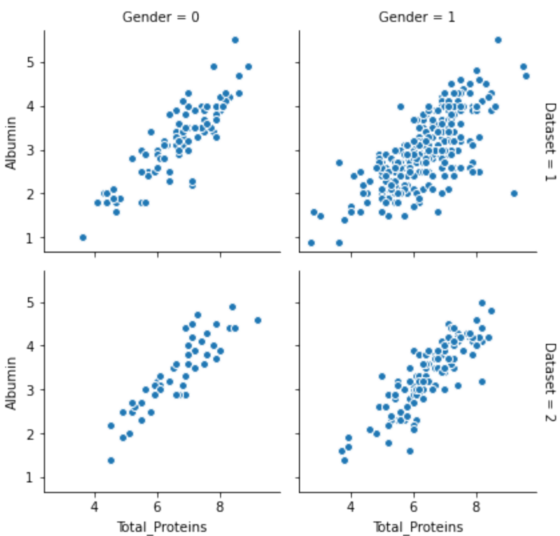
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. Frc
  FutureWarning
<seaborn.axisgrid.JointGrid at 0x7ff5f1431350>
```




```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Alkaline_Phosphatase", "Alkaline_Aminotransferase"   ,  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

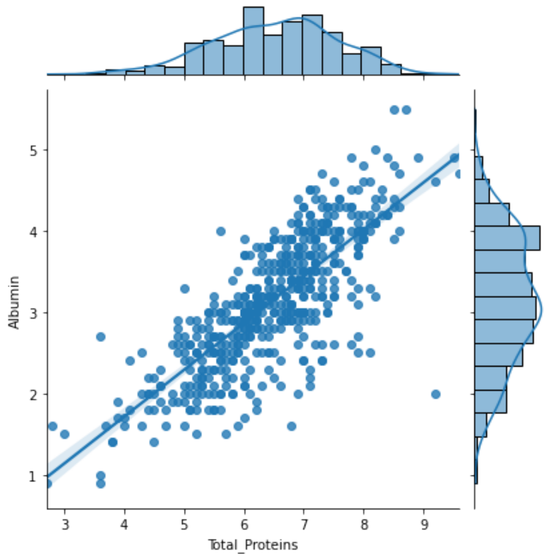



```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Total_Proteins", "Albumin",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```

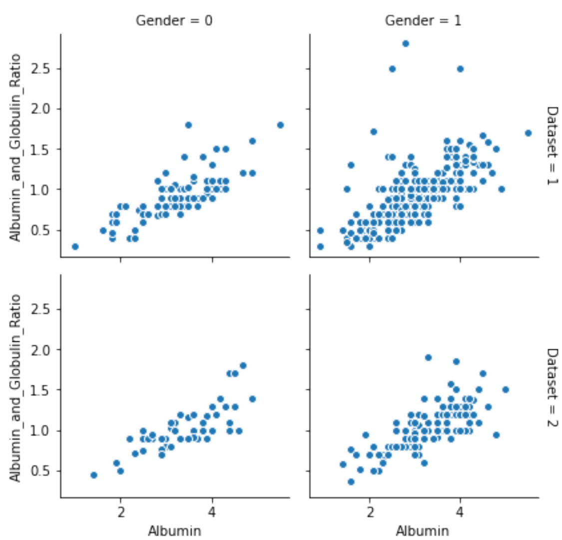



```
sns.jointplot("Total_Proteins", "Albumin", data=datasets, kind="reg")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. Frc
  FutureWarning
<seaborn.axisgrid.JointGrid at 0x7ff5ee6e9b50>
```

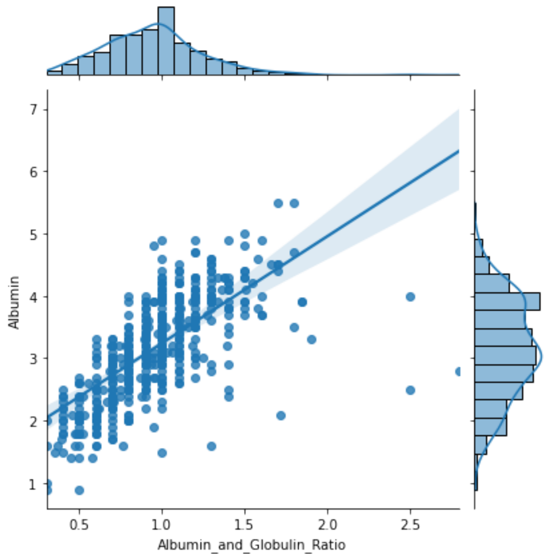



```
g = sns.FacetGrid(datasets, col="Gender", row="Dataset", margin_titles=True)
g.map(plt.scatter,"Albumin", "Albumin_and_Globulin_Ratio",  edgecolor="w")
plt.subplots_adjust(top=0.9)
```




```
sns.jointplot("Albumin_and_Globulin_Ratio","Albumin", data=datasets, kind="reg")
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. Frc
  FutureWarning
<seaborn.axisgrid.JointGrid at 0x7ff5ee8a9710>
```




```
features__datasets=datasets.drop(['Gender'],axis=1)
```

```
features__datasets
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspartate_Aminotransferase	Total_Proteins
0	65	0.7	0.1	187	16	18	6.8
1	62	10.9	5.5	699	64	100	7.5
2	62	7.3	4.1	490	60	68	7.0
3	58	1.0	0.4	182	14	20	6.8
4	72	3.9	2.0	195	27	59	7.3
...	...	...	...	...	...	...	...
578	60	0.5	0.1	500	20	34	5.9
579	40	0.6	0.1	98	35	31	6.0
580	52	0.8	0.2	245	48	49	6.4
581	31	1.3	0.5	184	29	32	6.8
582	38	1.0	0.3	216	21	24	7.3

564 rows × 10 columns

```
y = features__datasets['Dataset']
```

```
X = features__datasets.drop(['Dataset'], axis=1)
```

```
X
```

	Age	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Aspartate_Aminotransferase	Total_Proteins
0	65	0.7	0.1	187	16	18	6.8
1	62	10.9	5.5	699	64	100	7.5
2	62	7.3	4.1	490	60	68	7.0
3	58	1.0	0.4	182	14	20	6.8
4	72	3.9	2.0	195	27	59	7.3
...	...	...	...	...	...	...	...
578	60	0.5	0.1	500	20	34	5.9
579	40	0.6	0.1	98	35	31	6.0
580	52	0.8	0.2	245	48	49	6.4
581	31	1.3	0.5	184	29	32	6.8
582	38	1.0	0.3	216	21	24	7.3

564 rows × 9 columns

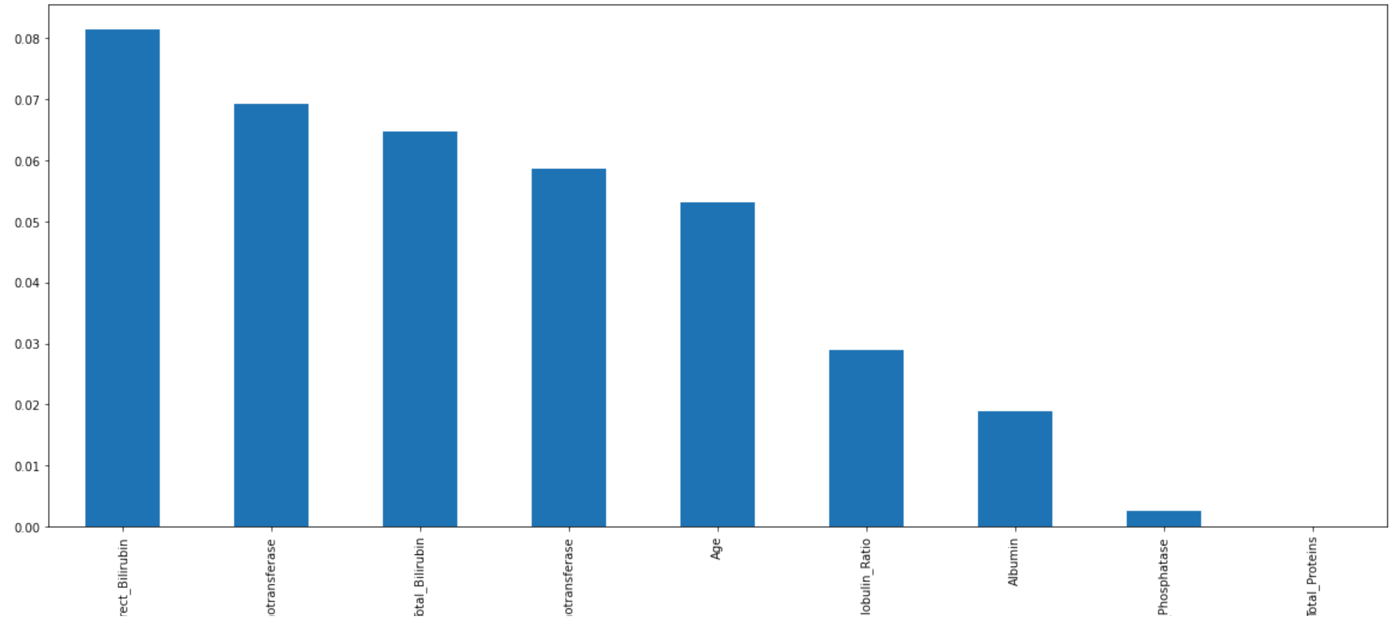
```
y
```

```
0      1
1      1
2      1
3      1
4      1
      ..
578    2
579    1
580    1
581    1
582    2
Name: Dataset, Length: 564, dtype: int64
```

Feature Selection

```
from sklearn.feature_selection import mutual_info_classif
# determine the mutual information
mutual_info = mutual_info_classif(X, y)
mutual_info
mutual_info = pd.Series(mutual_info)
mutual_info.index = X.columns
mutual_info.sort_values(ascending=False)
mutual_info.sort_values(ascending=False).plot.bar(figsize=(20, 8))
from sklearn.feature_selection import SelectKBest
sel_five_cols = SelectKBest(mutual_info_classif, k=5)
print(sel_five_cols)
sel_five_cols.fit(X, y)
print(sel_five_cols.get_support())
X=X.drop(['Total_Proteins','Direct_Bilirubin','Aspartate_Aminotransferase','Albumin_and_Globulin_Ratio'],axis=1)
```

```
SelectKBest(k=5, score_func=<function mutual_info_classif at 0x7ff5fb818290>)
[ True False  True False  True  True False False  True]
```



Dividing Data into train and test

ar lka un

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn import linear_model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
print (X_train.shape)
print (y_train.shape)
print (X_test.shape)
print (y_test.shape)
```

```
(338, 5)
(338,)
(226, 5)
(226,)
```

```
X
```

	Age	Total_Bilirubin	Alkaline_Phosphatase	Alkaline_Aminotransferase	Albumin
0	65	0.7	187	16	3.3
1	62	10.9	699	64	3.2
2	62	7.3	490	60	3.3
3	58	1.0	182	14	3.4
4	72	3.9	195	27	2.4
...	...	...	...	...	...
578	60	0.5	500	20	1.6
579	40	0.6	98	35	3.2
580	52	0.8	245	48	3.2
581	31	1.3	184	29	3.4
582	38	1.0	216	21	4.4

564 rows × 5 columns

```
y
```

```
0       1
1       1
2       1
3       1
4       1
```

```
       ..
578    2
579    1
580    1
581    1
582    2
Name: Dataset, Length: 564, dtype: int64
```

Machine Learning Models

Random Forest

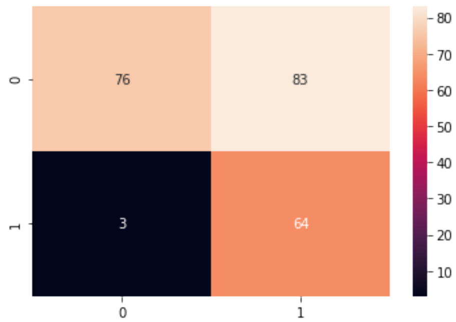
```
from sklearn.svm import SVC, LinearSVC
import numpy as np
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier
random_forest = RandomForestClassifier(n_estimators = 10)
random_forest.fit(X_train, y_train)
rf_predicted = random_forest.predict(X_test)
random_forest_score = round(random_forest.score(X_train, y_train) * 100, 2)
random_forest_score_test = round(random_forest.score(X_test, y_test) * 100, 2)
print('Random Forest Score: \n', random_forest_score)
print('Random Forest Test Score: \n', random_forest_score_test)
print('Accuracy: \n', accuracy_score(y_test,rf_predicted))
print(confusion_matrix(y_test,rf_predicted))
print(classification_report(y_test,rf_predicted))
```

```
Random Forest Score: 
 97.34
Random Forest Test Score: 
 70.8
Accuracy: 
 0.7079646017699115
[[144  15]
 [ 51  16]]
              precision    recall  f1-score   support

           1       0.74      0.91      0.81       159
           2       0.52      0.24      0.33        67

    accuracy                           0.71       226
   macro avg       0.63      0.57      0.57       226
weighted avg       0.67      0.71      0.67       226
```

Gaussian Naive Bayes

```
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_score
from sklearn.linear_model import Perceptron
gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
gauss_predicted = gaussian.predict(X_test)
gauss_score = round(gaussian.score(X_train, y_train) * 100, 2)
gauss_score_test = round(gaussian.score(X_test, y_test) * 100, 2)
print('Gaussian Score: \n', gauss_score)
print('Gaussian Test Score: \n', gauss_score_test)
print('Accuracy: \n', accuracy_score(y_test, gauss_predicted))
print('precision: \n',precision_score(y_test, gauss_predicted))
print(confusion_matrix(y_test,gauss_predicted))
print(classification_report(y_test,gauss_predicted))
gauss_score_precision=precision_score(y_test, gauss_predicted)
sns.heatmap(confusion_matrix(y_test,gauss_predicted),annot=True,fmt="d")
```

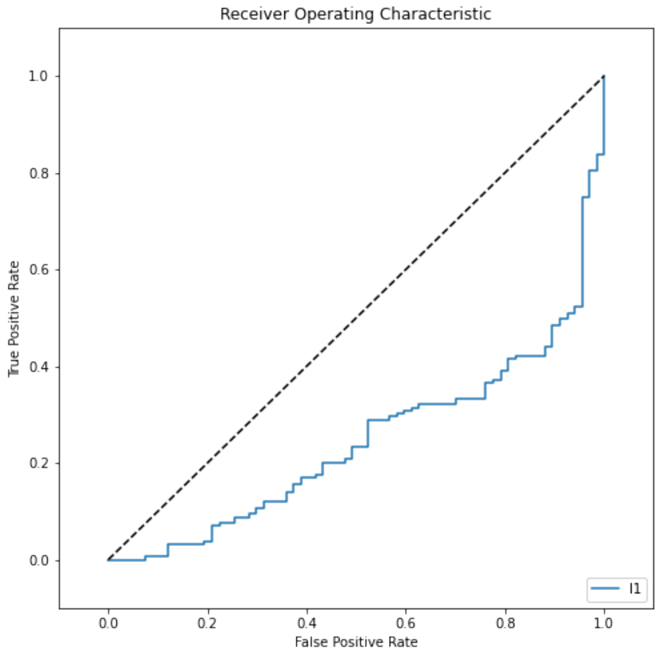
```
Gaussian Score:
 55.92
Gaussian Test Score:
 61.95
Accuracy:
 0.6194690265486725
precision:
 0.9620253164556962
[[76 83]
 [ 3 64]]
              precision    recall  f1-score   support

           1       0.96      0.48      0.64       159
           2       0.44      0.96      0.60        67

    accuracy                           0.62       226
   macro avg       0.70      0.72      0.62       226
weighted avg       0.81      0.62      0.63       226

<matplotlib.axes._subplots.AxesSubplot at 0x7ff5f18e4a50>
```




```
y_pred_proba = gaussian.predict_proba(X_test)[:,1]
```

```
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba,pos_label=1)
```

```
import matplotlib.pyplot as plt
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')

# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')

# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



MLP Neural Networks

```
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier
neural = MLPClassifier(hidden_layer_sizes=40,
                     activation='relu',
                     solver='adam',
                     alpha=0.001,
                     batch_size='auto',
                     max_iter=200,
                     random_state=137,
                     tol=0.0001,
                     early_stopping=False,
                     validation_fraction=0.1,
                     beta_1=0.9,
                     beta_2=0.999,
                     epsilon=1e-08,
                     learning_rate='constant',
                     power_t=0.5,
                     momentum=0.8,
                     nesterovs_momentum=True,
                     shuffle=True,
                     learning_rate_init=0.001)
neural.fit(X_train, y_train)
predicted = neural.predict(X_test)

neural_score = round(neural.score(X_train, y_train) * 100, 2)
mlp_precision_score=precision_score(y_test, predicted)
neural_score_test = round(neural.score(X_test, y_test) * 100, 2)
print('Neural Score: \n', neural_score)
print('Neural Test Score: \n', neural_score_test)
print('Accuracy: \n', accuracy_score(y_test, predicted))
print(confusion_matrix(predicted,y_test))
print(classification_report(y_test,predicted))
```

```
Neural Score:
 72.19
Neural Test Score:
 71.68
Accuracy:
 0.7168141592920354
[[144  49]
 [ 15  18]]
              precision    recall  f1-score   support

           1       0.75      0.91      0.82       159
           2       0.55      0.27      0.36        67

    accuracy                           0.72       226
```

```
    macro avg       0.65      0.59      0.59       226
 weighted avg       0.69      0.72      0.68       226
```

SVM

```
from sklearn import svm

from sklearn.metrics import f1_score
```

```
Svm = svm.SVC(kernel='rbf')
Svm.fit(X_train,y_train)
svm_yhat = Svm.predict(X_test)
```

```
from sklearn.metrics import f1_score
svm_f1 = f1_score(y_test, svm_yhat, average='weighted')
print(svm_f1)
```

```
0.5811056200436732
```

Random Forest Hypertuning

```
import numpy as np
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [4,6,8,10,20,40,60]
min_samples_split = [2, 5,4,]
min_samples_leaf = [1, 2,5]
bootstrap = [True, False]
param_grid = {'n_estimators': n_estimators,
              'min_samples_leaf': min_samples_leaf,
              'max_depth': max_depth,
              'bootstrap': bootstrap }
rf_Model = RandomForestClassifier()
from sklearn.model_selection import GridSearchCV
rf_Grid = GridSearchCV(estimator = rf_Model, param_grid = param_grid, cv = 3, verbose=2, n_jobs = 4)
```

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

```
Fitting 3 folds for each of 420 candidates, totalling 1260 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  45 tasks      | elapsed:    8.8s
[Parallel(n_jobs=4)]: Done 166 tasks      | elapsed:   35.6s
[Parallel(n_jobs=4)]: Done 369 tasks      | elapsed:  1.4min
[Parallel(n_jobs=4)]: Done 652 tasks      | elapsed:  2.4min
[Parallel(n_jobs=4)]: Done 1017 tasks      | elapsed:  3.5min
[Parallel(n_jobs=4)]: Done 1260 out of 1260 | elapsed:  4.3min finished
GridSearchCV(cv=3, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=4,
             param_grid={'bootstrap': [True, False],
                         'max_depth': [4, 6, 8, 10, 20, 40, 60],
                         'min_samples_leaf': [1, 2, 5],
                         'n_estimators': [10, 42, 74, 106, 138, 171, 203, 235,
                                          267, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

```
rf_Grid.best_params_
```

```
{'bootstrap': True, 'max_depth': 4, 'min_samples_leaf': 2, 'n_estimators': 300}
```

```
rf_Grid.score(X_train,y_train)
```

```
0.7751479289940828
```

```
rf_Grid.score(X_test,y_test)
```

```
0.7035398230088495
```

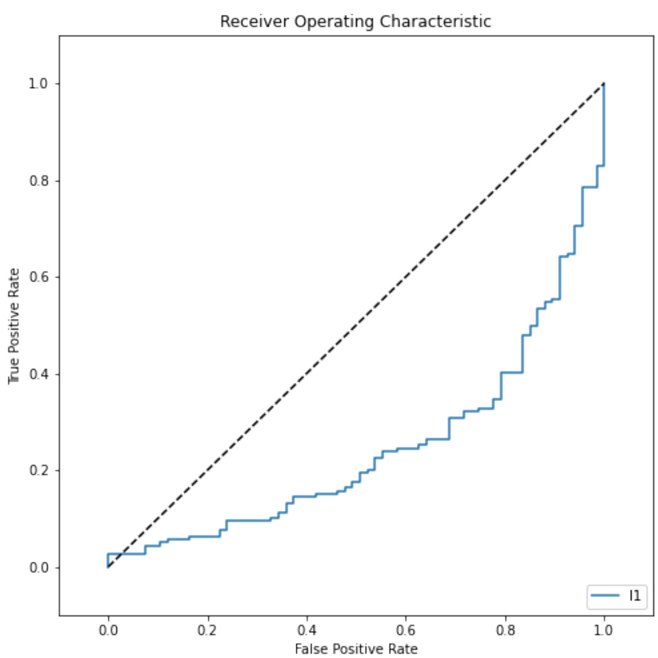
```
y_pred_proba =rf_Grid.predict_proba(X_test)[:,1]
```

```
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba,pos_label=1)
```

```
print(auc(fpr, tpr))
```

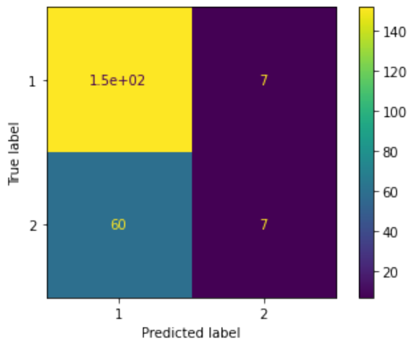
```
0.24490753778278423
```

```
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')

# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')

# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')

# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(rf_Grid, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff5ef3c7d90>
```



SVM Hypertuning

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Import Library of Support Vector Machine model
from sklearn import svm

# Create a Support Vector Classifier
svc = svm.SVC()

# Hyperparameter Optimization
parameters = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
]

# Run the grid search
grid_obj = GridSearchCV(svc, parameters)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the svc to the best combination of parameters
svc = grid_obj.best_estimator_

# Train the model using the training sets
svc.fit(X_train,y_train)
```

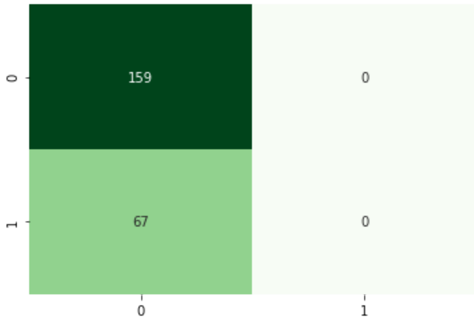
```
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
from sklearn import metrics
y_pred = svc.predict(X_test)
acc_svm = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
from sklearn.metrics import precision_score
svm_predictions=precision_score(y_test, y_pred)

print( 'Test Accuracy of SVM model : ', acc_svm )
```

```
Test Accuracy of SVM model :  70.35
```

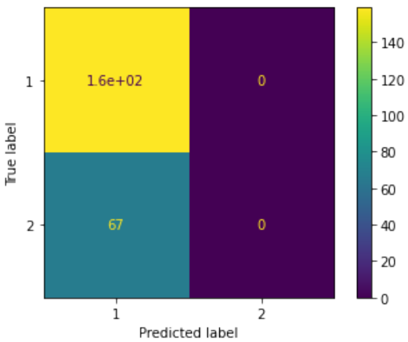
```
cm_svm = confusion_matrix(y_test,y_pred)
sns.heatmap(cm_svm,cbar=False,annot=True,cmap="Greens",fmt="d")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff5ef58f190>
```




```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(grid_obj, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff5ef603a90>
```



MLP Hypertuning

```
from sklearn.neural_network import MLPClassifier
mlp_gs = MLPClassifier(max_iter=100)
parameter_space = {
    'hidden_layer_sizes': [(10,30,10),(20,)],
    'activation': ['tanh', 'relu','logistic'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05,0.001],
    'learning_rate': ['constant','adaptive'],
}
from sklearn.model_selection import GridSearchCV
clf = GridSearchCV(mlp_gs, parameter_space, n_jobs=-1, cv=5)
clf.fit(X_train, y_train) # X is train samples and y is the corresponding labels
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:571: ConvergenceWarning: Stochastic Optimizer: M
  % self.max_iter, ConvergenceWarning)
GridSearchCV(cv=5, error_score=nan,
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_fun=15000,
                                     max_iter=100, momentum=0.9,
                                     n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_state...
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'activation': ['tanh', 'relu', 'logistic'],
                         'alpha': [0.0001, 0.05, 0.001],
                         'hidden_layer_sizes': [(10, 30, 10), (20,)],
                         'learning_rate': ['constant', 'adaptive'],
                         'solver': ['sgd', 'adam']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
print('Best parameters found:\n', clf.best_params_)
```

```
Best parameters found:
 {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd'}
```

```
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
0.701 (+/-0.036) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver
0.704 (+/-0.036) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver
0.707 (+/-0.035) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver
0.701 (+/-0.051) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver
0.707 (+/-0.021) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd
0.704 (+/-0.066) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'ada
0.686 (+/-0.039) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd
0.707 (+/-0.073) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'ada
0.704 (+/-0.044) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver':
0.692 (+/-0.061) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver':
0.710 (+/-0.014) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver':
0.689 (+/-0.043) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver':
0.680 (+/-0.084) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd'}
0.701 (+/-0.103) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'adam'
0.710 (+/-0.056) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd'}
0.704 (+/-0.071) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'adam'
0.713 (+/-0.022) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver'
0.695 (+/-0.055) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver'
0.710 (+/-0.021) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver'
0.707 (+/-0.069) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver'
0.686 (+/-0.047) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd'
0.695 (+/-0.099) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'adam
0.713 (+/-0.064) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd'
0.701 (+/-0.062) for {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'adam
0.713 (+/-0.012) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver
0.698 (+/-0.044) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver
0.713 (+/-0.012) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver
0.698 (+/-0.053) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver
0.707 (+/-0.024) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd
0.698 (+/-0.022) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'ada
0.707 (+/-0.021) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd
0.713 (+/-0.022) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'ada
0.713 (+/-0.012) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver':
0.701 (+/-0.045) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver':
0.710 (+/-0.010) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver':
0.713 (+/-0.061) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver':
0.707 (+/-0.024) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd'}
0.707 (+/-0.079) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'adam'
0.701 (+/-0.011) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd'}
0.713 (+/-0.031) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'adam'
0.713 (+/-0.012) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver'
0.686 (+/-0.039) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solver'
0.713 (+/-0.012) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver'
0.698 (+/-0.031) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'solver'
0.704 (+/-0.031) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'sgd'
0.701 (+/-0.047) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver': 'adam
0.710 (+/-0.037) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'sgd'
0.701 (+/-0.045) for {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver': 'adam
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'so
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'so
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'so
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'adaptive', 'so
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver':
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'constant', 'solver':
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver':
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.0001, 'hidden_layer_sizes': (20,), 'learning_rate': 'adaptive', 'solver':
0.713 (+/-0.012) for {'activation': 'logistic', 'alpha': 0.05, 'hidden_layer_sizes': (10, 30, 10), 'learning_rate': 'constant', 'solv
```

```
y_true, y_pred = y_test , clf.predict(X_test)
from sklearn.metrics import classification_report
print('Results on the test set:')
print(classification_report(y_true, y_pred))
print(confusion_matrix(y_pred,y_true))
```

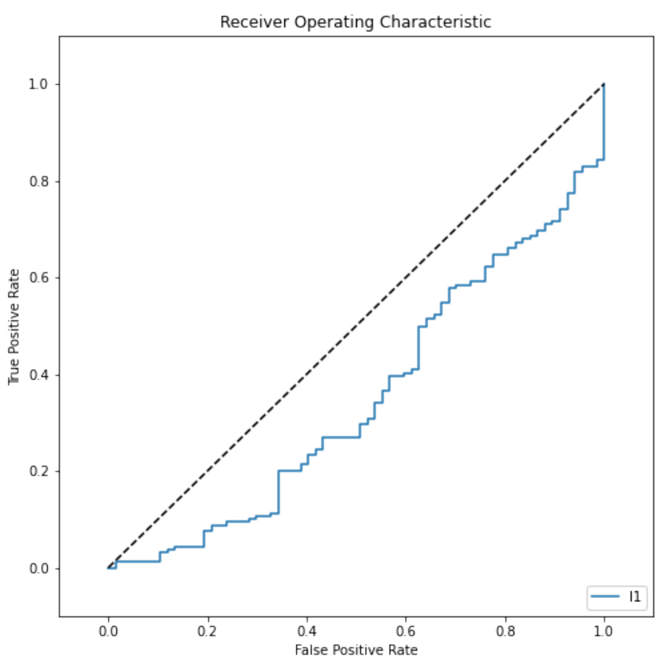
```
Results on the test set:
              precision    recall  f1-score   support

           1       0.76      0.89      0.82       159
           2       0.55      0.31      0.40        67

    accuracy                           0.72       226
   macro avg       0.65      0.60      0.61       226
```

```
weighted avg       0.70      0.72      0.69       226

[[142  46]
 [ 17  21]]
```

```
y_pred_proba = clf.predict_proba(X_test)[:,1]
```

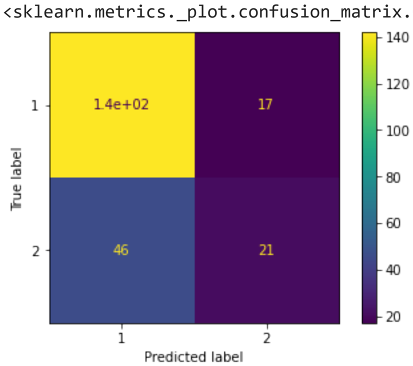
```
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba,pos_label=1)
```

```
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')

# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')

# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')

# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



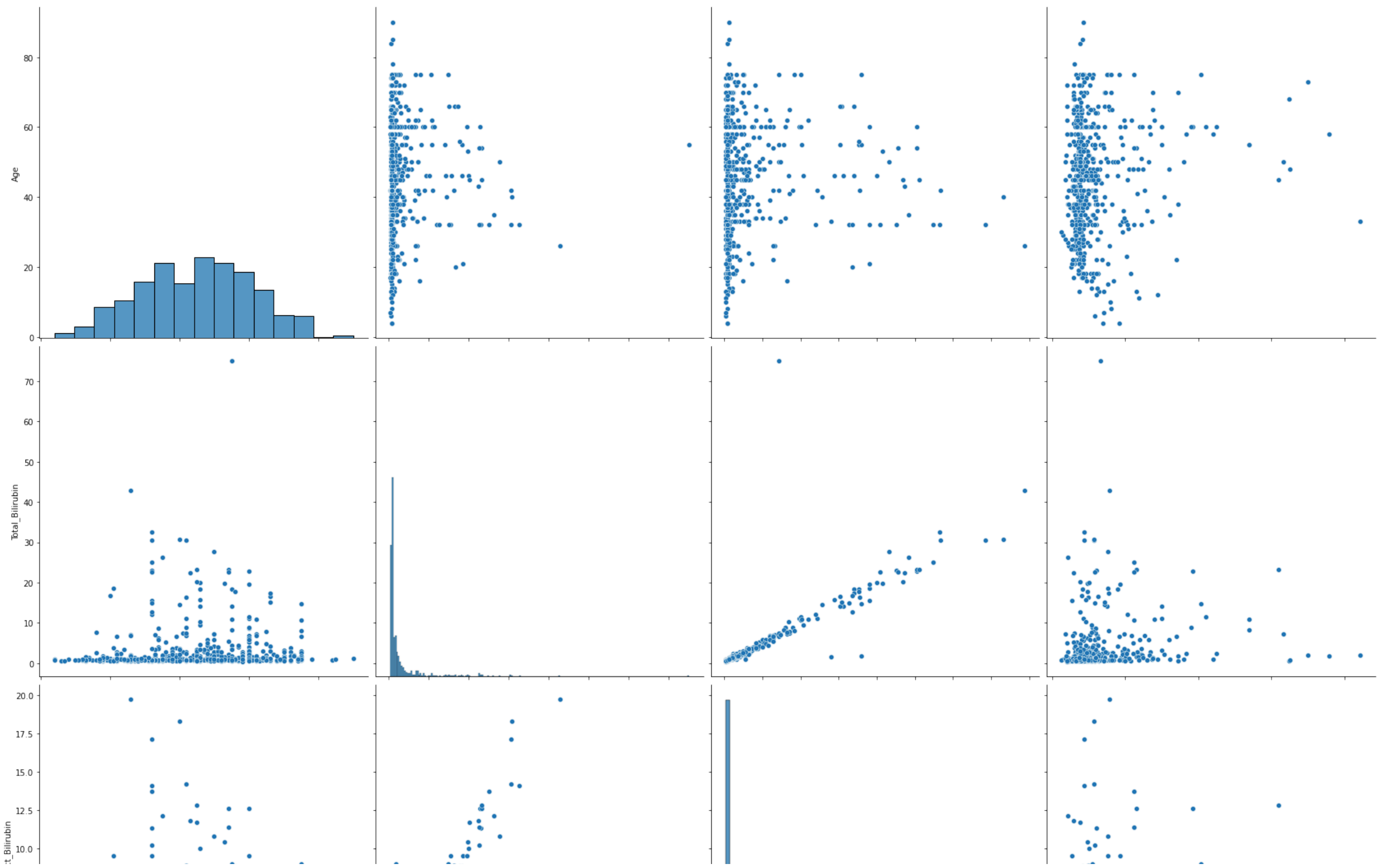

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(clf, X_test, y_test)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff5eee8dc10>
```



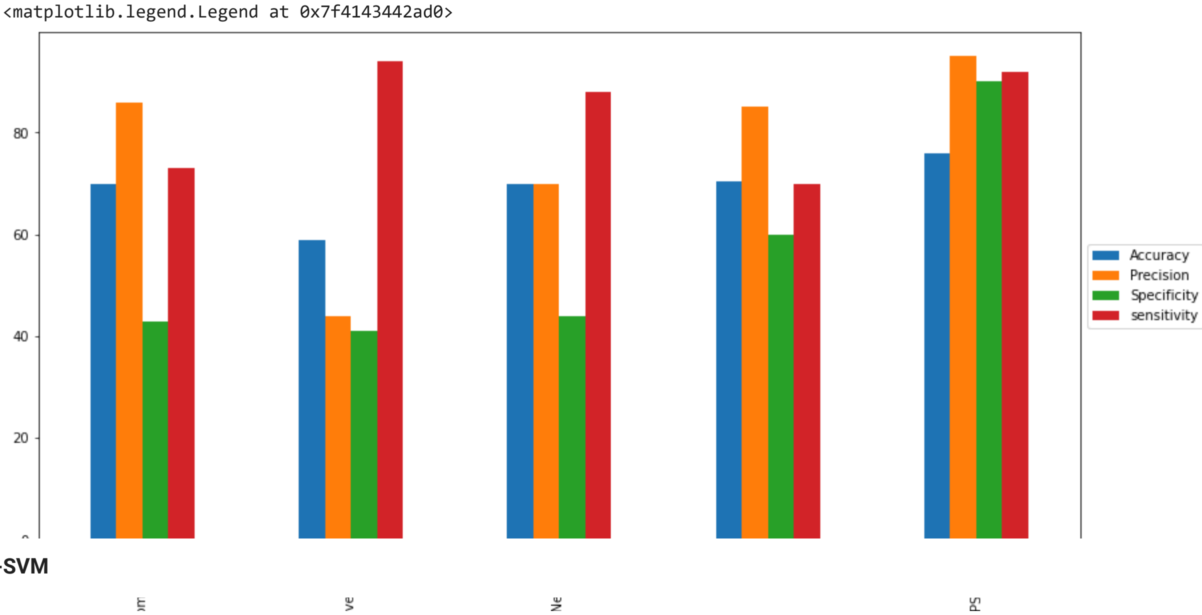
```
import pandas as pd
###Model evaluation
#We can now rank our evaluation of all the models to choose the best one for our problem.
models = pd.DataFrame({
    'Model': [ 'Random Forest' , 'Gaussian Naive Bayes', 'MLP Neural Networks','SVM','PSO-SVM'],
    'Accuracy': [ 70,59,70,70.35,76],
    'Precision':[86,44,70,85,95],
    'Specificity':[43,41,44,60,90],
    'sensitivity':[73,94,88,70,92]})
```

```
sns.pairplot(X.iloc[:, :4], height = 6)
plt.show()
```




```
import matplotlib.pyplot as plt
models.plot.bar(x="Model",figsize=(14,7))
plt.legend(loc='center left', bbox_to_anchor=(1,0.5))
```

```
<matplotlib.legend.Legend at 0x7f4143442ad0>
```



PSO-SVM

```
import numpy as np
import matplotlib.pyplot as plt
import sys
from sklearn import svm
from mpl_toolkits.mplot3d import axes3d, Axes3D
```

```
import pandas as pd
data = pd.read_csv('/content/Indian Liver Patient Dataset (ILPD).csv',names=['Age','Gender','Total_Bilirubin','Direct_Bilirubin','Alkaline_P
```

```
len(data)
```

```
583
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
data['Gender']= label_encoder.fit_transform(data['Gender'])
data['Gender'].unique()
data['Albumin_and_Globulin_Ratio'].fillna(value=data['Albumin_and_Globulin_Ratio'].mean(), inplace=True)
data = data.drop_duplicates()
print( data.shape )
data_n = data.copy()
data_n = (data - data.min())/(data.max() - data.min())
print(data_n)
```

```
(570, 11)
          Age  Gender  ...  Albumin_and_Globulin_Ratio  Dataset
0    0.709302     0.0  ...                       0.240      0.0
1    0.674419     1.0  ...                       0.176      0.0
2    0.674419     1.0  ...                       0.236      0.0
3    0.627907     1.0  ...                       0.280      0.0
4    0.790698     1.0  ...                       0.040      0.0
..        ...     ...  ...                         ...      ...
578  0.651163     1.0  ...                       0.028      1.0
579  0.418605     1.0  ...                       0.320      0.0
580  0.558140     1.0  ...                       0.280      0.0
581  0.313953     1.0  ...                       0.280      0.0
582  0.395349     1.0  ...                       0.480      1.0

[570 rows x 11 columns]
```

```
dimensions = 12
data_cn = pd.concat([data_n.shift(i) for i in range(0 + dimensions + 1)], axis = 1)

print(data_cn)
```

```
          Age  Gender  ...  Albumin_and_Globulin_Ratio  Dataset
0    0.709302     0.0  ...                         NaN      NaN
1    0.674419     1.0  ...                         NaN      NaN
2    0.674419     1.0  ...                         NaN      NaN
3    0.627907     1.0  ...                         NaN      NaN
4    0.790698     1.0  ...                         NaN      NaN
..        ...     ...  ...                         ...      ...
578  0.651163     1.0  ...                       0.280      1.0
579  0.418605     1.0  ...                       0.036      0.0
580  0.558140     1.0  ...                       0.288      0.0
581  0.313953     1.0  ...                       0.360      0.0
582  0.395349     1.0  ...                       0.240      0.0

[570 rows x 143 columns]
```

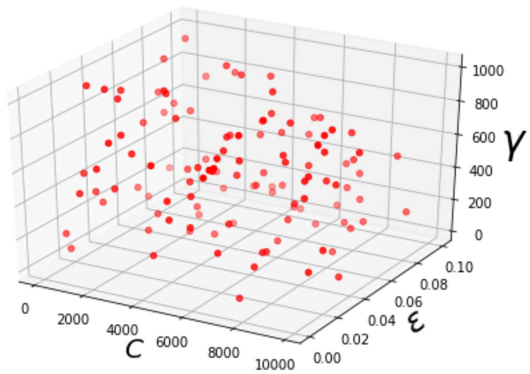
```
from sklearn.model_selection import train_test_split

x = data_cn.iloc[12:,1:]

Y = data_cn.iloc[12:,0]

x_train, x_val, Y_train, Y_val = train_test_split(x, Y, test_size = 0.4, shuffle = False)

x_val, x_test, Y_val, Y_test = train_test_split(x_val, Y_val, test_size = 0.5, shuffle = False)


print(len(Y_val))
print(len(Y_test))
print(len(Y_train))
```

```
112
112
334
```

```
def pso(n_particles, iterations, dimensions, inertia):

    # Range of SVR's hyperparameters (Particles' search space)
    # C, Epsilon and Gamma
    max_c = 1e4
    min_c = 1e-3
    max_e = 1e-1
    min_e = 1e-8
    max_g = 1e3
    min_g = 1e-3

    # Initializing particles' positions randomly, inside
    # the search space
    x = np.random.rand(n_particles, 1)*(max_c - min_c) + min_c
    y = np.random.rand(n_particles, 1)*(max_e - min_e) + min_e
    z = np.random.rand(n_particles, 1)*(max_g - min_g) + min_g

    c = np.concatenate((x,y,z), axis=1)

    # Initializing particles' parameters
    v = np.zeros((n_particles, dimensions))
    c1 = 2
    c2 = 2
    p_best = np.zeros((n_particles, dimensions))
    p_best_val = np.zeros(n_particles) + sys.maxsize
    g_best = np.zeros(dimensions)
    g_best_val = sys.maxsize

    best_iter = np.zeros(iterations)

    # Initializing regression variables
    p_best_RGS = np.empty((n_particles), dtype = object);
    g_best_RGS = sys.maxsize
    plot(c)

    from sklearn.metrics import mean_squared_error

    for i in range(iterations):

        for j in range(n_particles):
          # Starting Regression
          rgs = svm.SVR(C = c[j][0], epsilon = c[j][1], gamma = c[j][2])

          # Fitting the curve
          rgs.fit(x_train, Y_train)
          Y_predict = rgs.predict(x_val)

          # Using Mean Squared Error to verify prediction accuracy
          mse = mean_squared_error(Y_val, Y_predict)

          # If mse value for that search point, for that particle,
          # is less than its personal best point,
          # replace personal best
          if(mse < p_best_val[j]):   # mse < p_best_val[j]
              # The value below represents the current least Mean Squared Error
              p_best_val[j] = mse

              p_best_RGS[j] = rgs


              # The value below represents the current search coordinates for
              # the particle's current least Mean Squared Error found
              p_best[j] = c[j].copy()

          # Using auxiliar variable to get the index of the
          # particle that found the configuration with the
          # minimum MSE value
          aux = np.argmin(p_best_val)

          if(p_best_val[aux] < g_best_val):
              # Assigning Particle's current best MSE to the Group's best
              g_best_val = p_best_val[aux]

              # Assigning Particle's current best configuration to the Group's best
              g_best = p_best[aux].copy()

              # Group best regressor:
```

```
          # the combination of C, Epsilon and Gamma
          # that computes the best fitting curve
          g_best_RGS = p_best_RGS[aux]


      rand1 = np.random.random()
      rand2 = np.random.random()

      # The variable below influences directly the particle's velocity.
      # It can either make it smaller or bigger.
      w = inertia

      # The equation below represents Particle's velocity, which is
      # the rate of change in its position
      v[j] = w*v[j] + c1*(p_best[j] - c[j])*rand1 + c2*(g_best - c[j])*rand2

      # Change in the Particle's position
      c[j] = c[j] + v[j]

      # Below is a series of conditions that stop the particles from
      # leaving the search space
      if(c[j][2] < min_g):
        c[j][2] = min_g
      if(c[j][2] > max_g):
        c[j][2] = max_g
      if(c[j][1] < min_e):
        c[j][1] = min_e
      if(c[j][1] > max_e):
        c[j][1] = max_e
      if(c[j][0] < min_c):
        c[j][0] = min_c
      if(c[j][0] > max_c):
        c[j][0] = max_c


    # The variable below represents the least Mean Squared Error
    # of the current iteration
    best_iter[i] = g_best_val

    print('Best value iteration # %d = %f\n'%(i, g_best_val))

  # Coordinates found after all the iterations
  print('Group Best configuration found: ')
  print(g_best)
  print('\n')
  print('Best Regressor:\n')
  print(g_best_RGS)
  print('\n')
  # Displaying the MSE value variation throughout the iterations
  t = range(iterations)
  plt.plot(t, best_iter, label='Fitness Value')
  plt.legend()
  plt.show()

  # Displaying Particles' final configuration
  plot(c)

  # Making the prediction with the best configuration of C, Epsilon and
  # Gamma found by the particles
  predict_test = g_best_RGS.predict(x_test)


  # Displaying actual values and predicted values for
  # Group's best configuration found overall
  print(color.BOLD + 'Predictions with the Population Best Value found:\n' + color.END)
  evaluate(predict_test)
```

```
class color:
   PURPLE = '\033[95m'
   CYAN = '\033[96m'
   DARKCYAN = '\033[36m'
   BLUE = '\033[94m'
   GREEN = '\033[92m'
   YELLOW = '\033[93m'
   RED = '\033[91m'
   BOLD = '\033[1m'
   UNDERLINE = '\033[4m'
   END = '\033[0m'



def plot(some_list):

  ax = Axes3D(plt.figure())
  ax.scatter3D(some_list[:,0], some_list[:,1], some_list[:,2], color = 'r')
  ax.set_xlabel('$C$', fontsize = 20)
  ax.set_ylabel('$\epsilon$', fontsize = 25)
  ax.zaxis.set_rotate_label(False)
  ax.set_zlabel('$\gamma$', fontsize=30, rotation = 0)
  ax.zaxis._axinfo['label']['space_factor'] = 1.0
  plt.show()

  print('\n')
  print('\n')


def evaluate(predictions):

    from sklearn.metrics import mean_squared_error
    import statistics as st
    from sklearn.metrics import accuracy_score
    predict_test = predictions

    # To un-normalize the data:
    # Multiply the values by
    # data.to_numpy().max()

    plt.plot(range(len(Y_test)), Y_test, label='Real')
    plt.plot(range(len(predict_test)), predict_test, label='Predicted')
    plt.legend()
    plt.show()
    mse = mean_squared_error(Y_test, predict_test)
    print('\n')
    print('\n')
    print('Mean Squared Error for the Test Set:\t %f' %mse)
    print('\n')
    print('\n')
    print('Predictions Average:\t %f' %((predict_test.sum()/len(predict_test))))
    print('\n')
    print('\n')
    print('Predictions Median:\t %f' %(st.median(predict_test)))
    print('\n')
    print('\n')


pso(120, 100, 3, 1)
```




```
Best value iteration # 0 = 0.036103

Best value iteration # 1 = 0.036072

Best value iteration # 2 = 0.036072

Best value iteration # 3 = 0.036072

Best value iteration # 4 = 0.036032

Best value iteration # 5 = 0.036007

Best value iteration # 6 = 0.036007

Best value iteration # 7 = 0.036007

Best value iteration # 8 = 0.036007

Best value iteration # 9 = 0.036007

Best value iteration # 10 = 0.036007

Best value iteration # 11 = 0.036007

Best value iteration # 12 = 0.036007

Best value iteration # 13 = 0.036007

Best value iteration # 14 = 0.036007

Best value iteration # 15 = 0.036007

Best value iteration # 16 = 0.036007

Best value iteration # 17 = 0.036007
```

```
pso(120, 100, 3, 0.8)
```

```
Best value iteration # 19 = 0.036007
```

```
pso(120, 100, 3, 0.6)
```

```
pso(120, 100, 3, 0.2)
```

```
Best value iteration # 22 = 0.036007
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_val= sc.transform(x_val)
```

```
Best value iteration # 25 = 0.036007
```

```
# Import Library of Support Vector Machine model
from sklearn import svm
from sklearn.model_selection import GridSearchCV
# Create a Support Vector Classifier
pso_svc = svm.SVC()
# Hyperparameter Optimization
parameters ={'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

# Run the grid search
grid_obj = GridSearchCV(svc, parameters)
grid_obj = grid_obj.fit(x_train, Y_train)

# Set the svc to the best combination of parameters
pso_svc = grid_obj.best_estimator_

# Train the model using the training sets
pso_svc.fit(x_train,Y_train)
```

```
SVC(C=100, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
from sklearn import metrics
y_pred = svc.predict(X_test)
acc_svm = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )
from sklearn.metrics import precision_score
svm_predictions=precision_score(y_test, y_pred)
print( 'Test Accuracy of SVM :',acc_svm )
```

```
Test Accuracy of SVM :76

Best value iteration # 37 = 0.036007
```

```
import pandas as pd
import numpy as np
import random
import csv
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

liver= pd.read_csv('/content/Indian Liver Patient Dataset (ILPD).csv',names=['Age','Gender','Total_Bilirubin','Direct_Bilirubin','Alkaline_

liver_diseased=liver[liver['Dataset']==1]

liver_diseased = pd.get_dummies( liver , columns=["Gender"] , drop_first=False)

for i in range(583):
    if liver.iloc[i,10]==2:
        liver.iloc[i,10]=0

liver['Dataset'].unique()

liver['Gender'].unique()

for i in range(11):
    print(sum(liver.iloc[:,i].isnull()))

liver['Albumin_and_Globulin_Ratio'].fillna(liver['Albumin_and_Globulin_Ratio'].mean(),inplace=True)

sum(liver['Albumin_and_Globulin_Ratio'].isnull())

for i in range(583):
    if liver.iloc[i,1]=='Male':
        liver.iloc[i,1]=0
    else:
        liver.iloc[i,1]=1

# to perform hyperparameter tuning
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.model_selection import cross_val_score

# Machine Learning Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from matplotlib.colors import ListedColormap
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

# to save the final model on disk
from sklearn.externals import joblib

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(liver.drop('Dataset',axis=1))
scaled_features = scaler.transform(liver.drop('Dataset',axis=1))
liver_scaled = pd.DataFrame(scaled_features,columns=liver.columns[:-1])

X=liver_scaled
y=liver['Dataset']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)

rows=X_train.shape[0]
columns=X.shape[1]
print("No of rows in trainnig dataset : %s"%rows)
print("No of columns " ,columns)

from sklearn.metrics import classification_report,confusion_matrix

svc_linear = svm.SVC(kernel='linear', C=1)
svc_linear.fit(X_train, y_train)
predicted= svc_linear.predict(X_test)
```

```
Best value iteration # 71 = 0.036007
Best value iteration # 71 = 0.036007
Best value iteration # 72 = 0.036007
Best value iteration # 72 = 0.036007
Best value iteration # 73 = 0.036007
Best value iteration # 73 = 0.036007
Best value iteration # 74 = 0.036007
Best value iteration # 74 = 0.036007
Best value iteration # 75 = 0.036007
Best value iteration # 75 = 0.036007
Best value iteration # 76 = 0.036007
Best value iteration # 76 = 0.036007
Best value iteration # 77 = 0.036007
Best value iteration # 77 = 0.036007
Best value iteration # 78 = 0.036007
Best value iteration # 78 = 0.036007
Best value iteration # 79 = 0.036007
Best value iteration # 79 = 0.036007
Best value iteration # 80 = 0.036007
Best value iteration # 80 = 0.036007
Best value iteration # 81 = 0.036007
Best value iteration # 81 = 0.036007
Best value iteration # 82 = 0.036007
Best value iteration # 82 = 0.036007
Best value iteration # 83 = 0.036007
Best value iteration # 83 = 0.036007
Best value iteration # 84 = 0.036007
Best value iteration # 84 = 0.036007
Best value iteration # 85 = 0.036007
Best value iteration # 85 = 0.036007
Best value iteration # 86 = 0.036007
Best value iteration # 86 = 0.036007
Best value iteration # 87 = 0.036007
Best value iteration # 87 = 0.036007
Best value iteration # 88 = 0.036007
Best value iteration # 88 = 0.036007
Best value iteration # 89 = 0.036007
Best value iteration # 89 = 0.036007
Best value iteration # 90 = 0.036007
Best value iteration # 90 = 0.036007
Best value iteration # 91 = 0.036007
Best value iteration # 91 = 0.036007
```

```
pso_acc = accuracy_score(y_test,predicted)
pso_prec = precision_score(y_test,predicted)
pso_rec = recall_score(y_test,predicted)
pso_f1 = f1_score(y_test,predicted)
print("Accuracy using PSO-SVM : " , pso_acc)
print("Precision using PSO-SVM : " , pso_prec)
print("Recall using PSO-SVM : " , pso_rec)
print("F1_Score using PSO-SVM : " , pso_f1)
```

```
import random

population_size= rows
dimensions=columns
maxiter=0
fitness=0
weight_matrix=np.random.rand(rows,columns)

mapped_matrix=np.multiply(X_train,weight_matrix)
```