

NAVNEEL MONDAL

Reg No: 21BCE2654

DATE: 21-09-2023

AI ML ASSIGNMENT-4

1. Download the dataset: winequality_red.csv is downloaded.

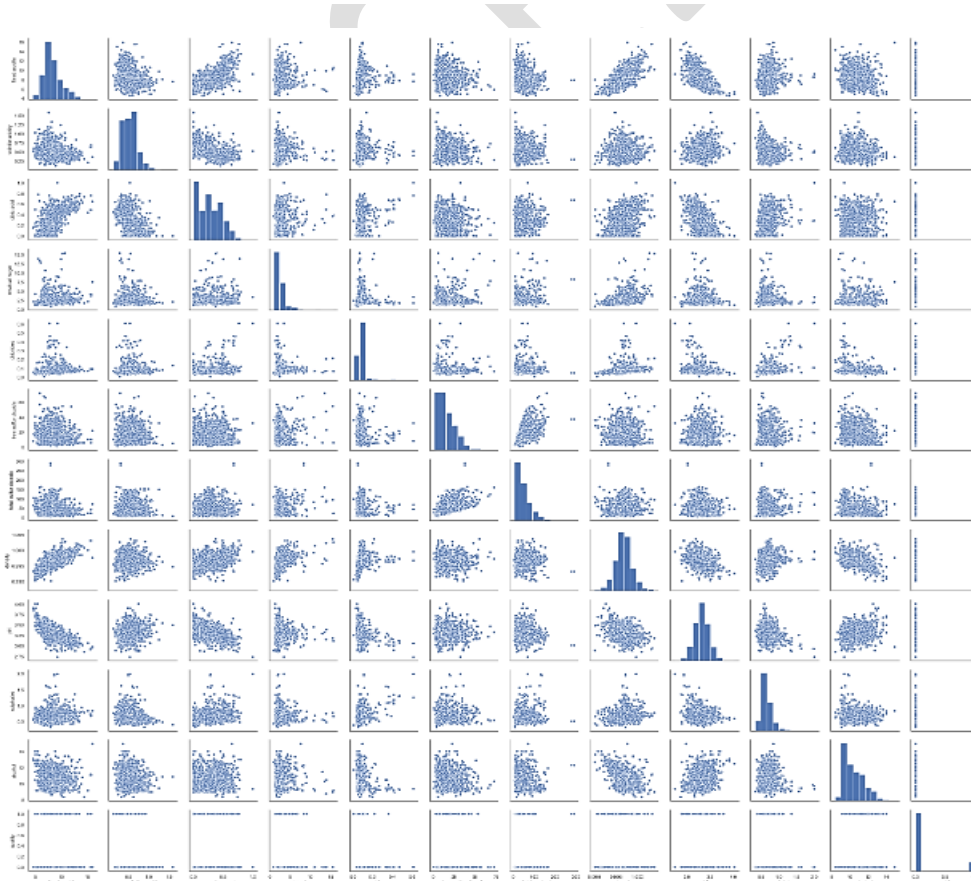
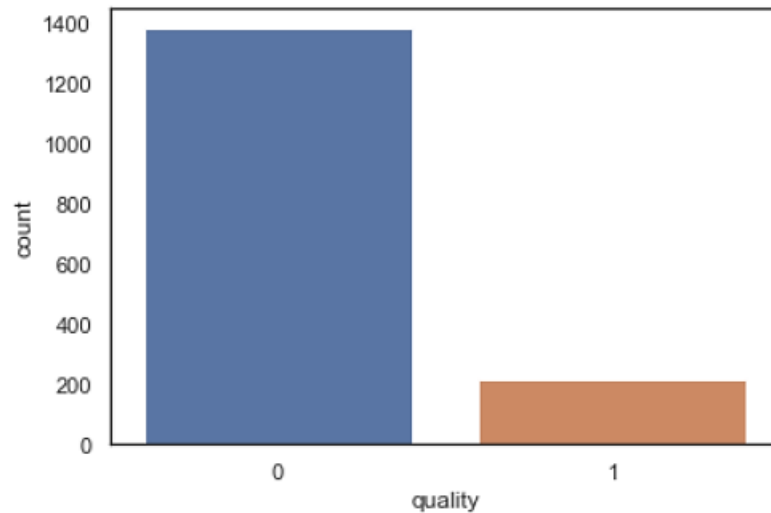
2. Load The dataset:

```
1  # Importing the libraries
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  import seaborn as sns
6  %matplotlib inline
7  import time
8  import random
9
10 random.seed(100)
11
12 # Importing the dataset
13 wine = pd.read_csv('winequality-red.csv')
14
15 wine.head()
16
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

3.Data preprocessing including visualization:

```
17 # Making binary classification for the response variable.
18 from sklearn.preprocessing import LabelEncoder
19 bins = (2, 6.5, 8)
20 group_names = ['bad', 'good']
21 wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
22 label_quality = LabelEncoder()
23 wine['quality'] = label_quality.fit_transform(wine['quality'])
24 wine['quality'].value_counts()
25
26
27 #plotting the response variable
28 sns.countplot(wine['quality'])
29
```



```

34
35 wine[wine.columns[:11]].describe()
36
37

```

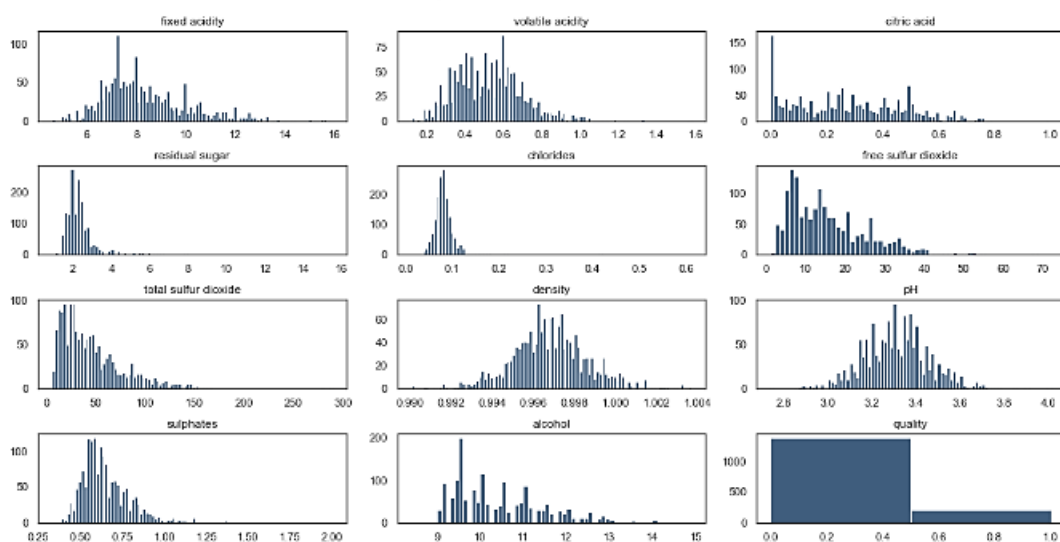
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000

```

38 ## Histograms
39 fig = plt.figure(figsize=(15, 12))
40 plt.suptitle('Histograms of Numerical Columns', fontsize=20)
41 for i in range(wine.shape[1]):
42     plt.subplot(6, 3, i + 1)
43     f = plt.gca()
44     f.set_title(wine.columns.values[i])
45
46     vals = np.size(wine.iloc[:, i].unique())
47     if vals >= 100:
48         vals = 100
49
50     plt.hist(wine.iloc[:, i], bins=vals, color='#3F5D7D')
51 plt.tight_layout(rect=[0, 0.03, 1, 0.95])
52

```

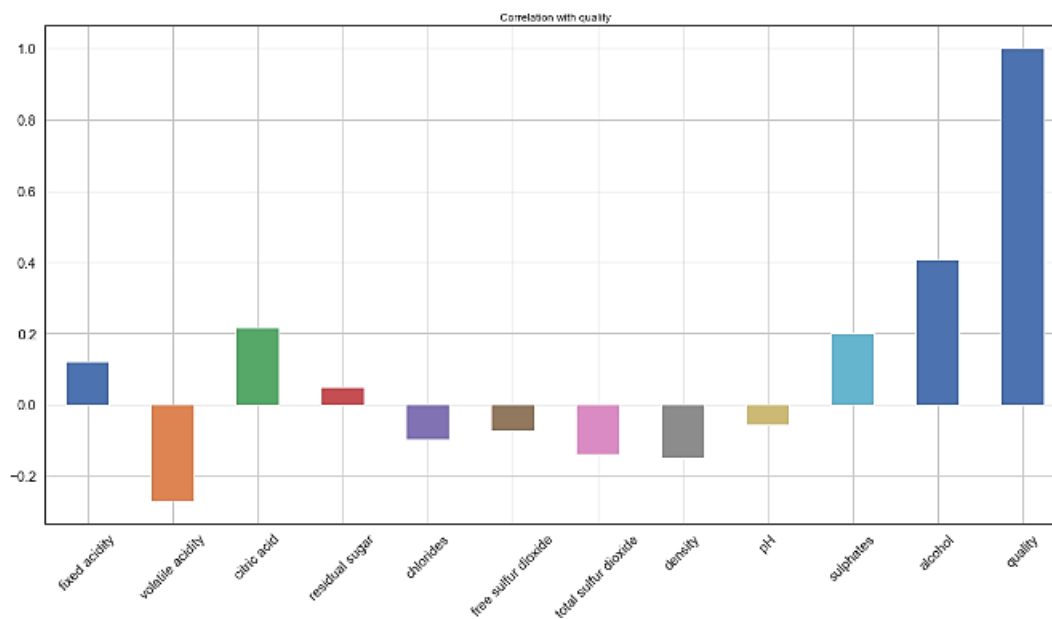
Histograms of Numerical Columns



```
52
53
54 wine.isna().any()
55
```

```
fixed acidity      False
volatile acidity   False
citric acid        False
residual sugar     False
chlorides          False
free sulfur dioxide False
total sulfur dioxide False
density           False
pH                False
sulphates          False
alcohol           False
quality           False
dtype: bool
```

```
55
56 #Correlation with Quality with respect to attributes
57 wine.corrwith(wine.quality).plot.bar(
58     figsize = (20, 10), title = "Correlation with quality", fontsize = 15,
59     rot = 45, grid = True)
60
```



```

62     ## Correlation Matrix
63
64     sns.set(style="white")
65
66     # Compute the correlation matrix
67     corr = wine.corr()
68
69     corr.head()

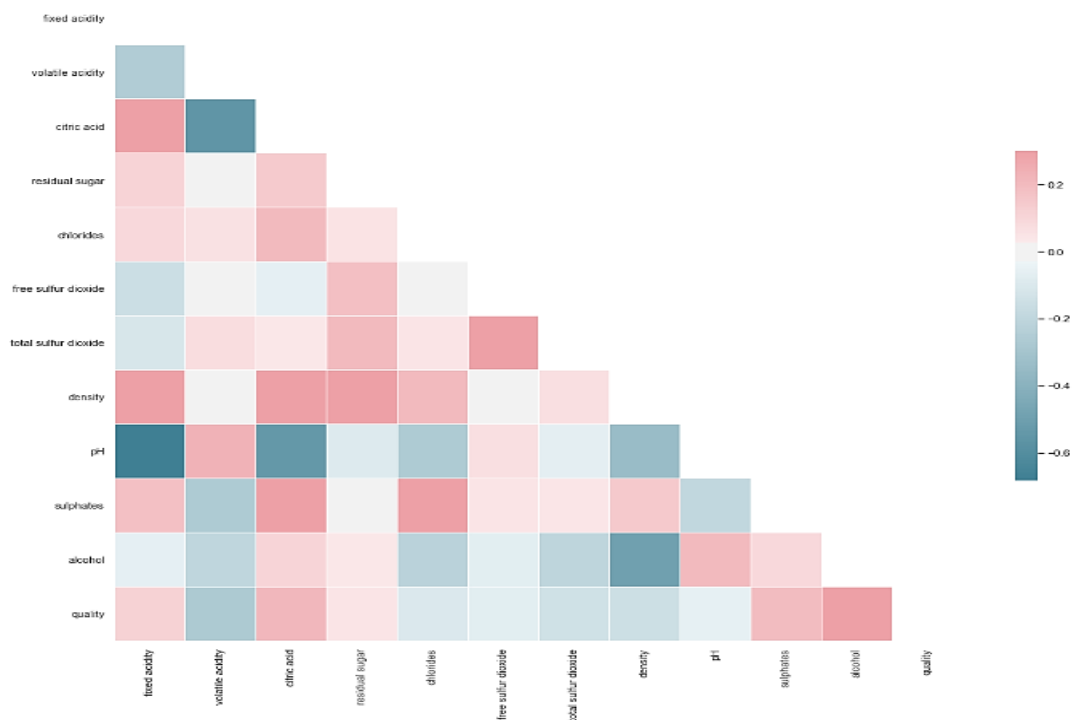
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978	0.183006	-0.061668	0.120061
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937	-0.260987	-0.202288	-0.270712
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904	0.312770	0.109903	0.214716
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652	0.005527	0.042075	0.047779
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026	0.371260	-0.221141	-0.097308

```

70
71     # Generate a mask for the upper triangle
72     mask = np.zeros_like(corr, dtype=np.bool)
73     mask[np.triu_indices_from(mask)] = True
74
75     # Set up the matplotlib figure
76     f, ax = plt.subplots(figsize=(18, 15))
77
78     # Generate a custom diverging colormap
79     cmap = sns.diverging_palette(220, 10, as_cmap=True)
80
81     # Draw the heatmap with the mask and correct aspect ratio
82     sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
83                 square=True, linewidths=.5, cbar_kws={"shrink": .5})

```



4. Machine Learning Model building:

```
85 #Assigning and dividing the dataset
86 X = wine.drop('quality',axis=1)
87 y=wine['quality']
88 X.head()
89
90 y.head()
91
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

```
90 y.head()
91
92 wine.columns[:11]
93
94
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
      'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
      'pH', 'sulphates', 'alcohol'],
      dtype='object')
```

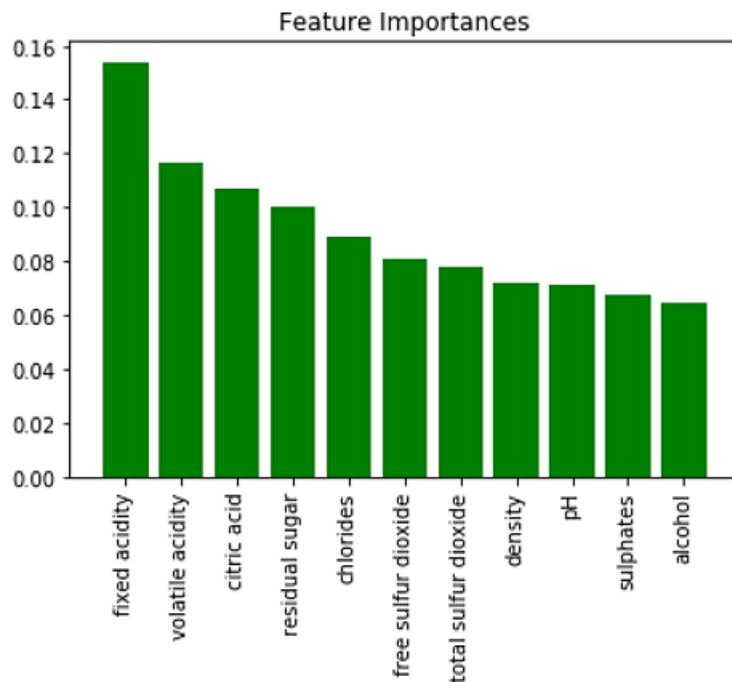
```
95 Fitting Random Forest Classification to the Training set
96 from sklearn.ensemble import RandomForestClassifier
97 classifier = RandomForestClassifier(n_estimators = 200, criterion = 'entropy', random_state = 0)
98 classifier.fit(X, y)
99 importances = classifier.feature_importances_
100 indices = np.argsort(importances)[::-1]
101 for i in range(X.shape[1]):
102     print ("%2d) %-*s %f" % (i + 1, 30, features_label[i],importances[indices[i]]))
103
104
```

```
1) fixed acidity          0.154052
2) volatile acidity       0.116418
3) citric acid            0.107016
4) residual sugar         0.099913
5) chlorides              0.089030
6) free sulfur dioxide    0.080517
7) total sulfur dioxide   0.077752
8) density                0.072258
9) pH                    0.071140
10) sulphates             0.067320
11) alcohol               0.064583
```

```

105 title('Feature Importances')
106 plt.bar(range(X.shape[1]),importances[indices], color="green", align="center")
107 plt.xticks(range(X.shape[1]),features_label, rotation=90)
108 plt.xlim([-1, X.shape[1]])
109 plt.show()
110
111

```



```

112 # Splitting the dataset into the Training set and Test set
113 from sklearn.model_selection import train_test_split
114 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 5)
115
116 # Feature Scaling
117 from sklearn.preprocessing import StandardScaler
118 sc = StandardScaler()
119 X_train2 = pd.DataFrame(sc.fit_transform(X_train))
120 X_test2 = pd.DataFrame(sc.transform(X_test))
121 X_train2.columns = X_train.columns.values
122 X_test2.columns = X_test.columns.values
123 X_train2.index = X_train.index.values
124 X_test2.index = X_test.index.values
125 X_train = X_train2
126 X_test = X_test2
127
128 #Using Principal Dimensional Reduction
129 from sklearn.decomposition import PCA
130 pca = PCA(n_components = 4)
131 X_train = pca.fit_transform(X_train)
132 X_test = pca.transform(X_test)
133 explained_variance = pca.explained_variance_ratio_
134 print(pd.DataFrame(explained_variance))
135

```

```

0
0 0.281687
1 0.171462
2 0.143245
3 0.114765

```

```

136 ##### Model Building #####
137
138 ### Comparing Models
139
140 ## Logistic Regression
141 from sklearn.linear_model import LogisticRegression
142 classifier = LogisticRegression(random_state = 0, penalty = 'l1')
143 classifier.fit(X_train, y_train)
144
145 # Predicting Test Set
146 y_pred = classifier.predict(X_test)
147 from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
148 acc = accuracy_score(y_test, y_pred)
149 prec = precision_score(y_test, y_pred)
150 rec = recall_score(y_test, y_pred)
151 f1 = f1_score(y_test, y_pred)
152
153 results = pd.DataFrame([['Logistic Regression', acc, prec, rec, f1]],
154                        columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
155 print(results)
156
157

```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.86875	0.6	0.26087	0.363636

```

158 ## SVM (Linear)
159 from sklearn.svm import SVC
160 classifier = SVC(random_state = 0, kernel = 'linear')
161 classifier.fit(X_train, y_train)
162
163 # Predicting Test Set
164 y_pred = classifier.predict(X_test)
165 acc = accuracy_score(y_test, y_pred)
166 prec = precision_score(y_test, y_pred)
167 rec = recall_score(y_test, y_pred)
168 f1 = f1_score(y_test, y_pred)
169
170 model_results = pd.DataFrame([['SVM (Linear)', acc, prec, rec, f1]],
171                             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
172
173 results = results.append(model_results, ignore_index = True)
174 print(results)
175
176

```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.86875	0.6	0.26087	0.363636
1	SVM (Linear)	0.85625	0.0	0.00000	0.000000

```

177 ## SVM (rbf)
178 from sklearn.svm import SVC
179 classifier = SVC(random_state = 0, kernel = 'rbf')
180 classifier.fit(X_train, y_train)
181
182 # Predicting Test Set
183 y_pred = classifier.predict(X_test)
184 acc = accuracy_score(y_test, y_pred)
185 prec = precision_score(y_test, y_pred)
186 rec = recall_score(y_test, y_pred)
187 f1 = f1_score(y_test, y_pred)
188
189 model_results = pd.DataFrame([['SVM (RBF)', acc, prec, rec, f1]],
190                             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
191
192 results = results.append(model_results, ignore_index = True)
193 print(results)
194

```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.86875	0.600000	0.260870	0.363636
1	SVM (Linear)	0.85625	0.000000	0.000000	0.000000
2	SVM (RBF)	0.87500	0.714286	0.217391	0.333333


```

195 ## Randomforest
196 from sklearn.ensemble import RandomForestClassifier
197 classifier = RandomForestClassifier(random_state = 0, n_estimators = 100,
198                                   criterion = 'entropy')
199 classifier.fit(X_train, y_train)
200
201 # Predicting Test Set
202 y_pred = classifier.predict(X_test)
203 acc = accuracy_score(y_test, y_pred)
204 prec = precision_score(y_test, y_pred)
205 rec = recall_score(y_test, y_pred)
206 f1 = f1_score(y_test, y_pred)
207
208 model_results = pd.DataFrame([['Random Forest (n=100)', acc, prec, rec, f1]],
209                               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
210
211 results = results.append(model_results, ignore_index = True)
212 print(results)
213

```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.868750	0.600000	0.260870	0.363636
1	SVM (Linear)	0.856250	0.000000	0.000000	0.000000
2	SVM (RBF)	0.875000	0.714286	0.217391	0.333333
3	Random Forest (n=100)	0.903125	0.741935	0.500000	0.597403

5. Evaluate the model:

```
214
215 ## K-fold Cross Validation
216 from sklearn.model_selection import cross_val_score
217 accuracies = cross_val_score(estimator = classifier, X= X_train, y = y_train,
218                             cv = 10)
219 print("Random Forest Classifier Accuracy: %0.2f (+/- %0.2f)" % (accuracies.mean(), accuracies.std() * 2))
220
```

Random Forest Classifier Accuracy: 0.90 (+/- 0.05)

```
220
221 # Applying Grid Search
222
223 # Round 1: Entropy
224 parameters = {"max_depth": [3, None],
225              'min_samples_split': [2, 5, 10],
226              'min_samples_leaf': [1, 5, 10],
227              "bootstrap": [True, False],
228              "criterion": ["entropy"]}
229
230
231 from sklearn.model_selection import GridSearchCV
232 grid_search = GridSearchCV(estimator = classifier, # Make sure classifier points to the RF model
233                           param_grid = parameters,
234                           scoring = "accuracy",
235                           cv = 10,
236                           n_jobs = -1)
237
238 t0 = time.time()
239 grid_search = grid_search.fit(X_train, y_train)
240 t1 = time.time()
241 print("Took %0.2f seconds" % (t1 - t0))
242
243 rf_best_accuracy = grid_search.best_score_
244 rf_best_parameters = grid_search.best_params_
245 rf_best_accuracy, rf_best_parameters
246
```

Took 80.70 seconds

```
(0.8999218139171228,
 {'bootstrap': False,
  'criterion': 'entropy',
  'max_depth': None,
  'min_samples_leaf': 1,
  'min_samples_split': 2})
```

```
246
247 # Round 2: Entropy
248 parameters = {"max_depth": [None],
249              'min_samples_split': [8, 10, 12],
250              'min_samples_leaf': [1, 2, 3],
251              "bootstrap": [True],
252              "criterion": ["entropy"]}
253
254
255 from sklearn.model_selection import GridSearchCV
256 grid_search = GridSearchCV(estimator = classifier, # Make sure classifier points to the RF model
257                           param_grid = parameters,
258                           scoring = "accuracy",
259                           cv = 10,
260                           n_jobs = -1)
261
262 t0 = time.time()
263 grid_search = grid_search.fit(X_train, y_train)
264 t1 = time.time()
265 print("Took %0.2f seconds" % (t1 - t0))
266
267 rf_best_accuracy = grid_search.best_score_
268 rf_best_parameters = grid_search.best_params_
269 rf_best_accuracy, rf_best_parameters
270
```

Took 37.32 seconds

```
(0.8866301798279906,
 {'bootstrap': True,
  'criterion': 'entropy',
  'max_depth': None,
  'min_samples_leaf': 1,
  'min_samples_split': 8})
```

```

271 # Round 1: Gini
272 parameters = {"max_depth": [3, None],
273               'min_samples_split': [2, 5, 10],
274               'min_samples_leaf': [1, 5, 10],
275               "bootstrap": [True, False],
276               "criterion": ["gini"]}
277
278 # Make sure classifier points to the RF model
279 from sklearn.model_selection import GridSearchCV
280 grid_search = GridSearchCV(estimator = classifier,
281                             param_grid = parameters,
282                             scoring = "accuracy",
283                             cv = 10,
284                             n_jobs = -1)
285
286 t0 = time.time()
287 grid_search = grid_search.fit(X_train, y_train)
288 t1 = time.time()
289 print("Took %.2f seconds" % (t1 - t0))
290
291 rf_best_accuracy = grid_search.best_score_
292 rf_best_parameters = grid_search.best_params_
293 rf_best_accuracy, rf_best_parameters
294
295

```

Took 62.63 seconds

```

(0.9046129788897577,
 {'bootstrap': True,
  'criterion': 'gini',
  'max_depth': None,
  'min_samples_leaf': 1,
  'min_samples_split': 2})

```

```

296 # Round 2: Gini
297 parameters = {"max_depth": [None],
298               'min_samples_split': [2, 3, 4],
299               'min_samples_leaf': [8, 10, 12],
300               "bootstrap": [True],
301               "criterion": ["gini"]}
302
303
304 from sklearn.model_selection import GridSearchCV
305 grid_search = GridSearchCV(estimator = classifier, # Make sure classifier points to the RF model
306                             param_grid = parameters,
307                             scoring = "accuracy",
308                             cv = 10,
309                             n_jobs = -1)
310
311 t0 = time.time()
312 grid_search = grid_search.fit(X_train, y_train)
313 t1 = time.time()
314 print("Took %.2f seconds" % (t1 - t0))
315
316 rf_best_accuracy = grid_search.best_score_
317 rf_best_parameters = grid_search.best_params_
318 rf_best_accuracy, rf_best_parameters
319

```

Took 29.75 seconds

```

(0.8772478498827209,
 {'bootstrap': True,
  'criterion': 'gini',
  'max_depth': None,
  'min_samples_leaf': 8,
  'min_samples_split': 2})

```

```
319  
320 rf_best_accuracy, rf_best_parameters  
321
```

```
(0.9046129788897577,  
{ 'bootstrap': True,  
  'criterion': 'gini',  
  'max_depth': None,  
  'min_samples_leaf': 1,  
  'min_samples_split': 2})
```

21BCE2654

6.Prediction results:

```
321
322
323 # Predicting Test Set
324 y_pred = grid_search.predict(X_test)
325 acc = accuracy_score(y_test, y_pred)
326 prec = precision_score(y_test, y_pred)
327 rec = recall_score(y_test, y_pred)
328 f1 = f1_score(y_test, y_pred)
329
330 model_results = pd.DataFrame([['Random Forest (n=100, GSx2 + Gini)', acc, prec, rec, f1]],
331                             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score'])
332
333 results = results.append(model_results, ignore_index = True)
334 results
335
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Logistic Regression	0.868750	0.600000	0.260870	0.363636
1	SVM (Linear)	0.856250	0.000000	0.000000	0.000000
2	SVM (RBF)	0.875000	0.714286	0.217391	0.333333
3	Random Forest (n=100)	0.903125	0.741935	0.500000	0.597403
4	Random Forest (n=100, GSx2 + Gini)	0.909375	0.757576	0.543478	0.632911
5	Random Forest (n=100, GSx2 + Gini)	0.909375	0.757576	0.543478	0.632911