American International University-Bangladesh (AIUB)

Department of Computer Science

Faculty of Science &Technology (FST)

Spring 22-23

Python Final Project

Programming in Python

Sec: **B**

Project submitted

By

Name	ID
SHOPNAMOY SAHA PRONOY	19-41199-2
JOYDEP DHAR	20-44237-3
RIDAN RABAB MAJUMDER	19-40314-1
NAIMUR RAHMAN	19-40397-1

Submitted to

DR. AKINUL ISLAM JONY
Associate Professor & Head (UG), Computer Science
American International University-Bangladesh

Table Of Content

1. Project overview

This project involves building a classification-based model using Python and a real dataset containing a minimum of 2000 instances. The dataset will be preprocessed, including data cleaning, and then subjected to exploratory data analysis.

The project will employ five classification models, namely Naive Bayes, K-Nearest Neighbors (KNN), Decision Tree, Logistic Regression, and Support Vector Machine (SVM). Each of these models will be trained using the preprocessed dataset, and their respective performances will be evaluated. The evaluation will be based on predictive accuracy, and the comparison will provide insights into which classifier is the most effective for this particular dataset.

Overall, the project aims to develop a classification model that can predict the target variable accurately, using the selected classifiers, and provide insights into which classifier works best for this dataset.

2. Dataset overview

2.1. Data source Url: https://data.world/makeovermonday/2021w14

2.2. Data set description:

Images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. A total of 16 features; 12 dimensions and 4 shape forms, were obtained from the grains.

Does the data set contain missing values?

No

Number of Instances (records in this data set):

13611

Number of Attributes (fields within each record):

17

Attribute Information:

1.) Area (A): The area of a bean zone and the number of pixels within its boundaries.

2.) Perimeter (P): Bean circumference is defined as the length of its border.

3.) Major axis length (L): The distance between the ends of the longest line that can be drawn from a bean.

4.) Minor axis length (l): The longest line that can be drawn from the bean while standing perpendicular to the main axis.

5.) Aspect ratio (K): Defines the relationship between L and l.

6.) Eccentricity (Ec): Eccentricity of the ellipse having the same moments as the region.

7.) Convex area (C): Number of pixels in the smallest convex polygon that can contain the area of a bean seed.

8.) Equivalent diameter (Ed): The diameter of a circle having the same area as a bean seed area.

9.) Extent (Ex): The ratio of the pixels in the bounding box to the bean area.

10.) Solidity (S): Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.

11.) Roundness (R): Calculated with the following formula: (4piA)/(P^2)

12.) Compactness (CO): Measures the roundness of an object: Ed/L

13.) ShapeFactor1 (SF1)

14.) ShapeFactor2 (SF2)

15.) ShapeFactor3 (SF3)

16.) ShapeFactor4 (SF4)

17.) Class (Seker, Barbunya, Bombay, Cali, Dermosan, Horoz and Sira)

The data set is too large to show all rows. Here are the first 20 rows below.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Bean ID	Area	Perimeter	MajorAxisl	MinorAxisl	AspectRati	Eccentricit	ConvexAre	EquivDiam	Extent	Solidity	roundness	Compactn	ShapeFact	ShapeFact	ShapeFact	ShapeFact	Class
2	1	28395	610.291	208.1781	173.8887	1.197191	0.549812	28715	190.1411	0.763923	0.988856	0.958027	0.913358	0.007332	0.003147	0.834222	0.998724	SEKER
3	2	28734	638.018	200.5248	182.7344	1.097356	0.411785	29172	191.2728	0.783968	0.984986	0.887034	0.953861	0.006979	0.003564	0.909851	0.99843	SEKER
4	3	29380	624.11	212.8261	175.9311	1.209713	0.562727	29690	193.4109	0.778113	0.989559	0.947849	0.908774	0.007244	0.003048	0.825871	0.999066	SEKER
5	4	30008	645.884	210.558	182.5165	1.153638	0.498616	30724	195.4671	0.782681	0.976696	0.903936	0.928329	0.007017	0.003215	0.861794	0.994199	SEKER
6	5	30140	620.134	201.8479	190.2793	1.060798	0.33368	30417	195.8965	0.773098	0.990893	0.984877	0.970516	0.006697	0.003665	0.9419	0.999166	SEKER
7	6	30279	634.927	212.5606	181.5102	1.171067	0.520401	30600	196.3477	0.775688	0.98951	0.943852	0.923726	0.00702	0.003153	0.85327	0.999236	SEKER
8	7	30477	670.033	211.0502	184.0391	1.146768	0.489478	30970	196.9886	0.762402	0.984081	0.85308	0.933374	0.006925	0.003242	0.871186	0.999049	SEKER
9	8	30519	629.727	212.9968	182.7372	1.165591	0.51376	30847	197.1243	0.770682	0.989367	0.967109	0.92548	0.006979	0.003158	0.856514	0.998345	SEKER
10	9	30685	635.681	213.5341	183.1571	1.165852	0.514081	31044	197.6597	0.771561	0.988436	0.95424	0.925658	0.006959	0.003152	0.856844	0.998953	SEKER
11	10	30834	631.934	217.2278	180.8975	1.200834	0.553642	31120	198.139	0.783683	0.99081	0.970278	0.912125	0.007045	0.003008	0.831973	0.999061	SEKER
12	11	30917	640.765	213.5601	184.4399	1.157885	0.504102	31280	198.4055	0.770805	0.988395	0.946258	0.929038	0.006908	0.003174	0.863112	0.999384	SEKER
13	12	31091	638.558	210.4863	188.3268	1.117665	0.446622	31458	198.963	0.786377	0.988334	0.958173	0.945254	0.00677	0.003334	0.893506	0.99864	SEKER
14	13	31107	640.594	214.6485	184.9693	1.160455	0.507366	31423	199.0142	0.761046	0.989944	0.952582	0.927163	0.0069	0.003145	0.859632	0.997564	SEKER
15	14	31158	642.626	216.4848	183.6443	1.178827	0.529514	31492	199.1773	0.798759	0.989394	0.948119	0.920052	0.006948	0.003071	0.846496	0.997872	SEKER
16	15	31158	641.105	212.067	187.193	1.132879	0.469924	31474	199.1773	0.781313	0.98996	0.952623	0.939219	0.006806	0.003267	0.882132	0.999349	SEKER
17	16	31178	636.888	212.9759	186.5621	1.141582	0.482352	31520	199.2412	0.76411	0.98915	0.9659	0.935511	0.006831	0.003227	0.87518	0.99909	SEKER
18	17	31202	644.454	215.6407	184.4717	1.168964	0.517871	31573	199.3179	0.779193	0.988249	0.944079	0.924306	0.006911	0.003112	0.854341	0.998693	SEKER
19	18	31203	639.782	215.0677	184.8749	1.163315	0.510947	31558	199.3211	0.762984	0.988751	0.957949	0.926783	0.006893	0.003137	0.858926	0.999202	SEKER
20	19	31272	638.666	212.4503	187.5359	1.132851	0.469883	31593	199.5413	0.770322	0.98984	0.963425	0.939238	0.006794	0.003261	0.882167	0.999364	SEKER

Information of the DataFrame.

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 13611 entries, 1 to 13611
Data columns (total 17 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Area             13611 non-null  int64
 1   Perimeter        13611 non-null  float64
 2   MajorAxisLength  13611 non-null  float64
 3   MinorAxisLength  13611 non-null  float64
 4   AspectRation     13611 non-null  float64
 5   Eccentricity     13611 non-null  float64
 6   ConvexArea       13611 non-null  int64
 7   EquivDiameter    13611 non-null  float64
 8   Extent           13611 non-null  float64
 9   Solidity         13611 non-null  float64
 10  roundness        13611 non-null  float64
 11  Compactness      13611 non-null  float64
 12  ShapeFactor1     13611 non-null  float64
 13  ShapeFactor2     13611 non-null  float64
 14  ShapeFactor3     13611 non-null  float64
 15  ShapeFactor4     13611 non-null  float64
 16  Class            13611 non-null  object
dtypes: float64(14), int64(2), object(1)
memory usage: 1.9+ MB
```

3. Data Preprocessing and Exploratory data analysis

Here by using the groupby and describe methods we can analyze the data and see that how many instances each class has. We can also see count, min and max values etc.

In [4]:
```
data.groupby('Class').size()
```

Out[4]:
```
Class
BARBUNYA    1322
BOMBAY       522
CALI        1630
DERMASON    3546
HOROZ       1928
SEKER       2027
SIRA        2636
dtype: int64
```

In [6]:
```
data.describe()
```

Out[6]:

	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity
count	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000	13611.000000
mean	53048.284549	855.283459	320.141867	202.270714	1.583242	0.750895	53768.200206	253.064220	0.749733	0.987143
std	29324.095717	214.289696	85.694186	44.970091	0.246678	0.092002	29774.915817	59.177120	0.049086	0.004660
min	20420.000000	524.736000	183.601165	122.512653	1.024868	0.218951	20684.000000	161.243764	0.555315	0.919246
25%	36328.000000	703.523500	253.303633	175.848170	1.432307	0.715928	36714.500000	215.068003	0.718634	0.985670
50%	44652.000000	794.941000	296.883367	192.431733	1.551124	0.764441	45178.000000	238.438026	0.759859	0.988283
75%	61332.000000	977.213000	376.495012	217.031741	1.707109	0.810466	62294.000000	279.446467	0.786851	0.990013
max	254616.000000	1985.370000	738.860153	460.198497	2.430306	0.911423	263261.000000	569.374358	0.866195	0.994677

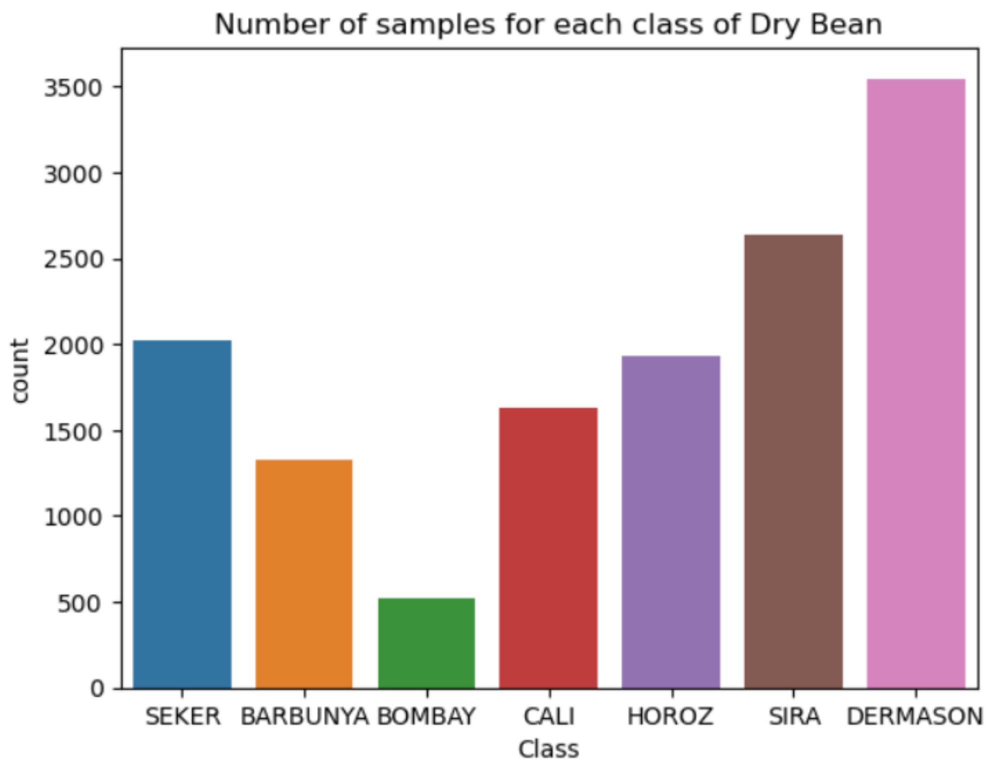
Then we make a copy of the dataset and remove the class attribute.

```
In [5]: temp_df = data.copy()
        temp_df['Class'] = pd.to_numeric(temp_df['Class'], errors='coerce')
        temp_df
```

We also plot the number of samples each class has.

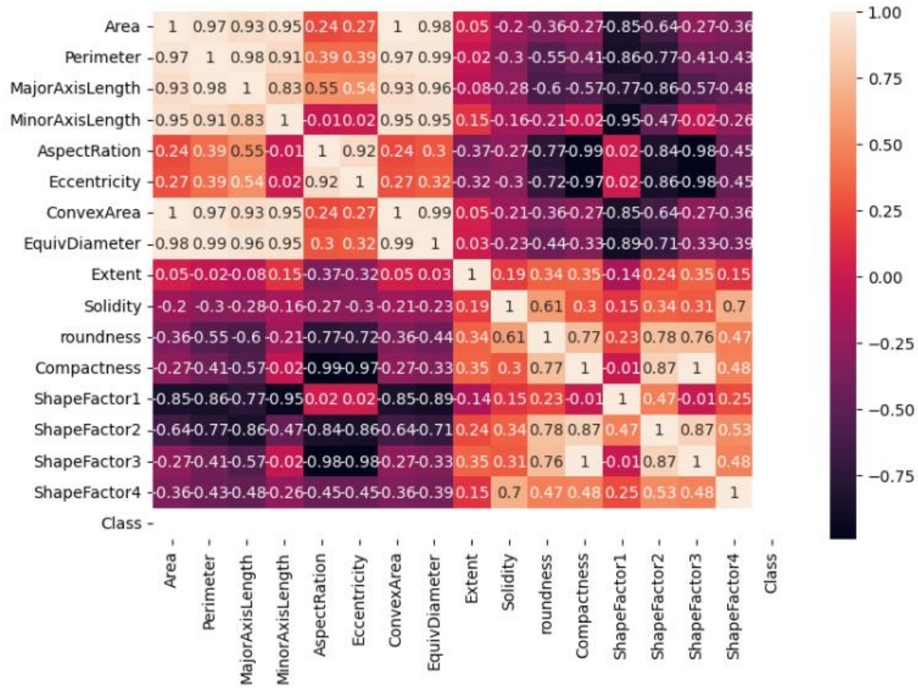
```
In [10]: sns.countplot(x='Class', data=data)
         plt.title("Number of samples for each class of Dry Bean")

Out[10]: Text(0.5, 1.0, 'Number of samples for each class of Dry Bean')
```



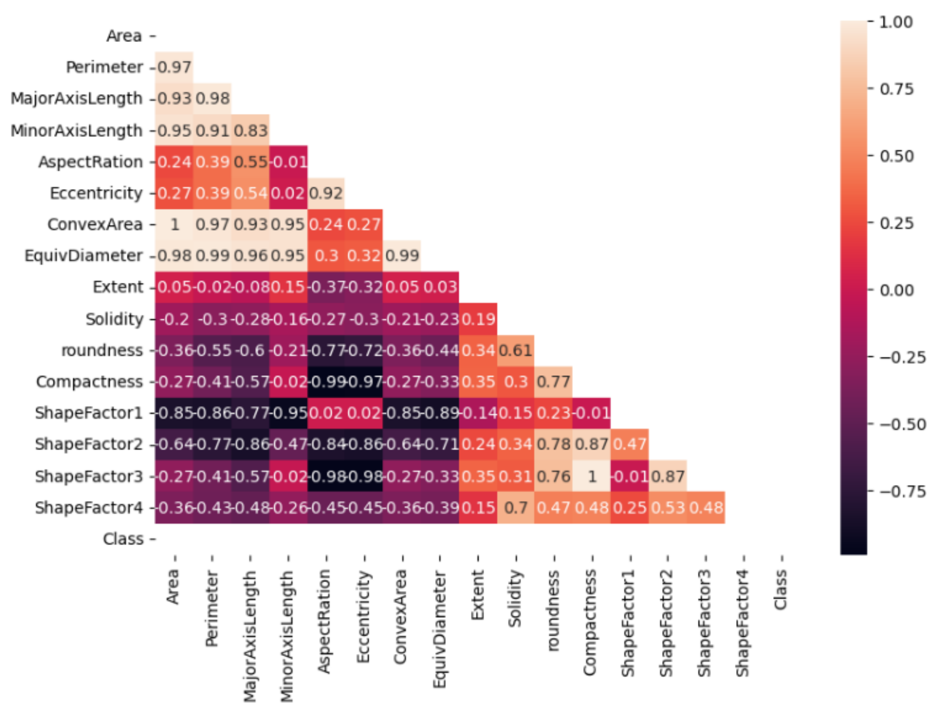
We then find the correlation matrix and make a heatmap out of it. From the correlation matrix we see that area, perimeter, MajorAxisLength, MinorAxisLength, ConvexArea and EquivDiameter has very good relationship. The other values are not very related. So, to make our model efficient we can select these attributes which are related and discard the other attribute as they will not affect in training the models.

In [7]:
```python
# correlation finding
corr_matrix = temp_df.corr().round(2)
plt.figure(figsize = (9, 6))
sns.heatmap(data=corr_matrix, annot=True);
```



In [12]:
```python
mask = np.zeros_like(corr_matrix)
mask[np.triu_indices_from(mask)] = True
plt.figure(figsize = (9, 6))
sns.heatmap(data=corr_matrix, annot=True, mask=mask)
```

Out[12]: <Axes: >



4. Model development

To build a model we have to prepare the data first. We keep the relevant attributes which will be used for training in separate data frame x.

```
In [26]: X = data[['Area', 'Perimeter', 'MajorAxisLength', 'MinorAxisLength', 'ConvexArea', 'EquivDiameter']]
         X
```

Out[26]:

Bean ID	Area	Perimeter	MajorAxisLength	MinorAxisLength	ConvexArea	EquivDiameter
1	28395	610.291	208.178117	173.888747	28715	190.141097
2	28734	638.018	200.524796	182.734419	29172	191.272750
3	29380	624.110	212.826130	175.931143	29690	193.410904
4	30008	645.884	210.557999	182.516516	30724	195.467062
5	30140	620.134	201.847882	190.279279	30417	195.896503
...	...	...	...	...	...	...
13607	42097	759.696	288.721612	185.944705	42508	231.515799
13608	42101	757.499	281.576392	190.713136	42494	231.526798
13609	42139	759.321	281.539928	191.187979	42569	231.631261
13610	42147	763.779	283.382636	190.275731	42667	231.653248
13611	42159	772.237	295.142741	182.204716	42600	231.686223

13611 rows × 6 columns

We also keep the Class attribute which are result in y.

```
In [22]: y = data['Class']
         y
```

```
Out[22]: Bean ID
         1            SEKER
         2            SEKER
         3            SEKER
         4            SEKER
         5            SEKER
                      ...
         13607     DERMASON
         13608     DERMASON
         13609     DERMASON
         13610     DERMASON
         13611     DERMASON
         Name: Class, Length: 13611, dtype: object
```

After that we split the dataset for training and testing. In this project we kept train/test ration 80/20.

```
In [29]: from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 16)

         print("X_train shape: ", X_train.shape)
         print("X_test shape: ", X_test.shape)
         print("y_train shape: ", y_train.shape)
         print("y_test shape: ", y_test.shape)

         X_train shape:  (10888, 6)
         X_test shape:  (2723, 6)
         y_train shape:  (10888,)
         y_test shape:  (2723,)
```

Then we applied different models to train and predict values.

Here we used SVM model. In our case svm is not performing very well in predicting accuracy

```
In [66]: #Model SVM

In [30]: from sklearn import svm
         from sklearn import metrics

         model_svm = svm.SVC()
         model_svm.fit(X_train, y_train)
         y_prediction_svm = model_svm.predict(X_test)

         score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
         print("----------------------------------")
         print('The accuracy of the SVM is: {}'.format(score_svm))
         print("----------------------------------")

         score = set()
         score.add(('SVM', score_svm))

         ----------------------------------
         The accuracy of the SVM is: 0.6416
         ----------------------------------
```

Here we used Decision tree classifier. It is giving good prediction results.

```
In [20]: #Dicision tree
         from sklearn.tree import DecisionTreeClassifier

         model_dt = DecisionTreeClassifier(random_state=4)
         model_dt.fit(X_train, y_train)
         y_prediction_dt = model_dt.predict(X_test)

         score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
         print("---------------------------------")
         print('The accuracy of the DT is: {}'.format(score_dt))
         print("---------------------------------")

         score.add(('DT', score_dt))

         ---------------------------------
         The accuracy of the DT is: 0.8707
         ---------------------------------
```

The next model is Knn. This model is giving decent accuracy but not very high.

In [21]:
```
from sklearn.neighbors import KNeighborsClassifier

model_knn = KNeighborsClassifier(n_neighbors=3)
model_knn.fit(X_train, y_train)
y_prediction_knn = model_knn.predict(X_test)

score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("----------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("----------------------------------")

score.add(('KNN', score_knn))
```

```
----------------------------------
The accuracy of the KNN is: 0.7363
----------------------------------
```

Then we used the Logistic Regression. This model doesn't give good accuracy on prediction.

In [22]:
```
#Logistic Regresssion
from sklearn.linear_model import LogisticRegression
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train)
y_prediction_lr = model_lr.predict(X_test)

score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
print("----------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("----------------------------------")

score.add(('LR', score_lr))
```

```
----------------------------------
The accuracy of the LR is: 0.6875
----------------------------------
```

Lastly, we used GaussianNB. This model gives better prediction but still less accurate.

In [23]:
```
#Gaussian
from sklearn.naive_bayes import GaussianNB

model_nb = GaussianNB()
model_nb.fit(X_train, y_train)
y_prediction_nb = model_nb.predict(X_test)

score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("----------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("----------------------------------")

score.add(('NB', score_nb))
```

```
----------------------------------
The accuracy of the NB is: 0.7602
----------------------------------
```

5. Discussion and conclusion

Based on the project requirements, we were tasked with building a classification-based model using Python on a real dataset consisting of a minimum of 2000 instances. We conducted data preprocessing, including data cleaning and exploratory data analysis. We then implemented several classification models, including Naive Bayes, K-Nearest Neighbors, Decision Tree, Logistic Regression, and Support Vector Machine.

After implementing the models, we compared their predictive accuracy and evaluated their performance. We found that the Decision Tree model performed better than the other models in terms of interpretability and ease of use.

```
In [20]: print("The accuracy scores of different Models:")
         print("----------------------------------------")
         for s in score:
          print(s)

The accuracy scores of different Models:
----------------------------------------
('DT', 0.8707)
('SVM', 0.6416)
('LR', 0.6875)
('NB', 0.7602)
('KNN', 0.7363)
```

In conclusion, we have successfully completed the project, which involved building a classification-based model using Python on a real dataset. We have demonstrated proficiency in data preprocessing, exploratory data analysis, and implementing multiple classification models. Our analysis and comparison of the models revealed that while SVM achieved the highest accuracy, Decision Tree provided the best balance between interpretability and performance.