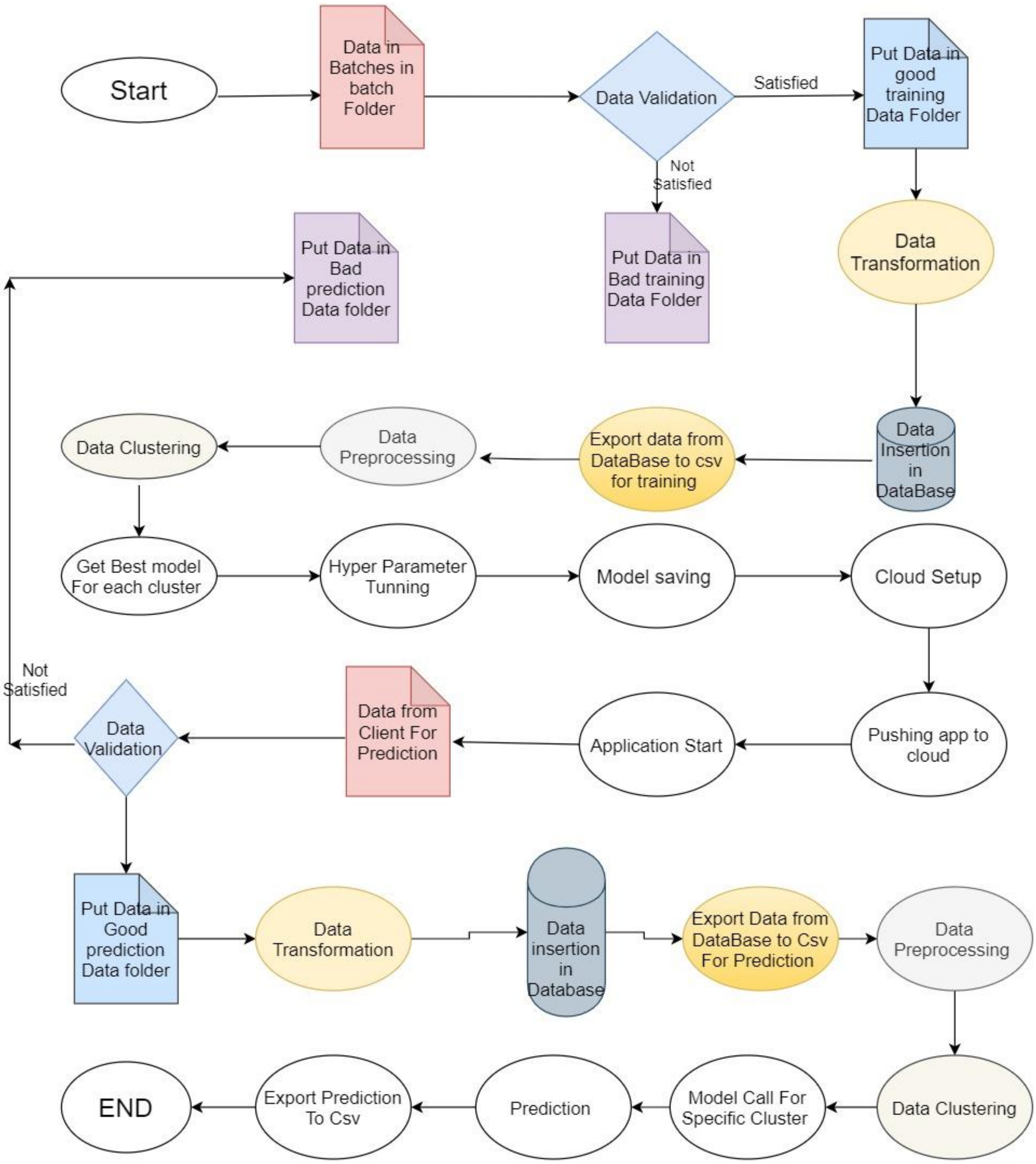
ABALONE'S AGE PREDICTION BY PREDICTING RINGS COUNTS

Title of Dataset: Abalone data

Problem Statement: Predicting the age of abalone from physical measurements. The age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope -- a boring and time-consuming task.

The number of rings is the value to predict: either as a continuous value or as a classification problem. No. of Rings +1.5 gives the age in years

Architecture:



Data Description :

Data Sources:

(a) Original owners of database:

Marine Resources Division

Marine Research Laboratories - Taroona

Department of Primary Industry and Fisheries, Tasmania

GPO Box 619F, Hobart, Tasmania 7001, Australia

(b) Donor of database:

Sam Waugh (Sam.Waugh@cs.utas.edu.au)

Department of Computer Science, University of Tasmania ,Australia

Data contains following attributes :

1. Number of Instances: 4177

2. Number of Attributes: 8

3. Attribute information Given is the attribute name, attribute type, the measurement unit and a brief description. The number of rings is the value to predict: it is a classification problem.

Name	**Data Types**	**Meas.**	**Description**
Sex	Nominal		M, F, and I (infant)
Length	Continuous	mm.	Longest shell measurement
Diameter	Continuous	mm.	perpendicular to length
Height	Continuous	mm.	with meat in shell
Whole weight	Continuous	grams	whole abalone
Shucked weight	Continuous	grams	weight of meat
Viscera weight	Continuous	grams	gut weight (after bleeding)
Shell weight	Continuous	grams	after being dried
Rings	Integer	Nos.	+1.5 gives the age in years

4.Statistics for numeric domains:

```
df=pd.read_csv('/content/abalone.csv')
df.describe()
```

	Length	Diameter	Height	Whole_Weight	Sucked_Weight	Viscera_Weight	Shell_Weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

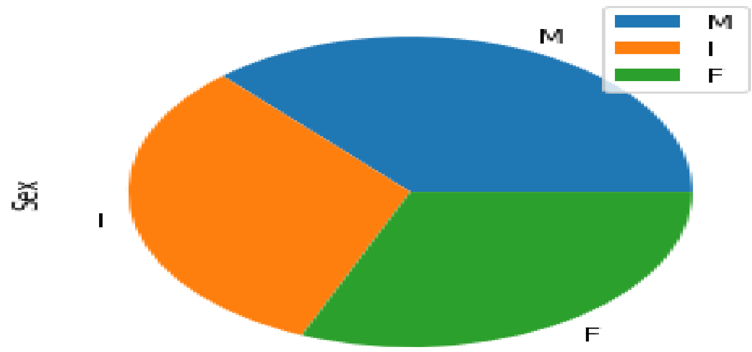
Conclusion from above result:-

- Mean of Length , diameter ,Height , Whole_Weight, Sucked_weight (weight without shell), shell Weight are 0.52 mm, 0.40mm,0.139mm,0.82 gm , 0.35gm,0.18gm,0.23gm
- Maximum length, diameter , height , whole weight , sucked weight , viscera weight , shell weight of abalone are 0.81mm,0.65mm,1.13mm,2.825gm,1.48gm,0.76gm,1.005gm respectively.

EDA:-

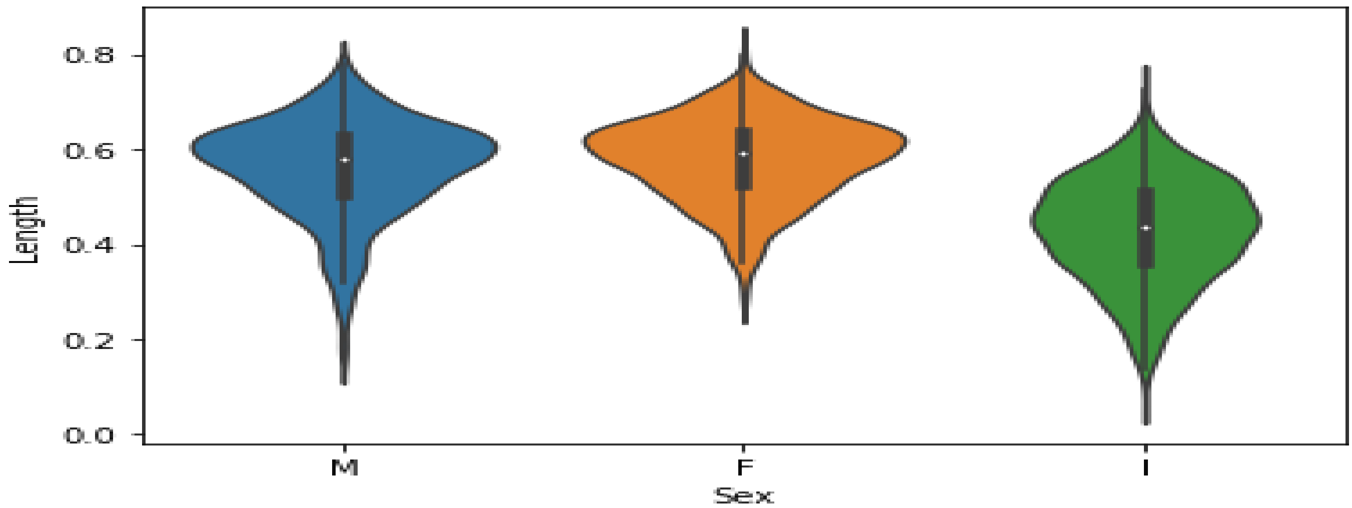
Distribution of sex (M,I,F) in data

```
data.Sex.value_counts().plot(kind='pie',legend=True);
```



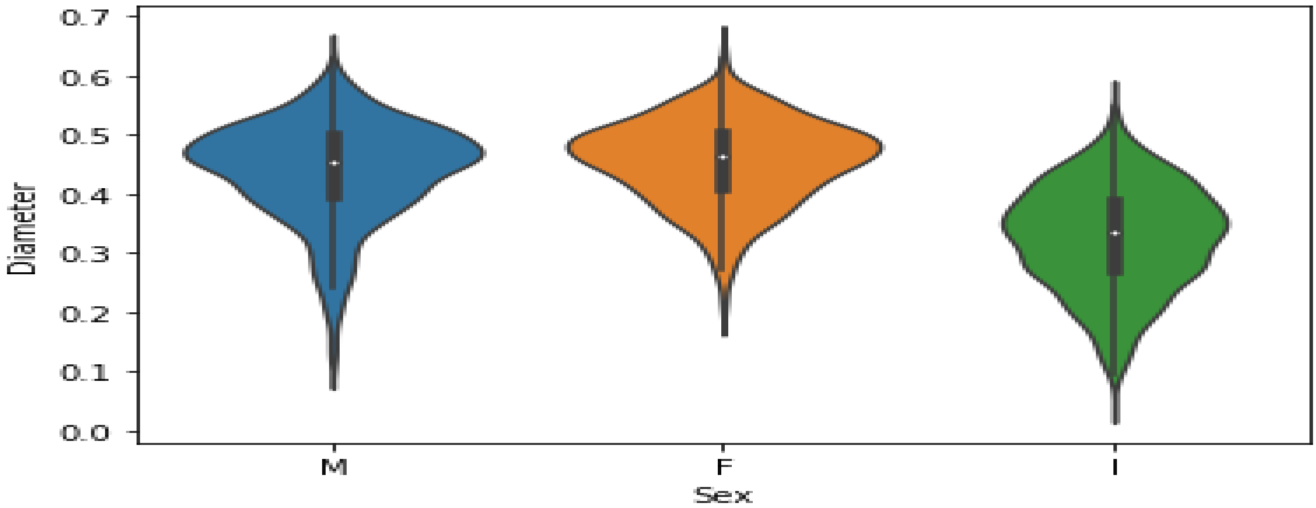
Below plot shows length with respect to Male , Female , Infant Abalone

```
sns.violinplot(x="Sex", y="Length", data=data);
```



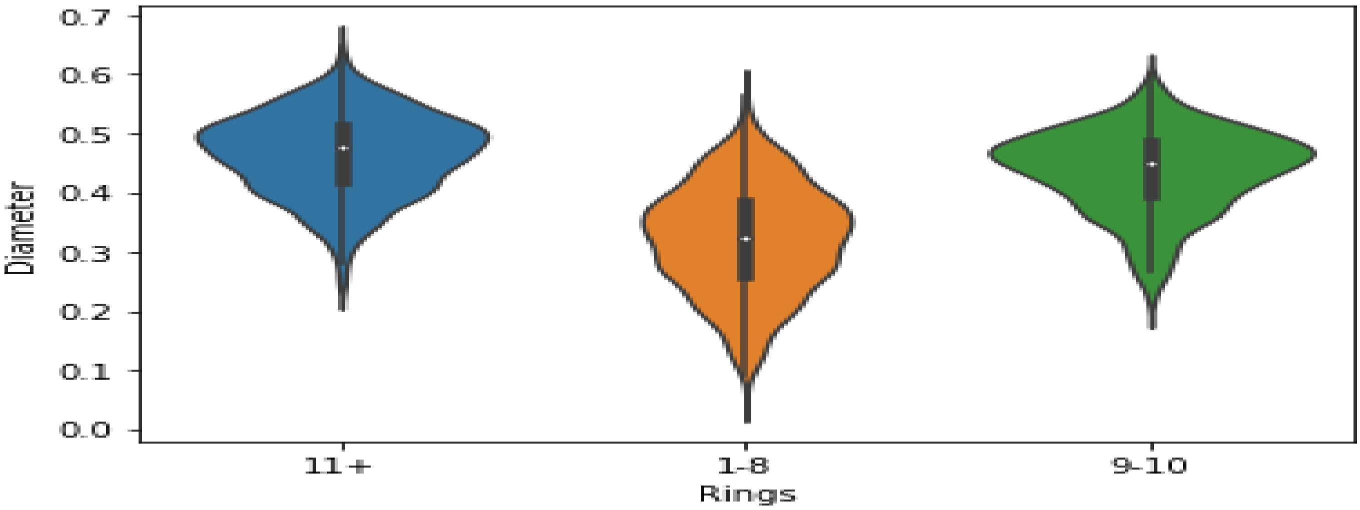
If we see then male and female have similar length and most of male and female having length between 0.5 to 0.7 mm where as infant length range is 0.28 to 0.5 mm

```
sns.violinplot(x="Sex",y="Diameter",data=data);
```



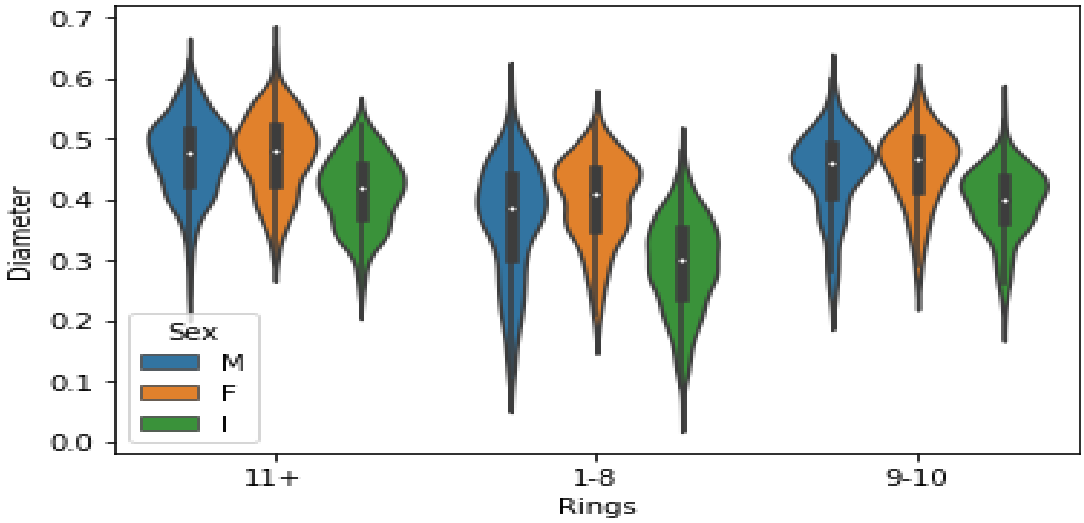
If we see distribution of diameter is similar to length distribution

```
sns.violinplot(x="Rings",y="Diameter",data=data,)
```



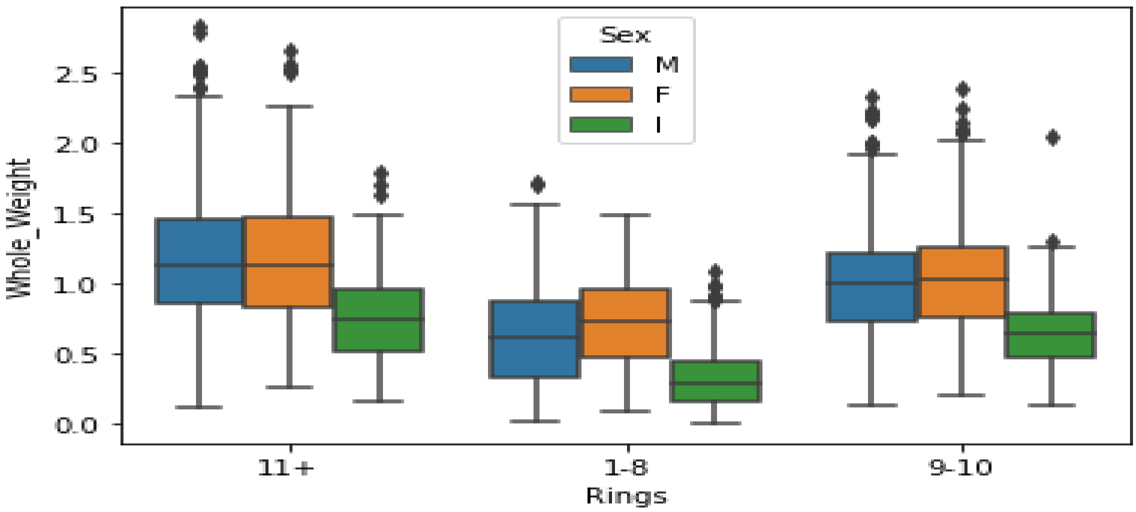
Here , in above plots those abalone whose no of rings are 1-8 having diameter range 0.3-0.6mm , where as for no of rings 9 to 29 having similar diameter with range 0.4 to 0.6mm

```
sns.violinplot(x="Rings",y="Diameter",data=data,hue='Sex');
```



If we see diameter of male under 1-8 no of rings are not normal , and in every group infant have less diameter in comparision to male and female.

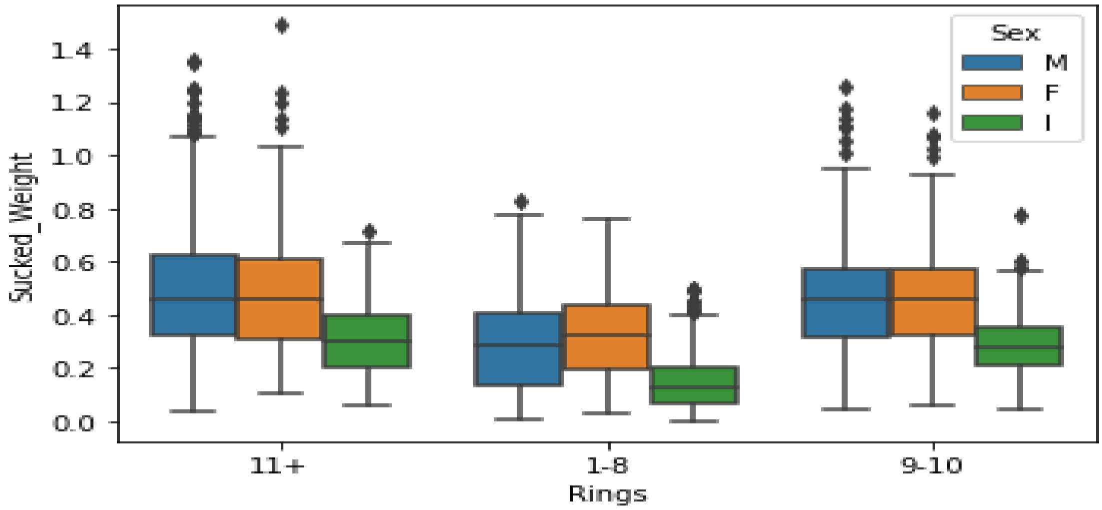
```
sns.boxplot(x="Rings",y="Whole_Weight",data=data,hue='Sex')
```



If we see in each category male and female Whole weight is same is same but in group 1-8 female abalone have higher weight than male. So we can conclude that generally female abalone are heavier than male abalone and infant abalone with respect to whole weight.

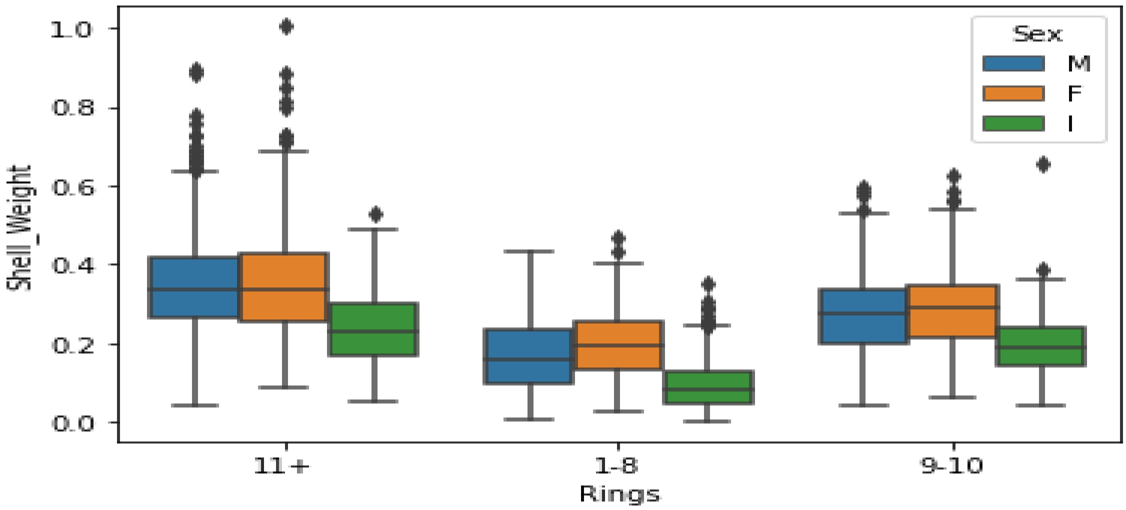
If

```
sns.boxplot(x="Rings",y="Sucked_Weight",data=data,hue='Sex');
```



In case of Sucked_Weight (Weight without shell) female have almost same in comparision to male , so we can conclude that weight of meat is higher in female abalone.

```
sns.boxplot(x="Rings",y="Shell_Weight",data=data,hue='Sex');
```



Shell weight of female abalone is higher than male in each category of age .

<u>5.Data Ditribution of Target Columns</u>:

```
df=data.Rings.value_counts()
df.sort_index()
```

1	1
2	1
3	15
4	57
5	115
6	259
7	391
8	568
9	689
10	634
11	487
12	267
13	203
14	126
15	103
16	67
17	58
18	42
19	32
20	26
21	14
22	6
23	9
24	2
25	1
26	1
27	2

But after grouping 1-8 in 1st group , 9-10 in 2nd group, rest in 11+ 3rd group

```
data['Rings'].value_counts()
```

11+	1447
1-8	1407
9-10	1323

Target Label:

Rings is the target columns which contains the no of rings counts. With right prediction of Rings on adding 1.5 years will give age of abalone

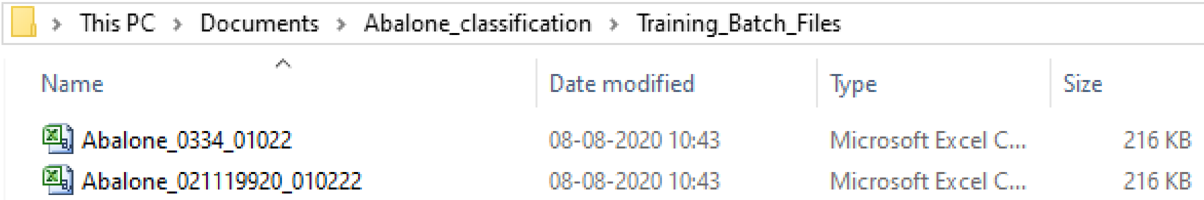
Data Validation :

In this step, we perform different sets of validation on the given set of training files.

1. Name Validation- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder."

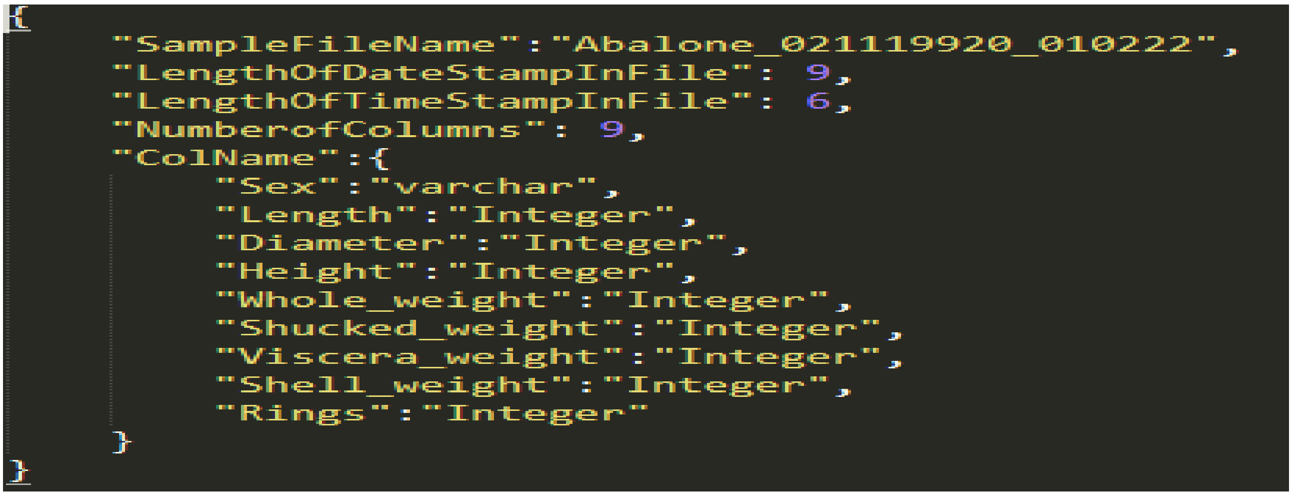
```
regex = "['Abalone']+['\_'']+[\d_]+[\d]+\.csv"
return regex
```

I created a regular expression whose structure will be Abalone_(digits)_(digits).csv.



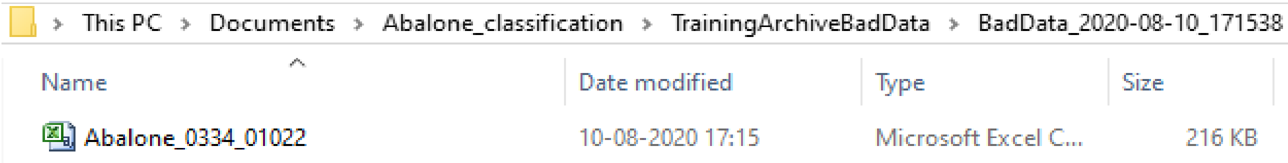
According to regular expression both file name given client is fine

But according to specification I created in Jason file below

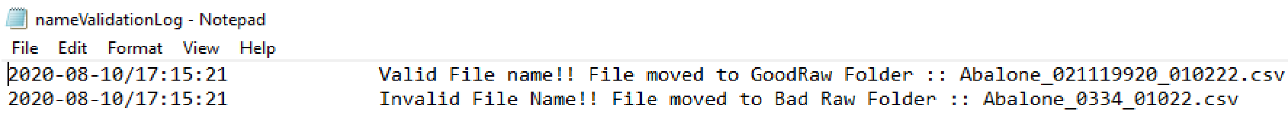



```
{
    "SampleFileName":"Abalone_021119920_010222",
    "LengthOfDateStampInFile": 9,
    "LengthOfTimeStampInFile": 6,
    "NumberofColumns": 9,
    "ColName":{
        "Sex":"varchar",
        "Length":"Integer",
        "Diameter":"Integer",
        "Height":"Integer",
        "Whole_weight":"Integer",
        "Shucked_weight":"Integer",
        "Viscera_weight":"Integer",
        "Shell_weight":"Integer",
        "Rings":"Integer"
    }
}
```


1st digit part(length_of_ DateStamp) and and 2nd digit part(length_of_timestamp) of is not correct for First file is correct so it is put to BadArchivefolder



After that logs will be generated like given below



2. Number of Columns - We validate the number of columns , Name of Columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder."

From the given Jason schema structure file given below

```
{
    "SampleFileName":"Abalone_021119920_010222",
    "LengthOfDateStampInFile": 9,
    "LengthOfTimeStampInFile": 6,
    "NumberofColumns": 9,
    "ColName":{
        "Sex":"varchar",
        "Length":"Integer",
        "Diameter":"Integer",
        "Height":"Integer",
        "Whole_weight":"Integer",
        "Shucked_weight":"Integer",
        "Viscera_weight":"Integer",
        "Shell_weight":"Integer",
        "Rings":"Integer"
    }
}
```

We extract Number of columns, Name of Columns using function in training_validation_insertion.py file given below:

```
# extracting values from prediction schema
LengthOfDateStampInFile, LengthOfTimeStampInFile, column_names, noofcolumns = self.raw_data.valuesFromSchema()
```

Now we pass these values to below function to validate as and also log file is generated as given below

```
valuesfromSchemaValidationLog - Notepad
File  Edit  Format  View  Help
2020-08-10/17:15:21     LengthOfDateStampInFile:: 9     LengthOfTimeStampInFile:: 6     NumberofColumns:: 9
```

Passing no of columns to the below function

```
# validating no of column  in the file
self.raw_data.validateColumnLength(noofcolumns)
```

it will check the no of columns in data using below function

```
def validateColumnLength(self, NumberofColumns):
    """
                  Method Name: validateColumnLength
                  Description: This function validates the number of colum
                               It is should be same as given in the schema
                               If not same file is not suitable for process
                               If the column number matches, file is kept 

                  Output: None
                  On Failure: Exception


              """
    try:
        f = open("Training_Logs/columnValidationLog.txt", 'a+')
        self.logger.log(f, "Column Length Validation Started!!")
        for file in listdir('Training_Raw_files_validated/Good_Raw/'):
            csv = pd.read_csv("Training_Raw_files_validated/Good_Raw/" + file)
            if csv.shape[1] == NumberofColumns:
```

after log file is maintained as given below

```
columnValidationLog - Notepad
File  Edit  Format  View  Help
2020-08-10/17:15:21          Column Length Validation Started!!
2020-08-10/17:15:21          Column Length Validation Completed!!
```

Similarly we validate the given below specification also

3. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4. The datatype of columns - The datatype of columns is given in the schema file. It is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Bad_Data_Folder".

5. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Bad_Data_Folder".
Here , we are checking is there any columns present in our dataset which have entire values null using below calling function in training_validation_insertion.py file

```
# validating if any column has all values missing
self.raw_data.validateMissingValuesInWholeColumn()
self.log_writer.log(self.file_object, "Raw Data Validation Complete!!")
```

It will navigate to the method given below

```
def validateMissingValuesInWholeColumn(self):
    """
                                    Method Name: validateMissingValuesInWholeColumn
                                    Description: This function validates if any column in
                                                 If all the values are missing, the file i
                                                 SUch files are moved to bad raw data.
                                    Output: None
                                    On Failure: Exception



                              """
    try:
        f = open("Training_Logs/missingValuesInColumn.txt", 'a+')
        self.logger.log(f, "Missing Values Validation Started!!")

        for file in listdir('Training_Raw_files_validated/Good_Raw/'):
            csv = pd.read_csv("Training_Raw_files_validated/Good_Raw/" + file)
            count = 0
            for columns in csv:
                if (len(csv[columns]) - csv[columns].count()) == len(csv[columns]):
                    count += 1
                    shutil.move("Training_Raw_files_validated/Good_Raw/" + file,
                                "Training_Raw_files_validated/Bad_Raw")
```

<u>Data Transformation:</u> In this phase we will replacing missing value with Null

method which we are calling in training_validation_insertion.py is given below:

```
self.log_writer.log(self.file_object, "Starting Data Transforamtion!!")
# replacing blanks in the csv file with "Null" values to insert in table
self.dataTransform.replaceMissingWithNull()
```

<u>Data Insertion in Database</u>

1) Database Creation and connection - Create a database with the given name passed. If the database has already been created, open a connection to the database. Here, I m using sqlite3 database.

```
# create database with given name, if present open the connection! Create table with columns given in schema
self.dBOperation.createTableDb('Training', column_names)
```

Below is the code given to buildup the connection to database

```
conn = self.dataBaseConnection(DatabaseName)
c = conn.cursor()
c.execute("SELECT count(name)  FROM sqlite_master WHERE type = 'table'AND name = 'Good_Raw_Data'")
if c.fetchone()[0] ==1:
    conn.close()
    file = open("Training_Logs/DbTableCreateLog.txt", 'a+')
    self.logger.log(file, "Tables created successfully!!")
    file.close()

    file = open("Training_Logs/DataBaseConnectionLog.txt", 'a+')
    self.logger.log(file, "Closed %s database successfully" % DatabaseName)
    file.close()
```

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created, and new files are inserted in the already present table as we want training to be done on new as well as old training files.

```
        for key in column_names.keys():
            type = column_names[key]


            try:
                conn.execute('ALTER TABLE Good_Raw_Data ADD COLUMN "{column_name}" {dataType}'.format(column_name=key,dataType=type))
            except:
                conn.execute('CREATE TABLE  Good_Raw_Data ({column_name} {dataType})'.format(column_name=key, dataType=type))

        conn.close()

        file = open("Training_Logs/DbTableCreateLog.txt", 'a+')
        self.logger.log(file, "Tables created successfully!!")
        file.close()

        file = open("Training_Logs/DataBaseConnectionLog.txt", 'a+')
        self.logger.log(file, "Closed %s database successfully" % DatabaseName)
        file.close()

except Exception as e:
    file = open("Training_Logs/DbTableCreateLog.txt", 'a+')
    self.logger.log(file, "Error while creating table: %s " % e)
    file.close()
    conn.close()
    file = open("Training_Logs/DataBaseConnectionLog.txt", 'a+')
```

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Calling function below method for database insertion from training_validation_insertion.py

```
self.dBOperation.insertIntoTableGoodData('Training')
self.log_writer.log(self.file_object, "Insertion in Table completed!!!")
self.log_writer.log(self.file_object, "Deleting Good Data Folder!!!")
# Delete the good data folder after loading files in table
```

Working of insertion of database given below

```
conn = self.dataBaseConnection(Database)
goodFilePath= self.goodFilePath
badFilePath = self.badFilePath
onlyfiles = [f for f in listdir(goodFilePath)]
log_file = open("Training_Logs/DbInsertLog.txt", 'a+')

for file in onlyfiles:
    try:
        with open(goodFilePath+'/'+file, "r") as f:
            next(f)
            reader = csv.reader(f, delimiter="\n")
            for line in enumerate(reader):
                for list_ in (line[1]):
                    try:
                        conn.execute('INSERT INTO Good_Raw_Data values ({values})'.format(values=(list_)))
                        self.logger.log(log_file," %s: File loaded successfully!!" % file)
                        conn.commit()
                    except Exception as e:
                        raise e
```

once data is inserted in the database , we will retrieve all the data to folder TrainingFilefromDB folder with name as input .csv and log file will be generated

```
# export data in table to csvfile
self.dBOperation.selectingDatafromtableintocsv('Training')
self.file_object.close()
```

Model Training :

1) **Data Export from Db -** The data in a stored database is exported as a CSV file to be used for model training.

```
# export data in table to csvfile
self.dBOperation.selectingDatafromtableintocsv('Training')
self.file_object.close()
```

2) **Data Preprocessing**

a) Drop the columns not required for prediction.

Function or method calling from trainingModel.py

```
data=preprocessor.remove_columns(data,[ ]) # remove the column as it doesn't contribute to prediction.
data.replace('?',np.NaN,inplace=True) # replacing '?' with NaN values for imputation
```

```
def remove_columns(self,data,columns):
    """
            Method Name: remove_columns
            Description: This method removes the given columns from
            Output: A pandas DataFrame after removing the specified
            On Failure: Raise Exception



    """
    self.logger_object.log(self.file_object, 'Entered the remove_co
    self.data=data
    self.columns=columns
    try:
        self.useful_data=self.data.drop(labels=self.columns, axis=1)
        self.logger_object.log(self.file_object, 'Column removal Su
        return self.useful_data
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured
        self.logger_object.log(self.file_object, 'Column removal Un
        raise Exception()
```

b) Check for null values in the columns. If present, impute the

```python
# check if missing values are present in the dataset
is_null_present, cols_with_missing_values = preprocessor.is_null_present(data)

# if missing values are there, replace them appropriately.
if (is_null_present):
    data = preprocessor.impute_missing_values(data, cols_with_missing_values)  # missing value imputation
```

Here , in below imputing method I m imputing numerical missing values by median imputation and categorical variable by frequent labels

```python
def impute_missing_values(self, data, cols_with_missing_values):
    """
                                    Method Name: impute_missing_values
                                    Description: This method replaces all the missing values in the Datafram
                                    Output: A Dataframe which has all the missing values imputed.
                                    On Failure: Raise Exception


                     """
    self.logger_object.log(self.file_object, 'Entered the impute_missing_values method of the Preprocessor c
    self.data= data
    self.cols_with_missing_values=cols_with_missing_values
    try:
        for col in self.cols_with_missing_values:
            if self.data[col].dtypes == 'O':
                self.imputer_s=mdi.CategoricalVariableImputer(imputation_method='frequent',variables=[col])
                self.data[col]=self.imputer_s.fit_transform(self.data[[col]])
            else:
                self.imputer_n=mdi.MeanMedianImputer(imputation_method='mean',variables=[col])
                self.data[col]=self.imputer_n.fit_transform(self.data[[col]])
```

c) Grouping the data based on target variable's label to make target variable balance and lower its high cardinality to low cardinality but after labeling our problem will be still multiclassification.

d) Replace and encode the categorical values with numeric values ,In this project I m doing this step after clustering because I m not including "sex" column in clustering . so, first I will remove "Sex", "Rings" columns then will perform clustering then I will attach the data for "Sex", "Rings" in clustered data.

```python
# create separate two data frames one on which we will perform cluster  and other is attached after performing
X, Y =preprocessor.separate_data_frame(data,label_column_name=['Rings','Sex'])
```

```python
def separate_data_frame(self, data, label_column_name):
    """
                Method Name: separate_label_feature
                Description: This method is used to seperate data frame in two group and on 1st group i will perform
                             2nd group to cluterd data frame
                Output: Returns two separate Dataframes, one containing dataframe on which clustering is perform and
                          to the clustered data.
                On Failure: Raise Exception


            """
    self.logger_object.log(self.file_object, 'Entered the separate_label_feature method of the Preprocessor class')
    try:
        self.X=data.drop(labels=label_column_name,axis=1) # drop the columns specified and separate the columns on which
        self.Y=data[label_column_name] # Filter the other data frame which we will not include in clustering
        self.logger_object.log(self.file_object, 'dataframe Separation for cluster is  Successful. Exited the separate_d
        return self.X,self.Y
```

3) **Clustering -** KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters using kneed method , The idea behind clustering is to implement different algorithms for different clusters.

The Kmeans model is trained over preprocessed data, and the model is saved for further use in prediction.

```
kmeans=clustering.KMeansClustering(self.file_object,self.log_writer) # object initialization.
number_of_clusters=kmeans.elbow_plot(X)  #  using the elbow plot to find the number of optimum clusters

# Divide the data into clusters
X=kmeans.create_clusters(X,number_of_clusters)

#create a new column in the dataset consisting of the corresponding cluster assignments.
X = pd.concat([X, Y], axis=1, sort=False)
```

```
def create_clusters(self,data,number_of_clusters):
    """
                                Method Name: create_clusters
                                Description: Create a new dataframe consisting of the cluster information.
                                Output: A datframe with cluster column
                                On Failure: Raise Exception
                        """
    self.logger_object.log(self.file_object, 'Entered the create_clusters method of the KMeansClustering class')
    self.data=data
    try:
        self.kmeans = KMeans(n_clusters=number_of_clusters, init='k-means++', random_state=42)
        #self.data = self.data[~self.data.isin([np.nan, np.inf, -np.inf]).any(1)]
        self.y_kmeans=self.kmeans.fit_predict(data) #  divide data into clusters

        self.file_op = file_methods.File_Operation(self.file_object,self.logger_object)
        self.save_model = self.file_op.save_model(self.kmeans, 'KMeans') # saving the KMeans model to directory
                                                                        # passing 'Model' as the functions need three parameters

        self.data['Cluster']=self.y_kmeans  # create a new column in dataset for storing the cluster information
        self.logger_object.log(self.file_object, 'succesfully created '+str(self.kn.knee)+ 'clusters. Exited the create_clusters method of 
        return self.data
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in create_clusters method of the KMeansClustering class. Exception messa
        self.logger_object.log(self.file_object,'Fitting the data to clusters failed. Exited the create_clusters method of the KMeansCluste
        raise Exception()
```

4)Attaching columns and Encoding categorical Values: here we r attaching the columns which we removed since they were not required for Cluster and after attaching encode the categorical values to numerical values

```
#create a new column in the dataset consisting of the corresponding cluster assignments.
X = pd.concat([X, Y], axis=1, sort=False)

# encode categorical data
X = preprocessor.encode_categorical_columns(X)

# getting the unique clusters from our dataset
list_of_clusters = X['Cluster'].unique()
```

```python
def encode_categorical_columns(self,data):
    """
                                          Method Name: encode_categorical_columns
                                          Description: This method encodes the categorical values to numeric values.
                                          Output: dataframe with categorical values converted to numerical values
                                          On Failure: Raise Exception


                      """
    self.logger_object.log(self.file_object, 'Entered the encode_categorical_columns method of the Preprocessor class')

    self.data=data
    try:
        self.cat_df = self.data.select_dtypes(include=['object']).copy()

        try:
            # code block for training
            self.cat_df['Rings'] = self.cat_df['Rings'].map({'1-8': 0, '11+': 1,'9-10':2})
            self.cols_to_drop=['Rings']
        except:
            # code block for Prediction
            self.cols_to_drop = [ ]
        # Using the dummy encoding to encode the categorical columns to numerical ones

        for col in self.cat_df.drop(columns=self.cols_to_drop).columns:
            self.cat_df = pd.get_dummies(self.cat_df, columns=[col], prefix=[col], drop_first=True)
```

5) **Model Selection –** After the clusters have been created, we find the best model for each cluster. We are using two algorithms, “Random Forest” and "XGBoost". For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the Accuracy scores & Roc_Auc score. we compare the model based on accuracy score for both models when entire columns have 1 lables and we compare and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

```python
for i in list_of_clusters:
    cluster_data = X[X['Cluster'] == i]  # filter the data for one cluster

    # Prepare the feature and Label columns
    cluster_features=cluster_data.drop(['Rings','Cluster'],axis=1)
    cluster_label= cluster_data['Rings']

    # splitting the data into training and test set for each cluster one by one
    x_train, x_test, y_train, y_test = train_test_split(cluster_features, cluster_label, test_size=1 / 3, random_state=100)


    model_finder=tuner.Model_Finder(self.file_object,self.log_writer) # object initialization

    #getting the best model for each of the clusters
    best_model_name,best_model,Roc_Auc_score=model_finder.get_best_model(x_train,y_train,x_test,y_test)

    #saving the best model to the directory.
    file_op = file_methods.File_Operation(self.file_object,self.log_writer)
    save_model=file_op.save_model(best_model,best_model_name+str(i))
    df = df.append({'Cluster_No' : i,  'Best_Model_Name' : best_model_name+str(i),'Roc_Auc_score':Roc_Auc_score},  ignore_index = True)

# logging the successful Training
self.log_writer.log(self.file_object, 'Successful End of Training')
self.file_object.close()
return df
```

Now below function is to find the best model for the particular cluster

```
def get_best_model(self,train_x,train_y,test_x,test_y):
    """
                                        Method Name: get_best_model
                                        Description: Find out the Model which has the best AUC score.
                                        Output: The best model name and the model object
                                        On Failure: Raise Exception



                                """
    self.logger_object.log(self.file_object,'Entered the get_best_model method of the Model_Finder class')
    # create best model for XGBoost
    try:
        self.xgboost= self.get_best_params_for_xgboost(train_x,train_y)
        self.prediction_xgboost = self.xgboost.predict(np.array(test_x))_# Predictions using the XGBoost Model

        if len(test_y.unique()) == 1: #if there is only one label in y, then roc_auc_score returns error. We will use accuracy in that case
            self.xgboost_score = accuracy_score(test_y, self.prediction_xgboost)
            self.logger_object.log(self.file_object, 'Accuracy for XGBoost:' + str(self.xgboost_score))  # Log AUC
        else:
            self.xgboost_score = self.multiclass_roc_auc_score(test_y, self.prediction_xgboost)_# AUC for XGBoost
            self.logger_object.log(self.file_object, 'AUC for XGBoost:' + str(self.xgboost_score))_# Log AUC

        # create best model for Random Forest
        self.rf=self.get_best_params_for_rf(train_x,train_y)
        self.prediction_rf=self.rf.predict(np.array(test_x))_# prediction using the SVM Algorithm
```

```
        # create best model for Random Forest
        self.rf=self.get_best_params_for_rf(train_x,train_y)
        self.prediction_rf=self.rf.predict(np.array(test_x))_# prediction using the Random Forest Algorithm

        if len(test_y.unique()) == 1:#if there is only one label in y, then roc_auc_score returns error. We will use accuracy in that case
            self.rf_score = accuracy_score(test_y,self.prediction_rf)
            self.logger_object.log(self.file_object, 'Accuracy for Random forest:' + str(self.rf_score))
        else:
            self.rf_score = self.multiclass_roc_auc_score(test_y, self.prediction_rf)_# AUC for Random Forest
            self.logger_object.log(self.file_object, 'AUC for Random Forest:' + str(self.rf_score))

        #comparing the two models
        if(self.rf_score <  self.xgboost_score):
            return 'XGBoost',self.xgboost,self.xgboost_score
        else:
            return 'RF',self.rf_classifier,self.rf_score

    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in get_best_model method of the Model_Finder class. Exception message:  ' +
        self.logger_object.log(self.file_object,'Model Selection Failed. Exited the get_best_model method of the Model_Finder class')
        raise Exception()
```

Result of training is shown on user interface as given below :

Prediction Results Or Trained Model Information

Results

Model for each cluster has been trained sucessfully, summary is given below:- !!!

result of training

	Cluster_No	Best_Model_Name	Roc_Auc_score
0	2	XGBoost2	0.695872
1	0	RF0	0.664862
2	1	XGBoost1	0.548081

!!!

Prediction Data Description :

The Client will send the data in multiple sets of files in batches at a given location. Data will contain the some physical measurement of various Abalone.

Apart from prediction files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns and their datatype.

Data Validation

In this step, we perform different sets of validation on the given set of Prediction files. These steps are similar to train Validation.

```
def prediction_validation(self):

    try:

        self.log_writer.log(self.file_object,'Start of Validation on files for prediction!!')
        #extracting values from prediction schema
        LengthOfDateStampInFile,LengthOfTimeStampInFile,column_names,noofcolumns = self.raw_data.valuesFromSchema()
        #getting the regex defined to validate filename
        regex = self.raw_data.manualRegexCreation()
        #validating filename of prediction files
        self.raw_data.validationFileNameRaw(regex,LengthOfDateStampInFile,LengthOfTimeStampInFile)
        #validating column length in the file
        self.raw_data.validateColumnLength(noofcolumns)
        #validating if any column has all values missing
        self.raw_data.validateMissingValuesInWholeColumn()
        self.log_writer.log(self.file_object,"Raw Data Validation Complete!!")

        self.log_writer.log(self.file_object,("Starting Data Transforamtion!!"))
        #replacing blanks in the csv file with "Null" values to insert in table
        self.dataTransform.replaceMissingWithNull()

        self.log_writer.log(self.file_object,"DataTransformation Completed!!!")

        self.log_writer.log(self.file_object,"Creating Prediction_Database and tables on the basis of given schema!!!")
        #create database with given name, if present open the connection! Create table with columns given in schema
        self.dBOperation.createTableDb('Prediction',column_names)
        self.log_writer.log(self.file_object,"Table creation Completed!!")
```

all the below given steps are done thorough above snippet of code written under prediction_validation_insertion.py

1) Name Validation- We validate the name of the files based on given Name in the schema file. We have created a regex pattern as per the name given in the schema file, to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of the timestamp in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder".

2) Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder".

3) Name of Columns - The name of the columns is validated and should be same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4) Datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is incorrect, then the file is moved to "Bad_Data_Folder".

5) Null values in columns - If any of the columns in a file has all the values as NULL or missing, we discard such file and move it to "Bad_Data_Folder".

Data Insertion in Database

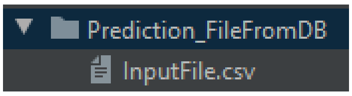
1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then a new table is not created, and new files are inserted into the already present table as we want training to be done on new as well old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Prediction:

1) Data Export from Db - The data in the stored database is exported as a CSV file to be used for prediction.



2) Data Preprocessing :

a) Drop the columns not required for prediction.

```
data = preprocessor.remove_columns(data,[])  # remove the column as it doesn't contribute to prediction.
```

```python
def remove_columns(self,data,columns):
    """
            Method Name: remove_columns
            Description: This method removes the given columns from a pandas dataframe.
            Output: A pandas DataFrame after removing the specified columns.
            On Failure: Raise Exception



    """
    self.logger_object.log(self.file_object, 'Entered the remove_columns method of the Preprocessor class')
    self.data=data
    self.columns=columns
    try:
        self.useful_data=self.data.drop(labels=self.columns, axis=1) # drop the labels specified in the columns
        self.logger_object.log(self.file_object, 'Column removal Successful.Exited the remove_columns method of the
        return self.useful_data
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in remove_columns method of the Preprocessor clas
        self.logger_object.log(self.file_object, 'Column removal Unsuccessful. Exited the remove_columns method of t
        raise Exception()
```

b) For this dataset, the null values were replaced with ‘?’ in the client data. Those ‘?’ have been replaced with NaN values.

```python
data.replace('?', np.NaN, inplace=True)  # replacing '?' with NaN values for imputation
```

c) Check for null values in the columns. If present, impute the numerical null values using the median imputer. Where as categorical imputer using frequent imputer.

```python
# if missing values are there, replace them appropriately.
if (is_null_present):
    data = preprocessor.impute_missing_values(data, cols_with_missing_values)  # missing value imputation
```

```python
def impute_missing_values(self, data, cols_with_missing_values):
    """
                                  Method Name: impute_missing_values
                                  Description: This method replaces all the missing values in the Dataframe using
                                  Output: A Dataframe which has all the missing values imputed.
                                  On Failure: Raise Exception


              """
    self.logger_object.log(self.file_object, 'Entered the impute_missing_values method of the Preprocessor class')
    self.data= data
    self.cols_with_missing_values=cols_with_missing_values
    try:
        for col in self.cols_with_missing_values:
            if self.data[col].dtypes == 'O':
                self.imputer_s=mdi.CategoricalVariableImputer(imputation_method='frequent',variables=[col])
                self.data[col]=self.imputer_s.fit_transform(self.data[[col]])
            else:
                self.imputer_n=mdi.MeanMedianImputer(imputation_method='mean',variables=[col])
                self.data[col]=self.imputer_n.fit_transform(self.data[[col]])

        self.logger_object.log(self.file_object, 'Imputing missing values Successful. Exited the impute_missing_valu
        return self.data
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in impute_missing_values method of the Preprocess
        self.logger_object.log(self.file_object,'Imputing missing values failed. Exited the impute_missing_values me
```

3) Clustering :- KMeans model created during training is loaded, and clusters for the preprocessed prediction data is predicted.

```python
kmeans = file_loader.load_model('KMeans')
```

```python
def load_model(self,filename):
    """
                Method Name: load_model
                Description: load the model file to memory
                Output: The Model file loaded in memory
                On Failure: Raise Exception


    """
    self.logger_object.log(self.file_object, 'Entered the load_model method of the File_Operation class')
    try:
        with open(self.model_directory + filename + '/' + filename + '.sav',
                  'rb') as f:
            self.logger_object.log(self.file_object,'Model File ' + filename + ' loaded. Exited the load_model method of the
            return pickle.load(f)
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in load_model method of the Model_Finder class. Exceptio
        self.logger_object.log(self.file_object,'Model File ' + filename + ' could not be saved. Exited the load_model meth
        raise Exception()
```

4) After Predicting the cluster no for test data we will encode the categorical variables.

```python
def encode_categorical_columns(self,data):
    """
                                        Method Name: encode_categorical_columns
                                        Description: This method encodes the categorical values to numeric values.
                                        Output: dataframe with categorical values converted to numerical values
                                        On Failure: Raise Exception


                            """
    self.logger_object.log(self.file_object, 'Entered the encode_categorical_columns method of the Preprocessor class')

    self.data=data
    try:
        self.cat_df = self.data.select_dtypes(include=['object']).copy()

        try:
            # code block for training
            self.cat_df['Rings'] = self.cat_df['Rings'].map({'1-8': 0, '11+': 1,'9-10':2})
            self.cols_to_drop=['Rings']
        except:
            # code block for Prediction
            self.cols_to_drop = [ ]
        # Using the dummy encoding to encode the categorical columns to numerical ones

        for col in self.cat_df.drop(columns=self.cols_to_drop).columns:
            self.cat_df = pd.get_dummies(self.cat_df, columns=[col], prefix=[col], drop_first=True)
```

5) Prediction - Based on the cluster number, the respective model is loaded and is used to predict the data for that cluster.

```python
for i in clusters:
    cluster_data = data[data['clusters'] == i]
    cluster_data = cluster_data.drop(['clusters'],axis=1)
    model_name = file_loader.find_correct_model_file(i)
    model = file_loader.load_model(model_name)
    result = (model.predict(np.array(cluster_data)))
    for res in result:
        if res == 0:
            predictions.append('1-8 Rings')
        elif res == 1:
            predictions.append('11+ Rings')
        else:
            predictions.append('9-10 Rings')
```

```python
def find_correct_model_file(self,cluster_number):
    """
                    Method Name: find_correct_model_file
                    Description: Select the correct model based on cluster number
                    Output: The Model file
                    On Failure: Raise Exception


        """
    self.logger_object.log(self.file_object, 'Entered the find_correct_model_file method of the File_Operation class')
    try:
        self.cluster_number= cluster_number
        self.folder_name=self.model_directory
        self.list_of_model_files = []
        self.list_of_files = os.listdir(self.folder_name)
        for self.file in self.list_of_files:
            try:
                if (self.file.index(str( self.cluster_number))!=-1):
                    self.model_name=self.file
            except:
                continue
        self.model_name=self.model_name.split('.')[0]
        self.logger_object.log(self.file_object,'Exited the find_correct_model_file method of the Model_Finder class.')
        return self.model_name
    except Exception as e:
        self.logger_object.log(self.file_object,'Exception occured in find_correct_model_file method of the Model_Finder cl
        self.logger_object.log(self.file_object,'Exited the find_correct_model_file method of the Model_Finder class with F
```

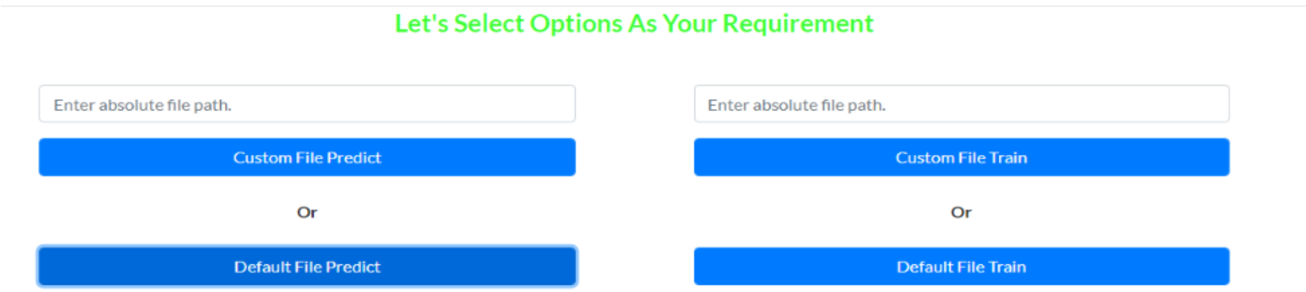
6) Once the prediction is made for all the clusters, the predictions along with the Rings Counts are saved in a CSV file at a given location, and the location is returned to the client on user interface.

```python
    final= pd.DataFrame(list(zip(predictions)),columns=['Predictions'])
    path="Prediction_Output_File/Predictions.csv"
    final.to_csv("Prediction_Output_File/Predictions.csv",header=True,mode='a+') #appends result to prediction file
    self.log_writer.log(self.file_object,'End of Prediction')
except Exception as ex:
    self.log_writer.log(self.file_object, 'Error occured while running the prediction!! Error:: %s' % ex)
    raise ex
return path , final
```

```python
# predicting for dataset present in database
path,result = pred.predictionFromModel()
X = pd.concat([pred_input_data,pd.DataFrame(result)],axis=1,sort=False)
print(X)
return Response("Prediction File created at %s!!!" %path +"   "+ "prediction results are given below %s" %X.to_html(
```

Result will be shown on user interface like this :

On clicking on defult file prediction as given below



Result of prediction will be shown in result box in user interface page

Prediction Results Or Trained Model Information

Results

"Prediction File created !!!

Prediction File created at Prediction_Output_File/Predictions.csv!!! prediction results are given below

	Sex	Length	Diameter	Height	Whole_weight	Shucked_weight	Viscera_weight	Shell_weight	Predictions
0	I	0.255	0.185	0.070	0.0750	0.0280	0.0180	0.0250	1-8 Rings
1	F	0.675	0.545	0.195	1.7345	0.6845	0.3695	0.6050	9-10 Rings
2	F	0.345	0.255	0.100	0.1970	0.0710	0.0510	0.0600	1-8 Rings
3	F	0.570	0.450	0.170	1.0980	0.4140	0.1870	0.4050	9-10 Rings
4	I	0.335	0.250	0.080	0.1830	0.0735	0.0400	0.0575	1-8 Rings
5	F	0.515	0.395	0.140	0.6860	0.2810	0.1255	0.2200	11+ Rings
6	F	0.635	0.495	0.195	1.2970	0.5560	0.2985	0.3700	11+ Rings
7	M	0.660	0.500	0.165	1.3195	0.6670	0.2690	0.3410	9-10 Rings
8	I	0.280	0.205	0.080	0.1270	0.0520	0.0390	0.0420	11+ Rings
9	I	0.305	0.230	0.075	0.1455	0.0595	0.0305	0.0500	11+ Rings