# Predicting the Age of the Abalone : from the given Dataset.

***Abalone is one of the PEARL producing animal.***

It is a type of snail. Which is actually a large marine gastropod mollusc,

Like most mollusks, they're permanently attached to their durable shells. Among the world's most expensive seafood, abalone is often sold live in the shell. It has extremely rich, flavorful, and highly prized meat that is considered a culinary delicacy .

## Goal of this Project :

***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

## Description of the Attributes :

```
Given is the attribute name, attribute type, the measurement unit and a
brief description.  The number of rings is the value to predict: either
as a continuous value or as a classification problem.

Name           DataType        Meas.        Description

Sex            nominal         M,F,I        Male, Female, 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 continuou      grams         weight of meat
Viscera weight continuous     grams         gut weight (after bleeding)
Shell weight   continuous     grams         after being dried
Rings           integer      +1.5            gives the age in years

No. of Rings is the Target varaible.
```

## Importing some Basic Libraries :

In [208]:

```
import os
os.getcwd()
```

Out[208]: 'C:\\Users\\shahe\\Desktop\\MachineLearning\\Project\\classification'

In [209]:

```
import numpy as np                # For Mathematical Calculations
import pandas as pd               # For Data Processing or I/O
import seaborn as sns             # For Visualization
import matplotlib.pyplot as plt  # For Visualization
```

In [210]:

```
import warnings
warnings.filterwarnings('ignore')
```

## Reading the Data :

In [211]:

```
colnames =["Sex","Length","Diameter","Height","WholeWeight","ShuckedWeight","Visce

df = pd.read_csv('Abalone.csv',names=colnames, header=None,na_values=' ')

# Adding  a Column
df['Age'] = df['Rings'] + 1.5
df.head(5)
```

Out[211]:

| | Sex | Length | Diameter | Height | WholeWeight | ShuckedWeight | VisceraWeight | ShellWeight | Rings |
|---|---|---|---|---|---|---|---|---|---|
| **0** | M | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 15 |
| **1** | M | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 7 |
| **2** | F | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 9 |
| **3** | M | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 10 |
| **4** | I | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 7 |

In [78]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Sex            4177 non-null   object 
 1   Length         4177 non-null   float64
 2   Diameter       4177 non-null   float64
 3   Height         4177 non-null   float64
 4   WholeWeight    4177 non-null   float64
 5   ShuckedWeight  4177 non-null   float64
 6   VisceraWeight  4177 non-null   float64
 7   ShellWeight    4177 non-null   float64
 8   Rings          4177 non-null   int64  
 9   Age            4177 non-null   float64
dtypes: float64(8), int64(1), object(1)
memory usage: 326.5+ KB
```

### Saving a Copy of the Original Data :

In [59]:

```
df_original = df.copy()
```

### Handling Missing Values :

In [60]:

```
df.isna().mean()
```

Out[60]:

```
Sex              0.0
Length           0.0
Diameter         0.0
Height           0.0
WholeWeight      0.0
ShuckedWeight    0.0
VisceraWeight    0.0
ShellWeight      0.0
Rings            0.0
Age              0.0
dtype: float64
```

### Checking for Duplicates :

In [61]:

```
df.duplicated().sum()
```

Out[61]:

```
0
```

### Checking the Structure of the Data :

In [62]:

```
## UNDERSTANDING THE DATA

### Checking the Shape of the Data Frame.

print ( f" The Shape of the Data Set: {df.shape} \n")
print ( f" Number of Observations: {df.shape[0]}\n " )
print ( f" Number of Columns: {df.shape[1]}\n " )
```

```
The Shape of the Data Set: (4177, 10)

Number of Observations: 4177

Number of Columns: 10
```

## Description of Data :

In [63]:

```
df.describe(include='all')
```

Out[63]:

| | Sex | Length | Diameter | Height | WholeWeight | ShuckedWeight | VisceraWeight |
|---|---|---|---|---|---|---|---|
| **count** | 4177 | 4177.000000 | 4177.000000 | 4177.000000 | 4177.000000 | 4177.000000 | 4177.000000 |
| **unique** | 3 | NaN | NaN | NaN | NaN | NaN | NaN |
| **top** | M | NaN | NaN | NaN | NaN | NaN | NaN |
| **freq** | 1528 | NaN | NaN | NaN | NaN | NaN | NaN |
| **mean** | NaN | 0.523992 | 0.407881 | 0.139516 | 0.828742 | 0.359367 | 0.180594 |
| **std** | NaN | 0.120093 | 0.099240 | 0.041827 | 0.490389 | 0.221963 | 0.109614 |
| **min** | NaN | 0.075000 | 0.055000 | 0.000000 | 0.002000 | 0.001000 | 0.000500 |
| **25%** | NaN | 0.450000 | 0.350000 | 0.115000 | 0.441500 | 0.186000 | 0.093500 |
| **50%** | NaN | 0.545000 | 0.425000 | 0.140000 | 0.799500 | 0.336000 | 0.171000 |
| **75%** | NaN | 0.615000 | 0.480000 | 0.165000 | 1.153000 | 0.502000 | 0.253000 |
| **max** | NaN | 0.815000 | 0.650000 | 1.130000 | 2.825500 | 1.488000 | 0.760000 |

```
Key insights :

        - No missing values in the dataset
        - All numerical features except 'sex'
        - Though features are not normaly distributed, are close to normality
        - None of the features have minimum = 0 except Height (requires re-check)
        - Each feature has difference scale range
```

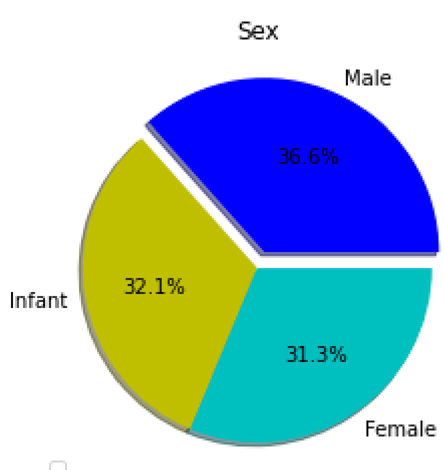
## UNIVARIATE ANALYSIS :

In [64]:
```python
df["Sex"].value_counts()
```

Out[64]:
```
M    1528
I    1342
F    1307
Name: Sex, dtype: int64
```

In [65]:
```python
Sex=df['Sex'].value_counts()
values = [Sex[0],Sex[1],Sex[2]]
colors = ['b', 'y','c']
labels = ['Male','Infant','Female']
explode = (0.1, 0,0)
plt.title('Sex')
plt.legend(labels,loc=3)
plt.pie(values, colors=colors, labels=labels,
explode=explode, autopct='%1.1f%%', counterclock=True, shadow=True)
```

Out[65]:
```
([<matplotlib.patches.Wedge at 0x1dc62e77c40>,
  <matplotlib.patches.Wedge at 0x1dc63e535b0>,
  <matplotlib.patches.Wedge at 0x1dc63e53e20>],
 [Text(0.4910229903302814, 1.09494128745203, 'Male'),
  Text(-1.0848393519507589, -0.18199884741134378, 'Infant'),
  Text(0.6099659291018239, -0.9153914820091724, 'Female')],
 [Text(0.2864300776926641, 0.638715751013684, '36.6%'),
  Text(-0.5917305556095048, -0.09927209858800569, '32.1%'),
  Text(0.3327086886009948, -0.49930444473227575, '31.3%')])
```




### Encoding : Label Encoding

In [212]:
```python
df.Sex = df.Sex.map(
    {
      "M" :0, "I" :2, "F" :4
    }
    )
```

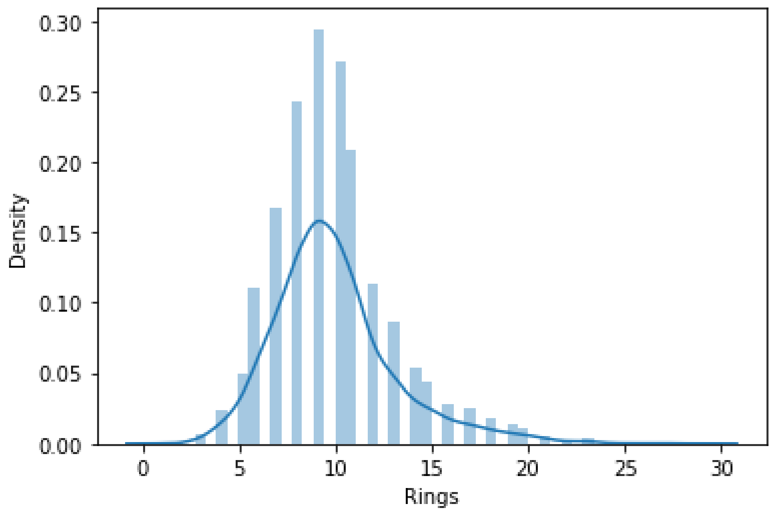
In [82]:
```
df["Sex"].value_counts()
```

Out[82]:
```
0    1528
2    1342
4    1307
Name: Sex, dtype: int64
```

## Visualizing the Data :

In [70]:
```
sns.distplot(df['Rings'])
```

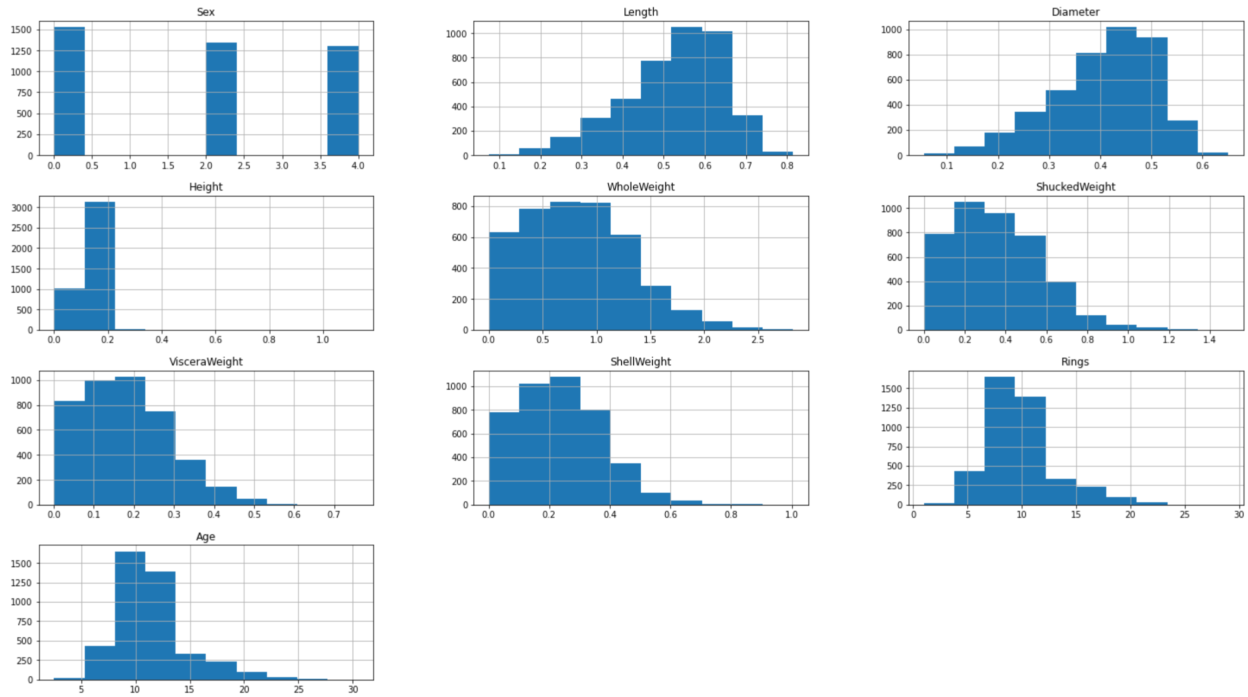
Out[70]: <AxesSubplot:xlabel='Rings', ylabel='Density'>




## Histogram gives the Distribution for Continous Features and Frequency of Discrete features.

In [83]:

```python
df.hist(figsize=(25,14))
plt.show()
```

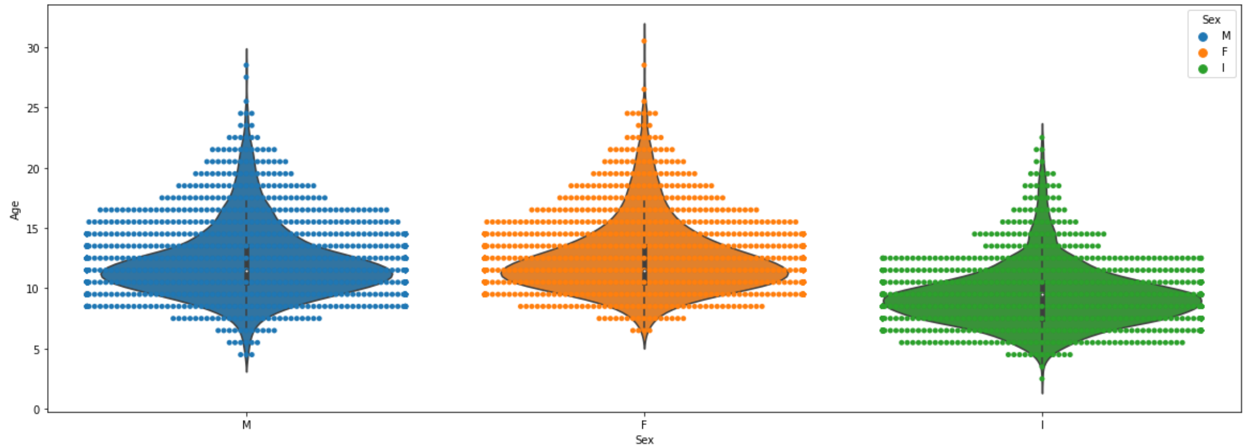



continous variables are skewed.
Discrete variable has 3 values, 0 is Male, 1 is Infant and 2 is Female.
The count of Male Abalone is slightly higher than Infant and Female Abalone.

- Height has highest skewedness followed by age, Shucked weight .

In [10]:
```
plt.figure(figsize = (20,7))
sns.swarmplot(x = 'Sex', y = 'Age', data = df, hue = 'Sex')
sns.violinplot(x = 'Sex', y = 'Age', data = df)
```

Out[10]: <AxesSubplot:xlabel='Sex', ylabel='Age'>

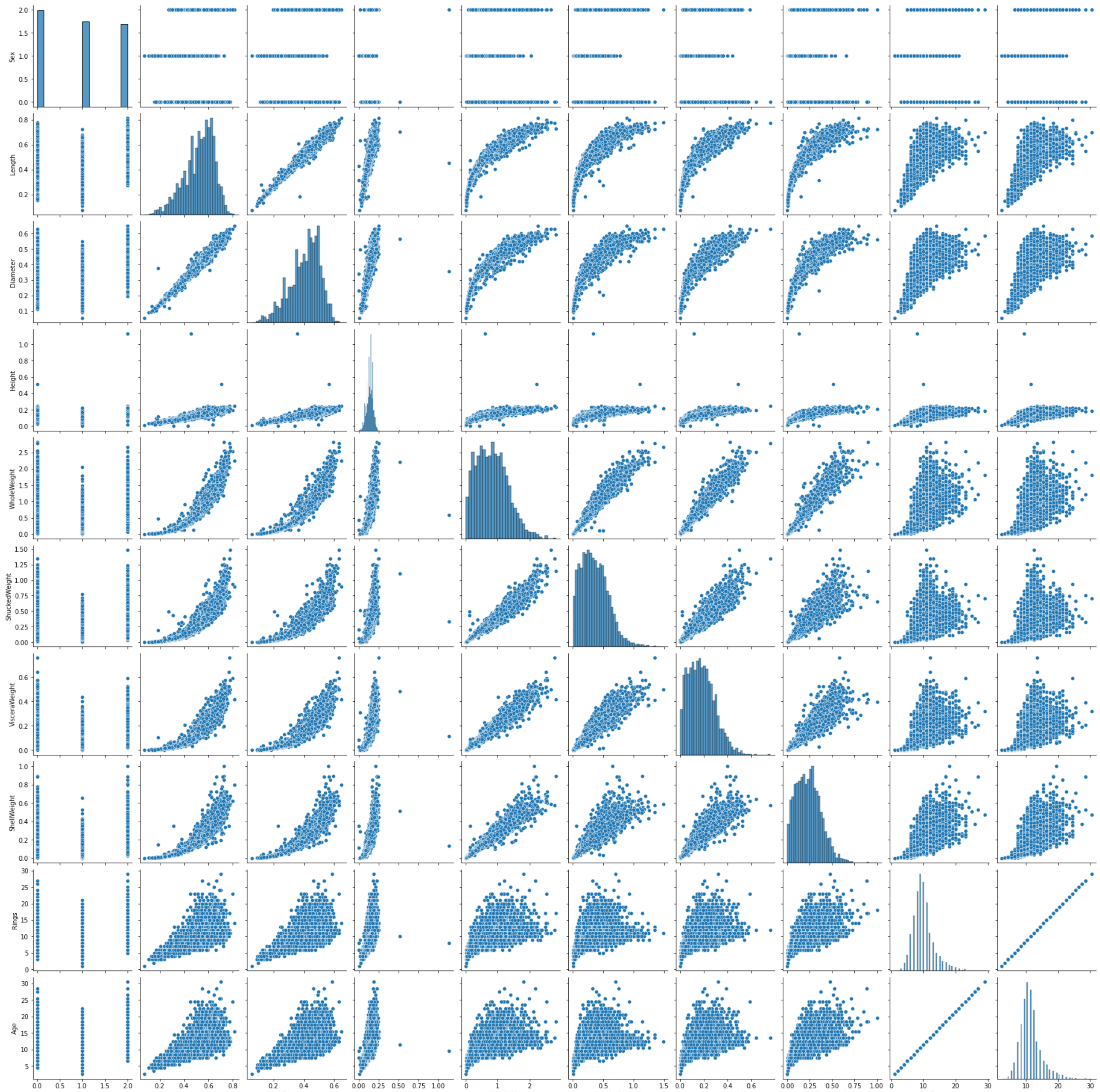



```
Male : age majority lies in between 7.5 years to 19 years
 Female: age majority lies in between 8 years to 19 years
 Immature: age majority lies in between 6 years to < 10 years
```

## BIVARIATE ANALYSIS :

In [16]:
```
sns.pairplot(df)
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x2315d8209a0>

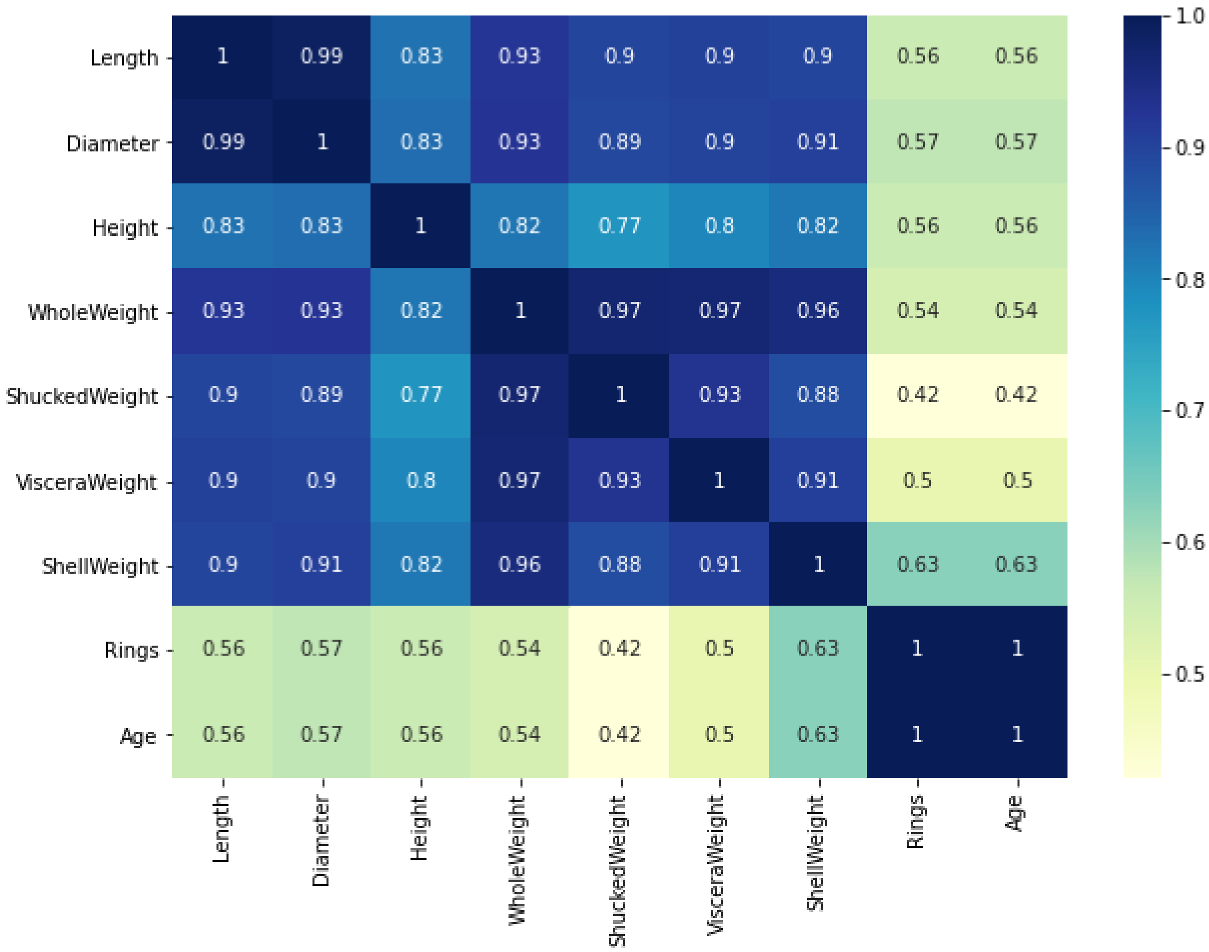


```
key insights :
    length is linearly correlated with diameter while, non-linear relation with
height,
    whole weight, shucked weight, viscera weight and shell weight
```

Pairplot will give visualization of the correlation among Continous variables. Though it is not very clear. we can view it clearly using Heatmap.

In [20]:
```python
dfcont=df[["Length","Diameter","Height","WholeWeight","ShuckedWeight","VisceraWeig
```

In [21]:
```python
plt.figure(figsize=(10,7))
dfcont = sns.heatmap(dfcont.corr(), cmap="YlGnBu", annot=True)
```



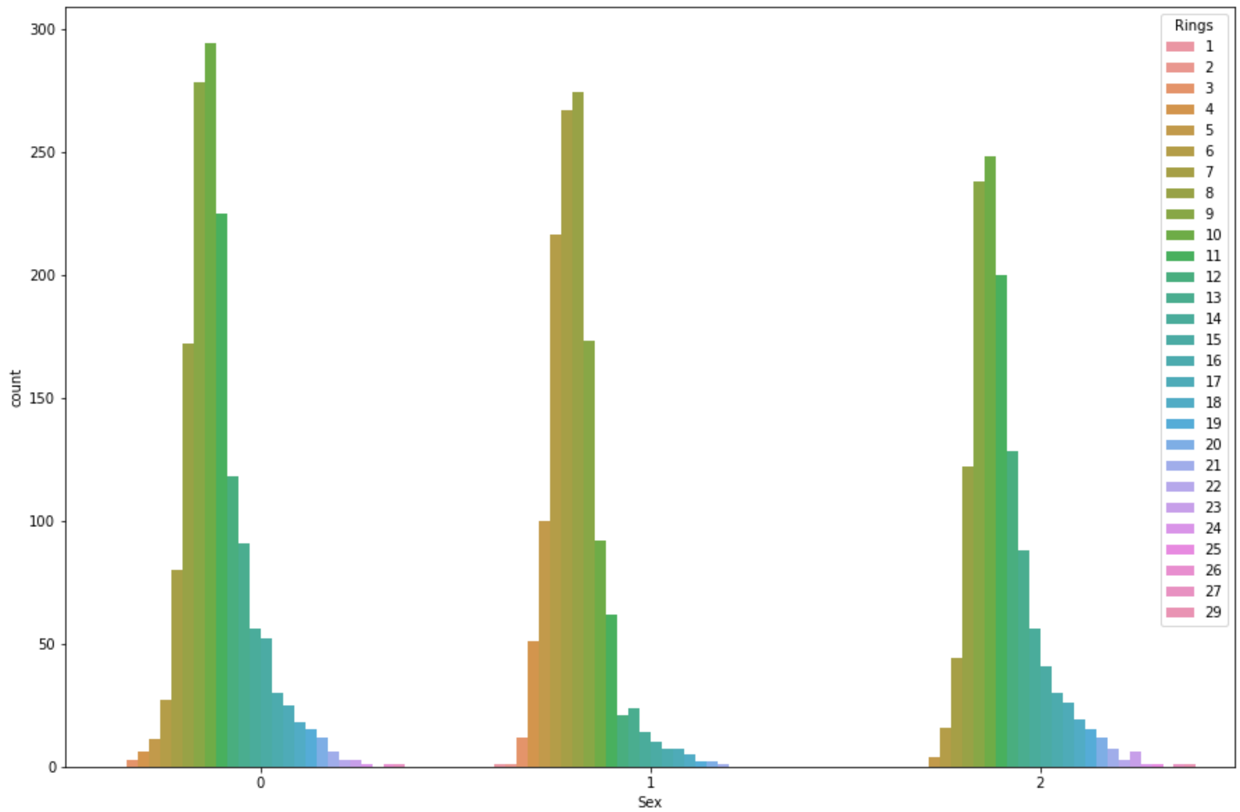

```
   Whole Weight is the most linearly correlated with all other features except age
    Heigh has least linearity with remaining features
    Age is most linearly proprtional with Shell Weight followed by Diameter and
length
```

```
Age is least correlated with Shucked Weight
```

```
Such high correlation coefficients among features can result into multi-
collinearity.
```

In [22]:
```python
plt.figure(figsize=(15,10))
sns.countplot(x=df['Sex'], hue=df['Rings'])
```

Out[22]: <AxesSubplot:xlabel='Sex', ylabel='count'>

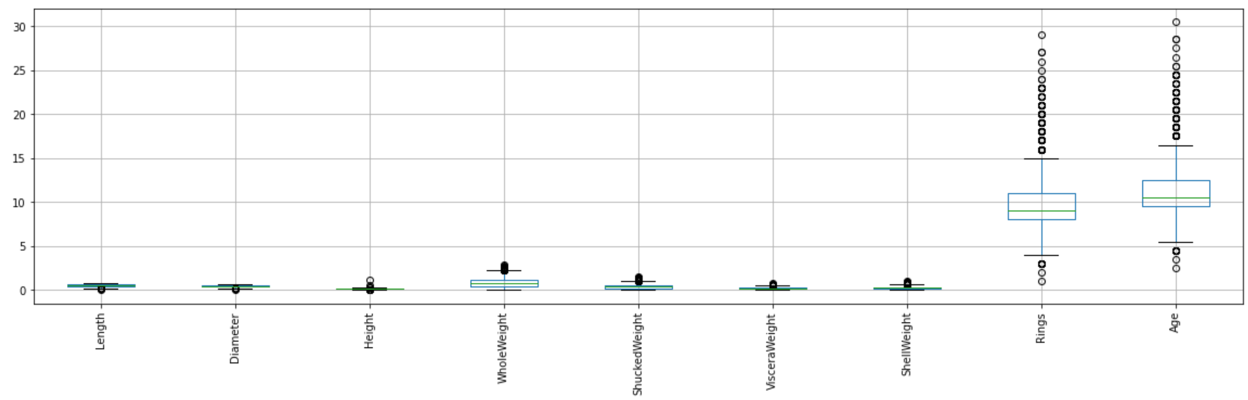



In [ ]:
```
The count of Male abalone is more than Infant and Female Abalone.
```

## OUTLIER ANALYSIS :

In [11]:
```python
df.boxplot( rot = 90, figsize=(20,5))
```

Out[11]: <AxesSubplot:>




## Splitting The Data to Independent variables(X), Dependent Variable(y).

In [213]:
```python
#extract dependent and independent variables
X = df.drop('Rings',axis=1)
y = df.Rings
```

In [214]:
```python
X.head(5)
```

Out[214]:

| | Sex | Length | Diameter | Height | WholeWeight | ShuckedWeight | VisceraWeight | ShellWeight | Age |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.455 | 0.365 | 0.095 | 0.5140 | 0.2245 | 0.1010 | 0.150 | 16.5 |
| **1** | 0 | 0.350 | 0.265 | 0.090 | 0.2255 | 0.0995 | 0.0485 | 0.070 | 8.5 |
| **2** | 4 | 0.530 | 0.420 | 0.135 | 0.6770 | 0.2565 | 0.1415 | 0.210 | 10.5 |
| **3** | 0 | 0.440 | 0.365 | 0.125 | 0.5160 | 0.2155 | 0.1140 | 0.155 | 11.5 |
| **4** | 2 | 0.330 | 0.255 | 0.080 | 0.2050 | 0.0895 | 0.0395 | 0.055 | 8.5 |

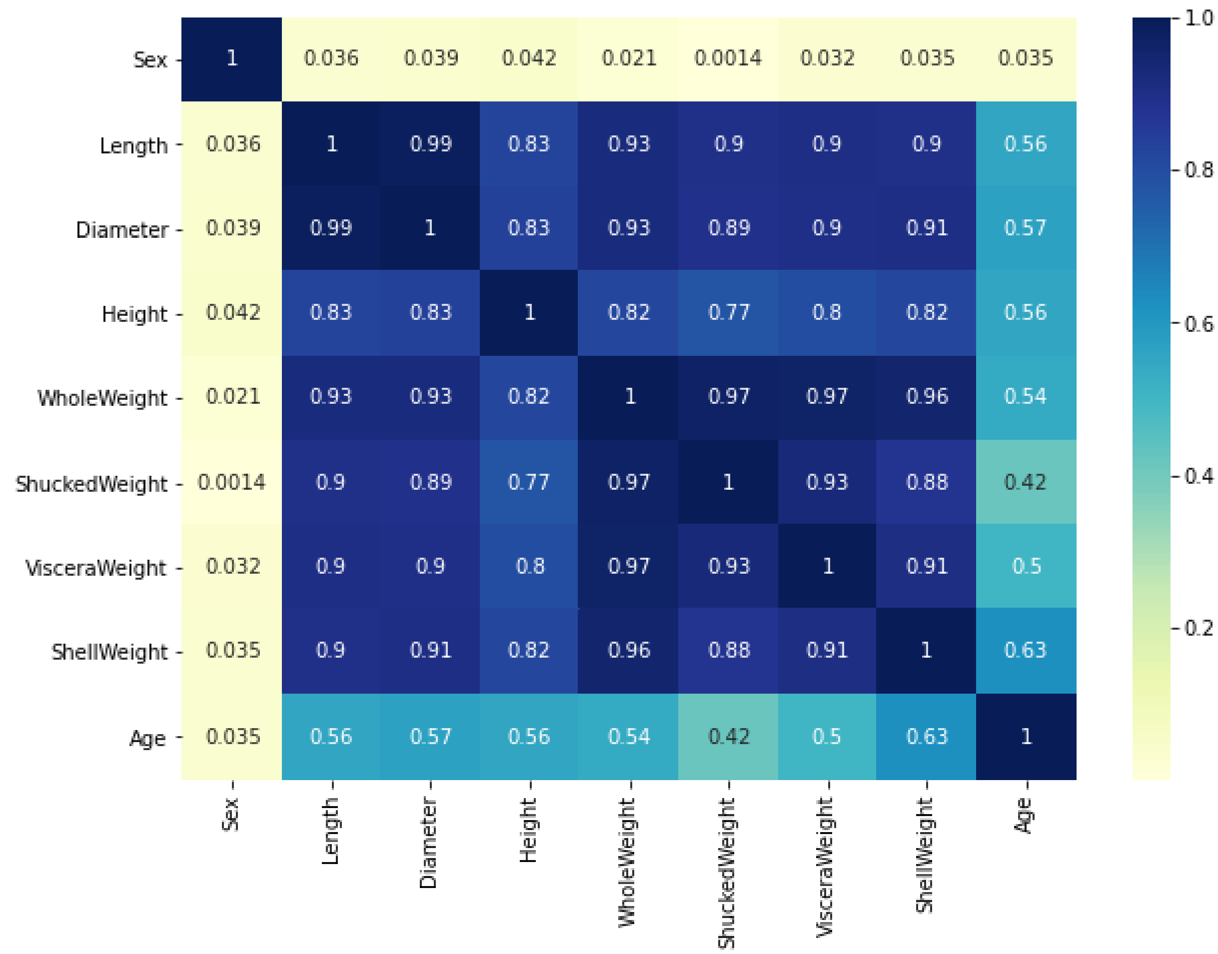
In [215]:
```python
y.head(5)
```

Out[215]:
```
0    15
1     7
2     9
3    10
4     7
Name: Rings, dtype: int64
```

## Checking the Correlation of only Independent variables :

In [216]:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,7))

ht = sns.heatmap(X.corr(), cmap="YlGnBu", annot=True)

# displaying heatmap
plt.show()
```




| | Sex | Length | Diameter | Height | WholeWeight | ShuckedWeight | VisceraWeight | ShellWeight | Age |
|---|---|---|---|---|---|---|---|---|---|
| Sex | 1 | 0.036 | 0.039 | 0.042 | 0.021 | 0.0014 | 0.032 | 0.035 | 0.035 |
| Length | 0.036 | 1 | 0.99 | 0.83 | 0.93 | 0.9 | 0.9 | 0.9 | 0.56 |
| Diameter | 0.039 | 0.99 | 1 | 0.83 | 0.93 | 0.89 | 0.9 | 0.91 | 0.57 |
| Height | 0.042 | 0.83 | 0.83 | 1 | 0.82 | 0.77 | 0.8 | 0.82 | 0.56 |
| WholeWeight | 0.021 | 0.93 | 0.93 | 0.82 | 1 | 0.97 | 0.97 | 0.96 | 0.54 |
| ShuckedWeight | 0.0014 | 0.9 | 0.89 | 0.77 | 0.97 | 1 | 0.93 | 0.88 | 0.42 |
| VisceraWeight | 0.032 | 0.9 | 0.9 | 0.8 | 0.97 | 0.93 | 1 | 0.91 | 0.5 |
| ShellWeight | 0.035 | 0.9 | 0.91 | 0.82 | 0.96 | 0.88 | 0.91 | 1 | 0.63 |
| Age | 0.035 | 0.56 | 0.57 | 0.56 | 0.54 | 0.42 | 0.5 | 0.63 | 1 |

There is Multicollinierity among almost all the independent features. as, it shows strong positive correlation.

PROJECT REQUIREMENTS : For classification using logistic regression, KNN, random forest, adaboost, SVM and one classification model of choice in SparkML For classification show: Accuracy, confusion matrix, (Macro recall and precision for multiclass Classification) Do hyper-parameter tuning using Grid Search

## Splitting the Data into Train (80%) and Test(20%)

In [222]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,
                                                 random_state=0,test_size=0.2)
```

In [223]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

#scale the features using training set

X_train_scaled = pd.DataFrame(sc.fit_transform(X_train))
X_test_scaled = pd.DataFrame(sc.transform(X_test))
```

## LOGISTIC REGRESSION :

In [224]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

model = LogisticRegression(multi_class='ovr',class_weight={9:1, 10:1, 8:1, 11:1.2,
                                                          15:6, 16:10, 17:10, 4:12
                                                          22:110, 24:320 , 27:320

cross_val_score(LogisticRegression(random_state=0),X_train_scaled,y_train,cv=4).me
```

Out[224]:
```
0.7443883763573331
```

In [225]:
```
#Training the model
model.fit(X_train_scaled,y_train)
```

Out[225]:
```
LogisticRegression(class_weight={1: 650, 2: 650, 3: 40, 4: 12, 5: 6, 6: 2.2,
                                 7: 1.5, 8: 1, 9: 1, 10: 1, 11: 1.2, 12: 2.2,
                                 13: 2.5, 14: 5, 15: 6, 16: 10, 17: 10, 18: 16,
                                 19: 18, 20: 26, 21: 40, 22: 110, 23: 75,
                                 24: 320, 25: 650, 26: 650, 27: 320},
                   multi_class='ovr')
```

In [226]:
```
#Validating the model

model.score(X_test_scaled,y_test)
```

Out[226]:
```
0.19617224880382775
```

## Hyper Parameter Tunning using GridSearchCV :

In [227]:

```
# Grid search cross validation
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

param = {"C":np.logspace(-3,3,7),
         "penalty":["l1","l2"]}# l1 lasso l2 ridge



lrmodel = GridSearchCV(LogisticRegression(),param,cv=10)
lrmodel.fit(X_train,y_train)

print("tuned hyperparameters :(best parameters) ",lrmodel.best_params_)
print("accuracy :",lrmodel.best_score_)
```

```
tuned hyperparameters :(best parameters)  {'C': 1.0, 'penalty': 'l2'}
accuracy : 0.4067584234516043
```

In [230]:

```
lrmodel1 = LogisticRegression(C=10,penalty="l2")
lrmodel1.fit(X_train,y_train)

print("model score",lrmodel1.score(X_test,y_test))
```

```
model score 0.4043062200956938
```

In [231]:
```
from sklearn.metrics import  classification_report
y_pred = lrmodel1.predict(X_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        11
           5       0.15      0.06      0.09        33
           6       0.00      0.00      0.00        47
           7       0.36      0.37      0.36        98
           8       0.46      0.49      0.47       113
           9       0.63      0.83      0.71       127
          10       0.56      0.85      0.67       107
          11       0.28      0.34      0.30        95
          12       0.00      0.00      0.00        66
          13       0.21      0.31      0.25        39
          14       0.00      0.00      0.00        26
          15       0.10      0.28      0.15        18
          16       0.00      0.00      0.00        14
          17       0.00      0.00      0.00        10
          18       0.00      0.00      0.00         5
          19       0.00      0.00      0.00         8
          20       0.00      0.00      0.00         8
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       0.00      0.00      0.00         2
          29       0.00      0.00      0.00         1

    accuracy                           0.40       836
   macro avg       0.12      0.16      0.14       836
weighted avg       0.32      0.40      0.35       836
```

## KNN Classifer :

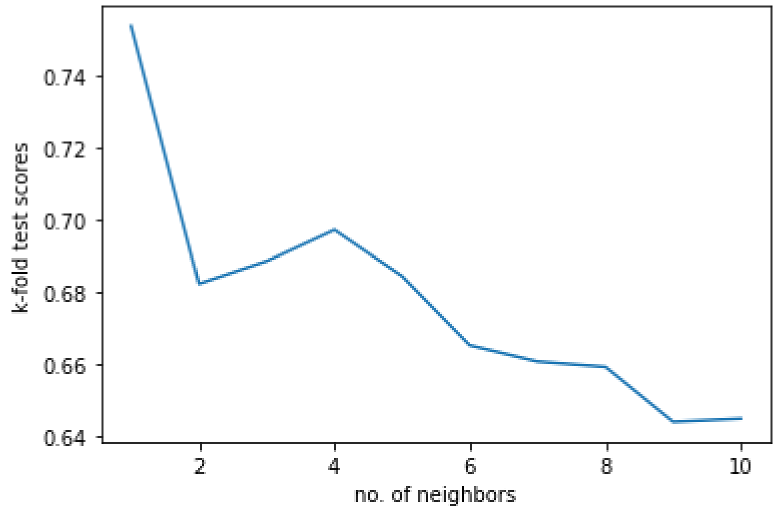
In [144]:
```
#import the knn model
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

In [145]:
```
#see the cross_validated score for cv=3
from sklearn.model_selection import cross_val_score
cross_val_score(knn,X_train_scaled,y_train,cv=3).mean()
```

Out[145]: 0.6468128956357675

In [146]:
```
#for no.of neighbors from 1 - 10, graph the k-fold scores
scores = []
for i in range(1,11,1):
    knn = KNeighborsClassifier(n_neighbors=i, weights='uniform')
    scores.append(cross_val_score(knn,X_train_scaled,y_train,cv=8).mean())
```

In [147]:
```
import matplotlib.pyplot as plt
plt.plot(range(1,11,1),scores)
plt.xlabel('no. of neighbors')
plt.ylabel('k-fold test scores')
plt.show()
```




**4-NN is theBest Model. Since, the curve is starting to drop from 4 of x-axis which is 'no.of neighbours'.**

In [148]:
```
knnmodel = KNeighborsClassifier(n_neighbors=4)
knnmodel.fit(X_train_scaled,y_train)
```

Out[148]:
```
KNeighborsClassifier(n_neighbors=4)
```

In [149]:
```
y_pred = knnmodel.predict(X_test)
```

In [150]:
```
knnmodel.score(X_test_scaled,y_test)
```

Out[150]:
```
0.7045454545454546
```

## Hyper Parameter Tuning Using GridSearchCV :

In [170]:

```python
from sklearn.model_selection import GridSearchCV
params = {'n_neighbors':[2,3,4,5,6,7,8,9],
          'weights':['uniform','distance'],
          'metric':['euclidean','manhattan']}

knnmodel1 = GridSearchCV(
                  KNeighborsClassifier(),
                  params,
                  cv = 4,
                  n_jobs = -1
                 )


knnmodel1.fit(X_train_scaled,y_train)
knnmodel1.best_params_
```

Out[170]:

```
{'metric': 'manhattan', 'n_neighbors': 4, 'weights': 'distance'}
```

In [171]:

```python
from sklearn.metrics import classification_report
y_pred = knnmodel1.predict(X_test_scaled)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       1.00      0.20      0.33         5
           4       0.64      0.82      0.72        11
           5       0.94      0.88      0.91        33
           6       0.91      0.91      0.91        47
           7       0.96      0.87      0.91        98
           8       0.87      0.90      0.89       113
           9       0.88      0.96      0.92       127
          10       0.83      0.89      0.86       107
          11       0.80      0.82      0.81        95
          12       0.75      0.67      0.70        66
          13       0.74      0.72      0.73        39
          14       0.64      0.54      0.58        26
          15       0.50      0.61      0.55        18
          16       0.46      0.43      0.44        14
          17       0.56      0.50      0.53        10
          18       0.25      0.20      0.22         5
          19       0.38      0.38      0.38         8
          20       0.50      0.38      0.43         8
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       0.67      1.00      0.80         2
          29       0.00      0.00      0.00         1

    accuracy                           0.81       836
   macro avg       0.60      0.58      0.57       836
weighted avg       0.81      0.81      0.81       836
```

### Random Forest Classifier :

In [172]:

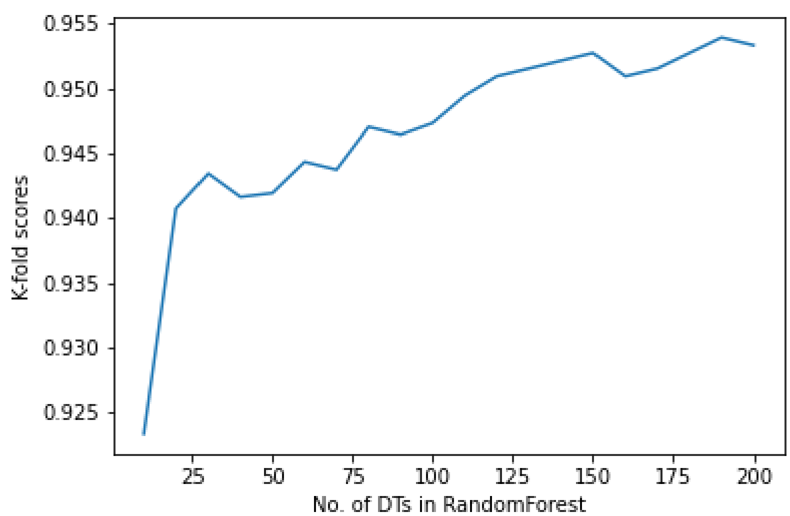
```
#import libraries
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
```

In [173]:

```
#Graph k-fold score vs no. of estimators in Random Forest
scores = []
for i in range(10,201,10):
    scores.append(cross_val_score(RandomForestClassifier(n_estimators=i,random_sta
                              X_train_scaled,y_train,cv=4).mean())
```

In [174]:

```
plt.plot(range(10,201,10),scores)
plt.xlabel('No. of DTs in RandomForest')
plt.ylabel('K-fold scores')
plt.show()
```




## Hyper Parameter Tuning Using GridSearchCV :

In [175]:

```
#including other params like max_depth, we will apply gridsearch to fine the best
params = {
            'n_estimators': [50,60,70,80,90],
            'max_depth': [12,14,16,18,20]
        }
model = GridSearchCV(RandomForestClassifier(random_state=0), params,cv=4)
model.fit(X_train_scaled,y_train)
```

Out[175]:

```
GridSearchCV(cv=4, estimator=RandomForestClassifier(random_state=0),
             param_grid={'max_depth': [12, 14, 16, 18, 20],
                         'n_estimators': [50, 60, 70, 80, 90]})
```

In [176]:
```
model.best_params_
```
Out[176]:
```
{'max_depth': 18, 'n_estimators': 80}
```

In [177]:
```
model.best_score_
```
Out[177]:
```
0.9476226971893533
```

In [178]:
```
best_model = model.best_estimator_
```

In [179]:
```
best_model.fit(X_train_scaled,y_train)
```
Out[179]:
```
RandomForestClassifier(max_depth=18, n_estimators=80, random_state=0)
```

In [180]:
```
y_pred = best_model.predict(X_test_scaled)
```

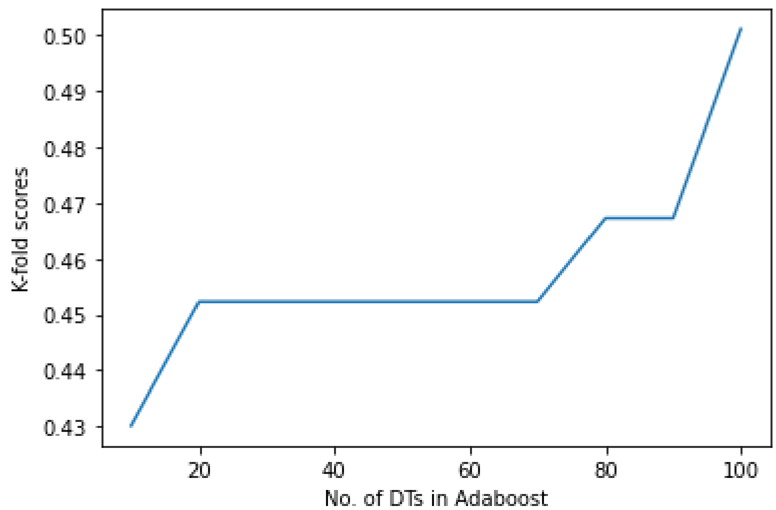
In [181]:
```

print(classification_report(y_test,y_pred))
```
```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00         5
           4       1.00      0.91      0.95        11
           5       0.97      0.97      0.97        33
           6       0.98      0.98      0.98        47
           7       0.99      0.99      0.99        98
           8       0.99      1.00      1.00       113
           9       1.00      1.00      1.00       127
          10       0.99      1.00      1.00       107
          11       1.00      0.99      0.99        95
          12       1.00      1.00      1.00        66
          13       1.00      1.00      1.00        39
          14       0.89      0.92      0.91        26
          15       0.69      1.00      0.82        18
          16       0.71      0.71      0.71        14
          17       0.45      0.50      0.48        10
          18       0.43      0.60      0.50         5
          19       0.40      0.25      0.31         8
          20       0.33      0.12      0.18         8
          21       0.00      0.00      0.00         2
          22       0.00      0.00      0.00         1
          23       1.00      0.50      0.67         2
          29       0.00      0.00      0.00         1

    accuracy                           0.96       836
   macro avg       0.72      0.70      0.70       836
weighted avg       0.95      0.96      0.95       836
```

```

Accuracy for Logistic Regression and KNN models is 34%, KNN classifier is 81%.
Random Forest model shows a huge improvement in Accuracy which is 96%.
```

## Ada Boost Classifier :

In [182]:

```
from sklearn.ensemble import AdaBoostClassifier

#Graph k-fold score vs no. of estimators in Adaboost which uses DT as base estimat
scores = []
for i in range(10,101,10):
    scores.append(cross_val_score(AdaBoostClassifier(n_estimators=i,random_state=0
                                  X_train_scaled,y_train,cv=4).mean())
plt.plot(range(10,101,10),scores)
plt.xlabel('No. of DTs in Adaboost')
plt.ylabel('K-fold scores')
plt.show()
```




## Hyper Parameter Tuning :

In [187]:
```python
from sklearn.tree import DecisionTreeClassifier
#including other params like max_depth, we will apply gridsearch to fine the best
params = {
            'n_estimators': [5,10,20,30,40,50,60,70,80],
            'base_estimator': [DecisionTreeClassifier(max_depth=8,random_state=0),
                               DecisionTreeClassifier(max_depth=9,random_state=0),
                               DecisionTreeClassifier(max_depth=10,random_state=0)
                              DecisionTreeClassifier(max_depth=11,random_state=0),
                              DecisionTreeClassifier(max_depth=12,random_state=0)]
        }
model = GridSearchCV(AdaBoostClassifier(random_state=0), params,cv=4)
model.fit(X_train_scaled,y_train)
```

Out[187]:
```
GridSearchCV(cv=4, estimator=AdaBoostClassifier(random_state=0),
             param_grid={'base_estimator': [DecisionTreeClassifier(max_depth=8,
                                                                   random_state=
0),
                                            DecisionTreeClassifier(max_depth=9,
                                                                   random_state=
0),
                                            DecisionTreeClassifier(max_depth=10,
                                                                   random_state=
0),
                                            DecisionTreeClassifier(max_depth=11,
                                                                   random_state=
0),
                                            DecisionTreeClassifier(max_depth=12,
                                                                   random_state=
0)],
                         'n_estimators': [5, 10, 20, 30, 40, 50, 60, 70, 80]})
```

In [188]:
```python
model.best_params_
```

Out[188]:
```
{'base_estimator': DecisionTreeClassifier(max_depth=9, random_state=0),
 'n_estimators': 10}
```

In [189]:
```python
model.best_score_
```

Out[189]:
```
0.9988034696157924
```

In [190]:
```python
best_model = model.best_estimator_
```

In [191]:
```python
best_model.fit(X_train_scaled,y_train)
```

Out[191]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=9,
                                                         random_state=0),
                   n_estimators=10, random_state=0)
```

In [192]:
```python
y_pred = best_model.predict(X_test_scaled)
```

In [193]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00         5
           4       1.00      1.00      1.00        11
           5       1.00      1.00      1.00        33
           6       1.00      1.00      1.00        47
           7       1.00      1.00      1.00        98
           8       1.00      1.00      1.00       113
           9       1.00      1.00      1.00       127
          10       1.00      1.00      1.00       107
          11       1.00      1.00      1.00        95
          12       1.00      1.00      1.00        66
          13       1.00      1.00      1.00        39
          14       1.00      1.00      1.00        26
          15       1.00      1.00      1.00        18
          16       1.00      1.00      1.00        14
          17       1.00      1.00      1.00        10
          18       1.00      1.00      1.00         5
          19       1.00      1.00      1.00         8
          20       1.00      1.00      1.00         8
          21       1.00      1.00      1.00         2
          22       1.00      1.00      1.00         1
          23       1.00      1.00      1.00         2
          27       0.00      0.00      0.00         0
          29       0.00      0.00      0.00         1

    accuracy                           1.00       836
   macro avg       0.91      0.91      0.91       836
weighted avg       1.00      1.00      1.00       836
```

## SVM Classifier :

In [195]:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

In [199]:

```
params = {
                        'C' : [0.1, 1, 10, 15],
                        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
                        'degree': [2,3],
                        'gamma' : [0.001,0.005,0.1]
                    }

model = GridSearchCV(SVC(random_state=0),param_grid=params,cv=4)
```

In [200]:
```
model.fit(X_train_scaled,y_train)
```
Out[200]:
```
GridSearchCV(cv=4, estimator=SVC(random_state=0),
             param_grid={'C': [0.1, 1, 10, 15], 'degree': [2, 3],
                         'gamma': [0.001, 0.005, 0.1],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']})
```
In [201]:
```
model.best_params_
```
Out[201]:
```
{'C': 10, 'degree': 2, 'gamma': 0.001, 'kernel': 'linear'}
```
In [202]:
```
model.best_score_
```
Out[202]:
```
0.9952110133799387
```
In [203]:
```
svm =model.best_estimator_
```
In [204]:
```
svm.fit(X_train_scaled,y_train)
```
Out[204]:
```
SVC(C=10, degree=2, gamma=0.001, kernel='linear', random_state=0)
```
In [205]:
```
svm.score(X_test_scaled,y_test)
```
Out[205]:
```
0.9988038277511961
```
In [206]:
```
y_pred = svm.predict(X_test_scaled)
```

In [207]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       1.00      1.00      1.00         5
           4       1.00      1.00      1.00        11
           5       1.00      1.00      1.00        33
           6       1.00      1.00      1.00        47
           7       1.00      1.00      1.00        98
           8       1.00      1.00      1.00       113
           9       1.00      1.00      1.00       127
          10       1.00      1.00      1.00       107
          11       1.00      1.00      1.00        95
          12       1.00      1.00      1.00        66
          13       1.00      1.00      1.00        39
          14       1.00      1.00      1.00        26
          15       1.00      1.00      1.00        18
          16       1.00      1.00      1.00        14
          17       1.00      1.00      1.00        10
          18       1.00      1.00      1.00         5
          19       1.00      1.00      1.00         8
          20       1.00      1.00      1.00         8
          21       1.00      1.00      1.00         2
          22       1.00      1.00      1.00         1
          23       1.00      1.00      1.00         2
          24       0.00      0.00      0.00         0
          29       0.00      0.00      0.00         1

    accuracy                           1.00       836
   macro avg       0.91      0.91      0.91       836
weighted avg       1.00      1.00      1.00       836
```

## Inference :

In [ ]:
```
SKLearn :

Logistic Regression   :  Accuracy : 25%          MacroAvg.: 15%
KNN Classifier        :  Accuracy : 25%          MacroAvg.: 15%
RandomForest Classifier : Accuracy : 96%         MacroAvg.: 70%
AdaBoost Classifier    :  Accuracy : 100%        MacrAvg. : 91%
SupportVectorclassifier: Accuracy : 100%         MacroAvg. : 91%

PySpark :
    Logistic Regression : Accuracy : 42.77%
    Random Forest Classifier : Accuracy : 97.44%

The RandomForest Classifier in Sklearn(96%) shows slightly less Accuracy compared

Conclusion :

By looking at the above Results, I can conclude that AdaBoost Classifier and SVC a
```

## PySpark :

In [ ]:
```
#set environment
import os
import sys

os.environ["SPARK_HOME"] = "/usr/hdp/current/spark2-client"
os.environ["PYLIB"] = os.environ["SPARK_HOME"] + "/python/lib"
# In below two lines, use /usr/bin/python2.7 if you want to use Python 2
os.environ["PYSPARK_PYTHON"] = "/usr/local/anaconda/bin/python"
os.environ["PYSPARK_DRIVER_PYTHON"] = "/usr/local/anaconda/bin/python"
sys.path.insert(0, os.environ["PYLIB"] +"/py4j-0.10.4-src.zip")
sys.path.insert(0, os.environ["PYLIB"] +"/pyspark.zip")
```

## Initiating the Spark Session :

In [ ]:
```
#import Sparksession driver
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Classification of Abalone Dataset") \
    .getOrCreate()
```

## Reading the Data into Pyspark :

In [ ]:
```
df = spark.read.csv('data/Abalone.csv',inferSchema=True)
df = df.toDF("Sex","Length","Diameter","Height","WholeWeight","ShuckedWeight","Vis
```

In [ ]:
```
import pyspark.sql.functions as F
from pyspark.sql.types import *
df = df.withColumn("Age", F.col("Rings") + 1.5)

#According to the MetaData,
# Adding Column 'Age' which is determined by adding 1.5 to the 'Rings' column

df.show(5)
```

In [ ]:
```
+---+------+--------+------+-----------+-------------+-------------+-----------+--
|Sex|Length|Diameter|Height|WholeWeight|ShuckedWeight|VisceraWeight|ShellWeight|Ri
+---+------+--------+------+-----------+-------------+-------------+-----------+--
|  M| 0.455|   0.365| 0.095|      0.514|       0.2245|        0.101|       0.15|
|  M|  0.35|   0.265|  0.09|     0.2255|       0.0995|       0.0485|       0.07|
|  F|  0.53|    0.42| 0.135|      0.677|       0.2565|       0.1415|       0.21|
|  M|  0.44|   0.365| 0.125|      0.516|       0.2155|        0.114|      0.155|
|  I|  0.33|   0.255|  0.08|      0.205|       0.0895|       0.0395|      0.055|
+---+------+--------+------+-----------+-------------+-------------+-----------+--
only showing top 5 rows
```

In [ ]:
```
for col in df.columns:
    print("no. of cells in column", col, "with null values:", df.filter(df[col].is
```

In [ ]:
```
for col in df.columns:
    print("no. of cells in column", col, "with null values:", df.filter(df[col].is

no. of cells in column Sex with null values: 0
no. of cells in column Length with null values: 0
no. of cells in column Diameter with null values: 0
no. of cells in column Height with null values: 0
no. of cells in column WholeWeight with null values: 0
no. of cells in column ShuckedWeight with null values: 0
no. of cells in column VisceraWeight with null values: 0
no. of cells in column ShellWeight with null values: 0
no. of cells in column Rings with null values: 0
no. of cells in column Age with null values: 0
```

## Label Encoding :

In [ ]:
```
#Label encoder
from pyspark.ml.feature import StringIndexer
indexed = df
for col in ["Sex"]:
    stringIndexer = StringIndexer(inputCol=col, outputCol=col+"_encoded")
    indexed = stringIndexer.fit(indexed).transform(indexed)
indexed.show()
```

In [ ]:

```
+---+------+--------+------+-----------+------------+-------------+-----------+--
|Sex|Length|Diameter|Height|WholeWeight|ShuckedWeight|VisceraWeight|ShellWeight|Ri
+---+------+--------+------+-----------+------------+-------------+-----------+--
|  M| 0.455|   0.365| 0.095|      0.514|      0.2245|        0.101|       0.15|
|  M|  0.35|   0.265|  0.09|     0.2255|      0.0995|       0.0485|       0.07|
|  F|  0.53|    0.42| 0.135|      0.677|      0.2565|       0.1415|       0.21|
|  M|  0.44|   0.365| 0.125|      0.516|      0.2155|        0.114|      0.155|
|  I|  0.33|   0.255|  0.08|      0.205|      0.0895|       0.0395|      0.055|
|  I| 0.425|     0.3| 0.095|     0.3515|       0.141|       0.0775|       0.12|
|  F|  0.53|   0.415|  0.15|     0.7775|       0.237|       0.1415|       0.33|
|  F| 0.545|   0.425| 0.125|      0.768|       0.294|       0.1495|       0.26|
|  M| 0.475|    0.37| 0.125|     0.5095|      0.2165|       0.1125|      0.165|
|  F|  0.55|    0.44|  0.15|     0.8945|      0.3145|        0.151|       0.32|
|  F| 0.525|    0.38|  0.14|     0.6065|       0.194|       0.1475|       0.21|
|  M|  0.43|    0.35|  0.11|      0.406|      0.1675|        0.081|      0.135|
|  M|  0.49|    0.38| 0.135|     0.5415|      0.2175|        0.095|       0.19|
|  F| 0.535|   0.405| 0.145|     0.6845|      0.2725|        0.171|      0.205|
|  F|  0.47|   0.355|   0.1|     0.4755|      0.1675|       0.0805|      0.185|
|  M|   0.5|     0.4|  0.13|     0.6645|       0.258|        0.133|       0.24|
|  I| 0.355|    0.28| 0.085|     0.2905|       0.095|       0.0395|      0.115|
|  F|  0.44|    0.34|   0.1|      0.451|       0.188|        0.087|       0.13|
|  M| 0.365|   0.295|  0.08|     0.2555|       0.097|        0.043|        0.1|
|  M|  0.45|    0.32|   0.1|      0.381|      0.1705|        0.075|      0.115|
+---+------+--------+------+-----------+------------+-------------+-----------+--
only showing top 20 rows
```

### Vector Assembling :

In [ ]:

```
#all the independent variables need to be packed into one column of vector type
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=["Length","Diameter","Height","WholeWeight",
                            outputCol="features")
feature_vec=assembler.transform(df).select('features','Rings')
feature_vec.show(5)
```

In [ ]:

```
+--------------------+-----+
|            features|Rings|
+--------------------+-----+
|[0.455,0.365,0.09...|   15|
|[0.35,0.265,0.09,...|    7|
|[0.53,0.42,0.135,...|    9|
|[0.44,0.365,0.125...|   10|
|[0.33,0.255,0.08,...|    7|
+--------------------+-----+
only showing top 5 rows
```

In [ ]:

```
#Count of target classes
feature_vec.groupBy('Rings').count().show()
#there is data imbalance
```

In [ ]:

```
+-----+-----+
|Rings|count|
+-----+-----+
|   26|    1|
|   27|    2|
|   12|  267|
|   22|    6|
|    1|    1|
|   13|  203|
|   16|   67|
|    6|  259|
|    3|   15|
|   20|   26|
|    5|  115|
|   19|   32|
|   15|  103|
|    9|  689|
|   17|   58|
|    4|   57|
|    8|  568|
|   23|    9|
|    7|  391|
|   10|  634|
+-----+-----+
only showing top 20 rows
```

### Splitting the Data :

In [ ]:

```
# Split the data into train and test sets
train_data, test_data = feature_vec.randomSplit([.75,.25],seed=0)
```

### Logistic Regression :

In [ ]:

```
from pyspark.ml.classification import LogisticRegression

# Create initial LogisticRegression model
lr = LogisticRegression(labelCol="Rings", featuresCol="features",
                        maxIter=100, regParam=0.0001, family="multinomial",
                        elasticNetParam=0.0)

# Train model with Training Data
lrModel = lr.fit(train_data)
predictions = lrModel.transform(test_data)
predictions.printSchema()
```

In [ ]:
```
root
 |-- features: vector (nullable = true)
 |-- Rings: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = true)
```

In [ ]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

evaluator = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol
evaluator.evaluate(predictions)
```

In [ ]:
```
0.5142576204523107
```

### Hyper Parameter tuning :

In [ ]:
```
#Grid Search
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = (ParamGridBuilder()\
             .addGrid(lr.regParam,[0.001,0.01,0.1,1])\
             .addGrid(lr.elasticNetParam,[0.0,0.5,1.0])\
             .build())

# Create 4-fold CrossValidator
cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluato

cvModel = cv.fit(train_data)
```

In [ ]:
```
#Best Model Params
score_params_list = list(zip(cvModel.avgMetrics, cvModel.getEstimatorParamMaps()))
max(score_params_list,key=lambda item:item[0])
```

In [ ]:
```
(0.39563360878492154,
 {Param(parent='LogisticRegression_4680a14087fe8f9ed89a', name='regParam', doc='re
  Param(parent='LogisticRegression_4680a14087fe8f9ed89a', name='elasticNetParam',
```

In [ ]:
```
predictions = cvModel.bestModel.transform(test_data)
```

In [ ]:
```
evaluator.evaluate(predictions)
```

In [ ]:
```
0.4277286135693215
```

### Random Forest Classifier :

In [ ]:

```python
from pyspark.ml.classification import RandomForestClassifier


model = RandomForestClassifier(labelCol='Rings', featuresCol="features",
                        maxDepth=15, minInfoGain=0.001, seed=0, numTrees=110)
rfModel = model.fit(train_data)

#Evaulation of the Model
predictions = rfModel.transform(test_data)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
evaluator = MulticlassClassificationEvaluator(predictionCol='prediction', labelCol
evaluator.evaluate(predictions)
```

In [ ]:

```
0.9744346116027532
```

## Hyper Parameter Tuning :

In [ ]:

```python
#Grid Search
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
model = RandomForestClassifier(labelCol='Rings', featuresCol="features",
                        minInfoGain=0.001, seed=0)
paramGrid = (ParamGridBuilder()\
             .addGrid(model.maxDepth,[13,14,15,16])\
             .addGrid(model.numTrees,[50,60,70,80])\
             .build())

# Create 4-fold CrossValidator
cv = CrossValidator(estimator=model, estimatorParamMaps=paramGrid, evaluator=evalu
```

In [ ]:

```python

cvModel = cv.fit(train_data)
```

In [ ]:

```python
#Best Model Params
score_params_list = list(zip(cvModel.avgMetrics, cvModel.getEstimatorParamMaps()))
max(score_params_list,key=lambda item:item[0])
```

In [ ]:

```
(0.9579278251236223,
 {Param(parent='RandomForestClassifier_4a1ba8f89b1889a6b01c', name='maxDepth', doc
  Param(parent='RandomForestClassifier_4a1ba8f89b1889a6b01c', name='numTrees', doc
```

In [ ]:

```python
predictions = cvModel.bestModel.transform(test_data)
```

In [ ]:

```python
evaluator.evaluate(predictions)
```

In [ ]:
```
0.9744346116027532
```

In [ ]:
```
spark.stop()
```

### Inference :

In [ ]:
```


PySpark :
    Logistic Regression :       Accuracy : 42.77 %
    Random Forest Classifier :  Accuracy : 97.44%

The Bestparameters for RandomForest Classifier in Pyspark needs maxdepth of 16 ,ar
and its Accuracy is 97.44% , which is huge jump from logistic regression accuracy

SKLearn :

Logistic Regression  :    Accuracy : 25%        MacroAvg.: 15%
KNN Classifier       :    Accuracy : 25%        MacroAvg.: 15%
RandomForest Classifier : Accuracy : 96%        MacroAvg.: 70%
AdaBoost Classifier   :   Accuracy : 100%       MacrAvg. : 91%
SupportVectorclassifier:  Accuracy : 100%       MacroAvg. : 91%

The RandomForest Classifier in Sklearn(96%) shows slightly less Accuracy compared

Conclusion :

By looking at the above Results, I can conclude that SVC and AdaBoost Classifier i
```