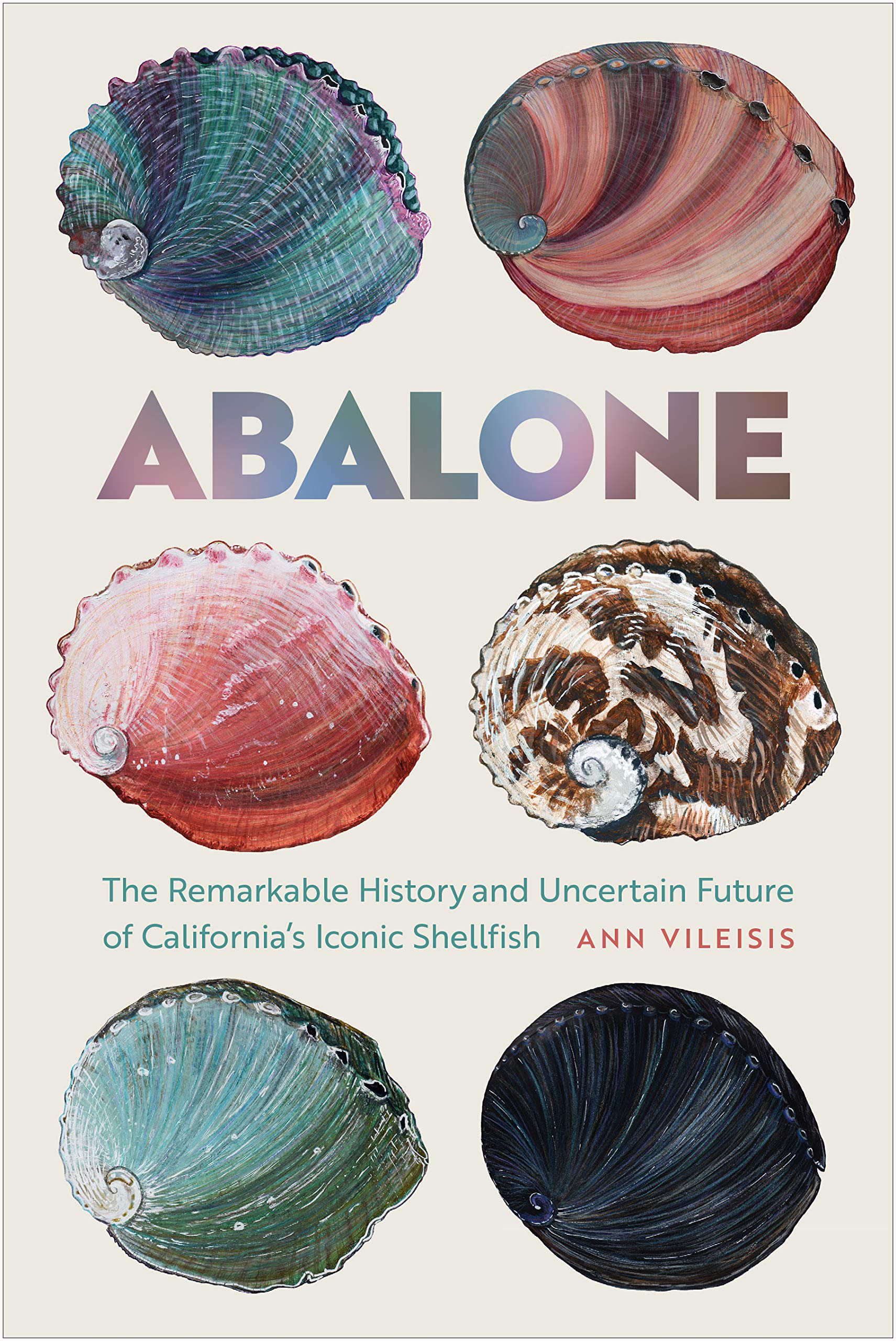
Abalone Case study



In this blog, I’ll go through the whole process of creating a machine learning model on Ablone dataset.

You can download the dataset from the given below link:

<https://github.com/dsrscientist/dataset1/blob/master/abalone.csv>

**Let’s know what is Ablone?**

**Abalone:**

Abalone are marine snails. Their taxonomy puts them in the family Haliotidae, which contains only one genus. Haliotis, which once contained six subgenera. These subgenera have become alternate representations of Haliotis. The number of species recognized worldwide ranges between 30 and 130 with over 230 sepcies-level taxa described. The most comprehensive treatment of the family considers 56 species valid with 18 additional subspecies. The shells of abalone have a low, open spiral structure and are characterized by several open respiratory pores in a row near the shell’s outer edge. The thick inner layer of the shell is composed of nacre(mother-of-pearl), which in many species is highly iridescent, giving rise to a range strong, changeable colors, which make the shells attractive to humans as decorative objects, jewelery and as a source of colorful mother-of-pearl. The flesh of abalones is widely considered to be a desirable food and is consumed raw or cooked by a variety of cultures.

**Problem Description:**

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 boaring and time conusming task. Other measurements, which are easier to obtain, are used to predict the age.

**Importing the necessary libraries:**

import numpy as np

import pandas as pd

import seaborn as sb

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.pipeline import Pipeline

from sklearn. preprocessing import MinMaxScaler

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import AdaBoostRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.metrics import mean\_squared\_error

import warnings

warnings.filterwarnings('ignore')

We have imported the all important libraries now load the dataset into the jupyter notebook.

**Code:**

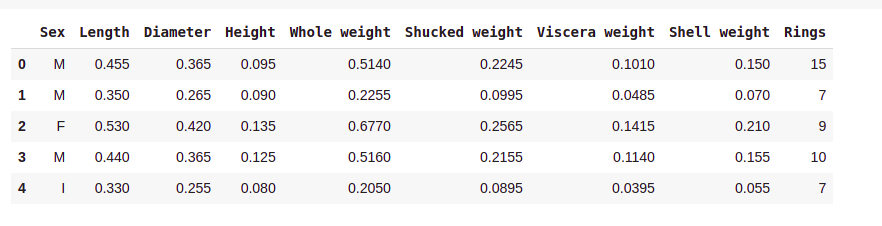
dataset=pd.read\_csv('AbaloneCaseStudy.csv')

### **Data Analysis**

Displaying the first row in our dataset.

**Code:**

dataset.head(5)



let's first compute the target variable of the problem 'Age' and assign it to the dataset. Age=1.5+Rings.

To do this first we have to drop Rings column and add age column of the abalone.

**Code:**

dataset['age']=dataset['Rings']+1.5

dataset.drop('Rings',axis=1,inplace=True)

**Now, let's check the shape of the dataset using .shape method.**

**Code:**

print(dataset.shape)

It would display the total number of row and column of the dataset.

Before proceding further we have to check is there any null value available in the dataset. For checking the null value use the following code:

dataset.isna().sum()

**output:**

Sex 0

Length 0

Diameter 0

Height 0

Whole weight 0

Shucked weight 0

Viscera weight 0

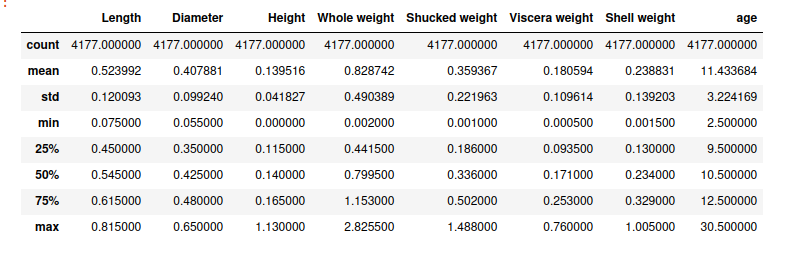
Shell weight 0

age 0

dtype: int64

we don’t have null or empty cells in our dataset so no need to apply imputer library to handle Empty Cell or empty values.

## **Descriptive Statistic describe each attribute**



Attribute age confidence interval range starts from 9 to 12 so it is not starting from zero so it will positive skew.

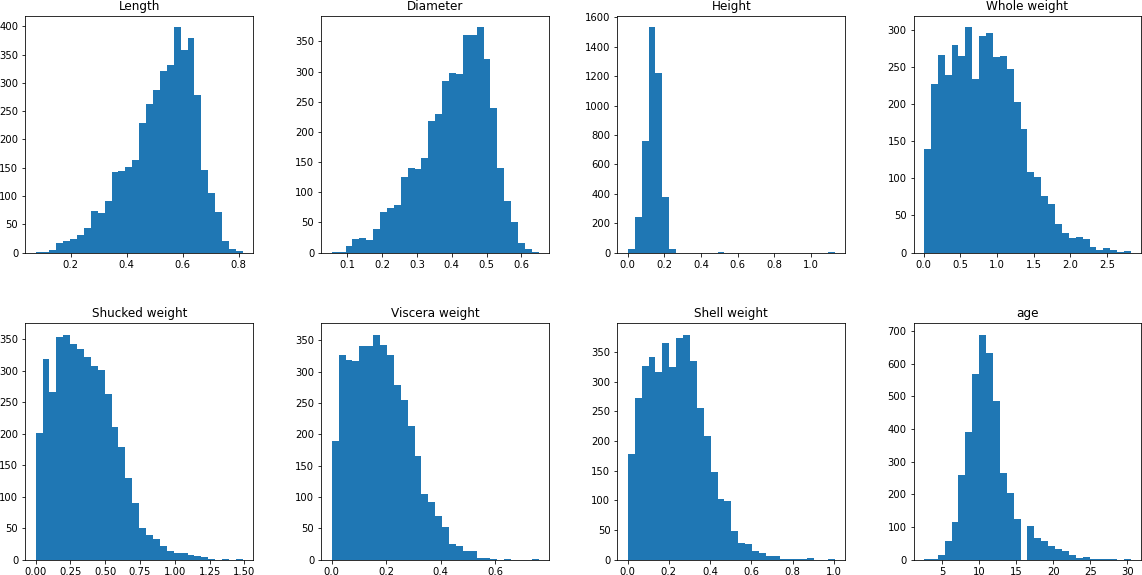
### Histrogram visualisation for each attribute to know about what kind of distribution it is?

**Code:**

dataset.hist(figsize=(20,10),grid=False, layout=(2,4),bins=30)

plt.show()

**Output:**



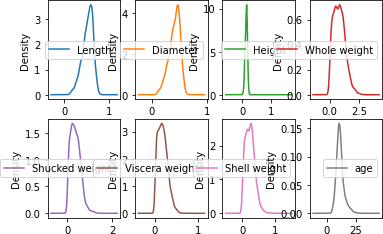
### density visualisation for all attributes

**Code:**

dataset.plot(kind='density',layout=(2,4), sharex=False, sharey=False,subplots=True, grid=False)

plt.show()

**Output:**

though features are not normaly distributed, are close to normality.

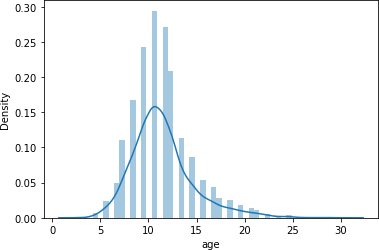
### Histogram descriptive visualisation distribution for output attribute age:

**Code:**

sb.distplot(dataset['age'])

plt.show()

**Output:**



as per above plot we got normal distribution for discreate output age column values.

## Analyzing the correlations with output and each input attribute and find outliers.

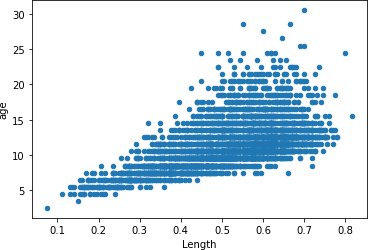
## histogram visualisation for age output and Longest shell measuruement input attributes

**Code:**

data\_plot=pd.concat([dataset['age'],dataset['Length']],axis=1)

data\_plot.plot.scatter(x='Length',y='age')

**Output:**



We don't have any outliers for age output and length input attributes as per above visualisation plot.

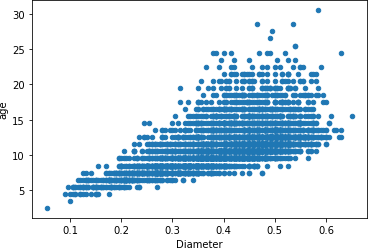
### Histrogram visualisation for age output and diameter input attributes.

**Code:**

data\_plot=pd.concat([dataset['Diameter'],dataset['age']],axis=1)

data\_plot.plot.scatter(x='Diameter',y='age')

**output:**



we don’t have any outliers for age output and Diameter input attributes as per above visulisation plot.

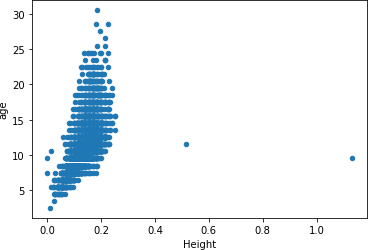
### Histogram visualisation for age output and height input attributes.

**Code:**

data\_plot=pd.concat([dataset['Height'],dataset['age']],axis=1)

data\_plot.plot.scatter(x='Height',y='age')

**output:**



Here we got 2 outlier point in between 0.4 to 1.2 values. Perhaps with this outlier value will effect the performance for our algorithm. so now we are going to remove this 2 outliers values.

### Removing outlier for age output and Height input attributes

**Code:**

dataset=dataset.drop(dataset[(dataset['Height']>0.4) & (dataset['Height']<1.4)].index)

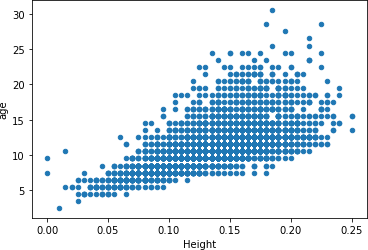
### Visualising again to know those outlier removed or not

**Code:**

data\_plot=pd.concat([dataset['Height'],dataset['age']],axis=1)

data\_plot.plot.scatter(x='Height',y='age')

**output:**



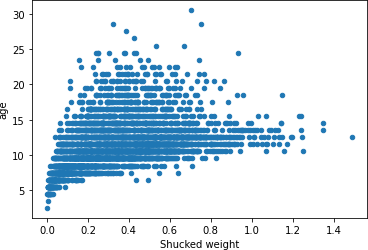
### Histogram visualisation for age output and whole weight input attributes.

**Code:**

data=pd.concat([dataset['Whole weight'],dataset['age']],axis=1)

data.plot.scatter(x='Whole weight',y='age')

**Output:**



As we can see above plot with high value we got less rings, so it was near to outlier so we are removing from 1.2 to 1.6 range values for rings output and shucked weight input attribute.

#### Removing the outlier values for age output and shucked weight input attribute

**Code:**

dataset=dataset.drop(dataset[(dataset['Shucked weight']>1.2)&(dataset['Shucked weight']<1.5)].index)

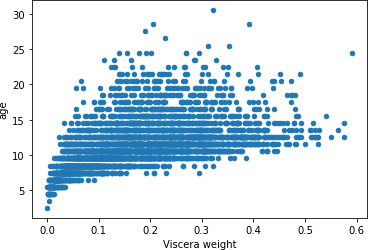
Visualising again to know whether those outlier values removed or not

**code:**

data=pd.concat([dataset['Shucked weight'],dataset['age']],axis=1)

data.plot.scatter(x='Shucked weight',y='age')

**output:**



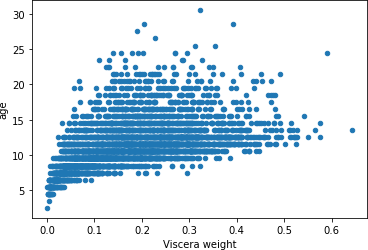
**Histogram visualisation for viscera weight input attribute and age output attribute**

**Code:**

data=pd.concat([dataset['Viscera weight'],dataset['age']],axis=1)

data.plot.scatter(x='Viscera weight',y='age')

**Output:**



We got again 1 outlier value for viscera weight input attribute and age output attribute. So now we are going to remove in between 0.6 to 15 value box.

## Removing the outlier value lies in between 0.6 to 15

## **Code:**

## dataset=dataset.drop(dataset[(dataset['Viscera weight']>0.6)&(dataset['Viscera weight']<15)].index)

## **Output:**

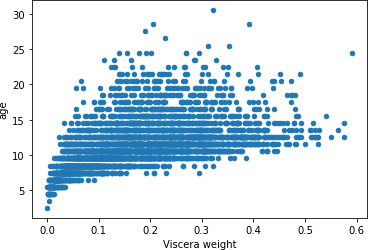
#### **Visualising again to check whether those outliers removed or not**

**Code:**

data=pd.concat([dataset['Viscera weight'],dataset['age']],axis=1)

data.plot.scatter(x='Viscera weight',y='age')

**Output:**



## **Histogram visualisation for shell weight input attribute and age output attribute.**

## **Code:**

## data=pd.concat([dataset['Shell weight'],dataset['age']],axis=1)

## data.plot.scatter(x='Shell weight',y='age')

## **Output:**

## 

We are not sure whether those values lies below the threshold 0.9 to 15 values we are going to remove those values.

## Removing **the outliers for shell weight input attribute and age output attribute.**

## **Code:**

## dataset=dataset.drop(dataset[(dataset['Shell weight']>0.9)&(dataset['Shell weight']<15)].index)

#### **Visualising again to check all outlier below the threshold removed or not.**

## Code:

## data=pd.concat([dataset['Shell weight'],dataset['age']],axis=1)

## data.plot.scatter(x='Shell weight',y='age')

## **Output:**

## 

## Correlation values between each attributes using heatmap.

## **Code:**

## corr\_value=dataset.corr()

## sb.heatmap(corr\_value,square=True)

## **Output:**

## 

## **Data Cleaning**

No need to apply cleaning to our dataset. Because we don’t have any error or empty or null values.

**Spliting the dataset into input and output attribute.**

**Code:**

x=dataset.drop(columns='age')

y=dataset['age']

**Label Encoder**

Here we are going to Encode the categorical value into numerical values.

**Code:**

labelencoder=LabelEncoder()

x['Sex']=labelencoder.fit\_transform(x['Sex'])

## **Splitting dataset into training and test set**

## Let’s split dataset into train and test.

## **Code:**

## x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,train\_size=0.7,test\_size=0.3,random\_state=5)

### **Classification Modelling**

**Code:**

n\_neighbors=5

models=[]

models.append(('LinearRegression',LinearRegression()))

models.append(('knn',KNeighborsRegressor(n\_neighbors=n\_neighbors)))

models.append(('SVR',SVR()))

models.append(("decision\_tree",DecisionTreeRegressor()))

#Evaluating Each model

names=[]

predictions=[]

error='neg\_mean\_squared\_error'

for name,model in models:

fold=KFold(n\_splits=10,random\_state=0)

result=cross\_val\_score(model,x\_train,y\_train,cv=fold,scoring=error)

predictions.append(result)

names.append(name)

msg="%s : %f (%f)"%(name,result.mean(),result.std())

print(msg)

# Visualizing the Model accuracy

fig=plt.figure()

fig.suptitle("Comparing Algorithms")

axis=fig.add\_subplot(111)

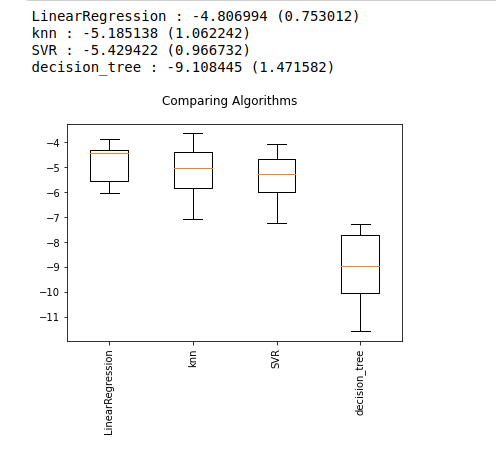
plt.boxplot(predictions)

axis.set\_xticklabels(names)

plt.xticks(rotation='90')

plt.show()

**Output:**

We listing top two accuracy models

* 1. LinearRegression: mean -4.857541(0.499037)
  2. Knn: mean:5.224915 std:(0.714804) Now we are applying regularisation tuning to LinearRegr

Before going to tuning those 2 algorithms we need to apply feature scaling to our dataset and predicting again then we will take top 2 algorithm which is scaled one.Because scaled algorithm always gives best predictions.

# spot checking and comparing algorithms with minmaxscalar scalar

**Code:**

pipelines=[]

pipelines.append(('scaled LinearRegression',Pipeline([('scaler',MinMaxScaler()),('LinearRegression',LinearRegression())])))

pipelines.append(('scaled KNN',Pipeline([('scaler',MinMaxScaler()),('KNN',KNeighborsRegressor(n\_neighbors=n\_neighbors))])))

pipelines.append(('scaled SVR',Pipeline([('scaler',MinMaxScaler()),('SVR',SVR())])))

pipelines.append(('scaled DecisionTree',Pipeline([('scaler',MinMaxScaler()),('decision',DecisionTreeRegressor())])))

# Evaluating Each model

names=[]

predictions=[]

for name,model in models:

fold=KFold(n\_splits=10,random\_state=0)

result=cross\_val\_score(model,x\_train,y\_train,cv=fold,scoring=error)

predictions.append(result)

names.append(name)

msg="%s : %f (%f)"%(name,result.mean(),result.std())

print(msg)

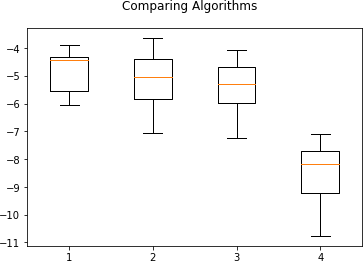
# Visualizing the Model accuracy

fig=plt.figure()

fig.suptitle("Comparing Algorithms")

plt.boxplot(predictions)

plt.show()



We listing top two accuracy models, After applying feature scaling performance get increased

Before Applying Feature Scaling

LinearRegression : mean -4.857541 (0.499037) knn : mean:-5.224915 std:(0.714804)

After Applying Feature Scaling

LinearRegression : mean -4.857541 (0.499037) knn : mean:-5.224915 std:(0.714804) Now we are applying regularisation tuning to decision tree and supoort vector algorithms

**Linear Regression Algorithm tunning.**

**Code:**

import numpy as np

from sklearn.model\_selection import GridSearchCV

scaler=MinMaxScaler().fit(x\_train)

rescaledx=scaler.transform(x\_train)

param\_grid=dict()

model=LinearRegression()

fold=KFold(n\_splits=10,random\_state=5)

grid=GridSearchCV(estimator=model,param\_grid=param\_grid,scoring=error,cv=fold)

grid\_result=grid.fit(rescaledx,y\_train)

print("Best: %f using %s "%(grid\_result.best\_score\_,grid\_result.best\_params\_))

**Output:**

Best: -4.806994

**KNN Regression Tuning**

**code:**

import numpy as np

from sklearn.model\_selection import GridSearchCV

scaler=MinMaxScaler().fit(x\_train)

rescalex=scaler.transform(x\_train)

n\_neighbors=[3,4,5,6,7,8,9,10,15,20]

# With degree our model fit to training set overfitting so better not use for all algorithms except polyomial

#degree=[1,2,3,4,5,6,7,8,9]

param\_grid=dict(n\_neighbors=n\_neighbors)

model=KNeighborsRegressor()

fold=KFold(n\_splits=10,random\_state=5)

grid=GridSearchCV(estimator=model,param\_grid=param\_grid,scoring=error,cv=fold)

grid\_result=grid.fit(rescalex,y\_train)

print("Best: %f using %s "%(grid\_result.best\_score\_,grid\_result.best\_params\_))

**Output:**

Best: -5.090146 using {'n\_neighbors': 10}

We listing tunned two accuracy models

Linear Regression Algorithm Best: -4.857541 using {}

KNN Regression Algorithm Best: -5.125680 using {'n\_neighbors': 15}

**Ensemble and Boosting algorithm to imporve performance.**

**Code:**

ensembles=[]

ensembles.append(('scaledAB',Pipeline([('scale',MinMaxScaler()),('AB',AdaBoostRegressor())])))

ensembles.append(('scaledGBC',Pipeline([('scale',MinMaxScaler()),('GBc',GradientBoostingRegressor())])))

ensembles.append(('scaledRFC',Pipeline([('scale',MinMaxScaler()),('rf',RandomForestRegressor(n\_estimators=10))])))

ensembles.append(('scaledETC',Pipeline([('scale',MinMaxScaler()),('ETC',ExtraTreesRegressor(n\_estimators=10))])))

Output:

scaledAB : -7.598311 (0.835692)

scaledGBC : -4.724734 (0.960913)

scaledRFC : -5.241283 (1.055150)

scaledETC : -5.321892 (0.954267)

#### Visualizing the compared Ensemble Algorithms

**Code:**

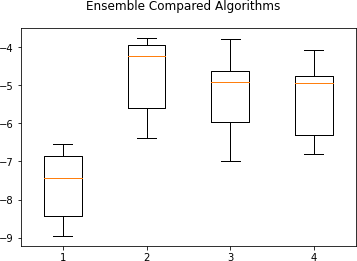
fig=plt.figure()

fig.suptitle('Ensemble Compared Algorithms')

plt.boxplot(results)

plt.show()

Output:



We listing tunned two accuracy models

Gradient Boosting Algorithm with mean -4.824129 and std(0.622529)

Random Forest Regressor Algorithm with mean -5.198858 and std(0.586180)

Now we are going to tuning this 2 algorithms.

**Code:**

scaler=MinMaxScaler().fit(x\_train)

rescaledx=scaler.transform(x\_train)

n\_estimators=[5,10,15,20,25,30,40,50,75,100]

param\_grid=dict(n\_estimators=n\_estimators)

model=RandomForestRegressor()

fold=KFold(n\_splits=10,random\_state=5)

grid=GridSearchCV(estimator=model,param\_grid=param\_grid,scoring=error,cv=fold)

grid\_result=grid.fit(rescaledx,y\_train)

print("Best: %f using %s "%(grid\_result.best\_score\_,grid\_result.best\_params\_))

#### **Gradient Boosting Algorithm Tuning**

Code:

scaler=MinMaxScaler().fit(x\_train)

rescaledx=scaler.transform(x\_train)

learning\_rate=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]

n\_estimators=[10,15,20,25,30,40,50,75,100,150,200]

param\_grid=dict(learning\_rate=learning\_rate,n\_estimators=n\_estimators)

model=GradientBoostingRegressor()

fold=KFold(n\_splits=10,random\_state=5)

grid=GridSearchCV(estimator=model,param\_grid=param\_grid,scoring=error,cv=fold)

grid\_result=grid.fit(rescaledx,y\_train)

print("Best: %f using %s "%(grid\_result.best\_score\_,grid\_result.best\_params\_))

We listing top four tunned algorithms.

Linear Regression Algorithm Best: -4.857541 using {} Regression Algorithm Best: -5.125680 using {'n\_neighbors': 15} Random Forest Regressor Algorithm Best: -4.797185 using {'n\_estimators': 75} Gradient Boosting Regressor Algorithm -4.802066 using {'learning\_rate': 0.2, 'n\_estimators': 75} As per above four tunned algorithm Decision Tree Classifier Algorithm.

**Finalize Model:**

we finalized the Random Forest Regressor algorithm and evaluate the model for Abalone Physical meansurements.

**Code:**

scaler=MinMaxScaler().fit(x\_train)

scaler\_x=scaler.transform(x\_train)

model=RandomForestRegressor(n\_estimators=75)

model.fit(scaler\_x,y\_train)

#Transform the validation test set data

scaledx\_test=scaler.transform(x\_test)

y\_pred=model.predict(scaledx\_test)

accuracy=mean\_squared\_error(y\_test,y\_pred)

print("accuracy :",accuracy)

Output:

accuracy : 4.889000675555556

We got Regression accuracy

training set accuracy: 4.797185

testing accuracy :4.622583966920101.

That’s all in this Blog.

**Thank you.**