1. Download the dataset

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew
%matplotlib inline

from sklearn.preprocessing import  StandardScaler
from sklearn.model_selection import  train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import  RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import  GradientBoostingRegressor
from sklearn.linear_model import  Ridge
from sklearn.svm import SVR
import warnings
```

2. Load the dataset into the tool.

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

```
data['age'] = data['Rings']+1.5
data.drop('Rings', axis = 1, inplace = True)
```

```
print('This dataset has {} observations with {} features.'.format(data.shape[0], data.shap
```

```
    This dataset has 4177 observations with 9 features.
```

```
data.info()
```

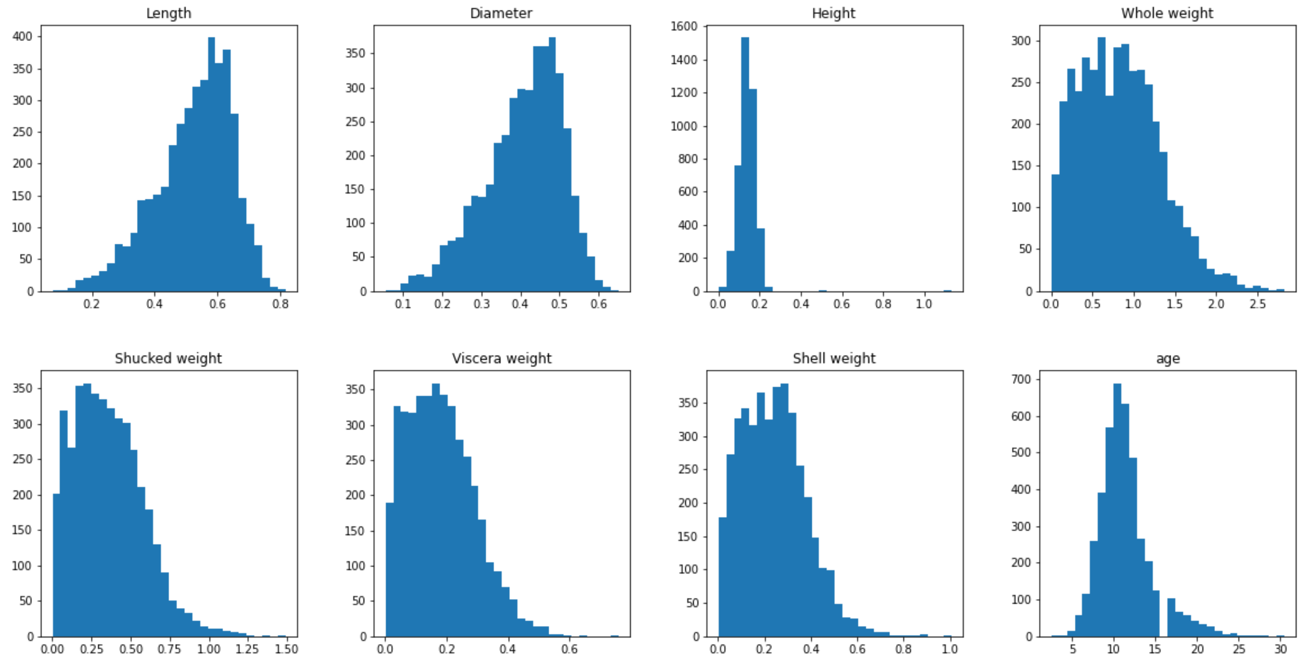
```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 4177 entries, 0 to 4176
    Data columns (total 9 columns):
     #   Column          Non-Null Count  Dtype  
    ---  ------          --------------  -----  
     0   Sex             4177 non-null   object 
     1   Length          4177 non-null   float64
     2   Diameter        4177 non-null   float64
     3   Height          4177 non-null   float64
     4   Whole weight    4177 non-null   float64
     5   Shucked weight  4177 non-null   float64
     6   Viscera weight  4177 non-null   float64
```

```
 7   Shell weight    4177 non-null   float64
 8   age             4177 non-null   float64
dtypes: float64(8), object(1)
memory usage: 293.8+ KB
```

Perform Below Visualizations:Univariate Analysis, Bi-Variate Analysis, Multi-Variate Analysis

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

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccc0d95d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccdf2a650>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccc076150>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccc02b750>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccbfe2d50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccbfa5390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccbf5ba10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f9ccbf12f90>]],
      dtype=object)
```



```
numerical_features = data.select_dtypes(include=[np.number]).columns
categorical_features = data.select_dtypes(include=[np.object]).columns
numerical_features
```

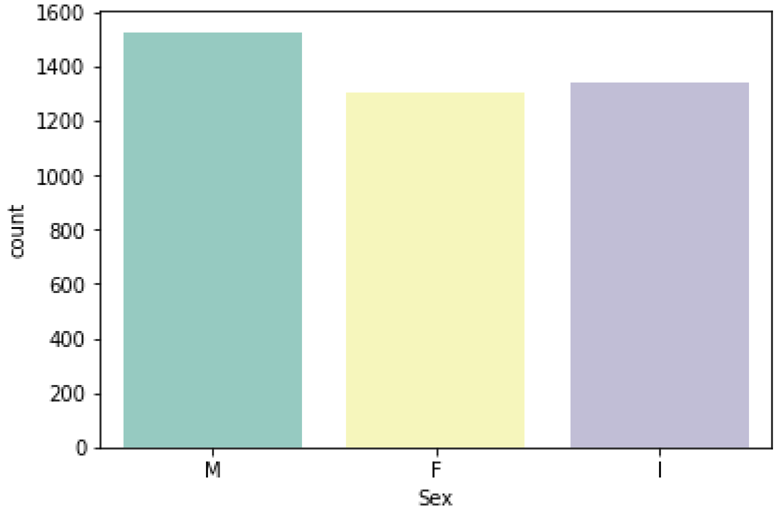
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: DeprecationWarning: `
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re

Index(['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'age'],
      dtype='object')
```

```
categorical_features
```

```
Index(['Sex'], dtype='object')
```

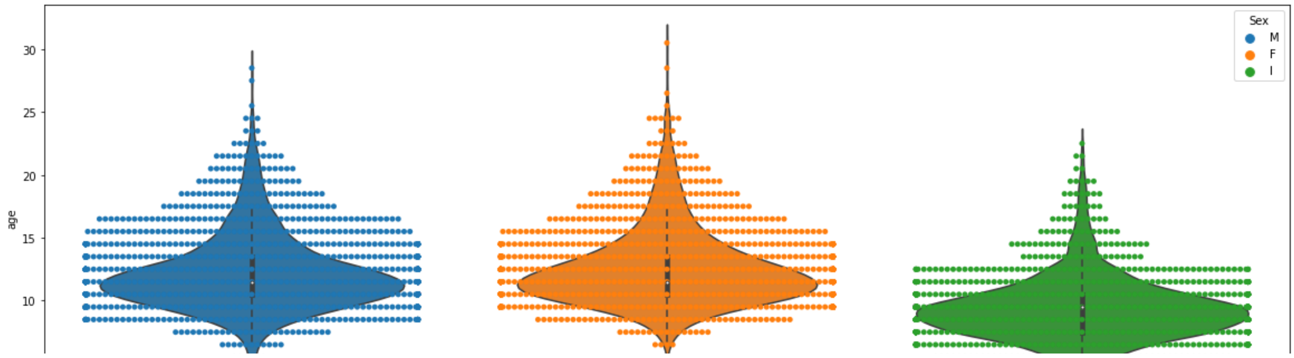
```
sns.countplot(x = 'Sex', data = data, palette="Set3")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ccb5df250>
```




```
plt.figure(figsize = (20,7))
sns.swarmplot(x = 'Sex', y = 'age', data = data, hue = 'Sex')
sns.violinplot(x = 'Sex', y = 'age', data = data)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 56.2
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 52.2
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 58.5
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ccdf2abd0>
```

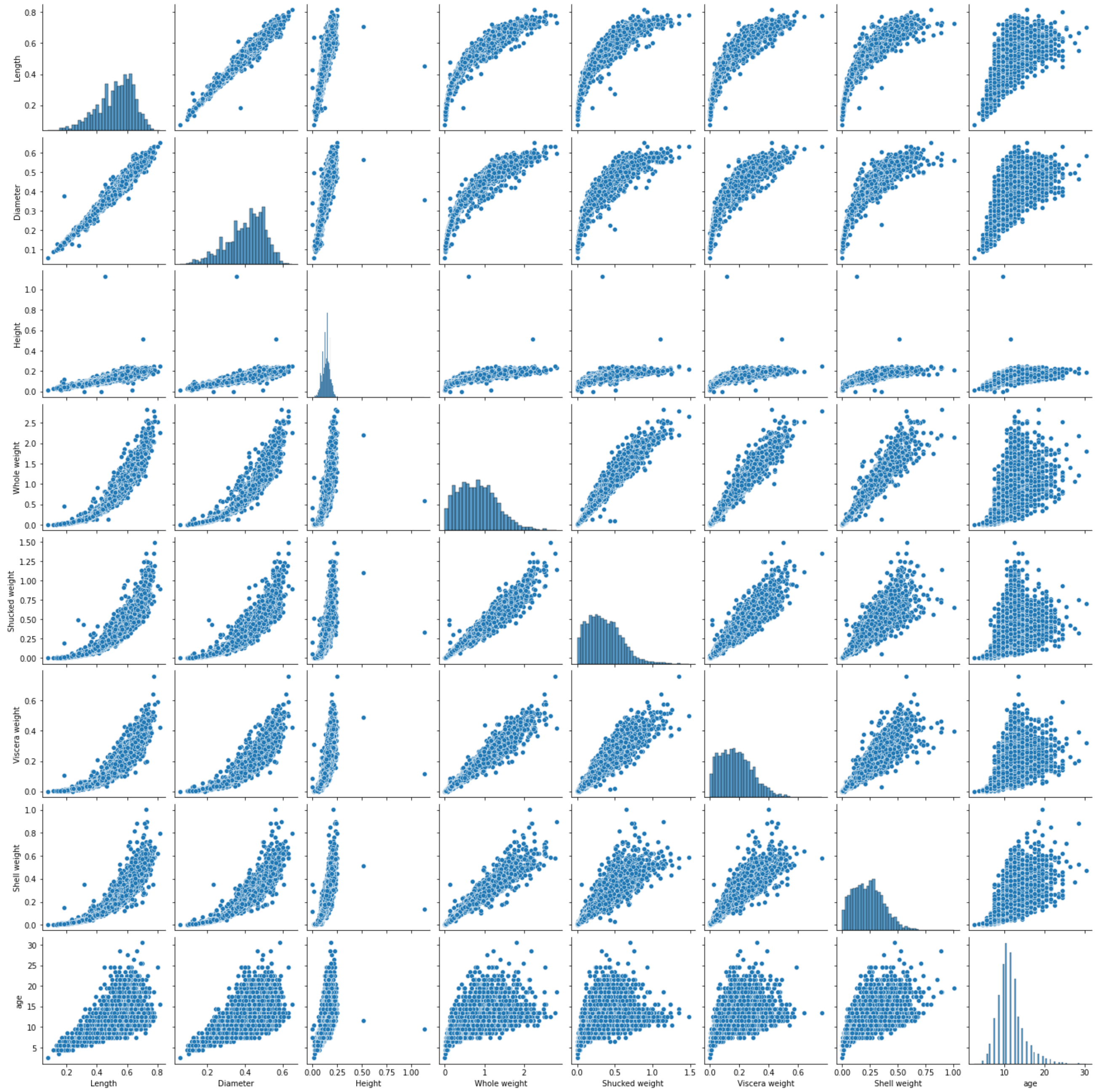



```
data.groupby('Sex')[['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'age']].mean().sort_values('age')
```

Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	age
I	0.427746	0.326494	0.107996	0.431363	0.191035	0.092010	0.128182	9.390462
M	0.561391	0.439287	0.151381	0.991459	0.432946	0.215545	0.281969	12.205497
F	0.579093	0.454732	0.158011	1.046532	0.446188	0.230689	0.302010	12.629304

```
sns.pairplot(data[numerical_features])
```

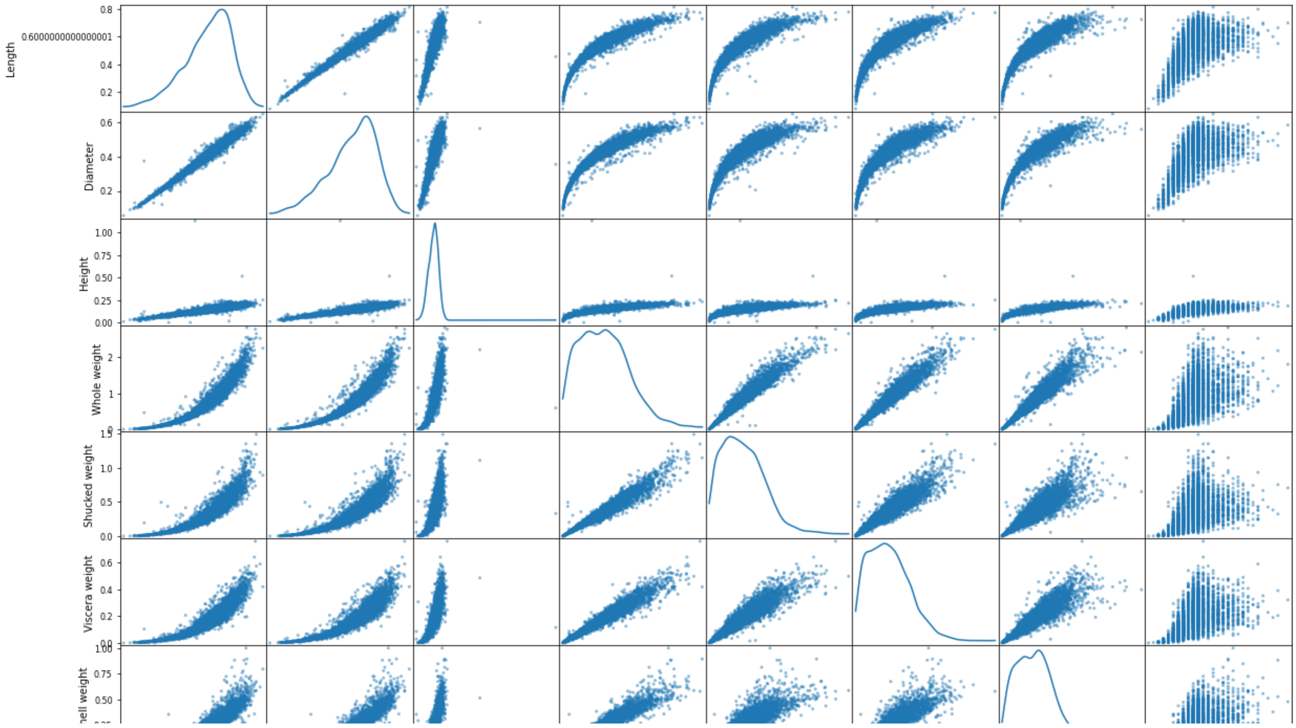
```
<seaborn.axisgrid.PairGrid at 0x7f9ccbc28390>
```



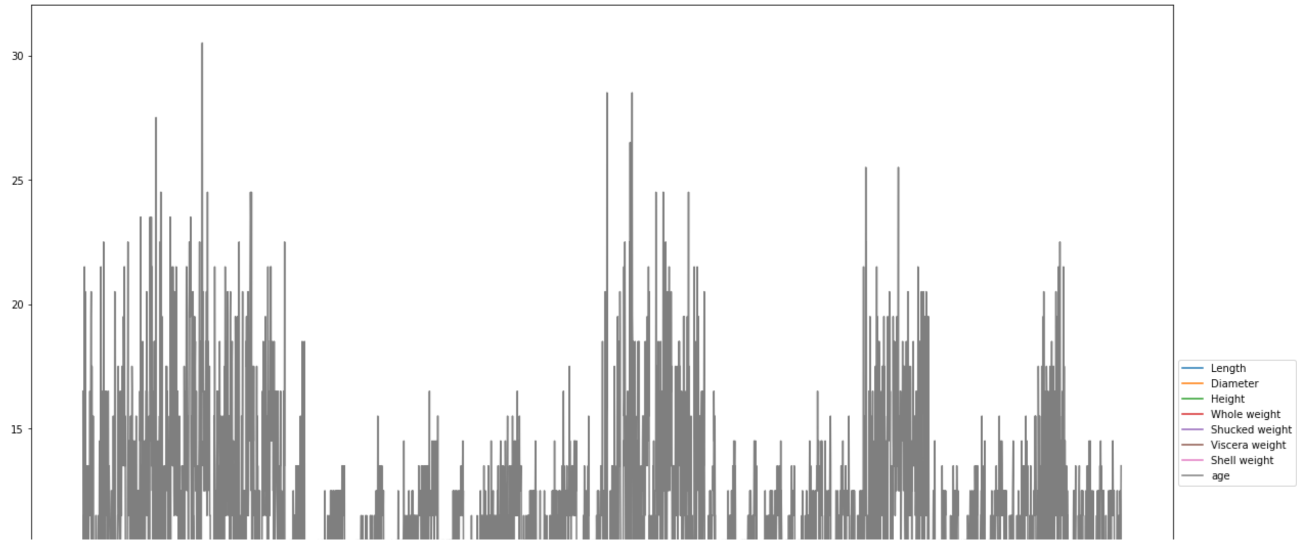

```
plt.figure(figsize=(20,7))
sns.heatmap(data[numerical_features].corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ce4c80dd0>
```


```
pd.plotting.scatter_matrix(data.loc[:, 'Sex':'age'], diagonal="kde",figsize=(20,15))
plt.show()
```




```
ax = data[['Sex','Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'age']].plot(figsize=(20,15))
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5));
```



4.Perform descriptive statistics on the dataset

```
data.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	41
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	

```
data['Sex'].describe()
```

```
count     4177
unique       3
top          M
freq      1528
Name: Sex, dtype: object
```

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

```
M    1528
I    1342
F    1307
Name: Sex, dtype: int64
```

```
data.kurtosis()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Droppi
  """Entry point for launching an IPython kernel.
Length            0.064621
Diameter         -0.045476
Height           76.025509
Whole weight     -0.023644
Shucked weight    0.595124
Viscera weight    0.084012
Shell weight      0.531926
age               2.330687
dtype: float64
```

```
skew_values = skew(data[numerical_features], nan_policy = 'omit')
dummy = pd.concat([pd.DataFrame(list(numerical_features), columns=['Features']),
           pd.DataFrame(list(skew_values), columns=['Skewness degree'])], axis = 1)
dummy.sort_values(by = 'Skewness degree' , ascending = False)
```

	Features	Skewness degree
2	Height	3.127694
7	age	1.113702
4	Shucked weight	0.718840
6	Shell weight	0.620704
5	Viscera weight	0.591640
3	Whole weight	0.530768
1	Diameter	-0.608979
0	Length	-0.639643

5.Check for Missing values and deal with them.

```
missing_values = data.isnull().sum().sort_values(ascending = False)
percentage_missing_values = (missing_values/len(data))*100
pd.concat([missing_values, percentage_missing_values], axis = 1, keys= ['Missing values',
```

	Missing values	% Missing
Sex	0	0.0
Length	0	0.0

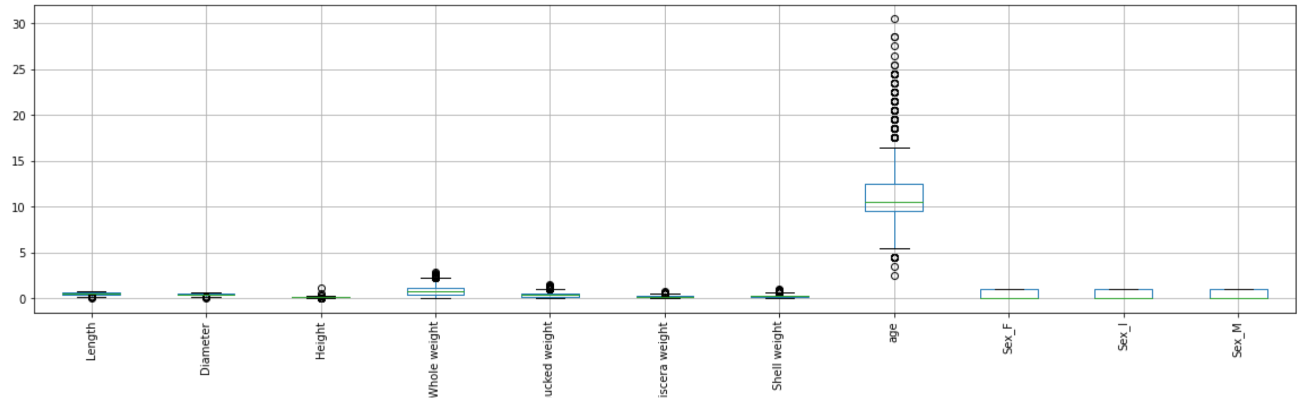
```
data.head
```

```
<bound method NDFrame.head of      Sex  Length  Diameter  Height  Whole weight  Shucked weight  \
0      M   0.455     0.365   0.095        0.5140          0.2245
1      M   0.350     0.265   0.090        0.2255          0.0995
2      F   0.530     0.420   0.135        0.6770          0.2565
3      M   0.440     0.365   0.125        0.5160          0.2155
4      I   0.330     0.255   0.080        0.2050          0.0895
...   ..     ...       ...     ...           ...             ...
4172   F   0.565     0.450   0.165        0.8870          0.3700
4173   M   0.590     0.440   0.135        0.9660          0.4390
4174   M   0.600     0.475   0.205        1.1760          0.5255
4175   F   0.625     0.485   0.150        1.0945          0.5310
4176   M   0.710     0.555   0.195        1.9485          0.9455

      Viscera weight  Shell weight   age
0             0.1010        0.1500  16.5
1             0.0485        0.0700   8.5
2             0.1415        0.2100  10.5
3             0.1140        0.1550  11.5
4             0.0395        0.0550   8.5
...              ...           ...   ...
4172          0.2390        0.2490  12.5
4173          0.2145        0.2605  11.5
4174          0.2875        0.3080  10.5
4175          0.2610        0.2960  11.5
4176          0.3765        0.4950  13.5

[4177 rows x 9 columns]>
```

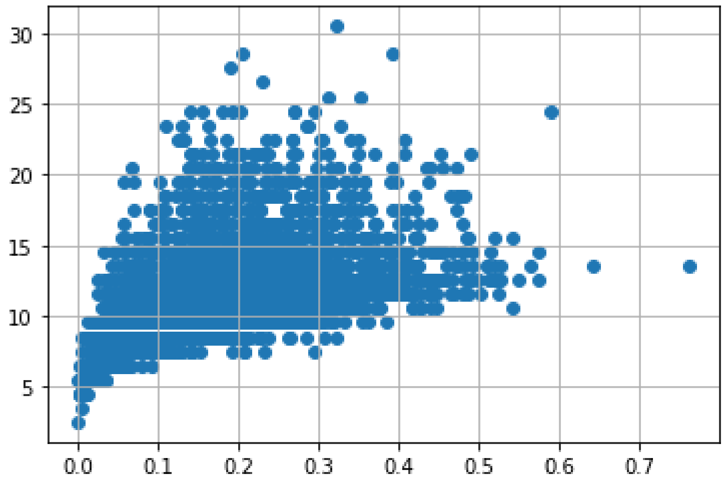
6.Find the outliers and replace them outliers

```
original_data = data.copy()
data = pd.get_dummies(data)
dummy_data = data.copy()
data.boxplot( rot = 90, figsize=(20,5))
```

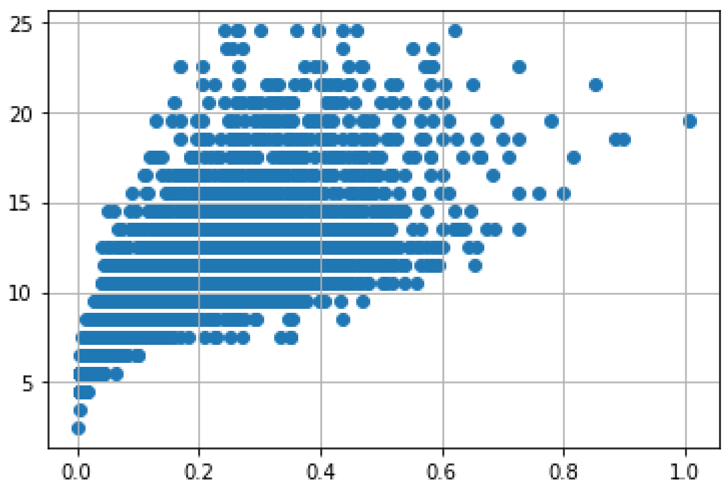
<matplotlib.axes._subplots.AxesSubplot at 0x7f9cc3f8a090>




```
var = 'Viscera weight'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```

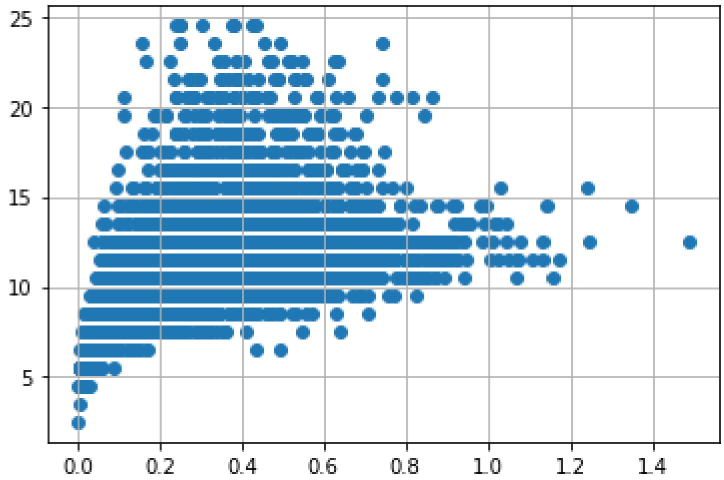



```
# outliers removal
data.drop(data[(data['Viscera weight']> 0.5) & (data['age'] < 20)].index, inplace=True)
data.drop(data[(data['Viscera weight']<0.5) & (data['age'] > 25)].index, inplace=True)

var = 'Shell weight'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```

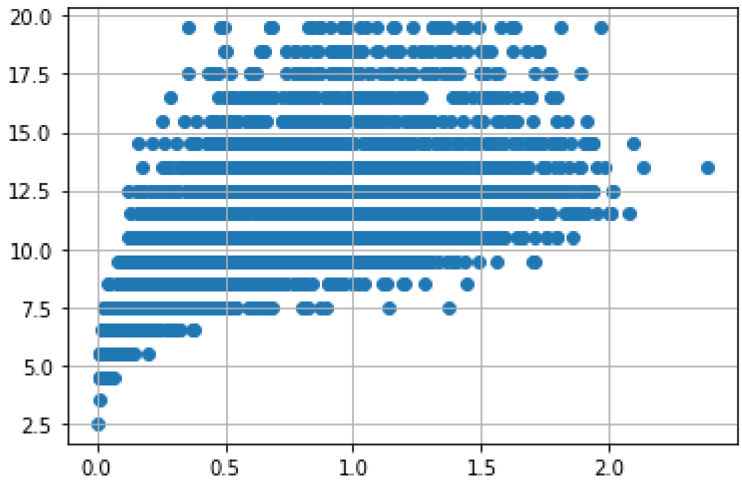



```
data.drop(data[(data['Shell weight']> 0.6) & (data['age'] < 25)].index, inplace=True)
```

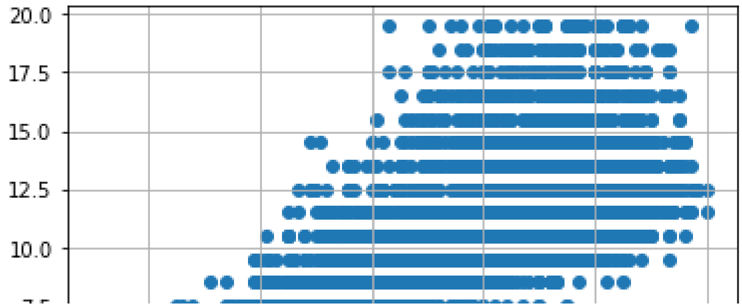
```
data.drop(data[(data['Shell weight']<0.8) & (data['age'] > 25)].index, inplace=True)

var = 'Shucked weight'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```



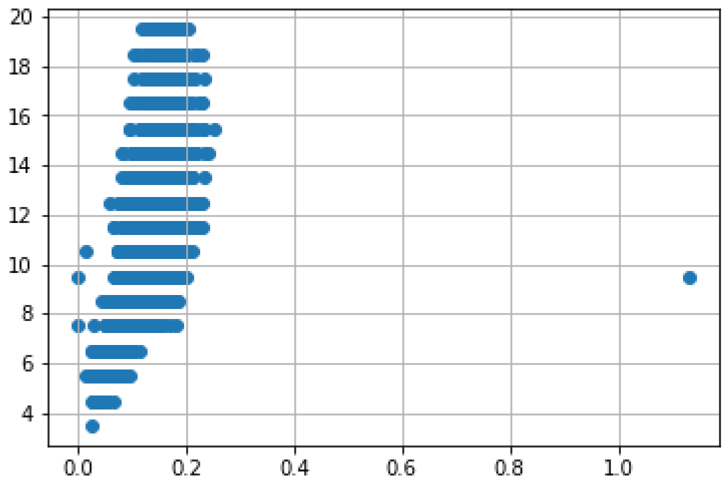

```
data.drop(data[(data['Shucked weight']>= 1) & (data['age'] < 20)].index, inplace=True)
data.drop(data[(data['Shucked weight']<1) & (data['age'] > 20)].index, inplace=True)

var = 'Whole weight'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```



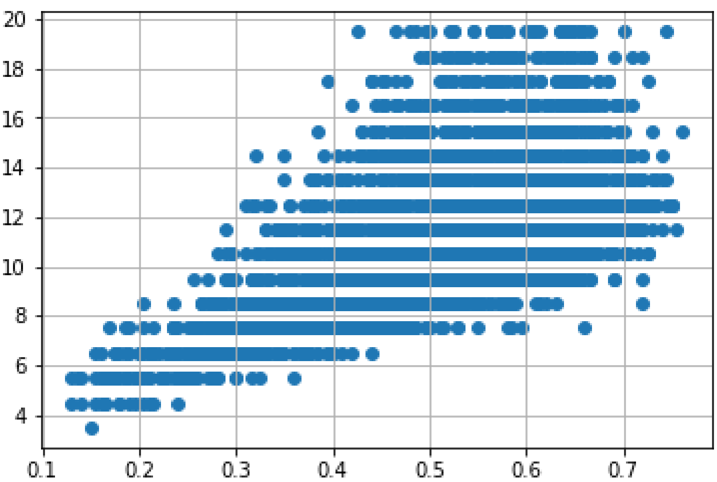

```
data.drop(data[(data['Whole weight']>= 2.5) & (data['age'] < 25)].index, inplace=True)
data.drop(data[(data['Whole weight']<2.5) & (data['age'] > 25)].index, inplace=True)

var = 'Diameter'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```




```
data.drop(data[(data['Diameter']<0.1) & (data['age'] < 5)].index, inplace=True)
data.drop(data[(data['Diameter']<0.6) & (data['age'] > 25)].index, inplace=True)
data.drop(data[(data['Diameter']>=0.6) & (data['age']< 25)].index, inplace=True)

var = 'Height'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```




```
data.drop(data[(data['Height']>0.4) & (data['age'] < 15)].index, inplace=True)
data.drop(data[(data['Height']<0.4) & (data['age'] > 25)].index, inplace=True)

var = 'Length'
plt.scatter(x = data[var], y = data['age'],)
plt.grid(True)
```




```
data.drop(data[(data['Length']<0.1) & (data['age'] < 5)].index, inplace=True)
```

```
data.drop(data[(data['Length']<0.8) & (data['age'] > 25)].index, inplace=True)
data.drop(data[(data['Length']>=0.8) & (data['age']< 25)].index, inplace=True)
```

7. Check for Categorical columns and perform encoding.

```
from sklearn import preprocessing
label = preprocessing.LabelEncoder()

original_data['Sex']= label.fit_transform(original_data['Sex'])
```

```
original_data.head()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	age
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	16.5
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	8.5
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	10.5
3	2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	11.5
4	1	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	8.5

8. Split the data into dependent and independent variables.

```
X = original_data.drop('age', axis = 1)
y = original_data['age']
```

```
X
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700

```
y
```

```
0       16.5
1        8.5
2       10.5
3       11.5
4        8.5
        ...
4172    12.5
4173    11.5
4174    10.5
4175    11.5
4176    13.5
Name: age, Length: 4177, dtype: float64
```

9. Scale the independent variables

```
# Normalized Y

from sklearn import preprocessing
Y=y.values.reshape(-1,1)

normalized_Y = preprocessing.normalize(Y)

print (normalized_Y)
```

```
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
```

```
# Standardized Y

standard_Y = Y.copy()

from sklearn import preprocessing

ss = preprocessing.StandardScaler()
ss.fit(standard_Y)

print (standard_Y)
```

```
[[16.5]
 [ 8.5]
 [10.5]
 ...
 [10.5]
 [11.5]
 [13.5]]
```

10. Split the data into training and testing

```
X = data.drop('age', axis = 1)
y = data['age']
```

```
standardScale = StandardScaler()
standardScale.fit_transform(X)

selectkBest = SelectKBest()
X_new = selectkBest.fit_transform(X, y)

X_train, X_test, y_train, y_test = train_test_split(X_new, y, test_size = 0.25)
```

11. Build the Model

12. Train the Model

```
np.random.seed(10)
def rmse_cv(model, X_train, y):
    rmse =- (cross_val_score(model, X_train, y, scoring='neg_mean_squared_error', cv=5))
    return(rmse*100)

models = [LinearRegression(),
             Ridge(),
             SVR(),
             RandomForestRegressor(),
             GradientBoostingRegressor(),
             KNeighborsRegressor(n_neighbors = 4),]

names = ['LR','Ridge','svm','GNB','RF','GB','KNN']

for model,name in zip(models,names):
    score = rmse_cv(model,X_train,y_train)
    print("{}    : {:.6f}, {:4f}".format(name,score.mean(),score.std()))
```

```
LR    : 362.218315, 31.358517
Ridge    : 367.492321, 28.808135
svm    : 389.796574, 30.207738
GNB    : 363.082453, 28.130258
```

```
RF    : 351.412403, 27.731600
GB    : 403.767053, 30.014933
```

```
def modelfit(alg, dtrain, predictors, performCV=True, printFeatureImportance=True, cv_fold
    #Fit the algorithm on the data
    alg.fit(dtrain[predictors], dtrain['age'])

    #Predict training set:
    dtrain_predictions = alg.predict(dtrain[predictors])
    #dtrain_predprob = alg.predict_proba(dtrain[predictors])[:,1]

    #Perform cross-validation:
    if performCV:
        cv_score = -cross_val_score(alg, dtrain[predictors], dtrain['age'], cv=cv_folds,
                                                      scoring='r2')

    #Print model report:
    print ("\nModel Report")
    print( "RMSE : %.4g" % mean_squared_error(dtrain['age'].values, dtrain_predictions))
    print( "R2 Score (Train): %f" % r2_score(dtrain['age'], dtrain_predictions))

    if performCV:
        print( "CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(
                                                                             np.min(cv

    #Print Feature Importance:
    if printFeatureImportance:
        feat_imp = pd.Series(alg.coef_, predictors).sort_values(ascending=False)
        plt.figure(figsize=(20,4))
        feat_imp.plot(kind='bar', title='Feature Importances')
        plt.ylabel('Feature Importance Score')
```

13. Test the Model

14. Measure the performance using Metrics

```
# Base Model
predictors = [x for x in data.columns if x not in ['age']]
lrm0 = Ridge(random_state=10)
modelfit(lrm0, data, predictors)
```

```
Model Report
RMSE : 3.593
R2 Score (Train): 0.529894
CV Score : Mean - -0.4503433 | Std - 0.08079434 | Min - -0.514565 | Max - -0.3061263
```