```
import pandas as pd
import numpy as np
import seaborn as sns
```

LOADING THE DATASET

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

```
df
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.1500	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
...	...	...	...	...	...	...	...	...	...
4172	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4173	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4174	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4175	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4176	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4177 rows × 9 columns

```
df.dtypes
```

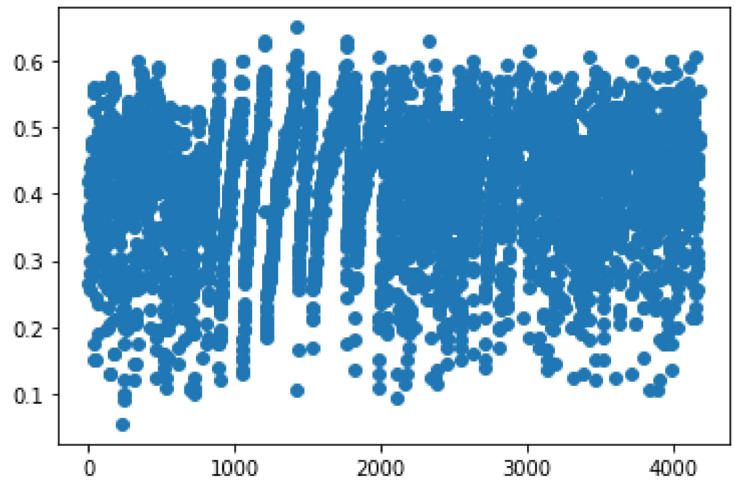
```
Sex                object
Length            float64
Diameter          float64
Height            float64
Whole weight      float64
Shucked weight    float64
Viscera weight    float64
Shell weight      float64
Rings               int64
dtype: object
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import seaborn as sns
```

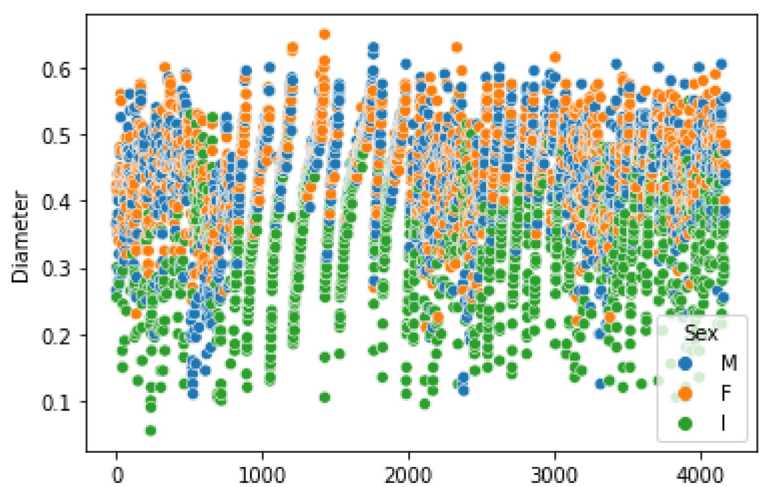
UNIVARIATE ANALYSIS

```
plt.scatter(df.index,df['Diameter'])
plt.show()
```



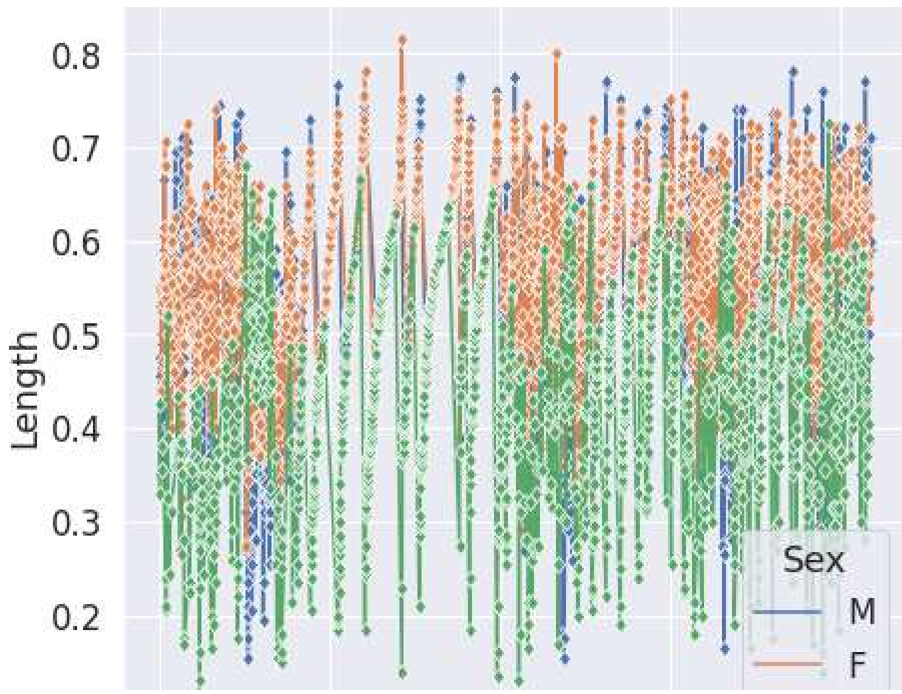

```
sns.scatterplot(x=df.index,y=df['Diameter'],hue=df['Sex'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f62aadf9cd0>



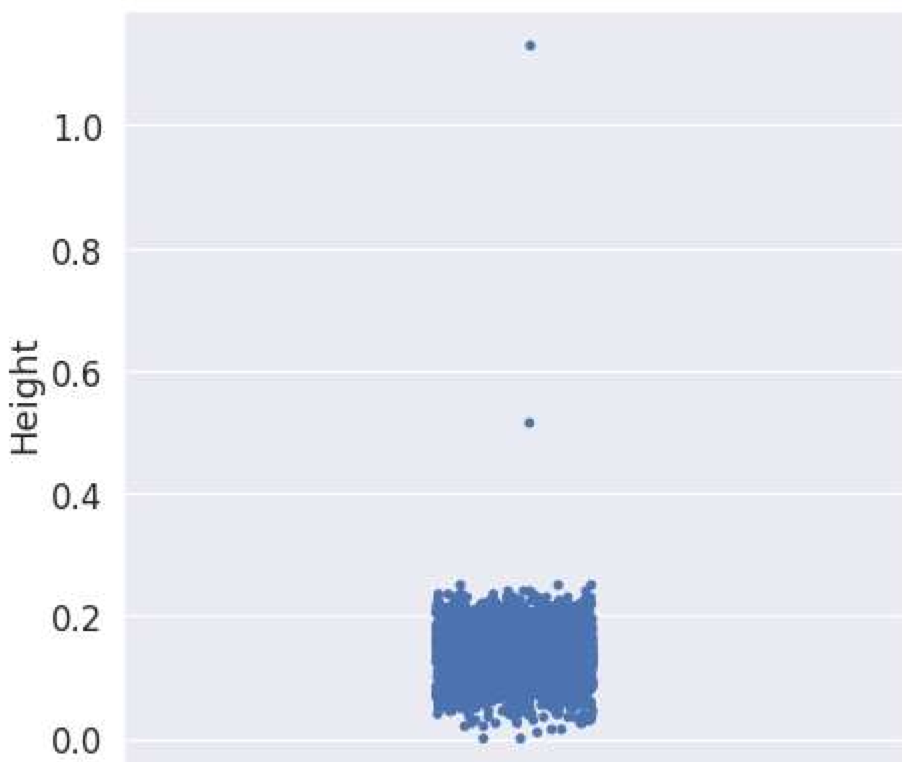

```
sns.set(rc={'figure.figsize': (7,7)})
sns.set (font_scale=1.5)
fig=sns.lineplot (x=df.index, y=df['Length'], markevery=1, marker='d', data=df, hue=df ['Sex'
fig.set(xlabel='index')
```

```
[Text(0.5, 0, 'index')]
```



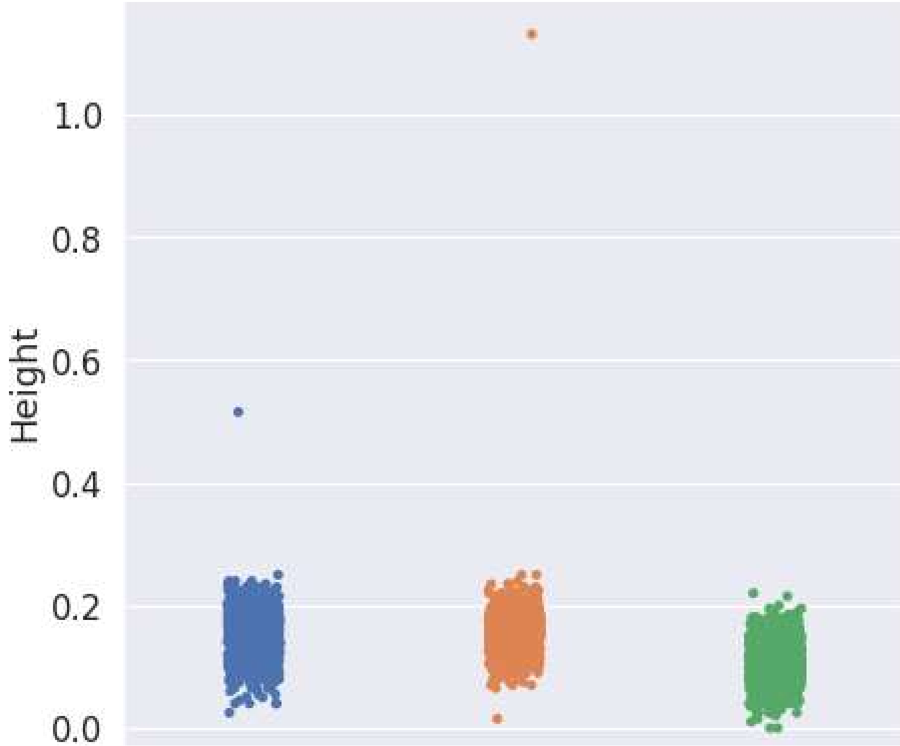

```
sns.stripplot (y=df['Height'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a9477b90>
```



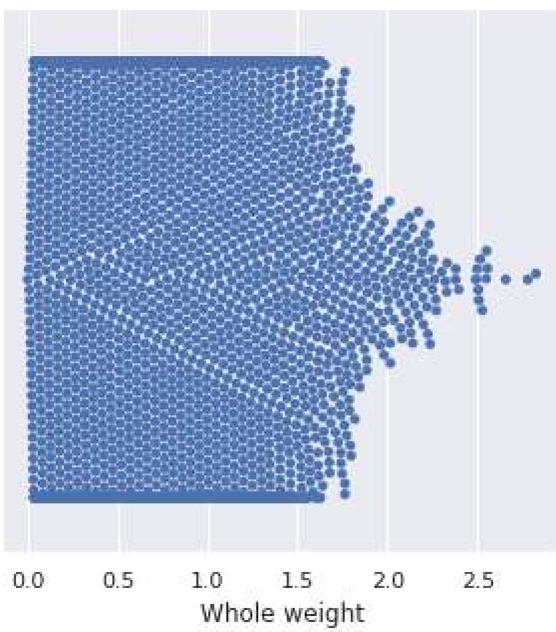

```
sns.stripplot (x=df['Sex'], y=df['Height'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a9445dd0>
```



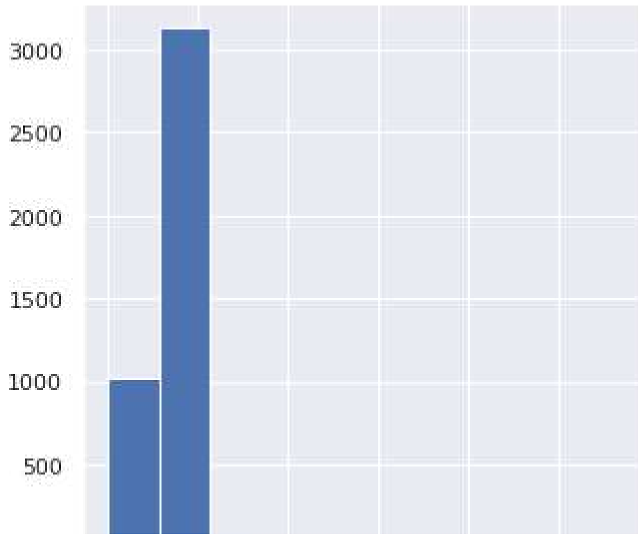

```
sns.set(rc={'figure.figsize': (5,5)})
sns.swarmplot (x=df['Whole weight'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:1296: UserWarning: 61.6% (
  warnings.warn(msg, UserWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a93ac150>
```



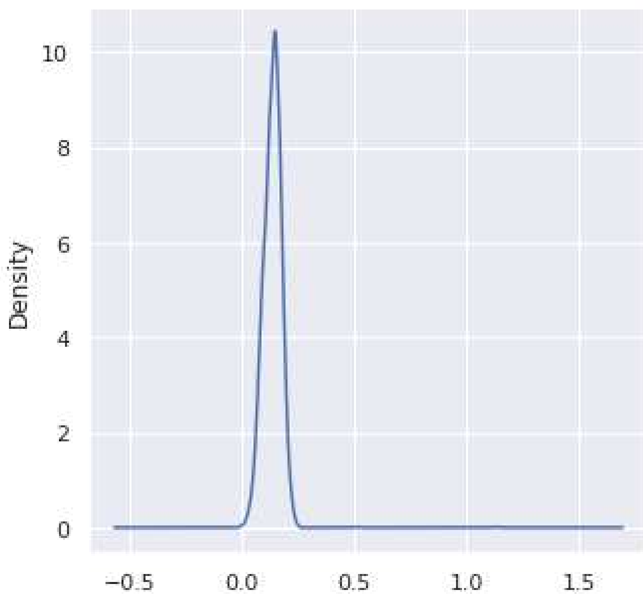

```
plt.hist(df['Height'])
```

```
(array([1.023e+03, 3.129e+03, 2.300e+01, 0.000e+00, 1.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00]),
 array([0.   , 0.113, 0.226, 0.339, 0.452, 0.565, 0.678, 0.791, 0.904,
        1.017, 1.13 ]),
 <a list of 10 Patch objects>)
```

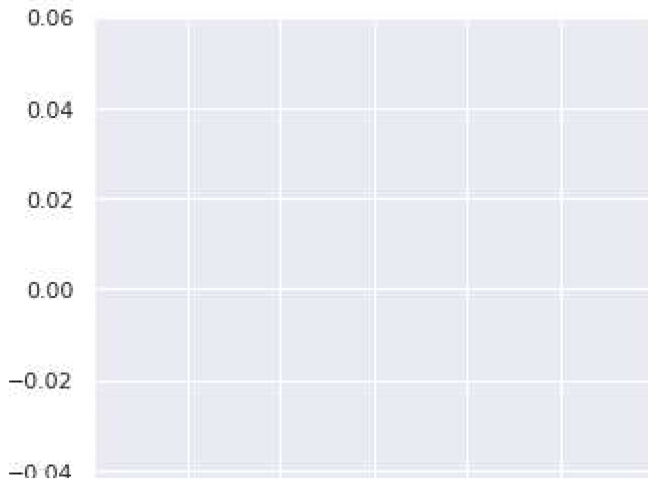



```
plt.figure (figsize=(5,5))
df ['Height'].plot (kind='density')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a6a78d10>
```

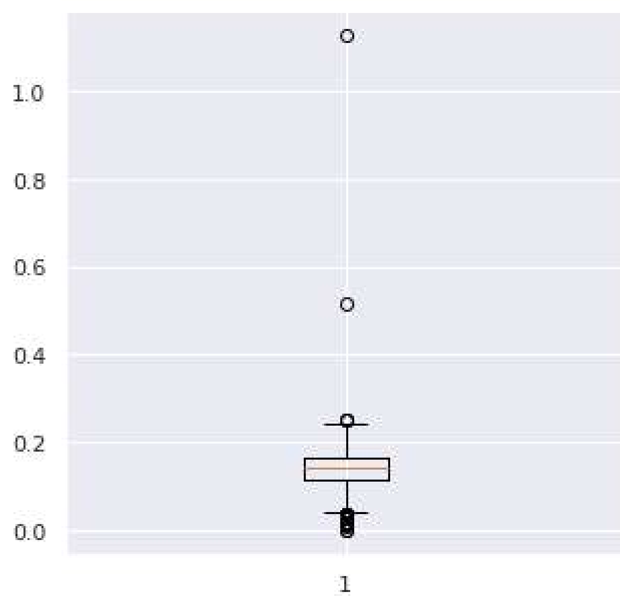



```
fig, ax = plt.subplots()
sns.rugplot (df ['Height'])
ax.set_xlim (3,9)
plt.show()
```




```
plt.boxplot(df['Height'])
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f62a6721190>,
  <matplotlib.lines.Line2D at 0x7f62a67216d0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f62a6721c10>,
  <matplotlib.lines.Line2D at 0x7f62a66a6190>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f62a671ab90>],
 'medians': [<matplotlib.lines.Line2D at 0x7f62a66a6710>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f62a66a6c50>],
 'means': []}
```

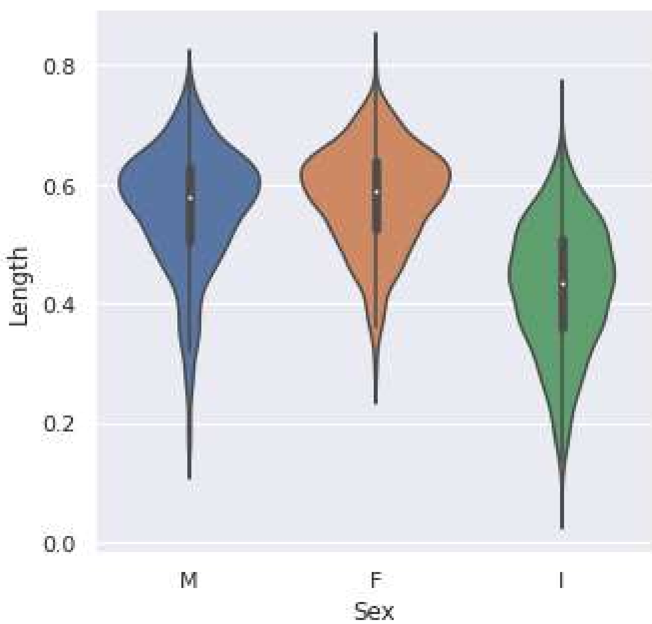


BIVARIATE ANALYSIS

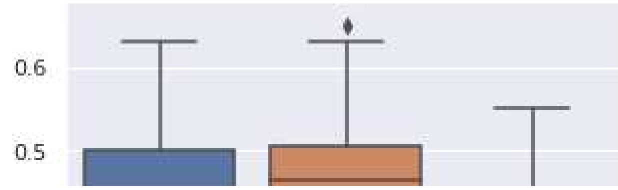
```
sns.barplot(x='Sex',y='Height',data=df)
sns.countplot(x='Sex',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a66cded0>
```

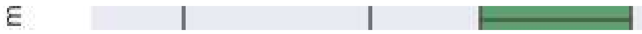

```
sns.violinplot (x="Sex", y="Length", data=df, size=8)
plt.show()
```




```
sns.boxplot (x='Sex',y='Diameter', data=df)
plt.show()
```

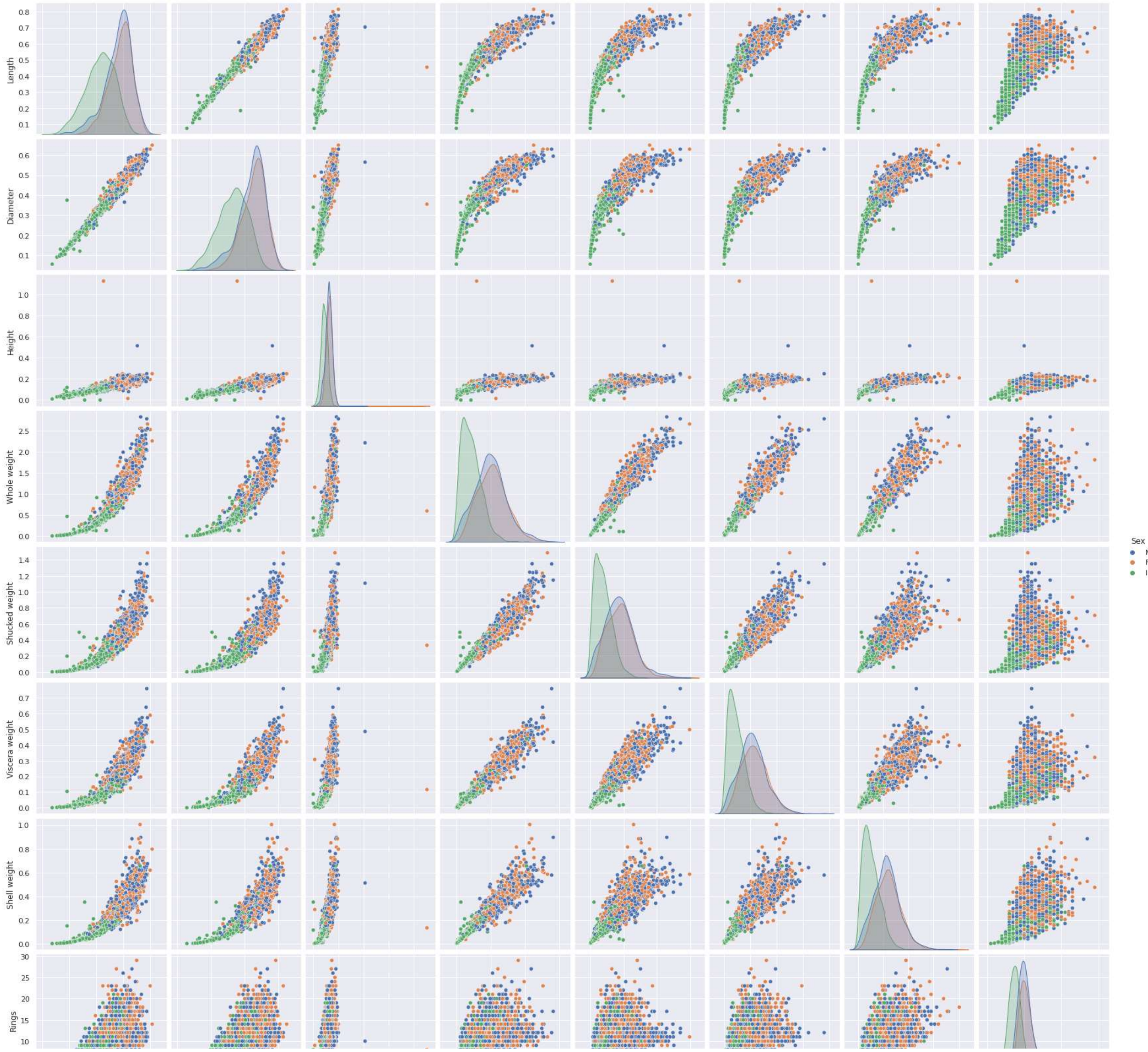


MULTIVARIATE ANALYSIS



```
sns.pairplot (df, hue="Sex", size=3)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:2076: UserWarning: The `size`
  warnings.warn(msg, UserWarning)
```



PERFORM DESCRIPTIVE STATISTICS ON THE DATASET

```
pd.set_option('display.width', 100)
pd.set_option('precision', 3)
description = df.describe()
print(description)
```

```
          Length  Diameter    Height  Whole weight  Shucked weight  Viscera weight  Shell
count   4177.000  4177.000  4177.000      4177.000        4177.000       4.177e+03      4
mean       0.524     0.408     0.140         0.829           0.359       1.806e-01
std        0.120     0.099     0.042         0.490           0.222       1.096e-01
min        0.075     0.055     0.000         0.002           0.001       5.000e-04
25%        0.450     0.350     0.115         0.442           0.186       9.350e-02
50%        0.545     0.425     0.140         0.799           0.336       1.710e-01
75%        0.615     0.480     0.165         1.153           0.502       2.530e-01
max        0.815     0.650     1.130         2.825           1.488       7.600e-01
```

```
         Rings
count  4177.000
mean      9.934
std       3.224
min       1.000
25%       8.000
50%       9.000
75%      11.000
max      29.000
```

Check for Missing values

```
df.isnull()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
4172	False	False	False	False	False	False	False
4173	False	False	False	False	False	False	False
4174	False	False	False	False	False	False	False
4175	False	False	False	False	False	False	False
4176	False	False	False	False	False	False	False

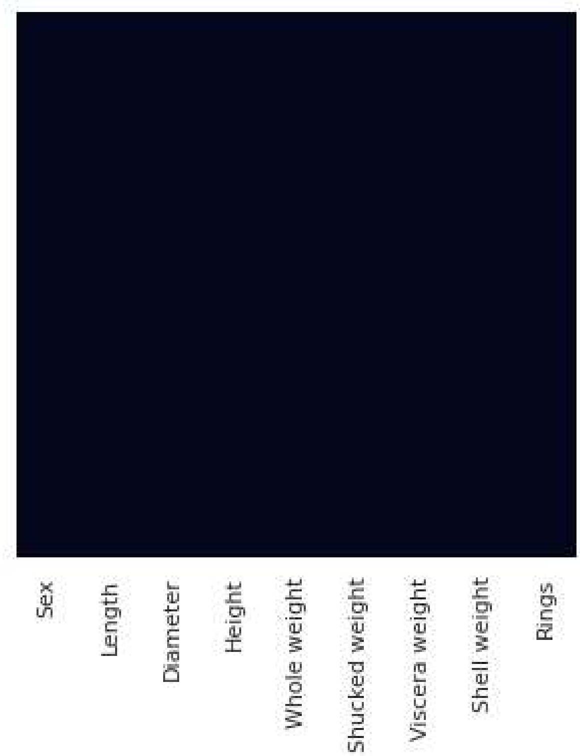
4177 rows × 9 columns

```
df.notnull()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	True	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	True	True	True
3	True	True	True	True	True	True	True	True	True
4	True	True	True	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

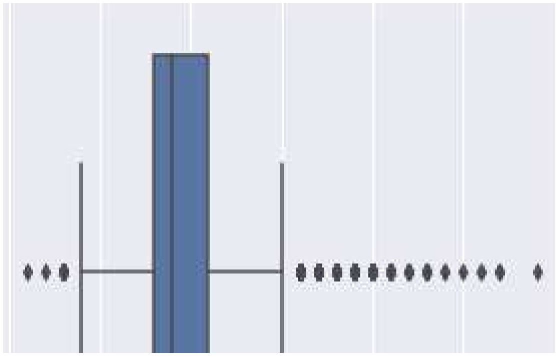
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a60e4490>
```



Find the outliers and replace them outliers

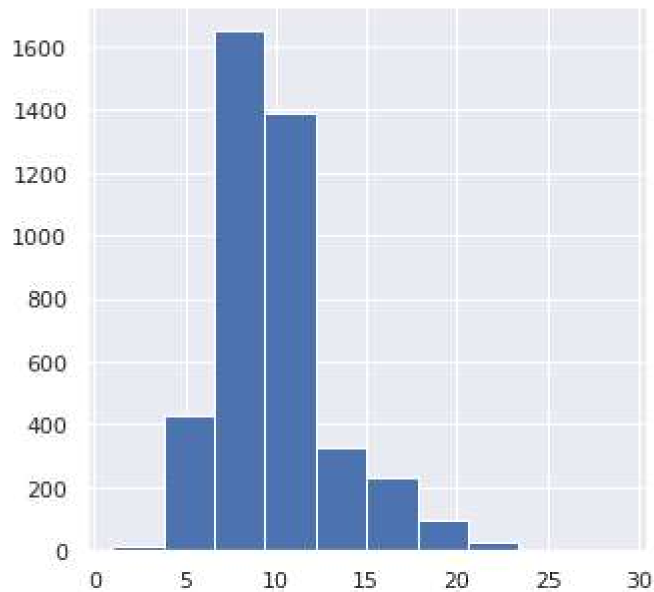
```
sns.boxplot(df['Rings'],data=df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a49d1850>
```



```
df['Rings'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a49352d0>
```



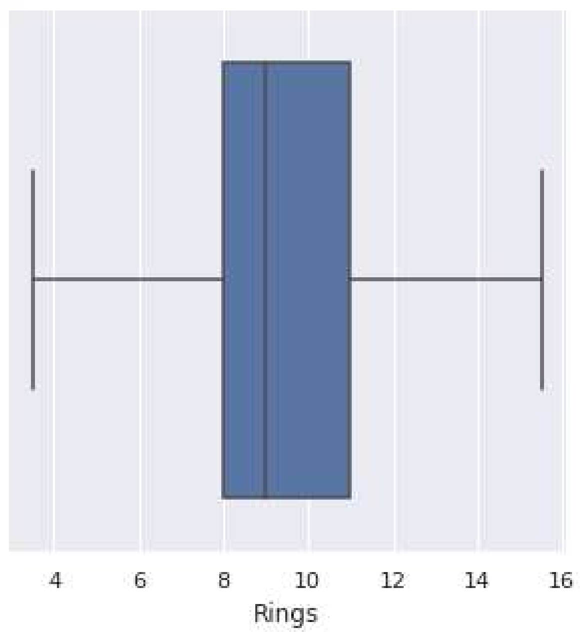

```
fare_mean = df['Rings'].mean()
fare_std = df['Rings'].std()
low= fare_mean -(3 * fare_std)
high= fare_mean + (3 * fare_std)
fare_outliers = df[(df['Rings'] < low) | (df['Rings'] > high)]
fare_outliers.head()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell
6	F	0.530	0.415	0.150	0.777	0.237	0.141	
72	F	0.595	0.475	0.170	1.247	0.480	0.225	
83	M	0.595	0.475	0.160	1.317	0.408	0.234	
166	F	0.725	0.575	0.175	2.124	0.765	0.452	
167	F	0.680	0.570	0.205	1.842	0.625	0.408	

```
Q1 = df['Rings'].quantile(0.25)
Q3 = df['Rings'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 -(whisker_width*IQR)
upper_whisker = Q3 +(whisker_width*IQR)
df['Rings']=np.where(df['Rings']>upper_whisker,upper_whisker,np.where(df['Rings']<lower_whisk
```

```
sns.boxplot(df['Rings'],data=df)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f62a48ce190>
```




```
Q1 = df['Rings'].quantile(0.10)
Q3 = df['Rings'].quantile(0.90)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
index=df['Rings'][(df['Rings']>upper_whisker)|(df['Rings']<lower_whisker)].index
df.drop(index,inplace=True)
```

Check for Categorical columns and perform encoding

```
from sklearn.compose import make_column_selector as selector

categorical_columns_selector = selector(dtype_include=object)
categorical_columns = categorical_columns_selector(df)
categorical_columns
```

```
['Sex']
```

```
data_categorical = df[categorical_columns]
data_categorical.head()
```

	Sex
0	M
1	M
2	F
3	M
4	I

```
from sklearn.preprocessing import OrdinalEncoder

Sex_column = data_categorical[["Sex"]]

encoder = OrdinalEncoder()
Sex_encoded = encoder.fit_transform(Sex_column)
Sex_encoded
```

```
array([[2.],
       [2.],
       [0.],
       ...,
       [2.],
       [0.],
       [2.]])
```

```
encoder.categories_
```

```
[array(['F', 'I', 'M'], dtype=object)]
```

```
data_encoded = encoder.fit_transform(data_categorical)
data_encoded[:5]
```

```
array([[2.],
       [2.],
       [0.],
       [2.],
       [1.]])
```

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)
```

```
Sex_encoded = encoder.fit_transform(Sex_column)
Sex_encoded
```

```
array([[0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       ...,
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.]])
```

```
feature_names = encoder.get_feature_names_out(input_features=["Sex"])
Sex_encoded = pd.DataFrame(Sex_encoded, columns=feature_names)
Sex_encoded
```

	Sex_F	Sex_I	Sex_M
0	0.0	0.0	1.0
1	0.0	0.0	1.0
2	1.0	0.0	0.0
3	0.0	0.0	1.0
4	0.0	1.0	0.0
...	...	...	...
4172	1.0	0.0	0.0
4173	0.0	0.0	1.0
4174	0.0	0.0	1.0
4175	1.0	0.0	0.0
4176	0.0	0.0	1.0

4177 rows × 3 columns

```
data_encoded = encoder.fit_transform(data_categorical)
data_encoded[:5]
```

```
array([[0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

Split the data into dependent and independent variables

```
X= df.iloc[ : , :-1].values
```

```
print(X)
```

```
[['M' 0.455 0.365 ... 0.2245 0.101 0.15]
 ['M' 0.35 0.265 ... 0.0995 0.0485 0.07]
 ['F' 0.53 0.42 ... 0.2565 0.1415 0.21]
 ...
 ['M' 0.6 0.475 ... 0.5255 0.2875 0.308]
 ['F' 0.625 0.485 ... 0.531 0.261 0.296]
 ['M' 0.71 0.555 ... 0.9455 0.3765 0.495]]
```

```
y= df.iloc[ : , 4].values
print(y)
```

```
[0.514  0.2255 0.677  ... 1.176  1.0945 1.9485]
```

Scale the independent variables

```
from sklearn import preprocessing
```

```
df.drop(labels="Sex",axis=1)
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
0	0.455	0.365	0.095	0.514	0.225	0.101	0.150	15.0
1	0.350	0.265	0.090	0.226	0.100	0.049	0.070	7.0
2	0.530	0.420	0.135	0.677	0.257	0.141	0.210	9.0
3	0.440	0.365	0.125	0.516	0.215	0.114	0.155	10.0
4	0.330	0.255	0.080	0.205	0.089	0.040	0.055	7.0
...	...	...	...	...	...	...	...	...
4172	0.565	0.450	0.165	0.887	0.370	0.239	0.249	11.0
4173	0.590	0.440	0.135	0.966	0.439	0.214	0.261	10.0
4174	0.600	0.475	0.205	1.176	0.525	0.287	0.308	9.0
4175	0.625	0.485	0.150	1.095	0.531	0.261	0.296	10.0
4176	0.710	0.555	0.195	1.948	0.946	0.377	0.495	12.0

4177 rows × 8 columns

Split the data into training and testing

```
from sklearn.model_selection import train_test_split
```

```
X=df.iloc[ : , :-1]
y=df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0)
```

```
X_train
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
678	F	0.450	0.380	0.165	0.817	0.250	0.192	0.265
3009	I	0.255	0.185	0.065	0.074	0.030	0.017	0.020
1906	I	0.575	0.450	0.135	0.825	0.338	0.211	0.239
768	F	0.550	0.430	0.155	0.785	0.289	0.227	0.233
2781	M	0.595	0.475	0.140	1.030	0.492	0.217	0.278
...	...	...	...	...	...	...	...	...
1033	M	0.650	0.525	0.185	1.622	0.664	0.323	0.477
3264	F	0.655	0.500	0.140	1.171	0.540	0.318	0.285
1653	M	0.595	0.450	0.145	0.959	0.463	0.206	0.254
2607	F	0.625	0.490	0.165	1.127	0.477	0.236	0.319
2732	I	0.410	0.325	0.110	0.326	0.133	0.075	0.101

3968 rows × 8 columns

```
y_train
```

```
678     15.5
3009     4.0
1906    11.0
768     11.0
2781    10.0
        ...
1033    10.0
3264    12.0
1653    10.0
2607     9.0
2732     8.0
Name: Rings, Length: 3968, dtype: float64
```

Build the Model -> Train and test the model

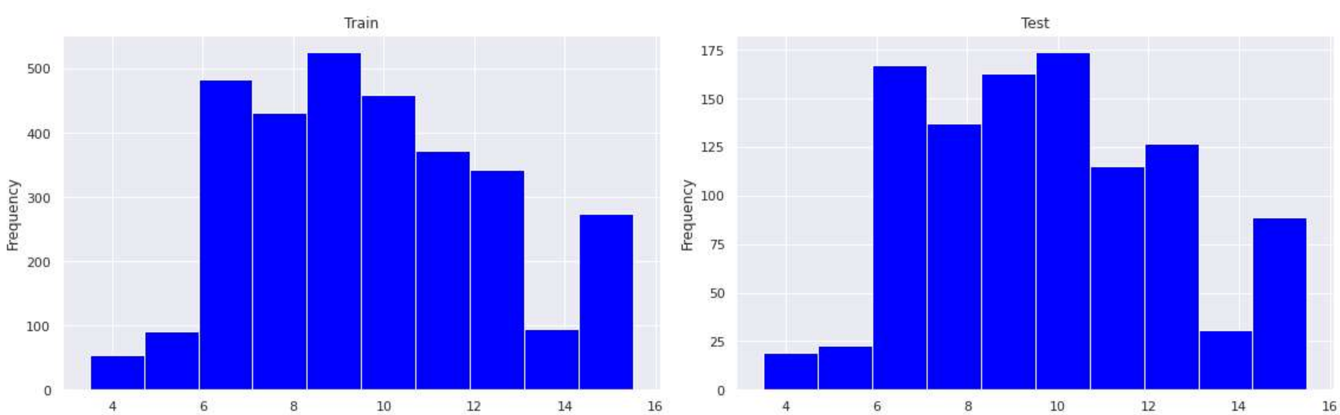
```
train, test = train_test_split(df, test_size=0.25, random_state=1)
print('Train data points :', len(train))
print('Test data points :', len(test))
```

```
Train data points : 3132
Test data points : 1045
```

```
train.Sex = train.Sex.replace({"M":1, "I":0, "F":-1})
test.Sex = test.Sex.replace({"M":1, "I":0, "F":-1})
```

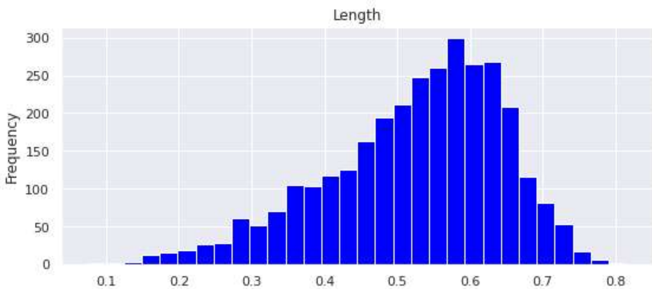
```
numerical_features = ["Length", 'Diameter', 'Height','Whole weight',
                      'Shucked weight', 'Viscera weight', 'Shell weight']

categorical_feature = "Sex"

features = numerical_features + [categorical_feature]

target = 'Rings'
```

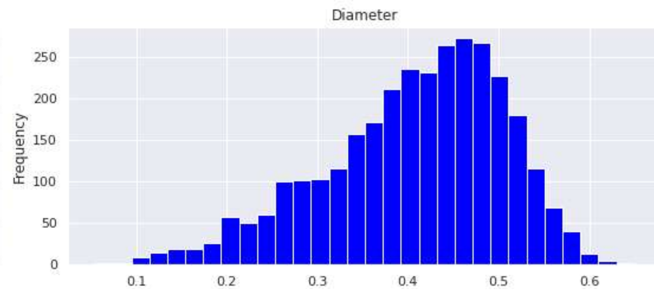
```
fig, axes = plt.subplots(ncols=2,figsize=(16, 5))

train[target].plot.hist(color='blue', ax=axes[0])
axes[0].set(title="Train")

test[target].plot.hist(color='blue', ax=axes[1])
axes[1].set(title="Test")

plt.tight_layout()
plt.show()
```



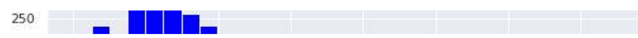

```
fig, axes = plt.subplots(4,2,figsize=(16, 14))
axes = np.ravel(axes)
```

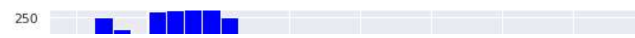
```
for i, c in enumerate(numerical_features):
    hist = train[c].plot(kind = 'hist', ax=axes[i], title=c, color='blue', bins=30)

plt.tight_layout()
plt.show()
```






```
idx = train.loc[train.Height>0.4].index
train.drop(idx, inplace=True)

idx = train.loc[train['Viscera weight']>0.6].index
train.drop(idx, inplace=True)

idx = train.loc[train[target]>25].index
train.drop(idx, inplace=True)
```






```
X_train = train[features]
y_train = train[target]

X_test = test[features]
y_test = test[target]

X_train.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Sex
4014	0.625	0.480	0.175	1.065	0.486	0.259	0.285	1
3252	0.480	0.380	0.130	0.618	0.300	0.142	0.175	1
305	0.200	0.145	0.060	0.037	0.013	0.009	0.011	0
1857	0.505	0.400	0.145	0.705	0.334	0.142	0.207	0
439	0.500	0.415	0.165	0.689	0.249	0.138	0.250	1

```
from sklearn.linear_model import LinearRegression
from sklearn. linear_model import Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor


models = {'linear_regression':LinearRegression(),

         'lasso':Lasso(random_state=1),

         'decision_tree':DecisionTreeRegressor(random_state=1),

         'random_forest':RandomForestRegressor(random_state=1),
```

```
          'xgboost':XGBRegressor(random_state=1),
        }
```

Measure the performance using Metrics.

```
# Linear regression
lr_params = {'fit_intercept':[True,False]}

# Lasso
lasso_params = {'alpha': [1e-4, 1e-3, 1e-2, 1, 10, 100]}

# Decision tree
dt_params =  {'max_depth': [4, 6, 8, 10, 12, 14, 16, 20],
            'min_samples_split': [5, 10, 20, 30, 40, 50],
            'max_features': [0.2, 0.4, 0.6, 0.8, 1],
            'max_leaf_nodes': [8, 16, 32, 64, 128,256]}

# Random Forest
rf_params = {'bootstrap': [True, False],
             'max_depth': [2, 5, 10, 20, None],
             'max_features': ['auto', 'sqrt'],
             'min_samples_leaf': [1, 2, 4],
             'min_samples_split': [2, 5, 10],
             'n_estimators': [100, 150, 200, 250]}

# XGBoost
xgb_params = {'n_estimators':[100, 200, 300] ,
             'max_depth':list(range(1,10)) ,
             'learning_rate':[0.006,0.007,0.008,0.05,0.09] ,
             'min_child_weight':list(range(1,10))}
```

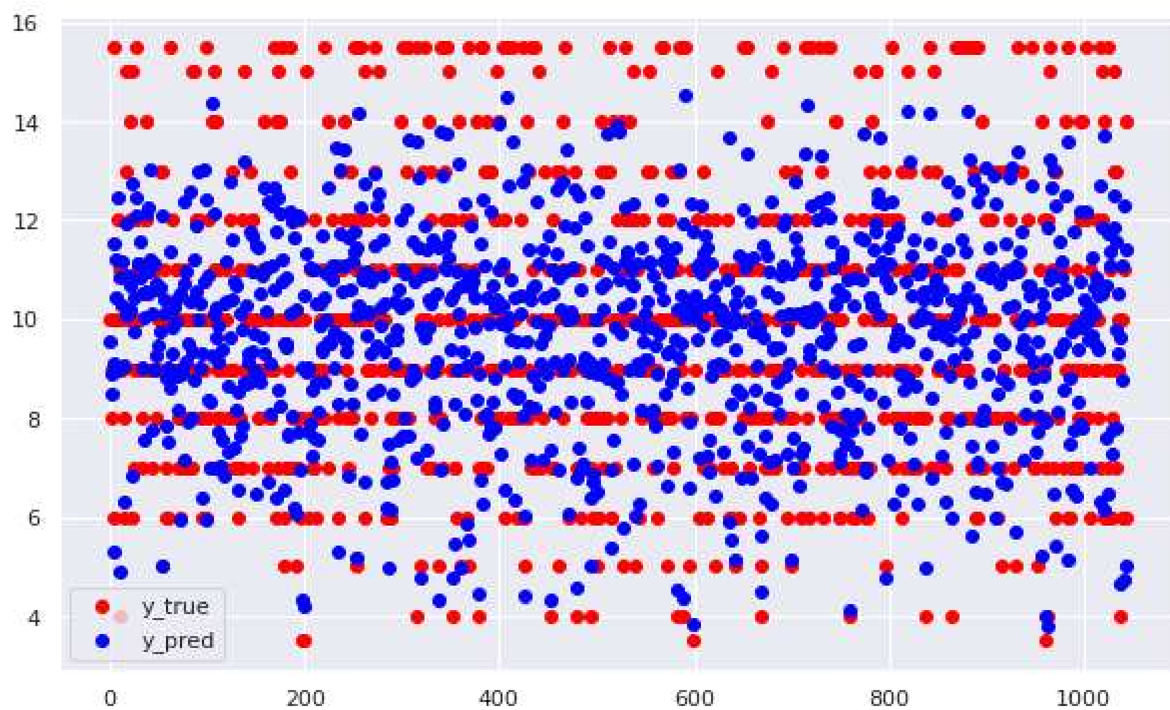
```
from sklearn.model_selection import RandomizedSearchCV
params = [lr_params, lasso_params, dt_params, rf_params, xgb_params]

# searching Hyperparameters
i=0
for name, model in models.items():
    print(name)
    regressor = RandomizedSearchCV(estimator = model,n_iter=10,param_distributions = params[i
    search = regressor.fit(X_train, y_train)
    print("Best params :",search.best_params_)
    print("RMSE :", -search.best_score_)
    i+=1
    print()
```

```
rf_params = {'n_estimators': 200,
```

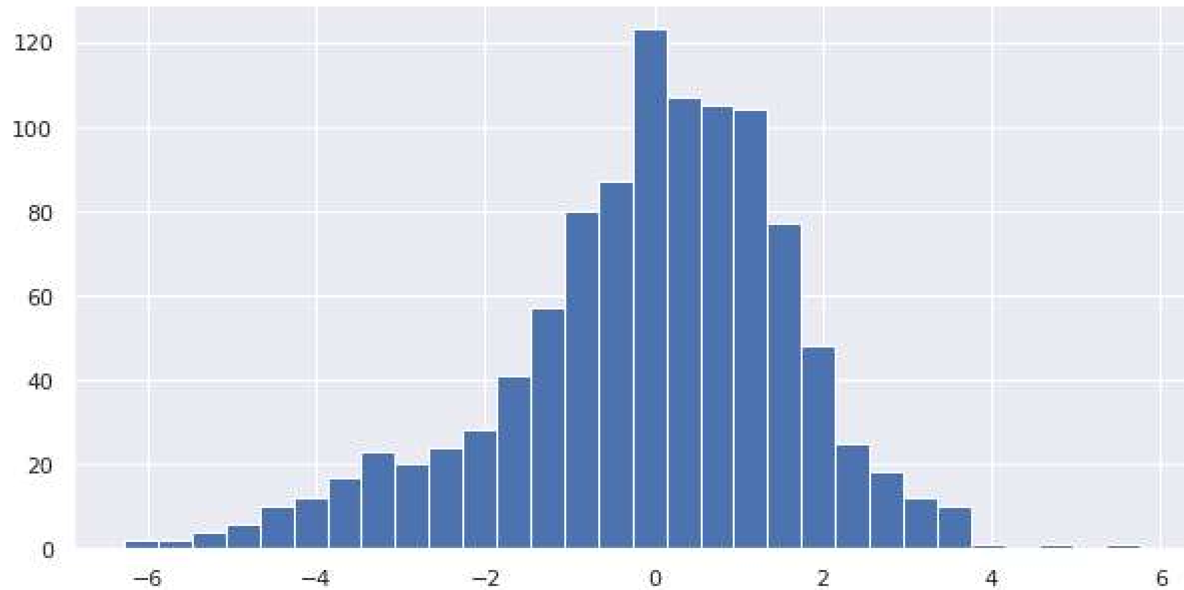
```
             'min_samples_split': 2,
             'min_samples_leaf': 4,
             'max_features': 'sqrt',
             'max_depth': None,
             'bootstrap': True}

model = RandomForestRegressor(random_state=1, **rf_params)

model.fit(X_train, y_train)
```

```
RandomForestRegressor(max_features='sqrt', min_samples_leaf=4, n_estimators=200,
                      random_state=1)
```

```
import pickle
with open("model.pkl", "wb") as f:
    pickle.dump(model, f)


y_pred = model.predict(X_test)

fig = plt.figure(figsize=(10, 6))
plt.scatter(range(y_test.shape[0]), y_test, color='red', label='y_true')
plt.scatter(range(y_test.shape[0]), y_pred, color='blue', label='y_pred')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(10,5))
plt.hist(y_pred-y_test, bins=30)
plt.show()
```




```
def predict_age(x):
    x = pd.DataFrame([x], columns=features)
    age = model.predict(x)
    return round(age[0],2)


with open("model.pkl", 'rb') as f:
    model = pickle.load(f)
ex = [0.295 , 0.225 , 0.08  , 0.124 , 0.0485, 0.032 , 0.04  , 0.]
print("Estimated age : ",predict_age(ex))
```

```
Estimated age :  7.26
```