```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from google.colab import files
upload=files.upload()
df = pd.read_csv('abalone.csv')
```

Choose Files abalone.csv
- **abalone.csv**(text/csv) - 191962 bytes, last modified: 11/4/2022 - 100% done

```
Saving abalone.csv to abalone (1).csv
```

```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4
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	

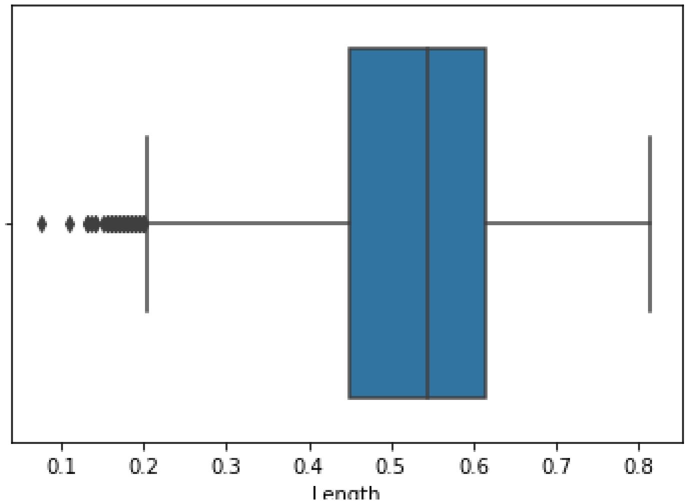
```
df.head()
```

	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	0.150	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

Univariate analysis

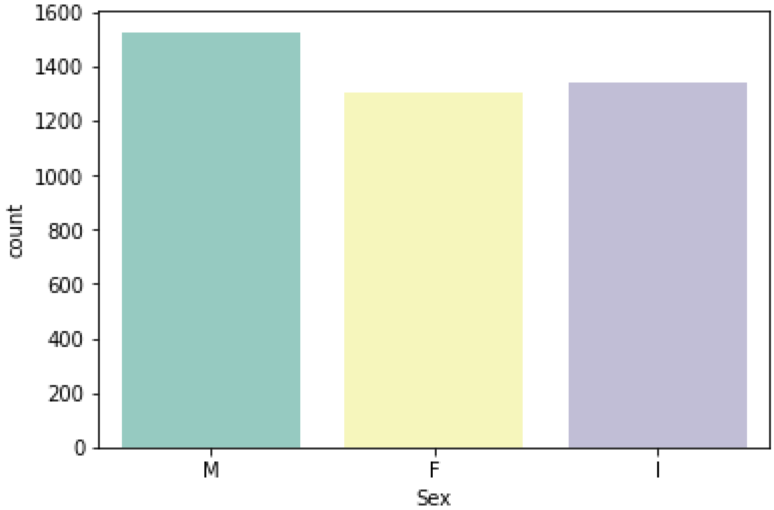
```
sns.boxplot(df.Length)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pas
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cd0d98e90>
```




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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cd1610050>
```



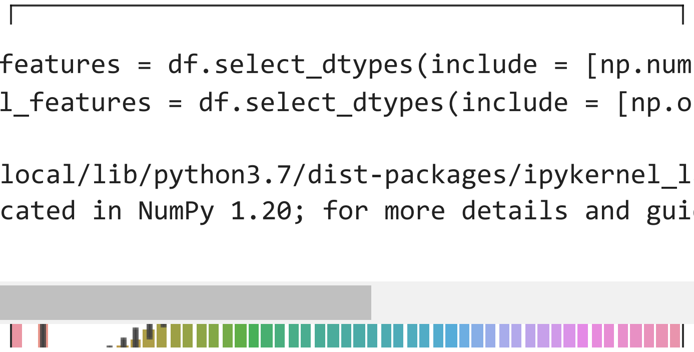
Bivariate analysis

```
sns.barplot(x=df.Height,y=df.Diameter)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cd0c6f2d0>
```

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

```
/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/r
```



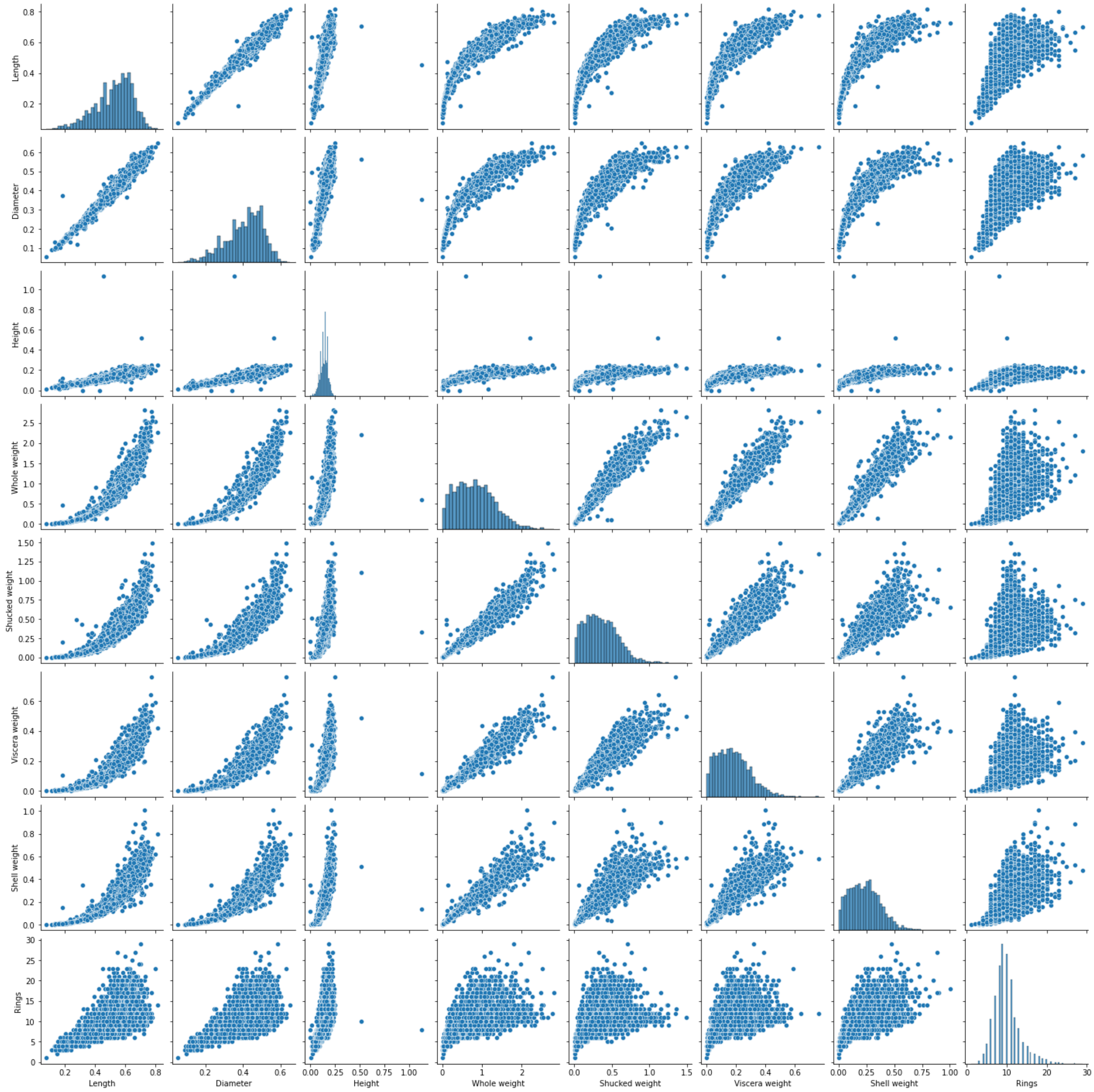
```
plt.figure(figsize = (20,7))
sns.heatmap(df[numerical_features].corr(),annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cd0577750>
```

Multivariate Analysis

```
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7f8ccc3f0610>
```



Perform descriptive model on the dataset

```
df['Height'].describe()
```

```
count    4177.000000
mean        0.139516
std         0.041827
min         0.000000
25%         0.115000
50%         0.140000
75%         0.165000
max         1.130000
Name: Height, dtype: float64
```

```
df['Height'].mean()
```

```
0.13951639932966242
```

```
df.max()
```

```
Sex                    M
Length             0.815
Diameter            0.65
Height              1.13
Whole weight      2.8255
Shucked weight     1.488
Viscera weight      0.76
Shell weight       1.005
Rings                 29
dtype: object
```

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

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

```
df[df.Height == 0]
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
1257	I	0.430	0.34	0.0	0.428	0.2065	0.0860	0.1150	8
3996	I	0.315	0.23	0.0	0.134	0.0575	0.0285	0.3505	6

```
df['Shucked weight'].kurtosis()
```

```
0.5951236783694207
```

```
df['Diameter'].median()
```

```
0.425
```

```
df['Shucked weight'].skew()
```

```
0.7190979217612694
```

Missing values

```
df.isna().any()
```

```
Sex               False
Length            False
Diameter          False
Height            False
Whole weight      False
Shucked weight    False
Viscera weight    False
Shell weight      False
Rings             False
dtype: bool
```

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

1 to 9 of 9 entries Filter

index	Missing values	% Missing
Sex	0	0.0
Length	0	0.0
Diameter	0	0.0
Height	0	0.0
Whole weight	0	0.0
Shucked weight	0	0.0
Viscera weight	0	0.0
Shell weight	0	0.0
Rings	0	0.0

Show 25 per page
Like what you see? Visit the [data table notebook]() to learn more about interactive tables.

Find the outliers

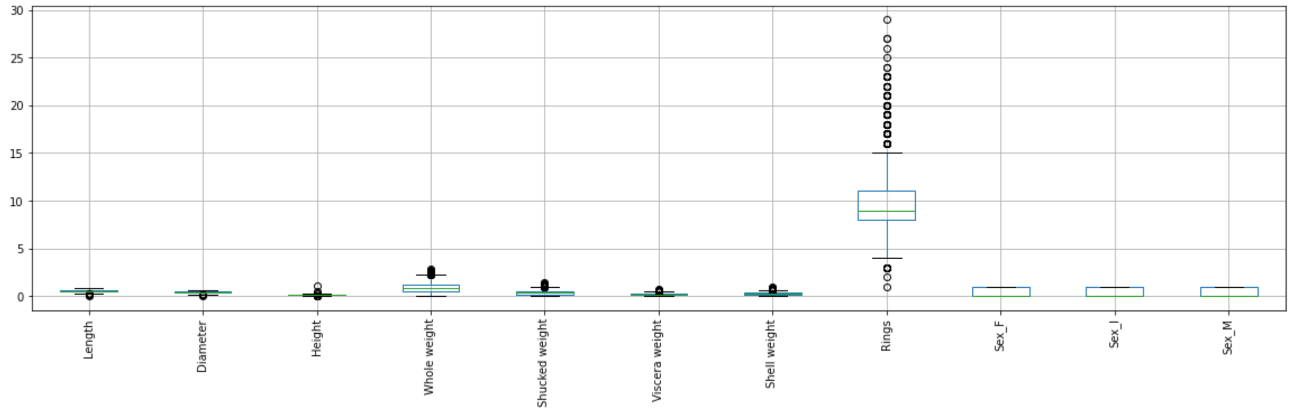
```
q1=df.Rings.quantile(0.25)
q2=df.Rings.quantile(0.75)
iqr=q2-q1
```

```
print(iqr)
```

```
3.0
```

```
df = pd.get_dummies(df)
dummy_df = df
df.boxplot( rot = 90, figsize=(20,5))
```

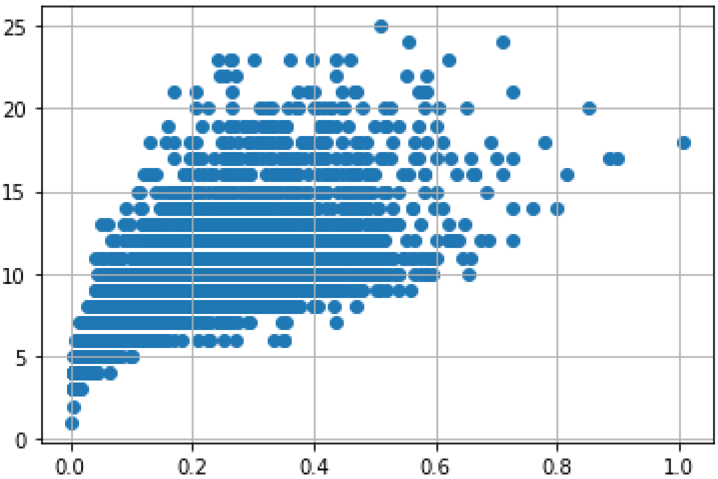
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8cc84cf450>
```




```
df['age'] = df['Rings']
df = df.drop('Rings', axis = 1)
```

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

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



Check for categorical columns and perform encoding

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

```
/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/r
```

```
numerical_features
categorical_features
```

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

```
abalone_numeric = df[['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight','Vi
```

```
abalone_numeric.head()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	age	Sex_F	Sex_I	Se
0	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	0	0	
1	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	0	0	
2	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	1	0	
3	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	0	0	

Dependent and Independent Variables

```
x = df.iloc[:, 0:1].values
```

```
y = df.iloc[:, 1]
```

```
y
```

```
0       0.365
1       0.265
2       0.420
3       0.365
4       0.255
        ...
4172    0.450
4173    0.440
4174    0.475
4175    0.485
4176    0.555
Name: Diameter, Length: 4150, dtype: float64
```

Scaling the Independent Variables

```
print ("\n ORIGINAL VALUES: \n\n", x,y)
```

```
 ORIGINAL VALUES: 

 [[0.455]
 [0.35 ]
 [0.53 ]
 ...
 [0.6  ]
 [0.625]
 [0.71 ]] 0       0.365
1       0.265
2       0.420
3       0.365
4       0.255
        ...  
4172    0.450
4173    0.440
4174    0.475
4175    0.485
4176    0.555
Name: Diameter, Length: 4150, dtype: float64
```

```
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler(feature_range =(0, 1))
new_y= min_max_scaler.fit_transform(x,y)
print ("\n VALUES AFTER MIN MAX SCALING: \n\n", new_y)
```

```
 VALUES AFTER MIN MAX SCALING: 

 [[0.51351351]
 [0.37162162]
 [0.61486486]
 ...
 [0.70945946]
 [0.74324324]
 [0.85810811]]
```

Split the data into Training and Testing

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

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.feature_selection import SelectKBest
```

```
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)
X_train
```

```
array([[0.685, 0.535, 0.155, ..., 0.   , 0.   , 1.   ],
       [0.36 , 0.265, 0.075, ..., 0.   , 1.   , 0.   ],
       [0.625, 0.495, 0.155, ..., 0.   , 0.   , 1.   ],
       ...,
       [0.665, 0.525, 0.155, ..., 0.   , 0.   , 1.   ],
       [0.445, 0.355, 0.095, ..., 0.   , 1.   , 0.   ],
       [0.43 , 0.35 , 0.11 , ..., 0.   , 0.   , 1.   ]])
```

```
y_train
```

```
1974    10
3529     6
3785     9
3132    10
543      8
        ..
3725     9
1468     8
3073    10
3540     8
11      10
Name: age, Length: 3112, dtype: int64
```

Build the model

Linear Regression

```
from sklearn import linear_model as lm
from sklearn.linear_model import LinearRegression
model=lm.LinearRegression()
results=model.fit(X_train,y_train)
```

```
accuracy = model.score(X_train, y_train)
print('Accuracy of the model:', accuracy)
```

```
Accuracy of the model: 0.5137651936009336
```

Training the model

```
lm = LinearRegression()
lm.fit(X_train, y_train)
y_train_pred = lm.predict(X_train)
y_train_pred
```

```
array([11.96875,  6.875  , 11.5    , ..., 13.4375 ,  8.4375 ,  9.59375])
```

```
X_train
```

```
array([[0.685, 0.535, 0.155, ..., 0.   , 0.   , 1.   ],
       [0.36 , 0.265, 0.075, ..., 0.   , 1.   , 0.   ],
       [0.625, 0.495, 0.155, ..., 0.   , 0.   , 1.   ],
       ...,
       [0.665, 0.525, 0.155, ..., 0.   , 0.   , 1.   ],
       [0.445, 0.355, 0.095, ..., 0.   , 1.   , 0.   ],
       [0.43 , 0.35 , 0.11 , ..., 0.   , 0.   , 1.   ]])
```

```
y_train
```

```
1974    10
3529     6
3785     9
3132    10
543      8
        ..
3725     9
1468     8
3073    10
3540     8
11      10
Name: age, Length: 3112, dtype: int64
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
s = mean_squared_error(y_train, y_train_pred)
print('Mean Squared error of training set :%2f'%s)
```

```
Mean Squared error of training set :4.838520
```

Testing the model

```
y_train_pred = lm.predict(X_train)
y_test_pred = lm.predict(X_test)
```

```
y_test_pred
```

```
array([10.53125,  6.03125,  9.5625 , ...,  9.59375,  6.8125 , 10.     ])
```

```
X_test
```

```
array([[0.61 , 0.45 , 0.15 , ..., 0.   , 0.   , 1.   ],
       [0.265, 0.205, 0.07 , ..., 0.   , 1.   , 0.   ],
       [0.57 , 0.445, 0.145, ..., 1.   , 0.   , 0.   ],
```

```
        ...,
        [0.54 , 0.425, 0.13 , ..., 0.   , 1.   , 0.   ],
        [0.35 , 0.25 , 0.07 , ..., 0.   , 1.   , 0.   ],
        [0.58 , 0.445, 0.135, ..., 0.   , 0.   , 1.   ]])
```

```
y_test
```

```
1363    10
323      5
1134     8
2231     9
2125     8
        ..
2078     8
3332    13
1606    10
818      6
1146     9
Name: age, Length: 1038, dtype: int64
```

```
p = mean_squared_error(y_test, y_test_pred)
print('Mean Squared error of testing set :%2f'%p)
```

```
Mean Squared error of testing set :4.362555
```

Measure the performance using metrices

```
from sklearn.metrics import r2_score
s = r2_score(y_train, y_train_pred)
print('R2 Score of training set:%.2f'%s)
```

```
R2 Score of training set:0.51
```

```
from sklearn.metrics import r2_score
p = r2_score(y_test, y_test_pred)
print('R2 Score of testing set:%.2f'%p)
```

```
R2 Score of testing set:0.59
```

✓ 0s completed at 9:51 PM