ASSIGNMENT – 3

Assignment Date	05 October 2022
Student Name	Abinaya.M
Student Roll Number	820319104002
Maximum Marks	2 Marks

Building a Regression Model

1. Download the dataset: [Dataset]

data=pd.read_csv("abalone.csv")

2. Load the dataset into the tool.

data.head()

```
In [2]: data.head()
```

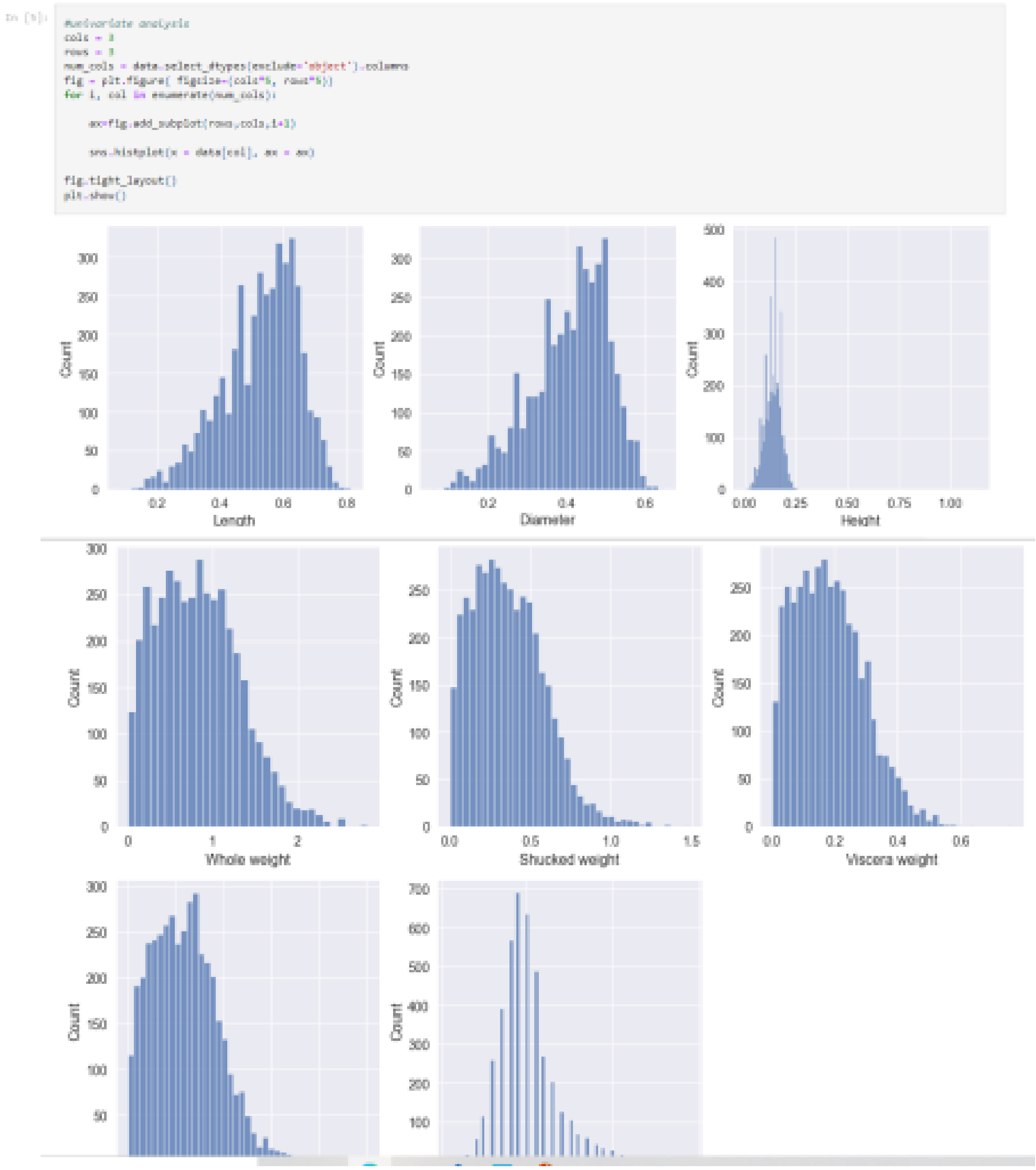
Out[2]:

	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

3. Perform Below Visualizations.

· Univariate Analysis

```
#univariate analysis
cols = 3
rows = 3
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure( figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):
ax=fig.add_subplot(rows,cols,i+1)
sns.histplot(x = data[col], ax = ax)
fig.tight_layout()
plt.show()
```



.

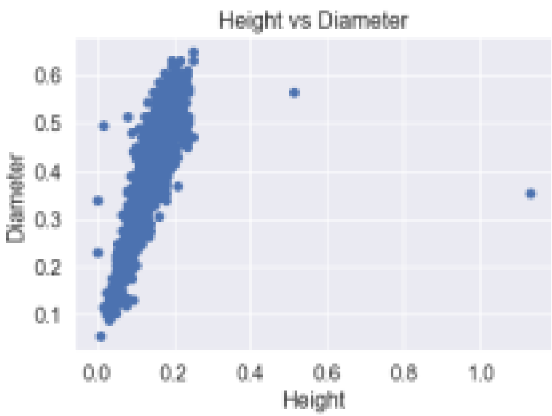
Bi-Variate Analysis

```python
#Bivariate analysis
import matplotlib.pyplot as plt
#create scatterplot of hours vs. score
plt.scatter(data.Height, data.Diameter)
plt.title('Height vs Diameter')
plt.xlabel('Height')
plt.ylabel
```

In [7]:

```
#Bivariate analysis
import matplotlib.pyplot as plt

#create scatterplot of hours vs. score
plt.scatter(data.Height, data.Diameter)
plt.title('Height vs Diameter')
plt.xlabel('Height')
plt.ylabel('Diameter')
```

Out[7]: Text(0, 0.5, 'Diameter')



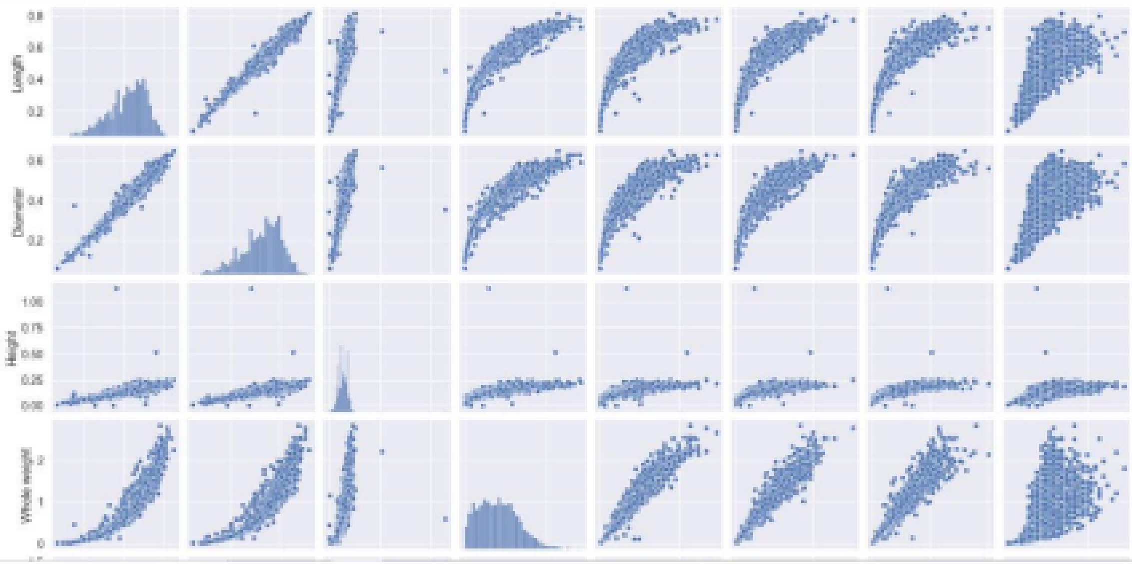
Multi-Variate Analysis

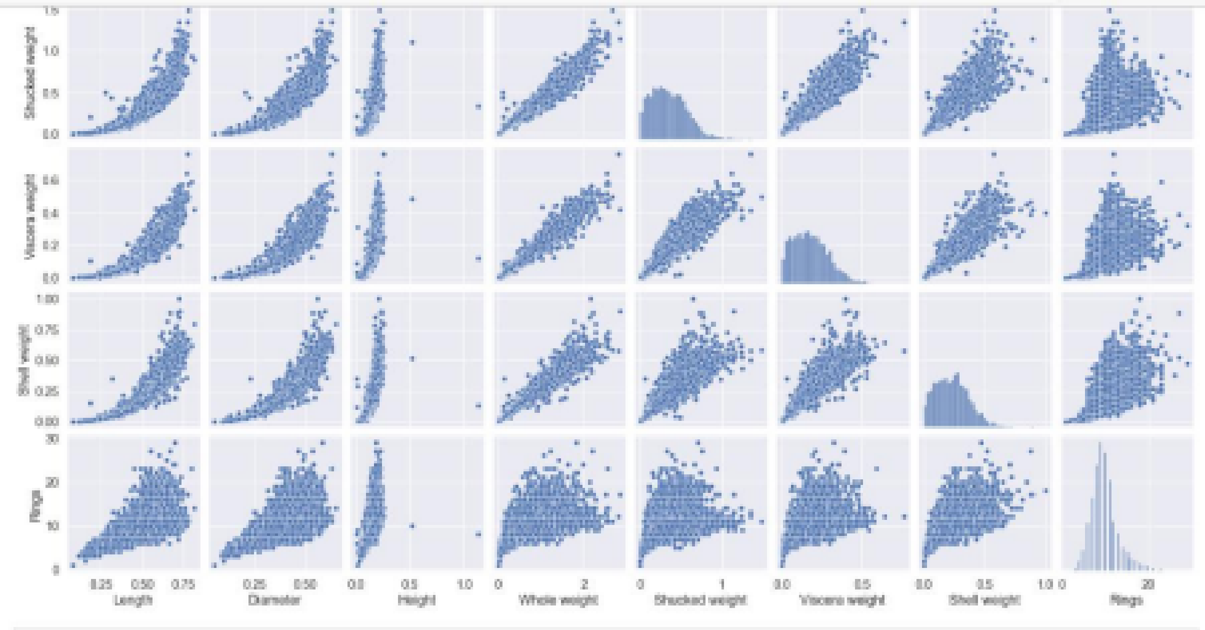
#multivariate analysis

sns.pairplot(d

In [8]:

```
#multivariate analysis
sns.pairplot(data);
```





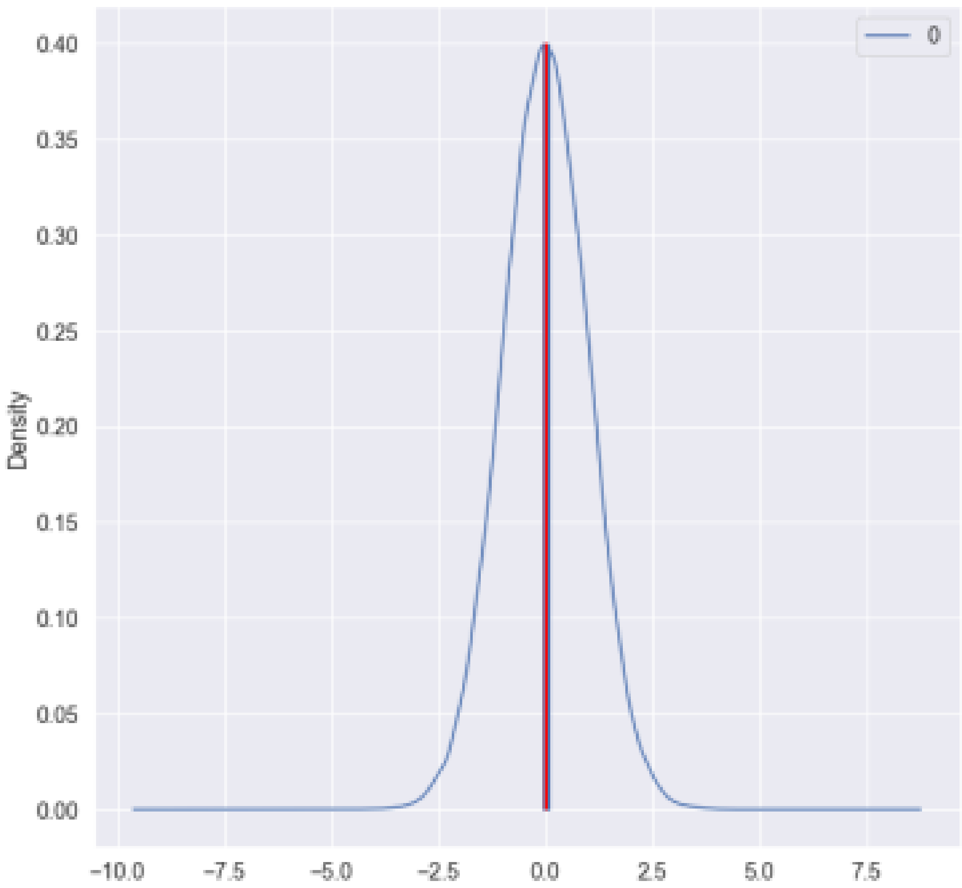
4. Perform descriptive statistics on the dataset

data.mean()

data.median()

```
In [9]: data.mean()

C:\Users\HP\AppData\Local\Temp\ipykernel_16792\883992179.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only
=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  data.mean()

Out[9]: Length            0.523992
        Diameter          0.407881
        Height            0.139516
        Whole weight      0.828742
        Shucked weight    0.359367
        Viscera weight    0.180594
        Shell weight      0.238831
        Rings             9.933684
        dtype: float64

In [10]: data.median()

C:\Users\HP\AppData\Local\Temp\ipykernel_16792\3972556868.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_onl
y=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  data.median()

Out[10]: Length            0.5450
         Diameter          0.4250
         Height            0.1400
         Whole weight      0.7995
         Shucked weight    0.3360
         Viscera weight    0.1710
         Shell weight      0.2340
         Rings             9.0000
         dtype: float64
```

```
In [11]:
norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
               figsize=(10,10));


plt.vlines(norm_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```



5. Check for Missing values and deal with them.

#identifying the missing value
df **=** pd**.**DataFrame(data)
df**.**isnull()

In [12]:
```
#identifying the missing value
df = pd.DataFrame(data)
df.isnull()
```

Out[12]:

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

4177 rows × 9 columns

#filling the missing value with previous value
df**.**fillna(method **=**'pad')

```
In [13]: #filling the missing value with previous value
         df.fillna(method ='pad')
```

Out[13]:

	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

#filling null values in missing values

data[0:]

```
In [14]: #filling null values in missing values
         data[0:]
```

Out[14]:

	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

6. Find the outliers and replace them outliers

#identifying the outliers

print(df['Shell weight'].skew())

df['Shell weight'].describe()

```
In [15]:
#identifying the outliers
print(df['Shell weight'].skew())
df['Shell weight'].describe()
```

```
0.6209268251392077
Out[15]:
count    4177.000000
mean        0.238831
std         0.139203
min         0.001500
25%         0.130000
50%         0.234000
75%         0.329000
max         1.005000
Name: Shell weight, dtype: float64
```

#replacing the outliers

print(df['Shell weight'].quantile(0.50))

print(df['Shell weight'].quantile(0.95))

df['Shell weight'] **=** np**.**where(df['Shell weight'] **>** 325, 140, df['Shell weight']) df**.**describe()

```
In [16]:
#replacing the outliers
print(df['Shell weight'].quantile(0.50))
print(df['Shell weight'].quantile(0.95))
df['Shell weight'] = np.where(df['Shell weight'] > 325, 140, df['Shell weight'])
df.describe()
```

```
0.234
0.48
```

Out[16]:

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

7. Check for Categorical columns and perform encoding.

#perform encoding

from sklearn.compose **import** make_column_selector **as** selector

categorical_columns_selector **=** selector(dtype_include**=**object)

categorical_columns **=** categorical_columns_selector(data)

categorical_columns

```
In [17]:
#perform encoding
from sklearn.compose import make_column_selector as selector

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

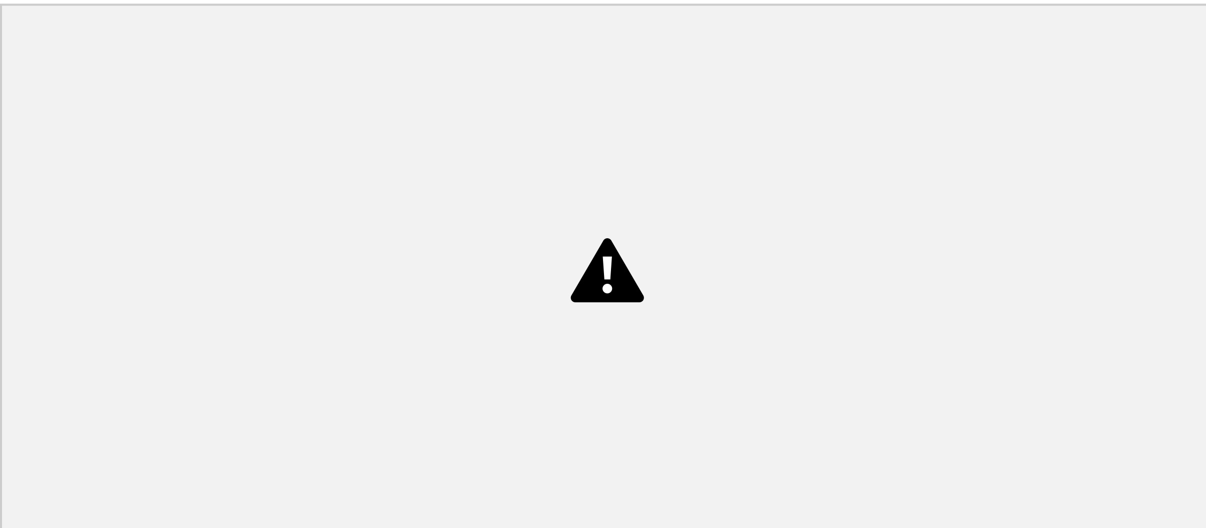
```
Out[17]: ['Sex']
```

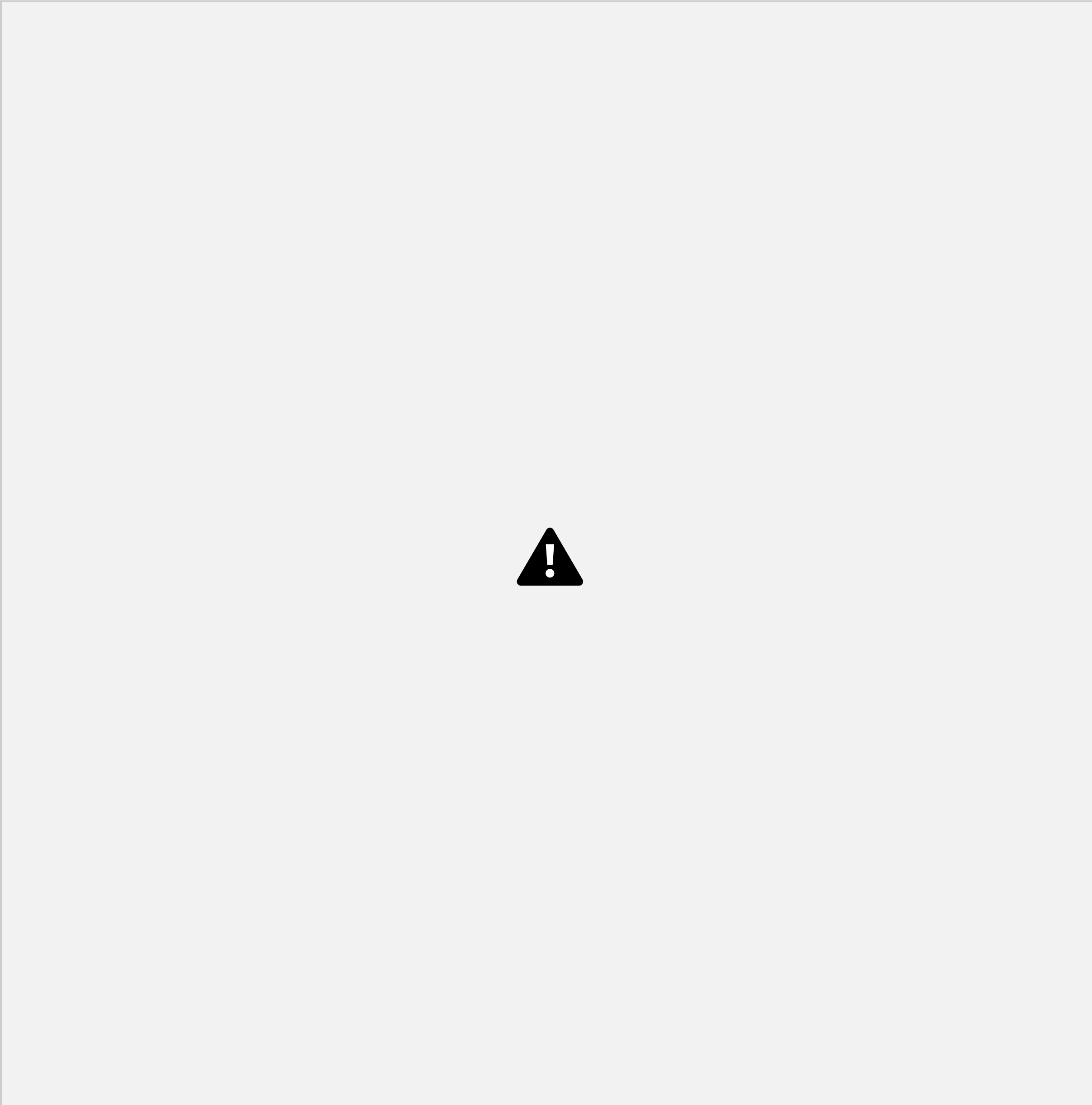
```
data_categorical = data[categorical_columns]
data_categorical.head()
```

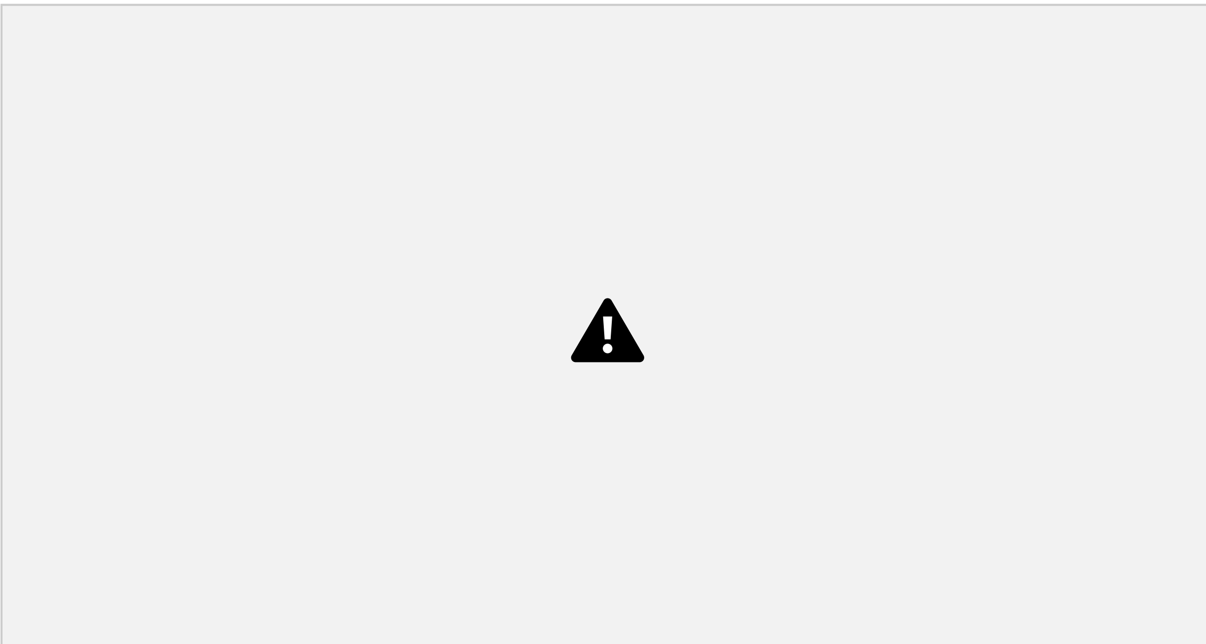
8. Split the data into dependent and independent variables.

```
from sklearn import preprocessing
# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()
# Encode labels in column 'species'.
df['Sex']= label_encoder.fit_transform(df['Sex'])
df['Sex'].unique()
X= data.iloc[ : , :-1].values
y= data.iloc[ : , 4].values
print(X,y)
# import packages
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# importing data
print(df.shape)
# head of the data
print('Head of the dataframe : ')
print(df.head())
print(df.columns)
X= df['Whole weight']
y=df['Shucked weight']
# using the train test split function
X_train, X_test, y_train, y_test = train_test_split(
X,y , random_state=104,test_size=0.25, shuffle=True)
```

```
# printing out train and test sets
print('X_train : ')
print(X_train.head())
print(X_train.shape)
print('')
print('X_test : ')
print(X_test.head())
print(X_test.shape)
print('')
print('y_train : ')
print(y_train.head())
print(y_train.shape)
print('')
print('y_test : ')
print(y_test.head())
print(y_test.shape)
```





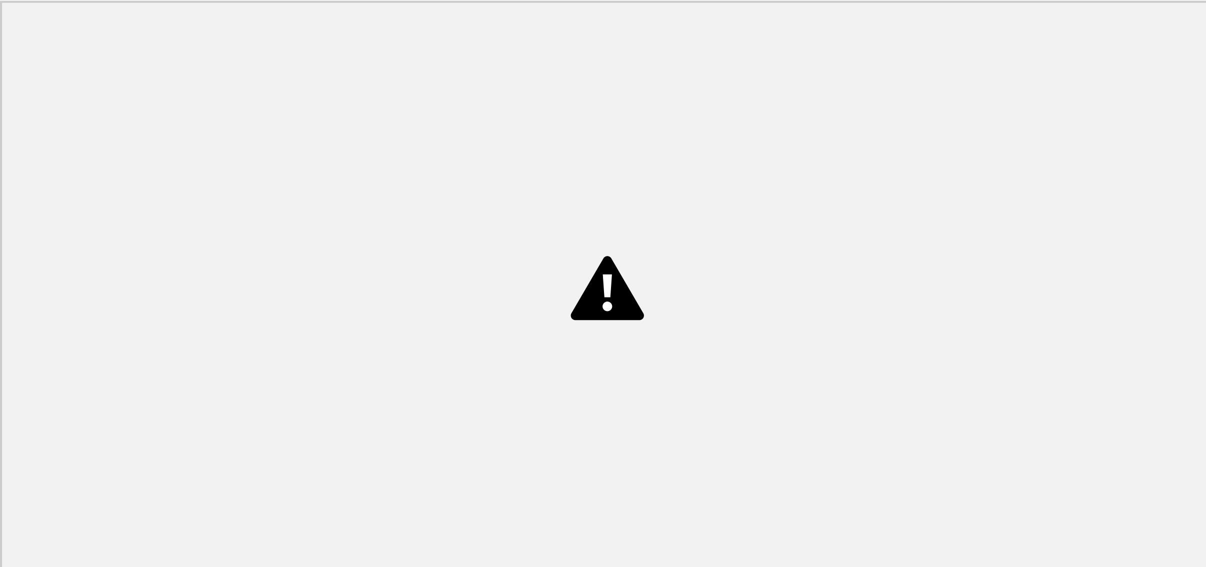


9. Scale the independent variables

#scaling

df_scaled = df.copy()

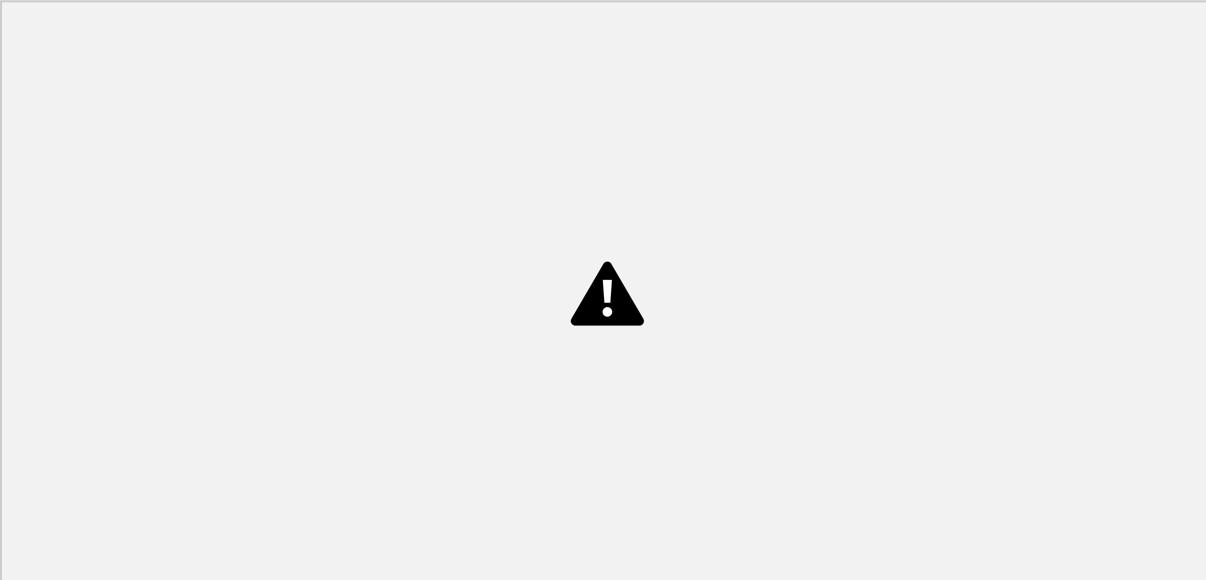
```
col_names = ['Shucked weight', 'Whole weight']
features = df_scaled[col_names]
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df_scaled[col_names] = scaler.fit_transform(features.values)
from sklearn.preproc
```

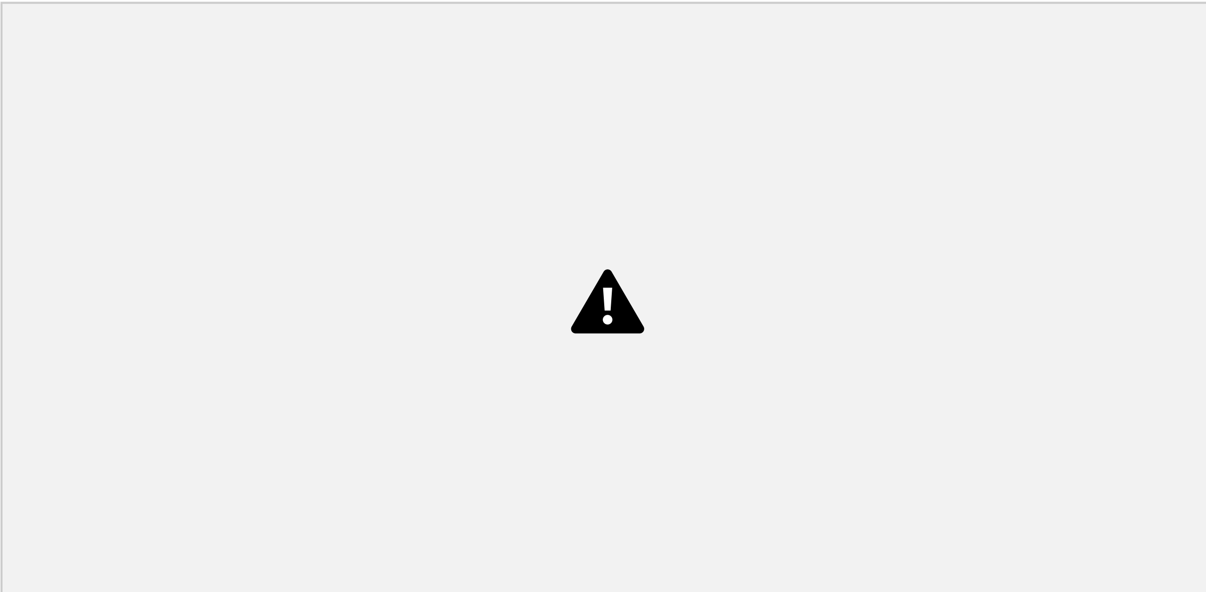


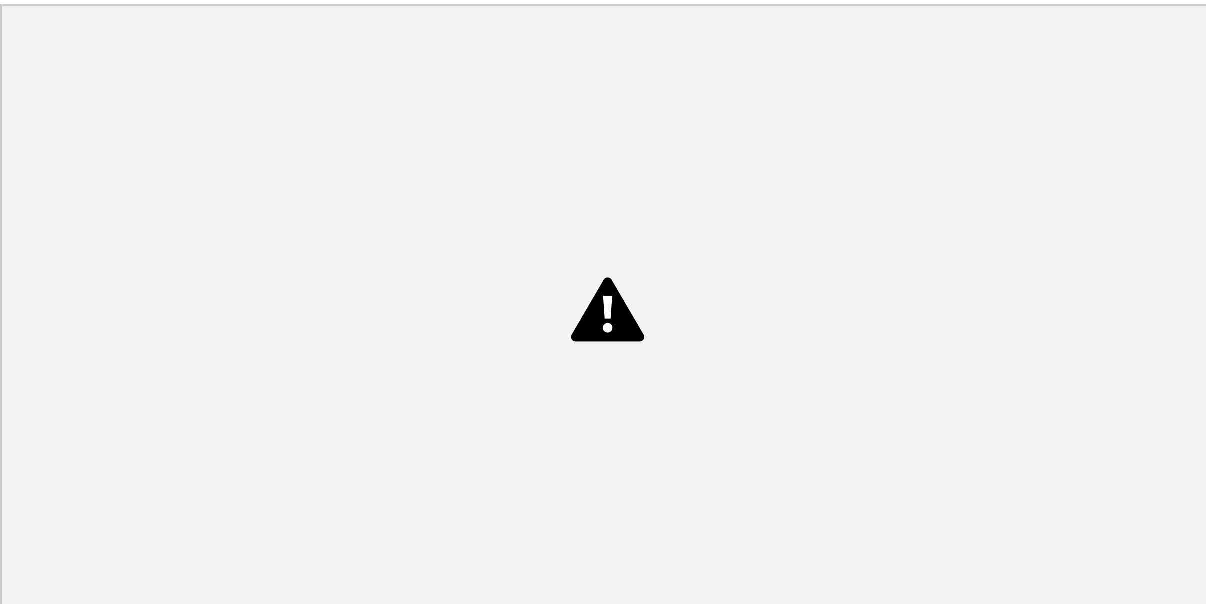
```
essing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(5, 10))
df_scaled[col_names] = scaler.fit_transform(features.values)
df_scaled
```

10. Split the data into training and testing

```
#testing and training
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.05, random_state=0)
print(X_train, X_test, y_train, y_test)
```

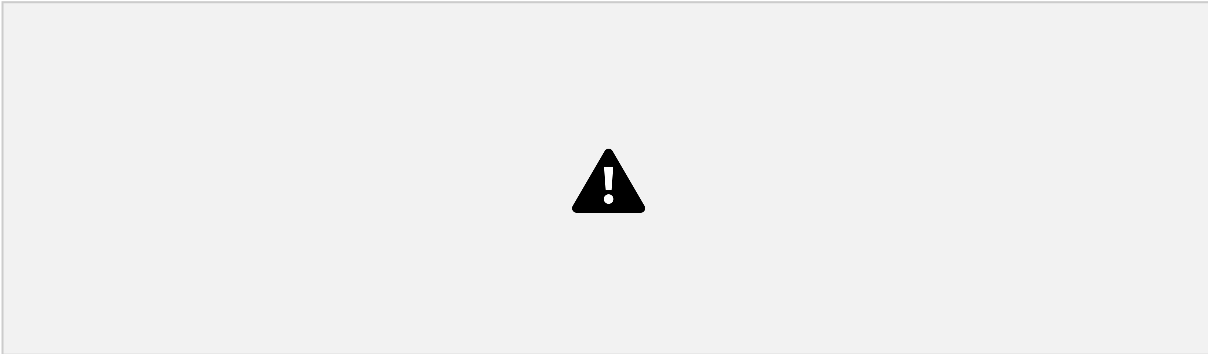






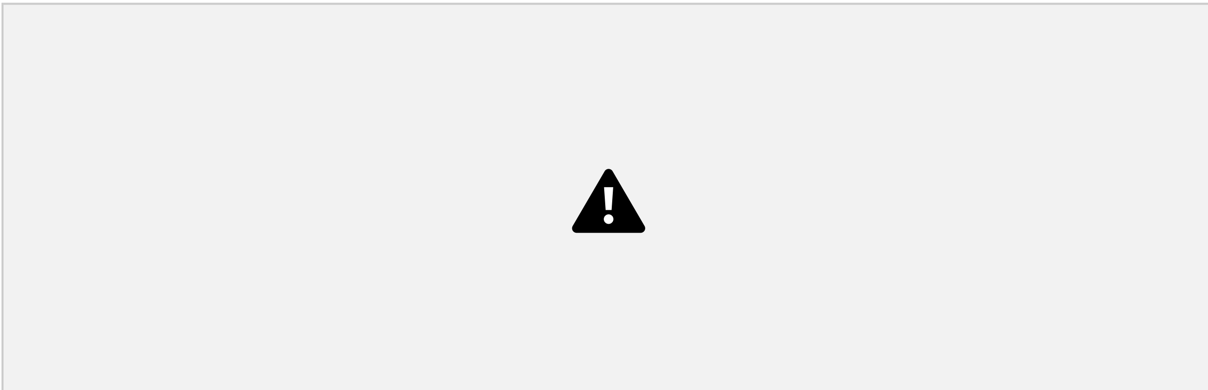
11. Build the Model

```
 # Evaluate the model on the test data
predictions = model.predict(X_test)
predictions
```



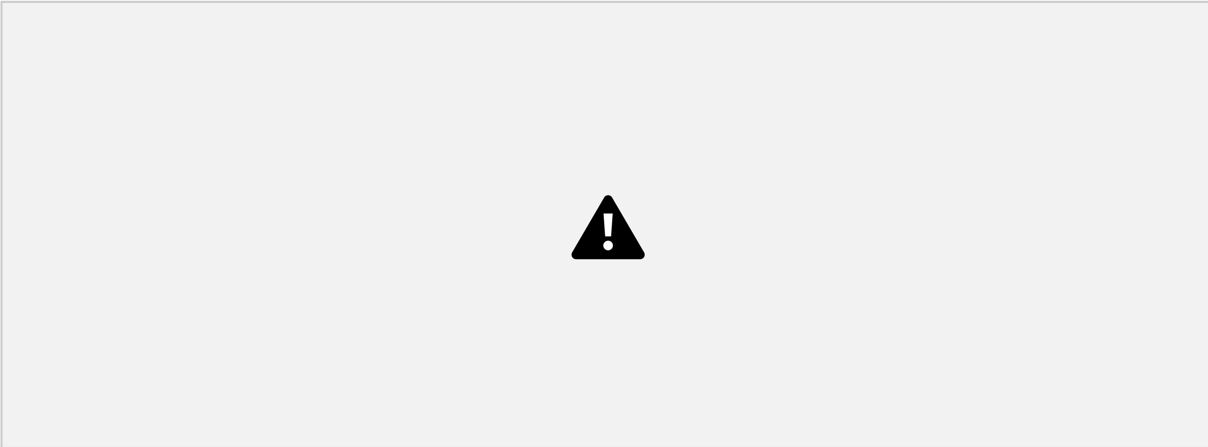
12. Train the Model

```
# Select algorithm
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
model = DecisionTreeClassifier()
# Fit model to the data
model.fit(X_train, y_train)
# Check model performance on training data
predictions = model.predict(X_train)
print(accuracy_score(y_train, predictions))
```



13. Test the Model

```
# Evaluate the model on the test data
predictions = model.predict(X_test)
predictions
```



14. Measure the performance using Metrics.

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files
(x86)/Graphviz2.38/bin' from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
X_actual = [1, 1, 0, 1, 0, 0, 1, 0, 0, 0]
Y_predic = [1, 0, 1, 1, 1, 0, 1, 1, 0, 0]
results = confusion_matrix(X_actual, Y_predic)
print ('Confusion Matrix :')
print(results)
print ('Accuracy Score is',accuracy_score(X_actual, Y_predic))
print ('Classification Report : ')
print (classification_report(X_actual, Y_predic))
print('AUC-ROC:',roc_auc_score(X_actual, Y_predic))
print('LOGLOSS Value is',log_loss(X_actual, Y_predic))
```