abalone

March 6, 2023

[1]:
```python
import pandas as pd
import numpy as np
from tensorflow.keras import layers
from tensorflow.keras import models
from sklearn.metrics import r2_score
```

[2]:
```python
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import math
from sklearn.model_selection import train_test_split
```

[3]:
```python
# to ignore warnings
import warnings
warnings.filterwarnings("ignore")
```

0.0.1 With the help of Pandas, read the ".csv" file and performing some task

[4]:
```python
data1 = pd.read_csv("abalone.csv")
```

[5]:
```python
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   @inputs Sex      4177 non-null   int64
 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   @outputs Rings   4177 non-null   int64
dtypes: float64(7), int64(2)
memory usage: 293.8 KB
```

0.0.2 Renaming the columns for better understanding

[6]:
```
data1.rename(columns = {'@inputs Sex':'Sex','@outputs Rings':'Rings'}, inplace␣
↪= True)
```

[7]:
```
data1.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   int64
 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   Rings           4177 non-null   int64
dtypes: float64(7), int64(2)
memory usage: 293.8 KB
```

[8]:
```
data1.head(4)
```

[8]:
```
   Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  \
0    3   0.400     0.305   0.100        0.3415          0.1760
1    2   0.635     0.500   0.150        1.3760          0.6495
2    3   0.370     0.270   0.090        0.1855          0.0700
3    1   0.680     0.540   0.155        1.5340          0.6710

   Viscera_weight  Shell_weight  Rings
0          0.0625        0.0865      7
1          0.3610        0.3100     10
2          0.0425        0.0650      7
3          0.3790        0.3840     10
```

[9]:
```
data1.tail(4)
```

[9]:
```
      Sex  Length  Diameter  Height  Whole_weight  Shucked_weight  \
4173    3   0.555     0.430   0.140        0.7665          0.3410
4174    3   0.485     0.380   0.120        0.4725          0.2075
4175    2   0.550     0.450   0.145        0.7410          0.2950
4176    3   0.530     0.415   0.145        0.9440          0.3845

      Viscera_weight  Shell_weight  Rings
4173          0.1650        0.2300      9
```

```
4174           0.1075        0.1470      6
4175           0.1435        0.2665     10
4176           0.1850        0.2650     21
```

[10]:
```
data1.isnull().any()
```

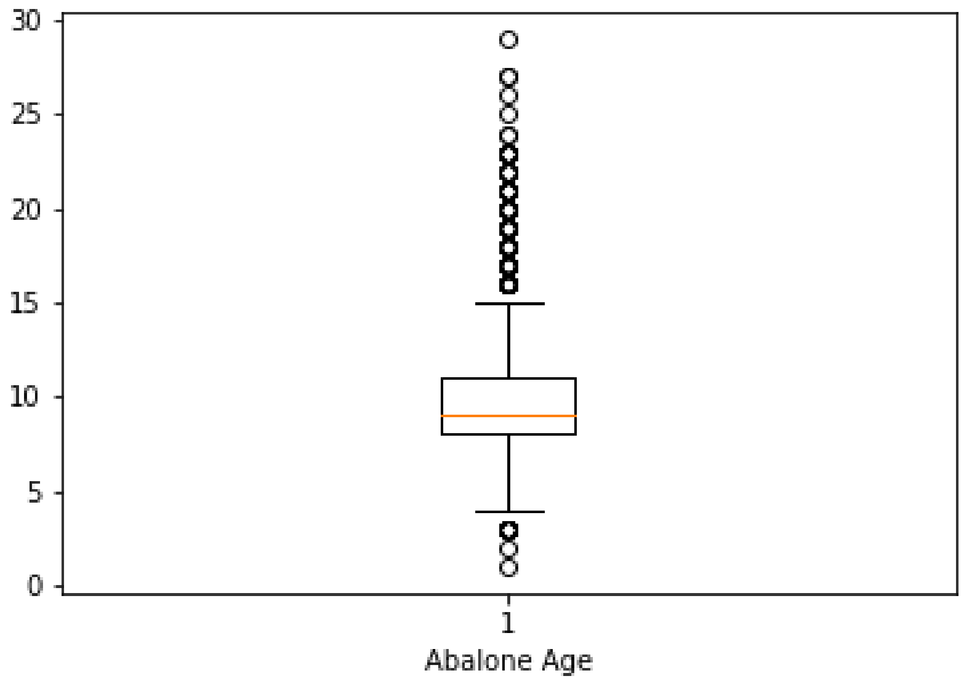
[10]:
```
Sex               False
Length            False
Diameter          False
Height            False
Whole_weight      False
Shucked_weight    False
Viscera_weight    False
Shell_weight      False
Rings             False
dtype: bool
```

[11]:
```
data1.isnull().sum()
```

[11]:
```
Sex               0
Length            0
Diameter          0
Height            0
Whole_weight      0
Shucked_weight    0
Viscera_weight    0
Shell_weight      0
Rings             0
dtype: int64
```

[12]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

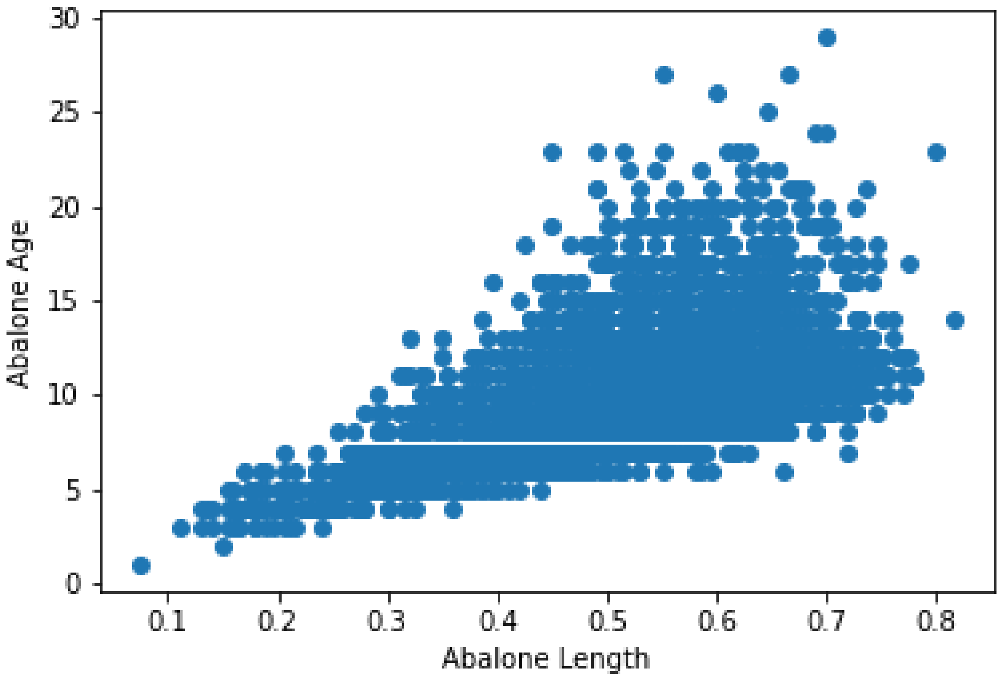
[13]:
```
# Create a boxplot of the age variable
plt.boxplot(data1["Rings"])
plt.xlabel("Abalone Age")
plt.show()
```



[14]:

```
# Create a scatterplot of Length vs Age
plt.scatter(data1["Length"], data1["Rings"])
plt.xlabel("Abalone Length")
plt.ylabel("Abalone Age")
plt.show()

#scatterplot is best used to detect Trends, Outilier detection
```




```
[15]: # Split the dataset into training and testing sets
X = data1.drop("Rings", axis=1) # Independent variables
y = data1["Rings"] # Dependent variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.0.3 Building the neural network model

```
[16]: def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu', input_shape=(X.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

0.0.4 Define and perform K-fold cross validation

```
[17]: kf = KFold(n_splits=5)
```

```
[18]: # Perform k-fold cross validation
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

0.0.5 With nural network

[19]:
```
 # Standardize the training and testing data
X_train_mean = X_train.mean()
X_train_std = X_train.std()
X_train = (X_train - X_train_mean) / X_train_std
X_test = (X_test - X_train_mean) / X_train_std
```

[20]:
```
# Build the model
model = build_model()
```

[21]:
```
history = model.fit(X_train, y_train, epochs=30 , batch_size=5, verbose=0)
```

[22]:
```
mse_scores = []
mae_scores = []
r2_scores = []
rmse_scores = []
```

[23]:
```
  # Predict the target values for the testing data
y_pred = model.predict(X_test).flatten()
```

```
27/27 [==============================] - 0s 1ms/step
```

[24]:
```
mse, mae = model.evaluate(X_test, y_test, verbose=0)
mse_scores.append(mse)
mae_scores.append(mae)
r2 = r2_score(y_test, y_pred)
r2_scores.append(r2)
```

[25]:
```
rmse = np.sqrt(mse)
rmse_scores.append(rmse)
```

[26]:
```
print('Mean Squared Error (MSE):', np.mean(mse_scores))
print('Mean Absolute Error (MAE):', np.mean(mae_scores))
print('Coefficient of determination R-squared (R2):', np.mean(r2_scores))
print("Root Mean Squared Error (RMSE): ", rmse)
```

```
Mean Squared Error (MSE): 4.532966136932373
Mean Absolute Error (MAE): 1.5163860321044922
Coefficient of determination R-squared (R2): 0.5779419097025543
Root Mean Squared Error (RMSE):  2.129076357703587
```

0.0.6 Intialize the linear regression model and fit the model

[27]:
```
model = LinearRegression()
```

[28]:
```
model.fit(X_train, y_train)
```

[28]:
```
LinearRegression()
```

[29]:
```
# Make predictions on the test data
y_pred = model.predict(X_test)
```

[30]:
```
# Initialize a list to store the mean squared errors
mse_scores = []
```

[31]:
```
# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)
mse_scores.append(mse)
```

[32]:
```
# Calculating the Mean Square Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE): ", mse)
```

```
Mean Squared Error (MSE):  5.585855026952151
```

[33]:
```
# Calculating the R2 Score
r2 = r2_score(y_test, y_pred)
print("Coefficient of determination R-squared (R2): ", r2)
```

```
Coefficient of determination R-squared (R2):  0.4799089551478021
```

[34]:
```
# Calculating the Root Mean Squared error
rmse = math.sqrt(mse)
print("Root Mean Squared Error (RMSE): ", rmse)
```

```
Root Mean Squared Error (RMSE):  2.363441352551857
```

[35]:
```
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE): ", mae)
```

```
Mean Absolute Error (MAE):  1.6357276858043999
```

0.0.7 By using XG Boost algorithm

[36]:
```
import xgboost as xgb
```

[37]:
```
# Define the XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror')
```

[38]:
```
xgb_model.fit(X_train, y_train)
```

[38]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
```

```
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, …)
```

[39]:
```python
# Predict the target values for the testing data
y_pred = xgb_model.predict(X_test)
```

[40]:
```python
mse_scores = []
mae_scores = []
r2_scores = []
rmse_scores = []
```

[41]:
```python
# Evaluate the model on the testing data
mse = mean_squared_error(y_test, y_pred)
mae = np.mean(np.abs(y_test - y_pred))
r2 = r2_score(y_test, y_pred)
rmse = math.sqrt(mse)
```

[42]:
```python
mse_scores.append(mse)
mae_scores.append(mae)
r2_scores.append(r2)
```

[43]:
```python
# Compute the mean evaluation metrics over all the folds
print('Mean Squared Error (MSE):', np.mean(mse_scores))
print('Mean Absolute Error (MAE):', np.mean(mae_scores))
print('Coefficient of determination R-squared (R2):', np.mean(r2_scores))
print("Root Mean Squared Error (RMSE): ", rmse)
```

```
Mean Squared Error (MSE): 5.557944584159364
Mean Absolute Error (MAE): 1.6627155004147285
Coefficient of determination R-squared (R2): 0.48250765691939257
Root Mean Squared Error (RMSE):  2.357529338981673
```

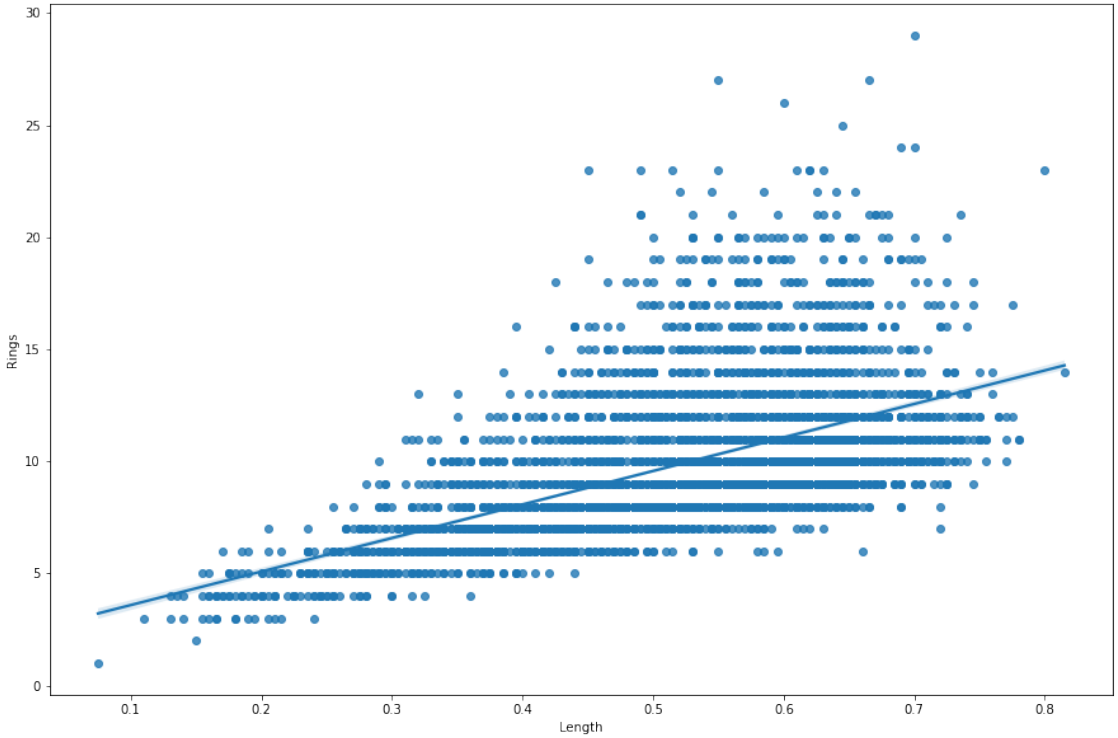
0.0.8 Plotting regression graph

[44]:
```python
# Selecting the relevant columns for the regression
x = data1['Length']
y = data1['Rings']

#enlarging the figure for better visualization
fig, ax = plt.subplots(figsize=(15, 10))

# Fit a regression model
sns.regplot(x, y, ax=ax)

# Show the plot
plt.show()
```



By Sushan Shankar