Data Analytics Assignment - 4: Abalon Age Prediction

Team ID : PNT2022TMID45269

Student Name: GRANAF P

Project Name :Visualizing and Predicting Heart Diseases with an Interactive Dash Board

Student Roll No:812119104008

Dataset : https://drive.google.com/file/d/1mOWrMc8b-ODshkEfyHB1UFwO5V5s3fcW/view

Import Necesssary packages

+ Code + Text

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression

from sklearn import  metrics
```

Download and Load the dataset

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

Perform descriptive statistics on the dataset

```
df.head()
```

```
df.tail()
```

	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
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

```
df.shape
```

```
(4177, 9)
```

```
df.describe()
```

	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.

```
df.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   Rings           4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
```

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

```
df.head()
```

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

Check for Missing values and deal with them

```
df.isnull().sum()
```

```
Sex               0
Length            0
Diameter          0
Height            0
Whole weight      0
Shucked weight    0
Viscera weight    0
Shell weight      0
age               0
dtype: int64
```

```
df.columns
```

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

Perform Below Visualizations

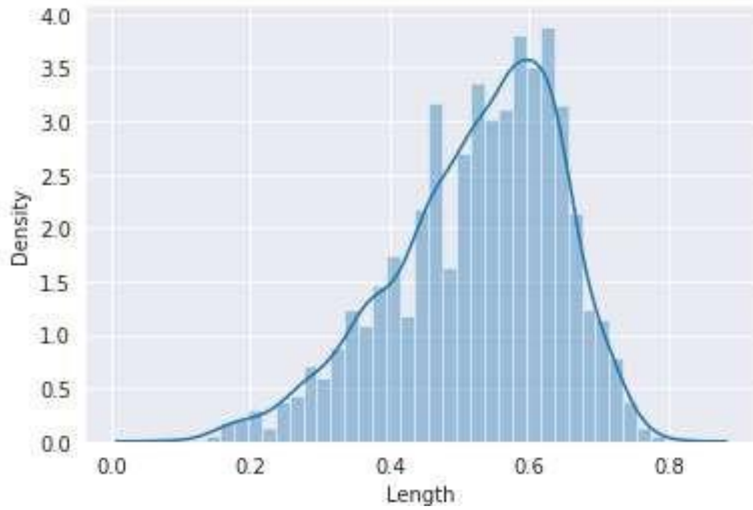
· Univariate Analysis

Bi-Variate Analysis

· Multi-Variate Analysis

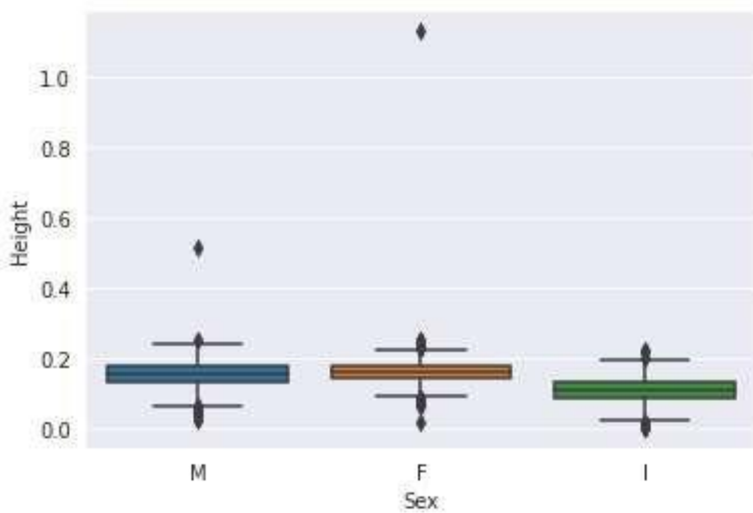
```
#univariate analysis
sns.distplot(df['Length'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f89eee6b4d0>
```



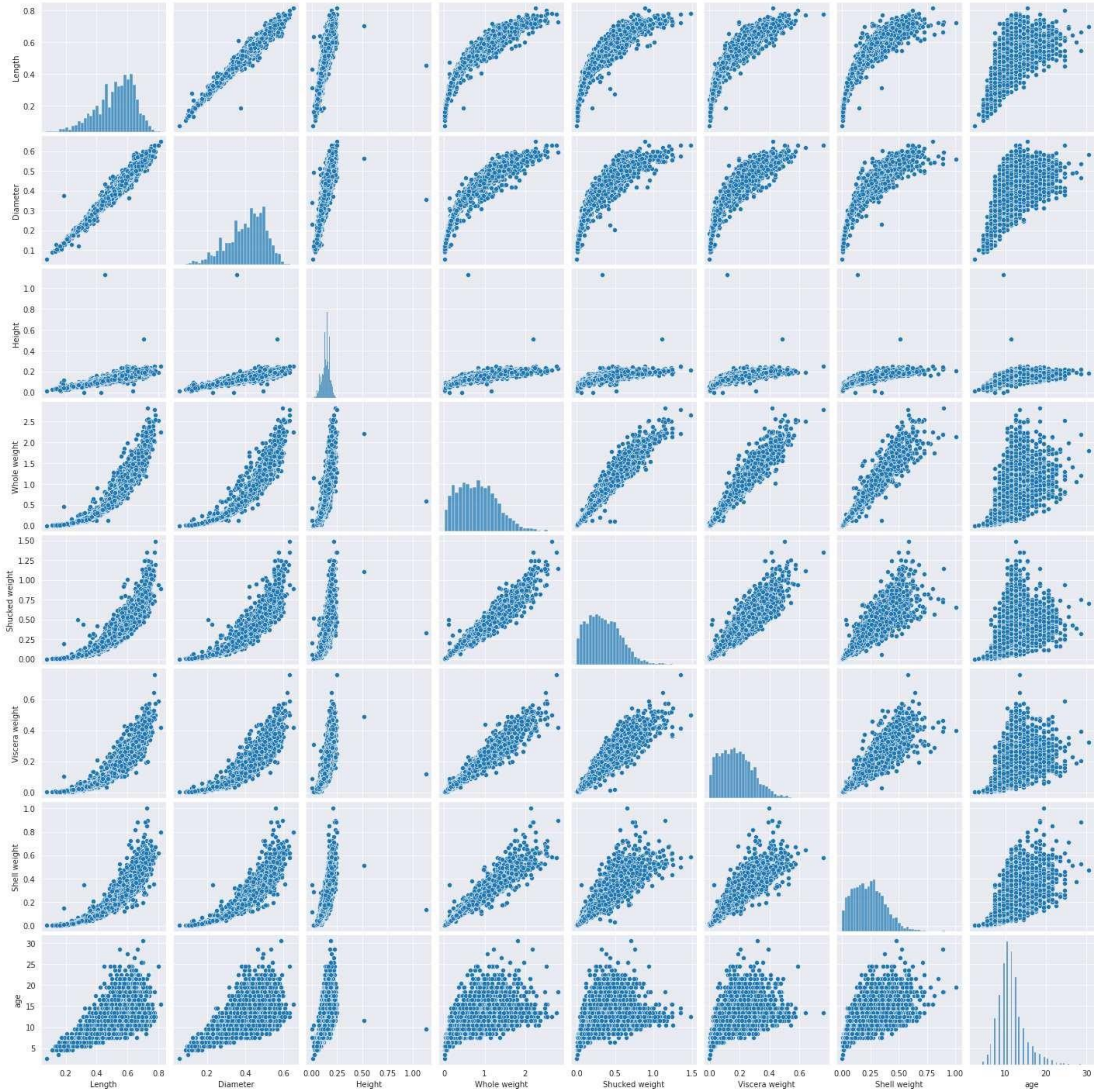

```
#Bi-variate analysis
sns.boxplot(df.Sex,df.Height)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f89ee86ca90>
```




```
#Multi-variate analysis
sns.pairplot(df)
```

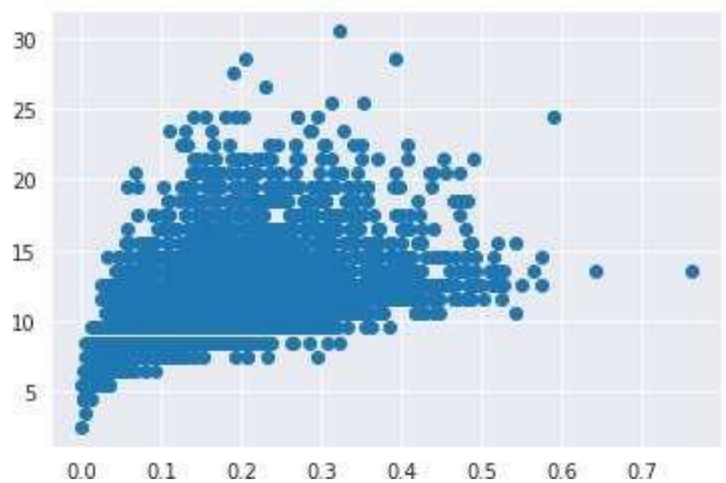
<seaborn.axisgrid.PairGrid at 0x7f89ee8162d0>



Find the outliers and replace them outliers

```
#Data Preprocessing
#Outlier handling
df = pd.get_dummies(df)
dummy_df = df
```

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



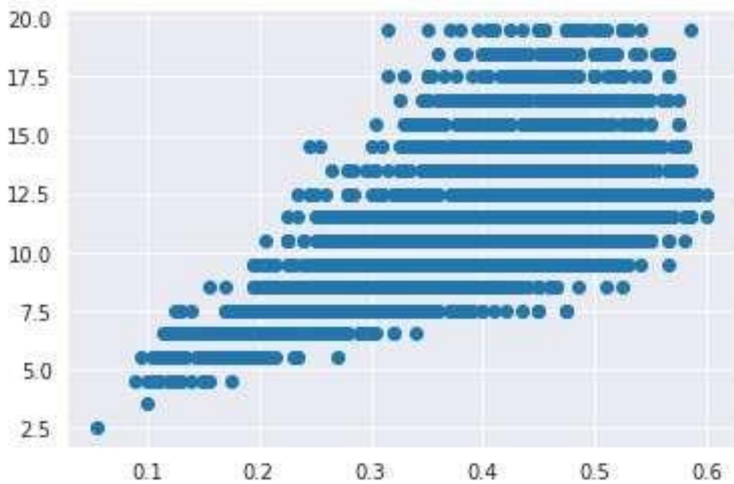

```
df.drop(df[(df['Viscera weight'] > 0.5) &
          (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<0.5) & (
df['age'] > 25)].index, inplace = True)
```

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

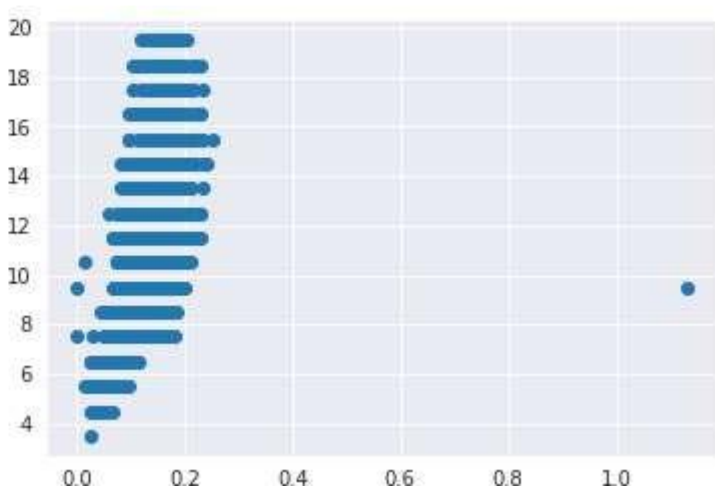

```
df.drop(df[(df['Shell weight'] > 0.6) & (df['age'] < 25)].index, inplace = True)
df.drop(df[(df['Shell weight']<0.8) & (df['age'] > 25)].index, inplace = True)
var = 'Shucked weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```


```
df.drop(df[(df['Shucked weight'] >= 1) & (df['age'] < 20)].index, inplace = True)
df.drop(df[(df['Viscera weight']<1) & (df['age'] > 20)].index, inplace = True)
var = 'Whole weight'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```

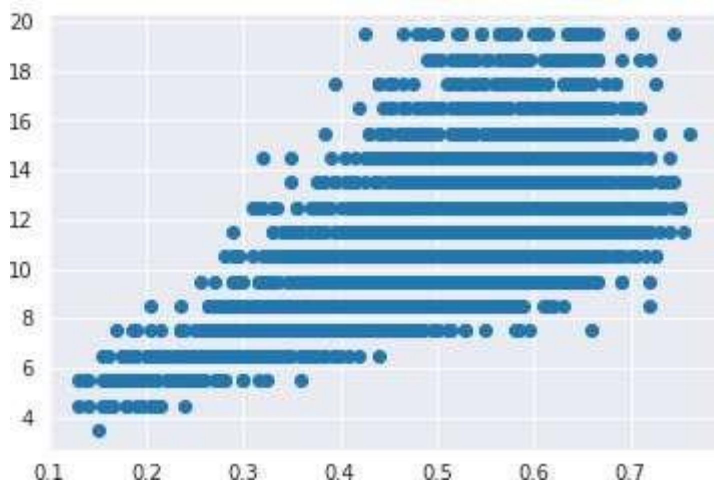

```
df.drop(df[(df['Whole weight'] >= 2.5) & (df['age'] < 25)].index, inplace = True)
df.drop(df[(df['Whole weight']<2.5) & (df['age'] > 25)].index, inplace = True)
var = 'Diameter'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```




```
df.drop(df[(df['Diameter'] <0.1) & (df['age'] < 5)].index, inplace = True)
df.drop(df[(df['Diameter']<0.6) & (df['age'] > 25)].index, inplace = True)
df.drop(df[(df['Diameter']>=0.6) & (df['age'] < 25)].index, inplace = True)
var = 'Height'
plt.scatter(x = df[var], y = df['age'])
plt.grid(True)
```




```
df.drop(df[(df['Height'] > 0.4) &(df['age'] < 15)].index, inplace = True)
df.drop(df[(df['Height']<0.4) & (df['age'] > 25)].index, inplace = True)
var = 'Length'
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: `np
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
```

```
numerical_features
```

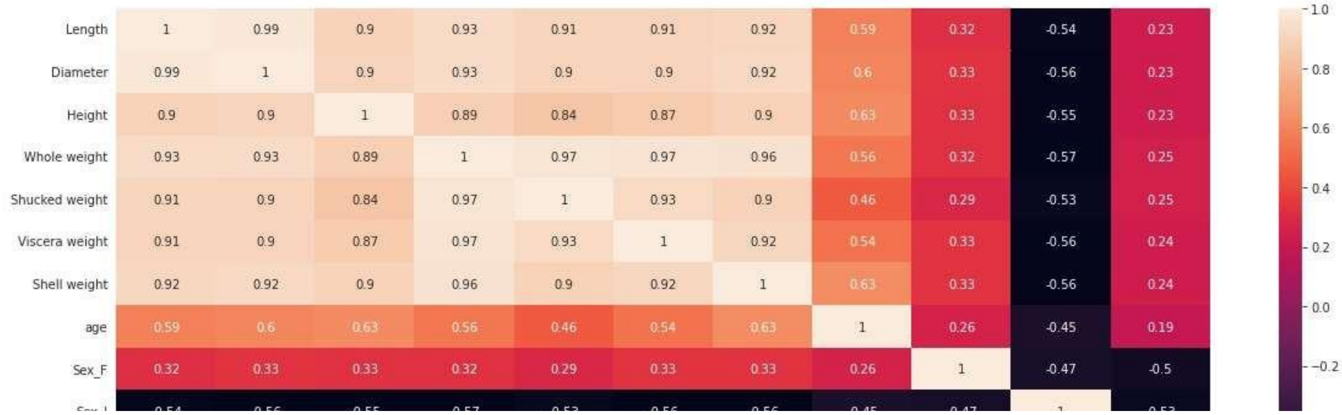
```
Index(['Length', 'Diameter', 'Height', 'Whole weight', 'Shucked weight',
       'Viscera weight', 'Shell weight', 'age', 'Sex_F', 'Sex_I', 'Sex_M'],
      dtype='object')
```

```
categorical_features
```

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

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f89e86a7390>
```



Whole Weight is almost linearly varying with all other features except age. Height has least linearity with remaining features. Age is most linearly proprtional with Shell Weight followed by Diameter and length. Age is least correlated with Shucked Weight.

KEY INSIGHT All numerical features but 'sex'

-> Though features are not normaly distributed, are close to normality

-> None of the features have minimum = 0 except Height (requires re-check)

-> Each feature has difference scale range

```
df.columns
```

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

Feature Selection and Standardization

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

LINEAR REGRESSION

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

```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression()
```

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

```
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)

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

```
Mean Squared Error of training set :3.569916
Mean Squared Error of testing set :3.526501
```

Note: The Lower the Mean Squared Error,better the forecast.

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

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

```
R2 Score of training set:0.53
R2 Score of testing set:0.53
```

Note: The ideal value of R-square is 1.

The closer the value of R-square to 1,better is the model fitted.