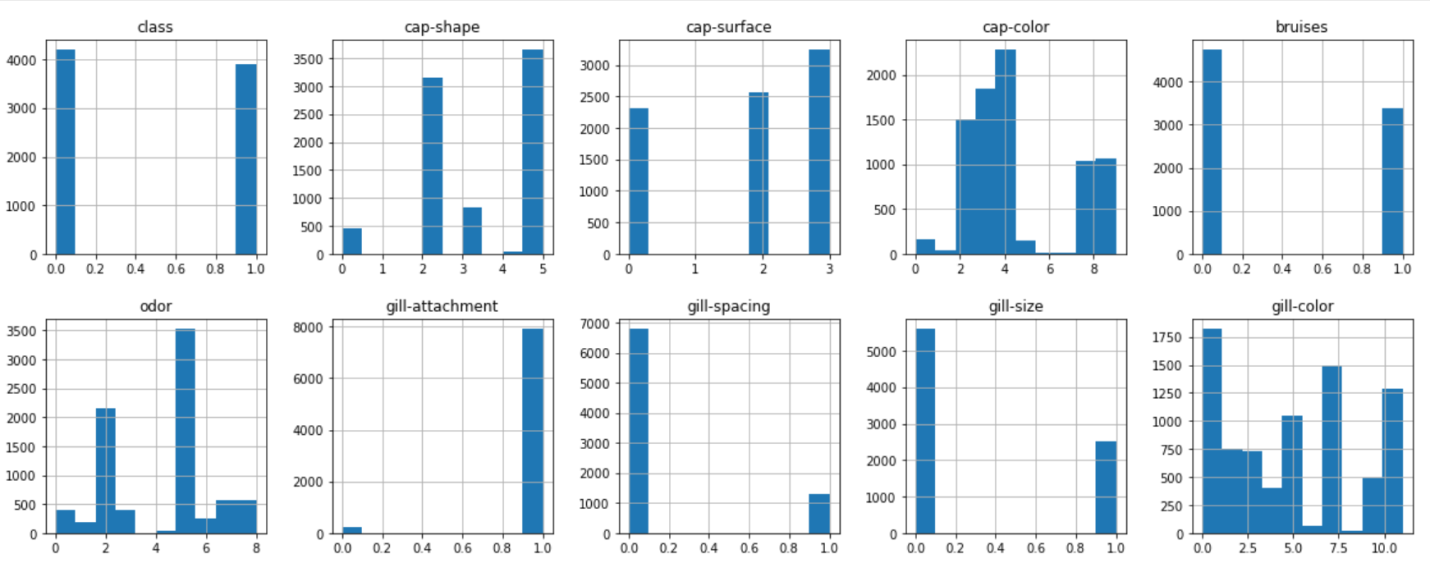
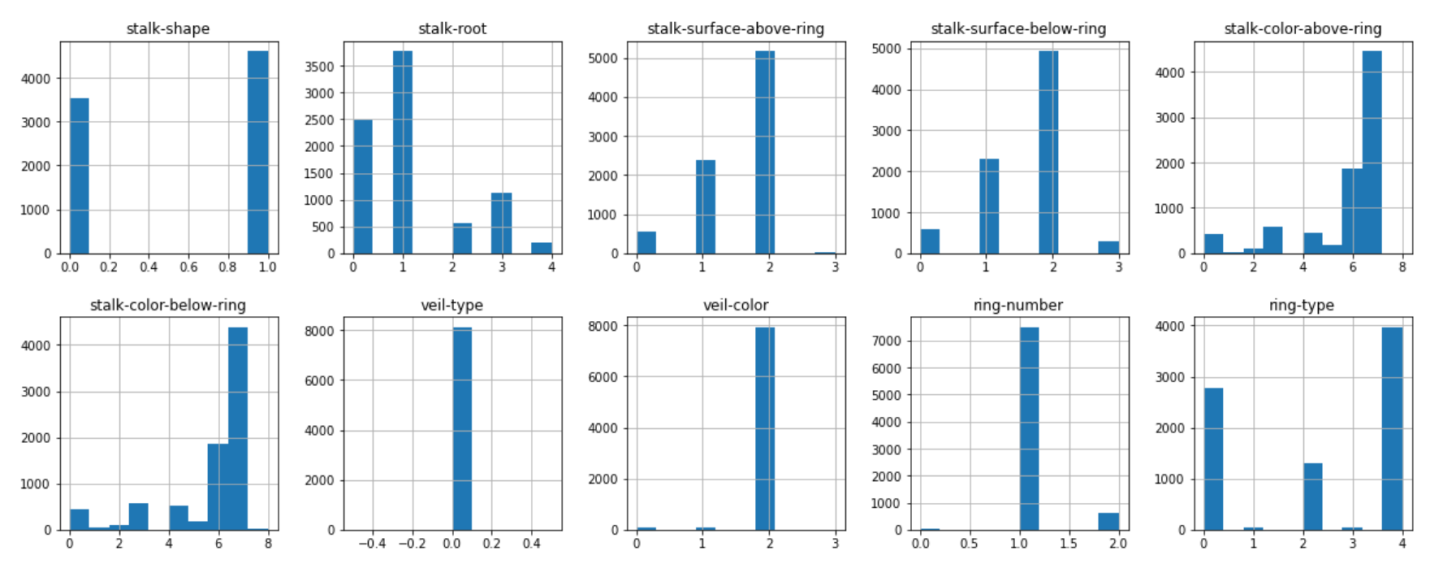
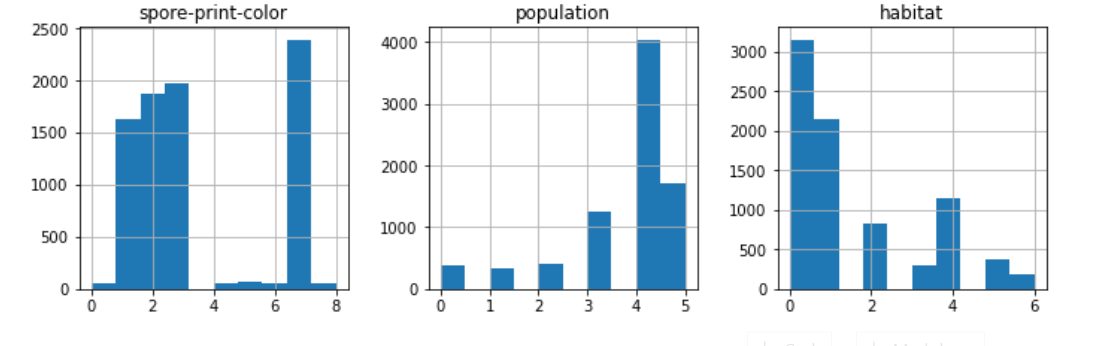
**EDA for Mushroom Dataset**

**Histogram**

data.hist(figsize=(20,20));



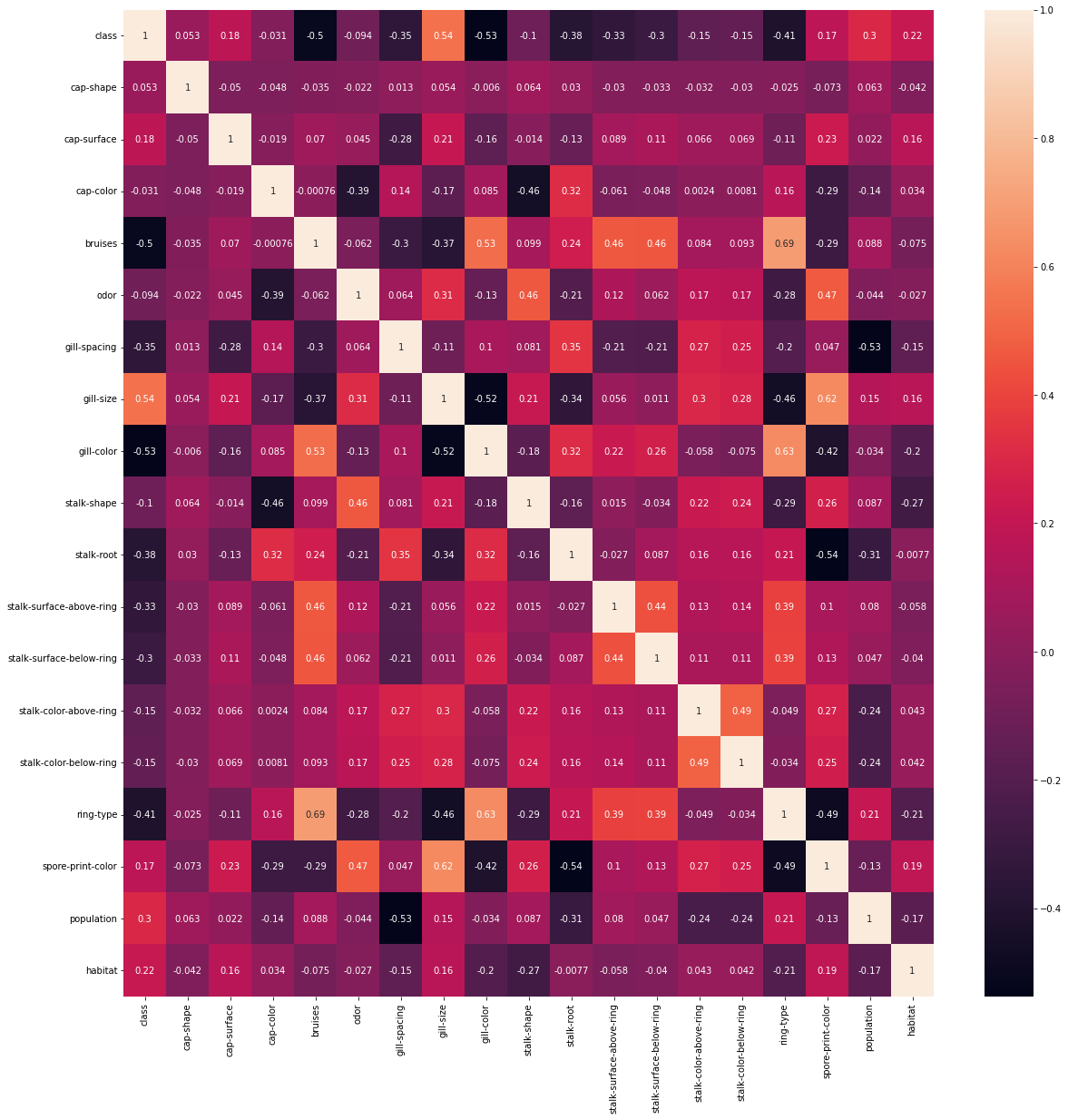




**Heatmap**

plt.figure(figsize = (20,20))

sns.heatmap(data.corr(), annot=True);



**Classification**

**Installing and Importing Libraries**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

**Reading Data**

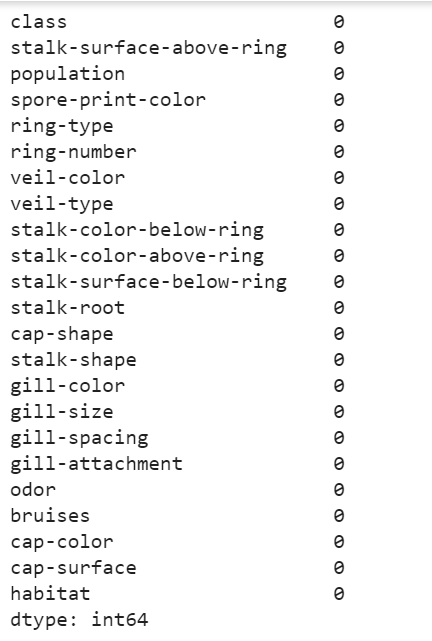
DATA = pd.read\_csv('../datasets/mushrooms.csv')

DATA.head()

DATA.shape

**Checking for missing values**

DATA.isna().sum().sort\_values(ascending=False)



**Label Encoding**

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

data = pd.DataFrame()

for col in DATA.columns:

  data[col] = encoder.fit\_transform(DATA[col])

**Feature Selection**

from feature\_engine.selection import SmartCorrelatedSelection, DropConstantFeatures, DropDuplicateFeatures

from sklearn.pipeline import Pipeline

pip = Pipeline([('constant', DropConstantFeatures(tol=0.9)), ('duplicate', DropDuplicateFeatures()), ('correlated', SmartCorrelatedSelection())])

data = pip.fit\_transform(data)

data.shape

**ML Model**

X = data.drop('class', axis=1)

Y = data['class']

X.shape, Y.shape



from sklearn.model\_selection import train\_test\_split

xTrain, xTest, yTrain, yTest = train\_test\_split(X, Y)

1. **Naïve Bayes**

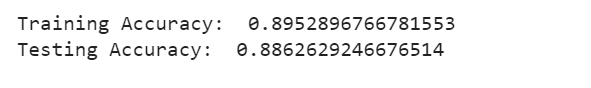
from sklearn.naive\_bayes import GaussianNB

nb = GaussianNB()

nb.fit(xTrain, yTrain)

print("Training Accuracy: ",nb.score(xTrain, yTrain))

print("Testing Accuracy: ",nb.score(xTest, yTest))



1. **KNN**

from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n\_neighbors=1)

knn.fit(xTrain, yTrain)

yPred1 = knn.predict(xTrain)

yPred2 = knn.predict(xTest)

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

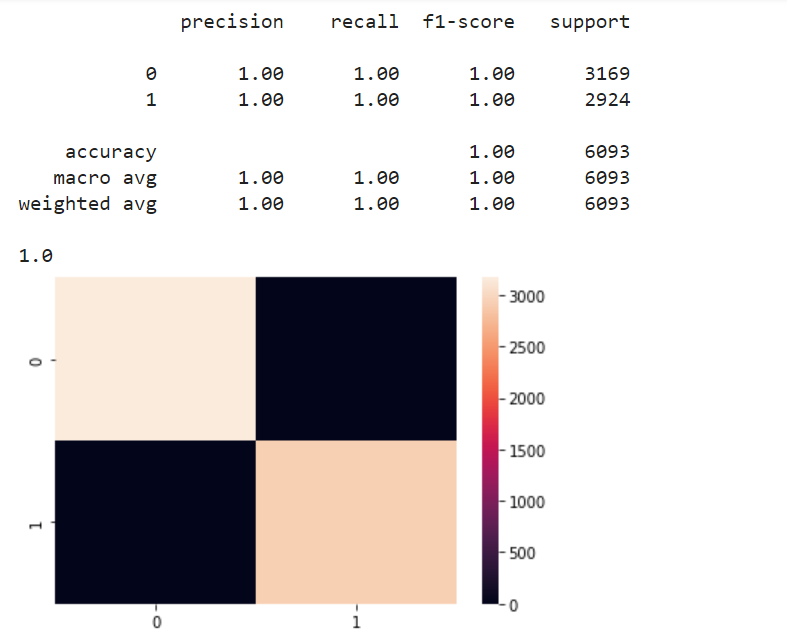
matrix = confusion\_matrix(yTrain, yPred1)

print(classification\_report(yTrain, yPred1))

sns.heatmap(matrix)

accuracy\_score(yTrain, yPred1)

**Result 1**



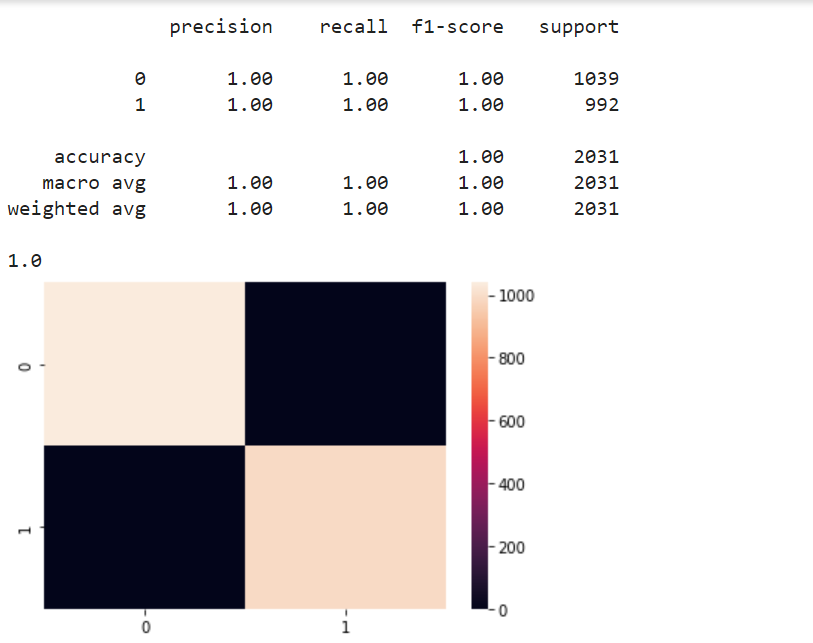
**Result 2**

matrix = confusion\_matrix(yTest, yPred2)

print(classification\_report(yTest, yPred2))

sns.heatmap(matrix)

accuracy\_score(yTest, yPred2)

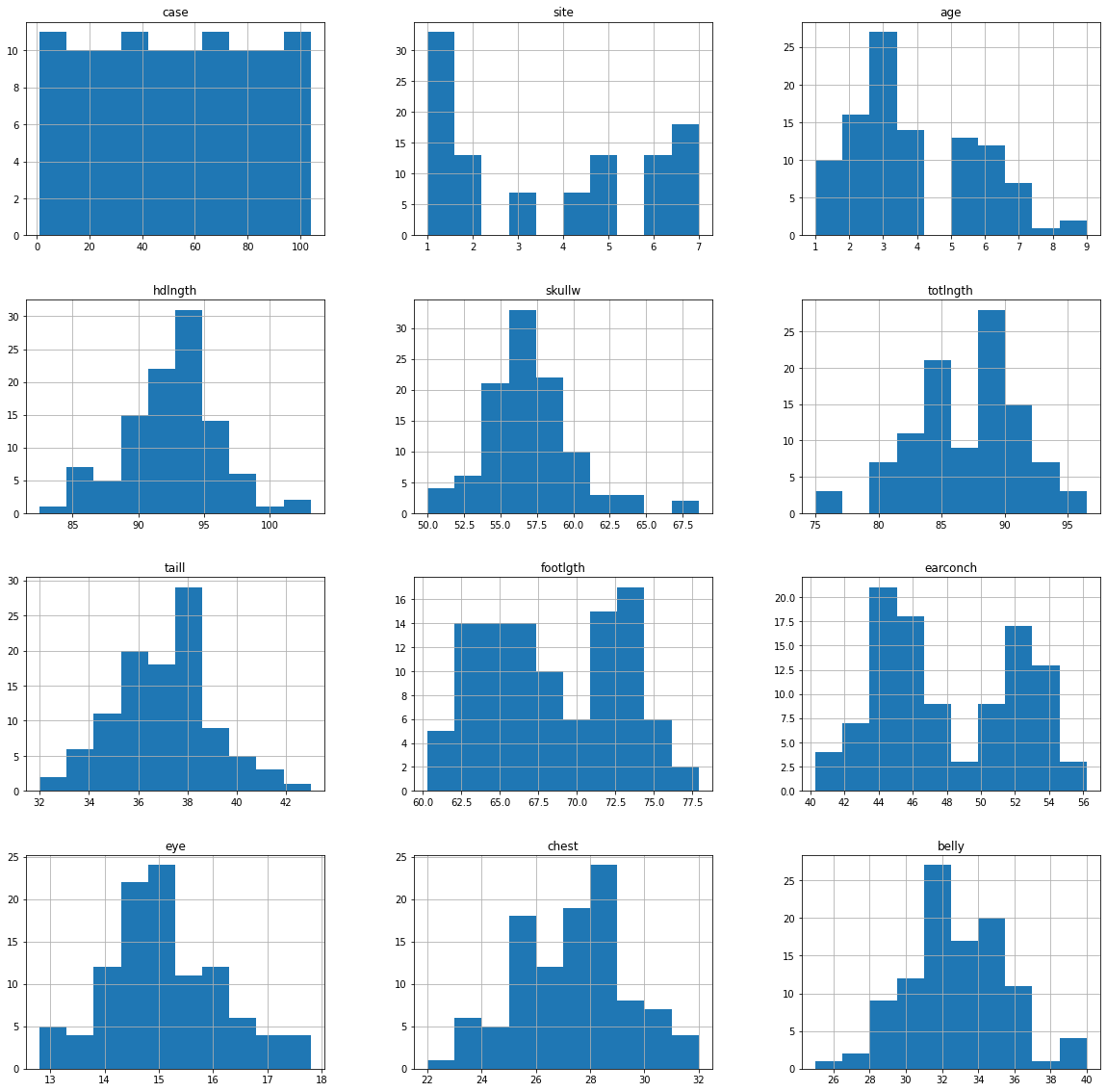


**Comparison:**

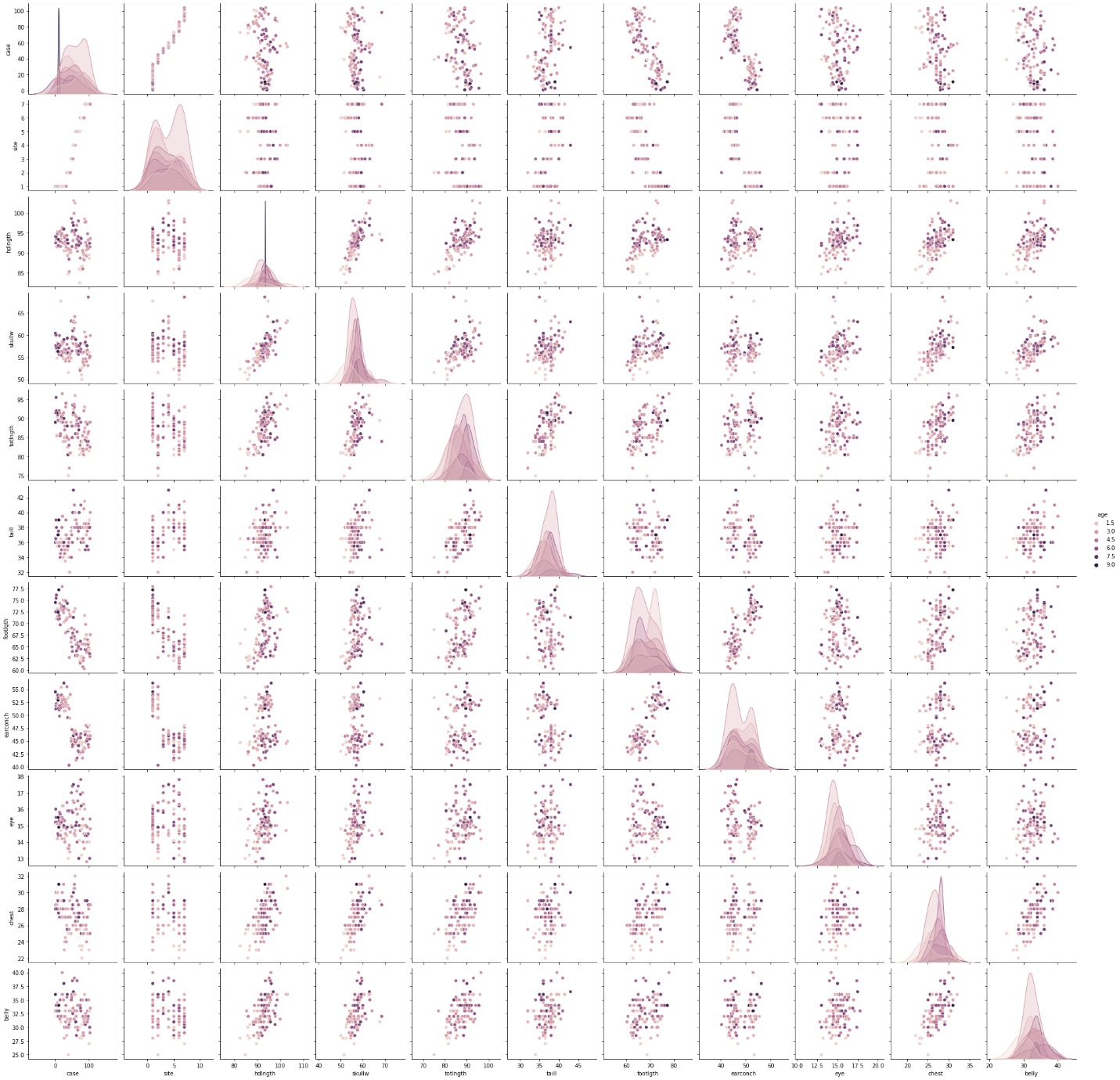
By analyzing all the steps of EDA and doing all the preprocessing, training-testing steps we can observed that both the models can predict the output with the same accuracy.

**EDA for Possum Dataset**

DATA.hist(figsize=(20,20));

****

sns.pairplot(DATA,hue='age');



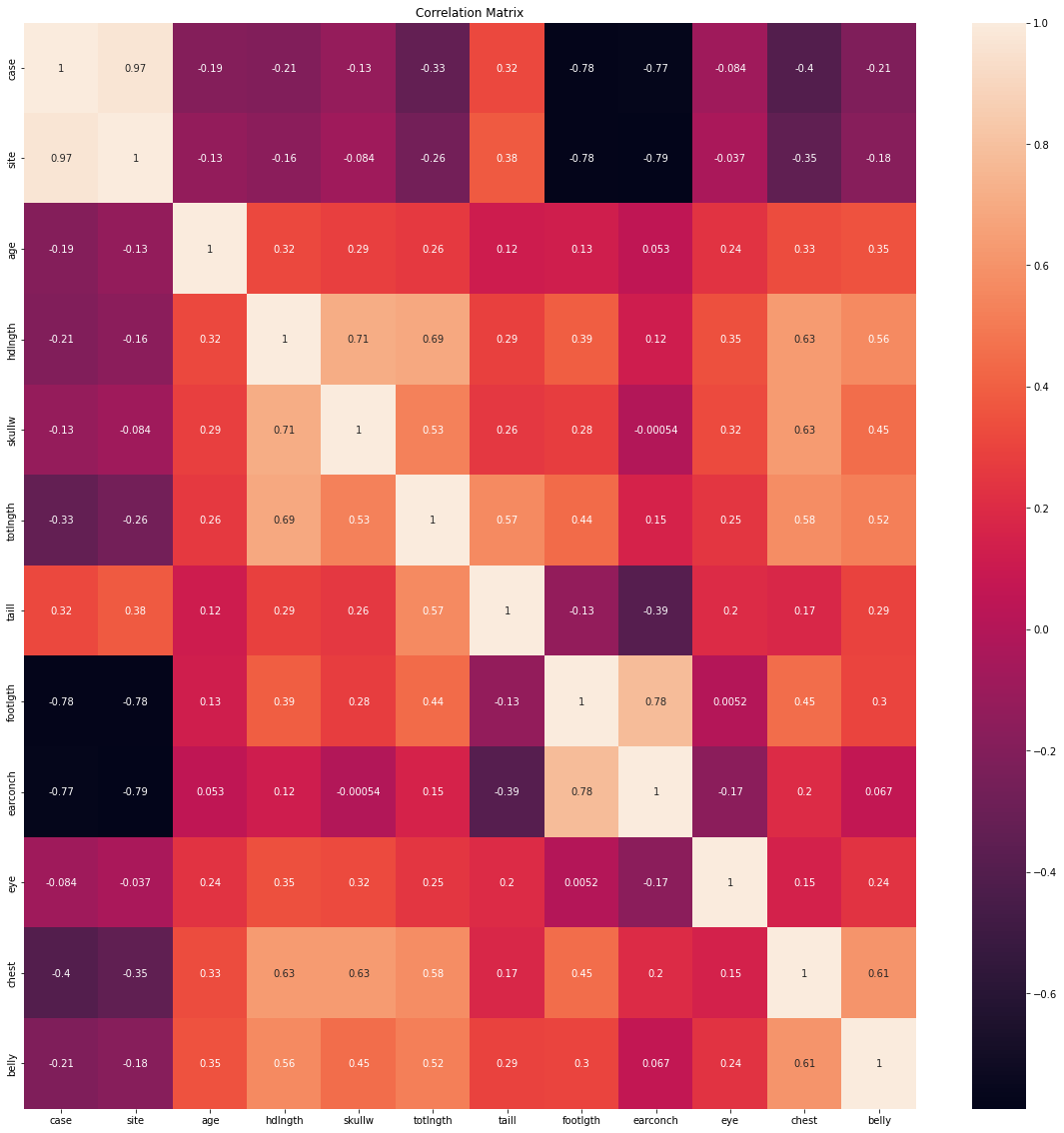
corr = DATA.corr()

plt.figure(figsize = (20,20))

sns.heatmap(corr, annot=True);

plt.title("Correlation Matrix");

plt.show()

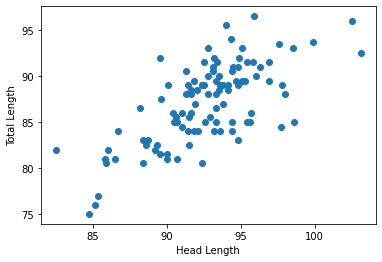
****

plt.scatter(DATA['hdlngth'], DATA['totlngth'])

plt.xlabel('Head Length')

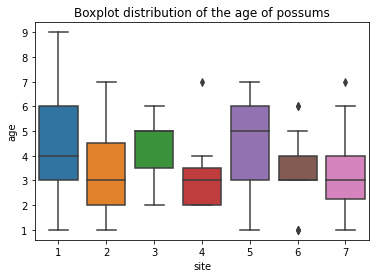
plt.ylabel('Total Length')

Text(0, 0.5, 'Total Length')

****

sns.boxplot(x="site", y="age", data= DATA);

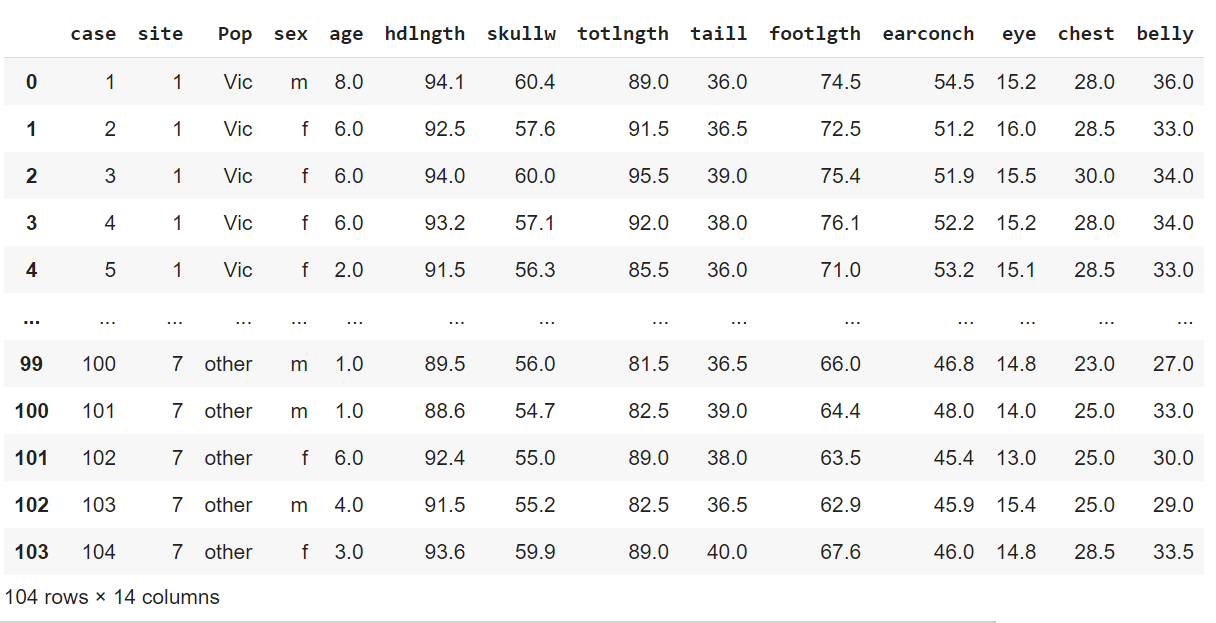
plt.title("Boxplot distribution of the age of possums");



**Regression**

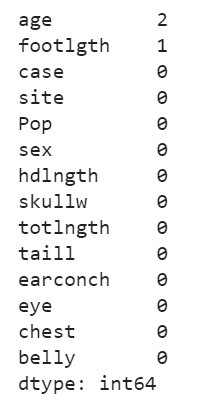
DATA = pd.read\_csv('../datasets/possum.csv')

DATA

****

**Null Values**

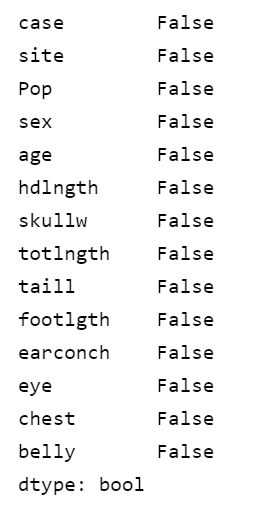
DATA.isna().sum().sort\_values(ascending=False)

****

DATA['footlgth'] = DATA['footlgth'].fillna(DATA['footlgth'].mean())

DATA = DATA.dropna()

DATA.isna().any()

****

**Label Encoding**

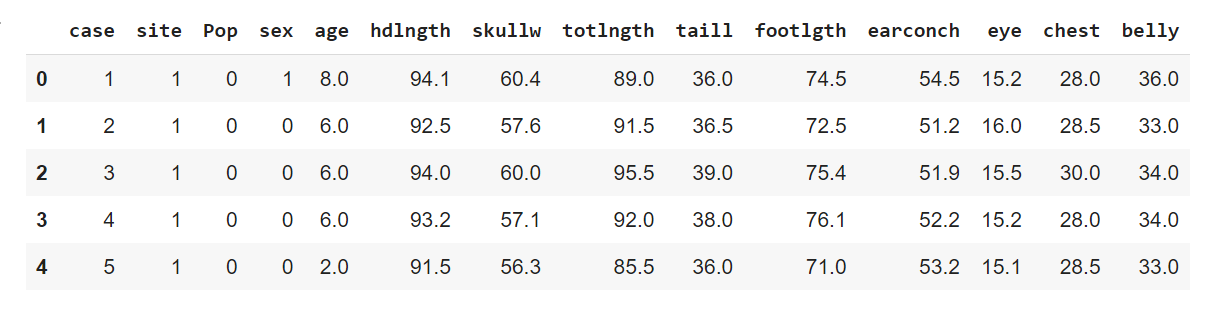
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

DATA['Pop'] = encoder.fit\_transform(DATA['Pop'])

DATA['sex'] = encoder.fit\_transform(DATA['sex'])

DATA.head()



**Scaling**

DATA = DATA.drop('case', axis=1)

X = DATA.drop('age', axis=1)

Y = DATA['age']

from sklearn.preprocessing import StandardScaler

std = StandardScaler()

x = std.fit\_transform(X)

**Feature Engineering**

from feature\_engine.selection import SmartCorrelatedSelection, DropConstantFeatures, DropDuplicateFeatures

from sklearn.pipeline import Pipeline

pip = Pipeline([('constant', DropConstantFeatures(tol=0.9)), ('duplicate', DropDuplicateFeatures()), ('correlated', SmartCorrelatedSelection())])

x = pip.fit\_transform(x)

x.shape

**Model Selection**

from sklearn.metrics import r2\_score, mean\_squared\_error

def evaluate(y\_true, y\_hat, label='test'):

    mse = mean\_squared\_error(y\_true, y\_hat)

    rmse = np.sqrt(mse)

    variance = r2\_score(y\_true, y\_hat)

    print('{} set RMSE:{}, R2:{}'.format(label, rmse, variance))

from sklearn.model\_selection import train\_test\_split

xTrain, xTest, yTrain, yTest = train\_test\_split(x, Y)

1. **Linear Regression**

from sklearn.linear\_model import LinearRegression

model = LinearRegression()

model.fit(xTrain, yTrain)

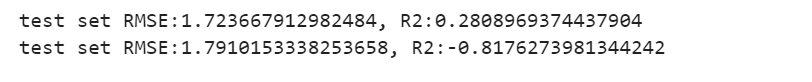
y\_hat\_train = model.predict(xTrain)

evaluate(yTrain, y\_hat\_train)

**Result**

y\_hat\_test = model.predict(xTest)

evaluate(yTest, y\_hat\_test)



1. **SVM**

from sklearn.svm import SVR

svr\_f = SVR()

svr\_f.fit(xTrain, yTrain)

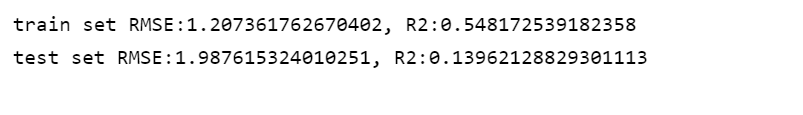
y\_hat\_train = svr\_f.predict(xTrain)

evaluate(yTrain, y\_hat\_train, 'train')

**Result**

y\_hat\_test = svr\_f.predict(xTest)

evaluate(yTest, y\_hat\_test)



**Comparison:**

It can be observed that both the models can predict the output with high accuracy, but the Linear Regression model produces a bit more error than the SVM model. Hence, SVM has better accuracy.