iris with multiple regression check

May 11, 2023

0.0.1 Loading the dataset

[1]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

df = pd.DataFrame(load_iris().data, columns=load_iris().feature_names)
df['target']=load_iris().target
df['species'] = df['target'].replace([0,1,2],
[ species for species in load_iris()['target_names']])
df.loc[139:141]
#df[species]=df.target.apply(lambda x: iris.target_names[x])
```

[1]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
139	6.9	3.1	5.4	2.1	2	virginica
140	6.7	3.1	5.6	2.4	2	virginica
141	6.9	3.1	5.1	2.3	2	virginica

[58]:
```
iris=load_iris()
dir(iris)
```

[58]:
```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

[59]:
```
iris.data_module
```

[59]:
```
'sklearn.datasets.data'
```

[60]:
```
iris.target_names
```

[60]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[61]:
```
# to display stats about data
df.describe()
```

[61]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

[62]:
```
# to basic info about datatype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int32
 5   species            150 non-null    object
dtypes: float64(4), int32(1), object(1)
memory usage: 6.6+ KB
```

[6]:
```
# to display no. of samples on each class
df['species'].value_counts()
```

[6]:
```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

0.0.2 Preprocessing the dataset

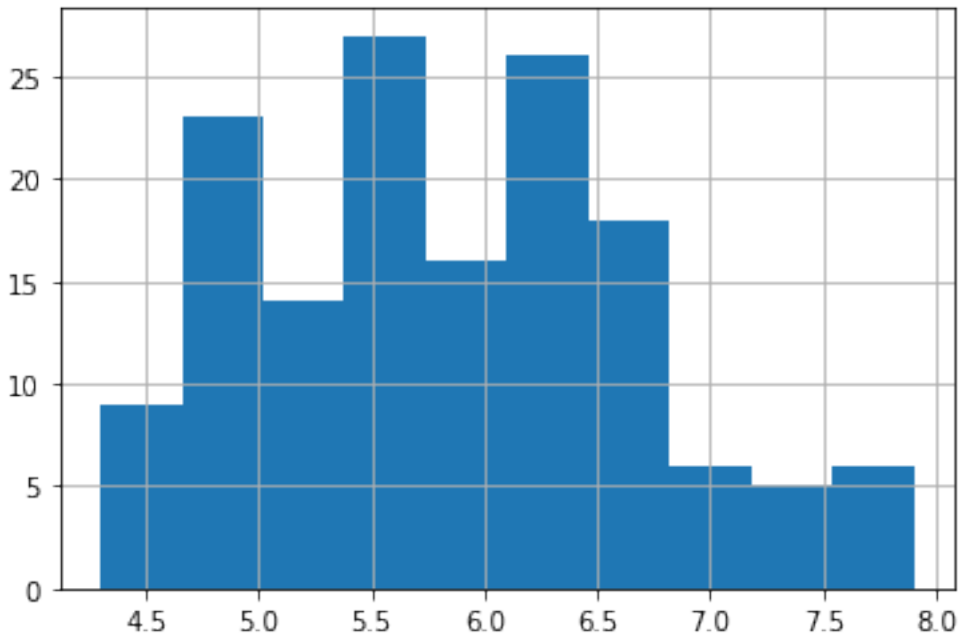
[64]:
```
# check for null values
df.isnull().sum()
```

[64]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
species              0
dtype: int64
```

0.0.3 histograms

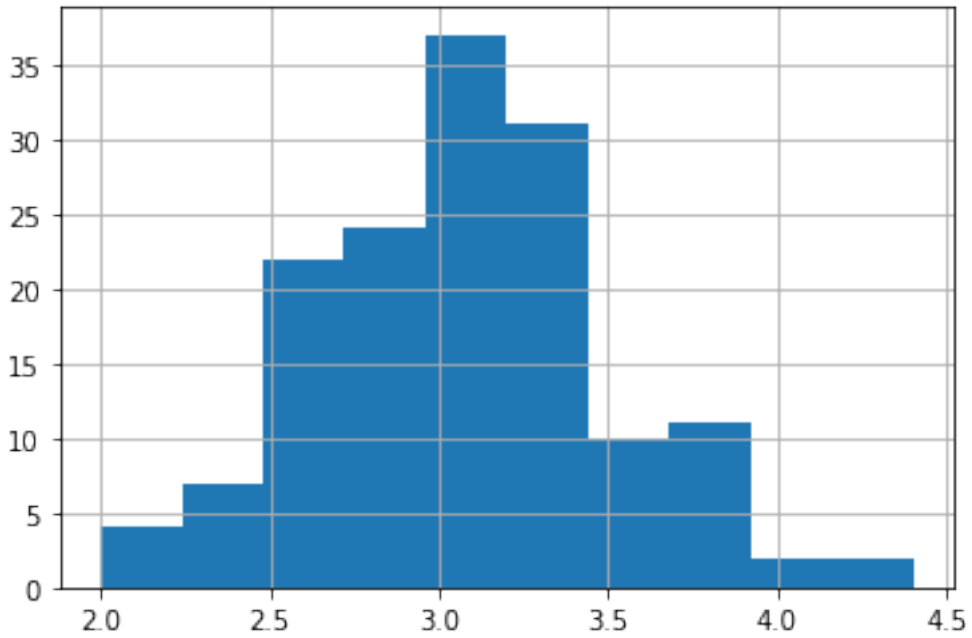
[65]:
```
df['sepal length (cm)'].hist()
```

[65]:
```
<AxesSubplot:>
```



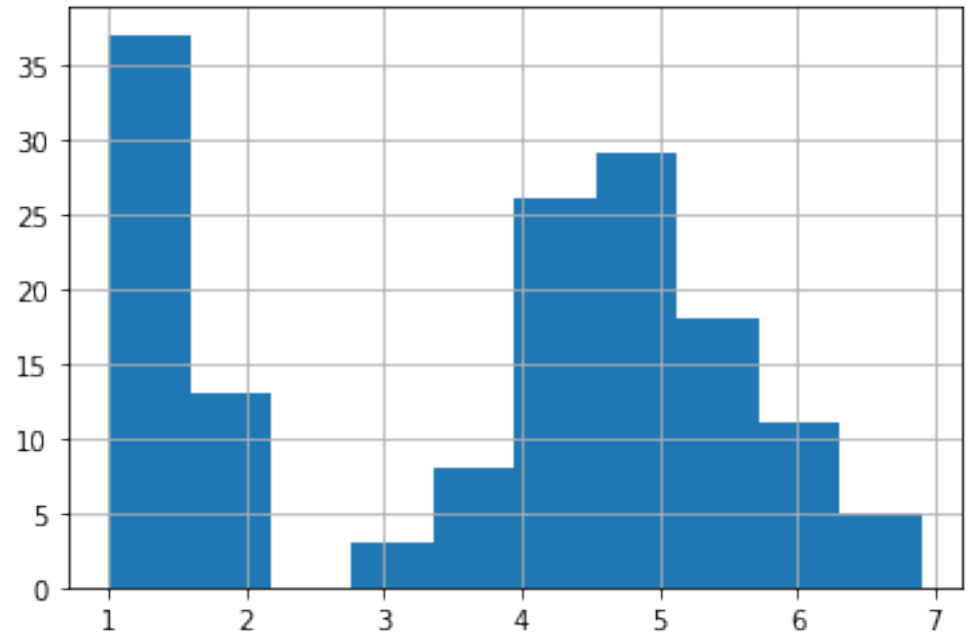
[66]:
```
df['sepal width (cm)'].hist()
```

[66]: <AxesSubplot:>



[67]:
```
df['petal length (cm)'].hist()
```

[67]: <AxesSubplot:>



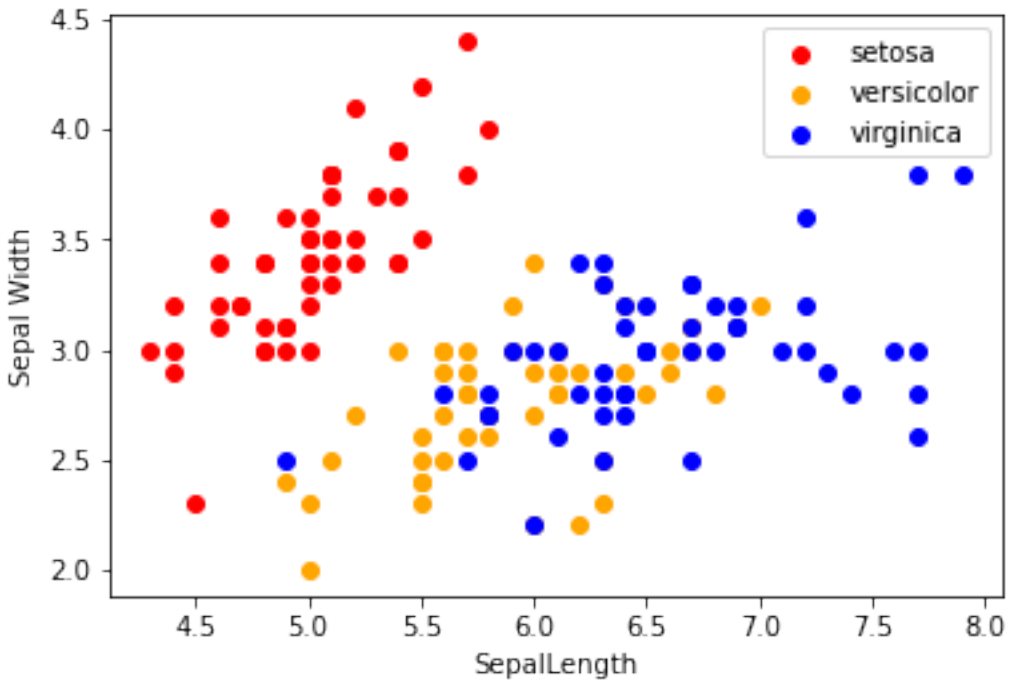
[68]:
```
# scatterplot
colors=['red','orange','blue']
species=['setosa','versicolor' ,'virginica']
```

[69]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

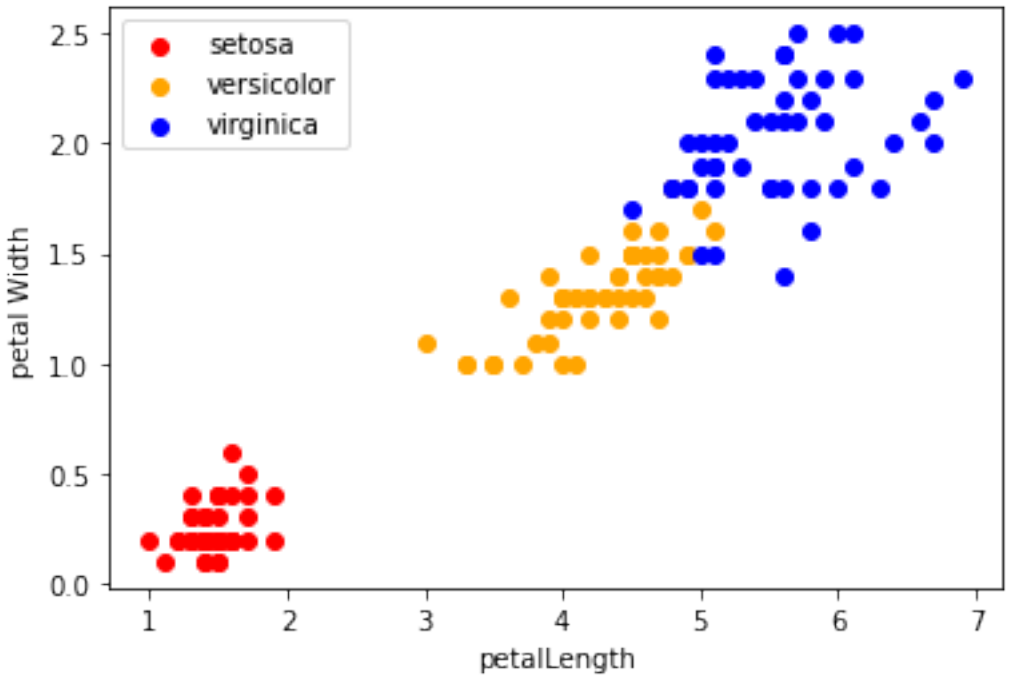
[71]:
```
import matplotlib.pyplot as plt
from importlib import reload
plt=reload(plt)
```

[73]:
```
for i in range(3):
    x=df[df['species']==species[i]]
    plt.scatter(x['sepal length (cm)'],x['sepal width␣
 ↪(cm)'],c=colors[i],label=species[i])
    plt.xlabel("SepalLength")
plt.ylabel('Sepal Width')
plt.legend()
```

[73]:
```
<matplotlib.legend.Legend at 0x2685a4695b0>
```

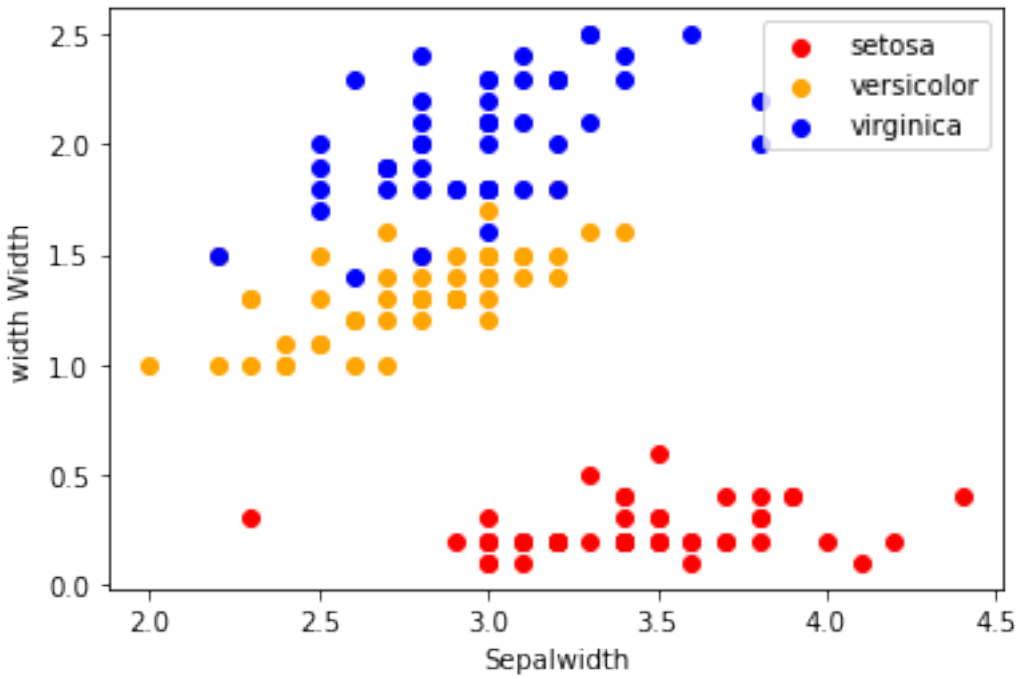


[74]:
```
for i in range(3):
    x=df[df['species']==species[i]]
    plt.scatter(x['petal length (cm)'],x['petal width␣
 ↪(cm)'],c=colors[i],label=species[i])
    plt.xlabel("petalLength")
    plt.ylabel('petal Width')
    plt.legend()
```



[76]:
```
for i in range(3):
    x=df[df['species']==species[i]]
    plt.scatter(x['sepal width (cm)'],x['petal width␣
 ↪(cm)'],c=colors[i],label=species[i])
    plt.xlabel("Sepalwidth")
plt.ylabel('width Width')
plt.legend()
```

[76]:
```
<matplotlib.legend.Legend at 0x2685a5d91c0>
```



0.0.4 Coorelation Matrix

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. The value is in the range of -1 to 1. If two varibles have high correlation, we can neglect one variable from those two.

[77]:
```
df.corr()
```

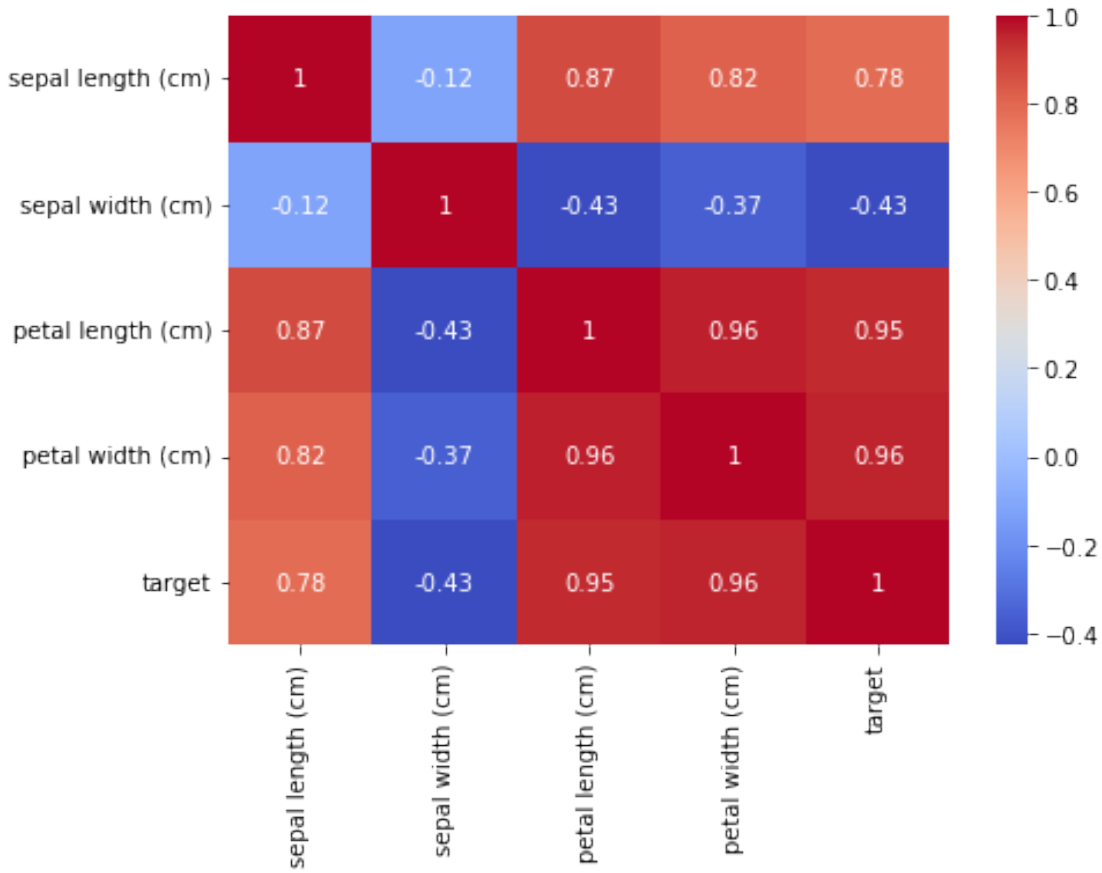
[77]:
```
                   sepal length (cm)  sepal width (cm)  petal length (cm)  \
sepal length (cm)           1.000000         -0.117570           0.871754
sepal width (cm)           -0.117570          1.000000          -0.428440
petal length (cm)           0.871754         -0.428440           1.000000
petal width (cm)            0.817941         -0.366126           0.962865
target                      0.782561         -0.426658           0.949035

                   petal width (cm)    target
sepal length (cm)          0.817941  0.782561
sepal width (cm)          -0.366126 -0.426658
petal length (cm)          0.962865  0.949035
petal width (cm)           1.000000  0.956547
target                     0.956547  1.000000
```

[83]:
```
corr=df.corr()
fig, ax=plt.subplots(figsize=(7,5))
import seaborn as sns
```

```
sns.heatmap(corr,annot=True,ax=ax, cmap="coolwarm")
```

[83]: <AxesSubplot:>



0.0.5 Label Encoder

In machine learning, we usually deal with datasets which contains multiple labels in one or more than one columns. These labels can be in the form of words or numbers. Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form

[84]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
```

[86]:
```
df['species']=le.fit_transform(df.species)
df.head()
```

[86]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	\
0	5.1	3.5	1.4	0.2	
1	4.9	3.0	1.4	0.2	
2	4.7	3.2	1.3	0.2	

```
3                4.6               3.1               1.5               0.2
4                5.0               3.6               1.4               0.2

   target  species
0       0        0
1       0        0
2       0        0
3       0        0
4       0        0
```

0.0.6 Model Training

[113]:
```
from sklearn.model_selection import train_test_split
x=df.drop(columns=['target','species'])
y=df['species']
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.3)
len(x_train)
```

[113]: 105

[114]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
```

[115]:
```
model.fit(x_train,y_train)
```

```
C:\Users\Rakesh\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[115]: LogisticRegression()

[116]:
```
model.score(x_test,y_test)*100
```

[116]: 100.0

[117]:
```
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier()
```

[118]:
```
model.fit(x_train,y_train)
```

[118]:
```
KNeighborsClassifier()
```

[119]:
```
model.score(x_test,y_test)*100
```

[119]:
```
100.0
```

[120]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
```

[121]:
```
model.fit(x_train,y_train)
```

[121]:
```
DecisionTreeClassifier()
```

[122]:
```
model.score(x_test,y_test)*100
```

[122]:
```
97.77777777777777
```

[ ]:
```

```