Linear Regression on Iris Dataset

```
import pandas as pd
from sklearn.datasets import load_iris
import seaborn as sns
iris = load_iris()
iris_df = pd.DataFrame(data= iris.data, columns= iris.feature_names)
target_df = pd.DataFrame(data= iris.target, columns= ['species'])

def converter(specie):
    if specie == 0:
        return 'setosa'
    elif specie == 1:
        return 'versicolor'
    else:
        return 'virginica'

target_df['species'] = target_df['species'].apply(converter)

iris_df = pd.concat([iris_df, target_df], axis= 1)
```

+ Code + Text

```
iris_df.describe()
```

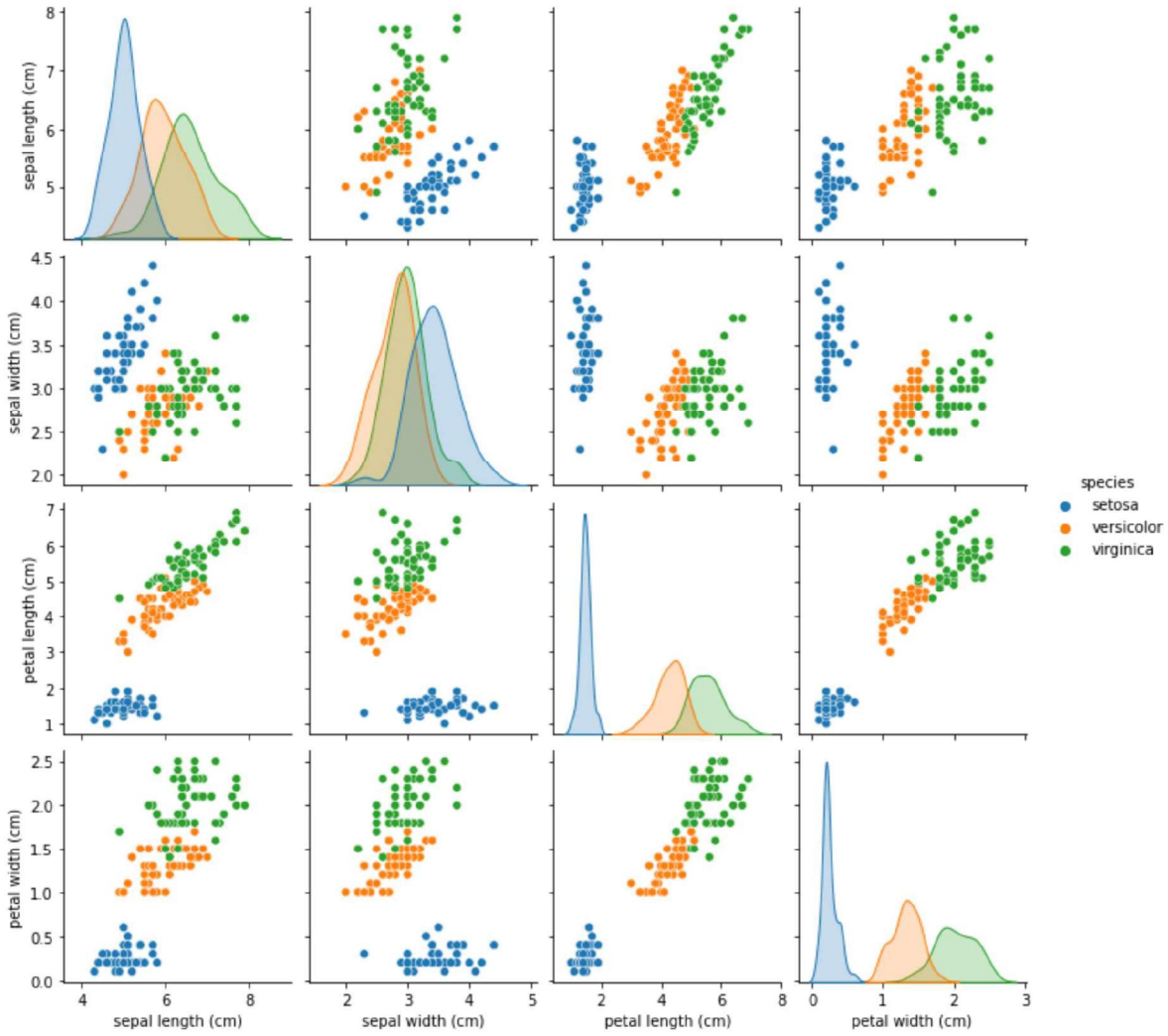
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000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

```
iris_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 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   species            150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
sns.pairplot(iris_df, hue= 'species')
```

```
<seaborn.axisgrid.PairGrid at 0x7f425f0619d0>
```




```
iris_df.drop('species', axis= 1, inplace= True)
target_df = pd.DataFrame(columns= ['species'], data= iris.target)
iris_df = pd.concat([iris_df, target_df], axis= 1)
X= iris_df.drop(labels= 'sepal length (cm)', axis= 1)
y= iris_df['sepal length (cm)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.33, random_state= 1

lr = LinearRegression()

lr.fit(X_train, y_train)
```

```
lr.predict(X_test)
pred = lr.predict(X_test)

print('Mean Absolute Error:', mean_absolute_error(y_test, pred))
print('Mean Squared Error:', mean_squared_error(y_test, pred))
print('Mean Root Squared Error:', np.sqrt(mean_squared_error(y_test, pred)))
```

```
Mean Absolute Error: 0.2595570975563036
Mean Squared Error: 0.10174529564238954
Mean Root Squared Error: 0.3189753840696638
```

```
iris_df.loc[6]
```

```
sepal length (cm)    4.6
sepal width (cm)     3.4
petal length (cm)    1.4
petal width (cm)     0.3
species              0.0
Name: 6, dtype: float64
```

```
d = {'sepal length (cm)' : [4.6],
    'sepal width (cm)' : [3.4],
    'petal length (cm)' : [1.4],
    'petal width (cm)' : [0.3],
    'species' : 0}
test_df = pd.DataFrame(data= d)
test_df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	4.6	3.4	1.4	0.3	0

```
pred = lr.predict(X_test)
print('Predicted Sepal Length (cm):', pred[0])
print('Actual Sepal Length (cm):', 4.6)
```

```
Predicted Sepal Length (cm): 5.461145872156033
Actual Sepal Length (cm): 4.6
```

Logistic Regression on Iris Dataset

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
dataset = pd.read_csv("/content/drive/MyDrive/Other Things/Deep Learning/iris.csv")
dataset.describe()
```

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000

```
X = dataset.iloc[:, [0,1,2, 3]].values
y = dataset.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =


from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)


from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0, solver='lbfgs', multi_class='auto')
classifier.fit(X_train, y_train)

     LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                        intercept_scaling=1, l1_ratio=None, max_iter=100,
                        multi_class='auto', n_jobs=None, penalty='l2',
                        random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                        warm_start=False)


y_pred = classifier.predict(X_test)
probs_y=classifier.predict_proba(X_test)
probs_y = np.round(probs_y, 2)

res = "{:<10} | {:<10} | {:<10} | {:<13} | {:<5}".format("y_test", "y_pred", "Setosa(%)",
res += "-"*65+"\n"
res += "\n".join("{:<10} | {:<10} | {:<10} | {:<13} | {:<10}".format(x, y, a, b, c) for x,
res += "\n"+"-"*65+"\n"
print(res)

     y_test     | y_pred     | Setosa(%)  | versicolor(%) | virginica(%)
     -----------------------------------------------------------------
     Virginica  | Virginica  | 0.0        | 0.03          | 0.97
     Versicolor | Versicolor | 0.01       | 0.95          | 0.04
     Setosa     | Setosa     | 1.0        | 0.0           | 0.0
     Virginica  | Virginica  | 0.0        | 0.08          | 0.92
     Setosa     | Setosa     | 0.98       | 0.02          | 0.0
     Virginica  | Virginica  | 0.0        | 0.01          | 0.99
     Setosa     | Setosa     | 0.98       | 0.02          | 0.0
     Versicolor | Versicolor | 0.01       | 0.71          | 0.28
     Versicolor | Versicolor | 0.0        | 0.73          | 0.27
```

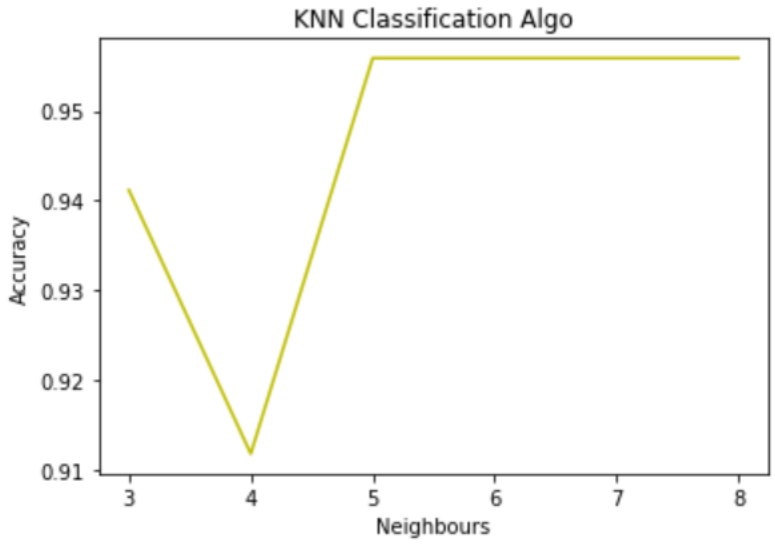
```
Versicolor | Versicolor | 0.02       | 0.89          | 0.08
Virginica  | Virginica  | 0.0        | 0.44          | 0.56
Versicolor | Versicolor | 0.02       | 0.76          | 0.22
Versicolor | Versicolor | 0.01       | 0.85          | 0.13
Versicolor | Versicolor | 0.0        | 0.69          | 0.3
Versicolor | Versicolor | 0.01       | 0.75          | 0.24
Setosa     | Setosa     | 0.99       | 0.01          | 0.0
Versicolor | Versicolor | 0.02       | 0.72          | 0.26
Versicolor | Versicolor | 0.03       | 0.86          | 0.11
Setosa     | Setosa     | 0.94       | 0.06          | 0.0
Setosa     | Setosa     | 0.99       | 0.01          | 0.0
Virginica  | Virginica  | 0.0        | 0.17          | 0.83
Versicolor | Versicolor | 0.04       | 0.71          | 0.25
Setosa     | Setosa     | 0.98       | 0.02          | 0.0
Setosa     | Setosa     | 0.96       | 0.04          | 0.0
Virginica  | Virginica  | 0.0        | 0.35          | 0.65
Setosa     | Setosa     | 1.0        | 0.0           | 0.0
Setosa     | Setosa     | 0.99       | 0.01          | 0.0
Versicolor | Versicolor | 0.02       | 0.87          | 0.11
Versicolor | Versicolor | 0.09       | 0.9           | 0.02
Setosa     | Setosa     | 0.97       | 0.03          | 0.0
Virginica  | Virginica  | 0.0        | 0.21          | 0.79
Versicolor | Versicolor | 0.06       | 0.69          | 0.25
Setosa     | Setosa     | 0.98       | 0.02          | 0.0
Virginica  | Virginica  | 0.0        | 0.35          | 0.65
Virginica  | Virginica  | 0.0        | 0.04          | 0.96
Versicolor | Versicolor | 0.07       | 0.81          | 0.11
Setosa     | Setosa     | 0.97       | 0.03          | 0.0
Versicolor | Virginica  | 0.0        | 0.42          | 0.58
-----------------------------------------------------------------
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
```

```
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
import warnings
warnings.simplefilter(action="ignore")



X,y=datasets.load_iris(return_X_y=True)

print(X)
print(X.shape)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.45, random_state =0)



# KNN

from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
a,tempList1,tempList2=[],[],[]
b=[]
for i in range(3,9):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    y_pred=knn.predict(X_test)
    a.append(metrics.accuracy_score(y_test, y_pred))
    b.append(i)
    # Calculate the accuracy of the model
```

```
    #print(knn.score(X_test, y_test))


plt.plot(b,a,color='y')
plt.xlabel("Neighbours")
plt.ylabel("Accuracy")
plt.title("KNN Classification Algo")
plt.show()
```



System Related Issues in IRIS Dataset using K Means and Linear and Logistic Regression:

- ➔ Sometimes not satisfying the system requirements might able to diminish the performance