Breast Cancer Prediction Using Python

In [34]:

```
# importing libraries
import numpy
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

In [35]:

```
# reading data from the file
df=pd.read_csv("data.csv")
```

In [36]:

```
df.head()
```

Out[36]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 33 columns

In [37]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [38]:
```
# return all the columns with null values count
df.isna().sum()
```

Out[38]:
```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

In [39]:
```
# return the size of dataset
df.shape
```

Out[39]:
```
(569, 33)
```

In [40]:
```
# remove the column
df=df.dropna(axis=1)
```

In [41]:
```
# shape of dataset after removing the null column
df.shape
```

Out[41]:
```
(569, 32)
```

In [46]:

```
# describe the dataset
df.describe()
```

Out[46]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.

8 rows × 31 columns

In [47]:

```
# Get the count of malignant<M> and Benign<B> cells
df['diagnosis'].value_counts()
```

Out[47]:

```
B    357
M    212
Name: diagnosis, dtype: int64
```

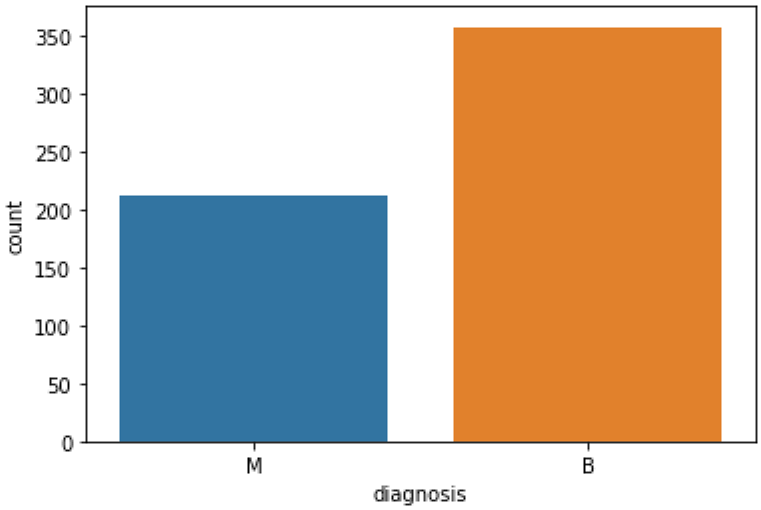
In [48]:

```
sns.countplot(df['diagnosis'],label="count")
```

```
/Applications/anaconda3/lib/python3.9/site-packages/seaborn/_decorators.p
y:36: FutureWarning: Pass the following variable as a keyword arg: x. Fro
m version 0.12, the only valid positional argument will be `data`, and pa
ssing other arguments without an explicit keyword will result in an error
or misinterpretation.
  warnings.warn(
```

Out[48]:

```
<AxesSubplot:xlabel='diagnosis', ylabel='count'>
```



In [49]:

```
# Label encoding(convert the value of M and B into 1 and 0)
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:,1]=labelencoder_Y.fit_transform(df.iloc[:,1].values)
```

In [50]:

```
df.head()
```

Out[50]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
0	842302	1	17.99	10.38	122.80	1001.0	
1	842517	1	20.57	17.77	132.90	1326.0	
2	84300903	1	19.69	21.25	130.00	1203.0	
3	84348301	1	11.42	20.38	77.58	386.1	
4	84358402	1	20.29	14.34	135.10	1297.0	

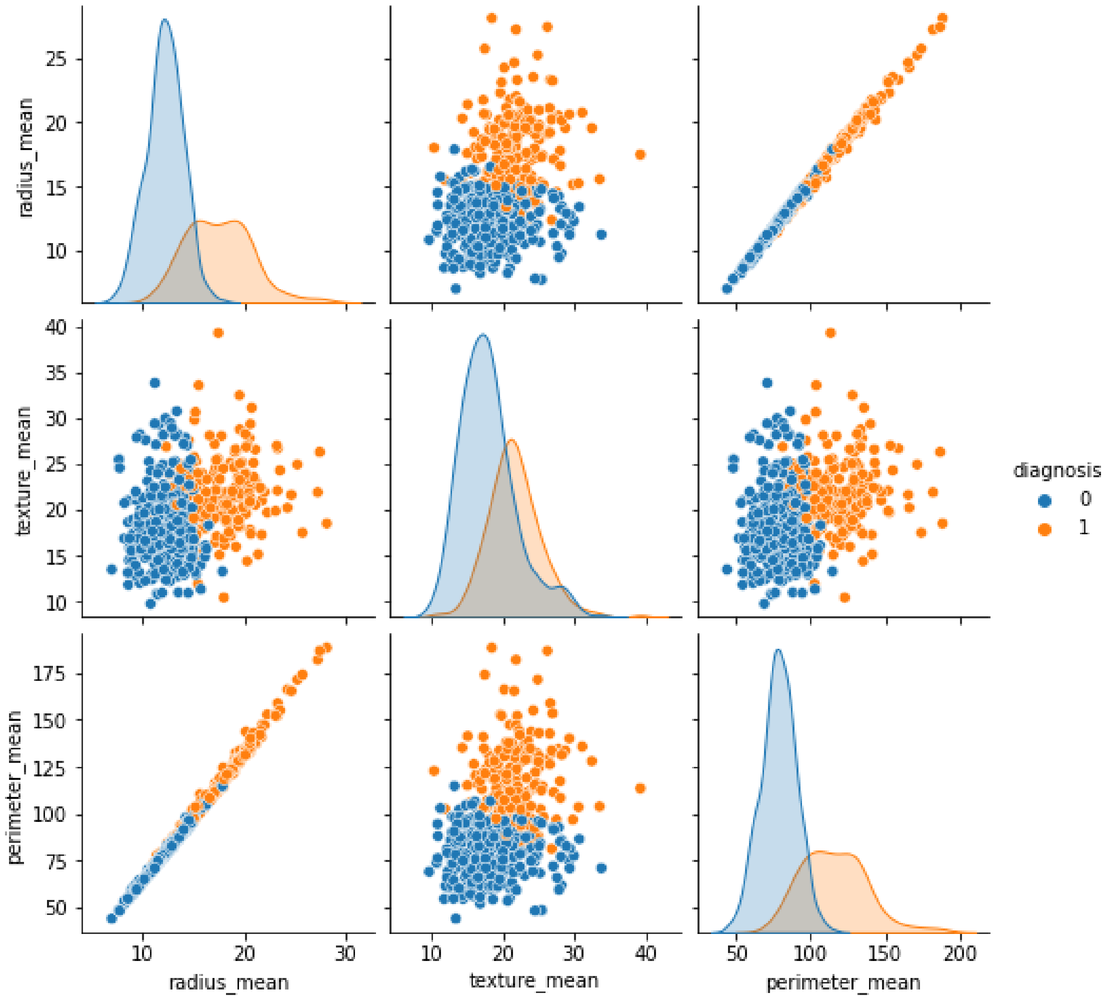
5 rows × 32 columns

In [16]:

```
sns.pairplot(df.iloc[:,1:5],hue="diagnosis")
```

Out[16]:

```
<seaborn.axisgrid.PairGrid at 0x7faeb2f65af0>
```



In [17]:

```
# get the correlation
df.iloc[:,1:32].corr()
```

Out[17]:

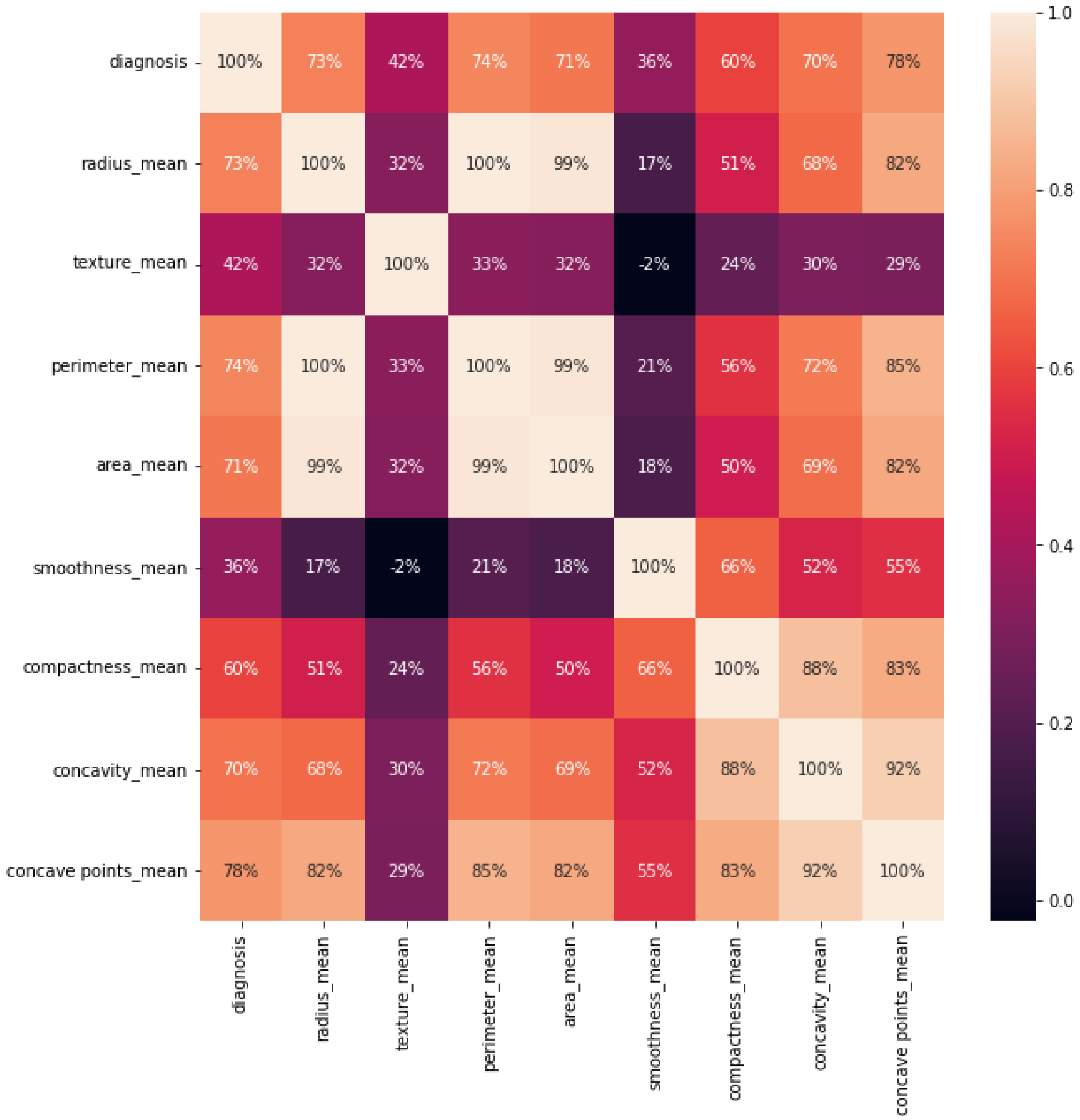
ean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst	perimeter_worst	area_wo
360	0.776614	0.330499	...	0.776454	0.456903	0.782914	0.733
764	0.822529	0.147741	...	0.969539	0.297008	0.965137	0.941
418	0.293464	0.071401	...	0.352573	0.912045	0.358040	0.343
136	0.850977	0.183027	...	0.969476	0.303038	0.970387	0.941
983	0.823269	0.151293	...	0.962746	0.287489	0.959120	0.959
984	0.553695	0.557775	...	0.213120	0.036072	0.238853	0.206
121	0.831135	0.602641	...	0.535315	0.248133	0.590210	0.509
000	0.921391	0.500667	...	0.688236	0.299879	0.729565	0.675
391	1.000000	0.462497	...	0.830318	0.292752	0.855923	0.809
667	0.462497	1.000000	...	0.185728	0.090651	0.219169	0.177
783	0.166917	0.479921	...	-0.253691	-0.051269	-0.205151	-0.231
925	0.698050	0.303379	...	0.715065	0.194799	0.719684	0.751
218	0.021480	0.128053	...	-0.111690	0.409003	-0.102242	-0.083
391	0.710650	0.313893	...	0.697201	0.200371	0.721031	0.730
427	0.690299	0.223970	...	0.757373	0.196497	0.761213	0.811
564	0.027653	0.187321	...	-0.230691	-0.074743	-0.217304	-0.182
279	0.490424	0.421659	...	0.204607	0.143003	0.260516	0.199
270	0.439167	0.342627	...	0.186904	0.100241	0.226680	0.188
260	0.615634	0.393298	...	0.358127	0.086741	0.394999	0.342
009	0.095351	0.449137	...	-0.128121	-0.077473	-0.103753	-0.110
301	0.257584	0.331786	...	-0.037488	-0.003195	-0.001000	-0.022
236	0.830318	0.185728	...	1.000000	0.359921	0.993708	0.984
879	0.292752	0.090651	...	0.359921	1.000000	0.365098	0.345
565	0.855923	0.219169	...	0.993708	0.365098	1.000000	0.977
987	0.809630	0.177193	...	0.984015	0.345842	0.977578	1.000
822	0.452753	0.426675	...	0.216574	0.225429	0.236775	0.209
968	0.667454	0.473200	...	0.475820	0.360832	0.529408	0.438
103	0.752399	0.433721	...	0.573975	0.368366	0.618344	0.543
323	0.910155	0.430297	...	0.787424	0.359755	0.816322	0.747
464	0.375744	0.699826	...	0.243529	0.233027	0.269493	0.209
930	0.368661	0.438413	...	0.093492	0.219122	0.138957	0.079

In [51]:

```
# visualize the correlation
plt.figure(figsize=(10,10))
sns.heatmap(df.iloc[:,1:10].corr(),annot=True,fmt=".0%")
```

Out[51]:

```
<AxesSubplot:>
```



In [52]:

```
# split the dataset into dependent(X) and Independent(Y) datasets
X=df.iloc[:,2:31].values
Y=df.iloc[:,1].values
```

In [53]:

```
# spliting the data into trainning and test dateset
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.20,random_s
```

In [54]:

```
# feature scaling
from sklearn.preprocessing import StandardScaler
X_train=StandardScaler().fit_transform(X_train)
X_test=StandardScaler().fit_transform(X_test)
```

In [55]:

```
# models/ Algorithms

def models(X_train,Y_train):
        #logistic regression
        from sklearn.linear_model import LogisticRegression
        log=LogisticRegression(random_state=0)
        log.fit(X_train,Y_train)


        #Decision Tree
        from sklearn.tree import DecisionTreeClassifier
        tree=DecisionTreeClassifier(random_state=0,criterion="entropy")
        tree.fit(X_train,Y_train)

        #Random Forest
        from sklearn.ensemble import RandomForestClassifier
        forest=RandomForestClassifier(random_state=0,criterion="entropy",n
        forest.fit(X_train,Y_train)

        print('[0]logistic regression accuracy:',log.score(X_train,Y_train
        print('[1]Decision tree accuracy:',tree.score(X_train,Y_train))
        print('[2]Random forest accuracy:',forest.score(X_train,Y_train))

        return log,tree,forest
```

In [56]:

```
model=models(X_train,Y_train)
```

```
[0]logistic regression accuracy: 0.9912087912087912
[1]Decision tree accuracy: 1.0
[2]Random forest accuracy: 0.9978021978021978
```

In [57]:

```
# testing the models/result

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

for i in range(len(model)):
    print("Model",i)
    print(classification_report(Y_test,model[i].predict(X_test)))
    print('Accuracy : ',accuracy_score(Y_test,model[i].predict(X_test)))
```

```
Model 0
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        67
           1       0.98      0.94      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy :  0.9649122807017544
Model 1
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        67
           1       0.93      0.91      0.92        47

    accuracy                           0.94       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.94      0.94      0.94       114

Accuracy :  0.9385964912280702
Model 2
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        67
           1       1.00      0.94      0.97        47

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Accuracy :  0.9736842105263158
```

In [58]:

```
# prediction of random-forest
pred=model[2].predict(X_test)
print('Predicted values:')
print(pred)
print('Actual values:')
print(Y_test)
```

```
Predicted values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1
0
 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0
0
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1
0
 1 1 0]
Actual values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1
0
 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0
1
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 1
0
 1 1 0]
```

In [67]:

```
plt.scatter()
plt.xlabel("actual ")
plt.ylabel("predicted")
plt.title("Actual VS Predicted")
plt.show()
```

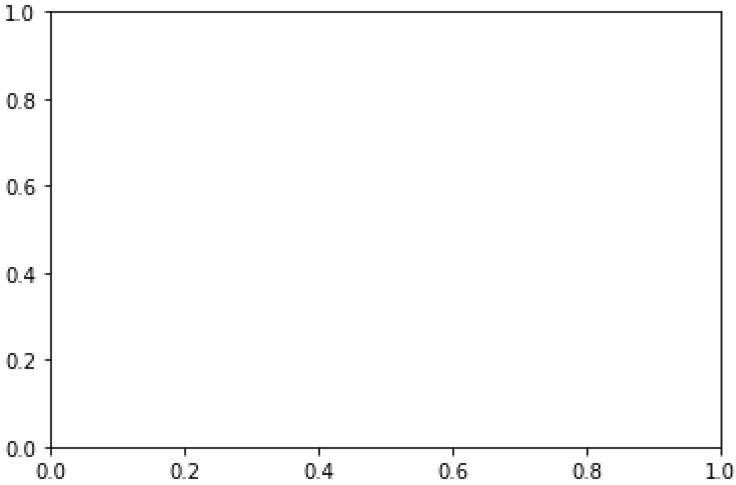
```
---------------------------------------------------------------------------
--
ValueError                                Traceback (most recent call las
t)
/var/folders/_1/z_zzbl013nbf5zyqc3lvq4xh0000gn/T/ipykernel_1184/181843946
2.py in <module>
----> 1 plt.scatter(X,Y)
      2 plt.xlabel("actual ")
      3 plt.ylabel("predicted")
      4 plt.title("Actual VS Predicted")
      5 plt.show()

/Applications/anaconda3/lib/python3.9/site-packages/matplotlib/pyplot.py
in scatter(x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths,
edgecolors, plotnonfinite, data, **kwargs)
   3066         vmin=None, vmax=None, alpha=None, linewidths=None, *,
   3067         edgecolors=None, plotnonfinite=False, data=None, **kwarg
s):
-> 3068     __ret = gca().scatter(
   3069         x, y, s=s, c=c, marker=marker, cmap=cmap, norm=norm,
   3070         vmin=vmin, vmax=vmax, alpha=alpha, linewidths=linewidths,

/Applications/anaconda3/lib/python3.9/site-packages/matplotlib/__init__.p
y in inner(ax, data, *args, **kwargs)
   1359     def inner(ax, *args, data=None, **kwargs):
   1360         if data is None:
-> 1361             return func(ax, *map(sanitize_sequence, args), **kwar
gs)
   1362 
   1363         bound = new_sig.bind(ax, *args, **kwargs)

/Applications/anaconda3/lib/python3.9/site-packages/matplotlib/axes/_axe
s.py in scatter(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha,
linewidths, edgecolors, plotnonfinite, **kwargs)
   4496         y = np.ma.ravel(y)
   4497         if x.size != y.size:
-> 4498             raise ValueError("x and y must be the same size")
   4499 
   4500         if s is None:

ValueError: x and y must be the same size
```



In [ ]:

In [ ]: