Breast Cancer Prediction Using K-NN

Using the Breast Cancer Wisconsin (Diagnostic) Database to create a classifier that can help diagnose patients.

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
```

The object returned by `load_breast_cancer()` is a scikit-learn Bunch object, which is similar to a dictionary.

In [4]:
```
cancer.keys()
```

Out[4]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

Question 0 (Example)

How many features does the breast cancer dataset have?

This function should return an integer.

In [5]:
```
len(cancer['feature_names'])
```

Out[5]:
```
30
```

Convert the sklearn.dataset `cancer` to a DataFrame.

This function should return a `(569, 31)` DataFrame with

columns =

```
['mean radius', 'mean texture', 'mean perimeter', 'mean area',
 'mean smoothness', 'mean compactness', 'mean concavity',
 'mean concave points', 'mean symmetry', 'mean fractal dimension',
 'radius error', 'texture error', 'perimeter error', 'area error',
 'smoothness error', 'compactness error', 'concavity error',
 'concave points error', 'symmetry error', 'fractal dimension error',
 'worst radius', 'worst texture', 'worst perimeter', 'worst area',
 'worst smoothness', 'worst compactness', 'worst concavity',
 'worst concave points', 'worst symmetry', 'worst fractal dimension',
 'target']
```

and index =

```
RangeIndex(start=0, stop=569, step=1)
```

In [11]:
```
columns = ['mean radius', 'mean texture', 'mean perimeter', 'mean area',
 'mean smoothness', 'mean compactness', 'mean concavity',
 'mean concave points', 'mean symmetry', 'mean fractal dimension',
 'radius error', 'texture error', 'perimeter error', 'area error',
 'smoothness error', 'compactness error', 'concavity error',
 'concave points error', 'symmetry error', 'fractal dimension error',
 'worst radius', 'worst texture', 'worst perimeter', 'worst area',
 'worst smoothness', 'worst compactness', 'worst concavity',
 'worst concave points', 'worst symmetry', 'worst fractal dimension',
 'target']

index = range(0, 569, 1)
df = pd.DataFrame(data=cancer['data'], index=index, columns = columns)
df['target'] = cancer['target']
df
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	worst
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	152.20	1575.00	1575.0	0.1
...																
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	166.10	2027.00	2226.0	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.00	1676.0	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	126.70	1124.00	891.0	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	184.60	1821.00	1645.0	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	59.16	268.60	2989.0	0.1

569 rows × 31 columns

class distribution

how many instances of `malignant` (encoded 0) and how many `benign` (encoded 1)

In [14]:
```
malignant_count = len(df['target'] == 0])
benign_count = len(df['target'] == 1])
index = ['malignant', 'benign']
class_distribution = pd.Series(data=[malignant_count, benign_count], index=index)

class_distribution
```

Out[14]:
```
malignant    212
benign       357
dtype: int64
```

Split the DataFrame into `X` (the data) and `y` (the labels).

- *X, a pandas DataFrame. has shape* `(569, 30)`
- *y, a pandas Series, has shape* `(569,)`.

In [21]:
```
X = df.iloc[:,:30]
y = df.target

X
```

Out[21]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	w
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.00	0.1
...																
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.00	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.00	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.00	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.00	0.1
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.60	2989.0

569 rows × 30 columns

In [18]:
```
y
```

Out[18]:
```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: target, Length: 569, dtype: int32
```

train_test_split

split `X` and `y` into training and test sets `(X_train, X_test, y_train, and y_test)`.

Set the random number generator state to 0 using `random_state=0` This function should return a tuple of length 4: `(X_train, X_test, y_train, y_test)`, where*

- `X_train` *has shape* `(426, 30)`
- `X_test` *has shape* `(143, 30)`
- `y_train` *has shape* `(426,)`
- `y_test` *has shape* `(143,)`

In [24]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
X_train
```

Out[24]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	w
293	11.850	17.46	75.54	432.7	0.08372	0.05642	0.026880	0.022600	0.1875	0.05715	...	13.060	25.75	84.35	517.8	0.1
332	11.220	19.86	71.94	387.3	0.10540	0.06779	0.005006	0.007593	0.1940	0.06028	...	11.980	25.78	76.91	436.1	0.1
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910	0.1752	0.05533	...	23.690	38.25	155.00	1731.00	0.1
279	13.690	17.84	86.24	572.3	0.07948	0.04052	0.019970	0.012380	0.1573	0.05520	...	15.500	26.10	98.91	739.1	0.1
489	16.690	20.20	107.10	857.6	0.07497	0.07112	0.036490	0.023070	0.1846	0.05325	...	19.180	26.56	127.30	1084.0	0.1
...																
277	16.810	19.98	120.90	1102.0	0.08023	0.05884	0.080200	0.036430	0.1550	0.04996	...	19.960	24.30	129.00	1236.0	0.1
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.085430	0.2030	0.08243	...	15.090	40.68	97.65	711.4	0.1
359	9.436	18.32	59.82	278.6	0.10090	0.05956	0.027100	0.014060	0.1506	0.06959	...	12.020	25.02	75.79	439.6	0.1
192	9.720	18.22	60.73	288.1	0.06950	0.02344	0.000000	0.000000	0.1653	0.06447	...	9.968	20.83	63.65	308.3	0.1
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050	0.1388	0.06570	...	12.480	37.16	82.28	474.2	0.1

426 rows × 30 columns

In [25]:
```
X_test
```

Out[25]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	w
512	13.40	20.52	88.64	556.7	0.11000	0.14990	0.14400	0.08172	0.2116	0.07325	...	16.41	29.66	114.10	795.4	0.1
407	13.21	25.25	84.10	537.9	0.08781	0.05020	0.02777	0.02068	0.1619	0.05584	...	14.35	34.23	91.29	632.9	0.1
429	14.26	19.65	97.83	629.9	0.07837	0.05581	0.02087	0.02652	0.1589	0.05586	...	14.91	19.31	96.53	688.9	0.1
298	14.26	18.17	91.22	633.1	0.06576	0.05220	0.02475	0.01374	0.1635	0.05586	...	16.22	25.26	105.80	819.7	0.1
37	13.03	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.02923	0.1467	0.05863	...	13.30	22.81	84.46	545.9	0.0
...																
236	23.21	26.97	153.50	1670.0	0.09509	0.16820	0.19900	0.12370	0.1909	0.06309	...	31.01	34.51	206.00	2944.0	0.1
113	10.51	20.19	68.64	334.2	0.11220	0.13030	0.06476	0.03068	0.1922	0.07782	...	11.16	22.75	72.62	374.4	0.1
527	12.34	12.27	78.94	468.5	0.09003	0.06307	0.02958	0.02647	0.1689	0.05808	...	13.61	19.27	87.22	564.9	0.1
76	13.53	10.94	87.91	559.2	0.12910	0.10470	0.06877	0.06556	0.2403	0.06641	...	14.08	12.49	91.36	605.5	0.1
182	19.59	18.15	130.70	1214.0	0.11200	0.16660	0.25080	0.12860	0.2027	0.06082	...	26.73	26.39	174.90	2232.0	0.1

143 rows × 30 columns

In [26]:
```
y_train
```

Out[26]:
```
293    1
332    1
565    0
279    1
489    0
      ..
277    0
9      0
359    1
192    1
559    1
Name: target, Length: 426, dtype: int32
```

In [27]:
```
y_test
```

Out[27]:
```
512    0
407    0
429    1
298    1
37     1
      ..
236    0
113    1
527    1
76     1
182    0
Name: target, Length: 143, dtype: int32
```

Train

Using KNeighborsClassifier, fit a k-nearest neighbors (knn) classifier with `X_train`, `y_train` and using one nearest neighbor (`n_neighbors = 1`).

In [30]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train, y_train)
```

Out[30]:
```
KNeighborsClassifier(n_neighbors=1)
```

predict the class label using the mean value

`cancerdf.mean()[:-1].values.reshape(1, -1)` which gets the mean value for each feature, ignores the target column, and reshapes the data from 1 dimension to 2 (necessary for the precist method of KNeighborsClassifier).

This function returns a numpy array either `array([0.])` or `array([1.])`.

In [33]:
```
means = (df.mean()[:-1].values.reshape(1, -1))
prediction = knn.predict(means)
prediction = np.array(prediction)
prediction
```

Out[33]:
```
array([1])
```

Predict class for test set

In [39]:
```
prediction = knn.predict(X_test)
prediction= np.array(prediction)

prediction
```

Out[39]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0])
```

accuracy

In [41]:
```
accuracy = knn.score(X_test, y_test)
accuracy
```

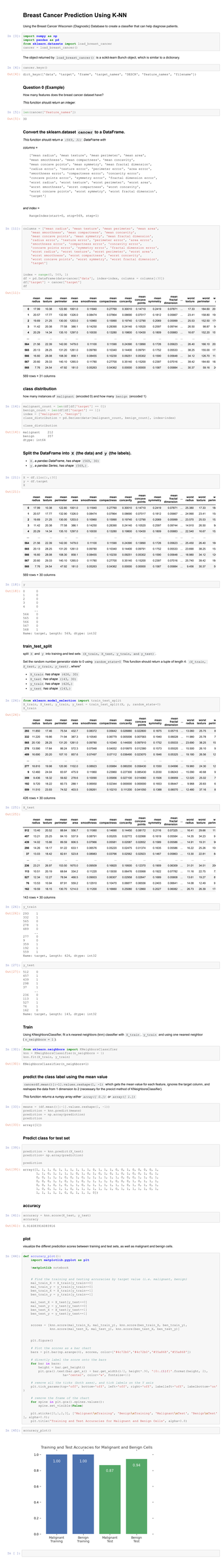
Out[41]:
```
0.9160839160839161
```

plot

visualize the different prediction scores between training and test sets, as well as malignant and benign cells.

In [46]:
```
def accuracy_plot():
    import matplotlib.pyplot as plt

    %matplotlib notebook

    # Find the training and testing accuracies by target value (i.e. malignant, benign)
    mal_train_X = X_train[y_train==0]
    mal_train_y = y_train[y_train==0]
    ben_train_X = X_train[y_train==1]
    ben_train_y = y_train[y_train==1]

    mal_test_X = X_test[y_test==0]
    mal_test_y = y_test[y_test==0]
    ben_test_X = X_test[y_test==1]
    ben_test_y = y_test[y_test==1]


    scores = [knn.score(mal_train_X, mal_train_y), knn.score(ben_train_X, ben_train_y),
              knn.score(mal_test_X, mal_test_y), knn.score(ben_test_X, ben_test_y)]


    plt.figure()

    # Plot the scores as a bar chart
    bars = plt.bar(np.arange(4), scores, color=['#4c72b0','#4c72b0','#55a868','#55a868'])

    # directly label the score onto the bars
    for bar in bars:
        height = bar.get_height()
        plt.gca().text(bar.get_x() + bar.get_width()/2, height*.90, '{0:.1%}'.format(height, 2),
                       ha='center', color='w', fontsize=11)

    # remove all the ticks (both axes), and tick labels on the Y axis
    plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', labelbottom='on')

    # remove the frame of the chart
    for spine in plt.gca().spines.values():
        spine.set_visible(False)

    plt.xticks([0,1,2,3], ['Malignant\nTraining', 'Benign\nTraining', 'Malignant\nTest', 'Benign\nTest'], alpha=0.8);
    plt.title('Training and Test Accuracies for Malignant and Benign Cells', alpha=0.8)
```

In [45]:
```
accuracy_plot()
```



In [ ]:
```

```