*You are currently looking at **version 1.3** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the [Jupyter Notebook FAQ](#) course resource.*

Assignment 1 - Introduction to Machine Learning

For this assignment, you will be using the Breast Cancer Wisconsin (Diagnostic) Database to create a classifier that can help diagnose patients. First, read through the description of the dataset (below).

In [4]:
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
cancer

#print(cancer.DESCR) # Print the data set description
```

Out[4]:
```
{'DESCR': 'Breast Cancer Wisconsin (Diagnostic) Database\n=============================================\n\nNotes\n---
--\nData Set Characteristics:\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 numeric, predictive att
ributes and the class\n\n    :Attribute Information:\n        - radius (mean of distances from center to points on th
e perimeter)\n        - texture (standard deviation of gray-scale values)\n        - perimeter\n        - area\n
- smoothness (local variation in radius lengths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavi
ty (severity of concave portions of the contour)\n        - concave points (number of concave portions of the contou
r)\n        - symmetry \n        - fractal dimension ("coastline approximation" - 1)\n\n        The mean, standard er
ror, and "worst" or largest (mean of the three\n        largest values) of these features were computed for each imag
e,\n        resulting in 30 features.  For instance, field 3 is Mean Radius, field\n        13 is Radius SE, field 23
is Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n                - WDBC-Benign\n\n    :Summary
Statistics:\n\n    ===================================== ====== ======\n                                           Mi
n    Max\n    ===================================== ====== ======\n    radius (mean):                        6.981  2
8.11\n    texture (mean):                       9.71   39.28\n    perimeter (mean):                     43.79  188.5
\n    area (mean):                          143.5  2501.0\n    smoothness (mean):                    0.053  0.163\n
compactness (mean):                   0.019  0.345\n    concavity (mean):                     0.0    0.427\n    conca
ve points (mean):                0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fractal di
mension (mean):            0.05   0.097\n    radius (standard error):              0.112  2.873\n    texture (standa
rd error):             0.36   4.885\n    perimeter (standard error):           0.757  21.98\n    area (standard erro
r):                6.802  542.2\n    smoothness (standard error):          0.002  0.031\n    compactness (standard er
ror):         0.002  0.135\n    concavity (standard error):           0.0    0.396\n    concave points (standard erro
r):      0.0    0.053\n    symmetry (standard error):            0.008  0.079\n    fractal dimension (standard erro
r):  0.001  0.03\n    radius (worst):                       7.93   36.04\n    texture (worst):
12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    area (worst):                         185.2
4254.0\n    smoothness (worst):                   0.071  0.223\n    compactness (worst):                  0.027  1.05
8\n    concavity (worst):                    0.0    1.252\n    concave points (worst):               0.0    0.291\n
symmetry (worst):                     0.156  0.664\n    fractal dimension (worst):            0.055  0.208\n    =====
================================ ====== ======\n\n    :Missing Attribute Values: None\n\n    :Class Distribution: 212
- Malignant, 357 - Benign\n\n    :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian\n\n    :Dono
r: Nick Street\n\n    :Date: November, 1995\n\nThis is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset
s.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image of a fine needle\naspirate (FNA) of a breas
t mass.  They describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating plane described above
was obtained using\nMultisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree\nConstruction Via Linear Programm
ing." Proceedings of the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp. 97-101, 1992], a cl
assification method which uses linear\nprogramming to construct a decision tree.  Relevant features\nwere selected us
ing an exhaustive search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual linear program used t
o obtain the separating plane\nin the 3-dimensional space is that described in:\n[K. P. Bennett and O. L. Mangasaria
n: "Robust Linear\nProgramming Discrimination of Two Linearly Inseparable Sets",\nOptimization Methods and Software
1, 1992, 23-34].\n\nThis database is also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-pro
g/cpo-dataset/machine-learn/WDBC/\n\nReferences\n----------\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuc
lear feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on \n     Electroni
c Imaging: Science and Technology, volume 1905, pages 861-870,\n     San Jose, CA, 1993.\n   - O.L. Mangasarian, W.N.
Street and W.H. Wolberg. Breast cancer diagnosis and \n     prognosis via linear programming. Operations Research, 43
(4), pages 570-577, \n     July-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning
techniques\n     to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994) \n     163-171.\n',
 'data': array([[  1.79900000e+01,   1.03800000e+01,   1.22800000e+02, ...,
          2.65400000e-01,   4.60100000e-01,   1.18900000e-01],
        [  2.05700000e+01,   1.77700000e+01,   1.32900000e+02, ...,
          1.86000000e-01,   2.75000000e-01,   8.90200000e-02],
        [  1.96900000e+01,   2.12500000e+01,   1.30000000e+02, ...,
          2.43000000e-01,   3.61300000e-01,   8.75800000e-02],
        ...,
        [  1.66000000e+01,   2.80800000e+01,   1.08300000e+02, ...,
          1.41800000e-01,   2.21800000e-01,   7.82000000e-02],
        [  2.06000000e+01,   2.93300000e+01,   1.40100000e+02, ...,
          2.65000000e-01,   4.08700000e-01,   1.24000000e-01],
        [  7.76000000e+00,   2.45400000e+01,   4.79200000e+01, ...,
          0.00000000e+00,   2.87100000e-01,   7.03900000e-02]]),
 'feature_names': array(['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'],
       dtype='<U23'),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
        1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1,
        0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
        0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
        0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0,
        1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
        1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
        1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
        0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
        1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
        0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1,
        1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
        1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 0, 1, 0, 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, 0, 0, 0, 0, 1]),
 'target_names': array(['malignant', 'benign'],
       dtype='<U9')}
```

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

In [2]:
```
cancer.keys()
```

Out[2]:
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

Question 0 (Example)

How many features does the breast cancer dataset have?

This function should return an integer.

In [3]:
```
# You should write your whole answer within the function provided. The autograder will call
# this function and compare the return value against the correct solution value
def answer_zero():
    # This function returns the number of features of the breast cancer dataset, which is an integer.
    # The assignment question description will tell you the general format the autograder is expecting
    return len(cancer['feature_names'])

# You can examine what your function returns by calling it in the cell. If you have questions
# about the assignment formats, check out the discussion forums for any FAQs
answer_zero()
```

Out[3]: 30

Question 1

Scikit-learn works with lists, numpy arrays, scipy-sparse matrices, and pandas DataFrames, so converting the dataset to a DataFrame is not necessary for training this model. Using a DataFrame does however help make many things easier such as munging data, so let's practice creating a classifier with a pandas DataFrame.

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 [5]:
```
def answer_one():

    # Your code here
#     print(len(cancer['feature_names']))
    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)
#     print(cancer['data'].shape)
    df = pd.DataFrame(data=cancer['data'], index=index, columns = columns[:30])
#     print(cancer['target'])
    df['target'] = cancer['target']

#     print(df.head())
    ans = df

    return ans


answer_one()
```

Out[5]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	wors smo
0	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.2419	0.07871	...	17.33	184.60	2019.0	0.16
1	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.1812	0.05667	...	23.41	158.80	1956.0	0.12
2	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.2069	0.05999	...	25.53	152.50	1709.0	0.14
3	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.2597	0.09744	...	26.50	98.87	567.7	0.20
4	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.1809	0.05883	...	16.67	152.20	1575.0	0.13
5	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890	0.2087	0.07613	...	23.75	103.40	741.6	0.17
6	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000	0.1794	0.05742	...	27.66	153.20	1606.0	0.14
7	13.710	20.83	90.20	577.9	0.11890	0.16450	0.093660	0.059850	0.2196	0.07451	...	28.14	110.60	897.0	0.16
8	13.000	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530	0.2350	0.07389	...	30.73	106.20	739.3	0.17
9	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.085430	0.2030	0.08243	...	40.68	97.65	711.4	0.18
10	16.020	23.24	102.70	797.8	0.08206	0.06669	0.032990	0.033230	0.1528	0.05697	...	33.88	123.80	1150.0	0.11
11	15.780	17.89	103.60	781.0	0.09710	0.12920	0.099540	0.066060	0.1842	0.06082	...	27.28	136.50	1299.0	0.13
12	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.206500	0.111800	0.2397	0.07800	...	29.94	151.70	1332.0	0.10
13	15.850	23.95	103.70	782.7	0.08401	0.10020	0.099380	0.053640	0.1847	0.05338	...	27.66	112.00	876.5	0.11
14	13.730	22.61	93.60	578.3	0.11310	0.22930	0.212800	0.080250	0.2069	0.07682	...	32.01	108.80	697.7	0.16
15	14.540	27.54	96.73	658.8	0.11390	0.15950	0.163900	0.073640	0.2303	0.07077	...	37.13	124.10	943.2	0.16
16	14.680	20.13	94.74	684.5	0.09867	0.07200	0.073950	0.052590	0.1586	0.05922	...	30.88	123.40	1138.0	0.14
17	16.130	20.68	108.10	798.8	0.11700	0.20220	0.172200	0.102800	0.2164	0.07356	...	31.48	136.80	1315.0	0.17
18	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.147900	0.094980	0.1582	0.05395	...	30.88	186.80	2398.0	0.15
19	13.540	14.36	87.46	566.3	0.09779	0.08129	0.066640	0.047810	0.1885	0.05766	...	19.26	99.70	711.2	0.14
20	13.080	15.71	85.63	520.0	0.10750	0.12700	0.045680	0.031100	0.1967	0.06811	...	20.49	96.09	630.5	0.13
21	9.504	12.44	60.34	273.9	0.10240	0.06492	0.029560	0.020760	0.1815	0.06905	...	15.66	65.13	314.9	0.13
22	15.340	14.26	102.50	704.4	0.10730	0.21350	0.207700	0.097560	0.2521	0.07032	...	19.08	125.10	980.9	0.13
23	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.109700	0.086320	0.1769	0.05278	...	35.59	188.00	2615.0	0.14
24	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700	0.1995	0.06330	...	31.56	177.00	2215.0	0.18
25	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100	0.3040	0.07413	...	21.40	152.40	1461.0	0.15
26	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830	0.2252	0.06924	...	33.21	122.40	896.9	0.15
27	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310	0.1697	0.05699	...	27.26	139.90	1403.0	0.13
28	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510	0.1926	0.06540	...	36.71	149.30	1269.0	0.16
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530	0.1739	0.06149	...	19.52	134.90	1227.0	0.12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
539	7.691	25.44	48.34	170.4	0.08668	0.11990	0.092520	0.013640	0.2037	0.07751	...	31.89	54.49	223.6	0.15
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.025940	0.1818	0.06782	...	19.68	78.78	457.8	0.13
541	14.470	24.99	95.81	656.4	0.08837	0.12300	0.100900	0.038900	0.1872	0.06341	...	31.73	113.50	808.9	0.13
542	14.740	25.42	94.70	668.6	0.08275	0.07214	0.041050	0.030270	0.1840	0.05680	...	32.29	107.40	826.4	0.10
543	13.210	28.06	84.88	538.4	0.08671	0.06877	0.029870	0.032750	0.1628	0.05781	...	37.17	92.48	629.6	0.10
544	13.870	20.70	89.77	584.8	0.09578	0.10180	0.036880	0.023690	0.1620	0.06688	...	24.75	99.17	688.6	0.12
545	13.620	23.23	87.19	573.2	0.09246	0.06747	0.029740	0.024430	0.1664	0.05801	...	29.09	97.58	729.8	0.12
546	10.320	16.35	65.31	324.9	0.09434	0.04994	0.010120	0.005495	0.1885	0.06201	...	21.77	71.12	384.9	0.12
547	10.260	16.58	65.85	320.8	0.08877	0.08066	0.043580	0.024380	0.1669	0.06714	...	22.04	71.08	357.4	0.14
548	9.683	19.34	61.05	285.7	0.08491	0.05030	0.023370	0.009615	0.1580	0.06235	...	25.59	69.10	364.2	0.11
549	10.820	24.21	68.89	361.6	0.08192	0.06602	0.015480	0.008160	0.1976	0.06328	...	31.45	83.90	505.6	0.12
550	10.860	21.48	68.51	360.5	0.07431	0.04227	0.000000	0.000000	0.1661	0.05948	...	24.77	74.08	412.3	0.10
551	11.130	22.44	71.49	378.4	0.09566	0.08194	0.048240	0.022570	0.2030	0.06552	...	28.26	77.80	436.6	0.10
552	12.770	29.43	81.35	507.9	0.08276	0.04234	0.019970	0.014990	0.1539	0.05637	...	36.00	88.10	594.7	0.12
553	9.333	21.94	59.01	264.0	0.09240	0.05605	0.039960	0.012820	0.1692	0.06576	...	25.05	62.86	295.8	0.11
554	12.880	28.92	82.50	514.3	0.08123	0.05824	0.061950	0.023430	0.1566	0.05708	...	35.74	88.84	595.7	0.12
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.027380	0.1593	0.06127	...	34.91	69.57	357.6	0.13
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.011160	0.1791	0.06331	...	22.88	67.88	347.3	0.12
557	9.423	27.88	59.26	271.3	0.08123	0.04971	0.000000	0.000000	0.1742	0.06059	...	34.24	66.50	330.6	0.10
558	14.590	22.68	96.39	657.1	0.08473	0.13300	0.102900	0.037360	0.1454	0.06147	...	27.27	105.90	733.5	0.10
559	11.510	23.93	74.52	403.5	0.09261	0.10210	0.111200	0.041050	0.1388	0.06570	...	37.16	82.28	474.2	0.12
560	14.050	27.15	91.38	600.4	0.09929	0.11260	0.044620	0.043040	0.1537	0.06171	...	33.17	100.20	706.7	0.12
561	11.200	29.37	70.67	386.0	0.07449	0.03558	0.000000	0.000000	0.1060	0.05502	...	38.30	75.19	439.6	0.09
562	15.220	30.62	103.40	716.9	0.10480	0.20870	0.255000	0.094290	0.2128	0.07152	...	42.79	128.70	915.0	0.14
563	20.920	25.09	143.00	1347.0	0.10990	0.22360	0.317400	0.147400	0.2149	0.06879	...	29.41	179.10	1819.0	0.14
564	21.560	22.39	142.00	1479.0	0.11100	0.11590	0.243900	0.138900	0.1726	0.05623	...	26.40	166.10	2027.0	0.14
565	20.130	28.25	131.20	1261.0	0.09780	0.10340	0.144000	0.097910	0.1752	0.05533	...	38.25	155.00	1731.0	0.11
566	16.600	28.08	108.30	858.1	0.08455	0.10230	0.092510	0.053020	0.1590	0.05648	...	34.12	126.70	1124.0	0.11
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.152000	0.2397	0.07016	...	39.42	184.60	1821.0	0.16
568	7.760	24.54	47.92	181.0	0.05263	0.04362	0.000000	0.000000	0.1587	0.05884	...	30.37	59.16	268.6	0.08

569 rows × 31 columns

Question 2

What is the class distribution? (i.e. how many instances of `malignant` (encoded 0) and how many `benign` (encoded 1)?)

This function should return a Series named `target` of length 2 with integer values and index = `['malignant', 'benign']`

In [10]:
```
def answer_two():
    cancerdf = answer_one()
    index = ['malignant', 'benign']
    # Your code here
    a = cancerdf.loc[cancerdf["target"]==0,"target"].value_counts().sum()
    b = cancerdf.loc[cancerdf["target"]==1,"target"].value_counts().sum()
    target = pd.Series(data=[a,b], index=index)
    return target# Return your answer


answer_two()
```

Out[10]:
```
malignant    212
benign       357
dtype: int64
```

Question 3

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

This function should return a tuple of length 2: `(X, y)`*, where*

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

In [16]:
```
def answer_three():
    cancerdf = answer_one()
    X = X = cancerdf.iloc[:,:30]
    y = cancerdf.target
    a = [X,y]
    # Your code here

    return a
```

Question 4

Using `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` to make sure your results match the autograder!

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 [19]:
```
from sklearn.model_selection import train_test_split

def answer_four():
    X, y = answer_three()
    X_train, X_test, y_train, y_test = train_test_split(X,y,random_state =0)

    # Your code here

    return X_train, X_test, y_train, y_test
```

Question 5

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

This function should return a `sklearn.neighbors.classification.KNeighborsClassifier`.

In [20]:
```
from sklearn.neighbors import KNeighborsClassifier

def answer_five():
    X_train, X_test, y_train, y_test = answer_four()
    knn = KNeighborsClassifier(n_neighbors = 1)
    knn.fit(X_train,y_train)

    # Your code here

    return knn # Return your answer
```

Question 6

Using your knn classifier, predict the class label using the mean value for each feature.

Hint: You can use `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 precict method of KNeighborsClassifier).

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

In [22]:
```
def answer_six():
    cancerdf = answer_one()
    means = cancerdf.mean()[:-1].values.reshape(1, -1)
    a = knn.predict(means)

    # Your code here

    return np.array(a)# Return your answer
```

Question 7

Using your knn classifier, predict the class labels for the test set `X_test`.

This function should return a numpy array with shape `(143,)` and values either `0.0` or `1.0`.

In [25]:
```
def answer_seven():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()
    r = knn.predict(X_test)
    r = np.array(r)

    # Your code here

    return r# Return your answer
```

Question 8

Find the score (mean accuracy) of your knn classifier using `X_test` and `y_test`.

This function should return a float between 0 and 1

In [27]:
```
def answer_eight():
    X_train, X_test, y_train, y_test = answer_four()
    knn = answer_five()
    a = knn.score(X_test,y_test)

    # Your code here

    return a # Return your answer
```

Out[27]: 0.91608391608391604

Optional plot

Try using the plotting function below to visualize the differet predicition scores between training and test sets, as well as malignant and benign cells.

In [28]:
```
def accuracy_plot():
    import matplotlib.pyplot as plt

    %matplotlib notebook

    X_train, X_test, y_train, y_test = answer_four()

    # 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]

    knn = answer_five()

    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}f}'.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)
```

Uncomment the plotting function to see the visualization.

Comment out the plotting function when submitting your notebook for grading.

In [1]:
```
#accuracy_plot()
```

In [ ]: