HW1

April 13, 2018

```
In [1]: import numpy as np
        import pandas as pd
```

1 Data Preprocessing

Question 1: Remove the rows with missing labels ('class') and rows with more than 7 missing features. Report the remaining number of rows.

```
In [2]: wine_modified = pd.read_csv("wine_modified.csv")
        wine_modified = wine_modified[wine_modified['class'].notnull()]
        wine_modified = wine_modified[np.isnan(wine_modified).sum(axis=1)<7]
        wine_modified
```

```
Out[2]:      class  Alcohol  Malic acid   Ash  Alcalinity of ash  Magnesium  \
        0      1.0   14.230        1.71  2.43               15.6      127.0
        1      1.0   13.200        1.78   NaN               11.2      100.0
        2      1.0   13.160        2.36   NaN               18.6      101.0
        4      1.0   13.240        2.59   NaN               21.0      118.0
        5      1.0   14.200        1.76   NaN               15.2      112.0
        6      1.0   14.390        1.87  2.45               14.6       96.0
        7      1.0   14.060        2.15  2.61               17.6      121.0
        8      1.0   14.830        1.64  2.17               14.0       97.0
        9      1.0   13.860        1.35   NaN               16.0       98.0
        10     1.0   14.100        2.16   NaN               18.0        NaN
        12     1.0   13.750        1.73   NaN               16.0       89.0
        13     1.0   14.750        1.73   NaN               11.4       91.0
        14     1.0   14.380        1.87  2.38               12.0      102.0
        17     1.0   13.830        1.57   NaN               20.0      115.0
        18     1.0   14.190        1.59   NaN               16.5      108.0
        19     1.0   13.640        3.10   NaN               15.2      116.0
        20     1.0   14.060        1.63  2.28               16.0      126.0
        21     1.0   12.930        3.80   NaN               18.6      102.0
        22     1.0   -8.226        1.86  2.36               16.6      101.0
        23     1.0   12.850        1.60  2.52               17.8       95.0
        24     1.0   13.500        1.81   NaN               20.0       96.0
        25     1.0   13.050        2.05  3.22               25.0      124.0
        26     1.0   13.390        1.77   NaN               16.1       93.0
```

27	1.0	13.300	1.72	NaN	17.0	94.0
28	1.0	13.870	1.90	NaN	19.4	107.0
29	1.0	14.020	1.68	NaN	16.0	96.0
30	1.0	13.730	1.50	NaN	22.5	101.0
33	1.0	13.760	1.53	2.70	19.5	132.0
34	1.0	13.510	1.80	NaN	19.0	110.0
35	1.0	13.480	1.81	2.41	20.5	100.0
..	...	...	...	...	...	...
144	3.0	12.250	3.88	2.20	18.5	112.0
145	3.0	13.160	3.57	NaN	21.0	102.0
146	3.0	13.880	5.04	2.23	20.0	80.0
148	3.0	13.320	3.24	NaN	21.5	92.0
150	3.0	13.500	3.12	NaN	24.0	123.0
151	3.0	12.790	2.67	NaN	22.0	112.0
152	3.0	13.110	1.90	NaN	25.5	NaN
153	3.0	13.230	3.30	2.28	18.5	98.0
154	3.0	12.580	1.29	NaN	20.0	103.0
155	3.0	13.170	5.19	NaN	22.0	93.0
156	3.0	13.840	4.12	2.38	19.5	89.0
157	3.0	12.450	3.03	NaN	27.0	97.0
158	3.0	14.340	1.68	2.70	25.0	98.0
159	3.0	13.480	1.67	NaN	22.5	89.0
160	3.0	12.360	3.83	2.38	21.0	88.0
161	3.0	13.690	3.26	NaN	20.0	107.0
162	3.0	12.850	3.27	NaN	22.0	106.0
163	3.0	-7.776	3.45	NaN	18.5	106.0
164	3.0	13.780	2.76	NaN	22.0	NaN
165	3.0	13.730	4.36	NaN	22.5	88.0
166	3.0	13.450	3.70	2.60	23.0	111.0
168	3.0	13.580	2.58	2.69	24.5	105.0
169	3.0	13.400	4.60	2.86	25.0	112.0
171	3.0	12.770	2.39	NaN	19.5	86.0
172	3.0	14.160	2.51	2.48	20.0	91.0
173	3.0	13.710	5.65	NaN	20.5	95.0
174	3.0	13.400	3.91	2.48	23.0	102.0
175	3.0	13.270	4.28	NaN	20.0	120.0
176	3.0	13.170	2.59	NaN	20.0	120.0
177	3.0	14.130	4.10	NaN	24.5	96.0

	Total phenols	Flavanoids	Nonflavanoid phenols	Proanthocyanins	\
0	2.80	3.06	0.28	2.29	
1	2.65	2.76	0.26	1.28	
2	2.80	3.24	0.30	2.81	
4	2.80	2.69	0.39	1.82	
5	3.27	NaN	0.34	1.97	
6	2.50	2.52	0.30	1.98	
7	2.60	2.51	0.31	1.25	
8	2.80	2.98	0.29	1.98	

9	2.98	NaN	0.22	1.85
10	2.95	NaN	0.22	2.38
12	2.60	2.76	0.29	1.81
13	3.10	3.69	0.43	2.81
14	3.30	NaN	0.29	2.96
17	2.95	3.40	0.40	1.72
18	3.30	NaN	0.32	1.86
19	2.70	NaN	0.17	1.66
20	3.00	3.17	0.24	2.10
21	2.41	2.41	0.25	1.98
22	2.61	2.88	0.27	1.69
23	2.48	2.37	0.26	1.46
24	2.53	2.61	0.28	1.66
25	2.63	2.68	0.47	1.92
26	2.85	NaN	0.34	1.45
27	2.40	2.19	0.27	1.35
28	2.95	2.97	0.37	1.76
29	2.65	2.33	0.26	1.98
30	3.00	3.25	0.29	2.38
33	2.95	2.74	0.50	1.35
34	2.35	2.53	0.29	1.54
35	2.70	NaN	0.26	1.86
..	...	...	...	...
144	1.38	NaN	0.29	1.14
145	1.50	0.55	0.43	1.30
146	0.98	0.34	0.40	0.68
148	1.93	0.76	0.45	1.25
150	1.40	1.57	0.22	1.25
151	1.48	NaN	0.24	1.26
152	2.20	1.28	0.26	1.56
153	1.80	0.83	0.61	1.87
154	1.48	0.58	0.53	1.40
155	1.74	0.63	0.61	1.55
156	1.80	0.83	0.48	1.56
157	1.90	0.58	0.63	1.14
158	2.80	1.31	0.53	2.70
159	2.60	NaN	0.52	2.29
160	2.30	0.92	0.50	1.04
161	1.83	0.56	0.50	0.80
162	1.65	NaN	0.60	0.96
163	1.39	0.70	0.40	0.94
164	1.35	0.68	0.41	1.03
165	1.28	0.47	0.52	1.15
166	1.70	0.92	0.43	1.46
168	1.55	NaN	0.39	1.54
169	1.98	0.96	0.27	1.11
171	1.39	0.51	0.48	0.64
172	1.68	0.70	0.44	1.24

173	1.68	0.61	0.52	1.06
174	1.80	NaN	0.43	1.41
175	1.59	0.69	0.43	1.35
176	1.65	0.68	0.53	1.46
177	2.05	NaN	0.56	1.35

	Color intensity	Hue	OD280/OD315	Proline
0	5.640000	1.04	3.92	1065.0
1	4.380000	1.05	3.40	1050.0
2	5.680000	1.03	3.17	1185.0
4	4.320000	1.04	2.93	735.0
5	6.750000	1.05	2.85	1450.0
6	5.250000	1.02	3.58	1290.0
7	5.050000	1.06	3.58	1295.0
8	5.200000	1.08	2.85	1045.0
9	7.220000	1.01	3.55	1045.0
10	5.750000	1.25	3.17	1510.0
12	5.600000	1.15	2.90	1320.0
13	5.400000	1.25	2.73	1150.0
14	7.500000	1.20	3.00	1547.0
17	6.600000	1.13	2.57	1130.0
18	8.700000	1.23	2.82	1680.0
19	5.100000	0.96	3.36	845.0
20	5.650000	1.09	3.71	780.0
21	4.500000	1.03	3.52	770.0
22	3.800000	1.11	4.00	1035.0
23	3.930000	1.09	3.63	1015.0
24	3.520000	1.12	3.82	845.0
25	3.580000	1.13	3.20	830.0
26	4.800000	0.92	3.22	1195.0
27	3.950000	1.02	2.77	1285.0
28	4.500000	1.25	3.40	915.0
29	4.700000	1.04	3.59	1035.0
30	5.700000	1.19	2.71	1285.0
33	5.400000	1.25	3.00	1235.0
34	4.200000	1.10	2.87	1095.0
35	5.100000	1.04	3.47	920.0
..	...	...	...	...
144	8.210000	0.65	2.00	855.0
145	4.000000	0.60	1.68	830.0
146	4.900000	0.58	1.33	415.0
148	8.420000	0.55	1.62	650.0
150	8.600000	0.59	1.30	500.0
151	10.800000	0.48	1.47	480.0
152	7.100000	0.61	1.33	425.0
153	10.520000	0.56	1.51	675.0
154	7.600000	0.58	1.55	640.0
155	7.900000	0.60	1.48	725.0

156	9.010000	0.57	1.64	480.0
157	7.500000	0.67	1.73	880.0
158	13.000000	0.57	1.96	660.0
159	11.750000	0.57	1.78	620.0
160	7.650000	0.56	1.58	520.0
161	5.880000	0.96	1.82	680.0
162	5.580000	0.87	2.11	570.0
163	5.280000	0.68	1.75	675.0
164	9.580000	0.70	1.68	615.0
165	6.620000	0.78	1.75	520.0
166	10.680000	0.85	1.56	695.0
168	8.660000	0.74	1.80	750.0
169	8.500000	0.67	1.92	630.0
171	9.899999	0.57	1.63	470.0
172	9.700000	0.62	1.71	660.0
173	7.700000	0.64	1.74	740.0
174	7.300000	0.70	1.56	750.0
175	10.200000	0.59	1.56	835.0
176	9.300000	0.60	1.62	840.0
177	9.200000	0.61	1.60	560.0

[154 rows x 14 columns]

In [50]:
```
wine_modified.shape
```

Out[50]: (154, 13)

Question 2: Remove features with > 50% of missing values. For other features with missing values fill them with the mean of the corresponding features. Report the removed features (if any) and standard deviation of features with missing values after filling. (2 marks)

In [3]:
```
print ("Removed features:")
print (list((np.isnan(wine_modified).sum()>len(wine_modified)/2).index[(np.isnan(wine_m

wine_modified = wine_modified[(np.isnan(wine_modified).sum()<len(wine_modified)/2).inde
features_with_missing_values = list(np.isnan(wine_modified).sum().index[(np.isnan(wine_
```

```
Removed features:
['Ash']
```

In [4]:
```
wine_modified = wine_modified.fillna(wine_modified.mean().to_dict())
wine_modified
```

Out[4]:

	class	Alcohol	Malic acid	Alcalinity of ash	Magnesium	Total phenols \
0	1.0	14.230	1.71	15.6	127.000000	2.80
1	1.0	13.200	1.78	11.2	100.000000	2.65
2	1.0	13.160	2.36	18.6	101.000000	2.80
4	1.0	13.240	2.59	21.0	118.000000	2.80

5	1.0	14.200	1.76	15.2	112.000000	3.27
6	1.0	14.390	1.87	14.6	96.000000	2.50
7	1.0	14.060	2.15	17.6	121.000000	2.60
8	1.0	14.830	1.64	14.0	97.000000	2.80
9	1.0	13.860	1.35	16.0	98.000000	2.98
10	1.0	14.100	2.16	18.0	99.496552	2.95
12	1.0	13.750	1.73	16.0	89.000000	2.60
13	1.0	14.750	1.73	11.4	91.000000	3.10
14	1.0	14.380	1.87	12.0	102.000000	3.30
17	1.0	13.830	1.57	20.0	115.000000	2.95
18	1.0	14.190	1.59	16.5	108.000000	3.30
19	1.0	13.640	3.10	15.2	116.000000	2.70
20	1.0	14.060	1.63	16.0	126.000000	3.00
21	1.0	12.930	3.80	18.6	102.000000	2.41
22	1.0	-8.226	1.86	16.6	101.000000	2.61
23	1.0	12.850	1.60	17.8	95.000000	2.48
24	1.0	13.500	1.81	20.0	96.000000	2.53
25	1.0	13.050	2.05	25.0	124.000000	2.63
26	1.0	13.390	1.77	16.1	93.000000	2.85
27	1.0	13.300	1.72	17.0	94.000000	2.40
28	1.0	13.870	1.90	19.4	107.000000	2.95
29	1.0	14.020	1.68	16.0	96.000000	2.65
30	1.0	13.730	1.50	22.5	101.000000	3.00
33	1.0	13.760	1.53	19.5	132.000000	2.95
34	1.0	13.510	1.80	19.0	110.000000	2.35
35	1.0	13.480	1.81	20.5	100.000000	2.70
..	...	...	...	...	...	...
144	3.0	12.250	3.88	18.5	112.000000	1.38
145	3.0	13.160	3.57	21.0	102.000000	1.50
146	3.0	13.880	5.04	20.0	80.000000	0.98
148	3.0	13.320	3.24	21.5	92.000000	1.93
150	3.0	13.500	3.12	24.0	123.000000	1.40
151	3.0	12.790	2.67	22.0	112.000000	1.48
152	3.0	13.110	1.90	25.5	99.496552	2.20
153	3.0	13.230	3.30	18.5	98.000000	1.80
154	3.0	12.580	1.29	20.0	103.000000	1.48
155	3.0	13.170	5.19	22.0	93.000000	1.74
156	3.0	13.840	4.12	19.5	89.000000	1.80
157	3.0	12.450	3.03	27.0	97.000000	1.90
158	3.0	14.340	1.68	25.0	98.000000	2.80
159	3.0	13.480	1.67	22.5	89.000000	2.60
160	3.0	12.360	3.83	21.0	88.000000	2.30
161	3.0	13.690	3.26	20.0	107.000000	1.83
162	3.0	12.850	3.27	22.0	106.000000	1.65
163	3.0	-7.776	3.45	18.5	106.000000	1.39
164	3.0	13.780	2.76	22.0	99.496552	1.35
165	3.0	13.730	4.36	22.5	88.000000	1.28
166	3.0	13.450	3.70	23.0	111.000000	1.70

168	3.0	13.580	2.58	24.5	105.000000	1.55
169	3.0	13.400	4.60	25.0	112.000000	1.98
171	3.0	12.770	2.39	19.5	86.000000	1.39
172	3.0	14.160	2.51	20.0	91.000000	1.68
173	3.0	13.710	5.65	20.5	95.000000	1.68
174	3.0	13.400	3.91	23.0	102.000000	1.80
175	3.0	13.270	4.28	20.0	120.000000	1.59
176	3.0	13.170	2.59	20.0	120.000000	1.65
177	3.0	14.130	4.10	24.5	96.000000	2.05

	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue \
0	3.060000	0.28	2.29	5.640000	1.04
1	2.760000	0.26	1.28	4.380000	1.05
2	3.240000	0.30	2.81	5.680000	1.03
4	2.690000	0.39	1.82	4.320000	1.04
5	1.937983	0.34	1.97	6.750000	1.05
6	2.520000	0.30	1.98	5.250000	1.02
7	2.510000	0.31	1.25	5.050000	1.06
8	2.980000	0.29	1.98	5.200000	1.08
9	1.937983	0.22	1.85	7.220000	1.01
10	1.937983	0.22	2.38	5.750000	1.25
12	2.760000	0.29	1.81	5.600000	1.15
13	3.690000	0.43	2.81	5.400000	1.25
14	1.937983	0.29	2.96	7.500000	1.20
17	3.400000	0.40	1.72	6.600000	1.13
18	1.937983	0.32	1.86	8.700000	1.23
19	1.937983	0.17	1.66	5.100000	0.96
20	3.170000	0.24	2.10	5.650000	1.09
21	2.410000	0.25	1.98	4.500000	1.03
22	2.880000	0.27	1.69	3.800000	1.11
23	2.370000	0.26	1.46	3.930000	1.09
24	2.610000	0.28	1.66	3.520000	1.12
25	2.680000	0.47	1.92	3.580000	1.13
26	1.937983	0.34	1.45	4.800000	0.92
27	2.190000	0.27	1.35	3.950000	1.02
28	2.970000	0.37	1.76	4.500000	1.25
29	2.330000	0.26	1.98	4.700000	1.04
30	3.250000	0.29	2.38	5.700000	1.19
33	2.740000	0.50	1.35	5.400000	1.25
34	2.530000	0.29	1.54	4.200000	1.10
35	1.937983	0.26	1.86	5.100000	1.04
..	...	...	...	...	...
144	1.937983	0.29	1.14	8.210000	0.65
145	0.550000	0.43	1.30	4.000000	0.60
146	0.340000	0.40	0.68	4.900000	0.58
148	0.760000	0.45	1.25	8.420000	0.55
150	1.570000	0.22	1.25	8.600000	0.59
151	1.937983	0.24	1.26	10.800000	0.48

152	1.280000	0.26	1.56	7.100000	0.61
153	0.830000	0.61	1.87	10.520000	0.56
154	0.580000	0.53	1.40	7.600000	0.58
155	0.630000	0.61	1.55	7.900000	0.60
156	0.830000	0.48	1.56	9.010000	0.57
157	0.580000	0.63	1.14	7.500000	0.67
158	1.310000	0.53	2.70	13.000000	0.57
159	1.937983	0.52	2.29	11.750000	0.57
160	0.920000	0.50	1.04	7.650000	0.56
161	0.560000	0.50	0.80	5.880000	0.96
162	1.937983	0.60	0.96	5.580000	0.87
163	0.700000	0.40	0.94	5.280000	0.68
164	0.680000	0.41	1.03	9.580000	0.70
165	0.470000	0.52	1.15	6.620000	0.78
166	0.920000	0.43	1.46	10.680000	0.85
168	1.937983	0.39	1.54	8.660000	0.74
169	0.960000	0.27	1.11	8.500000	0.67
171	0.510000	0.48	0.64	9.899999	0.57
172	0.700000	0.44	1.24	9.700000	0.62
173	0.610000	0.52	1.06	7.700000	0.64
174	1.937983	0.43	1.41	7.300000	0.70
175	0.690000	0.43	1.35	10.200000	0.59
176	0.680000	0.53	1.46	9.300000	0.60
177	1.937983	0.56	1.35	9.200000	0.61

	OD280/OD315	Proline
0	3.92	1065.0
1	3.40	1050.0
2	3.17	1185.0
4	2.93	735.0
5	2.85	1450.0
6	3.58	1290.0
7	3.58	1295.0
8	2.85	1045.0
9	3.55	1045.0
10	3.17	1510.0
12	2.90	1320.0
13	2.73	1150.0
14	3.00	1547.0
17	2.57	1130.0
18	2.82	1680.0
19	3.36	845.0
20	3.71	780.0
21	3.52	770.0
22	4.00	1035.0
23	3.63	1015.0
24	3.82	845.0
25	3.20	830.0

```
26           3.22   1195.0
27           2.77   1285.0
28           3.40    915.0
29           3.59   1035.0
30           2.71   1285.0
33           3.00   1235.0
34           2.87   1095.0
35           3.47    920.0
..            ...      ...
144          2.00    855.0
145          1.68    830.0
146          1.33    415.0
148          1.62    650.0
150          1.30    500.0
151          1.47    480.0
152          1.33    425.0
153          1.51    675.0
154          1.55    640.0
155          1.48    725.0
156          1.64    480.0
157          1.73    880.0
158          1.96    660.0
159          1.78    620.0
160          1.58    520.0
161          1.82    680.0
162          2.11    570.0
163          1.75    675.0
164          1.68    615.0
165          1.75    520.0
166          1.56    695.0
168          1.80    750.0
169          1.92    630.0
171          1.63    470.0
172          1.71    660.0
173          1.74    740.0
174          1.56    750.0
175          1.56    835.0
176          1.62    840.0
177          1.60    560.0

[154 rows x 13 columns]
```

In [5]:
```
print ("standard deviation of features with missing values after filling")
wine_modified.std()[features_with_missing_values]
```

```
standard deviation of features with missing values after filling
```

Out[5]:
```
Magnesium     14.440377
```

```
Flavanoids     0.873573
dtype: float64
```

In [6]:
```
wine_modified.std()
```

Out[6]:
```
class                     0.766522
Alcohol                   3.804067
Malic acid                1.116005
Alcalinity of ash         3.456794
Magnesium                14.440377
Total phenols             0.617237
Flavanoids                0.873573
Nonflavanoid phenols      0.127083
Proanthocyanins           0.587671
Color intensity           2.325204
Hue                       0.229412
OD280/OD315               0.723261
Proline                 303.033368
dtype: float64
```

Question 3: Detect and remove rows with any outliers/incorrect values in features 'alcohol' and 'proline' (if any). Clearly state the basis of your removal. (1 mark)

In [52]:
```
wine_modified['Alcohol'][wine_modified['Alcohol']<0]
```

Out[52]:
```
22    -8.226
42    -8.328
61    -7.584
118   -7.662
163   -7.776
Name: Alcohol, dtype: float64
```

In [53]:
```
wine_modified['Proline'][wine_modified['Proline']<0]
```

Out[53]:
```
Series([], Name: Proline, dtype: float64)
```

In [54]:
```
print ((wine_modified<0).sum())
print ("\n\nFrom the above output only Alcohol has values which are less than 0")
```

```
class                   0
Alcohol                 5
Malic acid              0
Alcalinity of ash       0
Magnesium               0
Total phenols           0
Flavanoids              0
Nonflavanoid phenols    0
Proanthocyanins         0
Color intensity         0
```

```
Hue                         0
OD280/OD315                 0
Proline                     0
dtype: int64
```

From the above output only Alcohol has values which are less than 0

In [55]:
```python
print ("Removing the rows which have incorrect values in features Alcohol")
```

```
Removing the rows which have incorrect values in features Alcohol
```

In [57]:
```python
wine_modified = wine_modified[wine_modified['Alcohol']>0]
wine_modified
```

Out[57]:
```
     class  Alcohol  Malic acid  Alcalinity of ash   Magnesium  Total phenols  \
0      1.0    14.23        1.71               15.6  127.000000           2.80
1      1.0    13.20        1.78               11.2  100.000000           2.65
2      1.0    13.16        2.36               18.6  101.000000           2.80
4      1.0    13.24        2.59               21.0  118.000000           2.80
5      1.0    14.20        1.76               15.2  112.000000           3.27
6      1.0    14.39        1.87               14.6   96.000000           2.50
7      1.0    14.06        2.15               17.6  121.000000           2.60
8      1.0    14.83        1.64               14.0   97.000000           2.80
9      1.0    13.86        1.35               16.0   98.000000           2.98
10     1.0    14.10        2.16               18.0   99.496552           2.95
12     1.0    13.75        1.73               16.0   89.000000           2.60
13     1.0    14.75        1.73               11.4   91.000000           3.10
14     1.0    14.38        1.87               12.0  102.000000           3.30
17     1.0    13.83        1.57               20.0  115.000000           2.95
18     1.0    14.19        1.59               16.5  108.000000           3.30
19     1.0    13.64        3.10               15.2  116.000000           2.70
20     1.0    14.06        1.63               16.0  126.000000           3.00
21     1.0    12.93        3.80               18.6  102.000000           2.41
23     1.0    12.85        1.60               17.8   95.000000           2.48
24     1.0    13.50        1.81               20.0   96.000000           2.53
25     1.0    13.05        2.05               25.0  124.000000           2.63
26     1.0    13.39        1.77               16.1   93.000000           2.85
27     1.0    13.30        1.72               17.0   94.000000           2.40
28     1.0    13.87        1.90               19.4  107.000000           2.95
29     1.0    14.02        1.68               16.0   96.000000           2.65
30     1.0    13.73        1.50               22.5  101.000000           3.00
33     1.0    13.76        1.53               19.5  132.000000           2.95
34     1.0    13.51        1.80               19.0  110.000000           2.35
35     1.0    13.48        1.81               20.5  100.000000           2.70
36     1.0    13.28        1.64               15.5  110.000000           2.60
..     ...      ...         ...                ...         ...            ...
```

143	3.0	13.62	4.95	20.0	92.000000	2.00
144	3.0	12.25	3.88	18.5	112.000000	1.38
145	3.0	13.16	3.57	21.0	102.000000	1.50
146	3.0	13.88	5.04	20.0	80.000000	0.98
148	3.0	13.32	3.24	21.5	92.000000	1.93
150	3.0	13.50	3.12	24.0	123.000000	1.40
151	3.0	12.79	2.67	22.0	112.000000	1.48
152	3.0	13.11	1.90	25.5	99.496552	2.20
153	3.0	13.23	3.30	18.5	98.000000	1.80
154	3.0	12.58	1.29	20.0	103.000000	1.48
155	3.0	13.17	5.19	22.0	93.000000	1.74
156	3.0	13.84	4.12	19.5	89.000000	1.80
157	3.0	12.45	3.03	27.0	97.000000	1.90
158	3.0	14.34	1.68	25.0	98.000000	2.80
159	3.0	13.48	1.67	22.5	89.000000	2.60
160	3.0	12.36	3.83	21.0	88.000000	2.30
161	3.0	13.69	3.26	20.0	107.000000	1.83
162	3.0	12.85	3.27	22.0	106.000000	1.65
164	3.0	13.78	2.76	22.0	99.496552	1.35
165	3.0	13.73	4.36	22.5	88.000000	1.28
166	3.0	13.45	3.70	23.0	111.000000	1.70
168	3.0	13.58	2.58	24.5	105.000000	1.55
169	3.0	13.40	4.60	25.0	112.000000	1.98
171	3.0	12.77	2.39	19.5	86.000000	1.39
172	3.0	14.16	2.51	20.0	91.000000	1.68
173	3.0	13.71	5.65	20.5	95.000000	1.68
174	3.0	13.40	3.91	23.0	102.000000	1.80
175	3.0	13.27	4.28	20.0	120.000000	1.59
176	3.0	13.17	2.59	20.0	120.000000	1.65
177	3.0	14.13	4.10	24.5	96.000000	2.05

	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue \
0	3.060000	0.28	2.29	5.640000	1.04
1	2.760000	0.26	1.28	4.380000	1.05
2	3.240000	0.30	2.81	5.680000	1.03
4	2.690000	0.39	1.82	4.320000	1.04
5	1.937983	0.34	1.97	6.750000	1.05
6	2.520000	0.30	1.98	5.250000	1.02
7	2.510000	0.31	1.25	5.050000	1.06
8	2.980000	0.29	1.98	5.200000	1.08
9	1.937983	0.22	1.85	7.220000	1.01
10	1.937983	0.22	2.38	5.750000	1.25
12	2.760000	0.29	1.81	5.600000	1.15
13	3.690000	0.43	2.81	5.400000	1.25
14	1.937983	0.29	2.96	7.500000	1.20
17	3.400000	0.40	1.72	6.600000	1.13
18	1.937983	0.32	1.86	8.700000	1.23
19	1.937983	0.17	1.66	5.100000	0.96

20	3.170000	0.24	2.10	5.650000	1.09
21	2.410000	0.25	1.98	4.500000	1.03
23	2.370000	0.26	1.46	3.930000	1.09
24	2.610000	0.28	1.66	3.520000	1.12
25	2.680000	0.47	1.92	3.580000	1.13
26	1.937983	0.34	1.45	4.800000	0.92
27	2.190000	0.27	1.35	3.950000	1.02
28	2.970000	0.37	1.76	4.500000	1.25
29	2.330000	0.26	1.98	4.700000	1.04
30	3.250000	0.29	2.38	5.700000	1.19
33	2.740000	0.50	1.35	5.400000	1.25
34	2.530000	0.29	1.54	4.200000	1.10
35	1.937983	0.26	1.86	5.100000	1.04
36	2.680000	0.34	1.36	4.600000	1.09
..	...	...	...	...	...
143	1.937983	0.47	1.02	4.400000	0.91
144	1.937983	0.29	1.14	8.210000	0.65
145	0.550000	0.43	1.30	4.000000	0.60
146	0.340000	0.40	0.68	4.900000	0.58
148	0.760000	0.45	1.25	8.420000	0.55
150	1.570000	0.22	1.25	8.600000	0.59
151	1.937983	0.24	1.26	10.800000	0.48
152	1.280000	0.26	1.56	7.100000	0.61
153	0.830000	0.61	1.87	10.520000	0.56
154	0.580000	0.53	1.40	7.600000	0.58
155	0.630000	0.61	1.55	7.900000	0.60
156	0.830000	0.48	1.56	9.010000	0.57
157	0.580000	0.63	1.14	7.500000	0.67
158	1.310000	0.53	2.70	13.000000	0.57
159	1.937983	0.52	2.29	11.750000	0.57
160	0.920000	0.50	1.04	7.650000	0.56
161	0.560000	0.50	0.80	5.880000	0.96
162	1.937983	0.60	0.96	5.580000	0.87
164	0.680000	0.41	1.03	9.580000	0.70
165	0.470000	0.52	1.15	6.620000	0.78
166	0.920000	0.43	1.46	10.680000	0.85
168	1.937983	0.39	1.54	8.660000	0.74
169	0.960000	0.27	1.11	8.500000	0.67
171	0.510000	0.48	0.64	9.899999	0.57
172	0.700000	0.44	1.24	9.700000	0.62
173	0.610000	0.52	1.06	7.700000	0.64
174	1.937983	0.43	1.41	7.300000	0.70
175	0.690000	0.43	1.35	10.200000	0.59
176	0.680000	0.53	1.46	9.300000	0.60
177	1.937983	0.56	1.35	9.200000	0.61

	OD280/OD315	Proline
0	3.92	1065.0

1	3.40	1050.0
2	3.17	1185.0
4	2.93	735.0
5	2.85	1450.0
6	3.58	1290.0
7	3.58	1295.0
8	2.85	1045.0
9	3.55	1045.0
10	3.17	1510.0
12	2.90	1320.0
13	2.73	1150.0
14	3.00	1547.0
17	2.57	1130.0
18	2.82	1680.0
19	3.36	845.0
20	3.71	780.0
21	3.52	770.0
23	3.63	1015.0
24	3.82	845.0
25	3.20	830.0
26	3.22	1195.0
27	2.77	1285.0
28	3.40	915.0
29	3.59	1035.0
30	2.71	1285.0
33	3.00	1235.0
34	2.87	1095.0
35	3.47	920.0
36	2.78	880.0
..	...	...
143	2.05	550.0
144	2.00	855.0
145	1.68	830.0
146	1.33	415.0
148	1.62	650.0
150	1.30	500.0
151	1.47	480.0
152	1.33	425.0
153	1.51	675.0
154	1.55	640.0
155	1.48	725.0
156	1.64	480.0
157	1.73	880.0
158	1.96	660.0
159	1.78	620.0
160	1.58	520.0
161	1.82	680.0
162	2.11	570.0

```
164          1.68    615.0
165          1.75    520.0
166          1.56    695.0
168          1.80    750.0
169          1.92    630.0
171          1.63    470.0
172          1.71    660.0
173          1.74    740.0
174          1.56    750.0
175          1.56    835.0
176          1.62    840.0
177          1.60    560.0

[149 rows x 13 columns]
```

In [58]:
```
wine_modified.shape
```

Out[58]:
```
(149, 13)
```

2 Decision Trees

Question 4: Train Decision Tree model on train data for criterions = {'gini', 'entropy'} and report the accuracies on the validation data. Select the best criterion and report the accuracy on the test data. (1 mark)

In [38]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import pydotplus
import pandas as pd
import numpy as np
```

In [61]:
```
# Load the wine train dataset
train_data = pd.read_csv('wine_train_data.csv')
train_label = pd.read_csv('wine_train_labels.csv')

# Load the wine validation dataset
val_data = pd.read_csv('wine_val_data.csv')
val_label = pd.read_csv('wine_val_labels.csv')

# Load the wine test dataset
test_data = pd.read_csv('wine_test_data.csv')
test_label = pd.read_csv('wine_test_labels.csv')
```

In [67]:
```
# Define Model for criterion='entropy'
clf_entropy = DecisionTreeClassifier(criterion='entropy')
# Train
clf_entropy.fit(train_data, train_label)
# Validate
val_pred = clf_entropy.predict(val_data)
```

```
    # Evaluate
    train_pred = clf_entropy.predict(train_data)
    print ('Train accuracy = ' + str(np.sum(train_pred == train_label['class'])*1.0/len(t
    print ('Validation accuracy = ' + str(np.sum(val_pred == val_label['class'])*1.0/len(v
```

```
Train accuracy = 1.0
Validation accuracy = 0.9743589743589743
```

```
In [66]: # Define Model for criterion='gini'
         clf_gini = DecisionTreeClassifier(criterion='gini')
         # Train
         clf_gini.fit(train_data, train_label)
         # Validate
         val_pred = clf_gini.predict(val_data)
         # Evaluate
         train_pred = clf_gini.predict(train_data)
         print ('Train accuracy = ' + str(np.sum(train_pred == train_label['class'])*1.0/len(t
         print ('Validation accuracy = ' + str(np.sum(val_pred == val_label['class'])*1.0/len(v
```

```
Train accuracy = 1.0
Validation accuracy = 0.9487179487179487
```

#Based on above validation accuracies, entropy is chosen as the best criterion

```
In [71]: #Training the model again for Train+Validation data
         clf = DecisionTreeClassifier(criterion='entropy')
         # Train
         combined_train_data = pd.concat([train_data, val_data])
         combined_train_label = pd.concat([train_label, val_label])
         clf.fit(combined_train_data, combined_train_label)

         # Predict
         test_pred = clf.predict(test_data)
         print ('Test accuracy = ' + str(np.sum(test_pred == test_label['class'])*1.0/len(test_
```

```
Test accuracy = 0.7692307692307693
```

Question 5: Use the criterion selected above to train Decision Tree model on train data for min samples split={2,5,10,20} and report the accuracies on the validation data. Select the best parameter and report the accuracy on the test data. (2 marks)

```
In [75]: #Min Sample Split
         for i in [2,5,10,20]:
             clf = DecisionTreeClassifier(criterion='entropy', min_samples_split=i)
             # Train
             clf.fit(train_data, train_label)
```

```
        # Validate
        val_pred = clf.predict(val_data)
        #Evaluate
        print ('Validation accuracy on mim_samples_split: ',i,' = ' + str(np.sum(val_pred
```

```
Validation accuracy on mim_samples_split:  2  = 0.9487179487179487
Validation accuracy on mim_samples_split:  5  = 0.9743589743589743
Validation accuracy on mim_samples_split:  10  = 0.9230769230769231
Validation accuracy on mim_samples_split:  20  = 0.9230769230769231
```

#Based on above validation accuracies, 5 is chosen as the best parameter

In [77]:
```
#Training the model again for Train+Validation data
clf = DecisionTreeClassifier(criterion='entropy', min_samples_split=5)
# Train
combined_train_data = pd.concat([train_data, val_data])
combined_train_label = pd.concat([train_label, val_label])
clf.fit(combined_train_data, combined_train_label)

# Predict
test_pred = clf.predict(test_data)
print ('Test accuracy = ' + str(np.sum(test_pred == test_label['class'])*1.0/len(test_
```

```
Test accuracy = 0.8205128205128205
```

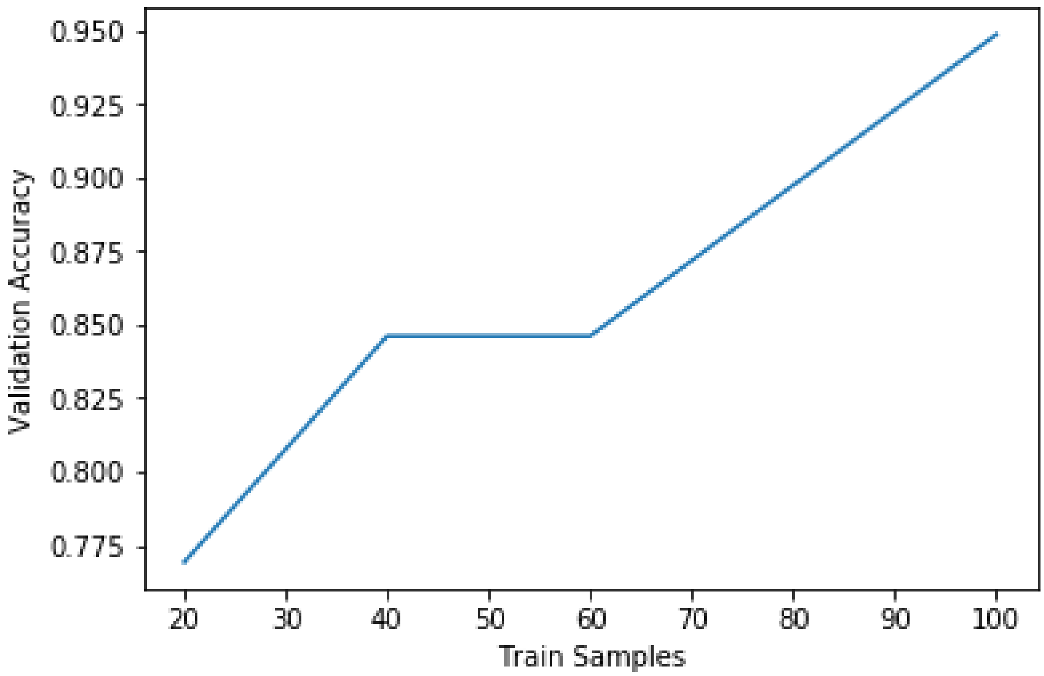
Question 6: Use the parameters selected above (Q4 and Q5) to train Decision Tree model using the first 20, 40, 60, 80 and 100 samples from train data. Keep the validation set unchanged during this analysis. Report and plot the accuracies on the validation data. (2 marks)

In [88]:
```
samples = [20,40,60,80,100]
val_accuracy = []
for i in samples:
    clf = DecisionTreeClassifier(criterion='entropy', min_samples_split=5)
    # Train
    clf.fit(train_data[0:i], train_label[0:i])
    # Validate
    val_pred = clf.predict(val_data)
    #Evaluate
    val_accuracy.append(np.sum(val_pred == val_label['class'])*1.0/len(val_data))
    print ('Train Samples: ',i,' Validation accuracy = ' + str(np.sum(val_pred == val_
```

```
Train Samples:  20  Validation accuracy = 0.7692307692307693
Train Samples:  40  Validation accuracy = 0.8461538461538461
Train Samples:  60  Validation accuracy = 0.8461538461538461
Train Samples:  80  Validation accuracy = 0.8974358974358975
Train Samples:  100  Validation accuracy = 0.9487179487179487
```

```
In [89]: import matplotlib.pyplot as plt
         plt.plot(samples, val_accuracy)
         plt.xlabel("Train Samples")
         plt.ylabel("Validation Accuracy")
```

```
Out[89]: Text(0,0.5,'Validation Accuracy')
```



3 Nearest Neighbor

Normalize Data: Normalize features such that for each feature the mean is 0 and the standard deviation is 1 in the train+validation data. Use the normal- izing factors calculated on train+validation data to modify the values in train, validation and test data.

```
In [90]: # Load the wine train dataset
         train_data = pd.read_csv('wine_train_data.csv')
         train_label = pd.read_csv('wine_train_labels.csv')

         # Load the wine validation dataset
         val_data = pd.read_csv('wine_val_data.csv')
         val_label = pd.read_csv('wine_val_labels.csv')

         # Load the wine test dataset
         test_data = pd.read_csv('wine_test_data.csv')
         test_label = pd.read_csv('wine_test_labels.csv')
```

In [91]:
```
train_val_data = pd.concat([train_data, val_data])
train_val_label = pd.concat([train_label, val_label])
```

In [92]:
```
mean = train_val_data.mean()
std = train_val_data.std()
normalized_train_val_data = (train_val_data-mean)/std

new_train_data = (train_data-mean)/std
new_val_data = (val_data-mean)/std
new_test_data = (test_data-mean)/std
```

Question 7: Train k-nn model on train + validation data and report accuracy on test data. Use Euclidean distance and k=3. (1 mark)

In [93]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [94]:
```
clf = KNeighborsClassifier(3, p=2)
clf.fit(normalized_train_val_data, train_val_label.values.ravel())

predictions = clf.predict(new_test_data)
print ('accuracy = ' + str(np.sum(predictions == test_label['class'])/(len(test_label
```

```
accuracy = 0.8717948717948718
```

Question 8: Train the model on train data for distance metrics defined by l1, linf, l2. Report the accuracies on the validation data. Select the best metric and report the accuracy on the test data for the selected metric. Use k=3. (1 mark)

In [95]:
```
clf = KNeighborsClassifier(3, p=1)
clf.fit(new_train_data, train_label.values.ravel())

val_predictions_l1 = clf.predict(new_val_data)
print ('accuracy = ' + str(np.sum(val_predictions_l1 == val_label['class'])/(len(val_l
```

```
accuracy = 0.9487179487179487
```

In [96]:
```
clf = KNeighborsClassifier(3, p=2)
clf.fit(new_train_data, train_label.values.ravel())

val_predictions_l2 = clf.predict(new_val_data)
print ('accuracy = ' + str(np.sum(val_predictions_l2 == val_label['class'])/(len(val_l
```

```
accuracy = 0.9230769230769231
```

In [97]:
```
clf = KNeighborsClassifier(3, p=np.inf)
clf.fit(new_train_data, train_label.values.ravel())

val_predictions_linf = clf.predict(new_val_data)
print ('accuracy = ' + str(np.sum(val_predictions_linf == val_label['class'])/(len(val
```

```
accuracy = 0.9230769230769231
```

```
In [98]: #Best metric is p=1 -> Manhattan distance
         clf = KNeighborsClassifier(3, p=1)
         clf.fit(normalized_train_val_data, train_val_label.values.ravel())

         predictions = clf.predict(new_test_data)
         print ('accuracy = ' + str(np.sum(predictions == test_label['class'])/(len(test_label
```

```
accuracy = 0.9743589743589743
```

Question 9: Train the k-nn model on train data for k=1,3,5,7,9. Report and plot the accuracies on the validation data. Select the best 'k' value and report the accuracy on the test data for the selected 'k'. Use Euclidean distance. (2 marks)

```
In [30]: K = [1,3,5,7,9]
         for k in K:
             clf = KNeighborsClassifier(k, p=2)
             clf.fit(new_train_data, train_label.values.ravel())

             predictions_val = clf.predict(new_val_data)
             print ('For k=',k,'accuracy = ' + str(np.sum(predictions_val == val_label['class']
```

```
For k= 1 accuracy = 0.9487179487179487
For k= 3 accuracy = 0.9230769230769231
For k= 5 accuracy = 0.9487179487179487
For k= 7 accuracy = 0.9743589743589743
For k= 9 accuracy = 0.9487179487179487
```

```
In [100]: #Best k=7 based on above accuracies
          clf = KNeighborsClassifier(7, p=2)
          clf.fit(normalized_train_val_data, train_val_label.values.ravel())

          predictions_test = clf.predict(new_test_data)
          print ('accuracy = ' + str(np.sum(predictions_test == test_label['class'])/(len(test_
```

```
accuracy = 0.9230769230769231
```

Question 10: Instead of using full train data, train the model using the first 20, 40, 60, 80 and 100 data samples from train data. Keep the validation set unchanged during this analysis. Report and plot the accuracies on the validation data. Use Euclidean distance and k=3. Note: Don't shuffle the data and use only the 'first n samples', otherwise your answers may differ. (2 marks)

```
In [48]: samples = [20,40,60,80,100]
         val_accuracy = []
```

```
    for i in samples:
        clf = KNeighborsClassifier(3, p=2)
        clf.fit(new_train_data[0:i], train_label[0:i].values.ravel())

        val_pred = clf.predict(new_val_data)

        val_accuracy.append(np.sum(val_pred == val_label['class'])*1.0/len(val_data))
        print ('Train Samples: ',i,' Validation accuracy = ' + str(np.sum(val_pred == val_
```

```
Train Samples:  20  Validation accuracy = 0.9487179487179487
Train Samples:  40  Validation accuracy = 1.0
Train Samples:  60  Validation accuracy = 1.0
Train Samples:  80  Validation accuracy = 1.0
Train Samples:  100  Validation accuracy = 0.9230769230769231
```

In [49]:
```
import matplotlib.pyplot as plt
plt.plot(samples, val_accuracy)
plt.xlabel("Train Samples")
plt.ylabel("Validation Accuracy")
```

Out[49]: Text(0,0.5,'Validation Accuracy')