public-schools

April 28, 2018

The two models that I have chosen for this projects are : 1) Linear Regression 2) Knn Neighbor Classifier The dataset I used is a dataset on public schools in the US. It contains Average SAT math score, % Economically Disadvantaged Students, Class Size, etc for its columns.

I use the linear regression model to predict the Average SAT math score from the % Economically Disadvantaged students in the school. I found that schools which have 10% more of economically disadvantaged students, will on average have a 27.8 lower average SAT math score compared to other schools

```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib
        %matplotlib inline
        import matplotlib.pyplot as plt
        plt.style.use('fivethirtyeight')
        import warnings
        warnings.simplefilter('ignore', FutureWarning)
        from matplotlib import patches
        from ipywidgets import interact, interactive, fixed
        import ipywidgets as widgets
        from sklearn.linear_model import LinearRegression
```

```
In [3]: data = pd.read_csv("MA_Public_Schools_2017.csv")
        data1 = pd.read_csv("MA_Public_Schools_datadict.csv")
        data.describe(include='all')
        list(data)
```

```
Out[3]: ['School Code',
         'School Name',
         'School Type',
         'Function',
         'Contact Name',
         'Address 1',
         'Address 2',
         'Town',
         'State',
         'Zip',
         'Phone',
         'Fax',
```

```
'Grade',
'District Name',
'District Code',
'PK_Enrollment',
'K_Enrollment',
'1_Enrollment',
'2_Enrollment',
'3_Enrollment',
'4_Enrollment',
'5_Enrollment',
'6_Enrollment',
'7_Enrollment',
'8_Enrollment',
'9_Enrollment',
'10_Enrollment',
'11_Enrollment',
'12_Enrollment',
'SP_Enrollment',
'TOTAL_Enrollment',
'First Language Not English',
'% First Language Not English',
'English Language Learner',
'% English Language Learner',
'Students With Disabilities',
'% Students With Disabilities',
'High Needs',
'% High Needs',
'Economically Disadvantaged',
'% Economically Disadvantaged',
'% African American',
'% Asian',
'% Hispanic',
'% White',
'% Native American',
'% Native Hawaiian, Pacific Islander',
'% Multi-Race, Non-Hispanic',
'% Males',
'% Females',
'Total # of Classes',
'Average Class Size',
'Number of Students',
'Salary Totals',
'Average Salary',
'FTE Count',
'In-District Expenditures',
'Total In-district FTEs',
'Average In-District Expenditures per Pupil',
'Total Expenditures',
```

```
'Total Pupil FTEs',
'Average Expenditures per Pupil',
'# in Cohort',
'% Graduated',
'% Still in School',
'% Non-Grad Completers',
'% GED',
'% Dropped Out',
'% Permanently Excluded',
'High School Graduates (#)',
'Attending Coll./Univ. (#)',
'% Attending College',
'% Private Two-Year',
'% Private Four-Year',
'% Public Two-Year',
'% Public Four-Year',
'% MA Community College',
'% MA State University',
'% UMass',
'AP_Test Takers',
'AP_Tests Taken',
'AP_One Test',
'AP_Two Tests',
'AP_Three Tests',
'AP_Four Tests',
'AP_Five or More Tests',
'AP_Score=1',
'AP_Score=2',
'AP_Score=3',
'AP_Score=4',
'AP_Score=5',
'% AP_Score 1-2',
'% AP_Score 3-5',
'SAT_Tests Taken',
'Average SAT_Reading',
'Average SAT_Writing',
'Average SAT_Math',
'MCAS_3rdGrade_Math_P+A #',
'% MCAS_3rdGrade_Math_P+A',
'MCAS_3rdGrade_Math_A #',
'% MCAS_3rdGrade_Math_A',
'MCAS_3rdGrade_Math_P #',
'% MCAS_3rdGrade_Math_P',
'MCAS_3rdGrade_Math_NI #',
'% MCAS_3rdGrade_Math_NI',
'MCAS_3rdGrade_Math_W/F #',
'% MCAS_3rdGrade_Math_W/F',
'MCAS_3rdGrade_Math_Stud. Incl. #',
```

```
'MCAS_3rdGrade_Math_CPI',
'MCAS_3rdGrade_Math_SGP',
'MCAS_3rdGrade_Math_Incl. in SGP(#)',
'MCAS_4thGrade_Math_P+A #',
'% MCAS_4thGrade_Math_P+A',
'MCAS_4thGrade_Math_A #',
'% MCAS_4thGrade_Math_A',
'MCAS_4thGrade_Math_P #',
'% MCAS_4thGrade_Math_P',
'MCAS_4thGrade_Math_NI #',
'% MCAS_4thGrade_Math_NI',
'MCAS_4thGrade_Math_W/F #',
'% MCAS_4thGrade_Math_W/F',
'MCAS_4thGrade_Math_Stud. Incl. #',
'MCAS_4thGrade_Math_CPI',
'MCAS_4thGrade_Math_SGP',
'MCAS_4thGrade_Math_Incl. in SGP(#)',
'MCAS_5thGrade_Math_P+A #',
'% MCAS_5thGrade_Math_P+A',
'MCAS_5thGrade_Math_A #',
'% MCAS_5thGrade_Math_A',
'MCAS_5thGrade_Math_P #',
'% MCAS_5thGrade_Math_P',
'MCAS_5thGrade_Math_NI #',
'% MCAS_5thGrade_Math_NI',
'MCAS_5thGrade_Math_W/F #',
'% MCAS_5thGrade_Math_W/F',
'MCAS_5thGrade_Math_Stud. Incl. #',
'MCAS_5thGrade_Math_CPI',
'MCAS_5thGrade_Math_SGP',
'MCAS_5thGrade_Math_Incl. in SGP(#)',
'MCAS_6thGrade_Math_P+A #',
'% MCAS_6thGrade_Math_P+A',
'MCAS_6thGrade_Math_A #',
'% MCAS_6thGrade_Math_A',
'MCAS_6thGrade_Math_P #',
'% MCAS_6thGrade_Math_P',
'MCAS_6thGrade_Math_NI #',
'% MCAS_6thGrade_Math_NI',
'MCAS_6thGrade_Math_W/F #',
'% MCAS_6thGrade_Math_W/F',
'MCAS_6thGrade_Math_Stud. Incl. #',
'MCAS_6thGrade_Math_CPI',
'MCAS_6thGrade_Math_SGP',
'MCAS_6thGrade_Math_Incl. in SGP(#)',
'MCAS_7thGrade_Math_P+A #',
'% MCAS_7thGrade_Math_P+A',
'MCAS_7thGrade_Math_A #',
```

```
'% MCAS_7thGrade_Math_A',
'MCAS_7thGrade_Math_P #',
'% MCAS_7thGrade_Math_P',
'MCAS_7thGrade_Math_NI #',
'% MCAS_7thGrade_Math_NI',
'MCAS_7thGrade_Math_W/F #',
'% MCAS_7thGrade_Math_W/F',
'MCAS_7thGrade_Math_Stud. Incl. #',
'MCAS_7thGrade_Math_CPI',
'MCAS_7thGrade_Math_SGP',
'MCAS_7thGrade_Math_Incl. in SGP(#)',
'MCAS_8thGrade_Math_P+A #',
'% MCAS_8thGrade_Math_P+A',
'MCAS_8thGrade_Math_A #',
'% MCAS_8thGrade_Math_A',
'MCAS_8thGrade_Math_P #',
'% MCAS_8thGrade_Math_P',
'MCAS_8thGrade_Math_NI #',
'% MCAS_8thGrade_Math_NI',
'MCAS_8thGrade_Math_W/F #',
'% MCAS_8thGrade_Math_W/F',
'MCAS_8thGrade_Math_Stud. Incl. #',
'MCAS_8thGrade_Math_CPI',
'MCAS_8thGrade_Math_SGP',
'MCAS_8thGrade_Math_Incl. in SGP(#)',
'MCAS_10thGrade_Math_P+A #',
'% MCAS_10thGrade_Math_P+A',
'MCAS_10thGrade_Math_A #',
'% MCAS_10thGrade_Math_A',
'MCAS_10thGrade_Math_P #',
'% MCAS_10thGrade_Math_P',
'MCAS_10thGrade_Math_NI #',
'% MCAS_10thGrade_Math_NI',
'MCAS_10thGrade_Math_W/F #',
'% MCAS_10thGrade_Math_W/F',
'MCAS_10thGrade_Math_Stud. Incl. #',
'MCAS_10thGrade_Math_CPI',
'MCAS_10thGrade_Math_SGP',
'MCAS_10thGrade_Math_Incl. in SGP(#)',
'MCAS_3rdGrade_English_P+A #',
'% MCAS_3rdGrade_English_P+A',
'MCAS_3rdGrade_English_A #',
'% MCAS_3rdGrade_English_A',
'MCAS_3rdGrade_English_P #',
'% MCAS_3rdGrade_English_P',
'MCAS_3rdGrade_English_NI #',
'% MCAS_3rdGrade_English_NI',
'MCAS_3rdGrade_English_W/F #',
```

```
'% MCAS_3rdGrade_English_W/F',
'MCAS_3rdGrade_English_Stud. Incl. #',
'MCAS_3rdGrade_English_CPI',
'MCAS_3rdGrade_English_SGP',
'MCAS_3rdGrade_English_Incl. in SGP(#)',
'MCAS_4thGrade_English_P+A #',
'% MCAS_4thGrade_English_P+A',
'MCAS_4thGrade_English_A #',
'% MCAS_4thGrade_English_A',
'MCAS_4thGrade_English_P #',
'% MCAS_4thGrade_English_P',
'MCAS_4thGrade_English_NI #',
'% MCAS_4thGrade_English_NI',
'MCAS_4thGrade_English_W/F #',
'% MCAS_4thGrade_English_W/F',
'MCAS_4thGrade_English_Stud. Incl. #',
'MCAS_4thGrade_English_CPI',
'MCAS_4thGrade_English_SGP',
'MCAS_4thGrade_English_Incl. in SGP(#)',
'MCAS_5thGrade_English_P+A #',
'% MCAS_5thGrade_English_P+A',
'MCAS_5thGrade_English_A #',
'% MCAS_5thGrade_English_A',
'MCAS_5thGrade_English_P #',
'% MCAS_5thGrade_English_P',
'MCAS_5thGrade_English_NI #',
'% MCAS_5thGrade_English_NI',
'MCAS_5thGrade_English_W/F #',
'% MCAS_5thGrade_English_W/F',
'MCAS_5thGrade_English_Stud. Incl. #',
'MCAS_5thGrade_English_CPI',
'MCAS_5thGrade_English_SGP',
'MCAS_5thGrade_English_Incl. in SGP(#)',
'MCAS_6thGrade_English_P+A #',
'% MCAS_6thGrade_English_P+A',
'MCAS_6thGrade_English_A #',
'% MCAS_6thGrade_English_A',
'MCAS_6thGrade_English_P #',
'% MCAS_6thGrade_English_P',
'MCAS_6thGrade_English_NI #',
'% MCAS_6thGrade_English_NI',
'MCAS_6thGrade_English_W/F #',
'% MCAS_6thGrade_English_W/F',
'MCAS_6thGrade_English_Stud. Incl. #',
'MCAS_6thGrade_English_CPI',
'MCAS_6thGrade_English_SGP',
'MCAS_6thGrade_English_Incl. in SGP(#)',
'MCAS_7thGrade_English_P+A #',
```

```
'% MCAS_7thGrade_English_P+A',
'MCAS_7thGrade_English_A #',
'% MCAS_7thGrade_English_A',
'MCAS_7thGrade_English_P #',
'% MCAS_7thGrade_English_P',
'MCAS_7thGrade_English_NI #',
'% MCAS_7thGrade_English_NI',
'MCAS_7thGrade_English_W/F #',
'% MCAS_7thGrade_English_W/F',
'MCAS_7thGrade_English_Stud. Incl. #',
'MCAS_7thGrade_English_CPI',
'MCAS_7thGrade_English_SGP',
'MCAS_7thGrade_English_Incl. in SGP(#)',
'MCAS_8thGrade_English_P+A #',
'% MCAS_8thGrade_English_P+A',
'MCAS_8thGrade_English_A #',
'% MCAS_8thGrade_English_A',
'MCAS_8thGrade_English_P #',
'% MCAS_8thGrade_English_P',
'MCAS_8thGrade_English_NI #',
'% MCAS_8thGrade_English_NI',
'MCAS_8thGrade_English_W/F #',
'% MCAS_8thGrade_English_W/F',
'MCAS_8thGrade_English_Stud. Incl. #',
'MCAS_8thGrade_English_CPI',
'MCAS_8thGrade_English_SGP',
'MCAS_8thGrade_English_Incl. in SGP(#)',
'MCAS_10thGrade_English_P+A #',
'% MCAS_10thGrade_English_P+A',
'MCAS_10thGrade_English_A #',
'% MCAS_10thGrade_English_A',
'MCAS_10thGrade_English_P #',
'% MCAS_10thGrade_English_P',
'MCAS_10thGrade_English_NI #',
'% MCAS_10thGrade_English_NI',
'MCAS_10thGrade_English_W/F #',
'% MCAS_10thGrade_English_W/F',
'MCAS_10thGrade_English_Stud. Incl. #',
'MCAS_10thGrade_English_CPI',
'MCAS_10thGrade_English_SGP',
'MCAS_10thGrade_English_Incl. in SGP(#)',
'Accountability and Assistance Level',
'Accountability and Assistance Description',
'School Accountability Percentile (1-99)',
'Progress and Performance Index (PPI) - All Students',
'Progress and Performance Index (PPI) - High Needs Students',
'District_Accountability and Assistance Level',
'District_Accountability and Assistance Description',
```

```
'District_Progress and Performance Index (PPI) - All Students',
 'District_Progress and Performance Index (PPI) - High Needs Students']
```

In [4]:
```
table = data.loc[:, ["12_Enrollment","Average Class Size", "% Attending College", '% F
```

In [5]:
```
df_12enrollment = table["12_Enrollment"]>0
table2 = table[df_12enrollment]
students_enrolled_12grade = table2
students_enrolled_12grade
```

Out[5]:

	12_Enrollment	Average Class Size	% Attending College	\
0	92	15.8	75.8	
8	315	16.8	81.6	
16	163	16.7	72.6	
17	11	7.6	NaN	
23	462	14.7	89.3	
33	295	14.3	85.0	
43	187	14.6	87.7	
49	14	7.6	21.9	
50	392	18.1	74.3	
60	157	17.3	79.1	
64	45	13.3	87.8	
66	334	15.3	75.5	
74	206	12.3	91.5	
78	167	14.0	89.9	
84	141	18.3	70.1	
86	3	8.6	NaN	
89	290	19.2	86.2	
99	332	18.9	76.6	
106	280	17.7	76.3	
115	57	16.8	81.4	
119	88	18.1	32.6	
120	96	14.6	77.8	
121	91	31.1	NaN	
122	121	16.5	81.7	
123	68	15.3	75.5	
124	379	25.8	94.1	
125	268	17.9	92.0	
127	192	17.8	66.2	
128	9	27.0	NaN	
131	195	15.2	65.5	
...	...	...	...	
1688	282	15.3	70.2	
1695	245	14.4	85.5	
1706	303	17.3	77.1	
1707	102	14.0	46.7	
1717	398	19.7	91.0	
1721	186	12.9	82.0	

1726	86	NaN	NaN
1732	254	15.8	87.9
1745	436	16.6	62.7
1753	297	20.9	76.4
1755	338	13.8	68.8
1763	230	14.5	86.9
1767	5	12.0	NaN
1768	67	15.2	68.2
1778	278	17.0	87.5
1781	125	18.3	80.9
1792	320	16.4	76.3
1796	211	15.3	68.3
1802	75	17.6	59.3
1805	329	13.5	75.2
1822	311	14.8	54.4
1826	293	16.0	60.6
1831	45	19.3	72.4
1838	358	16.2	67.2
1845	40	13.5	NaN
1851	30	25.2	NaN
1854	70	14.9	NaN
1855	12	20.3	NaN
1856	12	18.8	NaN
1858	117	25.7	NaN

	% First Language Not English	% Students With Disabilities	High Needs \
0	5.3	9.7	130.0
8	4.6	14.1	391.0
16	2.9	17.0	154.0
17	0.0	51.6	26.0
23	9.5	16.1	377.0
33	12.4	11.1	267.0
43	11.3	13.2	170.0
49	12.7	9.5	34.0
50	10.2	14.2	593.0
60	5.7	9.6	189.0
64	7.2	16.3	111.0
66	10.9	12.2	680.0
74	9.6	16.0	202.0
78	2.4	12.0	157.0
84	3.0	19.4	234.0
86	0.0	73.5	29.0
89	16.5	5.7	181.0
99	4.1	17.7	452.0
106	6.8	14.9	367.0
115	48.0	22.0	163.0
119	78.2	7.1	184.0
120	35.8	13.4	247.0

121	39.4	17.6	149.0
122	61.0	22.0	386.0
123	100.0	4.2	376.0
124	32.7	2.1	499.0
125	41.6	2.9	622.0
127	59.3	23.8	699.0
128	55.2	100.0	29.0
131	61.9	25.3	795.0
...	...	...	...
1688	28.8	18.9	595.0
1695	18.3	10.7	211.0
1706	5.4	15.1	444.0
1707	7.6	28.3	290.0
1717	10.1	9.8	224.0
1721	13.2	17.0	177.0
1726	3.1	15.3	178.0
1732	3.4	18.1	212.0
1745	8.0	15.4	693.0
1753	1.4	13.1	304.0
1755	3.4	25.4	624.0
1763	2.4	13.0	173.0
1767	0.0	35.7	21.0
1768	3.5	21.5	138.0
1778	11.8	18.0	305.0
1781	14.2	18.1	229.0
1792	17.6	14.3	470.0
1796	44.0	25.8	718.0
1802	78.0	11.7	444.0
1805	47.9	16.5	916.0
1822	57.7	25.4	1064.0
1826	57.3	23.8	1078.0
1831	68.5	11.0	174.0
1838	40.6	12.6	792.0
1845	29.9	24.2	203.0
1851	1.0	18.1	368.0
1854	37.5	17.7	256.0
1855	16.1	31.1	172.0
1856	51.2	11.5	172.0
1858	6.2	20.4	558.0

	% Economically Disadvantaged	% Asian	% Hispanic	% White	\
0	21.5	1.5	9.1	85.8	
8	22.7	2.2	5.8	88.8	
16	14.6	1.2	4.2	90.7	
17	74.2	0.0	6.5	87.1	
23	6.3	14.5	5.0	76.3	
33	10.3	10.7	5.7	75.6	
43	10.3	9.0	9.4	77.2	

49	46.0	1.6	11.1	76.2
50	25.6	4.1	12.5	73.6
60	15.2	4.3	6.4	84.0
64	23.8	5.6	4.7	48.3
66	25.5	2.8	11.5	72.2
74	7.8	11.4	6.4	68.7
78	13.5	2.0	4.1	90.3
84	15.6	3.1	5.3	88.4
86	44.1	0.0	11.8	85.3
89	6.5	16.8	3.5	69.9
99	23.8	2.7	10.8	81.2
106	15.2	6.4	4.6	84.3
115	61.4	5.8	39.9	7.2
119	51.3	2.0	37.1	5.1
120	43.6	4.5	39.4	12.5
121	66.3	1.0	39.4	14.5
122	56.0	4.8	51.2	5.8
123	60.3	3.4	45.5	1.9
124	15.7	29.0	12.1	46.9
125	31.7	21.9	22.6	29.3
127	61.0	3.1	45.7	5.4
128	75.9	13.8	17.2	13.8
131	62.9	17.0	43.5	5.2
...	...	...	...	...
1688	38.5	6.8	14.4	73.2
1695	7.6	21.2	4.5	70.1
1706	26.6	3.2	11.1	80.6
1707	34.4	1.9	6.3	88.0
1717	3.6	20.5	1.1	74.1
1721	5.8	17.0	4.8	67.1
1726	20.5	1.7	3.3	91.0
1732	4.1	7.0	1.9	86.5
1745	24.7	4.8	6.8	81.3
1753	17.5	1.3	2.5	91.8
1755	29.1	1.1	18.3	76.0
1763	8.1	5.2	3.0	87.5
1767	60.7	0.0	0.0	92.9
1768	37.0	3.1	8.0	84.1
1778	5.6	16.4	4.1	74.1
1781	26.6	1.7	11.8	81.4
1792	22.0	6.4	11.9	73.8
1796	54.7	5.9	38.2	32.2
1802	65.0	9.8	72.4	7.0
1805	43.9	7.8	31.7	40.6
1822	63.6	8.3	47.9	21.0
1826	58.6	12.7	43.7	23.9
1831	57.1	19.7	53.5	11.8
1838	43.8	6.6	36.8	37.1

1845	53.0	1.8	22.8	3.2
1851	47.8	2.2	16.8	64.3
1854	69.5	0.0	87.5	7.0
1855	81.3	0.5	69.4	3.1
1856	28.9	11.5	21.1	29.8
1858	36.1	3.3	12.2	70.5

	...	% Native Hawaiian, Pacific Islander \
0	...	0.2
8	...	0.1
16	...	0.0
17	...	0.0
23	...	0.0
33	...	0.2
43	...	0.0
49	...	0.0
50	...	0.2
60	...	0.0
64	...	0.0
66	...	0.4
74	...	0.6
78	...	0.1
84	...	0.0
86	...	0.0
89	...	0.1
99	...	0.2
106	...	0.3
115	...	0.4
119	...	1.0
120	...	0.0
121	...	0.0
122	...	0.0
123	...	0.8
124	...	0.0
125	...	0.2
127	...	0.1
128	...	3.4
131	...	0.0
...	...	...
1688	...	0.1
1695	...	0.0
1706	...	0.2
1707	...	0.2
1717	...	0.0
1721	...	0.0
1726	...	0.0
1732	...	0.0
1745	...	0.0

1753	...	0.2
1755	...	0.0
1763	...	0.1
1767	...	0.0
1768	...	0.0
1778	...	0.0
1781	...	0.0
1792	...	0.2
1796	...	0.0
1802	...	0.0
1805	...	0.0
1822	...	0.0
1826	...	0.0
1831	...	0.0
1838	...	0.0
1845	...	0.0
1851	...	0.3
1854	...	0.3
1855	...	0.0
1856	...	0.0
1858	...	0.2

	% Multi-Race, Non-Hispanic	% Males	% Females	Number of Students	\
0	0.9	45.6	54.4	451.0	
8	1.9	52.0	48.0	1242.0	
16	2.5	53.5	46.5	621.0	
17	6.5	64.5	35.5	33.0	
23	2.2	48.9	51.1	1799.0	
33	3.7	49.1	50.9	1255.0	
43	2.2	48.8	51.2	746.0	
49	7.9	54.0	46.0	72.0	
50	3.8	54.2	45.8	1724.0	
60	2.8	48.1	51.9	776.0	
64	3.4	48.3	51.4	320.0	
66	3.7	48.7	51.3	1837.0	
74	5.2	50.6	49.4	893.0	
78	1.4	50.3	49.7	707.0	
84	1.2	54.5	45.5	748.0	
86	2.9	64.7	32.4	40.0	
89	5.2	46.9	53.1	1249.0	
99	2.2	49.8	50.2	1285.0	
106	2.0	49.9	50.1	1388.0	
115	3.1	49.3	50.7	239.0	
119	1.0	48.7	51.3	245.0	
120	3.1	36.2	63.8	433.0	
121	1.0	51.8	48.2	259.0	
122	1.8	51.2	48.8	502.0	
123	1.1	51.9	48.1	518.0	

```
124                            3.8     45.1       54.9            2429.0
125                            2.9     41.9       58.1            1690.0
127                            1.0     60.3       39.7             958.0
128                            0.0     62.1       37.9              27.0
131                            1.1     60.1       39.9             952.0
...                            ...      ...        ...               ...
1688                           1.9     52.3       47.7            1235.0
1695                           2.3     49.3       50.7            1048.0
1706                           1.9     45.3       54.7            1261.0
1707                           1.3     63.9       36.1             502.0
1717                           3.6     51.0       49.0            1680.0
1721                           4.5     50.6       49.4             726.0
1726                           2.8     55.7       44.3               NaN
1732                           1.6     47.4       52.6             977.0
1745                           2.7     52.0       48.0            2012.0
1753                           2.1     48.5       51.5            1196.0
1755                           2.8     54.6       45.4            1327.0
1763                           1.7     49.4       50.6             876.0
1767                           7.1     64.3       35.7              28.0
1768                           2.4     53.6       46.4             325.0
1778                           3.9     50.3       49.7            1223.0
1781                           2.6     49.8       50.2             737.0
1792                           1.6     49.1       50.8            1338.0
1796                           4.5     51.8       48.2             990.0
1802                           1.7     50.4       49.6             492.0
1805                           3.5     54.5       45.5            1428.0
1822                           3.1     52.1       47.9            1241.0
1826                           2.5     52.9       47.1            1268.0
1831                           3.1     55.1       44.9             250.0
1838                           3.0     42.6       57.4            1348.0
1845                           1.1     42.3       57.7             228.0
1851                           4.9     44.8       55.2             637.0
1854                           0.0     44.8       55.2             312.0
1855                           4.7     51.3       48.7             179.0
1856                           1.2     45.0       55.0             267.0
1858                           7.1     43.4       56.6            1014.0

      Average In-District Expenditures per Pupil  \
0                                       12050.39
8                                       13546.48
16                                      13001.25
17                                      13001.25
23                                      14703.35
33                                      12316.61
43                                      12867.66
49                                      12517.75
50                                      12517.75
60                                      13185.35
```

64	14046.54
66	14952.35
74	15697.93
78	12262.88
84	12852.89
86	12852.89
89	11689.27
99	12220.91
106	14737.99
115	19224.89
119	19224.89
120	19224.89
121	19224.89
122	19224.89
123	19224.89
124	19224.89
125	19224.89
127	19224.89
128	19224.89
131	19224.89
...	...
1688	12440.45
1695	13568.55
1706	13466.39
1707	13466.39
1717	12435.21
1721	21826.20
1726	12707.37
1732	15349.89
1745	12778.32
1753	11153.06
1755	18295.68
1763	14116.94
1767	13711.13
1768	13711.13
1778	12109.26
1781	12112.16
1792	14451.77
1796	13588.16
1802	13588.16
1805	13588.16
1822	13588.16
1826	13588.16
1831	13588.16
1838	13588.16
1845	NaN
1851	NaN
1854	NaN

```
1855                                          NaN
1856                                          NaN
1858                                          NaN

      Average Expenditures per Pupil  Average SAT_Reading  \
0                           13270.84                520.0
8                           14363.21                496.0
16                          13771.87                531.0
17                          13771.87                  NaN
23                          15601.70                566.0
33                          13382.77                581.0
43                          13607.92                549.0
49                          12980.40                  NaN
50                          12980.40                494.0
60                          13580.88                488.0
64                          14284.49                467.0
66                          15012.81                505.0
74                          17839.45                566.0
78                          12591.99                544.0
84                          13220.48                515.0
86                          13220.48                  NaN
89                          13028.52                587.0
99                          13188.37                517.0
106                         15434.97                521.0
115                         19246.36                377.0
119                         19246.36                  NaN
120                         19246.36                437.0
121                         19246.36                  NaN
122                         19246.36                396.0
123                         19246.36                313.0
124                         19246.36                629.0
125                         19246.36                526.0
127                         19246.36                357.0
128                         19246.36                  NaN
131                         19246.36                346.0
...                              ...                  ...
1688                        13372.42                487.0
1695                        14745.64                583.0
1706                        13622.87                527.0
1707                        13622.87                452.0
1717                        13117.80                594.0
1721                        22768.35                604.0
1726                        13589.85                  NaN
1732                        15853.36                557.0
1745                        13675.12                503.0
1753                        11703.17                499.0
1755                        18273.29                468.0
1763                        15501.99                514.0
```

1767	13855.25	NaN
1768	13855.25	471.0
1778	12800.52	587.0
1781	12547.75	481.0
1792	15158.26	495.0
1796	13901.25	452.0
1802	13901.25	354.0
1805	13901.25	468.0
1822	13901.25	400.0
1826	13901.25	440.0
1831	13901.25	435.0
1838	13901.25	432.0
1845	NaN	NaN
1851	NaN	532.0
1854	NaN	413.0
1855	NaN	NaN
1856	NaN	NaN
1858	NaN	514.0

	Average SAT_Writing	Average SAT_Math
0	498.0	516.0
8	475.0	514.0
16	518.0	534.0
17	NaN	NaN
23	562.0	581.0
33	576.0	592.0
43	530.0	576.0
49	NaN	NaN
50	490.0	504.0
60	477.0	505.0
64	451.0	481.0
66	489.0	513.0
74	550.0	572.0
78	531.0	544.0
84	480.0	518.0
86	NaN	NaN
89	584.0	610.0
99	511.0	523.0
106	516.0	540.0
115	395.0	407.0
119	NaN	NaN
120	427.0	435.0
121	NaN	NaN
122	390.0	431.0
123	305.0	347.0
124	614.0	647.0
125	527.0	557.0
127	352.0	392.0

```
128                    NaN               NaN
131                  349.0             433.0
...                    ...               ...
1688                 479.0             508.0
1695                 577.0             608.0
1706                 503.0             526.0
1707                 399.0             458.0
1717                 580.0             612.0
1721                 603.0             627.0
1726                   NaN               NaN
1732                 548.0             578.0
1745                 491.0             510.0
1753                 499.0             518.0
1755                 435.0             468.0
1763                 508.0             527.0
1767                   NaN               NaN
1768                 449.0             463.0
1778                 586.0             617.0
1781                 469.0             494.0
1792                 481.0             516.0
1796                 446.0             457.0
1802                 360.0             371.0
1805                 453.0             484.0
1822                 384.0             409.0
1826                 432.0             473.0
1831                 428.0             471.0
1838                 422.0             437.0
1845                   NaN               NaN
1851                 505.0             515.0
1854                 410.0             410.0
1855                   NaN               NaN
1856                   NaN               NaN
1858                 491.0             472.0

[394 rows x 21 columns]
```

In [21]:
```python
poor = data.loc[:, ["12_Enrollment","% Economically Disadvantaged",'Average SAT_Math'
newpoor = poor["12_Enrollment"]>0
newpoor1 = poor[newpoor]
newpoor1.replace(to_replace = [np.inf, -np.inf], value = np.nan)
newpoor1 = newpoor1.dropna(axis = 0, how ='any')
newpoor1
#np.all(np.isfinite(newpoor1))
#np.any(np.isnan(newpoor1))
```

Out[21]:
```
      12_Enrollment  % Economically Disadvantaged  Average SAT_Math  \
0                92                          21.5             516.0
8               315                          22.7             514.0
```

16	163	14.6	534.0
23	462	6.3	581.0
33	295	10.3	592.0
43	187	10.3	576.0
50	392	25.6	504.0
60	157	15.2	505.0
64	45	23.8	481.0
66	334	25.5	513.0
74	206	7.8	572.0
78	167	13.5	544.0
84	141	15.6	518.0
89	290	6.5	610.0
99	332	23.8	523.0
106	280	15.2	540.0
115	57	61.4	407.0
120	96	43.6	435.0
122	121	56.0	431.0
123	68	60.3	347.0
124	379	15.7	647.0
125	268	31.7	557.0
127	192	61.0	392.0
131	195	62.9	433.0
134	63	66.8	374.0
142	29	72.7	405.0
145	30	54.7	352.0
148	354	46.0	415.0
153	110	64.6	405.0
154	81	48.3	437.0
...	...	...	...
1659	191	3.6	619.0
1661	88	40.9	491.0
1672	355	6.5	614.0
1675	61	17.0	525.0
1679	99	14.1	508.0
1688	282	38.5	508.0
1695	245	7.6	608.0
1706	303	26.6	526.0
1707	102	34.4	458.0
1717	398	3.6	612.0
1721	186	5.8	627.0
1732	254	4.1	578.0
1745	436	24.7	510.0
1753	297	17.5	518.0
1755	338	29.1	468.0
1763	230	8.1	527.0
1768	67	37.0	463.0
1778	278	5.6	617.0
1781	125	26.6	494.0

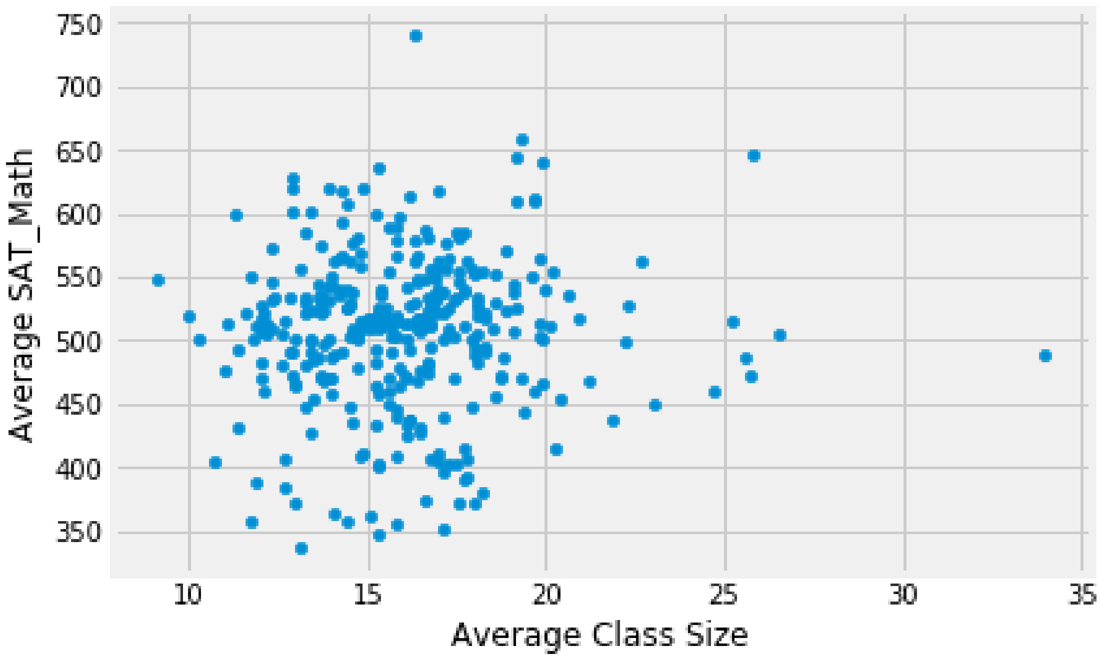
1792	320	22.0	516.0
1796	211	54.7	457.0
1802	75	65.0	371.0
1805	329	43.9	484.0
1822	311	63.6	409.0
1826	293	58.6	473.0
1831	45	57.1	471.0
1838	358	43.8	437.0
1851	30	47.8	515.0
1854	70	69.5	410.0
1858	117	36.1	472.0

	Average Class Size
0	15.8
8	16.8
16	16.7
23	14.7
33	14.3
43	14.6
50	18.1
60	17.3
64	13.3
66	15.3
74	12.3
78	14.0
84	18.3
89	19.2
99	18.9
106	17.7
115	16.8
120	14.6
122	16.5
123	15.3
124	25.8
125	17.9
127	17.8
131	15.2
134	16.6
142	10.7
145	17.1
148	17.7
153	17.0
154	21.9
...	...
1659	14.9
1661	15.7
1672	16.2
1675	14.4

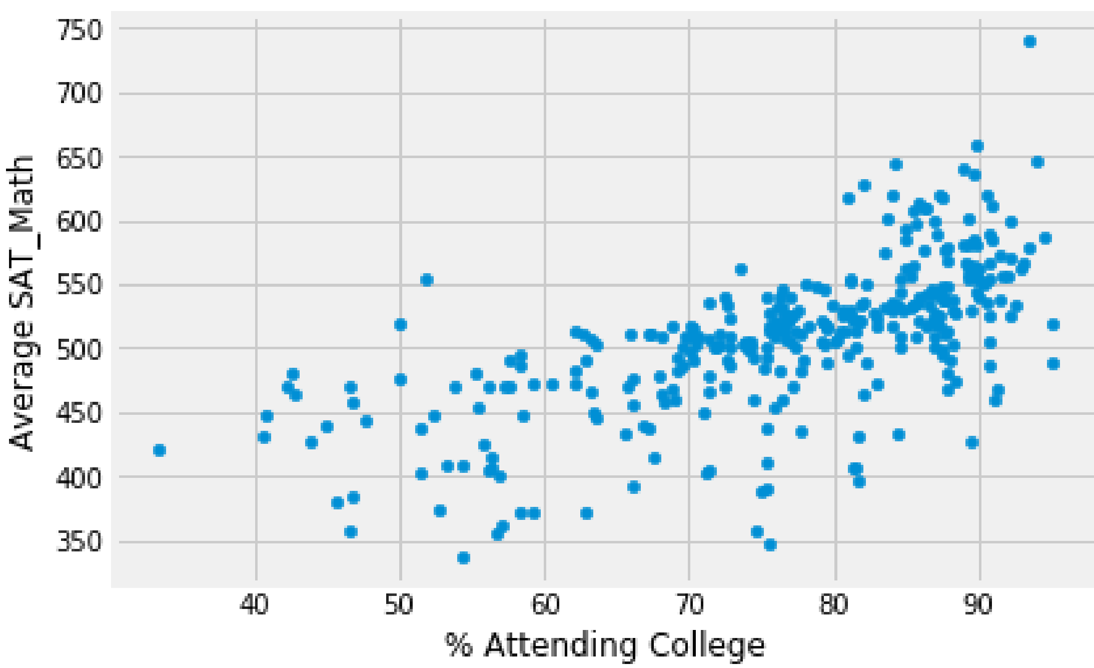
```
1679                17.3
1688                15.3
1695                14.4
1706                17.3
1707                14.0
1717                19.7
1721                12.9
1732                15.8
1745                16.6
1753                20.9
1755                13.8
1763                14.5
1768                15.2
1778                17.0
1781                18.3
1792                16.4
1796                15.3
1802                17.6
1805                13.5
1822                14.8
1826                16.0
1831                19.3
1838                16.2
1851                25.2
1854                14.9
1858                25.7

[328 rows x 4 columns]
```

df_12enrollment.describe()

```
In [7]: table2.plot.scatter("Average Class Size", 'Average SAT_Math', s=None, c=None)
        table2.plot.scatter( "% Attending College",'Average SAT_Math', s=None, c=None)
        table2.plot.scatter( '% First Language Not English','Average SAT_Math', s=None, c=None
        table2.plot.scatter( '% Students With Disabilities','Average SAT_Math', s=None, c=None
        table2.plot.scatter( "High Needs",'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Economically Disadvantaged",'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Asian" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Hispanic" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% White",'Average SAT_Math',  s=None, c=None)
        table2.plot.scatter("% Native American" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Native Hawaiian, Pacific Islander" ,'Average SAT_Math', s=None,

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x127c7510>
```

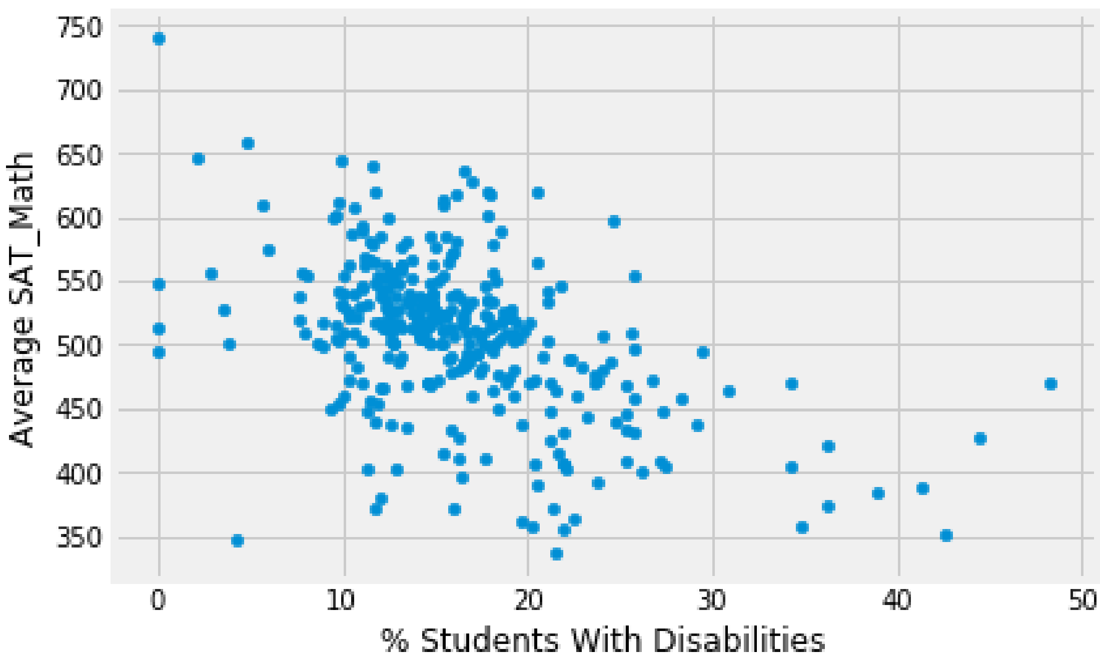


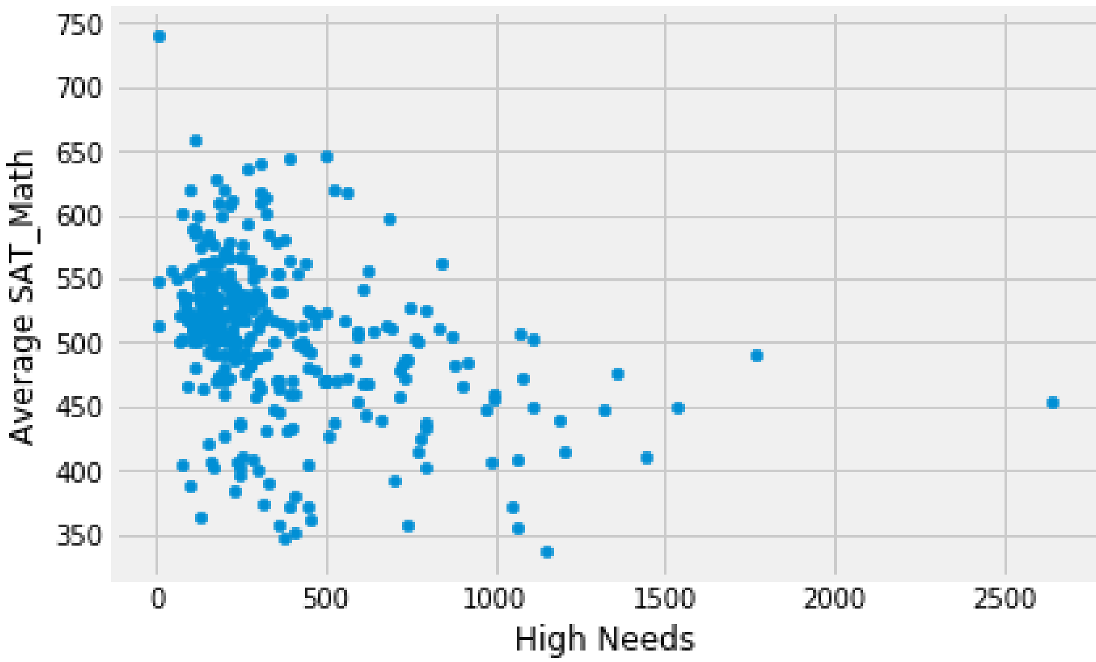
750
700
650
600
550
500
450
400
350
Average SAT_Math
10
15
20
25
30
35
Average Class Size

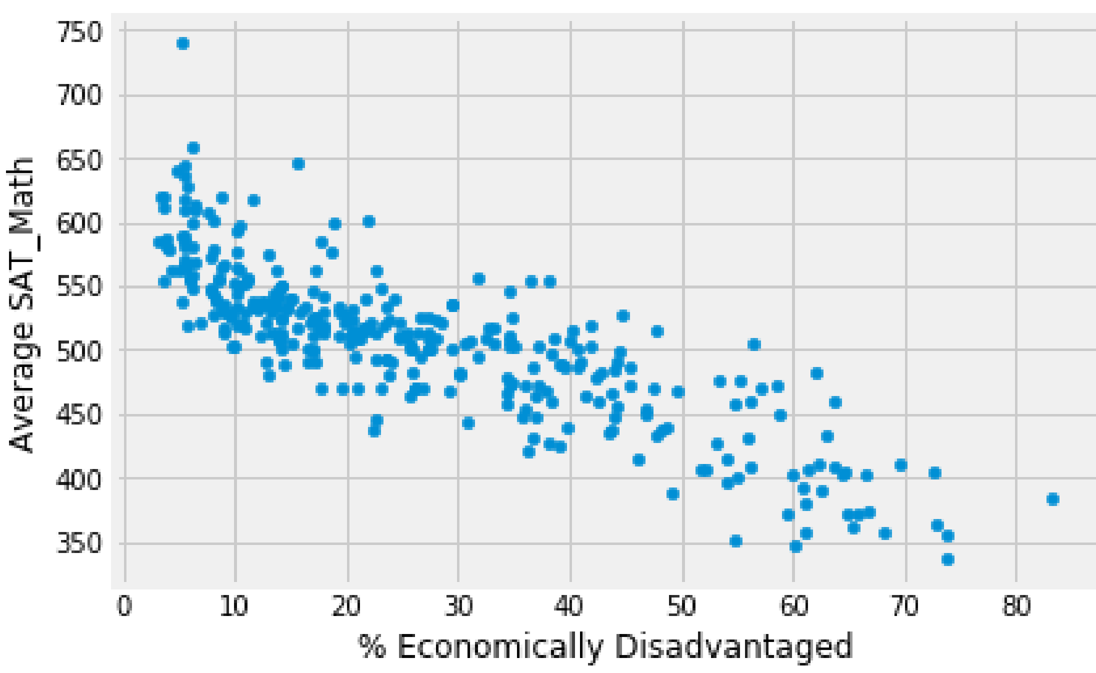


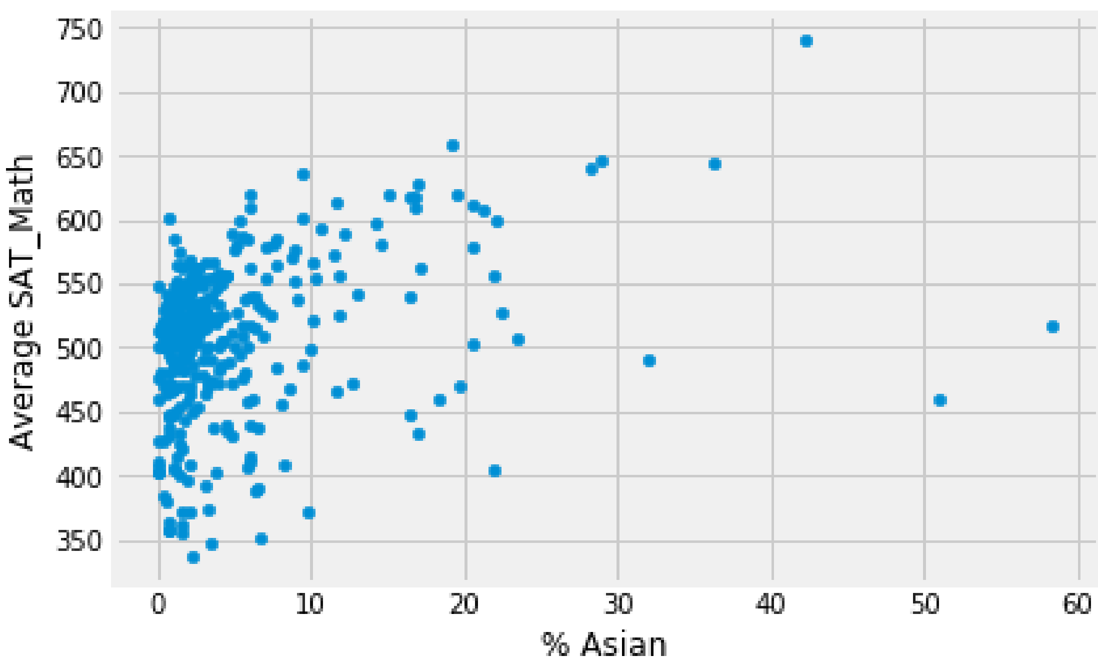
750
700
650
600
550
500
450
400
350
Average SAT_Math
40
50
60
70
80
90
% Attending College

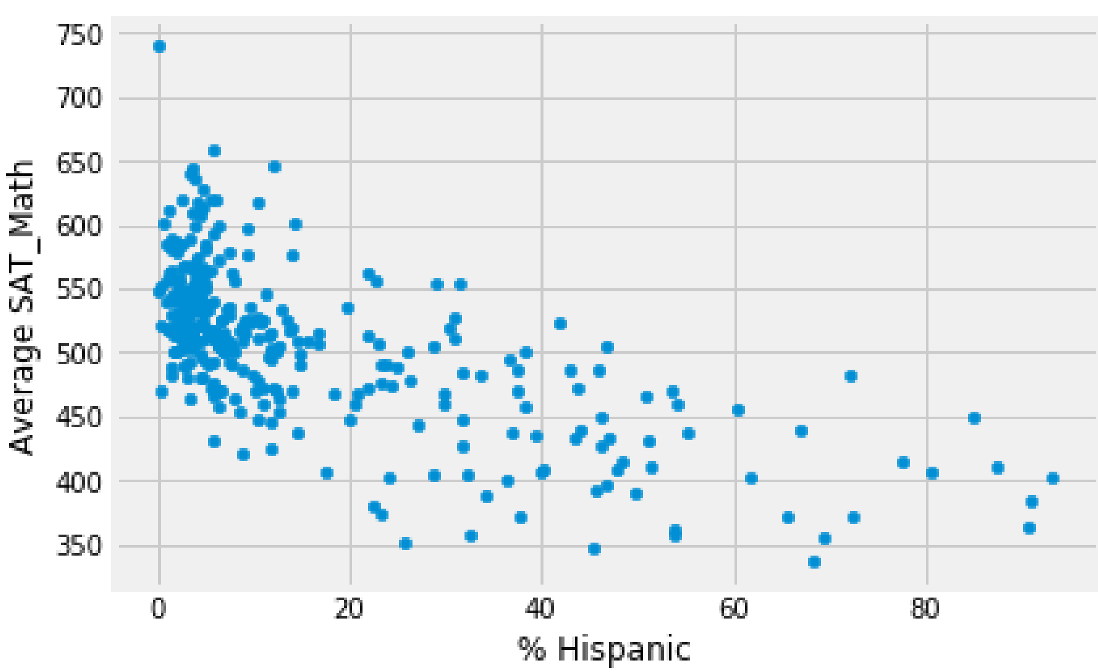

750
700
650
600
550
500
450
400
350
Average SAT_Math
0
20
40
60
80
100
% First Language Not English

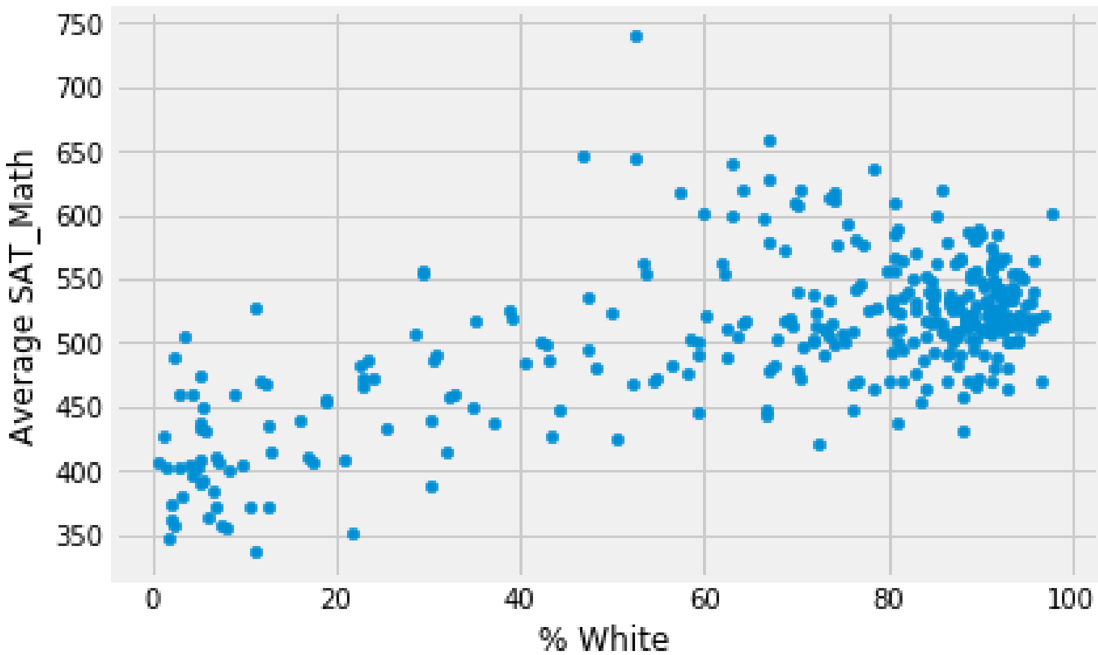

750
700
650
600
550
500
450
400
350
Average SAT_Math
0
10
20
30
40
50
% Students With Disabilities

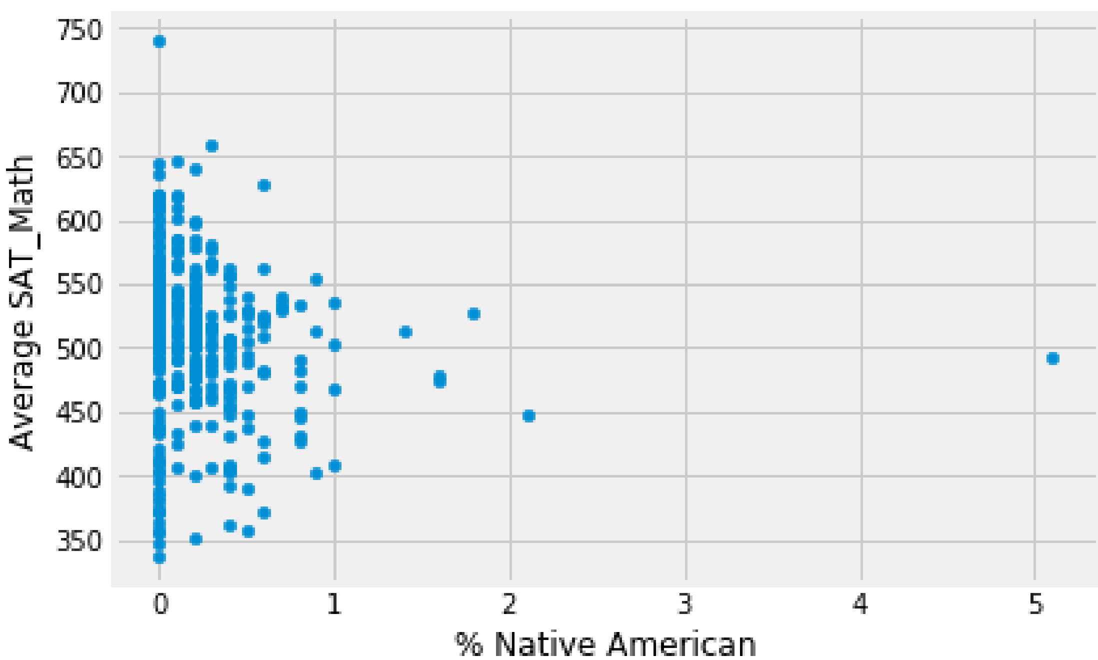

Average SAT_Math
750
700
650
600
550
500
450
400
350
0
500
1000
1500
2000
2500
High Needs

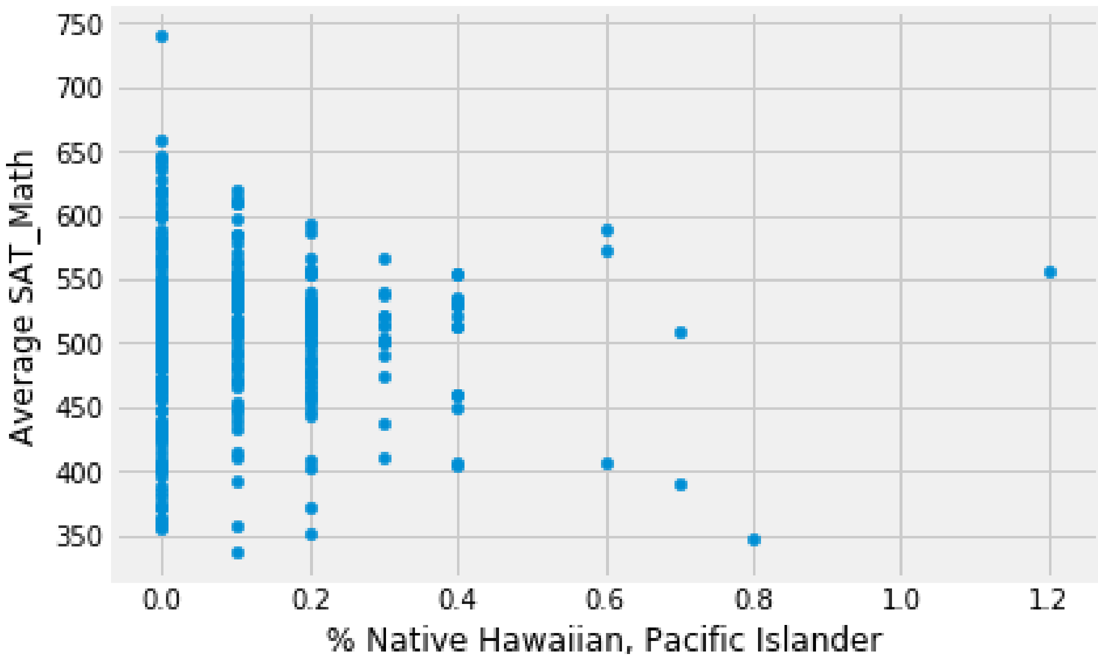

Average SAT_Math
750
700
650
600
550
500
450
400
350
0
10
20
30
40
50
60
70
80
% Economically Disadvantaged


Average SAT_Math
% Asian


Average SAT_Math
% Hispanic

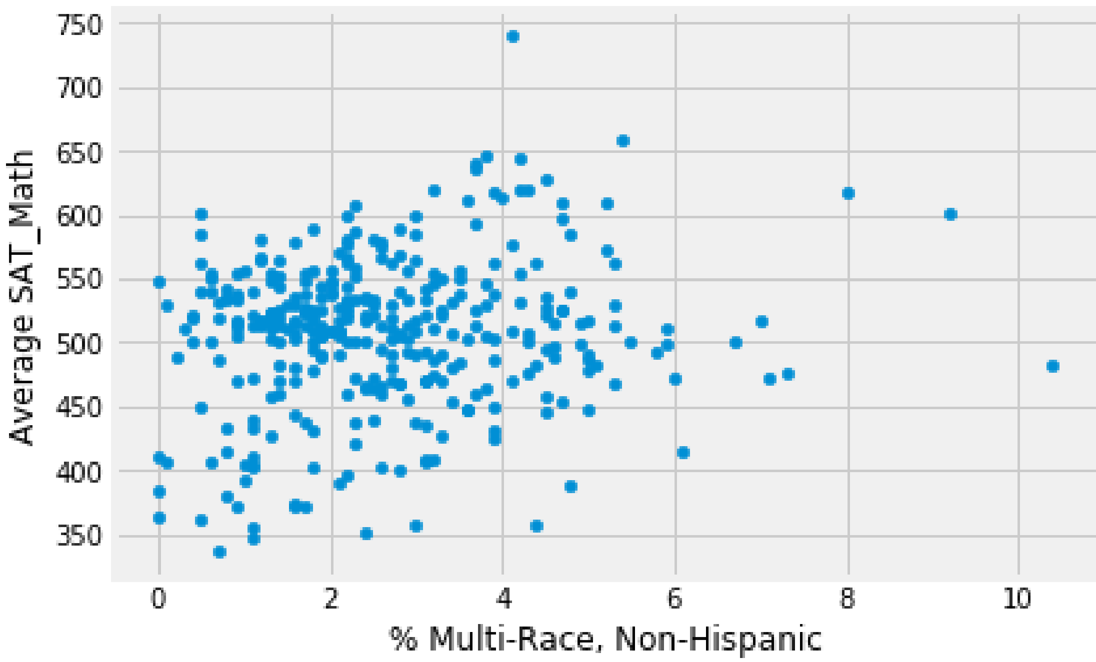

750
700
650
600
550
500
450
400
350
Average SAT_Math
0
20
40
60
80
100
% White

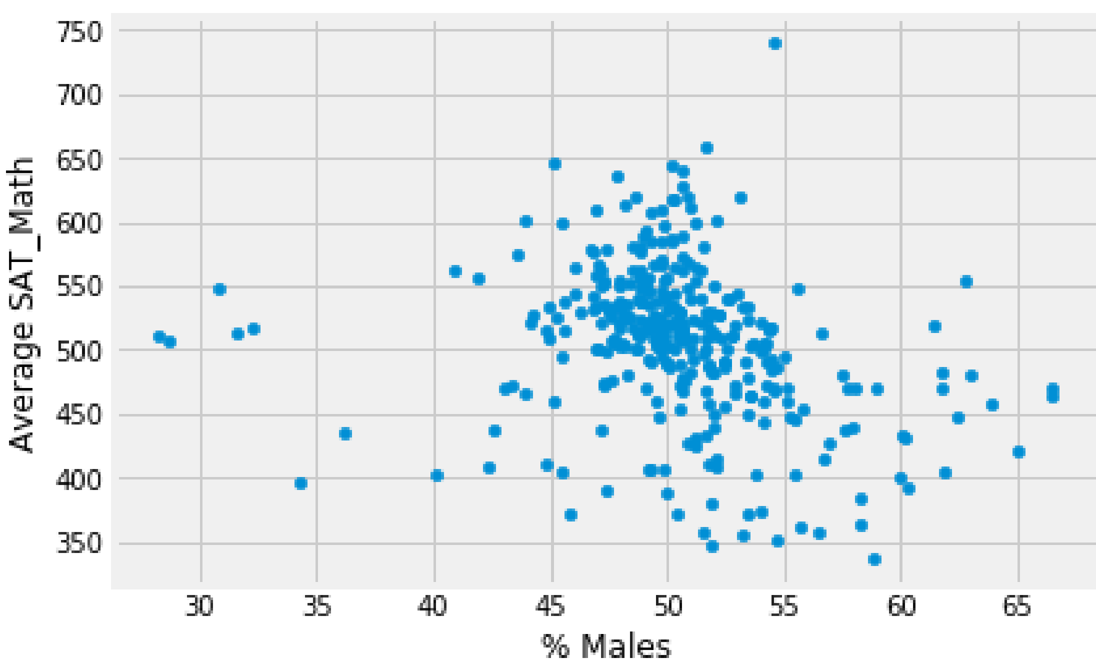

750
700
650
600
550
500
450
400
350
Average SAT_Math
0
1
2
3
4
5
% Native American

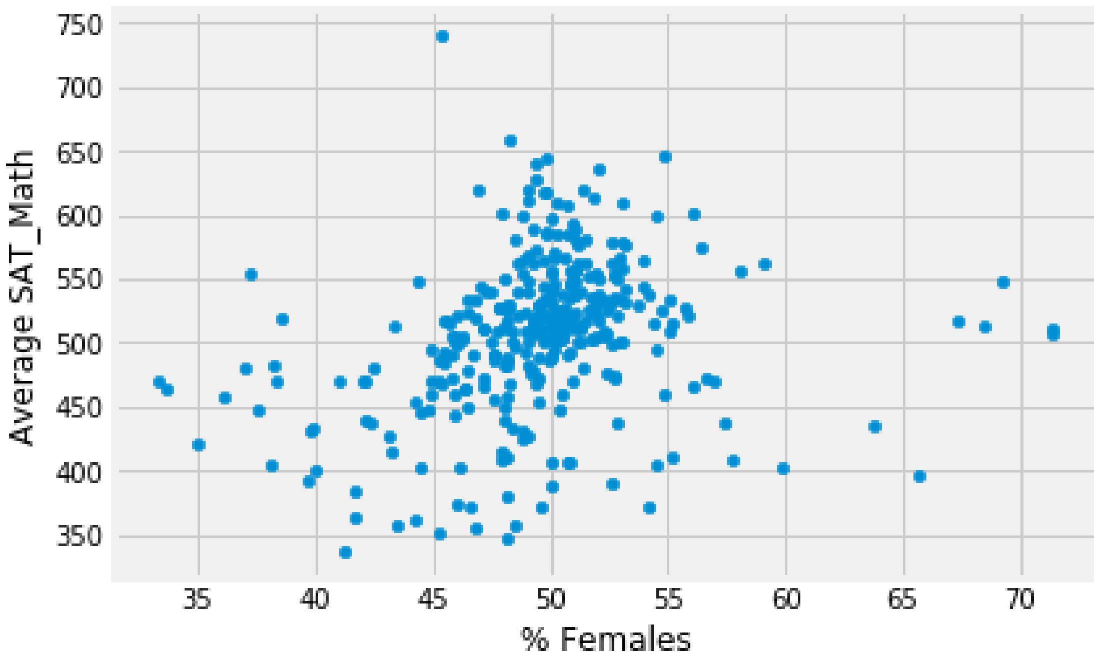


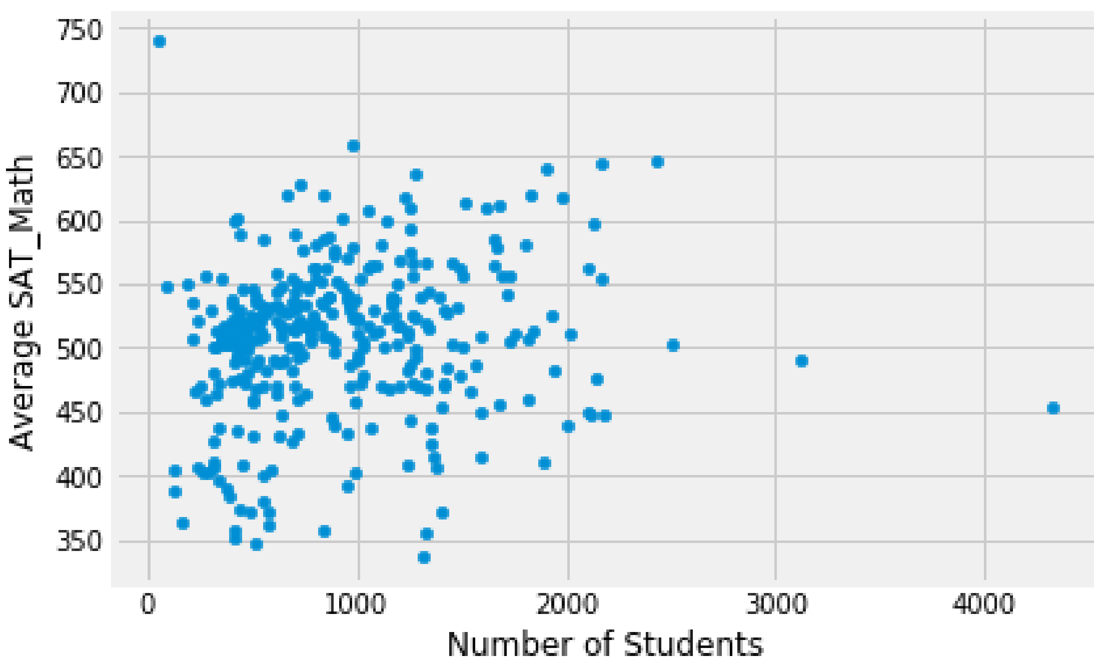

```
In [8]: table2.plot.scatter("% Multi-Race, Non-Hispanic",'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Males" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("% Females" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("Number of Students" ,'Average SAT_Math', s=None, c=None)
        table2.plot.scatter("Average In-District Expenditures per Pupil", 'Average SAT_Math', s
        table2.plot.scatter("Average Expenditures per Pupil" ,'Average SAT_Math', s=None, c=Nor
        table2.plot.scatter('Average SAT_Reading', 'Average SAT_Math', s=None, c=None)
        table2.plot.scatter('Average SAT_Writing' ,'Average SAT_Math', s=None, c=None)
```

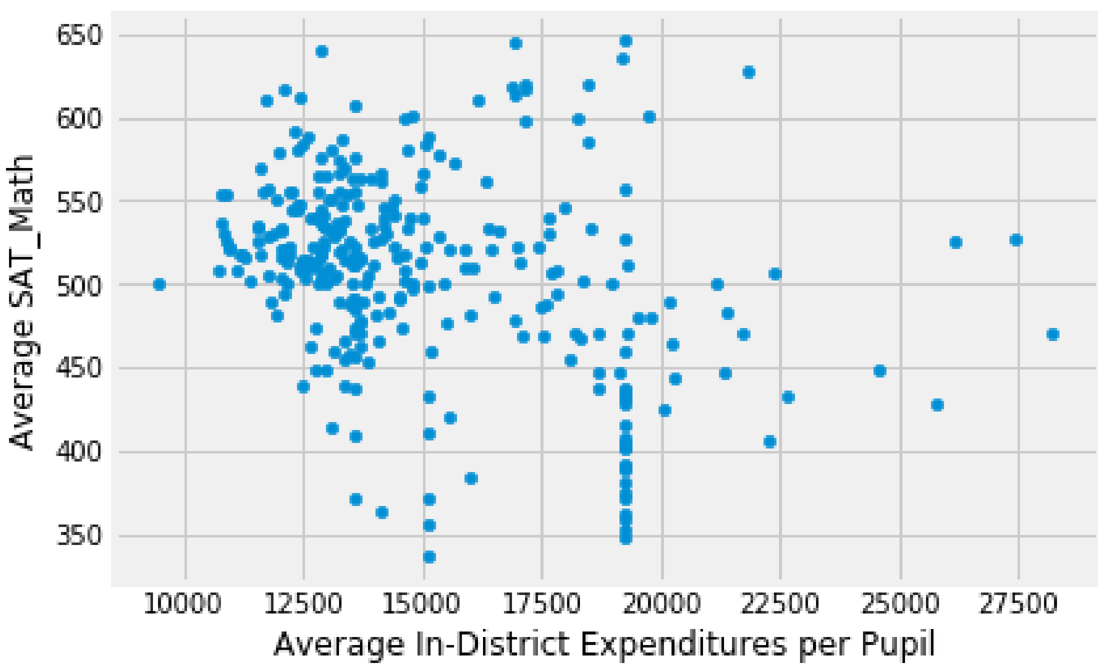
```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x126a4350>
```


750
700
650
600
550
500
450
400
350
Average SAT_Math
0
2
4
6
8
10
% Multi-Race, Non-Hispanic

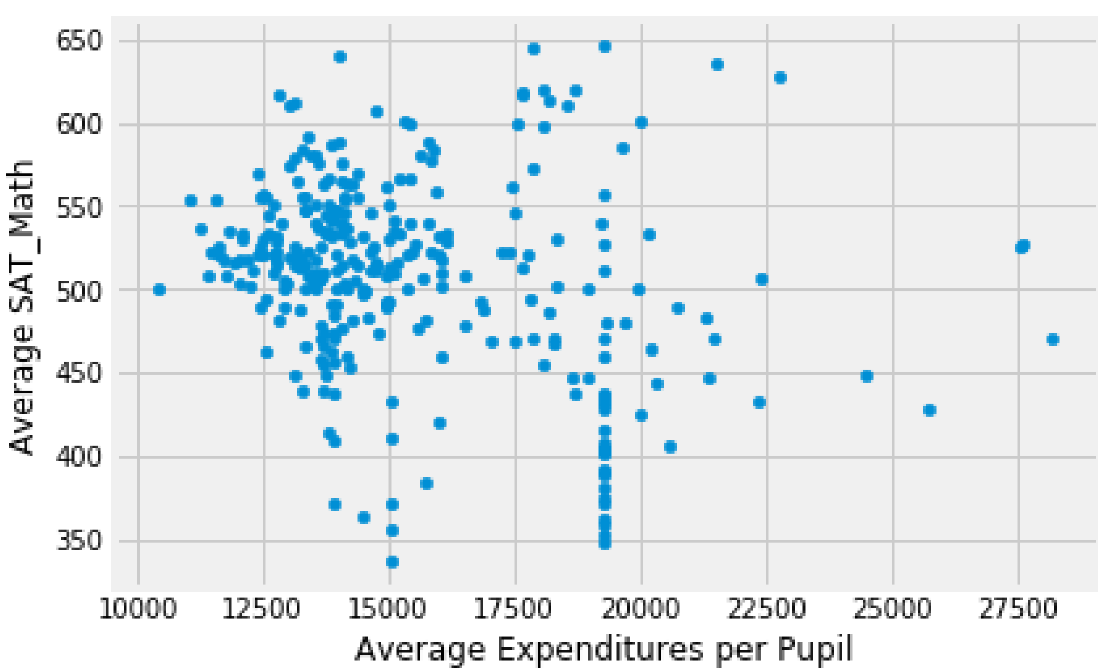

750
700
650
600
550
500
450
400
350
Average SAT_Math
30
35
40
45
50
55
60
65
% Males


Average SAT_Math
% Females
750
700
650
600
550
500
450
400
350
35
40
45
50
55
60
65
70

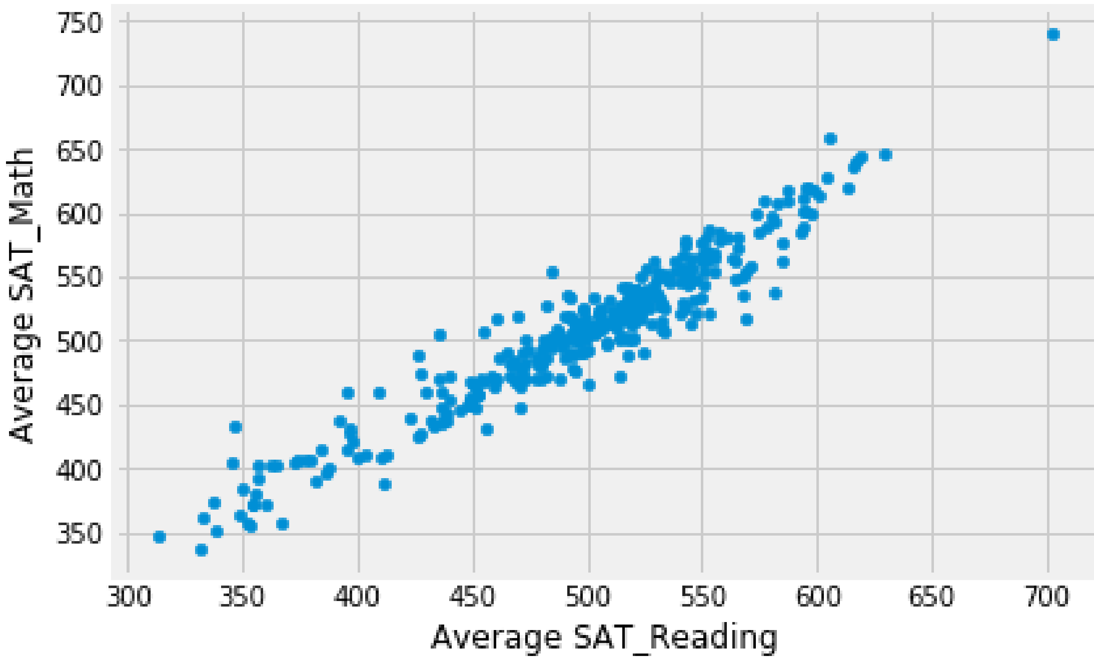

Average SAT_Math
Number of Students
750
700
650
600
550
500
450
400
350
0
1000
2000
3000
4000

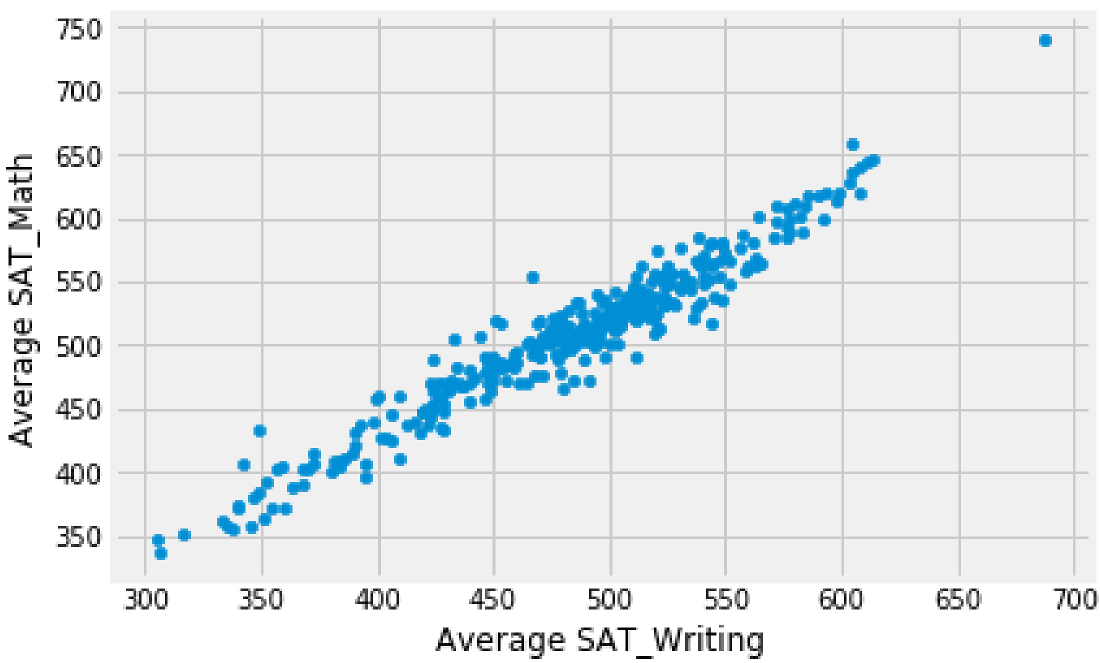


650
600
550
500
450
400
350
Average SAT_Math
10000
12500
15000
17500
20000
22500
25000
27500
Average In-District Expenditures per Pupil

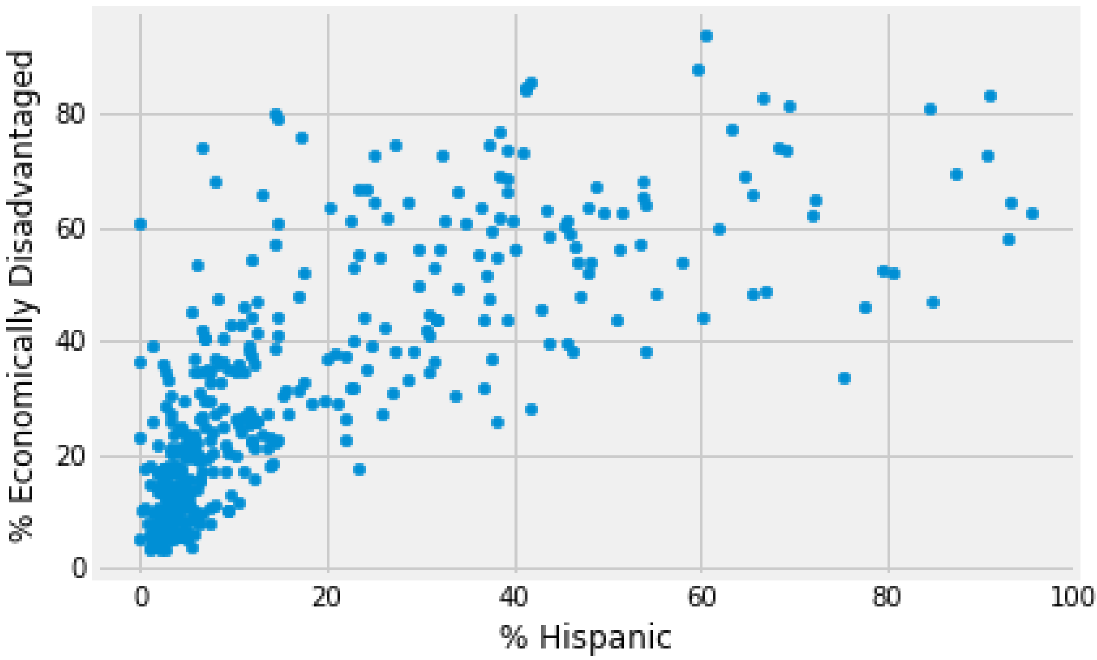


650
600
550
500
450
400
350
Average SAT_Math
10000
12500
15000
17500
20000
22500
25000
27500
Average Expenditures per Pupil






```
In [9]: table2.plot.scatter("% Hispanic", "% Economically Disadvantaged")

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x13c07d10>
```




```
In [10]: table2.replace(to_replace = [np.inf, -np.inf], value = np.nan)

         table2 = table2.dropna(axis = 0, how ='any')
         table2
         np.all(np.isfinite(newpoor1))
         np.any(np.isnan(newpoor1))

Out[10]: False

In [15]: x = newpoor1['% Economically Disadvantaged'].values[:np.newaxis]
         y = newpoor1['Average SAT_Math']

         # Reshaping
         x, y = x.reshape(-1,1), y.reshape(-1, 1)

         # Linear Regression Object
         lin_regression = LinearRegression()

         # Fitting linear model to the data
         lin_regression.fit(x,y)

         # Get slope of fitted line
         m = lin_regression.coef_

         # Get y-Intercept of the Line
```

```
b = lin_regression.intercept_

# Get Predictions for original x values
# you can also get predictions for new data
predictions = lin_regression.predict(x)

print (b)
print (m)
```

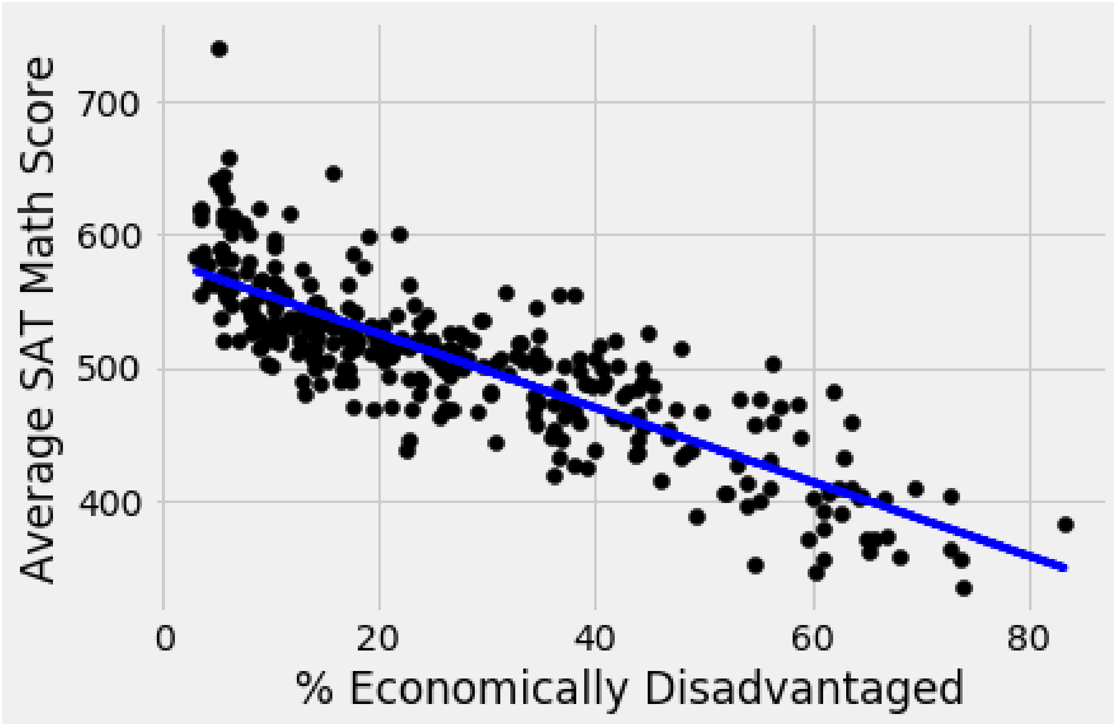
```
[581.28105413]
[[-2.779787]]
```

After doing multiple scatterplots using dependent variable y as % economically disadvantaged and testing different x variables such as class size, % students who graduated from the high school who attend college, % First Language not English, High Needs, % Students who are Economically Disadvantaged, % students who are Asian, % students who are Hispanic, % students who are Native American, % Native Hawaian, Pacific Islander, % Multirace, Non Hispanic, % Males as well as other variables, I found that the variable which highly correlates with Average SAT math score is % of students who are economically disadvantaged in the school. Thus, I decided to run linear regression with average SAT Math score as my dependent variable and % economically disadvantaged as my x variable.

After running linear regression, I found that the regression equation is Average_SAT_math = -2.78*(% Economically Disadvantaged) + 581

So, for instance, if the percentage of students in a particular high school who is economically disadvantaged is 10%, then the average SAT math score for a student in that high school is 27.8 points lower than a student who did not study in the school.

```
In [11]: plt.scatter(x, y,  color='black')
         a =plt.plot(x, predictions, color='blue',linewidth=3)
         plt.xlabel('% Economically Disadvantaged')
         plt.ylabel('Average SAT Math Score')
         plt.show()
```



In [176]:
```
#draw a scatterplot
def scatter(table, xcol, ycol, marker_color='blue'):

    #Cleaning missing and invalid data in table
    table.replace(to_replace = [np.inf, -np.inf], value = np.nan)
    table = table.dropna(axis = 0, how ='any')

    #Assigning axes
    x = table[xcol].values[:np.newaxis]
    y = table[ycol]

    # Reshaping
    x, y = x.reshape(-1,1), y.reshape(-1, 1)

    plt.scatter(x, y,  color = marker_color)


#Regress on a scatterplot with xcol and ycol (column names - str) from table
def scatter_and_regress(table, xcol, ycol, marker_color='blue'):

    #Cleaning missing and invalid data in table
    table.replace(to_replace = [np.inf, -np.inf], value = np.nan)
    table = table.dropna(axis = 0, how ='any')
```

```
    #Assigning axes
    x = table[xcol].values[:np.newaxis]
    y = table[ycol]

    # Reshaping
    x, y = x.reshape(-1,1), y.reshape(-1, 1)

    # Linear Regression Object
    lin_regression = LinearRegression()

    # Fitting linear model to the data
    lin_regression.fit(x,y)

    # Get slope of fitted line
    m = lin_regression.coef_

    # Get y-Intercept of the Line
    b = lin_regression.intercept_

    # Get Predictions for original x values
    predictions = lin_regression.predict(x)

    plt.scatter(x, y,  color = marker_color)
    plt.plot(x, predictions, color='black',linewidth=3)
    plt.xlabel(xcol)
    plt.ylabel(ycol)
    plt.show()
```

In [181]:
```
KNN = students_enrolled_12grade.loc[:, ['% Economically Disadvantaged','% Students Wi
KNN.head(3)

#reshuffle rows and divide into 2 sets - training and testing sets
```

Out[181]:
```
    % Economically Disadvantaged  % Students With Disabilities  High Needs  \
0                           21.5                           9.7       130.0
8                           22.7                          14.1       391.0
16                          14.6                          17.0       154.0

    % Attending College
0                  75.8
8                  81.6
16                 72.6
```

In [195]:
```
def cutoff_and_above(df, column, cutoff_value):
    """For each row, return True if the value in the column is equal to and above th
    classified = (df[column]>=cutoff_value)

    #Showing number of True (equal and above) and False values
```

```
        print(classified.value_counts())
        return classified


    def color_code(bool_array):
        """Return a color-coded array: 'Blue' for True values; 'Red' for False"""
        color = bool_array.apply(lambda row: 'Blue' if row else 'Red')
        return color


    def scatter_and_colorcode(table, xcol, ycol, color_col):
        """Draw a scatterplot w.r.t. Color column in table"""
        color = table[color]
        scatter(table, xcol, ycol, color)

    def distance_two_features(df, x_feature, y_feature):
        x1-x2 y1-y2
        """Compute the distance between x_feature and y_feature"""
        x = df[x_feature]
        y = df[y_feature]
        return np.sqrt(- rotem(x_feature))**2 + (row0.item(y_feature)-row1.item(y_feature
```

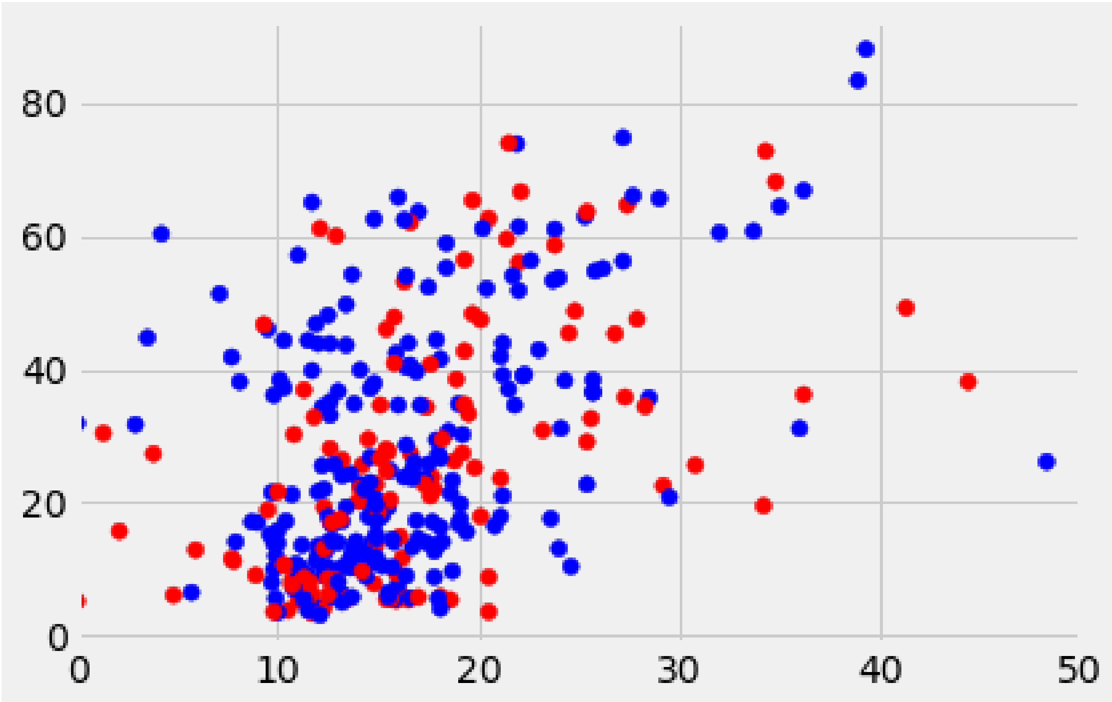
```
In [207]: #Divide schools into 2 classes using 70% Attending College as boundary
          classified = cutoff_and_above(KNN, '% Attending College', 70)
          KNN['Class'] = classified
          KNN['Color'] = color_code(classified)
          KNN.head(3)
```

```
True     248
False    146
Name: % Attending College, dtype: int64
```

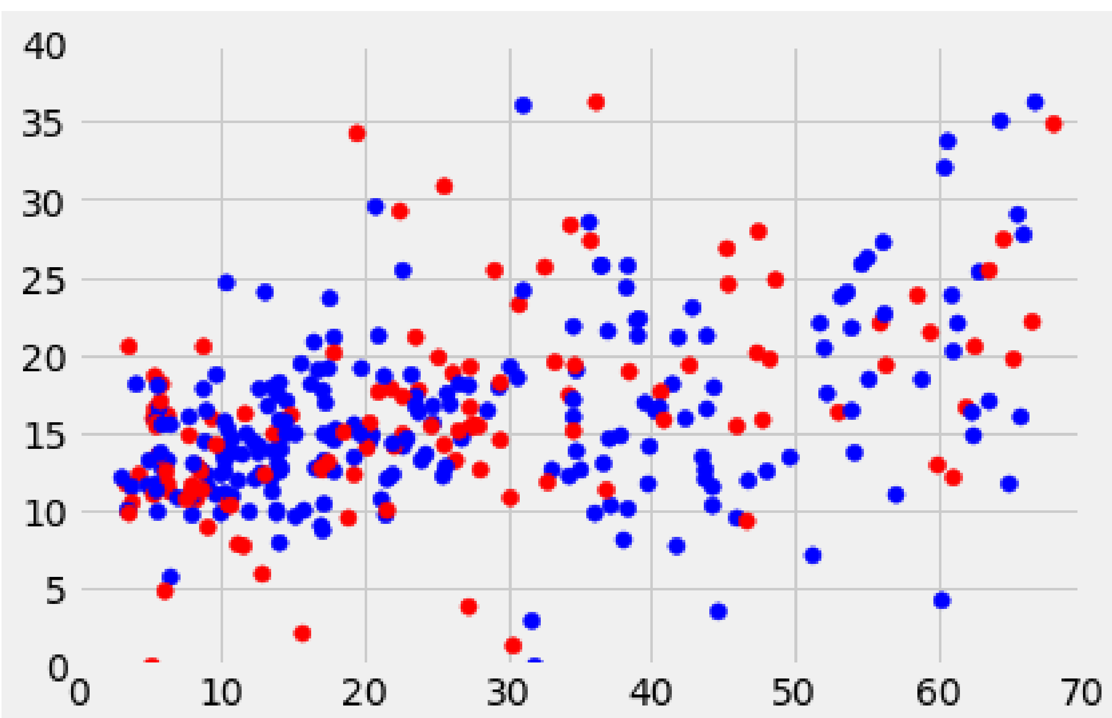
```
Out[207]:     % Economically Disadvantaged  % Students With Disabilities  High Needs  \
          0                           21.5                           9.7       130.0
          8                           22.7                          14.1       391.0
          16                          14.6                          17.0       154.0

              % Attending College  Class Color
          0                  75.8   True  Blue
          8                  81.6   True  Blue
          16                 72.6   True  Blue
```

```
In [209]: scatter_and_colorcode(KNN, '% Students With Disabilities','% Economically Disadvanta
          plt.xlim(0,50)
```

```
Out[209]: (0, 50)
```

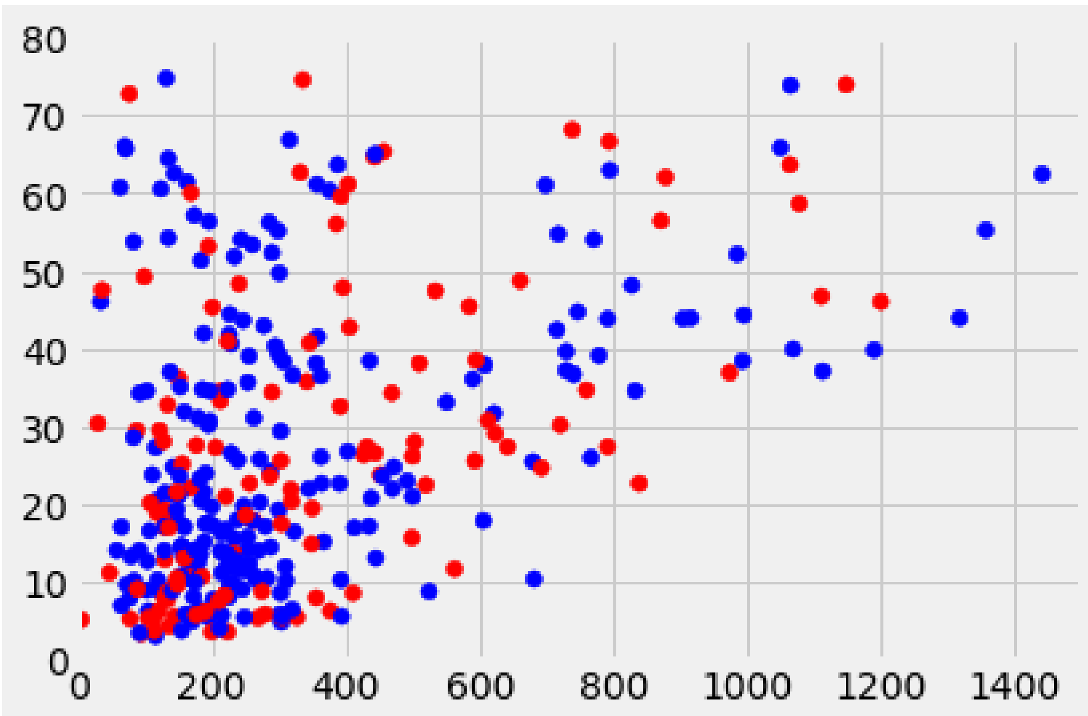



```
In [215]: scatter_and_colorcode(KNN, '% Economically Disadvantaged','% Students With Disabilit
          plt.xlim(0,70)
          plt.ylim(0,40)

Out[215]: (0, 40)
```



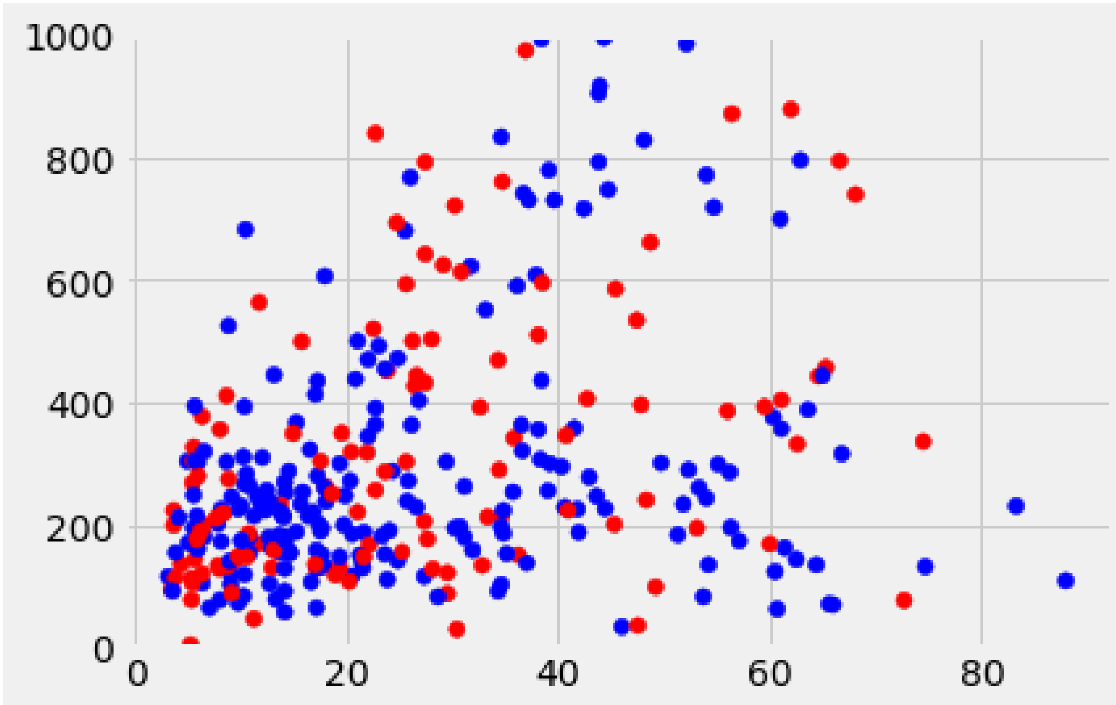
In [211]:
```
scatter_and_colorcode(KNN, 'High Needs','% Economically Disadvantaged', 'Color')
plt.xlim(0,1500)
plt.ylim(0,80)
```

Out[211]: (0, 80)



In [212]:
```
scatter_and_colorcode(KNN,'% Economically Disadvantaged', 'High Needs', 'Color')
plt.ylim(0,1000)
```

Out[212]: (0, 1000)



The second model that I use is K nearest neighbors. I wish to predict % of students attending college from a particular high school based on the the percentage of students who are economically disadvantaged and are disabled in a particular school. After building the model and running the code, I printed a list of my predictions. In the first row, the actual percentage of people attending college is 85.5% while the predicted percentage of people attending college is 81.44%

```
In [43]: KNN = students_enrolled_12grade.loc[:, ['% Economically Disadvantaged','% Students Wi
         #Cleaning missing and invalid data in table
         KNN.replace(to_replace = [np.inf, -np.inf], value = np.nan)
         KNN = table.dropna(axis = 0, how ='any')

         #Sourcecode: https://www.dataquest.io/blog/k-nearest-neighbors-in-python/
         import random
         import math
         from numpy.random import permutation

         # Randomly shuffle the index of KNN.
         random_indices = permutation(KNN.index)
         # Divide the data into half for training and testing set
         test_cutoff = math.floor(len(KNN)/2)
         # Generate the test set by taking the first 1/2 of the randomly shuffled indices.
         test = KNN.loc[random_indices[1:test_cutoff]]
         # Generate the train set with the rest of the data.
         train = KNN.loc[random_indices[test_cutoff:]]
         # The columns that we will be making predictions with.
```

```python
x_columns = ['% Economically Disadvantaged','% Students With Disabilities']
# The column that we want to predict.
y_column = ['% Attending College']

from sklearn.neighbors import KNeighborsRegressor
# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(train[x_columns], train[y_column])
# Make point predictions on the test set using the fit model.
predictions = knn.predict(test[x_columns])

# Get the actual values for the test set.
actual = test[y_column]
Actual_vs_Predictions = {'Actual': actual, 'Predicted': predictions}
print(Actual_vs_Predictions)

# Compute the mean squared error of our predictions.
mse = (((predictions - actual) ** 2).sum()) / len(predictions)
print(mse)
```

```
{'Actual':       % Attending College
687                 85.5
538                 72.2
1193                80.4
1451                71.9
758                 76.7
1554                72.5
668                 89.7
1621                87.5
957                 75.6
1353                71.8
833                 84.2
1661                70.3
516                 82.8
1654                85.6
972                 94.5
1278                62.8
1838                67.2
819                 75.3
290                 85.8
380                 79.3
1644                58.3
346                 77.2
1260                70.1
148                 56.3
807                 81.2
```

```
866                   77.9
879                   80.0
131                   65.5
993                   84.5
406                   90.0
...                    ...
1458                  81.4
1185                  57.6
1031                  57.7
330                   71.0
1826                  60.6
499                   51.5
1044                  53.8
1381                  84.6
635                   71.4
78                    89.9
219                   57.0
1560                  50.0
517                   79.3
1592                  83.0
412                   77.4
1822                  54.4
33                    85.0
1035                  79.3
742                   46.7
1675                  92.3
1145                  89.5
1745                  62.7
188                   75.0
656                   56.3
280                   80.9
374                   80.6
558                   71.3
1367                  69.6
1045                  84.9
960                   90.8

[145 rows x 1 columns], 'Predicted': array([[81.44],
       [78.3 ],
       [82.8 ],
       [70.62],
       [70.1 ],
       [76.68],
       [90.46],
       [81.74],
       [77.1 ],
       [76.9 ],
       [89.12],
```

```
[73.14],
[88.14],
[70.08],
[90.08],
[71.48],
[70.24],
[76.48],
[87.34],
[78.18],
[72.4 ],
[64.24],
[58.7 ],
[71.82],
[73.92],
[70.04],
[79.28],
[73.5 ],
[83.  ],
[81.72],
[76.68],
[72.08],
[86.06],
[88.76],
[76.5 ],
[72.4 ],
[79.28],
[66.66],
[67.46],
[66.66],
[76.92],
[69.66],
[78.36],
[67.94],
[87.58],
[73.4 ],
[61.5 ],
[88.32],
[84.76],
[74.74],
[67.94],
[81.56],
[88.32],
[63.46],
[59.98],
[87.68],
[87.06],
[63.46],
[87.82],
```

```
[83.56],
[62.16],
[61.74],
[75.52],
[65.52],
[78.78],
[87.06],
[89.82],
[77.2 ],
[72.98],
[72.4 ],
[87.8 ],
[78.54],
[85.94],
[86.8 ],
[70.38],
[83.24],
[88.94],
[83.24],
[81.96],
[76.72],
[87.6 ],
[87.64],
[84.64],
[88.94],
[87.68],
[76.74],
[65.02],
[86.08],
[88.94],
[71.06],
[65.02],
[77.46],
[88.  ],
[79.28],
[81.44],
[79.82],
[78.54],
[86.06],
[61.32],
[83.24],
[88.86],
[75.02],
[78.18],
[71.64],
[89.62],
[89.62],
[83.26],
```

```
       [68.54],
       [74.02],
       [77.3 ],
       [83.54],
       [73.24],
       [89.5 ],
       [74.8 ],
       [66.26],
       [83.56],
       [79.2 ],
       [54.78],
       [74.36],
       [70.74],
       [59.12],
       [81.56],
       [76.72],
       [67.  ],
       [84.64],
       [70.14],
       [74.42],
       [86.88],
       [86.16],
       [76.48],
       [73.5 ],
       [87.64],
       [60.96],
       [53.28],
       [78.3 ],
       [87.64],
       [77.08],
       [54.92],
       [76.06],
       [83.54],
       [79.82],
       [61.24],
       [64.68],
       [74.88],
       [89.44]])}
% Attending College    76.10944
dtype: float64
```

white-wine-quality

April 28, 2018

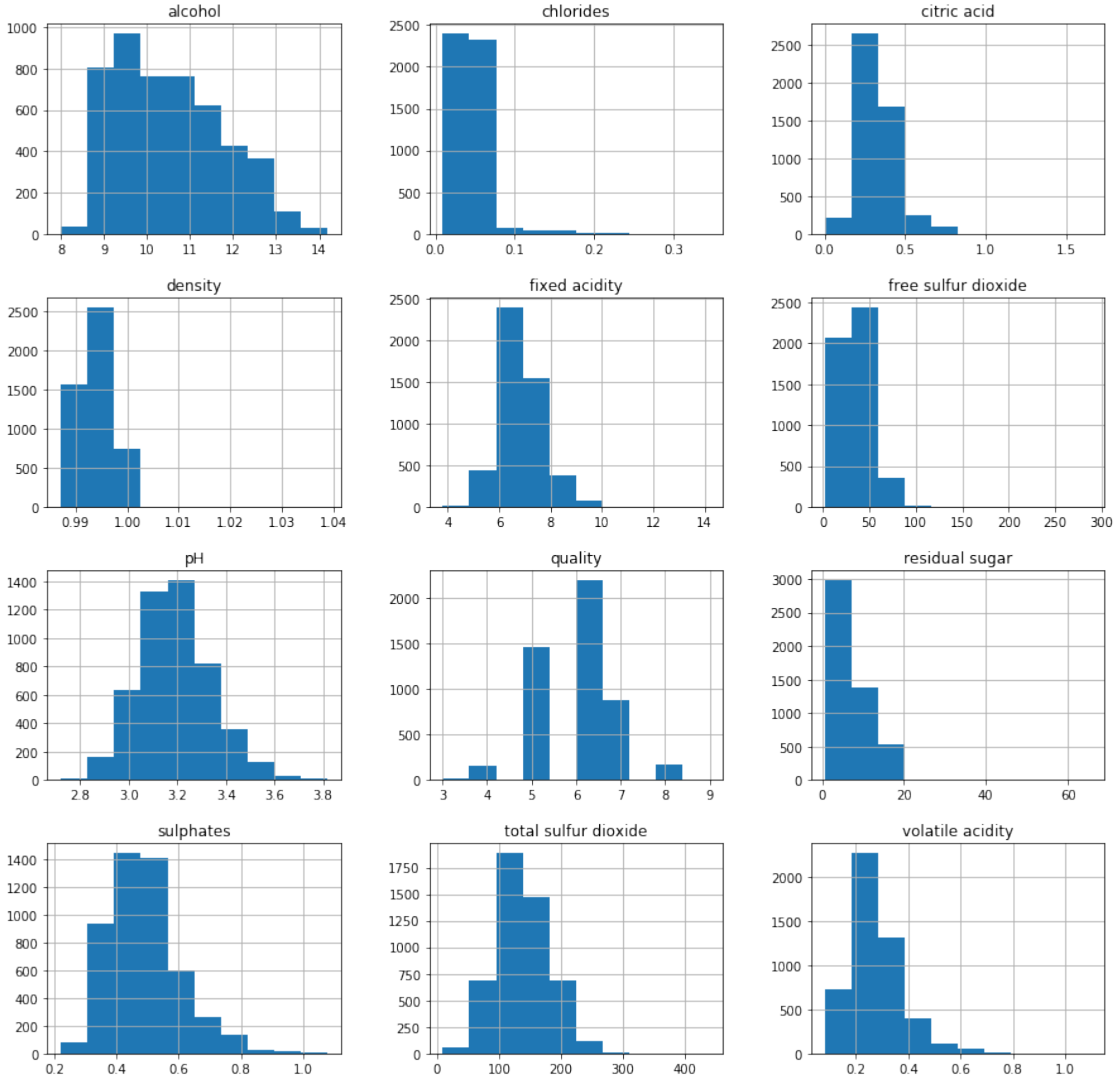
Predicting White Wine Quality

For this notebook we are going to work with the White Wine Quality dataset (https://archive.ics.uci.edu/ml/datasets/wine+quality).

We can first load the dataset into a pandas dataframe and plot the features we are going to use to predict the final quality to see what we are working with.

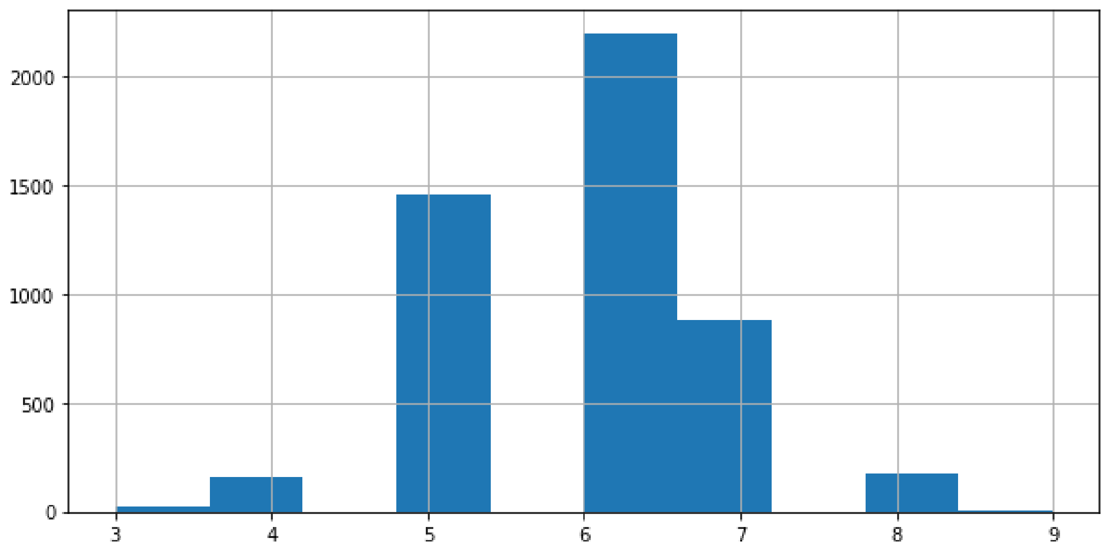
In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load wine dataset into pandas dataframe
df = pd.read_csv("white-wine-quality.csv", sep=";")
X = df.drop("quality", 1)
y = df["quality"]

#ăPlot features histograms
df.hist(figsize = [15, 15]);
```



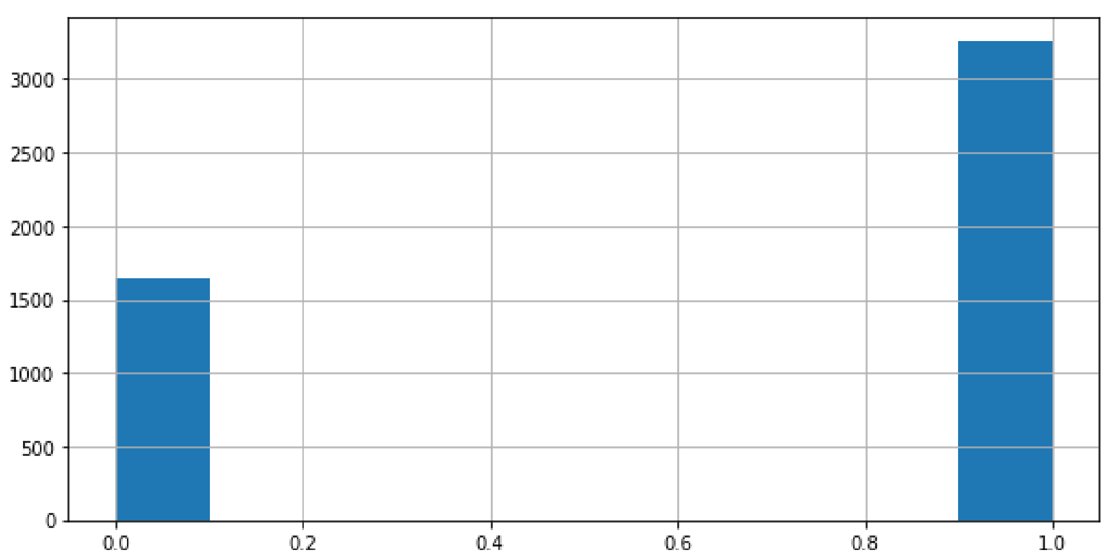
We can also check the quality outcome distribution:

```
In [3]: #ăPlot outcome histogram
        y.hist(figsize = [10, 5]);
```



For some of the quality classes we don't have enough datapoints so for the purpose of this project we are going to classify a wine in two categories: "good", when its quality is greater than 5, or "bad" when its quality is less than or equal to 5. Let's transform the outcome vector to a binary vector representing this classification.

In [4]:
```
y = y > 5
y.hist(figsize = [10, 5]);
```



k-Nearest Neighbors Model
As out first try to predict the quality of a white wine, we are going to train a k-Nearest Neighbors Model.

```
In [5]: from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn.neighbors import KNeighborsClassifier

        #ăSplit the data into test and train sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

        # Instantiate KNeighborsClassifier
        model = KNeighborsClassifier(n_neighbors = 5)
        model = model.fit(X_train, y_train)

        print("k-Nearest Neighbors accuracy for test set: %f" % model.score(X_test, y_test))
        y_true, y_pred = y_test, model.predict(X_test)
        print(classification_report(y_true, y_pred))
```

```
k-Nearest Neighbors accuracy for test set: 0.714286
             precision    recall  f1-score   support

      False       0.56      0.51      0.53       312
       True       0.78      0.81      0.79       668

avg / total       0.71      0.71      0.71       980
```

We obtained an accuracy score of 0.714286. But if we look at the ranges of values for the histograms presented at the beginning of this notebook, we can identify some differences between some of them of almost 100 times. For example, the "chlorides" feature ranges from 0 to 0.3, while the "total sulfure dioxide" ranges from 0 to 400. To normalize the ranges of values we are going to apply **data scaling**, and see if it improves the accuracy of our model.

```
In [6]: from sklearn.preprocessing import scale

        # Apply data scaling to features values
        X = scale(X)

        #ăSplit the data into test and train sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

        #ăTrain new model with scaled training data
        model = model.fit(X_train, y_train)

        print("Scaled k-Nearest Neighbors accuracy for test set: %f" % model.score(X_test, y_t
        y_true, y_pred = y_test, model.predict(X_test)
        print(classification_report(y_true, y_pred))
```

```
Scaled k-Nearest Neighbors accuracy for test set: 0.777551
             precision    recall  f1-score   support
```

```
       False       0.67      0.59      0.63       312
        True       0.82      0.86      0.84       668

 avg / total       0.77      0.78      0.77       980
```

We improved the accuracy to 0.777551. The data scaling process helped normalizing the ranges of values of the features.

Logistic Regression Model

Secondly, we can train a **Logistic Regression** model to see how it predicts the wine quality.

```
In [7]: from sklearn.linear_model import LogisticRegression

        # We need to reset the contents of X and y to its original values
        #ăLoad wine dataset into pandas dataframe
        df = pd.read_csv("white-wine-quality.csv", sep=";")
        X = df.drop("quality", 1)
        y = df["quality"]

        #ăSplit the data into test and train sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

        # Instantiate Logistic Regression Model
        model = LogisticRegression()

        #ăTrain model with training data
        model = model.fit(X_train, y_train)

        print("Logistic Regression accuracy for test set: %f" % model.score(X_test, y_test))
        y_true, y_pred = y_test, model.predict(X_test)
        print(classification_report(y_true, y_pred))
```

```
Logistic Regression accuracy for test set: 0.522449
             precision    recall  f1-score   support

          3       0.00      0.00      0.00         6
          4       0.00      0.00      0.00        26
          5       0.56      0.54      0.55       280
          6       0.51      0.78      0.62       452
          7       0.43      0.05      0.09       178
          8       0.00      0.00      0.00        36
          9       0.00      0.00      0.00         2

avg / total       0.47      0.52      0.46       980
```

We obtained an accuracy score of 0.525510. Let's see what happen now if we scale our data as we did before for the kNN model.

```
In [8]: # Apply data scaling to features values
        X = scale(X)

        #ăSplit the data into test and train sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

        #ăTrain new model with scaled training data
        model = model.fit(X_train, y_train)

        print("Scaled Logistic Regression score for test set: %f" % model.score(X_test, y_test
        y_true, y_pred = y_test, model.predict(X_test)
        print(classification_report(y_true, y_pred))
```

```
Scaled Logistic Regression score for test set: 0.530612
             precision    recall  f1-score   support

          3       0.00      0.00      0.00         6
          4       1.00      0.04      0.07        26
          5       0.56      0.54      0.55       280
          6       0.52      0.78      0.63       452
          7       0.46      0.09      0.15       178
          8       0.00      0.00      0.00        36
          9       0.00      0.00      0.00         2

avg / total       0.51      0.53      0.47       980
```

We got a similar accuracy score. It only improved to 0.530612.

As a conclusion we can state that the k-Nearest Neighbors Model performs much better than the Logistic Regression Model.

abalone

April 28, 2018

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)
from matplotlib import patches
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
from sklearn.linear_model import LinearRegression
```

In [2]:
```
data = pd.read_csv("abalone.csv")
data.describe(include='all')
list(data)
```

Out[2]:
```
['sex',
 'length',
 'diameter',
 'height',
 'whole weight',
 'shucked weight',
 'viscera weight',
 'shell weight',
 'rings']
```

In [3]:
```
table = data.loc[:, ['sex',
 'length',
 'diameter',
 'height',
 'whole weight',
 'shucked weight',
 'viscera weight',
 'shell weight',
 'rings']]
table
```

Out[3]:

	sex	length	diameter	height	whole weight	shucked weight	\
0	M	0.455	0.365	0.095	0.5140	0.2245	
1	M	0.350	0.265	0.090	0.2255	0.0995	
2	F	0.530	0.420	0.135	0.6770	0.2565	
3	M	0.440	0.365	0.125	0.5160	0.2155	
4	I	0.330	0.255	0.080	0.2050	0.0895	
5	I	0.425	0.300	0.095	0.3515	0.1410	
6	F	0.530	0.415	0.150	0.7775	0.2370	
7	F	0.545	0.425	0.125	0.7680	0.2940	
8	M	0.475	0.370	0.125	0.5095	0.2165	
9	F	0.550	0.440	0.150	0.8945	0.3145	
10	F	0.525	0.380	0.140	0.6065	0.1940	
11	M	0.430	0.350	0.110	0.4060	0.1675	
12	M	0.490	0.380	0.135	0.5415	0.2175	
13	F	0.535	0.405	0.145	0.6845	0.2725	
14	F	0.470	0.355	0.100	0.4755	0.1675	
15	M	0.500	0.400	0.130	0.6645	0.2580	
16	I	0.355	0.280	0.085	0.2905	0.0950	
17	F	0.440	0.340	0.100	0.4510	0.1880	
18	M	0.365	0.295	0.080	0.2555	0.0970	
19	M	0.450	0.320	0.100	0.3810	0.1705	
20	M	0.355	0.280	0.095	0.2455	0.0955	
21	I	0.380	0.275	0.100	0.2255	0.0800	
22	F	0.565	0.440	0.155	0.9395	0.4275	
23	F	0.550	0.415	0.135	0.7635	0.3180	
24	F	0.615	0.480	0.165	1.1615	0.5130	
25	F	0.560	0.440	0.140	0.9285	0.3825	
26	F	0.580	0.450	0.185	0.9955	0.3945	
27	M	0.590	0.445	0.140	0.9310	0.3560	
28	M	0.605	0.475	0.180	0.9365	0.3940	
29	M	0.575	0.425	0.140	0.8635	0.3930	
...	..	...	...	...	...	...	
4147	M	0.695	0.550	0.195	1.6645	0.7270	
4148	M	0.770	0.605	0.175	2.0505	0.8005	
4149	I	0.280	0.215	0.070	0.1240	0.0630	
4150	I	0.330	0.230	0.080	0.1400	0.0565	
4151	I	0.350	0.250	0.075	0.1695	0.0835	
4152	I	0.370	0.280	0.090	0.2180	0.0995	
4153	I	0.430	0.315	0.115	0.3840	0.1885	
4154	I	0.435	0.330	0.095	0.3930	0.2190	
4155	I	0.440	0.350	0.110	0.3805	0.1575	
4156	M	0.475	0.370	0.110	0.4895	0.2185	
4157	M	0.475	0.360	0.140	0.5135	0.2410	
4158	I	0.480	0.355	0.110	0.4495	0.2010	
4159	F	0.560	0.440	0.135	0.8025	0.3500	
4160	F	0.585	0.475	0.165	1.0530	0.4580	
4161	F	0.585	0.455	0.170	0.9945	0.4255	
4162	M	0.385	0.255	0.100	0.3175	0.1370	

4163	I	0.390	0.310	0.085	0.3440	0.1810
4164	I	0.390	0.290	0.100	0.2845	0.1255
4165	I	0.405	0.300	0.085	0.3035	0.1500
4166	I	0.475	0.365	0.115	0.4990	0.2320
4167	M	0.500	0.380	0.125	0.5770	0.2690
4168	F	0.515	0.400	0.125	0.6150	0.2865
4169	M	0.520	0.385	0.165	0.7910	0.3750
4170	M	0.550	0.430	0.130	0.8395	0.3155
4171	M	0.560	0.430	0.155	0.8675	0.4000
4172	F	0.565	0.450	0.165	0.8870	0.3700
4173	M	0.590	0.440	0.135	0.9660	0.4390
4174	M	0.600	0.475	0.205	1.1760	0.5255
4175	F	0.625	0.485	0.150	1.0945	0.5310
4176	M	0.710	0.555	0.195	1.9485	0.9455

	viscera weight	shell weight	rings
0	0.1010	0.1500	15
1	0.0485	0.0700	7
2	0.1415	0.2100	9
3	0.1140	0.1550	10
4	0.0395	0.0550	7
5	0.0775	0.1200	8
6	0.1415	0.3300	20
7	0.1495	0.2600	16
8	0.1125	0.1650	9
9	0.1510	0.3200	19
10	0.1475	0.2100	14
11	0.0810	0.1350	10
12	0.0950	0.1900	11
13	0.1710	0.2050	10
14	0.0805	0.1850	10
15	0.1330	0.2400	12
16	0.0395	0.1150	7
17	0.0870	0.1300	10
18	0.0430	0.1000	7
19	0.0750	0.1150	9
20	0.0620	0.0750	11
21	0.0490	0.0850	10
22	0.2140	0.2700	12
23	0.2100	0.2000	9
24	0.3010	0.3050	10
25	0.1880	0.3000	11
26	0.2720	0.2850	11
27	0.2340	0.2800	12
28	0.2190	0.2950	15
29	0.2270	0.2000	11
...	...	...	...
4147	0.3600	0.4450	11

```
4148         0.5260        0.3550     11
4149         0.0215        0.0300      6
4150         0.0365        0.0460      7
4151         0.0355        0.0410      6
4152         0.0545        0.0615      7
4153         0.0715        0.1100      8
4154         0.0750        0.0885      6
4155         0.0895        0.1150      6
4156         0.1070        0.1460      8
4157         0.1045        0.1550      8
4158         0.0890        0.1400      8
4159         0.1615        0.2590      9
4160         0.2170        0.3000     11
4161         0.2630        0.2845     11
4162         0.0680        0.0920      8
4163         0.0695        0.0790      7
4164         0.0635        0.0810      7
4165         0.0505        0.0880      7
4166         0.0885        0.1560     10
4167         0.1265        0.1535      9
4168         0.1230        0.1765      8
4169         0.1800        0.1815     10
4170         0.1955        0.2405     10
4171         0.1720        0.2290      8
4172         0.2390        0.2490     11
4173         0.2145        0.2605     10
4174         0.2875        0.3080      9
4175         0.2610        0.2960     10
4176         0.3765        0.4950     12

[4177 rows x 9 columns]
```

In [4]:
```python
df_male = table['sex'] == 'M'
tableM = table[df_male]
tableM
```

Out[4]:
```
     sex  length  diameter  height  whole weight  shucked weight  \
0      M   0.455     0.365   0.095        0.5140          0.2245
1      M   0.350     0.265   0.090        0.2255          0.0995
3      M   0.440     0.365   0.125        0.5160          0.2155
8      M   0.475     0.370   0.125        0.5095          0.2165
11     M   0.430     0.350   0.110        0.4060          0.1675
12     M   0.490     0.380   0.135        0.5415          0.2175
15     M   0.500     0.400   0.130        0.6645          0.2580
18     M   0.365     0.295   0.080        0.2555          0.0970
19     M   0.450     0.320   0.100        0.3810          0.1705
20     M   0.355     0.280   0.095        0.2455          0.0955
27     M   0.590     0.445   0.140        0.9310          0.3560
```

28	M	0.605	0.475	0.180	0.9365	0.3940
29	M	0.575	0.425	0.140	0.8635	0.3930
30	M	0.580	0.470	0.165	0.9975	0.3935
32	M	0.665	0.525	0.165	1.3380	0.5515
35	M	0.465	0.355	0.105	0.4795	0.2270
39	M	0.355	0.290	0.090	0.3275	0.1340
46	M	0.470	0.370	0.120	0.5795	0.2930
51	M	0.400	0.320	0.095	0.3030	0.1335
52	M	0.485	0.360	0.130	0.5415	0.2595
54	M	0.405	0.310	0.100	0.3850	0.1730
56	M	0.445	0.350	0.120	0.4425	0.1920
57	M	0.470	0.385	0.135	0.5895	0.2765
60	M	0.450	0.345	0.105	0.4115	0.1800
61	M	0.505	0.405	0.110	0.6250	0.3050
63	M	0.425	0.325	0.095	0.3785	0.1705
64	M	0.520	0.400	0.120	0.5800	0.2340
65	M	0.475	0.355	0.120	0.4800	0.2340
70	M	0.555	0.425	0.130	0.7665	0.2640
73	M	0.570	0.480	0.175	1.1850	0.4740
...	..	...	...	...	...	...
4099	M	0.670	0.525	0.180	1.4915	0.7280
4102	M	0.680	0.545	0.185	1.6720	0.7075
4103	M	0.700	0.545	0.215	1.9125	0.8825
4109	M	0.480	0.365	0.130	0.5305	0.2405
4110	M	0.510	0.410	0.155	1.2825	0.5690
4116	M	0.625	0.480	0.160	1.2415	0.6575
4118	M	0.650	0.525	0.185	1.3455	0.5860
4120	M	0.350	0.265	0.090	0.1970	0.0730
4126	M	0.550	0.420	0.145	0.7385	0.3210
4128	M	0.555	0.435	0.145	0.9205	0.4040
4130	M	0.580	0.450	0.140	0.8240	0.3465
4133	M	0.585	0.450	0.150	0.9970	0.4055
4137	M	0.630	0.505	0.155	1.1050	0.4920
4138	M	0.630	0.490	0.155	1.2290	0.5350
4142	M	0.655	0.525	0.180	1.4020	0.6240
4144	M	0.670	0.535	0.190	1.6690	0.7465
4145	M	0.670	0.525	0.200	1.7405	0.6205
4146	M	0.695	0.530	0.210	1.5100	0.6640
4147	M	0.695	0.550	0.195	1.6645	0.7270
4148	M	0.770	0.605	0.175	2.0505	0.8005
4156	M	0.475	0.370	0.110	0.4895	0.2185
4157	M	0.475	0.360	0.140	0.5135	0.2410
4162	M	0.385	0.255	0.100	0.3175	0.1370
4167	M	0.500	0.380	0.125	0.5770	0.2690
4169	M	0.520	0.385	0.165	0.7910	0.3750
4170	M	0.550	0.430	0.130	0.8395	0.3155
4171	M	0.560	0.430	0.155	0.8675	0.4000
4173	M	0.590	0.440	0.135	0.9660	0.4390

4174	M	0.600	0.475	0.205	1.1760	0.5255
4176	M	0.710	0.555	0.195	1.9485	0.9455

	viscera weight	shell weight	rings
0	0.1010	0.1500	15
1	0.0485	0.0700	7
3	0.1140	0.1550	10
8	0.1125	0.1650	9
11	0.0810	0.1350	10
12	0.0950	0.1900	11
15	0.1330	0.2400	12
18	0.0430	0.1000	7
19	0.0750	0.1150	9
20	0.0620	0.0750	11
27	0.2340	0.2800	12
28	0.2190	0.2950	15
29	0.2270	0.2000	11
30	0.2420	0.3300	10
32	0.3575	0.3500	18
35	0.1240	0.1250	8
39	0.0860	0.0900	9
46	0.2270	0.1400	9
51	0.0600	0.1000	7
52	0.0960	0.1600	10
54	0.0915	0.1100	7
56	0.0955	0.1350	8
57	0.1200	0.1700	8
60	0.1125	0.1350	7
61	0.1600	0.1750	9
63	0.0800	0.1000	7
64	0.1315	0.1850	8
65	0.1015	0.1350	8
70	0.1680	0.2750	13
73	0.2610	0.3800	11
...	...	...	...
4099	0.3430	0.3810	9
4102	0.3640	0.4800	11
4103	0.4385	0.5060	10
4109	0.1270	0.1390	8
4110	0.2910	0.3795	9
4116	0.2625	0.2785	9
4118	0.2780	0.3865	9
4120	0.0365	0.0770	7
4126	0.1485	0.2520	11
4128	0.2275	0.2550	8
4130	0.1765	0.2630	10
4133	0.2830	0.2510	11
4137	0.2260	0.3250	11

```
4138            0.2900         0.3350      11
4142            0.2935         0.3650      13
4144            0.2935         0.5080      11
4145            0.2970         0.6570      11
4146            0.4095         0.3850      10
4147            0.3600         0.4450      11
4148            0.5260         0.3550      11
4156            0.1070         0.1460       8
4157            0.1045         0.1550       8
4162            0.0680         0.0920       8
4167            0.1265         0.1535       9
4169            0.1800         0.1815      10
4170            0.1955         0.2405      10
4171            0.1720         0.2290       8
4173            0.2145         0.2605      10
4174            0.2875         0.3080       9
4176            0.3765         0.4950      12

[1528 rows x 9 columns]
```

```
In [5]: df_female = table['sex'] == 'F'
        tableF = table[df_female]
        tableF
```

```
Out[5]:      sex  length  diameter  height  whole weight  shucked weight  \
        2      F   0.530     0.420   0.135        0.6770          0.2565
        6      F   0.530     0.415   0.150        0.7775          0.2370
        7      F   0.545     0.425   0.125        0.7680          0.2940
        9      F   0.550     0.440   0.150        0.8945          0.3145
        10     F   0.525     0.380   0.140        0.6065          0.1940
        13     F   0.535     0.405   0.145        0.6845          0.2725
        14     F   0.470     0.355   0.100        0.4755          0.1675
        17     F   0.440     0.340   0.100        0.4510          0.1880
        22     F   0.565     0.440   0.155        0.9395          0.4275
        23     F   0.550     0.415   0.135        0.7635          0.3180
        24     F   0.615     0.480   0.165        1.1615          0.5130
        25     F   0.560     0.440   0.140        0.9285          0.3825
        26     F   0.580     0.450   0.185        0.9955          0.3945
        31     F   0.680     0.560   0.165        1.6390          0.6055
        33     F   0.680     0.550   0.175        1.7980          0.8150
        34     F   0.705     0.550   0.200        1.7095          0.6330
        36     F   0.540     0.475   0.155        1.2170          0.5305
        37     F   0.450     0.355   0.105        0.5225          0.2370
        38     F   0.575     0.445   0.135        0.8830          0.3810
        40     F   0.450     0.335   0.105        0.4250          0.1865
        41     F   0.550     0.425   0.135        0.8515          0.3620
        47     F   0.460     0.375   0.120        0.4605          0.1775
        49     F   0.525     0.425   0.160        0.8355          0.3545
```

53	F	0.470	0.360	0.120	0.4775	0.2105
55	F	0.500	0.400	0.140	0.6615	0.2565
59	F	0.505	0.400	0.125	0.5830	0.2460
62	F	0.530	0.410	0.130	0.6965	0.3020
66	F	0.565	0.440	0.160	0.9150	0.3540
67	F	0.595	0.495	0.185	1.2850	0.4160
68	F	0.475	0.390	0.120	0.5305	0.2135
...	..	...	...	...	...	...
4058	F	0.695	0.560	0.185	1.7715	0.8195
4063	F	0.630	0.515	0.160	1.3360	0.5530
4064	F	0.640	0.490	0.180	1.3600	0.6530
4084	F	0.575	0.480	0.170	1.1000	0.5060
4092	F	0.625	0.490	0.190	1.7015	0.7465
4095	F	0.635	0.485	0.155	1.0730	0.4670
4096	F	0.635	0.500	0.175	1.4770	0.6840
4098	F	0.650	0.495	0.160	1.3105	0.5770
4100	F	0.675	0.520	0.175	1.4940	0.7365
4101	F	0.675	0.510	0.150	1.1965	0.4750
4104	F	0.710	0.545	0.175	1.9070	0.8725
4105	F	0.715	0.565	0.180	1.7900	0.8440
4106	F	0.720	0.590	0.205	1.7495	0.7755
4112	F	0.560	0.420	0.180	1.6645	0.7755
4114	F	0.570	0.450	0.150	0.9645	0.5310
4115	F	0.605	0.465	0.155	1.1000	0.5470
4117	F	0.640	0.505	0.175	1.3185	0.6185
4134	F	0.595	0.455	0.140	0.9140	0.3895
4135	F	0.600	0.500	0.170	1.1300	0.4405
4136	F	0.615	0.495	0.155	1.0805	0.5200
4139	F	0.635	0.495	0.175	1.2355	0.5205
4140	F	0.645	0.535	0.190	1.2395	0.4680
4141	F	0.650	0.505	0.165	1.3570	0.5725
4143	F	0.655	0.500	0.220	1.3590	0.6420
4159	F	0.560	0.440	0.135	0.8025	0.3500
4160	F	0.585	0.475	0.165	1.0530	0.4580
4161	F	0.585	0.455	0.170	0.9945	0.4255
4168	F	0.515	0.400	0.125	0.6150	0.2865
4172	F	0.565	0.450	0.165	0.8870	0.3700
4175	F	0.625	0.485	0.150	1.0945	0.5310

	viscera weight	shell weight	rings
2	0.1415	0.2100	9
6	0.1415	0.3300	20
7	0.1495	0.2600	16
9	0.1510	0.3200	19
10	0.1475	0.2100	14
13	0.1710	0.2050	10
14	0.0805	0.1850	10
17	0.0870	0.1300	10

22	0.2140	0.2700	12
23	0.2100	0.2000	9
24	0.3010	0.3050	10
25	0.1880	0.3000	11
26	0.2720	0.2850	11
31	0.2805	0.4600	15
33	0.3925	0.4550	19
34	0.4115	0.4900	13
36	0.3075	0.3400	16
37	0.1165	0.1450	8
38	0.2035	0.2600	11
40	0.0910	0.1150	9
41	0.1960	0.2700	14
47	0.1100	0.1500	7
49	0.2135	0.2450	9
53	0.1055	0.1500	10
55	0.1755	0.2200	8
59	0.1300	0.1750	7
62	0.1935	0.2000	10
66	0.1935	0.3200	12
67	0.2240	0.4850	13
68	0.1155	0.1700	10
...	...	...	...
4058	0.3310	0.4370	10
4063	0.3205	0.3500	11
4064	0.3470	0.3050	9
4084	0.2485	0.3100	10
4092	0.4105	0.3855	11
4095	0.1975	0.3500	11
4096	0.3005	0.3900	12
4098	0.3315	0.3550	9
4100	0.3055	0.3700	9
4101	0.3040	0.3860	11
4104	0.4565	0.4750	11
4105	0.3535	0.5385	9
4106	0.4225	0.4800	11
4112	0.3500	0.4525	9
4114	0.1890	0.2090	9
4115	0.2665	0.2585	10
4117	0.3020	0.3315	9
4134	0.2225	0.2710	9
4135	0.2670	0.3350	11
4136	0.1900	0.3200	9
4139	0.3085	0.3470	10
4140	0.2385	0.4240	10
4141	0.2810	0.4300	11
4143	0.3255	0.4050	13
4159	0.1615	0.2590	9

```
4160          0.2170        0.3000     11
4161          0.2630        0.2845     11
4168          0.1230        0.1765      8
4172          0.2390        0.2490     11
4175          0.2610        0.2960     10

[1307 rows x 9 columns]
```

```
In [6]: df_infant = table['sex'] == 'I'
        tableI = table[df_infant]
        tableI
```

```
Out[6]:      sex  length  diameter  height  whole weight  shucked weight  \
        4      I   0.330     0.255   0.080        0.2050          0.0895
        5      I   0.425     0.300   0.095        0.3515          0.1410
        16     I   0.355     0.280   0.085        0.2905          0.0950
        21     I   0.380     0.275   0.100        0.2255          0.0800
        42     I   0.240     0.175   0.045        0.0700          0.0315
        43     I   0.205     0.150   0.055        0.0420          0.0255
        44     I   0.210     0.150   0.050        0.0420          0.0175
        45     I   0.390     0.295   0.095        0.2030          0.0875
        48     I   0.325     0.245   0.070        0.1610          0.0755
        50     I   0.520     0.410   0.120        0.5950          0.2385
        58     I   0.245     0.190   0.060        0.0860          0.0420
        69     I   0.310     0.235   0.070        0.1510          0.0630
        100    I   0.360     0.265   0.095        0.2315          0.1050
        112    I   0.435     0.320   0.080        0.3325          0.1485
        121    I   0.385     0.295   0.085        0.2535          0.1030
        124    I   0.360     0.280   0.080        0.1755          0.0810
        125    I   0.270     0.195   0.060        0.0730          0.0285
        126    I   0.375     0.275   0.090        0.2380          0.1075
        127    I   0.385     0.290   0.085        0.2505          0.1120
        133    I   0.350     0.260   0.095        0.2110          0.0860
        134    I   0.265     0.200   0.065        0.0975          0.0400
        147    I   0.280     0.205   0.080        0.1270          0.0520
        148    I   0.175     0.130   0.055        0.0315          0.0105
        149    I   0.170     0.130   0.095        0.0300          0.0130
        174    I   0.235     0.160   0.040        0.0480          0.0185
        175    I   0.360     0.260   0.090        0.1785          0.0645
        176    I   0.315     0.210   0.060        0.1250          0.0600
        177    I   0.315     0.245   0.085        0.1435          0.0530
        178    I   0.225     0.160   0.045        0.0465          0.0250
        193    I   0.355     0.275   0.085        0.2200          0.0920
        ...   ..     ...       ...     ...           ...             ...
        4077   I   0.550     0.435   0.125        0.7410          0.3480
        4080   I   0.555     0.430   0.125        0.7005          0.3395
        4082   I   0.565     0.465   0.150        1.1815          0.5810
        4087   I   0.595     0.475   0.155        0.9840          0.4865
```

4107	I	0.420	0.305	0.100	0.3415	0.1645
4108	I	0.480	0.350	0.100	0.5190	0.2365
4111	I	0.515	0.400	0.140	0.7165	0.3495
4113	I	0.560	0.420	0.140	0.8370	0.4140
4119	I	0.300	0.215	0.050	0.1185	0.0480
4121	I	0.455	0.350	0.130	0.4725	0.2150
4122	I	0.460	0.365	0.110	0.4495	0.1755
4123	I	0.490	0.375	0.115	0.5570	0.2275
4124	I	0.500	0.385	0.120	0.5160	0.1970
4125	I	0.540	0.415	0.135	0.7090	0.3195
4127	I	0.550	0.445	0.110	0.7935	0.3780
4129	I	0.570	0.425	0.140	0.7655	0.3310
4131	I	0.580	0.425	0.145	0.8300	0.3790
4132	I	0.585	0.470	0.170	0.9850	0.3695
4149	I	0.280	0.215	0.070	0.1240	0.0630
4150	I	0.330	0.230	0.080	0.1400	0.0565
4151	I	0.350	0.250	0.075	0.1695	0.0835
4152	I	0.370	0.280	0.090	0.2180	0.0995
4153	I	0.430	0.315	0.115	0.3840	0.1885
4154	I	0.435	0.330	0.095	0.3930	0.2190
4155	I	0.440	0.350	0.110	0.3805	0.1575
4158	I	0.480	0.355	0.110	0.4495	0.2010
4163	I	0.390	0.310	0.085	0.3440	0.1810
4164	I	0.390	0.290	0.100	0.2845	0.1255
4165	I	0.405	0.300	0.085	0.3035	0.1500
4166	I	0.475	0.365	0.115	0.4990	0.2320

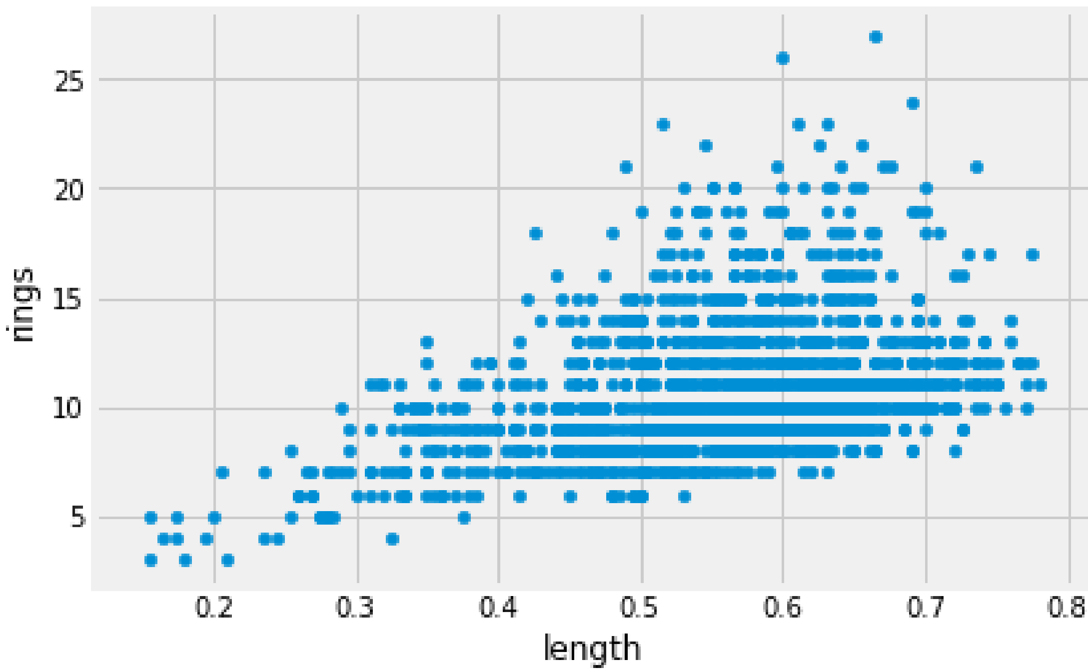
	viscera weight	shell weight	rings
4	0.0395	0.0550	7
5	0.0775	0.1200	8
16	0.0395	0.1150	7
21	0.0490	0.0850	10
42	0.0235	0.0200	5
43	0.0150	0.0120	5
44	0.0125	0.0150	4
45	0.0450	0.0750	7
48	0.0255	0.0450	6
50	0.1110	0.1900	8
58	0.0140	0.0250	4
69	0.0405	0.0450	6
100	0.0460	0.0750	7
112	0.0635	0.1050	9
121	0.0575	0.0850	7
124	0.0505	0.0700	6
125	0.0235	0.0300	5
126	0.0545	0.0700	6
127	0.0610	0.0800	8
133	0.0560	0.0680	7

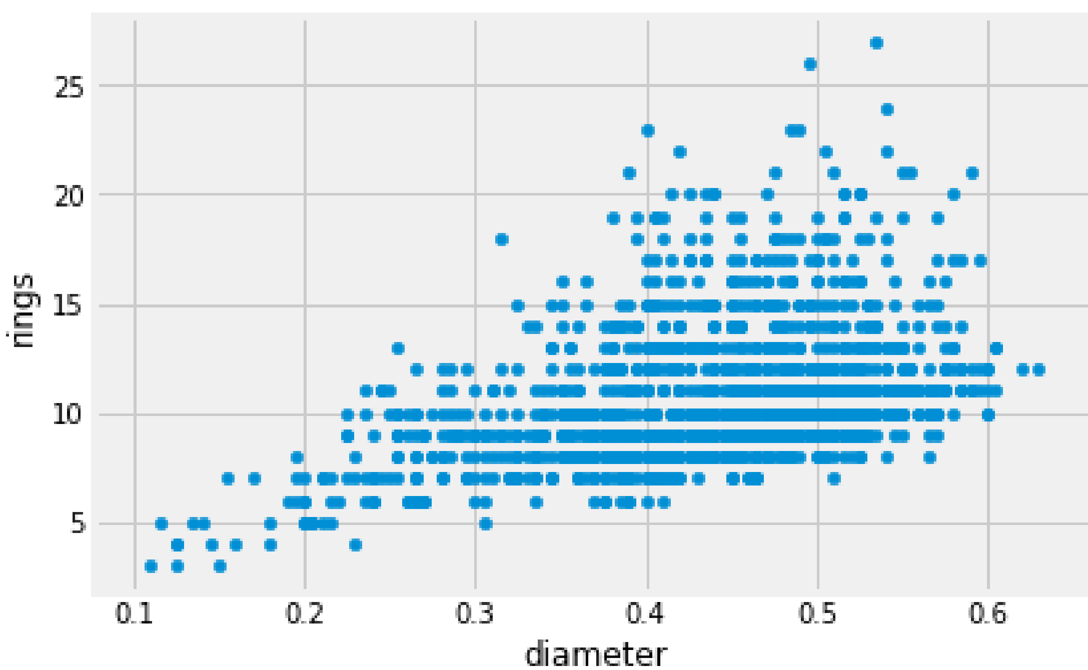
```
134           0.0205       0.0280      7
147           0.0390       0.0420      9
148           0.0065       0.0125      5
149           0.0080       0.0100      4
174           0.0180       0.0150      5
175           0.0370       0.0750      7
176           0.0375       0.0350      5
177           0.0475       0.0500      8
178           0.0150       0.0150      4
193           0.0600       0.1500      8
...              ...          ...    ...
4077          0.1585       0.2060      9
4080          0.1355       0.2095      8
4082          0.2215       0.3095      9
4087          0.1840       0.2755     10
4107          0.0775       0.0860      7
4108          0.1275       0.1260      7
4111          0.1595       0.1785      8
4113          0.2140       0.2000      8
4119          0.0225       0.0420      4
4121          0.0745       0.1500      9
4122          0.1020       0.1500      8
4123          0.1335       0.1765      8
4124          0.1305       0.1650      8
4125          0.1740       0.1850      9
4127          0.1420       0.2600     10
4129          0.1400       0.2400     10
4131          0.1605       0.2575     11
4132          0.2395       0.3150     10
4149          0.0215       0.0300      6
4150          0.0365       0.0460      7
4151          0.0355       0.0410      6
4152          0.0545       0.0615      7
4153          0.0715       0.1100      8
4154          0.0750       0.0885      6
4155          0.0895       0.1150      6
4158          0.0890       0.1400      8
4163          0.0695       0.0790      7
4164          0.0635       0.0810      7
4165          0.0505       0.0880      7
4166          0.0885       0.1560     10

[1342 rows x 9 columns]
```

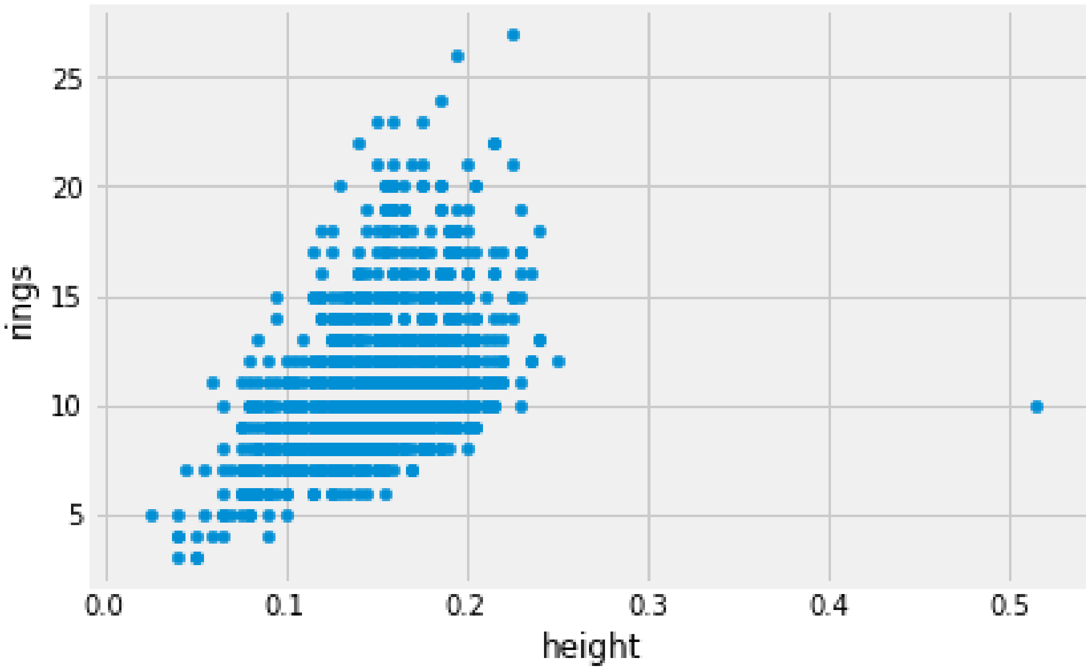
In [7]:
```
tableM.plot.scatter('length', 'rings', s=None, c=None)
tableM.plot.scatter('diameter', 'rings', s=None, c=None)
tableM.plot.scatter('height', 'rings', s=None, c=None)
tableM.plot.scatter('whole weight', 'rings', s=None, c=None)
```

```
tableM.plot.scatter('viscera weight', 'rings', s=None, c=None)
tableM.plot.scatter('shell weight', 'rings', s=None, c=None)
```

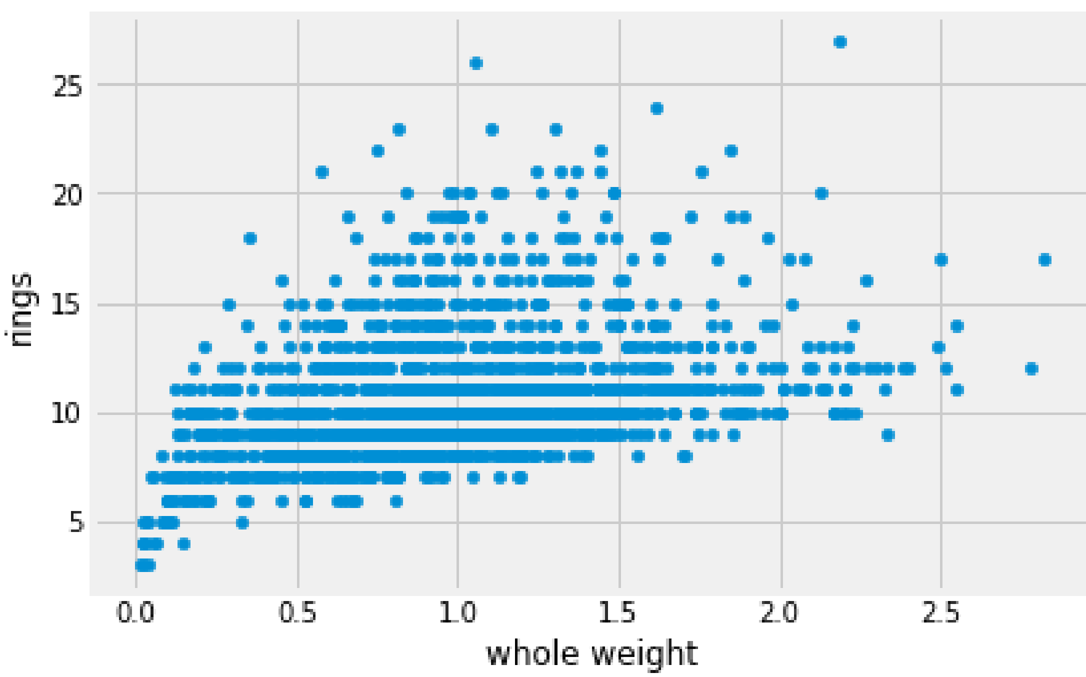
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x111b34400>



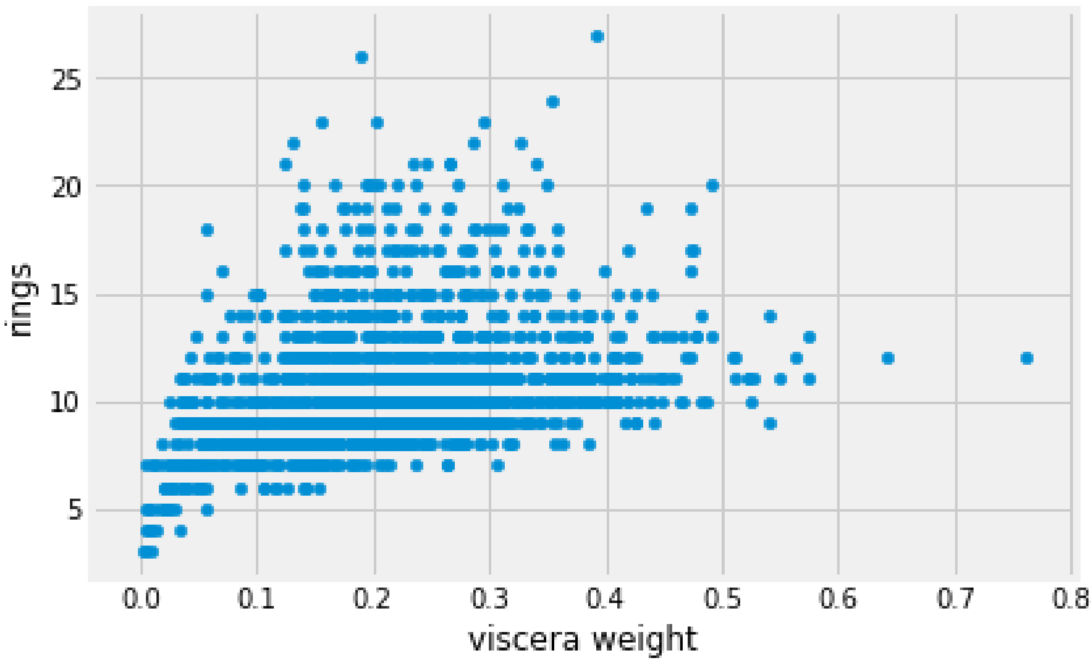


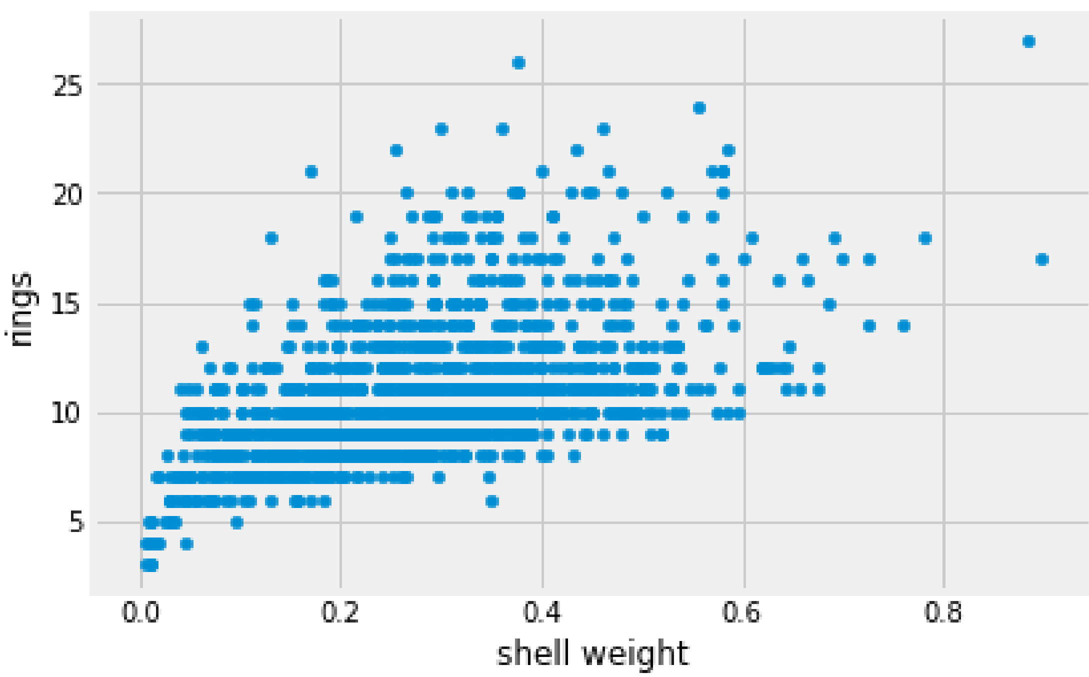


25
20
15
10
5
rings
0.0
0.1
0.2
0.3
0.4
0.5
height



25
20
15
10
5
rings
0.0
0.5
1.0
1.5
2.0
2.5
whole weight

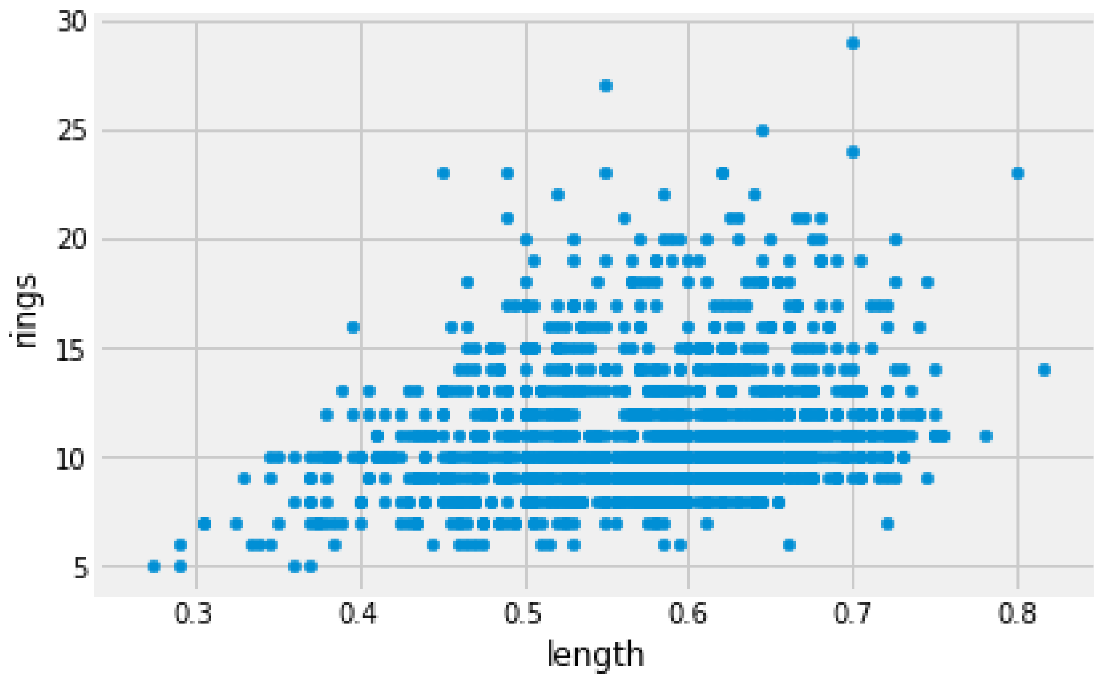


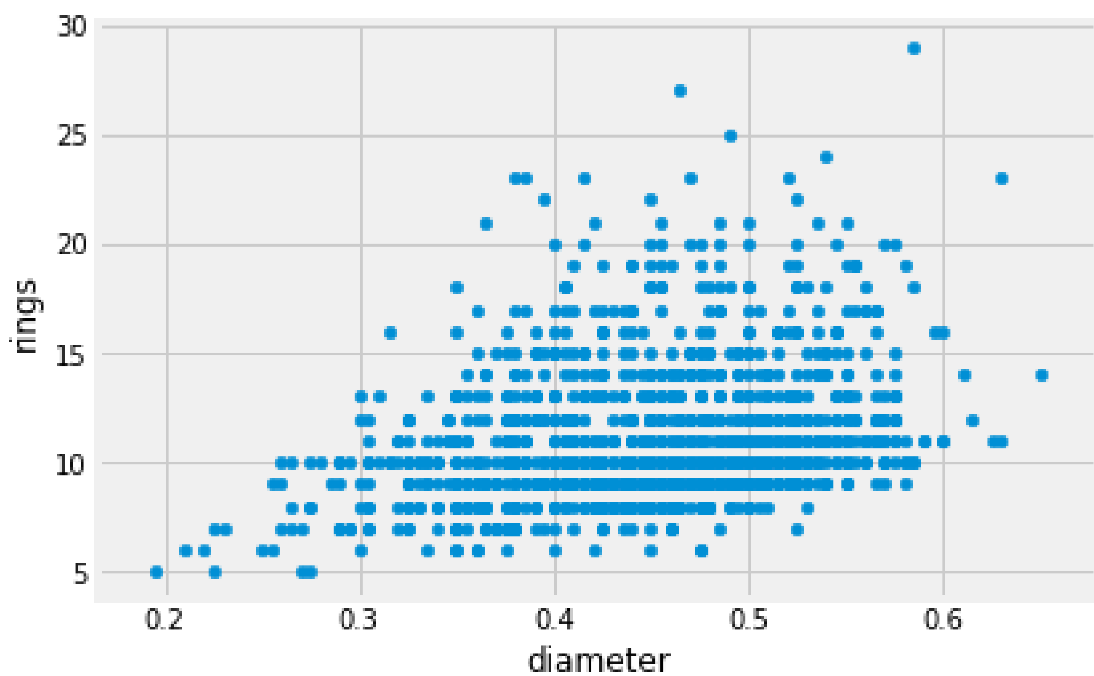


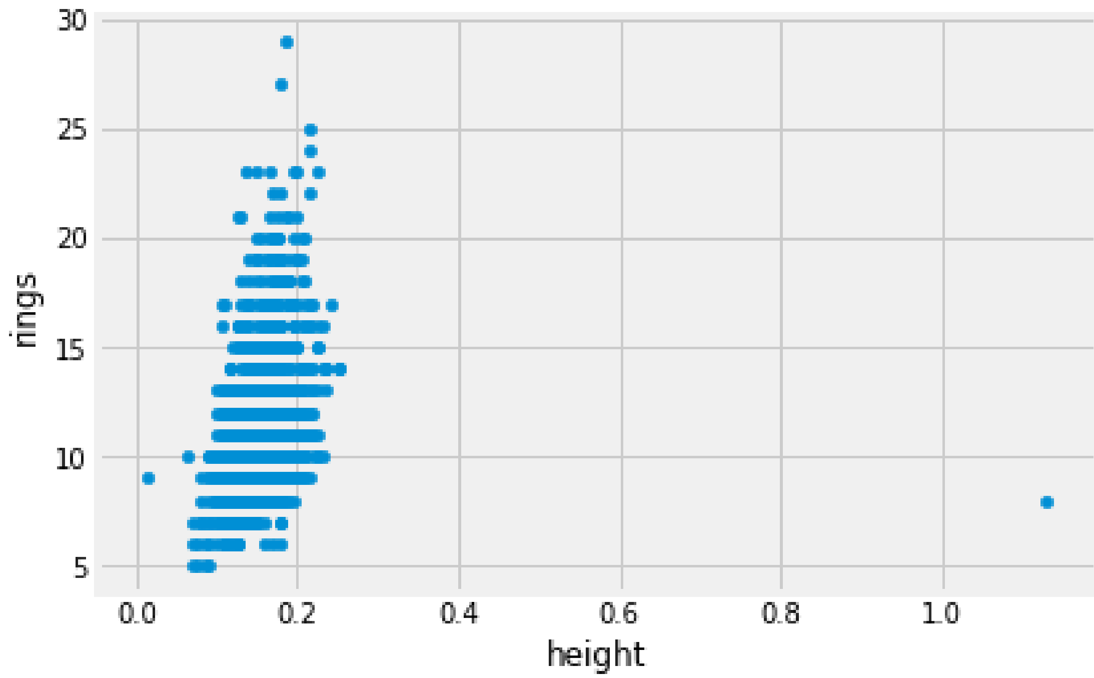

```
In [8]: tableF.plot.scatter('length', 'rings', s=None, c=None)
        tableF.plot.scatter('diameter', 'rings', s=None, c=None)
```

```
tableF.plot.scatter('height', 'rings', s=None, c=None)
tableF.plot.scatter('whole weight', 'rings', s=None, c=None)
tableF.plot.scatter('viscera weight', 'rings', s=None, c=None)
tableF.plot.scatter('shell weight', 'rings', s=None, c=None)
```

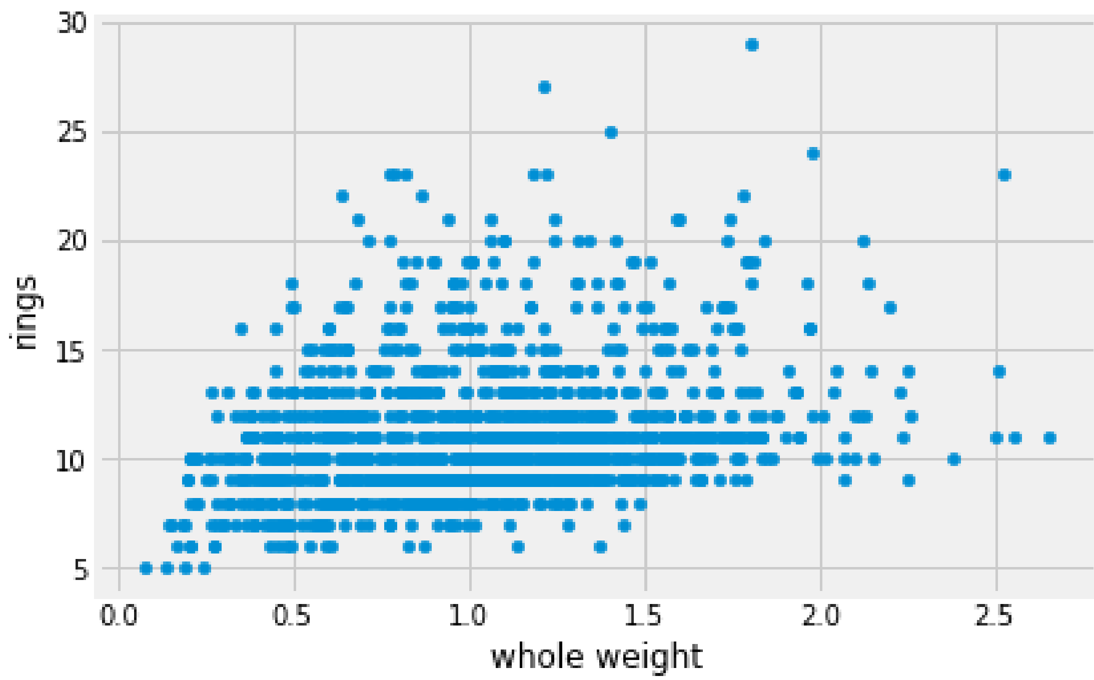
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1120dff98>



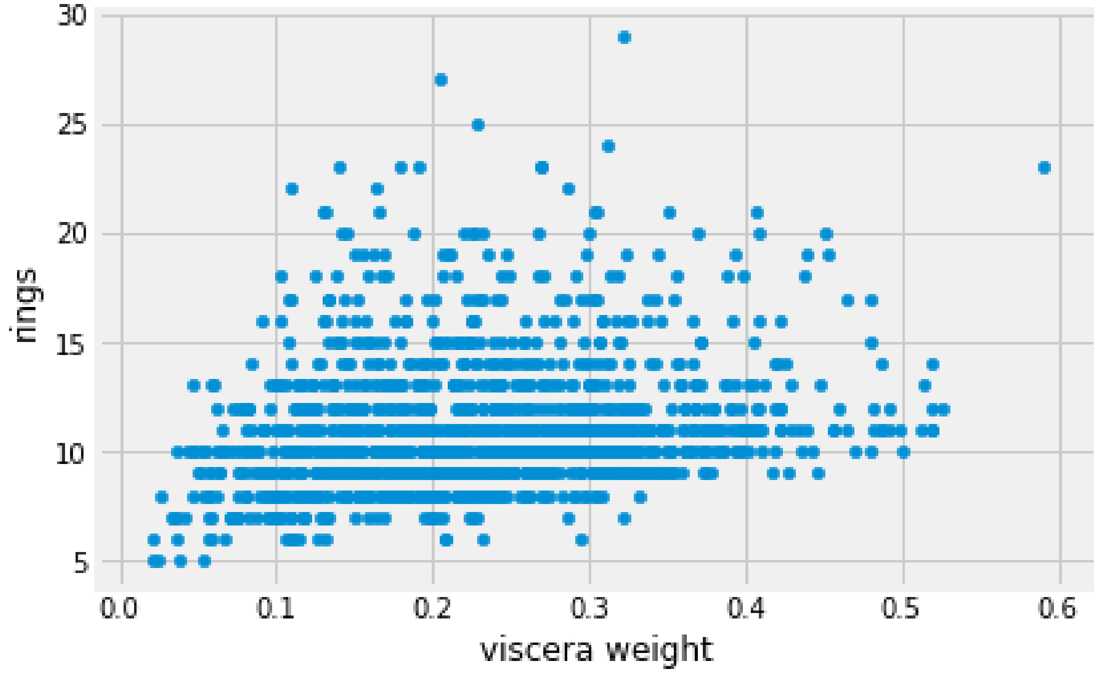


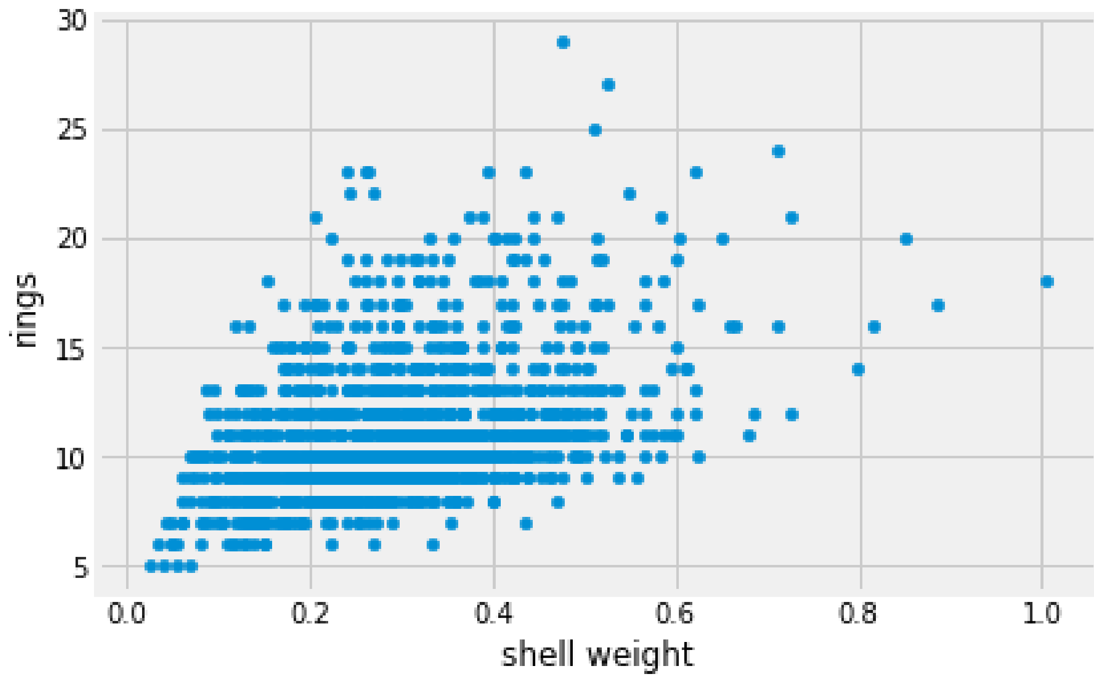


30
25
20
15
10
5
rings
0.0
0.2
0.4
0.6
0.8
1.0
height



30
25
20
15
10
5
rings
0.0
0.5
1.0
1.5
2.0
2.5
whole weight

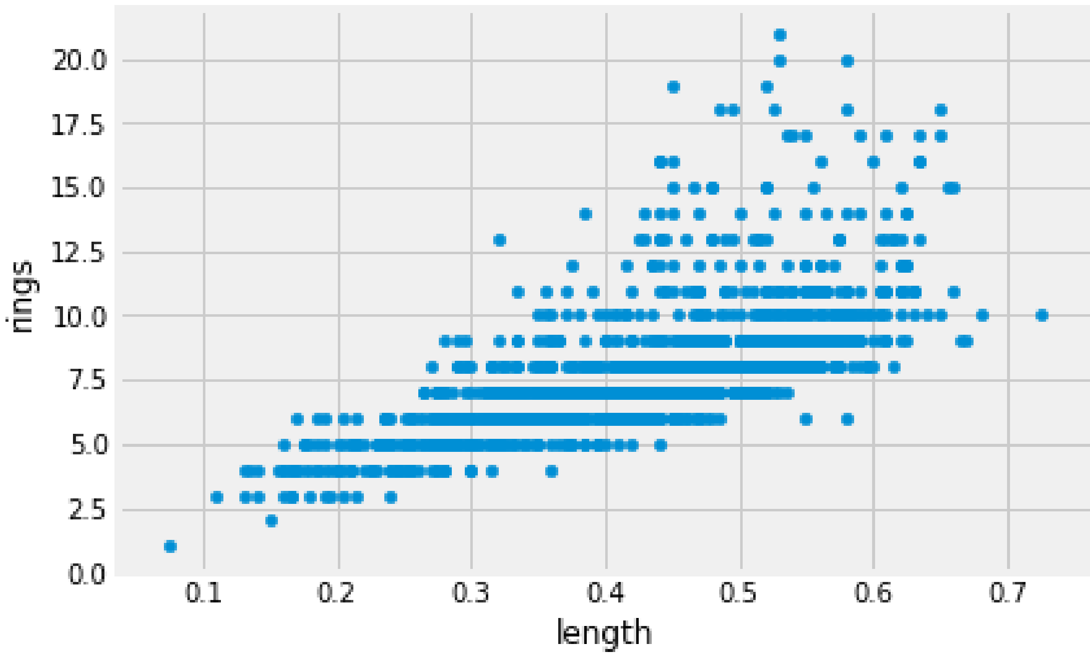


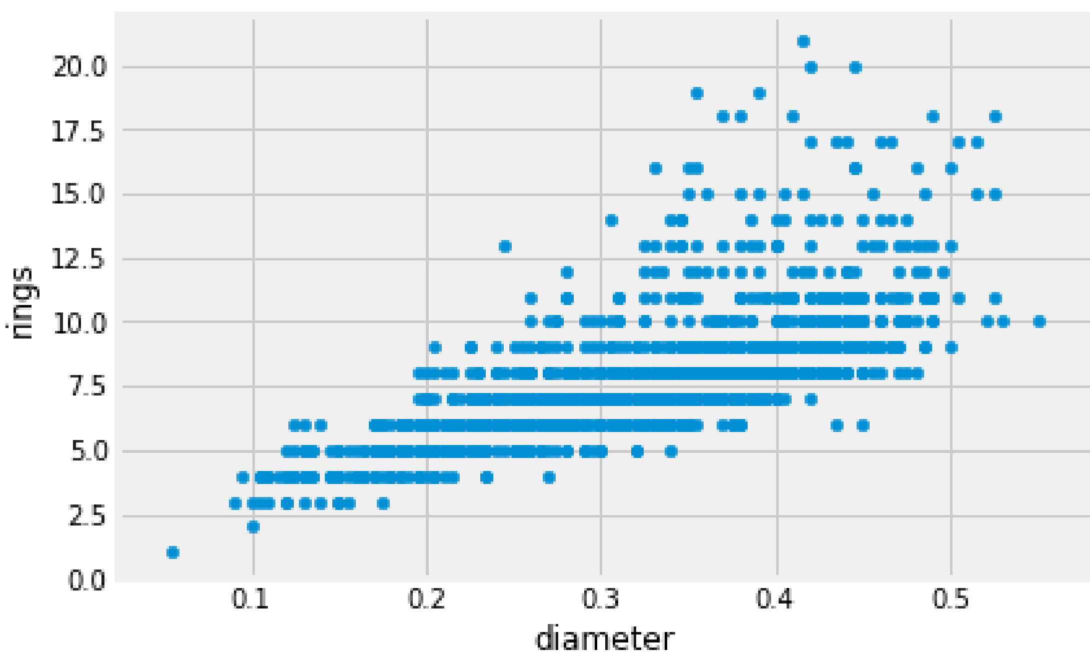


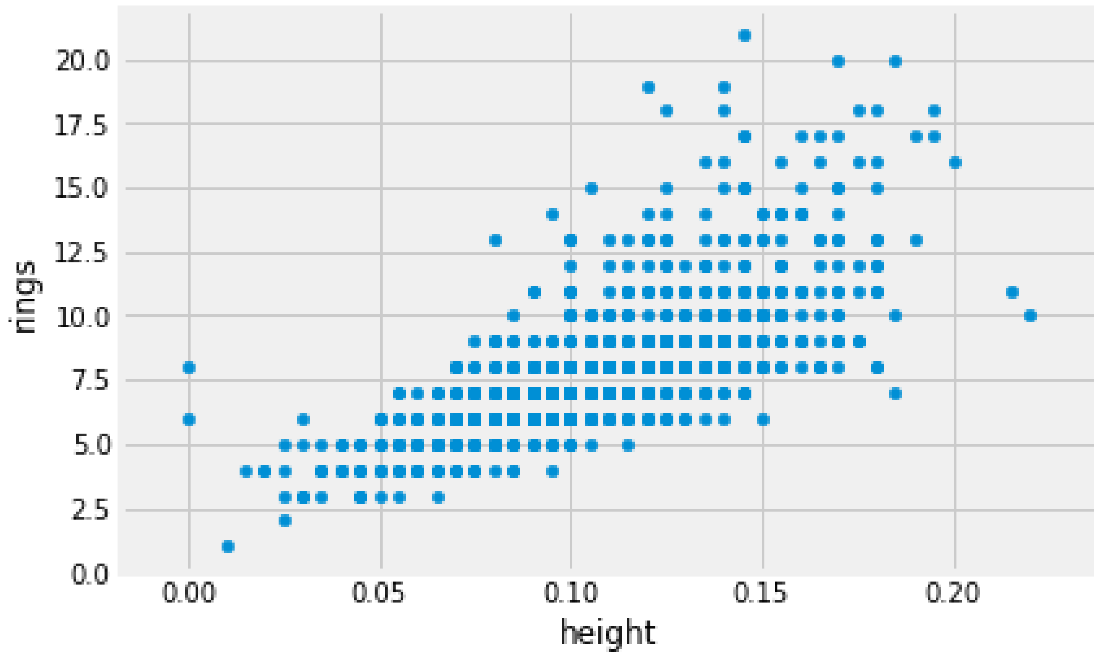

```
In [9]: tableI.plot.scatter('length', 'rings', s=None, c=None)
        tableI.plot.scatter('diameter', 'rings', s=None, c=None)
```

```
tableI.plot.scatter('height', 'rings', s=None, c=None)
tableI.plot.scatter('whole weight', 'rings', s=None, c=None)
tableI.plot.scatter('viscera weight', 'rings', s=None, c=None)
tableI.plot.scatter('shell weight', 'rings', s=None, c=None)
```

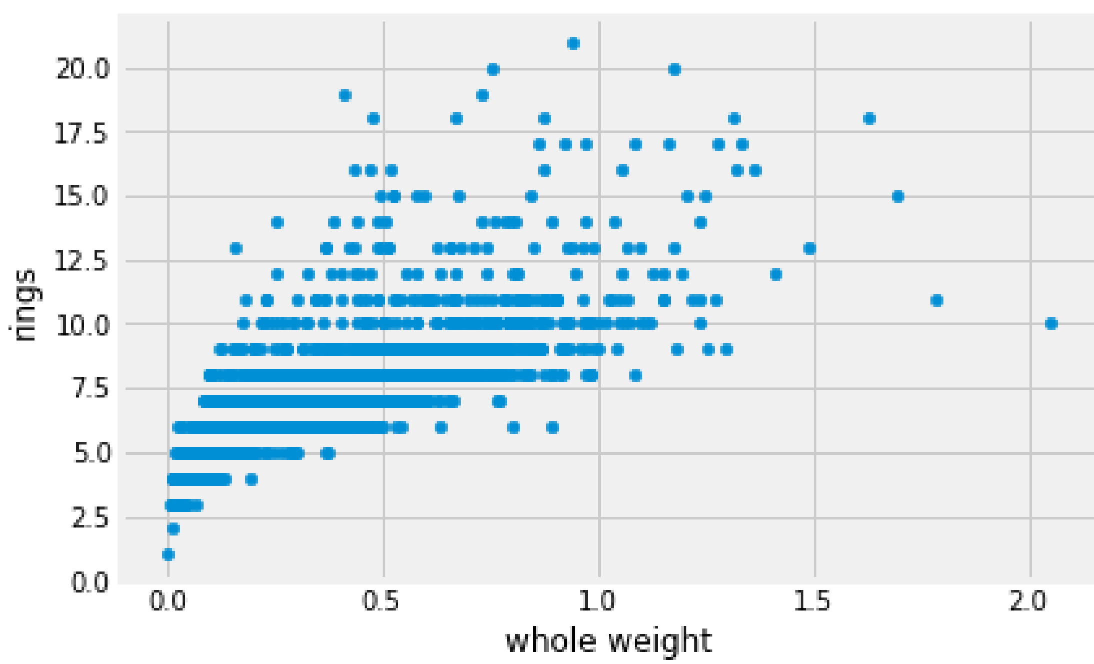
Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x11260add8>



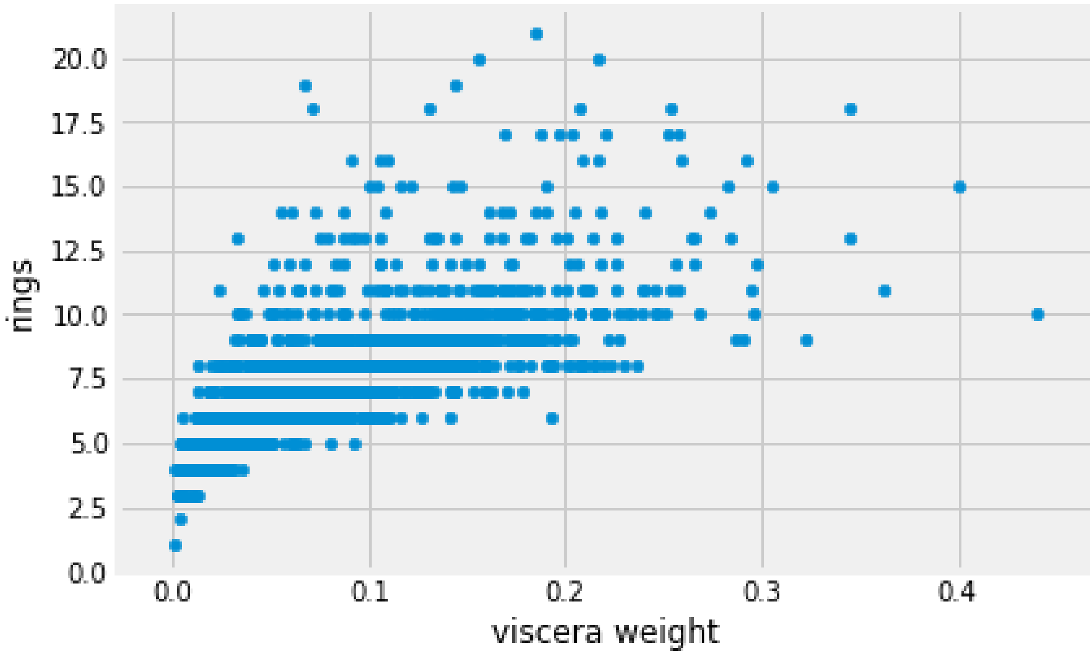


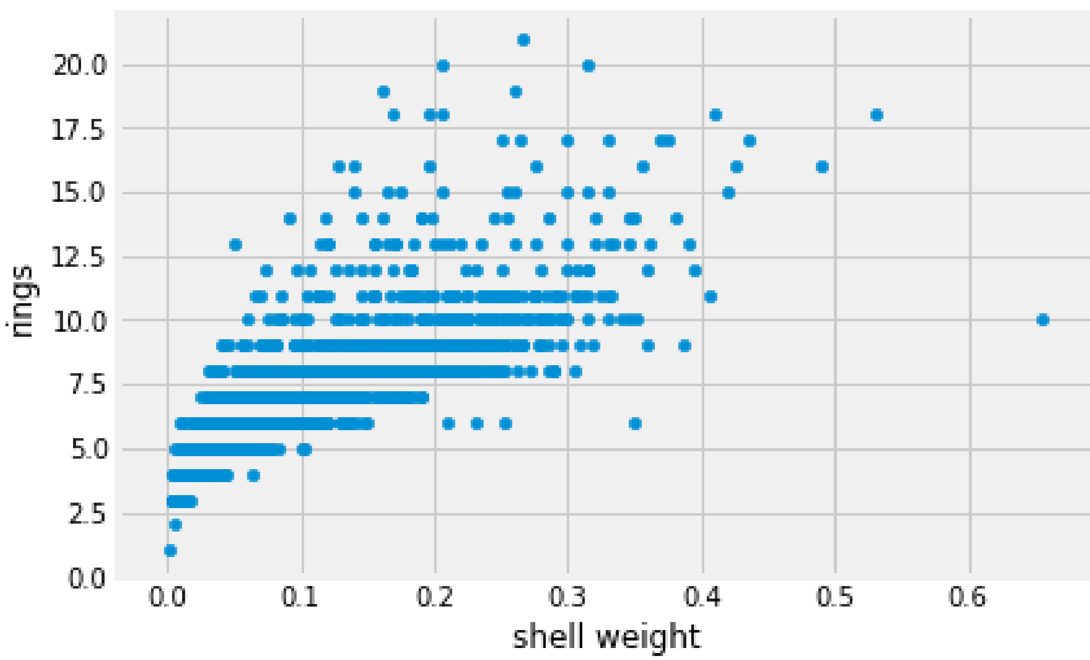


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
rings
0.00
0.05
0.10
0.15
0.20
height



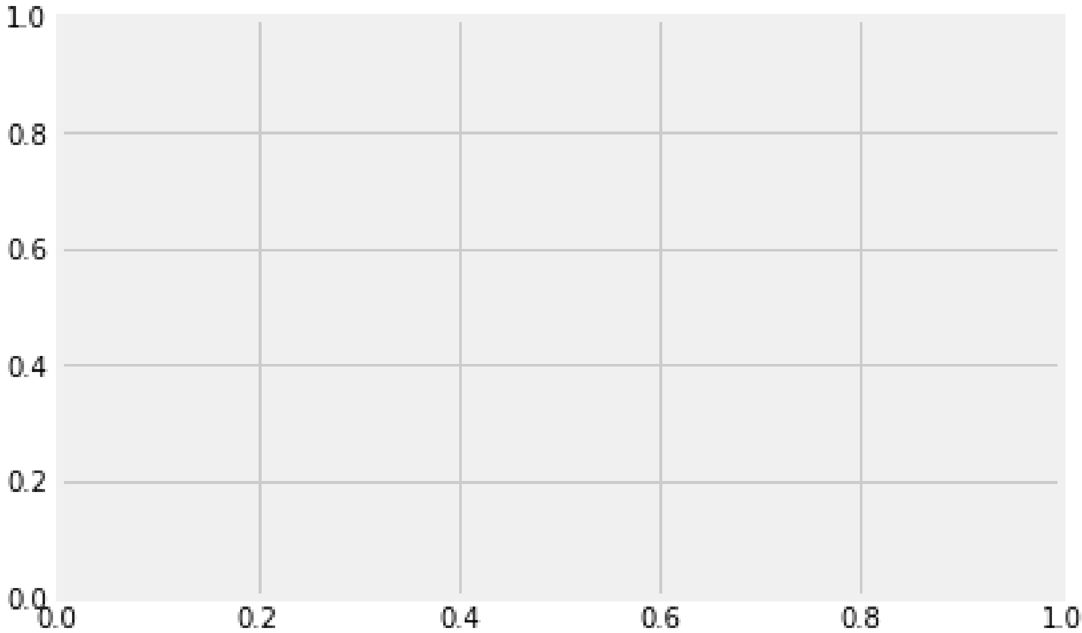
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
rings
0.0
0.5
1.0
1.5
2.0
whole weight

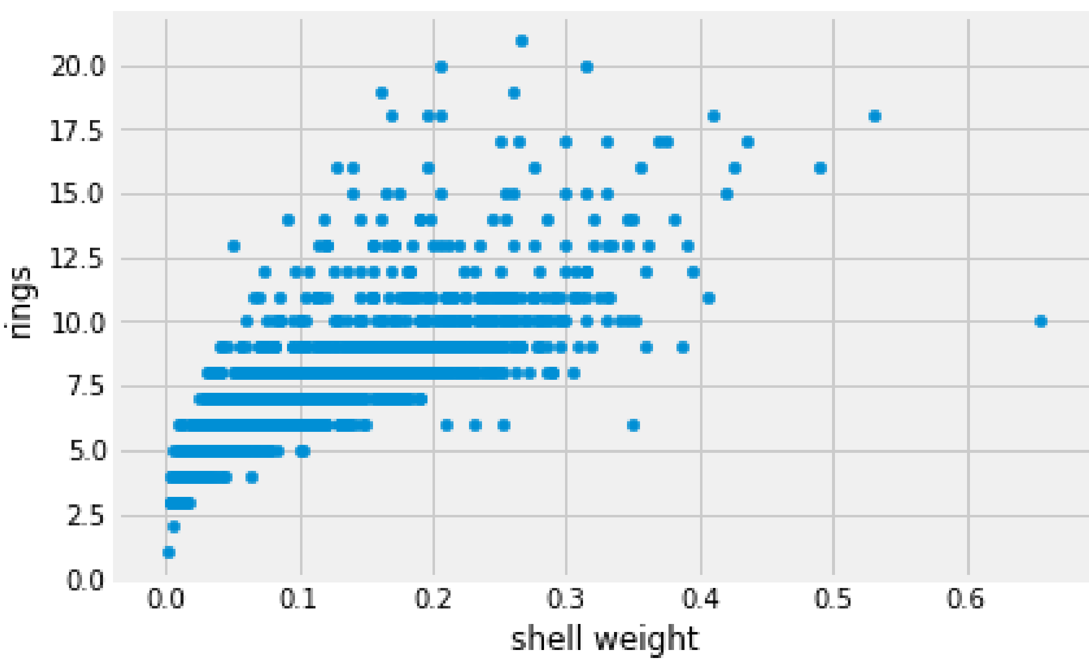





```
In [10]: fig = plt.figure()
         ax1 = fig.add_subplot(111)
         tableI.plot.scatter('shell weight', 'rings', s=None, c=None)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x1129d18d0>






```
In [11]: tableM.replace(to_replace = [np.inf, -np.inf], value = np.nan)
         tableM = tableM.dropna(axis = 0, how ='any')
```

```
tableM
#np.all(np.isfinite(newpoor1))
#np.any(np.isnan(newpoor1))
```

Out[11]:

```
      sex  length  diameter  height  whole weight  shucked weight  \
0       M   0.455     0.365   0.095        0.5140          0.2245
1       M   0.350     0.265   0.090        0.2255          0.0995
3       M   0.440     0.365   0.125        0.5160          0.2155
8       M   0.475     0.370   0.125        0.5095          0.2165
11      M   0.430     0.350   0.110        0.4060          0.1675
12      M   0.490     0.380   0.135        0.5415          0.2175
15      M   0.500     0.400   0.130        0.6645          0.2580
18      M   0.365     0.295   0.080        0.2555          0.0970
19      M   0.450     0.320   0.100        0.3810          0.1705
20      M   0.355     0.280   0.095        0.2455          0.0955
27      M   0.590     0.445   0.140        0.9310          0.3560
28      M   0.605     0.475   0.180        0.9365          0.3940
29      M   0.575     0.425   0.140        0.8635          0.3930
30      M   0.580     0.470   0.165        0.9975          0.3935
32      M   0.665     0.525   0.165        1.3380          0.5515
35      M   0.465     0.355   0.105        0.4795          0.2270
39      M   0.355     0.290   0.090        0.3275          0.1340
46      M   0.470     0.370   0.120        0.5795          0.2930
51      M   0.400     0.320   0.095        0.3030          0.1335
52      M   0.485     0.360   0.130        0.5415          0.2595
54      M   0.405     0.310   0.100        0.3850          0.1730
56      M   0.445     0.350   0.120        0.4425          0.1920
57      M   0.470     0.385   0.135        0.5895          0.2765
60      M   0.450     0.345   0.105        0.4115          0.1800
61      M   0.505     0.405   0.110        0.6250          0.3050
63      M   0.425     0.325   0.095        0.3785          0.1705
64      M   0.520     0.400   0.120        0.5800          0.2340
65      M   0.475     0.355   0.120        0.4800          0.2340
70      M   0.555     0.425   0.130        0.7665          0.2640
73      M   0.570     0.480   0.175        1.1850          0.4740
...    ..     ...       ...     ...           ...             ...
4099    M   0.670     0.525   0.180        1.4915          0.7280
4102    M   0.680     0.545   0.185        1.6720          0.7075
4103    M   0.700     0.545   0.215        1.9125          0.8825
4109    M   0.480     0.365   0.130        0.5305          0.2405
4110    M   0.510     0.410   0.155        1.2825          0.5690
4116    M   0.625     0.480   0.160        1.2415          0.6575
4118    M   0.650     0.525   0.185        1.3455          0.5860
4120    M   0.350     0.265   0.090        0.1970          0.0730
4126    M   0.550     0.420   0.145        0.7385          0.3210
4128    M   0.555     0.435   0.145        0.9205          0.4040
4130    M   0.580     0.450   0.140        0.8240          0.3465
4133    M   0.585     0.450   0.150        0.9970          0.4055
```

4137	M	0.630	0.505	0.155	1.1050	0.4920
4138	M	0.630	0.490	0.155	1.2290	0.5350
4142	M	0.655	0.525	0.180	1.4020	0.6240
4144	M	0.670	0.535	0.190	1.6690	0.7465
4145	M	0.670	0.525	0.200	1.7405	0.6205
4146	M	0.695	0.530	0.210	1.5100	0.6640
4147	M	0.695	0.550	0.195	1.6645	0.7270
4148	M	0.770	0.605	0.175	2.0505	0.8005
4156	M	0.475	0.370	0.110	0.4895	0.2185
4157	M	0.475	0.360	0.140	0.5135	0.2410
4162	M	0.385	0.255	0.100	0.3175	0.1370
4167	M	0.500	0.380	0.125	0.5770	0.2690
4169	M	0.520	0.385	0.165	0.7910	0.3750
4170	M	0.550	0.430	0.130	0.8395	0.3155
4171	M	0.560	0.430	0.155	0.8675	0.4000
4173	M	0.590	0.440	0.135	0.9660	0.4390
4174	M	0.600	0.475	0.205	1.1760	0.5255
4176	M	0.710	0.555	0.195	1.9485	0.9455

	viscera weight	shell weight	rings
0	0.1010	0.1500	15
1	0.0485	0.0700	7
3	0.1140	0.1550	10
8	0.1125	0.1650	9
11	0.0810	0.1350	10
12	0.0950	0.1900	11
15	0.1330	0.2400	12
18	0.0430	0.1000	7
19	0.0750	0.1150	9
20	0.0620	0.0750	11
27	0.2340	0.2800	12
28	0.2190	0.2950	15
29	0.2270	0.2000	11
30	0.2420	0.3300	10
32	0.3575	0.3500	18
35	0.1240	0.1250	8
39	0.0860	0.0900	9
46	0.2270	0.1400	9
51	0.0600	0.1000	7
52	0.0960	0.1600	10
54	0.0915	0.1100	7
56	0.0955	0.1350	8
57	0.1200	0.1700	8
60	0.1125	0.1350	7
61	0.1600	0.1750	9
63	0.0800	0.1000	7
64	0.1315	0.1850	8
65	0.1015	0.1350	8

```
70             0.1680        0.2750     13
73             0.2610        0.3800     11
...               ...           ...    ...
4099           0.3430        0.3810      9
4102           0.3640        0.4800     11
4103           0.4385        0.5060     10
4109           0.1270        0.1390      8
4110           0.2910        0.3795      9
4116           0.2625        0.2785      9
4118           0.2780        0.3865      9
4120           0.0365        0.0770      7
4126           0.1485        0.2520     11
4128           0.2275        0.2550      8
4130           0.1765        0.2630     10
4133           0.2830        0.2510     11
4137           0.2260        0.3250     11
4138           0.2900        0.3350     11
4142           0.2935        0.3650     13
4144           0.2935        0.5080     11
4145           0.2970        0.6570     11
4146           0.4095        0.3850     10
4147           0.3600        0.4450     11
4148           0.5260        0.3550     11
4156           0.1070        0.1460      8
4157           0.1045        0.1550      8
4162           0.0680        0.0920      8
4167           0.1265        0.1535      9
4169           0.1800        0.1815     10
4170           0.1955        0.2405     10
4171           0.1720        0.2290      8
4173           0.2145        0.2605     10
4174           0.2875        0.3080      9
4176           0.3765        0.4950     12

[1528 rows x 9 columns]
```

In [12]:
```
x = tableM['shell weight'].values[:np.newaxis]
y = tableM['rings']

# Reshaping
x, y = x.reshape(-1,1), y.reshape(-1, 1)

# Linear Regression Object
lin_regression = LinearRegression()

# Fitting linear model to the data
lin_regression.fit(x,y)
```

```
    # Get slope of fitted line
    m = lin_regression.coef_

    # Get y-Intercept of the Line
    b = lin_regression.intercept_

    # Get Predictions for original x values
    # you can also get predictions for new data
    predictions = lin_regression.predict(x)

    print (b)
    print (m)
```

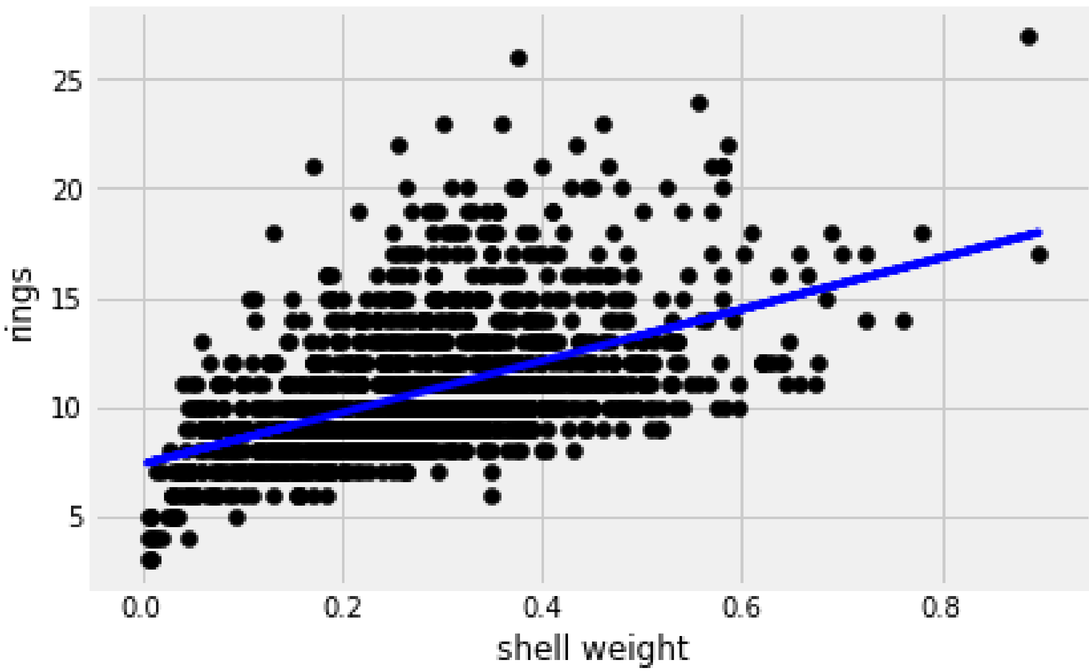
```
[7.37262799]
[[11.81997505]]
```

```
/Users/javier/Envs/data-x/lib/python3.6/site-packages/scipy/linalg/basic.py:1226: RuntimeWarnin
  warnings.warn(mesg, RuntimeWarning)
```

```
In [13]: print("""I ran a few multiple different scatterplots. I was really curious to see how
         I found that the variables that correlated the highest with rings were length and dia
         After running linear regression, I found that regression equation to be rings = 11.82
         """)
```

```
I ran a few multiple different scatterplots. I was really curious to see how the rings of an a
I found that the variables that correlated the highest with rings were length and diameter. Th
After running linear regression, I found that regression equation to be rings = 11.82*(shell w
```

```
In [14]: plt.scatter(x, y,  color='black')
         a =plt.plot(x, predictions, color='blue',linewidth=3)
         plt.xlabel('shell weight')
         plt.ylabel('rings')
         plt.show()
```




```
In [15]: #draw a scatterplot
         def scatter(table, xcol, ycol, marker_color='blue'):

             #Cleaning missing and invalid data in table
             table.replace(to_replace = [np.inf, -np.inf], value = np.nan)
             table = table.dropna(axis = 0, how ='any')

             #Assigning axes
             x = table[xcol].values[:np.newaxis]
             y = table[ycol]

             # Reshaping
             x, y = x.reshape(-1,1), y.reshape(-1, 1)

             plt.scatter(x, y,  color = marker_color)


         #Regress on a scatterplot with xcol and ycol (column names - str) from table
         def scatter_and_regress(table, xcol, ycol, marker_color='blue'):

             #Cleaning missing and invalid data in table
             table.replace(to_replace = [np.inf, -np.inf], value = np.nan)
             table = table.dropna(axis = 0, how ='any')

             #Assigning axes
```

```python
    x = table[xcol].values[:np.newaxis]
    y = table[ycol]

    # Reshaping
    x, y = x.reshape(-1,1), y.reshape(-1, 1)

    # Linear Regression Object
    lin_regression = LinearRegression()

    # Fitting linear model to the data
    lin_regression.fit(x,y)

    # Get slope of fitted line
    m = lin_regression.coef_

    # Get y-Intercept of the Line
    b = lin_regression.intercept_

    # Get Predictions for original x values
    predictions = lin_regression.predict(x)

    plt.scatter(x, y,  color = marker_color)
    plt.plot(x, predictions, color='black',linewidth=3)
    plt.xlabel(xcol)
    plt.ylabel(ycol)
    plt.show()
```

In [16]:
```python
KNN = table.loc[:, ['sex', 'rings', 'whole weight', 'shell weight', 'viscera weight']]
KNN.head(3)

#reshuffle rows and divide into 2 sets - training and testing sets
```

Out[16]:

	sex	rings	whole weight	shell weight	viscera weight
0	M	15	0.5140	0.15	0.1010
1	M	7	0.2255	0.07	0.0485
2	F	9	0.6770	0.21	0.1415

In [17]:
```python
def cutoff_and_above(df, column, cutoff_value):
    """For each row, return True if the value in the column is equal to and above the
    classified = (df[column]>=cutoff_value)

    #Showing number of True (equal and above) and False values
    print(classified.value_counts())
    return classified


def color_code(bool_array):
    """Return a color-coded array: 'Blue' for True values; 'Red' for False"""
```

```
    color = bool_array.apply(lambda row: 'Blue' if row == 'M' else 'Red' if row == 'F
    return color


def scatter_and_colorcode(table, xcol, ycol, color_col):
    """Draw a scatterplot w.r.t. Color column in table"""
    color = table[color_col]
    scatter(table, xcol, ycol, color)

def distance_two_features(df, x_feature, y_feature):
    x1-x2, y1-y2
    """Compute the distance between x_feature and y_feature"""
    x = df[x_feature]
    y = df[y_feature]
    return np.sqrt(- rotem(x_feature))**2 + (row0.item(y_feature)-row1.item(y_feature
```

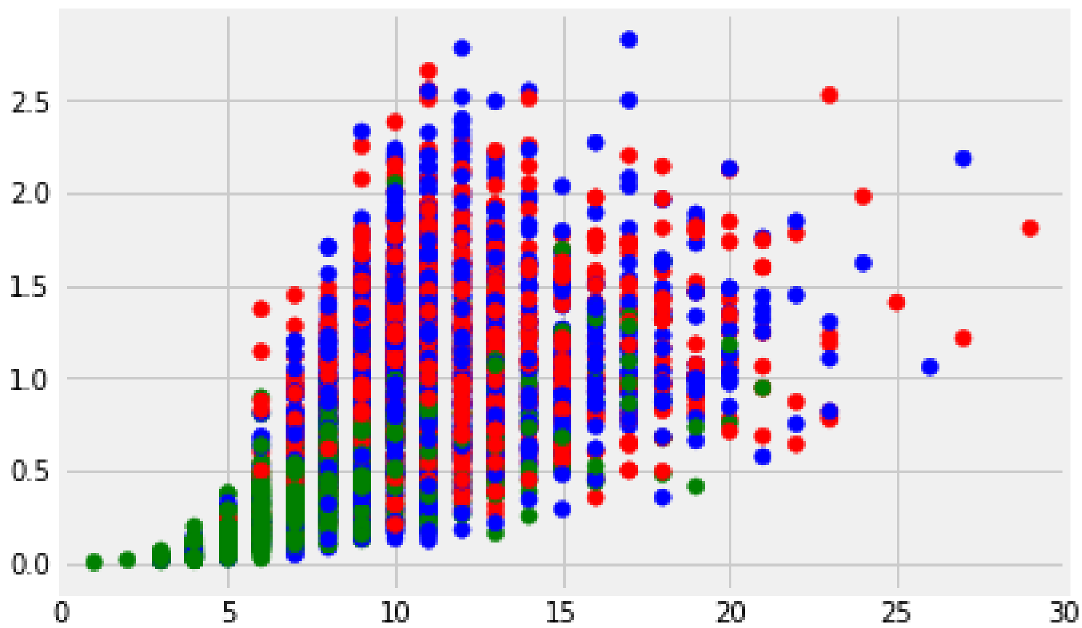
In [18]:
```
KNN['Color'] = color_code(table['sex'])
KNN.head(3)
```

Out[18]:

	sex	rings	whole weight	shell weight	viscera weight	Color
0	M	15	0.5140	0.15	0.1010	Blue
1	M	7	0.2255	0.07	0.0485	Blue
2	F	9	0.6770	0.21	0.1415	Red

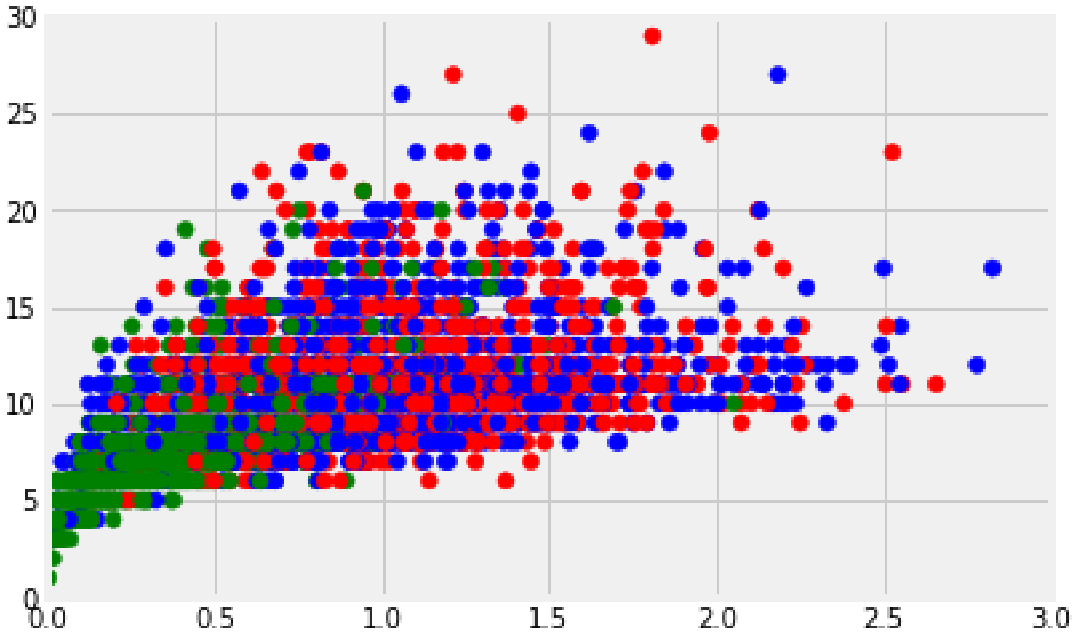
In [19]:
```
scatter_and_colorcode(KNN, 'rings','whole weight', 'Color')
plt.xlim(0,30)
```

Out[19]: (0, 30)



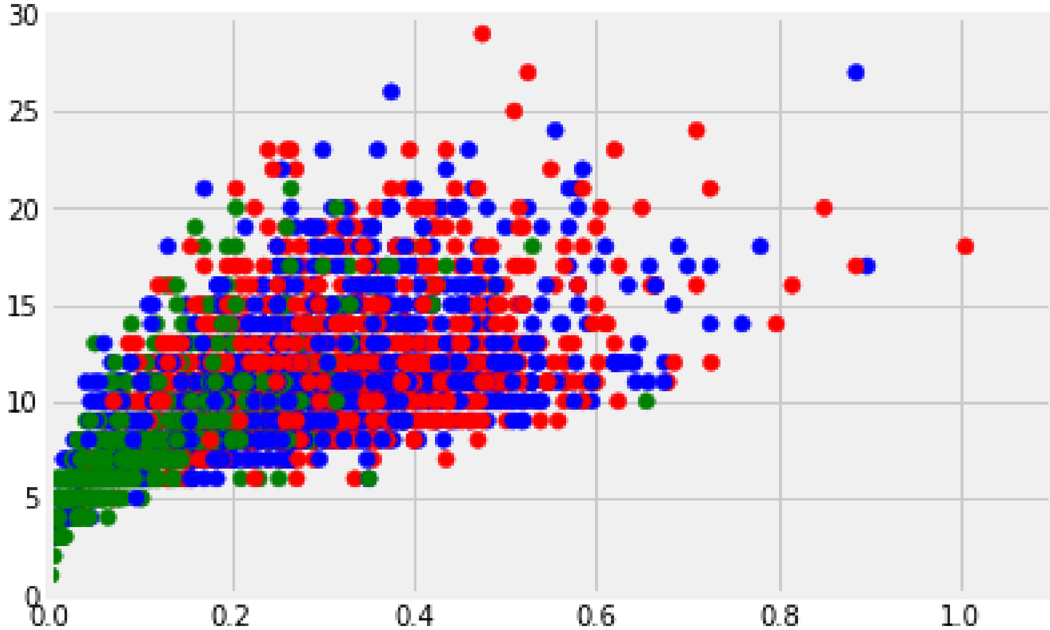
In [20]:
```
scatter_and_colorcode(KNN, 'whole weight','rings', 'Color')
plt.xlim(0,3)
plt.ylim(0,30)
```

Out[20]: (0, 30)



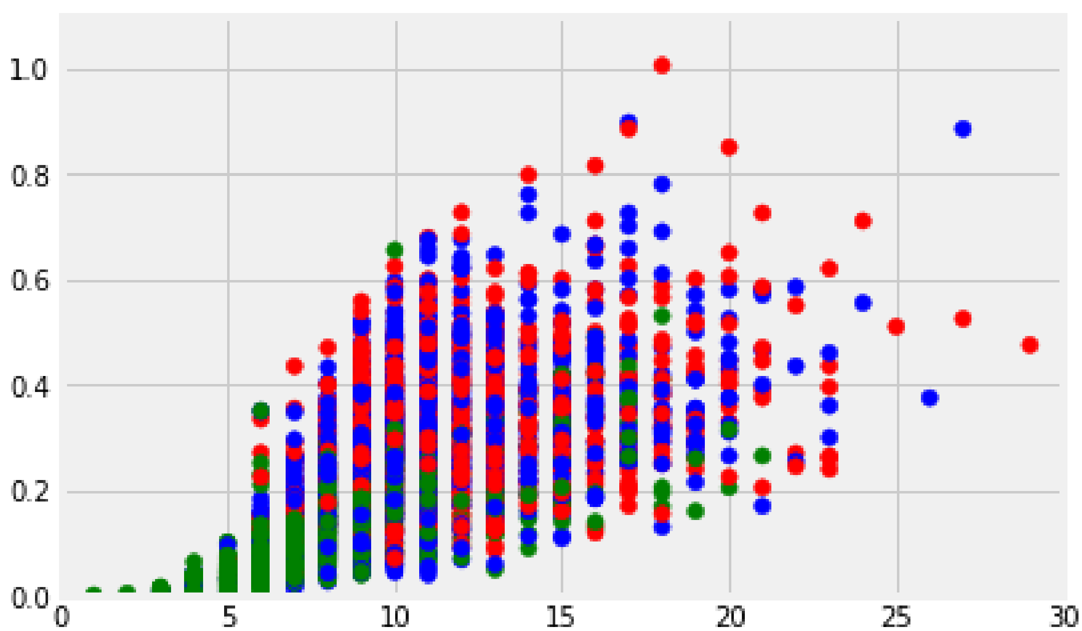
In [21]:
```
scatter_and_colorcode(KNN, 'shell weight','rings', 'Color')
plt.xlim(0,1.1)
plt.ylim(0,30)
```

Out[21]: (0, 30)



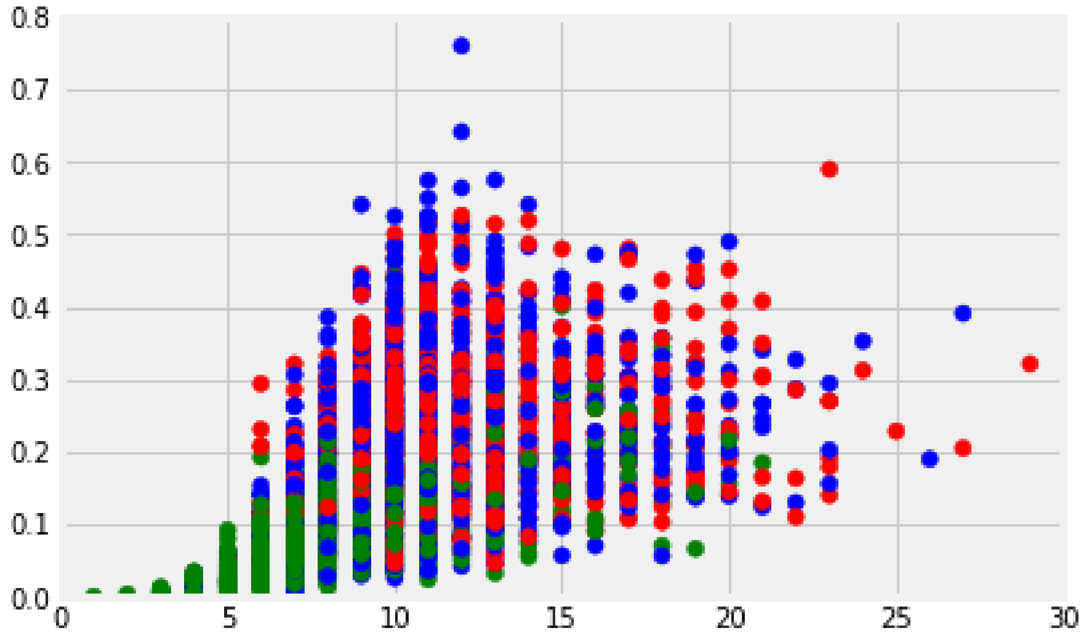
In [22]:
```
scatter_and_colorcode(KNN, 'rings', 'shell weight', 'Color')
plt.xlim(0,30)
plt.ylim(0,1.1)
```

Out[22]: (0, 1.1)



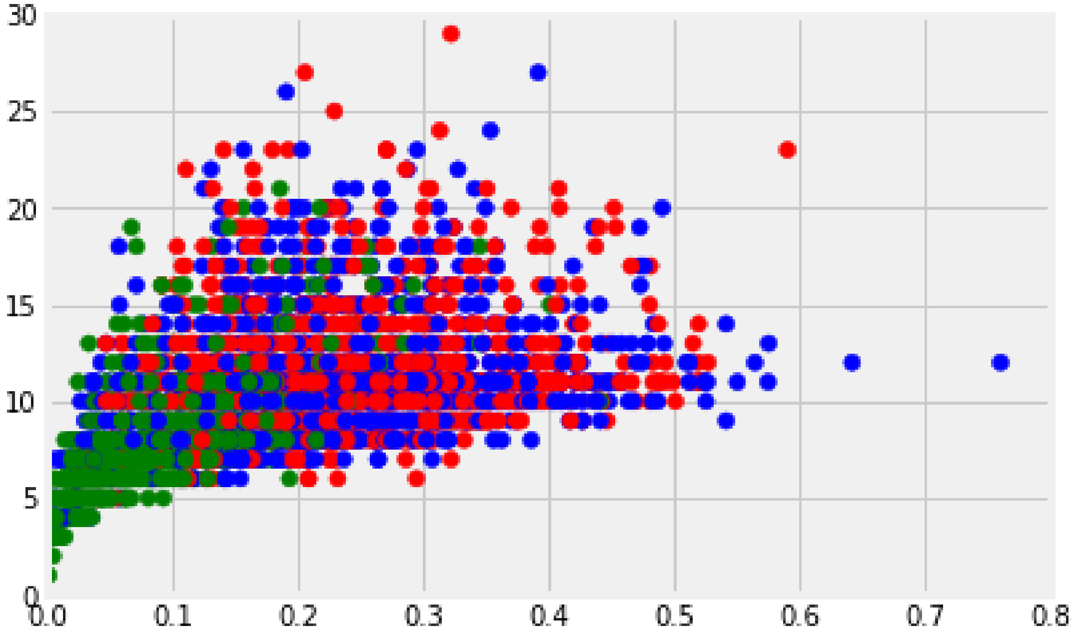
```
In [23]: scatter_and_colorcode(KNN, 'rings', 'viscera weight', 'Color')
         plt.xlim(0,30)
         plt.ylim(0,0.8)
```

```
Out[23]: (0, 0.8)
```




```
In [24]: scatter_and_colorcode(KNN, 'viscera weight', 'rings', 'Color')
         plt.xlim(0,0.8)
         plt.ylim(0,30)
```

```
Out[24]: (0, 30)
```




```
In [25]: print('The second model that I use is K nearest neighbors. I wanted to see how viscera
```

```
The second model that I use is K nearest neighbors. I wanted to see how viscera weight specifi
```

```
In [26]: KNN = tableF.loc[:, ['rings', 'viscera weight']]
         #Cleaning missing and invalid data in table
         KNN.replace(to_replace = [np.inf, -np.inf], value = np.nan)
         KNN = table.dropna(axis = 0, how ='any')

         #Sourcecode: https://www.dataquest.io/blog/k-nearest-neighbors-in-python/
         import random
         import math
         from numpy.random import permutation

         # Randomly shuffle the index of KNN.
         random_indices = permutation(KNN.index)
         # Divide the data into half for training and testing set
         test_cutoff = math.floor(len(KNN)/2)
         # Generate the test set by taking the first 1/2 of the randomly shuffled indices.
         test = KNN.loc[random_indices[1:test_cutoff]]
         # Generate the train set with the rest of the data.
         train = KNN.loc[random_indices[test_cutoff:]]
         # The columns that we will be making predictions with.
         x_columns = ['viscera weight']
         # The column that we want to predict.
```

```python
y_column = ['rings']

from sklearn.neighbors import KNeighborsRegressor
# Create the knn model.
# Look at the five closest neighbors.
knn = KNeighborsRegressor(n_neighbors=5)
# Fit the model on the training data.
knn.fit(train[x_columns], train[y_column])
# Make point predictions on the test set using the fit model.
predictions = knn.predict(test[x_columns])

# Get the actual values for the test set.
actual = test[y_column]
Actual_vs_Predictions = {'Actual': actual, 'Predicted': predictions}
print(Actual_vs_Predictions)

# Compute the mean squared error of our predictions.
mse = (((predictions - actual) ** 2).sum()) / len(predictions)
print(mse)
```

```
{'Actual':       rings
2535     11
821       6
1008     10
1729     11
3833     11
4167      9
3739      9
1755      7
1915     11
3304     16
304      10
926       7
2025     10
155      10
2521      9
361      12
67       13
3591      9
1816     10
4056     11
2140     10
3256     12
2246     10
170      14
2812      5
839       9
2317     13
```

```
806        6
4084      10
1695       8
...      ...
405       12
2556       6
15        12
478       21
475       17
2600      10
856        9
2799      10
292       15
2240      12
2561       6
1900       9
2286       9
1601       9
3136      11
2247       9
2936      11
2379      10
3618      11
693        9
3313      11
573       17
3892      13
2400      13
1907       9
3813       8
3891      12
1598       9
2644       8
3763      11

[2087 rows x 1 columns], 'Predicted': array([[10.6],
       [ 7.8],
       [ 9.4],
       ...,
       [ 7.6],
       [10. ],
       [10.8]])}
rings    8.399655
dtype: float64
```