Data-X HW1 Sp18

February 8, 2018

1 Homework 1

In this homework, you will get a chance to do some exercises with Numpy, Pandas, and Matplotlib to show us your understanding with this libraries.

If you have questions, Google! Additionally you can ask your peers questions on Piazza and/or go to Office Hours.

This homework is due **Thursday Feb. 8th, 2018 at 11:59 PM**. Please upload your .ipynb to your private repo on Github. Additionally, submit a pdf on bCourses and in the comment section include a link to your private repo.

This homework is long, please start early!

```
In [44]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt

         %matplotlib inline
```

1.1 NumPy Basics

Create two numpy arrays (`a` and `b`). `a` should be all integers between 10-19 (inclusive), and `b` should be ten evenly spaced numbers between 1-7. Print the results below.

```
In [45]: a = np.array(range(10,20))
         b = np.linspace(1,7,10)

         print("a: ", a)
         print("b: ", b)
```

```
a:  [10 11 12 13 14 15 16 17 18 19]
b:  [ 1.          1.66666667  2.33333333  3.          3.66666667  4.33333333
  5.          5.66666667  6.33333333  7.        ]
```

For `a` and `b` above do the follow and print out the results.

1. Square all the elements in both arrays (element-wise).

2. Add both the squared arrays (e.g. [1,2] + [3,4] = [4,6]).

3. Sum the elements with even indices of the added array.

4. Take the square root of the added array (element-wise square root).

```
In [46]: a = np.square(a)
         b = np.square(b)
         c = a + b
         d = c[::2]
         e = np.sqrt(c)
         print("1. ",a,"\n", b)
         print("2. ", c)
         print("3. ", d)
         print("4. ", e)
```

```
1.  [100 121 144 169 196 225 256 289 324 361]
 [  1.           2.77777778   5.44444444   9.          13.44444444
  18.77777778  25.          32.11111111  40.11111111  49.        ]
2.  [ 101.          123.77777778  149.44444444  178.          209.44444444
  243.77777778  281.          321.11111111  364.11111111  410.        ]
3.  [ 101.          149.44444444  209.44444444  281.          364.11111111]
4.  [ 10.04987562  11.12554618  12.22474721  13.34166406  14.47219556
  15.61338457  16.76305461  17.91957341  19.08169571  20.24845673]
```

Append `b` to `a`. Reshape the appended array so that it is a 5x4, 2D-array and store the results in a variable called `m`. Print `m`.

```
In [47]: m = np.append(a,b).reshape(5,4)

         print("m: ", m)
```

```
m:  [[ 100.          121.          144.          169.        ]
 [ 196.          225.          256.          289.        ]
 [ 324.          361.            1.            2.77777778]
 [   5.44444444    9.           13.44444444   18.77777778]
 [  25.          32.11111111   40.11111111   49.        ]]
```

Extract the second and the third column of the matrix `m`. Store the resulting 5x2 matrix in a new variable called `m2`. Print `m2`.

```
In [48]: m2 = m[:,1:3].reshape(5,2)

         print("m2: ", m2)
```

```
m2:  [[ 121.          144.        ]
 [ 225.          256.        ]
```

```
 [ 361.          1.        ]
 [   9.         13.44444444]
 [  32.11111111   40.11111111]]
```

Take the dot product of `m2` and `m` store the results in a matrix called `m3`. Print `m3`. Note that dot product of two matrices $A \cdot B = A^T B$

```
In [49]: m3 = np.transpose(m2).dot(m)

         print("m3: ", m3)
```

```
m3:  [[ 174015.77777778  196699.12345679   76794.01234568   88219.22222222]
 [  65975.97530864   76794.01234568   88062.65432099  100540.67901235]]
```

Round the `m3` matrix to two decimal points. Store the result in place and print the new `m3`.

```
In [50]: m3 = np.round(m3, decimals=2)

         print("m3: ", m3)
```

```
m3:  [[ 174015.78  196699.12   76794.01   88219.22]
 [  65975.98   76794.01   88062.65  100540.68]]
```

Sort the `m3` array so that the highest value is at the top left, the next highest value to the right of the highest, and the lowest value is at the bottom right. Print the sorted `m3` array.

```
In [51]: sorted_m3 = np.sort(m3, axis= 0,)
         sorted_m3 = -np.sort(-m3)
         print("sorted m3: ", sorted_m3)
```

```
sorted m3:  [[ 196699.12  174015.78   88219.22   76794.01]
 [ 100540.68   88062.65   76794.01   65975.98]]
```

1.2 NumPy and Masks

Create an array called `f` where there are 100 equally-spaced values from 0 to pi, inclusive. Take the sin of the array `f` (element-wise) and store that in place. Print `f`.

```
In [63]: f = np.linspace(0,np.pi,100)
         f = np.sin(f)
         print("f: ", f)
```

```
f:  [  0.00000000e+00   3.17279335e-02   6.34239197e-02   9.50560433e-02
   1.26592454e-01   1.58001396e-01   1.89251244e-01   2.20310533e-01
   2.51147987e-01   2.81732557e-01   3.12033446e-01   3.42020143e-01
   3.71662456e-01   4.00930535e-01   4.29794912e-01   4.58226522e-01
```

```
 4.86196736e-01   5.13677392e-01   5.40640817e-01   5.67059864e-01
 5.92907929e-01   6.18158986e-01   6.42787610e-01   6.66769001e-01
 6.90079011e-01   7.12694171e-01   7.34591709e-01   7.55749574e-01
 7.76146464e-01   7.95761841e-01   8.14575952e-01   8.32569855e-01
 8.49725430e-01   8.66025404e-01   8.81453363e-01   8.95993774e-01
 9.09631995e-01   9.22354294e-01   9.34147860e-01   9.45000819e-01
 9.54902241e-01   9.63842159e-01   9.71811568e-01   9.78802446e-01
 9.84807753e-01   9.89821442e-01   9.93838464e-01   9.96854776e-01
 9.98867339e-01   9.99874128e-01   9.99874128e-01   9.98867339e-01
 9.96854776e-01   9.93838464e-01   9.89821442e-01   9.84807753e-01
 9.78802446e-01   9.71811568e-01   9.63842159e-01   9.54902241e-01
 9.45000819e-01   9.34147860e-01   9.22354294e-01   9.09631995e-01
 8.95993774e-01   8.81453363e-01   8.66025404e-01   8.49725430e-01
 8.32569855e-01   8.14575952e-01   7.95761841e-01   7.76146464e-01
 7.55749574e-01   7.34591709e-01   7.12694171e-01   6.90079011e-01
 6.66769001e-01   6.42787610e-01   6.18158986e-01   5.92907929e-01
 5.67059864e-01   5.40640817e-01   5.13677392e-01   4.86196736e-01
 4.58226522e-01   4.29794912e-01   4.00930535e-01   3.71662456e-01
 3.42020143e-01   3.12033446e-01   2.81732557e-01   2.51147987e-01
 2.20310533e-01   1.89251244e-01   1.58001396e-01   1.26592454e-01
 9.50560433e-02   6.34239197e-02   3.17279335e-02   1.22464680e-16]
```

Use a 'mask' and print an array that is True when `f` >= 1/2 and False when `f` < 1/2. Print an array sequence that has only those values where `f` >= 1/2.

```
In [64]: mask = (f >= 1/2)
         for i in range(len(mask)):
             if mask[i]:
                 print(f[i])
```

```
0.513677391573
0.540640817456
0.567059863863
0.592907929055
0.618158986221
0.642787609687
0.666769000516
0.690079011482
0.712694171379
0.734591708658
0.755749574354
0.776146464292
0.795761840531
0.81457595205
0.832569854635
0.84972542995
0.866025403784
```

0.881453363448
0.895993774291
0.909631995355
0.922354294105
0.934147860265
0.945000818715
0.954902241444
0.96384215856
0.971811568324
0.978802446215
0.984807753012
0.989821441881
0.993838464461
0.996854775952
0.998867339183
0.999874127674
0.999874127674
0.998867339183
0.996854775952
0.993838464461
0.989821441881
0.984807753012
0.978802446215
0.971811568324
0.96384215856
0.954902241444
0.945000818715
0.934147860265
0.922354294105
0.909631995355
0.895993774291
0.881453363448
0.866025403784
0.84972542995
0.832569854635
0.81457595205
0.795761840531
0.776146464292
0.755749574354
0.734591708658
0.712694171379
0.690079011482
0.666769000516
0.642787609687
0.618158986221
0.592907929055
0.567059863863
0.540640817456

```
0.513677391573
```

1.3 NumPy and 2 Variable Prediction

Let x be the number of miles a person drives per day and y be the dollars spent on buying car fuel per day.

We have created 2 numpy arrays each of size 100 that represent x and y.
x (number of miles) ranges from 1 to 10 with a uniform noise of (0, 1/2).
y (money spent in dollars) will be from 1 to 20 with a uniform noise (0, 1).

Run the cell below.

```
In [65]: # seed the random number generator with a fixed value
         np.random.seed(500)

         x=np.linspace(1,10,100)+ np.random.uniform(low=0,high=.5,size=100)
         y=np.linspace(1,20,100)+ np.random.uniform(low=0,high=1,size=100)
         # print ('x = ',x)
         # print ('y= ',y)
```

Find the expected value of x and the expected value of y.

```
In [66]: mean_x = np.mean(x)
         mean_y = np.mean(y)
         print("mean for x and y:", mean_x, mean_y)

mean for x and y: 5.78253254159 11.0129816833
```

Find the variance for x and y.

```
In [67]: var_x = np.var(x)
         var_y = np.var(y)
         # you can set the axis, for matrix, axis = 0, variance across the row
         print("variance for x and y:", var_x, var_y)

variance for x and y: 7.03332752948 30.1139035755
```

Find the co-variance of x and y.

```
In [68]: cov = np.cov(x,y)
         print("Covriance:", cov)
         # why the cov(x) is different from var(x)

Covriance: [[  7.10437124  14.65774383]
 [ 14.65774383  30.41808442]]
```

Assume that the number of dollars spent on car fuel is only linearly dependent on the miles driven. Write code that uses a linear predictor to calculate a predicted value of y for each x.

i.e. $y_{predicted} = f(x) = mx + b$.

```
In [69]: m = cov[0][1]/var_x
         n =mean_y - mean_x*m
         def f(x):
             return m*x+n

         y_predicted = f(x)
```

Predict y for each value in x, put the error into an array called y_{error}.

```
In [70]: y_error = (y - y_predicted)**2
```

Write code that calculates the root mean square error (RMSE).

```
In [71]: rmse = np.mean(y_error**2)
         print(rmse)

0.0709399504515
```

1.4 Pandas

1.4.1 Reading a File

Read in a CSV file called 'data3.csv' into a dataframe called df.

Data description * Data source: http://www.fao.org/nr/water/aquastat/data/query/index.html * Data, units * GDP, current USD (CPI adjusted) * NRI, mm/yr * Population density, inhab/km^2 * Total area of the country, 1000 ha = 10km^2 * Total Population, unit 1000 inhabitants

Display the first 10 lines of the dataframe.

```
In [72]: df = pd.read_csv("data3.csv")
         df.tail(10)

Out[72]:                                                    Area  Area Id  \
         388                          United States of America    231.0
         389                          United States of America    231.0
         390                                               NaN      NaN
         391                                 E - External data      NaN
         392                             I - AQUASTAT estimate      NaN
         393                                K - Aggregate data      NaN
         394                                 L - Modelled data      NaN
         395                                 (c)  FAO of the UN      NaN
         396  The information contained in AQUASTAT is provi...      NaN
         397  FAO. 2016. AQUASTAT Main Database - Food and A...      NaN

                          Variable Name  Variable Id    Year   Value Symbol  Md
         388  National Rainfall Index (NRI)       4472.0  1996.0  1005.0      E NaN
```

```
389  National Rainfall Index (NRI)       4472.0  2002.0   938.7       E NaN
390                            NaN          NaN     NaN     NaN     NaN NaN
391                            NaN          NaN     NaN     NaN     NaN NaN
392                            NaN          NaN     NaN     NaN     NaN NaN
393                            NaN          NaN     NaN     NaN     NaN NaN
394                            NaN          NaN     NaN     NaN     NaN NaN
395                            NaN          NaN     NaN     NaN     NaN NaN
396                            NaN          NaN     NaN     NaN     NaN NaN
397                            NaN          NaN     NaN     NaN     NaN NaN
```

Display the column names.

```
In [73]: df.columns.values
```

```
Out[73]: array(['Area', 'Area Id', 'Variable Name', 'Variable Id', 'Year', 'Value',
                'Symbol', 'Md'], dtype=object)
```

1.4.2 Data Preprocessing

Create a mask of NAN values (i.e. apply `.isnull` on the dataframe). Inspect the mask for 'True' values, they denote NANs.
Hint: You will notice that the last 8 rows and the last column ('Other') have NAN values. You can also use df.tail() to see the last row.

Remove the bottom 8 rows from the dataframe because they contain NAN values. Also remove the column 'Other'.

```
In [74]: df = df[:-8]
         df = df.drop("Md", axis= 1)
```

All the columns in our dataframe are not required for analysis. Drop these columns: `Area Id`, `Variable Id`, and `Symbol` and save the new dataframe as `df1`.

```
In [75]: df1 = df.drop(["Area Id", "Symbol"], axis = 1)
```

Display all the unique values in your new dataframe for these columns: `Area`, `Variable Name`, and `Year`.

Note the Countries and the Metrics (ie.recorded variables) represented in your dataset. *Hint: Use .unique( ) method.*

```
In [76]: print(pd.unique(df1["Area"]))
         print("")
         print(pd.unique(df1["Variable Name"]))
         print("")
         print(pd.unique(df1["Year"]))
```

```
['Argentina' 'Australia' 'Germany' 'Iceland' 'Ireland' 'Sweden'
 'United States of America']

['Total area of the country' 'Total population' 'Population density'
```

```
 'Gross Domestic Product (GDP)' 'National Rainfall Index (NRI)']

[ 1962.  1967.  1972.  1977.  1982.  1987.  1992.  1997.  2002.  2007.
  2012.  2014.  2015.  1963.  1970.  1974.  1978.  1984.  1990.  1964.
  1981.  1985.  1996.  2001.  1969.  1973.  1979.  1993.  1971.  1975.
  1986.  1991.  1998.  2000.  1965.  1983.  1988.  1995.]
```

Convert the Year column string values to pandas datetime objects, where only the year is specified.
Hint: df1['Year'] = pd.to_datetime(pd.Series(df1['Year']).astype(int),format='%Y').dt.year
Run df1.tail() to see part of the result.

```
In [77]: df1["Year"] = pd.to_datetime(pd.Series(df1["Year"]).astype(int), format = "%Y").dt.yea
         df1.tail()
```

```
Out[77]:                          Area                 Variable Name  Variable Id  \
         385  United States of America  National Rainfall Index (NRI)       4472.0
         386  United States of America  National Rainfall Index (NRI)       4472.0
         387  United States of America  National Rainfall Index (NRI)       4472.0
         388  United States of America  National Rainfall Index (NRI)       4472.0
         389  United States of America  National Rainfall Index (NRI)       4472.0

              Year   Value
         385  1981   949.2
         386  1984   974.6
         387  1992  1020.0
         388  1996  1005.0
         389  2002   938.7
```

1.4.3 Extracting Statistics

Create a dataframe 'dftemp' to store rows where the `Area` is `Iceland`.

```
In [78]: dftemp = df1.loc[df1["Area"] == "Iceland"]
```

Print the years when the National Rainfall Index (NRI) was > 950 or < 900 in Iceland using the dataframe you created in the previous question.

```
In [79]: dftemp1 = dftemp.loc[dftemp["Variable Name"] == "National Rainfall Index (NRI)"]

         display(dftemp1.loc[(dftemp1["Value"] > 950) | (dftemp1["Value"] < 900)])
         # use | instead of or
```

```
        Area                  Variable Name  Variable Id  Year   Value
214  Iceland  National Rainfall Index (NRI)       4472.0  1967   816.0
215  Iceland  National Rainfall Index (NRI)       4472.0  1971   963.2
216  Iceland  National Rainfall Index (NRI)       4472.0  1975  1010.0
218  Iceland  National Rainfall Index (NRI)       4472.0  1986   968.5
```

```
219  Iceland  National Rainfall Index (NRI)       4472.0  1991  1095.0
220  Iceland  National Rainfall Index (NRI)       4472.0  1997   993.2
```

Get all the rows of df1 (from the preprocessed data section of this notebook) where the `Area` is `United States of America` and store that into a new dataframe called `df_usa`. Set the indices of the this dataframe to be the `Year` column.
Hint: Use .set_index()

```
In [80]: df_usa = df1.loc[df1["Area"] == "United States of America"]

         df_usa.set_index("Year", inplace= True)
         df_usa.head()
```

```
Out[80]:                           Area              Variable Name  Variable Id  \
         Year
         1962  United States of America  Total area of the country       4100.0
         1967  United States of America  Total area of the country       4100.0
         1972  United States of America  Total area of the country       4100.0
         1977  United States of America  Total area of the country       4100.0
         1982  United States of America  Total area of the country       4100.0

                  Value
         Year
         1962  962909.0
         1967  962909.0
         1972  962909.0
         1977  962909.0
         1982  962909.0
```

Pivot the dataframe so that the unique `Variable Name` entries become the column entries. The dataframe values should be the ones in the `Value` column. Do this by running the lines of code below.

```
In [81]: df_usa = df_usa.pivot(columns='Variable Name',values='Value')
         # # use the Variable Name as the column entries, group by different entries
         df_usa["Year"] = df_usa.index
         df_usa.head()
```

```
Out[81]: Variable Name  Gross Domestic Product (GDP)  National Rainfall Index (NRI)  \
         Year
         1962                           6.050000e+11                            NaN
         1965                                    NaN                          928.5
         1967                           8.620000e+11                            NaN
         1969                                    NaN                          952.2
         1972                           1.280000e+12                            NaN

         Variable Name  Population density  Total area of the country  \
         Year
```

```
1962                         19.93                    962909.0
1965                           NaN                         NaN
1967                         21.16                    962909.0
1969                           NaN                         NaN
1972                         22.14                    962909.0

Variable Name  Total population  Year
Year
1962                   191861.0  1962
1965                        NaN  1965
1967                   203713.0  1967
1969                        NaN  1969
1972                   213220.0  1972
```

Rename the corresponding columns to ['GDP','NRI','PD','Area','Population'].

```
In [82]: df_usa.rename(columns= {"Gross Domestic Product (GDP)": "GDP", "National Rainfall Inde
                       "Total area of the country": "Area", "Total population": "Population", '


         df_usa.head()
```

```
Out[82]: Variable Name           GDP    NRI     PD      Area  Population  Year
         Year
         1962           6.050000e+11    NaN  19.93  962909.0    191861.0  1962
         1965                    NaN  928.5    NaN       NaN         NaN  1965
         1967           8.620000e+11    NaN  21.16  962909.0    203713.0  1967
         1969                    NaN  952.2    NaN       NaN         NaN  1969
         1972           1.280000e+12    NaN  22.14  962909.0    213220.0  1972
```

Print the output of `df_usa.isnull().sum()`. This gives us the number of NAN values in each column. Replace the NAN values by 0, using `df_usa=df_usa.fillna(0)`. Print the output of `df_usa.isnull().sum()` again.

```
In [83]: display("Number of NAN values before: ", df_usa.isnull().sum())
         df_usa = df_usa.fillna(0)
         display("Number of NAN values after: ", df_usa.isnull().sum())
         df_usa.head()
```

```
'Number of NAN values before: '


Variable Name
GDP            7
NRI           11
PD             7
Area           7
Population     7
Year           0
dtype: int64
```

```
'Number of NAN values after: '


Variable Name
GDP           0
NRI           0
PD            0
Area          0
Population    0
Year          0
dtype: int64
```

```
Out[83]: Variable Name          GDP    NRI     PD      Area  Population  Year
         Year
         1962           6.050000e+11    0.0  19.93  962909.0    191861.0  1962
         1965           0.000000e+00  928.5   0.00       0.0         0.0  1965
         1967           8.620000e+11    0.0  21.16  962909.0    203713.0  1967
         1969           0.000000e+00  952.2   0.00       0.0         0.0  1969
         1972           1.280000e+12    0.0  22.14  962909.0    213220.0  1972
```

Calculate and print all the column averages and the column standard deviations.

```
In [84]: # are you sure not to delete all the NAN values first?
         # get rid of the zero entries
         # GDP_c = df_usa["GDP"].loc[df_usa["GDP"] != 0].reset_index()["GDP"]
         # NRI_c = df_usa["NRI"].loc[df_usa["NRI"] != 0].reset_index()["NRI"]
         # PD_c = df_usa["PD"].loc[df_usa["PD"] != 0].reset_index()["PD"]
         # Area_c = df_usa["Area"].loc[df_usa["Area"] != 0].reset_index()["Area"]
         # Population_c = df_usa["Population"].loc[df_usa["Population"] != 0].reset_index()["Po
         # df_usa = pd.DataFrame({"GDP": GDP_c, "NRI": NRI_c, "PD": PD_c, "Area": Area_c, "Pop

         print("Area: Mean: %.2f, Standard deviation: %.2f" % (np.mean(df_usa["Area"]), np.std
         print("GDP: Mean: %.2f, Standard deviation: %.2f" % (np.mean(df_usa["GDP"]), np.std(d
         print("Area: Mean: %.2f, Standard deviation: %.2f" % (np.mean(df_usa["NRI"]), np.std(
         print("Area: Mean: %.2f, Standard deviation: %.2f" % (np.mean(df_usa["PD"]), np.std(d
         print("Area: Mean: %.2f, Standard deviation: %.2f" % (np.mean(df_usa["Population"]), 

         df_usa
```

```
Area: Mean: 610314.74, Standard deviation: 466173.92
GDP: Mean: 4620894736842.11, Standard deviation: 5926262129793.53
Area: Mean: 409.27, Standard deviation: 480.39
Area: Mean: 16.70, Standard deviation: 13.19
Area: Mean: 161513.42, Standard deviation: 127876.43
```

```
Out[84]: Variable Name          GDP     NRI     PD      Area  Population  Year
         Year
```

1962	6.050000e+11	0.0	19.93	962909.0	191861.0	1962
1965	0.000000e+00	928.5	0.00	0.0	0.0	1965
1967	8.620000e+11	0.0	21.16	962909.0	203713.0	1967
1969	0.000000e+00	952.2	0.00	0.0	0.0	1969
1972	1.280000e+12	0.0	22.14	962909.0	213220.0	1972
1974	0.000000e+00	1008.0	0.00	0.0	0.0	1974
1977	2.090000e+12	0.0	23.17	962909.0	223091.0	1977
1981	0.000000e+00	949.2	0.00	0.0	0.0	1981
1982	3.340000e+12	0.0	24.30	962909.0	233954.0	1982
1984	0.000000e+00	974.6	0.00	0.0	0.0	1984
1987	4.870000e+12	0.0	25.49	962909.0	245425.0	1987
1992	6.540000e+12	1020.0	26.78	962909.0	257908.0	1992
1996	0.000000e+00	1005.0	0.00	0.0	0.0	1996
1997	8.610000e+12	0.0	28.34	962909.0	272883.0	1997
2002	1.100000e+13	938.7	29.95	963203.0	288471.0	2002
2007	1.450000e+13	0.0	31.32	963203.0	301656.0	2007
2012	1.620000e+13	0.0	32.02	983151.0	314799.0	2012
2014	0.000000e+00	0.0	0.00	983151.0	0.0	2014
2015	1.790000e+13	0.0	32.73	0.0	321774.0	2015

Using the `df_usa` dataframe, multiply the `Area` by 10 (so instead of 1000 ha, the unit becomes 100 ha = 1km^2). Store the result in place.

In [85]:
```
df_usa["Area"] = df_usa["Area"]*10
```

Create a new column in `df_usa` called `GDP/capita` and populate it with the calculated GDP per capita. Round the results to two decimal points. Store the result in place.

In [86]:
```
df_usa["GDP/capata"] = df_usa["GDP"]/df_usa["Population"]
df_usa["GDP/capata"] = np.round(df_usa["GDP/capata"], decimals=2)
```

Create a new column in `df_usa` called `PD2` (i.e. population density 2). Calculate the population density. **Note: the units should be inhab/km^2**. Round the reults to two decimal point. Store the result in place.

In [87]:
```
df_usa["PD2"] = df_usa["Population"]/df_usa["Area"]
df_usa = df_usa.fillna(0)
df_usa.head()
```

Out[87]:

Variable Name	GDP	NRI	PD	Area	Population	Year
Year						
1962	6.050000e+11	0.0	19.93	9629090.0	191861.0	1962
1965	0.000000e+00	928.5	0.00	0.0	0.0	1965
1967	8.620000e+11	0.0	21.16	9629090.0	203713.0	1967
1969	0.000000e+00	952.2	0.00	0.0	0.0	1969
1972	1.280000e+12	0.0	22.14	9629090.0	213220.0	1972

Variable Name	GDP/capata	PD2
Year		

```
1962           3153324.54  0.019925
1965                 0.00  0.000000
1967           4231443.26  0.021156
1969                 0.00  0.000000
1972           6003189.19  0.022143
```

Find the maximum value and minimum value of the 'NRI' column in the USA (using pandas methods). What years do the min and max values occur in?

```
In [88]: Max = np.max(df_usa["NRI"])
         Min = np.min(df_usa["NRI"])
         print("Max: %f, Min: %f" % (Max, Min))
         # df2 = df1.loc[df1["Area"] == "United States of America"]
         # df2.set_index("Year")
         # print(df2)
         max_year = df_usa[df_usa["NRI"] == Max].index.tolist()

         min_year = df_usa[df_usa["NRI"] == Min].index.tolist()



         print("Max occuried in %d, Min occuried in %d" % (max_year[0], min_year[0]))
```

```
Max: 1020.000000, Min: 0.000000
Max occuried in 1992, Min occuried in 1962
```

1.5 Matplotlib

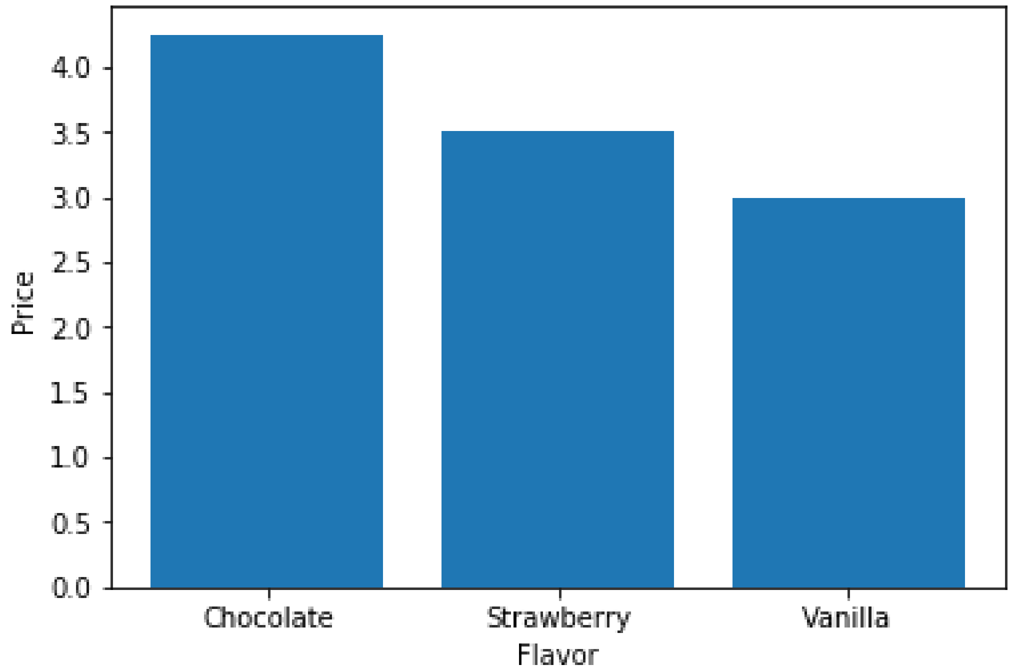
Create a dataframe called `icecream` that has column `Flavor` with entries `Strawberry`, `Vanilla`, and `Chocolate` and another column with `Price` with entries `3.50`, `3.00`, and `4.25`.

```
In [89]: icecream = pd.DataFrame({"Flavor": ["Strawberry", "Vanilla", "Chocolate"], "Price": [
```

Create a bar chart representing the three flavors and their associated prices.

```
In [90]: f, ax = plt.subplots()
         x1 = icecream["Flavor"]
         y1 = icecream["Price"]
         ax.set_xlabel("Flavor")
         ax.set_ylabel("Price")
         ax.bar(x1,y1)
```

```
Out[90]: <Container object of 3 artists>
```

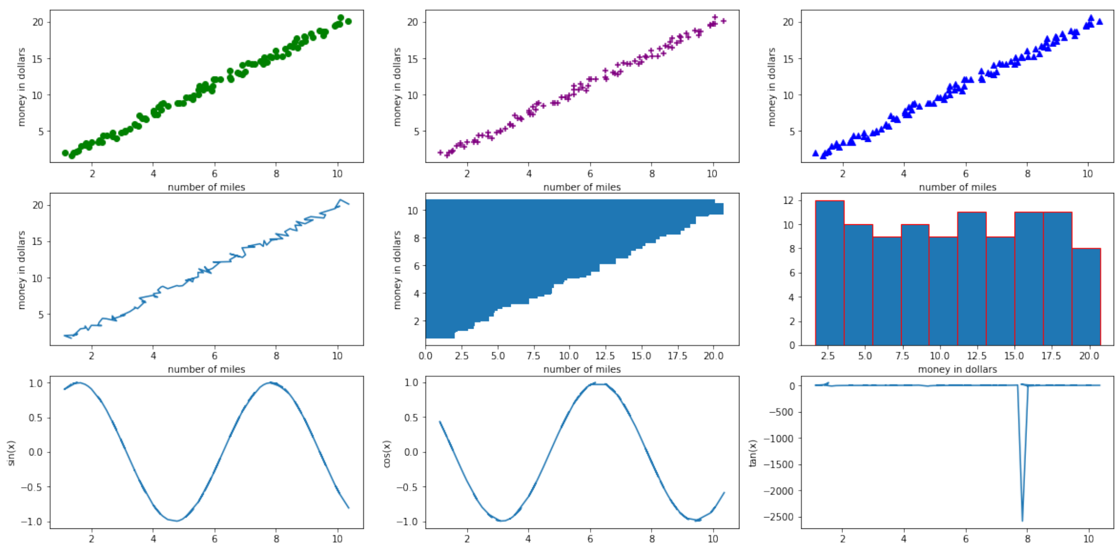


Create 9 random plots. The top three should be scatter plots (one with green dots, one with purple crosses, and one with blue triangles. The middle three graphs should be a line graph, a horizontal bar chart, and a histogram. The bottom three graphs should be trignometric functions (one sin, one cosine, one tangent).

```
In [91]: f = plt.subplots(figsize=(20, 10))
         ax1 = plt.subplot(3,3,1)
         ax1.scatter(x, y, color = "green")
         ax1.set_xlabel("number of miles")
         ax1.set_ylabel("money in dollars")

         ax2 = plt.subplot(3,3,2)
         ax2.scatter(x, y, marker="+", color = "purple")
         ax2.set_xlabel("number of miles")
         ax2.set_ylabel("money in dollars")

         ax3 = plt.subplot(3,3,3)
         ax3.scatter(x, y, marker="^", color = "blue")
         ax3.set_xlabel("number of miles")
         ax3.set_ylabel("money in dollars")

         ax4 = plt.subplot(3,3,4)
         ax4.plot(x,y)
         ax4.set_xlabel("number of miles")
         ax4.set_ylabel("money in dollars")
```

```
ax5 = plt.subplot(3,3,5)
ax5.barh(x,y,edgecolor = "red")
ax5.set_xlabel("number of miles")
ax5.set_ylabel("money in dollars")

ax6 = plt.subplot(3,3,6)
ax6.hist(y, edgecolor = "red", linewidth= 1.0)
ax6.set_xlabel("money in dollars")

ax7 = plt.subplot(3,3,7)
ax7.plot(x,np.sin(x))
ax7.set_xlabel("x")
ax7.set_ylabel("sin(x)")

ax8 = plt.subplot(3,3,8)
ax8.plot(x,np.cos(x))
ax8.set_xlabel("x")
ax8.set_ylabel("cos(x)")

ax9 = plt.subplot(3,3,9)
ax9.plot(x,np.tan(x))
ax9.set_xlabel("x")
ax9.set_ylabel("tan(x)")
```

Out[91]: Text(0,0.5,'tan(x)')



1.6 Extra Credit

Run the cell below to read in the data. See: https://www.quantshare.com/sa-43-10-ways-to-download-historical-stock-quotes-data-for-free

In [92]:
```
df_google = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=go
df_apple = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=aap

df_disney = pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=di
df_nike= pd.read_csv('https://finance.google.com/finance/historical?output=csv&q=nke')

df_apple.head()
```

Out[92]:

	Date	Open	High	Low	Close	Volume
0	8-Feb-18	160.29	161.00	155.03	155.15	54390516
1	7-Feb-18	163.08	163.40	159.07	159.54	51608580
2	6-Feb-18	154.83	163.72	154.00	163.03	68243838
3	5-Feb-18	159.10	163.88	156.00	156.49	72738522
4	2-Feb-18	166.00	166.80	160.10	160.50	86593825

Show a 3 x 3 correlation matrix for Nike, Apple, and Disney stock prices for the month of July, 2017.

Hint: Convert `Date` to a pandas datetime object. Change the indices of all the dataframes to `Date`. Use `Date` indices to filter rows. Create a new dataframe that stores values of the `Close` column from each dataframe. Use the `Close` column of each company's stock data to find the correlation using df.corr().

In [93]:
```
# df1["Year"] = pd.to_datetime(pd.Series(df1["Year"]).astype(int), format = "%Y").dt.y

df_google["Date"] = pd.to_datetime(pd.Series(df_google["Date"]))
df_google.set_index("Date", inplace= True)
df_google.sort_index(inplace= True)

df_apple["Date"] = pd.to_datetime(pd.Series(df_apple["Date"]))
df_apple.set_index("Date", inplace= True)
df_apple.sort_index(inplace= True)

df_disney["Date"] = pd.to_datetime(pd.Series(df_disney["Date"]))
df_disney.set_index("Date", inplace= True)
df_disney.sort_index(inplace= True)

df_nike["Date"] = pd.to_datetime(pd.Series(df_nike["Date"]))
df_nike.set_index("Date", inplace= True)
df_nike.sort_index(inplace= True)

display(df_google.head(), df_apple.head())
```

```
            Open    High     Low   Close   Volume
Date
```

```
2017-02-10  811.70  815.25  809.78  813.67  1134976
2017-02-13  816.00  820.96  815.49  819.24  1213324
2017-02-14  819.00  823.00  816.00  820.45  1054732
2017-02-15  819.36  823.00  818.47  818.98  1313617
2017-02-16  819.93  824.40  818.98  824.16  1287626
```

```
              Open    High     Low   Close    Volume
Date
2017-02-10  132.46  132.94  132.05  132.12  20065458
2017-02-13  133.08  133.82  132.75  133.29  23035421
2017-02-14  133.47  135.09  133.25  135.02  33226223
2017-02-15  135.52  136.27  134.62  135.51  35623100
2017-02-16  135.67  135.90  134.84  135.34  22584555
```

```
In [94]: df_new_july = pd.DataFrame({"google": df_google.loc[(df_google.index.year == 2017) &
                                     "apple": df_apple.loc[(df_apple.index.year == 2017) & (df_
                                     "disney": df_disney.loc[(df_disney.index.year == 2017) &
                                "nike": df_nike.loc[(df_nike.index.year == 2017) & (df_nike.inc
         df_new_july.corr()
```

```
Out[94]:            apple    disney    google      nike
         apple   1.000000  0.524912  0.805168  0.417947
         disney  0.524912  1.000000  0.188795  0.459045
         google  0.805168  0.188795  1.000000  0.351716
         nike    0.417947  0.459045  0.351716  1.000000
```

Show the same correlation matrix but over different time periods. 1. the last 20 days 2. the last 80 days

```
In [95]: # store the whole period
         df_new = pd.DataFrame({"google": df_google["Close"], "apple": df_apple["Close"], "dis
                                "nike": df_nike["Close"]})

         corr_20 = df_new[:20].corr()
         corr_80 = df_new[-80:].corr()
         display("Correlation for the first 20 days:", corr_20)
         print("")
         display("Correlation for the last 80 days:", corr_80)
```

```
'Correlation for the first 20 days:'
```

```
           apple    disney    google      nike
apple   1.000000  0.849213  0.798739  0.302779
disney  0.849213  1.000000  0.658487 -0.016865
google  0.798739  0.658487  1.000000  0.270564
nike    0.302779 -0.016865  0.270564  1.000000
```

```
'Correlation for the last 80 days:'
```

	apple	disney	google	nike
apple	1.000000	0.588532	0.493026	0.401821
disney	0.588532	1.000000	0.778008	0.885038
google	0.493026	0.778008	1.000000	0.839613
nike	0.401821	0.885038	0.839613	1.000000

Change the code so that it accepts a list of any stock symbols (i.e. ['NKE', 'APPL', 'DIS', …]) and creates a correlation matrix for the past 100 days.

```
In [96]: def get_corr(l):
             df_new = pd.DataFrame()
             for i in l:
                 link = r"https://finance.google.com/finance/historical?output=csv&q={}".forma
                 exec('df_{} = pd.read_csv("{}")'.format(str(i),link))
                 exec('df_{}["Date"] = pd.to_datetime(pd.Series(df_{}["Date"]))'.format(str(i)
                 exec('df_{}.set_index("Date", inplace= True)'.format(str(i)))
                 exec('df_{}.sort_index(inplace= True)'.format(str(i)))

             for i in l:
                 exec('df_new["{}"] = df_{}["Close"]'.format(str(i),str(i)))


             return df_new[l][-100:].corr()
```

```
In [97]: List = ['goog', 'aapl', 'dis', 'nke']
         corr_100 = get_corr(List)
         display(corr_100)
```

	goog	aapl	dis	nke
goog	1.000000	0.720482	0.832780	0.879636
aapl	0.720482	1.000000	0.721132	0.669868
dis	0.832780	0.721132	1.000000	0.911462
nke	0.879636	0.669868	0.911462	1.000000