Grashof_David_Assignment1

April 7, 2019

0.1 Assignment 1 | Exploring and Visualizing Data

0.1.1 Overview

The Northwestern MSDS program administrators are interested in finding out the best way to structure the program, going

forward. More specifically, they would like to know what programming language to teach the courses in and what electives

they will offer. To make a fully-informed decision, a survey was distributed to current students in the MSDS program to

gauge their interests in both programming languages and potential electives. The survey has been collected and analyzed in

various ways to help guide the administrators in enhancing the program.

In order to address each question, the data was parsed out accordingly from the original file. To answer the first

question, which programming language should be used primarily for future classes, a mean was taken for all responses. Due

to high level of variance, this value was not very meaningful. What it did show was that both R and Python had the highest

average response in the mid-30s. To get a better understanding of the responses, a histogram was constructed for each

language. It became increasingly obvious that both Java languages and SAS could be removed from consideration, leaving only

R and Python. Before making a final determination, it was necessary to understand if the responses changed based on the

context, such as industry, professional or personal. A correlation plot showed that the responses between each context for

a given language were highly correlated between .65 and .79, indicating that students mostly had the same response

regardless of context. Another consideration was the determine if there was a link between courses completed and responses,

with the assumption being that the answers of students with more classes completed would carry more weight than new

students who may not have any programming experience. The results showed that there was no relationship.

To determine which classes should be offered, first the mean was calculated for the four possible options with the Python course logging the highest mean score of 73.5, much greater than the second highest class, Foundations in Data Engineering,

with a mean score of 58. Likewise, this data exhibited high variance like the language preference responses. Next, the

distributions of the responses for each course were plotted, further showing that the Python class had the most responses

at or near 100.

It is recommended that given the high interest shown for Python in both questions, that the Python class be included as an

elective. Additionally, given the similar scores shown for both R and Python as the preferred programming language, future

courses should be offered as one or the other.

0.1.2 1.1 | Load Modules

```
In [1]: import pandas as pd  # data frame operations
        import nbconvert # convert to pdf
        import numpy as np  # arrays and math functions
        import matplotlib.pyplot as plt  # static plotting
        import seaborn as sns  # pretty plotting, including heat map
```

0.1.3 1.2 | Load Data

```
In [2]: #define filepath
        file = "C:/Users/David/OneDrive/MSDS/MSDS422/Week1/Assignment/mspa-software-survey-case
        df = pd.read_csv(file)
        #set RespondentID as index
        df.set_index('RespondentID', drop = True, inplace = True)
        #check strucuture of data
        df.head()
```

```
Out[2]:             Personal_JavaScalaSpark  Personal_JavaScriptHTMLCSS  \
        RespondentID
        5135740122                        0                           0
        5133300037                       10                          10
        5132253300                       20                           0
        5132096630                       10                          10
        5131990362                       20                           0

                    Personal_Python  Personal_R  Personal_SAS  \
        RespondentID
        5135740122                0          50            50
        5133300037               50          30             0
        5132253300               40          40             0
        5132096630               25          35            20
        5131990362                0          70            10

                    Professional_JavaScalaSpark  Professional_JavaScriptHTMLCSS  \
```

```
RespondentID
5135740122                                       0                               0
5133300037                                      25                              25
5132253300                                       0                               0
5132096630                                      10                              10
5131990362                                      20                               0

              Professional_Python  Professional_R  Professional_SAS  ...  \
RespondentID                                                         ...
5135740122                      0              25                75  ...
5133300037                     30              20                 0  ...
5132253300                     40              40                20  ...
5132096630                     25              35                20  ...
5131990362                      0              80                 0  ...

              PREDICT453  PREDICT454  PREDICT455  PREDICT456  PREDICT457  \
RespondentID
5135740122           NaN         NaN         NaN         NaN         NaN
5133300037           NaN         NaN         NaN         NaN         NaN
5132253300           NaN         NaN         NaN         NaN         NaN
5132096630           NaN         NaN         NaN         NaN         NaN
5131990362           NaN         NaN         NaN         NaN         NaN

              OtherPython  OtherR  OtherSAS             Other  Graduate_Date
RespondentID
5135740122            NaN     NaN       NaN               NaN            NaN
5133300037            NaN     NaN       NaN               NaN    Spring 2018
5132253300            NaN     NaN       NaN               NaN      Fall 2018
5132096630            NaN     NaN       NaN               NaN      Fall 2017
5131990362            NaN     NaN       NaN  CS-435 with Weka      Fall 2018

[5 rows x 40 columns]
```

0.1.4 1.3 | Pre-processing Data

In [3]:
```
#organize fields into easily callable groups by purpose of languge
personal_fields = ["Personal_JavaScalaSpark","Personal_JavaScriptHTMLCSS","Personal_Py
professional_fields = ["Professional_JavaScalaSpark","Professional_JavaScriptHTMLCSS","
industry_fields = ["Industry_JavaScalaSpark","Industry_JavaScriptHTMLCSS","Industry_Py
#organize fields into easily callable by langage regardless of purpose of use
jss_fields = ["Personal_JavaScalaSpark","Professional_JavaScalaSpark","Industry_JavaSca
jsh_fields = ["Personal_JavaScriptHTMLCSS","Professional_JavaScriptHTMLCSS","Industry_J
python_fields = ["Personal_Python","Professional_Python","Industry_Python"]
r_fields = ["Personal_R","Professional_R","Industry_R"]
sas_fields = ["Personal_SAS","Professional_SAS","Industry_SAS"]
#orangize classes taken fields
class_fields = ["PREDICT400","PREDICT401","PREDICT410","PREDICT411","PREDICT413","PREDI
                "PREDICT452","PREDICT453","PREDICT454","PREDICT455","PREDICT456","PREDI
```

```
#organize interest fields
interest_fields = ["Python_Course_Interest","Foundations_DE_Course_Interest","Analytics

# def encoder to create a binary nan value
def value_encoder(i):
    if i != i: catstr = "0"
    else: catstr = "1"
    catstr = int(catstr)
    return(catstr)

# apply encoder to class values in order to sum how many classes an individual has tak
for i in df.columns[20:38]:
    df[i] = df[i].map(value_encoder)

#sum number of programming classes taken per respondent as Courses_Completed may inclu
df["Courses_Completed_Prog"] = df[class_fields].sum(axis=1)

def courses_completed_group(i):
    i = int(i)
    if i > 0 and i <= 3: catstr = "0-3"
    elif i > 3 and i <= 6: catstr = "4-6"
    else: catstr = "7-10"
    return(catstr)

#map bucket
df["Courses_Completed_Bucket"] = df["Courses_Completed_Prog"].map(courses_completed_gr
#rebuild class organization of fields
class_fields = ["PREDICT400","PREDICT401","PREDICT410","PREDICT411","PREDICT413","PRED
                "PREDICT452","PREDICT453","PREDICT454","PREDICT455","PREDICT456","PRED
```

In [4]:
```
#overview of key statistics
for i in [personal_fields,professional_fields,industry_fields]:
    print(df[i].describe())
    print('-----------------------------------------------------------------------
#on a personal level R and Python are the leaders by mean but all language have a high
#on a professional level R and Python are again the leaders by mean but with equally h
#on an industry level R and Pytho are again again the leaders by mean with high std
```

```
       Personal_JavaScalaSpark  Personal_JavaScriptHTMLCSS  Personal_Python  \
count               207.000000                  207.000000       207.000000
mean                 10.135266                    4.797101        31.304348
std                  11.383477                    6.757764        15.570982
min                   0.000000                    0.000000         0.000000
25%                   0.000000                    0.000000        20.000000
50%                   9.000000                    0.000000        30.000000
75%                  20.000000                   10.000000        40.000000
max                  70.000000                   30.000000        90.000000
```

```
       Personal_R  Personal_SAS
count  207.000000    207.000000
mean    37.125604     16.637681
std     14.576003     13.626400
min      0.000000      0.000000
25%     30.000000      5.000000
50%     35.000000     15.000000
75%     50.000000     25.000000
max    100.000000     75.000000
--------------------------------------------------------------------------------
       Professional_JavaScalaSpark  Professional_JavaScriptHTMLCSS  \
count                   207.000000                      207.000000
mean                      9.251208                        5.840580
std                      13.167505                       10.812555
min                       0.000000                        0.000000
25%                       0.000000                        0.000000
50%                       5.000000                        0.000000
75%                      15.000000                       10.000000
max                      80.000000                      100.000000

       Professional_Python  Professional_R  Professional_SAS
count           207.000000      207.000000        207.000000
mean             30.028986       36.415459         18.463768
std              19.144802       20.847606         18.831841
min               0.000000        0.000000          0.000000
25%              20.000000       25.000000          0.000000
50%              30.000000       33.000000         15.000000
75%              40.000000       50.000000         30.000000
max             100.000000      100.000000        100.000000
--------------------------------------------------------------------------------
       Industry_JavaScalaSpark  Industry_JavaScriptHTMLCSS  Industry_Python  \
count               207.000000                  207.000000       207.000000
mean                 11.942029                    6.966184        29.772947
std                  14.706399                   10.030721        17.959816
min                   0.000000                    0.000000         0.000000
25%                   0.000000                    0.000000        20.000000
50%                   5.000000                    0.000000        30.000000
75%                  20.000000                   10.000000        40.000000
max                  70.000000                   50.000000        95.000000

       Industry_R  Industry_SAS
count  207.000000    207.000000
mean    32.434783     18.884058
std     15.912209     19.137623
min      0.000000      0.000000
25%     22.500000      0.000000
50%     30.000000     15.000000
75%     40.000000     30.000000
```

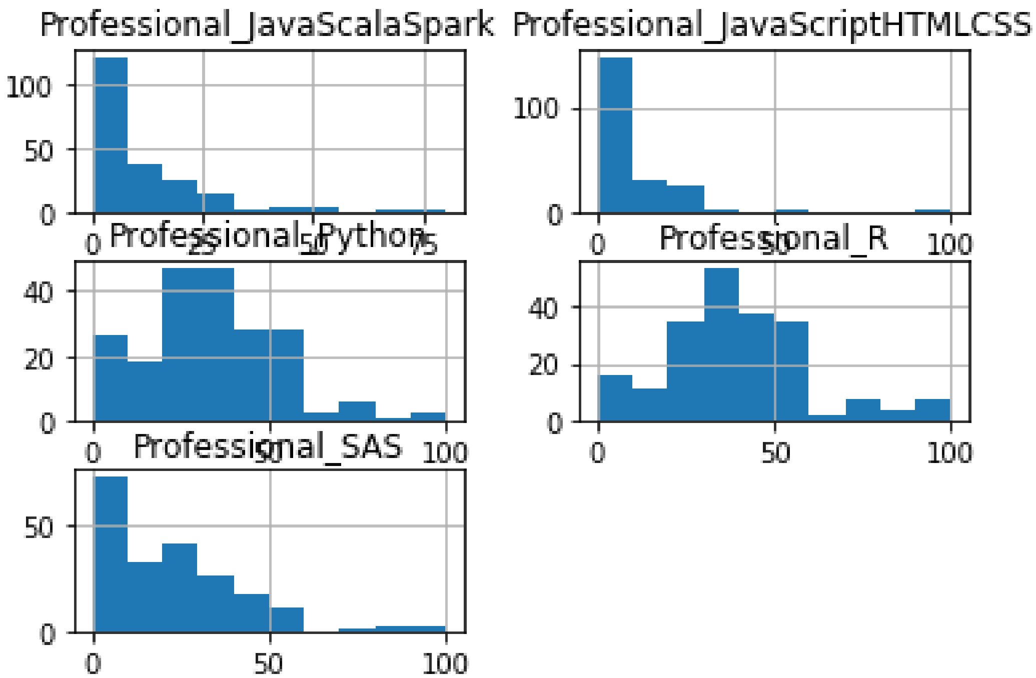
```
max     85.000000    100.000000
--------------------------------------------------------------------------------
```

```
In [5]: for i in [jss_fields,jsh_fields,python_fields,r_fields,sas_fields]:
            print(df[i].describe())
            print('-------------------------------------------------------------------------------
        # All 5 languages have near equal distribution of scores regardless of intent (persona
        # disregarding intent in further analysis may be acceptable. Scaling these scores may
```

```
       Personal_JavaScalaSpark  Professional_JavaScalaSpark  \
count               207.000000                   207.000000
mean                 10.135266                     9.251208
std                  11.383477                    13.167505
min                   0.000000                     0.000000
25%                   0.000000                     0.000000
50%                   9.000000                     5.000000
75%                  20.000000                    15.000000
max                  70.000000                    80.000000

       Industry_JavaScalaSpark
count               207.000000
mean                 11.942029
std                  14.706399
min                   0.000000
25%                   0.000000
50%                   5.000000
75%                  20.000000
max                  70.000000
--------------------------------------------------------------------------------
       Personal_JavaScriptHTMLCSS  Professional_JavaScriptHTMLCSS  \
count                  207.000000                      207.000000
mean                     4.797101                        5.840580
std                      6.757764                       10.812555
min                      0.000000                        0.000000
25%                      0.000000                        0.000000
50%                      0.000000                        0.000000
75%                     10.000000                       10.000000
max                     30.000000                      100.000000

       Industry_JavaScriptHTMLCSS
count                  207.000000
mean                     6.966184
std                     10.030721
min                      0.000000
25%                      0.000000
50%                      0.000000
75%                     10.000000
```

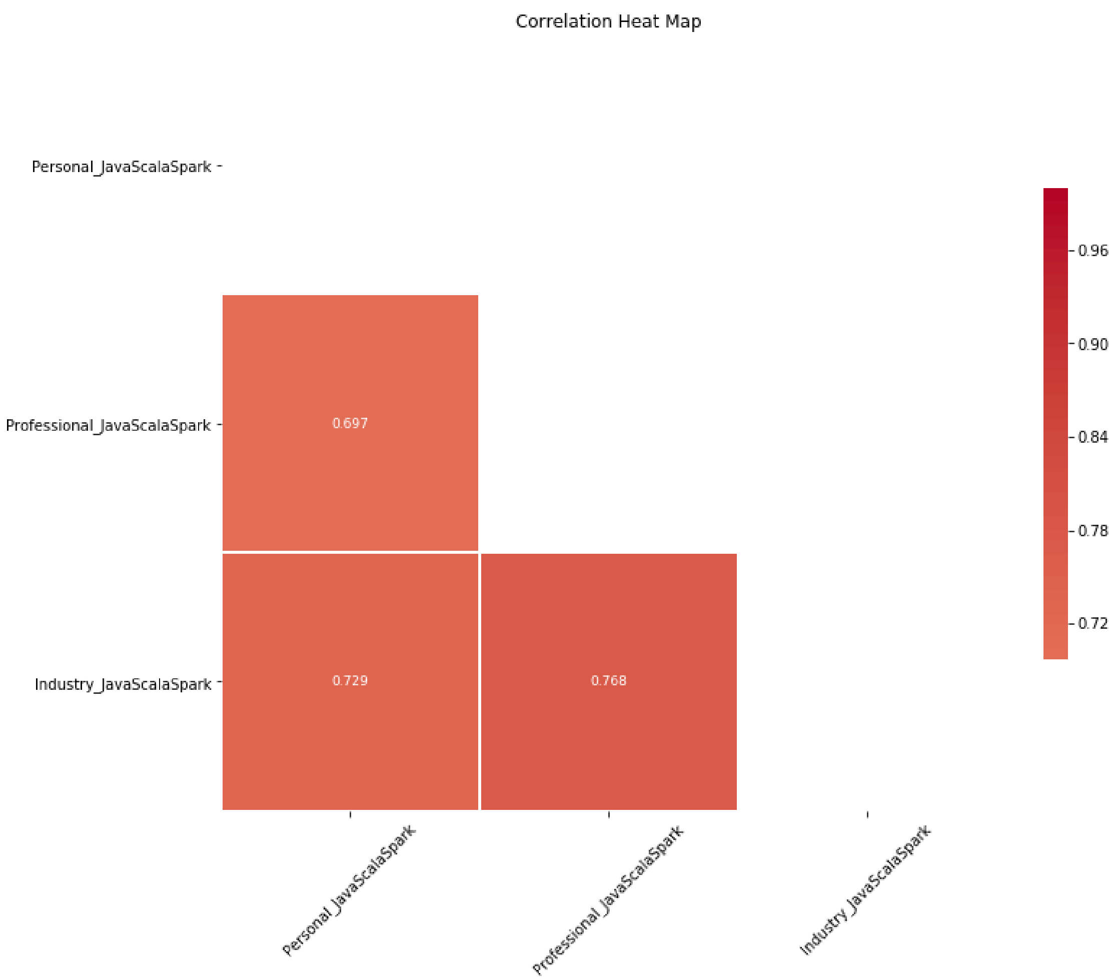
```
max                      50.000000
---------------------------------------------------------------------------------
       Personal_Python  Professional_Python  Industry_Python
count       207.000000           207.000000       207.000000
mean         31.304348            30.028986        29.772947
std          15.570982            19.144802        17.959816
min           0.000000             0.000000         0.000000
25%          20.000000            20.000000        20.000000
50%          30.000000            30.000000        30.000000
75%          40.000000            40.000000        40.000000
max          90.000000           100.000000        95.000000
---------------------------------------------------------------------------------
       Personal_R  Professional_R  Industry_R
count  207.000000      207.000000  207.000000
mean    37.125604       36.415459   32.434783
std     14.576003       20.847606   15.912209
min      0.000000        0.000000    0.000000
25%     30.000000       25.000000   22.500000
50%     35.000000       33.000000   30.000000
75%     50.000000       50.000000   40.000000
max    100.000000      100.000000   85.000000
---------------------------------------------------------------------------------
       Personal_SAS  Professional_SAS  Industry_SAS
count    207.000000        207.000000    207.000000
mean      16.637681         18.463768     18.884058
std       13.626400         18.831841     19.137623
min        0.000000          0.000000      0.000000
25%        5.000000          0.000000      0.000000
50%       15.000000         15.000000     15.000000
75%       25.000000         30.000000     30.000000
max       75.000000        100.000000    100.000000
---------------------------------------------------------------------------------
```

In [6]:
```
#plot histogram for all defined professional fields
df[professional_fields].hist()
plt.xlabel("Interest Score (0-100)")
plt.grid(True)
plt.show()
```

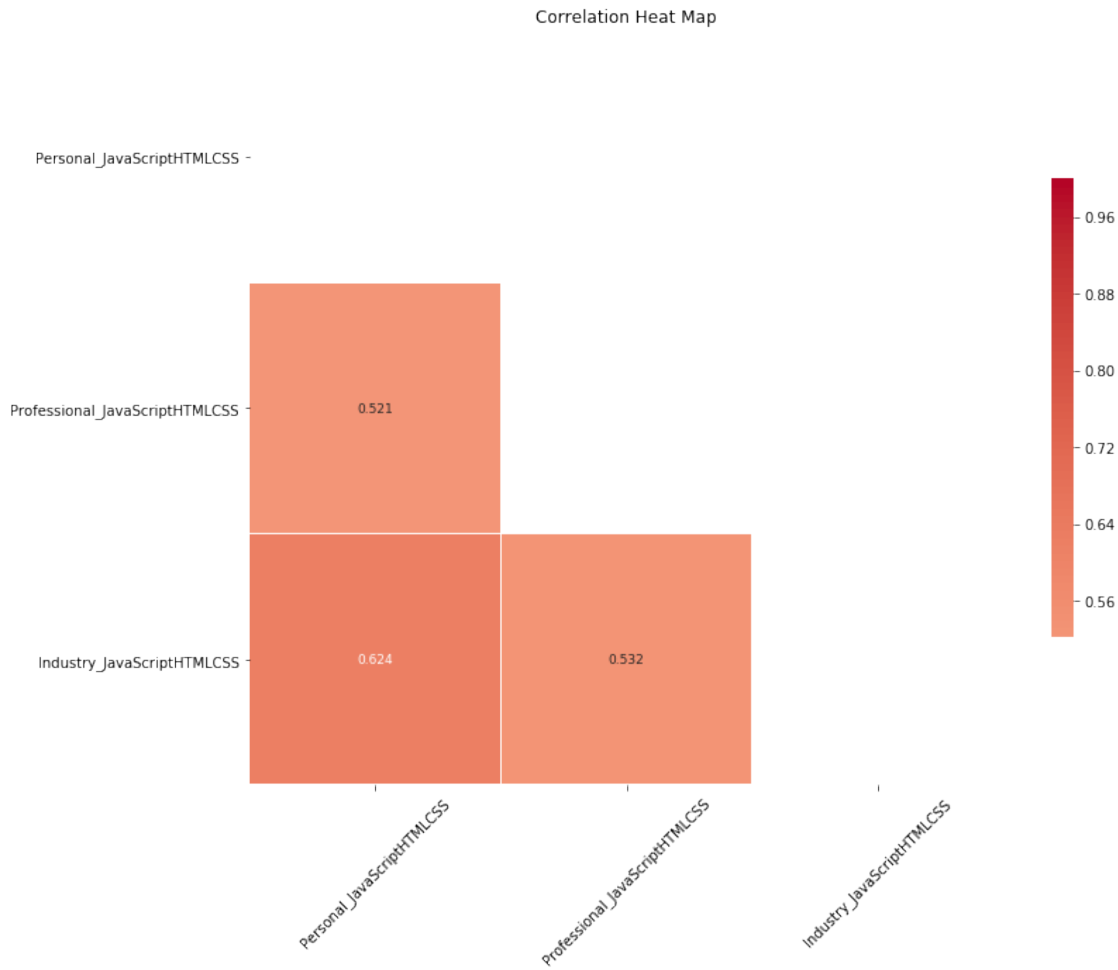



```
In [7]: #visualzing scatterplots to confirm similiaries of scores across intent
        def corr_chart(df_corr):
            corr=df_corr.corr()
            #screen top half to get a triangle
            top = np.zeros_like(corr, dtype=np.bool)
            top[np.triu_indices_from(top)] = True
            fig=plt.figure()
            fig, ax = plt.subplots(figsize=(12,12))
            sns.heatmap(corr, mask=top, cmap='coolwarm',
                center = 0, square=True,
                linewidths=.5, cbar_kws={'shrink':.5},
                annot = True, annot_kws={'size': 9}, fmt = '.3f')
            plt.xticks(rotation=45) # rotate variable labels on columns (x axis)
            plt.yticks(rotation=0) # use horizontal variable labels on rows (y axis)
            plt.title('Correlation Heat Map')
            plt.savefig('plot-corr-map.pdf',
                bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
                orientation='portrait', papertype=None, format=None,
                transparent=True, pad_inches=0.25, frameon=None)

        for i in [jss_fields,jsh_fields,python_fields,r_fields,sas_fields]:
            corr_chart(df_corr = df[i])

        #chart appears to indicate that language by intent is not as highly correlated as prev
        #JavascriptHTML has the weakest correlation by intent at .52 - .62 while Python shows
```

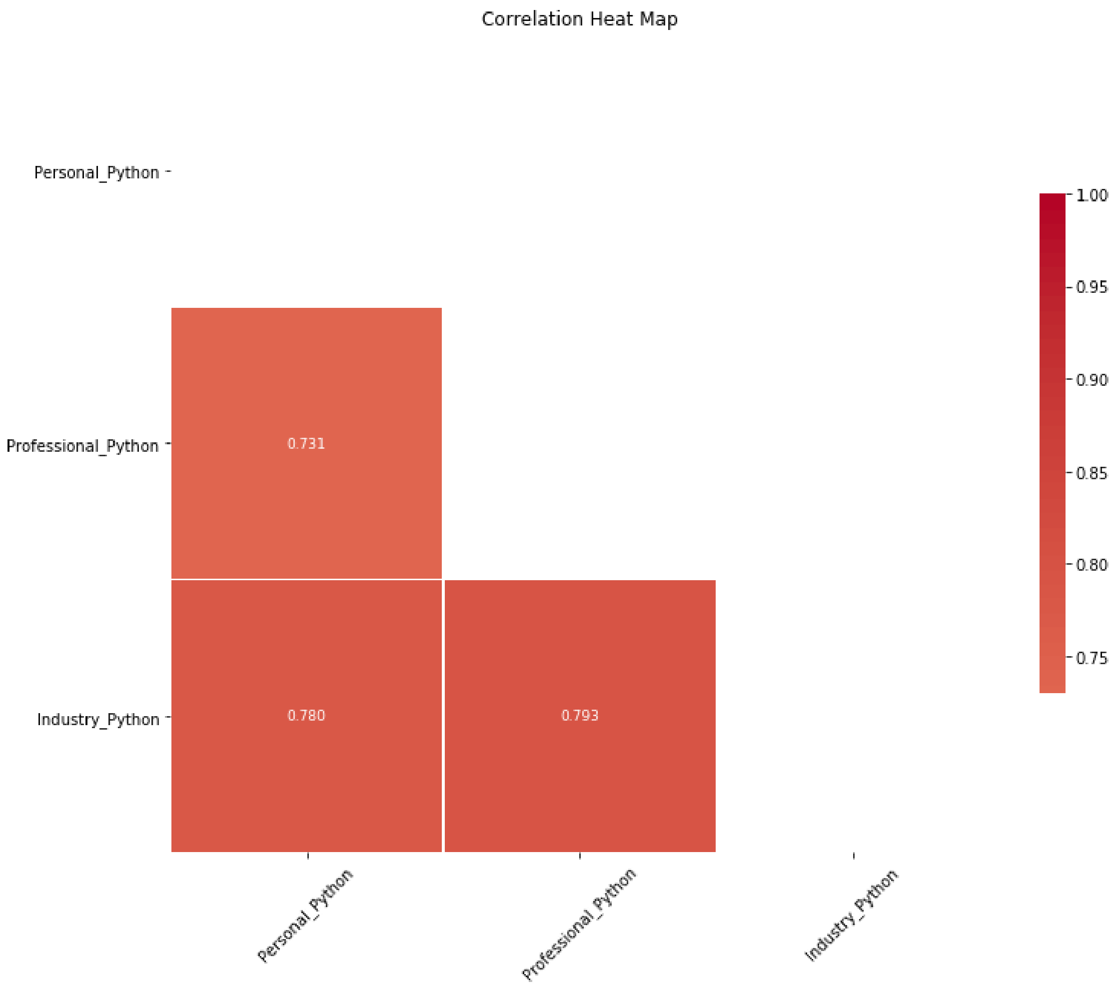
<Figure size 432x288 with 0 Axes>



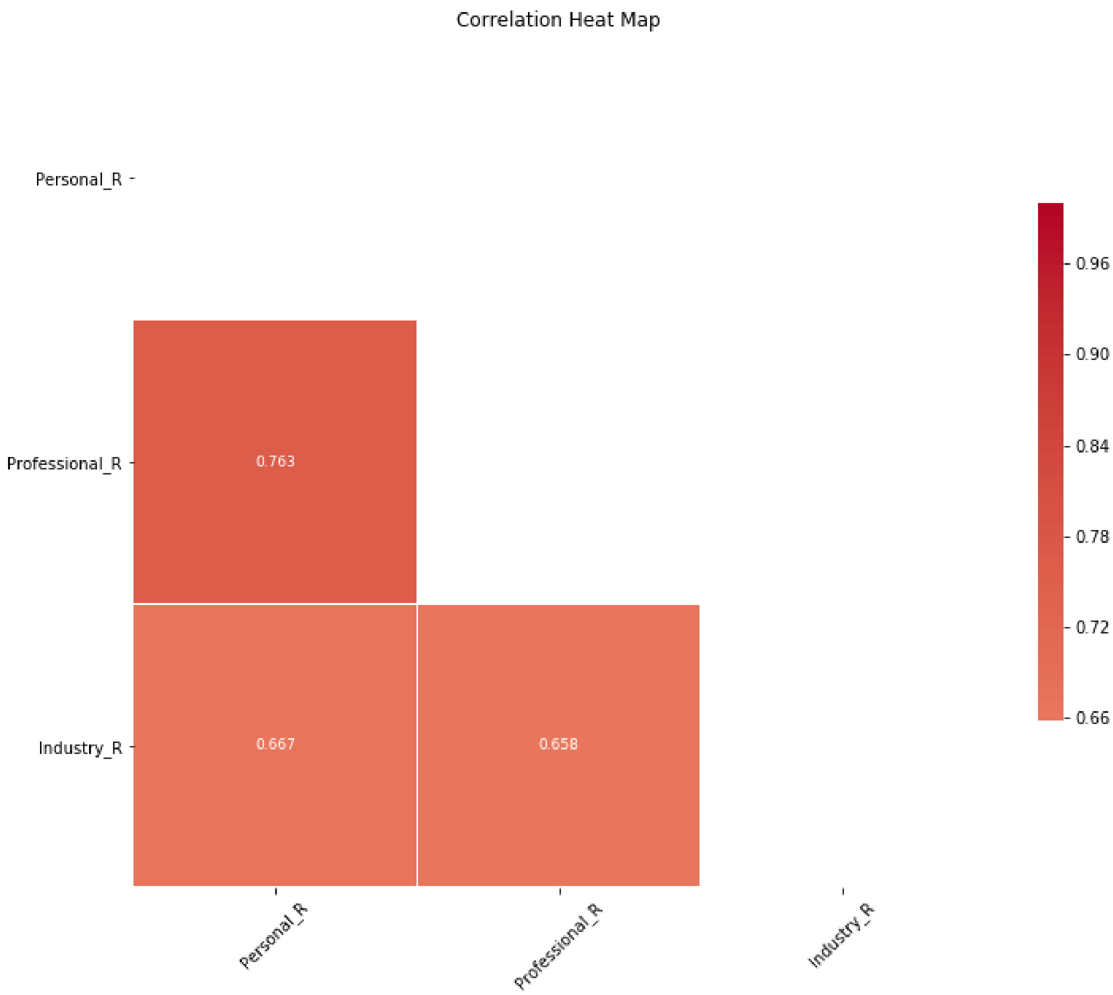
<Figure size 432x288 with 0 Axes>



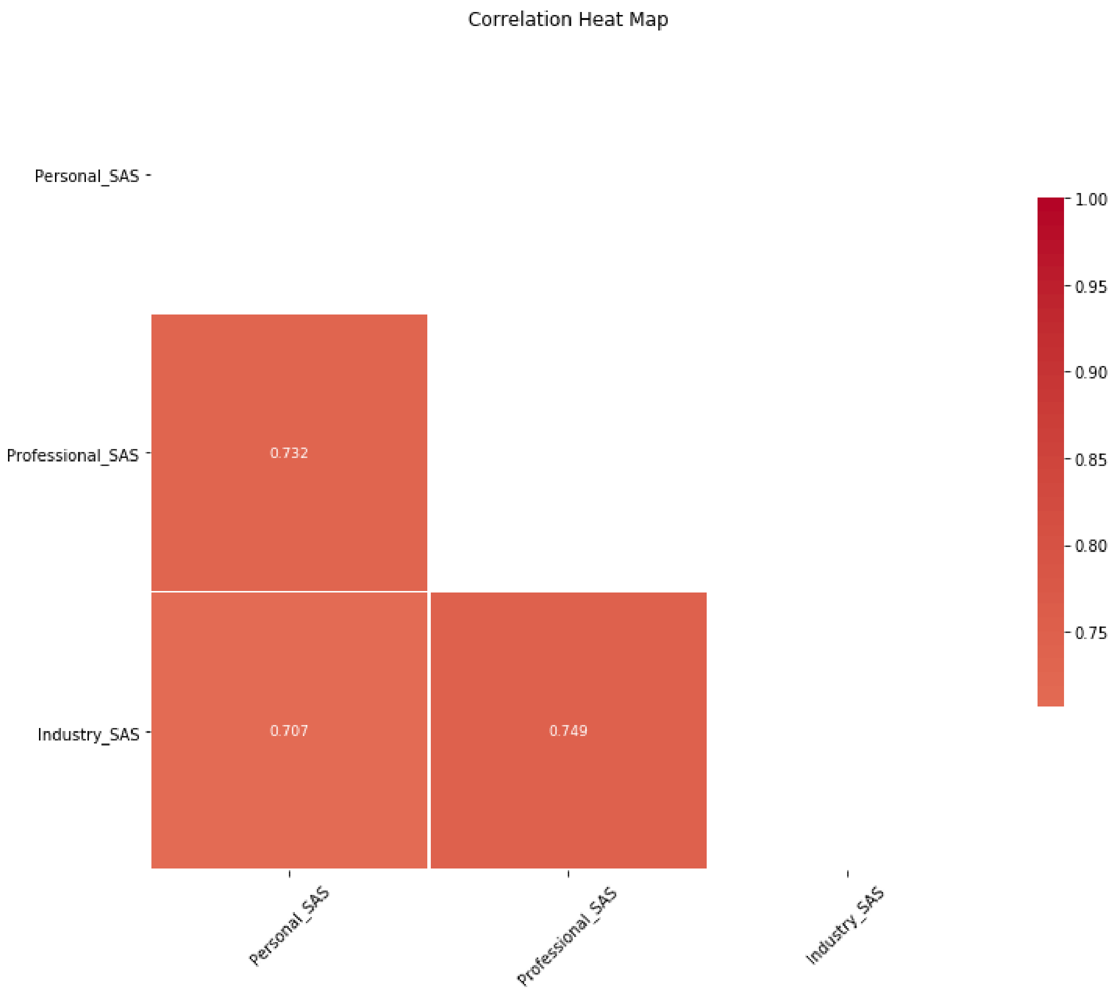

```
<Figure size 432x288 with 0 Axes>
```




```
&lt;Figure size 432x288 with 0 Axes&gt;
```



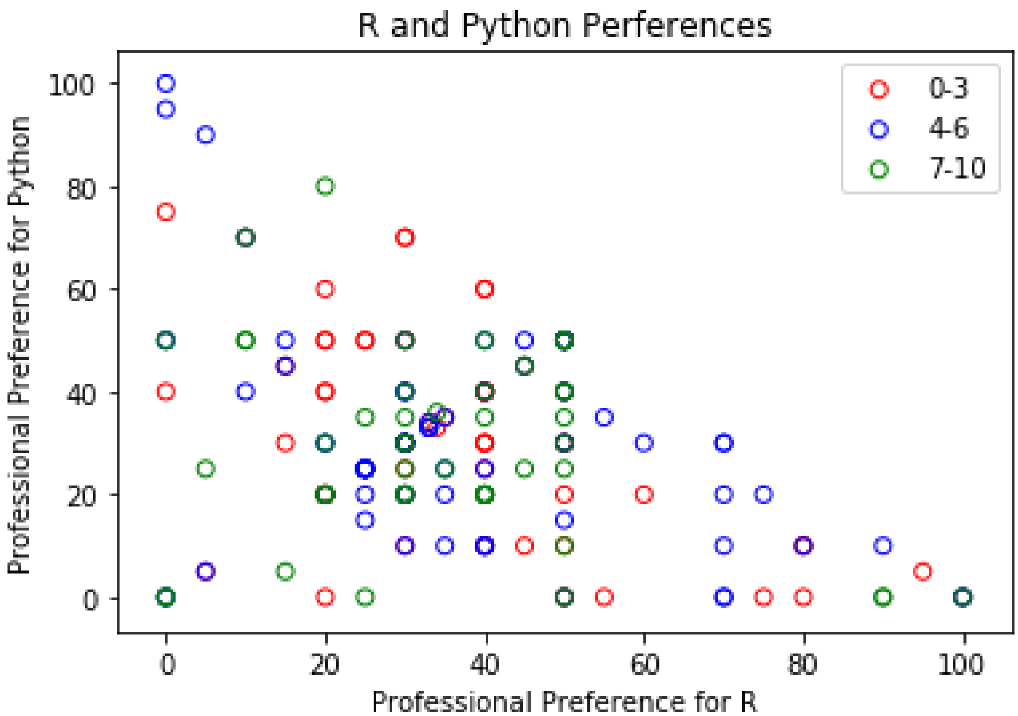
<Figure size 432x288 with 0 Axes>




```
In [8]: #visualize preferences by number of programming courses taken. Do students with more e
        # single scatter plot example
        fig, axis = plt.subplots()
        axis.set_xlabel('Professional Preference for R')
        axis.set_ylabel('Professional Preference for Python')
        group = ()
        group = df["Courses_Completed_Bucket"]
        plt.title('R and Python Perferences')
        scatter_plot = axis.scatter(df[df["Courses_Completed_Bucket"]=='0-3']['Professional_R']
            df[df["Courses_Completed_Bucket"]=='0-3']['Professional_Python'],
            facecolors = 'none',
            edgecolors = 'red',
            label = '0-3')
        scatter_plot = axis.scatter(df[df["Courses_Completed_Bucket"]=='4-6']['Professional_R']
            df[df["Courses_Completed_Bucket"]=='4-6']['Professional_Python'],
            facecolors = 'none',
            edgecolors = 'blue',
```

```
    label = '4-6')
scatter_plot = axis.scatter(df[df["Courses_Completed_Bucket"]=='7-10']['Professional_R
    df[df["Courses_Completed_Bucket"]=='7-10']['Professional_Python'],
    facecolors = 'none',
    edgecolors = 'green',
    label = '7-10')
plt.legend()

#there doesn't seem to be much a correlation between number of courses taken and prefe
```

Out[8]: <matplotlib.legend.Legend at 0x18807bf5cf8>

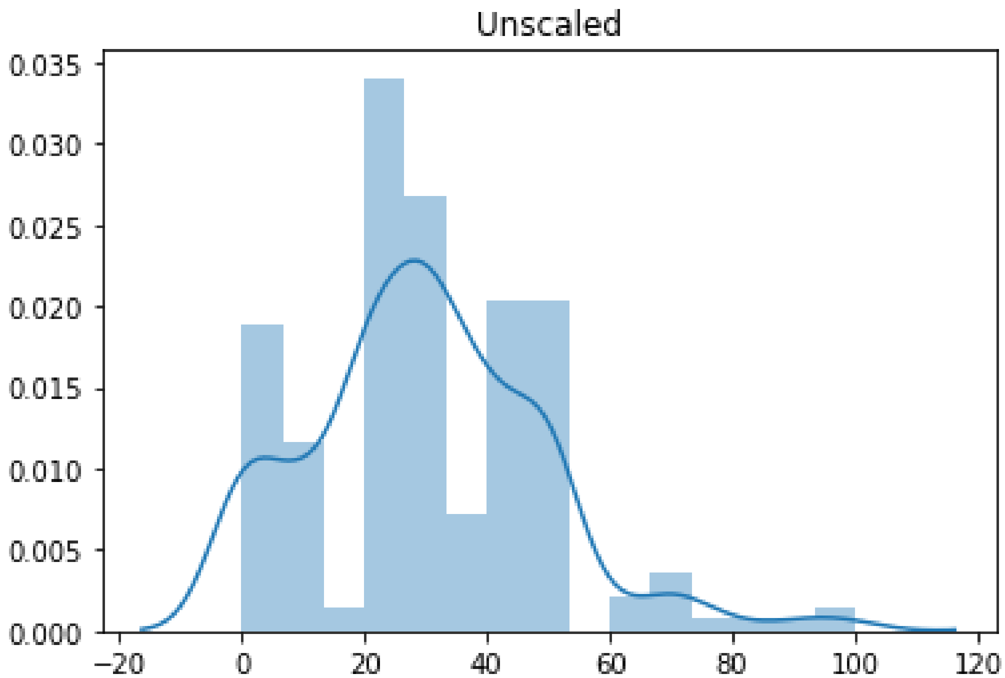


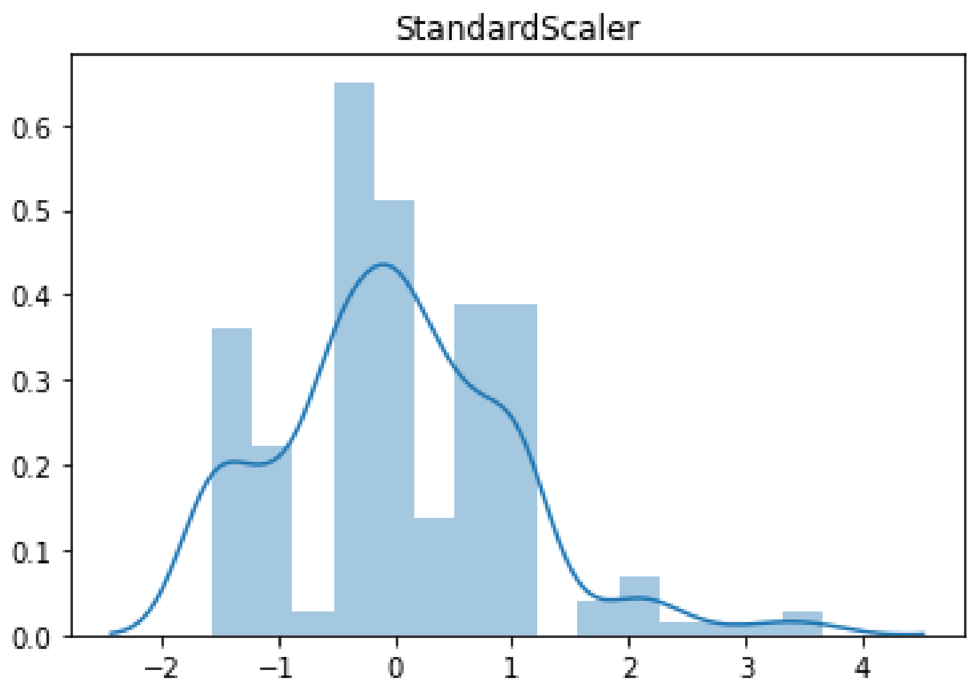
0.1.5 1.4 | Scaling of Variables

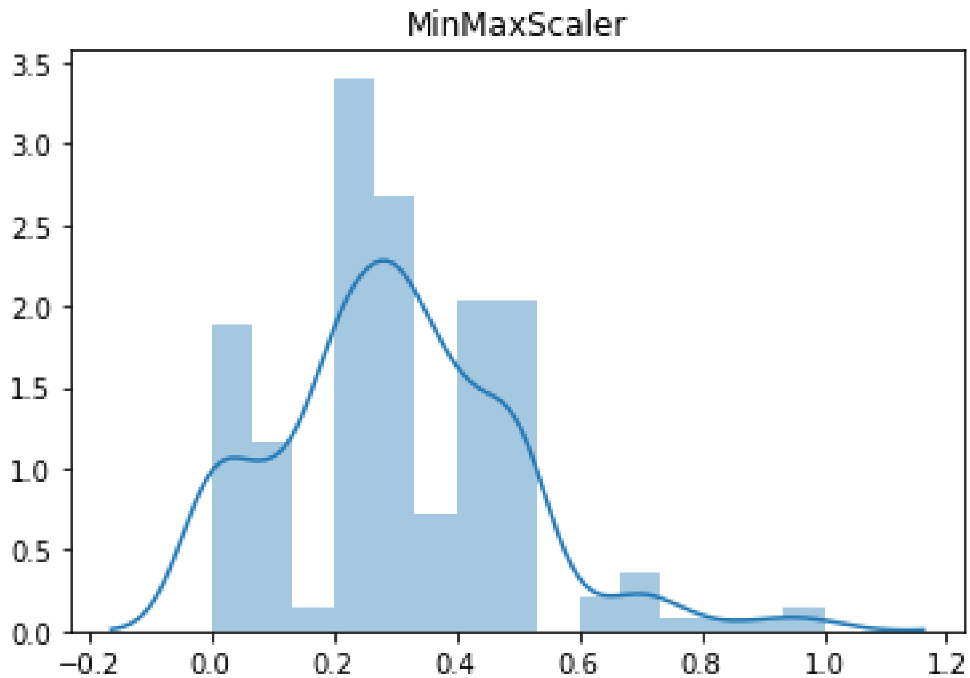
```
In [9]: from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.preprocessing import RobustScaler
        from sklearn.preprocessing import Normalizer
        from sklearn.preprocessing import QuantileTransformer

        X = df[["Professional_Python"]].dropna()
        unscaled_fig, ax = plt.subplots()
        sns.distplot(X).set_title('Unscaled')
        #unscaled_fig.savefig('Transformation-Unscaled' + '.pdf',
```

```
#    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
#    orientation='portrait', papertype=None, format=None,
#    transparent=True, pad_inches=0.25, frameon=None)

standard_fig, ax = plt.subplots()
sns.distplot(StandardScaler().fit_transform(X)).set_title('StandardScaler')
#standard_fig.savefig('Transformation-StandardScaler' + '.pdf',
#    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
#    orientation='portrait', papertype=None, format=None,
#    transparent=True, pad_inches=0.25, frameon=None)

minmax_fig, ax = plt.subplots()
sns.distplot(MinMaxScaler().fit_transform(X)).set_title('MinMaxScaler')
minmax_fig.savefig('Transformation-MinMaxScaler' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)

#plotting a variables against a few scaling options. StandardScaler and MinMaxScaler s
#but StandardScaler is based around 0 which is more intuitive. Additionally, the value
#extreme outliers, which if they were minmax might be be a better option.
```

```
C:\Users\David\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:625: DataConversionWa
  return self.partial_fit(X, y)
C:\Users\David\Anaconda3\lib\site-packages\sklearn\base.py:462: DataConversionWarning: Data wi
  return self.fit(X, **fit_params).transform(X)
C:\Users\David\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323: DataConversionWa
  return self.partial_fit(X, y)
```


Unscaled
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−20
0
20
40
60
80
100
120


StandardScaler
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2
−1
0
1
2
3
4




```
In [16]: #scale all numeric values in data
         df2 = pd.DataFrame(StandardScaler().fit_transform(df[personal_fields+professional_fiel
                                    columns = df[personal_fields+professional_fields+industry_fields+cl
         df2.head()
```

```
C:\Users\David\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:625: DataConversionWa
  return self.partial_fit(X, y)
C:\Users\David\Anaconda3\lib\site-packages\sklearn\base.py:462: DataConversionWarning: Data wi
  return self.fit(X, **fit_params).transform(X)
```

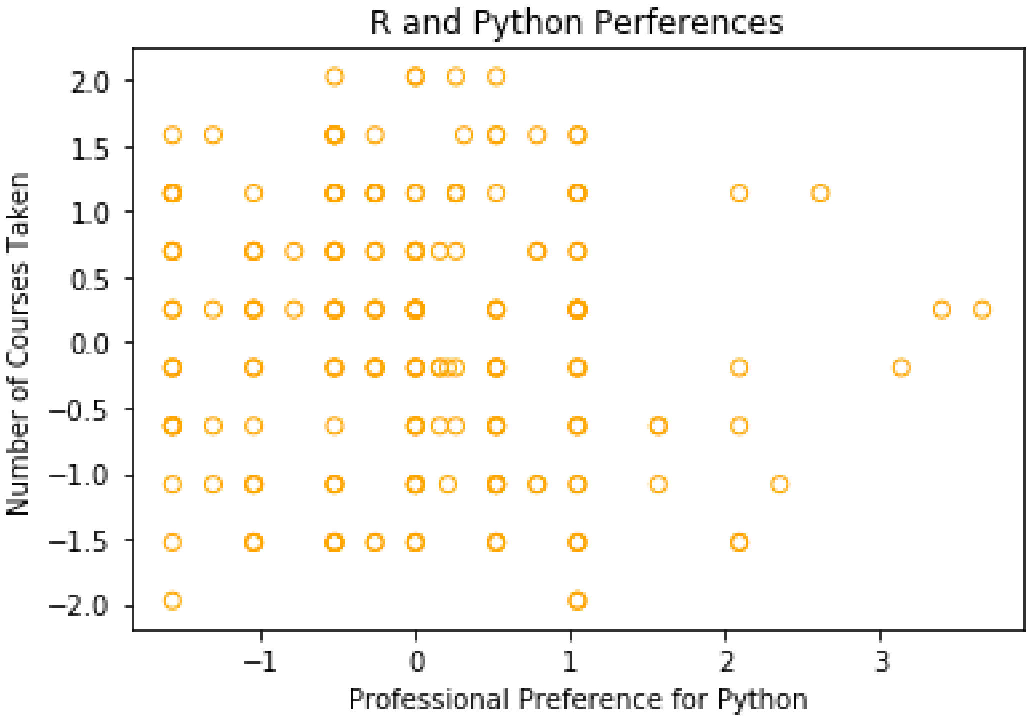
```
Out[16]:               Personal_JavaScalaSpark  Personal_JavaScriptHTMLCSS  \
         RespondentID
         5135740122                  -0.892507                   -0.711586
         5133300037                  -0.011911                    0.771781
         5132253300                   0.868684                   -0.711586
         5132096630                  -0.011911                    0.771781
         5131990362                   0.868684                   -0.711586

                       Personal_Python  Personal_R  Personal_SAS  \
         RespondentID
         5135740122          -2.015302    0.885401      2.454294
         5133300037           1.203583   -0.490044     -1.223949
         5132253300           0.559806    0.197679     -1.223949
         5132096630          -0.405860   -0.146183      0.247349
```

```
5131990362          -2.015302   2.260846    -0.488300

              Professional_JavaScalaSpark  Professional_JavaScriptHTMLCSS  \
RespondentID
5135740122                      -0.704282                       -0.541476
5133300037                       1.198934                        1.776256
5132253300                      -0.704282                       -0.541476
5132096630                       0.057005                        0.385617
5131990362                       0.818291                       -0.541476

              Professional_Python  Professional_R  Professional_SAS  ...  \
RespondentID                                                         ...
5135740122              -1.572322       -0.548894          3.009440  ...
5133300037              -0.001518       -0.789311         -0.982832  ...
5132253300               0.522084        0.172357          0.081774  ...
5132096630              -0.263318       -0.068060          0.081774  ...
5131990362              -1.572322        2.095694         -0.982832  ...

              PREDICT453  PREDICT454  PREDICT455  PREDICT456  PREDICT457  \
RespondentID
5135740122     -0.236902   -0.157329   -0.411693   -0.172774   -0.140372
5133300037     -0.236902   -0.157329   -0.411693   -0.172774   -0.140372
5132253300     -0.236902   -0.157329   -0.411693   -0.172774   -0.140372
5132096630     -0.236902   -0.157329   -0.411693   -0.172774   -0.140372
5131990362     -0.236902   -0.157329   -0.411693   -0.172774   -0.140372

              Courses_Completed_Prog  Python_Course_Interest  \
RespondentID
5135740122                 -1.972167               -0.790552
5133300037                  0.248935               -1.798517
5132253300                 -0.639506                0.889391
5132096630                  0.248935                0.385408
5131990362                 -0.639506               -0.454563

              Foundations_DE_Course_Interest  Analytics_App_Course_Interest  \
RespondentID
5135740122                          0.983034                      -0.123356
5133300037                         -0.247489                       1.021556
5132253300                          0.367773                       1.315124
5132096630                          0.060142                       1.021556
5131990362                         -1.478012                      -0.446280

              Systems_Analysis_Course_Interest
RespondentID
5135740122                           -0.108502
5133300037                           -0.108502
5132253300                            0.190402
5132096630                            0.847991
```

```
5131990362                                    0.788210

[5 rows x 35 columns]
```

0.1.6 1.4.1 | Analysis of Language Preferences (Scaled)

```
In [17]: # with scaled data I can compare two variables that before had two very different ran
         # single scatter plot example
         fig, axis = plt.subplots()
         axis.set_xlabel('Professional Preference for Python')
         axis.set_ylabel('Number of Courses Taken')
         plt.title('R and Python Perferences')
         scatter_plot = axis.scatter(df2['Professional_Python'],
             df2["Courses_Completed_Prog"],
             facecolors = 'none',
             edgecolors = 'orange')


         #this further cements the prognosis that there are no correlations between courses ta
```



0.1.7 1.5 | Analyis of New Courses

```
In [18]: df[interest_fields].head()
```

Out[18]:

```
                Python_Course_Interest  Foundations_DE_Course_Interest  \
RespondentID
5135740122                        50.0                            90.0
5133300037                        20.0                            50.0
5132253300                       100.0                            70.0
5132096630                        85.0                            60.0
5131990362                        60.0                            10.0

              Analytics_App_Course_Interest  Systems_Analysis_Course_Interest
RespondentID
5135740122                             51.0                              50.0
5133300037                             90.0                              50.0
5132253300                            100.0                              60.0
5132096630                             90.0                              82.0
5131990362                             40.0                              80.0
```

In [19]:

```python
df[interest_fields].describe()
```

Out[19]:

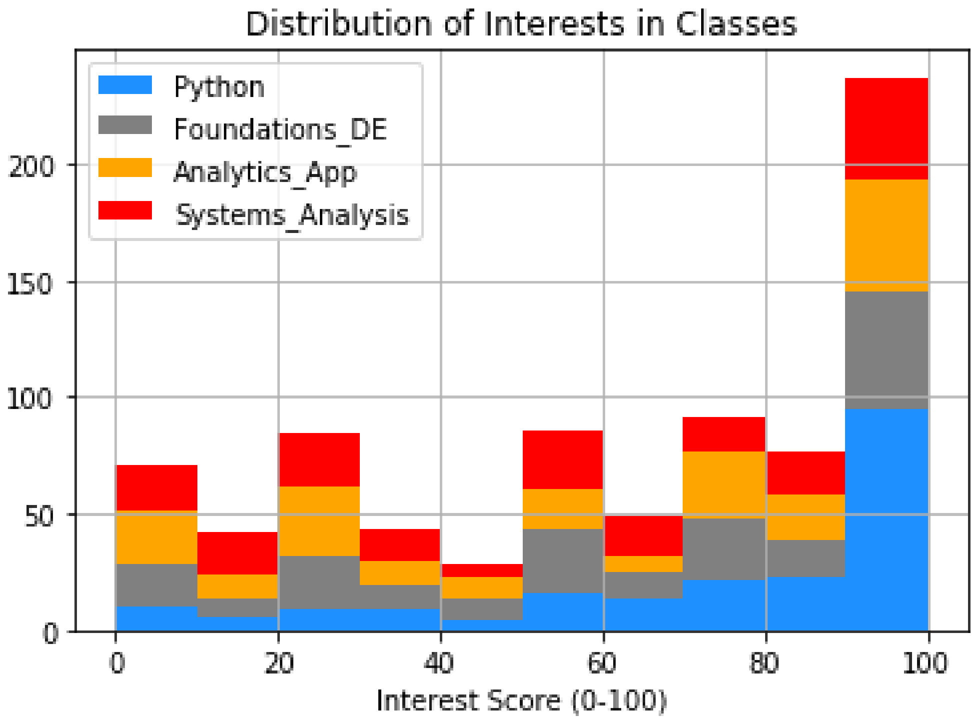
```
       Python_Course_Interest  Foundations_DE_Course_Interest  \
count              206.000000                      200.000000
mean                73.529126                       58.045000
std                 29.835429                       32.588079
min                  0.000000                        0.000000
25%                 53.000000                       29.500000
50%                 82.500000                       60.000000
75%                100.000000                       89.250000
max                100.000000                      100.000000

       Analytics_App_Course_Interest  Systems_Analysis_Course_Interest
count                     203.000000                        200.000000
mean                       55.201970                         53.630000
std                        34.147954                         33.539493
min                         0.000000                          0.000000
25%                        25.000000                         21.500000
50%                        60.000000                         51.500000
75%                        85.000000                         80.250000
max                       100.000000                        100.000000
```

In [20]:

```python
#isolate age values by class
x1 = df['Python_Course_Interest']
x2 = df['Foundations_DE_Course_Interest']
x3 = df['Analytics_App_Course_Interest']
x4 = df['Systems_Analysis_Course_Interest']

#combine threads and plot
plt.hist([x1,x2,x3,x4],stacked = True,color = ["dodgerblue","grey","orange","red"])
plt.title("Distribution of Interests in Classes")
plt.xlabel("Interest Score (0-100)")
```

```
plt.legend({'Python':'dodgerblue','Foundations_DE':"grey",'Analytics_App':"orange",'Sy
plt.grid(True)
plt.show()

#It looks like the Python course has garnered the most interest receieving the highes
# 100 scores.
```

```
C:\Users\David\Anaconda3\lib\site-packages\numpy\lib\histograms.py:824: RuntimeWarning: invalic
  keep = (tmp_a >= first_edge)
C:\Users\David\Anaconda3\lib\site-packages\numpy\lib\histograms.py:825: RuntimeWarning: invalic
  keep &= (tmp_a <= last_edge)
```



In [21]:
```
#isolate age values by class
x1 = df[df["Courses_Completed_Bucket"]=='0-3']['Python_Course_Interest']
x2 = df[df["Courses_Completed_Bucket"]=='4-6']['Python_Course_Interest']
x3 = df[df["Courses_Completed_Bucket"]=='7-10']['Python_Course_Interest']

#combine threads and plot
plt.hist([x1,x2,x3],stacked = True,color = ["dodgerblue","grey","orange"])
plt.title("Distribution of Interests by Courses Taken")
plt.xlabel("Interest Score (0-100)")
plt.legend({'0-3':'dodgerblue','4-6':"grey",'7-10':"orange"})
plt.grid(True)
```

```
plt.show()

#It looks like the Python course has garnered the most interest receieving the highes
# 100 scores.
```