Wanat_Assignment_1_Python

June 30, 2019

In [1]:
```
# Jump-Start Example: Python analysis of MSPA Software Survey

# Update 2017-09-21 by Tom Miller and Kelsey O'Neill
# Update 2018-06-30 by Tom Miller v005 transformation code added

# tested under Python 3.6.1 :: Anaconda custom (x86_64)
# on Windows 10.0 and Mac OS Sierra 10.12.2

# shows how to read in data from a comma-delimited text file
# manipuate data, create new count variables, define categorical variables,
# work with dictionaries and lambda mapping functions for recoding data

# visualizations in this program are routed to external pdf files
# so they may be included in printed or electronic reports

# prepare for Python version 3x features and functions
# these two lines of code are needed for Python 2.7 only
# commented out for Python 3.x versions
# from __future__ import division, print_function
# from future_builtins import ascii, filter, hex, map, oct, zip
```

In [2]:
```
# external libraries for visualizations and data manipulation
# ensure that these packages have been installed prior to calls
import pandas as pd  # data frame operations
import numpy as np  # arrays and math functions
import pandas_profiling
import matplotlib
import matplotlib.pyplot as plt  # static plotting
import seaborn as sns  # pretty plotting, including heat map
```

In [3]:
```
# correlation heat map setup for seaborn
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)

np.set_printoptions(precision=3)
```

In [4]:
```
# read in comma-delimited text file, creating a pandas DataFrame object
# note that IPAddress is formatted as an actual IP address
# but is actually a random-hash of the original IP address
valid_survey_input = pd.read_csv('mspa-survey-data.csv')

# use the RespondentID as label for the rows... the index of DataFrame
valid_survey_input.set_index('RespondentID', drop = True, inplace = True)

# examine the structure of the DataFrame object
print('\nContents of initial survey data ---------------')

# could use len() or first index of shape() to get number of rows/observations
print('\nNumber of Respondents =', len(valid_survey_input))
```

```
Contents of initial survey data ---------------

Number of Respondents = 207
```

In [5]:
```
#The shape attribute tells us the numbers of rows and columns in the data frame
print('\nThe shape of the dataframe (rows, columns):')
valid_survey_input.shape
```

```
The shape of the dataframe (rows, columns):
```

Out[5]: (207, 40)

In [6]:
```
# provide the number of columns in data frame
print('\nNumber of columns in data frame = ', valid_survey_input.shape[1])
```

```
Number of columns in data frame =  40
```

```
In [7]: # show the column/variable names of the DataFrame
        # note that RespondentID is no longer present
        print(valid_survey_input.columns)
```

```
Index(['Personal_JavaScalaSpark', 'Personal_JavaScriptHTMLCSS',
       'Personal_Python', 'Personal_R', 'Personal_SAS',
       'Professional_JavaScalaSpark', 'Professional_JavaScriptHTMLCSS',
       'Professional_Python', 'Professional_R', 'Professional_SAS',
       'Industry_JavaScalaSpark', 'Industry_JavaScriptHTMLCSS',
       'Industry_Python', 'Industry_R', 'Industry_SAS',
       'Python_Course_Interest', 'Foundations_DE_Course_Interest',
       'Analytics_App_Course_Interest', 'Systems_Analysis_Course_Interest',
       'Courses_Completed', 'PREDICT400', 'PREDICT401', 'PREDICT410',
       'PREDICT411', 'PREDICT413', 'PREDICT420', 'PREDICT422', 'PREDICT450',
       'PREDICT451', 'PREDICT452', 'PREDICT453', 'PREDICT454', 'PREDICT455',
       'PREDICT456', 'PREDICT457', 'OtherPython', 'OtherR', 'OtherSAS',
       'Other', 'Graduate_Date'],
      dtype='object')
```

```
In [8]: # examine the data types for each column in the data frame
        print(valid_survey_input.dtypes)
```

```
Personal_JavaScalaSpark               int64
Personal_JavaScriptHTMLCSS            int64
Personal_Python                       int64
Personal_R                            int64
Personal_SAS                          int64
Professional_JavaScalaSpark           int64
Professional_JavaScriptHTMLCSS        int64
Professional_Python                   int64
Professional_R                        int64
Professional_SAS                      int64
Industry_JavaScalaSpark               int64
Industry_JavaScriptHTMLCSS            int64
Industry_Python                       int64
Industry_R                            int64
Industry_SAS                          int64
Python_Course_Interest              float64
Foundations_DE_Course_Interest      float64
Analytics_App_Course_Interest       float64
Systems_Analysis_Course_Interest    float64
Courses_Completed                   float64
PREDICT400                           object
PREDICT401                           object
PREDICT410                           object
PREDICT411                           object
PREDICT413                           object
```

```
PREDICT420                         object
PREDICT422                         object
PREDICT450                         object
PREDICT451                         object
PREDICT452                         object
PREDICT453                         object
PREDICT454                         object
PREDICT455                         object
PREDICT456                         object
PREDICT457                         object
OtherPython                        object
OtherR                             object
OtherSAS                           object
Other                              object
Graduate_Date                      object
dtype: object
```

In [9]:
```python
#total number of NaN values in each column
valid_survey_input.isnull().sum()
```

Out[9]:
```
Personal_JavaScalaSpark              0
Personal_JavaScriptHTMLCSS           0
Personal_Python                      0
Personal_R                           0
Personal_SAS                         0
Professional_JavaScalaSpark          0
Professional_JavaScriptHTMLCSS       0
Professional_Python                  0
Professional_R                       0
Professional_SAS                     0
Industry_JavaScalaSpark              0
Industry_JavaScriptHTMLCSS           0
Industry_Python                      0
Industry_R                           0
Industry_SAS                         0
Python_Course_Interest               1
Foundations_DE_Course_Interest       7
Analytics_App_Course_Interest        4
Systems_Analysis_Course_Interest     7
Courses_Completed                   20
PREDICT400                          44
PREDICT401                          36
PREDICT410                          62
PREDICT411                          94
PREDICT413                         148
PREDICT420                          80
PREDICT422                         159
```

```
PREDICT450                        190
PREDICT451                        200
PREDICT452                        194
PREDICT453                        196
PREDICT454                        202
PREDICT455                        177
PREDICT456                        201
PREDICT457                        203
OtherPython                       202
OtherR                            193
OtherSAS                          205
Other                             181
Graduate_Date                       3
dtype: int64
```

0.1 Profile Analysis of Data Set

```
In [10]: #The ProfileReport in the pandas_profiling package provides a summary of the dataset

         pandas_profiling.ProfileReport(valid_survey_input)

Out[10]: <pandas_profiling.ProfileReport at 0x113ad55f8>

In [11]: # Save the pandas profile report to html file
         profile = pandas_profiling.ProfileReport(valid_survey_input)
         profile.to_file()
```

0.2 Analysis of Course Completion

```
In [12]: # Analysis of Course Completion
         # shorten the variable/column names for software preference variables
         survey_df = valid_survey_input.rename(index=str, columns={
             'Personal_JavaScalaSpark': 'My_Java',
             'Personal_JavaScriptHTMLCSS': 'My_JS',
             'Personal_Python': 'My_Python',
             'Personal_R': 'My_R',
             'Personal_SAS': 'My_SAS',
             'Professional_JavaScalaSpark': 'Prof_Java',
             'Professional_JavaScriptHTMLCSS': 'Prof_JS',
             'Professional_Python': 'Prof_Python',
             'Professional_R': 'Prof_R',
             'Professional_SAS': 'Prof_SAS',
             'Industry_JavaScalaSpark': 'Ind_Java',
             'Industry_JavaScriptHTMLCSS': 'Ind_JS',
             'Industry_Python': 'Ind_Python',
             'Industry_R': 'Ind_R',
             'Industry_SAS': 'Ind_SAS'})
```

```
In [13]: # Print the first five rows of the survey_df that contains
         # shortened variable/column names for software preference variables
         # to check that it worked

         print(pd.DataFrame.head(survey_df))
```

```
              My_Java  My_JS  My_Python  My_R  My_SAS  Prof_Java  Prof_JS  \
RespondentID
5135740122          0      0          0    50      50          0        0
5133300037         10     10         50    30       0         25       25
5132253300         20      0         40    40       0          0        0
5132096630         10     10         25    35      20         10       10
5131990362         20      0          0    70      10         20        0

              Prof_Python  Prof_R  Prof_SAS      ...        PREDICT453  \
RespondentID                                     ...
5135740122              0      25        75      ...               NaN
5133300037             30      20         0      ...               NaN
5132253300             40      40        20      ...               NaN
5132096630             25      35        20      ...               NaN
5131990362              0      80         0      ...               NaN

              PREDICT454  PREDICT455  PREDICT456  PREDICT457  OtherPython  \
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

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

[5 rows x 40 columns]
```

```
In [14]: # descriptive statistics for one variable
         print('\nDescriptive statistics for courses completed ---------------')
         print(survey_df['Courses_Completed'].describe())
```

```
Descriptive statistics for courses completed ---------------
count    187.000000
```

```
mean       6.342246
std        3.170849
min        1.000000
25%        4.000000
50%        6.000000
75%        9.000000
max       12.000000
Name: Courses_Completed, dtype: float64
```

In [15]:
```
#Summarize courses completed by displaying the number of students in each category
courses_completed_counts = survey_df['Courses_Completed'].value_counts()
print('\nThe number of courses completed by students: ')
courses_completed_counts.sort_index()
```

```
The number of courses completed by students:
```

Out[15]:
```
1.0      6
2.0     25
3.0     13
4.0     13
5.0     24
6.0     16
7.0     24
8.0     11
9.0     14
10.0    20
11.0    11
12.0    10
Name: Courses_Completed, dtype: int64
```

In [16]:
```
#Display the percentages of each course completed category
print('\nPercentage of the Number of Courses Completed: ')
((courses_completed_counts/sum(courses_completed_counts))*100)
```

```
Percentage of the Number of Courses Completed:
```

Out[16]:
```
2.0     13.368984
5.0     12.834225
7.0     12.834225
10.0    10.695187
6.0      8.556150
9.0      7.486631
3.0      6.951872
4.0      6.951872
```

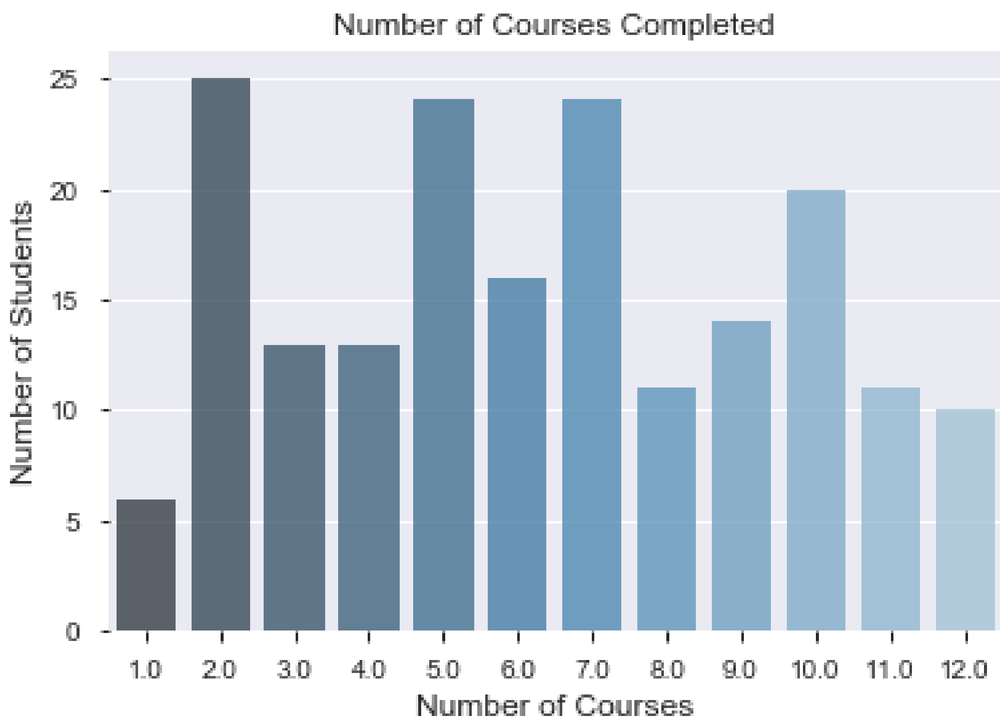
```
8.0      5.882353
11.0     5.882353
12.0     5.347594
1.0      3.208556
Name: Courses_Completed, dtype: float64
```

In [17]:
```
# Display the percentages of each course completed category
# Another way to calculate
print('\nPercentage of the Number of Courses Completed: ')
survey_df['Courses_Completed'].value_counts(normalize=True)*100
```

```
Percentage of the Number of Courses Completed:
```

Out[17]:
```
2.0     13.368984
5.0     12.834225
7.0     12.834225
10.0    10.695187
6.0      8.556150
9.0      7.486631
3.0      6.951872
4.0      6.951872
8.0      5.882353
11.0     5.882353
12.0     5.347594
1.0      3.208556
Name: Courses_Completed, dtype: float64
```

In [18]:
```
# Examine the number of courses completed by student with a bar graph

%matplotlib inline
course_completed_date_fig, ax = plt.subplots()
sns.barplot(y = courses_completed_counts.values,
            x = courses_completed_counts.index, alpha=0.8,
            palette="Blues_d").set_title('Number of Courses Completed')
ax.set_xlabel('Number of Courses', fontsize=12)
ax.set_ylabel('Number of Students', fontsize=12)
course_completed_date_fig.savefig('CourseCompleted' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```




```
In [19]: # The selected columns were placed into a new dataframe.
         # These columns are courses with a programming component
         # that were completed by the respondent at the time of the survey.
         # If the course was completed, the row contains a entry for that course.
         # If not completed, the row contains a 'NaN'.

         class_completed_df = valid_survey_input[['PREDICT400', 'PREDICT401', 'PREDICT410',
                           'PREDICT411', 'PREDICT413', 'PREDICT420',
                           'PREDICT422', 'PREDICT450', 'PREDICT451',
                           'PREDICT452', 'PREDICT453', 'PREDICT454',
                           'PREDICT455', 'PREDICT456', 'PREDICT457']]

In [20]: # View new dataframe for classes completed at time of survey

         class_completed_df.head()

Out[20]:                                          PREDICT400  \
         RespondentID
         5135740122                                      NaN
         5133300037    PREDICT 400 Math for Modelers (Python)
         5132253300    PREDICT 400 Math for Modelers (Python)
         5132096630    PREDICT 400 Math for Modelers (Python)
         5131990362                                      NaN
```

```
                                                     PREDICT401  \
RespondentID
5135740122                                                  NaN
5133300037    PREDICT 401 Introduction to Statistical Analys...
5132253300    PREDICT 401 Introduction to Statistical Analys...
5132096630    PREDICT 401 Introduction to Statistical Analys...
5131990362    PREDICT 401 Introduction to Statistical Analys...

                                                     PREDICT410  \
RespondentID
5135740122                                                  NaN
5133300037    PREDICT 410 Regression and Multivariate Analys...
5132253300                                                  NaN
5132096630    PREDICT 410 Regression and Multivariate Analys...
5131990362    PREDICT 410 Regression and Multivariate Analys...

                                               PREDICT411 PREDICT413  \
RespondentID
5135740122                                            NaN        NaN
5133300037    PREDICT 411 Generalized Linear Models (SAS)        NaN
5132253300                                            NaN        NaN
5132096630    PREDICT 411 Generalized Linear Models (SAS)        NaN
5131990362    PREDICT 411 Generalized Linear Models (SAS)        NaN

                                                     PREDICT420 PREDICT422  \
RespondentID
5135740122                                                  NaN        NaN
5133300037    PREDICT 420 Database Systems and Data Preparat...        NaN
5132253300    PREDICT 420 Database Systems and Data Preparat...        NaN
5132096630    PREDICT 420 Database Systems and Data Preparat...        NaN
5131990362                                                  NaN        NaN

             PREDICT450 PREDICT451 PREDICT452 PREDICT453 PREDICT454  \
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

             PREDICT455 PREDICT456 PREDICT457
RespondentID
5135740122          NaN        NaN        NaN
5133300037          NaN        NaN        NaN
5132253300          NaN        NaN        NaN
5132096630          NaN        NaN        NaN
5131990362          NaN        NaN        NaN
```

In [21]:
```
# It is known that the survey dataframe contains 207 rows.
# The class_completed_df will be examined for null values with the isnull function.
# The number of null values will be subtracted from 207, and the resulting number
# will be the number of respondents that have completed the course.

done_df = 207 - class_completed_df.isnull().sum()

print('\nThe number of respondents that have completed the following courses:')
done_df.sort_values(ascending=False)
```

```
The number of respondents that have completed the following courses:
```

Out[21]:
```
PREDICT401    171
PREDICT400    163
PREDICT410    145
PREDICT420    127
PREDICT411    113
PREDICT413     59
PREDICT422     48
PREDICT455     30
PREDICT450     17
PREDICT452     13
PREDICT453     11
PREDICT451      7
PREDICT456      6
PREDICT454      5
PREDICT457      4
dtype: int64
```

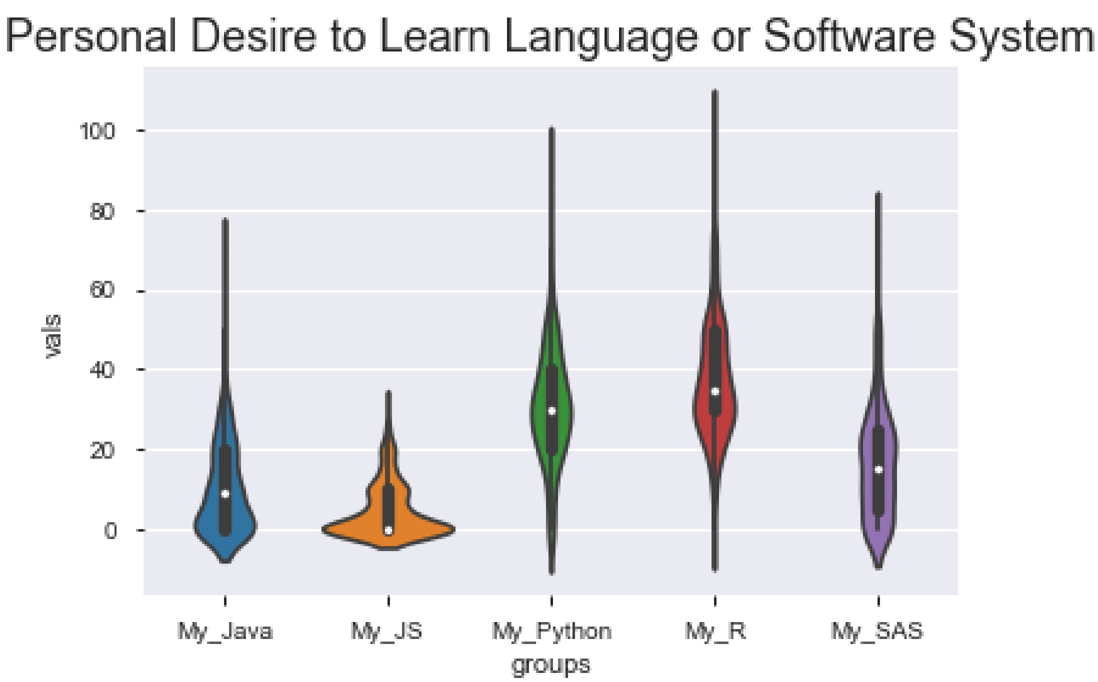
0.3 Analysis of Software Preference, Part I

In [22]:
```
# Examine the columns for respondent desire to learn for each
# of the five language/software options.
# The describe function provides a basic statistical analysis.

survey_df.iloc[:,0:5].describe()
```

Out[22]:

	My_Java	My_JS	My_Python	My_R	My_SAS
count	207.000000	207.000000	207.000000	207.000000	207.000000
mean	10.135266	4.797101	31.304348	37.125604	16.637681
std	11.383477	6.757764	15.570982	14.576003	13.626400
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	20.000000	30.000000	5.000000
50%	9.000000	0.000000	30.000000	35.000000	15.000000
75%	20.000000	10.000000	40.000000	50.000000	25.000000
max	70.000000	30.000000	90.000000	100.000000	75.000000

```
In [23]: # Place the columns for respondent desire to learn for each
         # of the five language/software options into a new data frame.
         # The dataframe columns are 'melted' together in order to examine in
         # a type of boxplot called a violinplot.
         # The shape in a violinplot demonstrates the distribution of the data.

         # The columns for respondent desire to learn each of the
         # five language/software options were examined.

         personal_df = survey_df.iloc[:,0:5]

         personal_melted_df = personal_df.melt(var_name='groups', value_name='vals')

         personal_desire_learn_fig, ax = plt.subplots()
         sns.violinplot(x="groups", y="vals", data=personal_melted_df)
         ax.set_title('Personal Desire to Learn Language or Software System', fontsize=18)
         personal_desire_learn_fig.savefig('PersonalDesireToLearn' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```



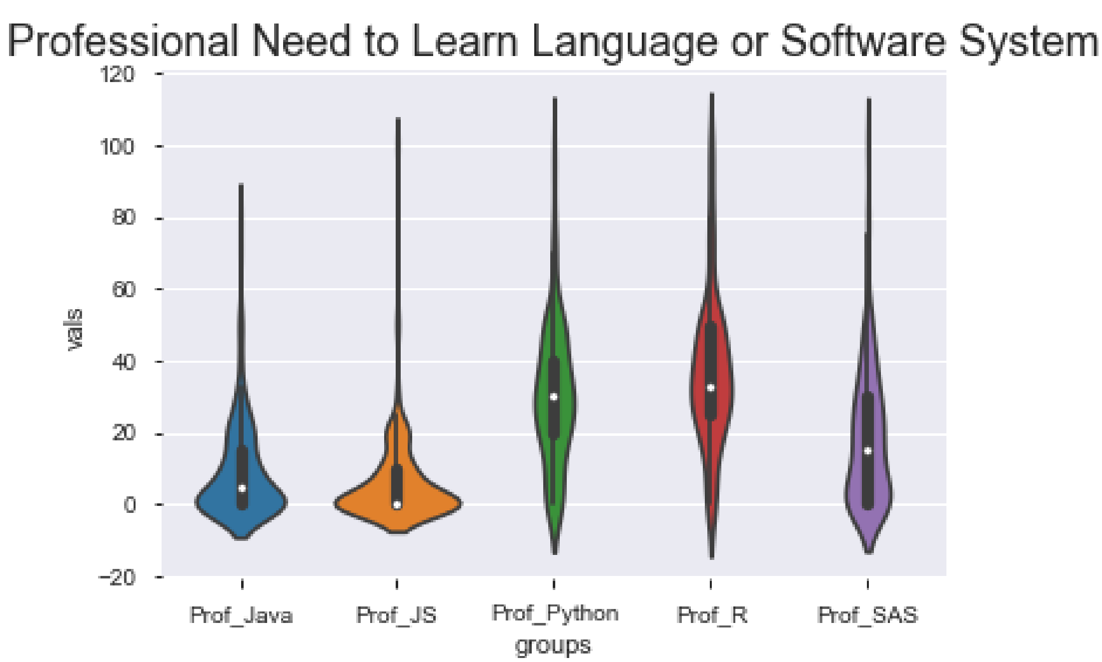

```
In [24]: # Examine the columns for respondent professional need to learn
         # for each of the five language/software options.
         # The describe function provides a basic statistical analysis.

         survey_df.iloc[:,5:10].describe()
```

Out[24]:

	Prof_Java	Prof_JS	Prof_Python	Prof_R	Prof_SAS
count	207.000000	207.000000	207.000000	207.000000	207.000000
mean	9.251208	5.840580	30.028986	36.415459	18.463768
std	13.167505	10.812555	19.144802	20.847606	18.831841
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	20.000000	25.000000	0.000000
50%	5.000000	0.000000	30.000000	33.000000	15.000000
75%	15.000000	10.000000	40.000000	50.000000	30.000000
max	80.000000	100.000000	100.000000	100.000000	100.000000

In [25]:

```
# Place the columns for respondent professional need to learn for each
# of the five language/software options into a new data frame.
# The dataframe columns are 'melted' together in order to examine in
# a type of boxplot called a violinplot.
# The shape in a violinplot demonstrates the distribution of the data.

# The columns for respondent professional need to learn each of the
# five language/software options were examined.
professional_df = survey_df.iloc[:,5:10]

professional_melted_df = professional_df.melt(var_name='groups', value_name='vals')

professional_need_learn_fig, ax = plt.subplots()
sns.violinplot(x="groups", y="vals", data=professional_melted_df)
ax.set_title('Professional Need to Learn Language or Software System', fontsize=18)
professional_need_learn_fig.savefig('ProfessionalNeedToLearn' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```

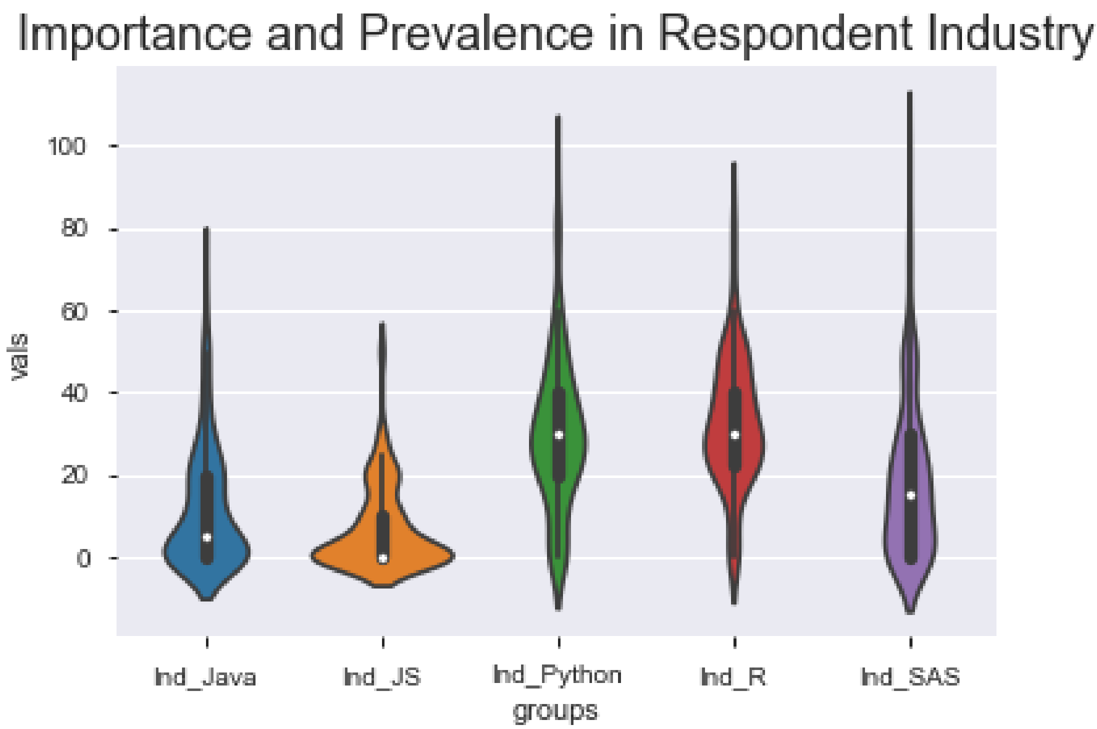


In [26]:
```python
# Examine the columns for respondent industry importance and prevalence to learn
# for each of the five language/software options.
# The describe function provides a basic statistical analysis.

survey_df.iloc[:,10:15].describe()
```

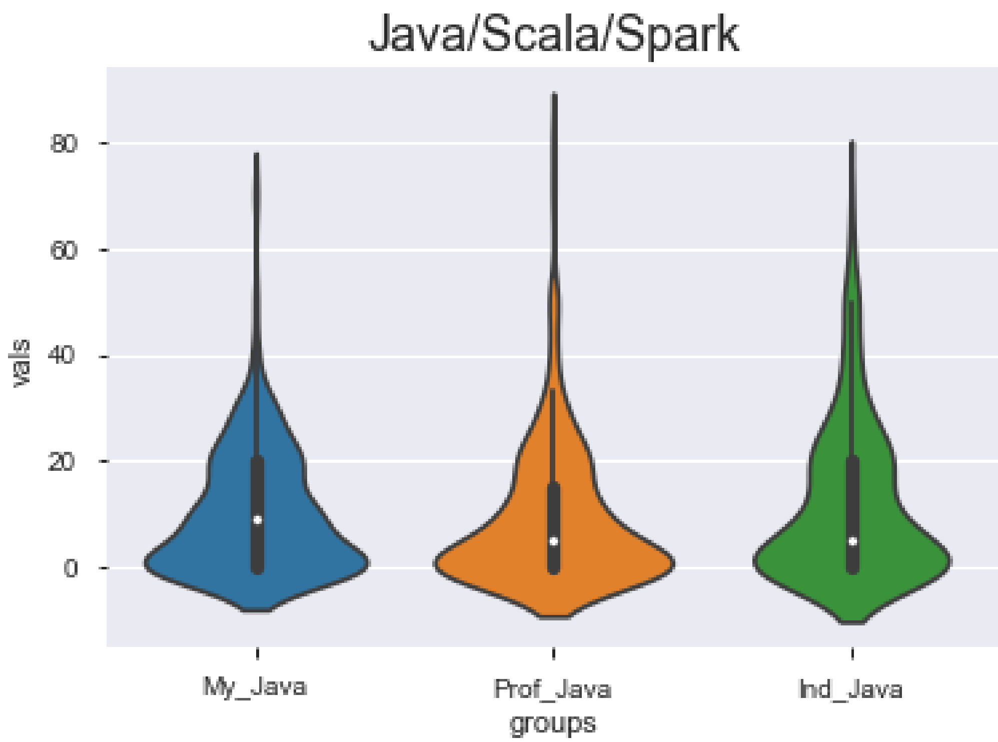
Out[26]:

	Ind_Java	Ind_JS	Ind_Python	Ind_R	Ind_SAS
count	207.000000	207.000000	207.000000	207.000000	207.000000
mean	11.942029	6.966184	29.772947	32.434783	18.884058
std	14.706399	10.030721	17.959816	15.912209	19.137623
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	20.000000	22.500000	0.000000
50%	5.000000	0.000000	30.000000	30.000000	15.000000
75%	20.000000	10.000000	40.000000	40.000000	30.000000
max	70.000000	50.000000	95.000000	85.000000	100.000000

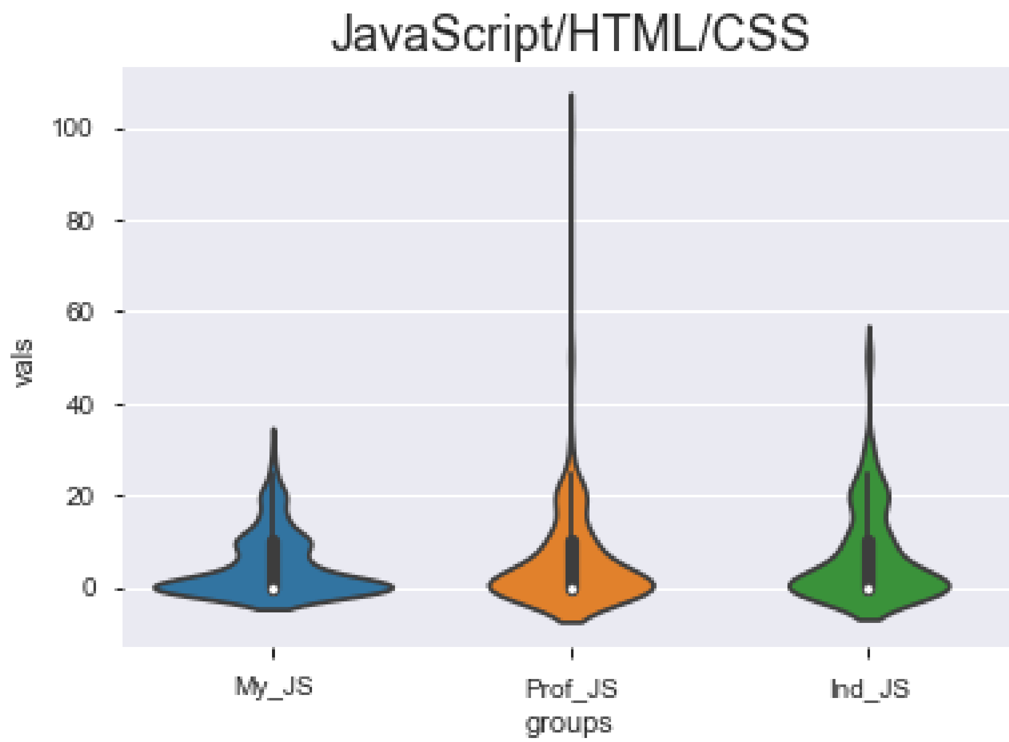
In [27]:
```python
# Place the columns for respondent industry importance and prevalence to learn for ea
# of the five language/software options into a new data frame.
# The dataframe columns are 'melted' together in order to examine in
# a type of boxplot called a violinplot.
# The shape in a violinplot demonstrates the distribution of the data.

# The columns for respondent industry importance and prevalence to learn
# each of the five language/software options were examined.
industry_df = survey_df.iloc[:,10:15]

industry_melted_df = industry_df.melt(var_name='groups', value_name='vals')


industry_importance_fig, ax = plt.subplots()
sns.violinplot(x="groups", y="vals", data=industry_melted_df)
ax.set_title('Importance and Prevalence in Respondent Industry', fontsize=18)
industry_importance_fig.savefig('IndustryImportance' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```



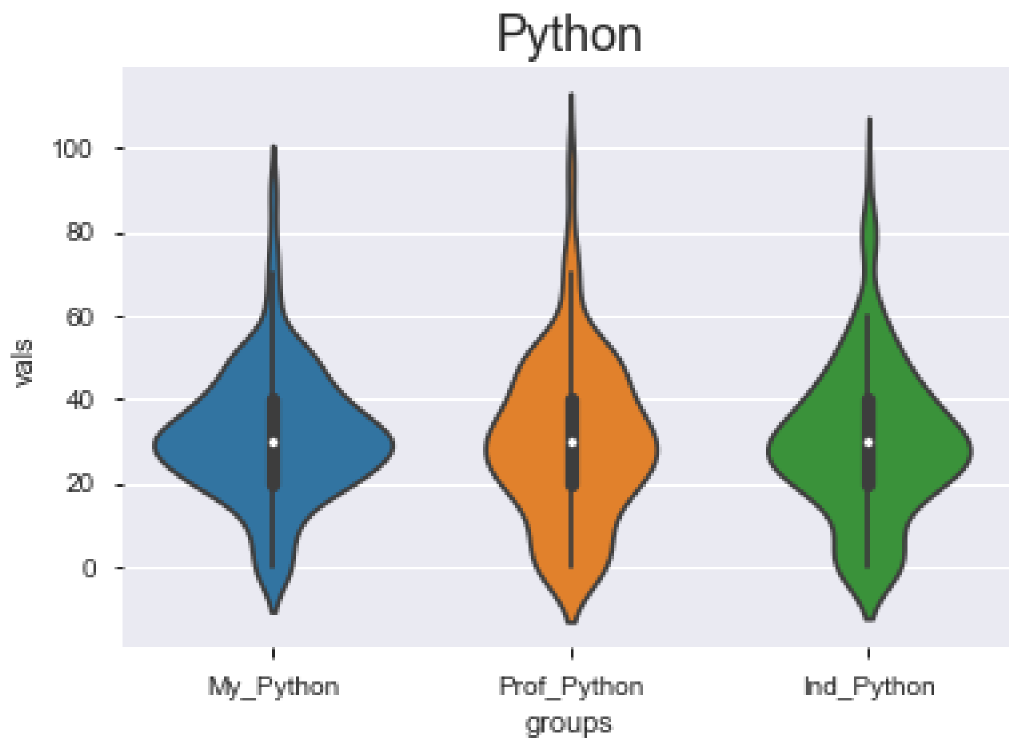
In [28]:
```
# Examining the Java/Scala/Spark responses

java_df = survey_df.iloc[:,[0,5,10]]
java_melted_df = java_df.melt(var_name='groups', value_name='vals')

java_scala_spark_fig, ax = plt.subplots()
sns.violinplot(x="groups", y="vals", data=java_melted_df)
ax.set_title('Java/Scala/Spark', fontsize=18)
java_scala_spark_fig.savefig('JavaScalaSpark' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```



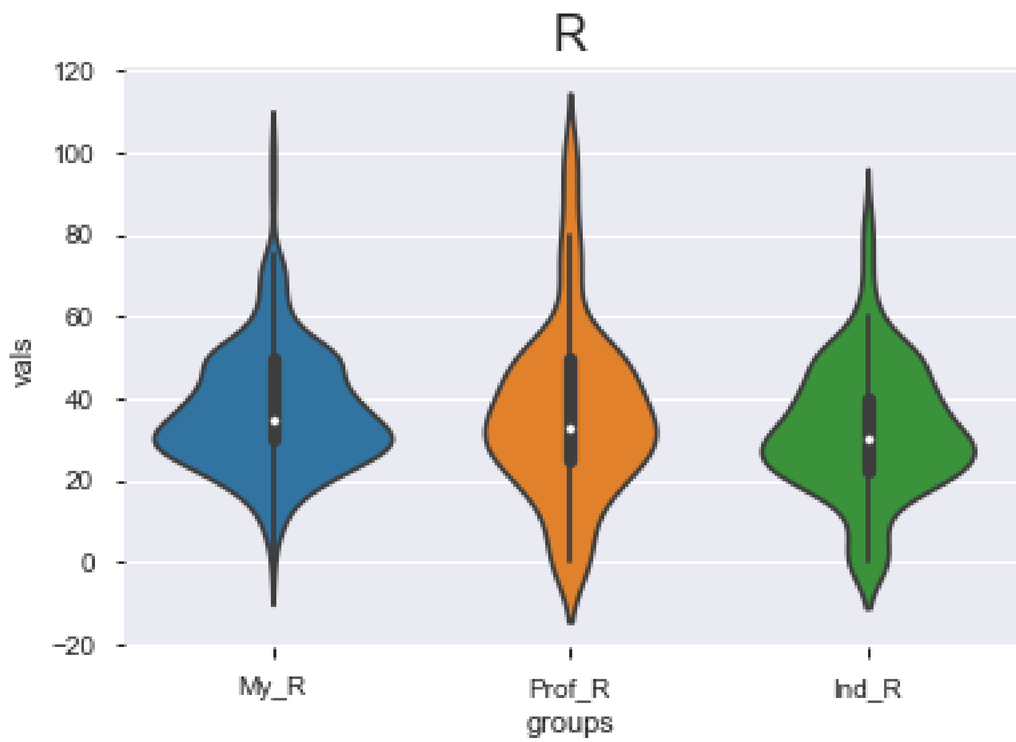

```
In [29]: # Examining the JavaScript/HTML/CSS responses
         js_df = survey_df.iloc[:,[1,6,11]]
         js_melted_df = js_df.melt(var_name='groups', value_name='vals')

         javascript_html_css_fig, ax = plt.subplots()
         sns.violinplot(x="groups", y="vals", data=js_melted_df)
         ax.set_title('JavaScript/HTML/CSS', fontsize=18)
         javascript_html_css_fig.savefig('JavaScriptHTMLCSS' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```



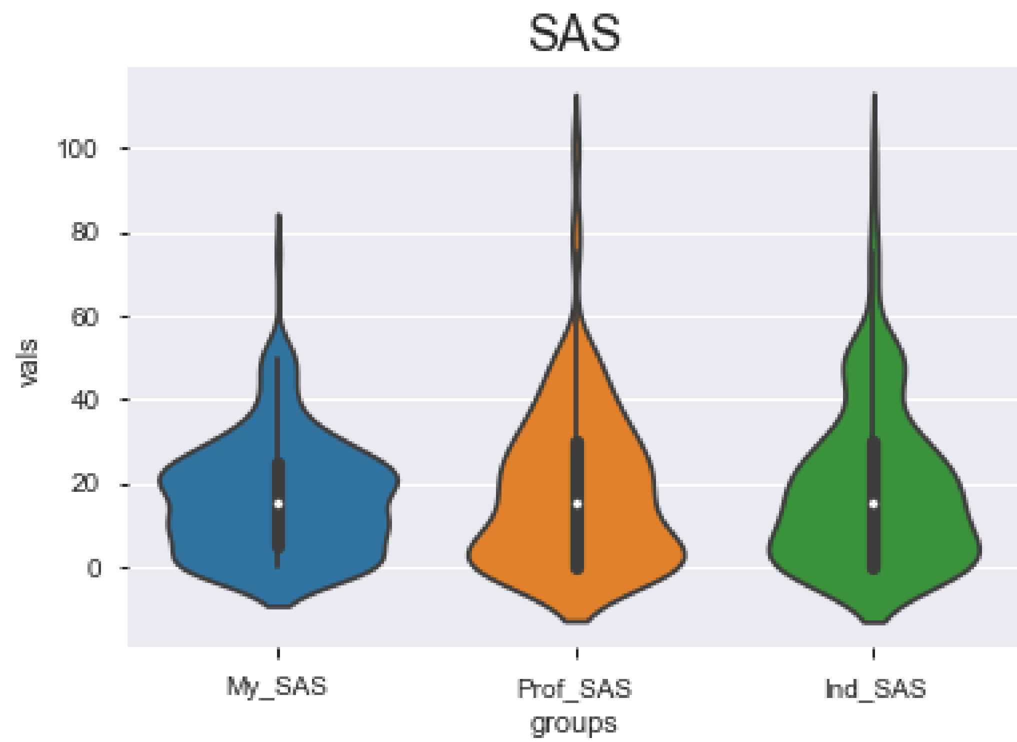
In [30]:
```
# Examining the Python responses
python_df = survey_df.iloc[:,[2,7,12]]
python_melted_df = python_df.melt(var_name='groups', value_name='vals')

python_fig, ax = plt.subplots()
sns.violinplot(x="groups", y="vals", data=python_melted_df)
ax.set_title('Python', fontsize=18)
python_fig.savefig('Python' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```




```
In [31]: # Examining the R responses
         r_df = survey_df.iloc[:,[3,8,13]]
         r_melted_df = r_df.melt(var_name='groups', value_name='vals')

         rlanguage_fig, ax = plt.subplots()
         sns.violinplot(x="groups", y="vals", data=r_melted_df)
         ax.set_title('R', fontsize=18)
         rlanguage_fig.savefig('Rlanguage' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```




```
In [32]: # Examining the SAS responses
         sas_df = survey_df.iloc[:,[4,9,14]]
         sas_melted_df = sas_df.melt(var_name='groups', value_name='vals')

         sas_fig, ax = plt.subplots()
         sns.violinplot(x="groups", y="vals", data=sas_melted_df)
         ax.set_title('SAS', fontsize=18)
         sas_fig.savefig('SAS' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```



0.4 Analysis of Software Preference, Part II

In [33]:
```
# define subset of survey DataFrame for analysis of software preferences
software_df = survey_df.loc[:, 'My_Java':'Ind_SAS']
```

In [34]:
```
# display the first five lines of the data frame that contains
# personal, professional, and industry preferences by language and software
display(pd.DataFrame.head(software_df))
```

RespondentID	My_Java	My_JS	My_Python	My_R	My_SAS	Prof_Java	Prof_JS	\
5135740122	0	0	0	50	50	0	0	
5133300037	10	10	50	30	0	25	25	
5132253300	20	0	40	40	0	0	0	
5132096630	10	10	25	35	20	10	10	
5131990362	20	0	0	70	10	20	0	

RespondentID	Prof_Python	Prof_R	Prof_SAS	Ind_Java	Ind_JS	Ind_Python	\
5135740122	0	25	75	0	0	0	
5133300037	30	20	0	20	25	40	
5132253300	40	40	20	30	0	30	
5132096630	25	35	20	10	10	25	

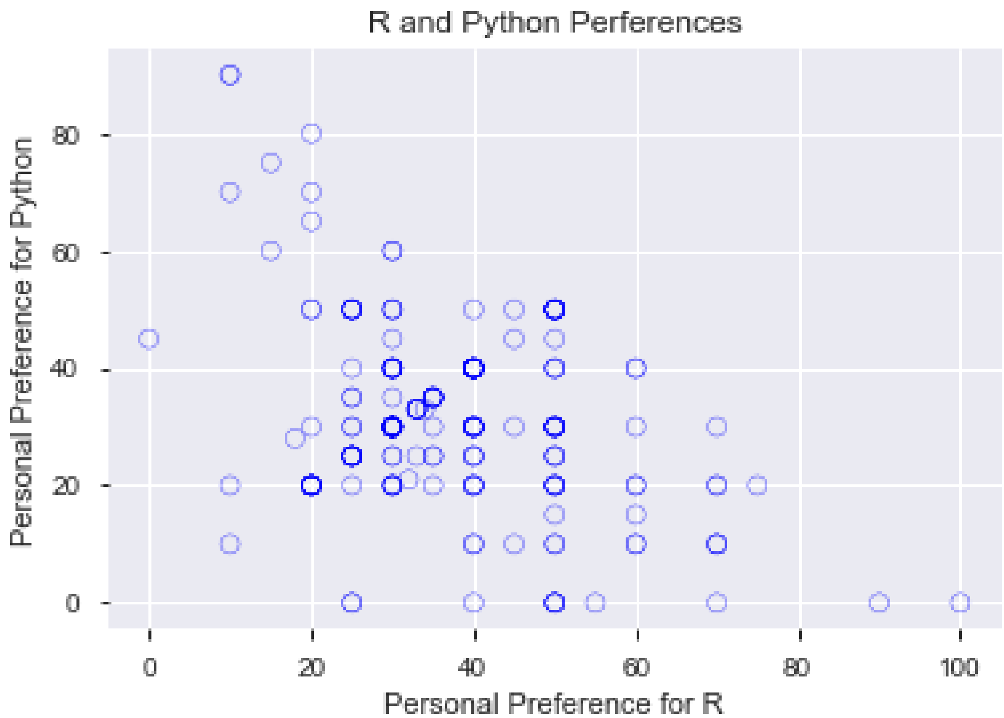
```
5131990362              0       80          0         40        0            0

              Ind_R  Ind_SAS
RespondentID
5135740122       50       50
5133300037       15        0
5132253300       40        0
5132096630       35       20
5131990362       60        0
```

```
In [35]: # descriptive statistics for software preference variables
         print('\nDescriptive statistics for survey data ---------------')
         print(software_df.describe())
```

```
Descriptive statistics for survey data ---------------
          My_Java       My_JS   My_Python        My_R      My_SAS   Prof_Java  \
count  207.000000  207.000000  207.000000  207.000000  207.000000  207.000000
mean    10.135266    4.797101   31.304348   37.125604   16.637681    9.251208
std     11.383477    6.757764   15.570982   14.576003   13.626400   13.167505
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.000000    0.000000   20.000000   30.000000    5.000000    0.000000
50%      9.000000    0.000000   30.000000   35.000000   15.000000    5.000000
75%     20.000000   10.000000   40.000000   50.000000   25.000000   15.000000
max     70.000000   30.000000   90.000000  100.000000   75.000000   80.000000

          Prof_JS  Prof_Python      Prof_R    Prof_SAS    Ind_Java  \
count  207.000000   207.000000  207.000000  207.000000  207.000000
mean     5.840580    30.028986   36.415459   18.463768   11.942029
std     10.812555    19.144802   20.847606   18.831841   14.706399
min      0.000000     0.000000    0.000000    0.000000    0.000000
25%      0.000000    20.000000   25.000000    0.000000    0.000000
50%      0.000000    30.000000   33.000000   15.000000    5.000000
75%     10.000000    40.000000   50.000000   30.000000   20.000000
max    100.000000   100.000000  100.000000  100.000000   70.000000

           Ind_JS  Ind_Python       Ind_R     Ind_SAS
count  207.000000  207.000000  207.000000  207.000000
mean     6.966184   29.772947   32.434783   18.884058
std     10.030721   17.959816   15.912209   19.137623
min      0.000000    0.000000    0.000000    0.000000
25%      0.000000   20.000000   22.500000    0.000000
50%      0.000000   30.000000   30.000000   15.000000
75%     10.000000   40.000000   40.000000   30.000000
max     50.000000   95.000000   85.000000  100.000000
```

```
In [36]: # single scatter plot example
```

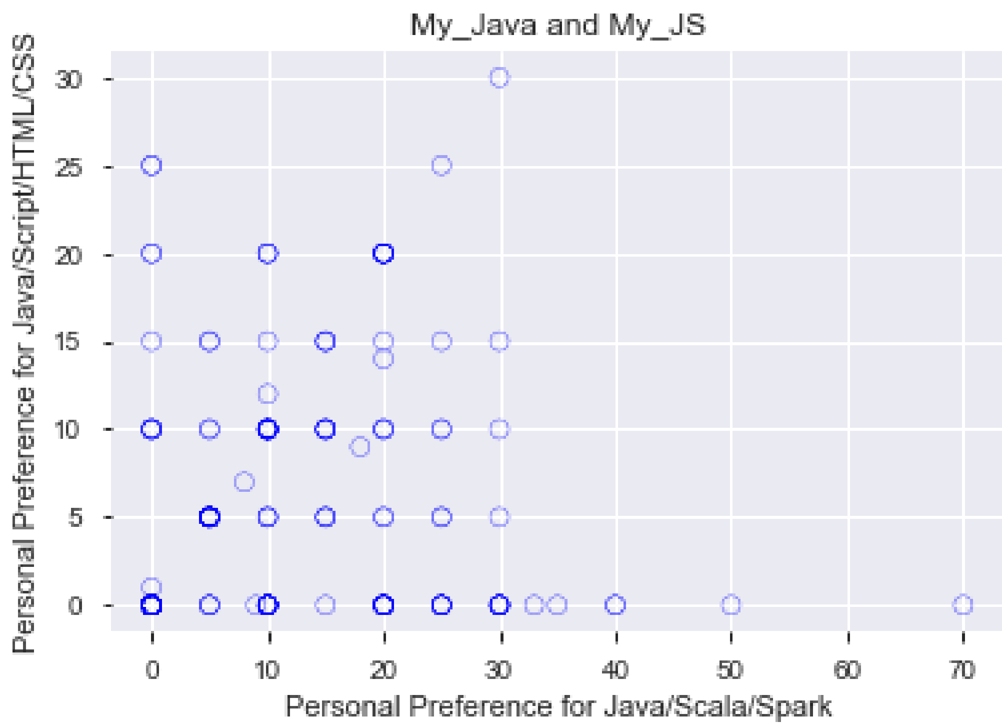
```
fig, axis = plt.subplots()
axis.set_xlabel('Personal Preference for R')
axis.set_ylabel('Personal Preference for Python')
plt.title('R and Python Perferences')
scatter_plot = axis.scatter(survey_df['My_R'],
    survey_df['My_Python'],
    facecolors = 'none',
    edgecolors = 'blue')
plt.savefig('plot-scatter-r-python.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```

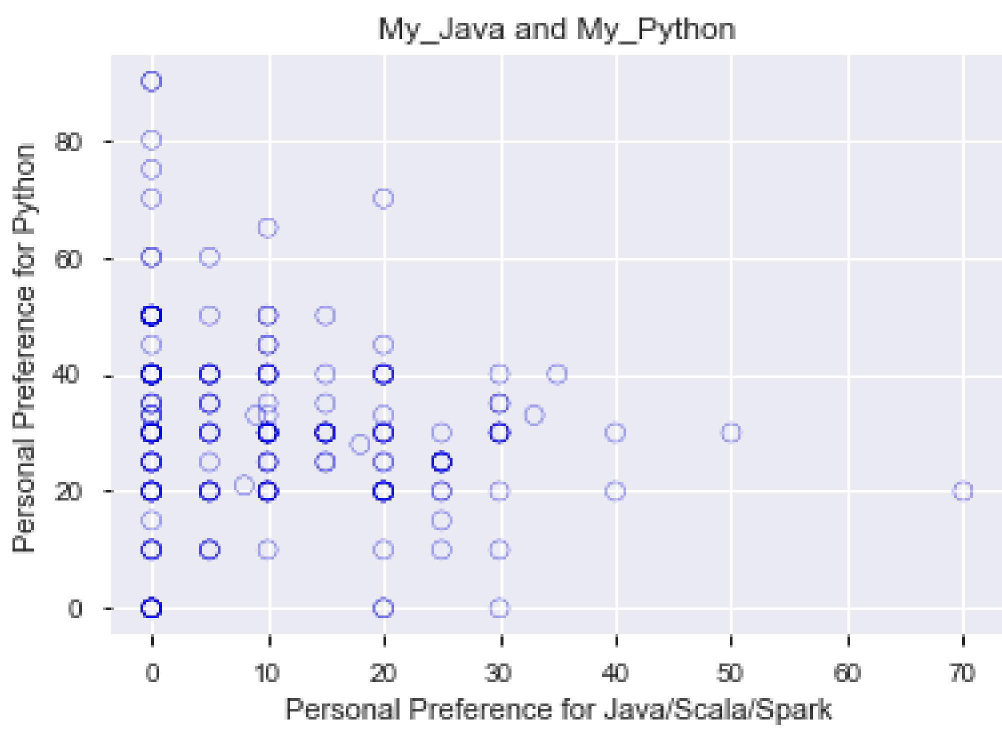


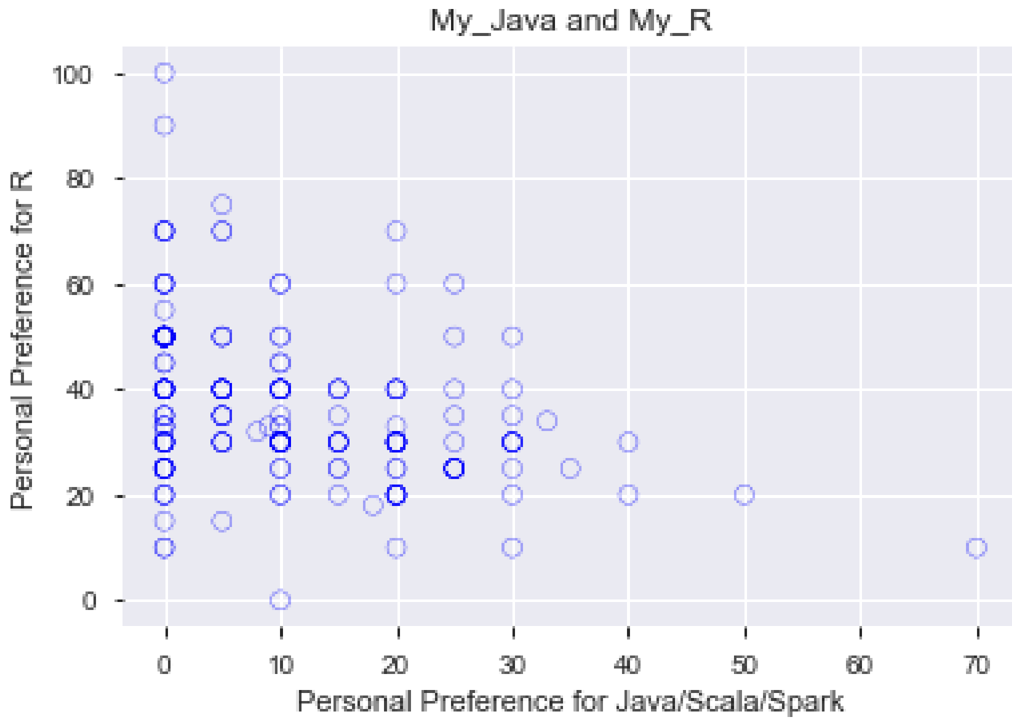
In [37]:
```
# Creating a list of the survey_df labels which will be used
# in a for loop to create graphs to compare responses

survey_df_labels = [
    'Personal Preference for Java/Scala/Spark',
    'Personal Preference for Java/Script/HTML/CSS',
    'Personal Preference for Python',
    'Personal Preference for R',
    'Personal Preference for SAS',
    'Professional Java/Scala/Spark',
    'Professional JavaScript/HTML/CSS',
```

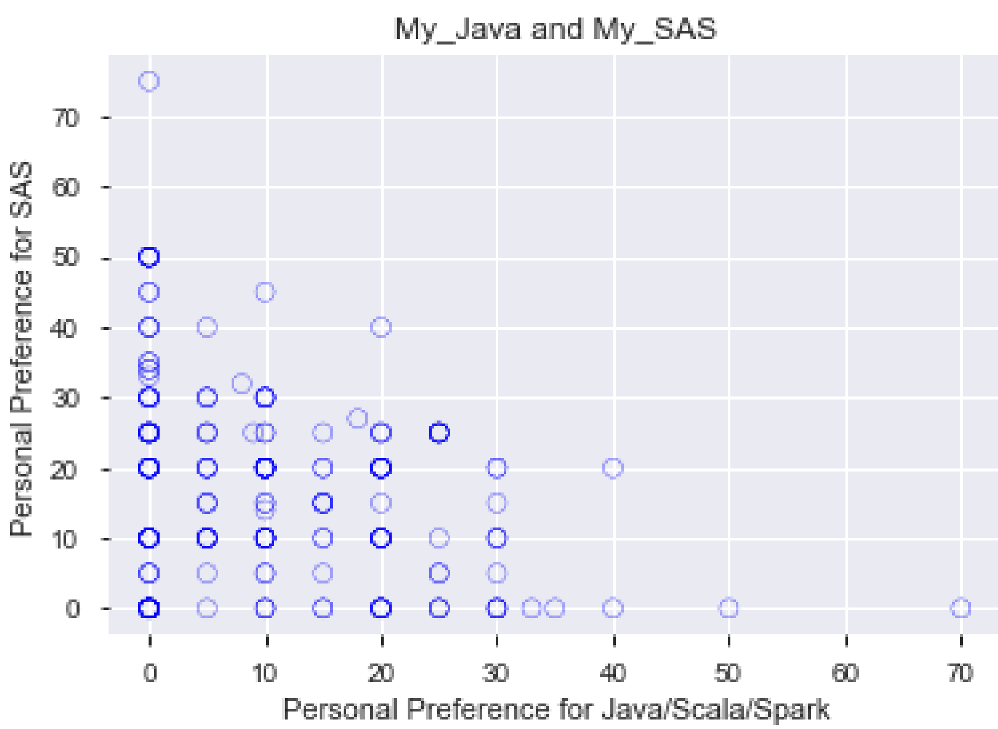
```
'Professional Python',
    'Professional R',
    'Professional SAS',
    'Industry Java/Scala/Spark',
    'Industry Java/Script/HTML/CSS',
    'Industry Python',
    'Industry R',
    'Industry SAS'
]
```

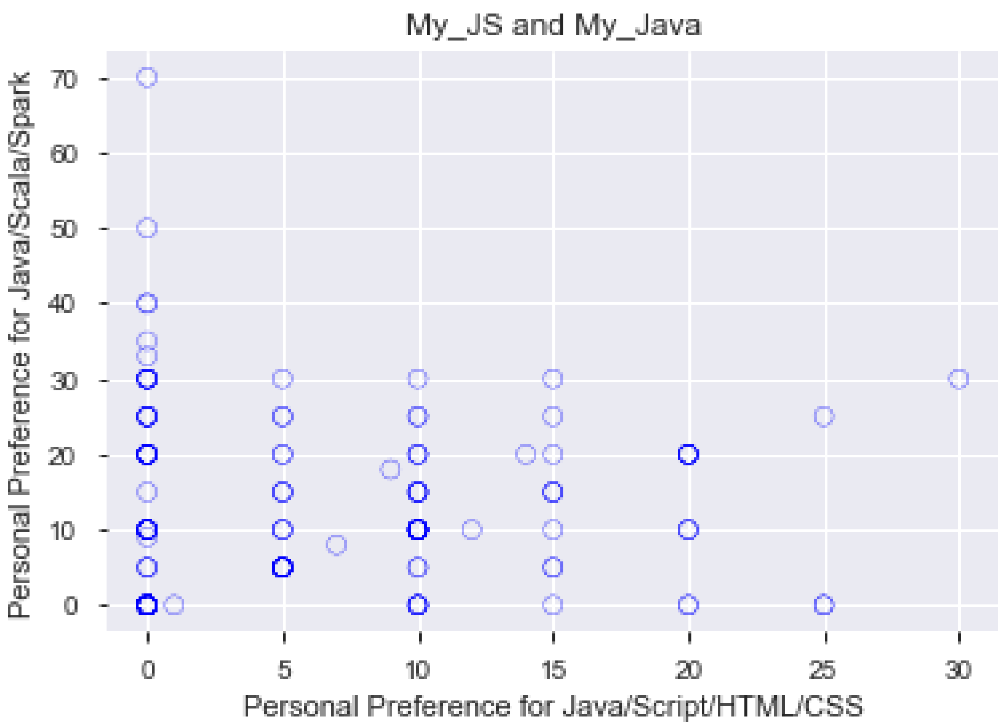
In [38]:
```
# create a set of scatter plots for personal preferences
for i in range(5):
    for j in range(5):
        if i != j:
            file_title = survey_df.columns[i] + '_and_' + survey_df.columns[j]
            plot_title = survey_df.columns[i] + ' and ' + survey_df.columns[j]
            fig, axis = plt.subplots()
            axis.set_xlabel(survey_df_labels[i])
            axis.set_ylabel(survey_df_labels[j])
            plt.title(plot_title)
            scatter_plot = axis.scatter(survey_df[survey_df.columns[i]],
            survey_df[survey_df.columns[j]],
            facecolors = 'none',
            edgecolors = 'blue')
            plt.savefig(file_title + '.pdf',
                bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
                orientation='portrait', papertype=None, format=None,
                transparent=True, pad_inches=0.25, frameon=None)
```

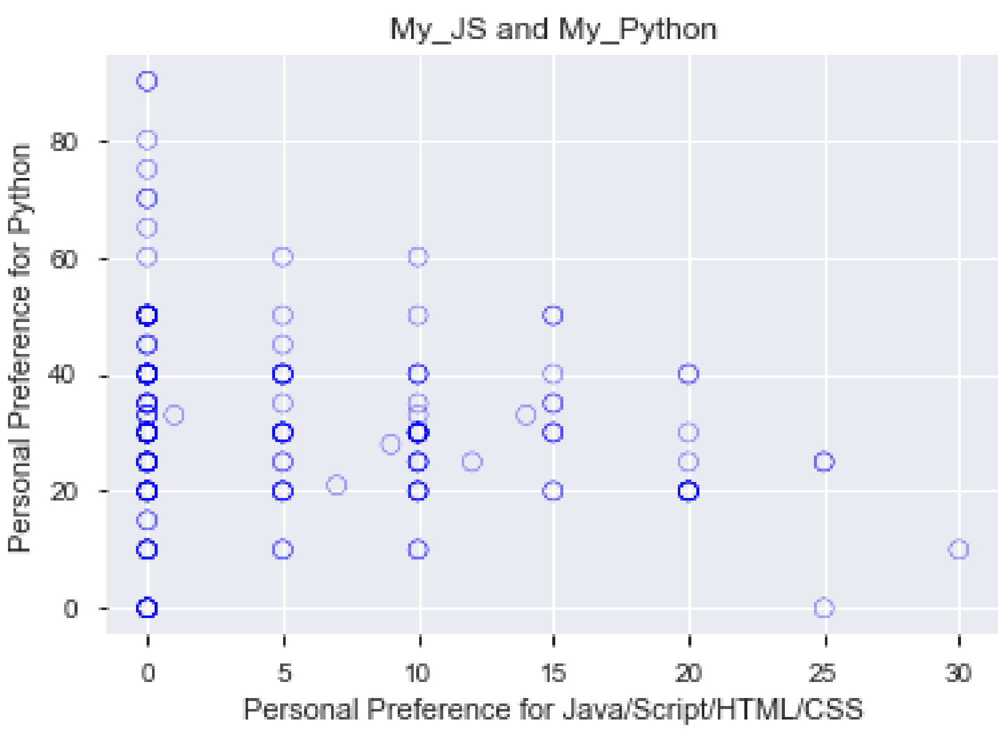

My_Java and My_JS
Personal Preference for Java/Script/HTML/CSS
Personal Preference for Java/Scala/Spark
0
5
10
15
20
25
30
0
10
20
30
40
50
60
70

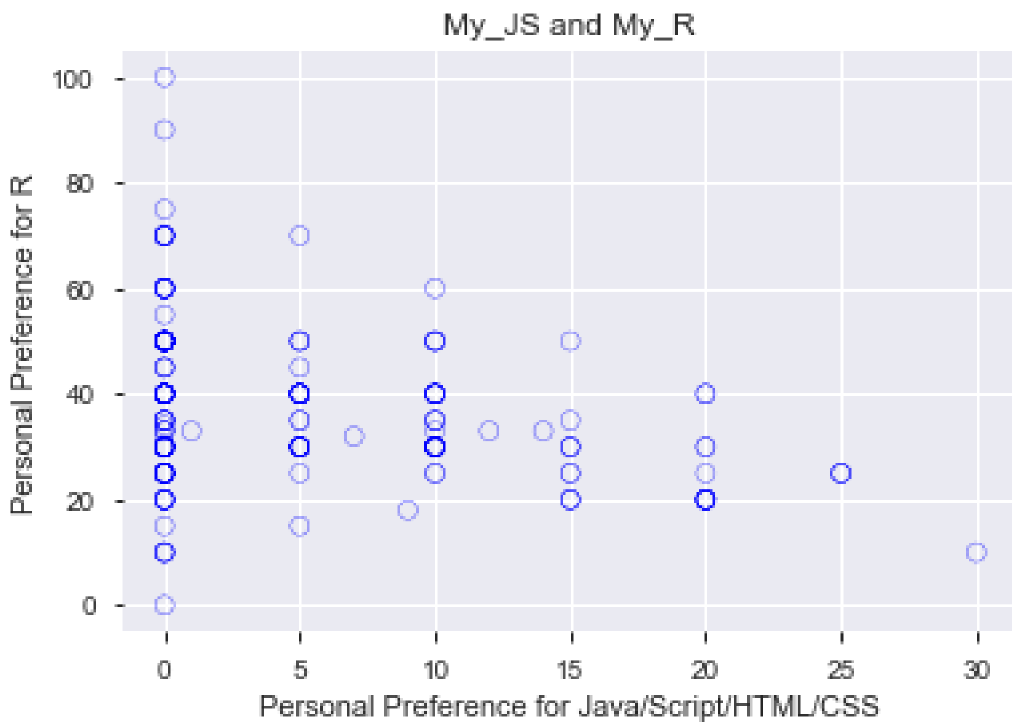

My_Java and My_Python
Personal Preference for Python
Personal Preference for Java/Scala/Spark
0
20
40
60
80
0
10
20
30
40
50
60
70

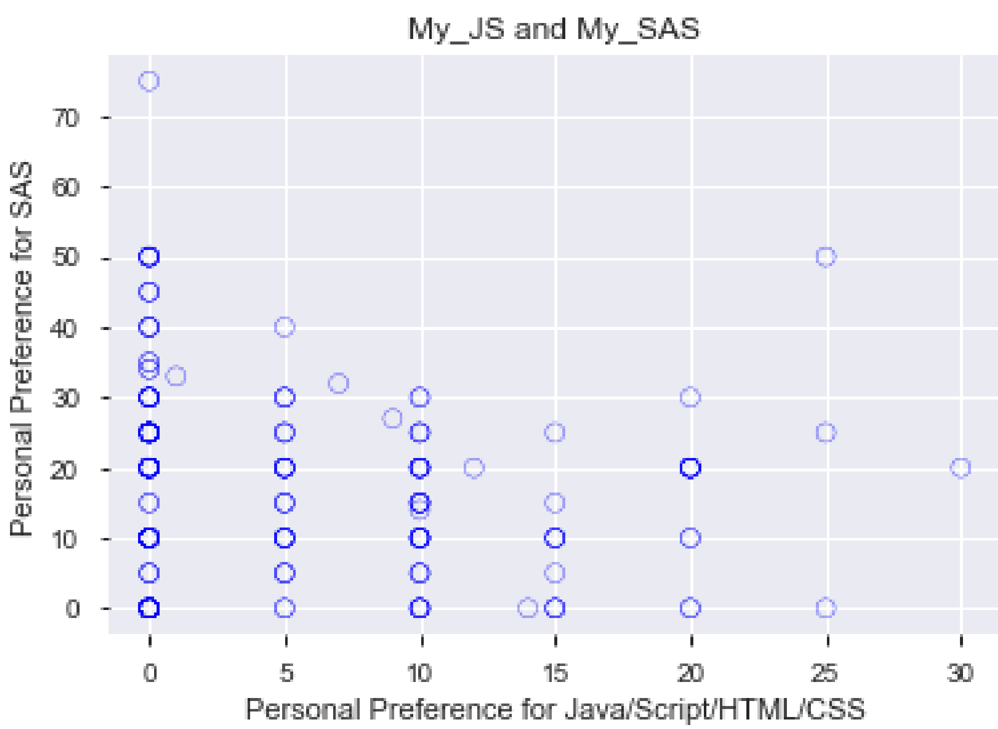

My_Java and My_R
Personal Preference for R
100
80
60
40
20
0
0
10
20
30
40
50
60
70
Personal Preference for Java/Scala/Spark

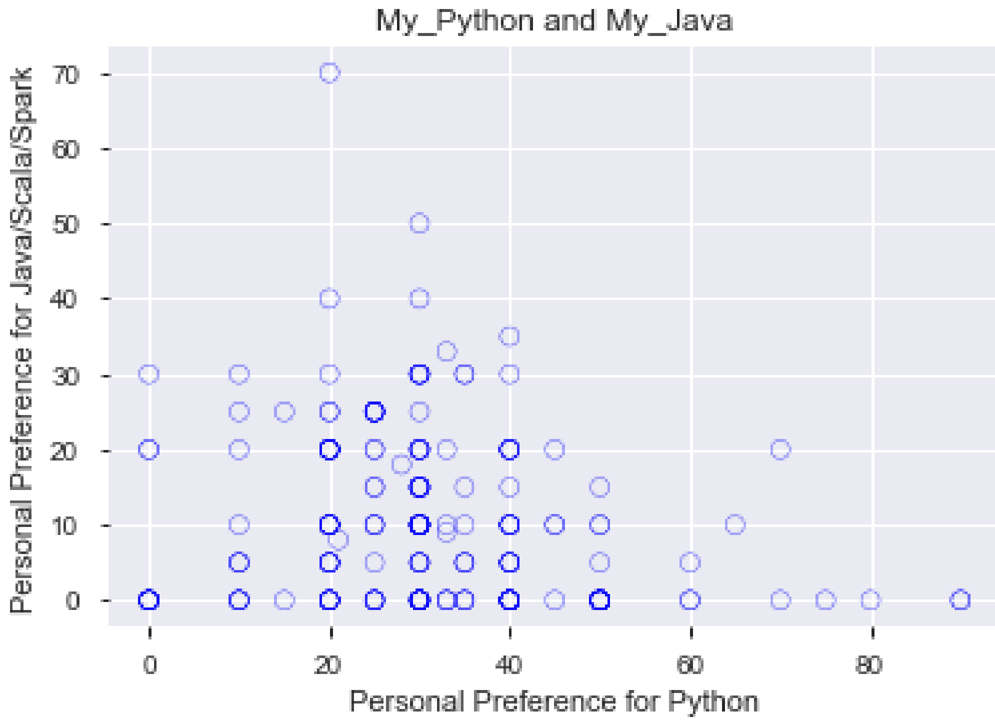

My_Java and My_SAS
Personal Preference for SAS
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
Personal Preference for Java/Scala/Spark


My_JS and My_Java
Personal Preference for Java/Scala/Spark
Personal Preference for Java/Script/HTML/CSS

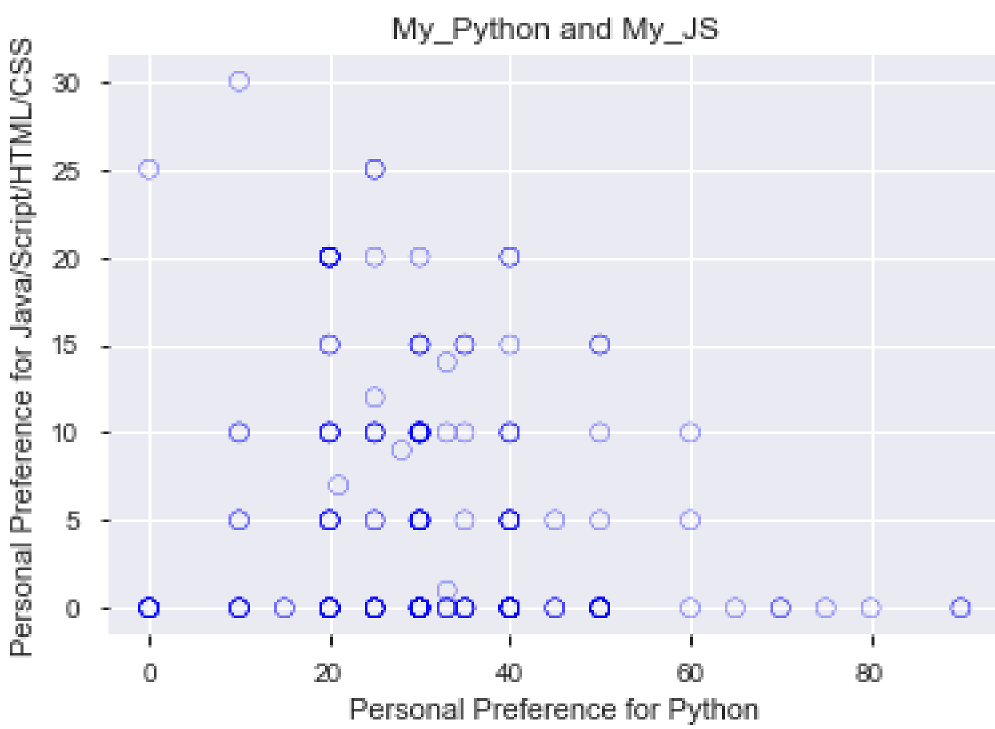

My_JS and My_Python
Personal Preference for Python
Personal Preference for Java/Script/HTML/CSS


My_JS and My_R
Personal Preference for R
100
80
60
40
20
0
0
5
10
15
20
25
30
Personal Preference for Java/Script/HTML/CSS

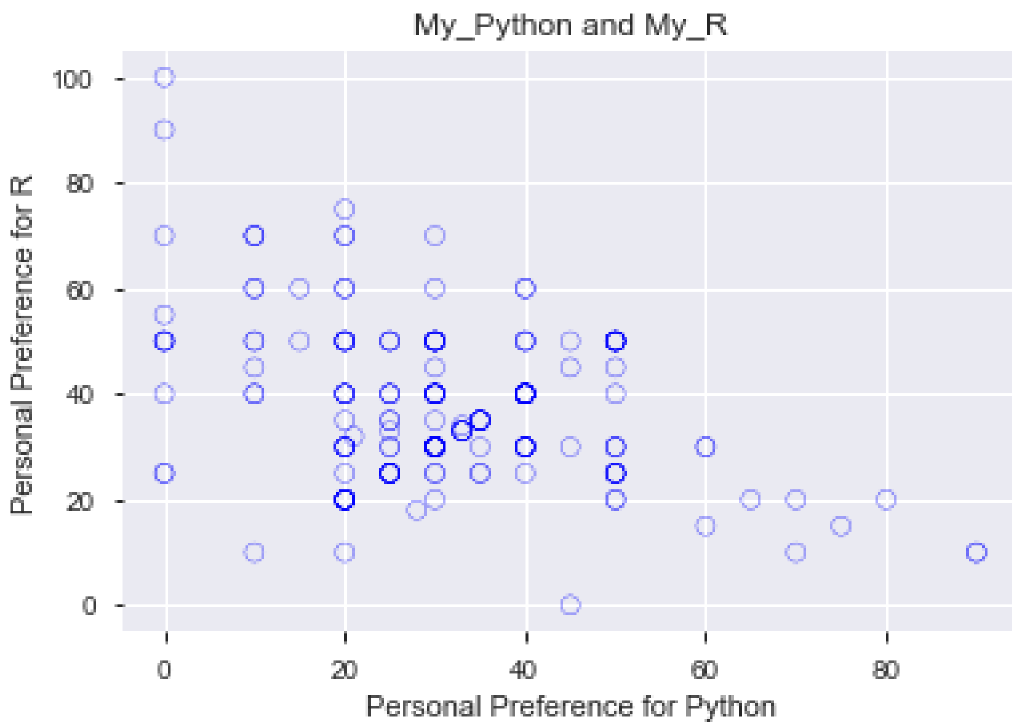

My_JS and My_SAS
Personal Preference for SAS
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
Personal Preference for Java/Script/HTML/CSS

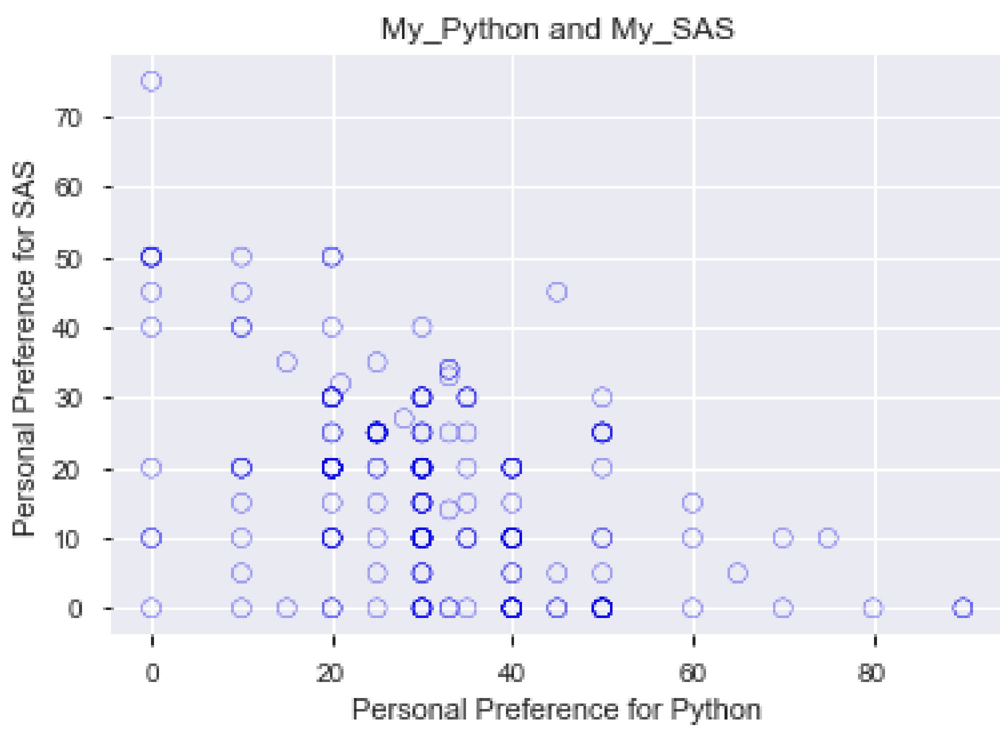


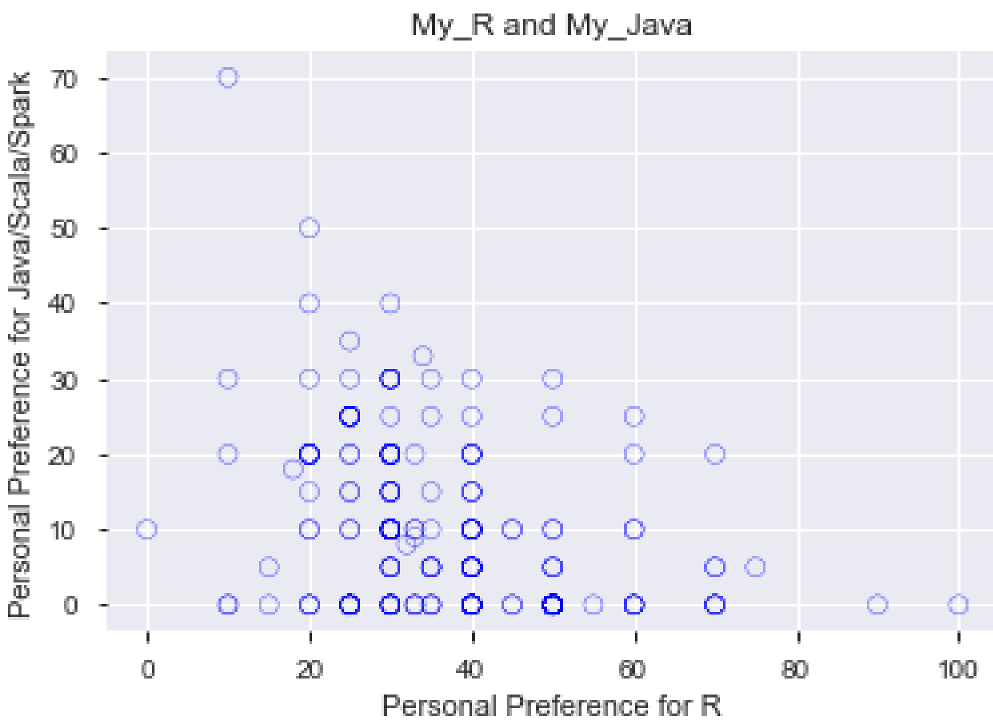
My_Python and My_Java
Personal Preference for Java/Scala/Spark
0
10
20
30
40
50
60
70
0
20
40
60
80
Personal Preference for Python



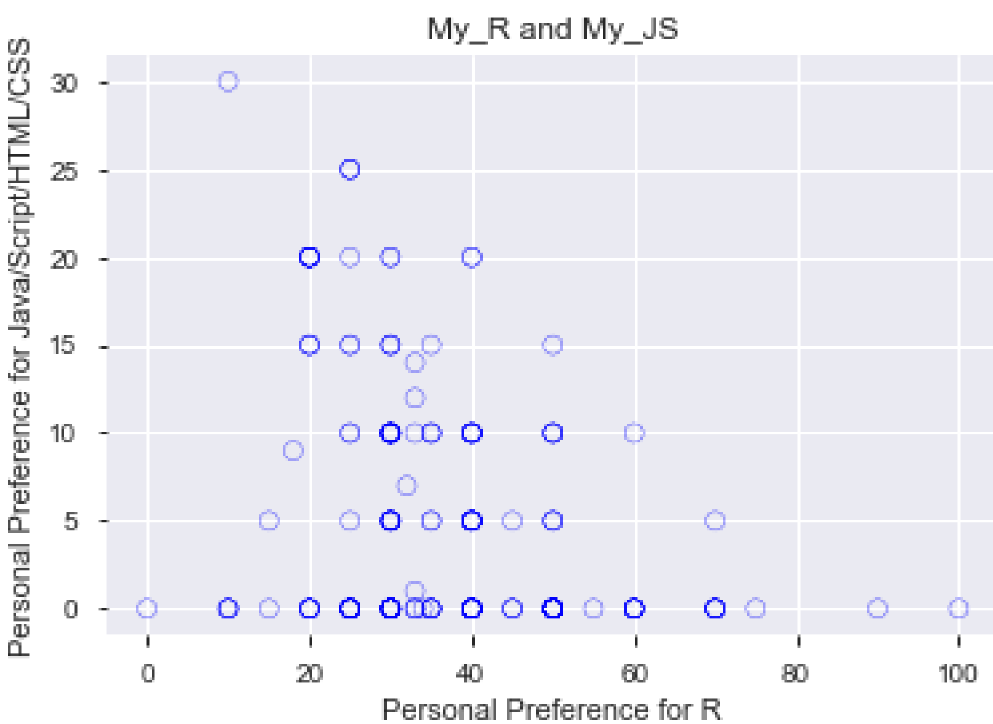
My_Python and My_JS
Personal Preference for Java/Script/HTML/CSS
0
5
10
15
20
25
30
0
20
40
60
80
Personal Preference for Python


My_Python and My_R
Personal Preference for R
Personal Preference for Python
0
20
40
60
80
100

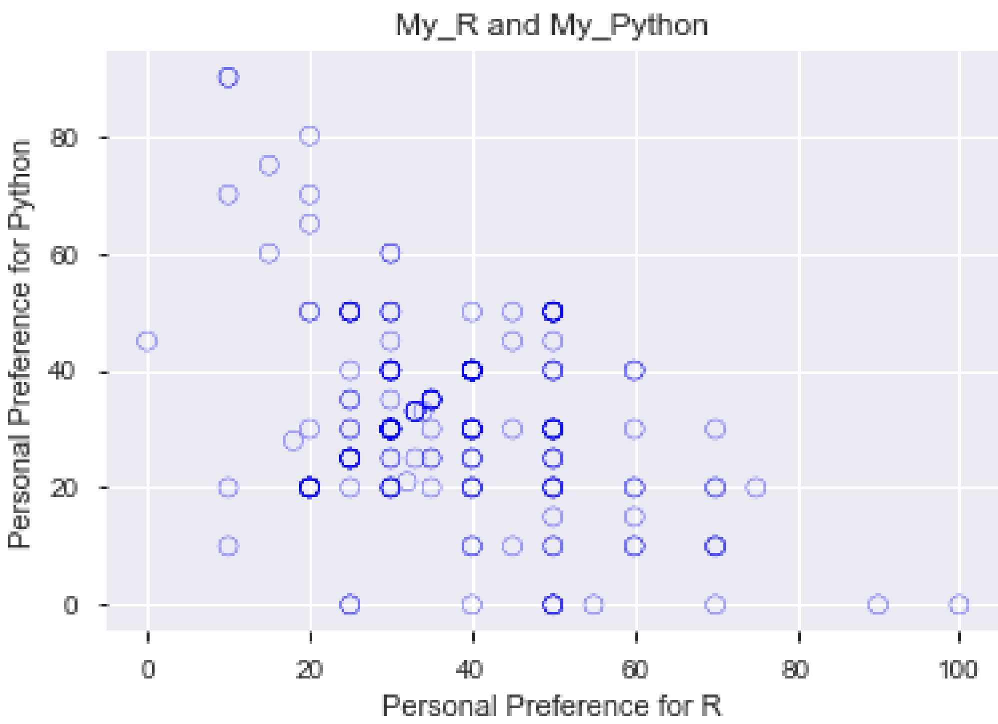

My_Python and My_SAS
Personal Preference for SAS
Personal Preference for Python
0
10
20
30
40
50
60
70
80

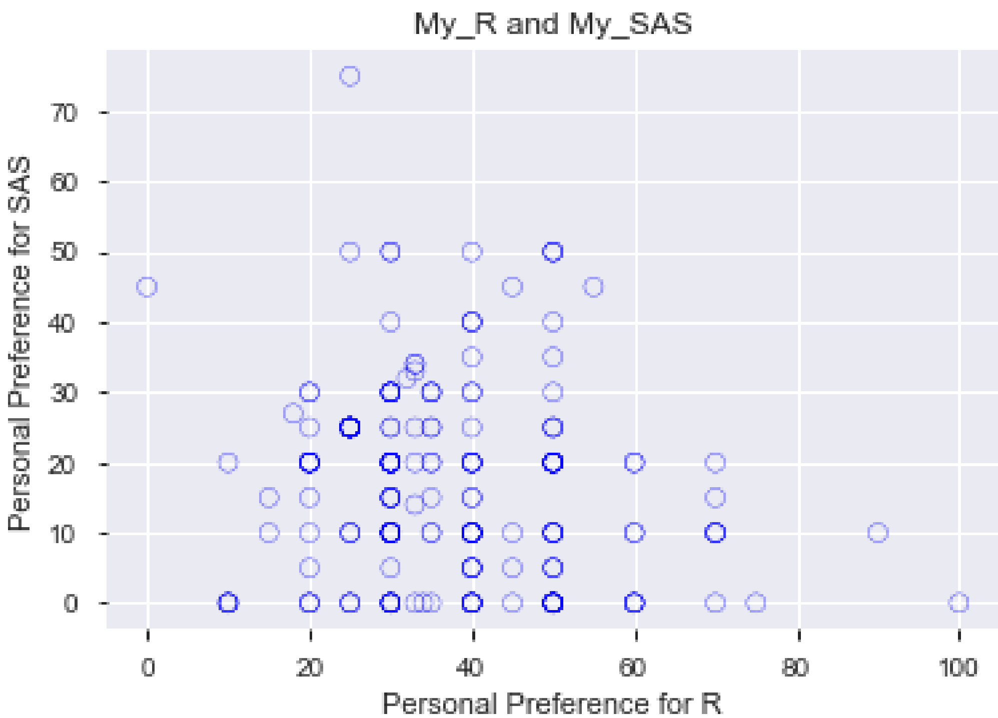


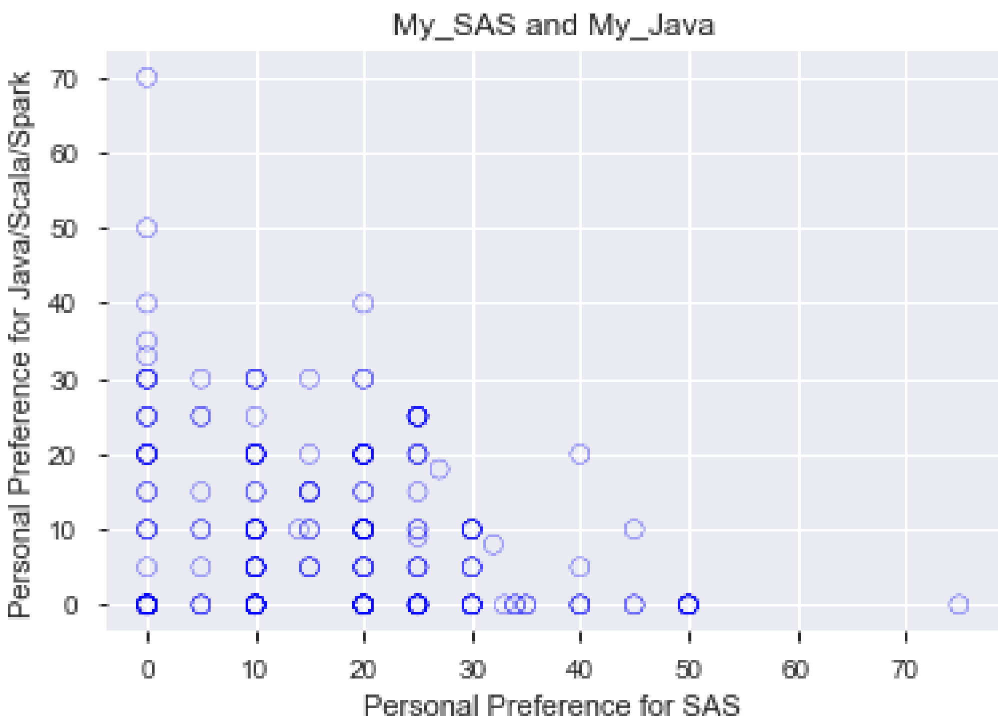
My_R and My_Java
Personal Preference for Java/Scala/Spark
70
60
50
40
30
20
10
0
0
20
40
60
80
100
Personal Preference for R

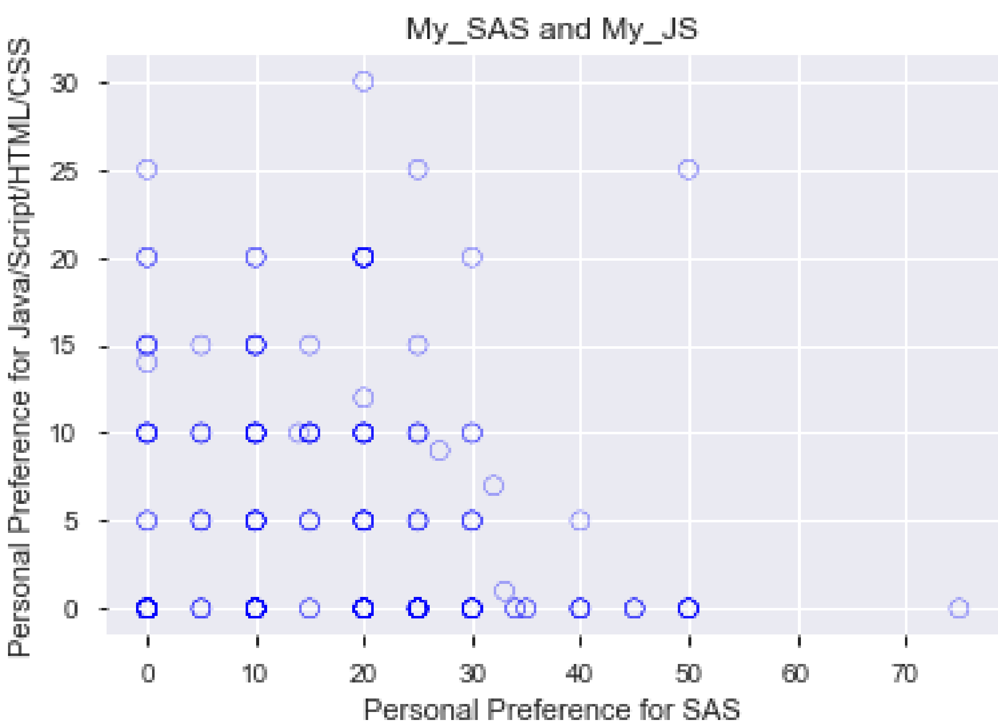


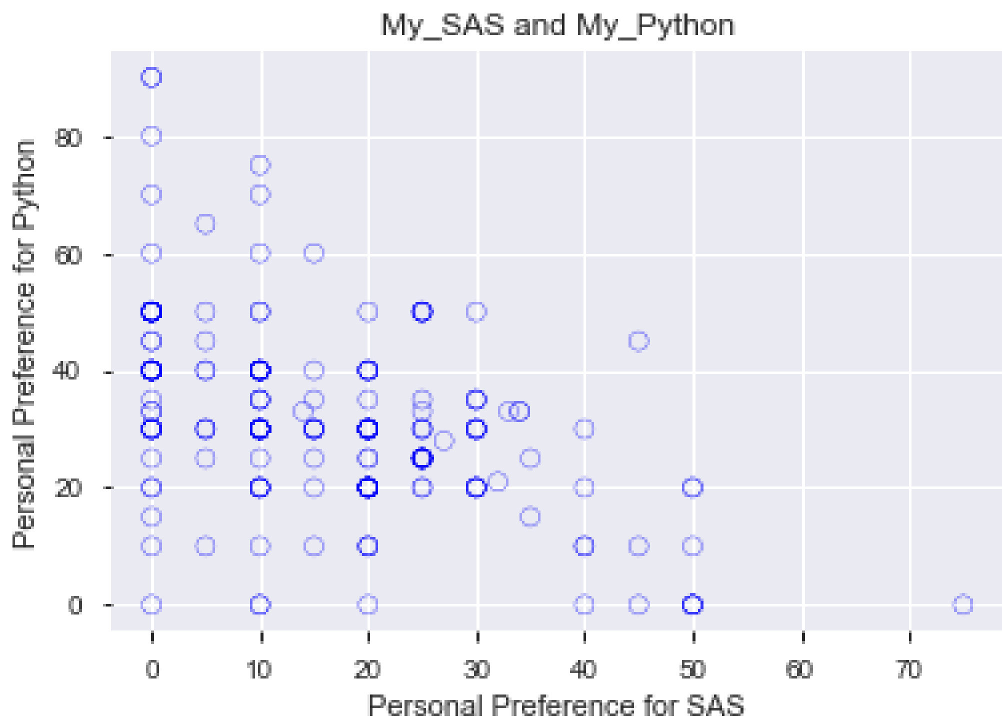
My_R and My_JS
Personal Preference for Java/Script/HTML/CSS
30
25
20
15
10
5
0
0
20
40
60
80
100
Personal Preference for R

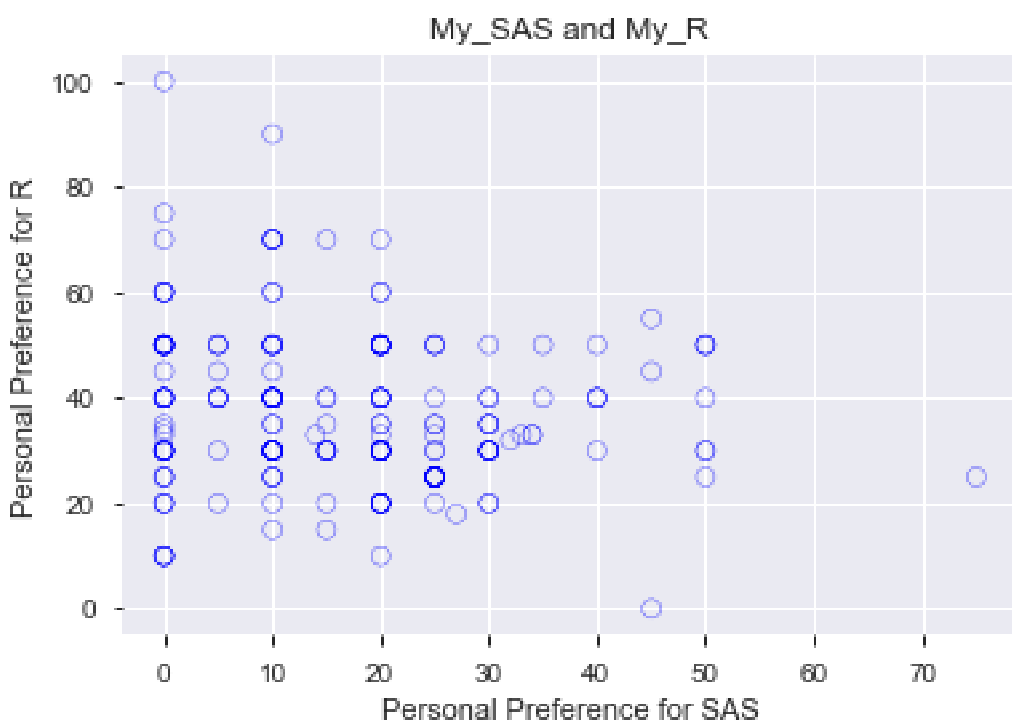

My_R and My_Python
Personal Preference for Python
Personal Preference for R
0
20
40
60
80
100


My_R and My_SAS
Personal Preference for SAS
Personal Preference for R
0
10
20
30
40
50
60
70
80
100


My_SAS and My_Java
Personal Preference for Java/Scala/Spark
0
10
20
30
40
50
60
70
Personal Preference for SAS
0
10
20
30
40
50
60
70

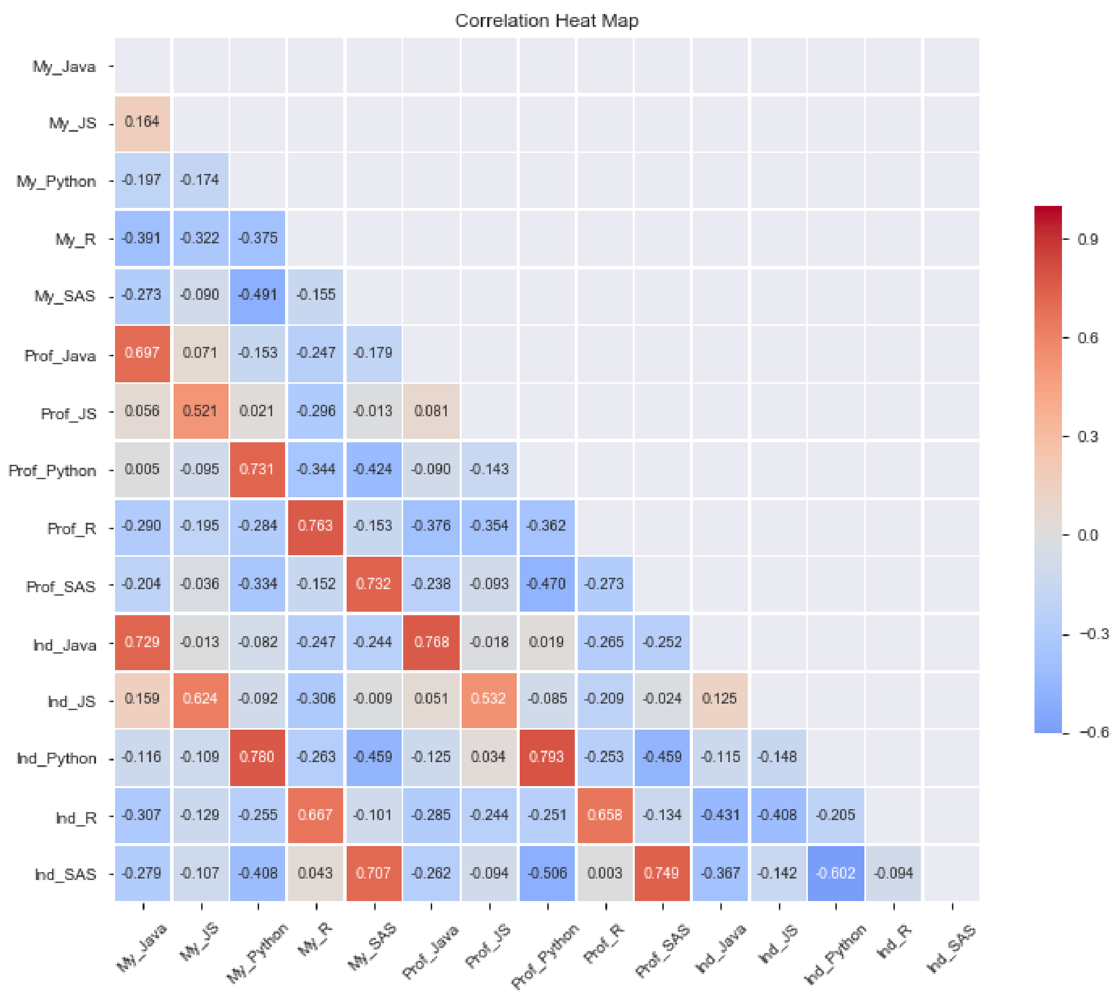

My_SAS and My_JS
Personal Preference for Java/Script/HTML/CSS
0
5
10
15
20
25
30
Personal Preference for SAS
0
10
20
30
40
50
60
70


My_SAS and My_Python
Personal Preference for Python
0
20
40
60
80
0
10
20
30
40
50
60
70
Personal Preference for SAS


My_SAS and My_R
Personal Preference for R
0
20
40
60
80
100
0
10
20
30
40
50
60
70
Personal Preference for SAS

```
In [39]: # examine intercorrelations among software preference variables
         # with correlation matrix/heat map
         corr_chart(df_corr = software_df)
```

<Figure size 432x288 with 0 Axes>




```
In [40]: corr_matrix = software_df.corr()
         corr_matrix["My_Java"].sort_values(ascending=False)
```

```
Out[40]: My_Java        1.000000
         Ind_Java       0.728855
         Prof_Java      0.697135
         My_JS          0.164302
         Ind_JS         0.158573
         Prof_JS        0.056456
         Prof_Python    0.004771
```

```
Ind_Python    -0.115839
My_Python     -0.197282
Prof_SAS      -0.203620
My_SAS        -0.273014
Ind_SAS       -0.278953
Prof_R        -0.290046
Ind_R         -0.307342
My_R          -0.391172
Name: My_Java, dtype: float64
```

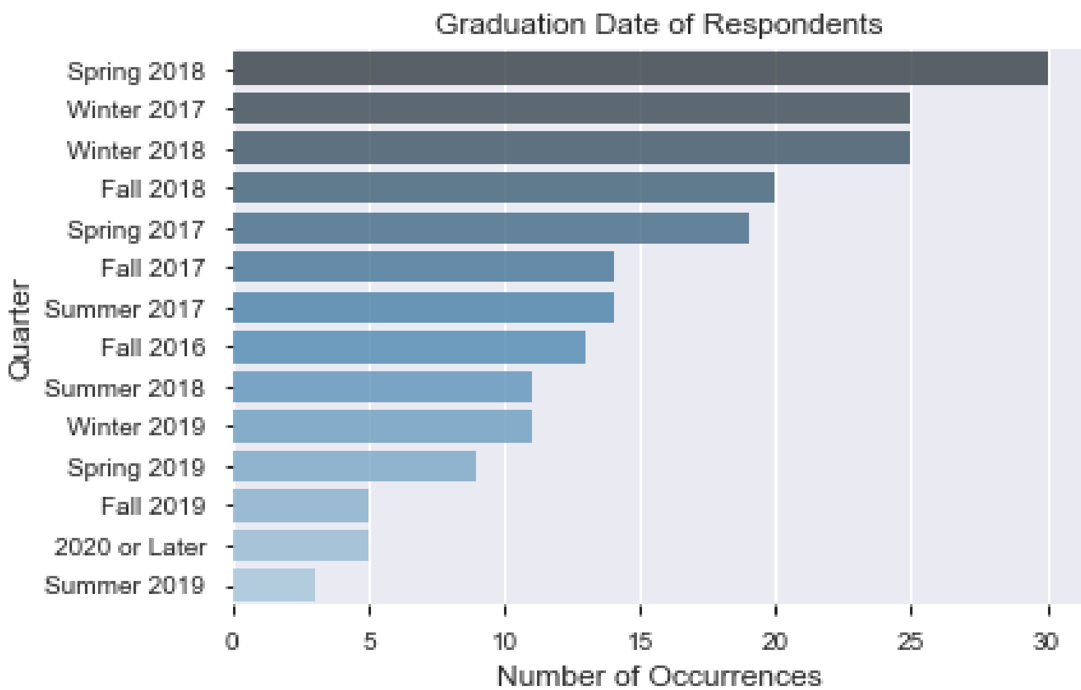
0.5 Analysis of New Course Interest and Graduation Date

In [41]:
```
# A new dataframe will be created to examine the relationship between
# course interest and expected graduation date.

# Select the graduation date column
example = survey_df.Graduate_Date

# Select the columns for course interest
example_2 = survey_df.iloc[:,15:19]

# Combine the two into a new dataframe and show the first five rows
interest_grad_date_df = pd.concat([example_2, example], axis=1, sort=False)
print(interest_grad_date_df.head())
```

```
              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

             Graduate_Date
RespondentID
5135740122             NaN
5133300037     Spring 2018
5132253300       Fall 2018
5132096630       Fall 2017
5131990362       Fall 2018
```

```
In [42]: # Examing the dataframe by grouping by graduation date
         # and checking that this works.

         interest_grad_date_df.groupby(['Graduate_Date']).groups.keys()
         # Out: dict_keys(['2020 or Later', 'Fall 2016', 'Fall 2017', 'Fall 2018',
         # 'Fall 2019', 'Spring 2017', 'Spring 2018', 'Spring 2019', 'Summer 2017', 'Summer 20
         # 'Summer 2019', 'Winter 2017', 'Winter 2018', 'Winter 2019'])

Out[42]: dict_keys(['2020 or Later', 'Fall 2016', 'Fall 2017', 'Fall 2018', 'Fall 2019', 'Sprin

In [43]: # Examing the dataframe by grouping by graduation date
         # and checking that this works.

         len(interest_grad_date_df.groupby(['Graduate_Date']).groups['Fall 2018'])
         #Out: 20

Out[43]: 20

In [44]: # Examine the number of respondents by graduation date

         graduate_date_counts = survey_df.Graduate_Date.value_counts()
         print('\nCount of students by graduation date: ')
         graduate_date_counts


Count of students by graduation date:


Out[44]: Spring 2018      30
         Winter 2017      25
         Winter 2018      25
         Fall 2018        20
         Spring 2017      19
         Fall 2017        14
         Summer 2017      14
         Fall 2016        13
         Summer 2018      11
         Winter 2019      11
         Spring 2019       9
         Fall 2019         5
         2020 or Later     5
         Summer 2019       3
         Name: Graduate_Date, dtype: int64

In [45]: # Examine the number of respondents by graduation date in bar chart

         graduation_date_fig, ax = plt.subplots()
```

```
sns.barplot(y = graduate_date_counts.index,
            x = graduate_date_counts.values, alpha=0.8,
            palette="Blues_d").set_title('Graduation Date of Respondents')
ax.set_xlabel('Number of Occurrences', fontsize=12)
ax.set_ylabel('Quarter', fontsize=12)
graduation_date_fig.savefig('GraduationDate' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```



In [46]:
```
# Examing the dataframe by grouping by graduation date.
# The count (number of respondents), min, max, and mean
# for each course interest question was evaluated.

interest_grad_date_df.groupby(['Graduate_Date']).agg(
    {# find the count min, max, and mean of each course interest column
        'Python_Course_Interest': ['count', min, max, 'mean'],
        'Foundations_DE_Course_Interest': ['count', min, max, 'mean'],
        'Analytics_App_Course_Interest': ['count', min, max, 'mean'],
        'Systems_Analysis_Course_Interest': ['count', min, max, 'mean']})
```

Out[46]:
```
              Python_Course_Interest                              \
                               count   min    max       mean
Graduate_Date
2020 or Later                      5  35.0  100.0  82.400000
```

	count	min	max	mean
Fall 2016	13	1.0	100.0	76.923077
Fall 2017	14	30.0	100.0	85.000000
Fall 2018	20	15.0	100.0	76.800000
Fall 2019	5	83.0	100.0	91.800000
Spring 2017	18	12.0	100.0	70.166667
Spring 2018	30	0.0	100.0	70.200000
Spring 2019	9	30.0	100.0	77.000000
Summer 2017	14	0.0	100.0	74.642857
Summer 2018	11	0.0	100.0	73.363636
Summer 2019	3	100.0	100.0	100.000000
Winter 2017	25	0.0	100.0	63.520000
Winter 2018	25	5.0	100.0	68.600000
Winter 2019	11	40.0	100.0	73.909091

	Foundations_DE_Course_Interest				\
	count	min	max	mean	
Graduate_Date					
2020 or Later	4	25.0	95.0	61.250000	
Fall 2016	12	0.0	100.0	70.250000	
Fall 2017	14	0.0	100.0	49.142857	
Fall 2018	19	10.0	100.0	58.894737	
Fall 2019	5	51.0	100.0	80.600000	
Spring 2017	18	18.0	100.0	70.444444	
Spring 2018	30	0.0	100.0	55.200000	
Spring 2019	9	25.0	100.0	62.666667	
Summer 2017	14	0.0	100.0	57.500000	
Summer 2018	9	0.0	100.0	53.666667	
Summer 2019	3	64.0	100.0	88.000000	
Winter 2017	25	0.0	100.0	49.920000	
Winter 2018	24	0.0	100.0	50.833333	
Winter 2019	11	0.0	100.0	56.636364	

	Analytics_App_Course_Interest				\
	count	min	max	mean	
Graduate_Date					
2020 or Later	5	50.0	90.0	72.200000	
Fall 2016	13	0.0	100.0	52.307692	
Fall 2017	14	0.0	100.0	60.714286	
Fall 2018	19	10.0	100.0	58.210526	
Fall 2019	5	20.0	100.0	60.200000	
Spring 2017	17	20.0	100.0	59.058824	
Spring 2018	30	0.0	100.0	49.133333	
Spring 2019	9	30.0	100.0	66.666667	
Summer 2017	14	0.0	100.0	59.214286	
Summer 2018	10	0.0	100.0	50.500000	
Summer 2019	3	100.0	100.0	100.000000	
Winter 2017	25	0.0	100.0	43.120000	
Winter 2018	25	0.0	100.0	61.800000	

```
Winter 2019                           11    0.0  100.0   43.818182

                 Systems_Analysis_Course_Interest
                                            count   min    max       mean
Graduate_Date
2020 or Later                                   5  10.0   70.0  32.000000
Fall 2016                                      13   3.0  100.0  59.307692
Fall 2017                                      14   0.0  100.0  61.785714
Fall 2018                                      18   0.0  100.0  56.611111
Fall 2019                                       5  10.0  100.0  55.400000
Spring 2017                                    18   4.0  100.0  55.111111
Spring 2018                                    29   0.0  100.0  48.689655
Spring 2019                                     9  30.0  100.0  77.333333
Summer 2017                                    14   0.0  100.0  52.857143
Summer 2018                                     9   0.0  100.0  45.555556
Summer 2019                                     3  90.0  100.0  96.666667
Winter 2017                                    25   0.0  100.0  40.400000
Winter 2018                                    25   0.0  100.0  54.280000
Winter 2019                                    10   0.0  100.0  55.700000
```

In [47]:
```
# Examing the dataframe by grouping by graduation date.
# The mean for each course interest question was evaluated.
# Checking to see if interest changes with graduation date.

interest_grad_date_df.groupby(['Graduate_Date']).agg(
    {# find the count min, max, and mean of each course interest column
        'Python_Course_Interest': ['mean'],
        'Foundations_DE_Course_Interest': ['mean'],
        'Analytics_App_Course_Interest': ['mean'],
        'Systems_Analysis_Course_Interest': ['mean']})
```

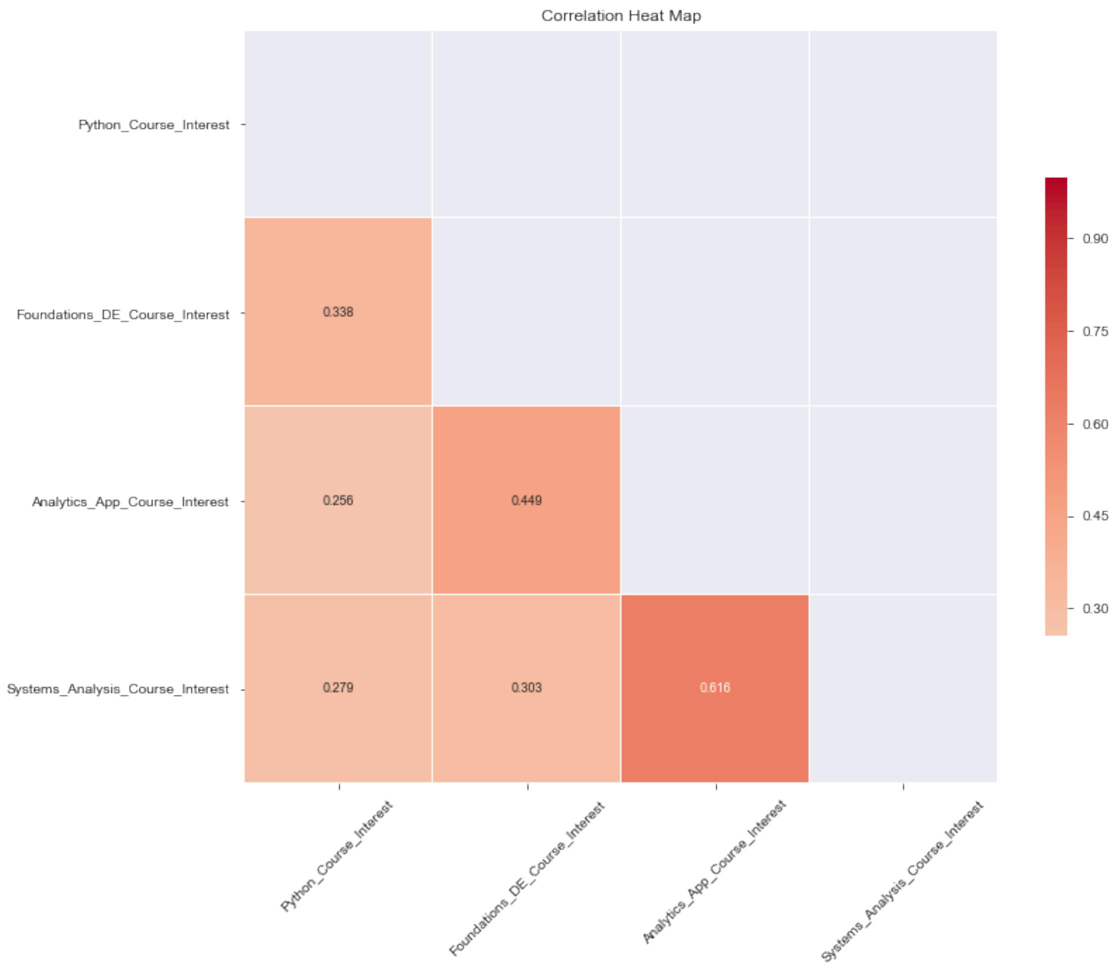
Out[47]:
```
                 Python_Course_Interest Foundations_DE_Course_Interest  \
                                   mean                           mean
Graduate_Date
2020 or Later                 82.400000                      61.250000
Fall 2016                     76.923077                      70.250000
Fall 2017                     85.000000                      49.142857
Fall 2018                     76.800000                      58.894737
Fall 2019                     91.800000                      80.600000
Spring 2017                   70.166667                      70.444444
Spring 2018                   70.200000                      55.200000
Spring 2019                   77.000000                      62.666667
Summer 2017                   74.642857                      57.500000
Summer 2018                   73.363636                      53.666667
Summer 2019                  100.000000                      88.000000
Winter 2017                   63.520000                      49.920000
Winter 2018                   68.600000                      50.833333
Winter 2019                   73.909091                      56.636364
```

```
                 Analytics_App_Course_Interest Systems_Analysis_Course_Interest
                                          mean                             mean
Graduate_Date
2020 or Later                        72.200000                        32.000000
Fall 2016                            52.307692                        59.307692
Fall 2017                            60.714286                        61.785714
Fall 2018                            58.210526                        56.611111
Fall 2019                            60.200000                        55.400000
Spring 2017                          59.058824                        55.111111
Spring 2018                          49.133333                        48.689655
Spring 2019                          66.666667                        77.333333
Summer 2017                          59.214286                        52.857143
Summer 2018                          50.500000                        45.555556
Summer 2019                         100.000000                        96.666667
Winter 2017                          43.120000                        40.400000
Winter 2018                          61.800000                        54.280000
Winter 2019                          43.818182                        55.700000
```

0.6 Analysis of Course Interest

In [48]:
```
# Is there any correlation between interest in classes
corr_chart(df_corr = interest_grad_date_df)
```

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



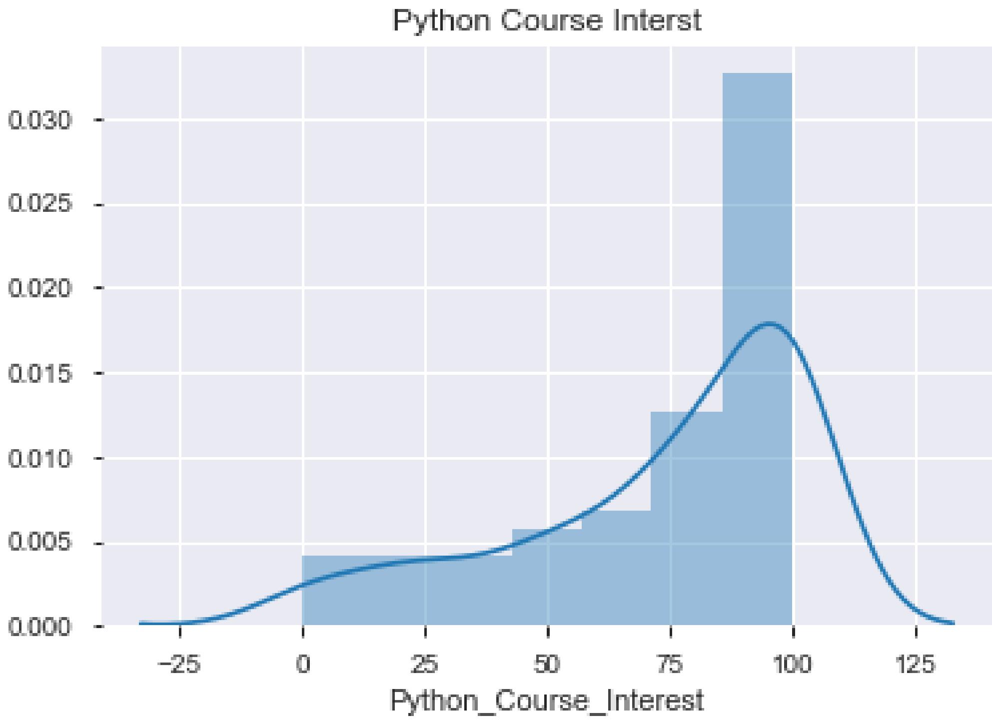
In [49]:
```
# Looking at course interest without graduation date using the describe function

what_df = survey_df.iloc[:,15:19]
what_df.describe()
```

Out[49]:
```
       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 [50]:
```
# Examining the python course interest distribution
python_course_interst_fig, ax = plt.subplots()
sns.distplot(what_df.Python_Course_Interest.dropna()).set_title('Python Course Interst
python_course_interst_fig.savefig('PythonCourseInterest' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [51]:
```
# Examining the course interest between the four courses via pairplot

course_interest_pairplot_fig = sns.pairplot(what_df.dropna())
course_interest_pairplot_fig.savefig('CourseInterestPairplot' + '.pdf',
```

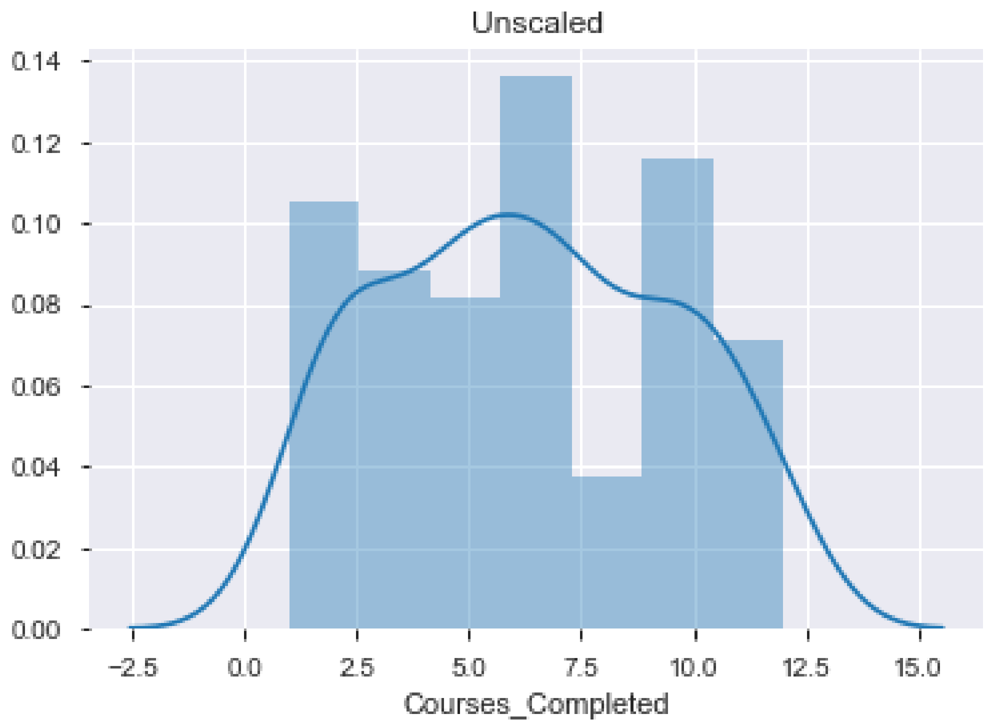
```
bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
orientation='portrait', papertype=None, format=None,
transparent=True, pad_inches=0.25, frameon=None)
```

0.7 Transformations

In [52]:
```
# ------------------------------------------------------------
# transformation code added with version v005
# ------------------------------------------------------------
# transformations a la Scikit Learn
# documentation at http://scikit-learn.org/stable/auto_examples/
#                  preprocessing/plot_all_scaling.html#sphx-glr-auto-
#                  examples-preprocessing-plot-all-scaling-py
```

```python
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

# transformations a la Scikit Learn
# select variable to examine, eliminating missing data codes
X = survey_df['Courses_Completed'].dropna()

# Seaborn provides a convenient way to show the effects of transformations
# on the distribution of values being transformed
# Documentation at https://seaborn.pydata.org/generated/seaborn.distplot.html

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)
```

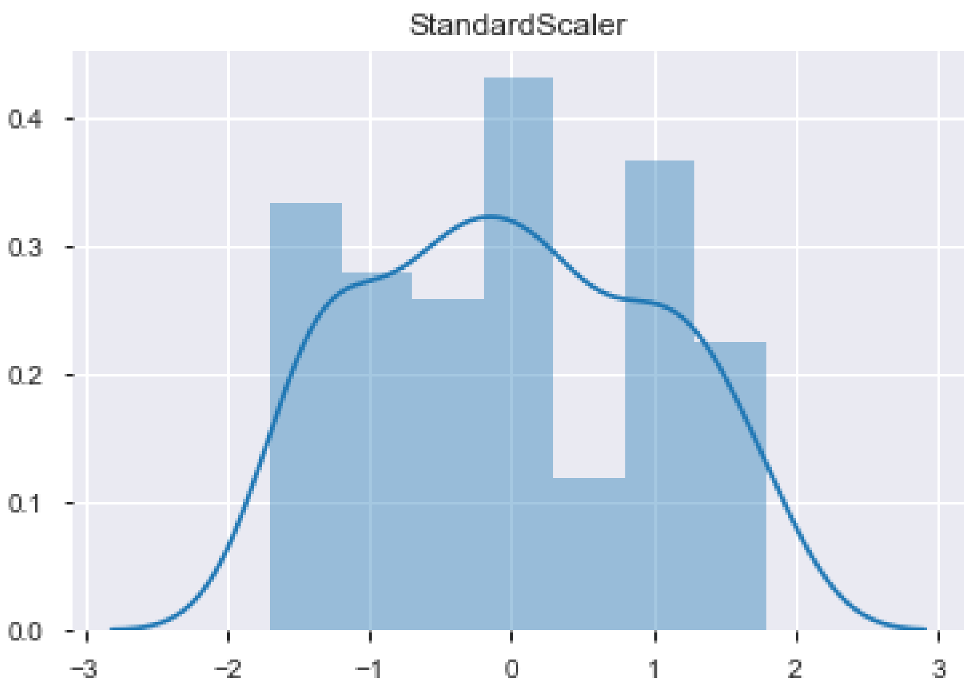
```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [53]:
```python
# Reshape your data either using array.reshape(-1, 1) if your data has a single featu
# or array.reshape(1, -1) if it contains a single sample.
```

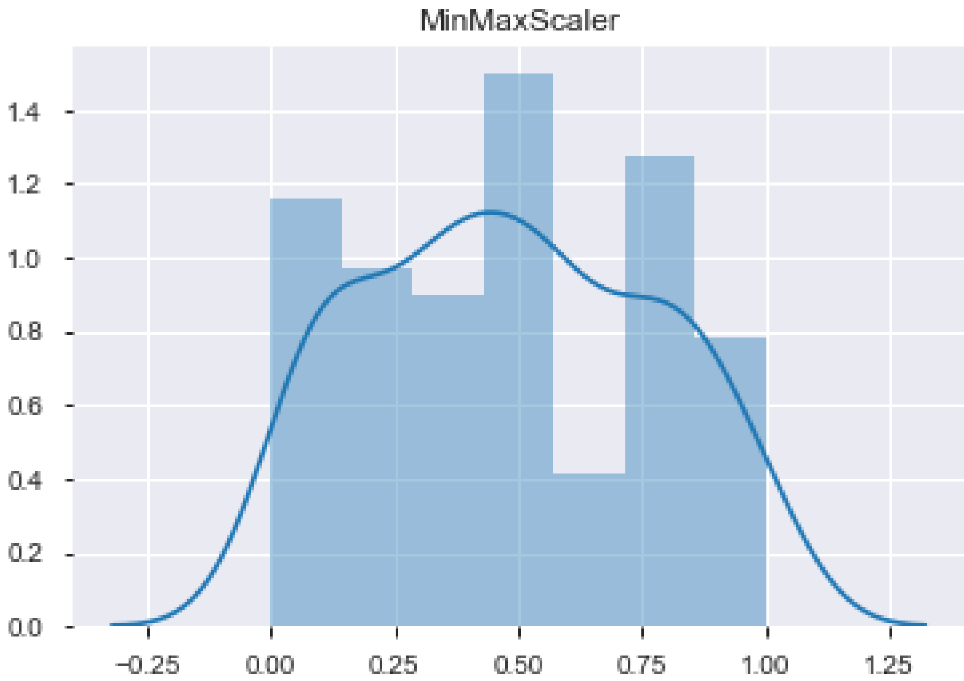
```
X = X.values.reshape(-1, 1)

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)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



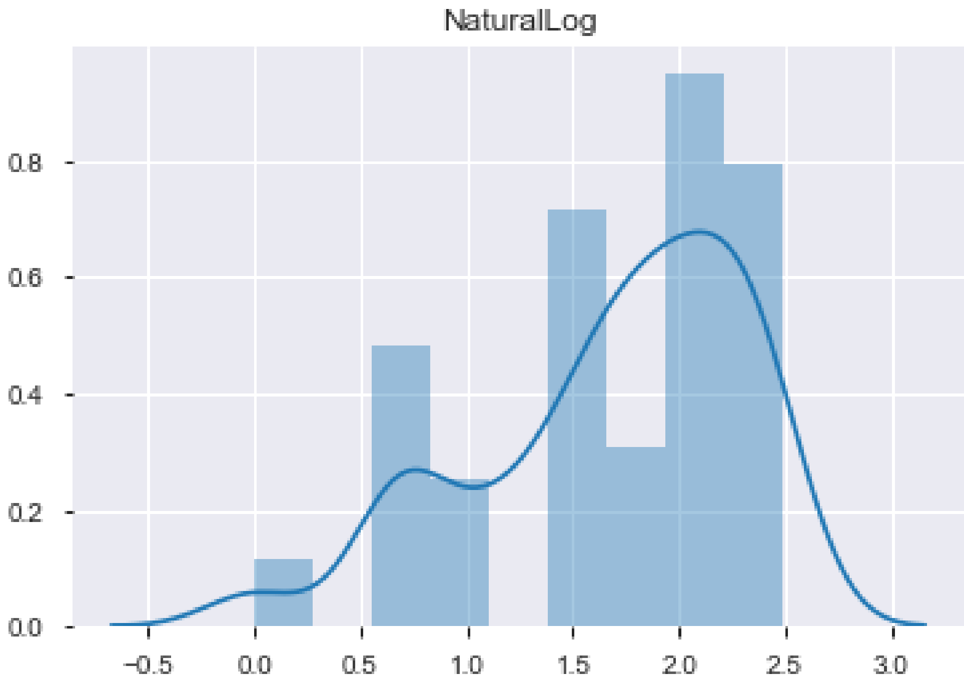
In [54]:
```
minmax_fig, ax = plt.subplots()
sns.distplot(MinMaxScaler().fit_transform(X)).set_title('MinMaxScaler')
minmax_fig.savefig('Transformation-MinMaxScaler' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
    transparent=True, pad_inches=0.25, frameon=None)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```




```
In [55]: log_fig, ax = plt.subplots()
         sns.distplot(np.log(X)).set_title('NaturalLog')
         log_fig.savefig('Transformation-NaturalLog' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```

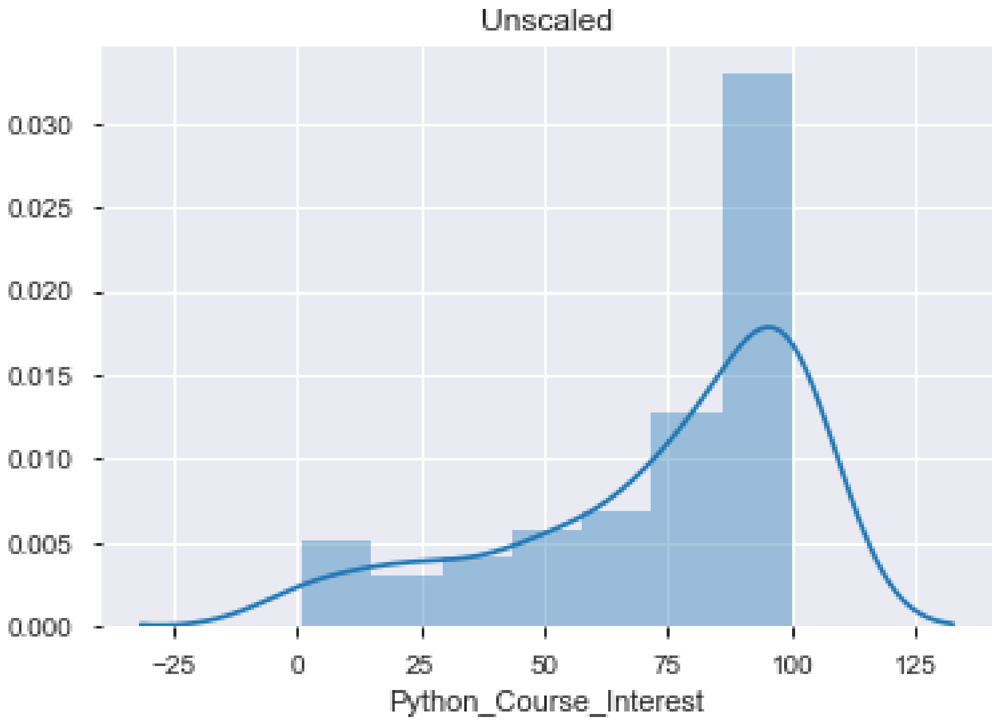
```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



0.8 Transformation of Selected Variable

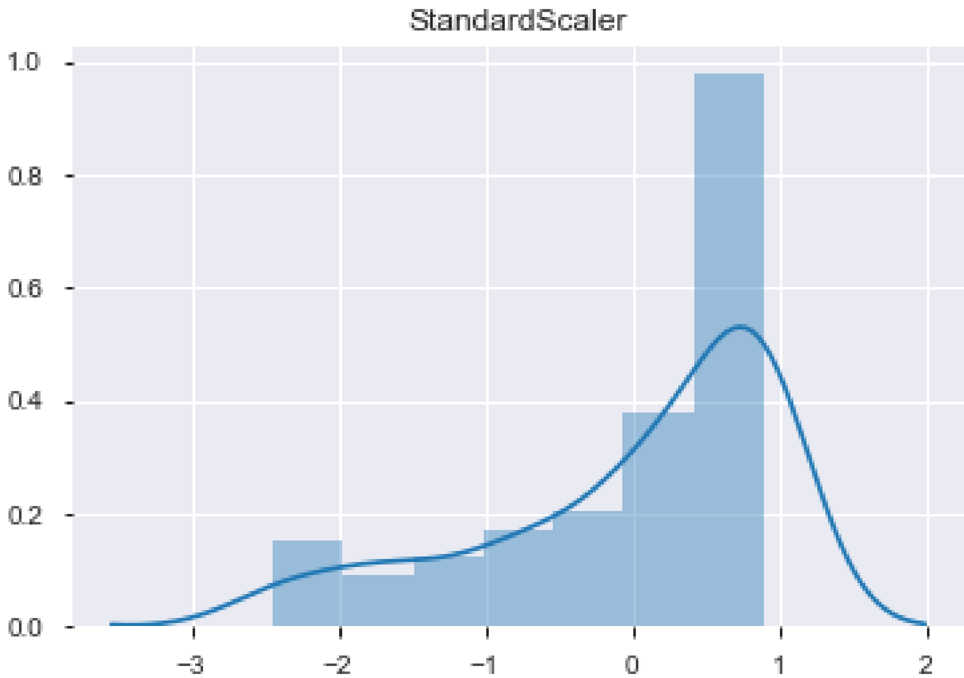
```
In [56]: # Get rid of NaN values
         P = survey_df['Python_Course_Interest'].dropna()
         # Change zero to 1 so ln transformation will work
         P[P == 0] = 1

         # Seaborn provides a convenient way to show the effects of transformations
         # on the distribution of values being transformed
         # Documentation at https://seaborn.pydata.org/generated/seaborn.distplot.html

         unscaledP_fig, ax = plt.subplots()
         sns.distplot(P).set_title('Unscaled')
         unscaledP_fig.savefig('Transformation-UnscaledP' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



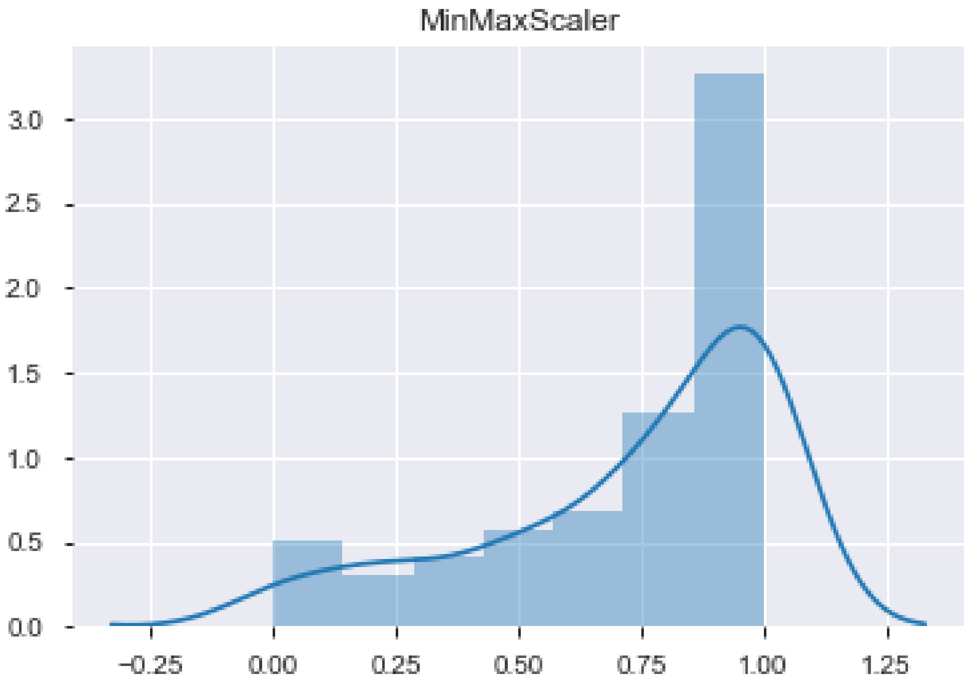

```
In [57]: P = P.values.reshape(-1, 1)

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

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



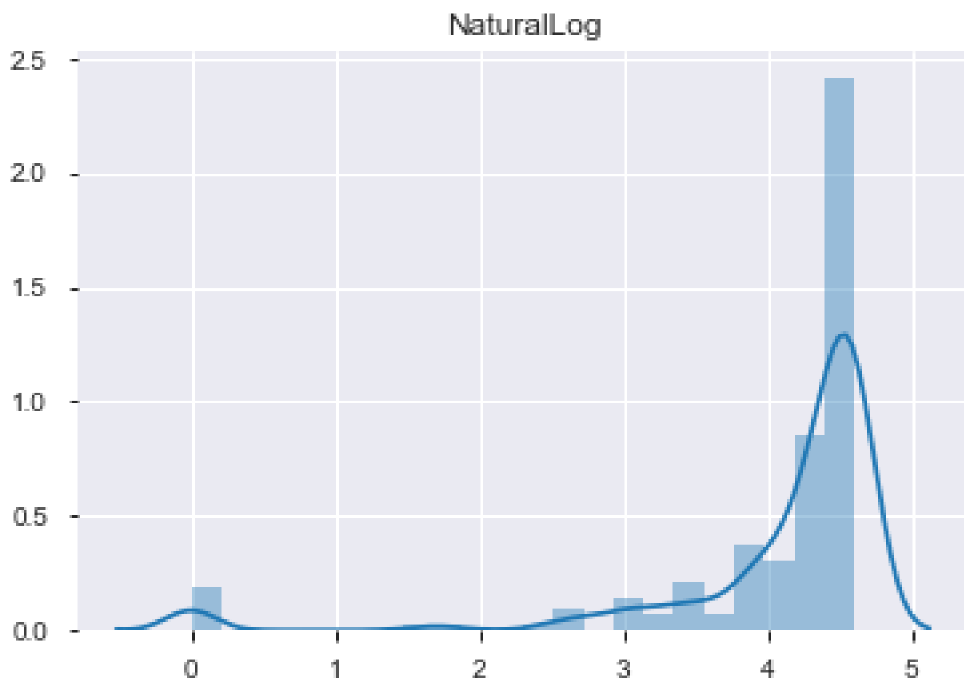

```
In [58]: minmax_figP, ax = plt.subplots()
         sns.distplot(MinMaxScaler().fit_transform(P)).set_title('MinMaxScaler')
         minmax_figP.savefig('Transformation-MinMaxScalerP' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```




```
In [59]: log_figP, ax = plt.subplots()
         sns.distplot(np.log(P)).set_title('NaturalLog')
         log_figP.savefig('Transformation-NaturalLogP' + '.pdf',
             bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
             orientation='portrait', papertype=None, format=None,
             transparent=True, pad_inches=0.25, frameon=None)
```

```
/Users/jmwanat/anaconda3/lib/python3.6/site-packages/matplotlib/axes/_axes.py:6462: UserWarning
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```


NaturalLog
2.5
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
5