In [1]:

```python
import pandas as pd  # data frame operations
import numpy as np  # arrays and math functions
import matplotlib.pyplot as plt  # static plotting
```

In [28]:

```python
# 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)
```

In [5]:

```python
valid_survey_input = pd.read_csv('mspa-survey-data.csv')

# use the RespondentID as label for the rows... the index of DataFrame
```

In [7]:

```python
print('\nContents of initial survey data ---------------')
```

```

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

Number of Respondents = 207
```

In [8]:

```
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 [9]:

```
print(pd.DataFrame.head(valid_survey_input))
#Here we can see the first few values of each of the columns. This illustrates how students distribute the poi
```

```
              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  \
RespondentID      ...
5135740122        ...               NaN         NaN         NaN         NaN
5133300037        ...               NaN         NaN         NaN         NaN
5132253300        ...               NaN         NaN         NaN         NaN
5132096630        ...               NaN         NaN         NaN         NaN
5131990362        ...               NaN         NaN         NaN         NaN

              PREDICT457  OtherPython  OtherR  OtherSAS             Other  \
RespondentID
5135740122           NaN          NaN     NaN       NaN               NaN
```

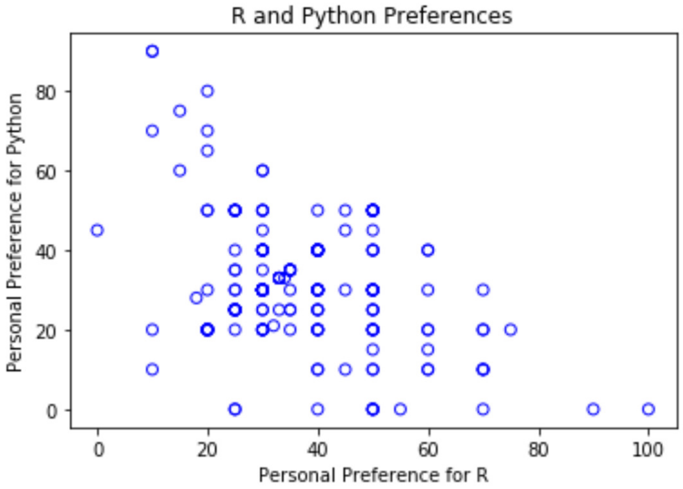
In [10]:

```
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',
```

In [11]:

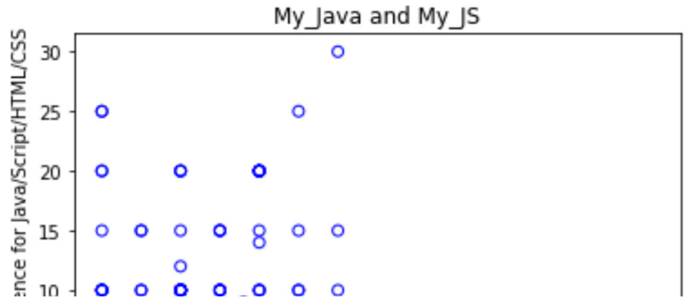
In [13]:

```
fig, axis = plt.subplots()
axis.set_xlabel('Personal Preference for R')
axis.set_ylabel('Personal Preference for Python')
plt.title('R and Python Preferences')
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)

#This graph shows us that the majority of the values in the personal category fall in the center of this graph
#The center of this graph contains numbers between 20-50 for python and 20-60 for R.
```



In [14]:

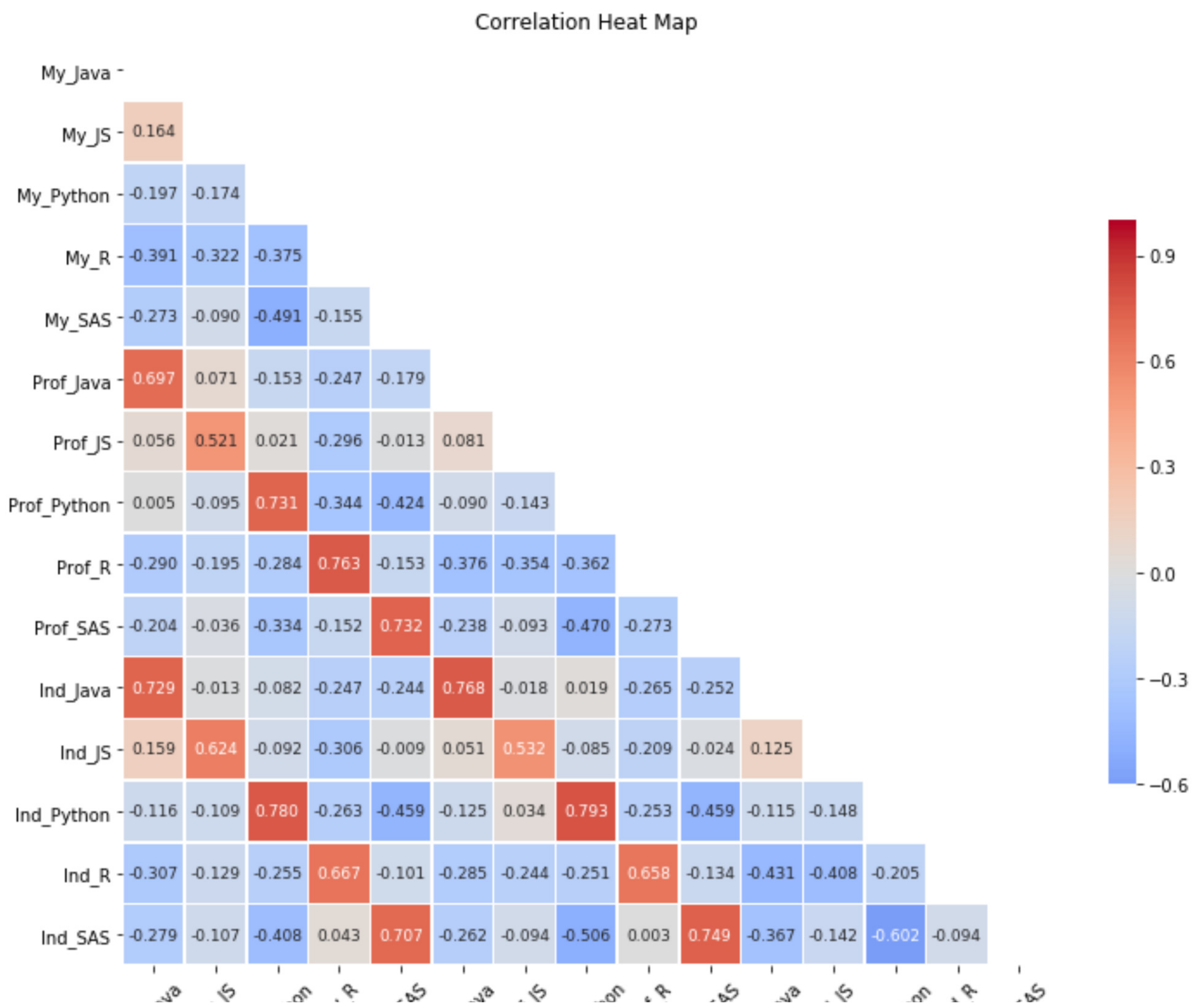
```
survey_df_labels = [
    'Personal Preference for Java/Scala/Spark',
    'Personal Preference for Java/Script/HTML/CSS',
    'Personal Preference for Python',
    'Personal Preference for R',
    'Personal Preference for SAS',
    'Professional Java/Scala/Spark',
    'Professional JavaScript/HTML/CSS',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

# 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,
```



In [15]:

```
corr_chart(df_corr = software_df)
#There are too many relationships between variables here to analyze briefly. But we can say that the
#stronger correlations in the graph belong between relationships of the same languages. This means that
#respondents' desire to learn languages(My_....) is strongly correlated with how important the languages are
#for the jobs they are aspiring to obtain and also it's strongly correlated with the languages that are
#relevant in the industry.
#The rest of the correlations are not significant in my opinion since I consider minimum 0.65 to
#be a good correlation but the closest one that doesn't fall in the cateogry previously described is the
#relationship between Industry Python and Industry SAS, the number tells us that about 60% of respondents
#that showed high interest in Python also showed in SAS.
```

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



In [16]:

```
print('\nDescriptive statistics for survey data ---------------')
print(software_df.describe())
#The descriptive stats below give Python and R a big win with R in first place and SAS comes behind these two.
#The values for relevance for languages across the 3 questions vary slightly which tells us that
#the desire for respondents is almost perfectly linearly related to what the industry and the jobs
```

```
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 [32]:

```
classes_df = survey_df.loc[:, 'Python_Course_Interest':'Systems_Analysis_Course_Interest']

print('\nDescriptive statistics for survey data ---------------')
print(classes_df.describe())
```

```
Descriptive statistics for survey data ---------------
       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 [17]:

```
print('\nDescriptive statistics for courses completed ---------------')
print(survey_df['Courses_Completed'].describe())

#These stats tell us that 20 people did not complete this part of the survey because there were 207
#responses as shown above and only 187 are shown on the count below. It also tell us that the average
```

```
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
```

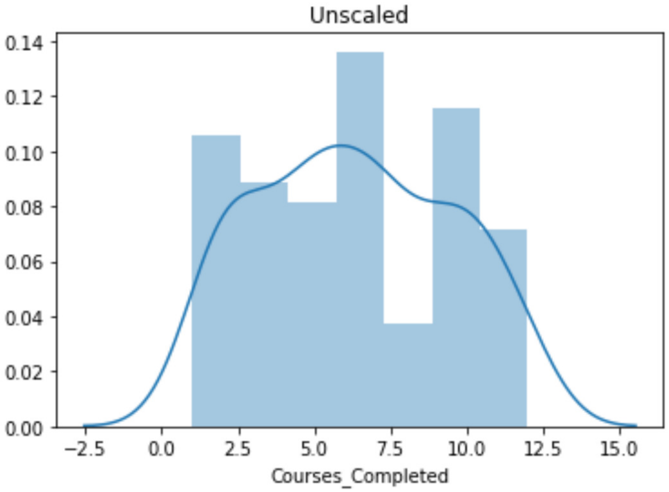
In [18]:
```
from sklearn.preprocessing import StandardScaler
```

In [20]:
```
X = survey_df['Courses_Completed'].dropna()
len(X)
```

Out[20]:
```
187
```

In [24]:
```
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)
```

```
C:\Users\gilad\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
ce for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future th
is will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a dif
ferent result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [25]:

```
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,
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-25-2740750c98f4> in <module>
      1 standard_fig, ax = plt.subplots()
----> 2 sns.distplot(StandardScaler().fit_transform(X)).set_title('StandardScaler')
      3 standard_fig.savefig('Transformation-StandardScaler' + '.pdf',
      4     bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
      5     orientation='portrait', papertype=None, format=None,

~\Anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    460         if y is None:
    461             # fit method of arity 1 (unsupervised transformation)
--> 462             return self.fit(X, **fit_params).transform(X)
    463         else:
    464             # fit method of arity 2 (supervised transformation)

~\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py in fit(self, X, y)
    623         # Reset internal state before fitting
    624         self._reset()
--> 625         return self.partial_fit(X, y)
    626 
    627     def partial_fit(self, X, y=None):

~\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py in partial_fit(self, X, y)
    647         X = check_array(X, accept_sparse=('csr', 'csc'), copy=self.copy,
    648                         warn_on_dtype=True, estimator=self, dtype=FLOAT_DTYPES,
--> 649                         force_all_finite='allow-nan')
    650 
    651         # Even in the case of `with_mean=False`, we update the mean anyway

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_s
parse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, wa
rn_on_dtype, estimator)
    550                     "Reshape your data either using array.reshape(-1, 1) if "
    551                     "your data has a single feature or array.reshape(1, -1) "
--> 552                     "if it contains a single sample.".format(array))
    553 
    554         # in the future np.flexible dtypes will be handled like object dtypes

ValueError: Expected 2D array, got 1D array instead:
array=[ 6.  4.  7.  7.  5. 11.  2.  3.  6.  3.  2.  7.  3.  4.  2. 12.  7.  5.
```

In [26]:

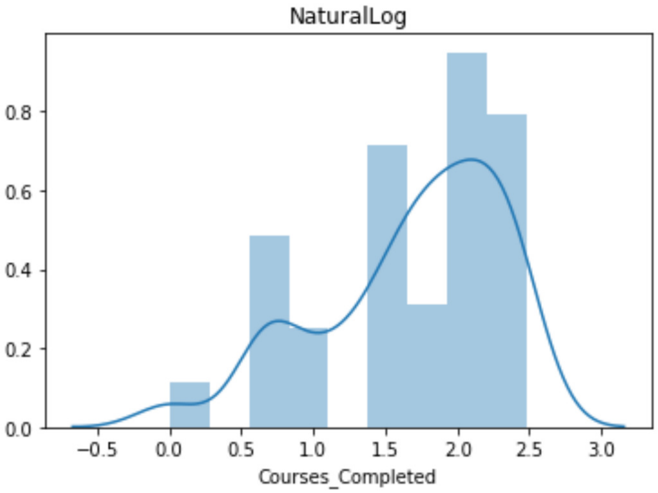
```
minmax_fig, ax = plt.subplots()
sns.distplot(MinMaxScaler().fit_transform(X)).set_title('MinMaxScaler')
minmax_fig.savefig('Transformation-MinMaxScaler' + '.pdf',
    bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
    orientation='portrait', papertype=None, format=None,
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-ea06a6d08a5f> in <module>
      1 minmax_fig, ax = plt.subplots()
----> 2 sns.distplot(MinMaxScaler().fit_transform(X)).set_title('MinMaxScaler')
      3 minmax_fig.savefig('Transformation-MinMaxScaler' + '.pdf',
      4     bbox_inches = 'tight', dpi=None, facecolor='w', edgecolor='b',
      5     orientation='portrait', papertype=None, format=None,

~\Anaconda3\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit_params)
    460         if y is None:
    461             # fit method of arity 1 (unsupervised transformation)
--> 462             return self.fit(X, **fit_params).transform(X)
    463         else:
    464             # fit method of arity 2 (supervised transformation)

~\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py in fit(self, X, y)
    321         # Reset internal state before fitting
    322         self._reset()
--> 323         return self.partial_fit(X, y)
    324 
    325     def partial_fit(self, X, y=None):

~\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py in partial_fit(self, X, y)
    349         X = check_array(X, copy=self.copy, warn_on_dtype=True,
    350                         estimator=self, dtype=FLOAT_DTYPES,
--> 351                         force_all_finite="allow-nan")
    352 
    353         data_min = np.nanmin(X, axis=0)

~\Anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, warn_on_dtype, estimator)
    550                     "Reshape your data either using array.reshape(-1, 1) if "
    551                     "your data has a single feature or array.reshape(1, -1) "
--> 552                     "if it contains a single sample.".format(array))
    553 
    554         # in the future np.flexible dtypes will be handled like object dtypes

ValueError: Expected 2D array, got 1D array instead:
array=[ 6.  4.  7.  7.  5. 11.  2.  3.  6.  3.  2.  7.  3.  4.  2. 12.  7.  5.
```

In [27]:

```
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)
```

```
C:\Users\gilad\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequen
ce for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future th
is will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a dif
ferent result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [ ]:

In [ ]: