homework1-skel

September 15, 2020

NAME: Nigel Mansell

1 Homework 1

1.0.1 Objectives

- Basic numpy operations to access data
- Basic plotting of subsets of data
- Simple descriptive statistics
- Do not save work within the mlp_2020 folder
  - create a folder in your home directory for assignments, and copy the templates there

1.0.2 General References

- Sci-kit Learn Iris Dataset
- Numpy Reference
- Summary of matplotlib
  - Plot
  - Boxplots
  - Histograms
  - Scatter plots
  - Colormap Plots

1.0.3 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook:
  - Submit this file (.ipynb) to the Canvas HW1 dropbox
- PDF:
  - File/Export Notebook As/PDF -> Produces a copy of the notebook in PDF format
  - Submit the PDF file to the Gradescope HW1 dropbox

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from IPython import get_ipython
from sklearn.datasets import load_iris
```

2 LOAD IRIS DATA SET

[2]:
```
"""
Load the dataset into the iris_dataset variable, by calling the
load_iris() function imported from sklearn.datasets.
Then display the iris_dataset object's list of keys. iris_dataset
is a dictionary object.
"""

"laoding the data and printing the keys"
iris_dataset = load_iris()
iris_dataset.keys()
```

[2]:
```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names',
'filename'])
```

2.0.1 Dataset Details

The `iris_dataset` variable is a dictionary with multiple fields: * `data` : m by n numpy array of the n observed feature values, for each of the m samples
* `target` : m by 1 numpy array of samples' classification as iris-setosa (i.e. 0), iris-versicolour (i.e. 1), or iris-virginica (i.e. 2) * `target_names` : 3 by 1 numpy array of the possible iris classifications
* `DESCR` : string containing a detailed description of the dataset
* `feature_names` : n by 1 numpy array of the names of the feature variables
* `filename` : string containing the absolute path to where the file containing all the data information is located on the local system

[3]:
```
"""
Print out the description of the data, by accessing the
'DESCR' field of the iris data set
"""

"Printing the discription from its key"
print(iris_dataset['DESCR'])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

2.1 SETUP USEFUL VARIABLES

[4]:
```
"""
Store the names of the features and the names
of the target classes, into the variables
feature_names and target_names respectively.
"""

"seting feature and target names to their associated keys"
feature_names =  iris_dataset['feature_names']
target_names = iris_dataset['target_names']

"""
Print the list of feature names and target names
"""

"printing feature and target names"
print(feature_names)
print(target_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width
(cm)']
['setosa' 'versicolor' 'virginica']
```

[5]:
```
"""
Create variables for the feature and target data
The X variable is a numpy array containing the data measured
for each feature for each sample. Each column of X is a
different feature for all the samples. Each row of X is a
different sample with all its features.
The y variable is a numpy array containing the classification
for each sample. A sample iris is either setosa, versicolor, or virginica.
"""

"setting X to the data from the key data and y to data from target"
X = iris_dataset['data']
y = iris_dataset['target']

"""
Print the dimensions of the X and y variables respectively
```

```
"""

"printing both x and y dimensions"
print(X.ndim)
print(y.ndim)
```

```
2
1
```

[6]:
```
"""
Store the number of samples and the number of features, by
accessing the values from the shape of X
"""

"setting nsamples to the row of X and nfeatures to the column of X"
nsamples, nfeatures = X.shape

"""
Print the print the number of samples and numberof features respectively
"""

"printing nsamples and nfeatures"
print(nsamples)
print(nfeatures)
```

```
150
4
```

2.2 SELECT SUBSET OF FEATURES

Not all available data is necessary or useful for making predictions and classifying observations. There are numerous feature selection algorithms that exist, which will be discussed in more detail later within the cousre. For now we are going to arbitrarly select sepal length, sepal width and petal length as our predictor variables. We will not yet be performing any predictions in this assignment; rather this term is used to conveniently distinuguish this subset of features from the full set of features.

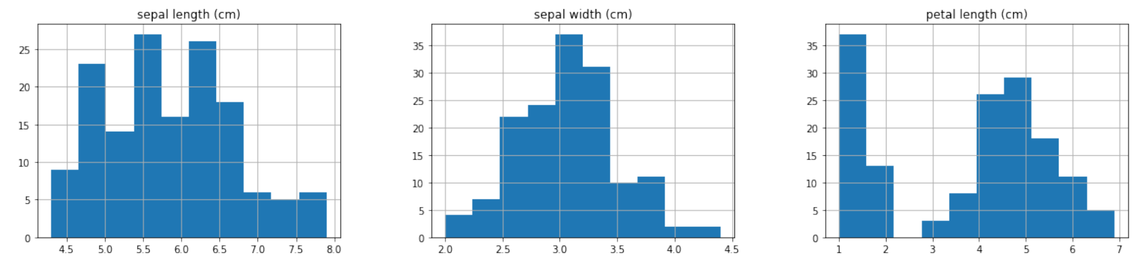
[7]:
```
""" PROVIDED
Feature Column Indices
The values observed for each feature resides within a particular
column of the feature matrix, X. For example, column 0 contains the
values of the mean radius for each observation, the column at index
3 contains the values for the mean area, and so on.
"""
sepal_length_idx = 0
sepal_width_idx = 1
petal_length_idx = 2
```

```
"""
Create a list of the select subset of features
"""
predictors = [sepal_length_idx, sepal_width_idx, petal_length_idx]

"""
Create a variable, storing the number of predictors
"""


"the length of predictors"
predictorSize = len(predictors)

"""
Create a list of corresponding names for the selected set of features.
This is conveniently done using list comprehension
"""

"the names of the predictors from feature_names array"
predictorNames = [feature_names[i] for i in range(predictorSize)]
"""
Print the list of predictor names
"""

"prints the names"
print(predictorNames)
```

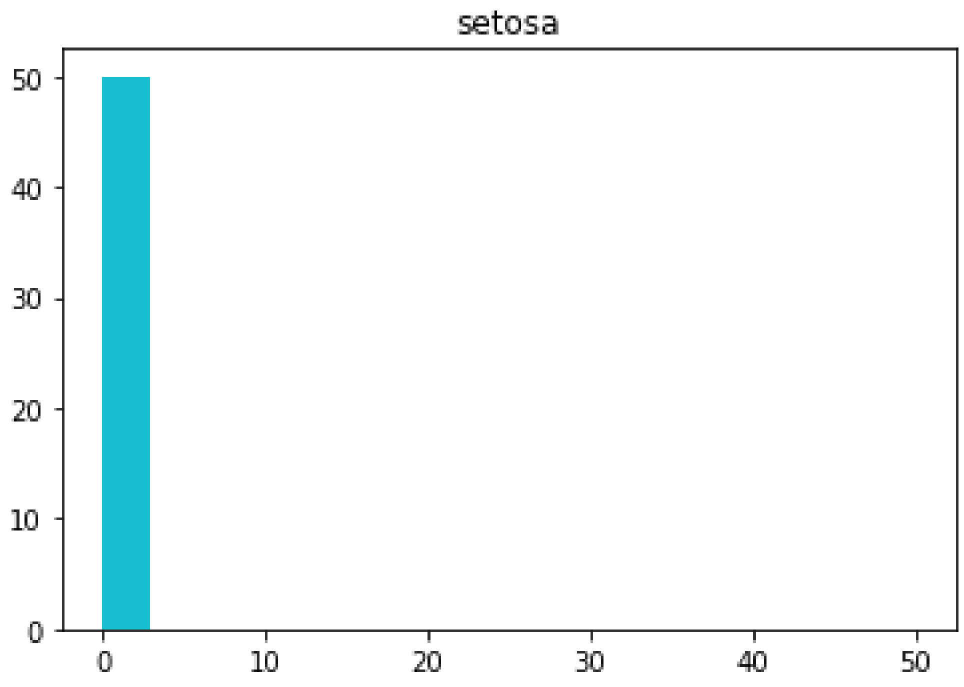
```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)']
```

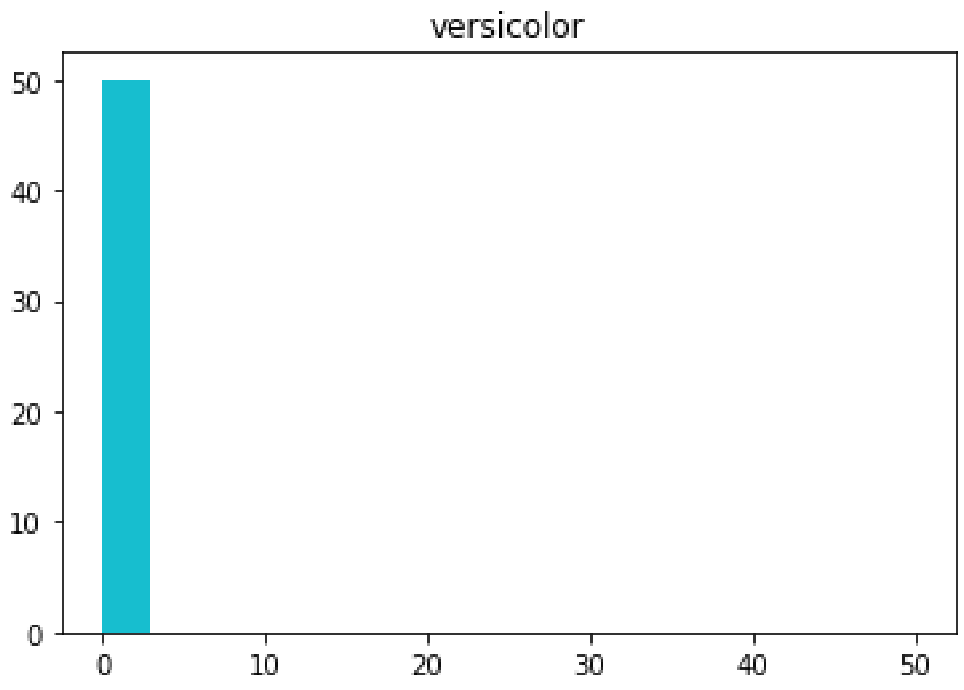
2.3 BASIC HISTOGRAMS OF FEATURES

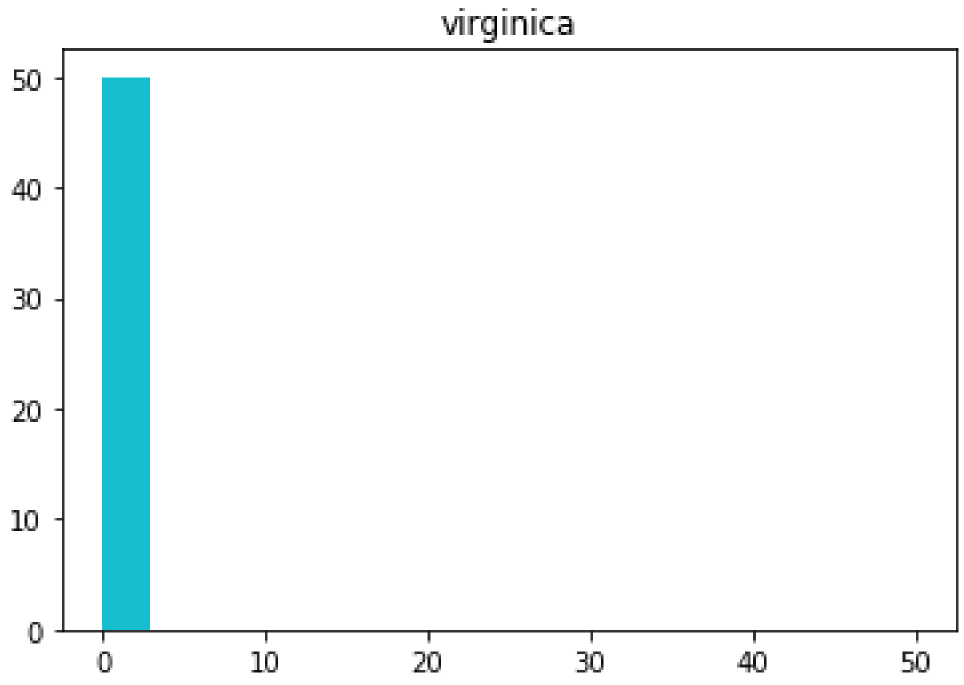
[8]:
```
""" TODO
HISTOGRAMS OF THE CHOSEN PREDICTOR FEATURES
Please plot histograms in their own subplot of
the same figure.
"""

"Turned X into a dataframe in order to iterate over columns easier"
df = pd.DataFrame(data=X,index=None,columns=feature_names)
plt.figure(figsize=(20,4))
plt.subplots_adjust(wspace=.3)
for i, fidx in enumerate(predictors, 1):
    plt.subplot(1, 3, i)
    plt.title(predictorNames[fidx])
    df[predictorNames[fidx]].hist()
    # TODO: Plot the histogram
```



[9]:
```
""" TODO
Create a histogram or barplot for the counts
for each target class
"""

"creates 3 histograms for each class and displays their count, or how often they␣
 ↪appear"
for i in y:
    plt.figure(i+1)
    plt.title(target_names[i])
    plt.hist(y, bins=np.arange(0, 51))
```




versicolor
0
10
20
30
40
50
0
10
20
30
40
50


virginica
0
10
20
30
40
50
0
10
20
30
40
50

2.4 BASIC BOXPLOTS OF FEATURES

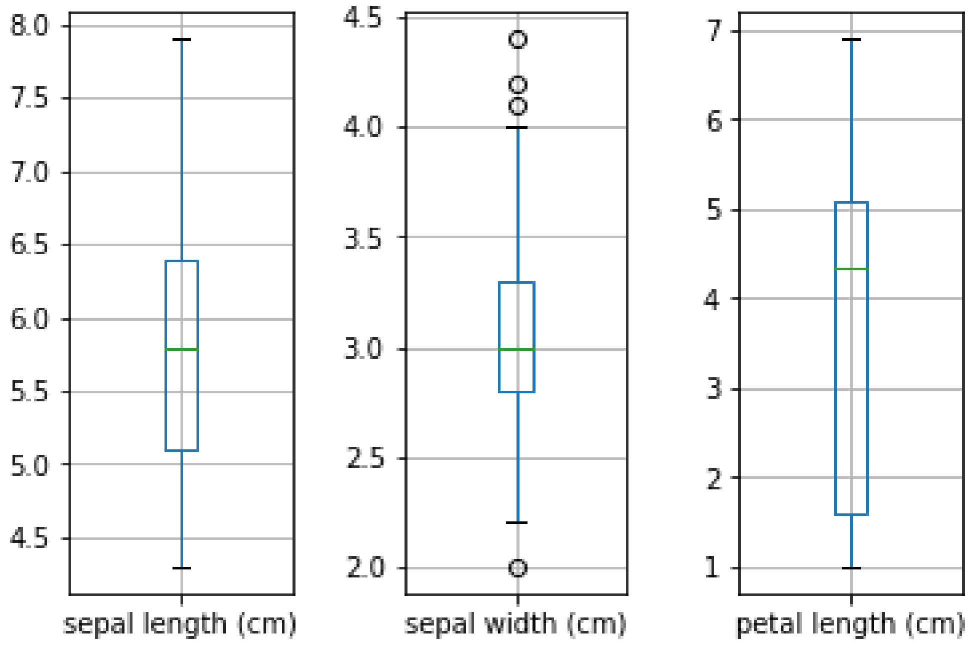
Boxplots or box-and-whisker plots are used to obtain a perspective of the distribution of the data. The box within the figure displays the 25th percentile (Q1), the median, and the 75th percentile (Q3) of the data. The range between the 75th percentile value and the 25th percentile value is the interquartile range (IQR = Q3 - Q1). The end of bottom line is Q1 - 1.5 * IQR. The end of top line is Q3 + 1.5 * IQR. Anything beyond the lines, the circles, are suggested outliers.

<

>

One can use the `boxplot(data_values, labels=[name])` to generate a boxplot. `data_values` would be the set of observed values for a paritucular feature and `labels` should be provided as a list, with the name of the feature, in place of `name`.

[10]:
```
""" TODO
BOXPLOTS OF THE CHOSEN PREDICTOR FEATURES
Please place the boxplots within their own
subplot of the same figure
"""

"Creates boxplots for predictors"
plt.subplots_adjust(wspace=.5)
for i, fidx in enumerate(predictors, 1):
    plt.subplot(1, 3, i)
    df.boxplot(column=predictorNames[fidx])
```



2.5 DESCRIPTIVE STATISTICS

[11]:
```
# Simply run this cell
"""
Create a separate variable of the data from the
predictors
"""
Xpreds = X[:, predictors]

"""
Check if any values are NaN (not a number)
"""
np.any(np.isnan(Xpreds))
```

[11]:
```
False
```

[12]:
```
""" TODO
Compute the following descriptive statistics of the
features ignoring NaN values, using numpy:
mean, median, standard deviation, min, and max

Make sure to compute the statistics of the columns
of X (i.e. of each feature). You can specify this
by setting axis=0 for each of the functions

Compute and print the results
"""

"Prints the non nan vales of each statistic"
print("Mean: ", np.nanmean(Xpreds, axis=0))
print("Median: ", np.nanmedian(Xpreds, axis=0))
print("Standard deviation: ", np.nanstd(Xpreds, axis=0))
print("Min: ", np.nanmin(Xpreds, axis=0))
print("Max: ", np.nanmax(Xpreds, axis=0))
```

```
Mean:  [5.84333333 3.05733333 3.758     ]
Median:  [5.8  3.   4.35]
Standard deviation:  [0.82530129 0.43441097 1.75940407]
Min:  [4.3 2.  1. ]
Max:  [7.9 4.4 6.9]
```

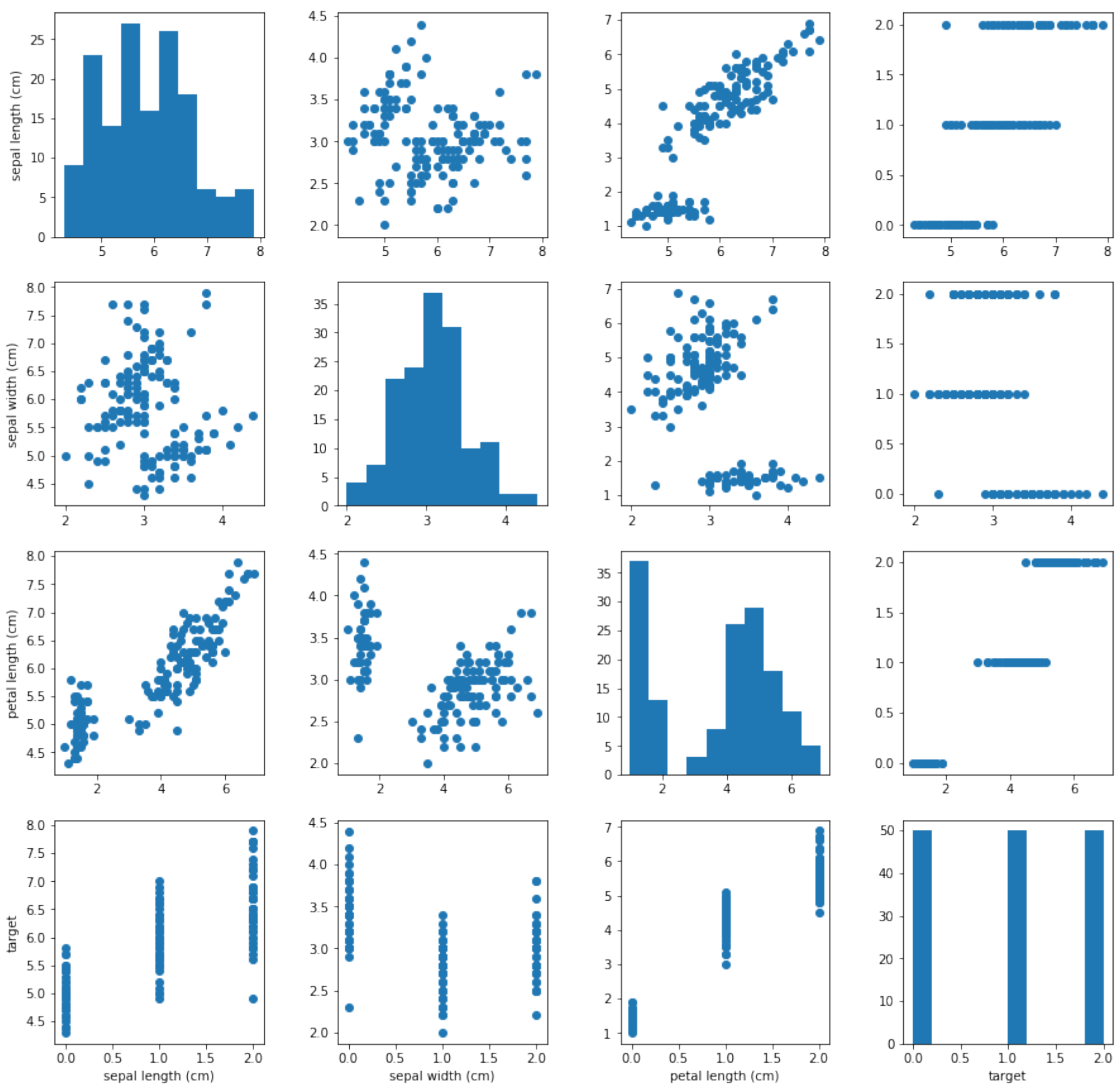
2.6 FEATURE CORRELATIONS

It's useful to know the correlation between various features, as well as each feature and the predicted label. Feature correlation is useful for feature selection and understanding the relationship between multiple variables within a dataset. Correlation is either positive, negative, or zero. When two features increase simultaneously, they are positively correlated. When one feature increases while the other decreases, the features are negatively correlated. Zero

correlation is when there is no relationship between the features. Correlation is on the range -1 (perfect negative correlation) and 1 (pefect positive correlation).
We can construct scatter plots of one feature versuses another to observe linear or nonlinear relationships.
Complete the following set of scatter plots:
<

>

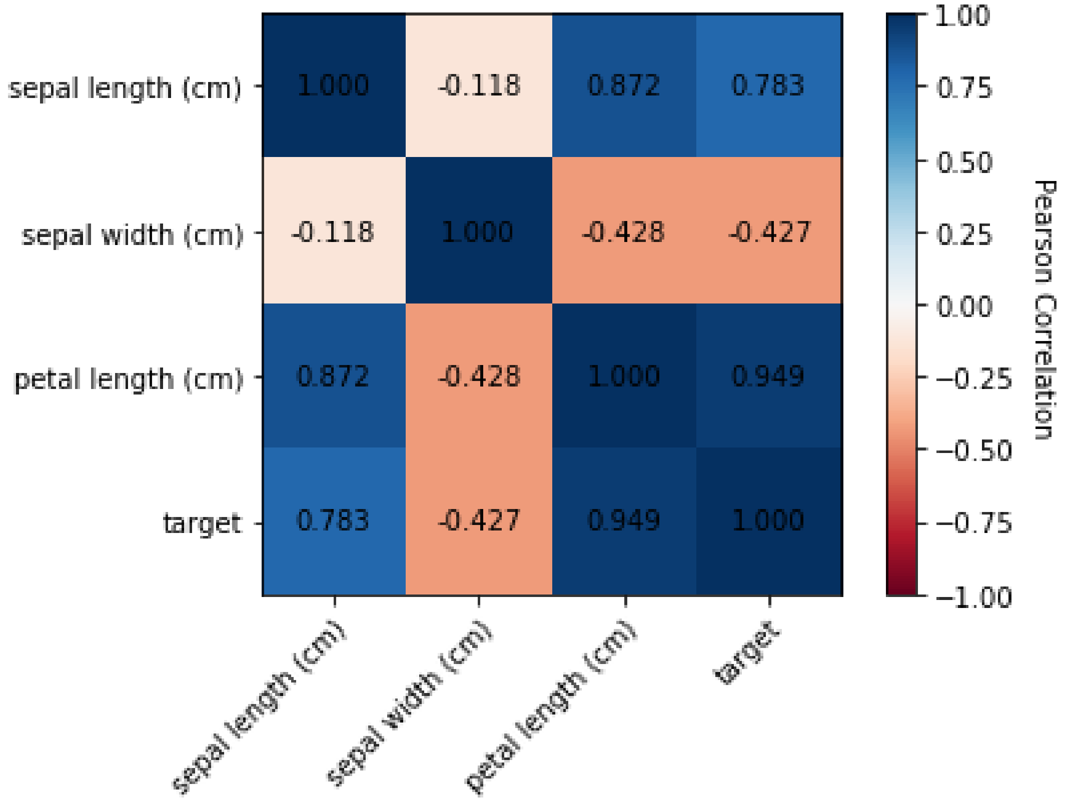
[13]:
```python
"""
Using the scatter plot function, construct plots depicting the
correlation between all pairings of the selected predictor features
and between all predictors and the determined target.
The figure will contain r by r subplots, where r = npredictors + 1.
Where subplot(i,j) is a scatter plot of the feature i versus feature j.
When i == j, plot the histogram of feature i instead of a scatter plot.
We are also interested in the correlation between each of the features
and the target classification, thus we will combine the predictors matrix
and the target vector into one large matrix for convenience.
"""
# Append the y to the end of the matrix of predictors
Xycombo = np.append(Xpreds, y.reshape(-1, 1), axis=1)
# Append the name 'target' to the end of the list of predictor names
Xycolnames = predictorNames + ['target']

df_Xycombo = pd.DataFrame(data=Xycombo,index=None, columns=Xycolnames)
# Create the scatter plots
fig, axs = plt.subplots(predictorSize+1, predictorSize+1, figsize=(15, 15))
fig.subplots_adjust(wspace=.35)
for f1 in range(predictorSize+1):
    for f2 in range(predictorSize+1):
        if f1 == f2:
            "creates a histogram if feature1 and feature2 are equal"
            axs[f1][f2].hist(df_Xycombo[Xycolnames[f1]])
        else:
            "scatter plots the difference between feature1 and feature2"
            axs[f1][f2].scatter(df_Xycombo[Xycolnames[f1]],␣
 ↪df_Xycombo[Xycolnames[f2]])
        if f1 == predictorSize:
            axs[f1, f2].set_xlabel(Xycolnames[f2])
        if f2 == 0:
            axs[f1, f2].set_ylabel(Xycolnames[f1])
```



2.7 IMAGES AND COLORMAPS

Create a colormap plot of the correlations between the all the predictors and the target

[15]:
```
""" PROVIDED
Generate a figure that plots the a correlation matrix
as a colormap.
PARAMS:
    corrs: matrix of correlations between the features
    varnames: list of the names of each of the features
              (e.g. the column names)
"""
def correlationmap(corrs, varnames):
    nvars = corrs.shape[0]
    # create the figure and plot the correlation matrix
    fig, ax = plt.subplots()
    im = ax.imshow(corrs, cmap='RdBu', vmin=-1, vmax=1)
    cbar = ax.figure.colorbar(im, ax=ax)
    cbar.ax.set_ylabel("Pearson Correlation", rotation=-90, va="bottom")

    # Specify the row and column ticks and labels for the figure
    ax.set_xticks(range(nvars))
    ax.set_yticks(range(nvars))
    ax.set_xticklabels(varnames)
    ax.set_yticklabels(varnames)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",␣
 ↪rotation_mode="anchor")

    # Loop over data dimensions and create text annotations.
    for i in range(nvars):
        for j in range(nvars):
            text = ax.text(j, i, "%.3f" % corrs[i, j],
                           ha="center", va="center", color="k")
# END DEF correlationmap


""" TODO
Compute the Pearson correlation between the columns of Xycombo using
the numpy function corrcoef(). The corrcoef() function performs the
the pairwise correlation on the rows of a matrix, thus you will need to
transpose the input.
"""

"transposes Xycombo then is used in np.corrcoef"
Xycorrs = np.corrcoef(np.transpose(Xycombo))

""" TODO
Call the function defined above, correlationmap(), to generate a
colormap plot of the correlations between columns of the Xycombo matrix
"""

"calling correlationmap by passing Xycorrs and Xycolnames"
correlationmap(Xycorrs, Xycolnames)
```


sepal length (cm)
sepal width (cm)
petal length (cm)
target
1.000
-0.118
0.872
0.783
-0.118
1.000
-0.428
-0.427
0.872
-0.428
1.000
0.949
0.783
-0.427
0.949
1.000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Pearson Correlation

[ ]: