CSC 522 HW 1

Group H29

September 6, 2022

Ethan Purnell (efpurne2)
Corey Capooci (cvcapooc)
Jackson Ingraham (jtingrah)
Github Repository: engr-ALDA-Fall2022-H29
`https://github.ncsu.edu/efpurne2/engr-ALDA-Fall2022-H29`

Question 1:

(a) Ratio and continuous
(b) Interval and continuous
(c) Nominal and binary
(d) Ordinal and discrete because there is no meaningful unit of measure between each placement.
(e) Ratio and continuous
(f) Ratio and discrete because the number of Legos is similar to counting the Legos. Therefore, 2 Legos is twice as many Legos as 1 Lego.
(g) Ratio and discrete because minute is a discernible unit of measurement and if it is minute 100 of the day, then it twice as far along in the day as opposed to minute 50.
(h) Nominal and discrete because it is a countably infinite set of values.
(i) Ordinal and discrete
(j) Ratio and continuous
(k) Ratio and discrete because the values are integers and one team can score twice as much as another.
(l) Ratio and continuous
(m) Ordinal and discrete
(n) Nominal and binary
(o) Ordinal and discrete because the page number of a book can indicate how far along in a book, but there is no discernible unit of measurement for a page in a book.
(p) Ordinal and discrete because the outcome of a dice roll is meaningless until it is applied to another context. Therefore, the values on a die provide order but with no unit of measurement by itself.
(q) Nominal and discrete
(r) Ratio and continuous because even though calories are often displayed as an integer the unit of measurement can be a real value.
(s) Ratio and continuous
(t) Ordinal and continuous

Question 2:

Please reference 'HW1Q2.py' for the entire solution of question 2. Relevant code and results for each part is shown below. Please note for all code checks, you will have to ensure that the same packages we used are installed in your python IDE!

Part A

Relevant code:

```
a = np.identity(5)
```

Value of a:

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

Part B

Relevant code:

```
a[:, 4] = 5
```

Value of a:

```
[[1. 0. 0. 0. 5.]
 [0. 1. 0. 0. 5.]
 [0. 0. 1. 0. 5.]
 [0. 0. 0. 1. 5.]
 [0. 0. 0. 0. 5.]]
```

Part C

Relevant code:

```
sum = 0
for x in np.nditer(a):
    sum += x
```

Value of the sum variable after this code is run is 29.

Part D

Relevant code:

```
a = a.T
```

Value of a:

```
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [5. 5. 5. 5. 5.]]
```

Part E

Relevant code:

```
sum = 0
for x in a[4, :]:
    sum += x
sum = 0
for x in np.diagonal(a):
    sum += x
sum = 0
for x in a[:, 0]:
    sum += x
```

The values for sum after this code is run are 25, 9, and 6

Part F

Relevant code:

```
b = np.random.standard_normal((5, 5))
```

Value of b:

```
[[-5.27899086e-04 -2.74901425e-01 -1.39285562e-01  1.98468616e+00
   2.82109326e-01]
 [ 7.60808658e-01  3.00981606e-01  5.40297269e-01  3.73497287e-01
   3.77813394e-01]
 [-9.02131926e-02 -2.30594327e+00  1.14276002e+00 -1.53565429e+00
  -8.63752018e-01]
 [ 1.01654494e+00  1.03396388e+00 -8.24492228e-01  1.89048564e-02
  -3.83343556e-01]
 [-3.04185475e-01  9.97291506e-01 -1.27273841e-01 -1.47588590e+00
  -1.94090633e+00]]
```

Part G

Relevant code:

```
c = np.empty((2, 5))
c[0, :] = np.subtract(b[0, :], a[0, :])
c[1, :] = np.add(a[4, :], b[4, :])
```

Value of c:

```
[[-1.0005279  -0.27490142 -0.13928556  1.98468616  0.28210933]
 [ 4.69581453  5.99729151  4.87272616  3.5241141   3.05909367]]
```

Part H

Relevant code:

```
d = np.multiply(np.arange(1,6,1), c)
```

Value of d:

```
[[-1.0005279  -0.54980285 -0.41785668  7.93874463  1.41054663]
 [ 4.69581453 11.99458301 14.61817848 14.09645639 15.29546836]]
```

Part I

Relevant code:

```
covmatrix = np.cov([x,y,z])
ccxy = np.corrcoef(x,y)
```

Values:

```
Covariance matrix
[[ 0.25        1.5        -0.5       ]
 [ 1.5       15.         -5.        ]
 [-0.5       -5.          1.66666667]]
Pearson correlation coefficients
[[1.         0.77459667]
 [0.77459667 1.        ]]
```

Part J

Relevant code:

```
xsm = np.mean(x**2) # Base mean to compare results to
xpstd = np.std(x) # Population Standard Deviation
xbar = m.sqrt((xm**2)+(xpstd**2))
xsstd = np.std(x, ddof=1) # Sample Standard Deviation
xbar = m.sqrt((xm**2)+(xsstd**2))
```

The value of the mean of the square of x was found to be 464.25
The value of the mean of the square of x using the population standard deviation (xpstd) was found to be 464.25
The value of the mean of the square of x using the sample standard deviation (xsstd) was found to be 464.5357142857143

Question 3:

Part A

Reports for each of the desired attributes (area, perimeter, length of kernel, width of kernel) are provided below. The tables are the outputs of our code which can be found in the Github repository listed on the title page. It should run without issue as long as the entire folder is downloaded to ensure that the data is included. The relevant file for this problem is HW1Q3.py.

Relevant Code:

```
# Area report
area = SeedDF['area'].describe()
area.loc['range'] = area.loc['max'] - area.loc['min']
area.loc['median'] = SeedDF['area'].median()
area.drop(['count', 'min', 'max'], inplace=True)
print('Area Report')
print(area)
print('\n')

# Perimeter report
perim = SeedDF['perimeter'].describe()
perim.loc['range'] = perim.loc['max'] - perim.loc['min']
perim.loc['median'] = SeedDF['perimeter'].median()
perim.drop(['count', 'min', 'max'], inplace=True)
print('Perimeter Report')
print(perim)
print('\n')

# Length of kernel report
kl = SeedDF['length of kernel'].describe()
kl.loc['range'] = kl.loc['max'] - kl.loc['min']
kl.loc['median'] = SeedDF['length of kernel'].median()
kl.drop(['count', 'min', 'max'], inplace=True)
print('Kernel Length Report')
print(kl)
print('\n')

# Width of kernel
kw = SeedDF['width of kernel'].describe()
kw.loc['range'] = kw.loc['max'] - kw.loc['min']
kw.loc['median'] = SeedDF['width of kernel'].median()
kw.drop(['count', 'min', 'max'], inplace=True)
print('Kernel Width Report')
```

```
print(kw)
print('\n')
```

Output:

Area Report
mean 14.847524
std 2.909699
25% 12.270000
50% 14.355000
75% 17.305000
range 10.590000
median 14.355000
Name: area, dtype: float64

Perimeter Report
mean 14.559286
std 1.305959
25% 13.450000
50% 14.320000
75% 15.715000
range 4.840000
median 14.320000
Name: perimeter, dtype: float64

Kernel Length Report
mean 5.628533
std 0.443063
25% 5.262250
50% 5.523500
75% 5.979750
range 1.776000
median 5.523500
Name: length of kernel, dtype: float64

Kernel Width Report
mean 3.258605
std 0.377714
25% 2.944000
50% 3.237000
75% 3.561750
range 1.403000
median 3.237000
Name: width of kernel, dtype: float64

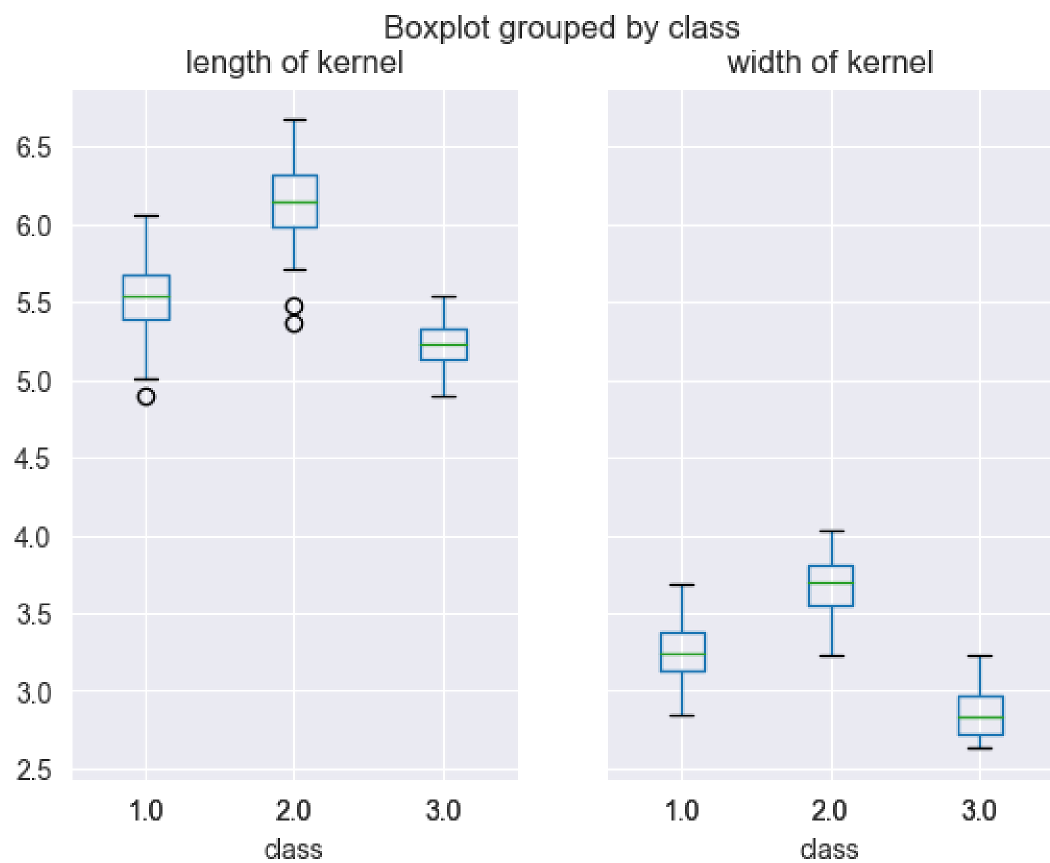
Part B

The box-and-whisker plot visualizing data for the length of kernel and width of kernel attributes, grouped by class, are shown below.

0.0.1 Relevant Code:

```
# B) Make a box-and-whisker plot for the attributes length of
   kernel and width of kernel where they are grouped by the
# class label. Include a title for each plot of what feature is
    being described.
plt.figure()
boxplot = SeedDF.boxplot(column=['length of kernel', 'width of
   kernel'], by='class')
plt.show()
```

Output:



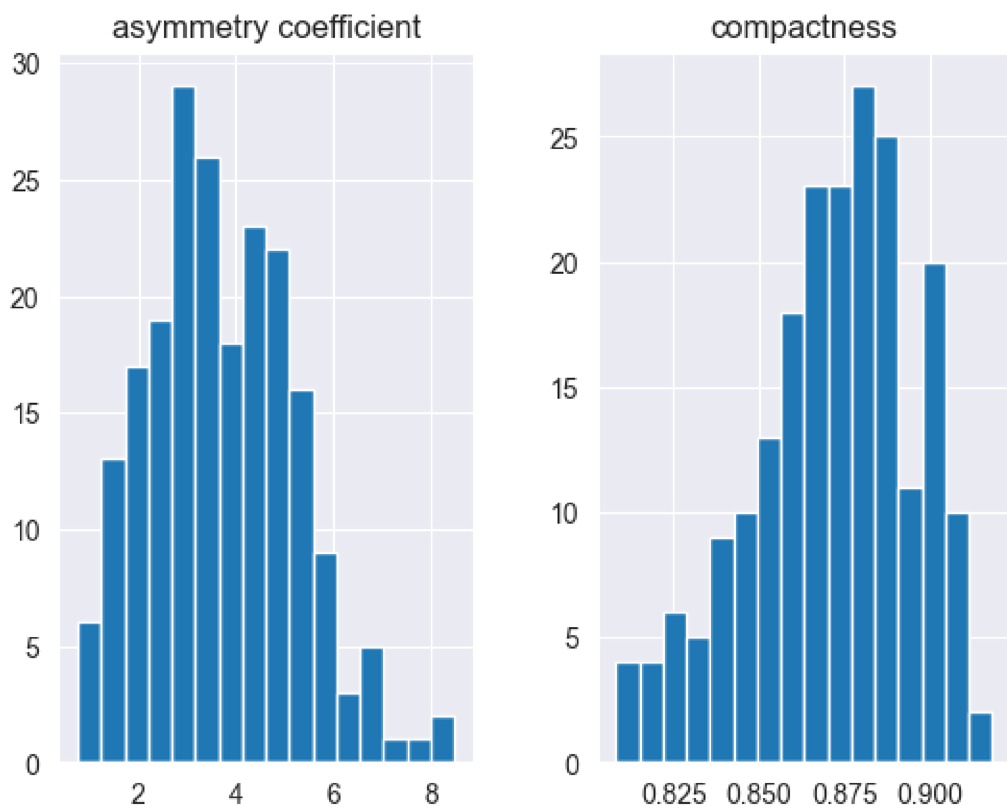
Part C

Histogram plots for the attributes asymmetry coefficient and compactness with 16 bins each are shown below.

Relevant Code:

```
# C) Make a histogram plot w/ 16 bins for the two features
   asymmetry coefficient and compactness, respectively
plt.figure()
hist = SeedDF.hist(column=['asymmetry coefficient', '
   compactness'], bins=16)
plt.show()
```

Output:



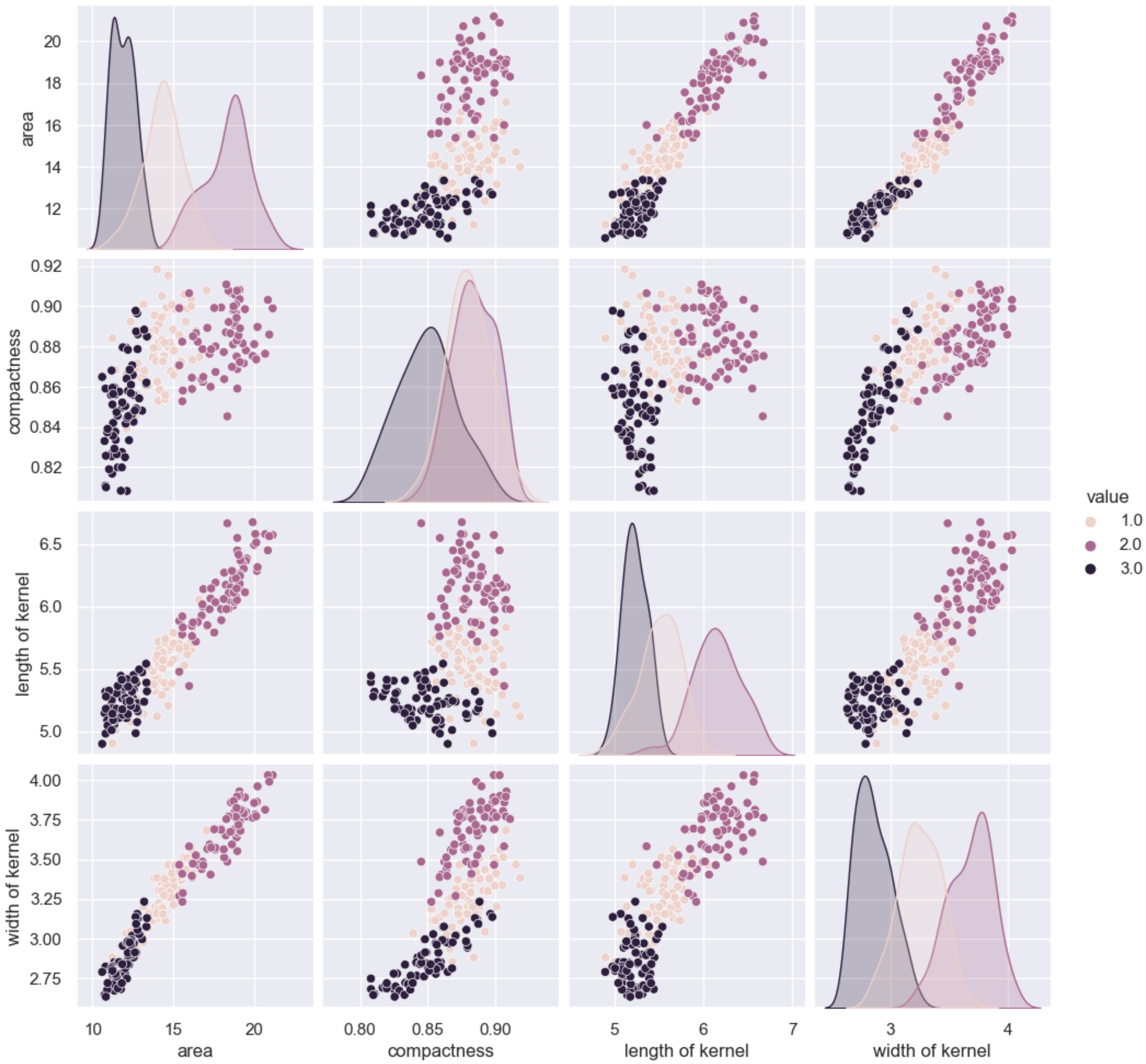
Part D

The scatter matrix for the features area, compactness, length of kernel, and width of kernel is shown below. The data points are colored by class, and the kernel density estimation (shown on the diagonal) is plotted for each class as well.

Relevant Code:

```
# D) Make a scatter matrix with area, compactness, length of
   kernel, width of kernel. Use class attribute to change
# the color of data points. For diagonal of scatter matrix,
   plot kernel density estimation
plt.figure()
scatdf = pd.melt(SeedDF, id_vars=['area', 'compactness', '
   length of kernel', 'width of kernel'], value_vars=['class'])
sbn.pairplot(scatdf, hue='value', diag_kind='kde')
plt.show()
```

Output:


area
compactness
length of kernel
width of kernel
value
1.0
2.0
3.0

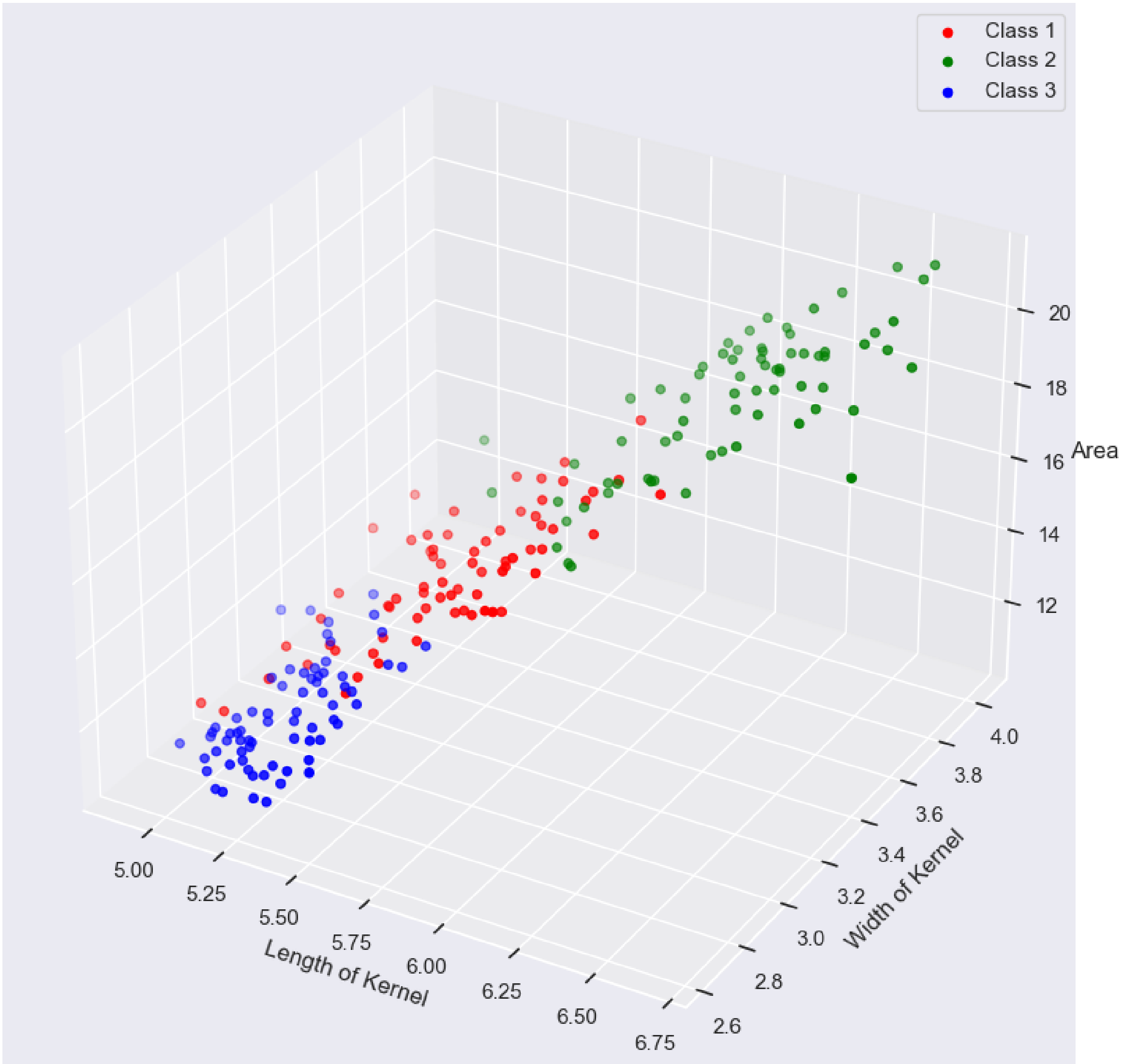
Part E

The 3D scatterplot across the dimensions area, length of kernel, and width of kernel, with data points colored by class, is shown below.

Relevant Code:

```
# E) Produce 3D scatter plot using length of kernel, width of
   kernel, and area as dimensions, and color data points
# according to class attribute
sbn.set()
scatplot = plt.figure(figsize=(10,10))
ax = plt.axes(projection='3d')
dfmelt1 = scatdf.loc[scatdf['value']==1.0]
dfmelt2 = scatdf.loc[scatdf['value']==2.0]
dfmelt3 = scatdf.loc[scatdf['value']==3.0]
ax.scatter3D(dfmelt1['length of kernel'], dfmelt1['width of
   kernel'], dfmelt1['area'], color='red', label='Class 1')
ax.scatter3D(dfmelt2['length of kernel'], dfmelt2['width of
   kernel'], dfmelt2['area'], color='green', label='Class 2')
ax.scatter3D(dfmelt3['length of kernel'], dfmelt3['width of
   kernel'], dfmelt3['area'], color='blue', label='Class 3')
ax.set_xlabel('Length of Kernel')
ax.set_ylabel('Width of Kernel')
ax.set_zlabel('Area')
ax.legend()

plt.show()
```

Output:


Class 1
Class 2
Class 3
Length of Kernel
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75
Width of Kernel
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
Area
12
14
16
18
20

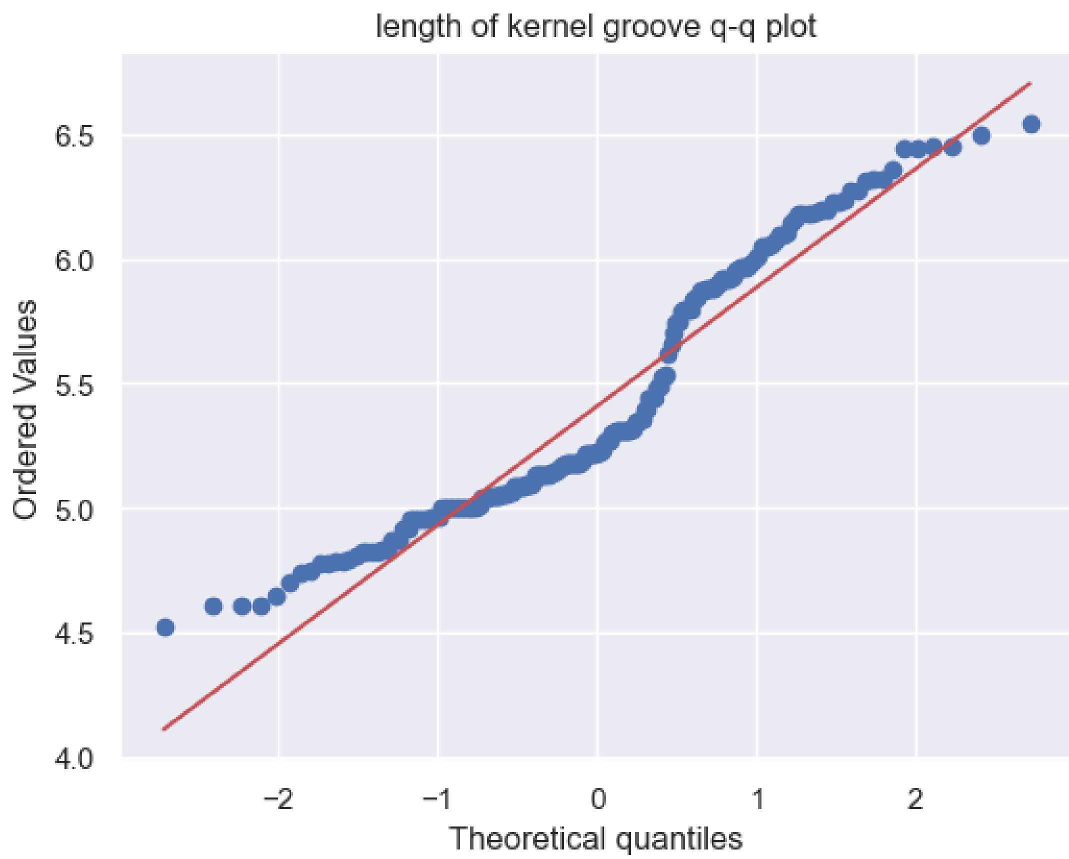
Part F

Quantile-quantile plots are shown below. These plots show that the quantiles from the data when compared to a normal distribution do not fall on a straight line very well, suggesting that our data is not very normally distributed. This is expected, as the compactness histogram from Part C appears to be skewed to the right and not normally distributed. The compactness data is skewed to the right of the straight line on its q-q plot, which is in line with what is observed on the histogram. The length of kernel groove q-q plot suggests that the data does not follow a normal distribution very well. It may be under-dispersed based on the shape.

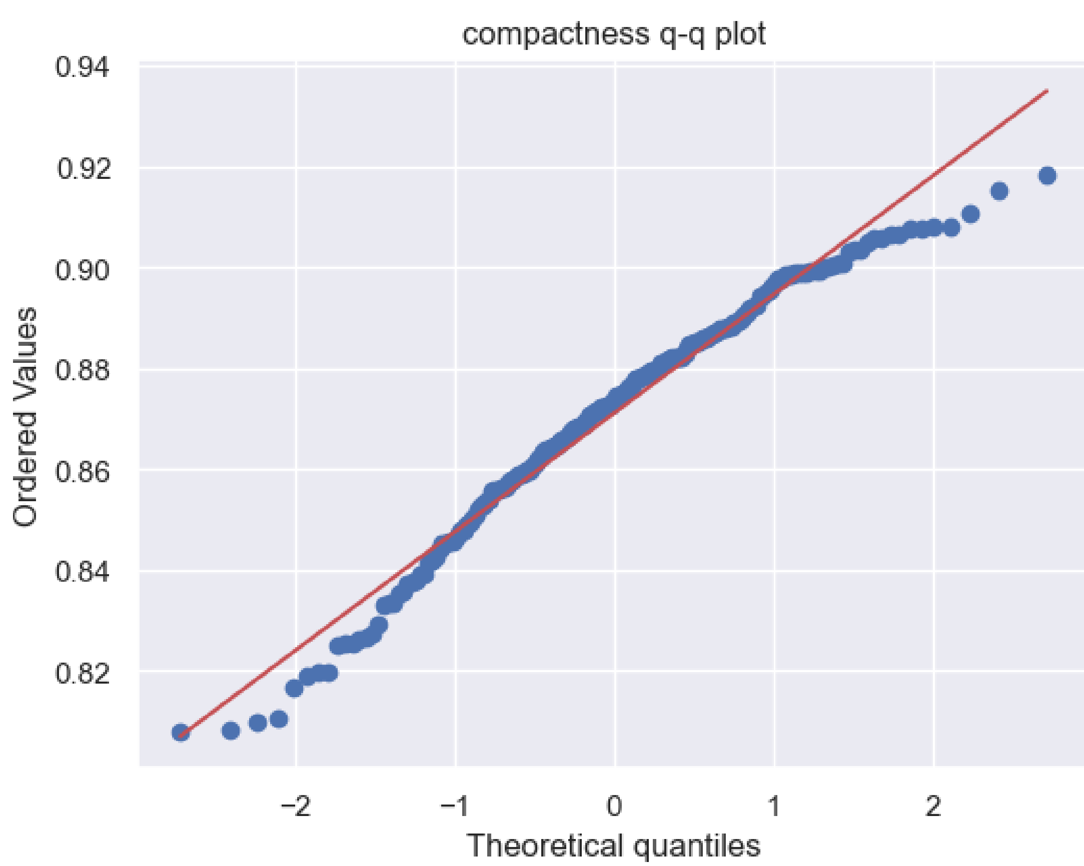
Relevant Code:

```
# F) Create a quantile quantile plot for length of kernel
   groove and compactness. Give a brief analysis for the two
# plots.

groove = SeedDF['length of kernel groove']
comp = SeedDF['compactness']
plt.figure()
qq1 = stats.probplot(groove, plot=plt)
plt.show()
plt.figure()
qq2 = stats.probplot(comp, plot=plt)
plt.show()
```

Output:



length of kernel groove q-q plot
Ordered Values
Theoretical quantiles
6.5
6.0
5.5
5.0
4.5
4.0
−2
−1
0
1
2



compactness q-q plot
Ordered Values
Theoretical quantiles
0.94
0.92
0.90
0.88
0.86
0.84
0.82
−2
−1
0
1
2

Question 4:

Part A

Subpart A.i

The simple matching technique would be a good technique to judge distance. An alternative would be to use Jaccard, but since there is no indication that non-zero attributes are regarded as important. It is more appropriate to use simple matching.

Subpart A.ii

Covariance is a measurement to determine the relationship between two dimensions, such as if the dimensions are correlated and the nature of the correlation like direct or inverse.

If A and B have a covariance of -1, then the two variables have an inverse relationship. If B and C have a covariance of 20, then the two variables have a direct relationship. The signs are important in determining the relationship because each item in the variable is compared to the mean. If both variables in the data object are always above or below the mean, then the covariance stays positive. Otherwise, the covariance is negative. If there is no relationship, the covariance is zero. The scale of the covariance is determined by each items distance from the mean. Therefore, variables with more variance will drive a higher covariance. Therefore the overall value of the covariance does not provide much indication of the strength of the covariance.

Part B

Subpart B.i

Noise is an error from the true value that may result from imperfections in techniques, imperfections in equipment, error, etc.. Noise may be desirable for aspects of privacy. Noise may allow user to extract information from a dataset without violating the privacy of individuals in the dataset. This can be done with the use of differential privacy. For example, a hospital system may want to learn more about cancer patients to infer which populations are most affected by a certain type of cancer without violating an individual cancer patient's privacy.

Subpart B.ii

An outlier is a data point that does not resemble the typical value for a data object within a dataset. Outliers are subjective, but there are some mathematical means to determine outliers in a dataset. For example, a data object could be an outlier if it is distant from the inter-quartile range (IQR) by 1.5 times the IQR for a variable in that object. Outliers are different from noise because noise are modifications applied to the data objects whereas an outlier is one data object that is peculiar in comparison to the dataset. In addition, outliers could be an edge case, or error.

Subpart B.iii

This is an example of noise in the dataset. The noise could have resulted in the measurements being outliers, but it is not guaranteed. Since there is no indication of the value of the measurements, no conclusion can be drawn on whether the values were outliers in the dataset.

Question 5:

Part A

The stratified sampling method with the number drawn being proportional to group size is most appropriate. The goal of the analysis to evaluate based on handedness of the data object. This requires having enough data objects from each group to make that analysis. The decision to have the method be proportional to group size is in order to ensure that the large group is appropriately represented in the analysis.

Part B

The simple random sampling method with replacement would be most appropriate. The analysis does not depend on gathering info from certain text categories so it does not involve taking data from each category. This sampling method accurately represents the text samples available.

Part C

The progressive sampling method would be the most appropriate. This method accounts for increasing the sample size which is required for this scenario.

Question 6:

Part A

Subpart A.i

An appropriate data transformation was determined to be:

$x' = \frac{1}{1+e^{-x}}$

This function is continuous across all real numbers, since no value of x results in a denominator of 0. Horizontal asymptotes occur at both y=0 and y=1 ensuring that all possible values of x fall within the provided range. The function monotonically increases since the limit at $x \to -\infty = 0$, the limit at $x \to \infty = 1$, and the deterministic factor of the function is e^{-x} which is monotonically continuous.

Subpart A.ii

An appropriate data transformation was determined to be:

$x' = \frac{2(x-a)}{b-a} - 1$

$\frac{x-a}{b-a}$ ensures a value between 0 and 1 in a given range. Multiplying that by 2 ensures a value between 0 and 2. Subtracting 1 from this result ensures a value in the domain [-1,1].

New Mean: 0.0429 which was found with direct conversion using the formula
New STD: 0.0679 which was calculated using f(98.6+0.95)-f(98.6)

Part B

Subpart B.i

One hot coding would work be a better word representation since the selected attributes are polarizing and broad. While there can be cases were context doesn't necessarily translate over well for this type of representation, most texts would still follow the model correctly making these scenarios outliers.

Subpart B.ii

Distributed word vectors would be best, since words can have multiple meanings and can become synonymous depending on how their used. Just because two words share a synonym doesn't mean they're necessarily synonymous either, so a word only being allowed one definition wouldn't work.

Question 7:

The code for this problem is titled HW1Q7.py in the main branch of our Github repository.

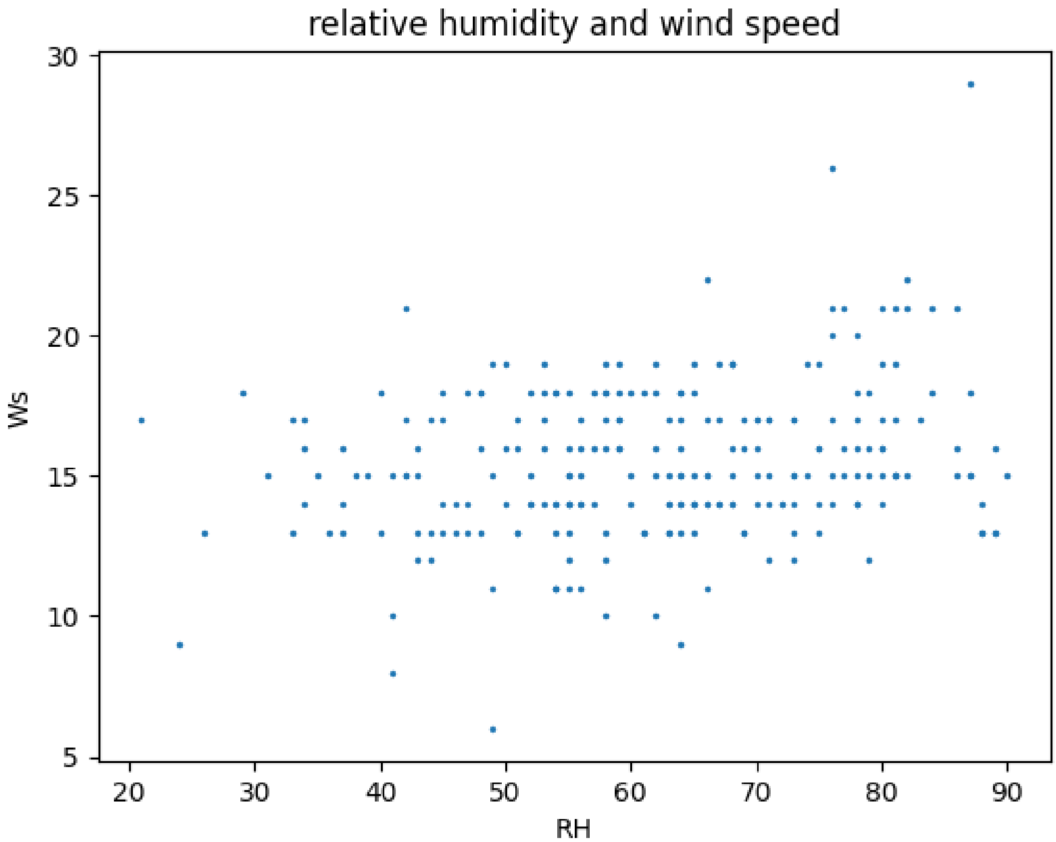
Part A

The original data set was cleaned using the tools available in the pandas package. The relative humidity and wind speed data were plotted as a scatter plot, shown below. From the plot, we can make the general interpretation that the wind speed and relative humidity are not very strongly correlated.

Relevant Code:

```
# Import and clean up the dataset
data = 'Algerian_forest_fires_dataset_UPDATE.csv'
df = pd.read_csv(data, header=1)
df.dropna(axis=0, how='any', thresh=None, subset=None, inplace=
   True)
df.drop(index=123, inplace=True)

# Since we are only concerned with RH and Ws, we'll grab those
   columns and make a new DF with the relevant data
reldat = df[[' RH', ' Ws']].copy()
reldat.rename(columns={' RH': 'RH', ' Ws': 'Ws'}, inplace=True)
reldat['RH'] = reldat['RH'].astype(float)
reldat['Ws'] = reldat['Ws'].astype(float)
plt.figure()
reldat.plot(kind='scatter', x='RH', y='Ws', title='relative
   humidity and wind speed', s=2)
plt.show()
```

Output:


relative humidity and wind speed
30
25
20
15
10
5
Ws
20
30
40
50
60
70
80
90
RH

Part B

The point P is included in the code available in our Github repository. It is also visible in the scatter plots produced in Part D.

0.0.2 Relevant Code:

```
# B) Define point P = (mean(RH), mean(Ws))
P = (reldat['RH'].mean(), reldat['Ws'].mean())
```

Part C

For each distance metric, the 6 closest points were computed using functions built around the metric equations, see code. The resulting points are listed below, along with the associated distance. The leftmost number is the dataframe index associated with the listed data point.

Relevant Code:

```
# Start by getting the list of points
pointlist = []
for index, rows in reldat.iterrows():
    entry = [rows.RH, rows.Ws]
    pointlist.append(entry)

# 1) Euclidean distance function. Takes the point P and the
   pointlist and returns a list of distances that can be
   appended to
# the original dataframe.
def euclidean(point, points):
    euc = []
    x1 = point[0]
    y1 = point[1]
    for item in points:
        x2 = item[0]
        y2 = item[1]
        dist = round(math.sqrt(((x2-x1)**2) + ((y2-y1)**2)), 2)
        euc.append(dist)
    return euc

ED = euclidean(P, pointlist)
reldat['Euclidean Distance'] = ED

# 2) Manhattan distance.

def manhattan(point, points):
    man = []
    x1 = point[0]
    y1 = point[1]
    for item in points:
        x2 = item[0]
        y2 = item[1]
        dist = round(abs(x1-x2) + abs(y1-y2), 2)
        man.append(dist)
    return man
```

```
man = manhattan(P, pointlist)
reldat['Manhattan Distance'] = man

# 3) Minkowski metric for power=7
def minkowski(point, points):
    mink = []
    x1 = point[0]
    y1 = point[1]
    for item in points:
        x2 = item[0]
        y2 = item[1]
        dist = round((((abs(x1-x2)**7) + (abs(y1-y2)**7)) **
           (1/7)), 2)
        mink.append(dist)
    return mink

mink = minkowski(P, pointlist)
reldat['Minkowski Distance'] = mink

# 4) Chebyshev distance
def chebyshev(point, points):
    cheb = []
    x1 = point[0]
    y1 = point[1]
    for item in points:
        x2 = item[0]
        y2 = item[1]
        dist = round(max(abs(x2-x1), abs(y2-y1)), 2)
        cheb.append(dist)
    return cheb

cheb = chebyshev(P, pointlist)
reldat['Chebyshev Distance'] = cheb

# 5) Cosine distance
def cosine(point, points):
    cos = []
    x1 = point[0]
    y1 = point[1]
    for item in points:
        x2 = item[0]
        y2 = item[1]
        num = (x1*y1) + (x2*y2)
        denomL = math.sqrt(x1**2 + x2**2)
```

```
        denomR = math.sqrt(y1**2 + y2**2)
        denom = denomL*denomR
        cossim = num / denom
        dist = round(1 - cossim, 5)
        cos.append(dist)
    return cos

cos = cosine(P, pointlist)
reldat['Cosine Distance'] = cos

# List the closest 6 points for each distance
# Euclidean
smallesteuc = reldat.nsmallest(6, ['Euclidean Distance'])
smallesteuc = smallesteuc.get(['RH', 'Ws', 'Euclidean Distance
   '])
print('Euclidean')
print(smallesteuc.to_string())
print('\n')

# Manhattan
smallestman = reldat.nsmallest(6, ['Manhattan Distance'])
smallestman = smallestman.get(['RH', 'Ws', 'Manhattan Distance
   '])
print('Manhattan')
print(smallestman.to_string())
print('\n')

# Minkowski
smallestmink = reldat.nsmallest(6, ['Minkowski Distance'])
smallestmink = smallestmink.get(['RH', 'Ws', 'Minkowski
   Distance'])
print('Minkowski')
print(smallestmink.to_string())
print('\n')

# Chebyshev
smallestcheb = reldat.nsmallest(6, ['Chebyshev Distance'])
smallestcheb = smallestcheb.get(['RH', 'Ws', 'Chebyshev
   Distance'])
print('Chebyshev')
print(smallestcheb.to_string())
print('\n')

# Cosine
smallestcos = reldat.nsmallest(6, ['Cosine Distance'])
```

```
smallestcos = smallestcos.get(['RH', 'Ws', 'Cosine Distance'])
print('Cosine')
print(smallestcos.to_string())
print('\n')
```

Output:

```
Euclidean
       RH    Ws  Euclidean Distance
222  62.0  15.0                0.50
149  62.0  16.0                0.51
73   63.0  15.0                1.08
35   63.0  14.0                1.78
63   63.0  14.0                1.78
177  63.0  17.0                1.79
```

```
Manhattan
       RH    Ws  Manhattan Distance
222  62.0  15.0                0.53
149  62.0  16.0                0.55
73   63.0  15.0                1.45
24   64.0  15.0                2.45
35   63.0  14.0                2.45
63   63.0  14.0                2.45
```

```
Chebyshev
       RH    Ws  Chebyshev Distance
222  62.0  15.0                0.49
149  62.0  16.0                0.51
73   63.0  15.0                0.96
35   63.0  14.0                1.49
63   63.0  14.0                1.49
177  63.0  17.0                1.51
```

```
Minkowski
       RH    Ws  Minkowski Distance
222  62.0  15.0                0.49
149  62.0  16.0                0.51
73   63.0  15.0                0.96
35   63.0  14.0                1.50
63   63.0  14.0                1.50
177  63.0  17.0                1.52
```

```
Cosine
       RH    Ws  Cosine Distance
12   84.0  21.0          0.00000
85   60.0  15.0          0.00000
88   64.0  16.0          0.00000
212  56.0  14.0          0.00000
238  56.0  14.0          0.00000
31   75.0  19.0          0.00002
```

Part D

Subpart D.i

Plots corresponding to each of the distances are shown on the following pages. The ”closest” points by the given metric are shown in green, point P is shown in red, and the remaining data is shown in blue. Please note that 20 green points are not visible on each plot, as some of the data points within the 20 smallest distances are duplicates, with the markers overlaid directly on top of each other and being indistinguishable.

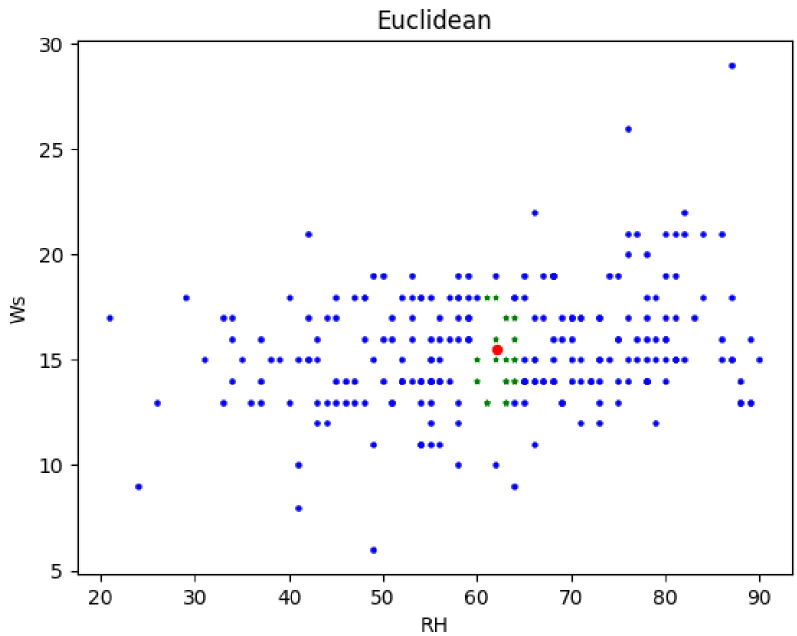
Relevant Code:

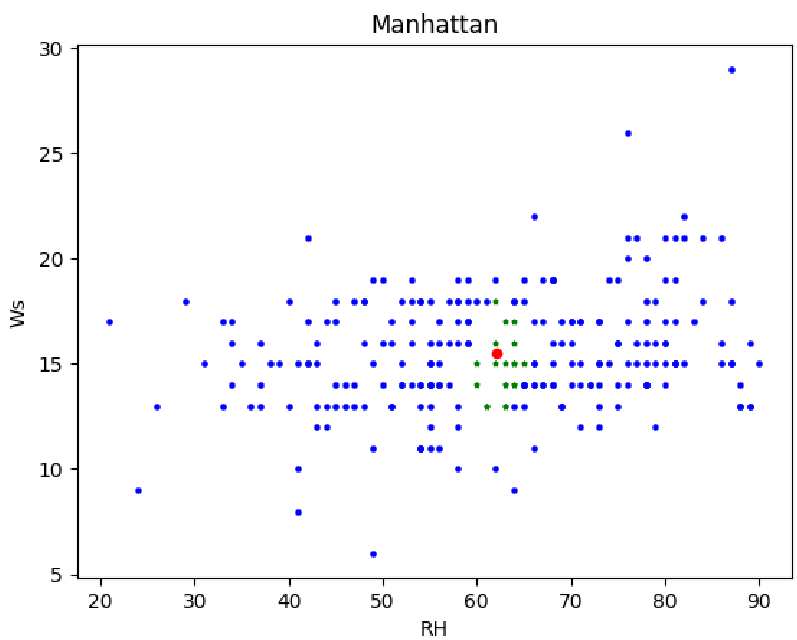
```
# i) Create a plot for each distance measure. Place P and mark
   the 20 closest points. Use different colors
# or shapes to mark them. Make sure the points can be uniquely
   identified.

eucpoints = reldat.nsmallest(20, ['Euclidean Distance'])
manpoints = reldat.nsmallest(20, ['Manhattan Distance'])
minkpoints = reldat.nsmallest(20, ['Minkowski Distance'])
chebpoints = reldat.nsmallest(20, ['Chebyshev Distance'])
cospoints = reldat.nsmallest(20, ['Cosine Distance'])

# Euclidean plot
eucind = eucpoints.index
todrop = []
for x in eucind:
    todrop.append(x)
eucremain = reldat.drop(todrop)


plt.figure()
Pdf = pd.DataFrame([P], columns=['RH', 'Ws'])
eucplot = eucremain.plot(kind='scatter', x='RH', y='Ws', color
   ='blue', title='Euclidean', s=5)
Pdf.plot(ax=eucplot, kind='scatter', x='RH', y='Ws', color='red
   ')
eucpoints.plot(ax=eucplot, kind='scatter', x='RH', y='Ws',
   color='green', marker='*', s=5)
plt.show()

# Manhattan plot
manind = manpoints.index
todrop = []
for x in manind:
```

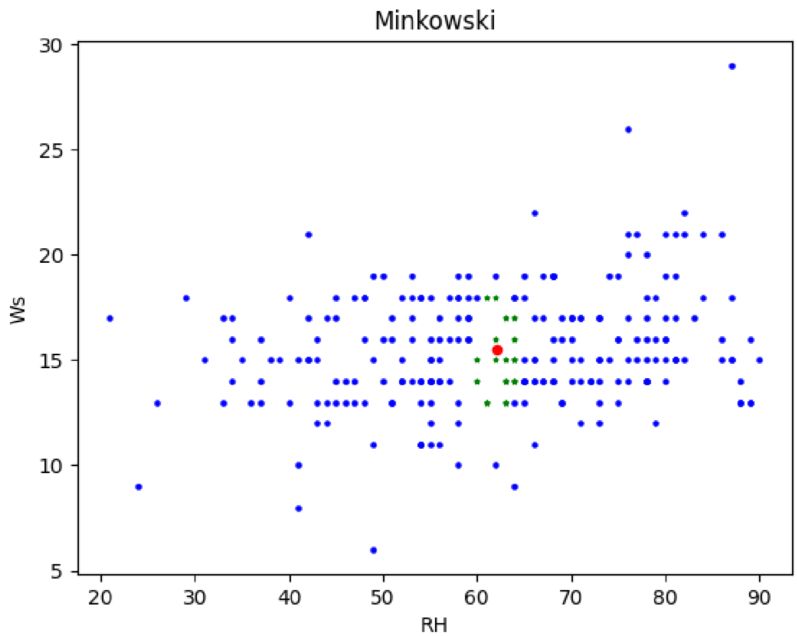
```
    todrop.append(x)
manremain = reldat.drop(todrop)

plt.figure()
manplot = manremain.plot(kind='scatter', x='RH', y='Ws', color
   ='blue', title='Manhattan', s=5)
Pdf.plot(ax=manplot, kind='scatter', x='RH', y='Ws', color='red
   ')
manpoints.plot(ax=manplot, kind='scatter', x='RH', y='Ws',
   color='green', marker='*', s=5)
plt.show()

# Minkowski
minkind = minkpoints.index
todrop = []
for x in minkind:
    todrop.append(x)
minkremain = reldat.drop(todrop)

plt.figure()
minkplot = minkremain.plot(kind='scatter', x='RH', y='Ws',
   color='blue', title='Minkowski', s=5)
Pdf.plot(ax=minkplot, kind='scatter', x='RH', y='Ws', color='
   red')
minkpoints.plot(ax=minkplot, kind='scatter', x='RH', y='Ws',
   color='green', marker='*', s=5)
plt.show()

# Chebyshev
chebind = chebpoints.index
todrop = []
for x in chebind:
    todrop.append(x)
chebremain = reldat.drop(todrop)

plt.figure()
chebplot = chebremain.plot(kind='scatter', x='RH', y='Ws',
   color='blue', title='Chebyshev', s=5)
Pdf.plot(ax=chebplot, kind='scatter', x='RH', y='Ws', color='
   red')
chebpoints.plot(ax=chebplot, kind='scatter', x='RH', y='Ws',
   color='green', marker='*', s=5)
plt.show()

# Cosine
```

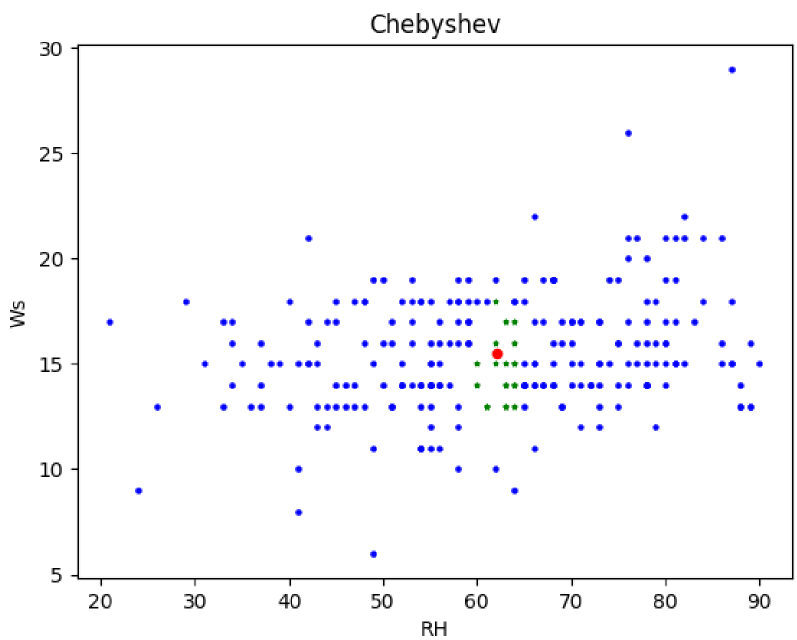
```
cosind = cospoints.index
todrop = []
for x in cosind:
    todrop.append(x)
cosremain = reldat.drop(todrop)

plt.figure()
cosplot = cosremain.plot(kind='scatter', x='RH', y='Ws', color
   ='blue', title='Cosine', s=5)
Pdf.plot(ax=cosplot, kind='scatter', x='RH', y='Ws', color='red
   ')
cospoints.plot(ax=cosplot, kind='scatter', x='RH', y='Ws',
   color='green', marker='*', s=5)
plt.show()
```

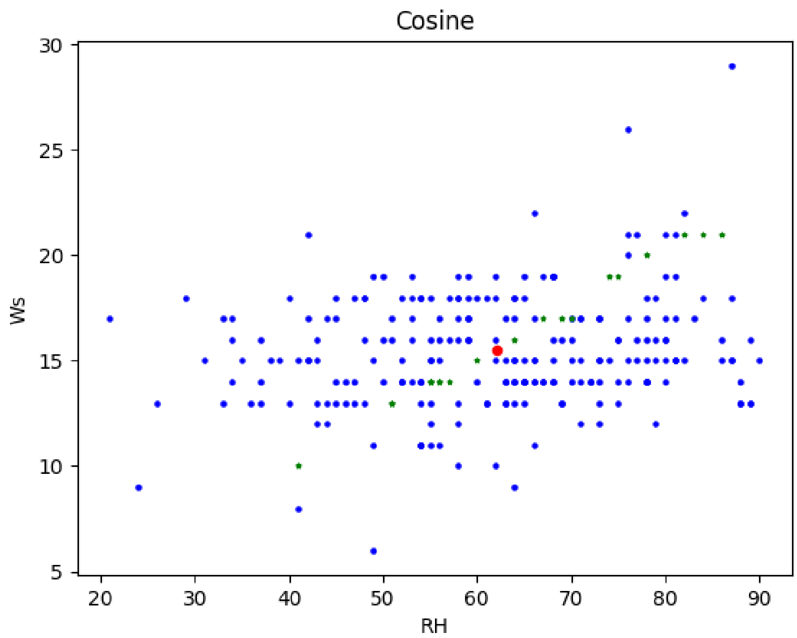
Output:






Minkowski
Ws
RH
30
25
20
15
10
5
20
30
40
50
60
70
80
90


Chebyshev
Ws
RH
30
25
20
15
10
5
20
30
40
50
60
70
80
90



Subpart D.ii

The set of points is not exactly the same across all the distance measures. However, 4 out of 5 of the measures do select very similar points (with the exception being the cosine distance). Some slight variation is to be expected as the distances are all calculated differently. It is also expected that the cosine distance would not be similar to the other distances, as it takes a fundamentally different approach to determining distance. The cosine distance is looking at angular distance, which is why points nearest to a certain angle on the plot are identified as the nearest points, rather than being in the closest rectangular proximity to the point P.