FINAL PROJECT REPORT

Nathaniel B. Moody | INFO-I590, Python

SUMMARY

The following report is submitted in fulfillment of the final project for INFO-I590, Python. A k-means clustering algorithm was used on the Wisconsin Breast Cancer dataset, in an attempt to correctly classify benign and malignant cells. The algorithm was successful, overall, with an error rate of 2.38, though the implementation was extremely inefficient. The code for this script can be found appended to this report.

PHASE 1 – DATA ACQUISITION AND EXPLORATION

In phase 1 of the project, the data was acquired from the UCI Machine Learning Repository, at the following url:

https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

The data was downloaded and read into the program, then cleaned by imputing any missing values with the column mean. Several exploratory plots were created (see Appendix B), and basic descriptive statistics for the data were calculated as follows:

Column	Mean	Median	Standard Deviation	Variance
A1	4.417739628	4	2.815740659	7.928395456
A2	3.134477825	1	3.05145911	9.3114027
A3	3.207439199	1	2.971912767	8.832265496
A4	2.806866953	1	2.855379239	8.1531906
A5	3.21602289	2	2.214299887	4.903123988
A6	3.54465593	1	3.60185164	12.97333524
A7	3.43776824	3	2.438364252	5.945620227
A8	2.86695279	1	3.053633894	9.324679956
A9	1.589413448	1	1.715077943	2.941492349

PHASES 2-3 – IMPLEMENTATION AND EVALUATION

In phase 2 of the project, the k-means clustering algorithm was implemented, and 1500 iterations were carried out to cluster the dataset into two classifications: benign (mu_2) and malignant (mu_4). These classifications, also carrying the ID for each observation to ensure accurate matching, were consolidated into a single new dataset in preparation for evaluation.

The first twenty observations of this new dataset:

ID	A2	A3	A4	A5	A6	A7	A8	A9	A10	CLASS	ID	Predicted Class
1000025	5	1	1	1	2	1	3	1	1	2	1000025	4
1002945	5	4	4	5	7	10	3	2	1	2	1002945	2
1015425	3	1	1	1	2	2	3	1	1	2	1015425	4
1016277	6	8	8	1	3	4	3	7	1	2	1016277	2
1017023	4	1	1	3	2	1	3	1	1	2	1017023	4
1017122	8	10	10	8	7	10	9	7	1	4	1017122	2
1018099	1	1	1	1	2	10	3	1	1	2	1018099	4
1018561	2	1	2	1	2	1	3	1	1	2	1018561	4
1033078	2	1	1	1	2	1	1	1	5	2	1033078	4
1033078	4	2	1	1	2	1	2	1	1	2	1033078	4
1035283	1	1	1	1	1	1	3	1	1	2	1035283	4
1036172	2	1	1	1	2	1	2	1	1	2	1036172	4
1041801	5	3	3	3	2	3	4	4	1	4	1041801	4
1043999	1	1	1	1	2	3	3	1	1	2	1043999	4
1044572	8	7	5	10	7	9	5	5	4	4	1044572	2
1047630	7	4	6	4	6	1	4	3	1	4	1047630	2
1048672	4	1	1	1	2	1	2	1	1	2	1048672	4
1049815	4	1	1	1	2	1	3	1	1	2	1049815	4
1050670	10	7	7	6	4	10	4	1	2	4	1050670	2
1050718	6	1	1	1	2	1	3	1	1	2	1050718	4

RESULTS

The algorithm achieved an general level of success in solving the classification problem, with an error rate of 2.38. While successful, this level of error would not be helpful in actual medical classification, due to the high number of type-1 and type-2 errors it generated. The algorithm is also, as implemented in the script that accompanies this report, woefully inefficient and time consuming. Further refinement of accuracy and optimization would be recommended as next steps for improvement.

APPENDIX A – SCRIPT

The following script was executed in Python 3.6, Anaconda.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 11 19:14:16 2017
Author: Nathaniel B. Moody
File: main.py
Notes: Created in fulfillment of INFO-590 Python, Final Project
Compiler: Python 3, Anaconda
"""

import pandas as pd
import numpy.random as npr
import numpy.linalg as npla
# from scipy.spatial import distance (not as fast as numpy)
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages



def getData():
    """
    Downloads the Wisconsin Breast Cancer Data from the UCI Machine Learning
Repository, and
    returns it as a Pandas DataFrame. Translates all "?" to NaN.

    Paramters: none
    Returns: "df" = Pandas DataFrame containing the Wisdoncsin Breast Cancer
Data.
    """
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-
cancer-wisconsin/breast-cancer-wisconsin.data"
    df = pd.read_csv(url, na_values = "?", names = ["ID", "A2", "A3", "A4",
                                                    "A5", "A6", "A7", "A8",
                                                    "A9", "A10", "CLASS"])
    return df



def imputeNaN(df):
    """
    Takes a Pandas Dataframe as an argument, imputes missing values using the
    column-mean, and returns the DataFrame.

    Parameters: "df" = Pandas DataFrame.
    Returns: "imputedDF" = Pandas DataFrame with missing values filled.
    """
    imputedDF = df.fillna(df.mean())
    return imputedDF
```

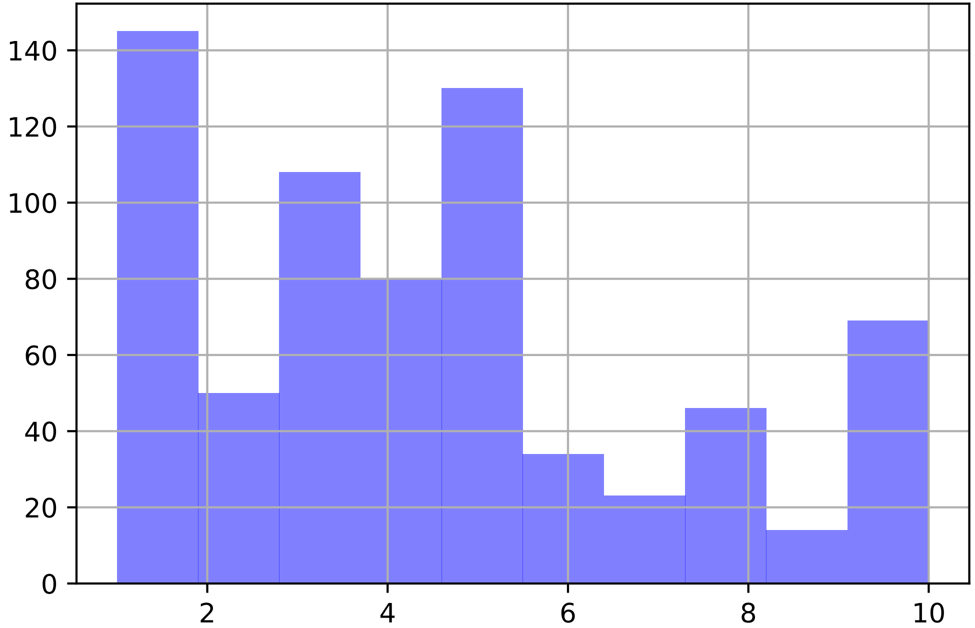
```
def basicPlots(df):
    """
    Takes a Pandas DataFrame and generates a basic set of exploratory plots.

    Parameters: "df" = Pandas DataFrame.
    Returns: None.
    """
    with PdfPages("Histogram_plots.pdf") as pdf:
        df.A2.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A3.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A4.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A5.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A6.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A7.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A8.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A9.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
        df.A10.hist(bins=10, color="b", alpha=0.5)
        pdf.savefig()
        plt.close()
    print("Plots saved as Histogram_plots.pdf, in the working directory.")



def descStats(df):
    """
    Takes a Pandas DataFrame and calculates basic descriptive statistics for
relevant columns.

    Parameters: "df" = Pandas DataFrame.
    Returns: None.
    """
    stats = pd.DataFrame(columns = ["Column", "Mean", "Median",
"Standard_Deviation", "Variance"])
    for i in range(1, 10):
        mean = df.iloc[:,i].mean()
        median = df.iloc[:,i].median()
        std = df.iloc[:,i].std()
        var = df.iloc[:,i].var()
        name = "A%d" % i
        stats = stats.append({"Column":name, "Mean":mean, "Median":median,
"Standard_Deviation":std, "Variance":var}, ignore_index=True)

    stats.to_csv("Descriptive_Statistics.csv")
    print(stats)
    print("Descriptive Statistics saved to Descriptive_Statistics.csv, in the
working directory.")



def init(df):
    """
    Takes original data and chooses two datapoints at random to serve as the
    initial cluster-means.

    Parameters: "df" = DataFrame containing the original data.
    Returns: "m2" = Series containing the randomly chosen datapoint for mu_2.
             "m4" = Series containing the randomly chosen datapoint for mu_4.
    """
    rand2 = round((npr.random()*699))
    rand4 = round((npr.random()*699))
    m2 = df.iloc[rand2,1:9]
    m4 = df.iloc[rand4,1:9]
    return m2,m4



def assign(df,m2,m4):
    """
    Takes original data and makes predictions for which cluster a point
belongs
    to based on euclidean distance to the two cluster-means.

    Parameters: "df" = DataFrame containing the original data.
                "m2" = Series containing the mean for all datapoints in
cluster 2.
                "m4" = Series containing the mean for all datapoints in
cluster 4.
    Returns: "joinedFrame" = DataFrame containing original data, plus
predictions.
                             the ID is also duplicated, so ensure correct
index-matching.
    """
    distFrame = pd.DataFrame(columns = ["ID", "Predicted_Class"])
    for i in range(0, len(df)):
        ID = df.iloc[i,0]
        d2 = npla.norm(df.iloc[i,1:9] - m2)
        d4 = npla.norm(df.iloc[i,1:9] - m4)
        if d2 < d4:
            distFrame = distFrame.append({"ID":ID, "Predicted_Class":2},
ignore_index=True)
```

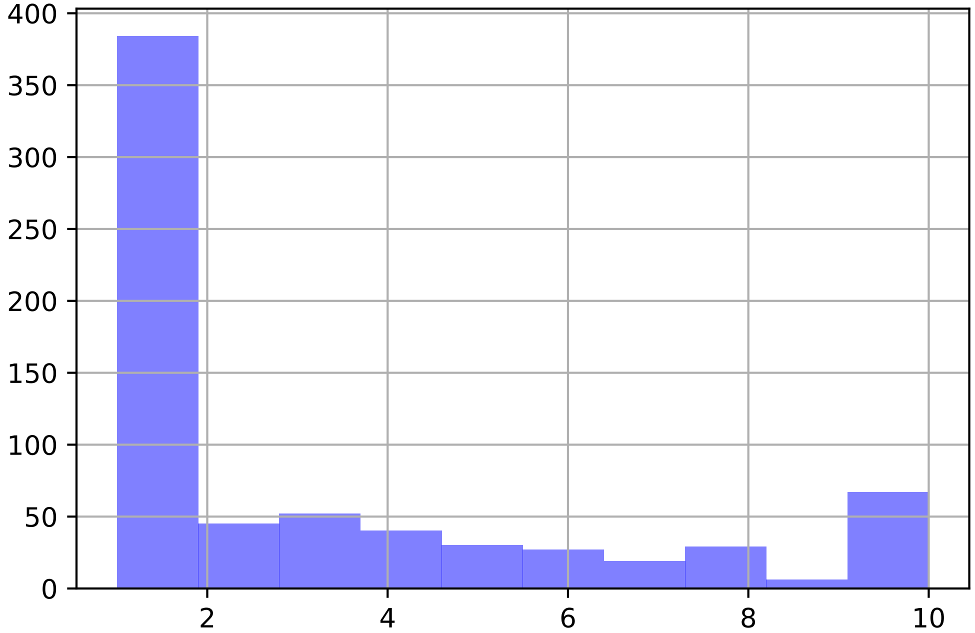
```
        else:
            distFrame = distFrame.append({"ID":ID, "Predicted_Class":4},
ignore_index=True)

    # Join distFrame to df.
    joinedFrame = pd.concat([df, distFrame], axis=1, join='inner')
    return joinedFrame



def recalc(df):
    """
    Recalculates mu_2 and mu_4 based on previous predictions.

    Parameters: "df" = DataFrame containing the predicted clusters for each
datapoint.
    Returns: "m2" = Series containing the mean for all datapoints in cluster
2.
             "m4" = Series containing the mean for all datapoints in cluster
4.
    """
    p2 = df.Predicted_Class == 2.0
    p4 = df.Predicted_Class == 4.0
    cluster2 = df[p2]
    cluster4 = df[p4]

    m2 = cluster2.iloc[:,1:9].mean()
    m4 = cluster4.iloc[:,1:9].mean()
    return m2, m4



def makeReport(m2, m4, df):
    """
    Takes results of analysis and writes it out to a .csv file.

    Parameters: "m2" = Series containing the final means used in clustering
group mu_2.
                "m4" = Series containing the final means used in clustering
group mu_4.
                "df" = DataFrame containing the original data plus
predictions.

    Returns: None.
    """
    path = "Clustering_Results.csv"
    m2.to_csv(path, mode='w', index=True)
    m4.to_csv(path, mode='a', index=True)
    df.head(20).to_csv(path, mode='a', header=True, index=False)
```

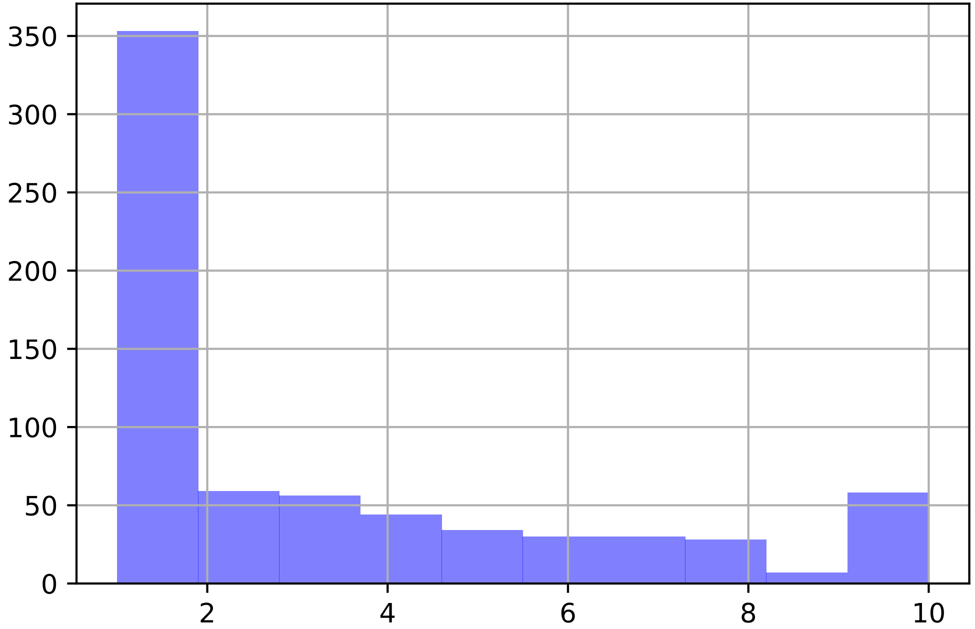
```
def ErrorRate(cluster, actualValue):
    """
    Takes the cluster a datapoint is assigned to (the predicted class),
    and the actual class, and returns 1 if the prediction is an error,
    returns 0 if predictions is accurate.

    Parameters: "cluster" = Integer representing predicted class.
                "actualValue" = Integer representing actual class.
    Returns: 1 if the prediction is an error, 0 if it is accurate.
    """
    if cluster != actualValue:
        return 1
    else:
        return 0


#-------------------------------End of function definitions------------------
------------------------

def main():

    # Acquire and import the Wisconsin Breast Cancer Dataset from the UCI-
MLR.
    print("\nDownloading data from UCI-MLR...")
    try:
        bdata = getData()
        print("Data downloaded/imported succesfully. Sample:")
        print(bdata.head())
    except:
        print("Data not acquired. Verify data source and restart script.")
        return

    # Impute missing values in the data.
    print("\nImputing missing data using column-means...")
    try:
        bdata = imputeNaN(bdata)
        print("Missing values succesfully imputed.")
    except:
        print("Values not imputed. Script may not process data correctly
while NaN values are present.")

    # Generate and save histogram plots.
    print("\nCreating exploratory histogram plots...")
    basicPlots(bdata)
    print("Succesfully created/saved histograms plots.")

    # Calcuate descriptive statistics for each column in the Dataset.
    print("\nCalculating descriptive statistics for the dataset.")
    descStats(bdata)
    print("Descriptive stats calculated.")

    # Initialize the k-means algorithm.
    print("\nInitializing k-means clustering algorithm...")
    m2, m4 = init(bdata)

    # Assign groups using randomly selected means from initialization.
    predictions = assign(bdata, m2, m4)

    # Recalculate the cluster-means (m2, m4) based on the predictions,
    # then predict again based on the new cluster-means.
    # Iterate through this process 1500 times.
    for i in range(1500):

        # Recalculate m2 and m4 based on the predictions.
        m2, m4 = recalc(predictions)
        # Recalculate predictions based on new m2 and m4.
        predictions = assign(bdata, m2, m4)
        print("Completed recalculation #", i, sep="")

    # Print the results of the k-means algorithm.
    print("---------------------------------Final Mean--------------------")
    print("mu_2:\n", m2)
    print("mu_4:\n", m4)
    print("\n\n")
    print("---------------------------------Cluster assignment--------------
------")
    print(predictions.iloc[:,[0,10,12]].head(20))

    # Write the results of the calculation out to a csv file.
    print("\nWriting results to csv...")
    makeReport(m2, m4, predictions)
    print("Results written to Clustering_Results.csv, in the working 
directory.")

    # Calculate the error rate for each cluster.
    print("\nCalculating the Error Rate for the clustering analysis...")
    errorPred2 = 0
    errorPred4 = 0
    totalPred2 = 0
    totalPred4 = 0

    for i in range(0, len(predictions)):

        # Accumulate the number of predictions and errors for each cluster. 
(index 10=CLASS, 12=Predicted_CLass)
        if predictions.iloc[i,12] == 2:
            totalPred2 += 1
            errorPred2 += ErrorRate(2, predictions.iloc[i,10])

        if predictions.iloc[i,12] == 4:
            totalPred4 += 1
            errorPred4 += ErrorRate(4, predictions.iloc[i,10])

    # Calculate error rates and total error.
    errorB = errorPred4/totalPred2
    errorM = errorPred2/totalPred4
    totalError = errorB + errorM
    print("Total Error Rate for Benign Classification:", round(errorB, 2))
    print("Total Error Rate for Malign Classification:", round(errorM, 2))
    print("The total Error Rate for this analysis is:", round(totalError, 2))


# Run the script.
if __name__ == "__main__":
    main()
```

APPENDIX B – EXPLORATORY PLOTS

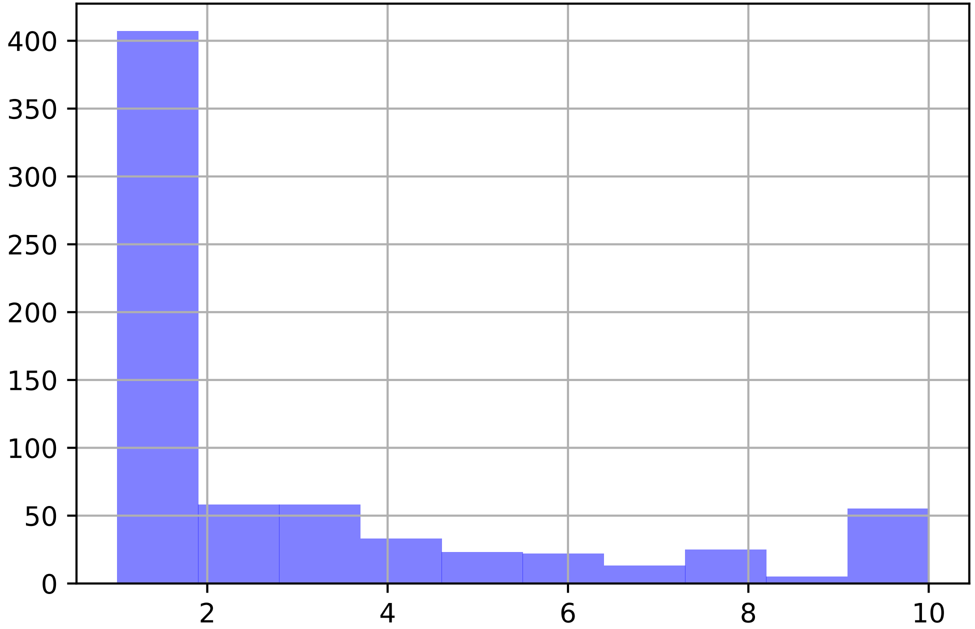
Attached are the exploratory plots that were generated during phase 1 of this project. Each plot is a histogram representing the dataset's column values A2-A10, in order.

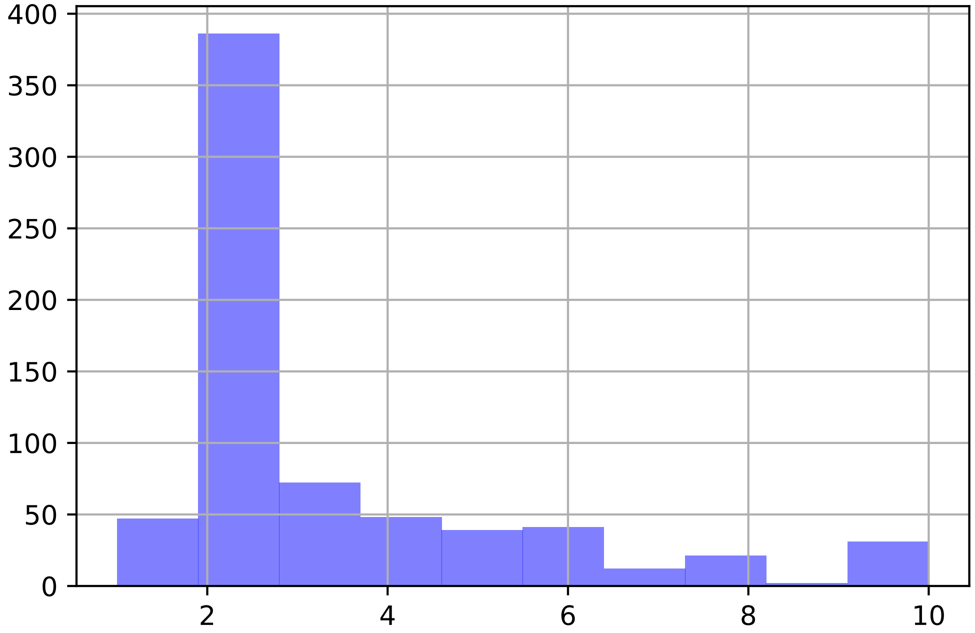

0
20
40
60
80
100
120
140
2
4
6
8
10

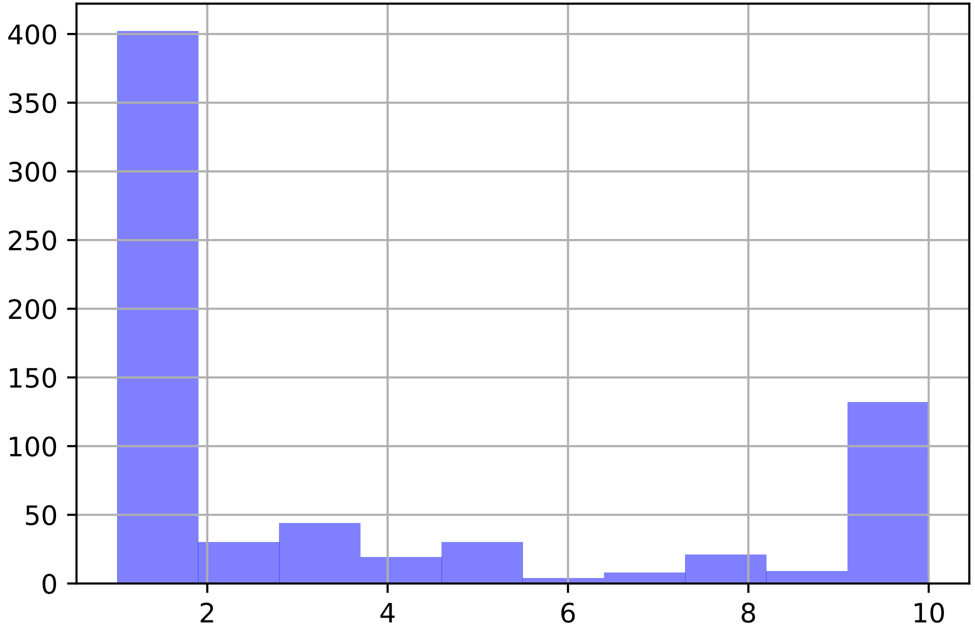

400
350
300
250
200
150
100
50
0
2
4
6
8
10

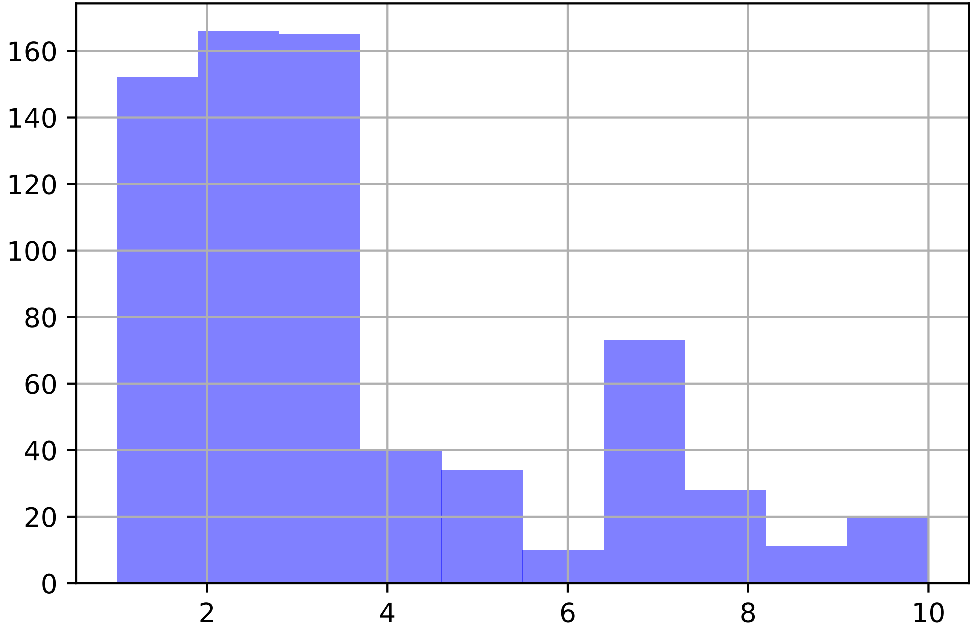


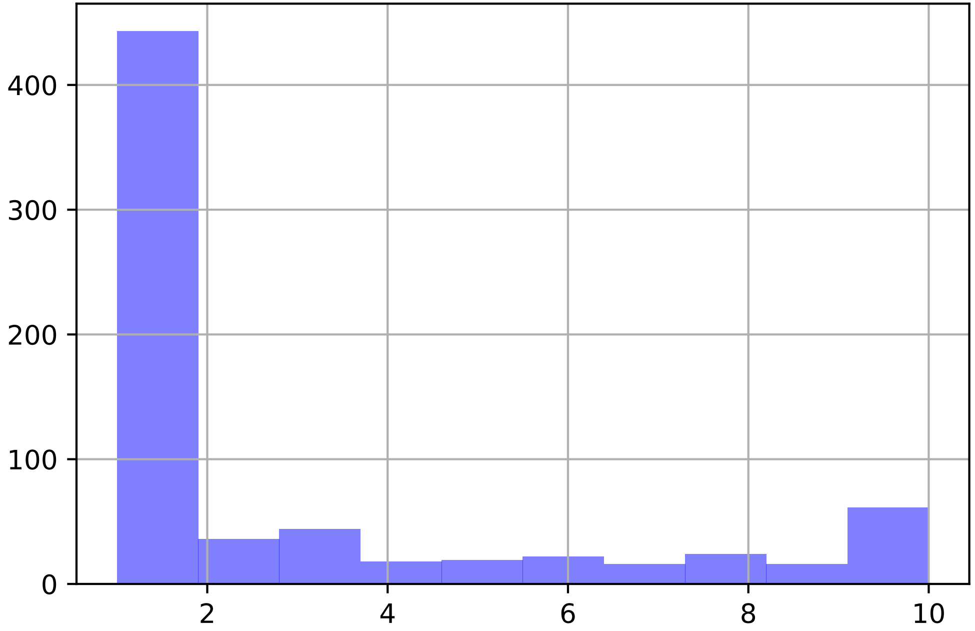
0
50
100
150
200
250
300
350
2
4
6
8
10

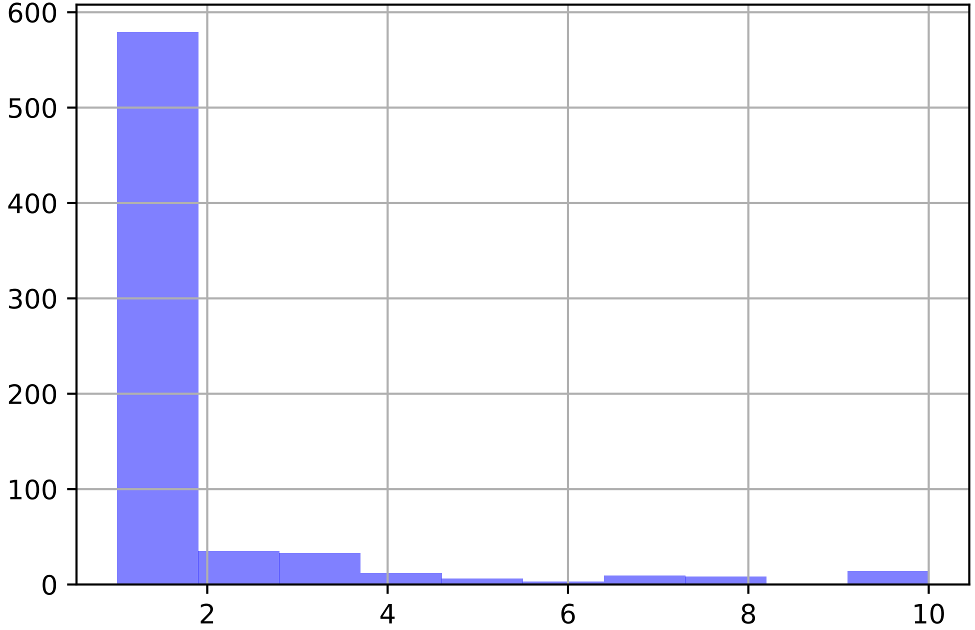

400
350
300
250
200
150
100
50
0
2
4
6
8
10


400
350
300
250
200
150
100
50
0
2
4
6
8
10


0
50
100
150
200
250
300
350
400
2
4
6
8
10


0
20
40
60
80
100
120
140
160
2
4
6
8
10


0
100
200
300
400
2
4
6
8
10



0
100
200
300
400
500
600
2
4
6
8
10