Quarter 1 Project FINAL REPORT: Predicting the Walkability Index of Regions in America

By Arnav Gupta and Raghav Kamineni (Dr. Yilmaz, Period 5)

September 23, 2024 - October 22, 2024

**Part 1: Statement/Project Goal**

The U.S.Environmental Protection Agency rates every Census block group (region of land) based on a number of factors, culminating in its Walkability Index (WI), or likelyhood of walking being used as a significantly used mode of transportation in that region. With growing socioeconomic inequality, walking has become one of the major modes of transportation in the modern world, and the U.S. government collected data from the Smart Location Database (a national geographic census-like database consisting of almost 100 factors) to determine the WI of regions across America. The various factors span across 8 broad measures of quality for each region, and they are listed below.

1. Housing Prices
2. Population and Land Density
3. Diversity of Land Utilization
4. Neighborhood Layouts and Designs
5. Location Accessibility
6. Transit Service
7. Employment
8. Individual Demographics

Based on these listed measures (and dozens of sub measures for each of these broader measures), a region’s WI is calculated from a range of 1-20 and utilized as a function of the given region’s walkability as a usable mode of transportation. However, given the classification nature of the project, we will discretize these values into classifications that we’ll cover below.

The WI helps people make better decisions on where to live, and given that almost 30% of individuals walk to work (and 50% for leisure), it’s important for the population to know which region they should live in if they are in the walking population. In this project, we will look at all the attributes available in the WI dataset and use them to train a classification model where we predict the region’s WI classification. After selecting the most informational attributes and testing which of our models attained the highest accuracy, we’ll be able to identify which attributes were most significant to a region’s WI and simplify the results for the population so they can make more informed decisions on the region to live in.

**Part 2: Description of the Dataset**

We found our dataset named ‘Walkability Index’ (found [here](https://catalog.data.gov/dataset/walkability-index3)) from the U.S. government’s public data catalog. After downloading the dataset as a CSV file and converting it into an ARFF file to manipulate on WEKA, we analyzed the characteristics of this dataset.

The complete dataset had 220,740 instances and 117 attributes including the class attribute, thus the dimension of the dataset is 116D. While the class attribute didn’t have any missing values, there were eleven other attributes with missing values, and the number of missing attributes ranged from 1-53,031 values. However, we noticed that some attributes had an abnormally high number of seemingly default values (such as zero), and we’ll make sure to sort that issue out during the preprocessing stage. As for the class attribute (which had no missing values), the original class attribute values are quantitative continuous values in the range [1-20); however we plan on binning these values into four equal-width classes for our classification models, and these classes are given below.

1. Least Walkable, [1,5.75)
2. Below Average Walkable, [5.75,10.5)
3. Above Average Walkable, [10.5,15.25)
4. Most Walkable, [15.25-20)

As for the class attribute distribution, the 220,740 values had a mean of 9.542 and a standard deviation of 4.374, which, given the range of [1,20), tells us that the distribution is skewed to the right.

**Part 3: Preprocessing**

On federal agency websites, the datasets are frequently just raw data, and the WI dataset was no exception. There are numerous steps we took in order to preprocess the dataset, thus making it more consistent, reliable, and even potentially increasing the long-term accuracy of our models. Below is a list of steps we took to preprocess this WI dataset in both Python and WEKA.

1. Binning, Discretizing, and Renaming the Class Attribute: using the raw dataset, we declared that the class attribute should be attribute index 115 (‘NatWalkInd’ or national WI) so that we could predict the WI of a given region. First, we opened the CSV file in Excel and changed the name of this attribute to a more user-friendly name like (‘Walkability\_Index’), and then manually shifted the new WI column to the end using Excel’s features so WEKA would also correctly declare it the class attribute. With this attribute being a quantitative continuous variable in the range [1, 20], we decided to bin this attribute into four equal-with bins and then discretize said bins into qualitative data labels (mentioned in dataset description) using the Python in the script below. Our class distribution after binning was 66989 instances for AboveAvgWalkable, 74795 instances for BelowAvgWalkable, 25630 instances for MostWalkable, and 53326 instances for LeastWalkable.

# PYTHON SCRIPT FOR EQUAL WIDTH BINNING AND DISCRETIZING CLASS ATTRIBUTE

import pandas as pd

df = pd.read\_csv("ModifiedDataset.csv")

column = df.pop("Walkability\_Index")

df["Walkability\_Index"] = column

def categorize\_walkability(value):

if 1 <= value <= 5.75:

return 'LeastWalkable'

elif 5.75 < value <= 10.5:

return 'BelowAvgWalkable'

elif 10.5 < value <= 15.25:

return 'AboveAvgWalkable'

elif 15.25 < value <= 20:

return 'MostWalkable'

df["Walkability\_Index"] = df["Walkability\_Index"].apply(categorize\_walkability)

df.to\_csv("ModifiedDataset2.csv", index=False)

1. Removing Default Values: While going through the dataset, we realized that numerous columns (relating to region size, location, and several other variables) had quite a number of default values, specifically the number zero. Going through the dataset, it seemed impossible for instances to have no area, location, employment, and various other factors, cementing our belief that these zeroes were incorrect default values. With these numbers being much less than 70% of the total number of values in a particular attribute, these incorrect default values needed to be treated like missing values, and thus, the process of removing default values and replacing them with WEKA’s placeholder for missing values (the question mark symbol) was carried out by Python in the script below.

# PYTHON SCRIPT FOR REPLACING DEFAULT VALUES WITH MISSING VALUE REPRESENTATION

import pandas as pd

df = pd.read\_csv("ModifiedDataset.csv")

for col in df.columns:

df[col].replace(0, '?', inplace=True)

df[col].replace(0.0, '?', inplace=True)

df[col].replace('0', '?', inplace=True)

df[col].replace('0.0', '?', inplace=True)

df.to\_csv("ModifiedDataset2.csv", index=False)

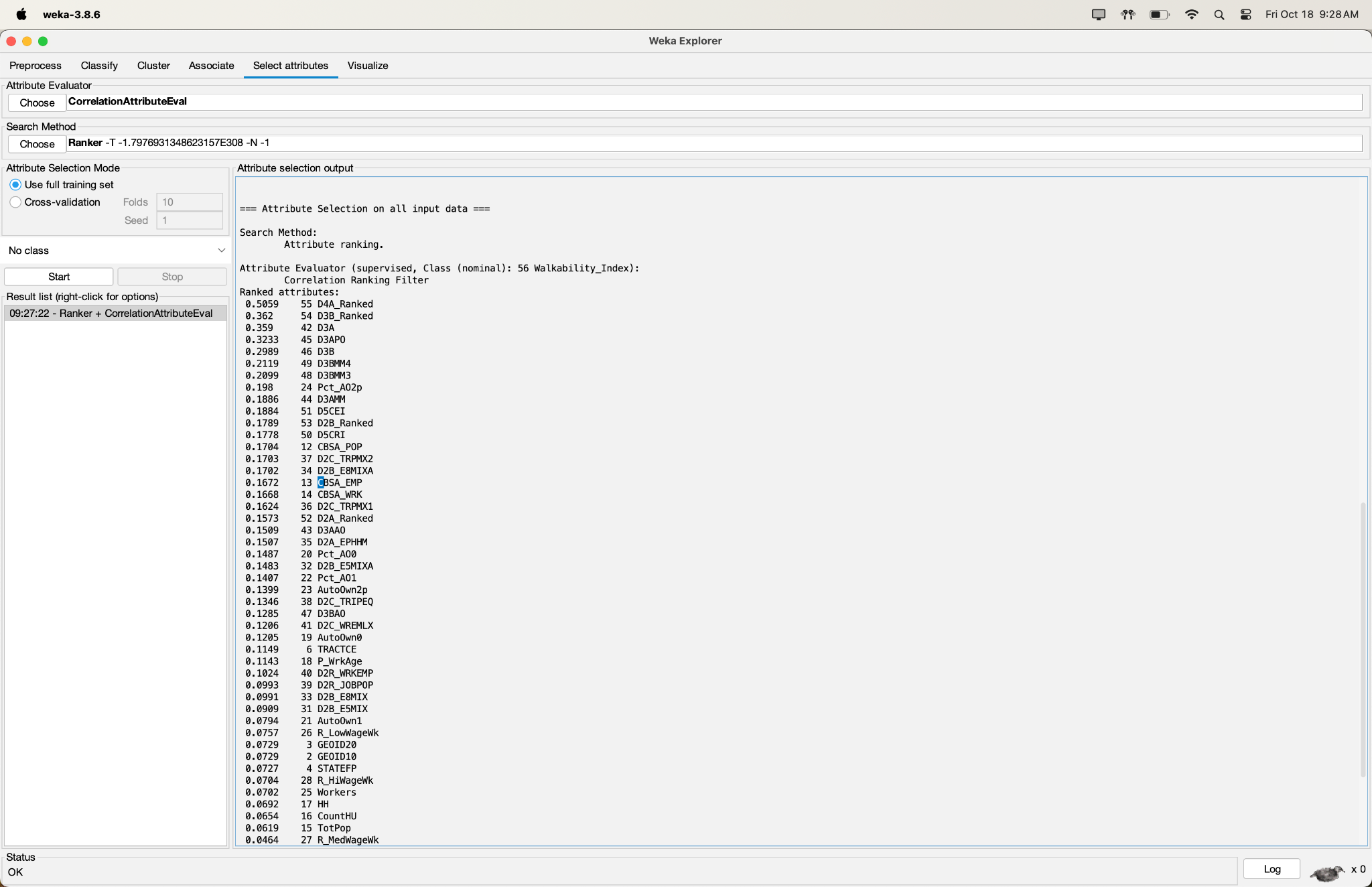
1. Removing Attributes With 70%+ Missing Values: After replacing the default values with missing values, we opened the CSV-file dataset in WEKA and manually went through the dataset to check for missing values within each attribute, and concluded that 60(!) columns (AC\_Total, AC\_Water, AC\_Land, AC\_Unpr, TotEmp, E5\_ Ret, E5\_Off, E5\_Ind, E5\_ Svc, E5\_Ent, E8\_Ret, E8\_off, E8\_ Ind, E8\_Svc, E8\_Ent, E8\_Ed, E8\_HIth, E8\_Pub, E\_LowWageWk, E\_MedWageWk, E\_HiWageWk, D1A, D1B, D1C, D1C5\_RET, D1C5\_OFF, D1C5\_IND, D1C5\_SVC, D1C5\_ENT, D1C8\_RET, D1C8\_OFF, D1C8\_IND, D1C8\_SVC, D1C8\_ENT, D1C8\_ED, D1C8\_HLTH, D1C8\_PUB, D1D, D1\_FLAG, D2A\_JPHH, DA\_WRKEMP, D3BPO4, D4A, D4B025, D4B050, D4C, D4D, D4E, D5AR, D5AE, D5BR, D5BE, D5CR, D5CE, D5DR, D5DRI, D5DE, D5DEI, Shape\_Length, Shape\_Area) needed to be removed due to their high number of now high number of missing values, specifically 70%+. Using WEKA’s remove attribute feature, we simply manipulated the dataset to remove the several above-mentioned attributes and saved our new dataset in WEKA right after.
2. Filling in Missing Values with Mean/Mode: With the dataset rid of attributes with numerous missing values and significantly skewed data, we changed all the missing values to the attribute’s mean or mode using WEKA’s useful ReplaceMissingValues feature in every single attribute index (note that the class attribute had no missing values in the raw dataset, so accidently replacing class attributes was not a concern).

After removing attributes that had 70% or more missing we had 55 attributes to use for the attribute selection algorithm.

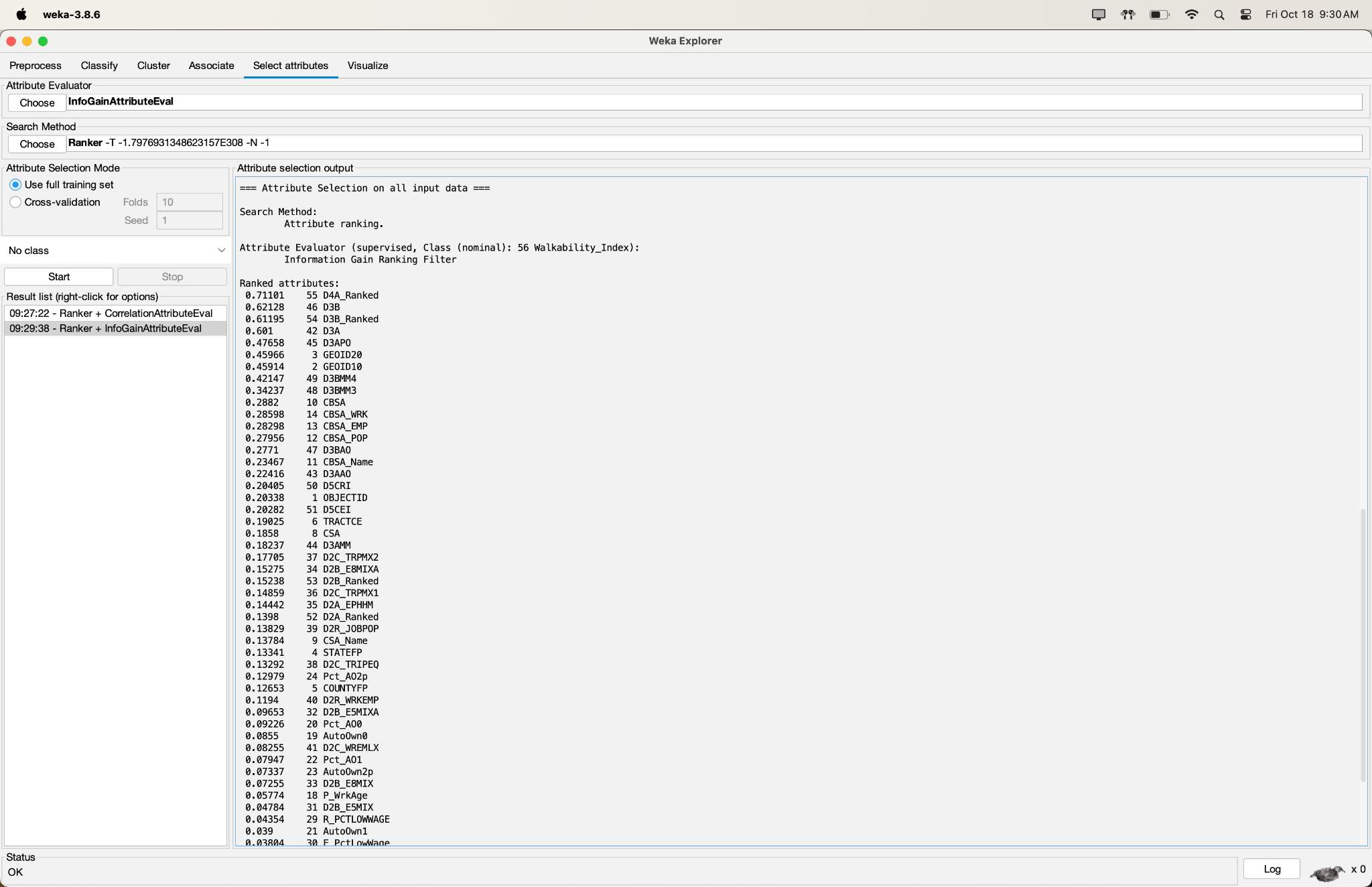
**Part 4: Attribute Selection Algorithms and Building Train/Test/Split Sets**

For this project, we will use four attribute selection algorithms. We’ll declare our thresholds and remaining attributes for each of the five attribute selection algorithms we’ve chosen below.

1. CorrelationAttributeEval

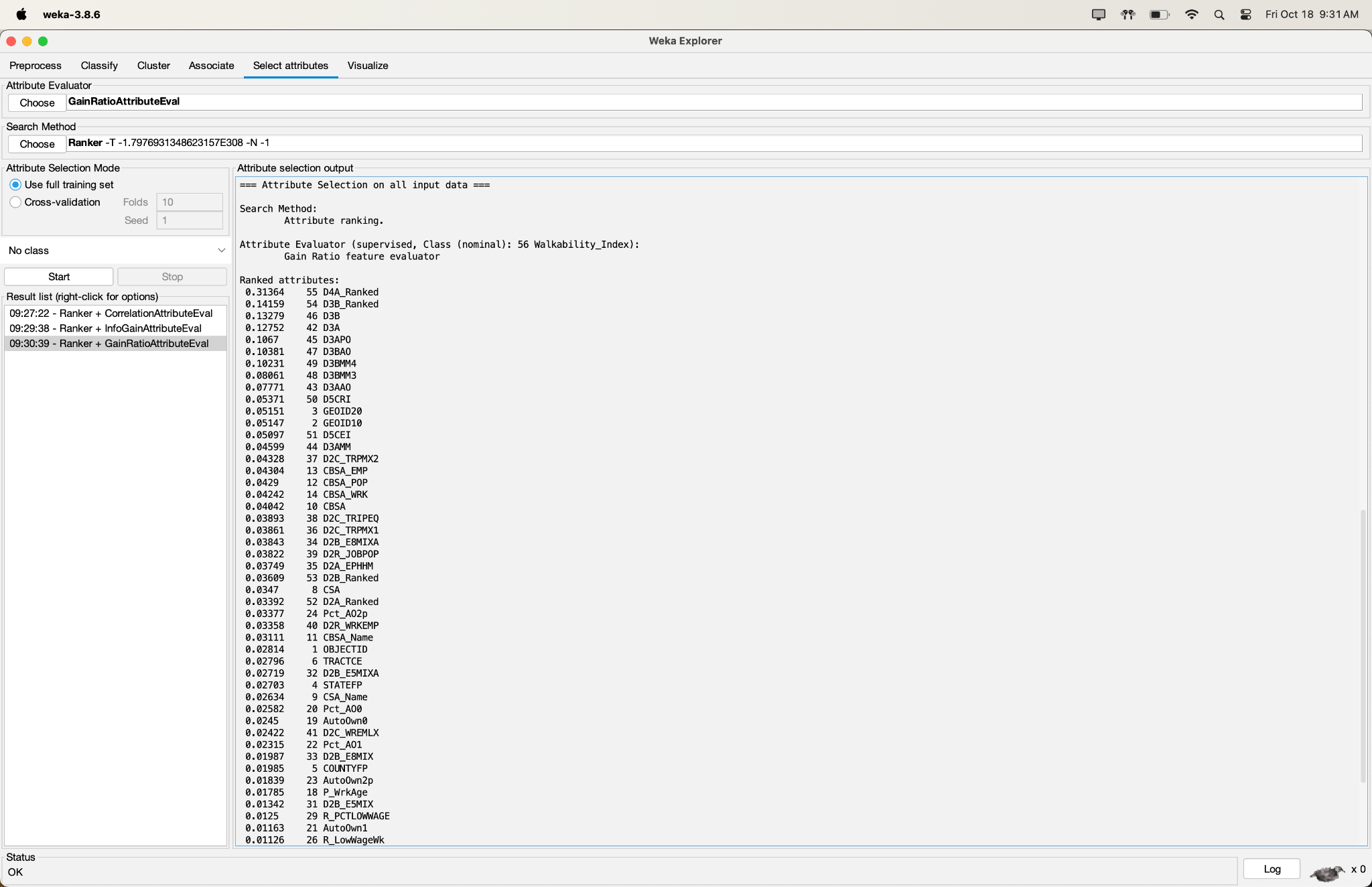
After running the correlation-based attribute selection algorithm, we decided to keep attributes with a correlation coefficient greater or equal to 0.1. The attributes remaining were the following: D4A\_Ranked, D3B\_Ranked, D3A, D3APO, D3B, D3BMM4, D3BMM3, Pct\_AO2p, D3AMM, D5CEI, D2B\_Ranked, D5CRI, CBSA\_POP, D2C\_TRPMX2, D2B\_E8MIXA, CBSA\_EMP, CBSA\_WRK, D2C\_TRPMX1, D2A\_Ranked, D3AAO, D2A\_EPHHM, Pct\_AO0, D2B\_E5MIXA, Pct\_AO1, AutoOwn2p, D2C\_TRIPEQ, D3BAO, D2C\_WREMLX, AutoOwn0, TRACTCE, P\_WrkAge, and D2R\_WRKEMP. 

1. InfoGainAttributeEval

After running the InfoGain-based attribute selection algorithm, we decided to keep attributes with a coefficient of greater or equal to 0.15. The attributes remaining were the following: D4A\_Ranked, D3B, D3B\_Ranked, D3A, D3APO, GEOID20, GEOID10, D3BMM4, D3BMM3, CBSA, CBSA\_WRK, CBSA\_EMP, CBSA\_POP, D3BAO, CBSA\_Name, D3AAO, D5CRI, OBJECTID, D5CEI, TRACTCE, CSA, D3AMM, D2C\_TRPMX2, D2B\_E8MIXA, and D2B\_Ranked. 

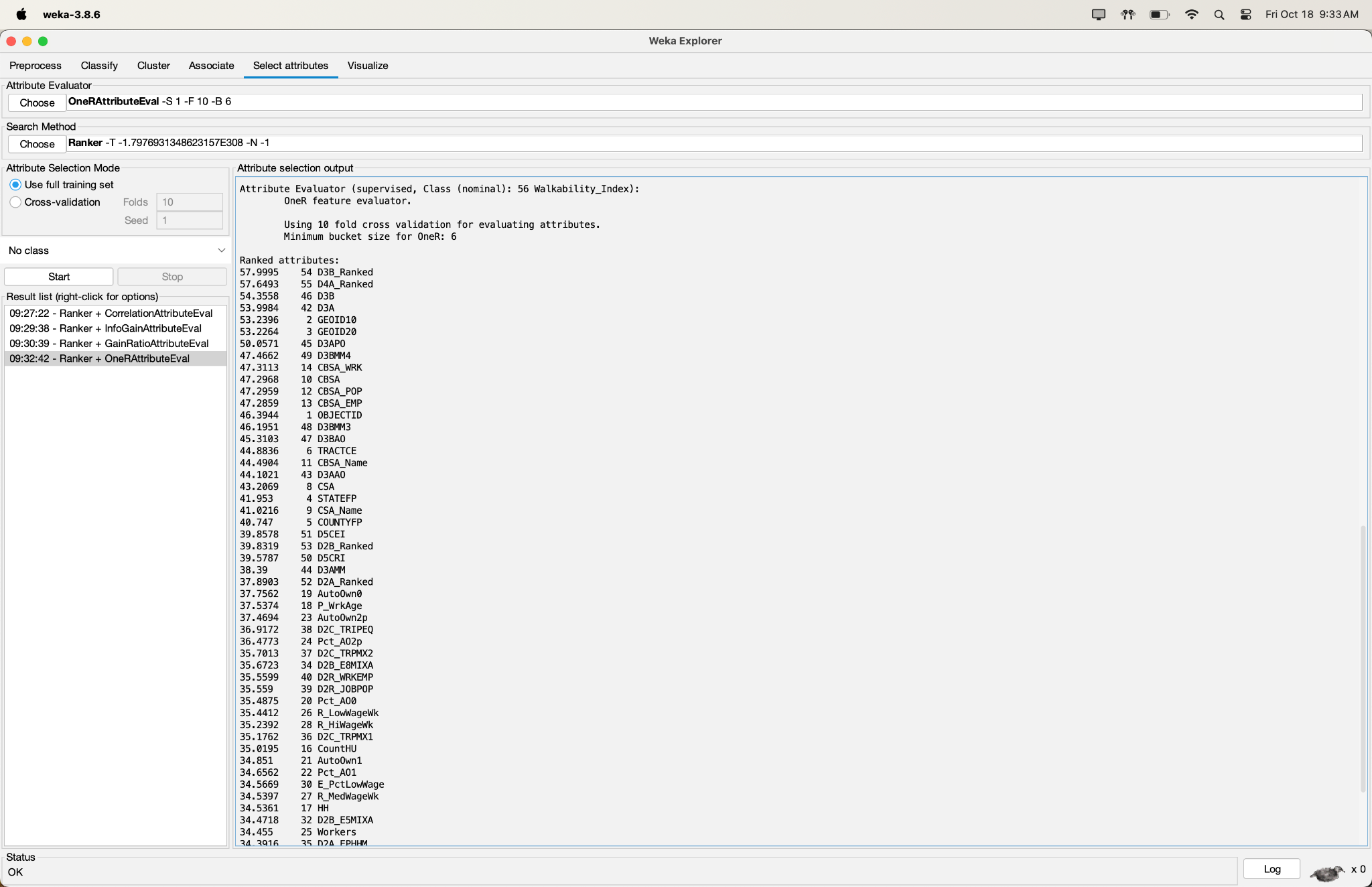
1. GainRatioAttributeEval

After running the GainRatio-based attribute selection algorithm, we decided to keep attributes with a coefficient of greater or equal to 0.03. The attributes remaining were the following: D4A\_Ranked, D3B\_Ranked, D3B, D3A, D3APO, D3BAO, D3BMM4, D3BMM3, D3AAO, D5CRI, GEOID20, GEOID10, D5CEI, D3AMM, D2C\_TRPMX2, CBSA\_EMP, CBSA\_POP, CBSA\_WRK, CBSA, D2C\_TRIPEQ, D2C\_TRPMX1, D2B\_E8MIXA, D2R\_JOBPOP, D2A\_EPHHM, D2B\_Ranked, CSA, D2A\_Ranked, Pct\_AO2p, D2R\_WRKEMP, and CBSA\_Name.



1. OneRAttributeEval

After running the OneR-based attribute selection algorithm, we decided to keep attributes with a coefficient greater than or equal to 36 with a minimum bucket size of 6. The attributes remaining were the following: D3B\_Ranked, D4A\_Ranked, D3B, D3A, GEOID10, GEOID20, D3APO, D3BMM4, CBSA\_WRK, CBSA, CBSA\_POP, CBSA\_EMP, OBJECTID, D3BMM3, D3BAO, TRACTCE, CBSA\_Name, D3AAO, CSA, STATEFP, CSA\_Name, COUNTYFP, D5CEI, D2B\_Ranked, D5CRI, D3AMM, D2A\_Ranked, AutoOwn0, P\_WrkAge, AutoOwn2p, D2C\_TRIPEQ, and Pct\_AO2p.



1. Attribute Selection Algorithm of Our Choice

With the four attribute selection algorithms we’ve chosen, we believe that a better attribute selection algorithm can be found by combining the best attributes in the previous algorithms. In lieu of that, we’ve selected (based on our interpretations of their rankings in the last four attribute selection algorithms) a list of the remaining attributes as follows: D4A\_Ranked, D3B\_Ranked, D3B, D3A, D3APO, D3BMM4, D3BMM3, Pct\_AO2p, D5CEI, D2B\_Ranked, CBSA\_POP, D2C\_TRPMX2, D2B\_E8MIXA, CBSA\_EMP, CBSA\_WRK, D3BAO, D3AAO, D5CRI, D3AMM, D2C\_TRIPEQ, D2R\_WRKEMP, TRACTCE, CBSA\_Name, CSA, AutoOwn0, and P\_WrkAge.

Train/Validation/Test Sets: After completing each of these attribute selection algorithms, we created a system based on our four classifications to then split the data randomly and effectively into train/test split data of 70% training, 15% validation, and 15% testing. First, we looped through each file post attribute selection, and for each file, we used the Scikit library to first use stratified random sampling on the file to split the file into 70% and 30%. Then we used stratified random sampling again on the 30% dataset to split it into two 15% data sets. After that, we saved the 70% set as the training set, 15% as the validation set, and the last 15% as the test set.

# PYTHON SCRIPT FOR STRATIFIED RANDOM SAMPLING SYSTEM

import pandas as pd

from sklearn.model\_selection import train\_test\_split

import os

def stratified\_sample\_and\_save(file\_path,name):

df = pd.read\_csv(file\_path)

X = df.iloc[:, :-1]

y = df.iloc[:, -1]

X\_train\_temp, X\_temp, y\_train\_temp, y\_temp = train\_test\_split(

X, y, test\_size=0.3, stratify=y, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(

X\_temp, y\_temp, test\_size=0.5, stratify=y\_temp, random\_state=42)

train\_df = pd.concat([X\_train\_temp, y\_train\_temp], axis=1)

val\_df = pd.concat([X\_val, y\_val], axis=1)

test\_df = pd.concat([X\_test, y\_test], axis=1)

train\_df.to\_csv(f"{name}\_train.csv", index=False)

val\_df.to\_csv(f"{name}\_val.csv", index=False)

test\_df.to\_csv(f"{name}\_test.csv", index=False)

csv\_files = ['ModifiedDatasetCORR.csv', 'ModifiedDatasetGAINRATIO.csv', 'ModifiedDatasetINFOGAIN.csv', 'ModifiedDatasetONER.csv', 'ModifiedDatasetINDIVIDUAL.csv']

names = ["corr","gainratio","infogain","oneR","indi"]

for file\_path,x in zip(csv\_files, range(5)):

stratified\_sample\_and\_save(file\_path,names[x])

When looking at the class distribution for each training set, we see that the distribution was preserved. For instance, when looking at the Correlation Training set, 30.3472% of the set was labeled AboveAvgWalkable, 33.8840% was labeled BelowAvgWalkable, 24.1577% was labeled LeastWalkable, and 11.6109% was labeled MostWalkable. When we now look at the initial distribution of the entire data set we see that they had a 30.3474% of AboveAvgWalkable, 33.8837% of BelowAvgWalkable, 23.1578% of LeastWalkable, and 11.6109% of MostWalkable. Comparing these values we see that they are very similar showing that the distribution was preserved, and although we only showed the values for the Correlation Training Set, the rest of the data sets also had the same proportions

As our attribute selection process comes to an end, we’ll have the desired train/validation/test split data in order for the next part of our project, which will be to run the classification models and attempt to attain the highest accuracy for the models based on the given attribute selection algorithms and classification model combinations.

Now that we have the final dataset after preprocessing and applying various attribute selection algorithms, we’ll need classification models to train, validate, and test on the given datasets. For this project, we’ll use four classification models, namely the models below:

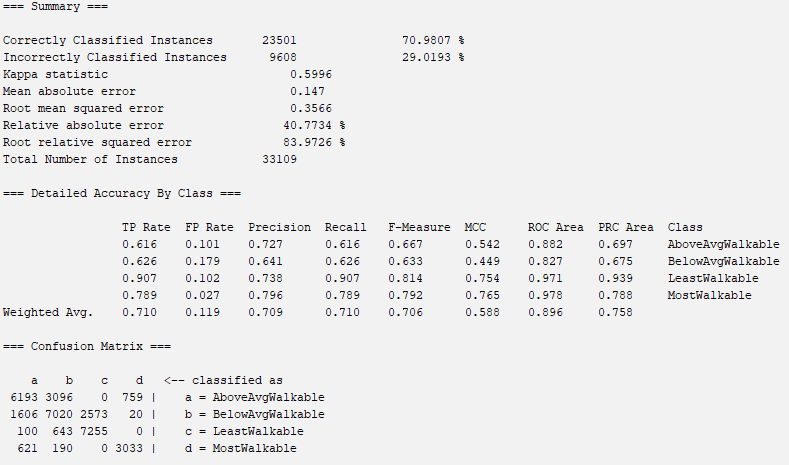
1. NaiveBayes (Bayes)
2. OneR (Rules)
3. J48 (Trees)
4. RandomForest (Trees)

While we have not learned the intricacies of any of these models (with the exception of OneR, which is a one-rule algorithm that independently chooses the best straight forward rule given each instances’ chosen attribute option based on each attribute), we have used the four of these models in our ML labs, and given our preprocessing and attribute selection algorithm success, we hope to achieve 95%+ accuracy for the majority of our 20 models, if not at least one model.

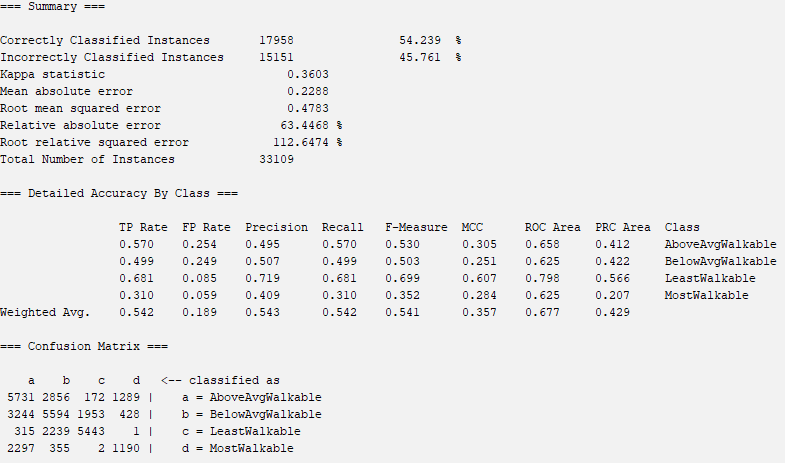
**Part 6: Results and Analysis**

Here are the results of each of our twenty models with each image containing the summary statistics, detailed accuracy by class, and the confusion matrix of the given testing data results.

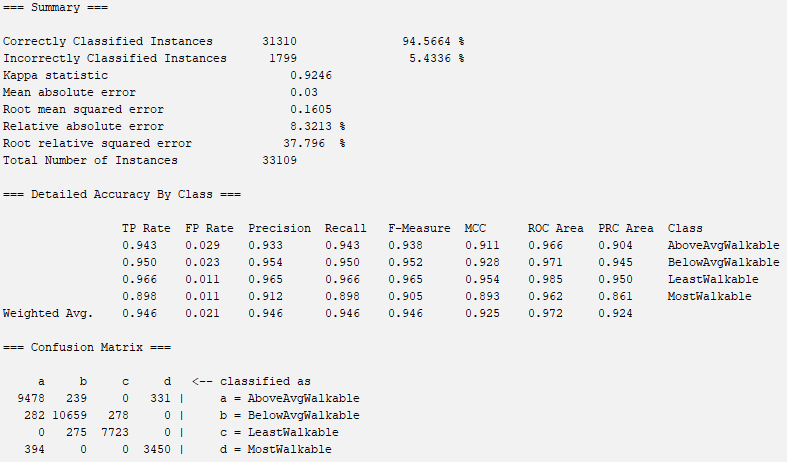
CorrelationAttributeEval with Naive Bayes:



CorrelationAttributeEval with OneR:



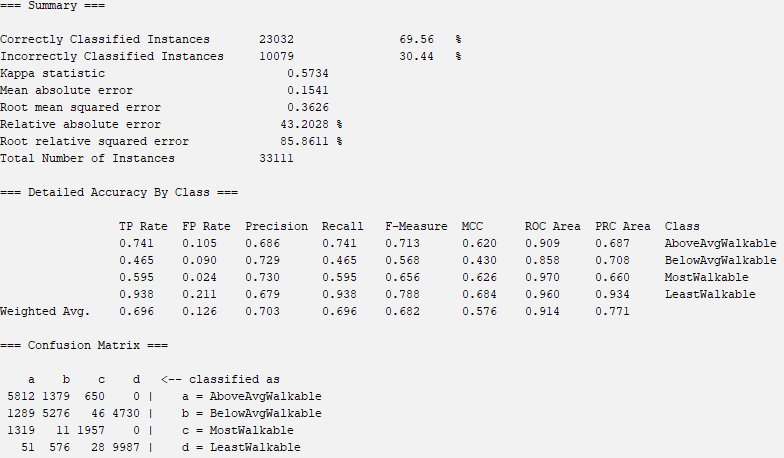
CorrelationAttributeEval with J48:



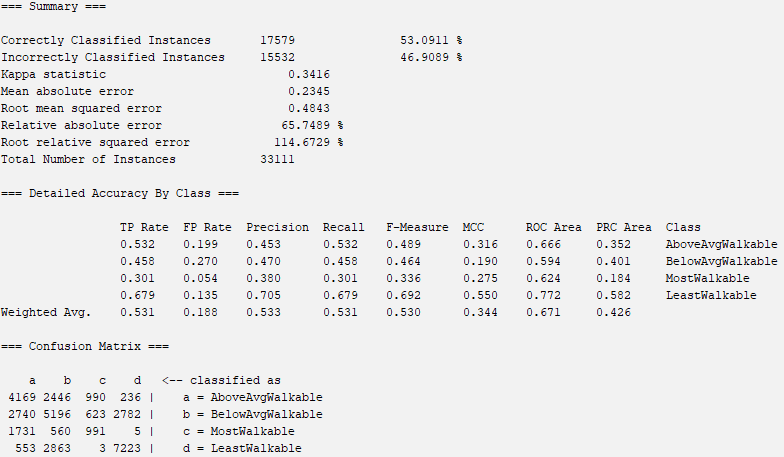
CorrelationAttributeEval with RandomForest:



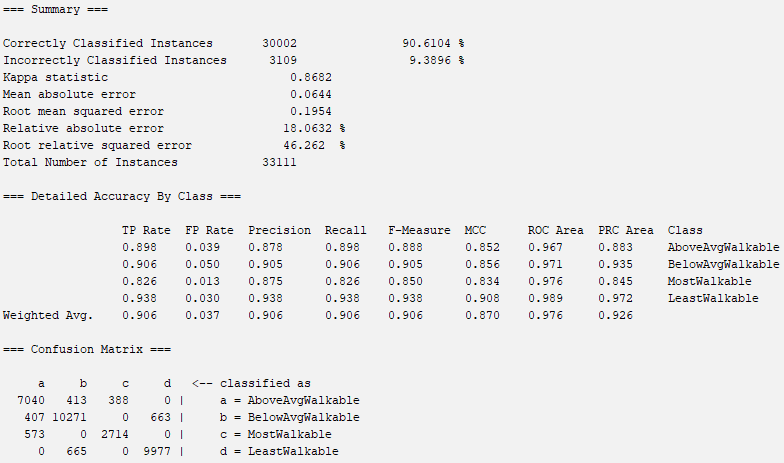
InfoGainAttributeEval with Naive Bayes:



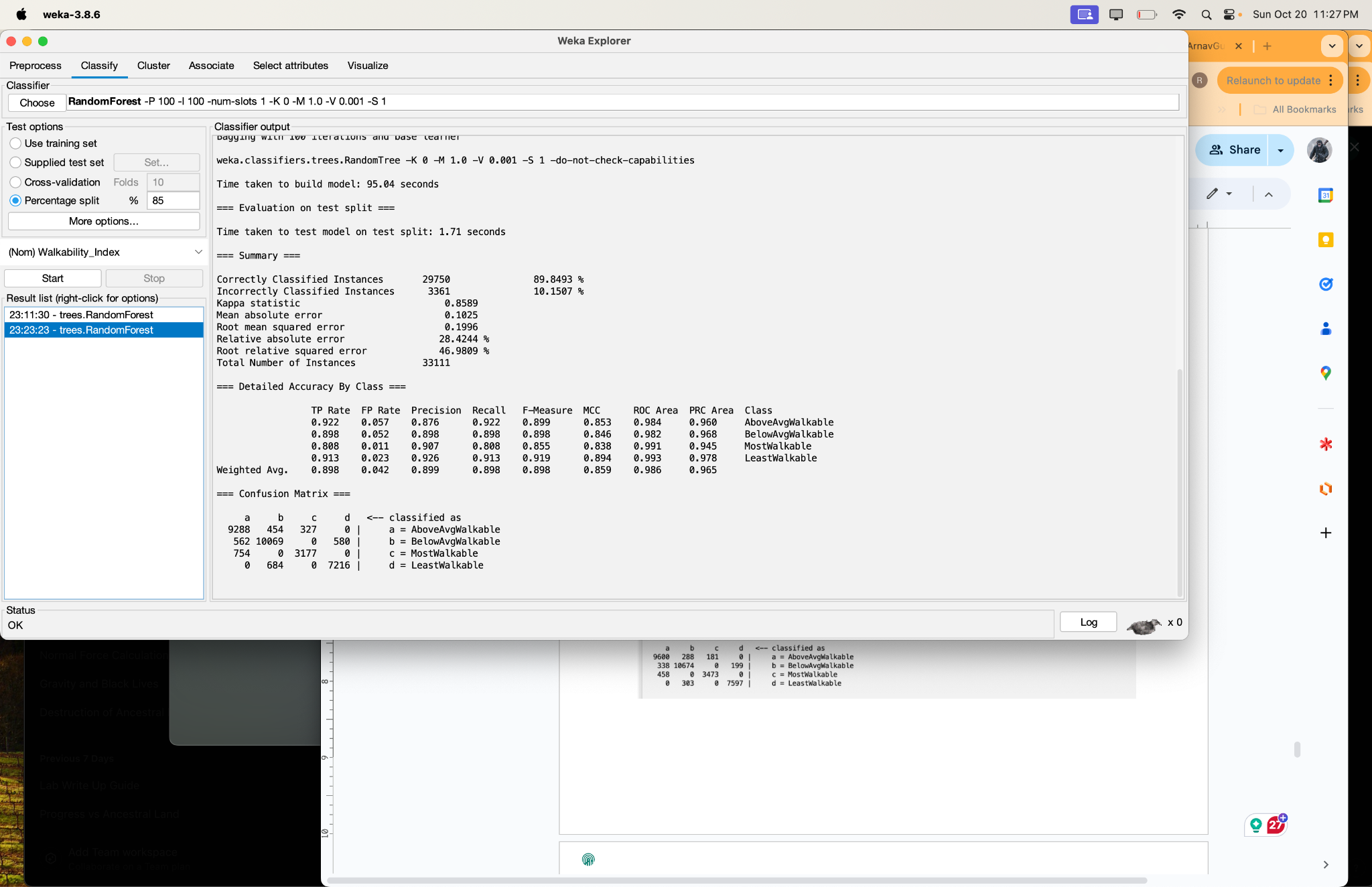
InfoGainAttributeEval with OneR:



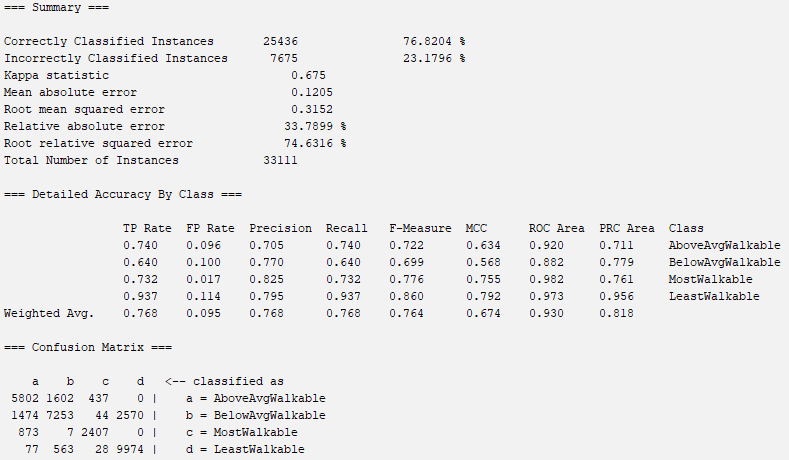
InfoGainAttributeEval with J48:



InfoGainAttributeEval with RandomForest:

****

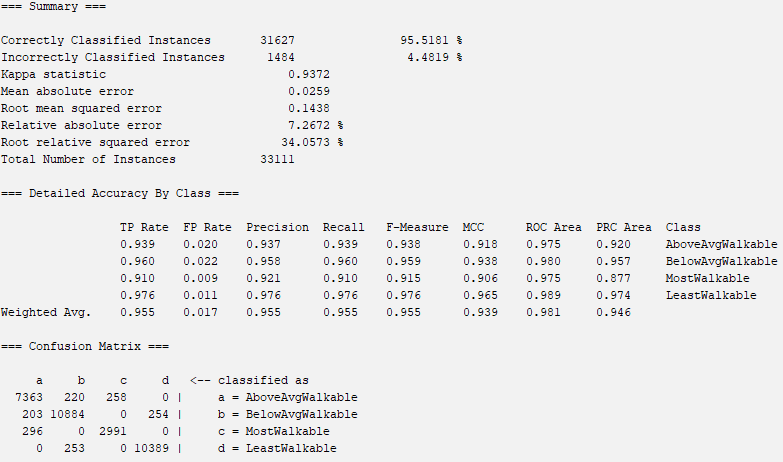
GainRatioAttributeEval with Naive Bayes:



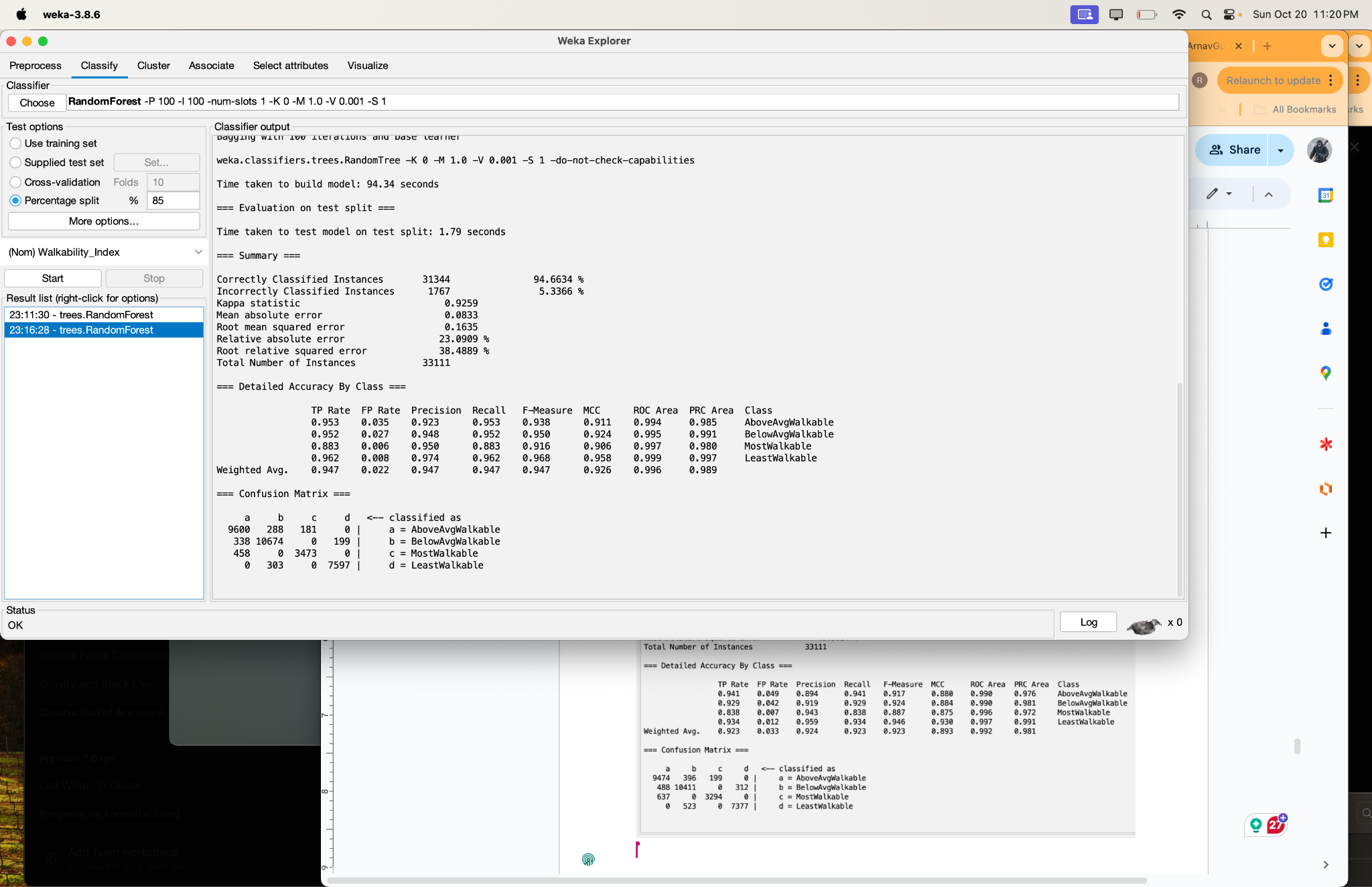
GainRatioAttributeEval with OneR:



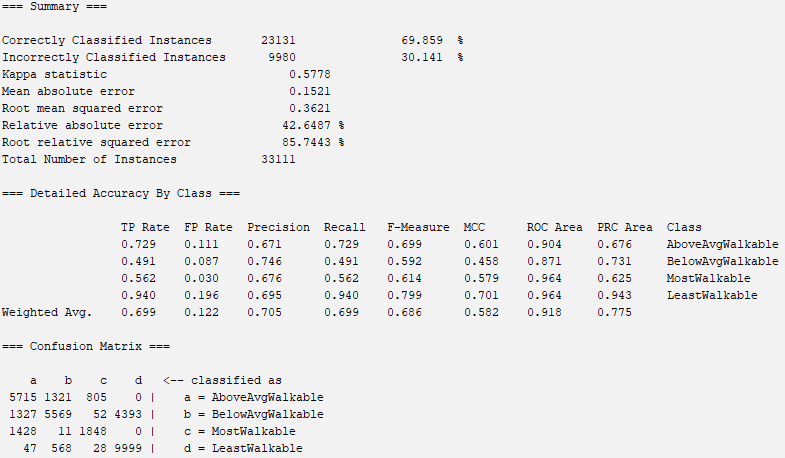
GainRatioAttributeEval with J48:



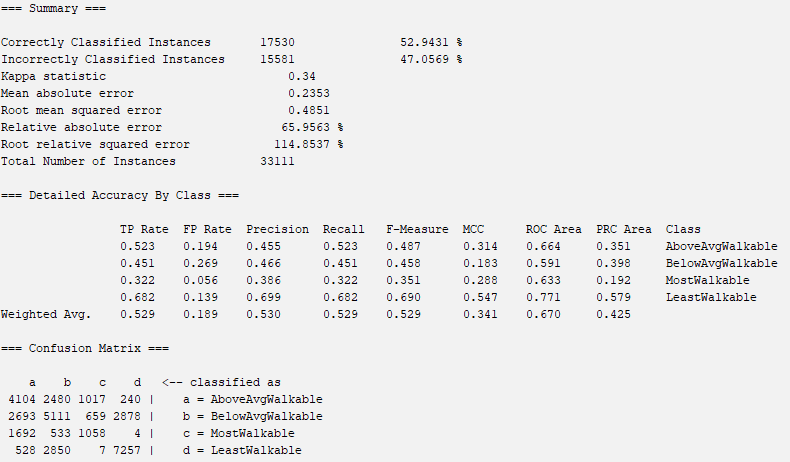
GainRatioAttributeEval with RandomForest:

****

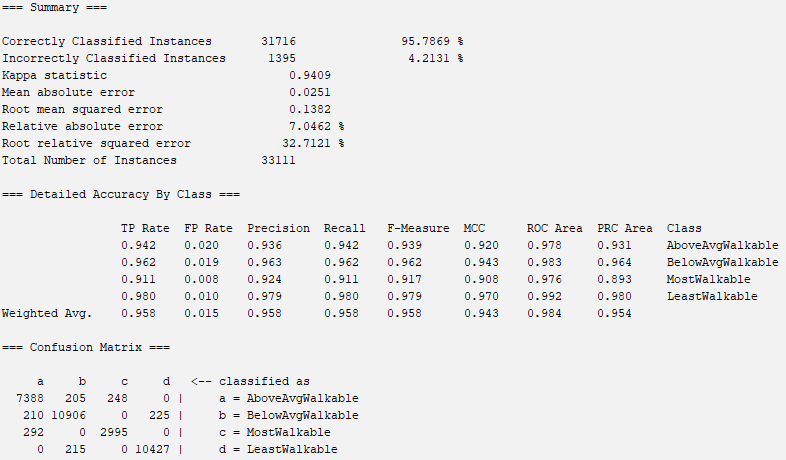
OneRAttributeEval with Naive Bayes:



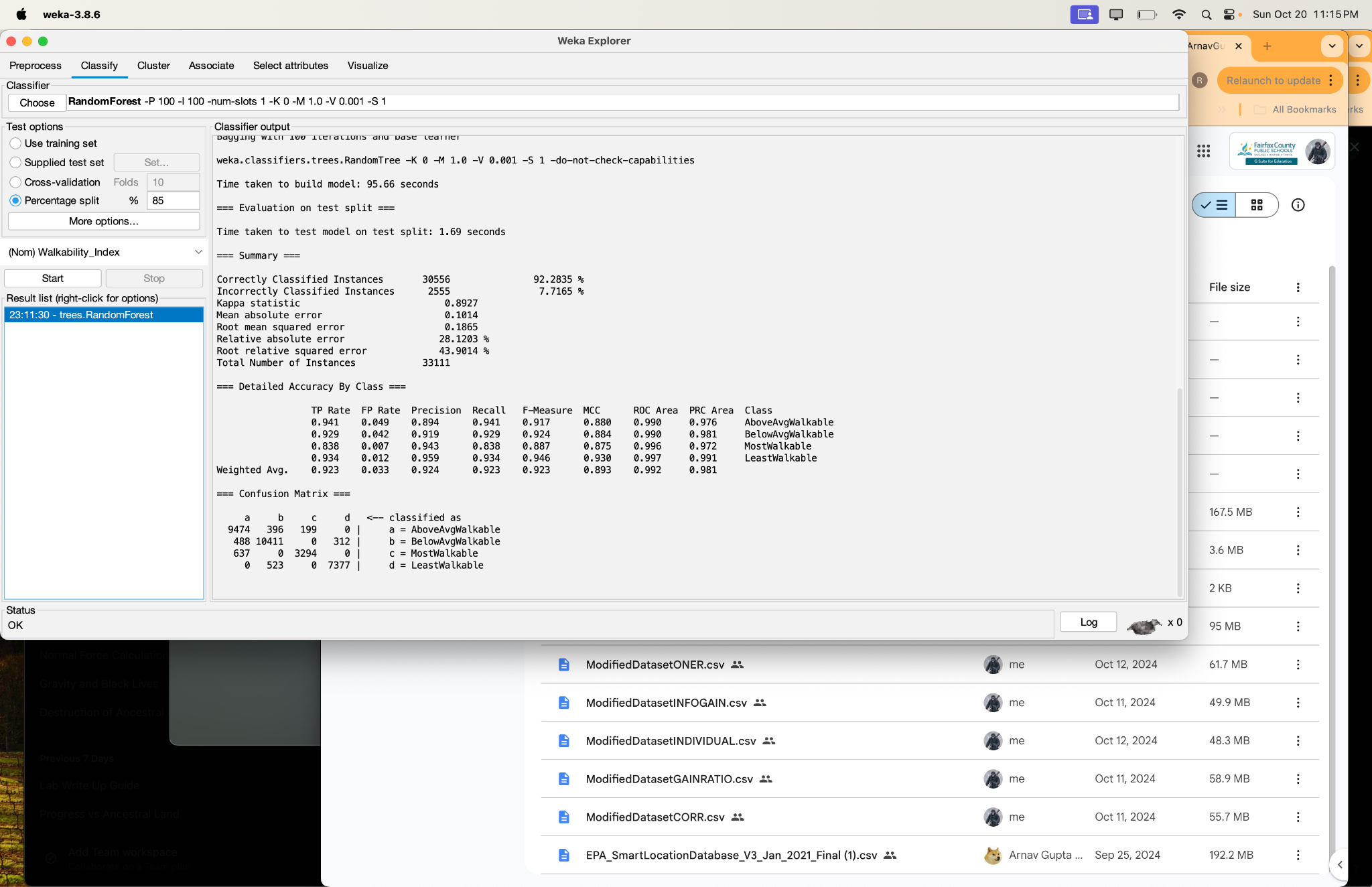
OneRAttributeEval with OneR:



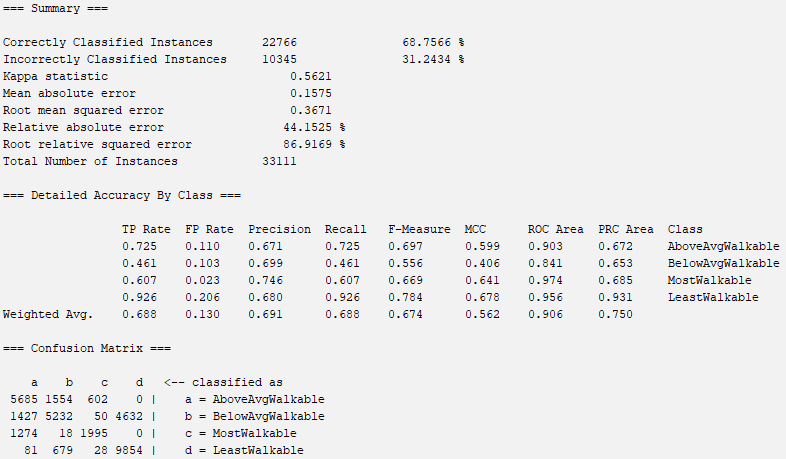
OneRAttributeEval with J48:



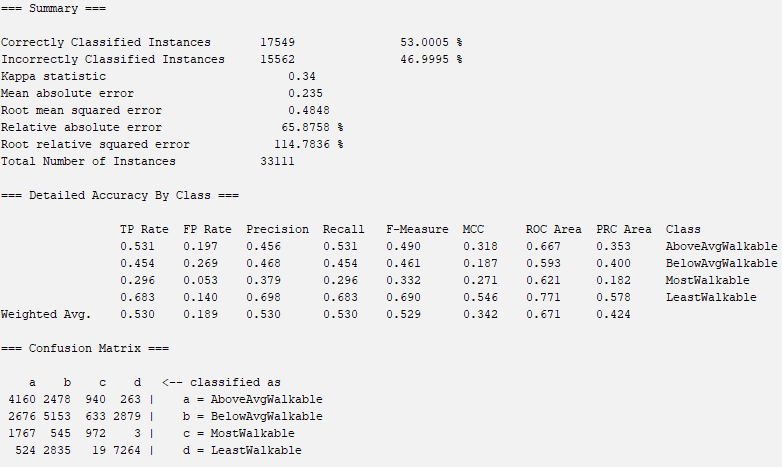
OneRAttributeEval with RandomForest:



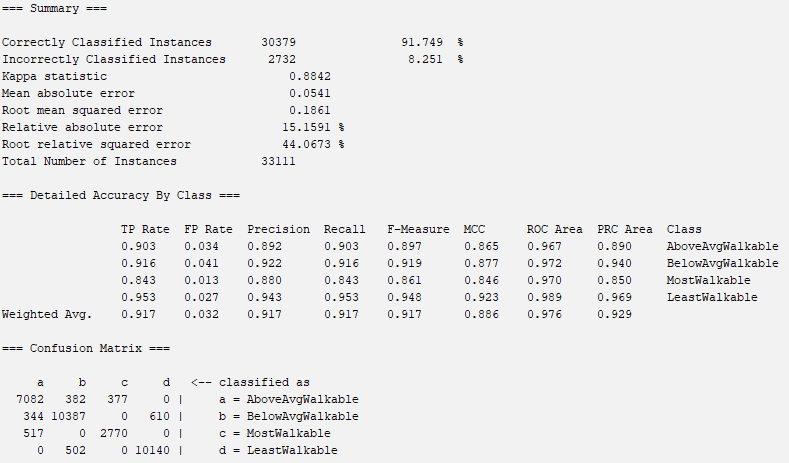
Team Choice “AttributeEval” with Naive Bayes:



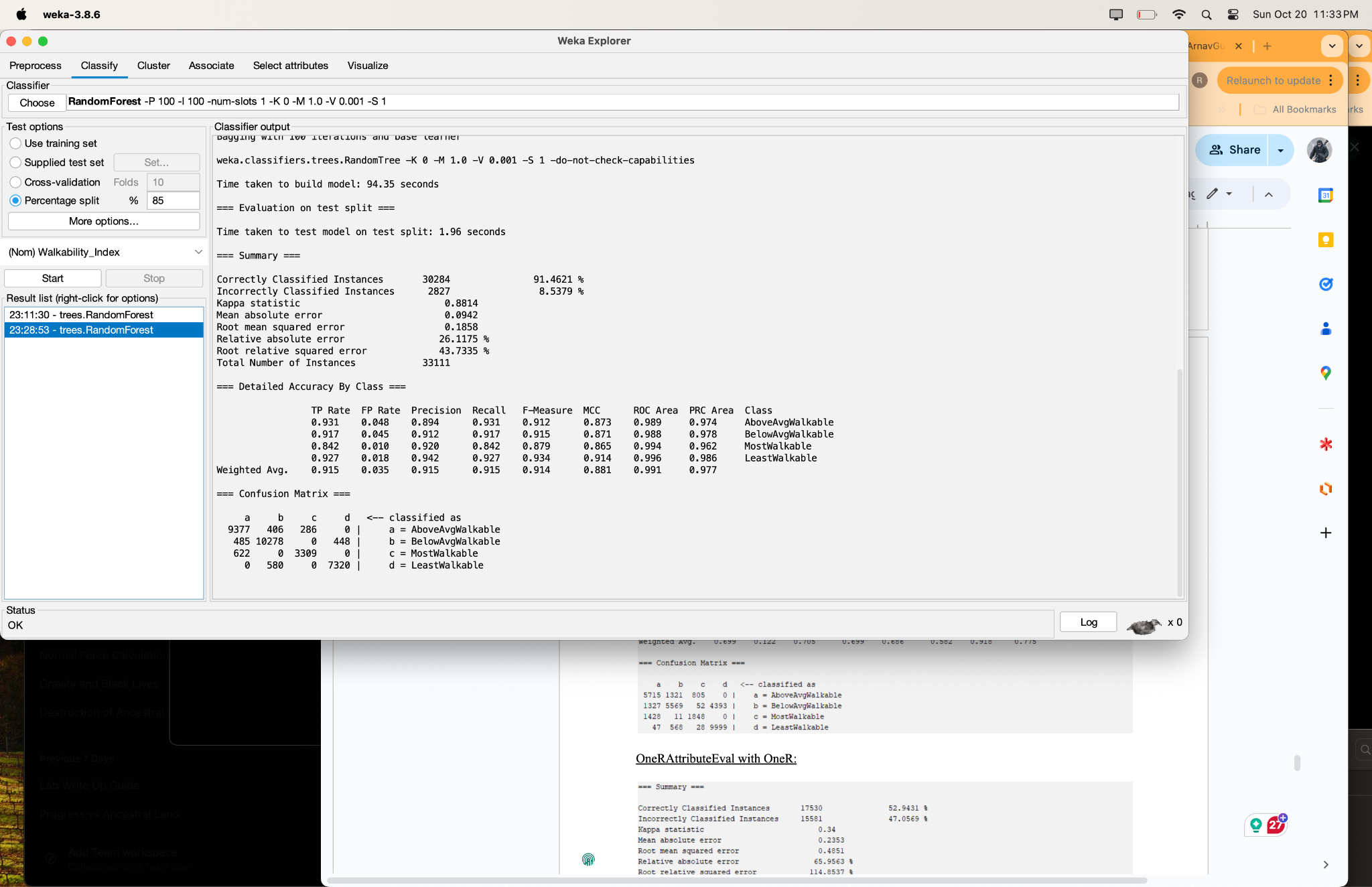
Team Choice “AttributeEval” with OneR:



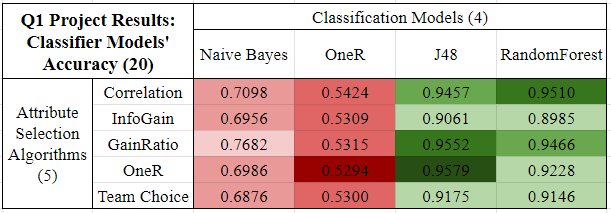
Team Choice “AttributeEval” with J48:



Team Choice “AttributeEval” with RandomForest:

****

After running the 20 models (4 different models on the five different attribute selection algorithm-processed datasets) using the train/test/validation sets, we created a table to view each combination of attribute selection algorithm and classifier model result in a clean way below



After obtaining the results of all 20 models’ performances, it’s clear that our best model is one of three options: Correlation AttributeEval with RandomForest, GainRatio AttributeEval with J48, or OneR AttributeEval with J48 due to their blanket accuracy percentages being so close to each other (making it seem like they could all converge in the long run to very similar results).

However, a closer look at the information in WEKA’s summary (and detailed summary) shows that the OneR AttributeEval with J48 Classifier Model is the best model. This is due to the fact that all of its error rates (Mean Absolute Error, Root Mean Squared Error, Relative Absolute Error, Root Relative Squared Error) are the lowest amongst the three models, while both its curves (ROC and PRC curves) exhibit the highest values amongst the three models, thus cementing its position as the best model amongst the twenty models.

**Part 7: Discussion, Conclusion, and Model Replication**

As discovered above, our best model was the OneR AttributeEval with J48 Classifier, which obtained the maximum accuracy of all twenty models: 95.79%. With this accuracy on roughly 33,000 instances of testing data, we believe our project, and especially this model, proved to be highly successful. Despite being successfully able to train, test, and validate the classification model that predicted an American regions’ walkability index with strong consistency, we believe there is definitely room for improvement. Namely, we maintained over 20 attributes after each attribute selection process, and if we had assigned higher, more stringent thresholds, we might have been able to potentially increase our overall accuracy by a significant percentage. If we were to replicate this project, we would have made sure to test various thresholds for the attribute selection algorithms (starting with the higher ones) and/or research the several classifier models to determine the optimal model for our type of project (classification).

In conclusion, we learned numerous concepts about machine learning in general, as well as effectively utilized our theoretical knowledge to potentially help others in a real-world setting. First, we learned how to find a raw dataset from reputable sources. Next we learned how to comb through raw files with both Python and WEKA to conduct numerous preprocessing steps and attribute selection algorithms and techniques that we learned in the class labs. Then, after learning how to make train/test/split files in Python, we learned how to run several models in WEKA to predict accuracies of our testing datasets based on manual input, cross-fold validation techniques, or raw train/test split features. Lastly, we learned how to interpret the results with more sophistication than the accuracy, where we effectively analyzed various error rates, area curves, and the popular confusion matrix to determine our best model. We are confident in our new skills learned in this project, and we look forward to applying them to real world applications similar to this project.

To reproduce our datasets, models, and results, there are numerous steps that’ll need to be taken in order to achieve the final datasets and models we’ve created. Below, you’ll find the ordered process to recreate our best model (OneR AttributeEval with J48 Classifier) from scratch.

1. Navigate to the publicly available dataset on the U.S. Government’s database of datasets called Walkability Index by clicking [here](https://catalog.data.gov/dataset/walkability-index7), and download the CSV file of this dataset.
2. Open the CSV file in Microsoft Excel and manually change the name of the 115th column from ‘NatWalkInd’ to ‘Walkability\_Index’ and shift this column two columns over from the 115th column to the 117th (last) column. Save the dataset as ModifiedDataset.csv
3. Open VSCode and run the first Python script on the fifth page of this report. This process will discretize the numeric values of the class attribute Walkability\_Index and then bin them into one of four bins depending on their WI.
4. Run the second Python script on the sixth page of this report. This process should remove all default values (namely 0 and 0.0) and replace them with commas (WEKA’s default missing value symbol).
5. Open the new CSV file (now named ModifiedDataset2.csv after the last step saving process) and manually remove all the attributes with 70%+ missing values. By this process, sixty attributes were removed, namely AC\_Total, AC\_Water, AC\_Land, AC\_Unpr, TotEmp, E5\_ Ret, E5\_Off, E5\_Ind, E5\_ Svc, E5\_Ent, E8\_Ret, E8\_off, E8\_ Ind, E8\_Svc, E8\_Ent, E8\_Ed, E8\_HIth, E8\_Pub, E\_LowWageWk, E\_MedWageWk, E\_HiWageWk, D1A, D1B, D1C, D1C5\_RET, D1C5\_OFF, D1C5\_IND, D1C5\_SVC, D1C5\_ENT, D1C8\_RET, D1C8\_OFF, D1C8\_IND, D1C8\_SVC, D1C8\_ENT, D1C8\_ED, D1C8\_HLTH, D1C8\_PUB, D1D, D1\_FLAG, D2A\_JPHH, DA\_WRKEMP, D3BPO4, D4A, D4B025, D4B050, D4C, D4D, D4E, D5AR, D5AE, D5BR, D5BE, D5CR, D5CE, D5DR, D5DRI, D5DE, D5DEI, Shape\_Length, and Shape\_Area.
6. Fill in missing values by finding WEKA’s ReplaceMissingValues filter located in filters > Choose > filters > unsupervised > attribute > ReplaceMissingValues. Next, click ‘Apply’ to apply this filter to the entire dataset.
7. Run the attribute selection on the current dataset and note the threshold; for our best model, these features are OneR. To do this, select the ‘Select Attributes’ tab. Next, in Attribute Evaluator, click ‘Choose’ > attributeSelection > OneRAttributeEval, and accept WEKA’s pop-up to make the Search Method Ranker. Click Start, and note the attributes and their coefficients. Go back to the Preprocess tab and remove all the attributes with a coefficient less than 36 (with a minimum bucket size of 6). All the attributes should be removed except the following: D3B\_Ranked, D4A\_Ranked, D3B, D3A, GEOID10, GEOID20, D3APO, D3BMM4, CBSA\_WRK, CBSA, CBSA\_POP, CBSA\_EMP, OBJECTID, D3BMM3, D3BAO, TRACTCE, CBSA\_Name, D3AAO, CSA, STATEFP, CSA\_Name, COUNTYFP, D5CEI, D2B\_Ranked, D5CRI, D3AMM, D2A\_Ranked, AutoOwn0, P\_WrkAge, AutoOwn2p, D2C\_TRIPEQ, and Pct\_AO2p.
8. Save the dataset as a CSV file, and run the third Python script on the fifteenth page of this report to complete the stratified random sampling process to maintain class distributions for the next step, building train/validation/testing datasets.
9. Run the fourth Python script on the sixteenth page of this report to make the three train, validation, and testing datasets on 70/15/15 splits. Note that this will create three files.
10. Open the largest file (training dataset) in WEKA, and head to the Classify tab for the last parts of the project: uploading the testing dataset, building the model, and obtaining the final results.
11. Change the classifier by clicking Choose > trees > J48 (for our best model) and in the test options, click ‘Supplied test set’ and open then upload the testing dataset to this location.
12. Click the ‘Start’ button and wait a couple minutes for the model to build and results to be calculated and cleanly outputted to the Classifier Output space.

The above process should result in the Classifier Output showing the same results as the first image on the twenty-fifth page of this report. This concludes the replication of our best model.

**Part 8: Team Members and Task Distribution**

The entirety of this project was completed by only two students, Arnav Gupta and Raghav Kamineni, who both are in Dr. Yilmaz’s fifth period Machine Learning 1 class. Below is a list of the parts Arnav Gupta, Raghav Kamineni, and both teammates completed together (with the first name contributing significantly more for that specific task at hand than the second name).

Finding Dataset and Building Proposal: Arnav Gupta

Preprocessing Initial Attempt: Raghav Kamineni

Preprocessing & Project Update: Raghav Kamineni and Arnav Gupta

Non-WEKA Attribute Selection Algorithm: Raghav Kamineni

Attribute Selection Algorithms and Classifiers: Arnav Gupta

Results Output: Raghav Kamineni

Results Analysis: Arnav Gupta

Building Final Report: Arnav Gupta and Raghav Kamineni

**Part 9: Sources and Acknowledgements**

For this project, we used the U.S. Government’s Data Catalog to find our dataset. Out of the 300,000+ datasets, we chose the Walkability Index dataset from the U.S. Environmental Protection Agency, which can be found here: <https://catalog.data.gov/dataset/walkability-index7>.

In addition to the dataset, we also used Visual Studio Code to run our several Python scripts, and lastly, we used WEKA to complete the entirety of our attribute selection algorithm and classifier model processes to obtain the final results and analysis.

Lastly, we’d like to thank the U.S. Environment Protection Agency for uploading their comprehensive Walkability Index dataset to a publicly available dataset database, and more importantly, we’d like to thank Dr. Yilmaz for supporting our team throughout the whole process: from teaching us to use WEKA through labs to reviewing our dataset and deliverables with frequently valuable feedback for us. Thank you.

END OF QUARTER ONE PROJECT BY ARNAV GUPTA AND RAGHAV KAMINENI