FALL, 2020

CS 59000-014 Machine Learning Project

GINI COEFFICIENT: INEQUALITY BETWEEN RICH & POOR

GARVIT, ANITA, MYRIAM, LUKE
PURDUE UNIVERSITY NORTHWEST
2200 169th street, Hammond, IN 46323

Table of Content

Team Members

Name	Title	Job
Mutyala, Anitha Chowdary	Member	Data Collection Attributes Search Helps in data Preprocessing
Changoluisa Toapanta, Myriam	Member	Data Preprocessing Attributes Selection Filtering Normalization
Garvit Agrawal	Team Lead	Giving KT to the team. Model Training (Decision Tree) Analysis Result
Luke James Conway	Member	Help in data collection User Interface

Introduction

In economics, the **Gini coefficient**, sometimes called the **Gini index** or **Gini ratio**, is a measure of statistical dispersion intended to represent the income inequality or wealth inequality within a nation or any other group of people. It was developed by the Italian statistician and sociologist Corrado Gini and published in his 1912 paper *Variability and Mutability*

The Gini coefficient measures the inequality among values of a frequency distribution (for example, levels of income). A Gini coefficient of zero expresses perfect equality, where all values are the same (for example, where everyone has the same income). A Gini coefficient of one (or 100%) expresses maximal inequality among values (e.g., for many people where only one person has all the income or consumption and all others have none, the Gini coefficient will be nearly one).

Previous course projects and the study conducted at the University of Michigan have utilized data mining technologies to develop data models that analyze substantial amounts of data which contains information on the income, expenses, and the financial footprints of families in the United States. These models and data were used to analyze the ever-widening gap between the rich and the poor and the major elements that govern this widening behavior are of great interest and concern in the modern day.

Objective

The objective of this project is to develop a Gini Coefficient Prediction System that examine the gap between the rich and the poor.

Chapter 1: Data Collection

Data collection is the process of gathering and measuring information on targeted variables in an established system, which then enables one to answer relevant questions and evaluate outcomes. While methods vary by discipline, the emphasis on ensuring accurate and honest collection remains the same. The goal for all data collection is to capture quality evidence that allows analysis to lead to the formulation of convincing and credible answers to the questions that have been posed.

Step 1: Define the aim of your research

Before you start the process of data collection, you need to identify exactly what you want to achieve. You can start by writing a problem statement: what is the practical or scientific issue that you want to address and why does it matter?

Next, formulate one or more research questions that precisely define what you want to find out. Depending on your research questions, you might need to collect quantitative or qualitative data:

- **Quantitative data** is expressed in numbers and graphs and is analyzed through statistical methods.
- **Qualitative data** is expressed in words and analyzed through interpretations and categorizations.

If your aim is to test a hypothesis, measure something precisely, or gain large-scale statistical insights, collect quantitative data. If your aim is to explore ideas, understand experiences, or gain detailed insights into a specific context, collect qualitative data. If you have several aims, you can use a mixed methods approach that collects both types of data.

Step 2: Choose your data collection method

Based on the data you want to collect, decide which method is best suited for your research.

- Experimental research is primarily a quantitative method.
- Interviews/focus groups and ethnography are qualitative methods.
- Surveys, observations, archival research and secondary data collection can be quantitative or qualitative methods.

Carefully consider what method you will use to gather data that helps you directly answer your research questions.

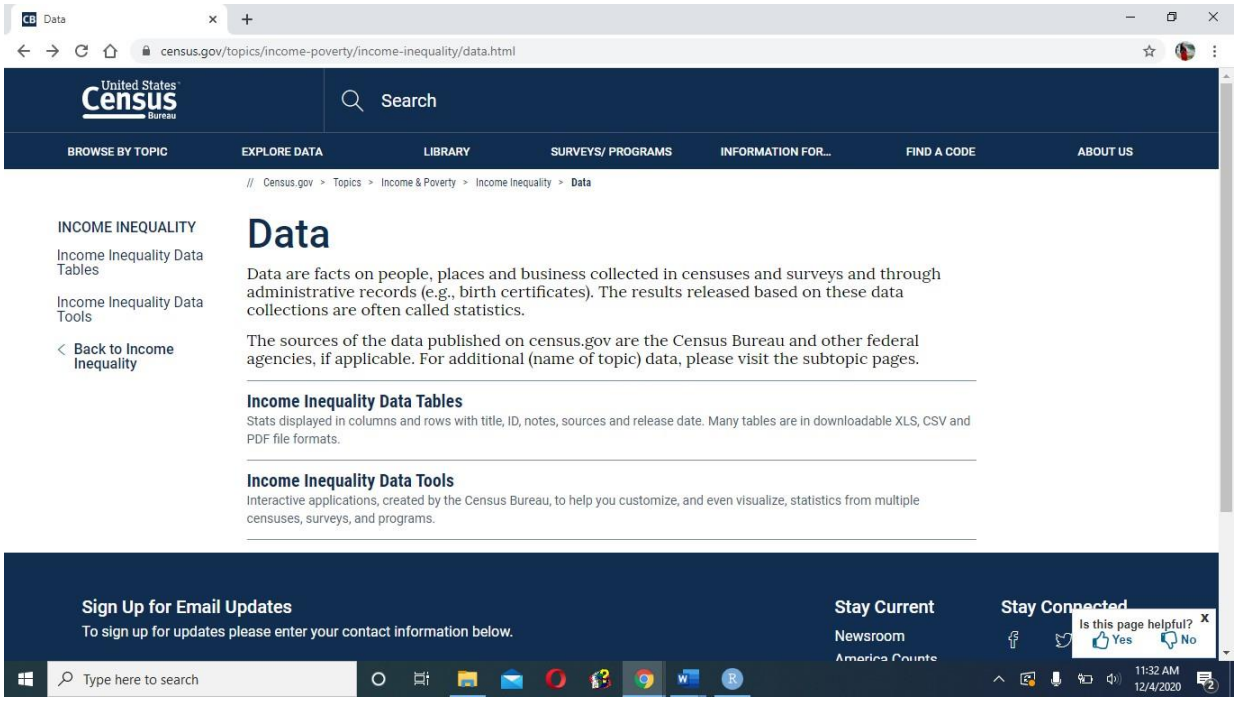
Method Used: Secondary Data Collection

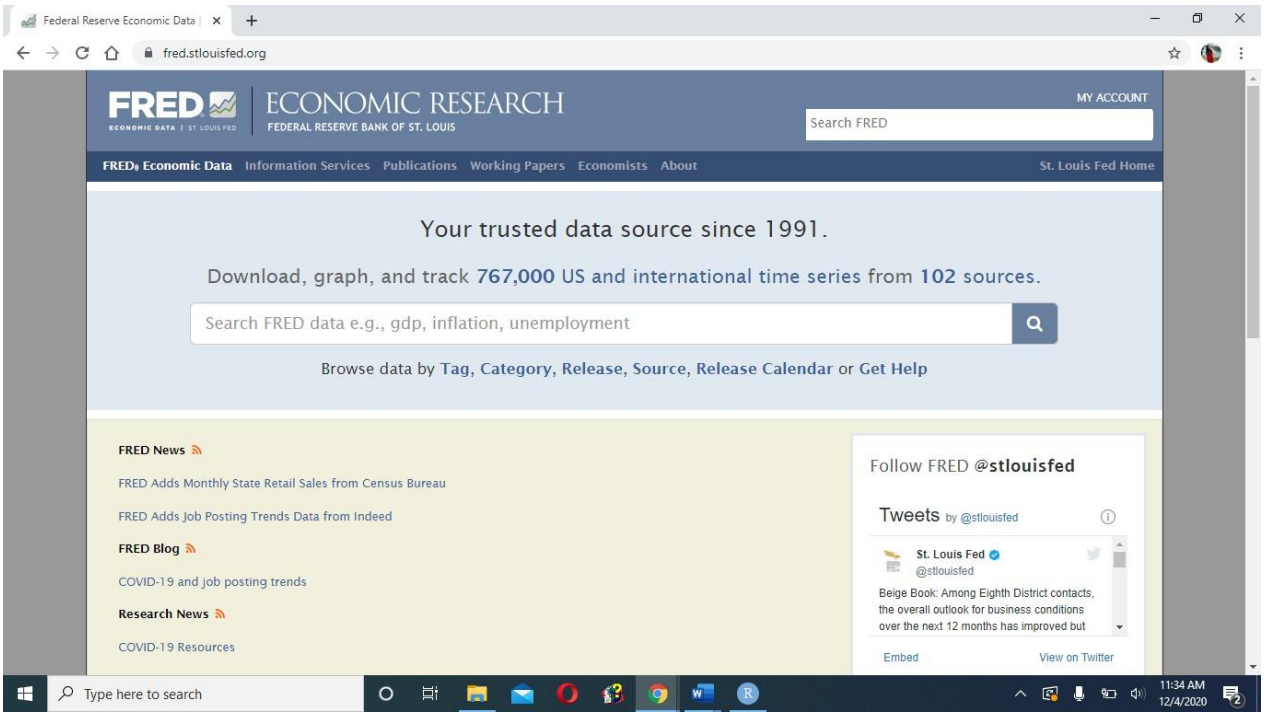
This method is used to find existing datasets that have already been collected, from sources such as government agencies or research organizations. To analyze data from populations that you can't access first-hand.

Responsibilities:

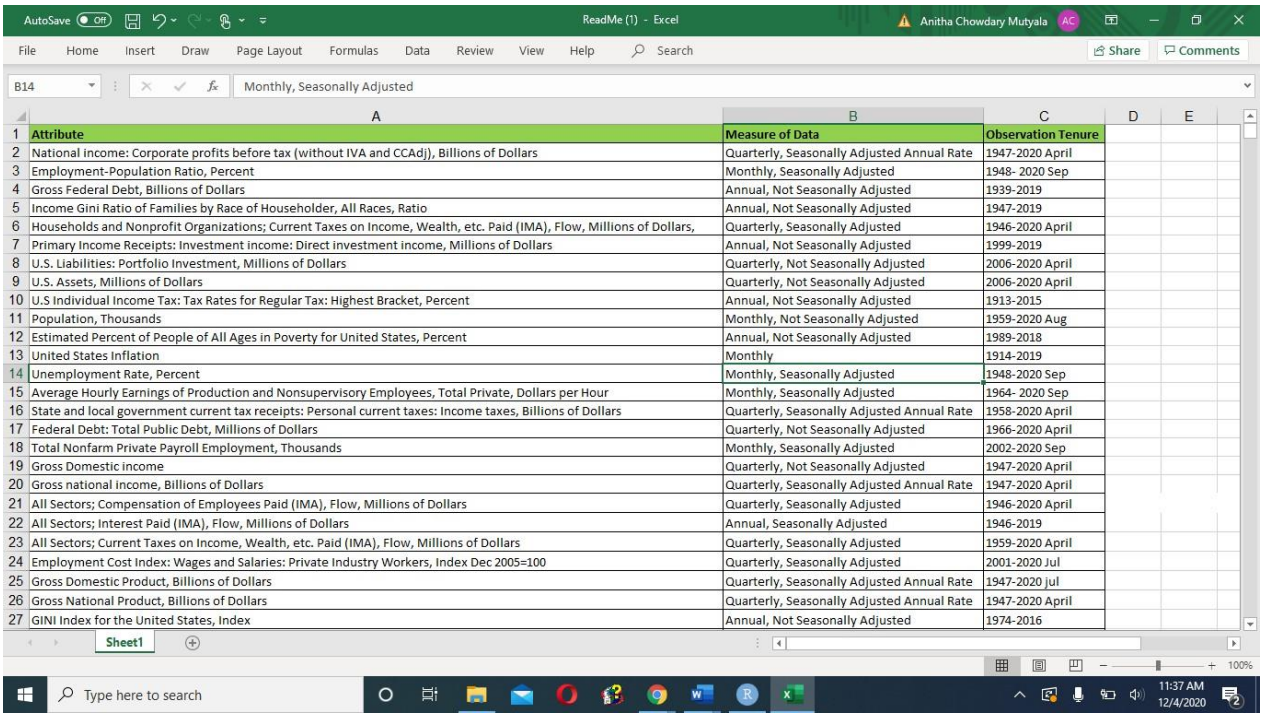
- Primarily responsible for collecting data from various websites. We have discussed and finalized some attributes that would be relevant to this project. I worked for about 6-8 weeks to identify appropriate data from websites(primarily https://www.census.gov/ and https://fred.stlouisfed.org)

- Collecting and updating data on services including their suitability to meet needs of project and how they can be accessed. Sharing appropriate information with team members and discussing about the attributes collected.



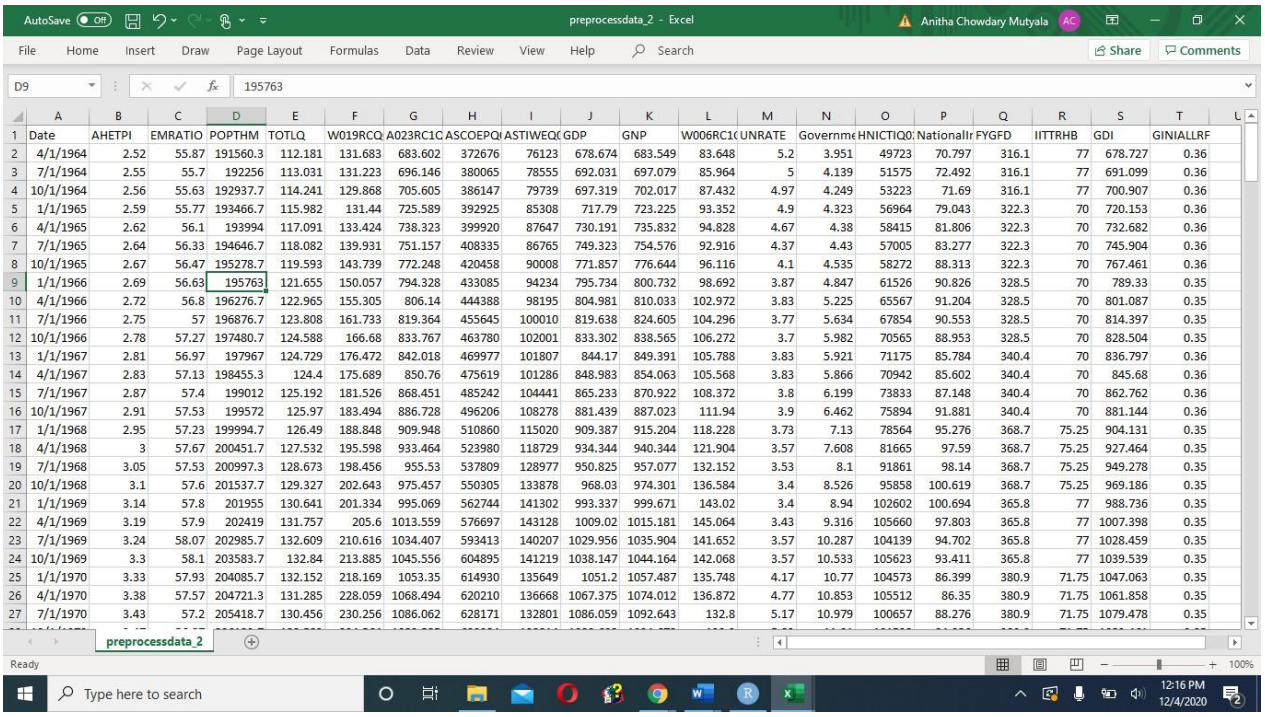


- Overall, we have collected 30 attributes in the initial stage.



	A	B	C
1	**Attribute**	**Measure of Data**	**Observation Tenure**
2	National income: Corporate profits before tax (without IVA and CCAdj), Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April
3	Employment-Population Ratio, Percent	Monthly, Seasonally Adjusted	1948- 2020 Sep
4	Gross Federal Debt, Billions of Dollars	Annual, Not Seasonally Adjusted	1939-2019
5	Income Gini Ratio of Families by Race of Householder, All Races, Ratio	Annual, Not Seasonally Adjusted	1947-2019
6	Households and Nonprofit Organizations; Current Taxes on Income, Wealth, etc. Paid (IMA), Flow, Millions of Dollars,	Quarterly, Seasonally Adjusted	1946-2020 April
7	Primary Income Receipts: Investment income: Direct investment income, Millions of Dollars	Annual, Not Seasonally Adjusted	1999-2019
8	U.S. Liabilities: Portfolio Investment, Millions of Dollars	Quarterly, Not Seasonally Adjusted	2006-2020 April
9	U.S. Assets, Millions of Dollars	Quarterly, Not Seasonally Adjusted	2006-2020 April
10	U.S Individual Income Tax: Tax Rates for Regular Tax: Highest Bracket, Percent	Annual, Not Seasonally Adjusted	1913-2015
11	Population, Thousands	Monthly, Not Seasonally Adjusted	1959-2020 Aug
12	Estimated Percent of People of All Ages in Poverty for United States, Percent	Annual, Not Seasonally Adjusted	1989-2018
13	United States Inflation	Monthly	1914-2019
14	Unemployment Rate, Percent	Monthly, Seasonally Adjusted	1948-2020 Sep
15	Average Hourly Earnings of Production and Nonsupervisory Employees, Total Private, Dollars per Hour	Monthly, Seasonally Adjusted	1964- 2020 Sep
16	State and local government current tax receipts: Personal current taxes: Income taxes, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1958-2020 April
17	Federal Debt: Total Public Debt, Millions of Dollars	Quarterly, Not Seasonally Adjusted	1966-2020 April
18	Total Nonfarm Private Payroll Employment, Thousands	Monthly, Seasonally Adjusted	2002-2020 Sep
19	Gross Domestic income	Quarterly, Not Seasonally Adjusted	1947-2020 April
20	Gross national income, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April
21	All Sectors; Compensation of Employees Paid (IMA), Flow, Millions of Dollars	Quarterly, Seasonally Adjusted	1946-2020 April
22	All Sectors; Interest Paid (IMA), Flow, Millions of Dollars	Annual, Seasonally Adjusted	1946-2019
23	All Sectors; Current Taxes on Income, Wealth, etc. Paid (IMA), Flow, Millions of Dollars	Quarterly, Seasonally Adjusted	1959-2020 April
24	Employment Cost Index: Wages and Salaries: Private Industry Workers, Index Dec 2005=100	Quarterly, Seasonally Adjusted	2001-2020 Jul
25	Gross Domestic Product, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 jul
26	Gross National Product, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April
27	GINI Index for the United States, Index	Annual, Not Seasonally Adjusted	1974-2016

- We have decided on the most appropriate attributes amongst the 30 attributes and finalized our data with 19 attributes **for the** interval (1964-2020) and 225 cases/rows.



AutoSave Off | preprocessdata_2 - Excel | Anitha Chowdary Mutyala

File Home Insert Draw Page Layout Formulas Data Review View Help Search Share Comments

D9 195763

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Date	AHETPI	EMRATIO	POPTHM	TOTLQ	W019RCQ	A023RC1C	ASCOEPQ	ASTIWEQ	GDP	GNP	W006RC1C	UNRATE	Governme	HNICTIQ0	NationalIr	FYGFD	IITTRHB	GDI	GINIALLRF
2	4/1/1964	2.52	55.87	191560.3	112.181	131.683	683.602	372676	76123	678.674	683.549	83.648	5.2	3.951	49723	70.797	316.1	77	678.727	0.36
3	7/1/1964	2.55	55.7	192256	113.031	131.223	696.146	380065	78555	692.031	697.079	85.964	5	4.139	51575	72.492	316.1	77	691.099	0.36
4	10/1/1964	2.56	55.63	192937.7	114.241	129.868	705.605	386147	79739	697.319	702.017	87.432	4.97	4.249	53223	71.69	316.1	77	700.907	0.36
5	1/1/1965	2.59	55.77	193466.7	115.982	131.44	725.589	392925	85308	717.79	723.225	93.352	4.9	4.323	56964	79.043	322.3	70	720.153	0.36
6	4/1/1965	2.62	56.1	193994	117.091	133.424	738.323	399920	87647	730.191	735.832	94.828	4.67	4.38	58415	81.806	322.3	70	732.682	0.36
7	7/1/1965	2.64	56.33	194646.7	118.082	139.931	751.157	408335	86765	749.323	754.576	92.916	4.37	4.43	57005	83.277	322.3	70	745.904	0.36
8	10/1/1965	2.67	56.47	195278.7	119.593	143.739	772.248	420458	90008	771.857	776.644	96.116	4.1	4.535	58272	88.313	322.3	70	767.461	0.36
9	1/1/1966	2.69	56.63	195763	121.655	150.057	794.328	433085	94234	795.734	800.732	98.692	3.87	4.847	61526	90.826	328.5	70	789.33	0.35
10	4/1/1966	2.72	56.8	196276.7	122.965	155.305	806.14	444388	98195	804.981	810.033	102.972	3.83	5.225	65567	91.204	328.5	70	801.087	0.35
11	7/1/1966	2.75	57	196876.7	123.808	161.733	819.364	455645	100010	819.638	824.605	104.296	3.77	5.634	67854	90.553	328.5	70	814.397	0.35
12	10/1/1966	2.78	57.27	197480.7	124.588	166.68	833.767	463780	102001	833.302	838.565	106.272	3.7	5.982	70565	88.953	328.5	70	828.504	0.35
13	1/1/1967	2.81	56.97	197967	124.729	176.472	842.018	469977	101807	844.17	849.391	105.788	3.83	5.921	71175	85.784	340.4	70	836.797	0.36
14	4/1/1967	2.83	57.13	198455.3	124.4	175.689	850.76	475619	101286	848.983	854.063	105.568	3.83	5.866	70942	85.602	340.4	70	845.68	0.36
15	7/1/1967	2.87	57.4	199012	125.192	181.526	868.451	485242	104441	865.233	870.922	108.372	3.8	6.199	73833	87.148	340.4	70	862.762	0.36
16	10/1/1967	2.91	57.53	199572	125.97	183.494	886.728	496206	108278	881.439	887.023	111.94	3.9	6.462	75894	91.881	340.4	70	881.144	0.36
17	1/1/1968	2.95	57.23	199994.7	126.49	188.848	909.948	510860	115020	909.387	915.204	118.228	3.73	7.13	78564	95.276	368.7	75.25	904.131	0.35
18	4/1/1968	3	57.67	200451.7	127.532	195.598	933.464	523980	118729	934.344	940.344	121.904	3.57	7.608	81665	97.59	368.7	75.25	927.464	0.35
19	7/1/1968	3.05	57.53	200997.3	128.673	198.456	955.53	537809	128977	950.825	957.077	132.152	3.53	8.1	91861	98.14	368.7	75.25	949.278	0.35
20	10/1/1968	3.1	57.6	201537.7	129.327	202.643	975.457	550305	133878	968.03	974.301	136.584	3.4	8.526	95858	100.619	368.7	75.25	969.186	0.35
21	1/1/1969	3.14	57.8	201955	130.641	201.334	995.069	562744	141302	993.337	999.671	143.02	3.4	8.94	102602	100.694	365.8	77	988.736	0.35
22	4/1/1969	3.19	57.9	202419	131.757	205.6	1013.559	576697	143128	1009.02	1015.181	145.064	3.43	9.316	105660	97.803	365.8	77	1007.398	0.35
23	7/1/1969	3.24	58.07	202985.7	132.609	210.616	1034.407	593413	140207	1029.956	1035.904	141.652	3.57	10.287	104139	94.702	365.8	77	1028.459	0.35
24	10/1/1969	3.3	58.1	203583.7	132.84	213.885	1045.556	604895	141219	1038.147	1044.164	142.068	3.57	10.533	105623	93.411	365.8	77	1039.539	0.35
25	1/1/1970	3.33	57.93	204085.7	132.152	218.169	1053.35	614930	135649	1051.2	1057.487	135.748	4.17	10.77	104573	86.399	380.9	71.75	1047.063	0.35
26	4/1/1970	3.38	57.57	204721.3	131.285	228.059	1068.494	620210	136668	1067.375	1074.012	136.872	4.77	10.853	105512	86.35	380.9	71.75	1061.858	0.35
27	7/1/1970	3.43	57.2	205418.7	130.456	230.256	1086.062	628171	132801	1086.059	1092.643	132.8	5.17	10.979	100657	88.276	380.9	71.75	1079.478	0.35

preprocessdata_2

Ready 100%

Type here to search 12:16 PM 12/4/2020

Chapter 2: Attributes

Below is the table which shows the attributes along with the description and the source from where the data has been collected:

Attribute	**Measure of Data**	**Observation Tenure**	**Source**
National income: Corporate profits before tax (without IVA and CCAdj), Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April	https://fred.stlouisfed.org/series/A053RC1Q027SBEA
Employment-Population Ratio, Percent	Monthly, Seasonally Adjusted	1948- 2020 Sep	https://fred.stlouisfed.org/series/EMRATIO
Gross Federal Debt, Billions of Dollars	Annual, Not Seasonally Adjusted	1939-2019	https://fred.stlouisfed.org/series/FYGFD
Income Gini Ratio of Families by Race of Householder, All Races, Ratio	Annual, Not Seasonally Adjusted	1947-2019	https://fred.stlouisfed.org/series/GINIALLRF
Households and Nonprofit Organizations; Current Taxes on Income, Wealth, etc. Paid (IMA), Flow, Millions of Dollars,	Quarterly, Seasonally Adjusted	1946-2020 April	https://fred.stlouisfed.org/series/HNICTIQ027S
Primary Income Receipts: Investment income: Direct investment income, Millions of Dollars	Annual, Not Seasonally Adjusted	1999-2019	https://fred.stlouisfed.org/series/IEAXID
U.S. Liabilities: Portfolio Investment, Millions of Dollars	Quarterly, Not Seasonally Adjusted	2006-2020 April	https://fred.stlouisfed.org/series/IIPPORTLQ

U.S. Assets, Millions of Dollars	Quarterly, Not Seasonally Adjusted	2006-2020 April	https://fred.stlouisfed.org/series/WALCL
U.S Individual Income Tax: Tax Rates for Regular Tax: Highest Bracket, Percent	Annual, Not Seasonally Adjusted	1913-2015	https://fred.stlouisfed.org/series/IITTRHB
Population, Thousands	Monthly, Not Seasonally Adjusted	1959-2020 Aug	https://fred.stlouisfed.org/series/POPTHM
Estimated Percent of People of All Ages in Poverty for United States, Percent	Annual, Not Seasonally Adjusted	1989-2018	https://fred.stlouisfed.org/series/PPAAUS00000A156NCEN
United States Inflation	Monthly	1914-2019	https://fred.stlouisfed.org/series/T10YIE
Unemployment Rate, Percent	Monthly, Seasonally Adjusted	1948-2020 Sep	https://fred.stlouisfed.org/series/UNRATE
Average Hourly Earnings of Production and Nonsupervisory Employees, Total Private, Dollars per Hour	Monthly, Seasonally Adjusted	1964- 2020 Sep	https://fred.stlouisfed.org/series/AHETPI
State and local government current tax receipts: Personal current taxes: Income taxes, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1958-2020 April	https://fred.stlouisfed.org/series/B245RC1Q027SBEA
Federal Debt: Total Public Debt, Millions of Dollars	Quarterly, Not Seasonally Adjusted	1966-2020 April	https://fred.stlouisfed.org/series/GFDEBTN
Total Nonfarm Private Payroll Employment, Thousands	Monthly, Seasonally Adjusted	2002-2020 Sep	https://fred.stlouisfed.org/series/NPPTTL

Gross Domestic income	Quarterly, Not Seasonally Adjusted	1947-2020 April	https://fred.stlouisfed.org/series/GDPC1
Gross national income, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April	https://fred.stlouisfed.org/series/GDP
All Sectors; Compensation of Employees Paid (IMA), Flow, Millions of Dollars	Quarterly, Seasonally Adjusted	1946-2020 April	https://fred.stlouisfed.org/series/ASCOEPQ027S
All Sectors; Interest Paid (IMA), Flow, Millions of Dollars	Annual, Seasonally Adjusted	1946-2019	https://fred.stlouisfed.org/series/ASINPAA027N
All Sectors; Current Taxes on Income, Wealth, etc. Paid (IMA), Flow, Millions of Dollars	Quarterly, Seasonally Adjusted	1959-2020 April	https://fred.stlouisfed.org/series/ASTIWEQ027S
Employment Cost Index: Wages and Salaries: Private Industry Workers, Index Dec 2005=100	Quarterly, Seasonally Adjusted	2001-2020 Jul	https://fred.stlouisfed.org/series/ECIWAG
Gross Domestic Product, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 jul	https://fred.stlouisfed.org/series/GDP
Gross National Product, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April	https://fred.stlouisfed.org/series/GDPC1
GINI Index for the United States, Index	Annual, Not Seasonally Adjusted	1974-2016	https://fred.stlouisfed.org/series/SIPOVGINIUSA
Hours of Wage and Salary Workers on Nonfarm Payrolls: Total, Billions of Hours	Quarterly, Seasonally Adjusted Annual Rate	1964-2020	https://fred.stlouisfed.org/series/TOTLQ

Consumer Price Index for All Urban Consumers (CPI-U), United States	Monthly	1913-2019	https://fred.stlouisfed.org/series/CPIAUCSL
Federal government current tax receipts, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1947-2020 April	https://fred.stlouisfed.org/series/W006RC1Q027SBEA
Federal government total expenditures, Billions of Dollars	Quarterly, Seasonally Adjusted Annual Rate	1959-2020 Jul	https://fred.stlouisfed.org/series/W019RCQ027SBEA

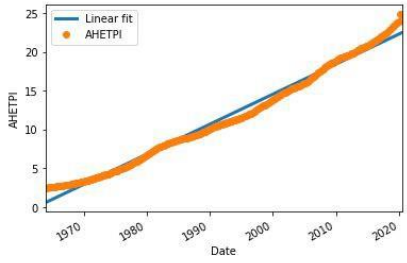
Chapter 3: Data Preprocessing

The probability of anomalous data has increased in current data due to its enormous size and its origin in diversified sources. Data preprocessing has become vital and a fundamental step in the machine learning pipeline considering the fact that high-quality data leads to better models and predictions.

Linear Regression on Time Series

Linear regression is commonly used as a type of predictive analysis. In this project, it is used to fill in missing data.

```
f, ax = plt.subplots(1, 1)
ax.plot(xf1, yf,label='Linear fit', lw=3)
AHETPI['AHETPI'].dropna().plot(ax=ax,marker='o', ls='')
plt.ylabel('AHETPI')
ax.legend();
```



Linear regression fit of Average Hourly Expenses vs quarterly interval time

Time series data is a collection of quantities that are assembled over even intervals in time and ordered chronologically. The time interval at which data is collected is generally referred to as the time series frequency. For this project, the time series data comes from the Economic Research Federal Reserve Bank of St. Louis and the data time intervals are yearly, quarterly, monthly and daily. The new dataset time interval is quarterly after the data preprocessing.

Data preprocessing steps:

The following steps are used to preprocess the data for this project.

1. Import packages
2. Import datasets
3. Create linear regression function
4. Create time series function
5. Predict values for 19 attributes
6. Concatenate data frames to export the new dataset

1. Import packages

The predefined Python libraries can perform specific data preprocessing jobs. The core Python libraries used for this data preprocessing, time series and simple linear regression are:

```
import pandas as pd
import numpy as np
import scipy.stats as sp
import matplotlib.pyplot as plt
%matplotlib inline
import time
from datetime import datetime
from sklearn import linear_model
```

2. Import dataset

In this step, I imported the 19 time series datasets that were previously collected.

```
EMRATIO = pd.read_excel (r'C:\datafile\EMRATIO_quaterly.xlsx', skiprows = range(1,65) )
EMRATIO.set_index('Date', inplace=True)
```

The new dataset interval time is from [1964 - 04 - 01] to [2020 - 10 - 01] quarterly. This interval time was selected after analyzing the interval dates of all datasets. The skiprows value range is different for all the 19 attributes.

3. Create linear regression function

For this project, I implemented a linear function to predict missing values for 19 attributes.

```
def linearFunctionToPredictValues(nameString):
        y=np.array(nameString.values, dtype=float)
        x=np.array(pd.to_datetime(nameString.dropna()).index.values, dtype=float)
        slope, intercept, r_value, p_value, std_err =sp.linregress(x,y)
        xf = np.linspace(min(x),max(x),100) xf1
        = xf.copy()
        xf1 = pd.to_datetime(xf1) yf
        = (slope*xf)+intercept
        return slope, intercept
```

4. Create time series function

For this project, I implemented a function to create time series data to be used as the input value in the linear regression function to predict missing output values for 19 attributes.

```
def createNewDates(initial, final, slope1, intercept1,intervals):
        index = pd.date_range(initial, final, intervals)
        index, len(index)
        new_arr = np.array(index.values, dtype = float)
        yf_new_values = (slope1*new_arr)+intercept1 print('The
        new predicted values are:' , yf_new_values)
```

5. Predict values for 19 attributes

Applied the two created functions to predict the missing values for the 19 attributes of the dataset.

```
import datetime
slope , intercept = linearFunctionToPredictValues(AHETPI['AHETPI'])

start1 = datetime.datetime(2020, 7, 1)
end1 = datetime.datetime(2020, 10, 1)
interval = 2

createNewDates(start1, end1, slope, intercept, interval )
```

Result:

```
The new predicted values are: [22.48347585 22.58103358]
```

The new predicted values are attached to the dataset to fill in the missing values for the given range data related to start1 and end1 in the code. For this specific case, only two values were missing in the AHETPI attribute for the following interval date [2020 - 07 - 01] and [2020 - 10 - 01].

1. Concatenate data frames to export the new dataset

In this final step, the data is concatenated to create the new dataset with 19 attributes which will be used for the analysis and development of an intelligent inflation prediction system.

```
horizontal_stack = pd.concat([AHETPI, EMRATIO, POPTHM, TOTLQ, W019RCQ027SBEA,
A023RC1Q027SBEA, ASCOEPQ027S, ASTIWEQ027S, GDP, GNP,
W006RC1Q027SBEA, UNRATE, Government, HNICTIQ027S, NationalIncome, FYGFD, IITTRHB,
GDI,GINIALLRF ], axis=1)
newdata = horizontal_stack.dropna()
newdata.to_csv('preprocessdata_2.csv', index = True)
```

After preprocessing the data, the new dataset is more accurate since the incorrect or missing values are deleted that are there as a result of the human factor or bugs. It also makes the dataset more complete since missing values from different attributes are filled in using linear regression. Finally, the new data boosts consistency and smoothness which makes the data easy to use, interpret and affects the accuracy of the results. The final dataset has 19 attributes and 225 rows.

Date	AHETPI	EMRATIO	POPTHM	TOTLQ	W019RCQ027SBEA	A023RC1Q027SBEA	ASCOEPQ027S	ASTIWEQ027S	GDP	GNP
1964-04-01	2.52	55.87	191560.33	112.181	131.683	683.602	372676.0	76123.0	678.674	683.549
1964-07-01	2.55	55.70	192256.00	113.031	131.223	696.146	380065.0	78555.0	692.031	697.079
1964-10-01	2.56	55.63	192937.67	114.241	129.868	705.605	386147.0	79739.0	697.319	702.017
1965-01-01	2.59	55.77	193466.67	115.982	131.440	725.589	392925.0	85308.0	717.790	723.225
1965-04-01	2.62	56.10	193994.00	117.091	133.424	738.323	399920.0	87647.0	730.191	735.832
...	...	...	...	...	...	...	...	...	...	...
2019-10-01	23.80	61.00	329186.34	257.332	4939.740	22002.259	11577444.0	2533521.0	21747.394	22028.479
2020-01-01	23.98	60.77	329529.34	255.269	5029.832	21914.503	11686291.0	2507944.0	21561.139	21804.322
2020-04-01	24.94	52.90	329897.34	227.746	9225.115	19616.236	10890349.0	2334261.0	19520.114	19671.986
2020-07-01	24.74	56.07	331327.10	241.324	7334.744	18971.764	10091311.0	2231664.0	21157.635	18908.044
2020-10-01	22.58	62.20	331967.85	261.840	4383.420	19067.960	10141872.0	2242857.0	18850.400	19003.691

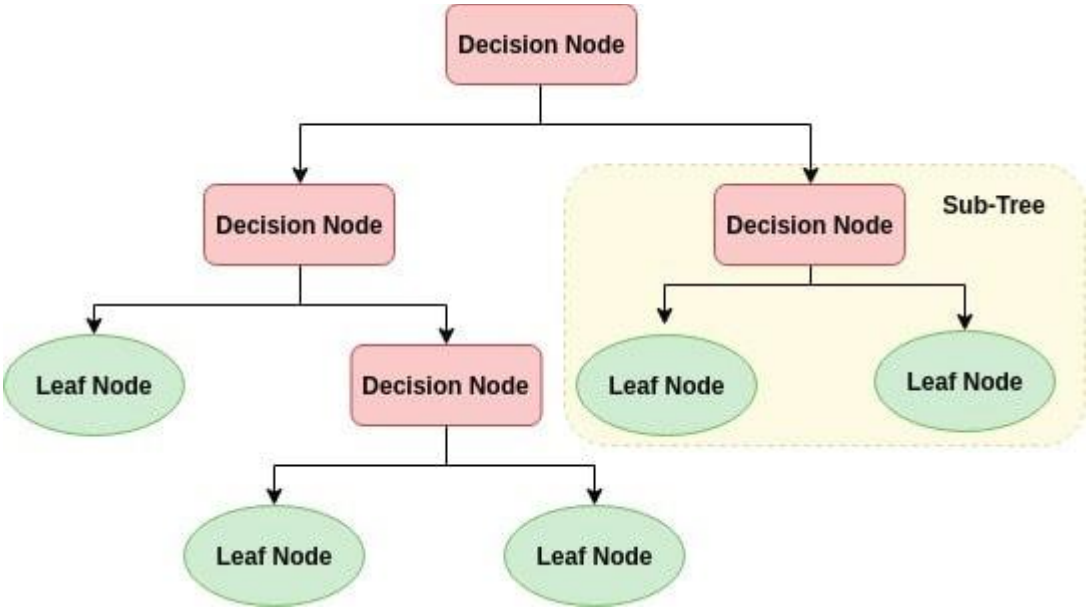
227 rows × 19 columns

Final dataset with 19 attributes and 225 rows

Chapter 4: Decision Tree

Decision Tree Algorithm

A decision tree is a flowchart-like tree structure where an internal node represents feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition based on the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



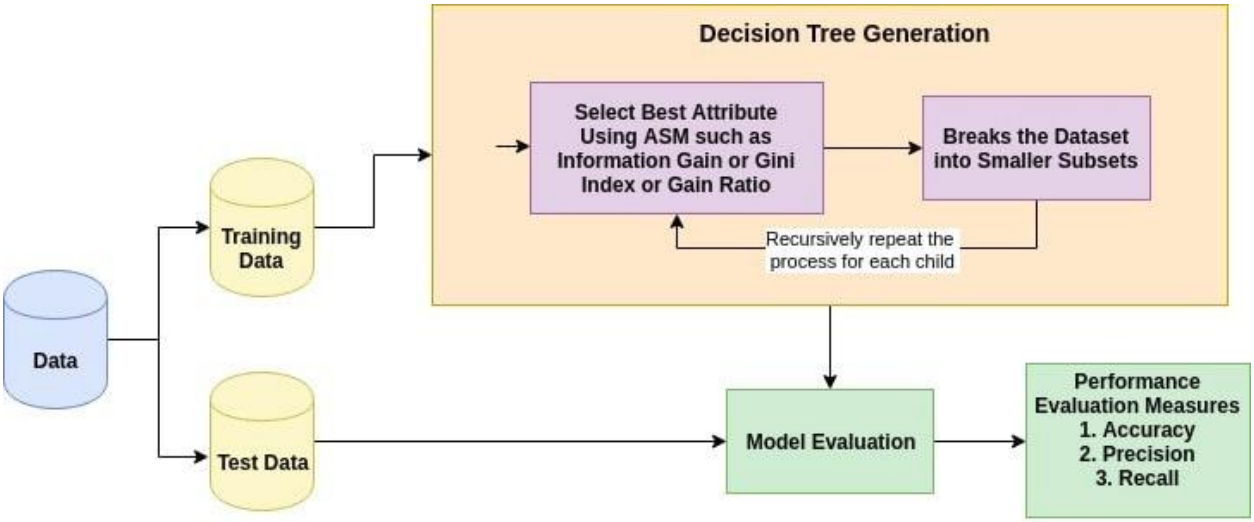
Decision Tree is a white box type of ML algorithm. It shares internal decision-making logic, which is not available in the black box type of algorithms such as Neural Network. Its training time is faster compared to the neural network algorithm. The time complexity of decision trees is a function of the number of records and number of attributes in the given data. The decision tree is a distribution-free or non-parametric method, which does not depend upon probability distribution assumptions. Decision trees can handle high dimensional data with good accuracy.

How does the Decision Tree algorithm work?

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures (ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.

3. Starts tree building by repeating this process recursively for each child until one of the conditions will match:
    - All the tuples belong to the same attribute value.
    - There are no more remaining attributes.
    - There are no more instances.



Attribute Selection Measures

Attribute selection measure is a heuristic for selecting the splitting criterion that partition data into the best possible manner. It is also known as splitting rules because it helps us to determine breakpoints for tuples on a given node. ASM provides a rank to each feature (or attribute) by explaining the given dataset. Best score attribute will be selected as a splitting attribute. In the case of a continuous-valued attribute, split points for branches also need to define. Most popular selection measures are Information Gain, Gain Ratio, and Gini Index.

Information Gain

Shannon invented the concept of entropy, which measures the impurity of the input set. In physics and mathematics, entropy referred as the randomness or the impurity in the system. In information theory, it refers to the impurity in a group of examples. Information gain is the decrease in entropy. Information gain computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

$$\text{Info}(D) = -\sum_{i=1}^{m} pi \log_2 pi$$

Where, Pi is the probability that an arbitrary tuple in D belongs to class Ci.

$$\text{Info}_A(D) = \sum_{j=1}^{V} \frac{|Dj|}{|D|} \quad \text{X Info}(D_j)$$

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D)$$

Where,

- Info(D) is the average amount of information needed to identify the class label of a tuple in D.
- |Dj|/|D| acts as the weight of the jth partition.
- InfoA(D) is the expected information required to classify a tuple from D based on the partitioning by A.

The attribute A with the highest information gain, Gain(A), is chosen as the splitting attribute at node N ().

Gain Ratio

Information gain is biased for the attribute with many outcomes. It means it prefers the attribute with many distinct values. For instance, consider an attribute with a unique identifier such as customer has zero info(D) because of pure partition. This maximizes the information gain and creates useless partitioning.

C4.5, an improvement of ID3, uses an extension to information gain known as the gain ratio. Gain ratio handles the issue of bias by normalizing the information gain using Split Info. Java implementation of the C4.5 algorithm is known as J48, which is available in WEKA data mining tool.

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2 \left(\frac{|D_j|}{|D|} \right)$$

Where,

- |Dj|/|D| acts as the weight of the jth partition.
- v is the number of discrete values in attribute A.

The gain ratio can be defined as

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo_A(D)}$$

The attribute with the highest gain ratio is chosen as the splitting attribute.

Gini index

Another decision tree algorithm CART (Classification and Regression Tree) uses the Gini method to create split points.

$$\text{Gini(D)}=1 - \sum_{i=1}^{m} \text{Pi}^2$$

Where, pi is the probability that a tuple in D belongs to class Ci.

The Gini Index considers a binary split for each attribute. You can compute a weighted sum of the impurity of each partition. If a binary split on attribute A partitions data D into D1 and D2, the Gini index of D is:

$$\text{Gini}_A\text{(D)}=\frac{|D1|}{|D|}\text{Gini}(D_1)+\frac{|D2|}{|D|}\text{Gini}(D_2)$$

In case of a discrete-valued attribute, the subset that gives the minimum gini index for that chosen is selected as a splitting attribute. In the case of continuous-valued attributes, the strategy is to select each pair of adjacent values as a possible split-point and point with smaller gini index chosen as the splitting point.

$$\Delta Gini(A) = Gini(D) - Gini_A(D).$$

The attribute with minimum Gini index is chosen as the splitting attribute.

Chapter 5: Analysis

Data preparation for the decision tree model

Once all the data had been collected and preprocessed, it was used to train a decision tree model. This model was coded in Python using the SKLearn Machine Learning library and the Pandas library for importing and manipulating data.

The collected data was first loaded into Python using Pandas:

Loading Data

We first load the dataset using pandas' read CSV function.

```
#reading the data file
dataset = pd.read_csv("C:\\Users\\agraw\\Documents\\PNW\\Fall 2020\\ML project\\Dataset\\Final_Data\\preprocessdata_2.csv")
```

```
#showing the head of the data
dataset.head()
```

	Date	AHETPI	EMRATIO	POPTHM	TOTLQ	W019RCQ027SBEA	A023RC1Q027SBEA	ASCOEPQ027S	ASTIWEQ027S	GDP	GNP	W006RC1Q027S
0	4/1/1964	2.52	55.87	191560.33	112.181	131.683	683.602	372676	76123	678.674	683.549	8
1	7/1/1964	2.55	55.70	192256.00	113.031	131.223	696.146	380065	78555	692.031	697.079	8
2	10/1/1964	2.56	55.63	192937.67	114.241	129.868	705.605	386147	79739	697.319	702.017	8
3	1/1/1965	2.59	55.77	193466.67	115.982	131.440	725.589	392925	85308	717.790	723.225	9
4	4/1/1965	2.62	56.10	193994.00	117.091	133.424	738.323	399920	87647	730.191	735.832	9

```
#showing the dataset size
dataset.shape
```

```
(225, 20)
```

Feature Selection

Here, we need to divide given columns into two types of variables dependent (or target variable) and independent variable (or feature variables).

```
#preparing dependednt and independent values
dataset = dataset.drop('Date', axis=1)
x = dataset.drop('GINIALLRF' , axis=1)
y = dataset['GINIALLRF']
```

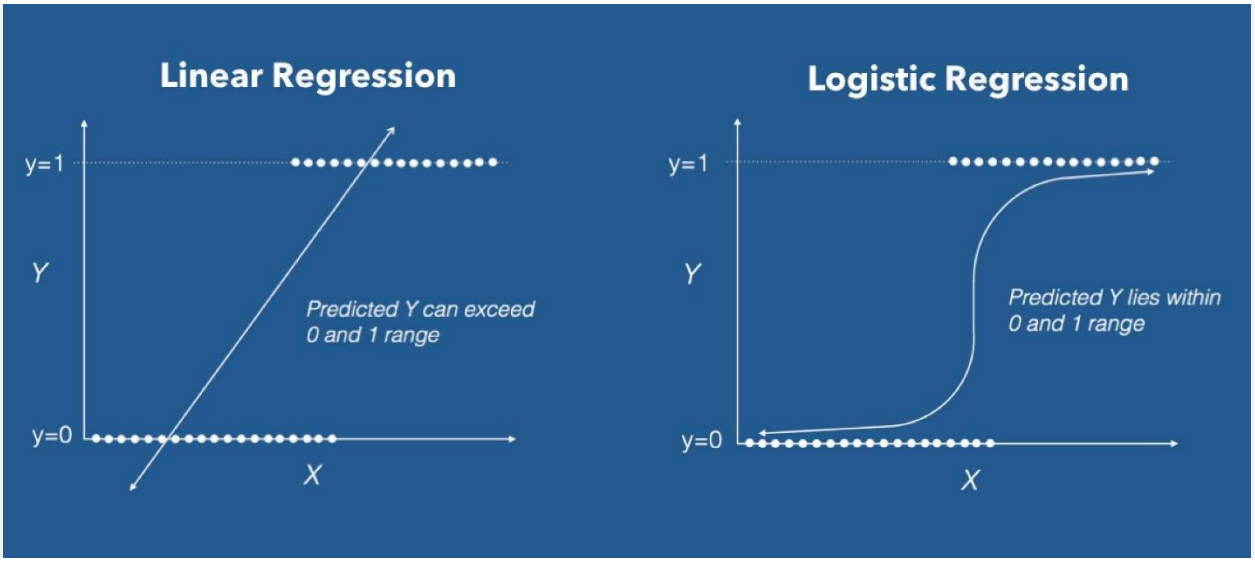
```
#preprocessing the data
from sklearn.preprocessing import MinMaxScaler
X_norm = MinMaxScaler().fit_transform(x)
Y = y.to_numpy()
```

```
#using lable encoder encoding the independent value
from sklearn import utils
from sklearn import preprocessing
lab_enc = preprocessing.LabelEncoder()
Y_encod = lab_enc.fit_transform(Y)
print(Y_encod)
print(utils.multiclass.type_of_target(Y))
print(utils.multiclass.type_of_target(Y.astype('int')))
print(utils.multiclass.type_of_target(Y_encod))
```

```
[ 1  1  1  1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0  0  0  0  0  0
  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  1  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  2  2  3
  3  3  3  3  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4  4  4  4  4  5
  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  5  7  7  7  7  7  7
  7  7  7  6  6  6  6  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7  7
  7  7  7  8  8  8  8  7  7  7  7  8  8  8  8  8  8  8  8  8  8  8  8  8
  8  8  8  7  7  7  7  8  8  8  8  8  8  8  8  8  8  8  9  9  9  9  9  9
  9  9  9 10 10 10 10  9  9  9  9  9  9  9  9  9  9  9 10 10 10 10  9  9
  9  9  9  9  9  9 11 11]
continuous
binary
multiclass
```

Logistic regression

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.



Linear Regression VS Logistic Regression Graph| Image: Data Camp

We can call a Logistic Regression a Linear Regression model, but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.
The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

$$0 \leq h_\theta(x) \leq 1$$

Logistic regression hypothesis expectation

We use logistic regression to get the good attributes for the model to increase our model accuracy.

```
#Selecting the attributes by using logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
model = LogisticRegression()
rfe = RFE(model, 12)
fit = rfe.fit(X_norm, Y_encod)
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

```
Num Features: 12
Selected Features: [ True  True  True  True False  True False  True False False  True  True
 False False  True  True  True  True]
Feature Ranking: [1 1 1 1 7 1 2 1 3 5 1 1 6 4 1 1 1 1]
```

```
#droping the attributes as suggested by logistic regression
data = dataset.drop(['W019RCQ027SBEA','ASCOEPQ027S','GDP','GNP','Government','HNICTIQ027S'], axis=1)
X_norm = MinMaxScaler().fit_transform(data)
```

Splitting Data

To understand model performance, dividing the dataset into a training set and a test set is a good strategy.

Then we split the dataset by using function train_test_split(). You need to pass 3 parameters features, target, and test_set size.

```
#preparing the test and train data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_norm, Y_encod, test_size=0.30)
```

Then we create a Decision Tree Model using Scikit-learn.

```
#training the model
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = "gini",max_depth=6)
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=6, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
#Testing the model
y_pred = classifier.predict(X_test)
```

Evaluating Model

Then we estimate, how accurately the classifier or model can predict the type of cultivars. Accuracy can be computed by comparing actual test set values and predicted values.

```
#Checking the accuracy and error
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 3  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  9  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  2  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  6  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  3  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  0  0  0  0]
 [ 0  0  0  0  2  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 12  0  0  0]
 [ 0  0  0  0  0  0  0  0  0 11  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  3  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         6
           4       0.60      1.00      0.75         3
           5       1.00      1.00      1.00         3
           6       0.00      0.00      0.00         2
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        12
           9       1.00      1.00      1.00        11
          10       1.00      1.00      1.00         3
          11       1.00      1.00      1.00         1

    accuracy                           0.97        68
   macro avg       0.88      0.92      0.90        68
weighted avg       0.95      0.97      0.96        68
```

```
from sklearn import metrics
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
print('Accuracy:', round(metrics.accuracy_score(y_test, y_pred),3)*100)
print('Balanced Accuracy:', round(metrics.balanced_accuracy_score(y_test, y_pred),3)*100)
```

```
Accuracy: 0.9705882352941176
Accuracy: 97.1
Balanced Accuracy: 91.7
```

We got a classification rate of 97.1%, accuracy.

Visualizing Decision Trees

You can use Scikit-learn's *export_graphviz* function for display the tree within a Jupyter notebook. For plotting tree, you also need to install graphviz.

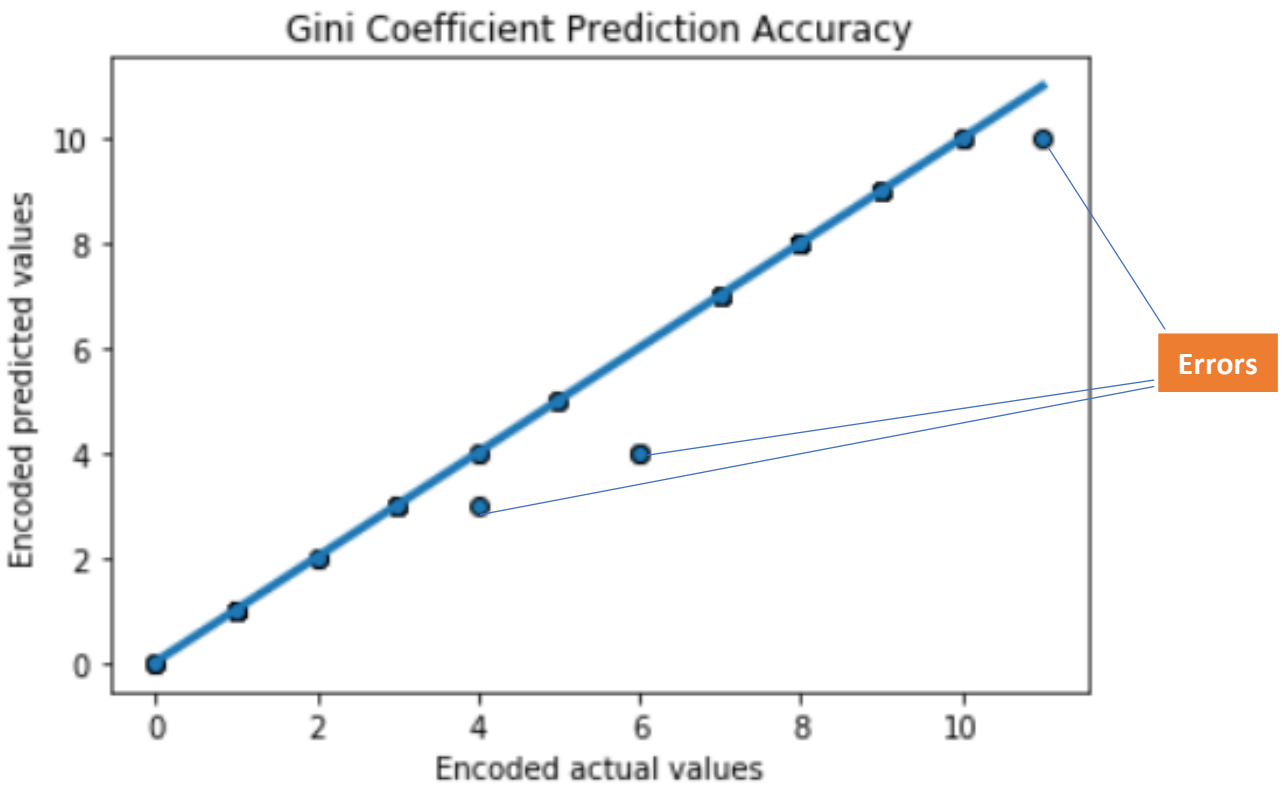
```
pip install graphviz
```

export_graphviz function converts decision tree classifier into dot file and then convert this dot file to png or displayable form on Jupyter.

```
#representing the tree
from sklearn import tree
text_representation = tree.export_text(classifier)
print(text_representation)
```

```
|--- feature_9 <= 0.02
|   |--- feature_12 <= 0.04
|   |   |--- class: 0
|   |--- feature_12 >  0.04
|   |   |--- class: 1
|--- feature_9 >  0.02
|   |--- feature_12 <= 0.71
|   |   |--- feature_12 <= 0.62
|   |   |   |--- feature_10 <= 0.23
|   |   |   |   |--- class: 5
|   |   |   |--- feature_10 >  0.23
|   |   |   |   |--- feature_7 <= 0.42
|   |   |   |   |   |--- feature_9 <= 0.05
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_9 >  0.05
|   |   |   |   |   |   |--- class: 4
|   |   |   |   |--- feature_7 >  0.42
|   |   |   |   |   |--- class: 3
|   |   |--- feature_12 >  0.62
|   |   |   |--- class: 7
|   |--- feature_12 >  0.71
|   |   |--- feature_0 <= 0.75
|   |   |   |--- class: 8
|   |   |--- feature_0 >  0.75
|   |   |   |--- feature_12 <= 0.88
|   |   |   |   |--- class: 9
|   |   |   |--- feature_12 >  0.88
|   |   |   |   |--- feature_10 <= 0.00
|   |   |   |   |   |--- class: 11
|   |   |   |   |--- feature_10 >  0.00
|   |   |   |   |   |--- class: 10
```

```
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred, edgecolors=(0, 0, 0))
ax.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], lw=3)
ax.set_xlabel('Encoded actual values')
ax.set_ylabel('Encoded predicted values')
ax.set_title("Gini Coefficient Prediction Accuracy")
plt.show()
```

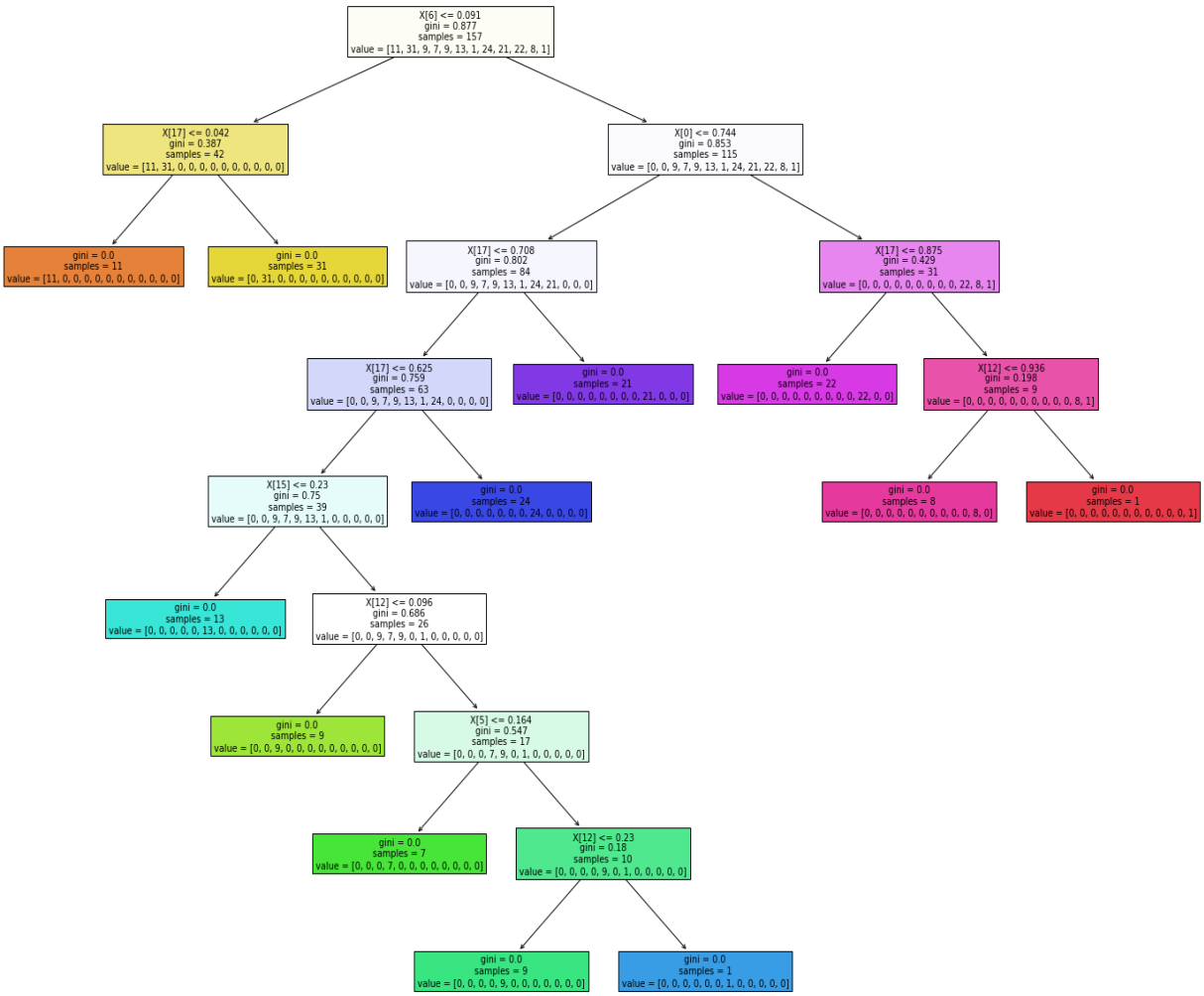


This graph represents the accuracy of the model where the x_lable and the y_lable shows the encode value of the GINI coefficients as

Encoded	0	1	2	3	4	5	6	7	8	9	10	11
GINI	0.35	0.36	0.37	0.38	0.39	0.40	0.42	0.43	0.44	0.45	0.46	0.47

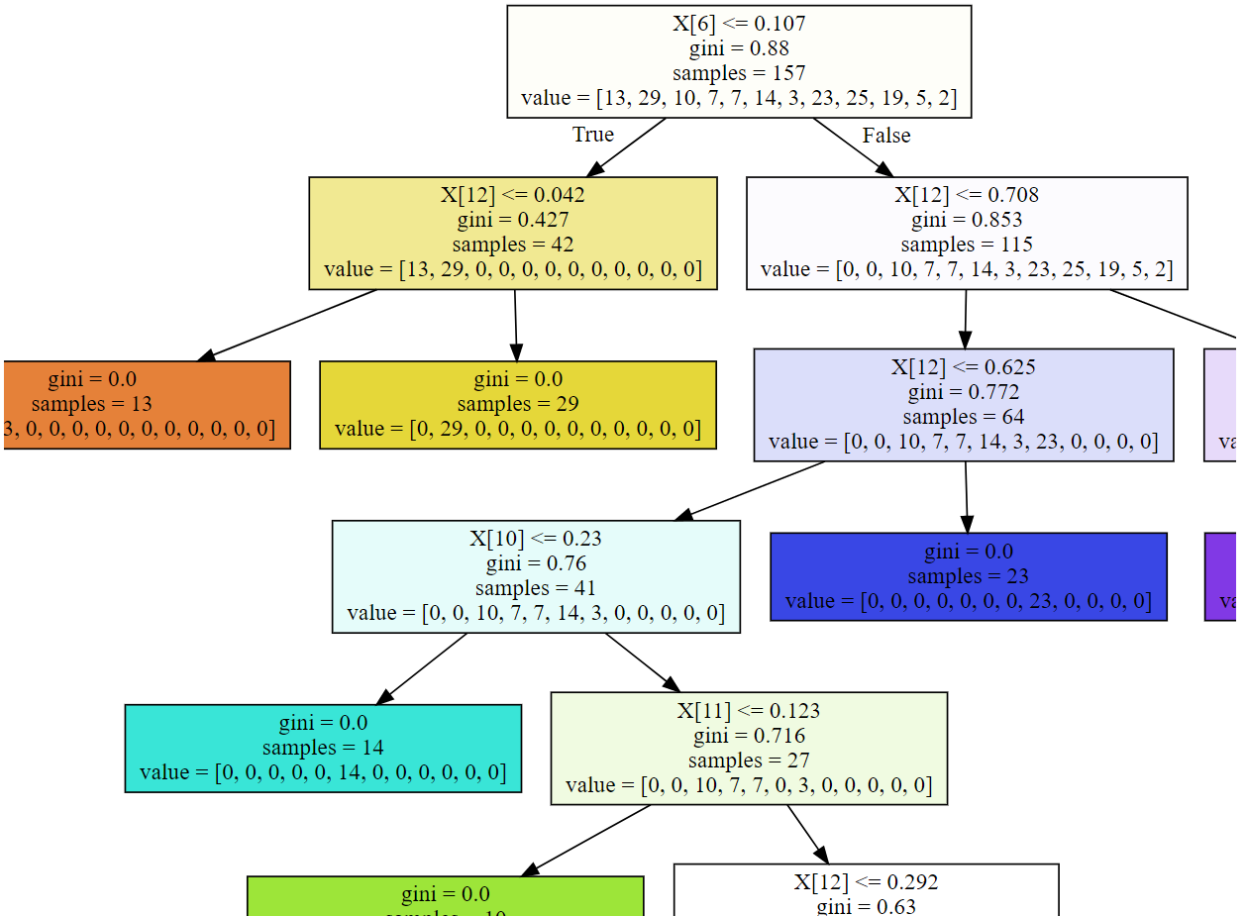
For example, At 2 it shows the actual GINI coefficient is 0.37 and our model also predict 0.37.

```
fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(classifier, filled=True)
```



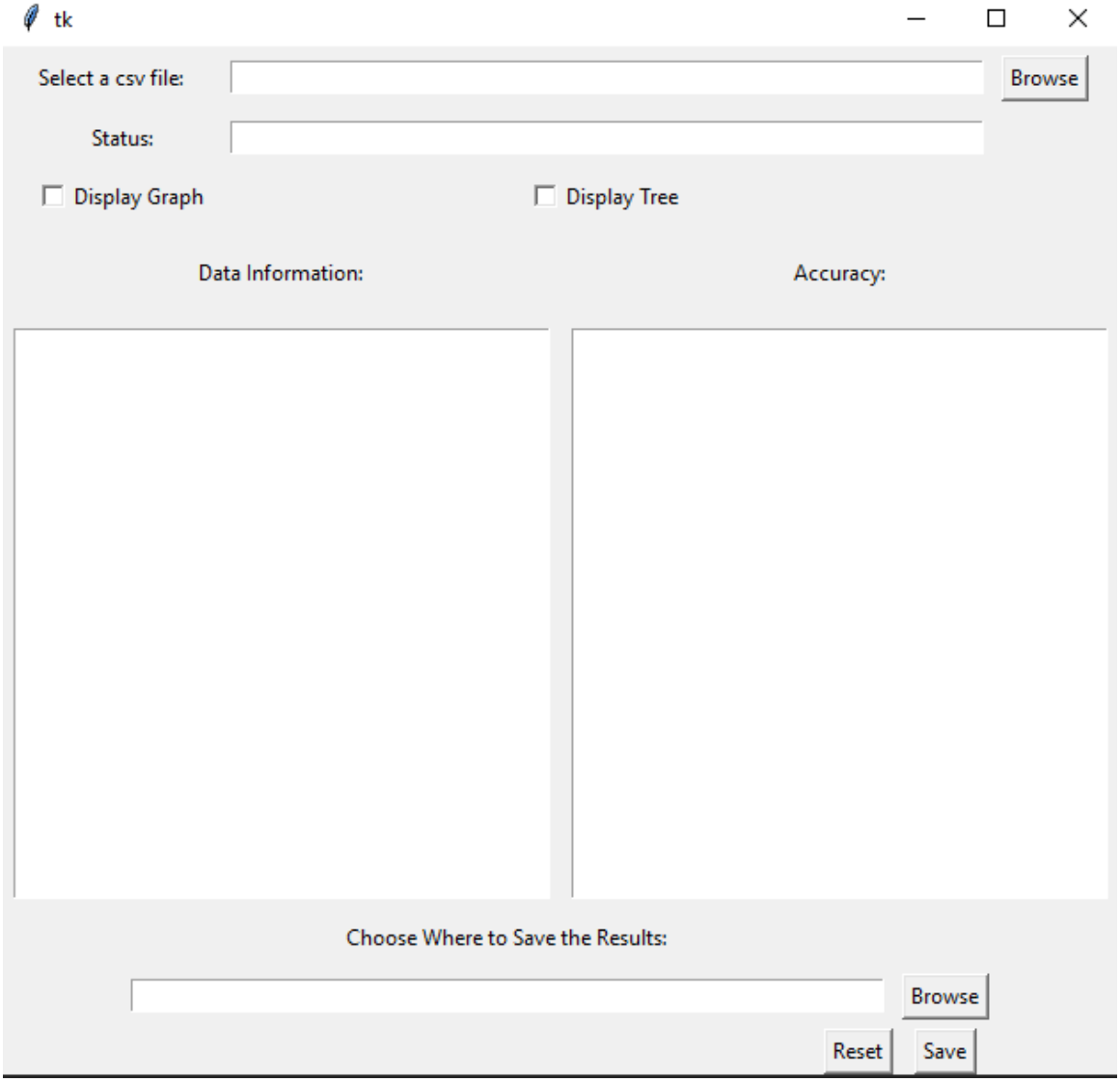
In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred as Gini ratio, which measures the impurity of the node. You can say a node is pure when all its records belong to the same class, such nodes known as the leaf node.

```
import graphviz
dot_data = tree.export_graphviz(classifier, out_file=None,
                                filled=True)
graphviz.Source(dot_data, format="png")
```



Chapter 6: User Interface

To be able to work with models in real time, designing a user interface was ideal. The design goals were to produce a simple, easy to understand, and intuitive to use. The figure below shows the structure of the UI. The GUI uses the tkinter library and runs on Python 3.



UI Fields

1. The Input Data field(Select a CSV File:)

    a. The field on the top of the GUI is where the user selects the data for training the model. Note that the selected file must be a CSV file or an error message will be displayed

2. Status field

    a. Displays a message that lets the user know whether the file they have selected is valid.

3. Display Graph Checkbox

    a. Checking this box will display a Graph upon selecting a valid data file
    b. Note that the user must check this box before selecting a data file in order to display the graph
    c. Displays in a new window

4. Display Tree Checkbox

    a. Checking this box will display the decision corresponding decision tree
    b. Similar to the graph, the box must be checked before selecting a data file
    c. Displays in a new window

5. Data Information field

    a. Shows all the information about the selected Data file
    b. This field will only show information if a valid file has been selected
    c. Automatically updates upon selection of a data file

6. Accuracy field

    a. Displays information related to the accuracy of the model
    b. Automatically updates on selection of a data file

7. Save Location field

    a. Displays the location in which to save the Data and accuracy information

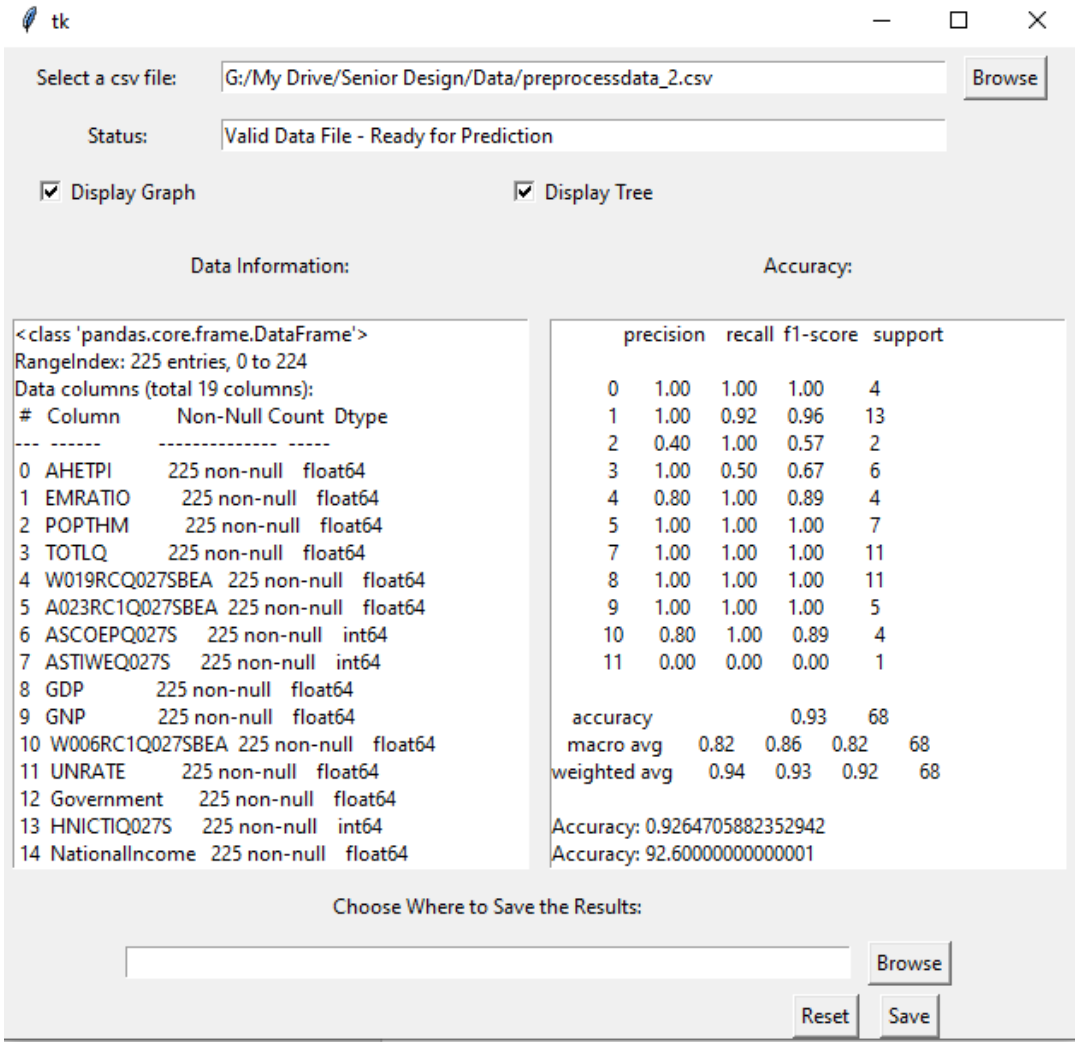
8. Save button

    a. Upon selecting a save location, the data in the Data information field and the accuracy field are saved in the location specified in the previous section

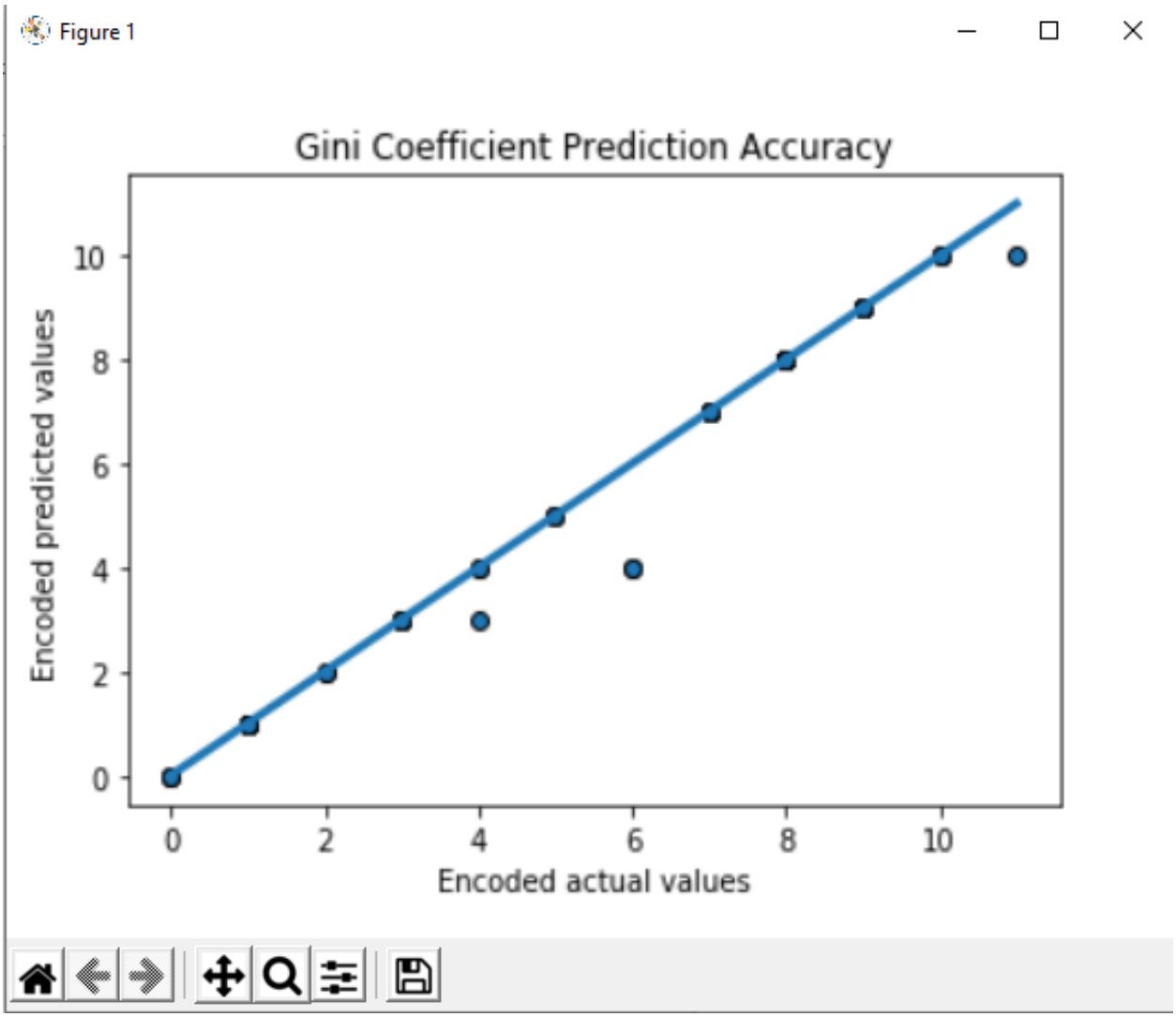
9. Reset Button

    a. Clears all fields

The Figure below shows the UI after selecting a csv file which shows the results



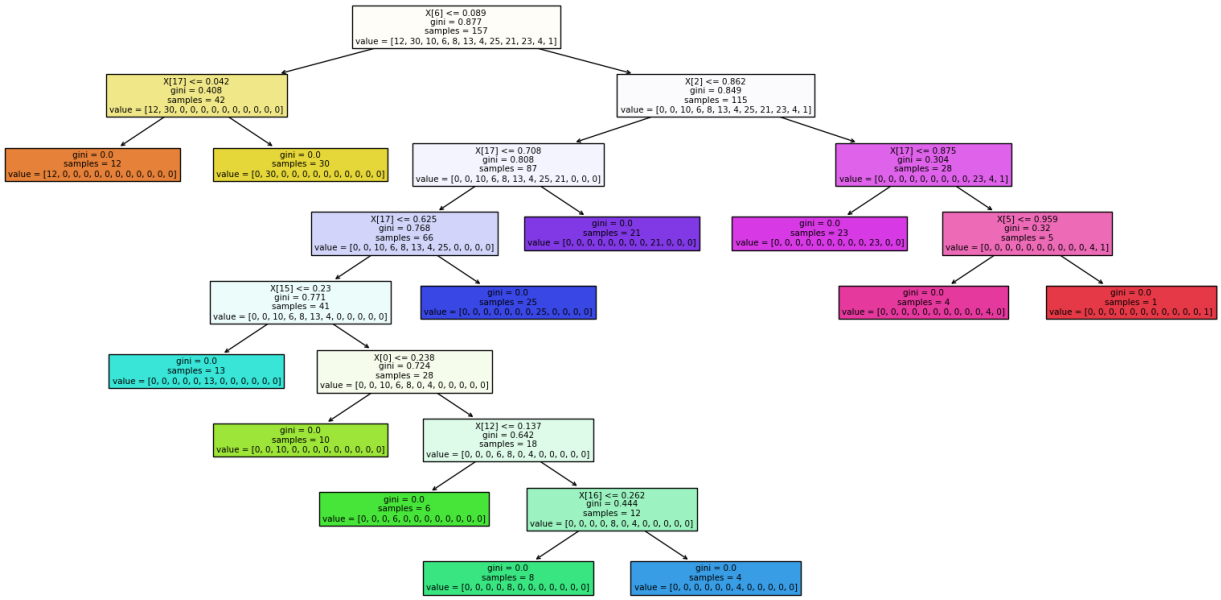
The following figures show an example of the graph and decision tree should the user elect to display them.



This graph represents the accuracy of the model where the x_lable and the y_lable shows the encode value of the GINI coefficients as

Encoded	0	1	2	3	4	5	6	7	8	9	10	11
GINI	0.35	0.36	0.37	0.38	0.39	0.40	0.42	0.43	0.44	0.45	0.46	0.47

For example, At 2 it shows the actual GINI coefficient is 0.37 and our model also predict 0.37.



UI Code Layout:

The GUI window is composed of three frames, the top middle and bottom. The top frames contain the data file selection field and the status field. The middle frame contains two fields where the accuracy information and the data information are displayed. The bottom frame has all the items to select a save location and save the results.

```
lblSelectFile = Label(frameTop, text="Select a csv file:")
lblModelFile = Label(frameTop, text="Model File:")
lblStatus = Label(frameTop, text="Status:")
lblPrediction = Label(frameMiddle, text="Data Information:")
lblAccuracy = Label(frameMiddle, text="Accuracy:")
lblSaveLocation = Label(frameBottom, text="Choose Where to Save the Results:")

btnBrowse = Button(frameTop, text="Browse", command=fileBrowse)
btnSaveBrowse = Button(frameBottom, text="Browse", command=fileSave)
btnSave = Button(frameBottom, text="Save", command=save)
btnReset = Button(frameBottom, text="Reset", command=reset)
entryFilePath = Entry(frameTop, width=70)
entryStatus = Entry(frameTop, width=70)
entrySaveLocation = Entry(frameBottom, width=70)
lstDataInfo = Listbox(frameMiddle, width=50, height=20)
lstAccuracy = Listbox(frameMiddle, width=50, height=20)
chkBoxDisplayGraph = Checkbutton(frameTop, text='Display Graph', variable=boolDisplayGraph, onvalue=True, offvalue=False)
chkBoxDisplayTree = Checkbutton(frameTop, text='Display Tree', variable=boolDisplayTree, onvalue=True, offvalue=False)

lblSelectFile.grid(row=0, column=0, padx=5, pady=5, sticky=W)
btnBrowse.grid(row=0, column=2, padx=5, pady=5)
entryFilePath.grid(row=0, column=1, padx=5, pady=5, sticky=W)
lblStatus.grid(row=2, column=0, padx=5, pady=5)
entryStatus.grid(row=2, column=1, padx=5, pady=5)
chkBoxDisplayGraph.grid(row=3, column=0, padx=5, pady=5)
chkBoxDisplayTree.grid(row=3, column=1, padx=5, pady=5)
lblPrediction.grid(row=3, column=0, padx=5, pady=15)
lblAccuracy.grid(row=3, column=1, padx=5, pady=15)
lstDataInfo.grid(row=4, column=0, padx=5, pady=5)
lstAccuracy.grid(row=4, column=1, padx=5, pady=5)
lblSaveLocation.grid(row=5, column=0, padx=5, pady=5)
entrySaveLocation.grid(row=6, column=0, padx=5, pady=5)
btnSaveBrowse.grid(row=6, column=1, padx=5, pady=5)
btnReset.grid(row=7, column=0, sticky=E)
btnSave.grid(row=7, column=1)
```

Future Changes:

Currently, the model does not have options to insert the dependent and independent variable fields individually by the user. Currently, those values can only be entered within the code itself. Future versions of the model will be able to take input fields from the user dynamically to make it more functionable.

Chapter 7: Next Steps

We will continue to optimize the model to improve the accuracy. We would allow users to have the option to pass full data sets of hypothetical values into the model to generate multiple predictions. We would like to publish this as an application that can predict in real time by accepting more attributes in the future.

References

- https://developer.ibm.com/technologies/data-science/articles/data-preprocessing-in-detail/
- https://www.aptech.com/blog/introduction-to-the-fundamentals-of-time-series-data-and-analysis/ https://fred.stlouisfed.org/series/GDP
- http://www.kasimte.com/2020/02/09/linear-regression-from-time-series-data-using-scikit-learn.html

Decision tree:

- **https://towardsdatascience.com/decision-tree-algorithm-explained-83beb6e78ef4**
- **https://en.wikipedia.org/wiki/Decision_tree_learning**
- **https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart**
- **https://www.youtube.com/watch?v=LDRbO9a6XPU&ab_channel=GoogleDevelopers**
- **http://www.ijoart.org/docs/Construction-of-Decision-Tree--Attribute-Selection-Measures.pdf**

Confusion matrix

- **https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/#:~:text=A%20confusion%20matrix%20is%20a,the%20true%20values%20are%20known.**

Classification report

- **https://muthu.co/understanding-the-classification-report-in-sklearn/**

Machine Learning:

- **https://machinelearningmastery.com/machine-learning-in-python-step-by-step/**
- **https://www.youtube.com/watch?v=KNAWp2S3w94&ab_channel=TensorFlow**

Gini Coefficient:

- **https://en.wikipedia.org/wiki/Gini_coefficient**
- **http://www3.nccu.edu.tw/~jthuang/Gini.pdf**
- **https://www.wallstreetmojo.com/gini-coefficient/**
- **https://www.minneapolisfed.org/institute/working-papers-institute/iwp9.pdf**
- **http://www.iariw.org/dresden/gindelsky.pdf**
- **https://www.bea.gov/system/files/papers/WP2018-7_0.pdf**