School of Information Studies
SYRACUSE UNIVERSITY

Course: IST644 Natural Language Process

Assignment: Final Project

Group members: Meichan Huang

Nicholas Nguyen

Eryka Gosselin-Preston

Date due: December 18, 2022

Date submitted: December 20, 2022

Data Report for Final Project on Wine Reviews

Introduction:

For the final project, we chose to work with wine reviews classification tasks. In this report, we will discuss the data acquiring and data wrangling process, then we will discuss the nature of the data. Lastly, we will report the accuracy of the classification models we ran. The goal of this project was to understand how to use descriptive data in wine reviews to predict the outcomes of the following three variables: price, points, and varieties.

Step 1: Data acquisition

In this section, we will describe the data we have used in this project.

For the final project, we used a wine review dataset that was scrapped in 2017 from Wine Enthusiast (available on Kaggle: https://www.kaggle.com/datasets/zynicide/wine-reviews). There were 10 columns in the data and approximately 150K entries of data. The columns included:

Fields	Description
index	Index number
country	Country the wine is produced
description	Detailed comments about the wine
designation	The vineyard within the winery where the grapes that made the wine are from
points	Scores the wine received
price	Price per bottle of wine in dollar amount
province	The province or state that the wine is from
region 1	The wine growing area in a province or state
region 2	The more specific regions specified within a wine growing area (ie Rutherford inside the Napa Valley)
variety	The type of wine
winery	Winery the wine was produced

For our analysis, we focused mainly on the attributes of description, points, and price. We did some descriptive analysis with country and variety as well.

Step 2: Data loading and preprocessing

Data loading: In this step, we started with loading the csv files that were saved in the local directory. More specifically:

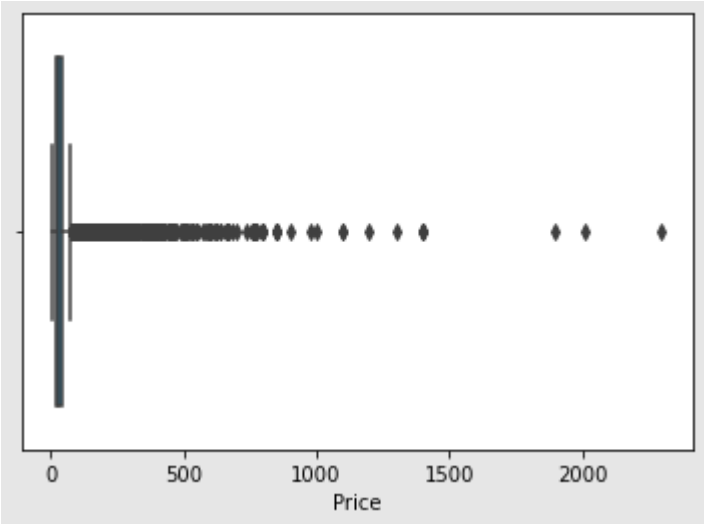
1. Using pd.read_csv(), we loaded the dataset into the Jupyter notebook and named the dataframe as **reviewdf**.

2. Renaming the dataset columns to the following: "Index", "Country", "Review", "Designation", "Points", "Price", "Province", "Region_1", "Region_2", "Variety", "Winery"

Data cleaning and preprocessing: Clean up null values, remove unwanted columns, and preprocess the text column “Review”.

Cleaning up missing data: we first used :is.null.sum() to determine the proportion of missing values from the merged

(1) For “country” and “province” attributes, there were 5 total values missing from the dataset, which is very little. Therefore, we just dropped the missing values.
(2) For “variety” attribute, only 1 value was missing, therefore, the same treatment was done, using the following code:
winereviews.dropna(how="any", subset = ["country", "province"])
(3) For “price” attribute, there were 13643 values missing. That was a significantly larger number of missing values, therefore, it would be problematic if we tried to drop them. Instead, after inspecting the distribution of the dataset using boxplot in seaborn, see below:



we decided to take the mode of the price, using the following code:

```
winereviews['Price'] = winereviews['Price'].fillna(winereviews['Price'].mode()[0])
```

After the imputation, we did descriptive statistical analysis with price and points, and the output were as followed:

	Points	Price
count	150924.000000	150924.000000
mean	87.888474	31.939069
std	3.222233	34.836879
min	80.000000	4.000000
25%	86.000000	16.000000
50%	88.000000	22.000000
75%	90.000000	38.000000
max	100.000000	2300.000000

Text data "Review" preprocess:

In this dataset, the one column that needed most attention was "Review". There were 150k reviews in this dataset. Obviously, there would be a lot of noises with the punctuation, numbers, and stop words if we do not clean up the dataset, which can ultimately impact the results of the text analysis. Therefore, we have taken two types of processing to clean up the data, (1) POS tagging and extract only nouns and adjectives as these are probably the best language features of the wine reviews as each wine review is a description of the reviewer's tastes and feelings about the wine. (2) Tokenized words with all punctuations, numbers, and stop words (including a customized list of words) removed.

(1) POS tagging and extract nouns, adjectives, adjective phrases, and noun phrases: First, we were interested in only the nouns and adjectives used in each review. Therefore, to extract the nouns and adjectives for each review, we had to first use NLTK pos_tagged method to tag all words in the sentences. We did not move any punctuation in this step because it is an important indicator for POS tagging, nor did we eliminate any stop words. The code we used in this step are:

- Lower case all the text:

```
##start with all lower cases
reviewdf['Review'] = reviewdf['Review'].astype(str).str.lower()
```

- Tokenize sentences into words within the nested list and apply POS tagging to the tokens.

```
#Tokenize the sentence and attach POS to each tokens. https://stackoverflow.com/questions/41674573/how-to-ap
reviewdf['TaggedReview'] = nltk.pos_tag_sents(reviewdf['Review'].apply(nltk.word_tokenize).tolist() )
```

It is tricky to work with pandas dataframe since the column has 150k entries; therefore, we had to use apply(nltk.word_tokenize) to all the reviews and then pos_tag_sents tag the sentences within each review.

- Extract nouns and adjectives from each review and saved the results into the new columns "nouns" and "adjectives"

Then we used regex expressions to retrieve two types of POS. To retrieve nouns, we used the POS tags “NN”, “NNS”, “NNP”, AND “NNPS” because they stand for singular nouns, plural nouns, singular proper nouns, and plural proper nouns. Then we used a lambda function to apply this expression to each row in the column “TaggedReview”. The results were saved as a new column “nouns”.

```
#Extract the nouns in each row only:
reviewdf['nouns'] = reviewdf['TaggedReview'].apply(lambda x: [word for word, tag in x if tag in ['NN', 'NNS', 'NNP', 'NNPS']])
```

Then we applied the same procedure to extract all adjectives from the Tagged Review column with regex expression 'JJ', 'JJR', 'JJS'. All adjectives that were retrieved were saved in the new column “adjectives”.

```
#Extract the adjs in each row only:
reviewdf['adjectives'] = reviewdf['TaggedReview'].apply(lambda x: [word for word, tag in x if tag in ['JJ', 'JJR', 'JJS']])
```

- Extract noun phrases and adjective phrases using regex and saved them into new columns “nounphrases” and “adjphrases”.

For the adjective phrases, we used the regex expression: grammar_adjph = "ADJPH: {<RB.?>+<JJ.?>}". Then we used the chunking technique to parse the adjective phrases from the subtrees when the label matches the regex parser we set up. Then all the adjective phrases were extracted and appended back to the panda dataframe reviewdf as “adjphrases”.

For the noun phrases, we used the regex expression grammar_nounph = "NP:{<DT>?<JJ>*<NN>}" and repeated the same process and saved the results to the “nounphrases” column.

(2) Tokenize text attribute: Since we are also interested in looking at some word frequency distributions of the wine description data, we used the NLTK package to perform the following tasks to tokenize each review (which has been lower cased in the first step of POS tagging) to create a bag of words.

- Tokenize using nltk.word_tokenize()

```
reviewdf['tokenized_words'] = reviewdf['Review'].apply(nltk.word_tokenize).tolist()
```

- Using Regex expressions to capture non-alpha characters (punctuations and numbers)

```
#Remove the punctuation and numbers
import re
def alpha_filter(w):
    pattern = re.compile('^[^a-z]+$')
    if (pattern.match(w)):
        return True
    else:
        return False
reviewdf['word_token_cleaned'] = reviewdf['tokenized_words'].apply(lambda x:[i for i in x if not alpha_filter(i)])
```

- Remove stopwords using stopwords from the nltk package
- Then we customized a list once stopwords came with the nltk package were removed. We did a brief inspection of the dataset and noted the following words not meaningful in the analysis: “'drink' , 'now', 'wine' ,'flavour','flavor', "'s",'still', 'flavours', 'flavors', "'ll"”

```
#Remove stopwords and punctuation
from nltk.corpus import stopwords
# Make a list of english stopwords
stopwords = nltk.corpus.stopwords.words("english")
mystopwords = ['drink' , 'now', 'wine' ,'flavour','flavor', "'s", 'still', 'flavours', 'flavors', "'ll", '%']
stopwords.extend(mystopwords)

#Save a new column to store the preprocessed word tokens
reviewdf['word_token_cleaned'] = reviewdf['word_token_cleaned'].apply(lambda x:[i for i in x if i not in stopwords]
```

Once the cleaning was done to all the text data, we saved the cleaned tokens into a new column called “word_token_clean”, which was heavily used in the classification task.

Here is a preview of the winereview dataset I worked with in most of the business questions. The data was also saved as “winereviewupdated.csv” for the ease of retrieval for later analysis:

	Country	Review	Designation	Points	Price	Province	Region_1	Region_2	Variety	Winery	TaggedReview	nouns	adjectives	tokenized_words	word_token_cleaned	adjphrases	nounphrases
1	Spain	ripe aromas of fig, blackberry and cassis are ...	Carodorum Selección Especial Reserva	96	110.0	Northern Spain	Toro	NaN	Tinta de Toro	Bodega Carmen Rodríguez	[(ripe, JJ), (aromas, NN), (of, IN), (fig, NN)...	[aromas, fig, blackberry, cassis, slathering, ...	[ripe, oaky, full, intense, rich, black, heady]	[ripe, aromas, of, fig, ,, blackberry, and, ca...	[ripe, aromas, fig, blackberry, cassis, soften...	[tremendously delicious]	[mac , watson , the memory , a wine , mother ,...
2	US	mac watson honors the memory of a wine once ma...	Special Selected Late Harvest	96	90.0	California	Knights Valley	Sonoma	Sauvignon Blanc	Macauley	[(mac, NN), (watson, NN), (honors, VBZ), (the,...	[mac, watson, memory, wine, mother, gold, colo...	[delicious, balanced, complex, white, dark, pe...	[mac, watson, honors, the, memory, of, a, wine...	[mac, watson, honors, memory, made, mother, tr...	[]	[% , new french oak , fruit , ponzi , aurora ,...
3	US	this spent 20 months in 30% new french oak, an...	Reserve	96	65.0	Oregon	Willamette Valley	Willamette Valley	Pinot Noir	Ponzi	[(this, DT), (spent, VBD), (20, CD), (months, ...	[months, %, oak, fruit, ponzi, aurora, abetina...	[new, french, aromatic, dense, toasty, aromas,...	[this, spent, 20, months, in, 30, %, new, fren...	[spent, months, new, french, oak, incorporates...	[]	[the top wine , la bégude , point , the vineya...

Step 3: Basic descriptive data analysis and further data preprocessing

After data cleaning, the last step is to understand the basic descriptive data of the dataset. In this example, I have used ***df[‘col1’].describe()*** to calculate the max, mean or count, etc. for the attributes, depending on the data types of the attributes, they have different results. I have also used df.std() to obtain the standard deviation for both price and points attributes.

For example, for price and points, which are continuous variables, using describe(), we were able to get the descriptive stats. On the left side, we can see the length, mean, std, min, max and the quantiles of points. We can see that the points wines in this dataset received ranged from 80 – 100, which was anticipated because Wine Enthusiast magazine only included wines that received 80 points or above in their reviews. We could see that the data has a relatively small standard deviation of 3.03, with the median around 89 points and mean at 88.8, which we could assume the data points are normally distributed.

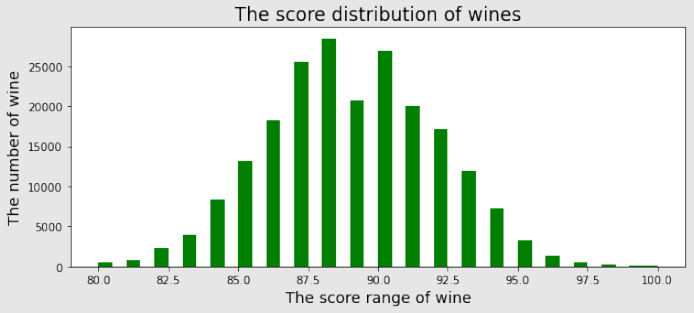
The table on the right side is the descriptive stats for price and we can see this attribute varied in a significant range from 4 – 3400 dollars. The most expensive bottle of wine is 3400 dollars.

```
count    211017.000000
mean         88.806850
std           3.028514
min          80.000000
25%          87.000000
50%          89.000000
75%          91.000000
max         100.000000
Name: points, dtype: float64
```

```
count    211017.000000
mean         35.757456
std          42.733794
min           4.000000
25%          18.000000
50%          25.000000
75%          42.000000
max        3400.000000
Name: price, dtype: float64
```

We also plotted the histogram for the distribution of points and price in order to understand the data more. In the following section,

Treatment for the points attribute: For points the wines in this dataset received, as one can see, the points that wines in the dataset received are normally distributed, with its peak around 88 points.



It is clear that the wine review points in this dataset is somewhat normally distributed. It has its peak between around 87-89. The mean of the points given to the varieties was around 87.9. The maximum was 100 which was very few. The minimum was 80, which was again very few. And the Standard deviation of 3.22 suggests that the points given to each wine was 88 (approx.) +/- 3 which means anything ranging from 85 to 91 which is a very good score.

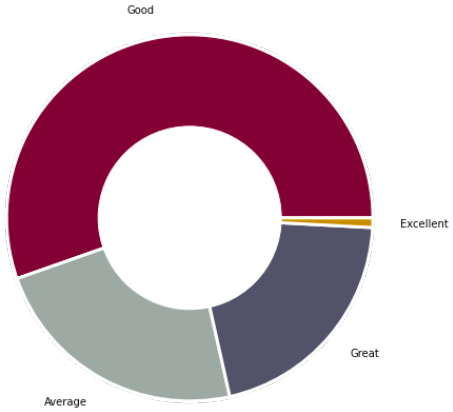
Based on the results, we could divide the variable of Points (continuous) into a categorical variable when conducting classification tasks:

- 80 to 85 points = "average"
- 85 to 90 points = "good"
- 90 to 95 points = "great"
- 95 to 100 points = "excellent

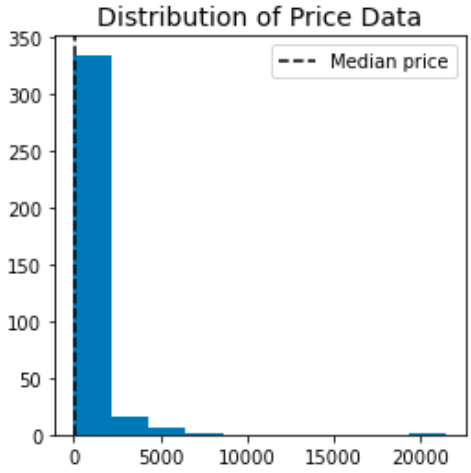
Using the following code, we created a new column called Quality to house the four categorical labels of all wines based on the points they received.

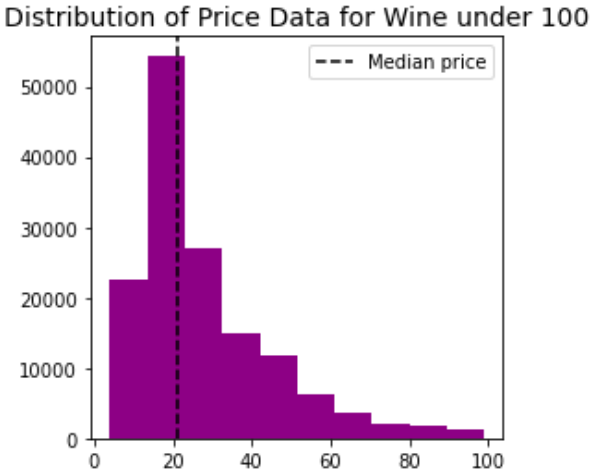
```
#First, we group the Point datapoints based on the following criteria and create a new column called Quality
reviewdf1.loc[(reviewdf1['Points'] > 80) & (reviewdf1['Points']<=85), 'Quality'] = 'Average'
reviewdf1.loc[(reviewdf1['Points'] > 85) & (reviewdf1['Points'] <=90), 'Quality'] = 'Good'
reviewdf1.loc[(reviewdf1['Points'] > 90) & (reviewdf1['Points'] <=95), 'Quality'] = 'Great'
reviewdf1.loc[(reviewdf1['Points'] > 95) & (reviewdf1['Points']<=100), 'Quality'] = 'Excellent'
reviewdf1 = reviewdf1[reviewdf1['Quality'].apply(lambda x: type(x) == str)]
```

We also graphed a pie chart to visualize the number of data in each category. We can see that the majority of the data has fallen into good, great, and excellent, and only a few were categorized in average.



Treatment for the price attribute: Whereas for the price distribution of wines in the data, we can see that the data is highly skewed to the right. The number of high-priced wines is barely invisible in the histogram. When zooming in to only visualize the distribution of wine price under 100 dollars, we can clearly see the data is positive skewed with most of the wine at 15-25 range and the number gradually declined from 40 – 100 dollars.





The most common price point for the wines in the dataset is around 20 dollars. So based on the descriptive analysis of the Price, the mean is around 30 which is much below 50. The data also has a very high standard deviation i.e. 34 so the prices vary from 4 to 60 dollars.

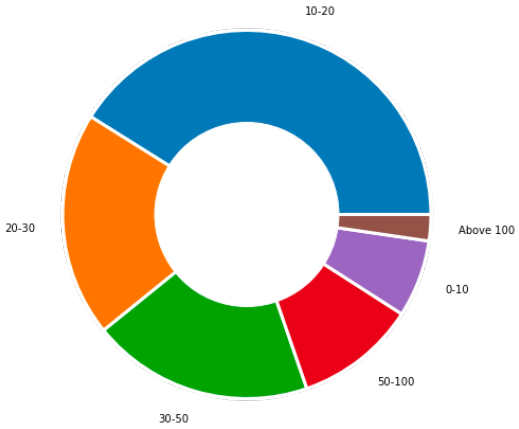
Based on this information, we divided up the price into a categorical variable as follows:

- 0-10
- 10-20
- 20-30
- 30-50
- 50-100
- Above 100

Using the following codes, we were able to create a new column called Price_val:

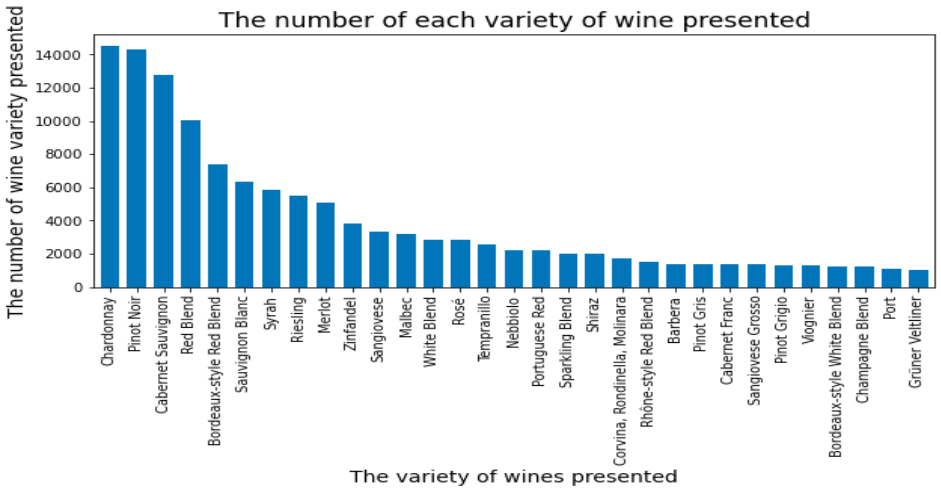
```
#Next, we group the Price datapoints based on the following criteria and create a new column called Price_val
reviewdf1.loc[(reviewdf1['Price'] > 0) & (reviewdf1['Price'] <=10) , 'Price_val'] = '0-10'
reviewdf1.loc[(reviewdf1['Price'] > 10) & (reviewdf1['Price'] <=20) , 'Price_val'] = '10-20'
reviewdf1.loc[(reviewdf1['Price'] > 20) & (reviewdf1['Price'] <=30) , 'Price_val'] = '20-30'
reviewdf1.loc[(reviewdf1['Price'] > 30) & (reviewdf1['Price'] <=50) , 'Price_val'] = '30-50'
reviewdf1.loc[(reviewdf1['Price'] > 50) & (reviewdf1['Price'] <=100) , 'Price_val'] = '50-100'
reviewdf1.loc[(reviewdf1['Price'] > 100), 'Price_val'] = 'Above 100'
```

After dividing up the price column, we did a pie chart to visualize the distribution of the price category. We can see that 1/3 of the data fell into the category above 100 dollars and the rest were equally distributed into the 10-100 range. Only a small proportion of the data has $0-10 price tags.



Treatment of variable attribute: The last variable that we did analysis on was the variety of wine. In total, there were 643 varieties of wine, and the below is a bar plot of the counts of varieties of wines that have over 1000 reviews only because it was impossible to plot all the wines that have one or two entries of reviews only.

In addition, the first four that had over 10k reviews each were: Chardonnay, Pinot Noir, Cabernet Sauvignon, and Red Blend. We subset these four top-reviewed wines later in our classification models.



Word frequency distribution

In this section, we discuss the distribution of unigrams frequencies, 20 most frequent nouns, 20 most frequent adjectives, 20 most frequent noun phrases and 20 most frequent adjective phrases. Overall, the data contained 3,441,539 tokens, just over 3.4 million tokens after we removed all punctuations, stop words, and numbers, which is quite impressive for its size. There were 1,980,620 tokens of nouns in the reviews, almost 2 million tokens. Adjectives are fewer in the corpus, which is just under 1 million (981,885) tokens.

Then we did a frequency distribution for all the categories of tokens we extracted from the wine reviews. The following is a brief discussion of the features:

For the 20 most frequent unigrams, we can see that fruit and finish are the two most frequent words, followed by aromas, acidity, and tannings. These are common descriptors for wines, which is not surprising. In addition, the majority of the most frequently-used words are either nouns or adjectives. Then we started to see more descriptors specific to certain wines, e.g. cherry, ripe, black, dry, spice, etc. There are two assumptions that could help us with the classification tasks:

(1) Nouns and adjectives are more important than the other POS in the wine review classification.
(2) The results might indicate that reviews could work well with classifications of wine varieties, because the descriptors get more specific to describe certain types of wines going down the tokenized word list.

We also plotted a word cloud to visualize the most common tokens in this corpus and we can see the most frequent words in the biggest fonts.

```
[('fruit', 56505),
 ('finish', 37724),
 ('aromas', 35820),
 ('acidity', 32602),
 ('tannins', 32181),
 ('cherry', 30659),
 ('palate', 29008),
 ('ripe', 26720),
 ('black', 24590),
 ('dry', 22978),
 ('spice', 22643),
 ('sweet', 21286),
 ('rich', 21172),
 ('oak', 19675),
 ('notes', 19606),
 ('red', 19187),
 ('soft', 17745),
 ('fresh', 17666),
 ('good', 17291),
 ('berry', 17098)]
```

For the 20 most common nouns, we can see a lot of overlapping in this list compared to the tokenized words. Overall, we can see that the frequent nouns have a huge overlap with the most 20 common unigram tokens, in which fruit and finish ranked 3rd and 4th in this frequent noun list, tannins, cherry, palate, spice, notes, and oak as well are overlapping with the word tokens. We suspect that this would cause the performance of using nouns as input for the classification to yield comparable results as using the tokenized words as input.

Since we did not remove any stop words when POS tagging the sentences, there are a few words that could be considered stop words that could be removed from the results, e.g. wine, and flavors. Also, interestingly, the sign "%" was tagged as nouns. We think the NLTK POS tagging performed fine, but there were some inconsistency, as we could see the mismatches between the numbers of nouns and the number of tokens of the same words, e.g. (finish as a noun has 33,140 tokens and finish has a total of 33,7724 tokens and some of them might be verbs); it would be nice if we could do the backward tagging to remove some of these words, which might yield better results extracting the nouns. Here is the top 20 frequent noun list and the word cloud side by side. The size of the fonts in the word cloud indicates the frequency of the nouns in the corpus. The larger the fonts, the more frequent the nouns.

```
[('wine', 88260),
 ('flavors', 77815),
 ('fruit', 56053),
 ('finish', 33140),
 ('acidity', 32557),
 ('aromas', 32195),
 ('tannins', 31111),
 ('palate', 28440),
 ('cherry', 26067),
 ('spice', 22030),
 ('notes', 19508),
 ('oak', 18153),
 ('berry', 16106),
 ('%', 15768),
 ('blackberry', 14217),
 ('blend', 13285),
 ('vanilla', 13209),
 ('plum', 13032),
 ('years', 12759),
 ('fruits', 12147)]
```

For the 20 most frequent adjectives, we can see that "black" and "dry" are the most frequent adjectives, which overlapped with the tokenized words list. The adjective list contained mostly descriptors of the color and taste of wines. However, "ripe" was not at the top of the list with only 17069 tokens, as compared to 26720 tokens in the unigram, which was quite surprising, because ripe can only be used as adj. In the word cloud, we can see the black, fresh, dry, and red and rich are the most frequent adjectives based on the size of their fonts.

```
[('black', 24590),
 ('dry', 21426),
 ('rich', 20957),
 ('red', 18328),
 ('soft', 17710),
 ('fresh', 17511),
 ('good', 17258),
 ('ripe', 17069),
 ('sweet', 16854),
 ('white', 11852),
 ('green', 10642),
 ('full', 9017),
 ('bright', 8884),
 ('dark', 7756),
 ('clean', 7709),
 ('light', 7656),
 ('tannic', 7508),
 ('fine', 7370),
 ('more', 7210),
 ('great', 6879)]
```

For the 20 most frequent bigrams, we see that the most frequent bigram two types (1) wine descriptors regarding their flavors: “black cherry”, “black fruit”, “ripe fruit”, “berry fruit”, “tropical fruit”, “red pepper”, “green apple” etc, and (2) wine varieties: cabernet sauvignon, pinot noir, sauvignon blank.

```
(('black', 'cherry'), 0.00180762153211107)
(('cabernet', 'sauvignon'), 0.0013726417163949037)
(('pinot', 'noir'), 0.0012305541212812058)
(('black', 'fruit'), 0.0009431826865829502)
(('cherry', 'fruit'), 0.0007865667075107967)
(('sauvignon', 'blanc'), 0.0007836610307191056)
(('black', 'currant'), 0.0007615778871022528)
(('crisp', 'acidity'), 0.0007429815556354294)
(('ripe', 'fruit'), 0.0007206078443394075)
(('berry', 'fruit'), 0.0006947473208933562)
(('tropical', 'fruit'), 0.0006656905529764445)
(('red', 'fruit'), 0.0006642377145805989)
(('firm', 'tannins'), 0.0006636924352738702)
(('green', 'apple'), 0.0006357620820220256)
(('black', 'pepper'), 0.000611644964650989)
(('barrel', 'sample'), 0.0005761957077923569)
(('stone', 'fruit'), 0.0005517880227421511)
(('red', 'berry'), 0.0005294143114461292)
(('blackberry', 'cherry'), 0.0005143047921293351)
(('long', 'finish'), 0.0005122708183751514)
```

Groupby results

In order to organize and gather information easier from our datasets we decided to use the groupby function within python for various variables. The grouping of variables was by Country, Points, Price and Variety. Groupby also allows us to extract information from each category with ease. For example, to figure out which countries had the lowest or highest points, we will run the function along with .min and .max functions.

Countries with lowest points (80): Argentina, Australia, Bulgaria, Chile, Cyprus, Germany, Greece, Hungary, Israel, Italy, Mexico, New Zealand, Portugal, Romania, South Africa, Spain, US, and Uruguay.

Countries with the highest points (100): Australia, France, Italy, and the US.

Groupby helps organize all the countries together and gives us an output of the variable we desire much easier than reading through the csv file by row ourselves. Next, we used the same function to figure out which countries had the cheapest and most expensive bottles of wine.

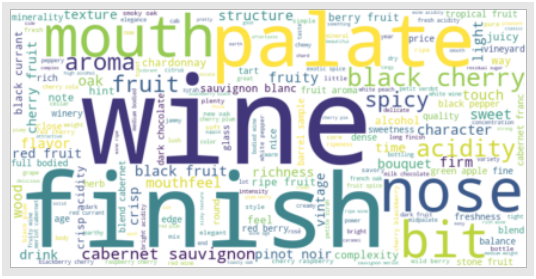
Countries with the cheapest wine ($4): Argentina, Portugal, Romania, Spain and the US.

Countries with the most expensive wine ($2300): France

There were some commonalities between the points and price that we found interesting. The countries that we had the cheapest wine did in fact also have the lowest points. While the country with the most expensive wine, France, had the highest score you could possibly achieve.

Step 3: Generate Word Clouds

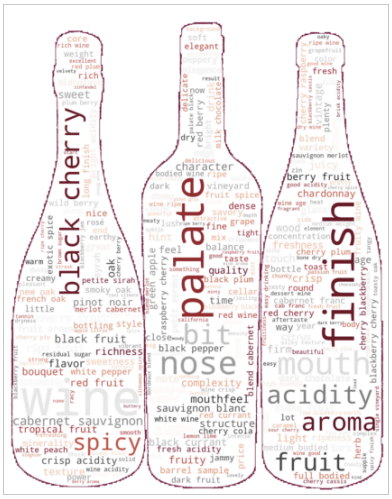
We also wanted to have some visualization of the word frequencies, so we generated some word clouds with noun and adjective phrases that we extracted from all wine reviews. By using the matplotlib, numpy, pandas and wordcloud packages, we created basic word cloud images. As the default background is set to black, we changed it to white in order to make the cloud easier to read.



From there, we masked the word clouds into a shape by converting jpeg images into a numpy arrays, changed the text color themes and the amount of text within each image.

For the adjective phrases word cloud, we used a wine bottle and glass jpeg, set the maximum word count to 250 words, changed the text color pattern using the ‘twilight’ colormap theme from the matplotlib library, and used a light grey contour color to mimic the appearance of white wine.

The same method was used to configure the word cloud for noun phrases. We used a different image to create the shape, set the maximum word count at 300 words, used the ‘RdGy’ colormap, and made the contour a dark red hue to imitate bottles of red wine.



Findings

By using word clouds, we were able to visualize the sentiment, commonalities and themes of the wine reviews in our dataset. The different sizes and colors of text represent the prevalence and importance of commonly used words and phrases. The most used adjectives revolved around taste followed by sentiment rather than other attributes such as appearance and smell. Sweet and dry tend to be used when pairing wine with food, while terms like rich and soft refer to the level of complexity and fullness. Other terms such as good and almost could insinuate the reviewer's sentiment about their experience purchasing wine or how they felt about the wine itself. As for the most frequently used nouns, there appeared to be more variety among the types of words used. For obvious reasons, wine would be one of the most commonly used nouns, as well as the terms finish and palate since they are the most significant characteristic in assessing wine quality. Overall, we found the word clouds to be highly effective in visualizing text data to get a general sense of all that was written in the reviews.

Step 4: Classification Tasks

In this step, we discuss our results with the classification models we did.

As we mentioned in step 3 of data descriptive analysis, it would be infeasible to classify the review data using individual points and price, since there was a huge variation among the numeric variables. Also, since the points and price ranges can be vast, it would be more helpful to give the range specific values to each one. Therefore, we made the decision to turn the continuous variables of “price” and “points” into categorical variables. These new columns are defined as wine_quality and price_val. make running our classification tools run without any errors.

We used 2 separate dataframes to see different classification results. The first dataframe used is “reviewdf2”, and the second dataframe used is “classifydf”. The reason for using different dataframes is because we wanted to test the classification models on both the pretokenized

dataframe (reviewdf2) and the tokenized dataframe (classifydf). Most of the results were most successful using the tokenized dataframe, which is what we included in our final Jupyter notebook for all the codes and in this report.

We used the Naïve Bayes model for our classification tasks. To begin we created a make_pipeline that has the TF-IDF vectorizer and Multinomial Naïve Bayes functions inside. This was the main model that we used for our classification of various variables and our code for modeling:

```
#Our model using multinomialNB and Tfidf vectorizer
model2 = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

We chose to use TF-IDF vectorizer as we thought it was a better option than Count Vectorizer. TF-IDF means Term Frequency - Inverse Document Frequency, which is statistic based on the frequency of a word in the corpus. In addition, it provides a numerical representation of the importance of a word for statistical analysis. TF-IDF is better than Count Vectorizers because it not only focuses on the frequency of words present in the corpus but also provides the importance of the words. Words that were less important for analysis were removed, hence making the model building less complex by reducing the input dimensions.

For each model training, we first built a new dataframe to only include the two columns that we'd like to model. The example below is our first model we ran with "word_token_cleaned" and "Quality". Then we used the following codes to split the train test set by 70% and 30% and randomized the dataset.

```
#Split the train test sets for both x and y variables
x1 = classifydf1.word_token_cleaned
y1 = classifydf1.Quality
x_train1, x_test1, y_train1, y_test1 = tts(x1, y1, test_size=0.30, random_state=1, stratify=y1)
```

Next, we trained the model with x_train and y_train and made predictions with x_test set. Lastly, we calculated the accuracy score for the classification model and the model accuracy score which was presented in percentage.

```
#Train and predict
training1 = model2.fit(x_train1, y_train1)
predicted1 = model2.predict(x_test1)
scores_qual1 = 100 * model2.score(x_test1,y_test1)
```

We also ran several codes to obtain the classification results:

(1) Cross-validation scores with 10 kfolds:

```
#Calculate the cross-validation score for the model with tokenized reviews and wine quality:
tokenscores1 = cross_val_score(model2, x_train1, y_train1, cv = 10, scoring='accuracy')
print('Cross-validation scores for the model with tokenized words with kfolds of 10 is:{}'.format(tokenscores1))
print('Average cross-validation score for this model with kfolds of 10 is: {:.4f}'.format(tokenscores1.mean()))
```

(2) Cross-validation scores with 5 kfolds:

```
tokenscores2 = cross_val_score(model2, x_train1, y_train1, cv = 5, scoring='accuracy')
print('Cross-validation scores for the model with tokenized words with kfolds of 5 is:{}'.format(tokenscores2))
print('Average cross-validation score for this model with kfolds of 5 is: {:.4f}'.format(tokenscores2.mean()))
```

(3) Classification report that contained precision, recall, f1-score, and support scores for the said model.

```
 score of Mulitnomial Naive Bayes for the quality of wine is with tokens is {} %".format(scores_qual1))
el accuracy score with tokens to predict the quality of wine is: {0:0.4f}'.format(accuracy_score(y_test1, predicted
sification_report(y_test1, predicted1))
```

After building the model, we first tested the model by classifying wine_quality with the variety of wines, just to test if the model would run smoothly without error codes. Of course, that is not any of the models we intended to run, but still, we can see the results are less than favorable.

```
Cross-validation scores:[0.55551324 0.55703675 0.55632261 0.55539685 0.55506356]
Average cross-validation score: 0.5559
```

	precision	recall	f1-score	support
Average	0.56	0.03	0.05	10412
Good	0.56	0.98	0.71	24925
Great	0.48	0.04	0.08	9292
Perfect	0.00	0.00	0.00	379
accuracy			0.56	45008
macro avg	0.40	0.26	0.21	45008
weighted avg	0.54	0.56	0.42	45008

With a k-fold of 5 the average cross-validation score was only 0.5559. As one can tell, the prediction was not able to predict any of the "Excellent" scores given for the classification. This is a problem that we did not want, so we tried a multitude of different classification predictions using various variables to try and find the best accuracy we can get.

In the following sections, we will report the best scores of three groups of modeling we did using tokenized wine reviews as input and wine quality (categorized points), price_val (categorized price), and wine variety. We will report our results from the worst performed model with reviews to predict wine price to the best performed model with reviews to predict wine varieties.

Classification using reviews to predict wine price_val

In this section, we will report the classification results of wine reviews to predict the labeling of wine price, using the model Nicholas has built using TFIDF vectorizer. As we mentioned before, we categorized the price that each wine received into 6 categories "0-10", "10-20", "20-30", "30-50", "50-100" and "Above 100". We also preprocessed the wine reviews to only contain tokenized words (with all punctuations, numbers, and stop words removed). Then we carried out three models, all using Naïve Bayes algorithm, to see which texts helped best predict the price of wine.

(1) Modeling with tokenized words and wine price using Naïve Bayes
(2) Modeling with nouns and wine price oing Naïve Bayes
(3) Modeling with adjectives and wine price using Naïve Bayes

For this modeling, we had the best score with tokenized words and price_val. With a wide range of prices and the number of words used in the reviews we knew the accuracy for this specific classification would be rather low. Also, due to the imbalance in the dataset (very small number

of above 100 dollars in the dataset), we have 0 in precision score and f1-score in the category of "Above 100 dollars".

```
The score of Mulitnomial Naive Bayes for predicting the price of wine with tokenized words on
ly is 49.32900817632421 %
Model accuracy score for this model is: 0.4933
/Users/meichanhuang/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/_classificatio
n.py:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

        0-10       0.83      0.00      0.01      3050
       10-20       0.50      0.95      0.65     18496
       20-30       0.56      0.08      0.14      8883
       30-50       0.45      0.41      0.43      8758
      50-100       0.61      0.06      0.12      4782
   Above 100       0.00      0.00      0.00      1039

    accuracy                           0.49     45008
   macro avg       0.49      0.25      0.22     45008
weighted avg       0.52      0.49      0.39     45008
```

Similarly, none of the parts of speech (nouns and adjectives) performed well either. In fact, they had lower accuracy scores compared to just using tokenized words, as we can see from the snapshot of the results from the adjectives predicting wine prices. Interestingly, this time we were able to capture the precision scores for all categories and in fact, adjectives have the highest precision score in predicting price over 100 dollars.

```
The score of Mulitnomial Naive Bayes for predicting the price of wine with adjective tokens only is 46.42507998578031 %
Model accuracy score for this second model is: 0.4643
              precision    recall  f1-score   support

        0-10       0.64      0.01      0.02      3050
       10-20       0.47      0.95      0.63     18496
       20-30       0.46      0.07      0.12      8883
       30-50       0.42      0.28      0.33      8758
      50-100       0.49      0.06      0.10      4782
   Above 100       0.75      0.00      0.01      1039

    accuracy                           0.46     45008
   macro avg       0.54      0.23      0.20     45008
weighted avg       0.48      0.46      0.36     45008
```

Classification using reviews to predict wine quality

In this section, we will report the classification results of wine reviews to predict the labeling of wine quality, using the model Nicholas has built using TFIDF vectorizer. As we mentioned before, we categorized the points that each wine received into 4 categories "average", "good", "great", and "excellent". We also preprocessed the wine reviews to only contain tokenized words (with all punctuations, numbers, and stop words removed). Then we carried out three models, all using Naïve Bayes algorithm, to see which texts helped best predict the quality of wine.

(4) Modeling with tokenized words and wine quality using Naïve Bayes
(5) Modeling with nouns and wine quality using Naïve Bayes
(6) Modeling with adjectives and wine quality using Naïve Bayes

Overall, using wine reviews, we got an accuracy score ranging from 62.85% (nouns to predict wine quality) to 67.75% (tokenized words to predict wine quality), with tokenized words performed the best in predicting the wine quality. The precision scores for each wine quality

label can be found in the classification report below. We did run into the issue with having 0 in the Excellent category, which means there was not enough data in this category to support the classification.

```
Model accuracy score with tokens to predict the quality of wine is:
0.6775
```

```
/Users/meichanhuang/opt/anaconda3/lib/python3.9/site-packages/sklear
n/metrics/_classification.py:1318: UndefinedMetricWarning: Precision
and F-score are ill-defined and being set to 0.0 in labels with no p
redicted samples. Use `zero_division` parameter to control this beha
vior.
  _warn_prf(average, modifier, msg_start, len(result))
```

	precision	recall	f1-score	support
Average	0.87	0.39	0.54	10412
Excellent	0.00	0.00	0.00	379
Good	0.64	0.96	0.77	24925
Great	0.78	0.27	0.41	9292
accuracy			0.68	45008
macro avg	0.58	0.40	0.43	45008
weighted avg	0.72	0.68	0.63	45008

However, this is only the average performing model in our experiment.

Classification using tokenized words to predict wine variety

Lastly, we ran three models to use wine reviews to predict the top 4 varieties of wines. As we mentioned in the descriptive data report, there were over 600 varieties of wines, and it would be impossible to model the data with so many vectors. Therefore, to make the classification tasks easier, we only used the top four varieties of wine, namely “Cabernet Sauvignon”, “Chardonnay”, “Pinot Noir”, and “Red Blend”, because these 4 had the most reviews at over 10,000.

(7) Modeling with tokenized words and top 4 wine varieties using Naïve Bayes
(8) Modeling with nouns and top 4 wine varieties using Naïve Bayes
(9) Modeling with adjectives and top 4 wine varieties using Naïve Bayes

Our highest score that we were able to achieve was an accuracy score of 88.75%. This came when we decided to use the word_token_cleaned variable with the Variety of wines. The classification report is also shown below. All categories of wine varieties received over 0.85 point in precision scores, with weighted average of 0.89, which is excellent outcome for us.

```
The score of Mulitnomial Naive Bayes for predicting the variety of wine with tokenized words only is 88.74610591900311 %
Model accuracy score for this second model is: 0.8875
```

	precision	recall	f1-score	support
Cabernet Sauvignon	0.81	0.86	0.83	3813
Chardonnay	0.98	0.98	0.98	4323
Pinot Noir	0.87	0.93	0.90	4276
Red Blend	0.88	0.73	0.80	2996
accuracy			0.89	15408
macro avg	0.89	0.87	0.88	15408
weighted avg	0.89	0.89	0.89	15408

This was exciting to see as the variety of wine was able to help coincide with the words they

were being reviewed as at a high rate of prediction. Using the top 4 wine varieties also assisted in creating a high accuracy score for our model. Instead of using all the wine varieties, the top 4 varieties created a relatively balanced dataset and were able to predict their reviewed words.

Next, we wanted to compare classification results of nouns and adjectives to the top 4 wine varieties. We can see that the accuracy score for nouns predicting the varieties performed better in this model, compared to the adjectives. We suggested that it is possible that the tokenized words may have already contained some descriptors for wines that were easy to categorize, based on our observations in the word frequency distribution. When it comes to the parts of speech accuracy score, nouns had about a 17% higher score. This is surprising because there were more nouns than adjectives within our dataset, yet the prediction of the correct nouns to variety was higher than adjectives.

(1) Accuracy score and classification report for nouns predicting wine varieties.

```
The score of Mulitnomial Naive Bayes for predicting the variety of wine with nouns only is 87.54543094496366 %
Model accuracy score for this second model is: 0.8755
                    precision    recall  f1-score   support

Cabernet Sauvignon       0.81      0.83      0.82      3813
        Chardonnay       0.96      0.97      0.96      4323
        Pinot Noir       0.86      0.92      0.89      4276
         Red Blend       0.86      0.73      0.79      2996

          accuracy                           0.88     15408
         macro avg       0.87      0.86      0.87     15408
      weighted avg       0.88      0.88      0.87     15408
```

(2) Accuracy score and classification report for adjective predicting wine varieties.

```
The score of Mulitnomial Naive Bayes for predicting the variety of wine with adjectives only is 70.18431983385256 %
Model accuracy score for this second model is: 0.7018
                    precision    recall  f1-score   support

Cabernet Sauvignon       0.61      0.64      0.62      3813
        Chardonnay       0.81      0.86      0.83      4323
        Pinot Noir       0.64      0.72      0.68      4276
         Red Blend       0.79      0.53      0.64      2996

          accuracy                           0.70     15408
         macro avg       0.71      0.69      0.69     15408
      weighted avg       0.71      0.70      0.70     15408
```

We suggested that the high accuracy could be explained by that fact the tokenized words contained certain descriptors that are used to describe certain types of wines, which helped when the algorithm tried to classify them into different wine varieties.

SVM models with wine reviews and wine varieties (did not work out as expected)

As per the professor's feedback, we tried again with svm algothrism with a smaller sample set. We took the best performing model with wine reviews predicting wine varieties and subset the data to 10% of its original size using the following codes. We shuffled the data so that it was randomized and include a balanced dataset of all four varieties of wines.

```
classifydf10 = classifydf.loc[:, ['word_token_cleaned', 'Variety']]

#Choose the top four with more than 10k reviews each and subset the data
wines = ['Chardonnay', 'Pinot Noir', 'Cabernet Sauvignon', 'Red Blend']
classifydf10 = classifydf10[classifydf10.Variety.isin(wines)]

shuffled = classifydf10.sample(frac=0.1, random_state=1).reset_index()
print(shuffled.head())
values = shuffled['Variety'].value_counts()
print(values)
```

The data included around 1000 reviews for each of the categories.

After building the model with svm with “rbf” kernal, the results came out only 29% accurate and the precision scores were 0 in predicting 3 out of 4 wine varieties. Therefore, we felt the codes could be modified to get better output, but due to our limitations, we left this part out for the professor’s input (see the Jupyter notebook submission).

Future research

To conclude, there is still so much to be desired for this project. For instance, there are a few areas that could be researched on:

(1) We could have used noun phrases and adjective phrases for the classification tasks.
(2) We could remove wine varieties names from the tokenized words and see if that changes the classification accuracy.
(3) We could vary the data sample size and adjust on the categories we had.
(4) Try other classification algorithms, such as Decision Tree.

References

Kaplan, D. (2022). Machine Learning 101: CountVectorizer Vs TFIDFVectorizer. Retrieved from: https://enjoymachinelearning.com/blog/countvectorizer-vs-tfidfvectorizer/#:~:text=CountVectorizer%20simply%20counts%20the%20number,is%20to%20the%20whole%20corpus.

Kiren, J. (2022). Text mining and sentiment analysis with NLTK and pandas in Python. Retrieved from: https://www.kirenz.com/post/2021-12-11-text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/text-mining-and-sentiment-analysis-with-nltk-and-pandas-in-python/

sklearn.feature_extraction.text.TfidfVectorizer. (2022). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Zackhoutt (2017). Wine Reviews. Retrieved from https://www.kaggle.com/datasets/zynicide/wine-reviews

Appendix A – Team member workload division (please add your own contribution)

Meichan Huang:

(1) Data loading, cleaning, and preprocessing stage work:

- data loading
- data column renaming
- removing null values and impute missing values of prices
- POS tagging the "review" data column and retrieved the adjectives (new column), nouns (new column), and created functions to retrieve the adjective phrases (new column), and noun phrases (new columns)
- tokenize the "review" data column and remove all punctuation, numbers, and stopwords with customized stopwords list and created a new column called word_token_cleaned.
- save the cleaned and preprocessed attributes into a dataset called winereviewcategorized.csv for the group to work with.

(2) Created the word cloud for tokenized words, nouns, and adjectives.

(3) Ran 9 classification models with wine reviews and other variables in the dataset:

- 3 models with quality attribute: tokenized words and quality attribute, adjectives and quality attribute, and nouns and quality attribute using naïve bayes method
- 3 models with price_val: tokenized words and price_val attribute, adjectives and price_val attribute, and nouns and price_val attribute.
- 3 models with top 4varieties of wines: tokenized words and varieties attribute, adjectives and varieties l attribute, and nouns and varieties attribute.
- Collaborated with Nicholas to run the last svm model that did not work out well.

Nicholas Nguyen:

(1) Descriptive data analysis with grouping pricing and points based on other attributes: countries, varieties.

(2) Created categorized price and points columns is as "quality" and "price_val" for classification analysis.

(3) Created the original classification codes for all naïve bayes modeling and adopted TF-IDF vectorizer.

(4) Run the naïve bayes classification model with wine quality and variety attributes.

(5) Created heat maps for the confusion matrix for all top-performing models in each of three modeling.

(6) Test run the svm classification model, but the model took too long to run and did not have any presentable results.

Eryka Grosselin-Preston:

(1) Created the word clouds based on adjective phrases and nouns phrases.

(2) Created PowerPoint slides for the group presentation.

Appendix – B Datasets for the Classification Tasks

1. Original dataset from Kaggle: winemag-data_first150k.csv
2. Tokenized and POS tagged dataset: winereview150k_updated.csv
3. Final dataset for classification in which categorized price and point variables included: winereviewcategorized.csv