Wine Analysis and Recommendations

Outline

It is a dataset containing wine reviews published to Wine Enthusiast (Wine Enthusiast, or WE for short, is an American magazine that invites professional wine tasters to rate wines every year), including taster name, wine price, variety, score, country, county, vineyard, etc. There are a large number of red wine varieties and brands, and even the same region can have many different red wines. Accordingly, different tasters have tasted different wines. Perhaps even a taster can't taste all the wines. In order to improve efficiency and let the tasters taste a satisfactory wine, I would like to develop a content-based method to recommend wines among tasters. At the same time, I would like to make a recommendation system for recommending wines to people at different stages in the field(tasting wine).

Analysis

Before we recommend it, let's analyse this dataset.

Basic Analysis of the Data Set

In [54]:
```
unique_countries = df['country'].unique()
num_countries = len(unique_countries)
print("Number of unique countries:", num_countries)
```
```
Number of unique countries: 42
```

In [55]:
```
unique_provinces = df['province'].unique()
num_provinces = len(unique_provinces)
print("Number of unique provinces:", num_provinces)
```
```
Number of unique provinces: 355
```

In [56]:
```
unique_titles = df['title'].unique()
num_titles = len(unique_titles)
print("Number of unique titles:", num_titles)
```
```
Number of unique titles: 39344
```

In [57]:
```
unique_taster_names = df['taster_name'].unique().tolist()
num_taster_names = len(unique_taster_names)
print("Number of unique taster names:", num_taster_names)
```
```
Number of unique taster names: 20
```

In conclusion, we can see that in this dataset there are 42 counties, 255 provinces, 20 tasters and 39,344 wines.

Recommendation

Our recommendations start from the shallow and go deep.

Part 1

1.1 For those who have never drunk wine or rarely drink wine, we recommend that wine starts with the country region and variety. To avoid small probability events, I need to have requirements for the number of entries

We can see that the Top 10 countries are Austria, France, US, Italy, Portugal, Spain, Argentina and Chile.
We can see that the Top 10 provinces are Champagne, Mosel ,Alsace ,Burgundy ,Piedmont ,South Australia, Rhône Valley ,Douro ,Oregon ,Tuscany

Recommended wines from different countries（To avoid small probability events, I need to have requirements for the number of entries）

In [49]:
```
country_points = df[['country', 'points']]

country_counts = country_points['country'].value_counts()

valid_countries = country_counts[country_counts >= 1000].index
valid_country_points = country_points[country_points['country'].isin(valid_countries)]
average_points_by_country = valid_country_points.groupby('country')['points'].mean()

top_countries = average_points_by_country.sort_values(ascending=False).head(10)
```

In [50]:
```
top_countries
```

Out[50]:
```
country
Austria      90.149140
France       88.847597
US           88.577100
Italy        88.492023
Portugal     88.190793
Spain        87.290839
Argentina    86.709544
Chile        86.628998
Name: points, dtype: float64
```

Recommended wines from different province（To avoid small probability events, I need to have requirements for the number of entries）

In [41]:
```
province_points = df[['province', 'points']]

province_counts = province_points['province'].value_counts()

valid_province = province_counts[province_counts >= 200].index
valid_province_points = province_points[province_points['province'].isin(valid_province)]
average_points_by_province = valid_province_points.groupby('province')['points'].mean()

top_province = average_points_by_province.sort_values(ascending=False).head(10)
```

In [42]:
```
top_province
```

Out[42]:
```
province
Champagne          90.243043
Mosel              90.075163
Alsace             89.552113
Burgundy           89.442478
Piedmont           89.360533
South Australia    89.283251
Rhône Valley       89.094937
Douro              89.060386
Oregon             89.058288
Tuscany            88.909432
Name: points, dtype: float64
```

Knowing that Champagne is the highest rated province, I would like to know the top 10 variety here

In [64]:
```
province = 'Champagne'
subset = df[df['province'] == province]

average_scores = subset.groupby('variety')['points'].mean()
sorted_varieties = average_scores.sort_values(ascending=False)

top_10_varieties = sorted_varieties.head(10)
```

In [67]:
```
top_10_varieties
```

Out[67]:
```
variety
Chardonnay         91.510000
Pinot Noir         90.047619
Pinot Blanc        90.000000
Champagne Blend    89.951456
Pinot Meunier      89.750000
Name: points, dtype: float64
```

Knowing that Chardonnay is the highest rated variety, I would like to know the wine of Chardonnay. Here we can make recommendations according to price.

```
In [75]: chardonnay_ratings_prices
```

Out[75]:

	title	points	price
30110	Olivier Leflaive 2014 Montrachet	97	886.0
36529	Krug 2002 Clos du Mesnil Brut Blanc de Blancs ...	99	800.0
30131	Olivier Leflaive 2014 Chevalier-Montrachet	95	710.0
353	Louis Latour 2014 Le Montrachet (Montrachet)	96	630.0
30121	Olivier Leflaive 2014 Bâtard-Montrachet	95	569.0
...	...	...	...
32942	Pine & Post 2006 Chardonnay (Washington)	87	6.0
30465	Gallo Family Vineyards 2005 Twin Valley Chardo...	83	5.0
8428	Earth's Harvest 2014 Organic Grapes Chardonnay...	85	5.0
37951	Earth's Harvest 2014 Organic Grapes Chardonnay...	85	5.0
31530	Bandit NV Chardonnay (California)	84	4.0

3350 rows × 3 columns

1.2 We can see that the top 10 varieties are Picolit, Furmint, Savagnin, Tokaji, Neuburger, Roter, Veltliner, Gros and Petit Manseng, Alsace white blend, Austrian white blend, Scheurebe.

Recommended wines from different varieties (To avoid small probability events, I need to have requirements for the number of entries)

```
In [35]: variety_points = df[['variety', 'points']]

         variety_counts = variety_points['variety'].value_counts()

         valid_variety = variety_counts[variety_counts >= 5].index

         valid_variety_points = variety_points[variety_points['variety'].isin(valid_variety)]

         average_points_by_variety = valid_variety_points.groupby('variety')['points'].mean()

         top_variety = average_points_by_variety.sort_values(ascending=False).head(10)
```

```
In [36]: top_variety

Out[36]: variety
         Picolit                   92.416667
         Furmint                   91.285714
         Savagnin                  91.285714
         Tokaji                    91.000000
         Neuburger                 90.666667
         Roter Veltliner           90.666667
         Gros and Petit Manseng    90.583333
         Alsace white blend        90.555556
         Austrian white blend      90.529412
         Scheurebe                 90.500000
         Name: points, dtype: float64
```

And then I want to find wines with the variety "Picolit" and obtain the corresponding title, price, and points, sorted by score in descending order

```
In [78]: wanted_wines = df[df['variety'] == 'Picolit']
         wanted_wines = wanted_wines.dropna(subset=['price', 'points'])
         sorted_wanted_wines = wanted_wines.sort_values('points', ascending=False)
         selected_columns = ['title', 'price', 'points']
         wanted_data = sorted_wanted_wines[selected_columns]
```

```
In [83]: wanted_data
```

Out[83]:

	title	price	points
33842	Livio Felluga 2007 Picolit Picolit (Colli Orie...	90.0	97
33843	Livio Felluga 2006 Picolit Picolit (Colli Orie...	90.0	96
15613	Livio Felluga 2004 Picolit Picolit (Colli Orie...	100.0	95
36498	Rocca Bernarda 2004 Picolit (Colli Orientali d...	50.0	94
25960	Comelli 2011 Eoos Picolit (Colli Orientali del...	45.0	93
36501	Valchiarò 2004 Picolit (Colli Orientali del Fr...	90.0	93
23380	Jacùss 2007 Picolit (Colli Orientali del Friuli)	55.0	92
18510	Comelli 2009 Eoos Picolit (Colli Orientali del...	20.0	90
23862	Marco Cecchini 2005 Picolit (Colli Orientali d...	30.0	90
23864	Conte d'Attimis-Maniago 2004 Picolit (Colli Or...	60.0	90
39602	Ronchi di Cialla 2008 Cialla Picolit (Colli Or...	75.0	88

I can now recommend a wine in Picolit of the top 10 varieties with a combination of price and points.

Part 2（Code learn from lesson week 2.1）

Now we are getting into the deep end of our recommendations. For those who have drunk wine or have had some experience with wine, there is a wine that they like to drink. Enjoying a wine you like means that you like the combination of aroma, taste, texture and colour of the wine. This allows us to personalise our recommendations to the different characteristics of the wine.

Now I would like to know how to recommend to a taster or a people who likes one type of wine, other types of wines that he might like.I noticed the "description" in the dataset. We can see the tasters describe the wine from several angles, such as aroma, taste and texture, color, etc. I believe this detailed and visual description can help the recommendation system to perform better. This is one of the taster's detailed evaluation of the wine.

'From it's musky, perfumed to nose to it's intensely honeyed, dried-peach- and apricot-flavored palate, everything about this wine is mysterious and ethereal. Crafted by monks in an ancient Georgian monastery, this amber-hued wine is made from white grapes that were fermented and aged in clay vessels (qvevri) and that underwent extended skin contact, which resulted in a wine with puckering tannins and a richly textural mouthfeel.'

1.1 Let's get a handle on the structure of this dataset and the important data

```
In [7]: df.columns

Out[7]: Index(['Unnamed: 0', 'country', 'description', 'designation', 'points',
               'price', 'province', 'region_1', 'region_2', 'taster_name',
               'taster_twitter_handle', 'title', 'variety', 'winery'],
              dtype='object')
```

```
In [28]: taster_name_ids = df["taster_name"].unique().tolist()
         title_ids = df["title"].unique().tolist()
```

```
In [29]: #Make a dictionary mapping ids (keys) to indexes (values)
         taster_name_id_to_index = {x: i for i, x in enumerate(taster_name_ids)}
         title_id_to_index = {x: i for i, x in enumerate(title_ids)}
```

```
In [12]: taster_counts = df['taster_name'].value_counts()
         taster_names = taster_counts.index.tolist()

         print("Number of taster names:", len(taster_names))
         print("List of taster names:")
         print(taster_names)

         Number of taster names: 19
         List of taster names:
         ['Roger Voss', 'Michael Schachner', 'Kerin O'Keefe', 'Paul Gregutt', 'Virginie Boone', 'Matt Kettmann', 'Joe Czerwins
         ki', 'Sean P. Sullivan', 'Anna Lee C. Iijima', 'Jim Gordon', 'Anne Krebiehl\xa0MW', 'Lauren Buzzeo', 'Susan Kostrzew
         a', 'Jeff Jenssen', 'Mike DeSimone', 'Alexander Peartree', 'Carrie Dykes', 'Fiona Adams', 'Christina Pickard']
```

```
In [78]: print("Number of wines:", len(title_id_to_index))

         Number of wines: 39344
```

```
In [75]: num_titles = len(df['title'])
         print("Total number of titles:", num_titles)

         Total number of titles: 40430
```

We can see that this dataset has 15 columns and 40430 rows. 19 of them are tasters and 39344 kinds of wines, which means that most of the wines are tasted by only one taster. This means that it is basically difficult to do a recommendation system by the number of tasters, we need to look at the other.

we did some data pre-processing, such as stop words, then calculate TF-IDF for all words of "description".

```
In [6]: #Import TfIdfVectorizer from scikit-learn
        from sklearn.feature_extraction.text import TfidfVectorizer

        #Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
        tfidf = TfidfVectorizer(stop_words='english')

        #Replace NaN with an empty string
        df['description'] = df['description'].fillna('')

        #Construct the required TF-IDF matrix by fitting and transforming the data
        tfidf_matrix = tfidf.fit_transform(df['description'])

        #Output the shape of tfidf_matrix
        tfidf_matrix.shape

Out[6]: (40430, 20234)
```

Measuring the similarity of embeddings using a distance metric (cosine similarity)

```
In [7]: # Import cosine similarity
        from sklearn.metrics.pairwise import cosine_similarity

        # Compute the cosine similarity matrix
        cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

Define the recommendation generator and generate recommendation results.

```python
In [11]: def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the wine that matches the title
    idx = indices[title]

    # Get the pairwise similarity scores of all wines with that wine
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the wines based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar wines
    sim_scores = sim_scores[1:11]

    # Get the wine indices
    wine_indices = [i[0] for i in sim_scores]

    # Get the titles and points of the top 10 most similar wines
    recommended_wines = df.iloc[wine_indices][['title', 'points']]

    return recommended_wines
```

```python
In [13]: title = "Feudi del Pisciotto 2013 Baglio del Sole Inzolia (Sicilia)"
recommendations = get_recommendations(title)
print(recommendations)
```

```
                                                title  points
16249  Marilena Barbera 2013 Coste al Vento Grillo (S...      86
5991   Barone Sergio 2015 Alègre Grillo (Terre Sicili...      87
9271     Terrose 2014 Salterio  (Rosso di Montepulciano)      86
9969       Terrazze dell'Etna 2009 Cirneco Rosso  (Etna)      87
23715               Ada Nada 2012 Valeirano  (Barbaresco)      88
18353         Casale Daviddi 2012  Rosso di Montepulciano      87
15915  Barone Sergio 2016 Alègre Grillo (Terre Sicili...      88
0                         Nicosia 2013 Vulkà Bianco  (Etna)      87
34212               Montresor 2014 Gran Guardia  (Lugana)      86
20304                     Gino Fasoli NV Brut  (Prosecco)      86
```

2.2 We can see that the ratings of the recommended wines are close (86-88), which is a way to verify the accuracy of the recommendation system. In the future, if you want to find a wine similar to your favorite wine, you will be able to find the recommended content quickly

With the help of this dataset, I would like to recommend cost-effective wines to the average person based on professional tasters' ratings, requiring wines with a score of 95 or more and a price under 100.

Find top 10 cost-effective wines with points above 95 and prices below 100 with price tags

```python
In [32]: # Filter the DataFrame for wines with points above 95 and prices below 100
cost_effective_wines = df[(df['points'] > 95) & (df['price'] < 100)]

# Sort the wines by ascending price
sorted_wines = cost_effective_wines.sort_values(by='price')

# Get the top 10 cost-effective wines
top_10_wines = sorted_wines.head(10)

# Print the titles, points, and prices
for i, row in top_10_wines.iterrows():
    title = row['title']
    points = row['points']
    price = row['price']
    print(f"{i+1}. {title} - {points} points, ${price}")
```

```
40310. Isole e Olena 2010  Chianti Classico - 96 points, $27.0
9902. Domaines Schlumberger 2014 Saering Grand Cru Riesling (Alsace) - 96 points, $29.0
16529. Trisaetum 2016 Ribbon Ridge Estate Dry Riesling (Ribbon Ridge) - 96 points, $32.0
34506. Williams Selyem 2007 Late Harvest Muscat (Russian River Valley) - 96 points, $40.0
15852. Stolo 2014 Estate Syrah (San Luis Obispo County) - 96 points, $40.0
16525. Taylor Fladgate NV 325 Anniversary  (Port) - 97 points, $40.0
9905. Kuentz-Bas 2015 Geisberg Grand Cru Riesling (Alsace) - 96 points, $42.0
33846. Samsara 2008 Las Hermanas Vineyard Pinot Noir (Sta. Rita Hills) - 96 points, $44.0
33845. Woodward Canyon 2009 Chardonnay (Washington) - 96 points, $44.0
26893. Iron Horse 2012 Wedding Cuvée Estate Bottled Sparkling (Green Valley) - 96 points, $44.0
```

2.3 Further, in order for the average consumer to buy high priced wines that they are not satisfied with and feel are not value for money, I would like to give the average consumer a list of the top 10 low-point and high-priced wines to help them have a better wine experience.

Find top 10 low-rated and high-priced wines with points below 90 and prices above 200

In [41]:
```python
# Filter the DataFrame for wines with points below 90 and prices above 500
low_rated_high_priced_wines = df[(df['points'] < 90) & (df['price'] > 200)]

# Sort the wines by descending price
sorted_wines = low_rated_high_priced_wines.sort_values(by='price', ascending=False)

# Get the top 10 low-rated and high-priced wines
top_10_wines = sorted_wines.head(10)

# Print the titles, points, and prices
for i, row in top_10_wines.iterrows():
    title = row['title']
    points = row['points']
    price = row['price']
    print(f"{i+1}. {title} - {points} points, ${price}")
```

```
27519. Vega Sicilia 2008 Unico  (Ribera del Duero) - 89 points, $500.0
26152. Armand de Brignac NV Brut Rosé  (Champagne) - 89 points, $450.0
26865. Armand de Brignac NV Brut Rosé  (Champagne) - 89 points, $450.0
31711. Matarromera 2000 Prestigio Pago de las Solanas  (Ribera del Duero) - 88 points, $325.0
26206. Capichera 2011 Albori di Lampata Red (Isola dei Nuraghi) - 88 points, $320.0
34029. Villa Canestrari 2005 10 Anni Riserva  (Amarone della Valpolicella) - 87 points, $300.0
6906. Nathaniel Rose 2012 Left Bank Abigail's Vineyard Domaine Barrien Cabernet Sauvignon (Lake Michigan Shore) - 87
points, $250.0
20838. Domaine Sophie Cinier 2012  Saint-Véran - 89 points, $250.0
38330. Domaine du Pegau 2015 Cuvée à Tempo White (Châteauneuf-du-Pape) - 87 points, $250.0
30715. Buglioni 2007 Riserva  (Amarone della Valpolicella Classico) - 88 points, $249.0
```

I searched every taster's top 10 wines and found that each taster's rating scale is different (there are high and low top 10 ratings), but there is little difference between the ratings of the top 10 of the same taster.

```
Taster: Alexander Peartree
4900. Lovingston 2012 Josie's Knoll Merlot (Monticello) - 91 points
31279. Bel Lago 2013 Chardonnay (Leelanau Peninsula) - 91 points
31598. King Family 2012 Meritage (Monticello) - 90 points
7935. Canyon Wind 2012 Clone 4 Cabernet Sauvignon (Grand Valley) - 90 points
31581. The Infinite Monkey Theorem 2013 Cabernet Franc (Grand Valley) - 90 points
26819. CrossKeys 2013 Touriga (Virginia) - 90 points
21794. Snowy Peaks 2011 Malbec (Grand Valley) - 89 points
21920. Michael Shaps 2014 Viognier (Virginia) - 89 points
23803. Gill's Pier 2012 Cabernet Franc-Merlot (Leelanau Peninsula) - 89 points
36967. Fabbioli Cellars 2012 Tre Sorélle Red (Virginia) - 88 points

Taster: Anna Lee C. Iijima
16523. Robert Weil 2015 Kiedrich Gräfenberg Trockenbeerenauslese Riesling (Rheingau) - 98 points
348. Robert Weil 2014 Kiedrich Gräfenberg Trockenbeerenauslese Riesling (Rheingau) - 97 points
16526. Domdechant Werner 2015 Hochheimer Domdechaney Trockenbeerenauslese Grosse Lage Riesling (Rheingau) - 96 points
31404. Maximin Grünhäuser 2015 Abtsberg Beerenauslese Grosse Lage Riesling (Mosel) - 96 points
31406. Robert Weil 2015 Kiedrich Gräfenberg Beerenauslese Riesling (Rheingau) - 96 points
355. Robert Weil 2014 Kiedrich Gräfenberg Beerenauslese Riesling (Rheingau) - 96 points
```

***Part 3*　（*Code learn from lesson week 6.1*）**

I want to find out the similarity of each taster's top 10 wines and the difference of a whole top 10 wines by embedding. To get a sense of the differences between each taster's top 10. I can also use embedding to make further recommendations

Then I trained the embedding model, intending to use what I learned in Week 6.1 - Embeddings, embedding taster_name with wine name and then calculating the similarity of the top 10 wines of the 20 tasters.

Training models

```
In [42]: torch.save(model.state_dict(), 'model_weights_1.pth')
```

```
In [43]: model = RecommenderNet(num_taster_name, num_title, EMBEDDING_SIZE)
         model.load_state_dict(torch.load('model_weights_1.pth'))
         model.eval()

Out[43]: RecommenderNet(
           (taster_name_embedding): Embedding(20, 16)
           (taster_name_bias): Embedding(20, 1)
           (title_embedding): Embedding(39344, 16)
           (title_bias): Embedding(39344, 1)
           (sig): Sigmoid()
         )
```

```
In [44]: num_taster_name, EMBEDDING_SIZE, model.taster_name_embedding

Out[44]: (20, 16, Embedding(20, 16))
```

I can find out the top 10 wines from 20 tasters

```
In [50]: # iterate over all user indices
         for taster_name_index in range(num_taster_name):
             # Get top 10 movies for the current user
             top_title = get_top_n(taster_name_index, 10)

             # Print the results
             print("taster_name", taster_name_index)
             print("Top 10 title:")
             for title_index in top_title:
                 title = get_names_for_indexes([title_index])  # Pass a list with a single index
                 print(title)

         taster_name 0
         Top 10 title:
         ['Conte Collalto NV Brut  (Valdobbiadene Prosecco Superiore)']
         ["Wayfarer 2012 Paige's Ridge Pinot Noir (Fort Ross-Seaview)"]
         ['Paratus 2012 Cabernet Sauvignon (Mount Veeder)']
         ['Spring Valley Vineyard 2009 Uriah Estate Grown Red Wine Red (Walla Walla Valley (WA))']
         ['Viña Godeval 2013 Godello (Valdeorras)']
         ['Castillo De Feliciana 2011 Reserve Tempranillo (Columbia Valley (WA))']
         ['Xavier Flouret 2005 Château Haut-Meneau La Victoire  (Premieres Côtes de Blaye)']
         ['Harlow Ridge 2011 Chardonnay (Lodi)']
         ['Larkspur 2013 Pinot Noir (Oregon)']
         ["Paradise Ridge 2003 Ladi's Vineyard Cabernet Sauvignon (Sonoma County)"]
         taster_name 1
         Top 10 title:
         ['Tenute Silvio Nardi 1999  Brunello di Montalcino']
         ['Uccelliera 2006 Riserva  (Brunello di Montalcino)']
         ['Ridge 2012 Bento Dusi Ranch Zinfandel (Paso Robles)']
         ['Frank Family 2012 Reserve Zinfandel (Chiles Valley)']
         ['Boeckel 2015 Zotzenberg Grand Cru Sylvaner (Alsace)']
```

However, when calculating the similarity, the result is 0, which also leads to the inability to calculate the difference later on.

```
In [115]: def calculate_mean_difference(taster_name=0, n=10):
              similarity_matrix = calculate_similarity_matrix(taster_name, n)
              difference_matrix = 1 - similarity_matrix
              return difference_matrix

          def calculate_difference_matrix_for_all_taster_name(n=10):
              difference_matrices = []
              for taster_name in taster_name_ids:
                  difference_matrix = calculate_difference_matrix(num_taster_name-1, n)
                  difference_matrices.append(difference_matrix)
              return difference_matrices

          # Example usage for calculating difference matrices for all users' top 10 movies
          difference_matrices = calculate_difference_matrix_for_all_taster_name(n=10)
          for i, taster_name in enumerate(taster_name_ids):
              print("Difference matrix for top 10 movies of user", taster_name)
              print(difference_matrices[i])
              print()
```

```
Difference matrix for top 10 movies of user Kerin O'Keefe
[[0.]]

Difference matrix for top 10 movies of user Roger Voss
[[0.]]

Difference matrix for top 10 movies of user Paul Gregutt
[[0.]]

Difference matrix for top 10 movies of user Alexander Peartree
[[0.]]
```

```
In [116]: #define calculate_mean_difference_for_dataset
          def calculate_mean_difference_for_dataset(n=10):
              mean_differences = []
              for taster_name in taster_name_ids:
                  mean_difference = calculate_mean_difference(num_taster_name-1, n)
                  mean_differences.append(mean_difference)
              mean_difference_dataset = np.mean(mean_differences)
              return mean_difference_dataset

          # example usage for calculating the mean difference for the entire dataset
          mean_difference_dataset = calculate_mean_difference_for_dataset(n=10)
          print("Mean difference for the entire dataset (all users' top 10 movies):")
          print(mean_difference_dataset) #here we get result
```

```
Mean difference for the entire dataset (all users' top 10 movies):
0.0
```

It can be argued that the completion of the recommendation system is imperfect.

Follow-up

Due to technical problems, I was not able to solve the code problem of the last explored direction in the time available, which is a pity. The next step is to find out what the problem is and solve it by finding out the similarity of the top 10 wines of taster.