RED WINE RECOMMENDATION SYSTEMS

Catherine Sanso

WORKFLOW

WEB SCRAPING

- Content extraction from Wine.com
- Python libraries: Beautiful Soup, Requests

EDA

- Analysis and investigation of data
- Pattern recognition for wine purchases and ratings

RECOMMENDATION SYSTEMS

- Popularity Model
- Content-Based Filtering Model
- Collaborative-Based Filtering Model

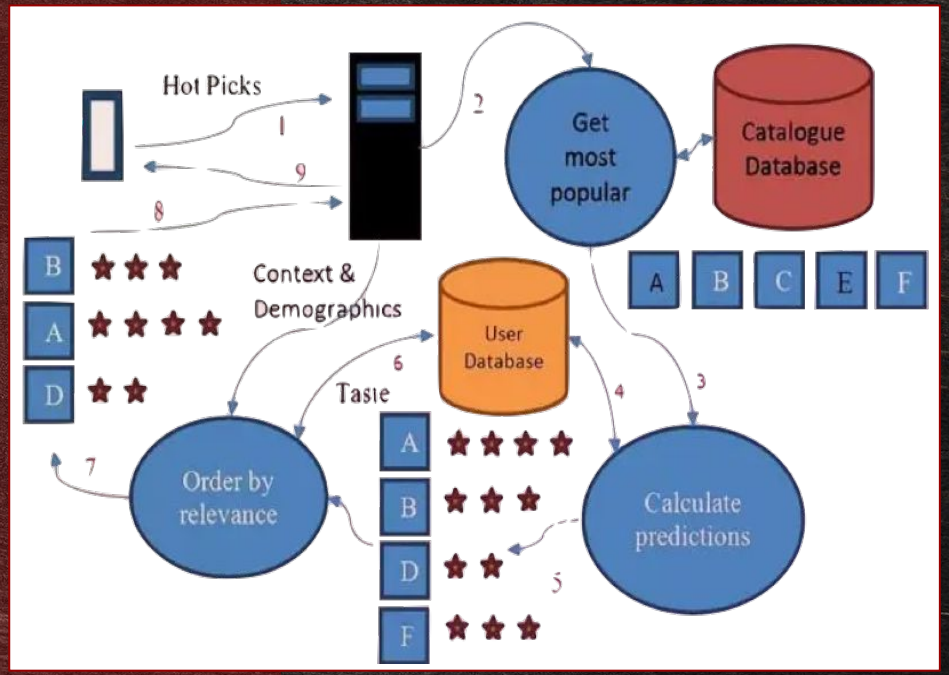
RECOMMENDATION SYSTEMS

ML technology designed to provide personalized suggestions and recommendations to users, based on their preferences and behavior.

Increases business revenue and connects customers with suitable products

Common applications include:

- E-commerce
- Content Streaming Services
- Social Media
- News & Content Aggregation
- Online Advertising
- Financial Product Personalization



01. WEB SCRAPING

Using Beautiful Soup and Requests

```python
def scrape_wine_data(url):
    req = requests.get(url)
    soup = BeautifulSoup(req.content, 'html.parser')
    results = []

    # Extracting the number of elements available for each property
    num_elements = len(soup.body.find_all('meta', class_='schema_productID'))

    for idx in range(num_elements):
        new_entry = {}

        # Define the elements and their corresponding class names
        elements_info = [
            ('product_id', 'meta', 'schema_productID'),
            ('wine_type', 'span', 'listGridItemOrigin_varietal'),
            ('wine_name', 'span', 'listGridItemInfo_name'),
            ('wine_origin', 'span', 'listGridItemOrigin_text'),
            ('rating_avg', 'span', 'averageRating_average'),
            ('rating_num', 'span', 'averageRating_number'),
            ('price_current', 'span', 'productPrice_price-saleWhole'),
            ('price_prediscount', 'span', 'productPrice_price-regWhole'),
            ('discount_nom', 'span', 'productPrice_savings-amount'),
            ('savings_percent', 'span', 'productPrice_savings-percentage')
        ]

        for entry_name, tag, class_name in elements_info:
            elements = soup.body.find_all(tag, class_=class_name)
            if idx < len(elements):
                new_entry[entry_name] = elements[idx].text if tag == 'span' else elements[idx].get('content')

        results.append(new_entry)

    # Creating the DataFrame
    df = pd.DataFrame(results)
    return df
```

```python
dataframes[0].head()
```

	product_id	wine_type	wine_name	wine_origin
0	1405250	Malbec	Chateau Du Caillau Cahors 2021	Cahors, Southwest, France
1	1358914	Tempranillo	Bodegas Lan D-12 2019	Rioja, Spain
2	1288787	Other Red Blends	Bodegas La Purisma Old Vines Red Blend 2019	Yecla, Spain
3	1301819	Tempranillo	Eguren Ugarte Cosecha 2021	Rioja, Spain
4	1384660	Gamay	Domaine Gilles Coperet Brouilly Saburin 2021	Beaujolais, Burgundy, France

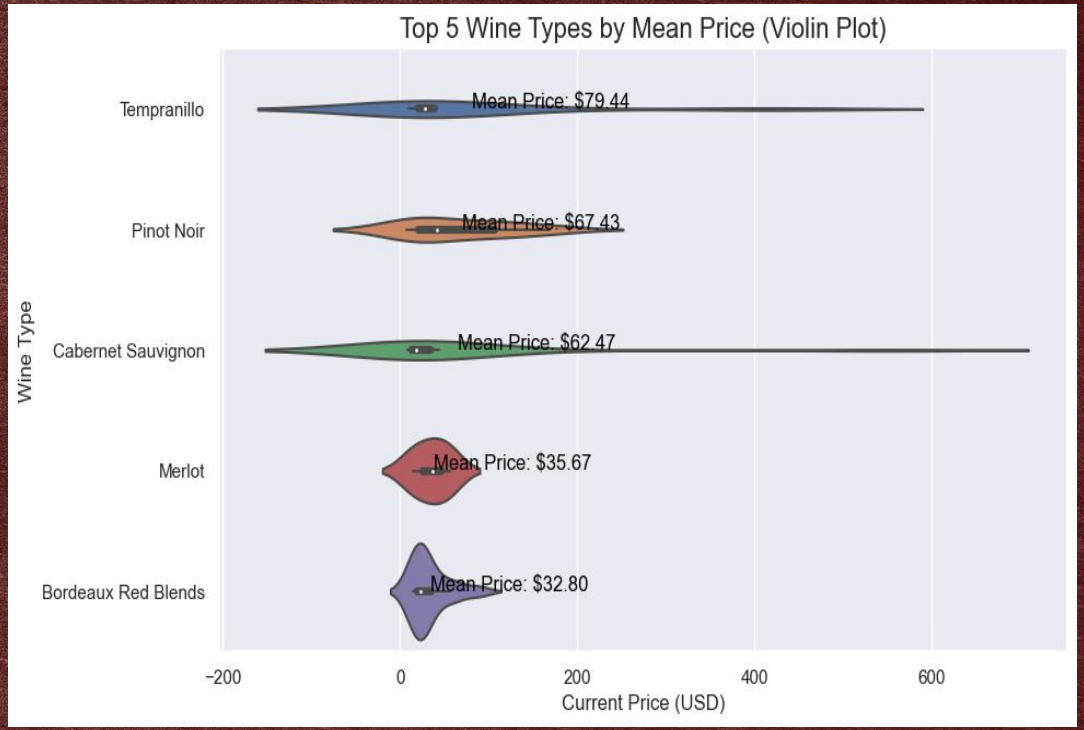
rating_avg	rating_num	price_current	price_prediscount	discount_nom	savings_percent
5.0	19	15	39	$23.01	59
4.8	28	10	19	$2.01	15
4.3	61	22	13	$7.01	23
4.6	27	11	10	$6.01	33
4.4	12	29	18	$5.01	14

02. EDA

Exploratory Data Analysis

- SUMMARY STATISTICS
- VIOLIN PLOTS
- HISTOGRAMS
- RIDGELINE CHARTS
- PAIRPLOTS

VIOLIN PLOT



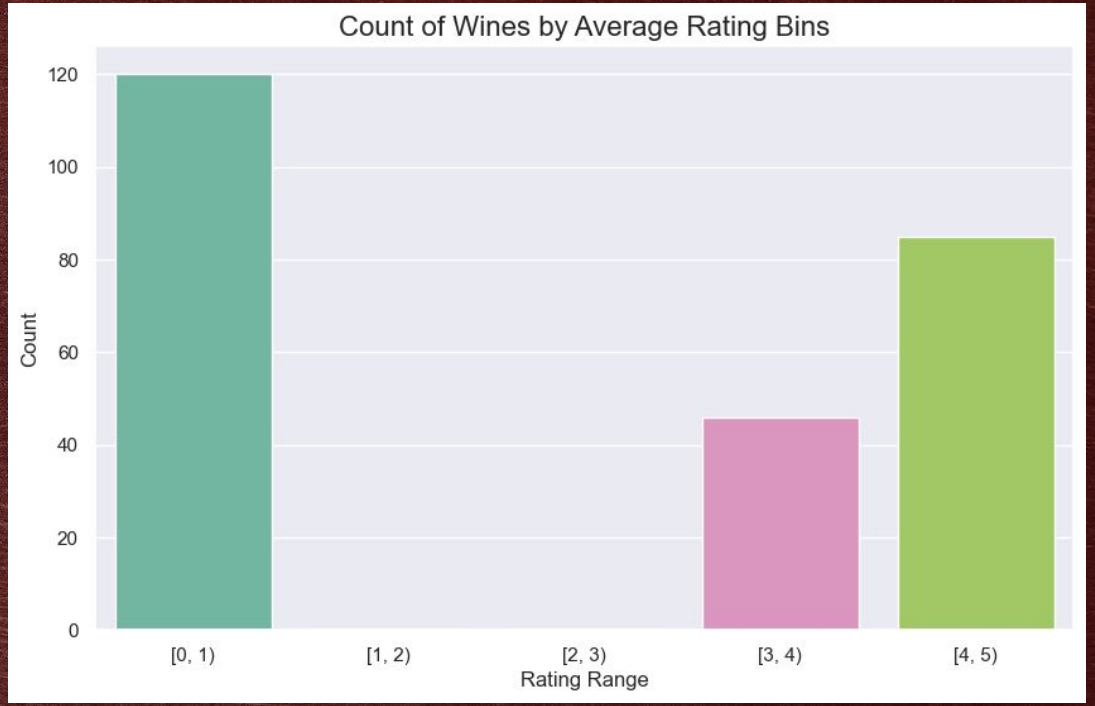
ANALYSIS

- Most extreme outliers: the Cabernet Sauvignons and Tempranillos
- The merlots most resemble a normal distribution, while the other four wines show normal distributions with positive (rightward) skew of varying magnitudes.
- The pinot noirs have the largest interquartile range (IQR) and largest 1.5x IQR, as evidenced by the thick gray bar and the adjacent thinner gray bar in the center of its plot, respectively.

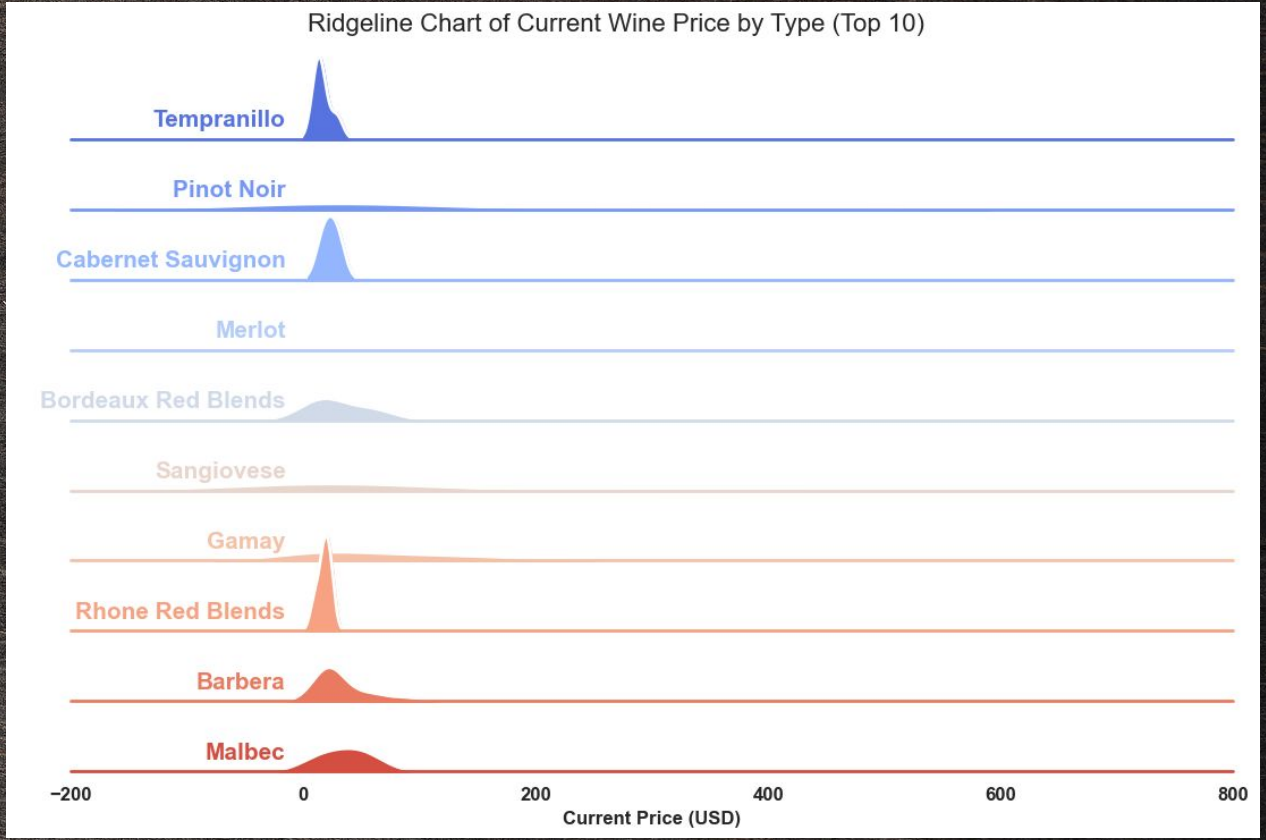
ANALYSIS

- The absence of ratings from [1, 3] could be attributed to a common user behavior where individuals tend to either rate wines very positively (above 3) or not provide a rating at all (0).
- This pattern may suggest that users who are satisfied with a wine are more likely to take the time to rate it, while those who are dissatisfied might simply choose not to rate it, resulting in a skewed distribution towards higher ratings and a lack of ratings in the 1 to 3 range.

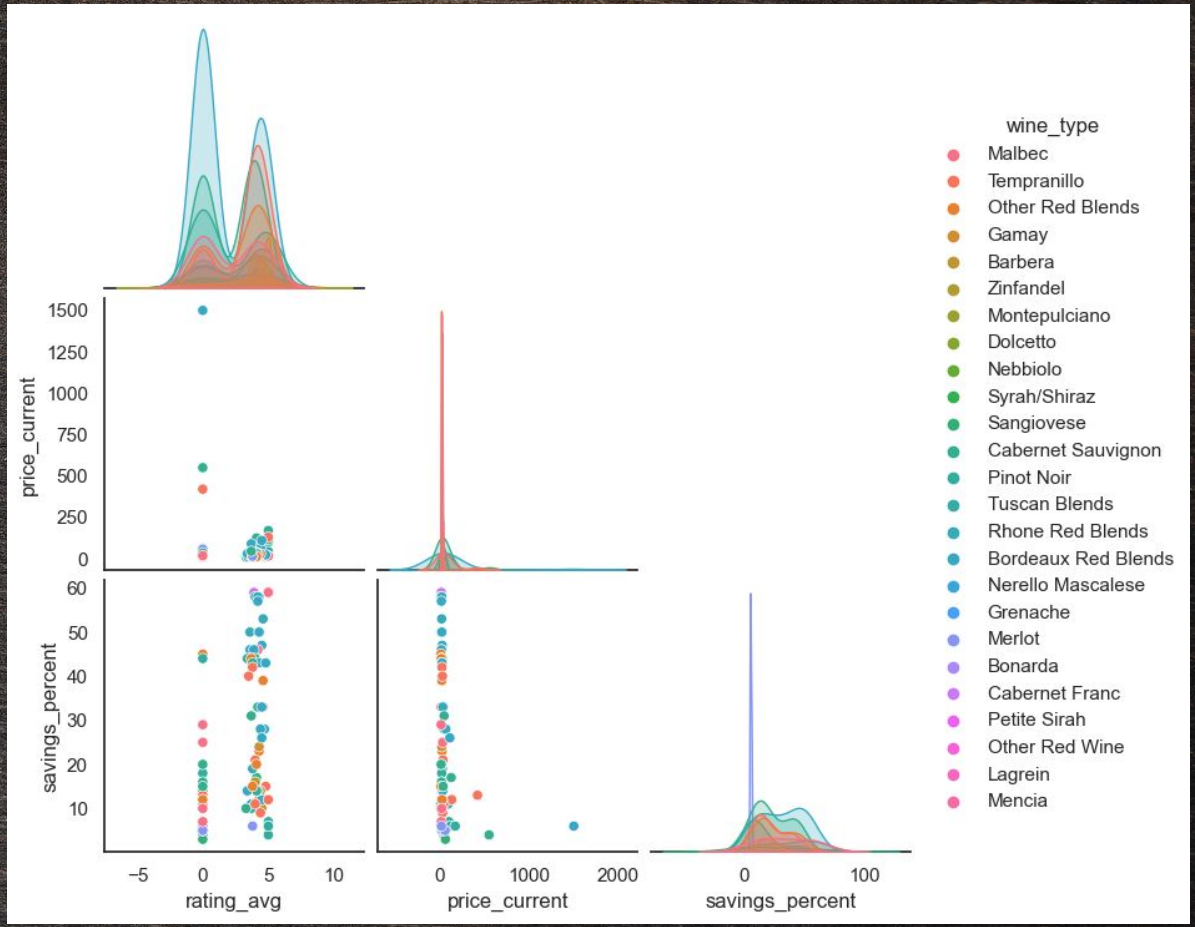
HISTOGRAM



RIDGELINE CHART



PARTIAL PAIRPLOT



03. RECOMMENDATION SYSTEMS

3 TYPES OF MODELS USED:

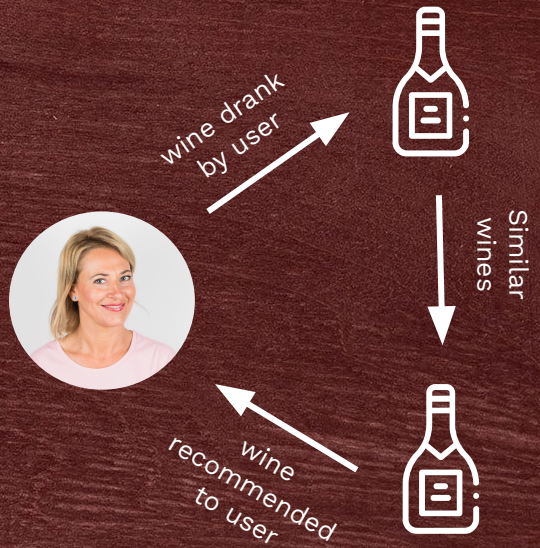
POPULARITY MODEL:

Top 10 wines:

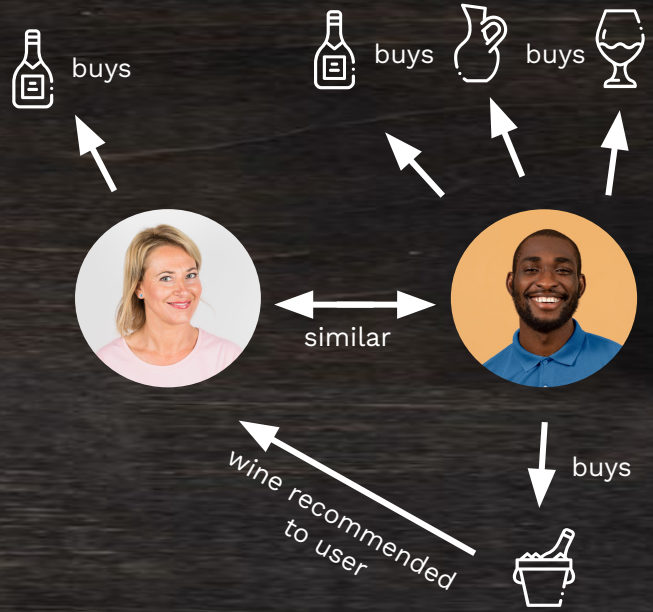
1.
2.
3.
4.

etc...

CONTENT-BASED FILTERING MODEL:



COLLABORATIVE FILTERING MODEL:



	PRINCIPLE	PERSONALIZATION	USER DATA	SCALABILITY	DIVERSITY	USE CASES
POPULARITY MODEL	Recommends items solely based on overall popularity	Lacks personalization; all users receive the same set of popular items	No user-specific data or historical interactions required	Simple, scalable, straightforward	Lack of diversity, tends to recommend the most popular items	Suitable for scenarios where personalization is not critical, such as news articles, trending products, or new user onboarding
CONTENT-BASED FILTERING MODEL	Recommends items with content similar to those the user has liked previously	Considers the user's past behavior and the content of the items.	Requires user profile information and item metadata	May be resource-intensive for large catalogs with extensive feature data	Recommends items based on content similarity	Effective for platforms with item metadata and well-defined preferences
COLLABORATIVE FILTERING MODEL	Recommends items based on the preferences of similar users or the similarity between items	Strong personalization	Requires user-item interaction data	Scales well for large datasets with efficient algorithms like matrix factorization and memory-based methods	Can provide diverse suggestions	Used in e-commerce, music streaming, and any industry where user interactions are abundant

POPULARITY MODEL:

```
# Computing the most popular items:
df_most_popular = df_all_interactions.groupby('product_id')['event_strength'].sum().sort_values(ascending= False).reset_index()
df_most_popular.head(10)
```

	product_id	event_strength
0	1145574	70.0
1	1405250	55.5
2	938102	44.0
3	583739	40.5
4	1232190	39.0
5	202370	31.0
6	1017295	30.0
7	202479	26.0
8	583740	22.0
9	117139	19.0

```
def recommend_top_10(user_id, df_interactions, df_most_popular):
    # Get the list of products the user has already interacted with
    user_interactions = df_interactions[df_interactions['user_id'] == user_id]['product_id'].unique()

    # Filter out products that the user has already interacted with
    top_10_recommendations = df_most_popular[~df_most_popular['product_id'].isin(user_interactions)].head(10)

    return top_10_recommendations
```

```
# Example: Calling the function
user_id = '121711'
top_10_recommendations = recommend_top_10(user_id, df_all_interactions, df_most_popular)

# Print the top 10 recommendations for the user
print(top_10_recommendations)
```

CONTENT-BASED FILTERING MODEL:

```
# Step 1: Feature Extraction
# Combine textual features into a single column and handle missing values
df_all_interactions['wine_features'] = df_all_interactions['wine_name'] + ' ' + df_all_interactions['wine_type'] + ' ' + df_all_interactions['wine_origin']
df_all_interactions['wine_features'].fillna('', inplace=True)

# TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()
wine_tfidf_matrix = tfidf_vectorizer.fit_transform(df_all_interactions['wine_features'])

# Step 2: User Profiles
def build_user_profile(user_id):
    user_interactions = df_all_interactions[df_all_interactions['user_id'] == user_id].copy()

    # Handle missing interaction strengths (NaN) by replacing with zeros
    user_interactions['event_strength'].fillna(0, inplace=True)
    user_item_strengths = user_interactions['event_strength'].values.reshape(-1, 1)

    user_profile = (user_item_strengths.T @ wine_tfidf_matrix[user_interactions.index].toarray()) / max(user_item_strengths.sum(), 1)
    return user_profile

# Step 3: Wine Profiles
wine_profiles = wine_tfidf_matrix

# Step 4: Recommendation Generation
def recommend_wines(user_id, num_recommendations=10):
    user_profile = build_user_profile(user_id)

    # Calculate cosine similarity between user profile and all wine profiles
    cosine_similarities = linear_kernel(user_profile, wine_profiles).flatten()

    # Get indices of top N wines with highest similarity scores
    top_indices = cosine_similarities.argsort()[-num_recommendations:][::-1]

    # Get the corresponding wine IDs
    recommended_wine_ids = df_all_interactions.iloc[top_indices]['product_id'].tolist()

    return recommended_wine_ids
```

```
# Example: Recommend 10 wines for a user with user_id '121711'
user_id_to_recommend = '121711'
recommended_wines = recommend_wines(user_id_to_recommend, num_recommendations=10)
print(recommended_wines)
```

```
[1230356, 1153834, 583740, 583740, 583740, 583740, 583740, 583740, 583740, 583740]
```

COLLABORATIVE FILTERING MODEL:

```
# Step 1: Preprocess the data
reader = Reader(rating_scale=(0, 5))  # Define the rating scale
data = Dataset.load_from_df(df_all_interactions[['user_id', 'product_id', 'event_strength']], reader)

# Step 2: Create a user-item interaction matrix
# done implicitly by Surprise

# Step 3: Apply collaborative filtering algorithm
# Choose a collaborative filtering algorithm (e.g., SVD)
algo = SVD()

# Train the model on the data
trainset = data.build_full_trainset()
algo.fit(trainset)

# Example: Recommend wines for a specific user (user_id)
user_id_to_recommend = '121711'
items_to_ignore = df_all_interactions[df_all_interactions['user_id'] == user_id_to_recommend]['product_id'].tolist()

# Get top N recommendations for the user
top_n = 10  # Number of recommendations to retrieve
user_recommendations = []

for product_id in df_all_interactions['product_id'].unique():
    if product_id not in items_to_ignore:
        predicted_rating = algo.predict(user_id_to_recommend, product_id).est
        user_recommendations.append((product_id, predicted_rating))

# Sort recommendations by predicted rating (higher is better)
user_recommendations.sort(key=lambda x: x[1], reverse=True)

# Get the top N recommendations
top_recommendations = user_recommendations[:top_n]

# Print or use top_recommendations as needed
print(top_recommendations)
```

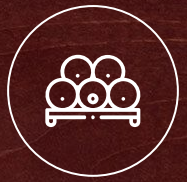
```
[(878447, 2.1125384100281934), (1220861, 2.1121085617210236), (1393318, 2.09657245713216), (1193738, 2.070056735569075
5), (534558, 2.069045645129811), (202370, 2.0618055914075164), (1070730, 2.055649476379271), (1069468, 2.0411214683891
346), (931216, 2.0320598864918336), (1232190, 2.0296291966656606)]
```

DRIVING WINE SALES THROUGH TARGETED RECOMMENDATIONS

FURTHER RECOMMENDATIONS

HYBRID MODEL

Use of a hybrid model that combined content-based and collaborative filtering techniques.



ADDITIONAL DATA

Additional user data and product data from Wine.com will increase the accuracy of the recommendation systems.