WINE QUALITY ANALYSIS

MACHINE LEARNING MODEL

Team: Samuel Heaton, Alice Johnson, AJ Domingo, Jananee Arjunan, Mia Tsivitse, William Julius

APPENDIX

01. Project Scope

02. Data ETL

03. Data EDA

04. Data Pre-Processing

05. Data Model Implementation

06. Data Model Optimization

.01

PROJECT SCOPE

01. Project Scope

In this project, we explored data from wine reviews and supplemented our dataset with additional aggregated weather and elevation data. Using machine learning, our goal was to create a model that could predict the final rating of the wine as an indicator of its quality ('classic' or 'good').

- 90-100 Points – Classic: a great wine of superior character and style
- 80-89 Points – Good: a solid, well-made wine

Initial Data Source: [Wine Reviews Dataset (Kaggle)]

DATA - VARIETY - COUNTRY - WINE - REVIEWS - PRICE - WINERY NAME - ELEV
INE - REVIEWS - PRICE - WINERY NAME - WEATHER DATA - VARIETY - COUNTR
COUNTRY - WINE - REVIEWS - PRICE - WINERY NAME - WEATHER DATA - VARIET

02.
DATA ETL

02. Data ETL

We narrowed our dataset to only wines produced in the United States (dropped data points with insufficient location information)

We supplemented the data with API calls:

1. **Google Places API** - queried the Winery name and Region for latitude and longitude.
2. **Visual Crossing's History Summary API** - queried the coordinates and returned summary weather data from 2022.
3. **Open Meteo API** - queried the coordinates and returned data points for elevation.

02. Data ETL

```python
for index,row in new_data.iterrows():
    zeroResultCount=0
    name = row["Location1"]
    params = { "key": g_key,
              "input":name,
             "inputtype":"textquery",
            "fields": "geometry"}
    url = "https://maps.googleapis.com/maps/api/place/findplacefromtext/json?"
#     print(url+"::"+params['input'])
    try:
        response = requests.get(url,params=params).json()

        if(response['status']== "ZERO_RESULTS"):
                zeroResultCount+=1
                name = row["Location2"]
                params = { "key": g_key,
              "input":name,
             "inputtype":"textquery",
            "fields": "geometry"}
        response = requests.get(url,params=params).json()
        new_data.loc[index,"Latitude"] = response["candidates"][0]["geometry"]["location"]["lat"]
        new_data.loc[index,"Longitude"] = response["candidates"][0]["geometry"]["location"]["lng"]
    except(KeyError, IndexError, JSONDecodeError):
        print(f"{index} {name} not found. Skipping...")
    except requests.ConnectionError:
        print("ConnectionError...")
    except requests.Timeout:
        print("Request Timeout...")
print(zeroResultCount)
```

```
21 Cocobon winery California Other not found. Skipping...
48 Clark-Clauden winery Napa not found. Skipping...
123 Bridlewood winery Central Coast not found. Skipping...
291 Patton Valley winery Willamette Valley not found. Skipping...
369 Expression 44° winery Willamette Valley not found. Skipping...
```

```python
for index,row in data.iterrows():
    lat = row["Latitude"]
    lng = row["Longitude"]
    base_url ="https://weather.visualcrossing.com/VisualCrossingWebServices/rest/services/weatherdata/historysummary?aggregat
    query = (f"&locations={lat},{lng}&key={weather_api_key}")
    url = base_url+query
#     print(url)
     # Use try and except to skip the missing data
    try:
        print(index)
        response = requests.get(url).json()
        data.loc[index,"Min_temp"]=response['location']['values'][0]['mint']
        data.loc[index,"Max_temp"]=response['location']['values'][0]['maxt']
        data.loc[index,"Precip"]=response['location']['values'][0]['precip']
        data.loc[index,"Humidity"]=response['location']['values'][0]['humidity']
        data.loc[index,"Heat_Index"]=response['location']['values'][0]['heatindex']
    except (KeyError, IndexError, JSONDecodeError):
        print("Data not found... skipping.")
    except requests.Timeout:
        print("Request Timeout...")
    except requests.ConnectionError:
        print("ConnectionError...")
```

```python
for index,row in location_data.iterrows():
    lat = row["Latitude"]
    lng = row["Longitude"]
    base_url ="https://api.open-meteo.com/v1/elevation"
    query = (f"?latitude={lat}&longitude={lng}")
    url = base_url+query
    # print(url)
    # Use try and except to skip the missing data
    try:
        print(index)
        response = requests.get(url).json()
        location_data.loc[index,"Elevation"]=response['elevation'][0]
    except (KeyError, IndexError, JSONDecodeError):
        print("Data not found... skipping.")
    except requests.Timeout:
        print("Request Timeout...")
    except requests.ConnectionError:
        print("ConnectionError...")
```

.03

DATA EDA

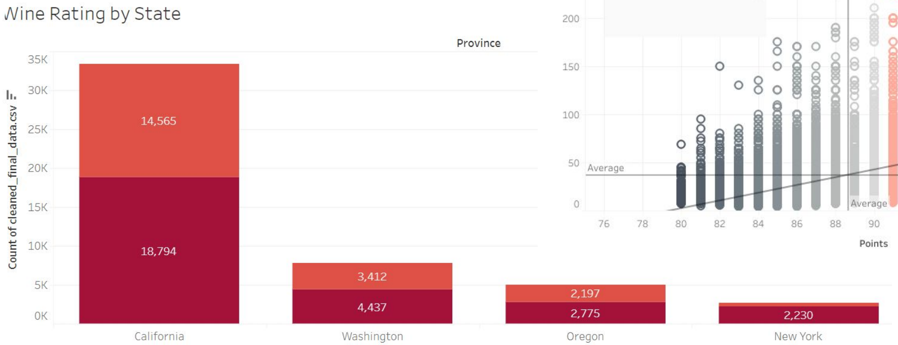
03. Data EDA

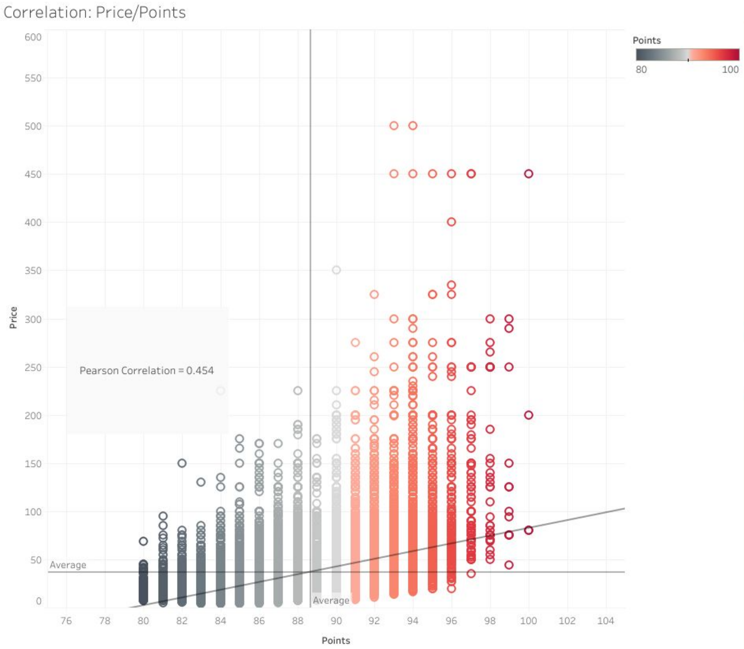
Tableau Notebook #1

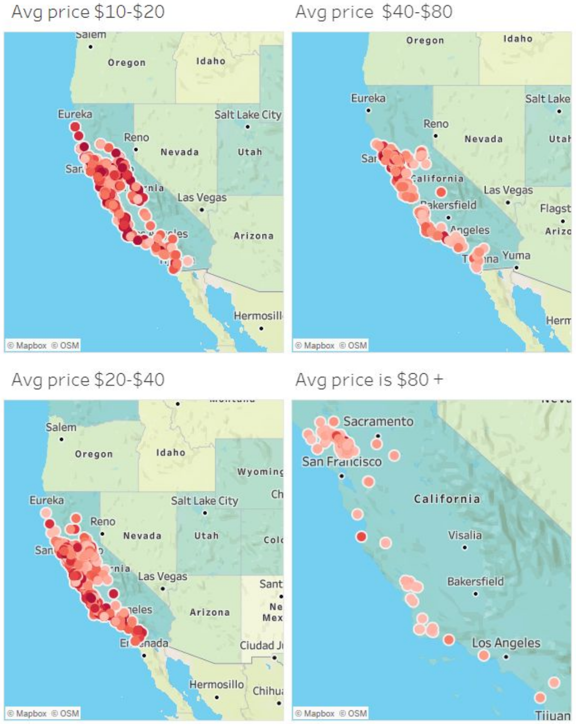
Tableau Notebook #2

Tableau Notebook #3

```
In [9]:  #average wine rating by state
         wine_country_mean = wine_df.groupby('province').mean()['points']
         wine_country_mean

Out[9]:  province
         California    88.644324
         New York      87.181477
         Oregon        88.967619
         Washington    88.980125
         Name: points, dtype: float64

In [10]: #average wine price by state
         wine_price_mean = wine_df.groupby('province').mean()['price']
         wine_price_mean

Out[10]: province
         California    39.568212
         New York      22.827522
         Oregon        35.978681
         Washington    32.629125
         Name: price, dtype: float64
```







04.
DATA PRE-PROCESSING

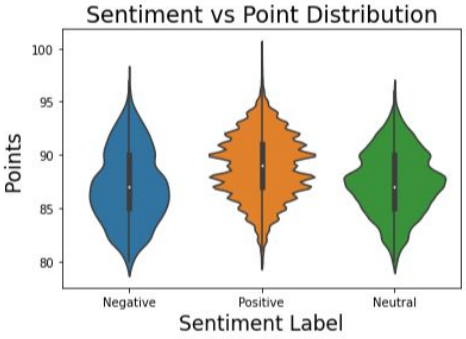
04. Data Pre-Processing

Sentiment Analysis

- We utilized **SentimentAnalyzer** from **Natural Language Toolkit (NLTK)** to create numerical data around the sentiment of the wine description.

```
import seaborn as sns
import numpy as np
a = sns.violinplot(data=NLP_data, x="sentiment", y="points")
a.set_title("Sentiment vs Point Distribution", fontsize=19)
a.set_ylabel("Points ", fontsize=17)
a.set_xlabel("Sentiment Label", fontsize=17)
```

```
Text(0.5, 0, 'Sentiment Label')
```

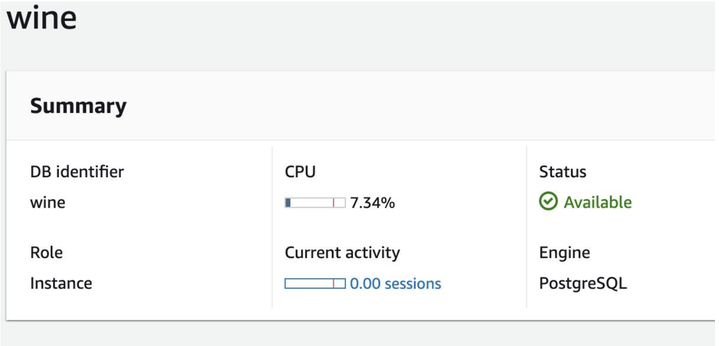


	description	points	polarity_score	review_neu	review_neg	review_pos	sentiment
1	Overripe and Porty, with raisin, prune and cho...	81	0.4417	0.829	0.000	0.171	Positive
2	Strong aromas of blueberry paste, cracked pepp...	92	0.8658	0.805	0.000	0.195	Positive
3	A vegetal note drags down the enjoyment. On th...	84	0.0644	0.710	0.129	0.162	Positive
4	Larry Stanton patiently waits to release the w...	93	0.5434	0.865	0.039	0.096	Positive
5	High alcohol gives the wine heat, especially i...	85	0.6423	0.895	0.000	0.105	Positive

04. Data Pre-Processing

Loading Data

- Using sqlalchemy, the final dataset was loaded into an **Amazon Web Services RDS Postgres SQL database** for ease of use in extracting the data while working in a cloud environment for the modeling steps.




```
In [10]: from sqlalchemy import inspect

         inspector = inspect(engine)

         inspector.get_table_names()

Out[10]: ['wine_data']

In [11]: full_data.to_sql(name='wine_data', con=conn, if_exists='append', index=False)
```

04. Data Pre-Processing

Final Steps

- Dropped unnecessary columns: "wine_id", "country", “winery_name”, "description", "designation","taster_name", "taster_twitter_handle", "title".
- Created bins for the prices
- Created bins for the points into 2 target values:
  - 0 for below 90 points
  - 1 for above 90 points
- Binned some varieties into “other” in order to contain outliers
- Used pd.dummies to create dummies for our categorical/non-numerical features
- Split the data into testing and training data
- Scaled the data

04. Data Pre-Processing

```python
# Choose a cutoff value and create a list of varieties to be replaced
cutoff = 1500
variety_types_to_replace = list(variety_values[variety_values <= cutoff].index)

# Replace in dataframe
for variety in variety_types_to_replace:
    data['variety'] = data['variety'].replace(variety,"Other")

# Check to make sure binning was successful
data['variety'].value_counts()
```

```
Other                       11644
Pinot Noir                   8868
Cabernet Sauvignon           6811
Chardonnay                   6241
Syrah                        2978
Red Blend                    2548
Zinfandel                    2515
Merlot                       2137
Sauvignon Blanc              1891
Bordeaux-style Red Blend     1664
Riesling                     1550
Name: variety, dtype: int64
```

```python
data["point_range"].value_counts()
```

```
1    28236
0    20611
Name: point_range, dtype: int64
```

```python
#create bins for price ranges
def price_to_range(price):
  if (price < 20):
    return 0
  elif (price >20) and (price <= 45):
    return 1
  if (price >45) and (price <= 80):
    return 2
  if (price >80) and (price <= 100):
    return 3
  if (price >140) and (price <= 300):
    return 4
  else:
    return 5

data["price"] = data["price"].apply(price_to_range)
data = data.rename(columns={"price": "price_range"})
data.head()
```

.05

DATA MODEL IMPLEMENTATION

05. Data Model Implementation

TensorFlow Neural Network

- Input features: 247 (length of our X_train after getting dummies)
- 2 hidden layers
  - 400 nodes - layer 1
  - 200 nodes - layer 2
- Relu-activation function for the hidden layers
- Trained over 100 epochs

```
input_features = len(X_train[0])
hidden_layer1 = 400
hidden_layer2= 100
# nodes3 =
nn = tf.keras.models.Sequential()
# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer1, activation="relu", input_dim=input_features))
# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_layer2, activation="relu"))
# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))
# Check the structure of the model
nn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 400)	98400
dense_1 (Dense)	(None, 100)	40100
dense_2 (Dense)	(None, 1)	101

Total params: 138,601
Trainable params: 138,601
Non-trainable params: 0

05. Data Model Implementation

- Achieved a 71% testing accuracy after training the model
- Our model was overfitting (Training: ~83%, Test: ~71%)

```
[ ] # # Evaluate the model using the test data
    model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
    print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

    382/382 - 1s - loss: 0.9691 - accuracy: 0.7091 - 963ms/epoch - 3ms/step
    Loss: 0.9690999984741211, Accuracy: 0.7091385722160339
```

05. Data Model Implementation

Random Forest Classifier

- Testing Accuracy of 77%, but was overfitting (99% training)

```
[35] from sklearn.ensemble import RandomForestClassifier
     from sklearn.metrics import accuracy_score

     # create and fit the model
     clf = RandomForestClassifier(random_state=42, n_estimators=100).fit(X_train_scaled, y_train)
     # Evaluate the model
     y_predict = clf.predict(X_test_scaled)
     score = accuracy_score(y_test,y_predict)
     score
```

```
0.7661316737635113
```

```
print(f'Training Score: {clf.score(X_train_scaled, y_train)}')
print(f'Testing Score: {clf.score(X_test_scaled, y_test)}')
```

```
Training Score: 0.9941585915108503
Testing Score: 0.7661316737635113
```

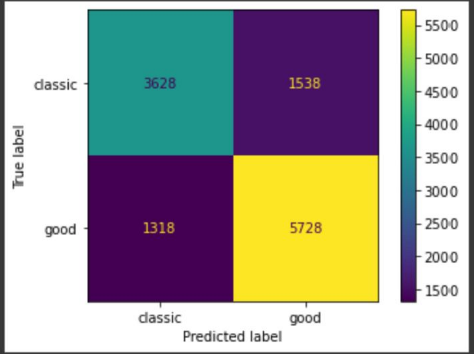
05. Data Model Implementation

Classification Report and Confusion Matrix

- We compared the predicted values from the model against the actual values.

```
[17] from sklearn.metrics import confusion_matrix, classification_report

[18] y_true = y_test
     y_pred = clf.predict(X_test_scaled)
     confusion_matrix(y_true, y_pred)

     array([[3628, 1538],
            [1318, 5728]])

     target_names=["classic","good"]
     print(classification_report(y_true, y_pred,target_names=target_names))

                   precision    recall  f1-score   support

         classic       0.73      0.70      0.72      5166
            good       0.79      0.81      0.80      7046

        accuracy                           0.77     12212
       macro avg       0.76      0.76      0.76     12212
    weighted avg       0.77      0.77      0.77     12212
```

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
import matplotlib.pyplot as plt
disp=ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=target_names)
disp.plot()
plt.show()
```



06.
DATA MODEL OPTIMIZATION

06. Data Model Optimization

MinMaxScalar

```
[28] # create minmaxscaler instance
     min_max_scaler = MinMaxScaler()

     x_minmax = min_max_scaler.fit(X_train)

     x_mm_train = x_minmax.transform(X_train)
     x_mm_test = x_minmax.transform(X_test)
```

```
# # Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(x_mm_test,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

382/382 - 1s - loss: 0.6923 - accuracy: 0.5287 - 998ms/epoch - 3ms/step
Loss: 0.6922512650489807, Accuracy: 0.5286603569984436
```

PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.99)
pca_data = pca.fit_transform(X_dummies)
pca_data_df=pd.DataFrame(pca_data)
pca_data_df
```

```
1145/1145 [==============================] - 2s 2ms/step - loss: 0.6616 - accuracy: 0.5922

# # Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

382/382 - 1s - loss: 0.6613 - accuracy: 0.5824 - 533ms/epoch - 1ms/step
Loss: 0.6612696647644043, Accuracy: 0.5823779702186584
```

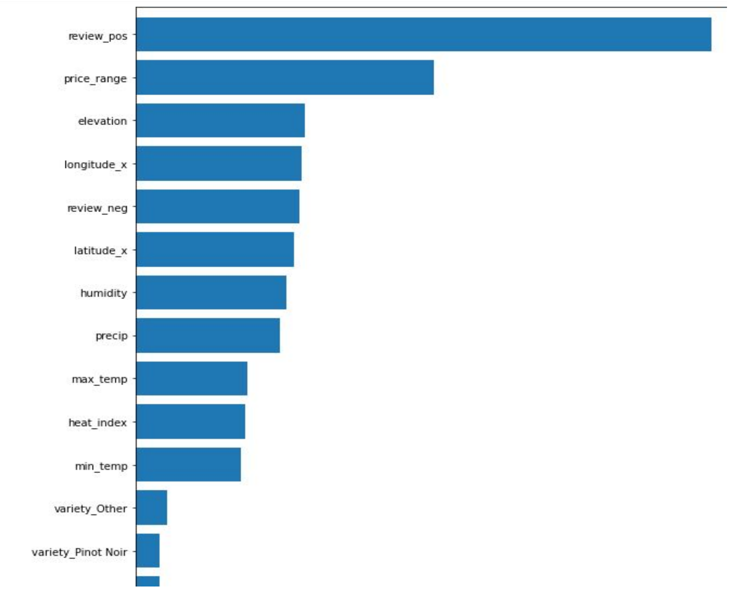
06. Data Model Optimization

SelectFromModel Feature Selection

- Feature importance used to look for best features using SelectFromModel (Top 20 Features)

```python
from matplotlib import pyplot as plt
import numpy as np
features = sorted(zip(feature_names, clf.feature_importances_), key = lambda x: x[1])
cols = [f[0] for f in features]
width = [f[1] for f in features]

fig, ax = plt.subplots()

fig.set_size_inches(10,200)
plt.margins(y=0.001)

ax.barh(y=cols, width=width)

plt.show()
```

```
Epoch 49/50
1145/1145 [==============================] - 3s 2ms/step - loss: 0.5314 - accuracy: 0.7237
Epoch 50/50
1145/1145 [==============================] - 3s 2ms/step - loss: 0.5314 - accuracy: 0.7249

[36] # # Evaluate the model using the test data
     model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

     382/382 - 1s - loss: 0.5369 - accuracy: 0.7195 - 755ms/epoch - 2ms/step
     Loss: 0.5369265675544739, Accuracy: 0.719538152217865
```



06. Data Model Optimization

KearsTuner

```
Epoch 100/100
1145/1145 [==============================] - 2s 2ms/step - loss: 0.4773 - accuracy: 0.7592

[100] # # Evaluate the model using the test data
      model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
      print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

      382/382 - 1s - loss: 0.5526 - accuracy: 0.7204 - 657ms/epoch - 2ms/step
      Loss: 0.5525580048561096, Accuracy: 0.7204388976097107
```

```
[26] model_loss, model_accuracy = best_model.evaluate(X_test_scaled,y_test,verbose=2)
     print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

     382/382 - 1s - loss: 0.5520 - accuracy: 0.7284 - 836ms/epoch - 2ms/step
     Loss: 0.5519906878471375, Accuracy: 0.7283819317817688

     best_hyper = tuner.get_best_hyperparameters(1)[0]
     best_hyper.values

     {'activation': 'relu',
      'first_units': 221,
      'num_layers': 1,
      'units_0': 81,
      'units_1': 81,
      'units_2': 121,
      'tuner/epochs': 15,
      'tuner/initial_epoch': 5,
      'tuner/bracket': 1,
      'tuner/round': 1,
      'tuner/trial_id': '0023'}
```

	Final Result	Hyperparameters
TF Neural Network Model	71%	247 Input Features | 2 Hidden Layers Layer 1 = 400 Nodes | Layer 2 = 200 Nodes Relu Activation Function | 100 epochs
Random Forest Classifier	77%	(random_state=42, n_estimators=100)
PCA and NN model	58%	(n_components=0.99) Input Features = 3 |Model: "sequential_1" Layer 1 = 10 Nodes / activation="relu" | Layer 3 = 1 Node / activation="sigmoid"
SelectFromModel Feature Selection	72%	Input Features = 85 Layer 1 = 120 Nodes / activation="relu" Layer 3 activation="sigmoid" Model: "sequential_1"
KerasTuner	73%	400/20 Max/Step Ratio 'activation': 'relu', | 'first_units': 221| 'num_layers': 1 'units_0': 81 | 'units_1': 81, | 'units_2': 121, 'tuner/epochs': 15 | 'tuner/initial_epoch': 5, | 'tuner/bracket': 1, 'tuner/round': 1 | 'tuner/trial_id': '0023'

CONCLUSION

Final Insights:

- There were a total of **48.8K** wines analyzed in our dataset. Overall, **68%** were from California, **16%** from Washington, **10%** from Oregon, and **6%** from New York.

- The highest count of wines rated as ‘Classic’ were found in California (**14K**) as well as in the region of Napa Valley (**2.1K**), when compared to other states and regions.

- Every wine in our dataset that was >$300 in price was rated as ‘Classic’. In comparison, only **11%** of the **10.2K** wines in the >$20 price range were rated as ‘Classic’.

- There was a moderate correlation between price and the review score (points) at **0.45** (Pearson Correlation).

- We believe our dataset could be improved in the future by increasing our total sample size from additional states and countries. Expanding our dataset could have further improved model accuracy.

- In the future, this data could also be used for investments/marketing purposes, such as recommending the best wineries based on ideal weather conditions.

QUESTIONS?