Wine Quality Prediction
Machine Learning Project

Pengxi Chen
McMaster University
February 29, 2024

[44]:
```python
# (2)
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Load the dataset
file_path = 'winequalityN.csv'
wine_data = pd.read_csv(file_path)
wine_data.head()
```

[44]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar
0	white	7.0	0.27	0.36	20.7
1	white	6.3	0.30	0.34	1.6
2	white	8.1	0.28	0.40	6.9
3	white	7.2	0.23	0.32	8.5
4	white	7.2	0.23	0.32	8.5

	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	0.045	45.0	170.0	1.0010	3.00
1	0.049	14.0	132.0	0.9940	3.30
2	0.050	30.0	97.0	0.9951	3.26
3	0.058	47.0	186.0	0.9956	3.19
4	0.058	47.0	186.0	0.9956	3.19

	sulphates	alcohol	quality
0	0.45	8.8	6
1	0.49	9.5	6
2	0.44	10.1	6
3	0.40	9.9	6
4	0.40	9.9	6

[45]:
```python
#(3)
# The number of features and observations
num_observations, num_features = wine_data.shape
# Variable types
variable_types = wine_data.dtypes
num_observations, num_features, variable_types
```

[45]: (6497,
 13,
 type   object
 fixed acidity   float64
 volatile acidity   float64
 citric acid   float64
 residual sugar   float64
 chlorides   float64
 free sulfur dioxide   float64
 total sulfur dioxide   float64
 density   float64
 pH   float64
 sulphates   float64
 alcohol   float64
 quality   int64
 dtype: object)

[46]:
```
#(4)
# summary statistics
summary_statistics = wine_data.describe()
summary_statistics
```

[46]:

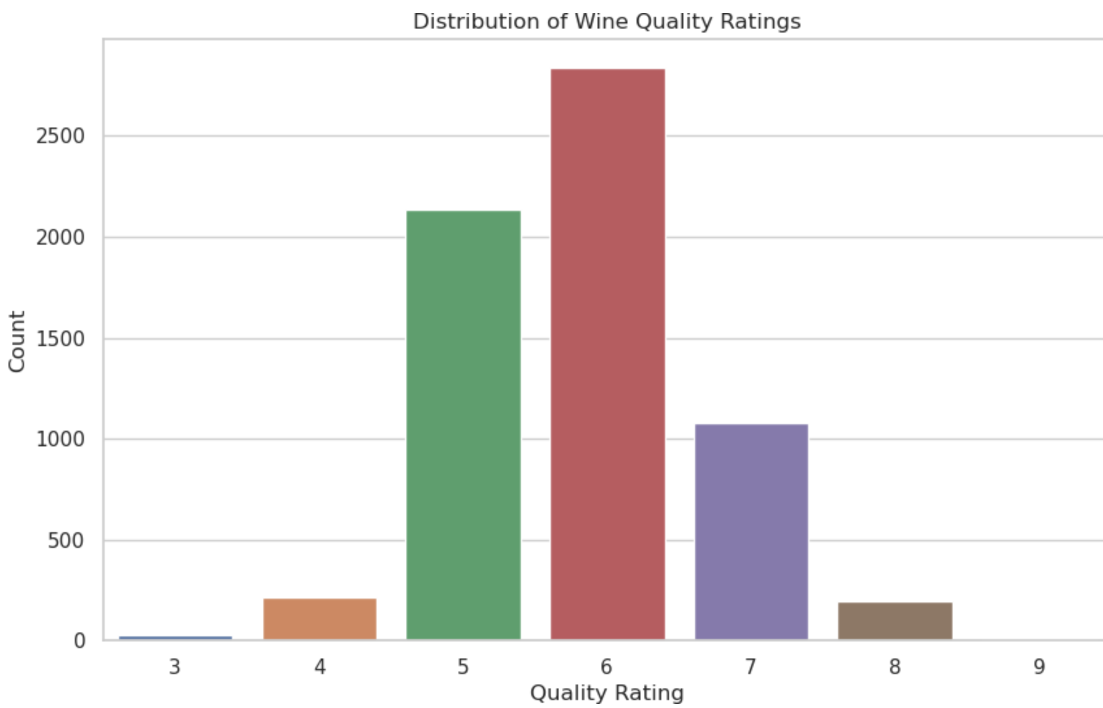
	fixed acidity	volatile acidity	citric acid	residual sugar
count	6487.000000	6489.000000	6494.000000	6495.000000
mean	7.216579	0.339691	0.318722	5.444326
std	1.296750	0.164649	0.145265	4.758125
min	3.800000	0.080000	0.000000	0.600000
25%	6.400000	0.230000	0.250000	1.800000
50%	7.000000	0.290000	0.310000	3.000000
75%	7.700000	0.400000	0.390000	8.100000
max	15.900000	1.580000	1.660000	65.800000

	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	6495.000000	6497.000000	6497.000000	6497.000000
mean	0.056042	30.525319	115.744574	0.994697
std	0.035036	17.749400	56.521855	0.002999
min	0.009000	1.000000	6.000000	0.987110
25%	0.038000	17.000000	77.000000	0.992340
50%	0.047000	29.000000	118.000000	0.994890
75%	0.065000	41.000000	156.000000	0.996990
max	0.611000	289.000000	440.000000	1.038980

	pH	sulphates	alcohol	quality
count	6488.000000	6493.000000	6497.000000	6497.000000
mean	3.218395	0.531215	10.491801	5.818378
std	0.160748	0.148814	1.192712	0.873255
min	2.720000	0.220000	8.000000	3.000000

25%	3.110000	0.430000	9.500000	5.000000
50%	3.210000	0.510000	10.300000	6.000000
75%	3.320000	0.600000	11.300000	6.000000
max	4.010000	2.000000	14.900000	9.000000

[47]:
```
#(5)
# Create a histogram for the quality ratings
plt.figure(figsize=(10, 6))
sns.countplot(data=wine_data, x='quality')
plt.title('Distribution of Wine Quality Ratings')
plt.xlabel('Quality Rating')
plt.ylabel('Count')
plt.show()
```

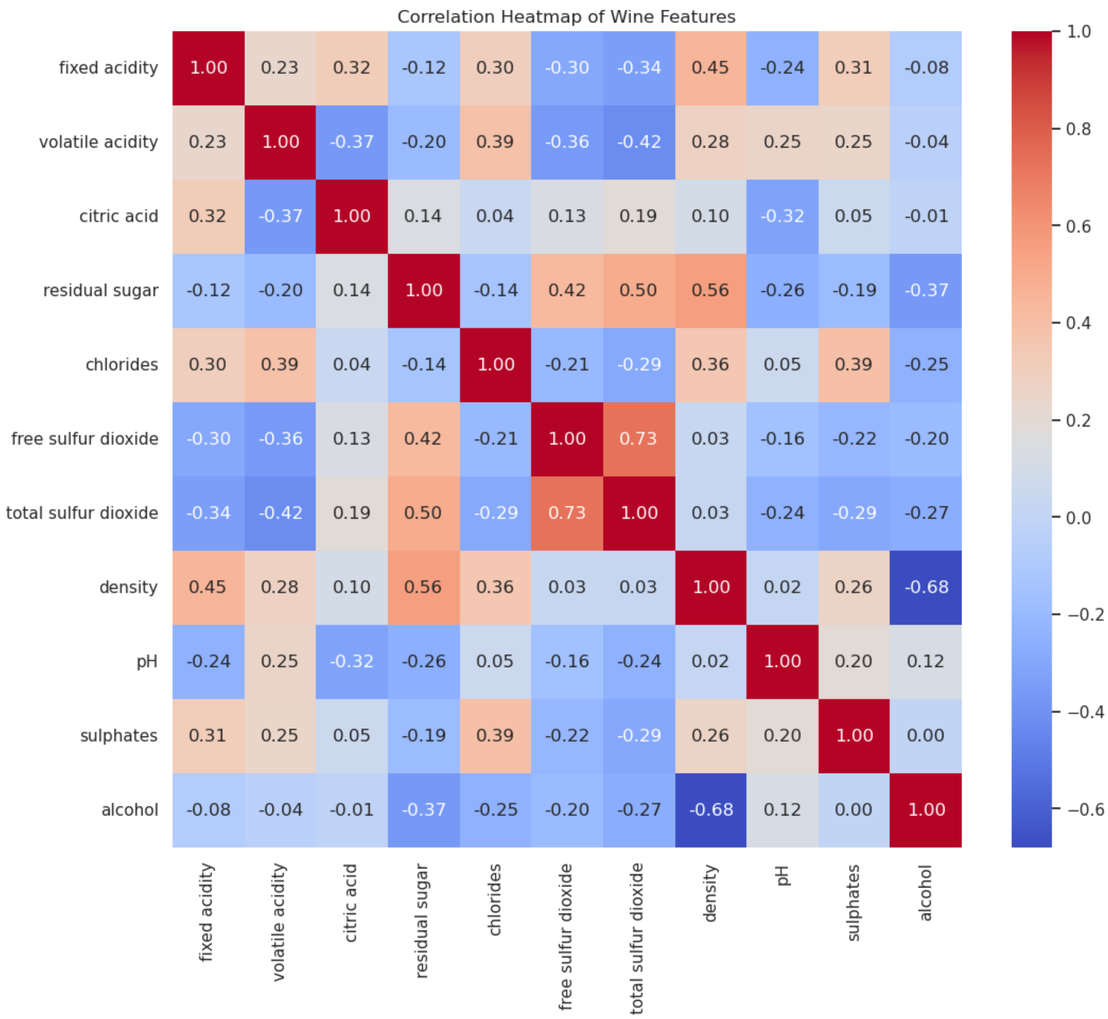


[48]:
```
# 6)
# Drop observations with quality scores <=4 and >=8
filtered_data = wine_data[(wine_data['quality'] > 4) & (wine_data['quality'] <
 8)]
```

[49]:
```
#(7)
# Count of observations and unique quality scores in the filtered data
num_observations_filtered = filtered_data.shape[0]
unique_quality_scores = filtered_data['quality'].nunique()
num_observations_filtered, unique_quality_scores
```

[49]: (6053, 3)

[50]:
```
#8
# Exclude 'type' and 'quality' variables
df_excluded = filtered_data.drop(['type', 'quality'], axis=1)
# Correlation matrix
corr_matrix = df_excluded.corr()
# Plot the correlation heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Wine Features')
plt.show()
```



[51]:
```
#(9)
# Check for missing values in the filtered dataset
```

```python
missing_values = filtered_data.isnull().sum()
# Display columns with missing values
missing_values[missing_values > 0]
# Drop observations with missing values in the filtered dataset
filtered_data_cleaned = filtered_data.dropna()
```

[52]:
```python
#(10)
# Standardize the predictor variables
from sklearn.preprocessing import StandardScaler
X = filtered_data_cleaned.drop(['type', 'quality'], axis=1)
y = filtered_data_cleaned['quality']
# Standardize the predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled.shape
```

[52]: (6022, 11)

[53]:
```python
#(11)
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.
↪25, random_state=42, stratify=y)
```

[54]:
```python
#(12) KNN classifier with k=5
knn = KNeighborsClassifier(n_neighbors=5)
```

[55]:
```python
#(13) Train the model on the training set
knn.fit(X_train, y_train)
# Predicte test set
y_pred = knn.predict(X_test)
y_pred
```

[55] : array([6, 6, 5, …, 5, 5, 5])

[56] :
```python
#(14) Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

[56]: 0.5942895086321381

[57] :
```python
#(15)
from sklearn.metrics import confusion_matrix
# Confusion matrix!
conf_matrix = confusion_matrix(y_test, y_pred)
```

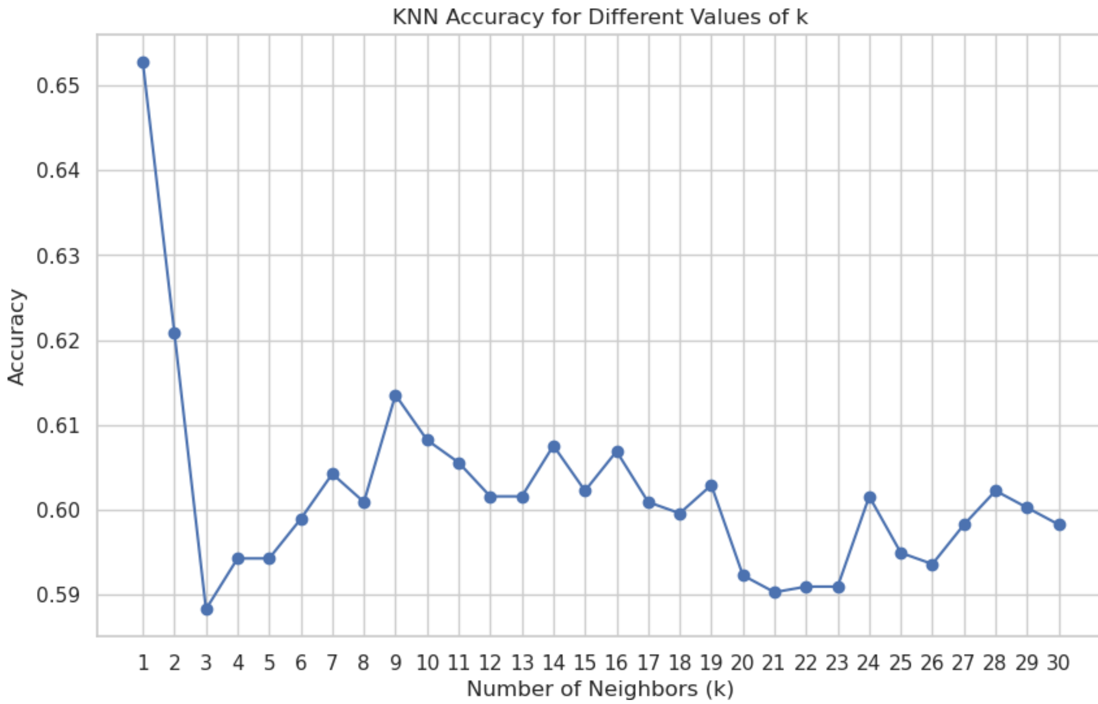
```
conf_matrix
```

```
[57]: array([[336, 184,  12],
             [187, 446,  72],
             [ 22, 134, 113]])
```

```
[58]: #(17)
# loop different values of k to find the optimal number of neighbors
k_values = range(1, 31)
accuracies = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
# Identify the k with the highest accuracy
max_accuracy = max(accuracies)
optimal_k = k_values[accuracies.index(max_accuracy)]
max_accuracy, optimal_k
```

```
[58]: (0.6527224435590969, 1)
```

```
[59]: #(18)
# Plot the accuracies for different values of k
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracies, marker='o')
plt.title('KNN Accuracy for Different Values of k')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```


KNN Accuracy for Different Values of k
Accuracy
0.65
0.64
0.63
0.62
0.61
0.60
0.59
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Neighbors (k)