Department of Computer Engineering
University of Peradeniya

CO 544 Machine Learning and Data Mining
Lab 03

May 3, 2023

1. Objectives

- To create an interactive dashboard using Python for exploratory data analysis, such as the correlation between different features, and explore model prediction accuracy in predicting the quality of the wine based on the input feature values.
- To discuss the performance of a model using various performance metrics.
- To apply logistic regression to the wine quality dataset to predict the quality of wine as "Good" or "Bad".

2. Preliminaries

You are expected to have:

- Understanding of basic data analysis and visualization techniques.
- Familiarity with the Wine Quality dataset, its features, and quality labels.
- Hands-on experience in HTML, CSS, and JavaScript to customize the layout and styles of the dashboard.

3. Introduction

Dashboarding is a method to present data in a visually appealing and interactive manner. It displays large amounts of data in an easy-to-understand format for quick analysis and decision-making. Dashboards are used in businesses, organizations, and government agencies to provide real-time insights and monitor key performance indicators. Dashboards typically consist of one or more graphical forms, such as charts, graphs, or maps, that display data in a meaningful way. These visualizations can be customized and interactive, allowing users to drill down into specific data points, filter data, or change the view of the data.

In this lab sheet, we use the Wine Quality dataset (winequality-red.csv) from UCI Machine Learning Repository to build an interactive dashboard using the Python libraries, such as Dash, Plotly, and Seaborn, and use logistic regression to predict wine quality.

3. Importing Required Libraries

In this lab, we use Pandas for data loading and preprocessing, Scikit-learn for model implementation and performance evaluation, and Dash and Plotly for dashboard creation.

```
# For data analysis
import pandas as pd

# For model creation and performance evaluation
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_curve, roc_auc_score

# For visualizations and interactive dashboard creation
import dash
from dash import dcc, html
from dash.dependencies import Input, Output, State
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

4. Exploratory Data Analysis (EDA) and Data Preprocessing

4.1 Loading Data

In this lab, the red wine dataset is used. The dataset consists of 1,599 samples and 12 features, including the wine quality score, which ranges from zero to ten. The features include fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality.

```
# Load dataset
data = pd.read_csv("data/winequality-red.csv")
```

4.2 Handling Missing Values and Duplicates

Handling missing values and duplicates is crucial for accurate and reliable analysis. The presence of missing values can affect the analysis and result in biased and inaccurate predictions. Here, we remove rows containing at least one missing value to handle missing data in the dataset. While it is quite straightforward, you should be careful as it may lead to loss of information and decrease the size of the dataset.

```
# check for missing values
print(data.isnull().sum())

# drop rows with missing values
data.dropna(inplace=True)
```

Duplicates refer to the presence of identical rows in a dataset. They can also negatively impact the analysis of the dataset as it can lead to incorrect statistical measures and biased results. Therefore, it is important to identify and drop duplicate data before proceeding with the analysis.

```
# Drop duplicate rows
data.drop_duplicates(keep='first')
```

4.3 EDA

Analysing the Distribution of Wine Quality Scores

Understanding the distribution of wine quality scores across the dataset is important. This helps identify whether the data set is balanced or imbalanced in terms of the distribution of quality ratings.

```
# Check wine quality distribution
plt.figure(dpi=100)
sns.countplot(data=data, x="quality")
plt.xlabel("Count")
plt.ylabel("Quality Score")
plt.show()
```

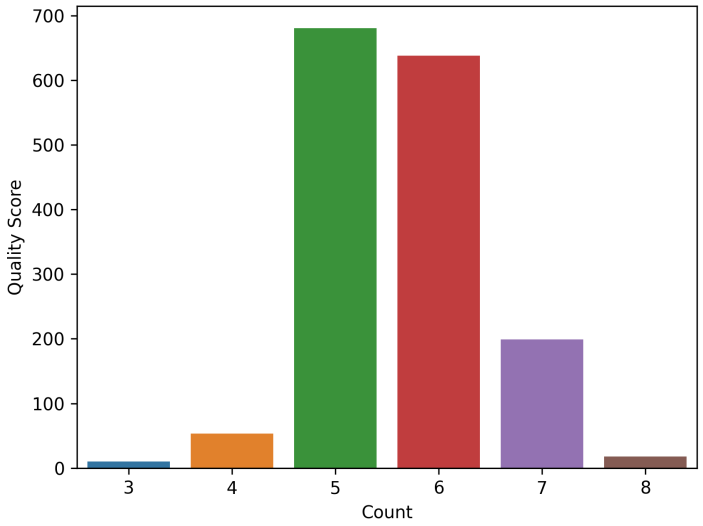

Figure 1. Distribution of wine quality scores.

Preparing the Dataset for Binary Classification

In this dataset, the quality score ranges from three to eight, where a higher score represents a better quality wine. However, for the purpose of binary classification, we label the quality score as **1** (i.e., represents "Good") and **0** (i.e., represents "Bad") based on a threshold value. In this case, we use the score of **six** as the threshold value, where a quality score greater than or equal to six can be labeled as **1 (Good)** and a quality score less than six can be labeled as **0 (Bad)**.

```
# Label quality into Good (1) and Bad (0)
data['quality'] = data['quality'].apply(lambda x: 1 if x >= 6.0 else 0)
```

Gaining Insights from the Correlation Matrix

The correlation matrix demonstrates the correlation coefficients between different features (a.k.a. variables or columns) in the dataset. The correlation coefficient is a value between -1 and 1 that measures the strength and direction of the linear relationship between two features.

```
# Calculate the correlation matrix
corr_matrix = data.corr()

# Plot heatmap
plt.figure(figsize=(12, 8), dpi=100)
```

```
sns.heatmap(corr_matrix, center=0, cmap='Blues', annot=True)
plt.show()
```

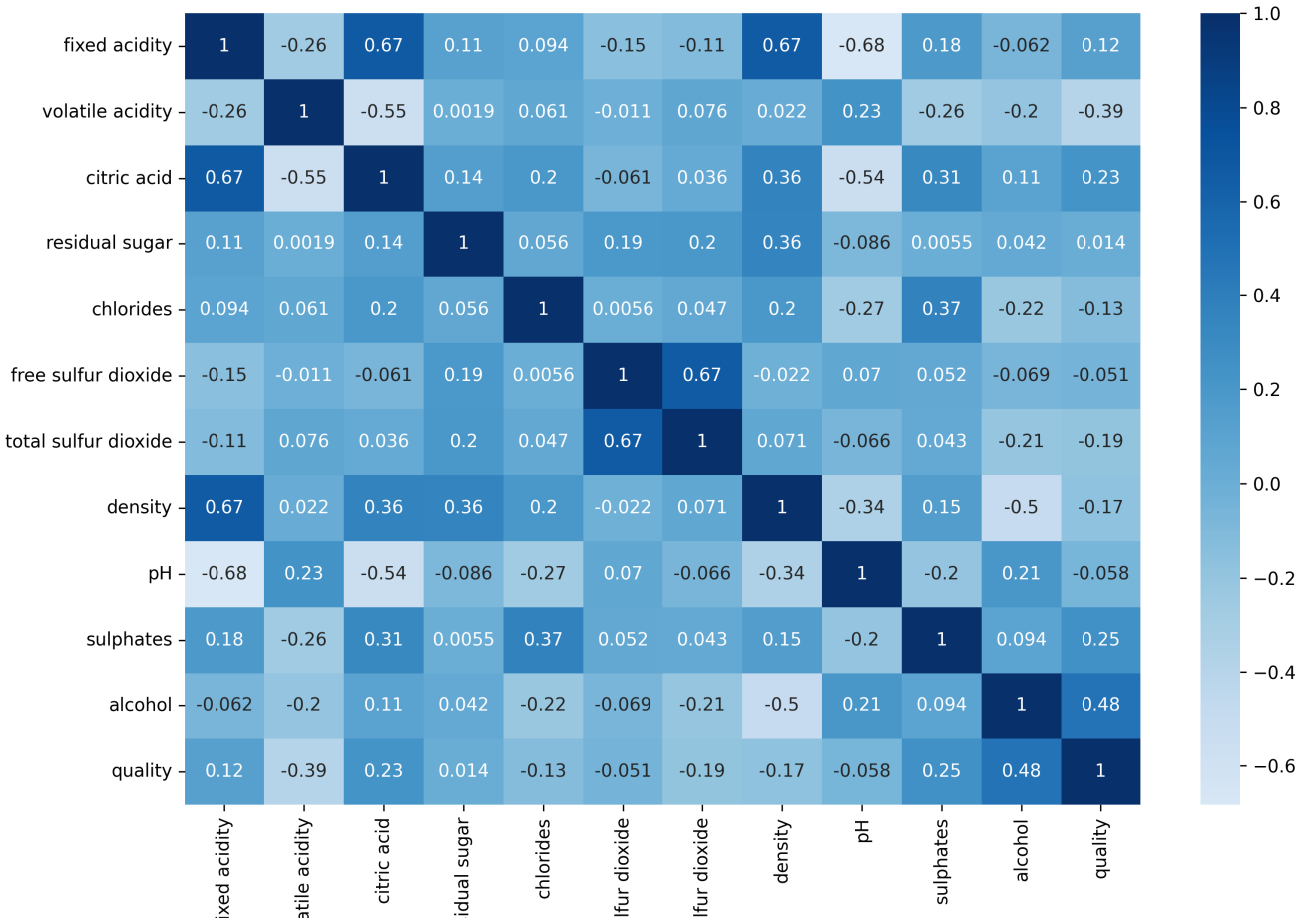

Figure 2. Correlation matrix.

TODO 1: Discuss how Figure 2 can be used to gain insights into the features that most influence the wine quality and to identify highly correlated features.

Preparing the dataset for model training

Firstly, we split the data into features (a.k.a. X) and target variables (a.k.a. y). In the case of our dataset, the features are the various chemical attributes of wine and the target variable is the quality of the wine.

```
# Drop the target variable
X = data.drop('quality', axis=1)
# Set the target variable as the label
y = data['quality']
```

Next, we split the data into training and testing sets. This is typically performed by randomly choosing a portion of the data as the test set and the remaining data as the training set. In this case, we use a split ratio of 4:1, where the larger potion is used for model training and the remaining smaller portion is used for model testing.

```
# Split the data into training and testing sets (20% testing and 80% training)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

5. Applying Logistic Regression

Logistic regression is a model for binary classification problems, where the goal is to predict the probability of an event occurring. We apply logistic regression to our dataset to predict whether a particular wine is Good (1) or Bad (0) based on its various input features (i.e., chemical and physical properties). Here, the Scikit-learn library can be used to implement the logistic regression model as follows:

```
# Create an object of the logistic regression model
logreg_model = LogisticRegression()
```

We can then train the model using the `fit` method of the scikit-learn's `LogisticRegression` class.

```
# Fit the model to the training data
logreg_model.fit(X_train, y_train)
```

Once the model is trained, we can use it to make predictions on the test data as follows:

```
# Predict the labels of the test set
y_pred = logreg_model.predict(X_test)
```

6. Evaluating Model Performance

Evaluating the performance of a model is crucial in determining how well the model is performing on the given dataset. One approach to evaluating the performance of a binary classification model such as logistic regression is to utilize a confusion matrix. It represents the number of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions obtained by the trained model (**Note:** You can expect to learn more about confusion matrices in Lab 4).

```
# Create the confusion matrix
confusion_mat = confusion_matrix(y_test, y_pred)
```

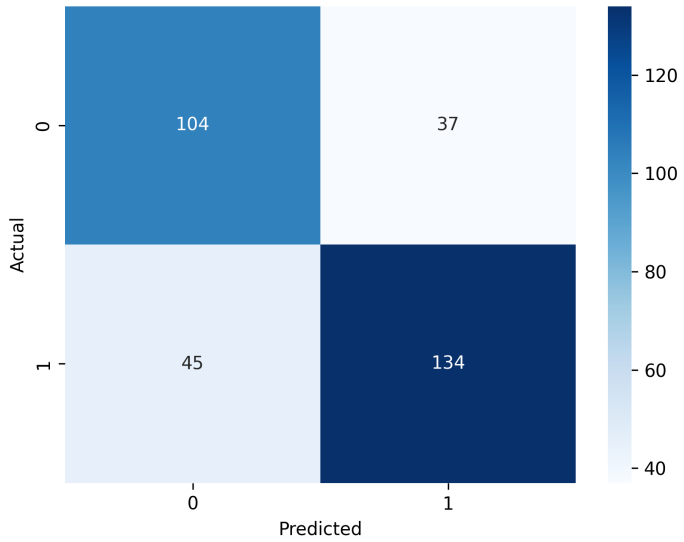

Figure 3. Confusion matrix.

Using the confusion matrix, we can calculate several performance metrics, such as **accuracy**, **precision**, **recall**, and **F1-score**.

Accuracy

This is the ratio of correct predictions to the total number of predictions.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. In other words, it refers to how precisely our model is able to predict the actual positives.

$$Precision = \frac{TP}{(TP + FP)}$$

Recall

Recall (a.k.a. sensitivity), on the other hand, is the ratio of correctly predicted positive observations to the total actual positive observations. In other words, it refers to how precisely our model is able to identify positive instances.

$$Recall = \frac{TP}{(TP + FN)}$$

F1-Score

This is the harmonic mean of precision and recall metrics. F1-score ranges from zero to one, where a value of one represents better precision and recall measures, and a value of zero represents that the model performs no better than random guessing.

$$F1 = 2.\frac{(Precision \times Recall)}{(Precision + Recall)}$$

To implement the above metrics, we can use `scikit-learn`.

```
# Compute the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)

# Compute the precision of the model
precision = precision_score(y_test, y_pred)

# Compute the recall of the model
recall = recall_score(y_test, y_pred)

# Compute the F1 score of the model
f1 = f1_score(y_test, y_pred)
```

The Receiver Operating Characteristic (ROC) Curve and the Area Under the Curve (AUC)

Moreover, the ROC curve and AUC are generally used to evaluate the performance of binary classifiers, including logistic regression models like ours. By visualizing the ROC curve and calculating the AUC, we can assess how well the model is able to distinguish between positive and negative samples and choose an appropriate classification threshold based on the expected trade-off between the TP rate and the FP rate.

```
# y_true and y_score are the true labels and predicted scores, respectively
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)

plt.figure(dpi=100)
plt.plot(fpr, tpr, color='blue', label='ROC curve (AUC = %0.2f)' % auc_score)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

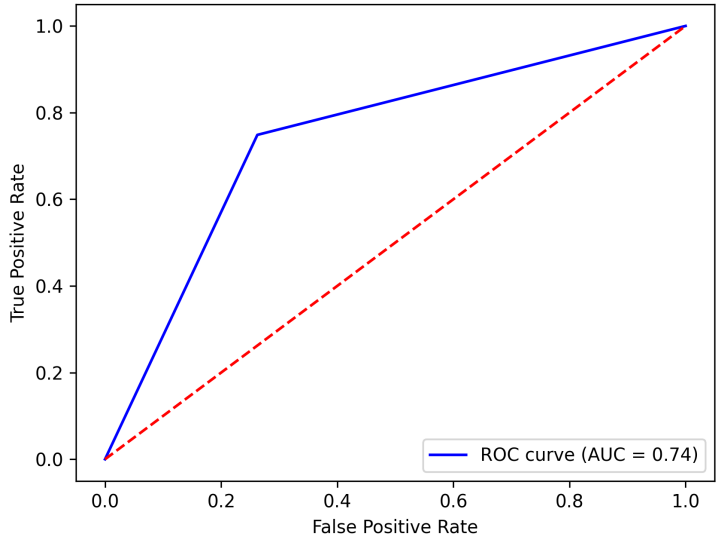

Figure 4. ROC curve.

TODO 2: What is the most appropriate performance metric to evaluate the model's performance? Briefly explain why.
TODO 3: Eliminate irrelevant features that may negatively affect the performance of the model and discuss how that may affect the model's performance using your chosen metric.
TODO 4: Try different test/train split ratios and evaluate the model performance in terms of your chosen metric to find the optimal split ratio.
TODO 5: Perform hyperparameter tuning for the logistic regression model using grid search to optimize the model's performance with the dataset. Try different hyperparameters, such as `C` (regularization strength), `penalty` (regularization type), `solver` (optimization algorithm), and `max_iter` (maximum number of iterations to converge). Compare the results with our initial model and choose the best set of hyperparameters.

7. Creating an Interactive Dashboard

7.1 Creating the Layout

Creating a dashboard layout involves defining the structure and design of the dashboard interface. In Dash, the layout of a dashboard is defined using HTML and CSS web technologies. Layouts are typically created using an integration of HTML tags, CSS classes, and Dash-specific components, such as `html.Div`, `dcc.Input`, and `dcc.Dropdown`.

In this lab, we first focus on creating a simple dashboard, which allows users to choose two features and visualize their correlation matrix as a scatter plot. We then integrate the function of predicting wine quality based on the user's input feature values using our logistic regression model already trained on the dataset.

```
# Create the Dash app
app = dash.Dash(__name__)

# Define the layout of the dashboard
app.layout = html.Div(
    children=[
    html.H1('CO544-2023 Lab 3: Wine Quality Prediction'),
    # Layout for exploratory data analysis: correlation between two selected features
    html.Div([
        html.H3('Exploratory Data Analysis'),
        html.Label('Feature 1 (X-axis)'),
        dcc.Dropdown(
            id='x_feature',
            options=[{'label': col, 'value': col} for col in data.columns],
            value=data.columns[0]
        )
    ], style={'width': '30%', 'display': 'inline-block'}),

    html.Div([
        html.Label('Feature 2 (Y-axis)'),
        dcc.Dropdown(
            id='y_feature',
            options=[{'label': col, 'value': col} for col in data.columns],
            value=data.columns[1]
        )
    ], style={'width': '30%', 'display': 'inline-block'}),

    dcc.Graph(id='correlation_plot'),

    # Layout for wine quality prediction based on input feature values
    html.H3("Wine Quality Prediction"),
    html.Div([
        html.Label("Fixed Acidity"),
        dcc.Input(id='fixed_acidity', type='number', required=True),
        html.Label("Volatile Acidity"),
        dcc.Input(id='volatile_acidity', type='number', required=True),
        html.Label("Citric Acid"),
        dcc.Input(id='citric_acid', type='number', required=True),
        html.Br(),

        html.Label("Residual Sugar"),
        dcc.Input(id='residual_sugar', type='number', required=True),
        html.Label("Chlorides"),
        dcc.Input(id='chlorides', type='number', required=True),
        html.Label("Free Sulfur Dioxide"),
        dcc.Input(id='free_sulfur_dioxide', type='number', required=True),
        html.Br(),
```

```
        html.Label("Total Sulfur Dioxide"),
        dcc.Input(id='total_sulfur_dioxide', type='number', required=True),
        html.Label("Density"),
        dcc.Input(id='density', type='number', required=True),
        html.Label("pH"),
        dcc.Input(id='ph', type='number', required=True),
        html.Br(),

        html.Label("Sulphates"),
        dcc.Input(id='sulphates', type='number', required=True),
        html.Label("Alcohol"),
        dcc.Input(id='alcohol', type='number', required=True),
        html.Br(),
    ]),

    html.Div([
        html.Button('Predict', id='predict-button', n_clicks=0),
    ]),

    html.Div([
        html.H4("Predicted Quality"),
        html.Div(id='prediction-output')
    ])
])
```

7.2 Adding Interactivity

To add interactivity to the dashboard, we can use callback functions, which are triggered when a user interacts with a component in the layout, such as selecting a feature from the dropdown menu and clicking the 'Predict' button after entering all the feature values.

```
# Define the callback to update the correlation plot
@app.callback(
    dash.dependencies.Output('correlation_plot', 'figure'),
    [dash.dependencies.Input('x_feature', 'value'),
     dash.dependencies.Input('y_feature', 'value')]
)
def update_correlation_plot(x_feature, y_feature):
    fig = px.scatter(data, x=x_feature, y=y_feature, color='quality')
    fig.update_layout(title=f"Correlation between {x_feature} and {y_feature}")
    return fig

# Define the callback function to predict wine quality
@app.callback(
    Output(component_id='prediction-output', component_property='children'),
    [Input('predict-button', 'n_clicks')],
    [State('fixed_acidity', 'value'),
     State('volatile_acidity', 'value'),
     State('citric_acid', 'value'),
     State('residual_sugar', 'value'),
     State('chlorides', 'value'),
     State('free_sulfur_dioxide', 'value'),
     State('total_sulfur_dioxide', 'value'),
     State('density', 'value'),
```

```
     State('ph', 'value'),
     State('sulphates', 'value'),
     State('alcohol', 'value')]
)
def predict_quality(n_clicks, fixed_acidity, volatile_acidity, citric_acid,
        residual_sugar, chlorides, free_sulfur_dioxide, total_sulfur_dioxide,
        density, ph, sulphates, alcohol):
    # Create input features array for prediction
    input_features = np.array([fixed_acidity, volatile_acidity, citric_acid,
            residual_sugar, chlorides, free_sulfur_dioxide,
            total_sulfur_dioxide, density, ph, sulphates, alcohol]).reshape(1, -1)

    # Predict the wine quality (0 = bad, 1 = good)
    prediction = logreg_model.predict(input_features)[0]

    # Return the prediction
    if prediction == 1:
        return 'This wine is predicted to be good quality.'
    else:
        return 'This wine is predicted to be bad quality.'
```

7.3 Running the Dashboard

The final step is to run the dashboard. We can do this by calling the `run_server` method of the Dash object.

```
if __name__ == '__main__':
    app.run_server(debug=False)
```

After running the dashboard and opening the server address, a web page will open with the layout and components specified in the implementation. Now, you can interact with the components and track changes on the dashboard based on your inputs.

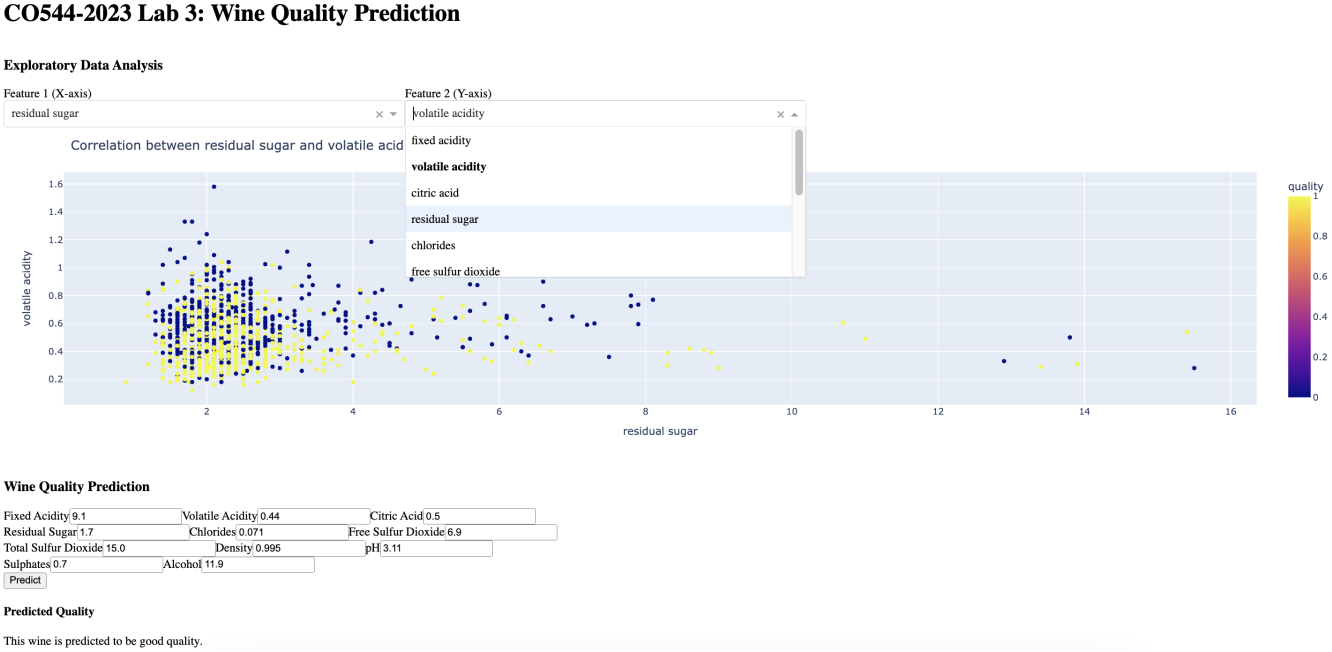

Figure 5. Our dashboard.

TODO 6: Deploy the dashboard on a cloud platform to make it accessible to a wider audience. The official documentation for deploying a Dash app to Heroku can be found at https://dash.plotly.com/deployment.

Repository: https://github.com/thusharabandara/co544-2023-lab3-interactive-python-dashboards

Instructions for Submitting the Report

Submit a short report containing your work on the given **TODOs** as a PDF file of no longer than five pages. Rename it as **e18xxxlab03.pdf**, where **xxx** is your registration number (Please note that you must include the shareable link of your Google Colab/Jupyter notebook as a reference to your report.)

***** End of Labsheet *****