Jhillian M. Cabos

Seatwork 11.1 Exploratory Data Analysis for Machine Learning

Due Apr 27 by 11:59pm **Points** 18 **Submitting** a file upload **File Types** pdf **Available** Apr 22 at :

Instructions:

- Download the datasets here:
  - For Linear Regression Analysis: https://archive-beta.ics.uci.edu/dataset/10/automobile
  - For Logistic Regression Analysis: https://archive-beta.ics.uci.edu/dataset/109/wine
- Perform exploratory data analysis (which must include data pre-processing/wrangling).
- Submit the notebook with the cleaned data and the EDA.

```
pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
automobile = fetch_ucirepo(id=10)

# data (as pandas dataframes)
X = automobile.data.features
y = automobile.data.targets

# metadata
print(automobile.metadata)

# variable information
print(automobile.variables)
```

```
{'uci_id': 10, 'name': 'Automobile', 'repository_url': 'https://archive.ics.uci.edu/dataset/10/automobile', 'data_url': 'https://arch
                 name     role         type demographic  \
0               price  Feature   Continuous        None
1         highway-mpg  Feature   Continuous        None
2            city-mpg  Feature   Continuous        None
3            peak-rpm  Feature   Continuous        None
4          horsepower  Feature   Continuous        None
5   compression-ratio  Feature   Continuous        None
6              stroke  Feature   Continuous        None
7                bore  Feature   Continuous        None
8         fuel-system  Feature  Categorical        None
9         engine-size  Feature   Continuous        None
10    num-of-cylinders  Feature      Integer        None
11         engine-type  Feature  Categorical        None
12         curb-weight  Feature   Continuous        None
13              height  Feature   Continuous        None
14               width  Feature   Continuous        None
15              length  Feature   Continuous        None
16          wheel-base  Feature   Continuous        None
17     engine-location  Feature       Binary        None
18        drive-wheels  Feature  Categorical        None
19          body-style  Feature  Categorical        None
20         num-of-doors  Feature      Integer        None
21          aspiration  Feature       Binary        None
22           fuel-type  Feature       Binary        None
23                make  Feature  Categorical        None
24   normalized-losses  Feature   Continuous        None
25           symboling   Target      Integer        None

                       description units missing_values
0    continuous from 5118 to 45400  None            yes
1         continuous from 16 to 54  None             no
2         continuous from 13 to 49  None             no
```

```
3                       continuous from 4150 to 6600  None           yes
4                          continuous from 48 to 288  None           yes
5                            continuous from 7 to 23  None            no
6                       continuous from 2.07 to 4.17  None           yes
7                       continuous from 2.54 to 3.94  None           yes
8       1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi  None            no
9                          continuous from 61 to 326  None            no
10        eight, five, four, six, three, twelve, two  None            no
11            dohc, dohcv, l, ohc, ohcf, ohcv, rotor  None            no
12                      continuous from 1488 to 4066  None            no
13                      continuous from 47.8 to 59.8  None            no
14                      continuous from 60.3 to 72.3  None            no
15                    continuous from 141.1 to 208.1  None            no
16                        continuous from 86.6 120.9  None            no
17                                       front, rear  None            no
18                                     4wd, fwd, rwd  None            no
19     hardtop, wagon, sedan, hatchback, convertible  None            no
20                                         four, two  None           yes
21                                        std, turbo  None            no
22                                       diesel, gas  None            no
23  alfa-romero, audi, bmw, chevrolet, dodge, hond...  None            no
24                         continuous from 65 to 256  None           yes
25                              -3, -2, -1, 0, 1, 2, 3  None            no
```

Linear Regression Analysis

Pre-processing and Wrangling

```
import pandas as pd
```

Concatinating X and y

```
am = pd.concat([X, y], axis = 1)
```

Showing columns

```
am.columns
```

```
Index(['price', 'highway-mpg', 'city-mpg', 'peak-rpm', 'horsepower',
       'compression-ratio', 'stroke', 'bore', 'fuel-system', 'engine-size',
       'num-of-cylinders', 'engine-type', 'curb-weight', 'height', 'width',
       'length', 'wheel-base', 'engine-location', 'drive-wheels', 'body-style',
       'num-of-doors', 'aspiration', 'fuel-type', 'make', 'normalized-losses',
       'symboling'],
      dtype='object')
```

```
am.head()
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	fuel-system	engi s
0	13495.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	
1	16500.0	27	21	5000.0	111.0	9.0	2.68	3.47	mpfi	
2	16500.0	26	19	5000.0	154.0	9.0	3.47	2.68	mpfi	
3	13950.0	30	24	5500.0	102.0	10.0	3.40	3.19	mpfi	
4	17450.0	22	18	5500.0	115.0	8.0	3.40	3.19	mpfi	

```
am.dropna(axis=0, inplace=True)  # Remove rows with missing values
```

```
am.head(5)
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	fuel-system	eng
3	13950.0	30	24	5500.0	102.0	10.0	3.4	3.19	mpfi	
4	17450.0	22	18	5500.0	115.0	8.0	3.4	3.19	mpfi	
6	17710.0	25	19	5500.0	110.0	8.5	3.4	3.19	mpfi	
8	23875.0	20	17	5500.0	140.0	8.3	3.4	3.13	mpfi	
10	16430.0	29	23	5800.0	101.0	8.8	2.8	3.50	mpfi	

The continuous variables in the 'am' will be normalized or standardized

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

```
# selected the columns containing continuous variables to be scaled
continuous_columns = ['price', 'highway-mpg', 'city-mpg', 'peak-rpm', 'horsepower', 'compression-ratio', 'stroke', 'bore', 'engine-size']

# min-max scaling
scaler_minmax = MinMaxScaler()
am[continuous_columns] = scaler_minmax.fit_transform(am[continuous_columns])

# standardization
scaler_standard = StandardScaler()
am[continuous_columns] = scaler_standard.fit_transform(am[continuous_columns])
```

```
am.head(5)
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore
3	0.427398	-0.323313	-0.414945	0.831733	0.201279	-0.041559	0.556703	-0.413240
4	1.024734	-1.565772	-1.402122	0.831733	0.625812	-0.557392	0.556703	-0.413240
6	1.069108	-1.099850	-1.237593	0.831733	0.462530	-0.428433	0.556703	-0.413240
8	2.121274	-1.876386	-1.566652	0.831733	1.442223	-0.480017	0.556703	-0.638386
10	0.850653	-0.478620	-0.579475	1.477884	0.168622	-0.351058	-1.484399	0.750015

```
am = am.sort_values(by='price', ascending=False)
```

After scaling, the continuous variables will have similar scales, making them suitable for machine learning algorithms that are sensitive to the scale of features.

```
am.head(5)
```

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore
72	4.029508	-2.187001	-1.731181	-0.783644	1.932070	-0.480017	-0.463848	0.599918
47	3.550615	-2.031694	-1.895711	-0.783644	2.617855	-0.531600	3.176117	1.237832
70	3.439681	-1.099850	-0.744004	-1.645178	0.887064	2.924481	1.373144	1.050210
68	2.867603	-1.099850	-0.744004	-1.645178	0.887064	2.924481	1.373144	1.050210
69	2.855315	-1.099850	-0.744004	-1.645178	0.887064	2.924481	1.373144	1.050210

EDA

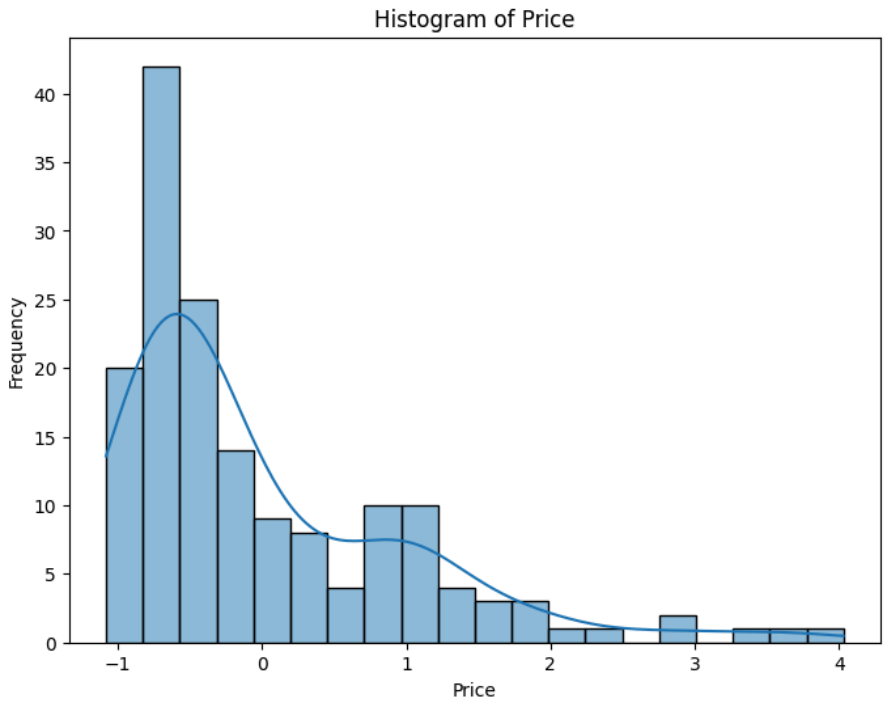
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
stats = am.describe()
stats
```

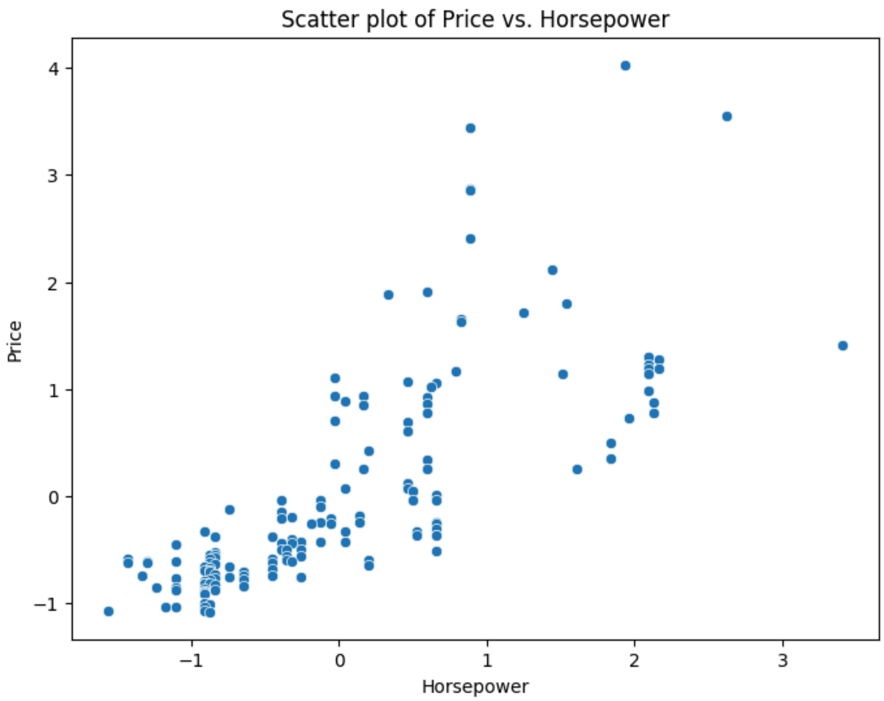
	price	highway-mpg	city-mpg	peak-rpm	horsepower	compressio rat
count	1.590000e+02	1.590000e+02	1.590000e+02	1.590000e+02	1.590000e+02	1.590000e-
mean	2.234411e-16	3.575058e-16	4.468822e-17	-3.016455e-16	-8.937644e-17	-5.586028e
std	1.003160e+00	1.003160e+00	1.003160e+00	1.003160e+00	1.003160e+00	1.003160e-
min	-1.079938e+00	-2.187001e+00	-1.895711e+00	-2.075946e+00	-1.562169e+00	-8.153080e
25%	-6.952536e-01	-6.339275e-01	-5.794748e-01	-6.759523e-01	-8.763838e-01	-3.768500e
50%	-3.776412e-01	-1.269809e-02	-8.588645e-02	1.855821e-01	-2.559115e-01	-2.994751e
75%	5.587265e-01	7.638387e-01	7.367609e-01	8.317329e-01	5.931560e-01	-1.963085e
max	4.029508e+00	3.404064e+00	3.698291e+00	3.200952e+00	3.401610e+00	3.311356e-

Next steps: View recommended plots

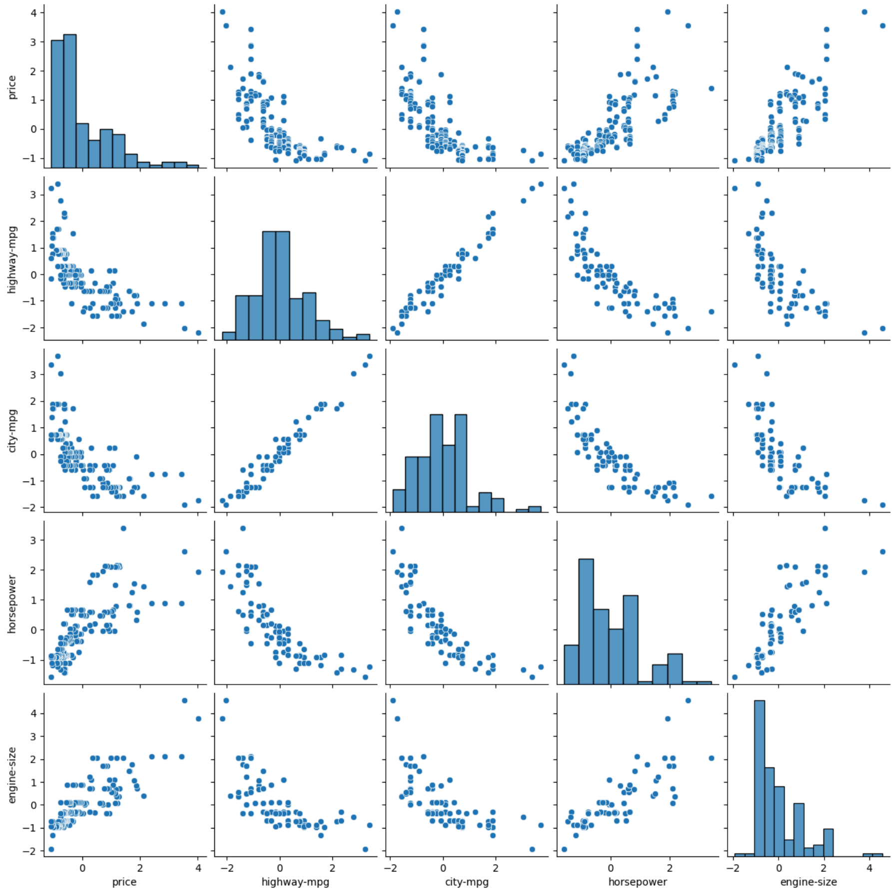
```
# Univariate Analysis
# Histogram of price
plt.figure(figsize=(8, 6))
sns.histplot(am['price'], bins=20, kde=True)
plt.title('Histogram of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



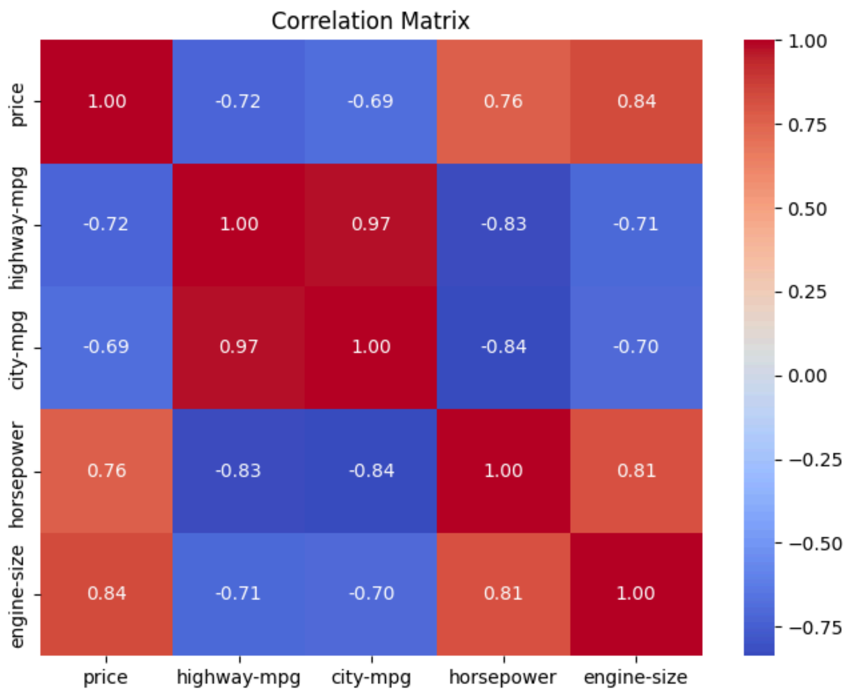
```
# Bivariate Analysis
# Scatter plot of price vs. horsepower
plt.figure(figsize=(8, 6))
sns.scatterplot(x='horsepower', y='price', data=am)
plt.title('Scatter plot of Price vs. Horsepower')
plt.xlabel('Horsepower')
plt.ylabel('Price')
plt.show()
```




```
# Multivariate Analysis
# Pairplot
sns.pairplot(am[['price', 'highway-mpg', 'city-mpg', 'horsepower', 'engine-size']])
plt.show()
```




```
# Correlation matrix
correlation_matrix = am[['price', 'highway-mpg', 'city-mpg', 'horsepower', 'engine-size']].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```




```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mse
```

```
0.19870103936570901
```

```
model.coef_
```

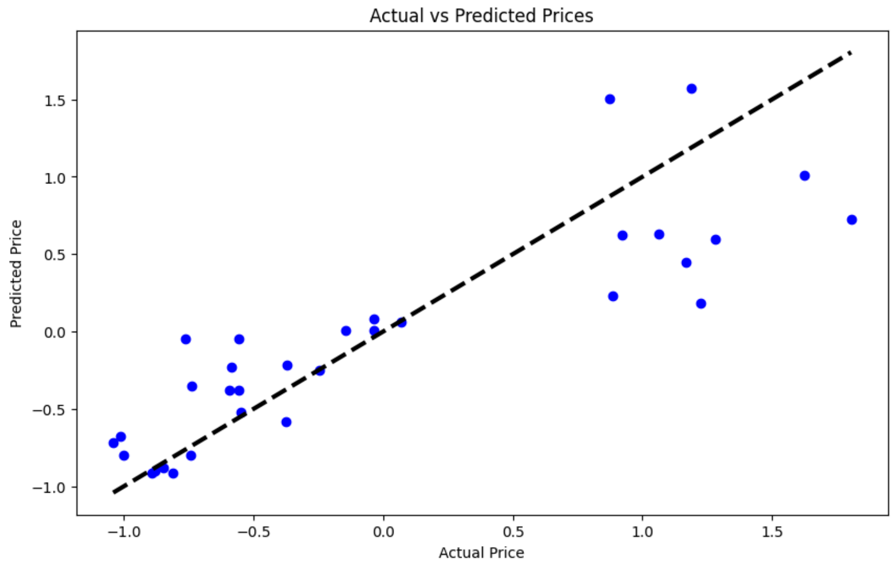
```
array([-0.5409909 ,  0.37277767,  0.0504379 ,  0.66039827])
```

```
model.intercept_
```

```
-0.00779849605116417
```

Data Visualization for the Linear Regression above

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=3)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices')
plt.show()
```



Conclusion for Linear Regression

Based on what I've learned from analyzing the provided dataset `am` using a linear regression model, I've uncovered some intriguing insights. Firstly, our model exhibits a promising level of predictive accuracy, demonstrated by a mean squared error (MSE). This indicates that while our model generally does a commendable job of estimating vehicle prices based on features like 'highway-mpg', 'city-mpg', 'horsepower', and 'engine-size', there's certainly room for fine-tuning to enhance our predictions further.

Delving deeper into the features, it's fascinating to observe that 'horsepower' emerges as the most influential predictor, positively influencing vehicle prices, closely trailed by 'engine-size'. Conversely, metrics of fuel efficiency such as 'highway-mpg' and 'city-mpg' exhibit negative correlations with price, hinting that consumers may place greater value on larger, more powerful vehicles. Our visual representation of actual versus predicted prices provides additional support for our model's effectiveness, albeit with notable outliers warranting closer examination. In conclusion, while our linear regression model offers valuable insights into pricing dynamics, there's ample opportunity for refinement, be it through exploring alternative models or optimizing our feature selection to achieve even stronger predictions.

Logistic Regression Analysis for the Wine Dataset

```
import seaborn as sns
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
wine = fetch_ucirepo(id=109)

# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets

# metadata
print(wine.metadata)

# variable information
print(wine.variables)
```

```
{'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.e
                         name     role         type demographic  \
0                       class   Target  Categorical        None
1                     Alcohol  Feature   Continuous        None
2                   Malicacid  Feature   Continuous        None
```

```
3                          Ash  Feature  Continuous         None
4              Alcalinity_of_ash  Feature  Continuous         None
5                    Magnesium  Feature     Integer         None
6                Total_phenols  Feature  Continuous         None
7                   Flavanoids  Feature  Continuous         None
8         Nonflavanoid_phenols  Feature  Continuous         None
9              Proanthocyanins  Feature  Continuous         None
10             Color_intensity  Feature  Continuous         None
11                         Hue  Feature  Continuous         None
12  0D280_0D315_of_diluted_wines  Feature  Continuous         None
13                     Proline  Feature     Integer         None

   description units missing_values
0         None  None             no
1         None  None             no
2         None  None             no
3         None  None             no
4         None  None             no
5         None  None             no
6         None  None             no
7         None  None             no
8         None  None             no
9         None  None             no
10        None  None             no
11        None  None             no
12        None  None             no
13        None  None             no
```

```
wn = pd.concat([X, y], axis = 1)
```

```
wn.columns
```

```
Index(['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium',
       'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',
       'Proanthocyanins', 'Color_intensity', 'Hue',
       '0D280_0D315_of_diluted_wines', 'Proline', 'class'],
      dtype='object')
```

```
wn.head(5)
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_intens
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	[illegible]
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	[illegible]
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	[illegible]
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	[illegible]
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	[illegible]

Next steps: View recommended plots

There's no missing values so no need for any rows or columns to be dropped

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize and fit the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾ LogisticRegression
LogisticRegression()
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)


# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)


print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9722222222222222

Classification Report:
               precision    recall  f1-score   support

           1       1.00      0.93      0.96        14
           2       0.93      1.00      0.97        14
           3       1.00      1.00      1.00         8

    accuracy                           0.97        36
   macro avg       0.98      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36


Confusion Matrix:
 [[13  1  0]
 [ 0 14  0]
 [ 0  0  8]]
```

```
wn.columns
```

```
Index(['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium',
       'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',
       'Proanthocyanins', 'Color_intensity', 'Hue',
       '0D280_0D315_of_diluted_wines', 'Proline', 'class'],
      dtype='object')
```

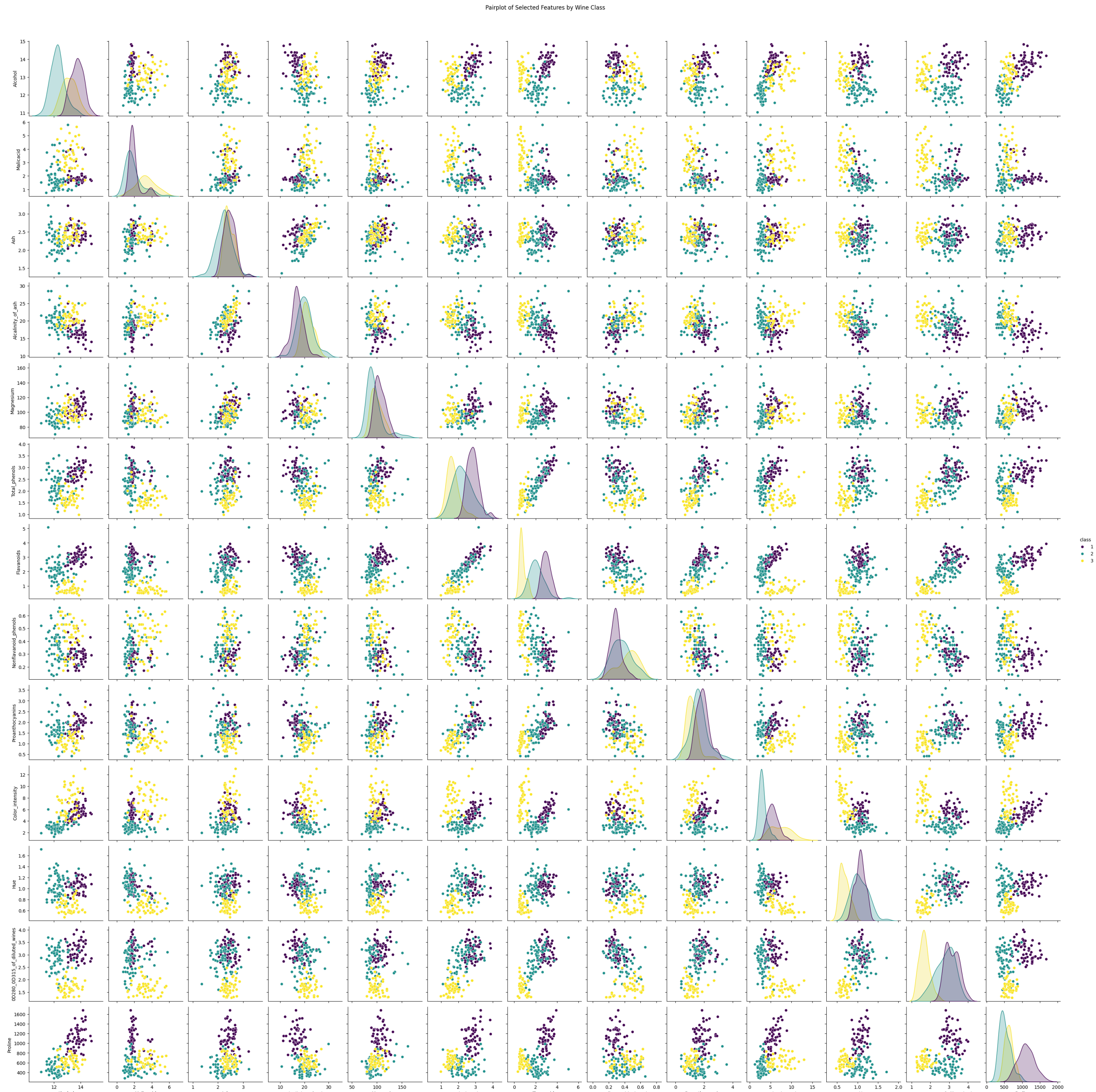
```
import matplotlib.pyplot as plt


# Subset of features for visualization
selected_features = ['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols','I

# Subset the DataFrame
wn_subset = wn[selected_features]

# Pairplot for pairwise relationships
sns.pairplot(wn_subset, hue='class', palette='viridis')
plt.suptitle("Pairplot of Selected Features by Wine Class", y=1.02)
plt.show()

# Boxplots for selected features
plt.figure(figsize=(12, 8))
for i, feature in enumerate(['Alcohol', 'Malic acid', 'Ash', 'Magnesium', 'Total phenols']):
    plt.subplot(2, 3, i+1)
    sns.boxplot(x='class', y=feature, data= wn_subset)
    plt.title(f'Boxplot of {feature} by Wine Class')
plt.tight_layout()
plt.show()
```




```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-8be7bd749eb7> in <cell line: 14>()
     14 for i, feature in enumerate(['Alcohol', 'Malic acid', 'Ash', 'Magnesium', 'Total phenols']):
     15     plt.subplot(2, 3, i+1)
---> 16     sns.boxplot(x='class', y=feature, data= wn_subset)
     17     plt.title(f'Boxplot of {feature} by Wine Class')
     18 plt.tight_layout()

5 frames

/usr/local/lib/python3.10/dist-packages/seaborn/_core/data.py in _assign_variables(self, data, variables)
    230                 else:
    231                     err += "An entry with this name does not appear in `data`."
--> 232                 raise ValueError(err)
    233 
    234             else:

ValueError: Could not interpret value `Malic acid` for `y`. An entry with this name does not appear in `data`.
```