Name: Dylan James N. Dejoras

Section: CPE22S3

source:
https://archive.ics.uci.edu/dataset/10/automobile
https://archive.ics.uci.edu/dataset/109/wine

Setup

```
pip install ucimlrepo
```

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

```
import pandas as pd
import numpy as np

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://archive
                 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
```

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

	price	highway-mpg	city-mpg	peak-rpm	horsepower	compression-ratio	stroke	bore	fuel-system	en
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	
...	...	...	...	...	...	...	...	...	...	
200	16845.0	28	23	5400.0	114.0	9.5	3.15	3.78	mpfi	
201	19045.0	25	19	5300.0	160.0	8.7	3.15	3.78	mpfi	
202	21485.0	23	18	5500.0	134.0	8.8	2.87	3.58	mpfi	
203	22470.0	27	26	4800.0	106.0	23.0	3.40	3.01	idi	
204	22625.0	25	19	5400.0	114.0	9.5	3.15	3.78	mpfi	

205 rows × 26 columns

```
auto_df = df.copy()
```

```
def check_duplicates(df):
  if df[df.duplicated()].shape[0] != 0:
    print(df[df.duplicated()].shape[0])
  else:
    print("No existing duplicates")
check_duplicates(auto_df)
```

```
No existing duplicates
```

```
df.isnull().sum()
```

```
price                4
highway-mpg          0
city-mpg             0
peak-rpm             2
horsepower           2
compression-ratio    0
stroke               4
bore                 4
fuel-system          0
engine-size          0
num-of-cylinders     0
engine-type          0
curb-weight          0
```

```
height               0
width                0
length               0
wheel-base           0
engine-location      0
drive-wheels         0
body-style           0
num-of-doors         2
aspiration           0
fuel-type            0
make                 0
normalized-losses   41
symboling            0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   price              201 non-null    float64
 1   highway-mpg        205 non-null    int64  
 2   city-mpg           205 non-null    int64  
 3   peak-rpm           203 non-null    float64
 4   horsepower         203 non-null    float64
 5   compression-ratio  205 non-null    float64
 6   stroke             201 non-null    float64
 7   bore               201 non-null    float64
 8   fuel-system        205 non-null    object 
 9   engine-size        205 non-null    int64  
 10  num-of-cylinders   205 non-null    int64  
 11  engine-type        205 non-null    object 
 12  curb-weight        205 non-null    int64  
 13  height             205 non-null    float64
 14  width              205 non-null    float64
 15  length             205 non-null    float64
 16  wheel-base         205 non-null    float64
 17  engine-location    205 non-null    object 
 18  drive-wheels       205 non-null    object 
 19  body-style         205 non-null    object 
 20  num-of-doors       203 non-null    float64
 21  aspiration         205 non-null    object 
 22  fuel-type          205 non-null    object 
 23  make               205 non-null    object 
 24  normalized-losses  164 non-null    float64
 25  symboling          205 non-null    int64  
dtypes: float64(12), int64(6), object(8)
memory usage: 41.8+ KB
```

```
def check_value_counts(column):
  print(df.value_counts(column))
```

```
check_value_counts('normalized-losses')
```

```
normalized-losses
161.0    11
91.0      8
150.0     7
128.0     6
104.0     6
134.0     6
95.0      5
94.0      5
74.0      5
65.0      5
103.0     5
85.0      5
168.0     5
102.0     5
122.0     4
148.0     4
106.0     4
118.0     4
93.0      4
101.0     3
154.0     3
115.0     3
83.0      3
125.0     3
```

```
137.0     3
87.0      2
188.0     2
158.0     2
153.0     2
81.0      2
145.0     2
192.0     2
89.0      2
129.0     2
194.0     2
197.0     2
119.0     2
113.0     2
110.0     2
108.0     2
164.0     2
186.0     1
231.0     1
142.0     1
77.0      1
78.0      1
98.0      1
90.0      1
121.0     1
107.0     1
256.0     1
Name: count, dtype: int64
```

```
def fill_missing_values(df, columns):
    for col in columns:
        df[col] = df[col].fillna(df[col].median())
    return df
```

```
na_counts = df.isnull().sum()
```

```
columns_with_na = na_counts[na_counts > 0].index.tolist()
```

```
columns_with_na
```

```
['price',
 'peak-rpm',
 'horsepower',
 'stroke',
 'bore',
 'num-of-doors',
 'normalized-losses']
```

```
auto_df = fill_missing_values(df, columns_with_na)
```

```
auto_df.isnull().sum()
```

```
price                0
highway-mpg          0
city-mpg             0
peak-rpm             0
horsepower           0
compression-ratio    0
stroke               0
bore                 0
fuel-system          0
engine-size          0
num-of-cylinders     0
engine-type          0
curb-weight          0
height               0
width                0
length               0
wheel-base           0
engine-location      0
drive-wheels         0
body-style           0
num-of-doors         0
aspiration           0
fuel-type            0
make                 0
normalized-losses    0
```

```
symboling            0
dtype: int64
```

```
auto_df.dtypes
```

```
price                float64
highway-mpg            int64
city-mpg               int64
peak-rpm             float64
horsepower           float64
compression-ratio    float64
stroke               float64
bore                 float64
fuel-system           object
engine-size            int64
num-of-cylinders       int64
engine-type           object
curb-weight            int64
height               float64
width                float64
length               float64
wheel-base           float64
engine-location       object
drive-wheels          object
body-style            object
num-of-doors         float64
aspiration            object
fuel-type             object
make                  object
normalized-losses    float64
symboling              int64
dtype: object
```

```
Fuel_system_type = list(df['fuel-system'].unique())
Engine_type = list(df['engine-type'].unique())
Engine_location_type = list(df['engine-location'].unique())
Drive_wheels_type = list(df['drive-wheels'].unique())
Body_style_type = list(df['body-style'].unique())
Aspiration_type = list(df['aspiration'].unique())
Fuel_type = list(df['fuel-type'].unique())
Make_type = list(df['make'].unique())
```

```
df['fuel-system'] = df.apply(lambda x: Fuel_system_type.index(x['fuel-system']) + 1, axis=1)
df['engine-type'] = df.apply(lambda x: Engine_type.index(x['engine-type']) + 1, axis=1)
df['engine-location'] = df.apply(lambda x: Engine_location_type.index(x['engine-location']) + 1, axis=1)
df['drive-wheels'] = df.apply(lambda x: Drive_wheels_type.index(x['drive-wheels']) + 1, axis=1)
df['body-style'] = df.apply(lambda x: Body_style_type.index(x['body-style']) + 1, axis=1)
df['aspiration'] = df.apply(lambda x: Aspiration_type.index(x['aspiration']) + 1, axis=1)
df['fuel-type'] = df.apply(lambda x: Fuel_type.index(x['fuel-type']) + 1, axis=1)
df['make'] = df.apply(lambda x: Make_type.index(x['make']) + 1, axis=1)
```

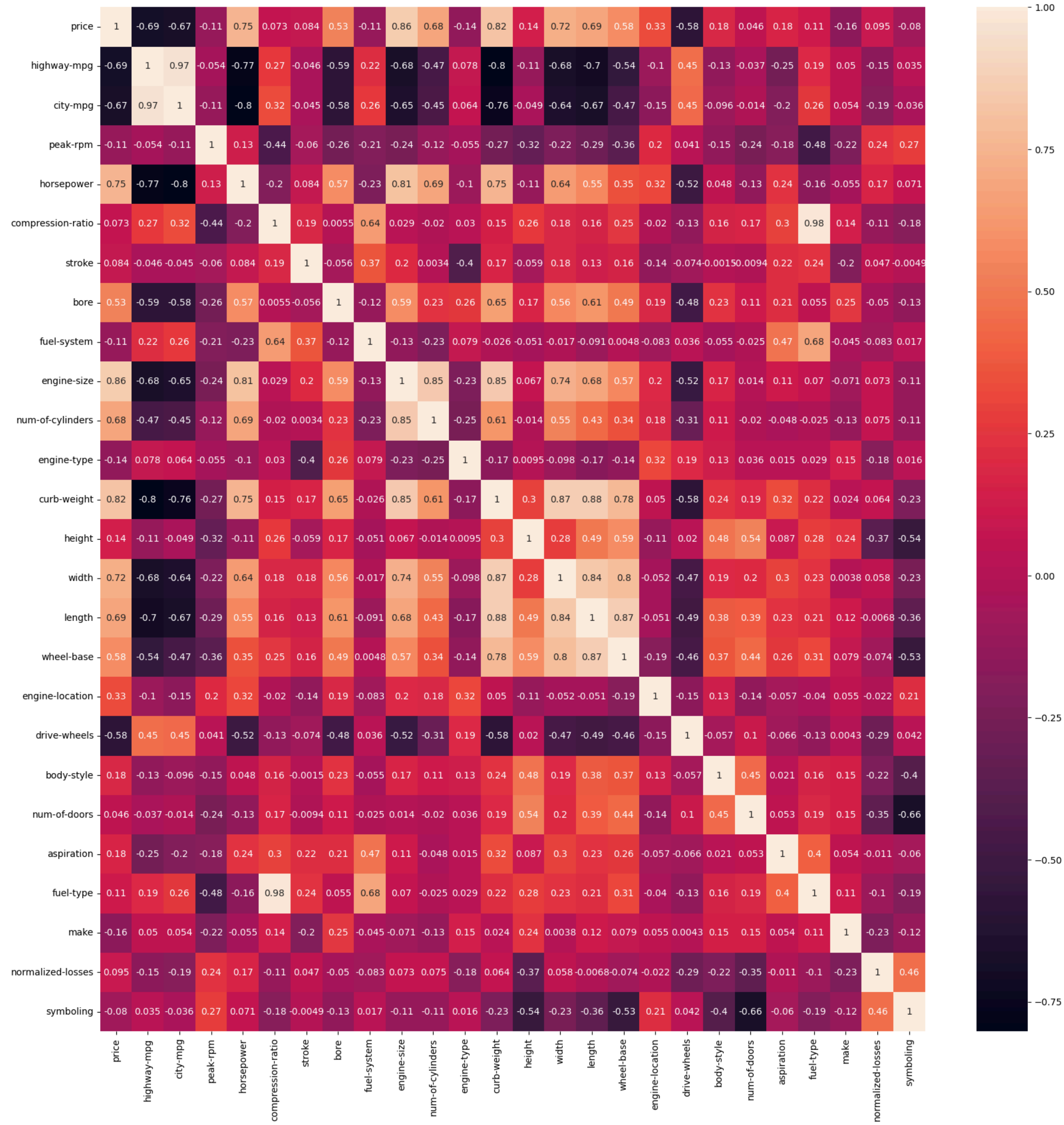
```
auto_df.dtypes
```

```
price                float64
highway-mpg            int64
city-mpg               int64
peak-rpm             float64
horsepower           float64
compression-ratio    float64
stroke               float64
bore                 float64
fuel-system            int64
engine-size            int64
num-of-cylinders       int64
engine-type            int64
curb-weight            int64
height               float64
width                float64
length               float64
wheel-base           float64
engine-location        int64
drive-wheels           int64
body-style             int64
num-of-doors         float64
aspiration             int64
fuel-type              int64
make                   int64
normalized-losses    float64
```

```
symboling             int64
dtype: object
```

```
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(20, 20))
sns.heatmap(auto_df.corr(), annot=True)
```

```
<Axes: >
```



```
X = df.drop('horsepower', axis=1)
y = df['horsepower']
```

```
print("X=",X.shape,"\ny=",y.shape)
```

```
X= (205, 25)
y= (205,)
```

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

```
X_train, X_test, y_train, y_test =  train_test_split(X, y, test_size=0.3, random_state=101)
```

```
X_train.shape
```

```
(143, 25)
```

```
X_test.shape
```

```
(62, 25)
```

```
model = LinearRegression()
```

```
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
model.coef_
```

```
array([-4.93081843e-04,  3.50592614e+00, -4.34114068e+00,  1.96062716e-02,
        5.22089428e+00,  7.30210367e+00,  2.44507227e+01, -7.58307630e-01,
        2.63253993e-01,  1.02400233e+01,  1.45818418e+00,  1.84326982e-02,
       -3.67802358e-02,  1.54274355e+00, -7.75070944e-02, -8.05226840e-01,
        2.00206382e+01, -9.60977760e+00, -9.58622509e-01, -1.46993056e+00,
        3.40275905e+01, -9.03368882e+01,  6.30398597e-03, -4.63055804e-02,
       -2.36585941e-01])
```

```
pd.DataFrame(model.coef_, X.columns, columns=['Coefficients'])
```

	Coefficients
price	-0.000493
highway-mpg	3.505926
city-mpg	-4.341141
peak-rpm	0.019606
compression-ratio	5.220894
stroke	7.302104
bore	24.450723
fuel-system	-0.758308
engine-size	0.263254
num-of-cylinders	10.240023
engine-type	1.458184
curb-weight	0.018433
height	-0.036780
width	1.542744
length	-0.077507
wheel-base	-0.805227
engine-location	20.020638
drive-wheels	-9.609778
body-style	-0.958623
num-of-doors	-1.469931
aspiration	34.027590
fuel-type	-90.336888
make	0.006304
normalized-losses	-0.046306
symboling	-0.236586

```
y_pred = model.predict(X_test)
```

```
MAE = metrics.mean_absolute_error(y_test, y_pred)
MSE = metrics.mean_squared_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
```

Average Error

```
MAE
```

```
9.592603226136523
```

Mean Square Error

```
MSE
```

```
180.35954074625744
```

Root Square Mean Error

```
RMSE
```

```
13.429800473062041
```

average of horsepower

```
df['horsepower'].mean()
```

```
104.16585365853659
```

subtracting y test and y pred yields test residual

```
test_residual = y_test - y_pred
```

```
!pip install hvplot
```

```
Requirement already satisfied: hvplot in /usr/local/lib/python3.10/dist-packages (0.9.2)
Requirement already satisfied: bokeh>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (3.3.4)
Requirement already satisfied: colorcet>=2 in /usr/local/lib/python3.10/dist-packages (from hvplot) (3.1.0)
Requirement already satisfied: holoviews>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.17.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from hvplot) (2.0.3)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.25.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from hvplot) (24.0)
Requirement already satisfied: panel>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.3.8)
Requirement already satisfied: param<3.0,>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (2.1.0)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (3.1.3)
Requirement already satisfied: contourpy>=1 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (1.2.1)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (9.4.0)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (6.0.1)
Requirement already satisfied: tornado>=5.1 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (6.3.3)
Requirement already satisfied: xyzservices>=2021.09.1 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (2024.4.0)
Requirement already satisfied: pyviz-comms>=0.7.4 in /usr/local/lib/python3.10/dist-packages (from holoviews>=1.11.0->hvplot) (3.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2024.1)
Requirement already satisfied: markdown in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (3.6)
Requirement already satisfied: markdown-it-py in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (3.0.0)
Requirement already satisfied: linkify-it-py in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (2.0.3)
Requirement already satisfied: mdit-py-plugins in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (0.4.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (2.31.0)
Requirement already satisfied: tqdm>=4.48.0 in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (4.66.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (6.1.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (4.11.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh>=1.0.0->hvplot) (2.1.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->hvplot) (1.16.0
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->panel>=0.11.0->hvplot) (0.5.1)
Requirement already satisfied: uc-micro-py in /usr/local/lib/python3.10/dist-packages (from linkify-it-py->panel>=0.11.0->hvplot) (1.0.3
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py->panel>=0.11.0->hvplot) (0.1.2
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (2.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (202
```

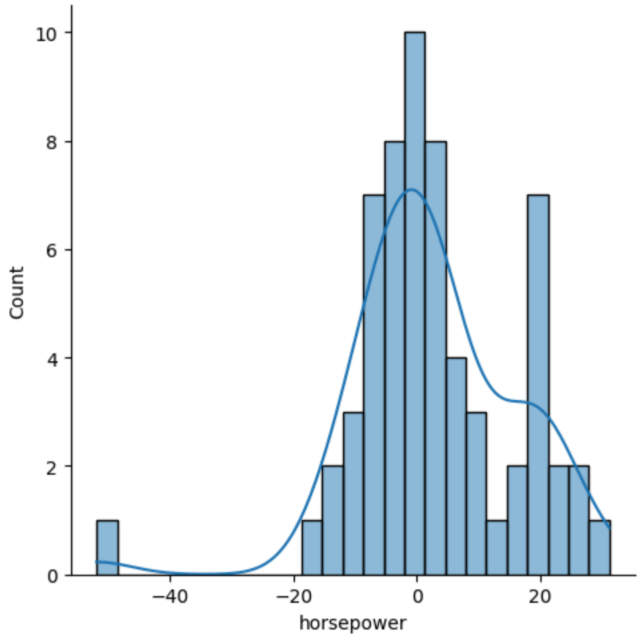
```
import hvplot.pandas
```

```
pd.DataFrame({'Error Values': (test_residual)}).hvplot.kde()
```

```
/usr/local/lib/python3.10/dist-packages/holoviews/core/util.py:1585: PanelDeprecationWarning: 'param_value_if_widget' is deprecated and
  value = param_value_if_widget(value)
```

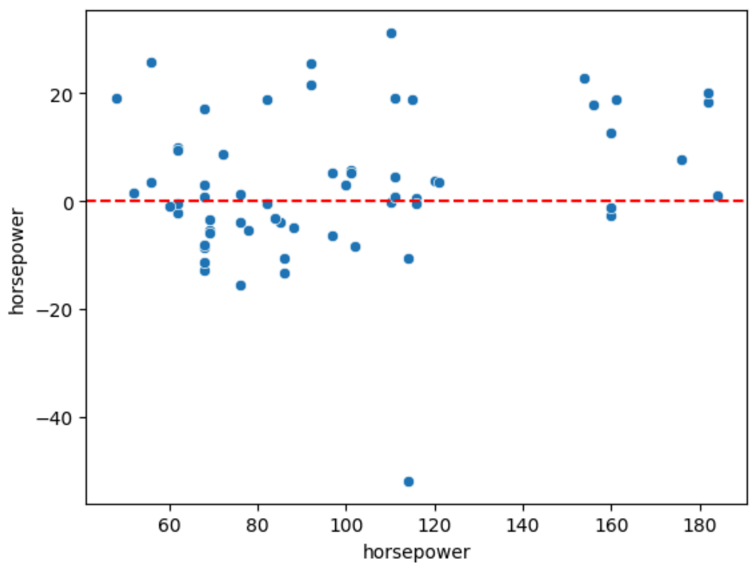
```
sns.displot(test_residual,bins=25, kde=True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7e92c81ed570>
```




```
sns.scatterplot(x=y_test, y=test_residual)
plt.axhline(y=0, color='r', ls='--')
```

```
<matplotlib.lines.Line2D at 0x7e92c8054c10>
```



```
# 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
```

```
dataFrames = [x,Y]
wine_df = pd.concat(dataFrames, axis = 1)
wine_df
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_inte
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	
...	...	...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	

178 rows × 14 columns

Next steps: View recommended plots

```
check_duplicates(wine_df)
```

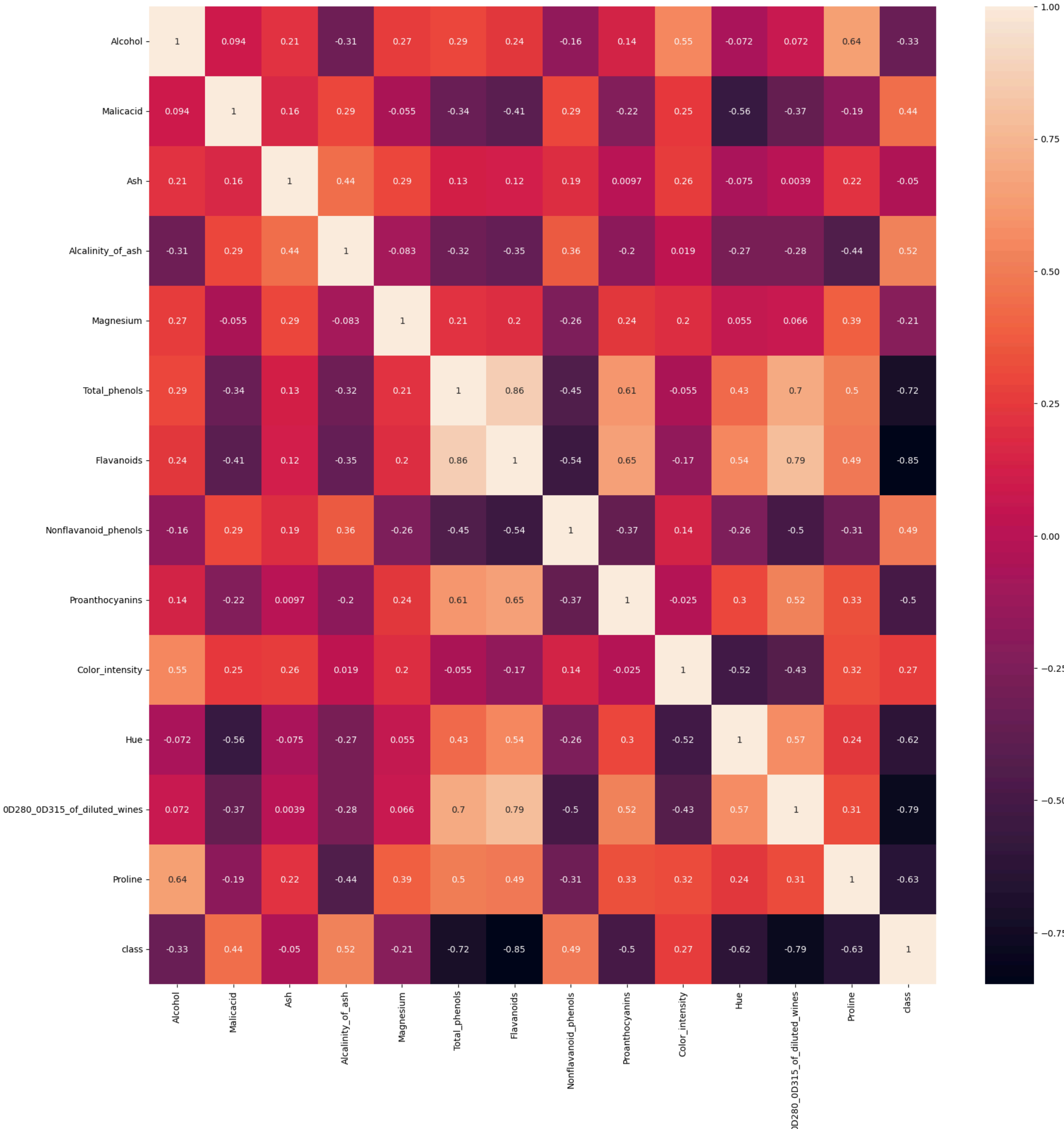
```
No existing duplicates
```

```
wine_df.isnull().sum()
```

```
Alcohol                         0
Malicacid                       0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
class                           0
dtype: int64
```

```
plt.figure(figsize=(20, 20))
sns.heatmap(wine_df.corr(), annot=True)
```

```
<Axes: >
```


Alcohol
Malicacid
Ash
Alcalinity_of_ash
Magnesium
Total_phenols
Flavanoids
Nonflavanoid_phenols
Proanthocyanins
Color_intensity
Hue
0D280_0D315_of_diluted_wines
Proline
class
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

```
wine_df.shape
```

```
(178, 14)
```

```
wine_df.head()
```

	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

```
wine_df.dtypes
```

```
Alcohol                         float64
Malicacid                       float64
Ash                             float64
Alcalinity_of_ash               float64
Magnesium                         int64
Total_phenols                   float64
Flavanoids                      float64
Nonflavanoid_phenols            float64
Proanthocyanins                 float64
Color_intensity                 float64
Hue                             float64
0D280_0D315_of_diluted_wines    float64
Proline                           int64
class                             int64
dtype: object
```

We would deal with continuous variables since there is no categorical variables as the usual variable for logistic regression.

```
categorical = [var for var in wine_df.columns if wine_df[var].dtype=='O']
print(categorical)
```

```
[]
```

```
numerical = [var for var in wine_df.columns if wine_df[var].dtype!='O']
print(numerical)
```

```
['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanin
```

```
wine_df[categorical].sum()
```

```
Series([], dtype: float64)
```

```
wine_df[numerical].isnull().sum()
```

```
Alcohol                 0
Malicacid               0
Ash                     0
Alcalinity_of_ash       0
```

```
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
class                           0
dtype: int64
```

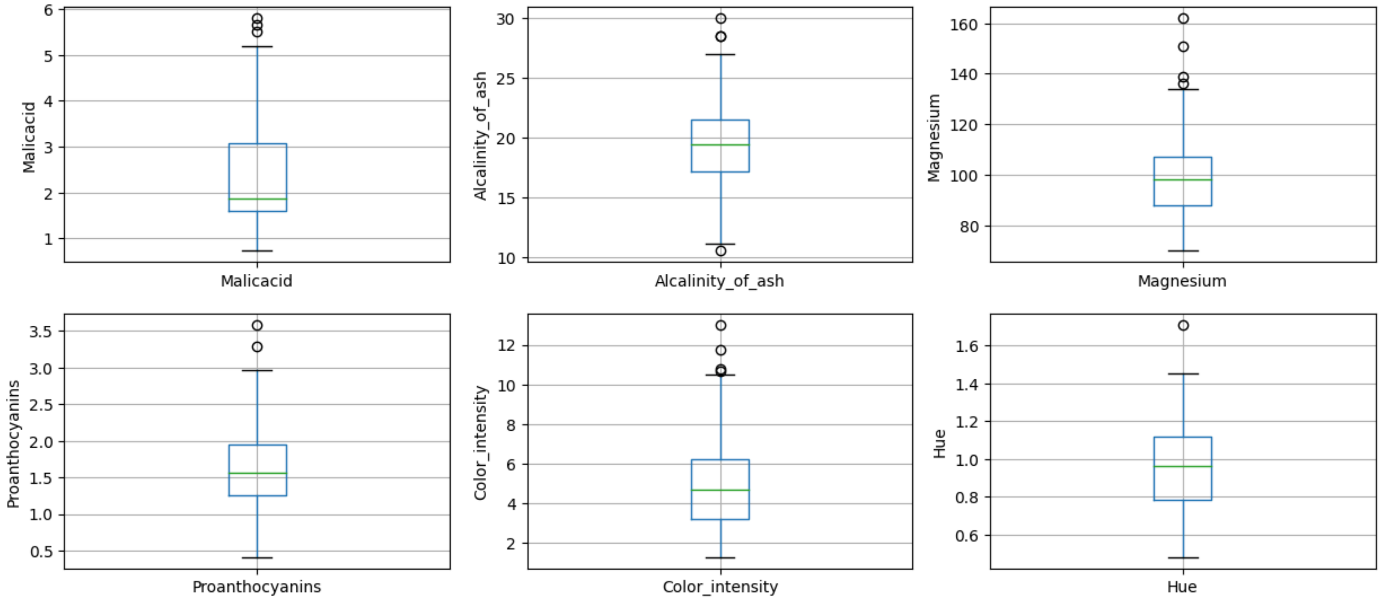
Outliers in our dataset

```
print(round(wine_df[numerical].describe()),2)
```

```
       Alcohol  Malicacid    Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
count    178.0      178.0  178.0              178.0      178.0          178.0
mean      13.0        2.0    2.0               19.0      100.0            2.0
std        1.0        1.0    0.0                3.0       14.0            1.0
min       11.0        1.0    1.0               11.0       70.0            1.0
25%       12.0        2.0    2.0               17.0       88.0            2.0
50%       13.0        2.0    2.0               20.0       98.0            2.0
75%       14.0        3.0    3.0               22.0      107.0            3.0
max       15.0        6.0    3.0               30.0      162.0            4.0

       Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity  \
count       178.0                 178.0            178.0            178.0
mean          2.0                   0.0              2.0              5.0
std           1.0                   0.0              1.0              2.0
min           0.0                   0.0              0.0              1.0
25%           1.0                   0.0              1.0              3.0
50%           2.0                   0.0              2.0              5.0
75%           3.0                   0.0              2.0              6.0
max           5.0                   1.0              4.0             13.0

         Hue  0D280_0D315_of_diluted_wines  Proline  class
count  178.0                         178.0    178.0  178.0
mean     1.0                           3.0    747.0    2.0
std      0.0                           1.0    315.0    1.0
min      0.0                           1.0    278.0    1.0
25%      1.0                           2.0    500.0    1.0
50%      1.0                           3.0    674.0    2.0
75%      1.0                           3.0    985.0    3.0
max      2.0                           4.0   1680.0    3.0   2
```

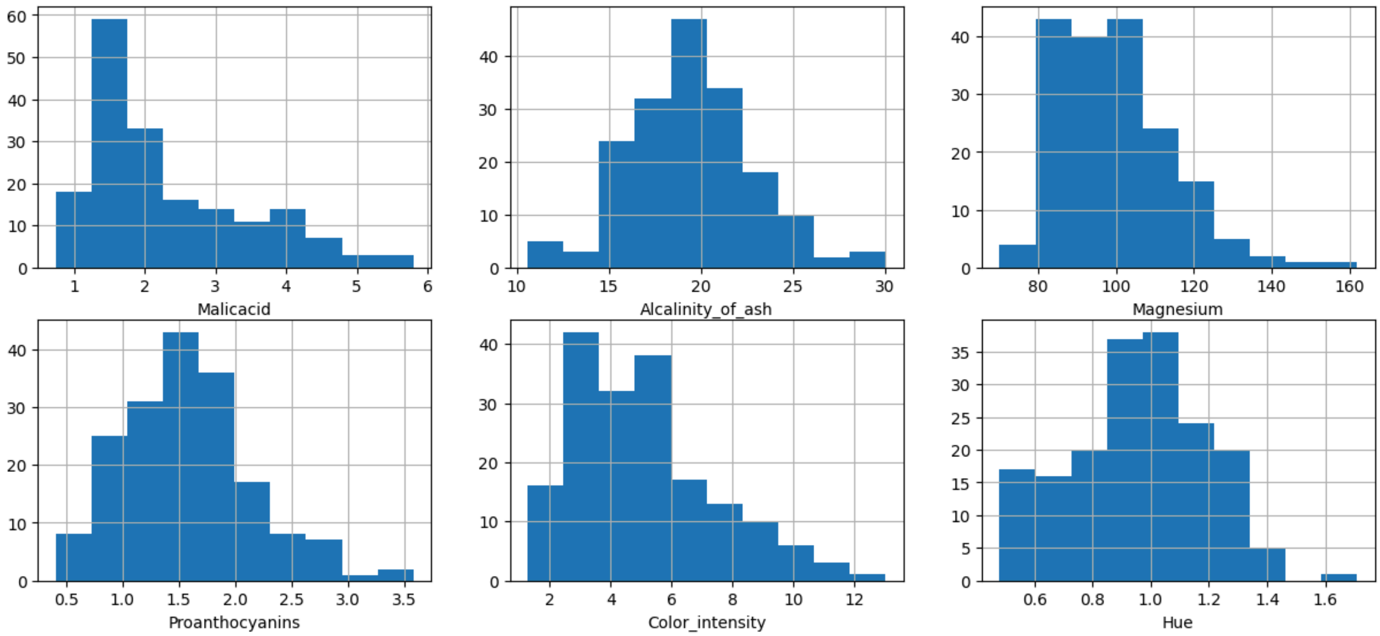
```
plt.figure(figsize=(15,10))

plt.subplot(3, 3, 1)
fig = wine_df.boxplot(column = 'Malicacid')
fig.set_title('')
fig.set_ylabel('Malicacid')

plt.subplot(3, 3, 2)
fig = wine_df.boxplot(column = 'Alcalinity_of_ash')
fig.set_title('')
fig.set_ylabel('Alcalinity_of_ash')

plt.subplot(3, 3, 3)
fig = wine_df.boxplot(column = 'Magnesium')
fig.set_title('')
fig.set_ylabel('Magnesium')

plt.subplot(3, 3, 4)
fig = wine_df.boxplot(column = 'Proanthocyanins')
fig.set_title('')
fig.set_ylabel('Proanthocyanins')

plt.subplot(3, 3, 5)
fig = wine_df.boxplot(column = 'Color_intensity')
fig.set_title('')
fig.set_ylabel('Color_intensity')

plt.subplot(3, 3, 6)
fig = wine_df.boxplot(column = 'Hue')
fig.set_title('')
fig.set_ylabel('Hue')
```

```
Text(0, 0.5, 'Hue')
```




```
plt.figure(figsize=(15,10))

plt.subplot(3, 3, 1)
fig = wine_df.Malicacid.hist(bins=10)
fig.set_xlabel('  ')
fig.set_ylabel('')

plt.subplot(3, 3, 2)
fig = wine_df.Alcalinity_of_ash.hist(bins=10)
fig.set_xlabel('Alcalinity_of_ash')
fig.set_ylabel('')

plt.subplot(3, 3, 3)
fig = wine_df.Magnesium.hist(bins=10)
fig.set_xlabel('Magnesium')
fig.set_ylabel('')

plt.subplot(3, 3, 4)
fig = wine_df.Proanthocyanins.hist(bins=10)
fig.set_xlabel('Proanthocyanins')
fig.set_ylabel('')

plt.subplot(3, 3, 5)
fig = wine_df.Color_intensity.hist(bins=10)
fig.set_xlabel('Color_intensity')
fig.set_ylabel('')

plt.subplot(3, 3, 6)
fig = wine_df.Hue.hist(bins=10)
fig.set_xlabel('Hue')
fig.set_ylabel('')
```

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




```
IQR = wine_df.Malicacid.quantile(0.75) - wine_df.Malicacid.quantile(0.25)
Lower_fence = wine_df.Malicacid.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Malicacid.quantile(0.75) + (IQR * 3)
print('Malicacid outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fence))
```

```
Malicacid outliers are values < -2.8374999999999995 or > 7.522499999999999
```

```
IQR = wine_df.Alcalinity_of_ash.quantile(0.75) - wine_df.Alcalinity_of_ash.quantile(0.25)
Lower_fence = wine_df.Alcalinity_of_ash.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Alcalinity_of_ash.quantile(0.75) + (IQR * 3)
print('Alcalinity_of_ash outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fe
```

```
Alcalinity_of_ash outliers are values < 4.299999999999997 or > 34.400000000000006
```

```
IQR = wine_df.Magnesium.quantile(0.75) - wine_df.Magnesium.quantile(0.25)
Lower_fence = wine_df.Magnesium.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Magnesium.quantile(0.75) + (IQR * 3)
print('Magnesium outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fence))
```

```
Magnesium outliers are values < 31.0 or > 164.0
```

```
IQR = wine_df.Proanthocyanins.quantile(0.75) - wine_df.Proanthocyanins.quantile(0.25)
Lower_fence = wine_df.Proanthocyanins.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Proanthocyanins.quantile(0.75) + (IQR * 3)
print('Proanthocyanins outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fenc
```

```
Proanthocyanins outliers are values < -0.8499999999999996 or > 4.05
```

```
IQR = wine_df.Color_intensity.quantile(0.75) - wine_df.Color_intensity.quantile(0.25)
Lower_fence = wine_df.Color_intensity.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Color_intensity.quantile(0.75) + (IQR * 3)
print('Color_intensity outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fenc
```

```
Color_intensity outliers are values < -5.7200000000000015 or > 15.14
```

```
IQR = wine_df.Hue.quantile(0.75) - wine_df.Hue.quantile(0.25)
Lower_fence = wine_df.Hue.quantile(0.25) - (IQR * 3)
Upper_fence = wine_df.Hue.quantile(0.75) + (IQR * 3)
print('Hue outliers are values < {lowerboundary} or > {upperboundary}'.format(lowerboundary=Lower_fence, upperboundary=Upper_fence))
```

```
Hue outliers are values < -0.23000000000000043 or > 2.1325000000000003
```

Declare feature vector and target variable

```
x = wine_df.drop(['class'], axis=1)
Y = wine_df['class']
```

Split data into separate training and test set

```
from sklearn.model_selection import train_test_split

x_train, x_test, Y_train, Y_test = train_test_split(x, Y, test_size = 0.2, random_state = 0)
```

```
x_train.shape, x_test.shape
```

```
((142, 13), (36, 13))
```

Engineering outliers in numerical variables

```
def max_value(df3, variable, top):
  return np.where(df3[variable]>top, top, df3[variable])

for df3 in [x_train, x_test]:
  df3['Malicacid'] = max_value(df3, 'Malicacid', 7.5)
  df3['Alcalinity_of_ash'] = max_value(df3, 'Alcalinity_of_ash', 34.4)
  df3['Magnesium'] = max_value(df3, 'Magnesium', 164)
  df3['Proanthocyanins'] = max_value(df3, 'Proanthocyanins', 4.05)
  df3['Color_intensity'] = max_value(df3, 'Color_intensity', 15.14)
  df3['Hue'] = max_value(df3, 'Hue', 2.13)
```

```
x_train.Malicacid.max(), x_test.Malicacid.max()
```

```
(5.65, 5.8)
```

```
x_train.Alcalinity_of_ash.max(), x_test.Alcalinity_of_ash.max()
```

```
(30.0, 28.5)
```

```
x_train.Magnesium.max(), x_test.Magnesium.max()
```

```
(162, 132)
```

```
x_train.Proanthocyanins.max(), x_test.Proanthocyanins.max()
```

```
(3.58, 2.45)
```

```
x_train.Color_intensity.max(), x_test.Color_intensity.max()
```

```
(13.0, 10.8)
```

```
x_train.Hue.max(), x_test.Hue.max()
```

```
(1.71, 1.38)
```

```
x_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000
mean	12.984859	2.372606	2.366901	19.554930	100.063380	2.258662	1.949155	0.363521	1.606056
std	0.807175	1.115360	0.269684	3.442549	14.249158	0.611691	0.975921	0.127709	0.591221
min	11.030000	0.740000	1.360000	10.600000	70.000000	1.100000	0.470000	0.130000	0.420000
25%	12.347500	1.602500	2.222500	17.250000	89.000000	1.705000	1.037500	0.270000	1.242500
50%	13.040000	1.895000	2.360000	19.500000	98.000000	2.210000	2.035000	0.340000	1.555000
75%	13.637500	3.222500	2.560000	21.500000	106.750000	2.735000	2.760000	0.450000	1.957500
max	14.750000	5.650000	3.220000	30.000000	162.000000	3.880000	3.740000	0.660000	3.580000

```
cols = x_train.columns
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

x_train = scaler.fit_transform(x_train)

x_test = scaler.transform(x_test)
```

```
x_train = pd.DataFrame(x_train, columns=[cols])
```

```
x_test = pd.DataFrame(x_test, columns=[cols])
```

```
x_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000
mean	0.525500	0.332506	0.541345	0.461594	0.326776	0.416785	0.452341	0.440606	0.375334
std	0.216983	0.227161	0.144991	0.177451	0.154882	0.220033	0.298447	0.240960	0.187095
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.354167	0.175662	0.463710	0.342784	0.206522	0.217626	0.173547	0.264151	0.260285
50%	0.540323	0.235234	0.537634	0.458763	0.304348	0.399281	0.478593	0.396226	0.359177
75%	0.700941	0.505601	0.645161	0.561856	0.399457	0.588129	0.700306	0.603774	0.486551
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Model Training

```
from sklearn.linear_model import LogisticRegression

logreg= LogisticRegression(solver='liblinear', random_state=0)

logreg.fit(x_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0, solver='liblinear')
```

Predict results

Predict type of classes

```
y_pred_test = logreg.predict(x_test)
```

```
y_pred_test
```

```
array([1, 3, 2, 1, 2, 2, 1, 3, 2, 2, 3, 3, 1, 2, 3, 2, 1, 1, 3, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1])
```

Prediction probability of getting 1

```
logreg.predict_proba(x_test)[:,0]
```

```
array([0.83411707, 0.08169456, 0.33794154, 0.79491944, 0.2372458 ,
       0.237406  , 0.87096795, 0.03861458, 0.16257044, 0.05946295,
       0.1482802 , 0.03716879, 0.92972498, 0.47744117, 0.0838979 ,
       0.13261824, 0.78069736, 0.95517064, 0.0825623 , 0.84310019,
       0.47020366, 0.67960296, 0.47193266, 0.24693731, 0.08366256,
       0.16146061, 0.21250699, 0.05442331, 0.07420845, 0.07434986,
       0.83480123, 0.85379414, 0.08617887, 0.8224449 , 0.87731904,
       0.6611851 ])
```

Prediction probability of getting 2

```
logreg.predict_proba(x_test)[:,1]
```

```
array([0.1240818 , 0.06966025, 0.65228402, 0.1607158 , 0.61192177,
       0.75455152, 0.08070117, 0.14839743, 0.77629276, 0.77373406,
       0.13986198, 0.07780088, 0.03752654, 0.51475621, 0.07579536,
       0.85273997, 0.16126423, 0.02246023, 0.44592902, 0.13936754,
       0.52143812, 0.24521823, 0.43573301, 0.71879674, 0.58483752,
       0.76777135, 0.73705714, 0.8411666 , 0.73358901, 0.06019109,
       0.12898216, 0.11348826, 0.61642905, 0.06319566, 0.08703462,
       0.31648131])
```

Check accuracy score

```
from sklearn.metrics import accuracy_score
```

```
print('Model accuracy score: {0:0.4f}'.format(accuracy_score(Y_test, y_pred_test)))
```

```
Model accuracy score: 0.9722
```

```
y_pred_train = logreg.predict(x_train)
```

```
y_pred_train
```

```
array([3, 2, 2, 3, 1, 1, 2, 2, 2, 1, 3, 2, 3, 1, 3, 3, 1, 3, 1, 2, 3, 3,
       2, 3, 3, 1, 3, 3, 2, 3, 3, 2, 1, 2, 2, 2, 1, 1, 2, 2, 3, 3, 2, 2,
       2, 3, 3, 1, 3, 2, 2, 2, 2, 2, 1, 1, 2, 1, 3, 1, 3, 1, 1, 2, 1, 2,
       2, 1, 3, 2, 1, 2, 2, 2, 3, 1, 3, 3, 1, 1, 2, 3, 1, 1, 2, 2, 1, 1,
       1, 3, 2, 1, 2, 3, 1, 2, 3, 3, 1, 1, 3, 1, 3, 2, 1, 1, 2, 1, 3, 2,
       3, 1, 3, 3, 3, 1, 2, 2, 2, 2, 3, 3, 2, 2, 2, 2, 3, 3, 1, 1, 3, 2,
       2, 2, 1, 1, 1, 2, 2, 2, 1, 3])
```

```
print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(Y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9789
```

Check for overfitting and underfitting

```
print('Training set score: {:.4f}'.format(logreg.score(x_train, Y_train)))

print('Test set score: {:.4f}'.format(logreg.score(x_test, Y_test)))
```

```
Training set score: 0.9789
Test set score: 0.9722
```

```
logreg100 = LogisticRegression(C=100, solver='liblinear', random_state=0)

logreg100.fit(x_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=100, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg100.score(x_train, Y_train)))

print('Test set score: {:.4f}'.format(logreg100.score(x_test, Y_test)))
```

```
Training set score: 1.0000
Test set score: 1.0000
```

```
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)

logreg001.fit(x_train, Y_train)
```

```
▾ LogisticRegression
LogisticRegression(C=0.01, random_state=0, solver='liblinear')
```

```
print('Training set score: {:.4f}'.format(logreg001.score(x_train, Y_train)))

print('Test set score: {:.4f}'.format(logreg001.score(x_test, Y_test)))
```

```
Training set score: 0.8944
Test set score: 0.8889
```

Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(Y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        14
           2       1.00      0.94      0.97        16
           3       0.86      1.00      0.92         6

    accuracy                           0.97        36
   macro avg       0.95      0.98      0.96        36
weighted avg       0.98      0.97      0.97        36
```

Classification accuracy

```
classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 1.0000
```

Precision

```
precision =  TP / float(TP+FP)

print('Precision: {0:0.4f}'.format(precision))
```

```
Precision: 1.0000
```

Recall

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 1.0000
```

True Positive Rate

```
true_positive_rate = TP / float(TP + FN)
print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 1.0000
```

Specificity

```
specificity = TN / (TN+FP)
print('Specifity : {0:0.4f}'.format(specificity))
```

```
Specifity : 1.0000
```

Adjusting the threshold level

```
y_pred_prob = logreg.predict_proba(x_test)[0:10]

y_pred_prob
```

```
array([[0.83411707, 0.1240818 , 0.04180113],
       [0.08169456, 0.06966025, 0.84864519],
       [0.33794154, 0.65228402, 0.00977444],
       [0.79491944, 0.1607158 , 0.04436476],
       [0.2372458 , 0.61192177, 0.15083243],
       [0.237406  , 0.75455152, 0.00804248],
       [0.87096795, 0.08070117, 0.04833088],
       [0.03861458, 0.14839743, 0.81298799],
       [0.16257044, 0.77629276, 0.0611368 ],
       [0.05946295, 0.77373406, 0.16680299]])
```

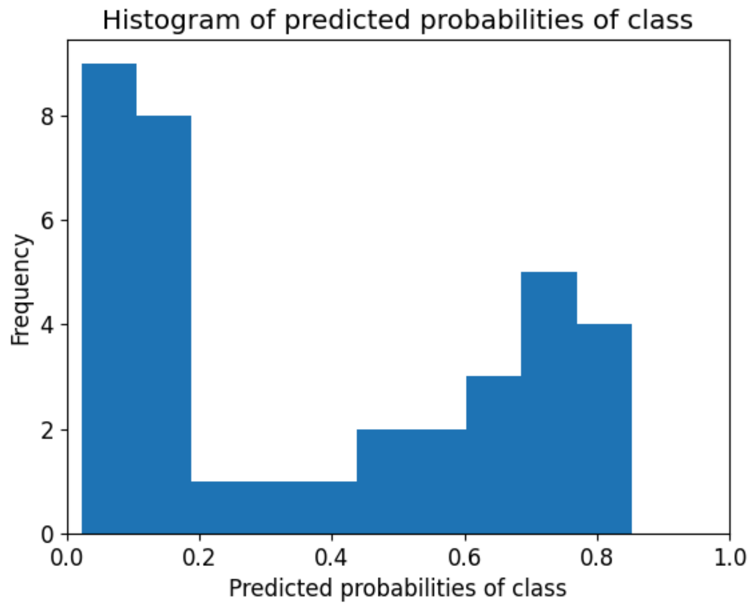
```
logreg.predict_proba(x_test)[0:10, 1]
```

```
array([0.1240818 , 0.06966025, 0.65228402, 0.1607158 , 0.61192177,
       0.75455152, 0.08070117, 0.14839743, 0.77629276, 0.77373406])
```

```
y_pred1 = logreg.predict_proba(x_test)[:, 1]
```

```
plt.rcParams['font.size'] = 12

plt.hist(y_pred1, bins = 10)

plt.title('Histogram of predicted probabilities of class')

plt.xlim(0,1)

plt.xlabel('Predicted probabilities of class')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



k-Fold Cross Validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(logreg, x_train, Y_train, cv=5, scoring = 'accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.93103448 1.         0.96428571 1.         0.92857143]
```

```
print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.9648
```

Hyperparameter Optimization using GridSearch CV

```
from sklearn.model_selection import GridSearchCV

parameters = [{'penalty':[']1',']2']},
          {'C':[1, 10, 100, 1000]}]

grid_search = GridSearchCV(estimator = logreg,
               param_grid = parameters,
               scoring = 'accuracy',
               cv = 5,
               verbose=0)

grid_search.fit(x_train, Y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
10 fits failed out of a total of 30.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1160, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter of LogisticRegression must be a str among {'none' (deprec

--------------------------------------------------------------------------------
5 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1160, in fit
    self._validate_params()
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'penalty' parameter of LogisticRegression must be a str among {'none' (deprec

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-fini
  warnings.warn(
```

▸ **GridSearchCV**

▸ **estimator: LogisticRegression**

▸ LogisticRegression

```
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

print('Parameters that give the best results :','\n\n', (grid_search.best_params_))

print('\n\nEstimator that was chosen by the search :', '\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9650


Parameters that give the best results :

 {'C': 10}


Estimator that was chosen by the search :

 LogisticRegression(C=10, random_state=0, solver='liblinear')
```

```
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(x_test, Y_test)))
```

```
GridSearch CV score on test set: 1.0000
```

Results and Conclusion

The selected dependent variable which is "horsepower" could be predicted successfully using all the other independent variables in linear regression. On the other hand, the model accuracy score after using logistic regression is 97.22% which yields that the classes of the columns are predicted accurately. The classification accuracy, precision, recall, true positive rate, and specificity are all 100% which means the model achieved perfect performance on the dataset. The GridSearch CV best score also yields 96.50%. The model has learned to perfectly discriminate between the classes in the dataset without making any errors.