Project4

March 19, 2024

1 ECE 219 Project 4

Group Members: Zan Xie (UID: 205364923), Joseph Gong (UID: 606073799), Anuk Fernando (UID: 805423707)

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
pip install pydantic-settings
```

```
Collecting pydantic-settings
  Downloading pydantic_settings-2.2.1-py3-none-any.whl (13 kB)
Requirement already satisfied: pydantic>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic-settings) (2.6.4)
Collecting python-dotenv>=0.21.0 (from pydantic-settings)
  Downloading python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.3.0->pydantic-
settings) (0.6.0)
Requirement already satisfied: pydantic-core==2.16.3 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.3.0->pydantic-
settings) (2.16.3)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2.3.0->pydantic-
settings) (4.10.0)
Installing collected packages: python-dotenv, pydantic-settings
Successfully installed pydantic-settings-2.2.1 python-dotenv-1.0.1
```

[ ]:
```
pip install pydantic==2.3.0
```

```
Collecting pydantic==2.3.0
  Downloading pydantic-2.3.0-py3-none-any.whl (374 kB)
                           374.5/374.5

kB 7.9 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic==2.3.0) (0.6.0)
```

```
Collecting pydantic-core==2.6.3 (from pydantic==2.3.0)
  Downloading
pydantic_core-2.6.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(1.9 MB)
                           1.9/1.9 MB
17.3 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic==2.3.0) (4.10.0)
Installing collected packages: pydantic-core, pydantic
  Attempting uninstall: pydantic-core
    Found existing installation: pydantic_core 2.16.3
    Uninstalling pydantic_core-2.16.3:
      Successfully uninstalled pydantic_core-2.16.3
  Attempting uninstall: pydantic
    Found existing installation: pydantic 2.6.4
    Uninstalling pydantic-2.6.4:
      Successfully uninstalled pydantic-2.6.4
Successfully installed pydantic-2.3.0 pydantic-core-2.6.3
```

[ ]:
```
pip install pandas_profiling
```

```
Collecting pandas_profiling
  Downloading pandas_profiling-3.6.6-py2.py3-none-any.whl (324 kB)
                           324.4/324.4

kB 5.2 MB/s eta 0:00:00
Collecting ydata-profiling (from pandas_profiling)
  Downloading ydata_profiling-4.6.5-py2.py3-none-any.whl (357 kB)
                           357.9/357.9

kB 11.8 MB/s eta 0:00:00
Requirement already satisfied: scipy<1.12,>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(1.11.4)
Requirement already satisfied: pandas!=1.4.0,<3,>1.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(1.5.3)
Requirement already satisfied: matplotlib<3.9,>=3.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(3.7.1)
Requirement already satisfied: pydantic>=2 in /usr/local/lib/python3.10/dist-
packages (from ydata-profiling->pandas_profiling) (2.3.0)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(6.0.1)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(3.1.3)
Collecting visions[type_image_path]==0.7.5 (from ydata-
```

```
profiling->pandas_profiling)
  Downloading visions-0.7.5-py3-none-any.whl (102 kB)
                           102.7/102.7

kB 11.9 MB/s eta 0:00:00
Requirement already satisfied: numpy<1.26,>=1.16.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(1.25.2)
Collecting htmlmin==0.1.12 (from ydata-profiling->pandas_profiling)
  Downloading htmlmin-0.1.12.tar.gz (19 kB)
  Preparing metadata (setup.py) … done
Collecting phik<0.13,>=0.11.1 (from ydata-profiling->pandas_profiling)
  Downloading
phik-0.12.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (686 kB)
                           686.1/686.1

kB 15.9 MB/s eta 0:00:00
Requirement already satisfied: requests<3,>=2.24.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(2.31.0)
Requirement already satisfied: tqdm<5,>=4.48.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(4.66.2)
Collecting seaborn<0.13,>=0.10.1 (from ydata-profiling->pandas_profiling)
  Downloading seaborn-0.12.2-py3-none-any.whl (293 kB)
                           293.3/293.3

kB 15.2 MB/s eta 0:00:00
Collecting multimethod<2,>=1.4 (from ydata-profiling->pandas_profiling)
  Downloading multimethod-1.11.2-py3-none-any.whl (10 kB)
Requirement already satisfied: statsmodels<1,>=0.13.2 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(0.14.1)
Collecting typeguard<5,>=4.1.2 (from ydata-profiling->pandas_profiling)
  Downloading typeguard-4.1.5-py3-none-any.whl (34 kB)
Collecting imagehash==4.3.1 (from ydata-profiling->pandas_profiling)
  Downloading ImageHash-4.3.1-py2.py3-none-any.whl (296 kB)
                           296.5/296.5

kB 24.6 MB/s eta 0:00:00
Requirement already satisfied: wordcloud>=1.9.1 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(1.9.3)
Collecting dacite>=1.8 (from ydata-profiling->pandas_profiling)
  Downloading dacite-1.8.1-py3-none-any.whl (14 kB)
Requirement already satisfied: numba<0.59.0,>=0.56.0 in
/usr/local/lib/python3.10/dist-packages (from ydata-profiling->pandas_profiling)
(0.58.1)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.10/dist-
```

```
packages (from imagehash==4.3.1->ydata-profiling->pandas_profiling) (1.5.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages
(from imagehash==4.3.1->ydata-profiling->pandas_profiling) (9.4.0)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.10/dist-
packages (from visions[type_image_path]==0.7.5->ydata-
profiling->pandas_profiling) (23.2.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-
packages (from visions[type_image_path]==0.7.5->ydata-
profiling->pandas_profiling) (3.2.1)
Collecting tangled-up-in-unicode>=0.0.4 (from
visions[type_image_path]==0.7.5->ydata-profiling->pandas_profiling)
  Downloading tangled_up_in_unicode-0.2.0-py3-none-any.whl (4.7 MB)
                           4.7/4.7 MB
25.6 MB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2<3.2,>=2.11.1->ydata-
profiling->pandas_profiling) (2.1.5)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib<3.9,>=3.2->ydata-profiling->pandas_profiling) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib<3.9,>=3.2->ydata-
profiling->pandas_profiling) (2.8.2)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba<0.59.0,>=0.56.0->ydata-
profiling->pandas_profiling) (0.41.1)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=1.4.0,<3,>1.1->ydata-profiling->pandas_profiling)
(2023.4)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.10/dist-
packages (from phik<0.13,>=0.11.1->ydata-profiling->pandas_profiling) (1.3.2)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas_profiling) (0.6.0)
```

```
Requirement already satisfied: pydantic-core==2.6.3 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas_profiling) (2.6.3)
Requirement already satisfied: typing-extensions>=4.6.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=2->ydata-
profiling->pandas_profiling) (4.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas_profiling) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.24.0->ydata-profiling->pandas_profiling) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas_profiling) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.24.0->ydata-
profiling->pandas_profiling) (2024.2.2)
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-
packages (from statsmodels<1,>=0.13.2->ydata-profiling->pandas_profiling)
(0.5.6)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.4->statsmodels<1,>=0.13.2->ydata-profiling->pandas_profiling)
(1.16.0)
Building wheels for collected packages: htmlmin
  Building wheel for htmlmin (setup.py) … done
  Created wheel for htmlmin: filename=htmlmin-0.1.12-py3-none-any.whl size=27080
sha256=1afe88acb1deabe7467af2b0c1aacd8f22acc7a9ce618d14124a3bfd1a2745b4
  Stored in directory: /root/.cache/pip/wheels/dd/91/29/a79cecb328d01739e64017b6
fb9a1ab9d8cb1853098ec5966d
Successfully built htmlmin
Installing collected packages: htmlmin, typeguard, tangled-up-in-unicode,
multimethod, dacite, imagehash, visions, seaborn, phik, ydata-profiling,
pandas_profiling
  Attempting uninstall: seaborn
    Found existing installation: seaborn 0.13.1
    Uninstalling seaborn-0.13.1:
      Successfully uninstalled seaborn-0.13.1
Successfully installed dacite-1.8.1 htmlmin-0.1.12 imagehash-4.3.1
multimethod-1.11.2 pandas_profiling-3.6.6 phik-0.12.4 seaborn-0.12.2 tangled-up-
in-unicode-0.2.0 typeguard-4.1.5 visions-0.7.5 ydata-profiling-4.6.5
```

[ ]:
```
!pip install pycountry_convert
```

```
Collecting pycountry_convert
  Downloading pycountry_convert-0.7.2-py3-none-any.whl (13 kB)
Collecting pprintpp>=0.3.0 (from pycountry_convert)
  Downloading pprintpp-0.4.0-py2.py3-none-any.whl (16 kB)
Collecting pycountry>=16.11.27.1 (from pycountry_convert)
```

```
  Downloading pycountry-23.12.11-py3-none-any.whl (6.2 MB)
                              6.2/6.2 MB
15.4 MB/s eta 0:00:00
Requirement already satisfied: pytest>=3.4.0 in
/usr/local/lib/python3.10/dist-packages (from pycountry_convert) (7.4.4)
Collecting pytest-mock>=1.6.3 (from pycountry_convert)
  Downloading pytest_mock-3.12.0-py3-none-any.whl (9.8 kB)
Collecting pytest-cov>=2.5.1 (from pycountry_convert)
  Downloading pytest_cov-4.1.0-py3-none-any.whl (21 kB)
Collecting repoze.lru>=0.7 (from pycountry_convert)
  Downloading repoze.lru-0.7-py3-none-any.whl (10 kB)
Requirement already satisfied: wheel>=0.30.0 in /usr/local/lib/python3.10/dist-
packages (from pycountry_convert) (0.43.0)
Requirement already satisfied: iniconfig in /usr/local/lib/python3.10/dist-
packages (from pytest>=3.4.0->pycountry_convert) (2.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from pytest>=3.4.0->pycountry_convert) (24.0)
Requirement already satisfied: pluggy<2.0,>=0.12 in
/usr/local/lib/python3.10/dist-packages (from pytest>=3.4.0->pycountry_convert)
(1.4.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in
/usr/local/lib/python3.10/dist-packages (from pytest>=3.4.0->pycountry_convert)
(1.2.0)
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from pytest>=3.4.0->pycountry_convert) (2.0.1)
Collecting coverage[toml]>=5.2.1 (from pytest-cov>=2.5.1->pycountry_convert)
  Downloading coverage-7.4.4-cp310-cp310-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (233 kB)
                              233.5/233.5

kB 10.9 MB/s eta 0:00:00
Installing collected packages: repoze.lru, pprintpp, pycountry, coverage,
pytest-mock, pytest-cov, pycountry_convert
Successfully installed coverage-7.4.4 pprintpp-0.4.0 pycountry-23.12.11
pycountry_convert-0.7.2 pytest-cov-4.1.0 pytest-mock-3.12.0 repoze.lru-0.7
```

[ ]:
```python
import pandas as pd
import warnings
from matplotlib import pyplot as plt

from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, mutual_info_regression,␣
 ↪f_regression
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neural_network import MLPRegressor
```

```python
from statsmodels.regression.linear_model import OLS
from sklearn.ensemble import RandomForestRegressor
import itertools
import pandas as pd
import numpy as np

import seaborn
import pycountry_convert as pc

from sklearn.tree import export_graphviz
import pydot
from statsmodels.api import add_constant
import itertools
```

2 Q1.1

[ ]:
```python
diamonds = pd.read_csv('/content/drive/Shareddrives/ECE219/Project4/
↪diamonds_ece219.csv')
```

[ ]:
```python
diamonds = diamonds.drop(columns=['Unnamed: 0'])
```

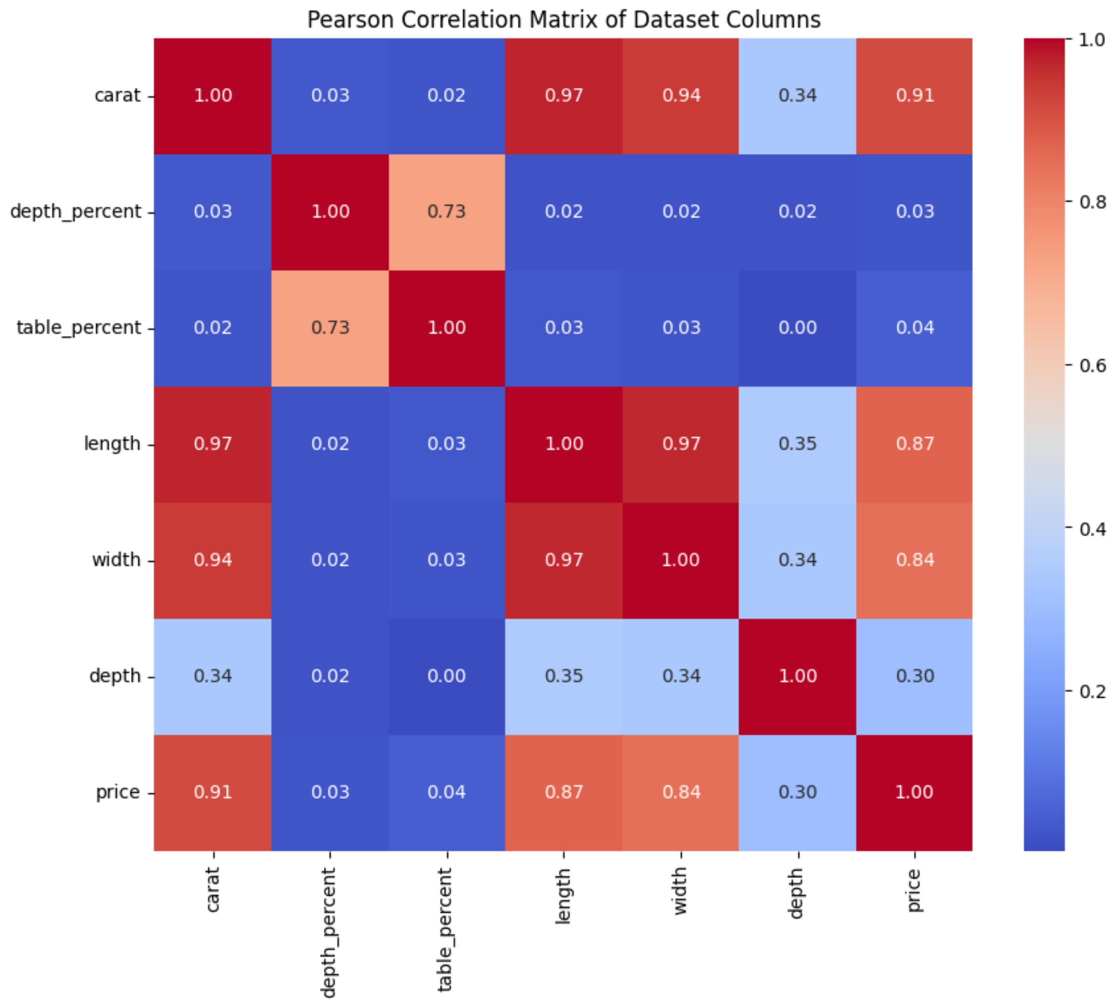
[ ]:
```python
diamonds.head()
```

[ ]:
```
  color clarity  carat        cut   symmetry     polish  depth_percent  \
0     E    VVS2   0.09  Excellent  Very Good  Very Good           62.7
1     E    VVS2   0.09  Very Good  Very Good  Very Good           61.9
2     E    VVS2   0.09  Excellent  Very Good  Very Good           61.1
3     E    VVS2   0.09  Excellent  Very Good  Very Good           62.0
4     E    VVS2   0.09  Very Good  Very Good  Excellent           64.9

   table_percent  length  width  depth girdle_min girdle_max  price
0           59.0    2.85   2.87   1.79          M          M    200
1           59.0    2.84   2.89   1.78        STK        STK    200
2           59.0    2.88   2.90   1.77         TN          M    200
3           59.0    2.86   2.88   1.78          M        STK    200
4           58.5    2.79   2.83   1.82        STK        STK    200
```

[ ]:
```python
import seaborn as sns
corr_matrix = diamonds.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Pearson Correlation Matrix of Dataset Columns')
plt.show()
```

```python
target_variable = 'price'  # Assuming 'price' is the target variable
correlation_with_target = corr_matrix[target_variable].drop(target_variable).
 ↪abs().sort_values(ascending=False)

correlation_with_target
```

```
<ipython-input-12-69067177c0d2>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it will
default to False. Select only valid columns or specify the value of numeric_only
to silence this warning.
  corr_matrix = diamonds.corr()
```




```
[ ]: carat            0.913479
     length           0.869521
     width            0.841887
     depth            0.299696
```

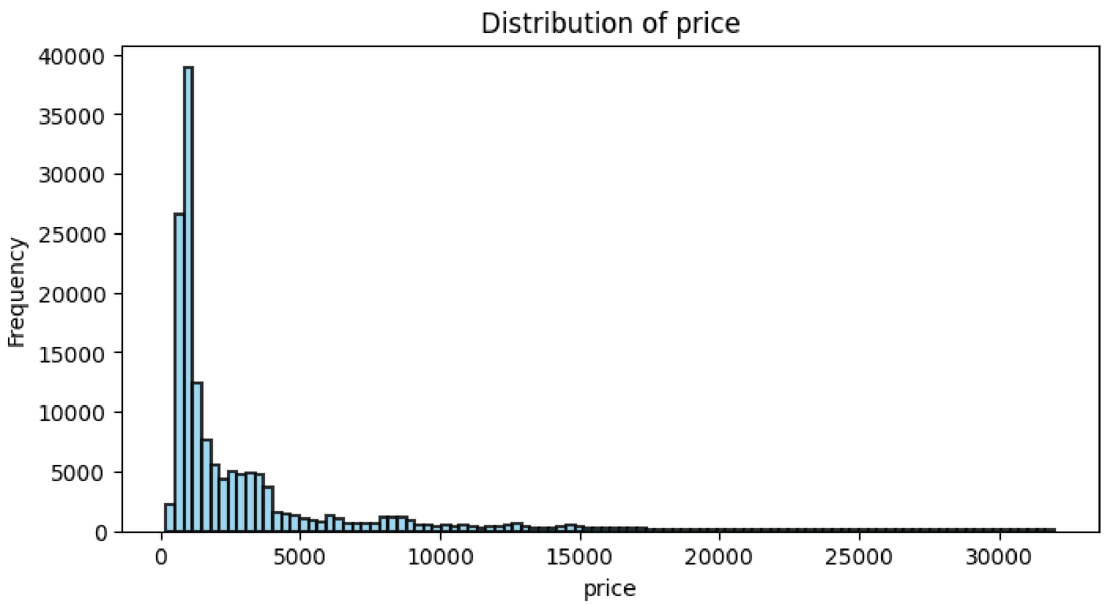
```
table_percent    0.042453
depth_percent    0.025469
Name: price, dtype: float64
```

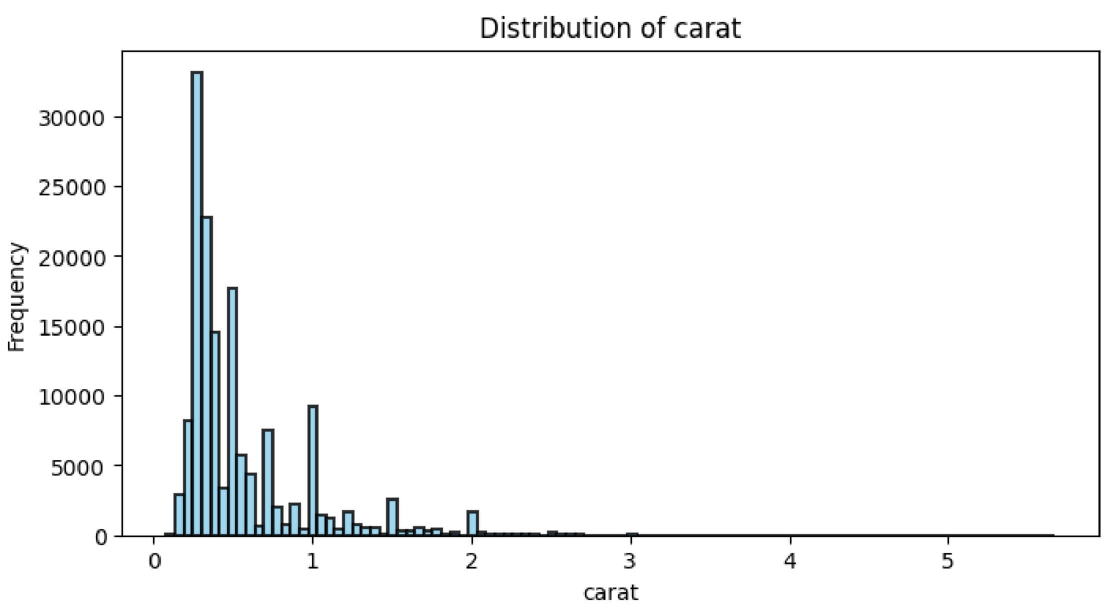
1. Carat with a correlation of 0.913479: This indicates that as the carat size increases, the price of the diamond tends to increase as well.
2. Length with a correlation of 0.869521: Similarly,this indicates that larger diamonds (length) are generally more expensive.
3. Width with a correlation of 0.841887: This also shows a strong positive correlation, supporting the idea that larger diamonds (width) have higher prices.
4. Depth, depth_percent, and table_pecent with a correlation which way much lower compare to the pervious three factors.

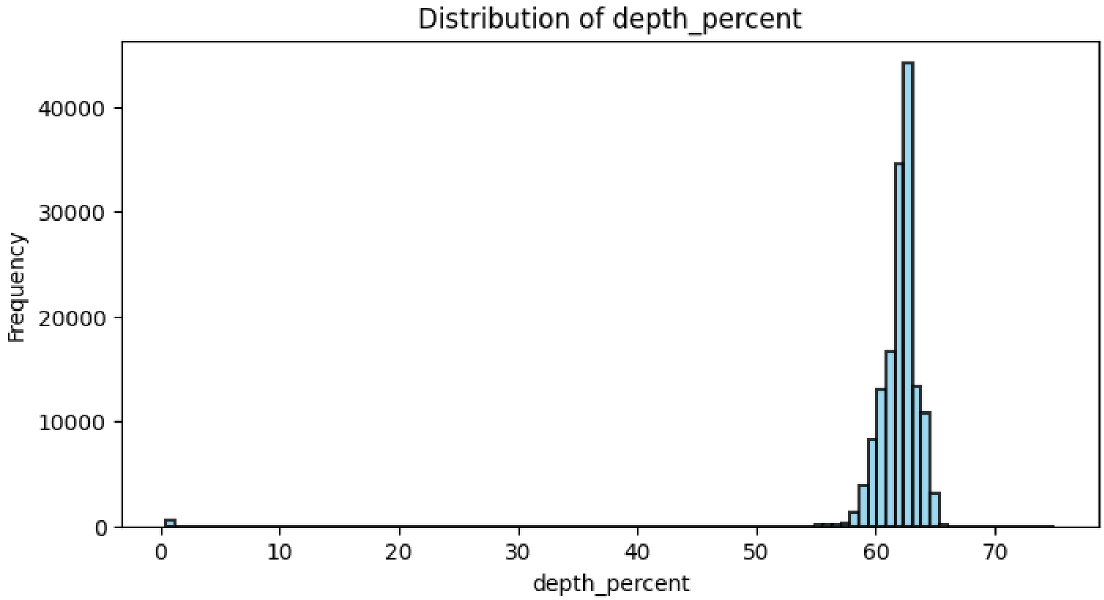
3 Q1.2

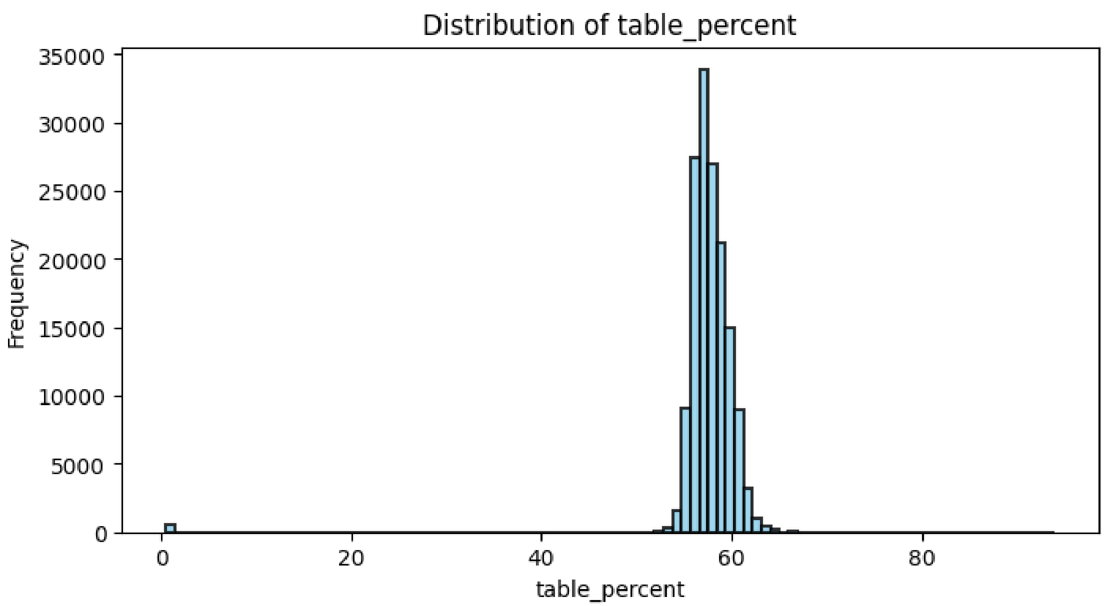
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

print(diamonds.columns)
numerical_features =
 ↪['price','carat','depth_percent','table_percent','length','width','depth']


for feature in numerical_features:
    plt.figure(figsize=(8, 4))
    plt.hist(diamonds[feature], bins=100, edgecolor='k', color='skyblue',
 ↪linewidth=1.5, alpha=0.8)
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {feature}')
    plt.show()
```

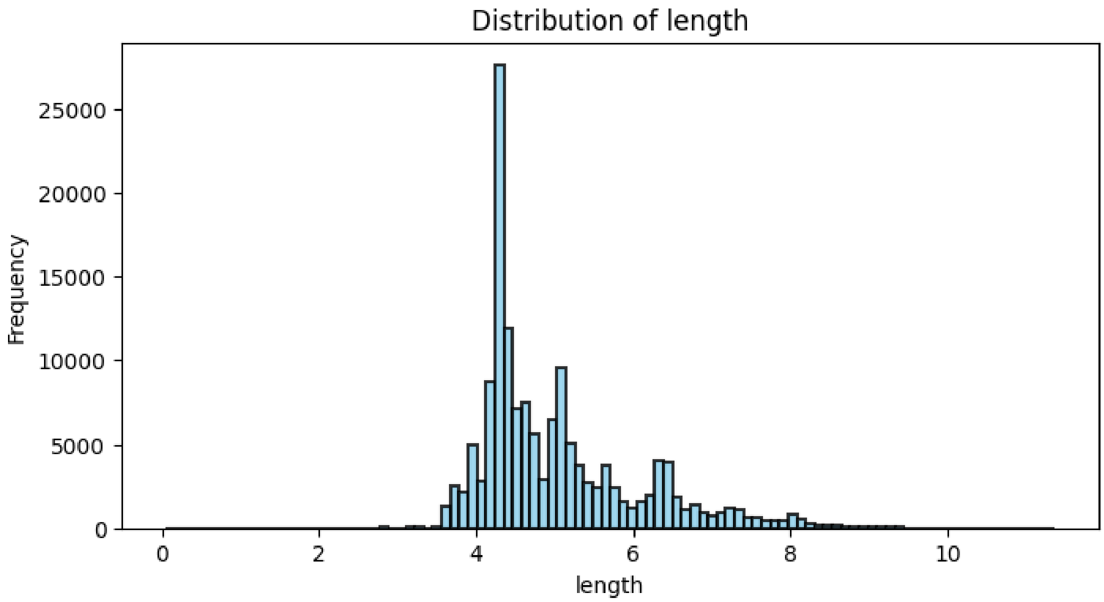
```
Index(['color', 'clarity', 'carat', 'cut', 'symmetry', 'polish',
       'depth_percent', 'table_percent', 'length', 'width', 'depth',
       'girdle_min', 'girdle_max', 'price'],
      dtype='object')
```

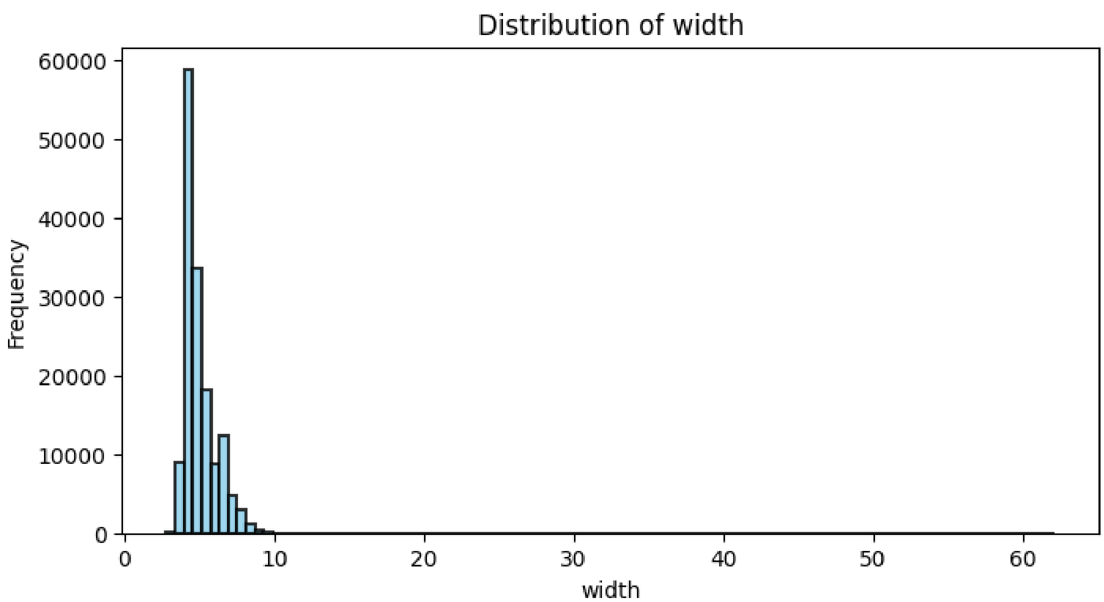

Distribution of price
Frequency
price
40000
35000
30000
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000
25000
30000

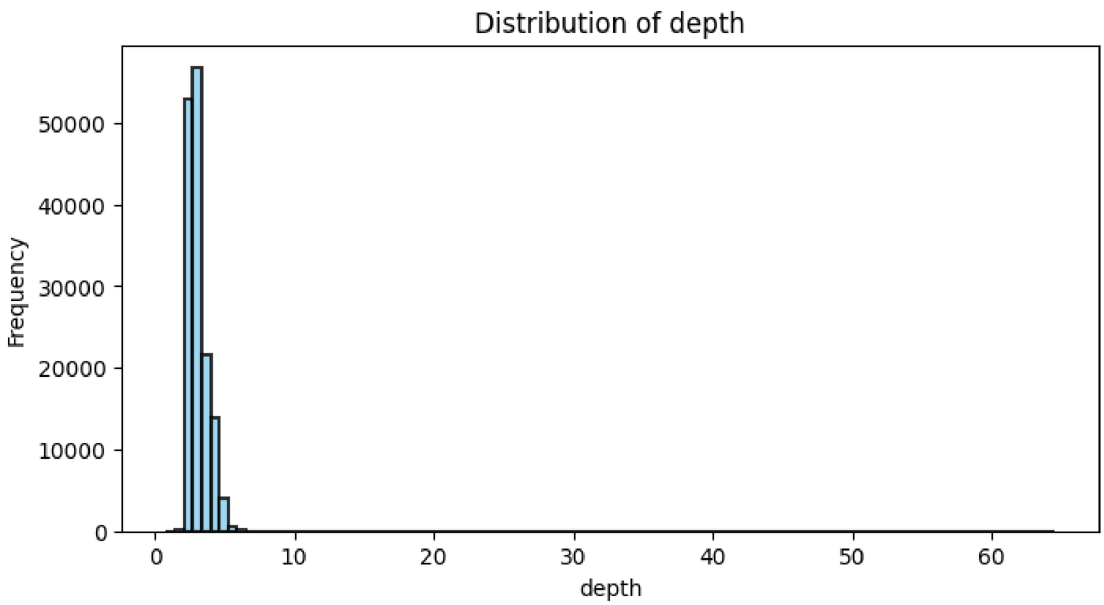

Distribution of carat
Frequency
carat
30000
25000
20000
15000
10000
5000
0
0
1
2
3
4
5


Distribution of depth_percent
Frequency
depth_percent
0
10000
20000
30000
40000
0
10
20
30
40
50
60
70


Distribution of table_percent
Frequency
table_percent
0
5000
10000
15000
20000
25000
30000
35000
0
20
40
60
80

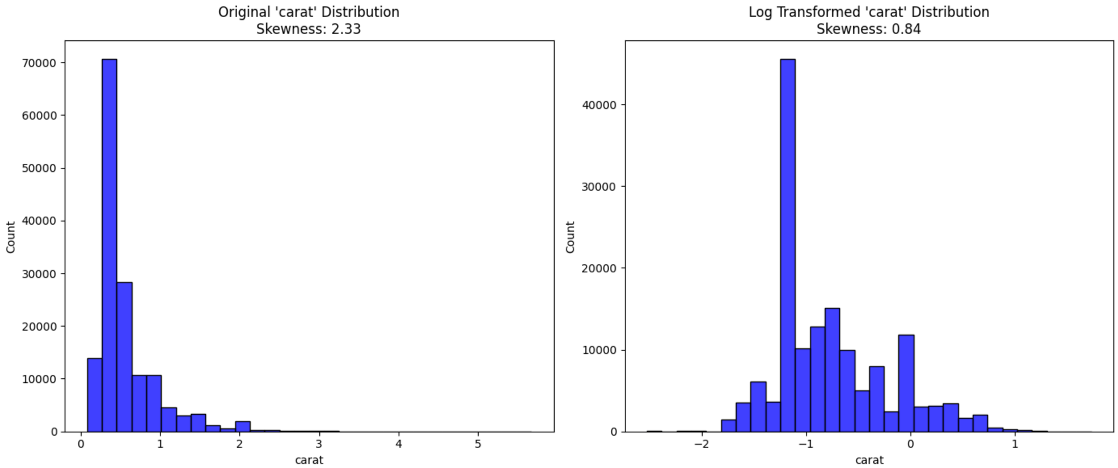

Distribution of length
Frequency
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
length


Distribution of width
Frequency
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
width



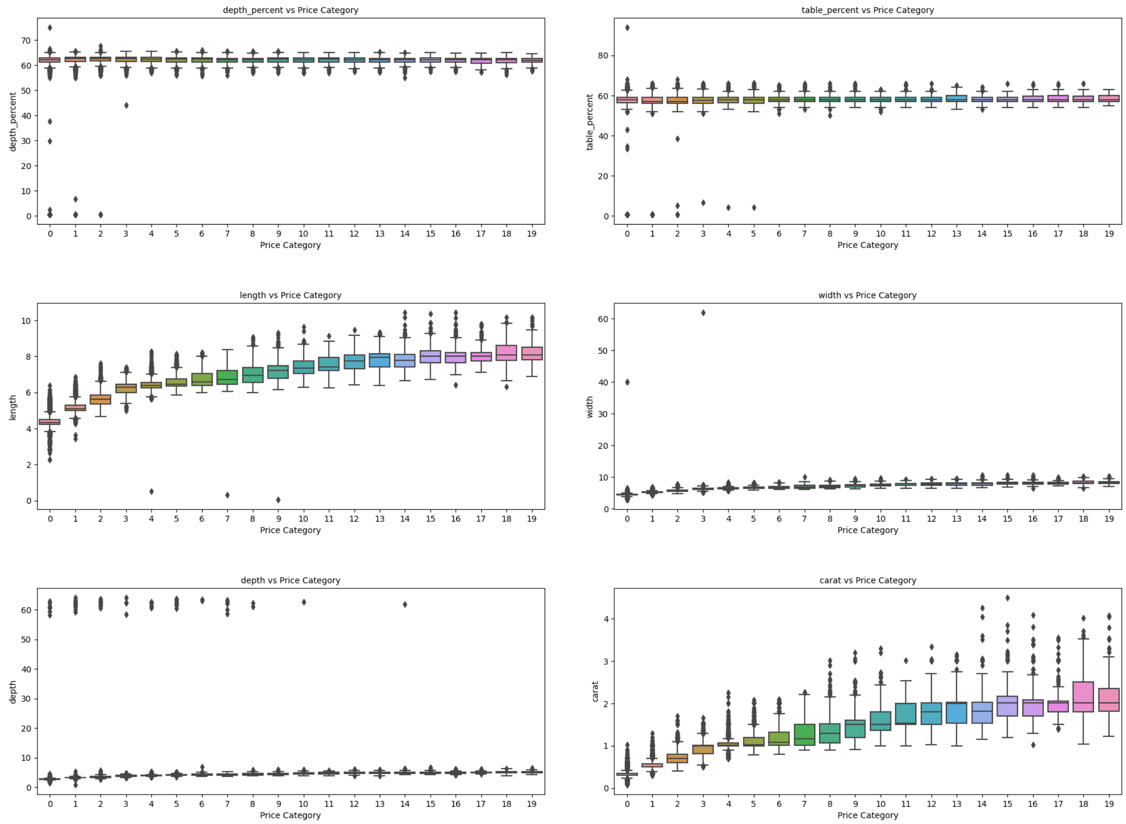
Log Transformation, Square Root Transformation can be done if the distribution of a feature has high skewness. The plots Below show the original and log-transformed distributions of the 'carat' feature.

[ ]:
```
original_skewness = diamonds['carat'].skew()

carat_log_transformed = np.log(diamonds['carat'])

transformed_skewness = carat_log_transformed.skew()

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

sns.histplot(diamonds['carat'], bins=30, ax=ax[0], color='blue',␣
 ↪edgecolor='black')
ax[0].set_title(f"Original 'carat' Distribution\nSkewness: {original_skewness:.
 ↪2f}")

sns.histplot(carat_log_transformed, bins=30, ax=ax[1], color='blue',␣
 ↪edgecolor='black')
ax[1].set_title(f"Log Transformed 'carat' Distribution\nSkewness:␣
 ↪{transformed_skewness:.2f}")

plt.tight_layout()
plt.show()
```



4 Q1.3

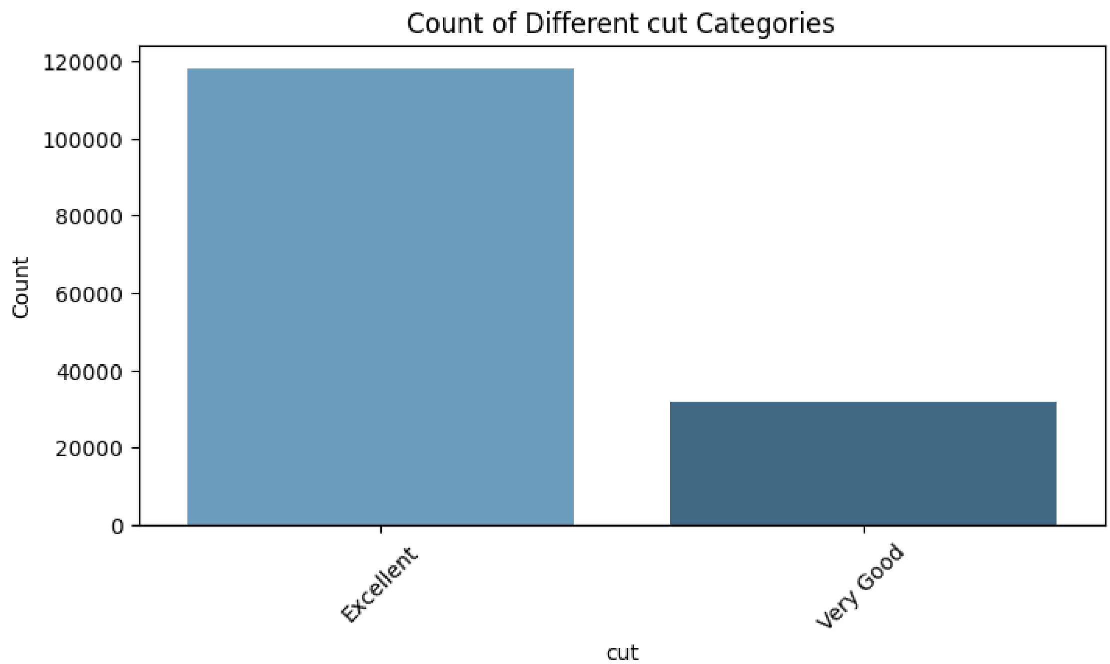
[ ]:
```
diamonds_sample = diamonds.sample(frac=0.5, random_state=1)

diamonds_sample['price_category'] = pd.cut(diamonds_sample['price'], bins=20,
  labels=range(20))

selected_features = ['depth_percent', 'table_percent', 'length', 'width',
  'depth','carat']

plt.figure(figsize=(20, 15))
for index, feature in enumerate(selected_features, 1):
    plt.subplot(3, 2, index)
    sns.boxplot(x='price_category', y=feature, data=diamonds_sample)
    plt.title(f'{feature} vs Price Category', fontsize=10)
    plt.xlabel('Price Category', fontsize=10)
    plt.ylabel(feature, fontsize=10)

plt.tight_layout(pad=5.0)
plt.show()
```



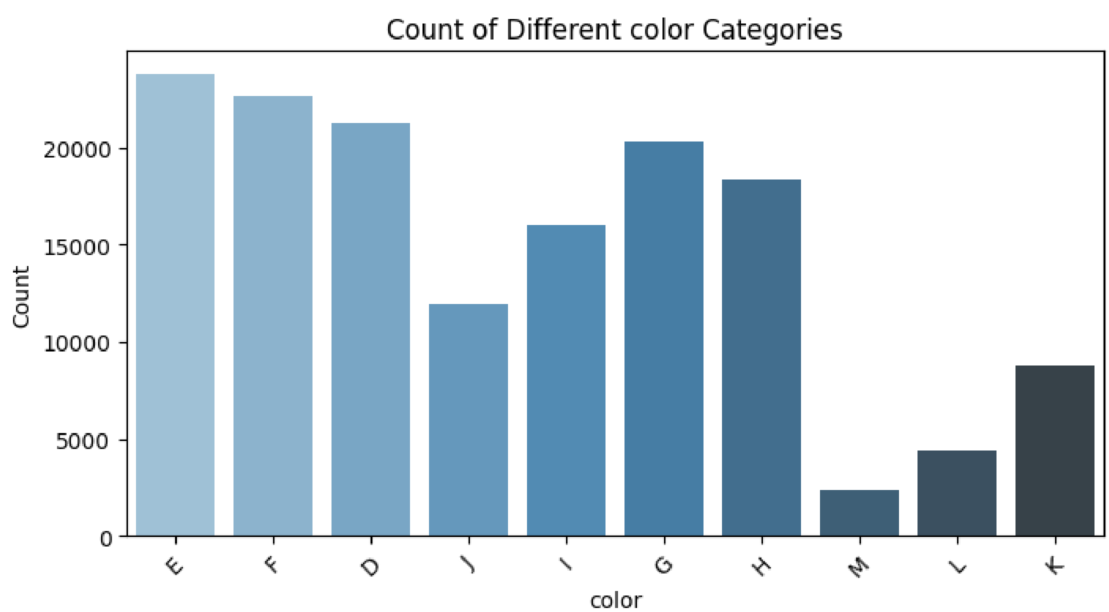
'length', 'width', and 'depth' might generally increase with higher price categories, reflecting that larger diamonds tend to be more expensive. diamond's 'carat' significantly affects its price

5 Q1.4

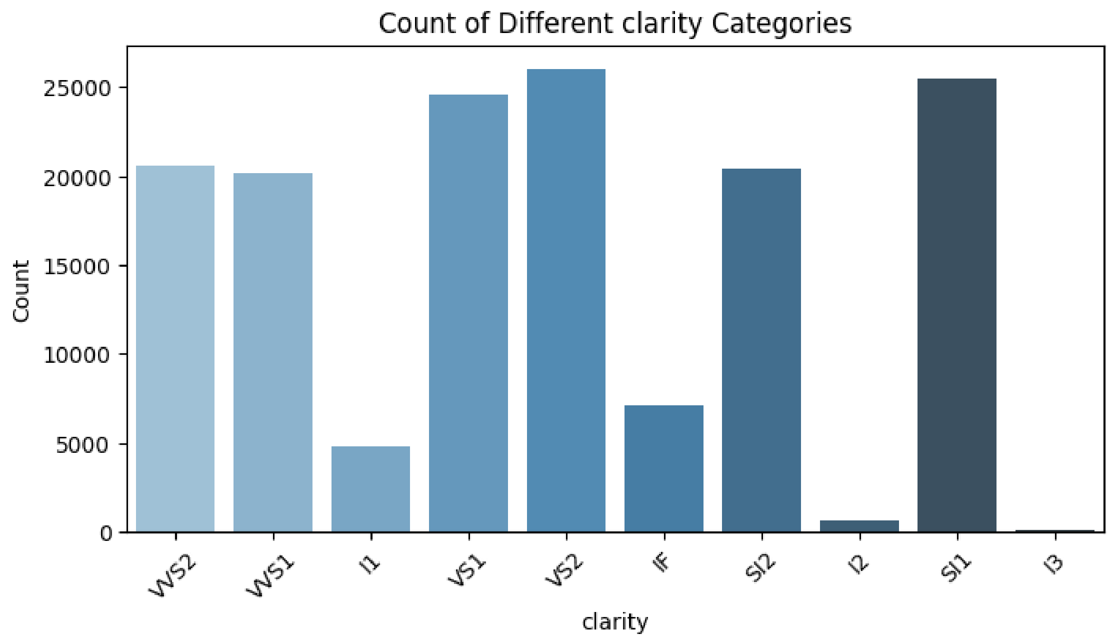
[ ]:
```
categorical_features = ['cut', 'color', 'clarity']
for feature in categorical_features:
    plt.figure(figsize=(8, 4))
    sns.countplot(data=diamonds, x=feature, palette='Blues_d')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.title(f'Count of Different {feature} Categories')
    plt.xticks(rotation=45)  # Rotate labels to prevent overlap
    plt.show()
```



Count of Different cut Categories
120000
100000
80000
60000
40000
20000
0
Count
Excellent
Very Good
cut



Count of Different color Categories
20000
15000
10000
5000
0
Count
E
F
D
J
I
G
H
M
L
K
color



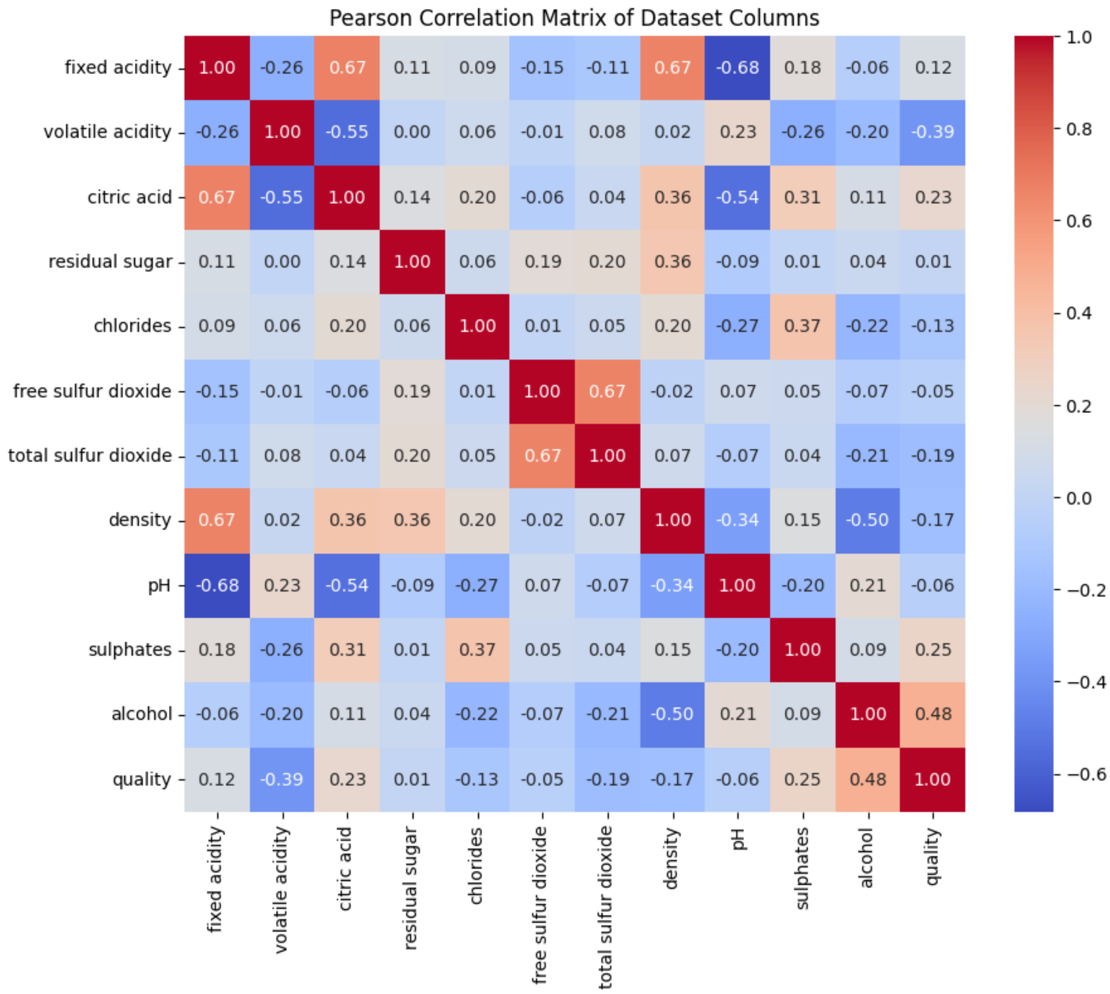
[ ]:
```
redwine = '/content/drive/Shareddrives/ECE219/Project4/winequality/
↪winequality-red.csv'  # Example path to the red wine data
whitewine = '/content/drive/Shareddrives/ECE219/Project4/winequality/
↪winequality-white.csv'  # Example path to the white wine data
```

[ ]:
```
dataset1 = pd.read_csv(redwine, delimiter=';')
dataset1.head()
```

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
```

```
3      9.8        6
4      9.4        5
```

[ ]:
```python
import seaborn as sns
corr_matrix = dataset1.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Pearson Correlation Matrix of Dataset Columns')
plt.show()

# Identify features with the highest absolute correlation with the target␣
↪variable
target_variable = 'quality'  # Assuming 'quality' is the target variable
correlation_with_target = corr_matrix[target_variable].drop(target_variable).
↪abs().sort_values(ascending=False)

correlation_with_target
```

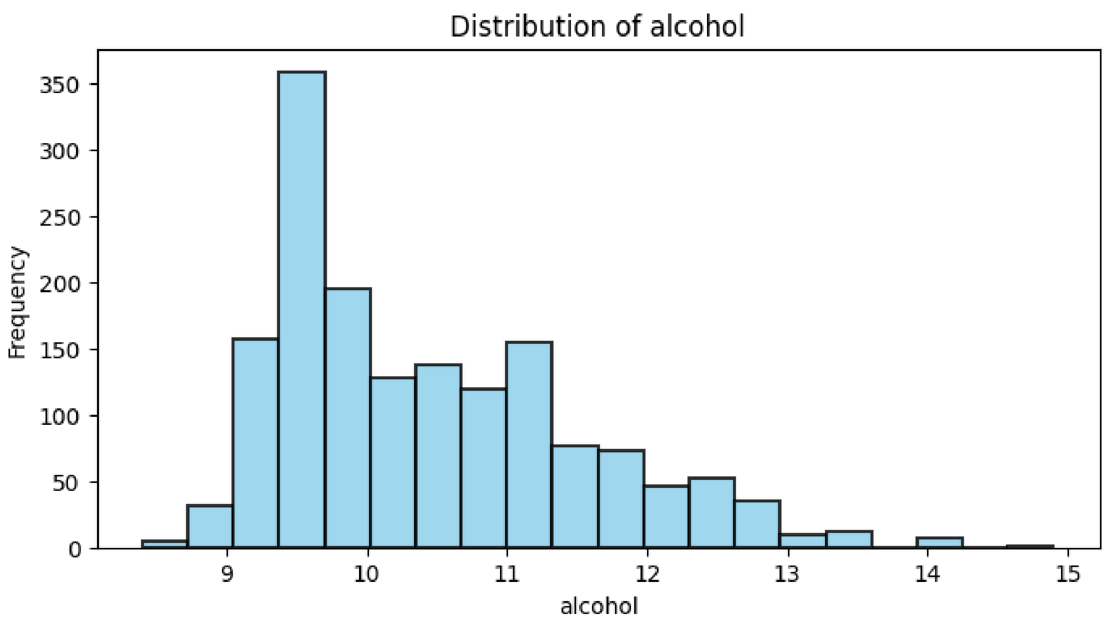


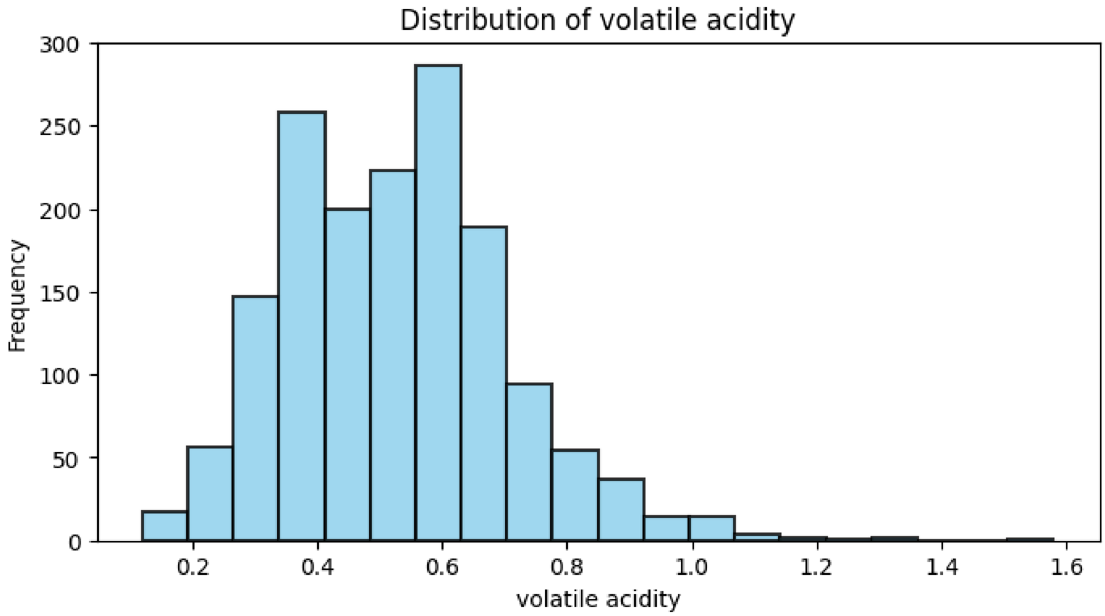

```
[ ]: alcohol                 0.476166
     volatile acidity        0.390558
     sulphates               0.251397
     citric acid             0.226373
     total sulfur dioxide    0.185100
     density                 0.174919
     chlorides               0.128907
     fixed acidity           0.124052
     pH                      0.057731
     free sulfur dioxide     0.050656
     residual sugar          0.013732
     Name: quality, dtype: float64
```

```
[ ]: print(dataset1.columns)
```

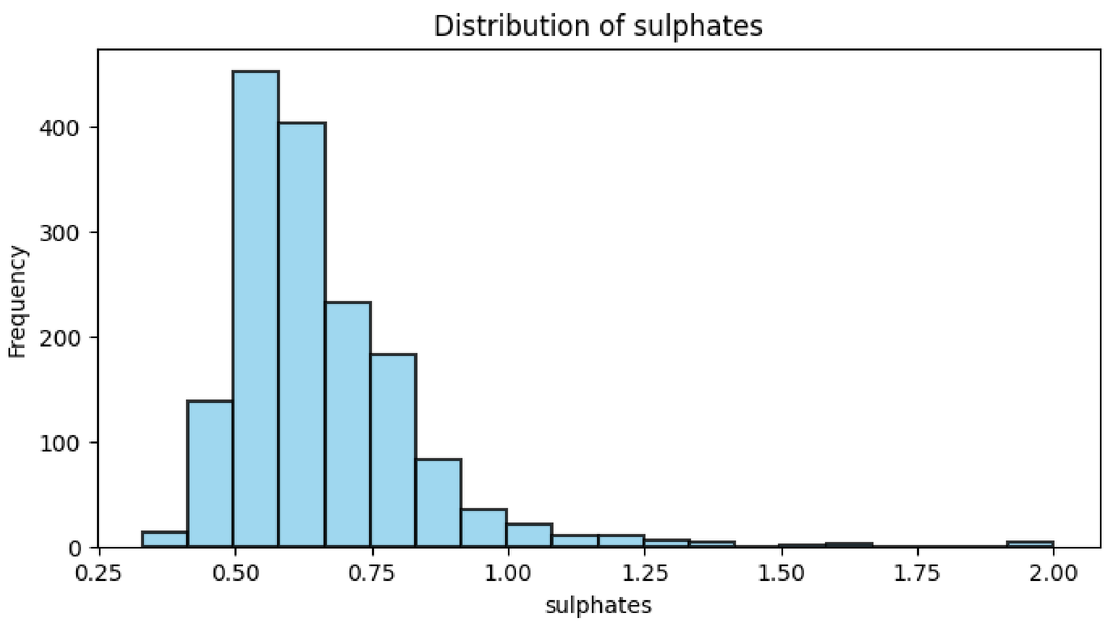
```
numerical_features = ['alcohol' , 'volatile acidity' ,'sulphates','citric acid'␣
↪,'total sulfur dioxide', 'density', 'chlorides'  ,'fixed acidity' , 'free␣
↪sulfur dioxide' ,'pH'  ,'residual sugar', 'quality' ]

# Plot histograms for numerical features
for feature in numerical_features:
    plt.figure(figsize=(8, 4))  # Adjust the figure size as necessary
    plt.hist(dataset1[feature], bins=20, edgecolor='k', color='skyblue',␣
↪linewidth=1.5, alpha=0.8)
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {feature}')
    plt.show()
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

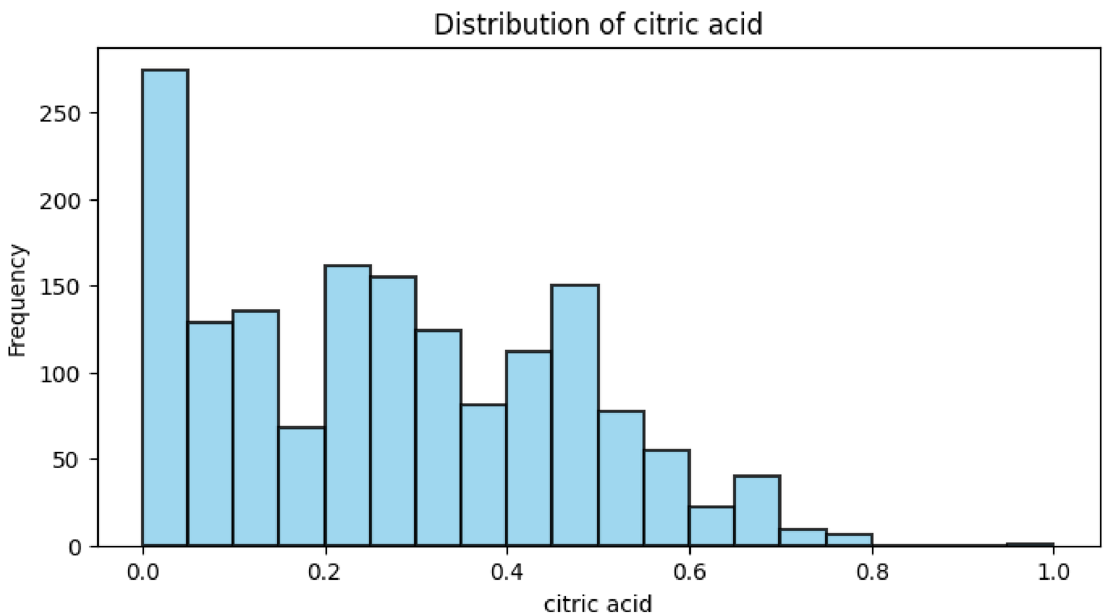


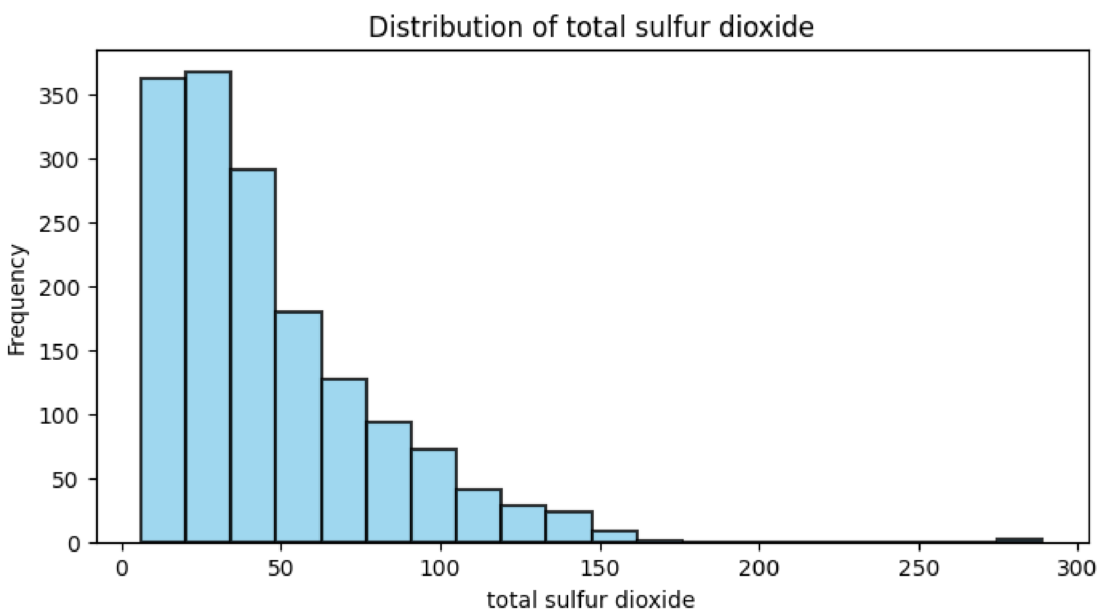


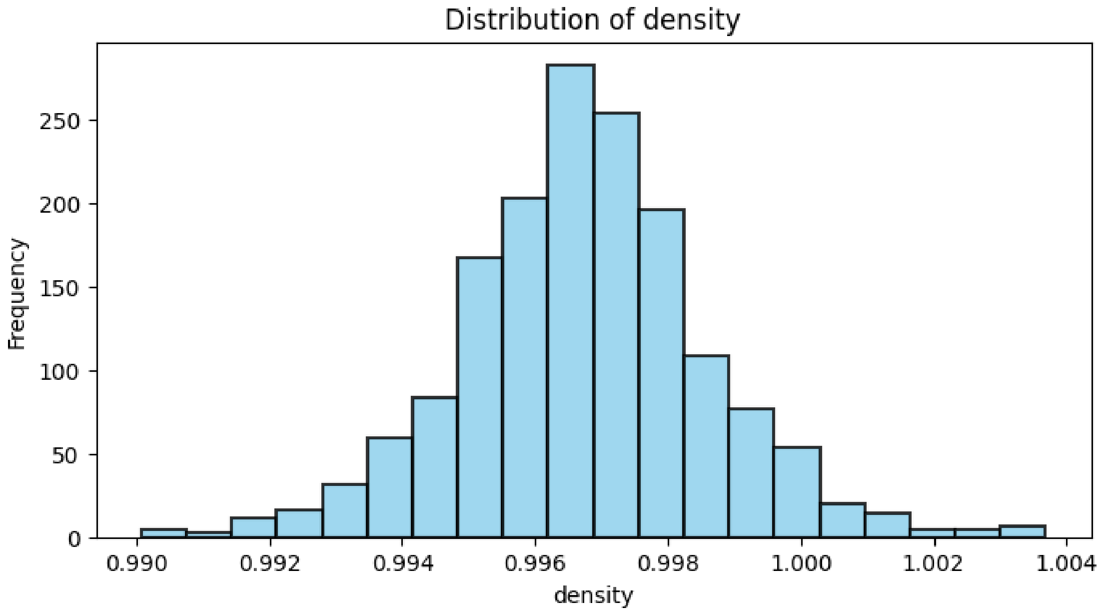
Distribution of volatile acidity
Frequency
300
250
200
150
100
50
0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
volatile acidity

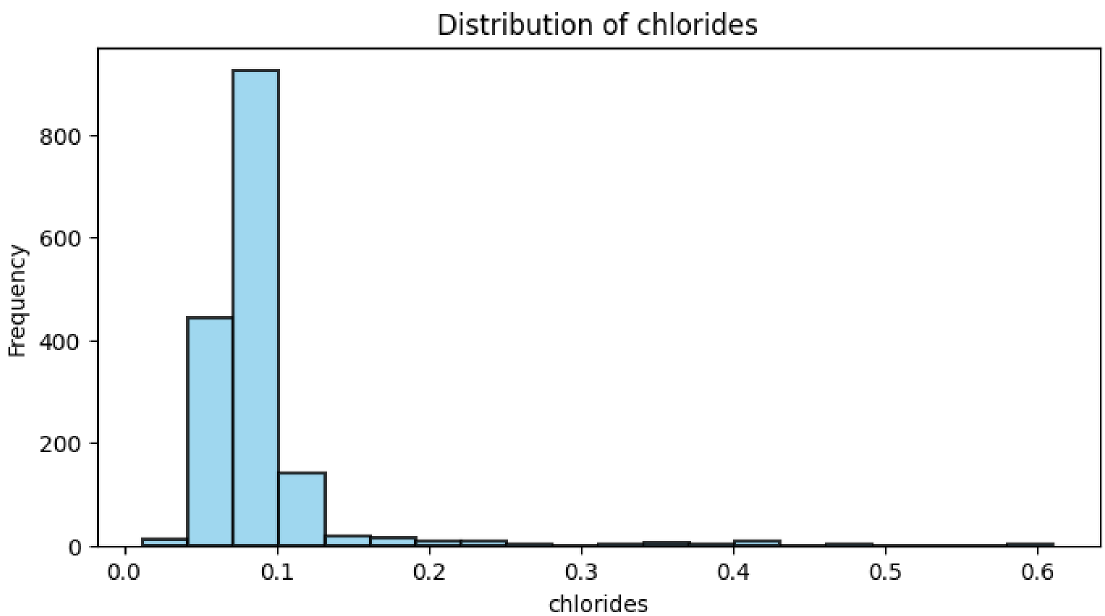


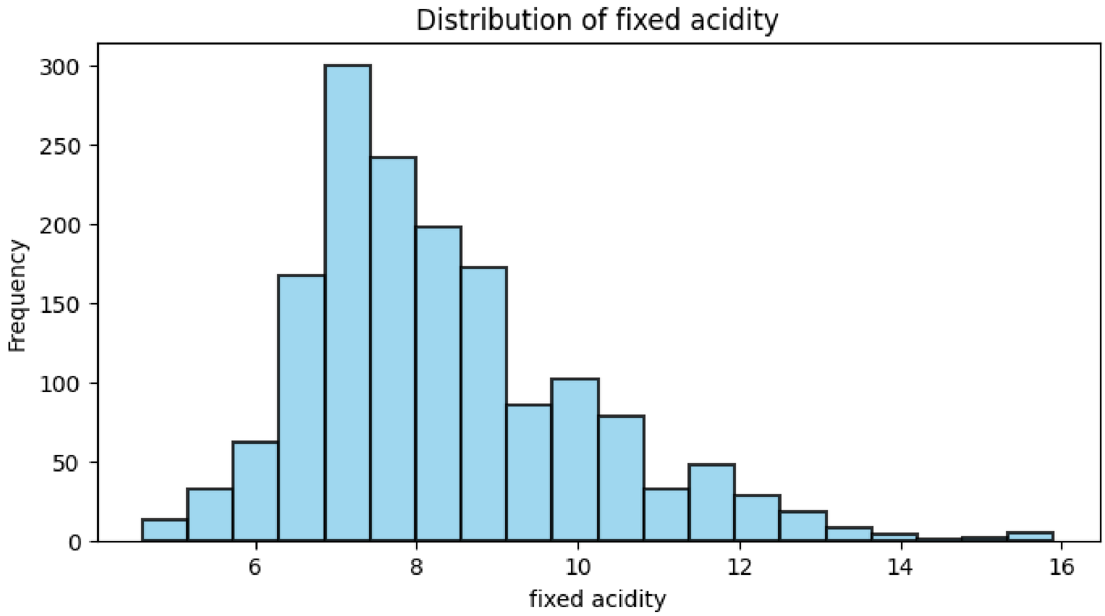
Distribution of sulphates
Frequency
400
300
200
100
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
sulphates

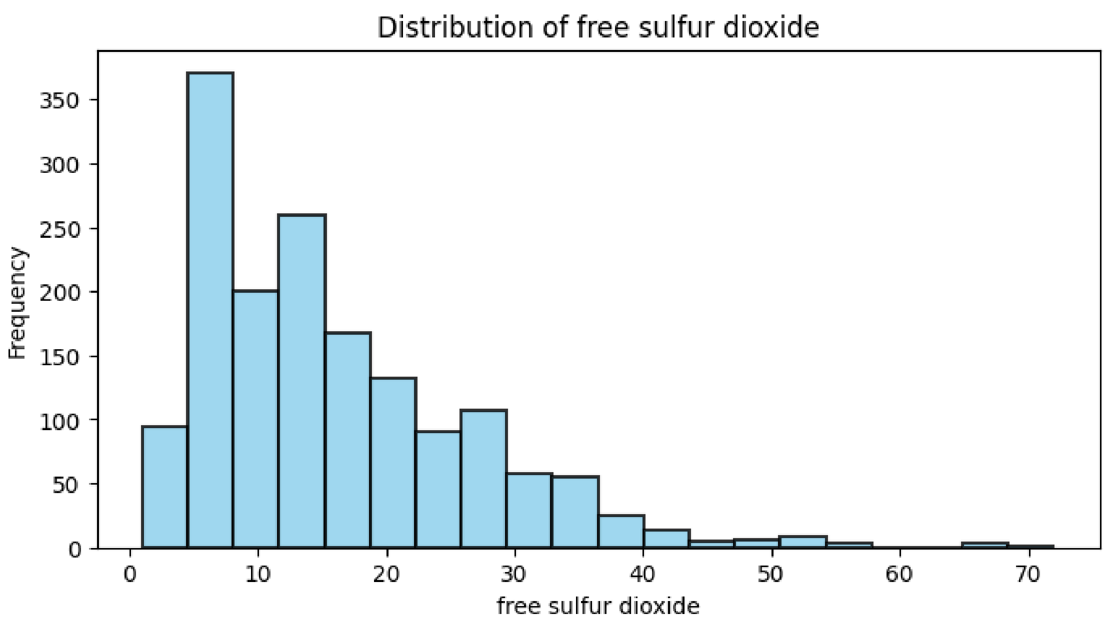

Distribution of citric acid
Frequency
0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0
citric acid

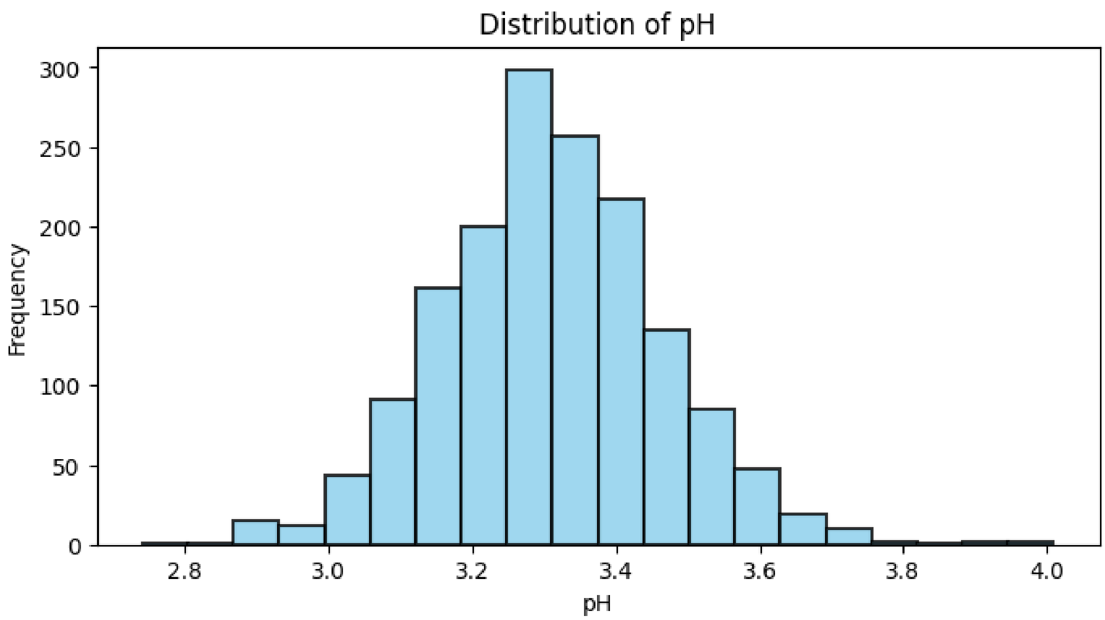

Distribution of total sulfur dioxide
Frequency
0
50
100
150
200
250
300
350
0
50
100
150
200
250
300
total sulfur dioxide


Distribution of density
Frequency
density
0
50
100
150
200
250
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004

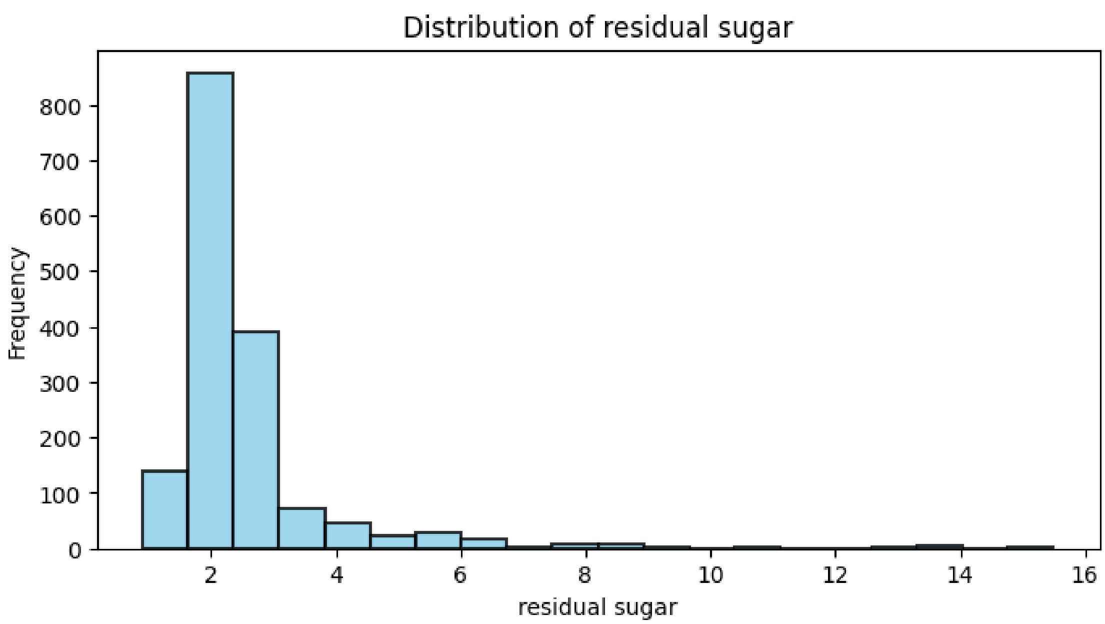

Distribution of chlorides
Frequency
chlorides
0
200
400
600
800
0.0
0.1
0.2
0.3
0.4
0.5
0.6


Distribution of fixed acidity
Frequency
300
250
200
150
100
50
0
6
8
10
12
14
16
fixed acidity

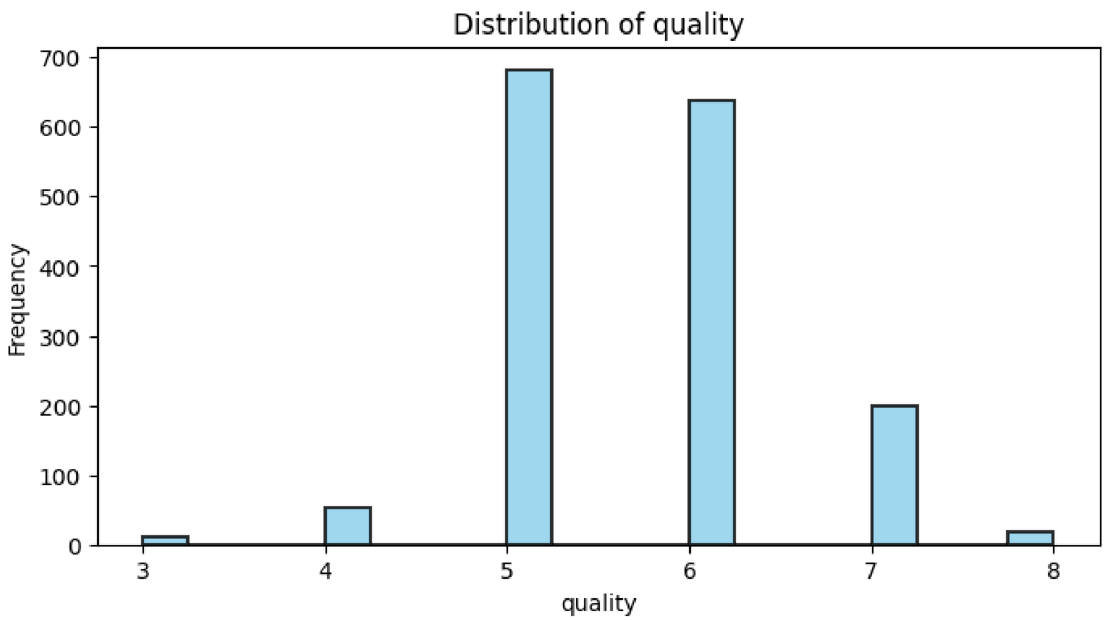

Distribution of free sulfur dioxide
Frequency
350
300
250
200
150
100
50
0
0
10
20
30
40
50
60
70
free sulfur dioxide



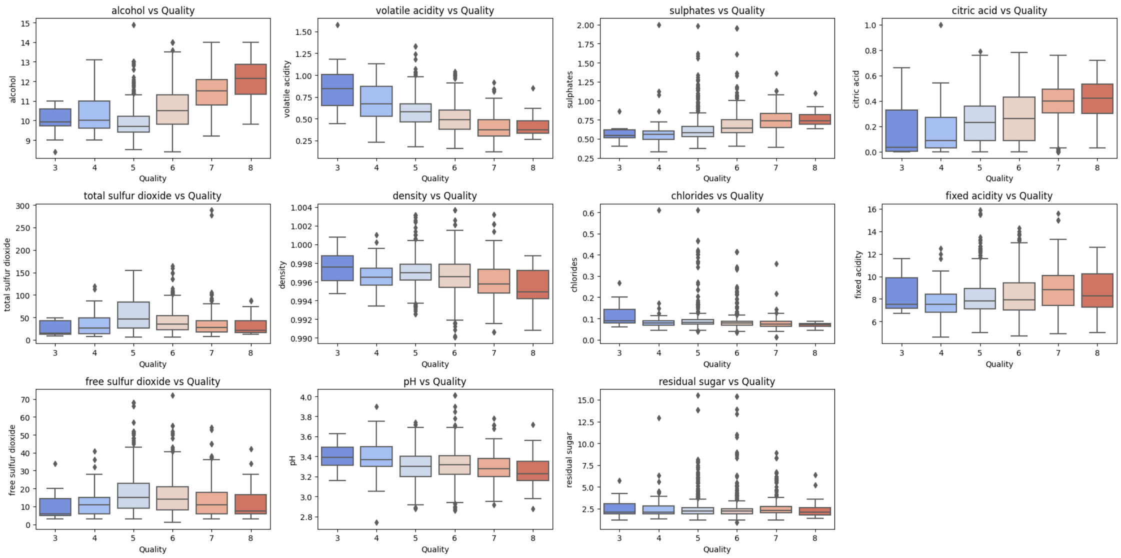
Distribution of pH
Frequency
pH
0
50
100
150
200
250
300
2.8
3.0
3.2
3.4
3.6
3.8
4.0



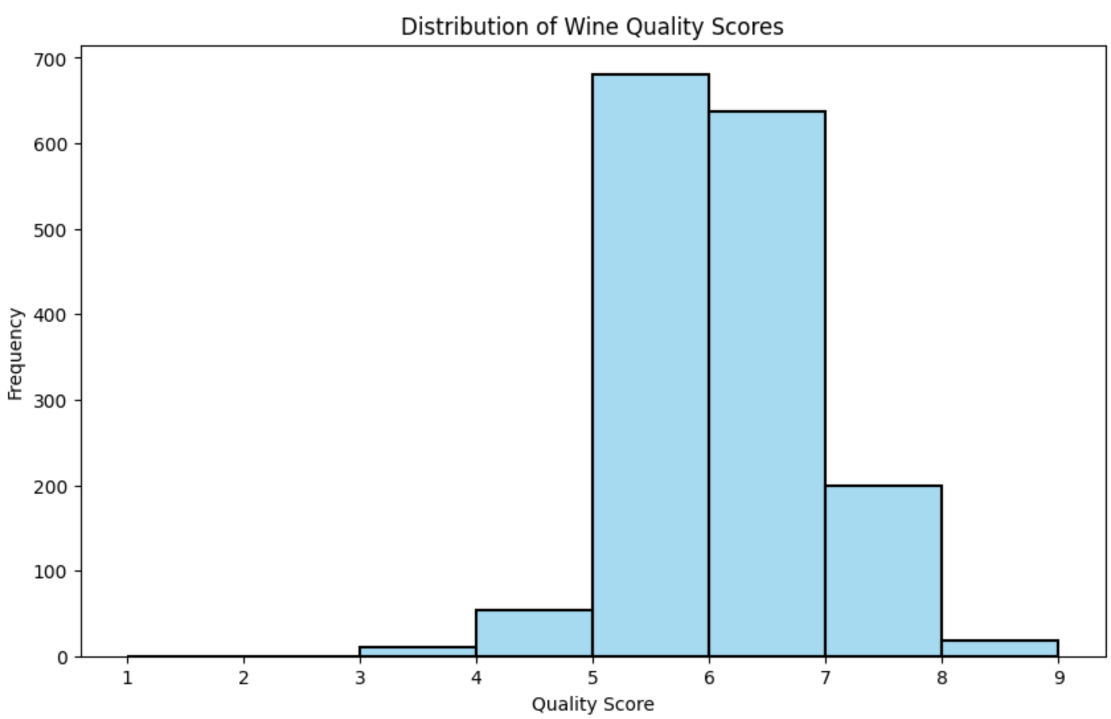
Distribution of residual sugar
Frequency
residual sugar
0
100
200
300
400
500
600
700
800
2
4
6
8
10
12
14
16



[ ]:
```
selected_features = ['alcohol' , 'volatile acidity' ,'sulphates','citric acid'␣
↪,'total sulfur dioxide', 'density', 'chlorides'  ,'fixed acidity' , 'free␣
↪sulfur dioxide' ,'pH'  ,'residual sugar']

# Plotting box plots for selected features against 'quality'
plt.figure(figsize=(20, 10))

for index, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, index)
    sns.boxplot(x='quality', y=feature, data=dataset1, palette='coolwarm')
    plt.title(f'{feature} vs Quality')
    plt.xlabel('Quality')
    plt.ylabel(feature)

plt.tight_layout()
plt.show()
```



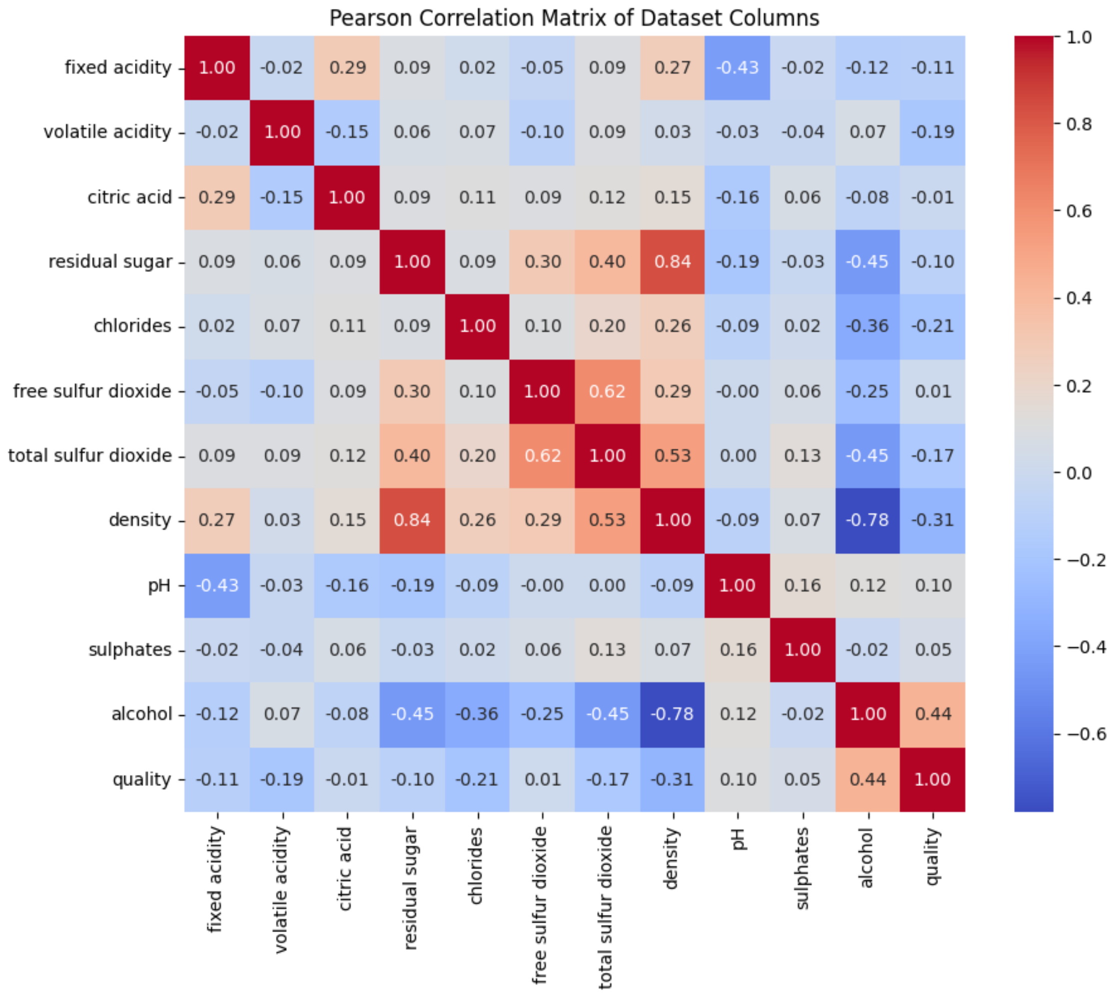

```
[ ]: plt.figure(figsize=(10, 6))
     sns.histplot(dataset1['quality'], kde=False, color='skyblue', bins=range(1,␣
      ↪10), edgecolor='k', linewidth=1.5)
     plt.title('Distribution of Wine Quality Scores')
     plt.xlabel('Quality Score')
     plt.ylabel('Frequency')
     plt.xticks(range(1, 10))  # Setting x-axis ticks to show each quality score
     plt.show()
```



[ ]:
```
dataset2 = pd.read_csv(whitewine, delimiter=';')
dataset2.head()
```

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.0              0.27         0.36            20.7      0.045
1            6.3              0.30         0.34             1.6      0.049
2            8.1              0.28         0.40             6.9      0.050
3            7.2              0.23         0.32             8.5      0.058
4            7.2              0.23         0.32             8.5      0.058

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 45.0                 170.0   1.0010  3.00       0.45
1                 14.0                 132.0   0.9940  3.30       0.49
2                 30.0                  97.0   0.9951  3.26       0.44
3                 47.0                 186.0   0.9956  3.19       0.40
4                 47.0                 186.0   0.9956  3.19       0.40

   alcohol  quality
0      8.8        6
1      9.5        6
2     10.1        6
3      9.9        6
4      9.9        6
```

[ ]:
```
import seaborn as sns
corr_matrix = dataset2.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Pearson Correlation Matrix of Dataset Columns')
plt.show()

# Identify features with the highest absolute correlation with the target␣
↪variable
target_variable = 'quality'  # Assuming 'quality' is the target variable
correlation_with_target = corr_matrix[target_variable].drop(target_variable).
↪abs().sort_values(ascending=False)

correlation_with_target
```

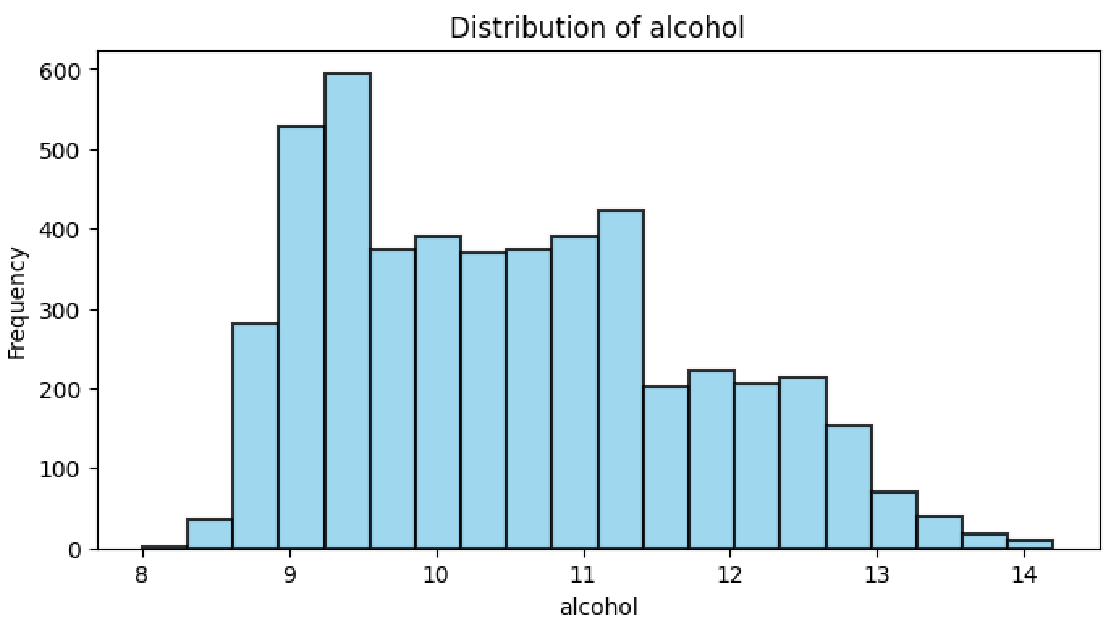


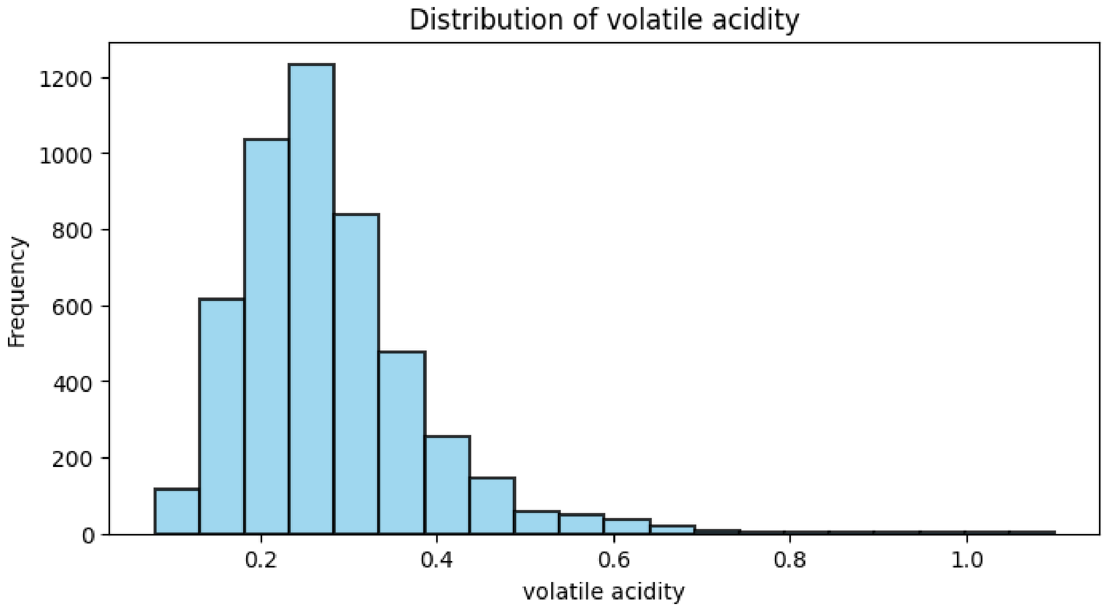

```
[ ]: alcohol                 0.435575
     density                 0.307123
     chlorides               0.209934
     volatile acidity        0.194723
     total sulfur dioxide    0.174737
     fixed acidity           0.113663
     pH                      0.099427
     residual sugar          0.097577
     sulphates               0.053678
     citric acid             0.009209
     free sulfur dioxide     0.008158
     Name: quality, dtype: float64
```

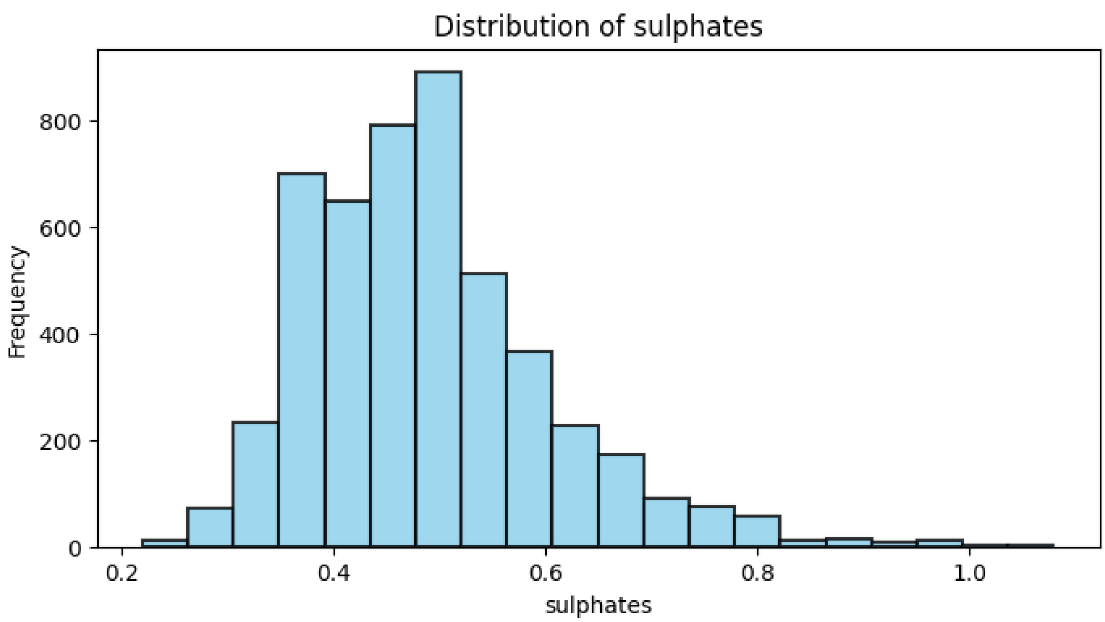
```
[ ]: print(dataset2.columns)
```

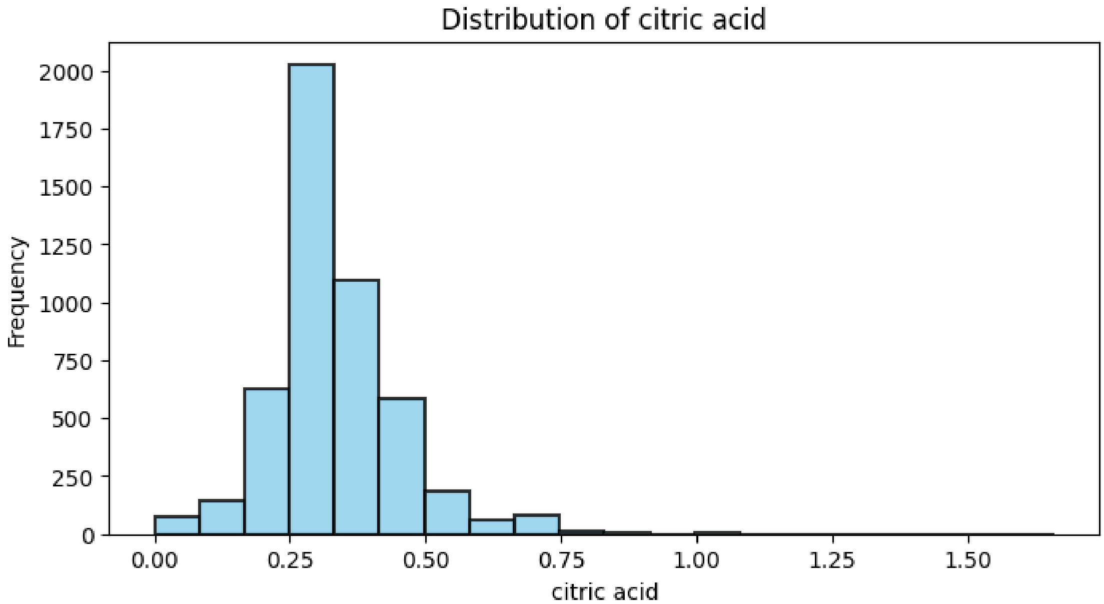
```python
numerical_features = ['alcohol' , 'volatile acidity' ,'sulphates','citric acid'␣
↪,'total sulfur dioxide', 'density', 'chlorides'  ,'fixed acidity' , 'free␣
↪sulfur dioxide' ,'pH'  ,'residual sugar', 'quality' ]

# Plot histograms for numerical features
for feature in numerical_features:
    plt.figure(figsize=(8, 4))  # Adjust the figure size as necessary
    plt.hist(dataset2[feature], bins=20, edgecolor='k', color='skyblue',␣
↪linewidth=1.5, alpha=0.8)
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {feature}')
    plt.show()
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

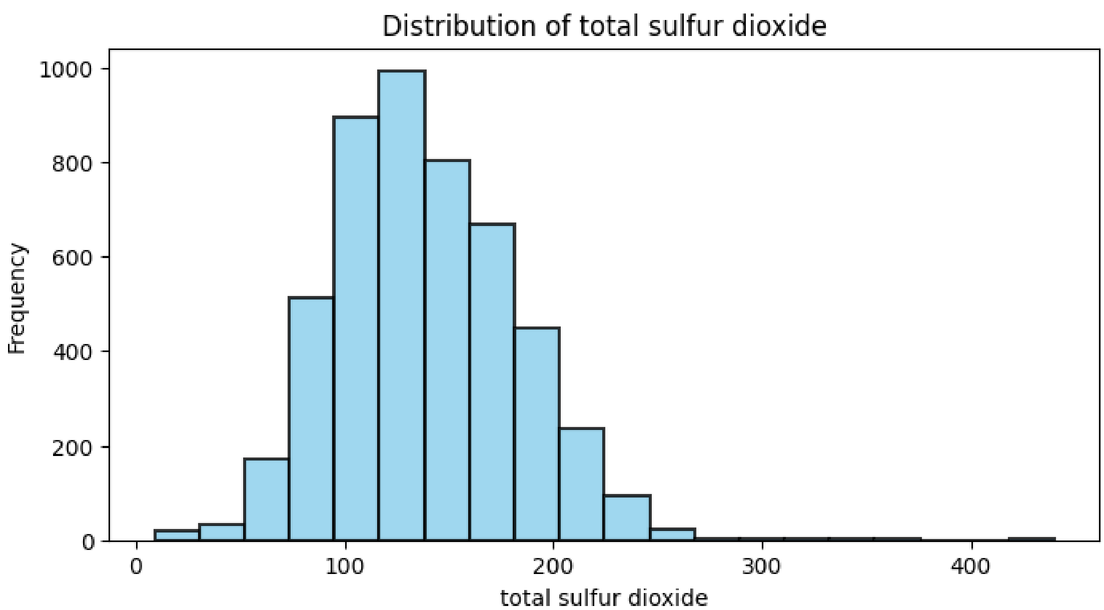



Distribution of volatile acidity
Frequency
volatile acidity
0
200
400
600
800
1000
1200
0.2
0.4
0.6
0.8
1.0

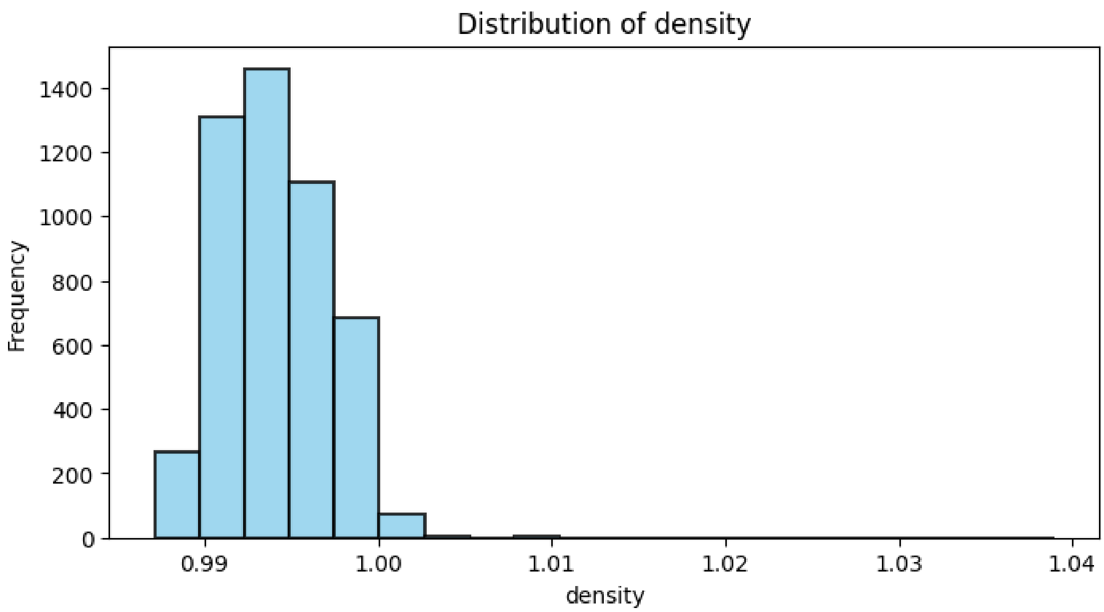

Distribution of sulphates
Frequency
sulphates
0
200
400
600
800
0.2
0.4
0.6
0.8
1.0

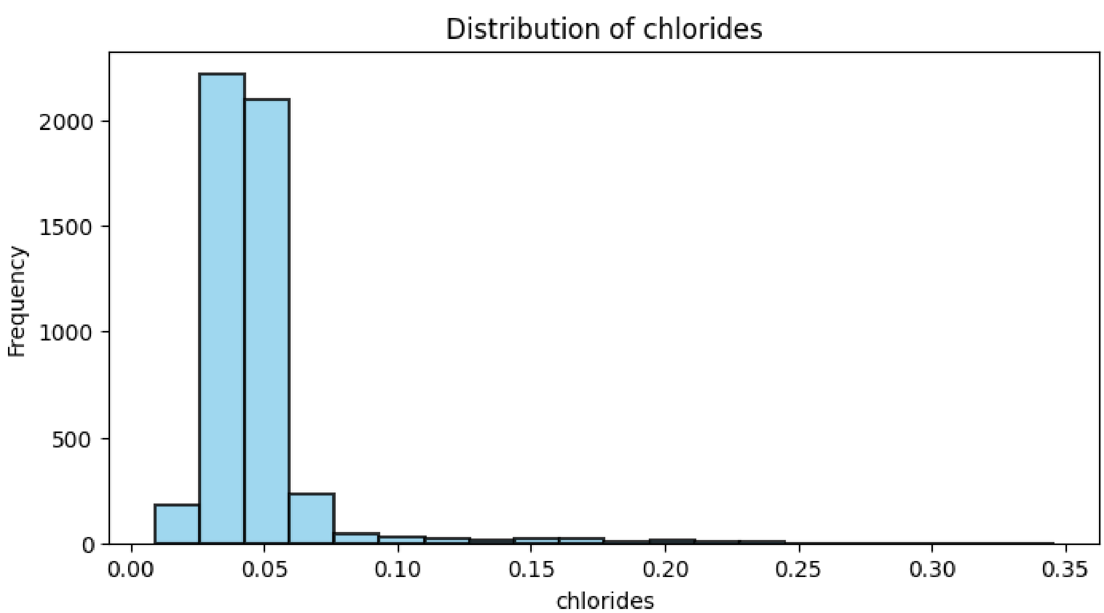


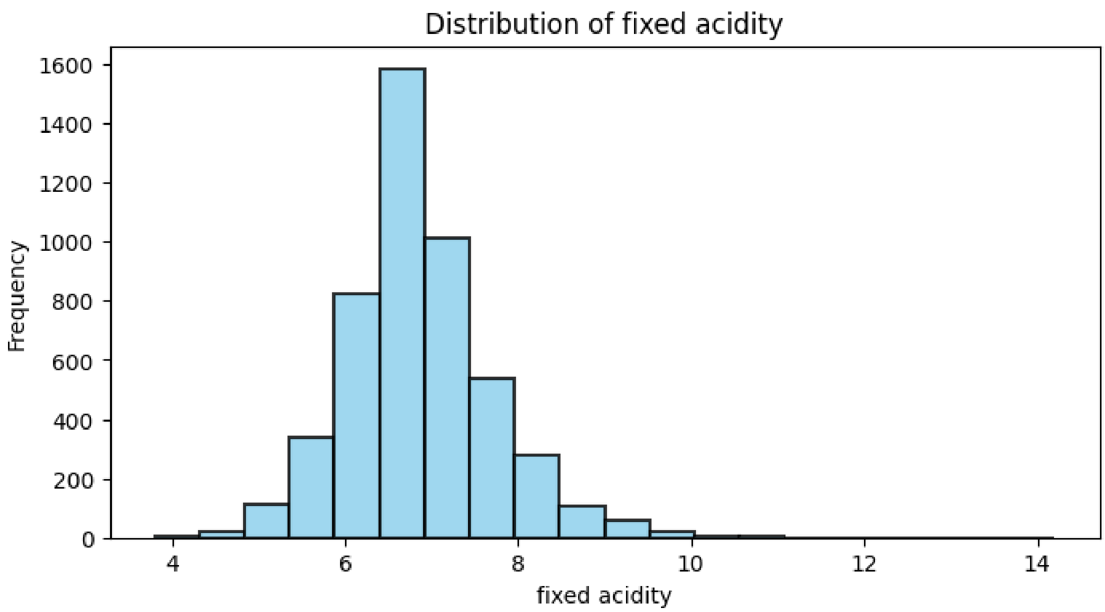
Distribution of citric acid
Frequency
0
250
500
750
1000
1250
1500
1750
2000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
citric acid



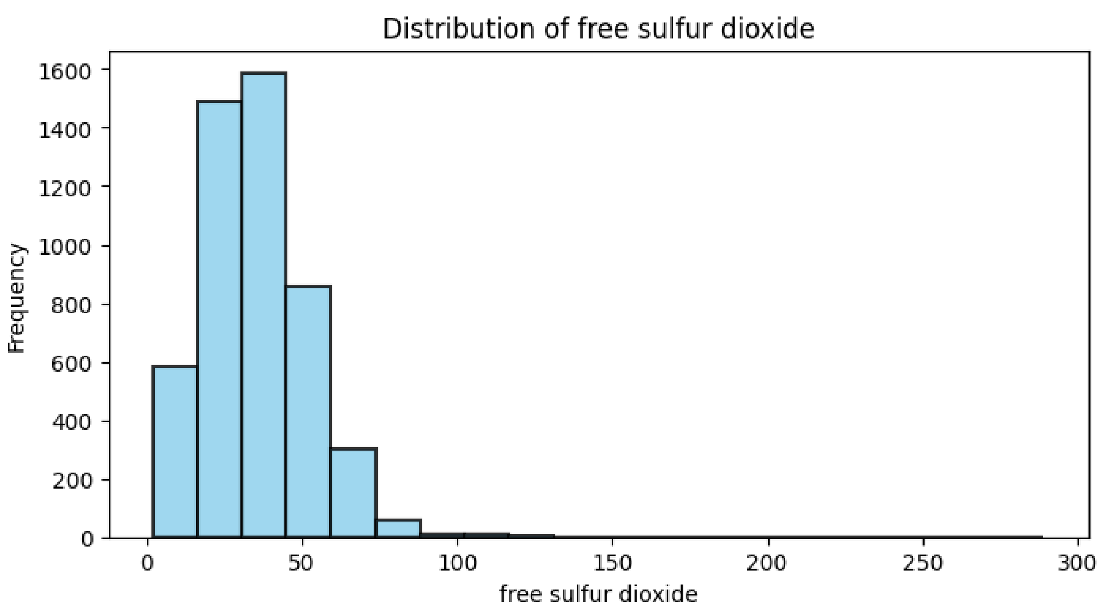
Distribution of total sulfur dioxide
Frequency
0
200
400
600
800
1000
0
100
200
300
400
total sulfur dioxide


Distribution of density
Frequency
0
200
400
600
800
1000
1200
1400
0.99
1.00
1.01
1.02
1.03
1.04
density

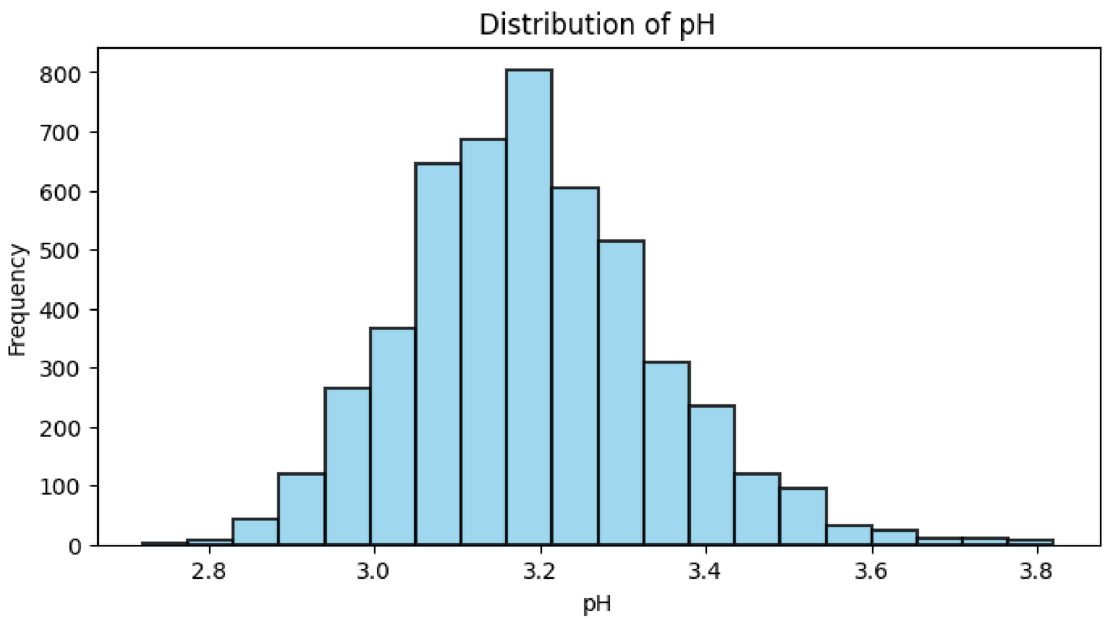

Distribution of chlorides
Frequency
0
500
1000
1500
2000
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
chlorides

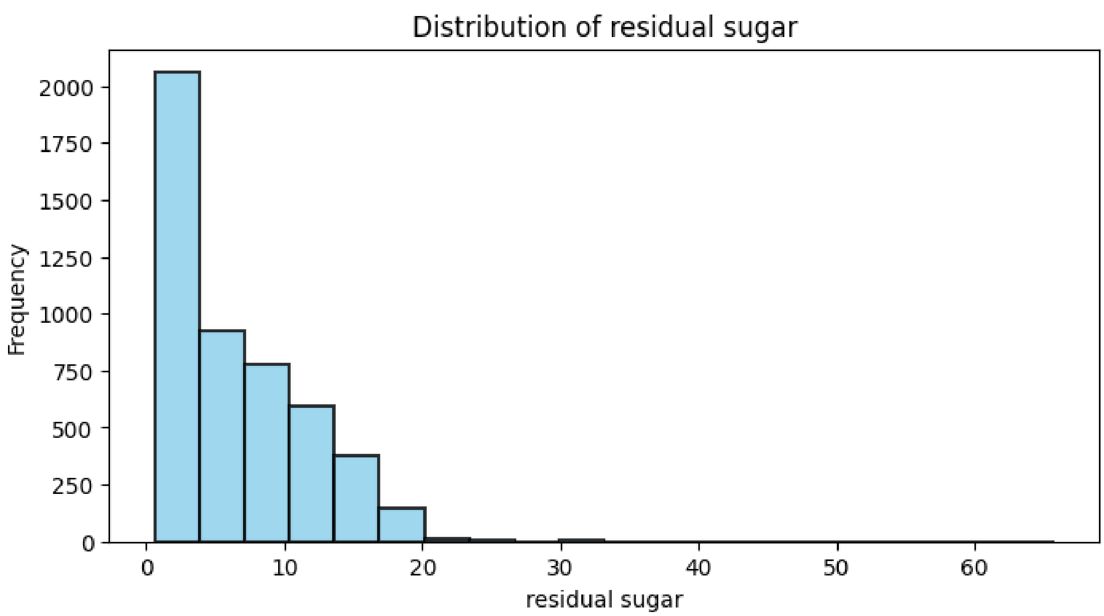


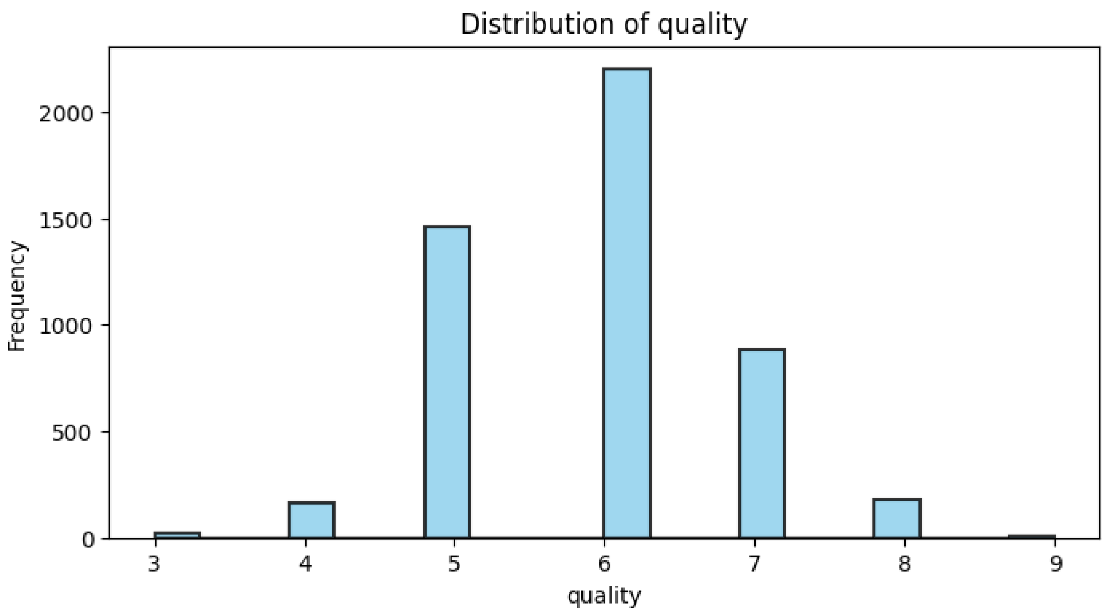
Distribution of fixed acidity
Frequency
fixed acidity



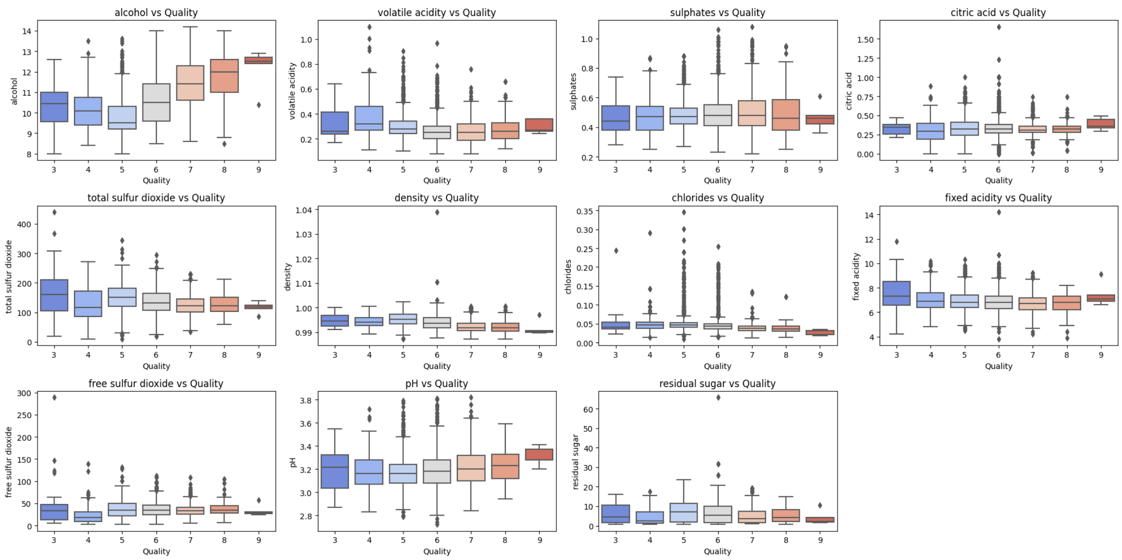
Distribution of free sulfur dioxide
Frequency
free sulfur dioxide


Distribution of pH
Frequency
0
100
200
300
400
500
600
700
800
2.8
3.0
3.2
3.4
3.6
3.8
pH

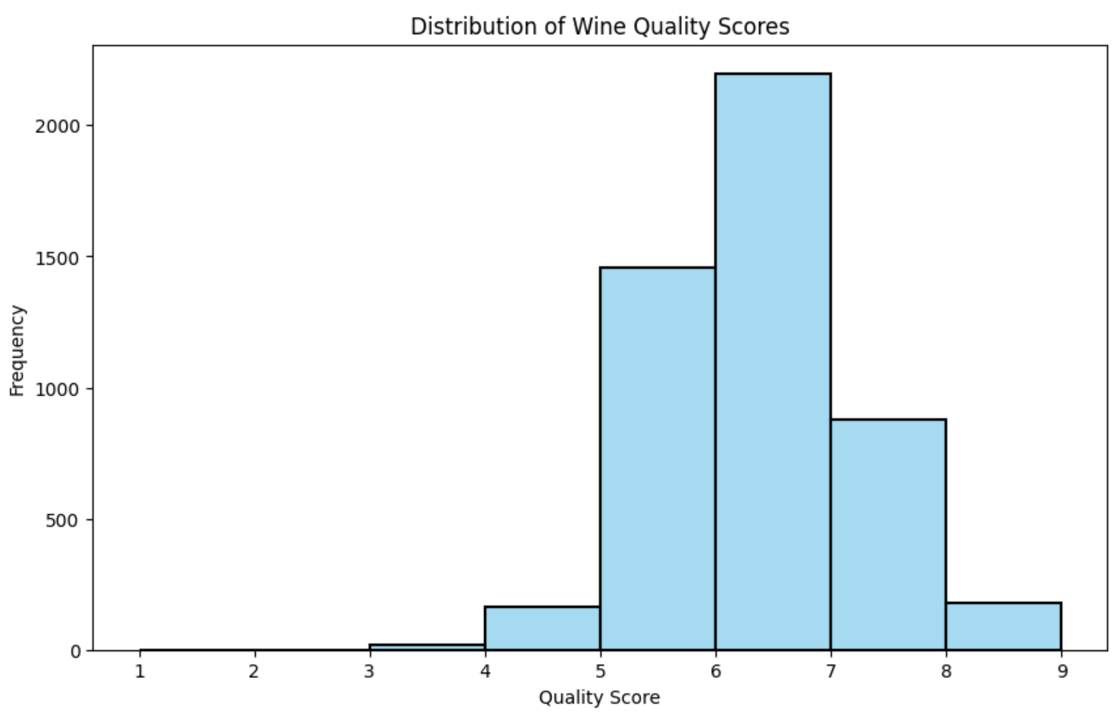

Distribution of residual sugar
Frequency
0
250
500
750
1000
1250
1500
1750
2000
0
10
20
30
40
50
60
residual sugar



[ ]:
```
selected_features = ['alcohol' , 'volatile acidity' ,'sulphates','citric acid'␣
 ↪,'total sulfur dioxide', 'density', 'chlorides'  ,'fixed acidity' , 'free␣
 ↪sulfur dioxide' ,'pH'  ,'residual sugar']

# Plotting box plots for selected features against 'quality'
plt.figure(figsize=(20, 10))

for index, feature in enumerate(selected_features, 1):
    plt.subplot(3, 4, index)
    sns.boxplot(x='quality', y=feature, data=dataset2, palette='coolwarm')
    plt.title(f'{feature} vs Quality')
    plt.xlabel('Quality')
    plt.ylabel(feature)

plt.tight_layout()
plt.show()
```




```
[ ]: plt.figure(figsize=(10, 6))
sns.histplot(dataset2['quality'], kde=False, color='skyblue', bins=range(1, 10), edgecolor='k', linewidth=1.5)
plt.title('Distribution of Wine Quality Scores')
plt.xlabel('Quality Score')
plt.ylabel('Frequency')
plt.xticks(range(1, 10))  # Setting x-axis ticks to show each quality score
plt.show()
```



6 Q2.1 Encoding for the Diamonds Dataset and Standardization

```
[ ]: dataset_d = diamonds.copy(deep=True)
```

```
[ ]: dataset_d.head()
```

```
[ ]:   color clarity  carat        cut  symmetry     polish  depth_percent  \
    0     E    VVS2   0.09  Excellent  Very Good  Very Good           62.7
    1     E    VVS2   0.09  Very Good  Very Good  Very Good           61.9
    2     E    VVS2   0.09  Excellent  Very Good  Very Good           61.1
    3     E    VVS2   0.09  Excellent  Very Good  Very Good           62.0
    4     E    VVS2   0.09  Very Good  Very Good  Excellent           64.9

       table_percent  length  width  depth girdle_min girdle_max  price
    0           59.0    2.85   2.87   1.79          M          M    200
    1           59.0    2.84   2.89   1.78        STK        STK    200
    2           59.0    2.88   2.90   1.77         TN          M    200
    3           59.0    2.86   2.88   1.78          M        STK    200
    4           58.5    2.79   2.83   1.82        STK        STK    200
```

```
[ ]: color_dict = {'M': 1, 'L': 2, 'K': 3, 'J': 4, 'I': 5, 'H': 6, 'G': 7, 'F': 8,␣
      ↪'E': 9, 'D': 10}
     cut_dict = {'Very Good': 1, 'Excellent': 2}
     symmetry_dict = {'Very Good': 1, 'Excellent': 2}
     polish_dict = {'Very Good': 1, 'Excellent': 2}
     girdle_min_dict = {'unknown': 1, 'XTK': 2, 'VTK': 3, 'TK': 4, 'STK': 5, 'M': 6,␣
      ↪'STN': 7, 'TN': 8, 'VTN': 9, 'XTN': 10}
     girdle_max_dict = {'XTN': 1, 'VTN': 2, 'TN': 3, 'STN': 4, 'M': 5, 'STK': 6,␣
      ↪'TK': 7, 'VTK': 8, 'XTK': 9, 'unknown': 10}
     clarity_dict = {'I3': 1, 'I2': 2, 'I1': 3, 'SI2': 4, 'SI1': 5, 'VS2': 6, 'VS1':␣
      ↪7, 'VVS2': 8, 'VVS1': 9, 'IF': 10}
```

```
[ ]: dataset_d['cut_encoded'] = dataset_d.cut.map(cut_dict)
     dataset_d['color_encoded'] = dataset_d.color.map(color_dict)
     dataset_d['clarity_encoded'] = dataset_d.clarity.map(clarity_dict)
     dataset_d['symmetry_encoded'] = dataset_d.symmetry.map(symmetry_dict)
     dataset_d['polish_encoded'] = dataset_d.polish.map(polish_dict)
     dataset_d['girdle_min_encoded'] = dataset_d.girdle_min.map(girdle_min_dict)
     dataset_d['girdle_max_encoded'] = dataset_d.girdle_max.map(girdle_max_dict)
```

```
[ ]: dataset_d.head()
```

```
[ ]:  color clarity  carat        cut   symmetry     polish  depth_percent  \
    0     E    VVS2   0.09  Excellent  Very Good  Very Good           62.7
    1     E    VVS2   0.09  Very Good  Very Good  Very Good           61.9
    2     E    VVS2   0.09  Excellent  Very Good  Very Good           61.1
    3     E    VVS2   0.09  Excellent  Very Good  Very Good           62.0
    4     E    VVS2   0.09  Very Good  Very Good  Excellent           64.9

       table_percent  length  width  …  girdle_min girdle_max price  \
    0           59.0    2.85   2.87  …           M          M   200
    1           59.0    2.84   2.89  …         STK        STK   200
    2           59.0    2.88   2.90  …          TN          M   200
    3           59.0    2.86   2.88  …           M        STK   200
    4           58.5    2.79   2.83  …         STK        STK   200

       cut_encoded  color_encoded  clarity_encoded  symmetry_encoded  \
    0            2              9                8                 1
    1            1              9                8                 1
    2            2              9                8                 1
    3            2              9                8                 1
    4            1              9                8                 1

       polish_encoded  girdle_min_encoded  girdle_max_encoded
    0               1                   6                   5
    1               1                   5                   6
    2               1                   8                   5
    3               1                   6                   6
    4               2                   5                   6

    [5 rows x 21 columns]
```

```
[ ]: dataset_encoded = dataset_d.
      ↪drop(columns=['cut','color','clarity','symmetry','polish','girdle_min','girdle_max'])
```

```
[ ]: dataset_encoded.head()
```

```
[ ]:   carat  depth_percent  table_percent  length  width  depth  price  \
    0   0.09           62.7           59.0    2.85   2.87   1.79    200
    1   0.09           61.9           59.0    2.84   2.89   1.78    200
    2   0.09           61.1           59.0    2.88   2.90   1.77    200
    3   0.09           62.0           59.0    2.86   2.88   1.78    200
    4   0.09           64.9           58.5    2.79   2.83   1.82    200

       cut_encoded  color_encoded  clarity_encoded  symmetry_encoded  \
    0            2              9                8                 1
    1            1              9                8                 1
    2            2              9                8                 1
    3            2              9                8                 1
```

```
4          1            9             8              1

   polish_encoded  girdle_min_encoded  girdle_max_encoded
0               1                   6                   5
1               1                   5                   6
2               1                   8                   5
3               1                   6                   6
4               2                   5                   6
```

[ ]:
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import mutual_info_regression, f_regression,␣
 ↪SelectKBest

numerical_cols = ['carat', 'depth_percent', 'table_percent', 'length', 'width',␣
 ↪'depth',
                  'cut_encoded', 'color_encoded', 'clarity_encoded',␣
 ↪'symmetry_encoded', 'polish_encoded',
                  'girdle_min_encoded', 'girdle_max_encoded', 'price']

numerical_data = dataset_encoded[numerical_cols]

scaler = StandardScaler()
standardized_data = scaler.fit_transform(numerical_data)

standardized_df = pd.DataFrame(standardized_data, columns=numerical_cols)

# Display the first few rows of the standardized data
standardized_df.head()
```

[ ]:
```
      carat  depth_percent  table_percent    length     width     depth  \
0 -1.157106       0.215866       0.345119 -2.146391 -2.078247 -0.730430
1 -1.157106       0.014689       0.345119 -2.156289 -2.059209 -0.735681
2 -1.157106      -0.186488       0.345119 -2.116697 -2.049690 -0.740932
3 -1.157106       0.039836       0.345119 -2.136493 -2.068728 -0.735681
4 -1.157106       0.769101       0.218693 -2.205778 -2.116324 -0.714676

   cut_encoded  color_encoded  clarity_encoded  symmetry_encoded  \
0     0.518390       0.916097         0.811981         -1.746964
1    -1.929051       0.916097         0.811981         -1.746964
2     0.518390       0.916097         0.811981         -1.746964
3     0.518390       0.916097         0.811981         -1.746964
4    -1.929051       0.916097         0.811981         -1.746964

   polish_encoded  girdle_min_encoded  girdle_max_encoded     price
0       -2.522184            0.663841           -1.191356 -0.659094
```

```
1        -2.522184            0.271002            -0.692649 -0.659094
2        -2.522184            1.449520            -1.191356 -0.659094
3        -2.522184            0.663841            -0.692649 -0.659094
4         0.396482            0.271002            -0.692649 -0.659094
```

7 Question2.2 Feature Selection

```
[ ]: X = standardized_df[numerical_cols[:-1]]  # Exclude 'price' as it's the target
y = standardized_df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Compute Mutual Information (MI) scores and select features
mi_scores = mutual_info_regression(X_train, y_train)
selector_mi = SelectKBest(mutual_info_regression, k='all').fit(X_train, y_train)
X_train_mi = selector_mi.transform(X_train)
X_test_mi = selector_mi.transform(X_test)
mi_results = pd.Series(mi_scores, index=X.columns, name="MI Scores").
 ↪sort_values(ascending=False)

# Compute F-scores and p-values
f_scores, p_values = f_regression(X_train, y_train)
f_results = pd.DataFrame({'F Score': f_scores, 'P Value': p_values}, index=X.
 ↪columns).sort_values(by="F Score", ascending=False)

# Select the top features based on F-scores
selector_f = SelectKBest(f_regression, k='all').fit(X_train, y_train)
X_train_f = selector_f.transform(X_train)
X_test_f = selector_f.transform(X_test)

# Optional: Display the results for analysis
print(mi_results)
print(f_results)
```

```
carat                 1.359443
width                 1.194142
length                1.185590
depth                 1.151173
color_encoded         0.174678
clarity_encoded       0.168540
depth_percent         0.043066
girdle_max_encoded    0.031447
cut_encoded           0.028569
symmetry_encoded      0.026849
girdle_min_encoded    0.022778
table_percent         0.022381
```

```
polish_encoded        0.007992
Name: MI Scores, dtype: float64
                          F Score       P Value
carat               605098.277047  0.000000e+00
length              371591.167281  0.000000e+00
width               300175.172484  0.000000e+00
depth                12198.889530  0.000000e+00
polish_encoded         369.416567  3.335977e-82
symmetry_encoded       279.562155  1.101999e-62
color_encoded          274.140524  1.663312e-61
table_percent          213.858280  2.177148e-48
clarity_encoded         89.768288  2.723880e-21
depth_percent           74.800842  5.269460e-18
cut_encoded             72.250572  1.916666e-17
girdle_min_encoded      70.375266  4.955407e-17
girdle_max_encoded      31.164850  2.375301e-08
```

[ ]:
```python
lowest_mi_features = mi_results.nsmallest(2).index.tolist()
lowest_mi_features
mi_results = pd.Series(mi_scores, index=X_train.columns).sort_values()
lowest_mi_scores = mi_results.nsmallest(2)
print(lowest_mi_scores)
```

```
polish_encoded    0.007992
table_percent     0.022381
dtype: float64
```

By selecting features with higer mutual information or significant F-scores, we effectively reduce the noise in the model. This can lead to lower test RMSE as the model makes predictions based on more relevant information. By using a subset of features, the complexity of the model can be reduced. This not only speeds up the training process but can also lead to more interpretable models. The qualitative impact of feature selection using mutual information and F-scores on test RMSE is generally positive, particularly for models where feature relevance and linear relationships are crucial.

8 3 Linear Regression for Diamonds

[ ]:
```python
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.feature_selection import SelectKBest, mutual_info_regression,
 ↪f_regression

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

models = {
    "LinearRegression": LinearRegression(),
```

```
    "Lasso": Lasso(),
    "Ridge": Ridge()
}

results = {
    "LinearRegression": {"MI": [], "F": []},
    "Lasso": {"MI": [], "F": []},
    "Ridge": {"MI": [], "F": []}
}

for i in range(1, X_train.shape[1] + 1):
    print(f'Testing models with top {i} features')

    selector_mi = SelectKBest(score_func=mutual_info_regression, k=i)
    X_train_mi = selector_mi.fit_transform(X_train, y_train)
    X_test_mi = selector_mi.transform(X_test)

    selector_f = SelectKBest(score_func=f_regression, k=i)
    X_train_f = selector_f.fit_transform(X_train, y_train)
    X_test_f = selector_f.transform(X_test)

    for name, model in models.items():
        cv_results_mi = cross_validate(model, X_train_mi, y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=10, n_jobs=-1)
        mean_rmse_mi = -cv_results_mi['test_score'].mean()

        cv_results_f = cross_validate(model, X_train_f, y_train,␣
 ↪scoring='neg_root_mean_squared_error', cv=10, n_jobs=-1)
        mean_rmse_f = -cv_results_f['test_score'].mean()

        results[name]["MI"].append(mean_rmse_mi)
        results[name]["F"].append(mean_rmse_f)

        print(f"{name} - k={i}: MI RMSE = {mean_rmse_mi}, F RMSE =␣
 ↪{mean_rmse_f}")

results
```

```
Testing models with top 1 features
LinearRegression - k=1: MI RMSE = 0.40666428458417664, F RMSE =
0.40666428458417664
Lasso - k=1: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=1: MI RMSE = 0.40666427988825227, F RMSE = 0.40666427988825227
Testing models with top 2 features
LinearRegression - k=2: MI RMSE = 0.4045651096064488, F RMSE =
0.3991077179724057
Lasso - k=2: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
```

```
Ridge - k=2: MI RMSE = 0.4045627117172515, F RMSE = 0.3991076690458337
Testing models with top 3 features
LinearRegression - k=3: MI RMSE = 0.3991403988206082, F RMSE =
0.3991403988206082
Lasso - k=3: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=3: MI RMSE = 0.3991405600664841, F RMSE = 0.3991405600664841
Testing models with top 4 features
LinearRegression - k=4: MI RMSE = 0.3991265114494793, F RMSE =
0.3991265114494793
Lasso - k=4: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=4: MI RMSE = 0.3991266700572979, F RMSE = 0.3991266700572979
Testing models with top 5 features
LinearRegression - k=5: MI RMSE = 0.3649984126186655, F RMSE =
0.3983113035747334
Lasso - k=5: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=5: MI RMSE = 0.3649985179770608, F RMSE = 0.39831145850139915
Testing models with top 6 features
LinearRegression - k=6: MI RMSE = 0.3438025870541922, F RMSE =
0.3971559110331241
Lasso - k=6: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=6: MI RMSE = 0.3438026402239046, F RMSE = 0.3971560323851326
Testing models with top 7 features
LinearRegression - k=7: MI RMSE = 0.3434431367567826, F RMSE =
0.36319820319746554
Lasso - k=7: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=7: MI RMSE = 0.3434431904229668, F RMSE = 0.36319828141401994
Testing models with top 8 features
LinearRegression - k=8: MI RMSE = 0.342972937630051, F RMSE =
0.36307753416682803
Lasso - k=8: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=8: MI RMSE = 0.34297298829574274, F RMSE = 0.36307761353472434
Testing models with top 9 features
LinearRegression - k=9: MI RMSE = 0.34220893065217584, F RMSE =
0.3432654350882277
Lasso - k=9: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=9: MI RMSE = 0.34220897489830215, F RMSE = 0.3432654811107309
Testing models with top 10 features
LinearRegression - k=10: MI RMSE = 0.3421980267399307, F RMSE =
0.34257600931346327
Lasso - k=10: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=10: MI RMSE = 0.3421980709609607, F RMSE = 0.34257605902379135
Testing models with top 11 features
LinearRegression - k=11: MI RMSE = 0.34191261545325574, F RMSE =
0.3421692376786013
Lasso - k=11: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=11: MI RMSE = 0.34191266270975273, F RMSE = 0.3421692856610001
Testing models with top 12 features
LinearRegression - k=12: MI RMSE = 0.3418640487727683, F RMSE =
```

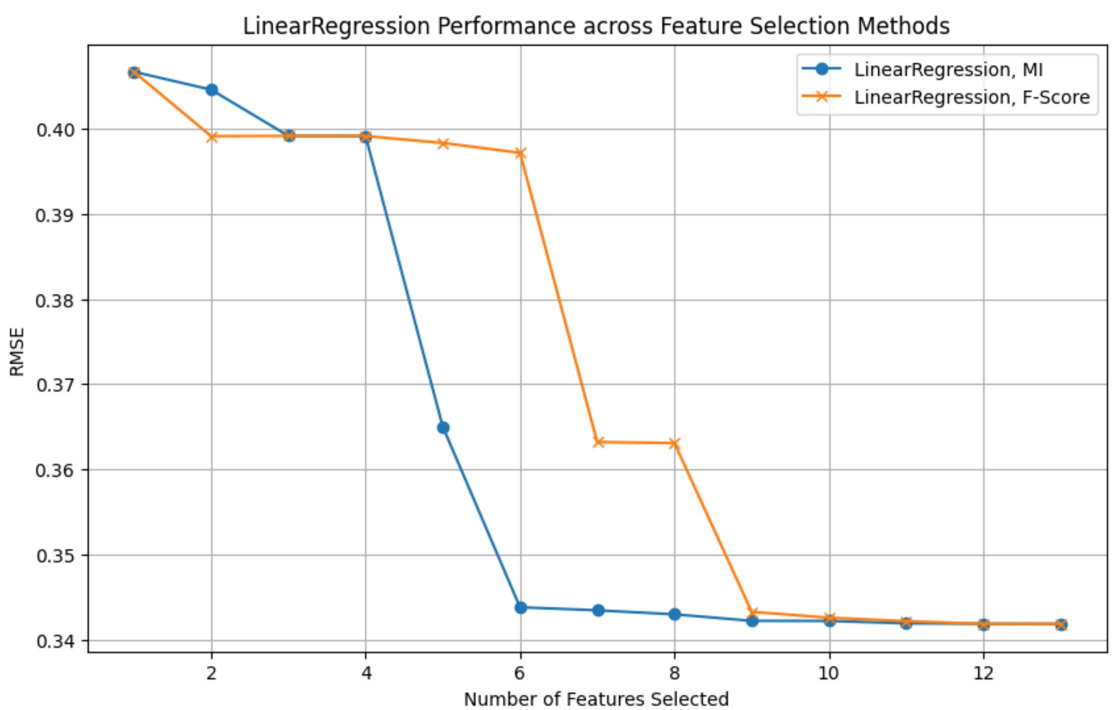
```
0.3418555942750469
Lasso - k=12: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=12: MI RMSE = 0.34186409289310105, F RMSE = 0.3418556395728873
Testing models with top 13 features
LinearRegression - k=13: MI RMSE = 0.34185351513246465, F RMSE =
0.34185351513246465
Lasso - k=13: MI RMSE = 0.999934846186267, F RMSE = 0.999934846186267
Ridge - k=13: MI RMSE = 0.3418535598620115, F RMSE = 0.3418535598620115
```

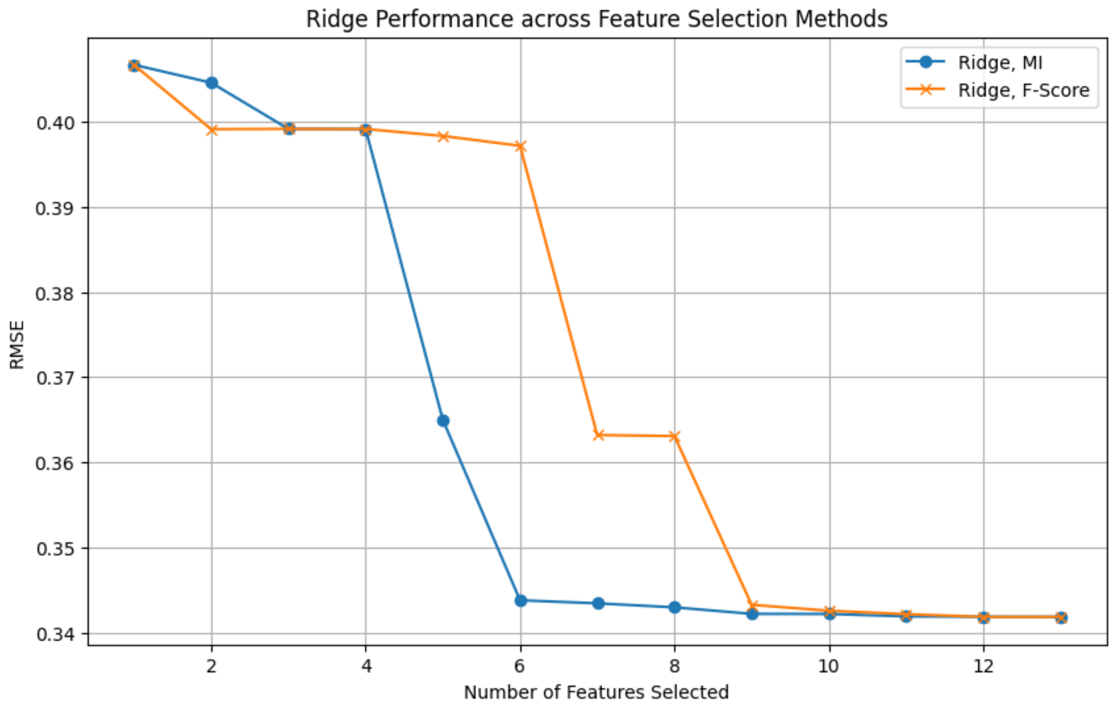
```
[ ]: {'LinearRegression': {'MI': [0.40666428458417664,
        0.4045651096064488,
        0.3991403988206082,
        0.3991265114494793,
        0.3649984126186655,
        0.3438025870541922,
        0.3434431367567826,
        0.342972937630051,
        0.34220893065217584,
        0.3421980267399307,
        0.34191261545325574,
        0.3418640487727683,
        0.34185351513246465],
       'F': [0.40666428458417664,
        0.3991077179724057,
        0.3991403988206082,
        0.3991265114494793,
        0.3983113035747334,
        0.3971559110331241,
        0.36319820319746554,
        0.36307753416682803,
        0.3432654350882277,
        0.34257600931346327,
        0.3421692376786013,
        0.3418555942750469,
        0.34185351513246465]},
      'Lasso': {'MI': [0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
        0.999934846186267,
```

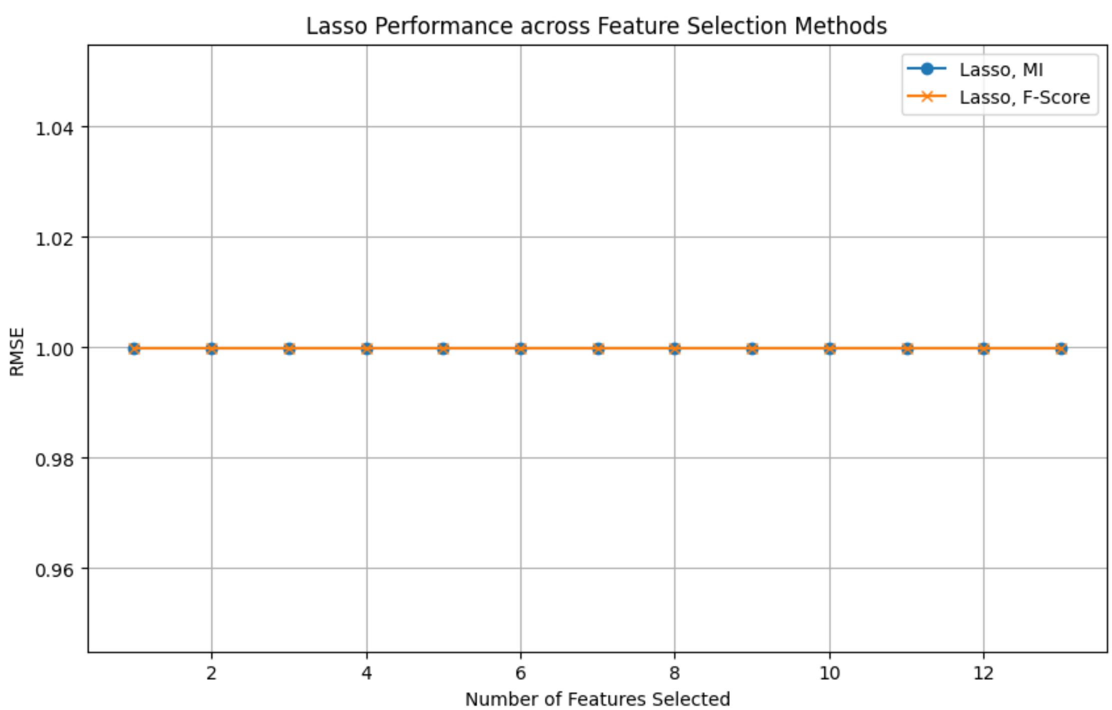
```
  0.999934846186267],
 'F': [0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267,
  0.999934846186267]},
'Ridge': {'MI': [0.40666427988825227,
  0.4045627117172515,
  0.3991405600664841,
  0.3991266700572979,
  0.3649985179770608,
  0.3438026402239046,
  0.3434431904229668,
  0.34297298829574274,
  0.34220897489830215,
  0.3421980709609607,
  0.34191266270975273,
  0.34186409289310105,
  0.3418535598620115],
 'F': [0.40666427988825227,
  0.3991076690458337,
  0.3991405600664841,
  0.3991266700572979,
  0.39831145850139915,
  0.3971560323851326,
  0.36319828141401994,
  0.36307761353472434,
  0.3432654811107309,
  0.34257605902379135,
  0.3421692856610001,
  0.3418556395728873,
  0.3418535598620115]}}
```

[ ]:
```
def plot_model_results(model_name, results, k_values):
    plt.figure(figsize=(10, 6))

    # Plotting Mutual Information (MI) scores
    mi_scores = results[model_name]["MI"]
    plt.plot(k_values, mi_scores, '-o', label=f'{model_name}, MI')
```

```
    # Plotting F-scores
    f_scores = results[model_name]["F"]
    plt.plot(k_values, f_scores, '-x', label=f'{model_name}, F-Score')

    plt.title(f'{model_name} Performance across Feature Selection Methods')
    plt.xlabel('Number of Features Selected')
    plt.ylabel('RMSE')
    plt.legend()
    plt.grid(True)
    plt.show()

# Assuming 'results' is correctly filled with your previous code
# Adjust 'k_values' based on the actual number of feature selections you made
k_values_linear = list(range(1, len(results['LinearRegression']['MI']) + 1))

k_values_ridge = list(range(1, len(results['Ridge']['MI']) + 1))
k_values_lasso = list(range(1, len(results['Lasso']['MI']) + 1))
# Corrected function calls with the right model keys
plot_model_results('LinearRegression', results, k_values_linear)

plot_model_results('Ridge', results, k_values_ridge)
plot_model_results('Lasso', results, k_values_lasso)
```







The objective function 1. Ordinary Least Square : min||Y − X ||ˆ2 2. Lasso : min||Y − X ||ˆ2+

|| ||1 3. Ridge : min||Y − X ||ˆ2+ || ||2

9 Question 4.1

1. Lasso Regression (L1 Regularization) The parameter controls the strength of the regularization. A larger leads to more coefficients being set to zero, increasing sparsity but potentially underfitting the data. Lasso tends to favor a solution with fewer nonzero coefficients, making it particularly useful when we believe many features are irrelevant or when we desire a model with simpler interpretation.
2. Ridge Regression (L2 Regularization) The regularization strength balances between fitting the training data well and reducing the magnitude of coefficients A larger results in greater shrinkage, leading to lower variance but potentially higher bias. Ridge is particularly useful when dealing with multicollinearity or when the number of parameters exceeds the number of observations.
3. L2 regularization term serves for shrinkage purpose while L1 can be used for feature selection or screening purposes.

10 Question 4.2

[ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LassoCV, RidgeCV
from sklearn.metrics import mean_squared_error
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

alphas = np.logspace(-6, 6, 13)

lasso_cv = LassoCV(alphas=alphas, cv=5, random_state=42).fit(X_train_scaled,␣
 ↪y_train)
ridge_cv = RidgeCV(alphas=alphas, scoring='neg_mean_squared_error', cv=5).
 ↪fit(X_train_scaled, y_train)

lasso_pred = lasso_cv.predict(X_test_scaled)
ridge_pred = ridge_cv.predict(X_test_scaled)
lasso_rmse = np.sqrt(mean_squared_error(y_test, lasso_pred))
ridge_rmse = np.sqrt(mean_squared_error(y_test, ridge_pred))

print(f"Lasso Best Alpha: {lasso_cv.alpha_}, RMSE: {lasso_rmse}")
print(f"Ridge Best Alpha: {ridge_cv.alpha_}, RMSE: {ridge_rmse}")
```

```python
if lasso_rmse < ridge_rmse:
    print("Lasso Regression is the best model.")
    best_model = 'Lasso'
    best_alpha = lasso_cv.alpha_
else:
    print("Ridge Regression is the best model.")
    best_model = 'Ridge'
    best_alpha = ridge_cv.alpha_

print(f"Best Regularization Scheme: {best_model}, Optimal Penalty Parameter:␣
↪{best_alpha}")
```

```
Lasso Best Alpha: 9.999999999999999e-06, RMSE: 0.3406201812319662
Ridge Best Alpha: 1.0, RMSE: 0.3406200781737286
Ridge Regression is the best model.
Best Regularization Scheme: Ridge, Optimal Penalty Parameter: 1.0
```

After finding the best alpha for each model, predictions are made on the test set. The Root Mean Squared Error (RMSE) is calculated for each model to assess their performance on unseen data. The RMSE is a common metric for regression tasks, providing an estimate of the standard deviation of the prediction errors.The model Ridge with the lower RMSE on the test set is considered the best model.

11 Question 4.3

For Ridge regularization,feature standardization often plays a significant role in improving model performance to ensure that the regularization is applied uniformly across all features, thus improving the model's performance and interpretability.

[ ]:
```python
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.feature_selection import SelectKBest, mutual_info_regression,␣
↪f_regression
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

models = {
    "Ridge": Ridge()
}

results = {
    "Non-Standardized": {model_name: {"MI": [], "F": []} for model_name in␣
↪models}
}
```

```python
# Function to evaluate models
def evaluate_models(data_variant, X_train, X_test, y_train, y_test, models,␣
↪results):
    print(f"Evaluating models for {data_variant} data...")
    for i in range(1, X_train.shape[1] + 1):
        print(f"\nSelecting top {i} features...")

        # Feature selection using mutual information
        selector_mi = SelectKBest(score_func=mutual_info_regression, k=i)
        X_train_mi = selector_mi.fit_transform(X_train, y_train)
        X_test_mi = selector_mi.transform(X_test)
        print(f"Selected features with MI: {selector_mi.
↪get_support(indices=True)}")

        # Feature selection using F-score
        selector_f = SelectKBest(score_func=f_regression, k=i)
        X_train_f = selector_f.fit_transform(X_train, y_train)
        X_test_f = selector_f.transform(X_test)
        print(f"Selected features with F-score: {selector_f.
↪get_support(indices=True)}")

        for name, model in models.items():
            print(f"\nEvaluating {name} model with MI-selected features...")
            cv_results_mi = cross_validate(model, X_train_mi, y_train,␣
↪scoring='neg_root_mean_squared_error', cv=10, n_jobs=-1)
            mean_rmse_mi = -cv_results_mi['test_score'].mean()
            print(f"Mean RMSE (MI): {mean_rmse_mi}")

            print(f"Evaluating {name} model with F-selected features...")
            cv_results_f = cross_validate(model, X_train_f, y_train,␣
↪scoring='neg_root_mean_squared_error', cv=10, n_jobs=-1)
            mean_rmse_f = -cv_results_f['test_score'].mean()
            print(f"Mean RMSE (F): {mean_rmse_f}")

            results[data_variant][name]["MI"].append(mean_rmse_mi)
            results[data_variant][name]["F"].append(mean_rmse_f)

    return results

# Now, call evaluate_models function and observe the printed outputs for each␣
↪step
results = evaluate_models("Non-Standardized", X_train, X_test, y_train, y_test,␣
↪models, results)
```

```python
# After evaluation, you can inspect the final results
print("\nFinal Results:")
print(results)
```

```
Evaluating models for Non-Standardized data…

Selecting top 1 features…
Selected features with MI: [0]
Selected features with F-score: [0]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.40666427988825227
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.40666427988825227

Selecting top 2 features…
Selected features with MI: [0 4]
Selected features with F-score: [0 3]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.4045627117172515
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3991076690458337

Selecting top 3 features…
Selected features with MI: [0 3 4]
Selected features with F-score: [0 3 4]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3991405600664841
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3991405600664841

Selecting top 4 features…
Selected features with MI: [0 3 4 5]
Selected features with F-score: [0 3 4 5]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3991266700572979
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3991266700572979

Selecting top 5 features…
Selected features with MI: [0 3 4 5 7]
Selected features with F-score: [ 0  3  4  5 10]

Evaluating Ridge model with MI-selected features…
```

```
Mean RMSE (MI): 0.3649985179770608
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.39831145850139915

Selecting top 6 features…
Selected features with MI: [0 3 4 5 7 8]
Selected features with F-score: [ 0  3  4  5  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3438026402239046
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3971560323851326

Selecting top 7 features…
Selected features with MI: [0 1 3 4 5 7 8]
Selected features with F-score: [ 0  3  4  5  7  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3434431904229668
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.36319828141401994

Selecting top 8 features…
Selected features with MI: [ 0  1  3  4  5  7  8 12]
Selected features with F-score: [ 0  2  3  4  5  7  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.34297298829574274
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.36307761353472434

Selecting top 9 features…
Selected features with MI: [ 0  1  3  4  5  7  8 11 12]
Selected features with F-score: [ 0  2  3  4  5  7  8  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3429727990809201
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3432654811107309

Selecting top 10 features…
Selected features with MI: [ 0  1  3  4  5  6  7  8 11 12]
Selected features with F-score: [ 0  1  2  3  4  5  7  8  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3421980709609607
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.34257605902379135
```

```
Selecting top 11 features…
Selected features with MI: [ 0  1  3  4  5  6  7  8  9 11 12]
Selected features with F-score: [ 0  1  2  3  4  5  6  7  8  9 10]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.34214841774362104
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3421692856610001

Selecting top 12 features…
Selected features with MI: [ 0  1  2  3  4  5  6  7  8  9 11 12]
Selected features with F-score: [ 0  1  2  3  4  5  6  7  8  9 10 11]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.34186409289310105
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3418556395728873

Selecting top 13 features…
Selected features with MI: [ 0  1  2  3  4  5  6  7  8  9 10 11 12]
Selected features with F-score: [ 0  1  2  3  4  5  6  7  8  9 10 11 12]

Evaluating Ridge model with MI-selected features…
Mean RMSE (MI): 0.3418535598620115
Evaluating Ridge model with F-selected features…
Mean RMSE (F): 0.3418535598620115

Final Results:
{'Non-Standardized': {'Ridge': {'MI': [0.40666427988825227, 0.4045627117172515,
0.3991405600664841, 0.3991266700572979, 0.3649985179770608, 0.3438026402239046,
0.3434431904229668, 0.34297298829574274, 0.3429727990809201, 0.3421980709609607,
0.34214841774362104, 0.34186409289310105, 0.3418535598620115], 'F':
[0.40666427988825227, 0.3991076690458337, 0.3991405600664841,
0.3991266700572979, 0.39831145850139915, 0.3971560323851326,
0.36319828141401994, 0.36307761353472434, 0.3432654811107309,
0.34257605902379135, 0.3421692856610001, 0.3418556395728873,
0.3418535598620115]}}}
```

12 Question 4.4

P-values of regression analysis provide a measure of the probability that the observed data would occur if the null hypothesis were true. If the p-value for some feature is very close to 0, we will have the confidence to say that particular feature is significant in the linear model

[ ]:
```
pip install statsmodels
```

```
Requirement already satisfied: statsmodels in /usr/local/lib/python3.10/dist-
packages (0.14.1)
Requirement already satisfied: numpy<2,>=1.18 in /usr/local/lib/python3.10/dist-
packages (from statsmodels) (1.25.2)
Requirement already satisfied: scipy!=1.9.2,>=1.4 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (1.11.4)
Requirement already satisfied: pandas!=2.1.0,>=1.0 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (1.5.3)
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-
packages (from statsmodels) (0.5.6)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels) (24.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas!=2.1.0,>=1.0->statsmodels)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas!=2.1.0,>=1.0->statsmodels) (2023.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from patsy>=0.5.4->statsmodels) (1.16.0)
```

[ ]:
```python
import statsmodels.api as sm
X_with_const = sm.add_constant(X)
model = sm.OLS(y, X_with_const).fit()

model_summary = model.summary()

print(model_summary)

p_values = model.pvalues
print("P-values for each feature:")
print(p_values)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.883
Model:                            OLS   Adj. R-squared:                  0.883
Method:                 Least Squares   F-statistic:                 8.729e+04
Date:                Mon, 18 Mar 2024   Prob (F-statistic):               0.00
Time:                        02:07:34   Log-Likelihood:                -51654.
No. Observations:              149871   AIC:                         1.033e+05
Df Residuals:                  149857   BIC:                         1.035e+05
Df Model:                          13
Covariance Type:            nonrobust
================================================================================
======
                         coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
```

```
------
const               -7.847e-17      0.001  -8.89e-14      1.000      -0.002
0.002
carat                   1.1873      0.004    313.393      0.000       1.180
1.195
depth_percent          -0.0292      0.001    -22.274      0.000      -0.032
-0.027
table_percent           0.0214      0.001     16.355      0.000       0.019
0.024
length                 -0.2109      0.005    -41.665      0.000      -0.221
-0.201
width                  -0.0077      0.003     -2.214      0.027      -0.015
-0.001
depth                  -0.0053      0.001     -5.569      0.000      -0.007
-0.003
cut_encoded             0.0197      0.001     18.189      0.000       0.018
0.022
color_encoded           0.1669      0.001    182.627      0.000       0.165
0.169
clarity_encoded         0.1215      0.001    132.253      0.000       0.120
0.123
symmetry_encoded        0.0054      0.001      4.978      0.000       0.003
0.008
polish_encoded          0.0026      0.001      2.746      0.006       0.001
0.004
girdle_min_encoded      0.0102      0.003      3.994      0.000       0.005
0.015
girdle_max_encoded     -0.0046      0.003     -1.807      0.071      -0.010
0.000
==============================================================================
Omnibus:                    66667.580   Durbin-Watson:                   1.086
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          4423325.728
Skew:                           1.312   Prob(JB):                         0.00
Kurtosis:                      29.485   Cond. No.                         12.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
P-values for each feature:
const                 1.000000e+00
carat                 0.000000e+00
depth_percent        1.002080e-109
table_percent         4.547285e-60
length                0.000000e+00
width                 2.680653e-02
depth                 2.568777e-08
cut_encoded           7.617413e-74
```

```
color_encoded          0.000000e+00
clarity_encoded        0.000000e+00
symmetry_encoded       6.438556e-07
polish_encoded         6.031770e-03
girdle_min_encoded     6.489660e-05
girdle_max_encoded     7.068782e-02
dtype: float64
```

Encoding for the red-wine and white wine Dataset and Standardization

[ ]:
```
dataset_redwine = dataset1.copy(deep=True)
```

[ ]:
```
dataset_redwine.head()
```

[ ]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

[ ]:
```
X_redwine = dataset_redwine.drop('quality', axis=1)  # Features
y_redwine = dataset_redwine['quality']  # Target

# Standardize features
scaler_redwine = StandardScaler()
X_redwine_standardized = scaler_redwine.fit_transform(X_redwine)

# Split the dataset into training and testing sets
X_train_redwine, X_test_redwine, y_train_redwine, y_test_redwine =
 ↪train_test_split(X_redwine_standardized, y_redwine, test_size=0.2,
 ↪random_state=42)

# Display the shape of the training and testing sets to confirm
```

```
(X_train_redwine.shape, X_test_redwine.shape, y_train_redwine.shape,␣
↪y_test_redwine.shape)
```

[ ]: ((1279, 11), (320, 11), (1279,), (320,))

[ ]:
```
from sklearn.feature_selection import mutual_info_regression, f_regression
mi_scores = mutual_info_regression(X_train_redwine, y_train_redwine)
mi_results = pd.Series(mi_scores, index=X_redwine.columns, name="MI Scores").
↪sort_values(ascending=False)
f_scores, p_values = f_regression(X_train_redwine, y_train_redwine)
f_results = pd.DataFrame({'F Score': f_scores, 'P Value': p_values},␣
↪index=X_redwine.columns).sort_values(by="F Score", ascending=False)
mi_results, f_results.head(), mi_results.nsmallest(2)
```

[ ]:
```
(alcohol                 0.192499
 total sulfur dioxide    0.110696
 volatile acidity        0.108452
 density                 0.093193
 sulphates               0.089162
 citric acid             0.082005
 fixed acidity           0.069736
 free sulfur dioxide     0.026747
 chlorides               0.012182
 residual sugar          0.003836
 pH                      0.000000
 Name: MI Scores, dtype: float64,
                          F Score       P Value
 alcohol               367.395573  3.596579e-72
 volatile acidity      213.369300  8.386233e-45
 sulphates              79.855018  1.376855e-18
 citric acid            62.565621  5.543422e-15
 total sulfur dioxide   53.245305  5.155447e-13,
 pH                0.000000
 residual sugar    0.003836
 Name: MI Scores, dtype: float64)
```

[ ]:
```
mi_scores = mutual_info_regression(X_redwine_standardized, y_redwine)

mi_scores_df = pd.DataFrame({'Feature': X_redwine.columns, 'MI Score':␣
↪mi_scores})

mi_scores_df.sort_values(by='MI Score', ascending=True, inplace=True)

mi_scores_df.head(2)
```

[ ]:
```
               Feature  MI Score
3       residual sugar  0.013788
```

```
5  free sulfur dioxide  0.020575
```

Encoding for the white-wine and white wine Dataset and Standardization

```
[ ]: dataset_whitewine = dataset2.copy(deep=True)
     dataset_whitewine.head()
```

```
[ ]:    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
     0            7.0              0.27         0.36            20.7      0.045
     1            6.3              0.30         0.34             1.6      0.049
     2            8.1              0.28         0.40             6.9      0.050
     3            7.2              0.23         0.32             8.5      0.058
     4            7.2              0.23         0.32             8.5      0.058

        free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
     0                 45.0                 170.0   1.0010  3.00       0.45
     1                 14.0                 132.0   0.9940  3.30       0.49
     2                 30.0                  97.0   0.9951  3.26       0.44
     3                 47.0                 186.0   0.9956  3.19       0.40
     4                 47.0                 186.0   0.9956  3.19       0.40

        alcohol  quality
     0      8.8        6
     1      9.5        6
     2     10.1        6
     3      9.9        6
     4      9.9        6
```

```
[ ]: from sklearn.preprocessing import StandardScaler

     # Separating features and target variable
     features_whitewine = dataset_whitewine.drop('quality', axis=1)
     target_whitewine = dataset_whitewine['quality']

     # Standardizing the features
     scaler = StandardScaler()
     features_whitewine_standardized = scaler.fit_transform(features_whitewine)

     # Convert the standardized features back into a DataFrame for better readability
     features_whitewine_standardized_df = pd.
      ↪DataFrame(features_whitewine_standardized, columns=features_whitewine.
      ↪columns)

     # Display the first few rows of the standardized features to verify
     features_whitewine_standardized_df.head()
```

```
[ ]:    fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
     0       0.172097         -0.081770     0.213280        2.821349  -0.035355
     1      -0.657501          0.215896     0.048001       -0.944765   0.147747
     2       1.475751          0.017452     0.543838        0.100282   0.193523
     3       0.409125         -0.478657    -0.117278        0.415768   0.559727
     4       0.409125         -0.478657    -0.117278        0.415768   0.559727

        free sulfur dioxide  total sulfur dioxide   density        pH  sulphates  \
     0             0.569932              0.744565  2.331512 -1.246921  -0.349184
     1            -1.253019             -0.149685 -0.009154  0.740029   0.001342
     2            -0.312141             -0.973336  0.358665  0.475102  -0.436816
     3             0.687541              1.121091  0.525855  0.011480  -0.787342
     4             0.687541              1.121091  0.525855  0.011480  -0.787342

         alcohol
     0 -1.393152
     1 -0.824276
     2 -0.336667
     3 -0.499203
     4 -0.499203
```

```
[ ]: from sklearn.model_selection import train_test_split

     # Splitting the dataset into training and testing sets
     X_train_whitewine, X_test_whitewine, Y_train_whitewine, Y_test_whitewine =␣
      ↪train_test_split(features_whitewine_standardized, target_whitewine,␣
      ↪test_size=0.2, random_state=42)

     # Output the shape of each set to verify the split
     X_train_whitewine.shape, X_test_whitewine.shape, Y_train_whitewine.shape,␣
      ↪Y_test_whitewine.shape
```

```
[ ]: ((3918, 11), (980, 11), (3918,), (980,))
```

```
[ ]: from sklearn.feature_selection import mutual_info_regression, f_regression

     # Compute Mutual Information between each feature and the target
     mi_scores = mutual_info_regression(X_train_whitewine, Y_train_whitewine)
     mi_results = pd.Series(mi_scores, index=features_whitewine.columns, name="MI␣
      ↪Scores").sort_values(ascending=False)

     # Compute F-scores and p-values for each feature
     f_scores, p_values = f_regression(X_train_whitewine, Y_train_whitewine)
     f_results = pd.DataFrame({'F Score': f_scores, 'P Value': p_values},␣
      ↪index=features_whitewine.columns).sort_values(by="F Score", ascending=False)

     # Display the Mutual Information scores and F-scores for comparison
```

```
mi_results, f_results.head(), mi_results.nsmallest(2)
```

[ ]:
```
(density                 0.194253
 alcohol                 0.153155
 residual sugar          0.099122
 total sulfur dioxide    0.090711
 chlorides               0.071327
 volatile acidity        0.066893
 free sulfur dioxide     0.047704
 citric acid             0.045638
 fixed acidity           0.031136
 sulphates               0.022064
 pH                      0.021452
 Name: MI Scores, dtype: float64,
                          F Score        P Value
 alcohol              896.871903  1.282903e-177
 density              388.880429   1.307999e-82
 volatile acidity     170.490487   3.585223e-38
 chlorides            161.756748   2.424743e-36
 total sulfur dioxide 106.233764   1.347675e-24,
 pH           0.021452
 sulphates    0.022064
 Name: MI Scores, dtype: float64)
```

[ ]:
```
mi_scores = mutual_info_regression(features_whitewine_standardized,␣
 ↪target_whitewine)

mi_scores_df = pd.DataFrame({'Feature': features_whitewine.columns, 'MI Score':␣
 ↪mi_scores})

mi_scores_df.sort_values(by='MI Score', ascending=True, inplace=True)

mi_scores_df.head(2)
```

[ ]:
```
         Feature  MI Score
9      sulphates  0.019798
0  fixed acidity  0.025912
```

[ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import mean_squared_error
import numpy as np

# Split the data into training and testing sets (80% train, 20% test)
X_train_whitewine, X_test_whitewine, y_train_whitewine, y_test_whitewine =␣
 ↪train_test_split(features_whitewine_standardized, target_whitewine,␣
 ↪test_size=0.2, random_state=42)
```

```python
# Initialize the models
ols = LinearRegression()
lasso = Lasso(random_state=42)
ridge = Ridge(random_state=42)

# Train the models
ols.fit(X_train_whitewine, y_train_whitewine)
lasso.fit(X_train_whitewine, y_train_whitewine)
ridge.fit(X_train_whitewine, y_train_whitewine)

# Predict on the testing set
ols_predictions = ols.predict(X_test_whitewine)
lasso_predictions = lasso.predict(X_test_whitewine)
ridge_predictions = ridge.predict(X_test_whitewine)

# Calculate RMSE for each model
ols_rmse = np.sqrt(mean_squared_error(y_test_whitewine, ols_predictions))
lasso_rmse = np.sqrt(mean_squared_error(y_test_whitewine, lasso_predictions))
ridge_rmse = np.sqrt(mean_squared_error(y_test_whitewine, ridge_predictions))

ols_rmse, lasso_rmse, ridge_rmse
```

[ ]: (0.7543373063311435, 0.8806495608493429, 0.7543901241092188)

13 Reprocessing the Diamonds Data to Prepare for Next Parts

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```python
pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.1-py2.py3-none-any.whl (107 kB)
                           107.7/107.7

kB 2.7 MB/s eta 0:00:00
Requirement already satisfied: joblib>=0.11 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.3.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-23.12.0-py3-none-any.whl (23 kB)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.25.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.10/dist-
packages (from scikit-optimize) (1.11.4)
```

```
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (1.2.2)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from scikit-optimize) (24.0)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikit-
optimize) (3.3.0)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-23.12.0 scikit-optimize-0.10.1
```

[ ]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, mutual_info_regression,␣
 ↪f_regression
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import RandomForestRegressor
from tempfile import mkdtemp
from shutil import rmtree
from joblib import Memory
from sklearn.tree import plot_tree
import lightgbm as lgb
from skopt import BayesSearchCV
```

[ ]:
```python
dataset = pd.read_csv('/content/drive/Shareddrives/ECE219/Project4/
 ↪diamonds_ece219.csv')
dataset.head()
```

[ ]:
```
   Unnamed: 0 color clarity  carat        cut   symmetry     polish  \
0           0     E    VVS2   0.09  Excellent  Very Good  Very Good
1           1     E    VVS2   0.09  Very Good  Very Good  Very Good
2           2     E    VVS2   0.09  Excellent  Very Good  Very Good
3           3     E    VVS2   0.09  Excellent  Very Good  Very Good
4           4     E    VVS2   0.09  Very Good  Very Good  Excellent

   depth_percent  table_percent  length  width  depth girdle_min girdle_max  \
0           62.7           59.0    2.85   2.87   1.79          M          M
1           61.9           59.0    2.84   2.89   1.78        STK        STK
2           61.1           59.0    2.88   2.90   1.77         TN          M
3           62.0           59.0    2.86   2.88   1.78          M        STK
```

```
4          64.9          58.5    2.79   2.83   1.82        STK        STK

   price
0    200
1    200
2    200
3    200
4    200
```

[ ]:
```
dataset = dataset.drop(['Unnamed: 0'], axis=1)
print(dataset['color'].unique())
print(dataset['cut'].unique())
print(dataset['clarity'].unique())
print(dataset['symmetry'].unique())
print(dataset['polish'].unique())
print(dataset['girdle_min'].unique())
print(dataset['girdle_max'].unique())
```

```
['E' 'F' 'D' 'J' 'I' 'G' 'H' 'M' 'L' 'K']
['Excellent' 'Very Good']
['VVS2' 'VVS1' 'I1' 'VS1' 'VS2' 'IF' 'SI2' 'I2' 'SI1' 'I3']
['Very Good' 'Excellent']
['Very Good' 'Excellent']
['M' 'STK' 'TN' 'TK' 'unknown' 'VTN' 'XTN' 'VTK' 'STN' 'XTK']
['M' 'STK' 'TK' 'unknown' 'TN' 'VTK' 'VTN' 'XTN' 'STN' 'XTK']
```

[ ]:
```
color_dict = {'M': 1, 'L': 2, 'K': 3, 'J': 4, 'I': 5, 'H': 6, 'G': 7, 'F': 8,
              'E': 9, 'D': 10}
cut_dict = {'Very Good': 1, 'Excellent': 2}
symmetry_dict = {'Very Good': 1, 'Excellent': 2}
polish_dict = {'Very Good': 1, 'Excellent': 2}
#girdle_min_dict = {'unknown': np.nan, 'XTN': 1, 'VTN': 2, 'TN': 3, 'STN': 4,␣
 ↪'M': 5, 'STK': 6, 'TK': 7, 'VTK': 8, 'XTK': 9}
#girdle_max_dict = {'unknown': np.nan, 'XTN': 1, 'VTN': 2, 'TN': 3, 'STN': 4,␣
 ↪'M': 5, 'STK': 6, 'TK': 7, 'VTK': 8, 'XTK': 9}
clarity_dict = {'I3': 1, 'I2': 2, 'I1': 3, 'SI2': 4, 'SI1': 5, 'VS2': 6,
                'VS1': 7, 'VVS2': 8, 'VVS1': 9, 'IF': 10}
```

[ ]:
```
dataset['color_encoded'] = dataset['color'].map(color_dict)
dataset['cut_encoded'] = dataset['cut'].map(cut_dict)
dataset['clarity_encoded'] = dataset['clarity'].map(clarity_dict)
dataset['symmetry_encoded'] = dataset['symmetry'].map(symmetry_dict)
dataset['polish_encoded'] = dataset['polish'].map(polish_dict)
#dataset['girdle_min_encoded'] = dataset['girdle_min'].map(girdle_min_dict)
#dataset['girdle_max_encoded'] = dataset['girdle_max'].map(girdle_max_dict)

dataset = pd.get_dummies(dataset, columns=['girdle_min', 'girdle_max'])
```

```python
dataset = dataset.drop('color', axis=1)
dataset = dataset.drop('cut', axis=1)
dataset = dataset.drop('clarity', axis=1)
dataset = dataset.drop('symmetry', axis=1)
dataset = dataset.drop('polish', axis=1)
#dataset = dataset.drop('girdle_min', axis=1)
#dataset = dataset.drop('girdle_max', axis=1)
```

[ ]:
```python
dataset.head()
```

[ ]:
```
   carat  depth_percent  table_percent  length  width  depth  price  \
0   0.09           62.7           59.0    2.85   2.87   1.79    200
1   0.09           61.9           59.0    2.84   2.89   1.78    200
2   0.09           61.1           59.0    2.88   2.90   1.77    200
3   0.09           62.0           59.0    2.86   2.88   1.78    200
4   0.09           64.9           58.5    2.79   2.83   1.82    200

   color_encoded  cut_encoded  clarity_encoded  …  girdle_max_M  \
0              9            2                8  …             1
1              9            1                8  …             0
2              9            2                8  …             1
3              9            2                8  …             0
4              9            1                8  …             0

   girdle_max_STK  girdle_max_STN  girdle_max_TK  girdle_max_TN  \
0               0               0              0              0
1               1               0              0              0
2               0               0              0              0
3               1               0              0              0
4               1               0              0              0

   girdle_max_VTK  girdle_max_VTN  girdle_max_XTK  girdle_max_XTN  \
0               0               0               0               0
1               0               0               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

   girdle_max_unknown
0                   0
1                   0
2                   0
3                   0
4                   0

[5 rows x 32 columns]
```

[ ]:
```
print(dataset.columns)
```

```
Index(['carat', 'depth_percent', 'table_percent', 'length', 'width', 'depth',
       'price', 'color_encoded', 'cut_encoded', 'clarity_encoded',
       'symmetry_encoded', 'polish_encoded', 'girdle_min_M', 'girdle_min_STK',
       'girdle_min_STN', 'girdle_min_TK', 'girdle_min_TN', 'girdle_min_VTK',
       'girdle_min_VTN', 'girdle_min_XTK', 'girdle_min_XTN',
       'girdle_min_unknown', 'girdle_max_M', 'girdle_max_STK',
       'girdle_max_STN', 'girdle_max_TK', 'girdle_max_TN', 'girdle_max_VTK',
       'girdle_max_VTN', 'girdle_max_XTK', 'girdle_max_XTN',
       'girdle_max_unknown'],
      dtype='object')
```

[ ]:
```
X_unscaled_all_pd = dataset.drop('price', axis=1)
X_unscaled_all = dataset.drop('price', axis=1).to_numpy()
y = dataset['price'].to_numpy()

# Standardize Feature Columns
standard_scaling = StandardScaler()
X_all = standard_scaling.fit_transform(X_unscaled_all)
```

[ ]:
```
print(X_unscaled_all_pd.shape)
print(X_all.shape)
print(y.shape)
```

```
(149871, 31)
(149871, 31)
(149871,)
```

[ ]:
```
selector = SelectKBest(score_func=f_regression, k=9)
X = selector.fit_transform(X_all, y)
selected_columns = selector.get_support()
selected_column_names = X_unscaled_all_pd.columns[selected_columns]
#print(selected_columns)
print("These are the selected features:")
print(list(selected_column_names))
print(X.shape)
```

```
These are the selected features:
['carat', 'table_percent', 'length', 'width', 'depth', 'color_encoded',
'symmetry_encoded', 'polish_encoded', 'girdle_min_TN']
(149871, 9)
```

14 5.1: Polynomial Regression on Diamonds Dataset - Salient Features

We tested out degrees 1, 2, 3, 4, 5, an 6 for polynomial regression. In addition to this, we also tried out various different regularization strengths (alpha values) for the Ridge Regression step: 10.0^(-5), 10.0^(-2), and 10.0^3. For this step, we had to limit cross validation to only 3 folds, in order to allow the Grid Search to finish in a reasonable amount of time.

After doing the Grid Search, we found the most salient features by accessing the coefficients of our trained model. The coefficients with the greatest absolute magnitude were the ones corresponding to the most salient features.

By following this process, we got the most salient features to be ***1) 'carat', 2) 'length', and 3) 'color_encoded'.*** This makes sense, because when we analyzed the dataset at the beginning, we saw that the 'carat' and 'length' features in particular had quite a high correlation with the target variable, price.

```
[ ]: degrees = [1, 2, 3, 4, 5, 6]
     degrees = np.array(degrees)
     alphas = [10.0**-6, 10.0**-5, 10.0**-4, 10.0**-3, 10.0**-2, 10.0**-1, 10.0**0,
               10.0**1, 10.0**2, 10.0**3, 10.0**4, 10.0**5, 10.0**6]
     alphas = np.array(alphas)
```

```
[ ]: !ulimit -Sv unlimited
```

```
[ ]: pipeline = Pipeline([
         #('poly', PolynomialFeatures()),
         ('ridge', Ridge())
     ])

     param_grid = {
         #'poly__degree': degrees,
         'ridge__alpha': alphas
     }

     grid_search0 = GridSearchCV(pipeline, param_grid=param_grid, cv=10,
                                 scoring='neg_root_mean_squared_error', verbose=1,
                                 return_train_score=True, n_jobs=-1)
     grid_search0.fit(X, y)
```

```
Fitting 10 folds for each of 13 candidates, totalling 130 fits
```

```
[ ]: GridSearchCV(cv=10, estimator=Pipeline(steps=[('ridge', Ridge())]), n_jobs=-1,
                  param_grid={'ridge__alpha': array([1.e-06, 1.e-05, 1.e-04, 1.e-03,
     1.e-02, 1.e-01, 1.e+00, 1.e+01,
            1.e+02, 1.e+03, 1.e+04, 1.e+05, 1.e+06])},
                  return_train_score=True, scoring='neg_root_mean_squared_error',
                  verbose=1)
```

[ ]:
```python
grid_search0.best_params_
```

[ ]: {'ridge__alpha': 1e-06}

[ ]:
```python
cachedir = mkdtemp()
memory = Memory(location=cachedir, verbose=10)
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('ridge', Ridge())
],
memory=memory
)

param_grid = {
    'poly__degree': degrees,
    'ridge__alpha': [10.0**-5, 10.0**-2, 10.0**3]
}

grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=3,
                           scoring='neg_root_mean_squared_error', verbose=1,
                           return_train_score=True, n_jobs=1)
grid_search.fit(X, y)
rmtree(cachedir)
```

```
Fitting 2 folds for each of 18 candidates, totalling 36 fits
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=1), array([[-0.558517, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.0s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=1), array([[-1.157106, …,
-0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/e4a29928901844d3faf8d14b2b229ebe
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/009187ae411dc3980585cd7cb9909cd9
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
```

```
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/e4a29928901844d3faf8d14b2b229ebe
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/009187ae411dc3980585cd7cb9909cd9
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(), array([[-0.558517, …, -0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.1s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(), array([[-1.157106, …, -0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.1s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/b5f138865197c45b40fc7a2a40e9e2bb
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/95070208ffe2963b1facf2a536b4933c
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/b5f138865197c45b40fc7a2a40e9e2bb
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/95070208ffe2963b1facf2a536b4933c
___________________________________fit_transform_one cache loaded - 0.0s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=3), array([[-0.558517, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.3s, 0.0min

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=4.636e-17): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T

________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=3), array([[-1.157106, …,
```

```
-0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.3s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/d8a67d13fcf21f7466bb8e40650e59cc
___________________________________fit_transform_one cache loaded - 0.1s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/ed69b700993cca4b02ea88014e974551
___________________________________fit_transform_one cache loaded - 0.1s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/d8a67d13fcf21f7466bb8e40650e59cc
___________________________________fit_transform_one cache loaded - 0.1s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/ed69b700993cca4b02ea88014e974551
___________________________________fit_transform_one cache loaded - 0.1s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=4), array([[-0.558517, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.8s, 0.0min

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=2.90834e-20): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T

________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=4), array([[-1.157106, …,
-0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.8s, 0.0min

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=4.56617e-19): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T

[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/85e61fb4e9de3f7c654c2fa846a095bf
___________________________________fit_transform_one cache loaded - 0.2s, 0.0min

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
```

```
LinAlgWarning: Ill-conditioned matrix (rcond=2.60509e-17): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/3c0e803b9f56f4ffdd8d1a874961782b
___________________________________fit_transform_one cache loaded - 0.2s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/85e61fb4e9de3f7c654c2fa846a095bf
___________________________________fit_transform_one cache loaded - 0.2s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/3c0e803b9f56f4ffdd8d1a874961782b
___________________________________fit_transform_one cache loaded - 0.2s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=5), array([[-0.558517, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 2.5s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=5), array([[-1.157106, …,
-0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 2.3s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/c988099d79f631835127c1b2c544ebb5
___________________________________fit_transform_one cache loaded - 0.5s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/f5f57c962eb19e151de07c5baa0057b1
___________________________________fit_transform_one cache loaded - 0.5s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/c988099d79f631835127c1b2c544ebb5
___________________________________fit_transform_one cache loaded - 0.6s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/f5f57c962eb19e151de07c5baa0057b1
___________________________________fit_transform_one cache loaded - 0.6s, 0.0min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=6), array([[-0.558517, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([ 1284, …, 31996]), None, message_clsname='Pipeline', message=None)
```

```
________________________________________________fit_transform_one - 8.2s, 0.1min
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=6), array([[-1.157106, …,
-0.308668],
       …,
       [-0.558517, …, -0.308668]]),
array([ 200, …, 1284]), None, message_clsname='Pipeline', message=None)
_______________________________________________fit_transform_one - 11.5s, 0.2min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/8da6ebc981fc7d4f81e69b8debf34b87
___________________________________fit_transform_one cache loaded - 1.2s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/c3be5419299a8ee9ca6862466c0a0609
___________________________________fit_transform_one cache loaded - 1.2s, 0.0min
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/8da6ebc981fc7d4f81e69b8debf34b87
___________________________________fit_transform_one cache loaded - 1.4s, 0.0min
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=2.61895e-20): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
[Memory]0.0s, 0.0min    : Loading _fit_transform_one from /tmp/tmp4ieizut7/jobli
b/sklearn/pipeline/_fit_transform_one/c3be5419299a8ee9ca6862466c0a0609
___________________________________fit_transform_one cache loaded - 1.4s, 0.0min
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_ridge.py:216:
LinAlgWarning: Ill-conditioned matrix (rcond=1.42996e-17): result may not be
accurate.
  return linalg.solve(A, Xy, assume_a="pos", overwrite_a=True).T
```

```
________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one…
_fit_transform_one(PolynomialFeatures(degree=1), array([[-1.157106, …,
-0.308668],
       …,
       [ 2.359604, …,  3.239725]]),
array([  200, …, 31996]), None, message_clsname='Pipeline', message=None)
________________________________________________fit_transform_one - 0.0s, 0.0min
```

[ ]:
```
params = grid_search.best_estimator_.get_params()
coefs = params['ridge'].coef_
coefs = coefs[1:]
coefs = np.array(coefs)
coefs = np.abs(coefs)
indices_of_largest = np.argsort(coefs)[-3:]
indices_of_largest = indices_of_largest[::-1]
```

```
print("Most Salient Features:")
print(list(selected_column_names[indices_of_largest]))
print("Absolute magnitudes of the coefficients for these features, respectively:
↪")
print(coefs[indices_of_largest])
```

```
Most Salient Features:
['carat', 'length', 'color_encoded']
```

15 5.2: Polynomial Regression on Diamonds Dataset - Optimal Polynomial Degree

It turned out that for this data, the ***polynomial degree of 1*** combined with a regularization strength of alpha=10^(-5) gave us the best results. We found this by doing a Grid Search which created polynomial regression models using each degree, and then picking out the model that worked the best, giving us the best RMSE value.

The test RMSE of this best estimator was: 3990.240

The train RMSE of this best estimator was: 1186.806

A very high-order polynomial implies that the model was very closely fitted to the training data. This means the training RMSE is a lot better, but such overfitting means worse performance on the testing data. For instance, for the training RMSE, the best model was actually degree 6 and alpha=10^(-5), which gave an RMSE of only 850.631. However, this model gave us an RMSE of 4.86914124e+10 for testing, showing that ***making the degree too high is a clear instance of overfitting, which means great performance on training data but very poor performance on testing data.***

[ ]:
```
print(grid_search.best_estimator_)
print(grid_search.best_params_)
print(grid_search.best_score_)
print(grid_search.cv_results_)
```

```
{'poly__degree': 1, 'ridge__alpha': 1e-05}
{'mean_fit_time': array([7.10432529e-02, 3.55254412e-02, 3.28900814e-02,
1.89299822e-01,
       9.44397449e-02, 9.88012552e-02, 5.08386970e-01, 2.63801932e-01,
       3.61155748e-01, 1.66459429e+00, 1.08093667e+00, 1.14974976e+00,
       3.61193115e+01, 3.40912163e+01, 4.36239743e+00, 2.26480782e+02,
       2.11217645e+02, 1.91937946e+01]), 'std_fit_time': array([7.25126266e-03,
9.93609428e-04, 2.86817551e-04, 5.88822365e-03,
       3.76939774e-04, 3.96668911e-03, 4.37462330e-03, 5.87809086e-03,
       1.63304806e-02, 5.25695086e-02, 1.08761311e-01, 1.27727270e-01,
       3.07183623e-01, 5.41460276e-01, 3.77693653e-01, 1.08504891e-01,
       2.09259748e-01, 8.53215456e-02]), 'mean_score_time': array([0.0072186 ,
0.00664055, 0.00614929, 0.05817068, 0.06089914,
       0.05868816, 0.21980596, 0.20971966, 0.22389185, 0.44472468,
```

```
       0.48095059, 0.46834052, 1.02065122, 0.99785244, 1.00476229,
       2.24632812, 2.2425735 , 2.20093203]), 'std_score_time':
array([1.23739243e-04, 6.28232956e-05, 1.64270401e-04, 2.21741199e-03,
       2.00474262e-03, 1.76525116e-03, 4.38451767e-04, 1.14829540e-02,
       2.84564495e-03, 1.42463446e-02, 2.96151638e-02, 9.02116299e-03,
       8.41319561e-03, 7.94875622e-03, 6.04736805e-03, 4.74858284e-03,
       8.32986832e-03, 9.81783867e-03]), 'param_poly__degree':
masked_array(data=[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_ridge__alpha': masked_array(data=[1e-05, 0.01,
1000.0, 1e-05, 0.01, 1000.0, 1e-05, 0.01,
                   1000.0, 1e-05, 0.01, 1000.0, 1e-05, 0.01, 1000.0,
                   1e-05, 0.01, 1000.0],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'poly__degree': 1, 'ridge__alpha':
1e-05}, {'poly__degree': 1, 'ridge__alpha': 0.01}, {'poly__degree': 1,
'ridge__alpha': 1000.0}, {'poly__degree': 2, 'ridge__alpha': 1e-05},
{'poly__degree': 2, 'ridge__alpha': 0.01}, {'poly__degree': 2, 'ridge__alpha':
1000.0}, {'poly__degree': 3, 'ridge__alpha': 1e-05}, {'poly__degree': 3,
'ridge__alpha': 0.01}, {'poly__degree': 3, 'ridge__alpha': 1000.0},
{'poly__degree': 4, 'ridge__alpha': 1e-05}, {'poly__degree': 4, 'ridge__alpha':
0.01}, {'poly__degree': 4, 'ridge__alpha': 1000.0}, {'poly__degree': 5,
'ridge__alpha': 1e-05}, {'poly__degree': 5, 'ridge__alpha': 0.01},
{'poly__degree': 5, 'ridge__alpha': 1000.0}, {'poly__degree': 6, 'ridge__alpha':
1e-05}, {'poly__degree': 6, 'ridge__alpha': 0.01}, {'poly__degree': 6,
'ridge__alpha': 1000.0}], 'split0_test_score': array([-1.60457462e+03,
-1.60457766e+03, -1.80826269e+03, -1.39592874e+03,
       -1.39581692e+03, -1.18729022e+03, -5.71280154e+04, -5.44912846e+04,
       -3.92188492e+03, -1.40652036e+07, -8.97177367e+05, -6.04698359e+03,
       -1.62728053e+09, -5.64772782e+07, -2.60143101e+05, -4.25827259e+10,
       -2.40643049e+09, -4.29136315e+07]), 'split1_test_score':
array([-6.37590584e+03, -6.37594441e+03, -6.70414913e+03, -1.06604454e+04,
       -1.06028585e+04, -7.86125267e+03, -1.01549761e+06, -2.35953848e+05,
       -6.89895817e+03, -1.86840315e+07, -6.93097032e+06, -1.71192701e+04,
       -5.71984988e+09, -4.39199192e+07, -8.25376869e+05, -5.48000989e+10,
       -3.34653000e+09, -3.78997895e+07]), 'mean_test_score':
array([-3.99024023e+03, -3.99026104e+03, -4.25620591e+03, -6.02818708e+03,
       -5.99933772e+03, -4.52427145e+03, -5.36312812e+05, -1.45222566e+05,
       -5.41042154e+03, -1.63746176e+07, -3.91407384e+06, -1.15831269e+04,
       -3.67356521e+09, -5.01985987e+07, -5.42759985e+05, -4.86914124e+10,
       -2.87648024e+09, -4.04067105e+07]), 'std_test_score':
array([2.38566561e+03, 2.38568337e+03, 2.44794322e+03, 4.63225834e+03,
```

```
       4.60352080e+03, 3.33698123e+03, 4.79184797e+05, 9.07312817e+04,
       1.48853662e+03, 2.30941394e+06, 3.01689647e+06, 5.53614327e+03,
       2.04628468e+09, 6.27867949e+06, 2.82616884e+05, 6.10868648e+09,
       4.70049757e+08, 2.50692100e+06]), 'rank_test_score': array([ 1,  2,  3,
7,  6,  4, 10,  9,  5, 13, 12,  8, 17, 15, 11, 18, 16,
       14], dtype=int32), 'split0_train_score': array([-2227.4281791 ,
-2227.4281791 , -2234.8906328 , -1770.55992643,
       -1770.5599266 , -1782.78693079, -1695.3579875 , -1695.41556348,
       -1714.46444746, -1657.69925387, -1658.97820446, -1676.81287346,
       -1618.23708127, -1624.36598225, -1655.48934216, -1568.20297598,
       -1585.09498018, -1633.0231358 ]), 'split1_train_score':
array([-146.1840258 , -146.18402581, -147.94533034, -139.52344329,
       -139.5235679 , -140.48204584, -138.40603329, -138.50519175,
       -139.87976655, -136.18066651, -136.54568308, -138.81422156,
       -134.69178256, -135.47477747, -138.1826011 , -133.05948837,
       -134.28895075, -137.65376854]), 'mean_train_score':
array([-1186.80610245, -1186.80610245, -1191.41798157,  -955.04168486,
        -955.04174725,  -961.63448832,  -916.8820104 ,  -916.96037761,
        -927.172107  ,  -896.93996019,  -897.76194377,  -907.81354751,
        -876.46443192,  -879.92037986,  -896.83597163,  -850.63123218,
        -859.69196546,  -885.33845217]), 'std_train_score':
array([1040.62207665, 1040.62207665, 1043.47265123,  815.51824157,
        815.51817935,  821.15244248,  778.4759771 ,  778.45518587,
        787.29234046,  760.75929368,  761.21626069,  768.99932595,
        741.77264936,  744.44560239,  758.65337053,  717.5717438 ,
        725.40301471,  747.68468363])}
-3990.2402294336907
```

16 6.1: Neural Network on Diamonds Dataset - Finding Good Hyperparameters

For the hidden layers, we tried various depths (up to 4) and different numbers of total neurons. These are the variants we tried: (10, 20, 30, 40), (10, 20, 30), (10, 20), (10,). Then, for the weight decay regularization we tried the following values: 10**-3, 10**-1, 1, 10, and 1000. For this step, we had to limit cross validation to only 3 folds, in order to allow the Grid Search to finish in a reasonable amount of time.

After running the Grid Search, we found that ***(10, 20, 30) was the best for the neural network's hidden layer sizes, and 10^(-3) was the best regularization value.***

[ ]:
```python
hidden_layers = [(10, 20, 30, 40), (10, 20, 30), (10, 20), (10,)]
nn_alphas = [10**-3, 10**-1, 1, 10, 1000]



cachedir = mkdtemp()
memory = Memory(location=cachedir, verbose=10)
```

```
pipeline_NN = Pipeline([
    ('NN', MLPRegressor(activation='identity'))
],
memory=memory
)

param_grid_NN = {
    'NN__hidden_layer_sizes': hidden_layers,
    'NN__alpha': nn_alphas
}

grid_search_NN = GridSearchCV(pipeline_NN, param_grid=param_grid_NN, cv=3,
                              scoring='neg_root_mean_squared_error', verbose=1,
                              return_train_score=True, n_jobs=-1)
grid_search_NN.fit(X, y)
rmtree(cachedir)
```

```
Fitting 2 folds for each of 20 candidates, totalling 40 fits
```

```
/usr/local/lib/python3.10/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py:686:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

[ ]:
```
print(grid_search_NN.best_params_)
print(grid_search_NN.best_score_)
print(grid_search_NN.cv_results_)
```

```
{'NN__alpha': 0.001, 'NN__hidden_layer_sizes': (10, 20, 30)}
-3328.36553172615
{'mean_fit_time': array([7.10432529e-02, 3.55254412e-02, 3.28900814e-02,
1.89299822e-01,
       9.44397449e-02, 9.88012552e-02, 5.08386970e-01, 2.63801932e-01,
       3.61155748e-01, 1.66459429e+00, 1.08093667e+00, 1.14974976e+00,
       3.61193115e+01, 3.40912163e+01, 4.36239743e+00, 2.26480782e+02,
       2.11217645e+02, 1.91937946e+01]), 'std_fit_time': array([7.25126266e-03,
9.93609428e-04, 2.86817551e-04, 5.88822365e-03,
       3.76939774e-04, 3.96668911e-03, 4.37462330e-03, 5.87809086e-03,
       1.63304806e-02, 5.25695086e-02, 1.08761311e-01, 1.27727270e-01,
       3.07183623e-01, 5.41460276e-01, 3.77693653e-01, 1.08504891e-01,
       2.09259748e-01, 8.53215456e-02]), 'mean_score_time': array([0.0072186 ,
0.00664055, 0.00614929, 0.05817068, 0.06089914,
       0.05868816, 0.21980596, 0.20971966, 0.22389185, 0.44472468,
       0.48095059, 0.46834052, 1.02065122, 0.99785244, 1.00476229,
       2.24632812, 2.2425735 , 2.20093203]), 'std_score_time':
array([1.23739243e-04, 6.28232956e-05, 1.64270401e-04, 2.21741199e-03,
       2.00474262e-03, 1.76525116e-03, 4.38451767e-04, 1.14829540e-02,
```

```
       2.84564495e-03, 1.42463446e-02, 2.96151638e-02, 9.02116299e-03,
       8.41319561e-03, 7.94875622e-03, 6.04736805e-03, 4.74858284e-03,
       8.32986832e-03, 9.81783867e-03]), 'param_poly__degree':
masked_array(data=[1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'param_ridge__alpha': masked_array(data=[1e-05, 0.01,
1000.0, 1e-05, 0.01, 1000.0, 1e-05, 0.01,
                   1000.0, 1e-05, 0.01, 1000.0, 1e-05, 0.01, 1000.0,
                   1e-05, 0.01, 1000.0],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False],
       fill_value='?',
            dtype=object), 'params': [{'poly__degree': 1, 'ridge__alpha':
1e-05}, {'poly__degree': 1, 'ridge__alpha': 0.01}, {'poly__degree': 1,
'ridge__alpha': 1000.0}, {'poly__degree': 2, 'ridge__alpha': 1e-05},
{'poly__degree': 2, 'ridge__alpha': 0.01}, {'poly__degree': 2, 'ridge__alpha':
1000.0}, {'poly__degree': 3, 'ridge__alpha': 1e-05}, {'poly__degree': 3,
'ridge__alpha': 0.01}, {'poly__degree': 3, 'ridge__alpha': 1000.0},
{'poly__degree': 4, 'ridge__alpha': 1e-05}, {'poly__degree': 4, 'ridge__alpha':
0.01}, {'poly__degree': 4, 'ridge__alpha': 1000.0}, {'poly__degree': 5,
'ridge__alpha': 1e-05}, {'poly__degree': 5, 'ridge__alpha': 0.01},
{'poly__degree': 5, 'ridge__alpha': 1000.0}, {'poly__degree': 6, 'ridge__alpha':
1e-05}, {'poly__degree': 6, 'ridge__alpha': 0.01}, {'poly__degree': 6,
'ridge__alpha': 1000.0}], 'split0_test_score': array([-1.60457462e+03,
-1.60457766e+03, -1.80826269e+03, -1.39592874e+03,
       -1.39581692e+03, -1.18729022e+03, -5.71280154e+04, -5.44912846e+04,
       -3.92188492e+03, -1.40652036e+07, -8.97177367e+05, -6.04698359e+03,
       -1.62728053e+09, -5.64772782e+07, -2.60143101e+05, -4.25827259e+10,
       -2.40643049e+09, -4.29136315e+07]), 'split1_test_score':
array([-6.37590584e+03, -6.37594441e+03, -6.70414913e+03, -1.06604454e+04,
       -1.06028585e+04, -7.86125267e+03, -1.01549761e+06, -2.35953848e+05,
       -6.89895817e+03, -1.86840315e+07, -6.93097032e+06, -1.71192701e+04,
       -5.71984988e+09, -4.39199192e+07, -8.25376869e+05, -5.48000989e+10,
       -3.34653000e+09, -3.78997895e+07]), 'mean_test_score':
array([-3.99024023e+03, -3.99026104e+03, -4.25620591e+03, -6.02818708e+03,
       -5.99933772e+03, -4.52427145e+03, -5.36312812e+05, -1.45222566e+05,
       -5.41042154e+03, -1.63746176e+07, -3.91407384e+06, -1.15831269e+04,
       -3.67356521e+09, -5.01985987e+07, -5.42759985e+05, -4.86914124e+10,
       -2.87648024e+09, -4.04067105e+07]), 'std_test_score':
array([2.38566561e+03, 2.38568337e+03, 2.44794322e+03, 4.63225834e+03,
       4.60352080e+03, 3.33698123e+03, 4.79184797e+05, 9.07312817e+04,
       1.48853662e+03, 2.30941394e+06, 3.01689647e+06, 5.53614327e+03,
       2.04628468e+09, 6.27867949e+06, 2.82616884e+05, 6.10868648e+09,
       4.70049757e+08, 2.50692100e+06]), 'rank_test_score': array([ 1,  2,  3,
```

```
7,  6,  4, 10,  9,  5, 13, 12,  8, 17, 15, 11, 18, 16,
       14], dtype=int32), 'split0_train_score': array([-2227.4281791 ,
-2227.4281791 , -2234.8906328 , -1770.55992643,
       -1770.5599266 , -1782.78693079, -1695.3579875 , -1695.41556348,
       -1714.46444746, -1657.69925387, -1658.97820446, -1676.81287346,
       -1618.23708127, -1624.36598225, -1655.48934216, -1568.20297598,
       -1585.09498018, -1633.0231358 ]), 'split1_train_score':
array([-146.1840258 , -146.18402581, -147.94533034, -139.52344329,
       -139.5235679 , -140.48204584, -138.40603329, -138.50519175,
       -139.87976655, -136.18066651, -136.54568308, -138.81422156,
       -134.69178256, -135.47477747, -138.1826011 , -133.05948837,
       -134.28895075, -137.65376854]), 'mean_train_score':
array([-1186.80610245, -1186.80610245, -1191.41798157,  -955.04168486,
        -955.04174725,  -961.63448832,  -916.8820104 ,  -916.96037761,
        -927.172107  ,  -896.93996019,  -897.76194377,  -907.81354751,
        -876.46443192,  -879.92037986,  -896.83597163,  -850.63123218,
        -859.69196546,  -885.33845217]), 'std_train_score':
array([1040.62207665, 1040.62207665, 1043.47265123,  815.51824157,
        815.51817935,  821.15244248,  778.4759771 ,  778.45518587,
        787.29234046,  760.75929368,  761.21626069,  768.99932595,
        741.77264936,  744.44560239,  758.65337053,  717.5717438 ,
        725.40301471,  747.68468363])}
```

17 6.2: Neural Network on Diamonds Dataset - Comparing Against Linear Regression

The test RMSE of the neural network best estimator was: 3328.366

Therefore, the performance is ***BETTER*** than linear regression. This is because the Neural Network is able to capture non-linear relationships in the data much better, whereas linear regression is linear in nature, and less able to capture non-linear relationships. Thus, we can see that the fully connected neural network is the more complex model between the two, and this explains why it performs better. Still, even though linear regression is more simple, this can sometimes be an advantage because the computations are much simpler and clearer, and this can help prevent overfitting and boost speed of training.

18 6.3: Neural Network on Diamonds Dataset - Output Activation Function

For the output, we used ***none (also known as identity)*** for the activation function. The reason is because we are doing a regression task, and our target variable, price, is continuous. We don't want to distort the scale of the output, and we don't want to inappropriately constrict the output to only include a limited range of values. Therefore, we choose to not add an activation funciton at the output. We want to allow the final output to take on any value between negative infinity and positive infinity.

19 6.4: Neural Network on Diamonds Dataset - Danger of Increasing Depth

If we increase the depth of network too far, the neural network model becomes prone to overfitting. This is because with greater depth it has more parameters that need to be tuned, and if we have too many paramters it's easy for the model to adapt to the training data too closely, resulting in good performance on the training data but poor performance on the testing data. Additionally, with more parameters we add to the computational complexity, and so training and retraining will take much longer, and this complexity might add delay when making the predictions as well.

20 7.1: Random Forest on Diamonds Dataset - Hyperparameters: Performance and Regularization Effect

For the maximum number of features, we tried between 1 and 8. Then, for the number of trees we tried the following values: values between 30-150 in increments of 30. For max depth we tried values between 5-17 in increments of 2. For this step, we had to limit cross validation to only 3 folds, in order to allow the Grid Search to finish in a reasonable amount of time.

After running the Grid Search, we found that 3 was the best for the maximum number of features, 120 was the best for number of trees, and 17 was the best for max depth.

The test RMSE of the random forest best estimator was: 3035.469

Maximum ***number of features*** can improve overall performance up to a certain extent, because considering more features in a given tree allows for greater model complexity. Considering more features can help our model be more intricate by taking more data overall into account before making a decision. However, we can't increase it too much because if the model considers too many of the features it is prone to overfitting. For instance, it might end up including features that aren't as relevant to making the decisions. Also, the trees in the random forest will not be as diverse because they will all include a great proportion of the features. Instead, the model would be more generizable if the trees were more diverse, and this is achievable by making the number of features lower. Therefore, keeping this value sufficiently low allows it to have a regularization effect.

Generally, it is good to increase ***number of trees*** because having more trees in the model ensures that our final prediction is more reliable and less affected by chance. However, we don't need to increase this too much because after we surpass a certain point, our results become reliable enough and reliability doesn't improve much by increasing number of trees further. There is not much of a regularization effect generated by this hyperparameter, but we can keep it from getting too high in order to ensure it doesn't encapsulate all the little variations in the training dataset, which we don't care about.

Max depth is good to increase up to a certain point because it makes the tree more complex, and so complex relationships in the data can be better accounted for. Plus, we consider more of the data and look at more of the features/values before making the final decision, and so our result is more precise. If we increase it too much though, the model could be prone to overfitting because it becomes too complex and too closely representative of the training data. Thus, we want to increase this but not too much in order to create a model that is both sufficiently complex and generalizable to unseen data. Therefore, keeping this value sufficiently low allows it to have a regularization effect.

```
[ ]: max_num_features = [i for i in range(1, 9)]
num_of_trees = [i for i in range(30, 180, 30)]
max_depth = [i for i in range(5, 18, 2)]

cachedir = mkdtemp()
memory = Memory(location=cachedir, verbose=10)

pipeline_RF = Pipeline([
    ('RF', RandomForestRegressor(oob_score=True))
],
memory=memory
)

param_grid_RF = {
    'RF__max_features': max_num_features,
    'RF__n_estimators': num_of_trees,
    'RF__max_depth': max_depth
}

grid_search_RF = GridSearchCV(pipeline_RF, param_grid=param_grid_RF, cv=3,
                              scoring='neg_root_mean_squared_error', verbose=1,
                              return_train_score=True, n_jobs=-1)
grid_search_RF.fit(X, y)
rmtree(cachedir)
```

```
Fitting 3 folds for each of 280 candidates, totalling 840 fits
```

```
[ ]: print(grid_search_RF.best_params_)
print(grid_search_RF.best_score_)
print(grid_search_RF.cv_results_['mean_test_score'])
```

21 7.2: Random Forest on Diamonds Dataset - Non-Linear Decision Boundary

Even though all we do at each layer is apply a threshold on a feature, random forests create highly non-linear decision boundaries and thus work fairly well for the regression task. For one, even though we just apply a threshold, we do this multiple times sequentially, and using many different features as well, and this alone allows us to incorporate fairly complex relationships between the features and target variable and create complex decision boundaries. In addition to this, there is a lot of randomness added to the whole process. For instance, the feature selection at each node is randomized by only allowing for a certain subset of features to be selected from. Plus, random forests create many trees and not just one, and each tree is different, and then the results are combined together at the end. Therefore, due to both the overall structure of the model as well as the significant randomness introduced amongst many different trees, highly non-linear decision boundaries can be formed.

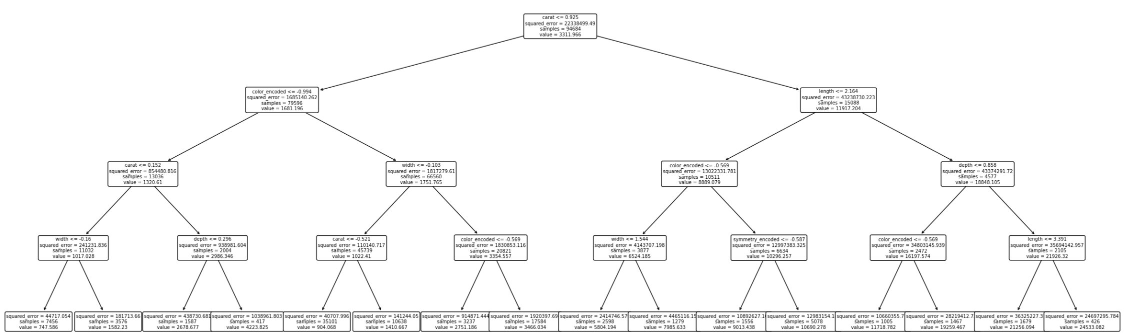
22 7.3: Random Forest on Diamonds Dataset - Visualizing a Tree

The structure of a randomly chosen tree in the random forest model has plotted below. 'carat' is chosen as the feature for branching at the root node. This makes a lot of sense because earlier in the project we showed that 'carat' is a highly important feature in the diamonds dataset because it is closely correlated with the target variable, price. Given that 'carat' is chosen for branching at the root node, we can infer that 'carat' is generally more important than all or most of the other features. Earlier in the project, we showed that the most salient features when doing the polynomial regression were 'carat', 'lenth', and 'color', and all these features can be found at the earlier levels of the tree structure found below, emphasizing their importance.

[ ]:
```
example_rf = RandomForestRegressor(max_depth=4, max_features=3, n_estimators=120)
example_rf.fit(X, y)
```

```
120
```

[ ]:
```
example_tree = example_rf.estimators_[100]
plt.figure(figsize=(30, 10))
plot_tree(example_tree, feature_names=list(selected_column_names), filled=False, fontsize=7, rounded=True)
plt.show()
```



23 7.4: Random Forest on Diamonds Dataset - Out-of-Bag Error

OOB Error is 0.062326425.

OOB error is a metric that tells us how poorly we do when testing decision trees using data samples that they did NOT get to see when they were being trained. Essentially, measuring OOB error helps us figure out how well our random forest model generalizes for unseen data. R2 score is a measure of how well our model can explain changes in the target variable, specifically for unseen data, using the inputted features. An R2 score close to 1 indicates that the predictions of the model are fairly close to the actual values when working with unseen data. On the other hand, an R2 score that's negative or close to 0 indicates that the predictions of the model do not accurately align with the actual values when working with unseen data. In our case, since our R2 score is 0.937673575, we know our random forest model works fairly well on unseen data.

```
[ ]: best_rf = grid_search_RF.best_estimator_
     print("OOB Error:", 1 - best_rf['RF'].oob_score_)
```

```
OOB Error: 0.062326425098034655
```

24 8.1: LightGBM on Diamonds Dataset - Important Hyperparameters and Search Space

After reading the documentation for LightGBM, we determined the important hyperparameters to be: learning_rate, n_estimators, num_leaves, max_depth, subsample, colsample_bytree, reg_alpha, and reg_lambda. The search space is displayed in the code below.

```
[ ]: lgb_model = lgb.LGBMRegressor()
     search_space = {
         'learning_rate': (0.01, 0.1, 'log-uniform'),
         'n_estimators': np.arange(50, 300, 5),
         'num_leaves': np.arange(50, 1000, 100),
         'max_depth': np.arange(1, 200, 20),
         'subsample': np.arange(0.6, 1.0, 0.2),
         'subsample_freq': [1, 2, 3],
         'colsample_bytree': np.arange(0.4, 1.0, 0.2),
         'reg_alpha': [10.0**-2, 10.0**-1, 1.0, 10.0, 100.0],
         'reg_lambda': [10.0**-2, 10.0**-1, 1.0, 10.0, 100.0],
     }
```

25 8.2: LightGBM on Diamonds Dataset - Applying Bayesian Optimization

The ideal hyperparameter combintation is: ('colsample_bytree', 0.6000000000000001), ('learning_rate', 0.1), ('max_depth', 61), ('n_estimators', 250), ('num_leaves', 450), ('reg_alpha', 1.0), ('reg_lambda', 10.0), ('subsample', 0.6), ('subsample_freq', 2) giving us an RMSE of 3924.344

```
[ ]: optimization = BayesSearchCV(
         estimator=lgb_model,
         search_spaces=search_space,
         scoring='neg_root_mean_squared_error',
         cv=2,
         n_iter=50,
         n_jobs=-1,
         verbose=1,
         return_train_score=True
     )
```

```
/usr/local/lib/python3.10/dist-packages/skopt/space/space.py:110: UserWarning:
Dimension array([0.6, 0.8]) was inferred to Real(low=0.6, high=0.8,
prior='uniform', transform='identity'). In upcoming versions of scikit-optimize,
it will be inferred to Categorical(categories=(0.6, 0.8), prior=None). See the
documentation of the check_dimension function for the upcoming API.
  warnings.warn(
```

[ ]:
```
optimization.fit(X, y)
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
```

```
/usr/local/lib/python3.10/dist-packages/skopt/space/space.py:110: UserWarning:
Dimension array([0.6, 0.8]) was inferred to Real(low=0.6, high=0.8,
prior='uniform', transform='identity'). In upcoming versions of scikit-optimize,
it will be inferred to Categorical(categories=(0.6, 0.8), prior=None). See the
documentation of the check_dimension function for the upcoming API.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/skopt/space/space.py:110: UserWarning:
Dimension array([0.6, 0.8]) was inferred to Real(low=0.6, high=0.8,
prior='uniform', transform='identity'). In upcoming versions of scikit-optimize,
it will be inferred to Categorical(categories=(0.6, 0.8), prior=None). See the
documentation of the check_dimension function for the upcoming API.
  warnings.warn(
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
```

```
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
Fitting 2 folds for each of 1 candidates, totalling 2 fits
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.003861 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1122
[LightGBM] [Info] Number of data points in the train set: 149871, number of used
features: 9
[LightGBM] [Info] Start training from score 3303.915487
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[ ]: BayesSearchCV(cv=2, estimator=LGBMRegressor(), n_jobs=-1,
                  return_train_score=True, scoring='neg_root_mean_squared_error',
                  search_spaces={'colsample_bytree': array([0.4, 0.6, 0.8]),
                                 'learning_rate': (0.01, 0.1, 'log-uniform'),
                                 'max_depth': array([  1,  21,  41,  61,  81, 101,
121, 141, 161, 181]),
                                 'n_estimators': array([ 50,  55,  60,  65,  70,
75,  80,  85,  90,  95, 100, 105, 110,
       115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175,
       180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240,
       245, 250, 255, 260, 265, 270, 275, 280, 285, 290, 295]),
                                 'num_leaves': array([ 50, 150, 250, 350, 450, 550,
650, 750, 850, 950]),
                                 'reg_alpha': [0.01, 0.1, 1.0, 10.0, 100.0],
                                 'reg_lambda': [0.01, 0.1, 1.0, 10.0, 100.0],
                                 'subsample': array([0.6, 0.8]),
                                 'subsample_freq': [1, 2, 3]},
                  verbose=1)
```

```
[ ]: print(optimization.best_params_)
     print(optimization.best_score_)
```

```
OrderedDict([('colsample_bytree', 0.6000000000000001), ('learning_rate', 0.1),
('max_depth', 61), ('n_estimators', 250), ('num_leaves', 450), ('reg_alpha',
1.0), ('reg_lambda', 10.0), ('subsample', 0.6), ('subsample_freq', 2)])
-3924.344586055418
```

26 8.3: LightGBM on Diamonds Dataset - Effect of Hyperparameters

The learning rate helps with performance by controlling the step size of gradient descent. It is also related to fitting efficiency because a smaller learning rate may take longer to train because it will take longer for convergence to occur. Number of estimators improves performance by getting rid of noise by adding more trees, each with a different structure. But we should reduce this number of estimators number if we want to improve fitting efficiency. The num leaves and max depth parameters affect performance because it determines how many leaves and layers to include in each tree, where more leaves and layers means more complexity, thus they also act as a regularization terms too because you need to make sure you don't increase it too much. Plus, if we increase these two hyperparameters too much, complexity gets so large that fitting efficiency diminishes as well. subsample and colsample_bytree relate to how much of the data to use when training and which features to use when training a particular tree. Therefore, it's important to get both to an appropriate value where they aren't incorporating too much of the training data and training features so that the model is able to generalize well. However, we want to include just enough to make performance good by adding sufficient complexity. Thus, here we see another trade off between the performance and regularization. Reg_alpha and reg_lambda both help with regularization, making sure that we don't overfit by using L1 and L2 regularization terms.

tweet_data_project

March 19, 2024

1 Twitter Data Project

Group Members: Zan Xie (UID: 205364923), Joseph Gong (UID: 606073799), Anuk Fernando (UID: 805423707)

[91]:
```
# library import
from google.colab import drive
drive.mount('/content/drive')

import json
import datetime
from collections import defaultdict
import matplotlib.pyplot as plt

# text cleaning
import pytz
import re
import nltk
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.corpus import wordnet
lemmatizer = nltk.stem.WordNetLemmatizer()
wordnet_lemmatizer = WordNetLemmatizer()

# feature extraction
import pytz
import pandas as pd
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# baseline model
import numpy as np
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.metrics import classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer

# LSTM model
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Input, Dense, Dropout, Conv1D, MaxPooling1D, Flatten
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package vader_lexicon to /root/nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!
```

#Q9.1

[2]:
```
# hashtag summary function

folder_path = '/content/drive/Shareddrives/ECE219/Project4/twitter_data/'
tweet_file = ['tweets_#gohawks.txt', 'tweets_#gopatriots.txt', 'tweets_#nfl.
 ↪txt', 'tweets_#patriots.txt', 'tweets_#sb49.txt', 'tweets_#superbowl.txt']
tweet_hashtag = ['#gohawks', '#gopatriots', '#nfl', '#patriots', '#sb49',␣
 ↪'#superbowl']

# open hashtag file, return tweet summary
def hashtag_summary(file_name):
  # initi variables
  time_list = []
  tweet_total = 0
  follower_total = 0
  retweet_total = 0

  with open(folder_path + file_name, 'r') as file:
    for line in file:
```

```
      json_object = json.loads(line)
      tweet_total += 1
      follower_total += json_object['author']['followers']
      retweet_total += json_object['metrics']['citations']['total']
      unix_time = json_object['citation_date']
      time_list.append(unix_time)
    file.close()

  # total hours
  time_start = min(time_list)
  time_end = max(time_list)
  time_total = (time_end - time_start) / 3600

  # summary
  tweet_avg = tweet_total / time_total
  follower_avg = follower_total / tweet_total
  retweet_avg = retweet_total / tweet_total

  return tweet_avg, follower_avg, retweet_avg
```

[ ]:
```
# implement
# run time 5 mins
for file_name, hashtag_name in zip(tweet_file, tweet_hashtag):
  # iterate over hashtag files
  tweet_avg, follower_avg, retweet_avg = hashtag_summary(file_name)
  print('Given hashtag:', hashtag_name)
  print('Average number of tweets per hour:', tweet_avg)
  print('Average number of followers of users posting the tweets per tweet:',␣
 ↪follower_avg)
  print('Average number of retweets per tweet:', retweet_avg)
  print('-'*20)
```

```
Given hashtag: #gohawks
Average number of tweets per hour: 292.48785062173687
Average number of followers of users posting the tweets per tweet:
2217.9237355281984
Average number of retweets per tweet: 2.0132093991319877
--------------------
Given hashtag: #gopatriots
Average number of tweets per hour: 40.95469800606194
Average number of followers of users posting the tweets per tweet:
1427.2526051635405
Average number of retweets per tweet: 1.4081919101697078
--------------------
Given hashtag: #nfl
Average number of tweets per hour: 397.0213901819841
Average number of followers of users posting the tweets per tweet:
```

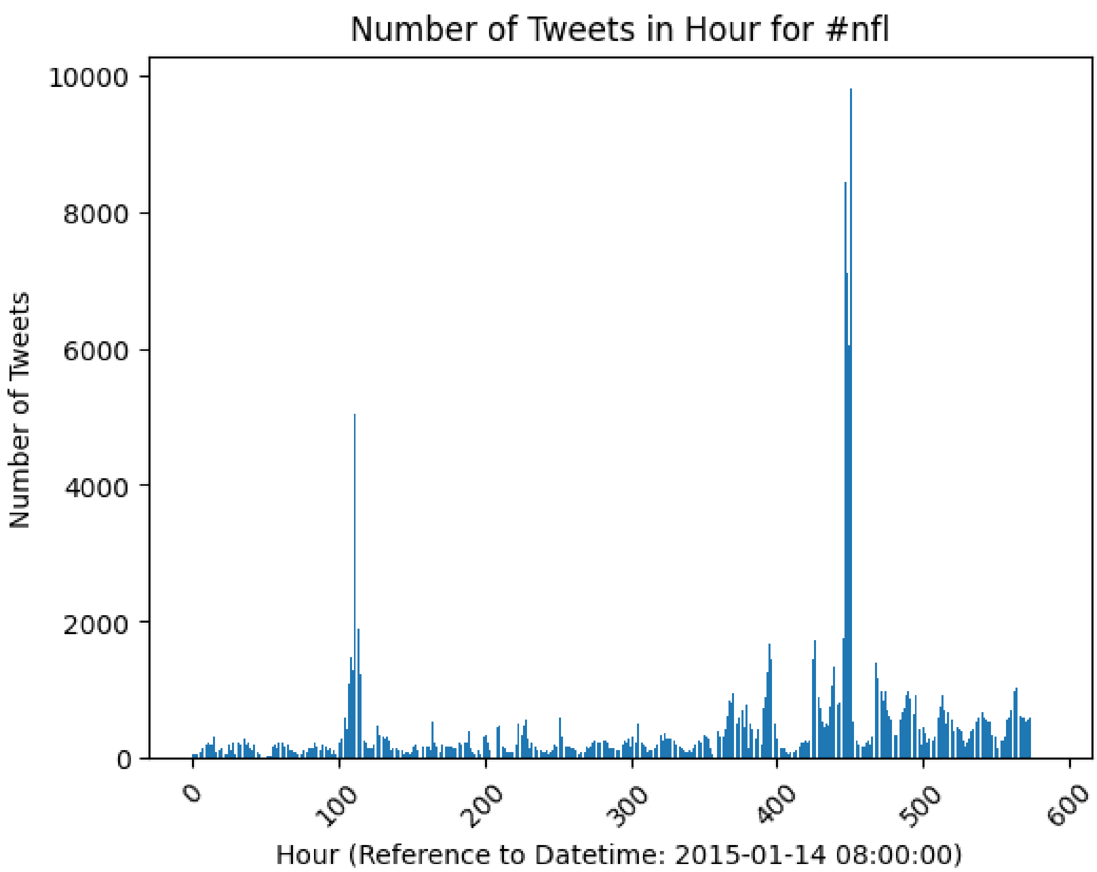
```
4662.37544523693
Average number of retweets per tweet: 1.5344602655543254
--------------------
Given hashtag: #patriots
Average number of tweets per hour: 750.89426460689
Average number of followers of users posting the tweets per tweet:
3280.4635616550277
Average number of retweets per tweet: 1.7852871288476946
--------------------
Given hashtag: #sb49
Average number of tweets per hour: 1276.8570598680474
Average number of followers of users posting the tweets per tweet:
10374.160292019487
Average number of retweets per tweet: 2.52713444111402
--------------------
Given hashtag: #superbowl
Average number of tweets per hour: 2072.11840170408
Average number of followers of users posting the tweets per tweet:
8814.96799424623
Average number of retweets per tweet: 2.3911895819207736
--------------------
```

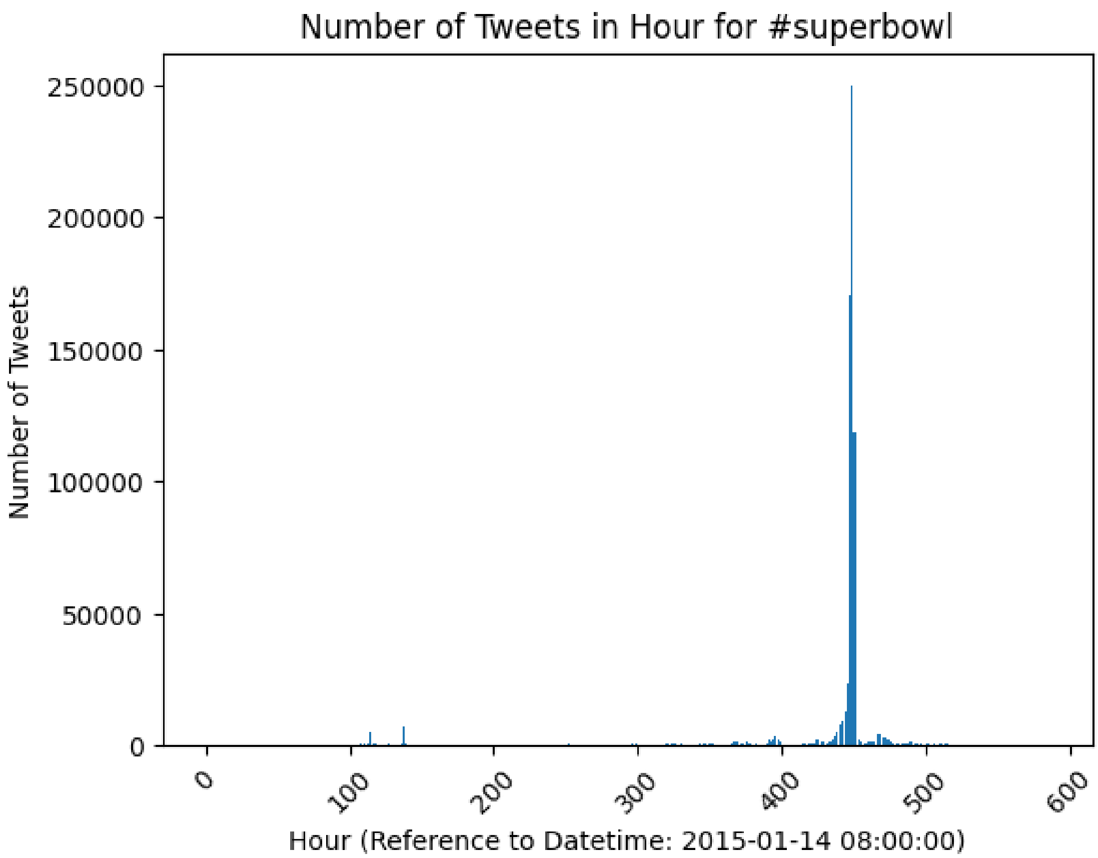
#Q9.2

[3]:
```
# Plot "number of tweets in hour" over time for #SuperBowl and #NFL

def tweet_in_hour(file_name):
  # initi variables
  time_list = []
  tweets_per_hour = defaultdict(int)

  with open(folder_path + file_name, 'r') as file:
    for line in file:
      json_object = json.loads(line)
      unix_time = json_object['citation_date']
      # reserve only hour information
      dt = datetime.datetime.fromtimestamp(unix_time).replace(minute=0,␣
 ↪second=0, microsecond=0)
      unix_time = int(dt.timestamp())
      time_list.append(unix_time)
    file.close()

  # get the earliest unix_time for reference
  time_start = min(time_list)
  datetime_start = datetime.datetime.fromtimestamp(time_start)

  # tweets time with respect to the reference
  time_list_ref = [((x - time_start) / 3600) for x in time_list]
```

```python
  for item in time_list_ref:
    tweets_per_hour[item] += 1

  return tweets_per_hour, str(datetime_start)
```

[ ]:
```python
# Plot nfl and superbowl

compare_file = ['tweets_#nfl.txt', 'tweets_#superbowl.txt']
compare_hashtag = ['#nfl', '#superbowl']

for file_name, hashtag_name in zip(compare_file, compare_hashtag):
  tweets_per_hour, datetime_start = tweet_in_hour(file_name)

  plt.bar(tweets_per_hour.keys(), tweets_per_hour.values())
  plt.xlabel(f'Hour (Reference to Datetime: {datetime_start})')
  plt.ylabel('Number of Tweets')
  plt.title(f'Number of Tweets in Hour for {hashtag_name}')
  plt.xticks(rotation=45)
  plt.show()
  print('\n')
```





#Q10

2 Time-Series Correlation between Scores and Tweets

3 Task Description

We analyze a dataset of tweets with timestamps to correlate them with football game scores over time, in this case, the Super Bowl. The aim is to create a model that, given a tweet, can predict the team that is currently winning or potentially the score.

4 Feature Engineering Process

1. Dataset: Randomly select 12138 samples from #superbowl file which has 1213813 samples. Construct a dataframe with corresponding datetime in PST time zone, follower_count, and retweet_count. Truncate the original dataset is helpful in reducing process overhead.
2. Text Preprocess: Clean and lemmatize tweets_title. Then, apply sentiment analysis on lemm_clean_text.
3. Sentiment Analysis: TextBlob and VADER(SentimentIntensityAnalyzer) are common tools for the purpose. I chose VADER since VADER is specifically designed for sentiment analysis in social media texts.
4. Advanced Text Representation: Beyond TF-IDF, consider using word embeddings like GloVe. These can capture semantic relationships between words better.

[4]:
```
# time zone
pst_tz = pytz.timezone('America/Los_Angeles')

# data path
folder_path = '/content/drive/Shareddrives/ECE219/Project4/twitter_data/'
tweet_file = ['tweets_#gohawks.txt', 'tweets_#gopatriots.txt', 'tweets_#nfl.
 ↪txt', 'tweets_#patriots.txt', 'tweets_#sb49.txt', 'tweets_#superbowl.txt']
tweet_hashtag = ['#gohawks', '#gopatriots', '#nfl', '#patriots', '#sb49',␣
 ↪'#superbowl']
superbowl_file_path = folder_path + tweet_file[5]

# retrive data feaurs
with open(superbowl_file_path, 'r') as f:
  line = f.readline()
  json_object = json.loads(line)
  data_features = list(json_object.keys())
  f.close()

print(data_features)

# Print the content of each feature
print("\nContents of each feature:")
for feature in data_features:
    print(f"{feature}: {json_object[feature]}")
```

```
['firstpost_date', 'title', 'url', 'tweet', 'author', 'original_author',
'citation_date', 'metrics', 'highlight', 'type', 'citation_url']

Contents of each feature:
firstpost_date: 1419883838
title: At http://t.co/Vd0RWOeAed -- #Seahawks #12thMAN #12 #SeahawkNation
#SuperBowlBound #Superbowl #Repeat #GoHawks … http://t.co/XSEFUKqEhN
url: http://twitter.com/HawksNationYes/status/549658771749101568
tweet: {'contributors': None, 'truncated': False, 'text': 'At
http://t.co/Vd0RWOeAed -- #Seahawks #12thMAN #12 #SeahawkNation #SuperBowlBound
```

#Superbowl #Repeat #GoHawks … http://t.co/XSEFUKqEhN',
'in_reply_to_status_id': None, 'id': 549658771749101568, 'favorite_count': 0,
'source': 'IFTTT', 'retweeted':
False, 'coordinates': None, 'timestamp_ms': '1419883838008', 'entities':
{'symbols': [], 'media': [{'expanded_url':
'http://twitter.com/HawksNationYes/status/549658771749101568/photo/1', 'sizes':
{'large': {'h': 640, 'resize': 'fit', 'w': 640}, 'small': {'h': 340, 'resize':
'fit', 'w': 340}, 'medium': {'h': 600, 'resize': 'fit', 'w': 600}, 'thumb':
{'h': 150, 'resize': 'crop', 'w': 150}}, 'url': 'http://t.co/XSEFUKqEhN',
'media_url_https': 'https://pbs.twimg.com/media/B6DHvZfIcAErwQ8.jpg', 'id_str':
'549658771648442369', 'indices': [115, 137], 'media_url':
'http://pbs.twimg.com/media/B6DHvZfIcAErwQ8.jpg', 'type': 'photo', 'id':
549658771648442369, 'display_url': 'pic.twitter.com/XSEFUKqEhN'}], 'hashtags':
[{'indices': [29, 38], 'text': 'Seahawks'}, {'indices': [39, 47], 'text':
'12thMAN'}, {'indices': [52, 66], 'text': 'SeahawkNation'}, {'indices': [67,
82], 'text': 'SuperBowlBound'}, {'indices': [83, 93], 'text': 'Superbowl'},
{'indices': [94, 101], 'text': 'Repeat'}, {'indices': [102, 110], 'text':
'GoHawks'}], 'user_mentions': [], 'trends': [], 'urls': [{'indices': [3, 25],
'url': 'http://t.co/Vd0RW0eAed', 'expanded_url': 'http://iqboom.com/seahawks',
'display_url': 'iqboom.com/seahawks'}]}, 'in_reply_to_screen_name': None,
'in_reply_to_user_id': None, 'retweet_count': 0, 'id_str': '549658771749101568',
'favorited': False, 'user': {'follow_request_sent': None,
'profile_use_background_image': True, 'geo_enabled': False, 'description':
'#seahawks', 'verified': False, 'profile_image_url_https':
'https://pbs.twimg.com/profile_images/450368952107950080/DYOcKrlw_normal.jpeg',
'profile_sidebar_fill_color': 'DDEEF6', 'is_translator': False, 'id':
2419495681, 'profile_text_color': '333333', 'followers_count': 811,
'profile_sidebar_border_color': 'C0DEED', 'id_str': '2419495681',
'default_profile_image': False, 'location': '', 'utc_offset': None,
'statuses_count': 82505, 'profile_background_color': 'C0DEED', 'friends_count':
17, 'profile_link_color': '0084B4', 'profile_image_url':
'http://pbs.twimg.com/profile_images/450368952107950080/DYOcKrlw_normal.jpeg',
'notifications': None, 'profile_background_image_url_https':
'https://abs.twimg.com/images/themes/theme1/bg.png',
'profile_background_image_url':
'http://abs.twimg.com/images/themes/theme1/bg.png', 'name': 'Hawks Nation, Yes',
'lang': 'en', 'profile_background_tile': False, 'favourites_count': 0,
'screen_name': 'HawksNationYes', 'url': 'http://hawksnationyes.tumblr.com',
'created_at': 'Sun Mar 30 20:28:01 +0000 2014', 'contributors_enabled': False,
'time_zone': None, 'protected': False, 'default_profile': True, 'following':
None, 'listed_count': 21}, 'geo': None, 'in_reply_to_user_id_str': None,
'possibly_sensitive': False, 'lang': 'und', 'created_at': 'Mon Dec 29 20:10:38
+0000 2014', 'filter_level': 'medium', 'in_reply_to_status_id_str': None,
'place': None, 'extended_entities': {'media': [{'expanded_url':
'http://twitter.com/HawksNationYes/status/549658771749101568/photo/1', 'sizes':
{'large': {'h': 640, 'resize': 'fit', 'w': 640}, 'small': {'h': 340, 'resize':
'fit', 'w': 340}, 'medium': {'h': 600, 'resize': 'fit', 'w': 600}, 'thumb':
{'h': 150, 'resize': 'crop', 'w': 150}}, 'url': 'http://t.co/XSEFUKqEhN',

'media_url_https': 'https://pbs.twimg.com/media/B6DHvZfIcAErwQ8.jpg', 'id_str':
'549658771648442369', 'indices': [115, 137], 'media_url':
'http://pbs.twimg.com/media/B6DHvZfIcAErwQ8.jpg', 'type': 'photo', 'id':
549658771648442369, 'display_url': 'pic.twitter.com/XSEFUKqEhN'}]}}
author: {'author_img':
'http://pbs.twimg.com/profile_images/508736487706206208/PHzhMVOj_normal.jpeg',
'name': 'Becca Delgado', 'url': 'http://twitter.com/beccadelgado67', 'nick':
'beccadelgado67', 'followers': 22.0, 'image_url':
'http://pbs.twimg.com/profile_images/508736487706206208/PHzhMVOj_normal.jpeg',
'type': 'twitter', 'description': 'Faith. Family. Football.'}
original_author: {'author_img':
'http://pbs.twimg.com/profile_images/450368952107950080/DYOcKrlw_normal.jpeg',
'description': '#seahawks', 'url': 'http://twitter.com/hawksnationyes', 'nick':
'hawksnationyes', 'followers': 1175.0, 'image_url':
'http://pbs.twimg.com/profile_images/450368952107950080/DYOcKrlw_normal.jpeg',
'type': 'twitter', 'name': 'Hawks Nation, Yes'}
citation_date: 1421468497
metrics: {'acceleration': 0, 'ranking_score': 3.2292066, 'citations':
{'influential': 0, 'total': 2, 'data': [{'timestamp': 1421468459, 'citations':
0}], 'matching': 1, 'replies': 0}, 'peak': 0, 'impressions': 5, 'momentum': 0}
highlight: At http://t.co/Vd0RWOeAed -- #Seahawks #12thMAN #12 #SeahawkNation
#SuperBowlBound #Superbowl #Repeat #GoHawks … http://t.co/XSEFUKqEhN
type: retweet:native
citation_url: http://twitter.com/BeccaDelgado67/status/556305315676037120

[5]:
```python
# GloVe Word Embedding
def load_glove_embeddings(glove_file):
  embeddings_dict = {}
  with open(glove_file, 'r', encoding='utf8') as f:
    for line in f:
      values = line.split()
      word = values[0]
      vector = np.asarray(values[1:], "float32")
      embeddings_dict[word] = vector
  return embeddings_dict

# loading the 300-dimensional GloVe embeddings
glove_embeddings = load_glove_embeddings('/content/drive/Shareddrives/ECE219/
 ↪Project4/GloVe/glove.6B.300d.txt')

# Ensure this dictionary is passed to the function correctly
def text_to_embedding(text, embeddings_dict):
  words = text.split()  # Assuming text is already preprocessed and␣
 ↪space-separated
  embeddings = [embeddings_dict.get(word, np.zeros(300)) for word in words]  #␣
 ↪Adjust 100 to your GloVe dimension
  if embeddings:
```

```
    return np.mean(embeddings, axis=0)
  else:
    return np.zeros(300)  # Adjust 100 to match your GloVe dimension
```

[6]:

```
# text cleaning process
lemmatizer = nltk.stem.WordNetLemmatizer()
wordnet_lemmatizer = WordNetLemmatizer()

def clean(text):
  text = re.sub(r'^https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE)
  texter = re.sub(r"<br />", " ", text)
  texter = re.sub(r"&quot;", "\"",texter)
  texter = re.sub('&#39;', "\"", texter)
  texter = re.sub('\n', " ", texter)
  texter = re.sub(' u '," you ", texter)
  texter = re.sub('`',"", texter)
  texter = re.sub(' +', ' ', texter)
  texter = re.sub(r"(!)\1+", r"!", texter)
  texter = re.sub(r"(\?)\1+", r"?", texter)
  texter = re.sub('&amp;', 'and', texter)
  texter = re.sub('\r', ' ',texter)
  texter = re.sub(r'\d+', '', texter) # exclude numbers
  texter = re.sub('[^a-zA-Z0-9\n]', ' ', texter) # Replace characters A-Za-z0-9␣
↪and decimal
  texter = re.sub('\s+',' ', texter) # Removing whitespace and newlines
  texter = texter.lower() # convert to lower case
  clean = re.compile('<.*?>')
  texter = texter.encode('ascii', 'ignore').decode('ascii')
  texter = re.sub(clean, '', texter)
  if texter == "":
      texter = ""
  return texter

#lemmatization
def nltk_tag_to_wordnet_tag(nltk_tag):
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None

def lemmatize_sentence(sentence):
```

```
    #tokenize the sentence and find the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])),␣
↪nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
            #if there is no available tag, append the token as is
            lemmatized_sentence.append(word)
        else:
            #else use the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)
```

[188]:

```
# tweets dataframe and save to file
# skip this step if the file has beeen saved
# read dataframe in the next code block

# initialize variables
title_list = []
time_list = []
follower_list = []
retweet_list = []
af_list = []
len_title_list = []
unix_time_list = []

# user post count
tweet_file = open(superbowl_file_path, 'r')
user_counts = {}
for line in tweet_file:
  json_object = json.loads(line)
  author = json_object['author']['name']
  if author in user_counts:
    user_counts[author] += 1
  else:
    user_counts[author] = 1

with open(superbowl_file_path, 'r') as f:
  for line in f:
    json_object = json.loads(line)
    title = json_object['title']
    follower = json_object['author']['followers']
    retweet = json_object['metrics']['citations']['total']
    unix_time = json_object['citation_date']
    datatime = datetime.datetime.fromtimestamp(unix_time, pst_tz)
```

```
    author = json_object['author']['name']
    len_title = len(title.split())

    # construct dataframe wrt current tweet
    title_list.append(title)
    time_list.append(datatime)
    follower_list.append(follower)
    retweet_list.append(retweet)
    af_list.append(user_counts[author])
    len_title_list.append(len_title)
    unix_time_list.append(unix_time)
  f.close()

# construct the dataframe
df_tweets = pd.DataFrame({
    'title': title_list,
    'unix time': unix_time_list,
    'timestamp': time_list,
    'follower_count': follower_list,
    'retweet_count': retweet_list,
    'active factor': af_list,
    'length title': len_title_list
})

# save the dataframe to file
df_tweets.to_pickle('/content/drive/Shareddrives/ECE219/Project4/
↪tweet_saved_file/df_tweet.pkl')
```

[7]:
```
# read the dataframe
df_tweets = pd.read_pickle('/content/drive/Shareddrives/ECE219/Project4/
↪tweet_saved_file/df_tweet.pkl')

# Print the first few rows of the DataFrame
display(df_tweets)
```

```
                                                    title   unix time  \
0        At http://t.co/Vd0RWOeAed -- #Seahawks #12thMA…  1421468497
1        You been 12ed pass it on #SeahawkNation #LOB #…  1421467579
2        27 days to the SuperBowl  \n#Katyperry #KatyC…  1421266957
3        Check out the cool event that  #budlight has p…  1421261298
4        Lenny Kravitz acompañará a Katy Perry en el #H…  1421316031
…                                                    …           …
1213808  Look at this crazy map of all the private jets…  1423328580
1213809  Where to start making money online for beginne…  1423330066
1213810  Still in superbowl mode  \n#SB49 #superbowl …  1423330744
1213811  @pscgt2015 Nice, Debbie! @futieton #SB49 #phx …  1423331367
1213812  Are you still celebrating the #SuperBowl win? …  1423332008
```

```
                        timestamp  follower_count  retweet_count  \
0       2015-01-16 20:21:37-08:00            22.0              2
1       2015-01-16 20:06:19-08:00            22.0             15
2       2015-01-14 12:22:37-08:00           858.0              2
3       2015-01-14 10:48:18-08:00         14335.0              2
4       2015-01-15 02:00:31-08:00          1143.0              7
…                             …               …             …
1213808 2015-02-07 09:03:00-08:00           297.0              1
1213809 2015-02-07 09:27:46-08:00          1759.0              1
1213810 2015-02-07 09:39:04-08:00         69827.0              5
1213811 2015-02-07 09:49:27-08:00          1085.0              1
1213812 2015-02-07 10:00:08-08:00         43330.0              4

         active factor  length title
0                    2            13
1                    2            14
2                    1            12
3                    1            14
4                    1            12
…                  …            …
1213808             38            19
1213809             46            13
1213810             13            13
1213811              3             9
1213812             20            10

[1213813 rows x 7 columns]
```

[8]:
```
# Scoring events
# if New England Patriots winning, label=1
# if Seattle Seahawks winning, label=0
# if even score, whoever wins that score goes to his label
# default score label=0, since Seattle Seahawks is the 'home' team

# Game start time, year-month-day-hour-min, time_zone
game_start = pst_tz.localize(datetime.datetime(2015, 2, 1, 15, 30))

# Function to calculate event time
def event_time(quarter, minutes, seconds):
    quarter_lengths = 15  # 15 minutes per quarter
    elapsed_since_start = datetime.timedelta(minutes=(quarter - 1) *
 ↪quarter_lengths)
    time_to_event = datetime.timedelta(minutes=(quarter_lengths - minutes),
 ↪seconds=(60 - seconds))
    event_datetime = game_start + elapsed_since_start + time_to_event
    return event_datetime
```

```
scoring_data = [
    (event_time(2, 9, 47), "New England Patriots", "7-0", 1),
    (event_time(2, 2, 16), "Seattle Seahawks", "7-7", 0),
    (event_time(2, 0, 31), "New England Patriots", "14-7", 1),
    (event_time(2, 0, 2), "Seattle Seahawks", "14-14", 0),
    (event_time(3, 11, 9), "Seattle Seahawks", "14-17", 0),
    (event_time(3, 4, 54), "Seattle Seahawks", "14-24", 0),
    (event_time(4, 7, 55), "New England Patriots", "21-24", 0),
    (event_time(4, 2, 2), "New England Patriots", "28-24", 1),
]

# Create DataFrame
df_scoring = pd.DataFrame(scoring_data, columns=['Event Time', 'Scoring Team',␣
↪'Score', 'Winning Label'])
print(df_scoring)
```

```
                 Event Time          Scoring Team  Score  Winning Label
0 2015-02-01 15:51:13-08:00  New England Patriots    7-0              1
1 2015-02-01 15:58:44-08:00      Seattle Seahawks    7-7              0
2 2015-02-01 16:00:29-08:00  New England Patriots   14-7              1
3 2015-02-01 16:00:58-08:00      Seattle Seahawks  14-14              0
4 2015-02-01 16:04:51-08:00      Seattle Seahawks  14-17              0
5 2015-02-01 16:11:06-08:00      Seattle Seahawks  14-24              0
6 2015-02-01 16:23:05-08:00  New England Patriots  21-24              0
7 2015-02-01 16:28:58-08:00  New England Patriots  28-24              1
```

[9]:
```
# truncate the dataset down to 12138 samples (100 times less samples)
# initialize empty columns for sentiment score and winning label

df_tweets_select = df_tweets.sample(n=12138, random_state=42)
df_tweets_select = df_tweets_select.sort_index()
df_tweets_select['clean text'] = None
df_tweets_select['sentiment score'] = None
df_tweets_select['winning label'] = None
sia = SentimentIntensityAnalyzer()

# Iterate through the df_tweets_select DataFrame
for i, row in df_tweets_select.iterrows():
  # Filter df_scoring to include events before the post's timestamp
  relevant_events = df_scoring[df_scoring['Event Time'] <= row['timestamp']]

  # clean text and sentiment score
  clean_text = clean(row['title'])
  lemm_clean_text = lemmatize_sentence(clean_text)
  sentiment_socre = sia.polarity_scores(lemm_clean_text)
```

```
  # The most recent scoring event append to dataframe
  if relevant_events.empty:
    df_tweets_select.at[i, 'winning label'] = 0
  else:
    recent_event = relevant_events.iloc[-1]
    df_tweets_select.at[i, 'winning label'] = int(recent_event['Winning Label'])

  df_tweets_select.at[i, 'sentiment score'] = sentiment_socre['compound']
  df_tweets_select.at[i, 'clean text'] = lemm_clean_text

# print the selected tweet dataframe
display(df_tweets_select)
```

```
                                                     title   unix time  \
31       Just a #SuperBowlChampion  #Pedestrian wide re…  1421224793
276      Gotta love that the entire city of #Seattle is…  1421261323
386      RT @liquidityinc: This might be ingenious, @cj…  1421269215
472      During #SB49, block on 1 St. b/w Jefferson &am…  1421275030
535      Impress your #SuperBowlXLIX guests and enjoy w…  1421282326
…                                                    …         …
1213483    Omg   @5SOS #SuperBowl http://t.co/ldsVSI7ncv  1423286621
1213607    #tevejonavic #letsclub #superbowl #katyper…  1423289803
1213696  .@MissyElliott cried tears of joy after her #S…  1423293311
1213750  #superbowl #seahawks #12thman Get your Superbo…  1423294785
1213804  Expert dating and relationships advice for men…  1423322132

                         timestamp  follower_count  retweet_count  \
31       2015-01-14 00:39:53-08:00           142.0              1
276      2015-01-14 10:48:43-08:00          4310.0              1
386      2015-01-14 13:00:15-08:00         11420.0              1
472      2015-01-14 14:37:10-08:00           515.0              2
535      2015-01-14 16:38:46-08:00          3995.0              2
…                              …              …             …
1213483  2015-02-06 21:23:41-08:00          2414.0              1
1213607  2015-02-06 22:16:43-08:00           481.0              1
1213696  2015-02-06 23:15:11-08:00        599655.0             15
1213750  2015-02-06 23:39:45-08:00          2916.0              1
1213804  2015-02-07 07:15:32-08:00          1759.0              1

         active factor  length title  \
31                   6            11
276                  4            17
386                  1            16
472                 10            21
535                  3            15
…                  …            …
1213483              1             5
```

```
1213607          41          7
1213696           6         10
1213750         163          9
1213804          46         13

                                                clean text sentiment score  \
31       just a superbowlchampion pedestrian wide recei…          0.0
276      get ta love that the entire city of seattle be…       0.8442
386      rt liquidityinc this might be ingenious cjwehl…          0.0
472      during sb block on st b w jefferson and washin…      -0.4404
535      impress your superbowlxlix guest and enjoy wat…       0.7096
…                                                   …            …
1213483               omg so superbowl http t co ldsvsincv          0.0
1213607  tevejonavic letsclub superbowl katyperry darkh…          0.0
1213696  missyelliott cry tear of joy after her superbo…       0.1779
1213750  superbowl seahawks thman get your superbowl sh…          0.0
1213804  expert date and relationship advice for men an…          0.0

        winning label
31                  0
276                 0
386                 0
472                 0
535                 0
…                 …
1213483             1
1213607             1
1213696             1
1213750             1
1213804             1

[12138 rows x 10 columns]
```

5 Baselines

Baseline models should be simple yet relevant. For this task, a logistic regression classifier could serve as a good baseline for binary classification.

Evaluation: A Logistic Regression Classifier is trained to serve as the baseline mode, and a dummy Logistic Regression Classifier is also trained for the purpose of comparison. Two classifiers' evaluation is shown below. Focus on model accuracy, the Losgistic Regression model reach to 0.680 while the dummy classifer is only 0.570. It suports the idea that the baseline logistic model has ability to predict winning team, even though it might not be at a super high acurracy.

[61]:
```python
# Evaluation function

def eval_model(model, X_test_svd, Y_test_label, y_pred, roc_idx):
```

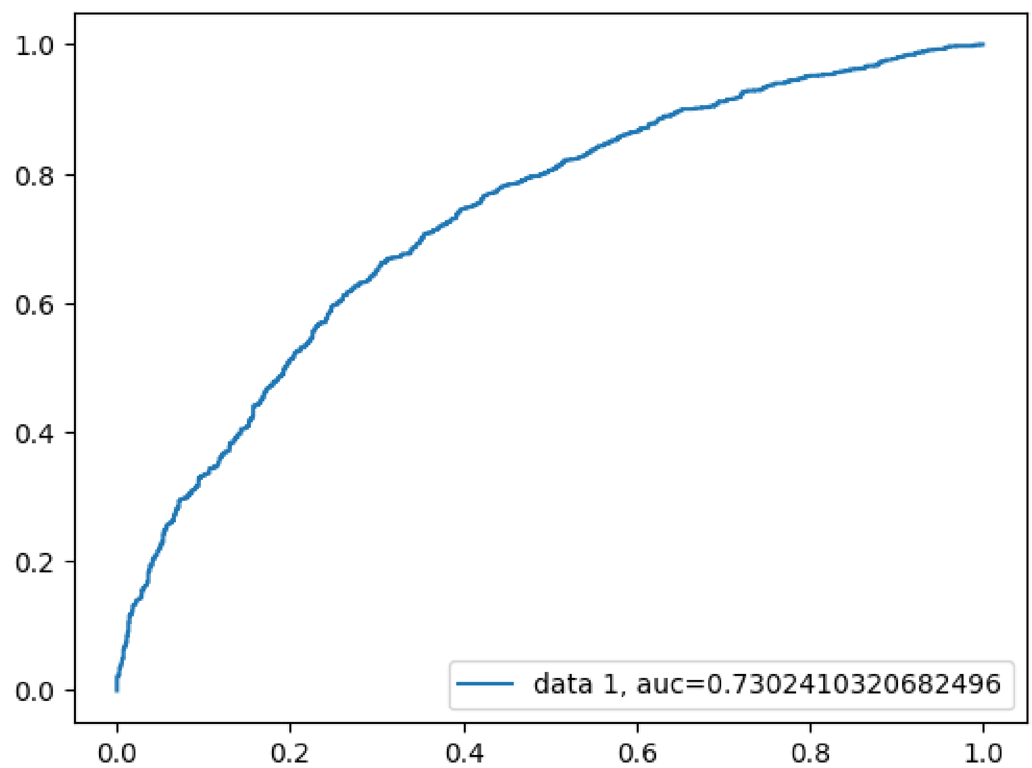
```
  #svc_disp = metrics.RocCurveDisplay.from_estimator(model, X_test_svd,␣
↪Y_test_label) # ROC curve
  cm = metrics.confusion_matrix(Y_test_label, y_pred) # Confusion matrix
  acc = metrics.accuracy_score(Y_test_label, y_pred) # Accuracy
  recall = metrics.recall_score(Y_test_label, y_pred) # Recall
  precision = metrics.precision_score(Y_test_label, y_pred) # Precision
  f1 = metrics.f1_score(Y_test_label, y_pred) # F-1 score

  # AUC curve with more decimal digits
  if (roc_idx):
    y_pred_proba = model.predict_proba(X_test_svd)[::,1]
    fpr, tpr, _ = metrics.roc_curve(Y_test_label,  y_pred_proba)
    auc = metrics.roc_auc_score(Y_test_label, y_pred_proba)
    plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
    plt.legend(loc=4)
    plt.show()

  #plt.show()
  print('Consufion Matrxi:')
  print(cm)
  print("Accuracy:", acc)
  print("Recall:", recall)
  print("Precision:", precision)
  print("F-1 score:", f1)
```

[12]:
```
# Logistic Regression Model
# Prepare dataset and labels

X = df_tweets_select[['clean text', 'retweet_count', 'active factor',␣
↪'sentiment score']]
y = df_tweets_select['winning label']
y = y.astype(int)

# GloVe Word Embedding
text_embedding = X['clean text'].apply(lambda x: text_to_embedding(x,␣
↪glove_embeddings))
expanded_text_embedding = text_embedding.apply(pd.Series)
expanded_text_embedding.columns = ['Feature_' + str(i) for i in range(1,␣
↪len(expanded_text_embedding.columns) + 1)]

# reconstruct dataframe with GloVe Word Embedding
X = X.drop('clean text', axis=1)
X = pd.concat([X, expanded_text_embedding], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)  # shuffle=False for time series data
```

```python
# Logistic Regression
# Initialize a Logistic Regression classifier and fit it to the training data
logreg = LogisticRegression(max_iter=2000)
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)

# Initialize a dummy classifier to always predict the most frequent class
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)
y_pred_dummy = dummy_clf.predict(X_test)
```

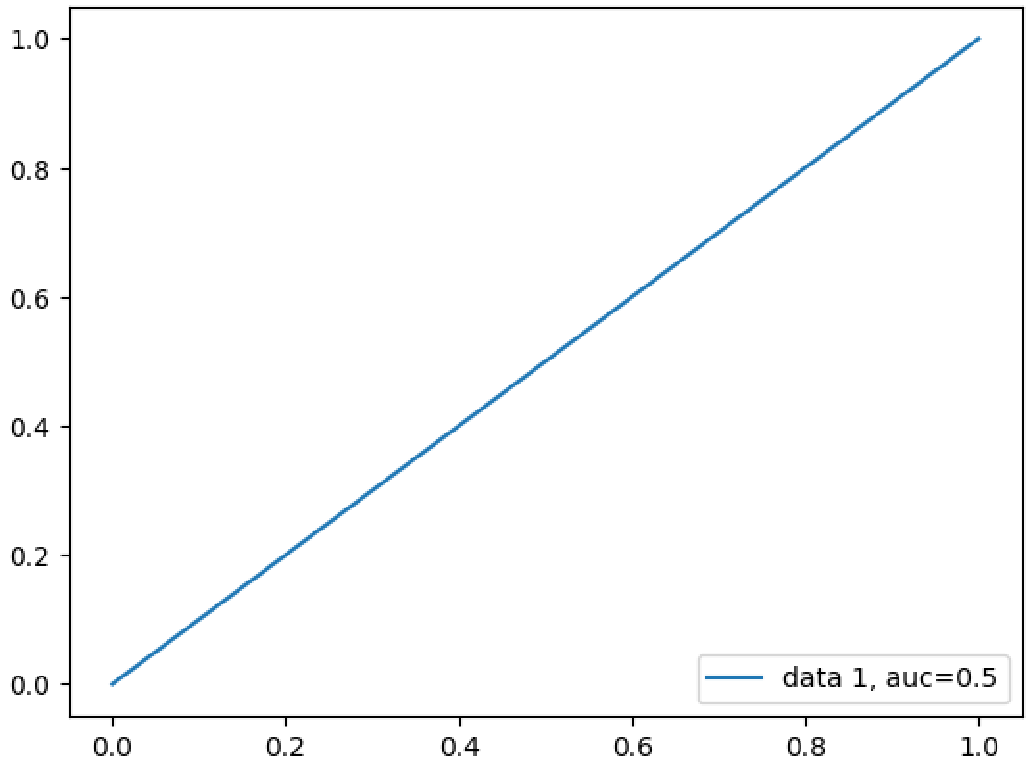
[13]:
```python
# Evaluate the Logistic Regression classifier
print("Logistic Regression Classifier Report")
eval_model(logreg, X_test, y_test, y_pred_logreg, 1)
print('\n')

# Evaluate the dummy classifier
print("Dummy Classifier Report")
eval_model(dummy_clf, X_test, y_test, y_pred_dummy, 1)
```

```
Logistic Regression Classifier Report
```



```
Consufion Matrxi:
[[ 568  481]
 [ 297 1082]]
Accuracy: 0.6795716639209226
Recall: 0.7846265409717187
Precision: 0.6922584772872681
F-1 score: 0.7355540448674373
```

```
Dummy Classifier Report
```




```
Consufion Matrxi:
[[   0 1049]
 [   0 1379]]
Accuracy: 0.5679571663920923
Recall: 1.0
Precision: 0.5679571663920923
F-1 score: 0.7244549514053059
```

6 Advanced Machine Learning Model

Shifting towards deep learning models, especially for tasks involving natural language processing (NLP), can significantly enhance the ability to model and understand complex relationships and patterns in text data. Given your dataset and the use of GloVe embeddings, an LSTM (Long Short-Term Memory) model is a good starting point for exploring deep learning techniques. LSTMs are effective at capturing long-term dependencies in sequence data, making them well-suited for text analysis tasks.

Evaluation: The LSTM model reaches an accuracy of 0.678 which shows little difference from the baseline logistic model. By oberserving Epoch training steps, the accuracy increase and loss decrease as step, indicating that the model is performing as expected. However, the accuracy is not high enough to beat the baseline model.

[90]:
```
# LSTM evaluation function

def eval_model_LSTM(model, X_test, Y_test_label, y_pred, roc_idx,␣
 ↪y_pred_proba=None):
  cm = metrics.confusion_matrix(Y_test_label, y_pred)
  acc = metrics.accuracy_score(Y_test_label, y_pred)
  recall = metrics.recall_score(Y_test_label, y_pred)
  precision = metrics.precision_score(Y_test_label, y_pred)
  f1 = metrics.f1_score(Y_test_label, y_pred)

  if roc_idx and y_pred_proba is not None:
    fpr, tpr, _ = metrics.roc_curve(Y_test_label,  y_pred_proba)
    auc = metrics.roc_auc_score(Y_test_label, y_pred_proba)
    plt.plot(fpr, tpr, label="AUC="+str(auc))
    plt.legend(loc=4)
    plt.show()

  print('Confusion Matrix:')
  print(cm)
  print("Accuracy:", acc)
  print("Recall:", recall)
  print("Precision:", precision)
  print("F-1 score:", f1)
```

[94]:
```
# LSTM model
# Convert pandas DataFrames to numpy arrays
X_train_np = X_train.values.astype('float32')
y_train_np = y_train.values.astype('float32').reshape(-1, 1)  # Reshaping if␣
 ↪y_train is a single column DataFrame
X_test_np = X_test.values.astype('float32')
y_test_np = y_test.values.astype('float32').reshape(-1, 1)  # Reshaping if␣
 ↪y_test is a single column DataFrame

# reshape dataset
```

```
X_train_reshaped = X_train_np.reshape((X_train_np.shape[0], X_train_np.
 ↪shape[1], 1))
X_test_reshaped = X_test_np.reshape((X_test_np.shape[0], X_test_np.shape[1], 1))

# construct model
model = Sequential([
    # Assuming each feature is a sequential step, adding Conv1D for feature␣
 ↪extraction
    Input(shape=(X_train_reshaped.shape[1], X_train_reshaped.shape[2])),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    MaxPooling1D(pool_size=2),
    Flatten(),  # Flatten the convolution output before feeding into dense␣
 ↪layers
    Dense(256, activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

# Define learning rate schedule
initial_learning_rate = 0.001
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=100,
    decay_rate=0.96,
    staircase=True)

# Compile the model with the new learning rate schedule
optimizer = Adam(learning_rate=lr_schedule)
model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Define early stopping callback and fit model
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)
model.fit(X_train_np, y_train_np, batch_size=64, epochs=30, validation_split=0.
 ↪1, callbacks=[early_stopping])
```

```
Epoch 1/30
137/137              17s 88ms/step -
accuracy: 0.5644 - loss: 0.7132 - val_accuracy: 0.6097 - val_loss: 0.6651
Epoch 2/30
```

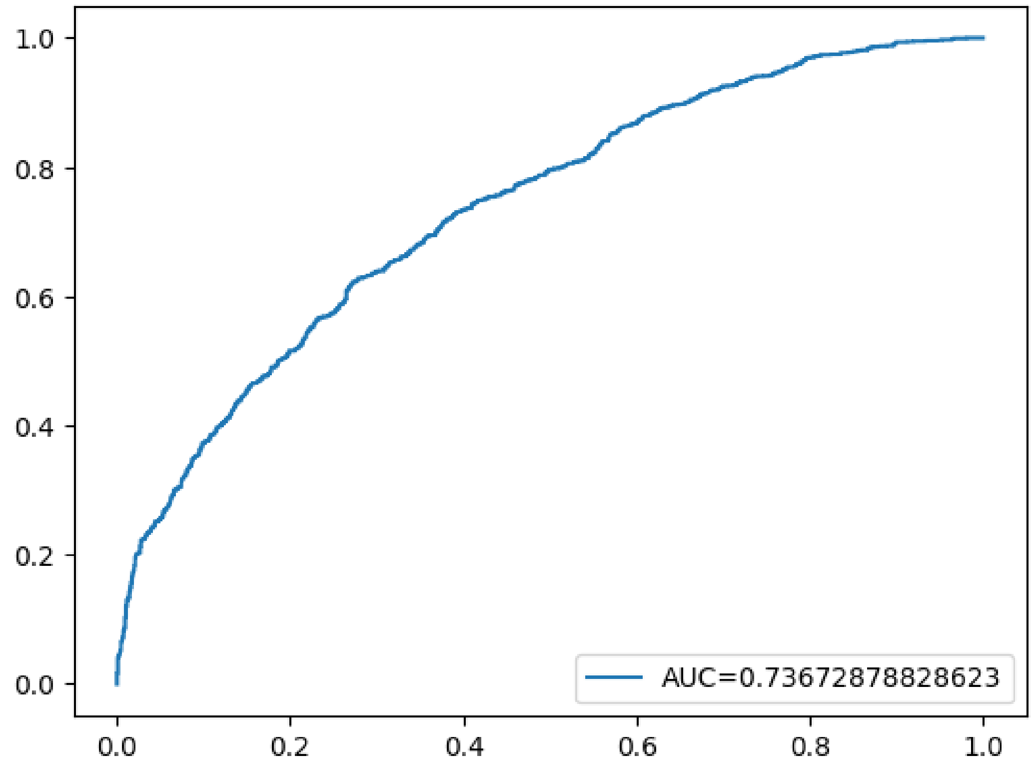
```
137/137 22s 102ms/step -
accuracy: 0.5999 - loss: 0.6693 - val_accuracy: 0.6406 - val_loss: 0.6365
Epoch 3/30
137/137 16s 72ms/step -
accuracy: 0.6269 - loss: 0.6499 - val_accuracy: 0.6519 - val_loss: 0.6285
Epoch 4/30
137/137 11s 79ms/step -
accuracy: 0.6515 - loss: 0.6338 - val_accuracy: 0.6416 - val_loss: 0.6261
Epoch 5/30
137/137 12s 85ms/step -
accuracy: 0.6527 - loss: 0.6204 - val_accuracy: 0.6395 - val_loss: 0.6285
Epoch 6/30
137/137 19s 73ms/step -
accuracy: 0.6791 - loss: 0.6102 - val_accuracy: 0.6468 - val_loss: 0.6169
Epoch 7/30
137/137 13s 92ms/step -
accuracy: 0.6714 - loss: 0.6046 - val_accuracy: 0.6529 - val_loss: 0.6180
Epoch 8/30
137/137 14s 103ms/step -
accuracy: 0.6877 - loss: 0.5836 - val_accuracy: 0.6560 - val_loss: 0.6156
Epoch 9/30
137/137 18s 84ms/step -
accuracy: 0.6861 - loss: 0.5886 - val_accuracy: 0.6488 - val_loss: 0.6082
Epoch 10/30
137/137 19s 74ms/step -
accuracy: 0.6915 - loss: 0.5811 - val_accuracy: 0.6189 - val_loss: 0.6349
Epoch 11/30
137/137 13s 92ms/step -
accuracy: 0.6895 - loss: 0.5764 - val_accuracy: 0.6365 - val_loss: 0.6156
Epoch 12/30
137/137 18s 72ms/step -
accuracy: 0.6993 - loss: 0.5682 - val_accuracy: 0.6416 - val_loss: 0.6092
Epoch 13/30
137/137 11s 80ms/step -
accuracy: 0.7033 - loss: 0.5599 - val_accuracy: 0.6509 - val_loss: 0.6154
Epoch 14/30
137/137 9s 63ms/step -
accuracy: 0.7292 - loss: 0.5487 - val_accuracy: 0.6375 - val_loss: 0.6148
```

[94]:
```
<keras.src.callbacks.history.History at 0x7da51c427730>
```

[95]:
```
# Generate probabilities
y_pred_proba = model.predict(X_test_np).flatten()

# Convert probabilities to binary predictions (0 or 1) based on a 0.5 threshold
y_pred_binary = (y_pred_proba > 0.5).astype(int)
```

```
# evaluate performance
eval_model_LSTM(model, X_test_np, y_test_np, y_pred_binary, roc_idx=1,
    y_pred_proba=y_pred_proba)
```

```
76/76                 1s 10ms/step
```




```
Confusion Matrix:
[[ 528  521]
 [ 285 1094]]
Accuracy: 0.6680395387149918
Recall: 0.7933284989122552
Precision: 0.6773993808049535
F-1 score: 0.7307949231796926
```

7 Evaluation

Compare performance of the baseline Logistic Model and LSTM model.

1. Accuracy Logistic Regression: 67.96% LSTM: 66.80% Analysis: The logistic regression model has a slightly higher accuracy compared to the LSTM model. Accuracy measures the overall correctness of the model across both classes but doesn't provide insight into the model's performance on individual classes.

2. Recall Logistic Regression: 78.46% LSTM: 79.33% Analysis: Both models have similar recall scores, with the LSTM model slightly outperforming the logistic regression model. Recall measures the model's ability to correctly identify positive instances. A higher recall indicates fewer false negatives. The similar recall scores suggest both models are comparably effective at identifying positive instances.

3. Precision Logistic Regression: 69.23% LSTM: 67.74% Analysis: The logistic regression model has a higher precision than the LSTM model. Precision measures the proportion of true positive predictions in all positive predictions made by the model. A higher precision indicates fewer false positives. The logistic regression model is slightly better at ensuring its positive predictions are correct.

4. F-1 Score Logistic Regression: 73.56% LSTM: 73.08% Analysis: The F-1 score is a harmonic mean of precision and recall, providing a single metric to assess the balance between them. The logistic regression model has a marginally higher F-1 score, suggesting a better balance between precision and recall compared to the LSTM model.

5. Overall Evaluation and Analysis The logistic regression model slightly outperforms the LSTM model across most metrics, with notable differences in precision and overall accuracy.

The reason that LSTM model did not perform outstandingly compare to the baseline logistic model can be that 1. Feature Extraction: The features extracted from original features might not be curical enough for LSTM model to achieve a better performance. 2. Model Parameters: Both models' performance can significantly depend on the choice of hyperparameters. The LSTM model might require more extensive hyperparameter tuning to optimize its architecture and training process for the specific task. 3. Model Complexity: LSTM might work better with a more complex model, comparing to the setup I used.