Wine Quality Prediction using classificaltion algorithms

In [2]:
```python
## importing librarie
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```python
df = pd.read_csv('C:\\Users\\navon\\Downloads\\winequalityN.csv')
df.head(5)
```

Out[3]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [4]:
```python
df.shape
```

Out[4]:
```
(6497, 13)
```

In [5]:
```python
df.describe()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6487.000000	6489.000000	6494.000000	6495.000000	6495.000000	6497.000000	6497.000000	6497.000000	6488.000000	6493.000000	6497.000000	6497.000000
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574	0.994697	3.218395	0.531215	10.491801	5.818378
std	1.296750	0.164649	0.145265	4.758125	0.035036	17.749400	56.521855	0.002999	0.160748	0.148814	1.192712	0.873255
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

In [25]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 634.5+ KB
```

In [6]:
```python
#checking null values
df.isnull().any()
```

Out[6]:
```
type                    False
fixed acidity            True
volatile acidity         True
citric acid              True
residual sugar           True
chlorides                True
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                       True
sulphates                True
alcohol                 False
quality                 False
dtype: bool
```

In [7]:
```python
#fill null values with their mean
df.fillna(df.mean(), inplace=True)
```

```
C:\Users\navon\AppData\Local\Temp/ipykernel_11412/2471838208.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns before calling the reduction.
  df.fillna(df.mean(), inplace=True)
```

In [8]:
```python
df.isnull().any()
```

Out[8]:
```
type                    False
fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

In [29]:
```python
#to show types of winery
df['type'].unique()
```

Out[29]:
```
array(['white', 'red'], dtype=object)
```

In [30]:
```python
df['type'].value_counts()
```

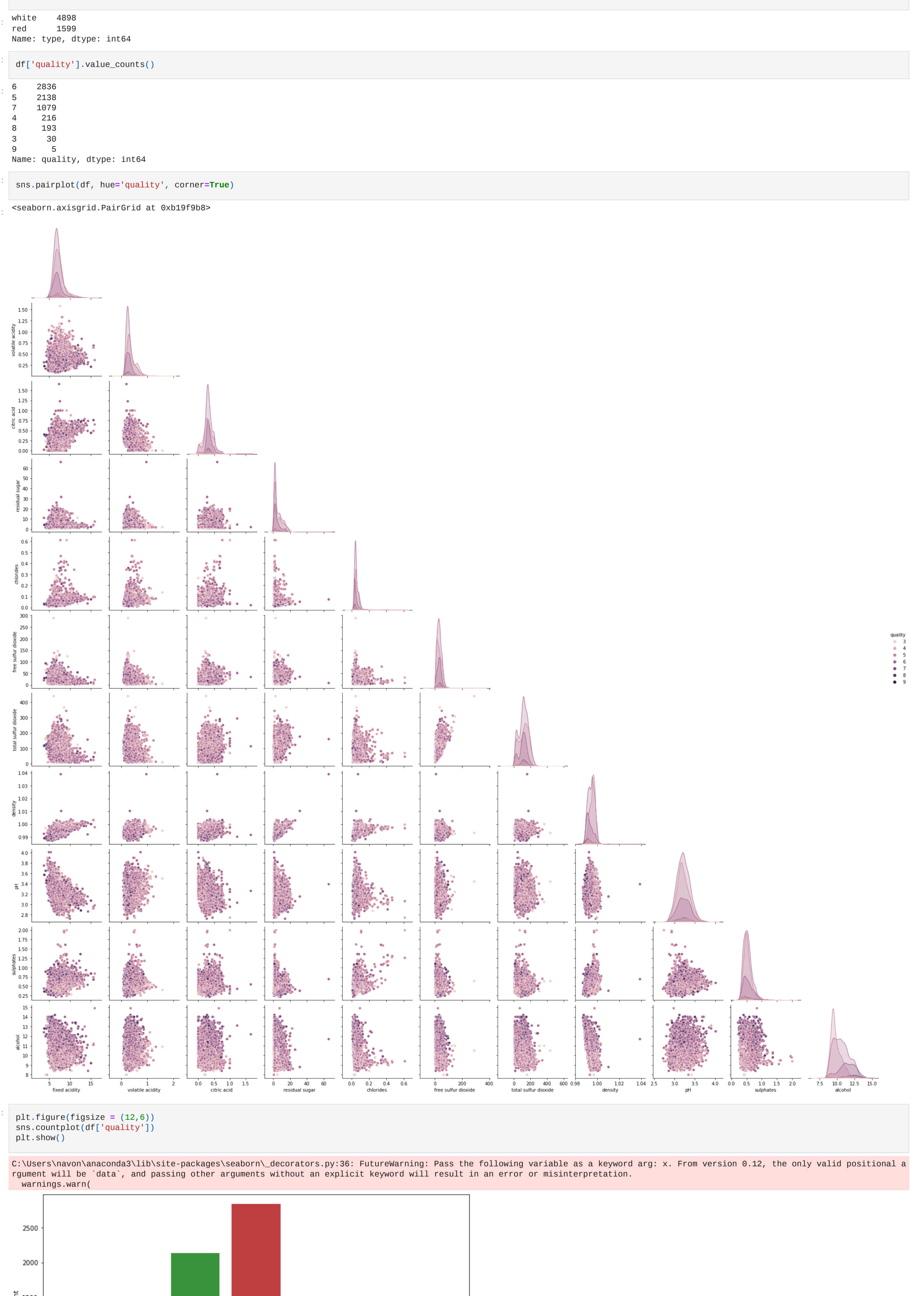
Out[30]:
```
white    4898
red      1599
Name: type, dtype: int64
```

In [31]:
```python
df['quality'].value_counts()
```

Out[31]:
```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

In [32]:
```python
sns.pairplot(df, hue='quality', corner=True)
```

Out[32]:
```
<seaborn.axisgrid.PairGrid at 0xb19f9b8>
```



In [39]:
```python
plt.figure(figsize = (12,6))
sns.countplot(df['quality'])
plt.show()
```

```
C:\Users\navon\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional a
rgument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

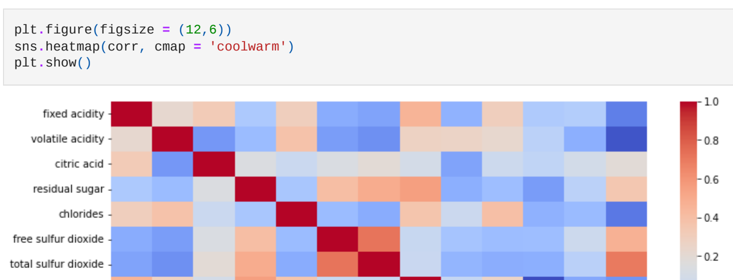
In [1]:
```python
sns.countplot(x="type", hue="quality", data=df)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11412/3183589349.py in <module>
----> 1 sns.countplot(x="type", hue="quality", data=df)

NameError: name 'sns' is not defined
```

In [41]:
```python
corr=df.corr()
```

In [42]:
```python
plt.figure(figsize = (12,6))
sns.heatmap(corr, cmap = 'coolwarm')
plt.show()
```



In [43]:
```python
#converting categorical to numerical
df = pd.get_dummies(df, drop_first=True)
df.head()
```

Out[43]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	wine_type
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1

In [44]:
```python
df = df.rename(columns={"type_white": "wine_type"})
df.head()
```

Out[44]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	wine_type
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	1
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	1
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1

we had an unbalanced dataset with respect to wine quality, we divide the qualities into two groups - 1 (for wine quality 7, 8, 9) and 0 (for 6 and below)

In [45]:
```python
df["wine_quality"] = [1 if x>6 else 0 for x in df.quality]
df.head()
```

Out[45]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	wine_type	wine_quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6	1	0
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6	1	0
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6	1	0
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1	0
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6	1	0

In [46]:
```python
y=df["wine_quality"]
```

In [47]:
```python
y.value_counts()
```

Out[47]:
```
0    5220
1    1277
Name: wine_quality, dtype: int64
```

In [48]:
```python
y
```

Out[48]:
```
0       0
1       0
2       0
3       0
4       0
       ..
6492    0
6493    0
6494    0
6495    0
6496    0
Name: wine_quality, Length: 6497, dtype: int64
```

In [49]:
```python
x = df.drop(["quality", "wine_quality"], axis=1)
```

In [50]:
```python
x
```

Out[50]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	wine_type
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.450000	8.8	1
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.490000	9.5	1
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.440000	10.1	1
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.400000	9.9	1
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.400000	9.9	1
...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.580000	10.5	0
6493	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.531215	11.2	0
6494	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.750000	11.0	0
6495	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.710000	10.2	0
6496	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.660000	11.0	0

6497 rows × 12 columns

Model Training

In [158…
```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [159…
```python
log = pd.DataFrame(columns=["model", "accuracy"])
```

In [160…
```python
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 42)
```

Logistic Regression

In [161…
```python
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='liblinear')

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

lr.score(X_train, y_train)

score = accuracy_score(y_test, y_pred)

confusion_matrix(y_test, y_pred)
```

Out[161…
```
array([[1013,   34],
       [ 195,   58]], dtype=int64)
```

In [162…
```python
log = log.append({"model": "logistic regression", "accuracy": score}, ignore_index=True)
```

Support Vector Classification - SVC

In [163…
```python
from sklearn.svm import SVC
```

In [164…
```python
Wq = SVC(kernel="rbf",C=1)

Wq.fit(X_train, y_train)

y_pred = Wq.predict(X_test)

score = accuracy_score(y_test, y_pred)

confusion_matrix(y_test, y_pred)
```

Out[164…
```
array([[1047,    0],
       [ 253,    0]], dtype=int64)
```

In [165…
```python
log = log.append({"model": "SVC", "accuracy": score}, ignore_index=True)
```

In [166…
```python
log
```

Out[166…

	model	accuracy
0	logistic regression	0.823846
1	SVC	0.805385

K Nearest Neighbour

In [167…
```python
from sklearn.neighbors import KNeighborsClassifier
```

In [168…
```python
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
clf.predict(X_test)
y_pred = clf.predict(X_test)
score = accuracy_score(y_test, y_pred)
```

In [169…
```python
log = log.append({"model": "K Nearest Neighbours", "accuracy": score}, ignore_index=True)
```

In [170…
```python
log
```

Out[170…

	model	accuracy
0	logistic regression	0.823846
1	SVC	0.805385
2	K Nearest Neighbours	0.806923

Decision Tree Classifier

In [171…
```python
from sklearn.tree import DecisionTreeClassifier
```

In [172…
```python
dtc = DecisionTreeClassifier(criterion='entropy',random_state=7)
dtc.fit(X_train,y_train)
y_pred = dtc.predict(X_test)
score = accuracy_score(y_test, y_pred)
```

In [173…
```python
log = log.append({"model": "Decision Tree", "accuracy": score}, ignore_index=True)
```

In [174…
```python
log
```

Out[174…

	model	accuracy
0	logistic regression	0.823846
1	SVC	0.805385
2	K Nearest Neighbours	0.806923
3	Decision Tree	0.838462

Random Forest Classifier

In [175…
```python
from sklearn.ensemble import RandomForestClassifier
```

In [176…
```python
rfc = RandomForestClassifier(random_state = 1)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)
score = accuracy_score(y_test, y_pred)
```

In [177…
```python
log = log.append({"model": "Random Forest", "accuracy": score}, ignore_index=True)
```

In [178…
```python
log
```

Out[178…

	model	accuracy
0	logistic regression	0.823846
1	SVC	0.805385
2	K Nearest Neighbours	0.806923
3	Decision Tree	0.838462
4	Random Forest	0.883846

Navie Bayes

In [179…
```python
from sklearn.naive_bayes import GaussianNB
```

In [180…
```python
Nb = GaussianNB()
Nb.fit(X_train, y_train)
Nb.predict(X_test)
y_pred = Nb.predict(X_test)
score = accuracy_score(y_test, y_pred)
```

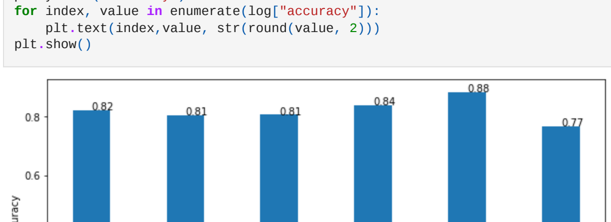
In [181…
```python
log = log.append({"model": "GaussianNB", "accuracy": score}, ignore_index=True)
```

In [182…
```python
log
```

Out[182…

	model	accuracy
0	logistic regression	0.823846
1	SVC	0.805385
2	K Nearest Neighbours	0.806923
3	Decision Tree	0.838462
4	Random Forest	0.883846
5	GaussianNB	0.769231

In [183…
```python
plt.figure(figsize=(10, 5))
plt.bar(log["model"], log["accuracy"], width=0.4)
plt.xlabel("Classifier")
plt.ylabel("accuracy")
for index, value in enumerate(log["accuracy"]):
    plt.text(index,value, str(round(value, 2)))
plt.show()
```



In [ ]: