Hyper Parameter Tuning GridSearchCV

- Data Ingestion
- EDA
- Feature Engineering
- Feature Scaling
- Model Training
- Performance Metrics
- Hyper parameter Tuning

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
data=pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/
```

In [2]:
```
data.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [3]:
```
data['quality'].value_counts()
```

Out[3]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [4]:
```
data['quality'].unique()
```

Out[4]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [5]:
```
data.columns
```

Out[5]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [7]:
```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

In [8]:
```
data.corr()
```

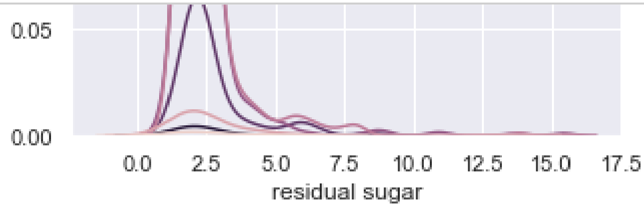
Out[8]:

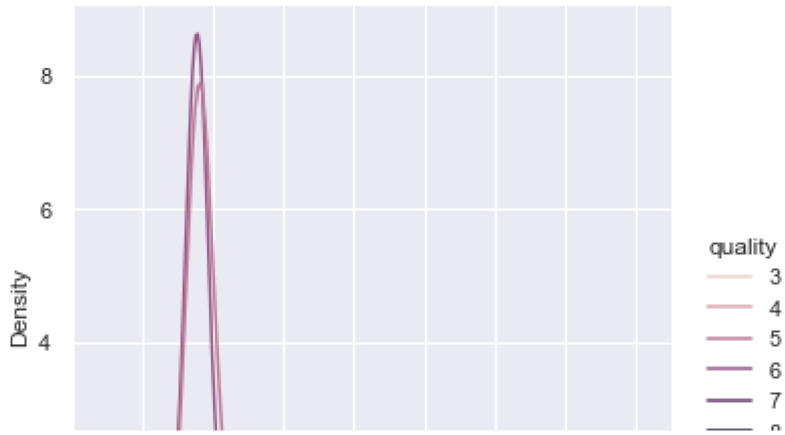
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	0.070377
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-0.066495
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-0.341699
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1.000000
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-0.196648
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	0.205633
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919	-0.057731

In [9]:
```
data.columns
```

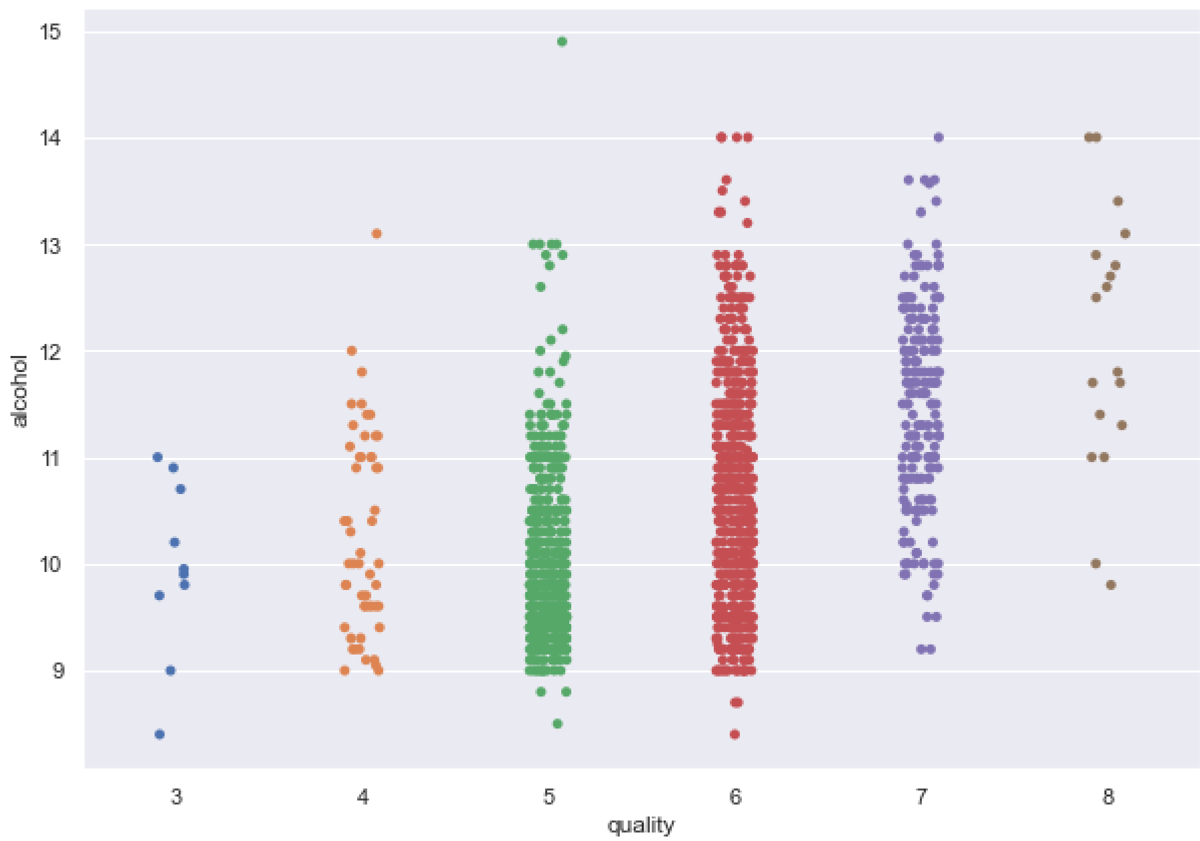
Out[9]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [10]:
```python
sns.set(rc={"figure.figsize":(10, 7)})
for col in data.columns:
    sns.displot(data=data,x=data[col],hue='quality',kind='kde')
```

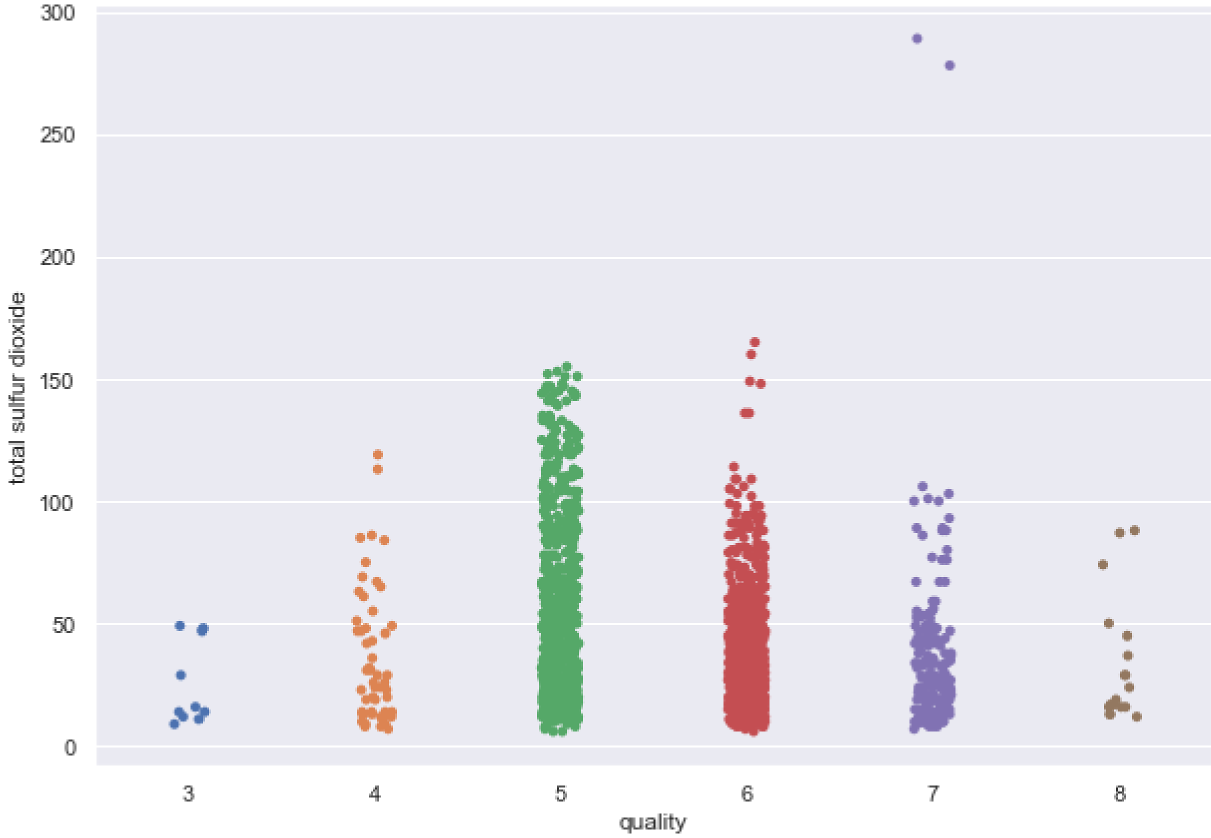




In [11]:
```python
sns.stripplot(data=data,y=data['alcohol'],x=data['quality'])
sns.set(rc={"figure.figsize":(10, 7)})
```

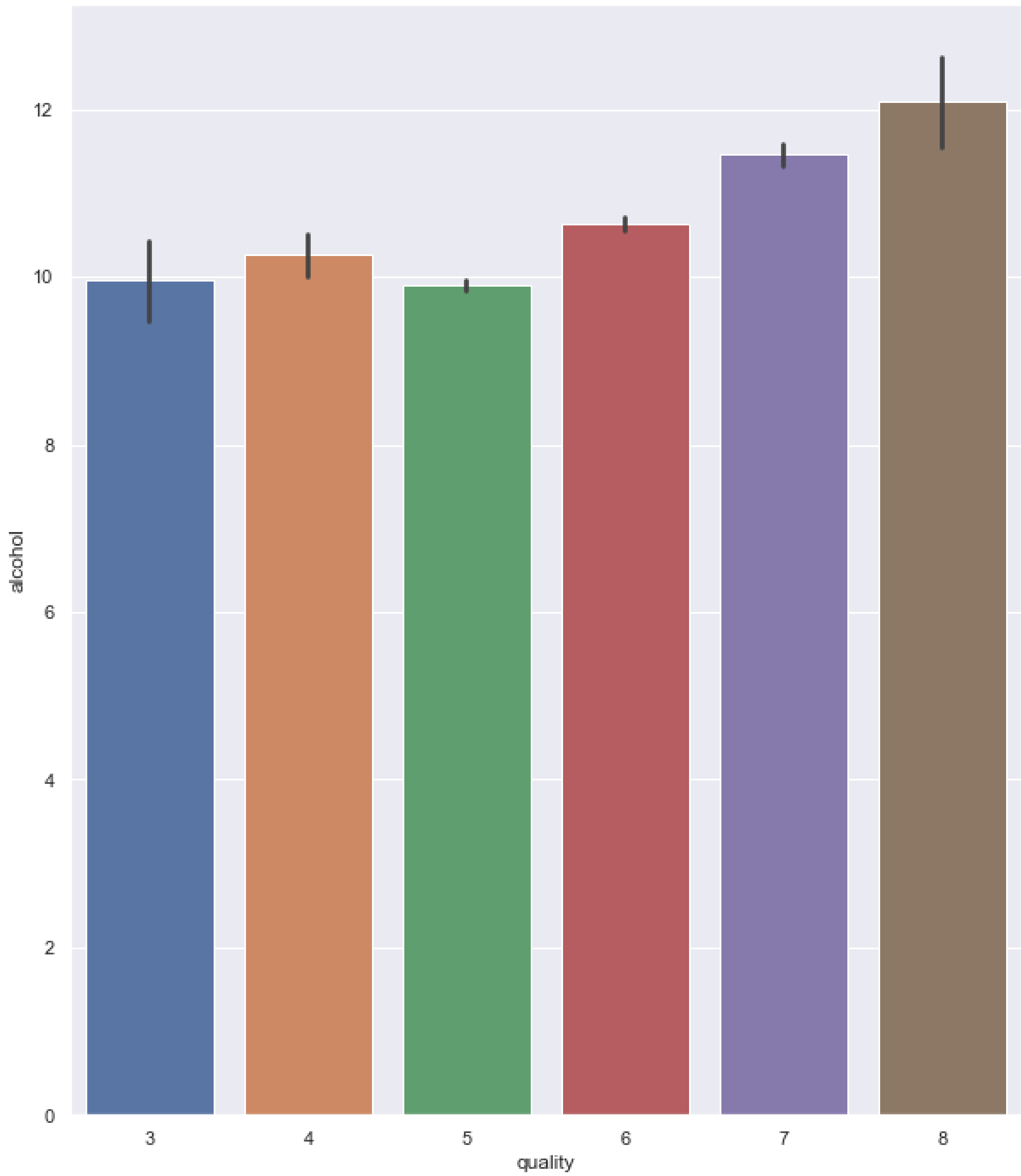


In [12]:
```
sns.stripplot(data=data,y=data['total sulfur dioxide'],x=data['quality'])
sns.set(rc={"figure.figsize":(10, 12)})
```



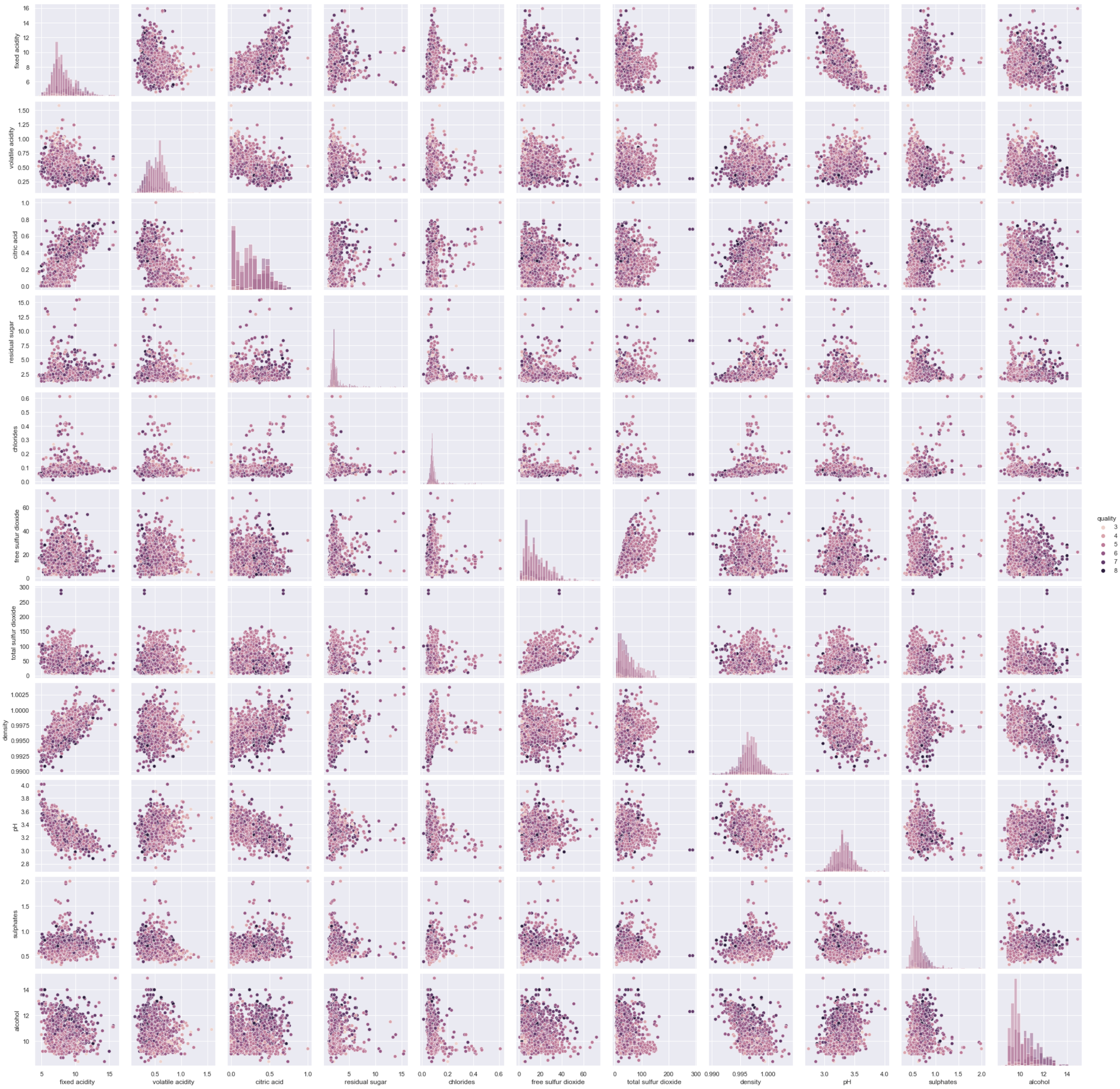
In [13]:
```python
sns.barplot(x='quality',data=data,y='alcohol')
```

Out[13]: <AxesSubplot:xlabel='quality', ylabel='alcohol'>



In [14]:
```
sns.pairplot(data,hue="quality", diag_kind="hist")
```

Out[14]: <seaborn.axisgrid.PairGrid at 0x2367df8e0d0>



In [15]:
```
data.head()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [16]:
```
X=data.drop('quality',axis=1)
```

In [17]:
```
X.head()
```

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [18]:
```
y=data['quality']
```

In [19]:
```
y.head()
```

Out[19]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

In [20]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=42)
```

In [21]:
```
X_test.head()
```

Out[21]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
803	7.7	0.56	0.08	2.50	0.114	14.0	46.0	0.9971	3.24	0.66	9.6
124	7.8	0.50	0.17	1.60	0.082	21.0	102.0	0.9960	3.39	0.48	9.5
350	10.7	0.67	0.22	2.70	0.107	17.0	34.0	1.0004	3.28	0.98	9.9
682	8.5	0.46	0.31	2.25	0.078	32.0	58.0	0.9980	3.33	0.54	9.8
1326	6.7	0.46	0.24	1.70	0.077	18.0	34.0	0.9948	3.39	0.60	10.6

In [22]:
```
X_train.head()
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
548	12.4	0.350	0.49	2.6	0.079	27.0	69.0	0.99940	3.12	0.75	10.4
355	6.7	0.750	0.01	2.4	0.078	17.0	32.0	0.99550	3.55	0.61	12.8
1296	6.6	0.630	0.00	4.3	0.093	51.0	77.5	0.99558	3.20	0.45	9.5
209	11.0	0.300	0.58	2.1	0.054	7.0	19.0	0.99800	3.31	0.88	10.5
140	8.4	0.745	0.11	1.9	0.090	16.0	63.0	0.99650	3.19	0.82	9.6

In [23]:
```
y_train.head()
```
Out[23]:
```
548     6
355     6
1296    5
209     7
140     5
Name: quality, dtype: int64
```
In [24]:
```
y_test.head()
```
Out[24]:
```
803     6
124     5
350     6
682     5
1326    6
Name: quality, dtype: int64
```
In [25]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler
```
Out[25]:
```
StandardScaler()
```
In [26]:
```
scaler.fit(X_train)##calculate the mean and std dev
```
Out[26]:
```
StandardScaler()
```
In [27]:
```
print(scaler.mean_)
```
```
[ 8.30345472  0.53246499  0.26933707  2.54691877  0.08772736 15.91223156
 46.76330532  0.99677933  3.31453782  0.65881419 10.41521942]
```
In [28]:
```
X_train_tf=scaler.fit_transform(X_train)
X_train_tf
```
Out[28]:
```
array([[ 2.40069523, -1.03103722,  1.12742595, ..., -1.26096312,
         0.52726134, -0.01431863],
       [-0.93967131,  1.22920403, -1.32502245, ...,  1.52622836,
        -0.28225704,  2.24363201],
       [-0.99827424,  0.55113165, -1.37611513, ..., -0.74241587,
        -1.20742091, -0.86105011],
       ...,
       [-0.6466567 ,  0.49462562, -1.06955908, ...,  1.26695473,
        -0.68701624, -0.86105011],
       [-0.23643625, -1.87862768,  0.4121285 , ...,  0.03540501,
         0.81637505,  1.39690052],
       [-1.46709761, -1.3700734 , -0.04770558, ...,  0.48913386,
        -0.68701624,  2.90220094]])
```
In [29]:
```
X_test_tf=scaler.transform(X_test)
```
In [30]:
```
from sklearn.svm import SVC
model=SVC(kernel='linear',random_state=42)
model
```
Out[30]:
```
SVC(kernel='linear', random_state=42)
```

In [31]:
```
model.fit(X_train_tf,y_train)
```

Out[31]:
```
SVC(kernel='linear', random_state=42)
```

In [32]:
```
model.score(X_train_tf,y_train)
```

Out[32]:
```
0.5994397759103641
```

In [33]:
```
y_pred=model.predict(X_test_tf)
```

In [34]:
```
y_test.head()
```

Out[34]:
```
803     6
124     5
350     6
682     5
1326    6
Name: quality, dtype: int64
```

In [35]:
```
from sklearn.metrics import accuracy_score
```

In [36]:
```
accuracy_score(y_pred,y_test)
```

Out[36]:
```
0.5587121212121212
```

In [37]:
```
from sklearn.model_selection import GridSearchCV
parameters=[{"C":[1,10,100,1000],'kernel':['linear']},
           {"C":[1,10,100,1000],'kernel':['rbf'],'gamma':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0
gridSearch=GridSearchCV(estimator=model,param_grid=parameters,
                        scoring="accuracy",
                        cv=10,
                        n_jobs=-1)
gridSearch=gridSearch.fit(X_train_tf,y_train)
```

```
C:\Users\prasa\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:676: User
Warning: The least populated class in y has only 8 members, which is less than n_split
s=10.
  warnings.warn(
```

In [38]:
```
accuracy=gridSearch.best_score_
```

In [39]:
```
accuracy
```

Out[39]:
```
0.6554690204222914
```

In [40]:
```
gridSearch.best_params_
```

Out[40]:
```
{'C': 10, 'gamma': 0.8, 'kernel': 'rbf'}
```

In [41]:
```
model=SVC(C=10,kernel='rbf',gamma=0.8,random_state=42)
model
```

Out[41]:
```
SVC(C=10, gamma=0.8, random_state=42)
```

In [42]:
```
model.fit(X_train_tf,y_train)
```

Out[42]:
```
SVC(C=10, gamma=0.8, random_state=42)
```

In [43]:
```
y_pred=model.predict(X_test_tf)
```

In [44]:
```
model.score(X_train_tf,y_train)
```

Out[44]:
```
0.9971988795518207
```

In [45]:
```
from sklearn.metrics import accuracy_score
accuracy=accuracy_score(y_test,y_pred)
```

In [46]:
```
accuracy
```

Out[46]:
```
0.6193181818181818
```

In [47]:
```
from sklearn.linear_model import LogisticRegression
regressor=LogisticRegression()
regressor
```

Out[47]:
```
LogisticRegression()
```

In [48]:
```
regressor.fit(X_train_tf,y_train)
```

Out[48]:
```
LogisticRegression()
```

In [49]:
```
y_pred2=regressor.predict(X_test_tf)
```

In [50]:
```
regressor.score(X_train_tf,y_train)
```

Out[50]:
```
0.6218487394957983
```

In [51]:
```
from sklearn.metrics import accuracy_score
```

In [52]:
```
accuracy_score(y_test,y_pred2)
```

Out[52]:
```
0.571969696969697
```

In [53]:
```
import numpy as np
C=np.linspace(1,100,10)
penalty = ['l1', 'l2']
C
```

Out[53]:
```
array([  1.,  12.,  23.,  34.,  45.,  56.,  67.,  78.,  89., 100.])
```

In [54]:
```
solver_options = ['newton-cg', 'lbfgs', 'liblinear', 'sag']
multi_class_options = ['ovr', 'multinomial']
class_weight_options = ['None', 'balanced']
param_grid = dict(C=C,penalty=penalty,solver = solver_options, multi_class = multi_class
```

In [55]:
```
grid = GridSearchCV(estimator=regressor, param_grid=param_grid, cv=10, scoring = 'accura
grid.fit(X_train_tf, y_train)
```

```
        nan        nan        nan        nan 0.51080824        nan
 0.43702838 0.43702838 0.50987366 0.410964         nan        nan
        nan        nan 0.40437002 0.40437002        nan 0.37541537
        nan        nan        nan        nan 0.60417965 0.60417965
        nan 0.60417965        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
 0.51174282        nan 0.43702838 0.43702838 0.50987366 0.38285739
        nan        nan        nan        nan 0.40437002 0.40437002
        nan 0.38101419]
  warnings.warn(
```

In [56]:
```
grid.best_score_
```

Out[56]:
```
0.6041796469366563
```

In [57]:
```
grid.best_params_
```

Out[57]:
```
{'C': 1.0,
 'class_weight': 'None',
 'multi_class': 'ovr',
 'penalty': 'l2',
 'solver': 'newton-cg'}
```

In [58]:
```
from sklearn.linear_model import LogisticRegression
regressor=LogisticRegression(C=5.5,class_weight='None',multi_class='ovr',solver='newton
regressor
```

Out[58]:
```
LogisticRegression(C=5.5, class_weight='None', multi_class='ovr',
                   solver='newton-cg')
```

In [59]:
```
regressor.fit(X_train_tf,y_train)
```

Out[59]:
```
LogisticRegression(C=5.5, class_weight='None', multi_class='ovr',
                   solver='newton-cg')
```

In [60]:
```
y_pred_Grid=regressor.predict(X_test_tf)
```

In [61]:
```
accuracy_score(y_test,y_pred_Grid)
```

Out[61]:
```
0.5606060606060606
```