```
#ML Class 20
#Wine Quality Prediction
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
#Loading the datasets
```

```
red_wine=pd.read_csv('winequality-red.csv',delimiter=';')
red_wine
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.6
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.6
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.5
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.5
...	...	...	...	...	...	...	...	...	...	..
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.7
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.7
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.7
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.6

Next steps: Generate code with red_wine View recommended plots

```
white_wine=pd.read_csv('winequality-white.csv',delimiter=';')
white_wine
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.4
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.4
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.4
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.4
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.4
...	...	...	...	...	...	...	...	...	...	..
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.5
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.4
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.4
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.3
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.3

Next steps: Generate code with white_wine View recommended plots

```
#Exploratory Data Analysis
```

```
print(red_wine['quality'].unique())
print('-----------------------------------')
print(white_wine['quality'].unique())
print('-----------------------------------')
print(red_wine.isna().sum())
print('-----------------------------------')
print(white_wine.isna().sum())
```

```
[5 6 7 4 8 3]
-----------------------------------
[6 5 7 8 4 3 9]
-----------------------------------
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
-----------------------------------
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

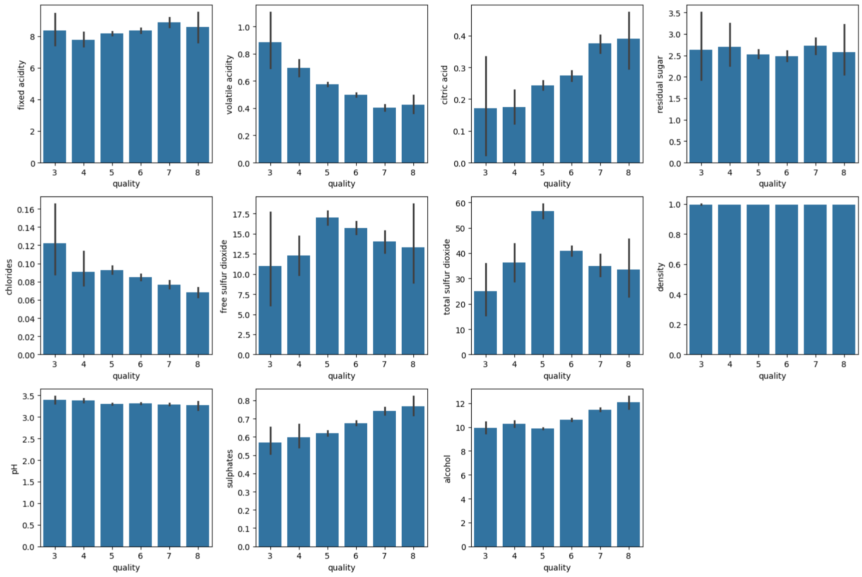
```
#Red wine quality is higher then white wine in the same range.
#No null values in both datasets.
```

```
print(red_wine.describe())
print('-----------------------')
print(white_wine.describe())
print('------------------------')
print(red_wine.info())
print('------------------------')
print(white_wine.info())
```

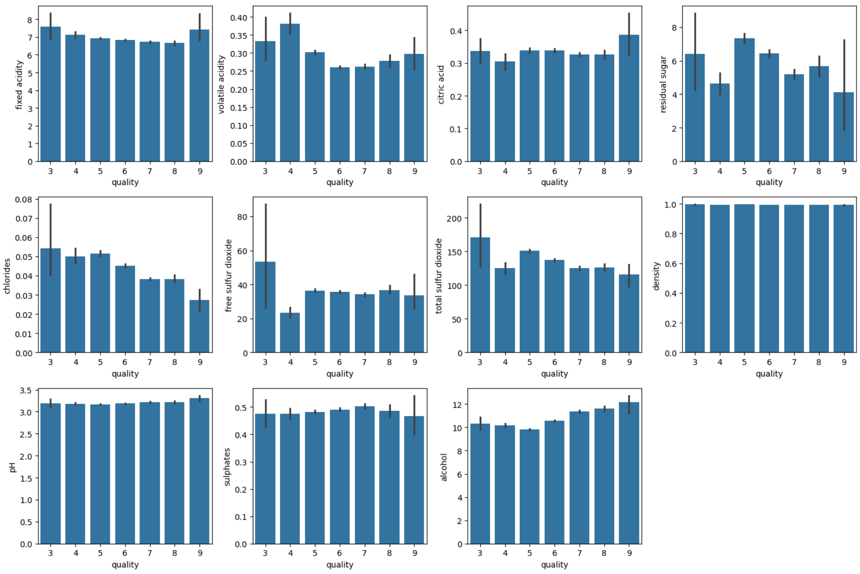
```
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
None
------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
None
```

```
#No categorical columns in both datasets.
```

```
#Quality vs Features for red wine
fig=plt.figure(figsize=(15,10))
plt.subplot(3,4,1)
sns.barplot(x='quality',y='fixed acidity',data=red_wine)
plt.subplot(3,4,2)
sns.barplot(x='quality',y='volatile acidity',data=red_wine)
plt.subplot(3,4,3)
sns.barplot(x='quality',y='citric acid',data=red_wine)
plt.subplot(3,4,4)
sns.barplot(x='quality',y='residual sugar',data=red_wine)
plt.subplot(3,4,5)
sns.barplot(x='quality',y='chlorides',data=red_wine)
plt.subplot(3,4,6)
sns.barplot(x='quality',y='free sulfur dioxide',data=red_wine)
plt.subplot(3,4,7)
sns.barplot(x='quality',y='total sulfur dioxide',data=red_wine)
plt.subplot(3,4,8)
sns.barplot(x='quality',y='density',data=red_wine)
plt.subplot(3,4,9)
sns.barplot(x='quality',y='pH',data=red_wine)
plt.subplot(3,4,10)
sns.barplot(x='quality',y='sulphates',data=red_wine)
plt.subplot(3,4,11)
sns.barplot(x='quality',y='alcohol',data=red_wine)
plt.tight_layout()
```




```
#Quality vs Features for white wine
fig=plt.figure(figsize=(15,10))
plt.subplot(3,4,1)
sns.barplot(x='quality',y='fixed acidity',data=white_wine)
plt.subplot(3,4,2)
sns.barplot(x='quality',y='volatile acidity',data=white_wine)
plt.subplot(3,4,3)
sns.barplot(x='quality',y='citric acid',data=white_wine)
plt.subplot(3,4,4)
sns.barplot(x='quality',y='residual sugar',data=white_wine)
plt.subplot(3,4,5)
sns.barplot(x='quality',y='chlorides',data=white_wine)
plt.subplot(3,4,6)
sns.barplot(x='quality',y='free sulfur dioxide',data=white_wine)
plt.subplot(3,4,7)
sns.barplot(x='quality',y='total sulfur dioxide',data=white_wine)
plt.subplot(3,4,8)
sns.barplot(x='quality',y='density',data=white_wine)
plt.subplot(3,4,9)
sns.barplot(x='quality',y='pH',data=white_wine)
plt.subplot(3,4,10)
sns.barplot(x='quality',y='sulphates',data=white_wine)
plt.subplot(3,4,11)
sns.barplot(x='quality',y='alcohol',data=white_wine)
plt.tight_layout()
```




```
temp_red=red_wine[red_wine['quality']>5.5]['volatile acidity']
temp1_red=red_wine[red_wine['quality']<=5.5]['volatile acidity']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

```
Quality Vs Volatile Acidity--> High Quality 0.4741461988304093 Low_Quality 0.589502688172043
```

```
temp_white=white_wine[white_wine['quality']>5.5]['volatile acidity']
temp1_white=white_wine[white_wine['quality']<=5.5]['volatile acidity']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 0.965 Low_Quality 1.1
```

```
temp_red=red_wine[red_wine['quality']>5.5]['chlorides']
temp1_red=red_wine[red_wine['quality']<=5.5]['chlorides']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

```
Quality Vs Volatile Acidity--> High Quality 0.0826608187134503 Low_Quality 0.09298924731182795
```

```
temp_white=white_wine[white_wine['quality']>5.5]['chlorides']
temp1_white=white_wine[white_wine['quality']<=5.5]['chlorides']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 0.255 Low_Quality 0.346
```

```
temp_red=red_wine[red_wine['quality']>5.5]['sulphates']
temp1_red=red_wine[red_wine['quality']<=5.5]['sulphates']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

```
Quality Vs Volatile Acidity--> High Quality 0.6926198830409357 Low_Quality 0.6185349462365591
```

```
temp_white=white_wine[white_wine['quality']>5.5]['sulphates']
temp1_white=white_wine[white_wine['quality']<=5.5]['sulphates']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 1.08 Low_Quality 0.88
```

```
temp_red=red_wine[red_wine['quality']>5.5]['alcohol']
temp1_red=red_wine[red_wine['quality']<=5.5]['alcohol']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

```
Quality Vs Volatile Acidity--> High Quality 10.85502923976608 Low_Quality 9.926478494623655
```

```
temp_white=white_wine[white_wine['quality']>5.5]['alcohol']
temp1_white=white_wine[white_wine['quality']<=5.5]['alcohol']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 14.2 Low_Quality 13.6
```

```
temp_red=red_wine[red_wine['quality']>5.5]['residual sugar']
temp1_red=red_wine[red_wine['quality']<=5.5]['residual sugar']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

```
Quality Vs Volatile Acidity--> High Quality 2.5359649122807015 Low_Quality 2.5420698924731187
```

```
temp_white=white_wine[white_wine['quality']>5.5]['residual sugar']
temp1_white=white_wine[white_wine['quality']<=5.5]['residual sugar']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 65.8 Low_Quality 23.5
```

```
temp_red=red_wine[red_wine['quality']>5.5]['total sulfur dioxide']
temp1_red=red_wine[red_wine['quality']<=5.5]['total sulfur dioxide']
print('Quality Vs Volatile Acidity-->','High Quality',temp_red.mean(),'Low_Quality',temp1_red.mean())
```

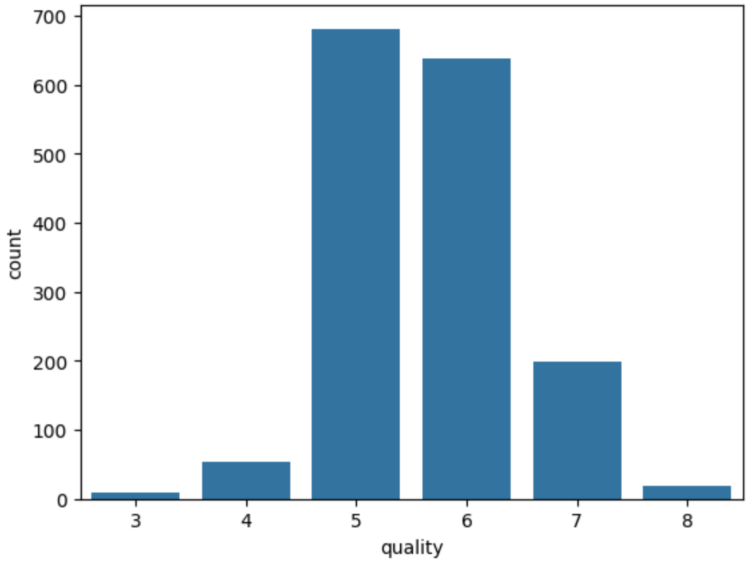
```
Quality Vs Volatile Acidity--> High Quality 39.35204678362573 Low_Quality 54.645161290322584
```

```
temp_white=white_wine[white_wine['quality']>5.5]['total sulfur dioxide']
temp1_white=white_wine[white_wine['quality']<=5.5]['total sulfur dioxide']
print('Quality Vs Volatile Acidity-->','High Quality',temp_white.max(),'Low_Quality',temp1_white.max())
```

```
Quality Vs Volatile Acidity--> High Quality 294.0 Low_Quality 440.0
```

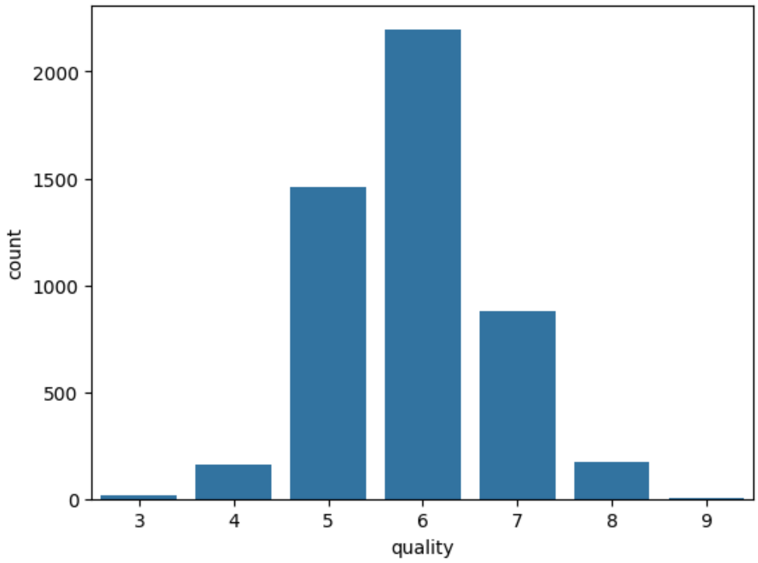
```
print(red_wine['quality'].value_counts())
print('----------------------------------------')
print(white_wine['quality'].value_counts())
print('----------------------------------------')
print(sns.countplot(x='quality',data=red_wine))
print('------------------------------------------------')
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
----------------------------------------
6    2198
5    1457
7     880
8     175
4     163
3      20
9       5
Name: quality, dtype: int64
-----------------------------------------
Axes(0.125,0.11;0.775x0.77)
-------------------------------------------------
```



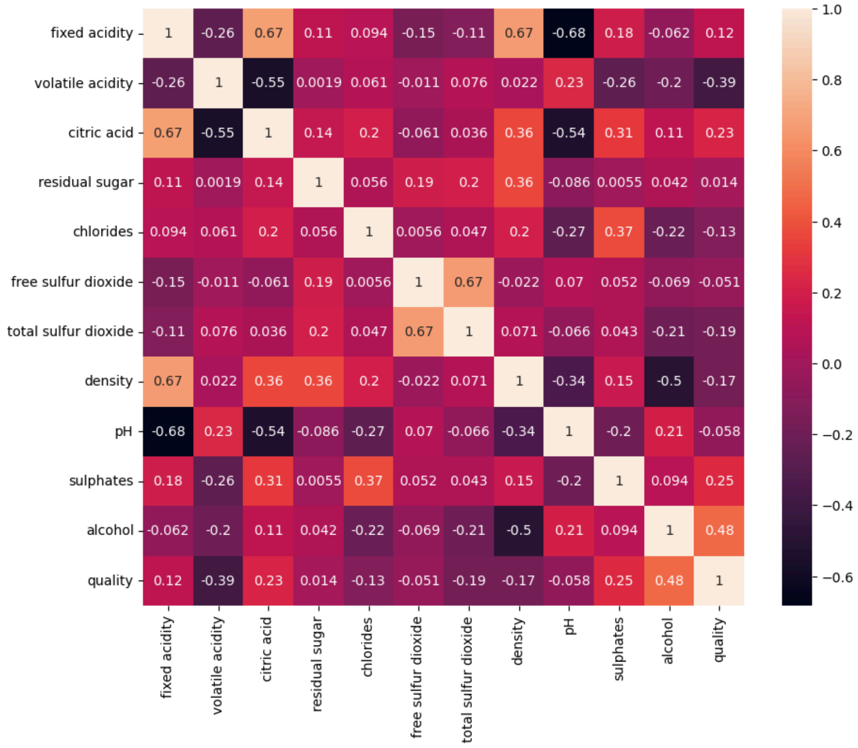

```
print(sns.countplot(x='quality',data=white_wine))
```

```
Axes(0.125,0.11;0.775x0.77)
```




```
plt.figure(figsize=(10,8))
#Correlation Matrix
sns.heatmap(red_wine.corr(),annot=True)
```

<Axes: >




```
#red_wine.corr().iloc[1,3]
#red_wine.corr()columns
abs(red_wine.corr().iloc[1,5])
```

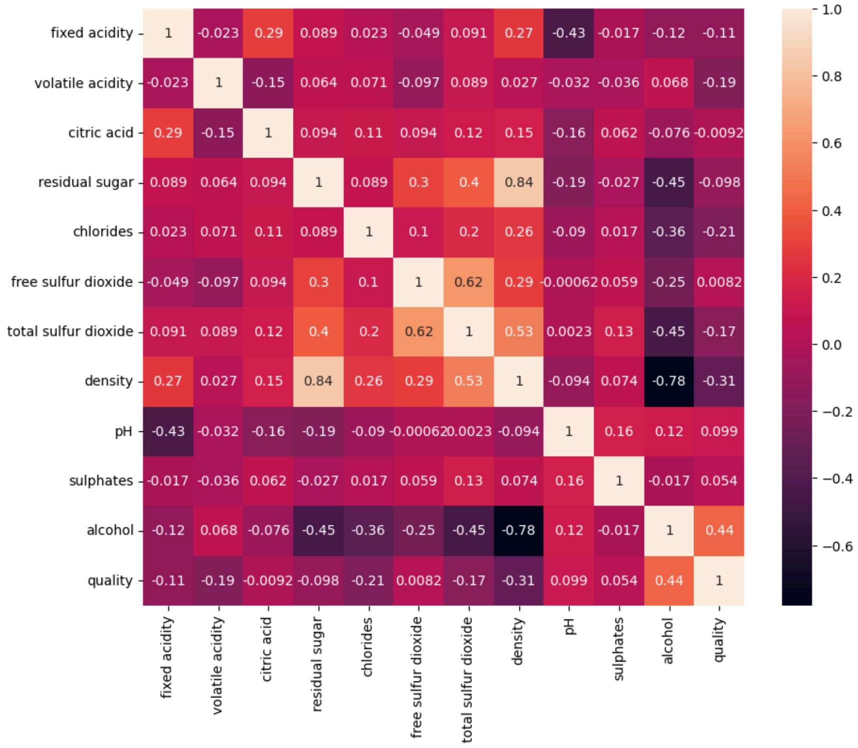
```
0.01050382700659221
```

```
for i in range(0,len(red_wine.columns)):
  for j in range(0,len(red_wine.columns)):
    if abs(red_wine.corr().iloc[i,j])>=0.5 and i!=j:
      print(red_wine.columns[i],",",red_wine.columns[j],(i,j))
```

```
fixed acidity , citric acid (0, 2)
fixed acidity , density (0, 7)
fixed acidity , pH (0, 8)
volatile acidity , citric acid (1, 2)
citric acid , fixed acidity (2, 0)
citric acid , volatile acidity (2, 1)
citric acid , pH (2, 8)
free sulfur dioxide , total sulfur dioxide (5, 6)
total sulfur dioxide , free sulfur dioxide (6, 5)
density , fixed acidity (7, 0)
pH , fixed acidity (8, 0)
pH , citric acid (8, 2)
```

```
#name=[]
#for a in range(len(red_wine.corr().columns)):
# for b in range(a):
#  if abs(red_wine.corr().iloc[a,b])>=0.5:
#   name.append((a,b))
#   name=red_wine.corr().columns[a]
#print(name)
```

```
plt.figure(figsize=(10,8))
sns.heatmap(white_wine.corr(),annot=True)
plt.show()
```




```
for i in range(0,len(white_wine.columns)):
  for j in range(0,len(white_wine.columns)):
    if abs(white_wine.corr().iloc[i,j])>=0.5 and i!=j:
      print(white_wine.columns[i],",",white_wine.columns[j],(i,j))
```

```
residual sugar , density (3, 7)
free sulfur dioxide , total sulfur dioxide (5, 6)
total sulfur dioxide , free sulfur dioxide (6, 5)
total sulfur dioxide , density (6, 7)
density , residual sugar (7, 3)
density , total sulfur dioxide (7, 6)
density , alcohol (7, 10)
alcohol , density (10, 7)
```

```
#Correlated features can be avoided in the final stage of model building.
#Categorization based on quality.
```

```
red_wine['rating']=red_wine['quality'].apply(lambda x: 'Good' if x>=6 else 'Cheap')
red_wine.head()
```

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

Next steps: Generate code with red_wine View recommended plots

```
white_wine['rating']=white_wine['quality'].apply(lambda x: 'Good' if x>=0 else 'Cheap')
white_wine.head()
```

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

Next steps: Generate code with white_wine View recommended plots

```
enc=LabelEncoder()
red_wine['target']=enc.fit_transform(red_wine['rating'])
red_wine.head()
```

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

Next steps: Generate code with red_wine View recommended plots

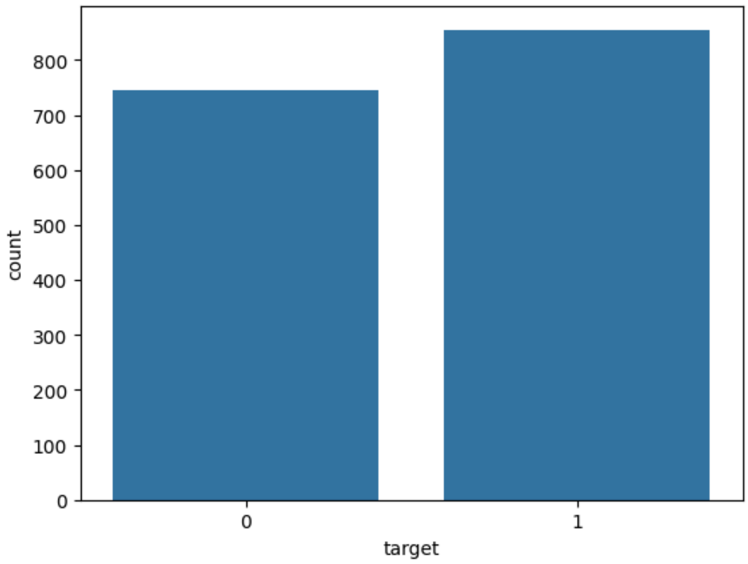
```
enc=LabelEncoder()
white_wine['target']=enc.fit_transform(white_wine['rating'])
white_wine.head()
```

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

Next steps: Generate code with white_wine View recommended plots

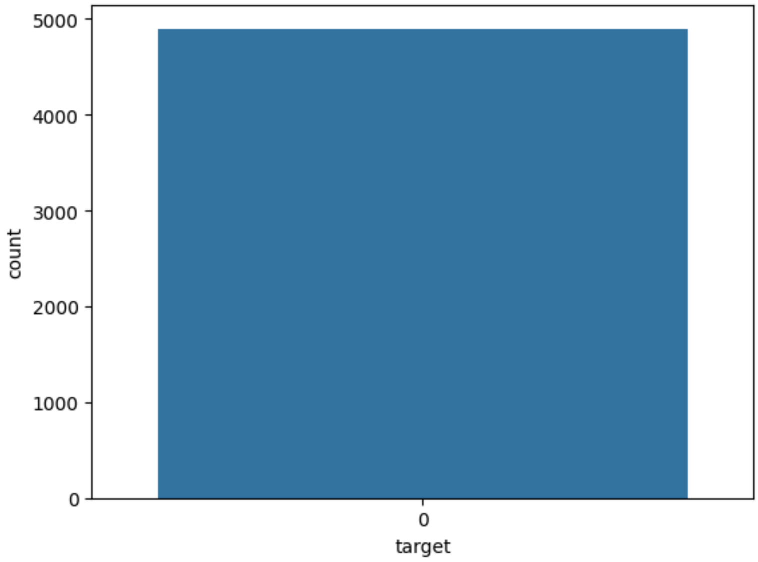
```
sns.countplot(x=red_wine['target'],data=red_wine)
print(red_wine['target'].value_counts())
```

```
1    855
0    744
Name: target, dtype: int64
```



```
sns.countplot(x=white_wine['target'],data=white_wine)
print(white_wine['target'].value_counts())
```

```
0    4898
Name: target, dtype: int64
```




```
#Count of white wine shows that there is a need to balance the dataset.
```

```
good_quality=white_wine[white_wine['target']==1]
bad_quality=white_wine[white_wine['target']==0]
good_quality=good_quality.sample(frac=1)
good_quality=good_quality[:1640]
new_white=pd.concat([good_quality,bad_quality])
new_white=new_white.sample(frac=1)
new_white
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
1461	6.4	0.145	0.49	5.4	0.048	54.0	164.0	0.99460	3.56	0.4
1659	6.6	0.325	0.49	7.7	0.049	53.0	217.0	0.99600	3.16	0.4
4413	5.8	0.300	0.38	4.9	0.039	22.0	86.0	0.98963	3.23	0.5
4194	6.7	0.200	0.24	6.5	0.044	28.0	100.0	0.99348	3.12	0.3
1349	9.2	0.350	0.39	0.9	0.042	15.0	61.0	0.99240	2.96	0.2
...	...	...	...	...	...	...	...	...	...	..
3544	6.0	0.330	0.20	1.8	0.031	49.0	159.0	0.99190	3.41	0.5
917	7.7	0.300	0.32	1.6	0.037	23.0	124.0	0.99190	2.93	0.3
4212	6.6	0.310	0.37	6.2	0.052	13.0	164.0	0.99602	3.24	0.3
3883	6.8	0.330	0.31	7.4	0.045	34.0	143.0	0.99226	3.06	0.5
538	6.1	0.240	0.30	1.5	0.045	22.0	61.0	0.99200	3.31	0.5

Next steps: Generate code with new_white | View recommended plots

```
new_white['target'].value_counts()
```

```
0    4898
Name: target, dtype: int64
```

```
#Split into training & testing
#Dropped fixed activity, pH, free sulfur dioxide from red wine
#Dropped density, free sulfur dioxide from white wine
```

```
Xr=red_wine.drop(['quality','target','rating','fixed acidity','pH','free sulfur dioxide'],axis=1)
yr=red_wine['target']
Xr_train,Xr_test,yr_train,yr_test=train_test_split(Xr,yr,test_size=0.3,random_state=100)
```

```
Xw=white_wine.drop(['quality','target','rating','fixed acidity','pH','free sulfur dioxide'],axis=1)
yw=white_wine['target']
Xw_train,Xw_test,yw_train,yw_test=train_test_split(Xw,yw,test_size=0.3,random_state=100)
```

```
len(yw_test)
```

```
1470
```

```
Xr_train.head()
```

	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	sulphates	alcohol
858	0.28	0.47	1.70	0.054	32.0	0.99686	0.67	10.6
654	0.47	0.47	2.40	0.074	29.0	0.99790	0.46	9.5
721	0.48	0.24	2.85	0.094	106.0	0.99820	0.53	9.2
176	0.38	0.21	2.00	0.080	35.0	0.99610	0.47	9.5

Next steps: Generate code with Xr_train View recommended plots

```
#Creating a normalization object
norm=MinMaxScaler()
norm_xrtrain=norm.fit_transform(Xr_train)
norm_xrtest=norm.transform(Xr_test)
#display values
print(norm_xrtrain)
```

```
[[0.10958904 0.47       0.05479452 ... 0.49853157 0.18404908 0.39285714]
 [0.23972603 0.47       0.10273973 ... 0.57488987 0.05521472 0.19642857]
 [0.24657534 0.24       0.13356164 ... 0.5969163  0.09815951 0.14285714]
 ...
 [0.34589041 0.2        0.04109589 ... 0.52349486 0.4601227  0.125     ]
 [0.33561644 0.02       0.10958904 ... 0.54185022 0.14110429 0.23214286]
 [0.17123288 0.43       0.09589041 ... 0.39867841 0.26993865 0.5       ]]
```

```
#Creating a normalization object
norm=MinMaxScaler()
norm_xwtrain=norm.fit_transform(Xw_train)
norm_xwtest=norm.transform(Xw_test)
#display values
print(norm_xwtrain)
```

```
[[0.14054054 0.22289157 0.01533742 ... 0.10198573 0.30232558 0.46774194]
 [0.07567568 0.1686747  0.04754601 ... 0.08077887 0.05813953 0.51612903]
 [0.20540541 0.28915663 0.08435583 ... 0.16483516 0.3255814  0.22580645]
 ...
 [0.04324324 0.21686747 0.02300613 ... 0.08270677 0.60465116 0.5483871 ]
 [0.4972973  0.1686747  0.07361963 ... 0.15018315 0.1744186  0.16129032]
 [0.15135135 0.18674699 0.0506135  ... 0.0746096  0.10465116 0.56451613]]
```

```
print(yr.value_counts())
print('-------------------')
print(yw.value_counts())
```

```
1    855
0    744
Name: target, dtype: int64
-------------------
0    4898
Name: target, dtype: int64
```

```
modeltypes={
    'LR':LogisticRegression(solver='liblinear'),
    'RFC':RandomForestClassifier(n_estimators=100),
    'KNN':KNeighborsClassifier(n_neighbors=8,algorithm='kd_tree'),
    'NBC':GaussianNB(),
    'XGBoosst':XGBClassifier(n_estimators=250,learning_rate=0.25,use_label_encoder=False)
}
```

```
#model=DecisionTreeClassifier()
#scores=cross_val_score(model,X,y,cv=skfold)
#print(np.mean(scores))
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
skfold=StratifiedKFold(n_splits=5)
```

```
def fit_and_score(modeltypes,X_train,y_train,X_test,y_test):
  model_scores={}
  scores={}
  trained_model={}
  for name,model in modeltypes.items():
    scores[name]=cross_val_score(model,X_train,y_train,cv=skfold)
    #fit model to the data
    trained_model[name]=model.fit(X_train,y_train)
    #y_predict[name]=model.predict(X_test)
    #Evaluate the model & append its score to model_scores
    model_scores[name]=model.score(X_test,y_test)
  return model_scores,trained_model,scores
```

```
m_scores,train_model,accuracy=fit_and_score(modeltypes=modeltypes,
                                            X_train=Xr_train,
                                            y_train=yr_train,
                                            X_test=Xr_test,
                                            y_test=yr_test)

print(m_scores)
```

```
{'LR': 0.7354166666666667, 'RFC': 0.7958333333333333, 'KNN': 0.6729166666666667, 'NBC': 0.7354166666666667, 'XGBoosst': 0.78125}
```

```
print(accuracy)
```

```
{'LR': array([0.79910714, 0.73214286, 0.71428571, 0.78125   , 0.70852018]), 'RFC': array([0.80803571, 0.77232143, 0.75      , 0.80803571, 0.70852018]), 'KNN': ar
```

```
xgb=XGBClassifier(n_estimators=100,learning_rate=0.25,use_label_encoder=False)
xgb.fit(Xr_train,yr_train,eval_set=[(Xr_test,yr_test)],early_stopping_rounds=None,verbose=False)
score_xgb=xgb.score(Xr_test,yr_test)
score_xgb
```

```
0.7875
```

```
xgb=XGBClassifier(n_estimators=100,learning_rate=0.25,use_label_encoder=False)
xgb.fit(Xw_train,yw_train,eval_set=[(Xw_test,yw_test)],early_stopping_rounds=None,verbose=False)
score_xgb=xgb.score(Xw_test,yw_test)
score_xgb
```

```
1.0
```

```
xgbred=XGBClassifier(n_estimators=100,learning_rate=0.25,use_label_encoder=False)
xgbred.fit(Xr_train,yr_train)
xr_predict=xgbred.predict(Xr_test)
red_predicted_df={'predicted_values':xr_predict,'original_values':yr_test}
#Creating a new Dataframe
pd.DataFrame(red_predicted_df).head()
```

	predicted_values	original_values
1254	1	0
1087	1	1
822	0	0
1514	0	1
902	1	1

```
xgbwhite=XGBClassifier(n_estimators=100,learning_rate=0.25,use_label_encoder=False)
xgbwhite.fit(Xw_train,yw_train)
xw_predict=xgbwhite.predict(Xw_test)
white_predicted_df={'predicted_values':xw_predict,'original_values':yw_test}
#Creating a new Dataframe
pd.DataFrame(white_predicted_df).head()
```

	predicted_values	original_values
828	0	0
1621	0	0
3091	0	0
2010	0	0
1433	0	0

```
print(classification_report(yr_test,xr_predict))
print(classification_report(yw_test,xw_predict))
print(confusion_matrix(yr_test,xr_predict))
print(confusion_matrix(yw_test,xw_predict))
```

```
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       224
           1       0.80      0.80      0.80       256

    accuracy                           0.79       480
   macro avg       0.79      0.79      0.79       480
weighted avg       0.79      0.79      0.79       480

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1470

    accuracy                           1.00      1470
   macro avg       1.00      1.00      1.00      1470
weighted avg       1.00      1.00      1.00      1470

[[174  50]
 [ 52 204]]
[[1470]]
```

```
param={'n_estimators':[50,100,500]}
grid_rf=GridSearchCV(RandomForestClassifier(),param,scoring='accuracy',cv=10)
grid_rf.fit(Xr_train,yr_train)
print('Best parameters -->',grid_rf.best_params_)
#Wine Quality Prediction
pred=grid_rf.predict(Xr_test)
print(confusion_matrix(yr_test,pred))
print('\n')
print(classification_report(yr_test,pred))
print('\n')
print(accuracy_score(yr_test,pred))
```

```
Best parameters --> {'n_estimators': 500}
[[173  51]
 [ 42 214]]


              precision    recall  f1-score   support

           0       0.80      0.77      0.79       224
           1       0.81      0.84      0.82       256

    accuracy                           0.81       480
   macro avg       0.81      0.80      0.80       480
weighted avg       0.81      0.81      0.81       480



0.80625
```

```
import pickle
pickle.dump(xgb,open('wine_quality','wb'))
model=pickle.load(open('wine_quality','rb'))
model
```

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.25, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
#ML Class 21
#Face Detection
```

```
#To find XML files
import os
import cv2
```

```
# Load the cascade
cascPathface = os.path.dirname(cv2.__file__) + '/haarcascade_frontalface_alt2.xml'
face_cascade = cv2.CascadeClassifier(cascPathface)

# Read the input image
img = cv2.imread('frnds.webp')

# Convert into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Check if the grayscale image is empty
if gray.size == 0:
    print('The input image is empty or could not be read.')
    exit()

# Detect faces of different sized in the input image
faces = face_cascade.detectMultiScale(gray, 1.1, 4)

# Draw rectangle around the faces
for (x, y, w, h) in faces:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 0, 255), 5)

# Export the result
cv2.imwrite('face_detected.png', img)
print('Suucessfully saved')

# Display the output
cv2.imshow('img', img)
cv2.waitKey()

# De-allocate any associated memory usage
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-65-e0ae8667ce07> in <cell line: 9>()
      7
      8 # Convert into grayscale
----> 9 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
     10
     11 # Check if the grayscale image is empty

error: OpenCV(4.8.0) /io/opencv/modules/imgproc/src/color.cpp:182: error: (-215:Assertion failed)
!_src.empty() in function 'cvtColor'
```

Next steps: Explain error

```
#ML Class 22
#Music Recommendation System
#MusicRecommender_KNN
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#Last fm is a music discovery service that gives you personalised recommends based on the music you listen to.
#Here we are going to do some machine learning & data analysis on the dataset of last.fm inorder to recommend the next song to the user.
```

```
musicdt=pd.read_csv('LastFM_Matrix.csv')
musicdt
```

	user	a perfect circle	abba	ac/dc	adam green	aerosmith	afi	air	alanis morissette	alexisonfire	...
0	1	0	0	0	0	0	0	0	0	0	...
1	33	0	0	0	1	0	0	0	0	0	...
2	42	0	0	0	0	0	0	0	0	0	...
3	51	0	0	0	0	0	0	0	0	0	...
4	62	0	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...
1252	19639	0	0	1	1	1	0	0	0	0	...
1253	19642	0	0	0	0	0	0	0	0	0	...
1254	19662	0	0	1	0	0	0	0	0	0	...
1255	19672	0	0	0	0	0	0	0	0	0	...
1256	19695	0	0	0	1	0	0	0	0	0	...

1257 rows × 286 columns

```
print(musicdt[musicdt['abba']==1]['user'].count())
print('-------------------------------------------------')
print(musicdt['user'].nunique())
print('-------------------------------------------------')
mdt=musicdt.drop(['user'],axis=1)
print(mdt)
print('-------------------------------------------------')
print(mdt.shape)
```

```
37
-------------------------------------------------
1257
-------------------------------------------------
      a perfect circle  abba  ac/dc  adam green  aerosmith  afi  air  \
0                    0     0      0           0          0    0    0
1                    0     0      0           1          0    0    0
2                    0     0      0           0          0    0    0
3                    0     0      0           0          0    0    0
4                    0     0      0           0          0    0    0
...                ...   ...    ...         ...        ...  ...  ...
1252                 0     0      1           1          1    0    0
1253                 0     0      0           0          0    0    0
1254                 0     0      1           0          0    0    0
1255                 0     0      0           0          0    0    0
1256                 0     0      0           1          0    0    0

      alanis morissette  alexisonfire  alicia keys  ...  timbaland  tom waits  \
0                     0             0            0  ...          0          0
1                     0             0            0  ...          0          0
2                     0             0            0  ...          0          0
3                     0             0            0  ...          0          0
4                     0             0            0  ...          0          0
...                 ...           ...          ...  ...        ...        ...
1252                  0             0            0  ...          0          0
1253                  0             0            0  ...          0          0
1254                  0             0            0  ...          0          0
1255                  0             0            0  ...          0          0
1256                  0             0            0  ...          0          1

      tool  tori amos  travis  trivium  u2  underoath  volbeat  yann tiersen
0        0          0       0        0   0          0        0             0
1        0          0       0        0   0          0        0             0
2        0          0       0        0   0          0        0             0
3        0          0       0        0   0          0        0             0
4        0          0       0        0   0          0        0             0
...    ...        ...     ...      ...  ..        ...      ...           ...
1252     0          0       0        0   0          0        0             0
1253     1          0       0        0   0          0        0             0
1254     0          0       0        0   0          0        0             0
1255     0          0       0        0   0          0        0             0
1256     0          0       0        0   0          0        0             0

[1257 rows x 285 columns]
-------------------------------------------------
(1257, 285)
```

```
#For calculating similarity matrix between different rows
from sklearn.metrics.pairwise import cosine_similarity
#Transpose, to get similarity between the songs
data_similarity=cosine_similarity(mdt.T)
data_similarity
```

```
array([[1.        , 0.        , 0.01791723, ..., 0.06506   , 0.05216405,
        0.        ],
       [0.        , 1.        , 0.05227877, ..., 0.        , 0.02536731,
        0.        ],
       [0.01791723, 0.05227877, 1.        , ..., 0.02039967, 0.13084898,
        0.        ],
       ...,
       [0.06506   , 0.        , 0.02039967, ..., 1.        , 0.        ,
        0.        ],
       [0.05216405, 0.02536731, 0.13084898, ..., 0.        , 1.        ,
        0.02969569],
       [0.        , 0.        , 0.        , ..., 0.        , 0.02969569,
        1.        ]])
```

```
print(data_similarity.shape)
print('--------------------------')
mdt1=pd.DataFrame(data_similarity,columns=mdt.columns,index=mdt.columns)
print(mdt1)
print('--------------------------')
#To check for duplicate songs/rows
print(mdt1.index.nunique())

#The principle behind nearest neighbor methods is to find a predefined number of training samples closest
#in distance to the new point, & predict the label from these. The number of samples can be user-defined
#constant(k-nearest neighbor learning), or vary based on the local density of points(radius-based neighbor learning)
#The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.
#Neighbors-based methods are known as non-generalizing machine learning methods, since they simply
#'remember' all of the training data(possibly transformed into a fast indexing structure)
#known as a Bali Tree or KD Tree). The classes in sklearn.neighbors can handle either NumPy
#arrays or sdpy.sparse matricies as input. For dense matricies, a large number of possible distance metrics
#are supported. For spare matricies, arbitary Minkowski metrics are supported for searches.
```

```
volbeat              0.052164  0.025367  0.130849  0.023531  0.057307
yann tiersen         0.000000  0.000000  0.000000  0.088045  0.000000

                      afi       air  alanis morissette  alexisonfire  \
a perfect circle  0.000000  0.051755           0.060718      0.000000
abba              0.000000  0.016779           0.029527      0.000000
ac/dc             0.067894  0.075730           0.038076      0.000000
adam green        0.000000  0.093386           0.000000      0.000000
aerosmith         0.000000  0.113715           0.100056      0.000000
...                    ...       ...                ...           ...
trivium           0.052650  0.000000           0.000000      0.085855
u2                0.023113  0.073657           0.103695      0.000000
underoath         0.154083  0.000000           0.000000      0.301511
volbeat           0.024708  0.031497           0.000000      0.000000
yann tiersen      0.030817  0.078567           0.000000      0.000000

                  alicia keys  ...  timbaland  tom waits      tool  tori amos  \
a perfect circle     0.000000  ...   0.047338   0.081200  0.394709   0.125553
abba                 0.000000  ...   0.000000   0.000000  0.000000   0.061056
ac/dc                0.088333  ...   0.044529   0.067894  0.058241   0.039367
adam green           0.025416  ...   0.000000   0.146516  0.083789   0.056637
aerosmith            0.061898  ...   0.052005   0.029735  0.025507   0.068966
...                       ...  ...        ...        ...       ...        ...
trivium              0.000000  ...   0.046041   0.000000  0.067746   0.000000
u2                   0.024056  ...   0.020211   0.069338  0.039653   0.080408
underoath            0.000000  ...   0.026948   0.030817  0.052870   0.035737
volbeat              0.000000  ...   0.021607   0.024708  0.063586   0.000000
yann tiersen         0.000000  ...   0.000000   0.123267  0.052870   0.035737

                    travis   trivium        u2  underoath   volbeat  \
a perfect circle  0.030359  0.111154  0.024398   0.065060  0.052164
abba              0.029527  0.000000  0.094916   0.000000  0.025367
ac/dc             0.000000  0.087131  0.122398   0.020400  0.130849
adam green        0.082169  0.025071  0.022011   0.000000  0.023531
aerosmith         0.033352  0.000000  0.214423   0.000000  0.057307
...                    ...       ...       ...        ...       ...
trivium           0.029527  1.000000  0.023729   0.126554  0.050735
```

```
trivium          0.000000
u2               0.000000
underoath        0.000000
volbeat          0.029696
yann tiersen     1.000000

[285 rows x 285 columns]
--------------------------
285
```

```
from sklearn.neighbors import NearestNeighbors
m_model=NearestNeighbors(n_neighbors=285)
m_model.fit(mdt1)
```

```
▾ NearestNeighbors
NearestNeighbors(n_neighbors=285)
```

```
#Predicted distances
indices=m_model.kneighbors(mdt1,return_distance=False)
```

```
indices
```

```
array([[  0, 277,  81, ...,  57,  32, 218],
       [  1, 221,  88, ..., 150, 125, 241],
       [  2, 128, 172, ..., 261,  17,  32],
       ...,
       [282,   8,  22, ..., 216,  19,  60],
       [283,  36, 196, ...,  39, 253,  60],
       [284,  54, 239, ..., 241, 125, 162]])
```

```
mdt2=pd.DataFrame(indices)
mdt2
```

	0	1	2	3	4	5	6	7	8	9	...	275	276	277	278	279	280
0	0	277	81	70	189	206	108	235	264	80	...	216	147	60	90	159	25
1	1	221	88	165	174	175	83	208	113	103	...	230	33	213	172	19	7
2	2	128	172	36	190	75	182	116	258	140	...	218	39	263	248	57	6
3	3	255	267	25	276	47	84	104	266	59	...	213	11	90	20	238	7
4	4	281	157	158	115	93	106	78	103	262	...	253	10	19	162	22	24
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
280	280	10	20	150	52	119	164	232	56	230	...	57	211	32	218	98	1
281	281	210	72	221	50	4	106	208	192	247	...	99	119	10	90	22	1
282	282	8	22	91	227	105	48	270	163	5	...	90	213	39	218	211	25
283	283	36	196	182	229	140	154	177	220	115	...	19	32	98	218	68	25
284	284	54	239	168	276	259	86	43	242	256	...	238	230	33	79	150	21

285 rows × 285 columns

```
mdt1.columns[0]
mdt1.index
```

```
Index(['a perfect circle', 'abba', 'ac/dc', 'adam green', 'aerosmith', 'afi',
       'air', 'alanis morissette', 'alexisonfire', 'alicia keys',
       ...
       'timbaland', 'tom waits', 'tool', 'tori amos', 'travis', 'trivium',
       'u2', 'underoath', 'volbeat', 'yann tiersen'],
      dtype='object', length=285)
```

```
#Gives names with respect to songs
final_model=pd.DataFrame(mdt1.columns[mdt2],index=mdt1.index)
final_model.head()
```

```
<ipython-input-79-c86700c31b30>:2: FutureWarning: Support for multi-dimensional indexing (e.g. `obj[
  final_model=pd.DataFrame(mdt1.columns[mdt2],index=mdt1.index)
```

	0	1	2	3	4	5	6	
a perfect circle	a perfect circle	tool	dredg	deftones	nine inch nails	porcupine tree	godsmack	
abba	abba	robbie williams	elvis presley	madonna	michael jackson	mika	duffy	
ac/dc	ac/dc	iron maiden	metallica	black sabbath	nirvana	die toten hosen	motorhead	ha
adam green	adam green	the libertines	the strokes	babyshambles	tom waits	bright eyes	editors	f
aerosmith	aerosmith	u2	led zeppelin	lenny kravitz	guns n roses	eric clapton	genesis	di

5 rows × 285 columns

```
top10=final_model[list(final_model.columns[:11])]
top10
```

	0	1	2	3	4	5	
a perfect circle	a perfect circle	tool	dredg	deftones	nine inch nails	porcupine tree	godsmac
abba	abba	robbie williams	elvis presley	madonna	michael jackson	mika	duf
ac/dc	ac/dc	iron maiden	metallica	black sabbath	nirvana	die toten hosen	motorhea
adam green	adam green	the libertines	the strokes	babyshambles	tom waits	bright eyes	edito
aerosmith	aerosmith	u2	led zeppelin	lenny kravitz	guns n roses	eric clapton	genes
...	...	...	...	...	...	...	.
trivium	trivium	all that remains	as i lay dying	killswitch engage	caliban	heaven shall burn	machir hea
u2	u2	r.e.m.	depeche mode	robbie williams	bruce springsteen	aerosmith	genes
underoath	underoath	alexisonfire	atreyu	enter shikari	silverstein	funeral for a friend	bring m th horizc
volbeat	volbeat	black sabbath	opeth	motorhead	slayer	judas priest	kreat

Next steps: Generate code with top10 View recommended plots

```
#Store data in csv file
top10.to_csv('top10.csv',index_label='Index')
```

```
pd.read_csv('top10.csv').head()
```

	Index	0	1	2	3	4	5	6
0	a perfect circle	a perfect circle	tool	dredg	deftones	nine inch nails	porcupine tree	godsmack
1	abba	abba	robbie williams	elvis presley	madonna	michael jackson	mika	duffy
2	ac/dc	ac/dc	iron maiden	metallica	black sabbath	nirvana	die toten hosen	motorhead
3	adam green	adam green	the libertines	the strokes	babyshambles	tom waits	bright eyes	editors

```
#We have created a model which recommends next song user will like to hear by using last.fm Germany data.
```

```
#ML Class 23
#Data Analysis
```

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

```
placement_df=pd.read_csv('Placement_Data.csv')
placement_df
```

	sl_no	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	wo
0	1	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	
1	2	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	
2	3	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	
3	4	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	
4	5	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	
...	...	...	...	...	...	...	...	...	...	
210	211	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	
211	212	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	
212	213	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	
213	214	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	
214	215	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	

Next steps: Generate code with placement_df | View recommended plots

```
placement_df['status'].value_counts()
```

```
Placed        148
Not Placed     67
Name: status, dtype: int64
```

```
placement_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 215 entries, 0 to 214
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sl_no           215 non-null    int64
 1   gender          215 non-null    object
 2   ssc_p           215 non-null    float64
 3   ssc_b           215 non-null    object
 4   hsc_p           215 non-null    float64
 5   hsc_b           215 non-null    object
 6   hsc_s           215 non-null    object
 7   degree_p        215 non-null    float64
 8   degree_t        215 non-null    object
 9   workex          215 non-null    object
 10  etest_p         215 non-null    float64
 11  specialisation  215 non-null    object
```

```
 12  mba_p          215 non-null    float64
 13  status         215 non-null    object
 14  salary         148 non-null    float64
dtypes: float64(6), int64(1), object(8)
memory usage: 25.3+ KB
```

```
placement_df['hsc_s'].unique()
```

```
array(['Commerce', 'Science', 'Arts'], dtype=object)
```

```
placement_df['hsc_b'].unique()
```

```
array(['Others', 'Central'], dtype=object)
```

```
placement_df['specialisation'].unique()
```

```
array(['Mkt&HR', 'Mkt&Fin'], dtype=object)
```

```
placement_df['gender'].unique()
```

```
array(['M', 'F'], dtype=object)
```

```
placement_df['ssc_b'].unique()
```

```
array(['Others', 'Central'], dtype=object)
```

```
placement_df['degree_t'].unique()
```

```
array(['Sci&Tech', 'Comm&Mgmt', 'Others'], dtype=object)
```

```
placement_df['workex'].unique()
```

```
array(['No', 'Yes'], dtype=object)
```

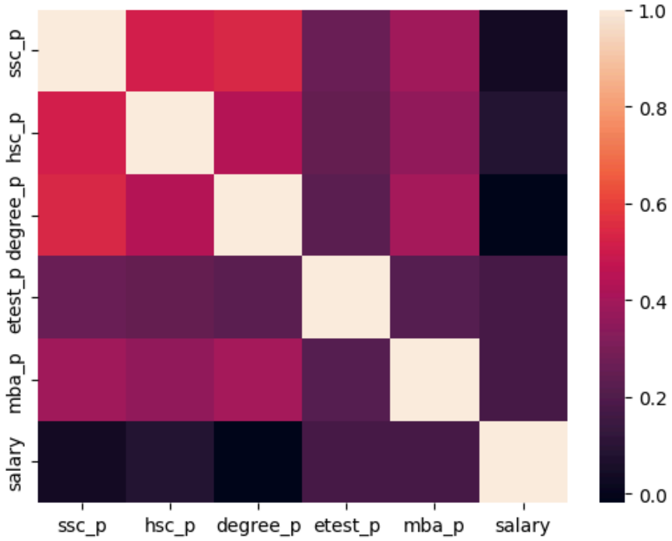
```
#No missing values
#Unwanted column- si_no
placement_df_required=placement_df.drop('sl_no',axis=1)
placement_df_required
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	e
0	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	
1	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	
2	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	
3	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	
4	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	
...	...	...	...	...	...	...	...	...	...	
210	M	80.60	Others	82.00	Others	Commerce	77.60	Comm&Mgmt	No	
211	M	58.00	Others	60.00	Others	Science	72.00	Sci&Tech	No	
212	M	67.00	Others	67.00	Others	Commerce	73.00	Comm&Mgmt	Yes	
213	F	74.00	Others	66.00	Others	Commerce	58.00	Comm&Mgmt	No	
214	M	62.00	Central	58.00	Others	Science	53.00	Comm&Mgmt	No	

Next steps: Generate code with placement_df_required View recommended plots

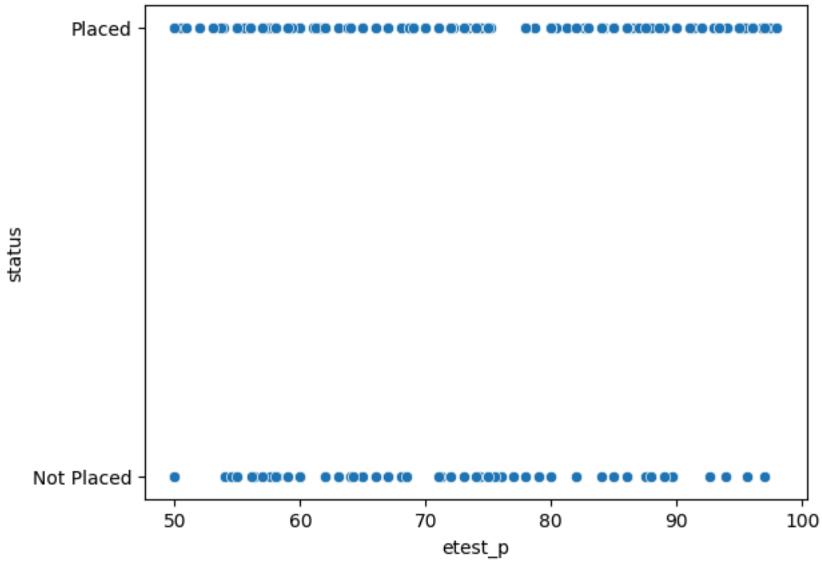
```
sns.heatmap(placement_df_required.corr())
```

```
<ipython-input-99-5bef27fb89c4>:1: FutureWarning: The default value of numeric_only in DataFrame.c
  sns.heatmap(placement_df_required.corr())
<Axes: >
```



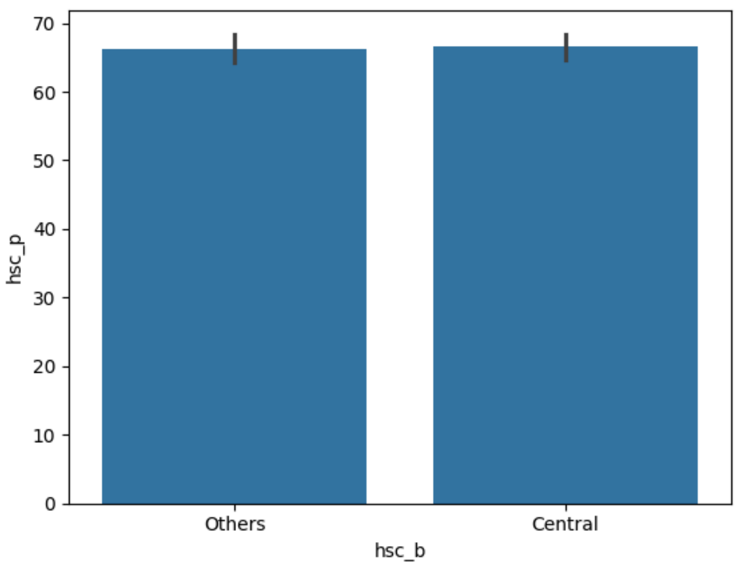

```
sns.scatterplot(x=placement_df_required['etest_p'],y=placement_df_required['status'])
```

```
<Axes: xlabel='etest_p', ylabel='status'>
```

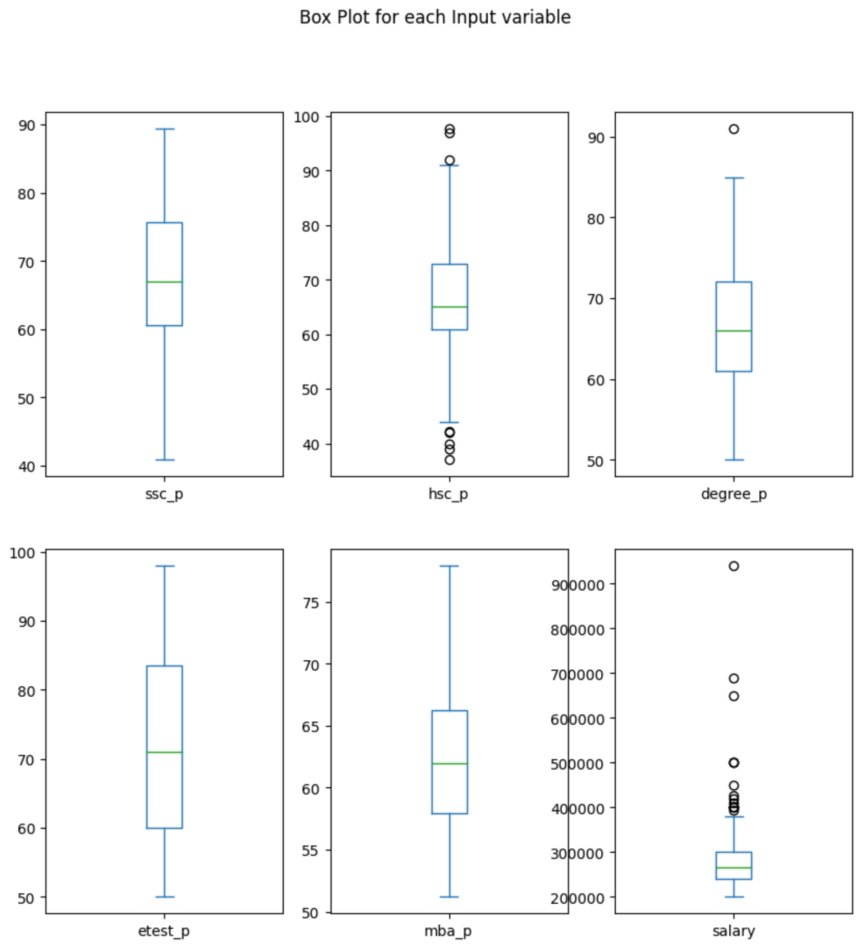



```
sns.barplot(x=placement_df_required['hsc_b'],y=placement_df_required['hsc_p'])
```

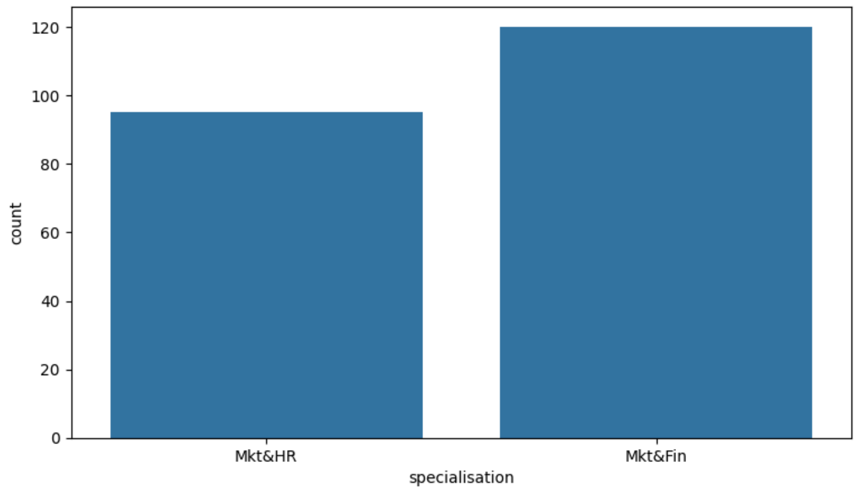
```
<Axes: xlabel='hsc_b', ylabel='hsc_p'>
```



```
placement_df_required.plot(kind='box',subplots=True,layout=(2,3),sharex=False,sharey=False,figsize=(10,10),title='Box Plot for each Input variable')
plt.savefig('placement_boxplot')
plt.show()
```

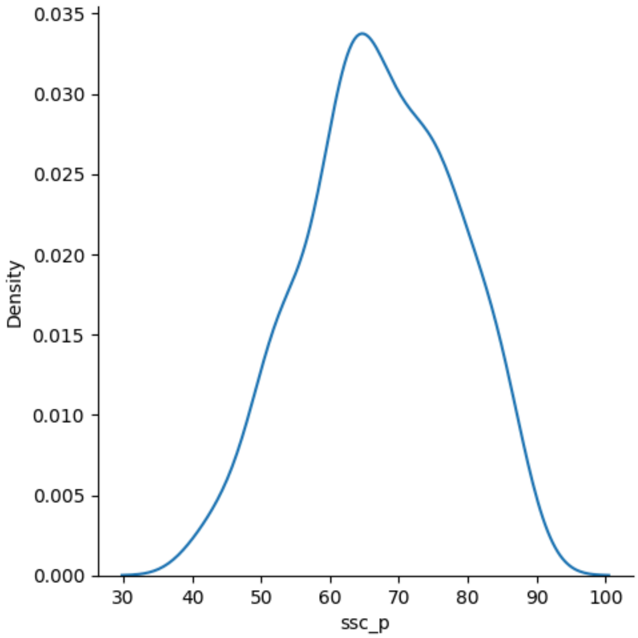



```
#Specialisation Distribution
plt.figure(figsize=(9,5))
sns.countplot(x=placement_df_required.specialisation)
plt.show()
```

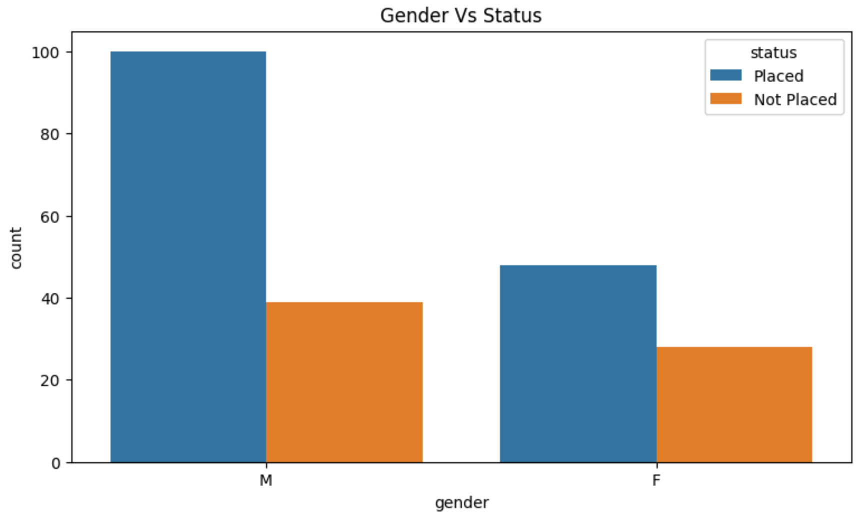



```
plt.figure(figsize=(9,5))
sns.displot(placement_df_required.ssc_p,kind='kde')
plt.show()
```

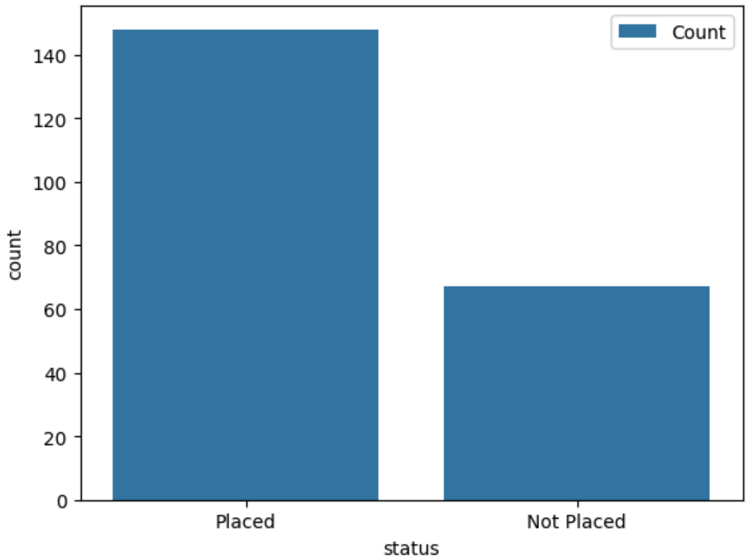
<Figure size 900x500 with 0 Axes>




```
plt.figure(figsize=(9,5))
sns.countplot(x='gender',hue='status',data=placement_df_required)
plt.title('Gender Vs Status')
plt.show()
```




```
sns.countplot(x=placement_df_required['status'],label='Count')
plt.show()
```




```
#Dataset is not a balanced one, which needs to be addressed
#Need to do encoding of categorical fields
#Pre-Processing
```

```
import pandas as pd
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
```

```
placement_df=pd.read_csv('Placement_Data.csv')
placement_df_required=placement_df.drop('sl_no',axis=1)
placement_df_required.head()
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	ete
0	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	
1	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	
2	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	
3	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	

Next steps: Generate code with placement_df_required View recommended plots

```
status_placed=placement_df_required[placement_df_required['status']=='Placed']
status_not_placed=placement_df_required[placement_df_required['status']=='Not Placed']
status_not_placed.shape
```

(67, 14)

```
predictor_df=placement_df_required.drop('status',axis=1)
target_df=placement_df_required[['status']]
target_df
```

	status
0	Placed
1	Placed
2	Placed
3	Not Placed
4	Placed
...	...
210	Placed
211	Placed
212	Placed
213	Placed
214	Not Placed

215 rows × 1 columns

Next steps: Generate code with target_df View recommended plots

```
ros=RandomOverSampler(random_state=23)
x_ros,y_ros=ros.fit_resample(predictor_df,target_df)
y_ros.value_counts()
```

```
status
Not Placed    148
Placed        148
dtype: int64
```

```
enc=LabelEncoder()
y_ros['status_binary']=enc.fit_transform(y_ros['status'])
y_ros.head()
```

	status	status_binary
0	Placed	1
1	Placed	1
2	Placed	1
3	Not Placed	0
4	Placed	1

Next steps: Generate code with y_ros View recommended plots

```
#F-95.8--->Central=83.6----->Central=Arts------->73.3------->Comm&Mgmt---->96.8------->Mkt&HR----->95.5------->YES
```

```
enc1=LabelEncoder()
x_ros['workex_binary']=enc1.fit_transform(x_ros['workex'])
x_ros
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	workex	e
0	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	No	
1	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	Yes	
2	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	No	
3	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	
4	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	No	
...	...	...	...	...	...	...	...	...	...	
291	F	63.40	Others	67.20	Others	Commerce	60.00	Comm&Mgmt	No	
292	M	52.00	Central	57.00	Central	Commerce	50.80	Comm&Mgmt	No	
293	M	61.08	Others	50.00	Others	Science	54.00	Sci&Tech	No	
294	M	52.00	Central	63.00	Others	Science	65.00	Sci&Tech	Yes	
295	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	No	

296 rows × 14 columns

Next steps: Generate code with x_ros View recommended plots

```
x_ros.drop('workex',axis=1,inplace=True)
```

```
ordinal_list=['Central','Others']
ct=ColumnTransformer([('ohe',OneHotEncoder(drop='first'),['gender','hsc_s','degree_t','specialisation']),
 ('oe',OrdinalEncoder(categories=[ordinal_list,ordinal_list]),['ssc_b','hsc_b']),],remainder='passthrough')
```

```
x_encoded=ct.fit_transform(x_ros)
x_encoded[1]
```

```
array([1.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 1.000e+00, 0.000e+00,
       0.000e+00, 1.000e+00, 7.933e+01, 7.833e+01, 7.748e+01, 8.650e+01,
       6.628e+01, 2.000e+05, 1.000e+00])
```

```
x_encoded[291]
```

```
array([ 0.  ,  1.  ,  0.  ,  0.  ,  0.  ,  1.  ,  1.  ,  1.  , 63.4 ,
       67.2 , 60.  , 58.06, 69.28,   nan,  0.  ])
```

```
x_encoded
```

```
array([[1.000e+00, 1.000e+00, 0.000e+00, ..., 5.880e+01, 2.700e+05,
        0.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 6.628e+01, 2.000e+05,
        1.000e+00],
       [1.000e+00, 0.000e+00, 0.000e+00, ..., 5.780e+01, 2.500e+05,
        0.000e+00],
       ...,
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 6.569e+01,       nan,
        0.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 5.609e+01,       nan,
        1.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 5.943e+01,       nan,
        0.000e+00]])
```

```
y_ros
```

	status	status_binary
0	Placed	1
1	Placed	1
2	Placed	1
3	Not Placed	0
4	Placed	1
...	...	...
291	Not Placed	0
292	Not Placed	0
293	Not Placed	0
294	Not Placed	0
295	Not Placed	0

296 rows × 2 columns

Next steps: Generate code with y_ros View recommended plots

```
X_train,X_test,y_train,y_test=train_test_split(x_encoded,y_ros[['status_binary']],test_size=0.30,random_state=15)
```

```
X_train[0]
```

```
array([ 1.  , 0.  , 1.  , 0.  , 1.  , 0.  , 1.  , 1.  , 67.  ,
     63.  , 64.  , 60.  , 61.87,  nan,  0.  ])
```

```
y_test.value_counts()
```

```
status_binary
0            46
1            43
dtype: int64
```

```
predictore1_df=pd.read_csv('trial.csv')
predictore1_df
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	etest_p	spe
0	F	82	Central	75	Central	Commerce	76	Comm&Mgmt	54.96	

```
enc.classes_
```

```
array(['Not Placed', 'Placed'], dtype=object)
```

```
predictore1_df['workex_binary']=enc1.transform(predictore1_df['workex'])
predictore1_df.drop('workex',axis=1,inplace=True)
predictore1_df
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	etest_p	spe
0	F	82	Central	75	Central	Commerce	76	Comm&Mgmt	54.96	

```
predictore1_df_encodedp=ct.fit_transform(predictore1_df)
predictore1_df_encodedp
```

```
array([[ 0.  , 0.  , 82.  , 75.  , 76.  , 54.96, 76.  , 1.  ]])
```

```
ct
```

ColumnTransformer

ohe	oe	remainder
OneHotEncoder	OrdinalEncoder	passthrough

```
#predictor_df_encodedpq=ct.transform(predictor_df)
#predictor_df_encodedpq
#predictor_df
```

```
x_ros
```

	gender	ssc_p	ssc_b	hsc_p	hsc_b	hsc_s	degree_p	degree_t	etest_p	s
0	M	67.00	Others	91.00	Others	Commerce	58.00	Sci&Tech	55.00	
1	M	79.33	Central	78.33	Others	Science	77.48	Sci&Tech	86.50	
2	M	65.00	Central	68.00	Central	Arts	64.00	Comm&Mgmt	75.00	
3	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	66.00	
4	M	85.80	Central	73.60	Central	Commerce	73.30	Comm&Mgmt	96.80	
...	...	...	...	...	...	...	...	...	...	
291	F	63.40	Others	67.20	Others	Commerce	60.00	Comm&Mgmt	58.06	
292	M	52.00	Central	57.00	Central	Commerce	50.80	Comm&Mgmt	67.00	
293	M	61.08	Others	50.00	Others	Science	54.00	Sci&Tech	71.00	
294	M	52.00	Central	63.00	Others	Science	65.00	Sci&Tech	86.00	
295	M	56.00	Central	52.00	Central	Science	52.00	Sci&Tech	66.00	

296 rows × 13 columns

Next steps: Generate code with x_ros View recommended plots

```
#Modelling
```

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
#from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB,ComplementNB
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
import pickle
#from processing.pre_processing import X_train,X_test,y_train,y_test
```

```
X_train
```

```
array([[1.000e+00, 0.000e+00, 1.000e+00, ..., 6.187e+01,       nan,
        0.000e+00],
       [0.000e+00, 1.000e+00, 0.000e+00, ..., 6.694e+01,       nan,
        0.000e+00],
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 6.099e+01, 2.750e+05,
        0.000e+00],
       ...,
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 5.272e+01, 2.550e+05,
        1.000e+00],
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 6.098e+01, 2.500e+05,
        1.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 6.022e+01,       nan,
        0.000e+00]])
```

```
y_test
```

	status_binary
86	1
75	0
127	1
47	1
97	0
...	...
249	0
251	0
99	0
280	0
295	0

89 rows × 1 columns

Next steps: Generate code with y_test View recommended plots

```
X_test
```

```
array([[1.000e+00, 1.000e+00, 0.000e+00, ..., 5.703e+01, 2.200e+05,
        0.000e+00],
       [0.000e+00, 1.000e+00, 0.000e+00, ..., 6.700e+01,       nan,
        0.000e+00],
       [0.000e+00, 0.000e+00, 1.000e+00, ..., 5.840e+01, 2.500e+05,
        0.000e+00],
       ...,
       [1.000e+00, 1.000e+00, 0.000e+00, ..., 5.947e+01,       nan,
        0.000e+00],
       [0.000e+00, 1.000e+00, 0.000e+00, ..., 5.729e+01,       nan,
        0.000e+00],
       [1.000e+00, 0.000e+00, 1.000e+00, ..., 5.943e+01,       nan,
        0.000e+00]])
```

```
#pre_processing.py
import pandas as pd
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split

placement_df=pd.read_csv("Campus_Selection.csv")
placement_df_required=placement_df.drop("sl_no",axis=1)
# placement_df_required.head()

status_placed=placement_df_required[placement_df_required['status']=='Placed']
status_not_placed=placement_df_required[placement_df_required['status']=='Not Placed']
# status_not_placed.shape

predictor_df=placement_df_required.drop('status',axis=1)
target_df=placement_df_required[['status']]

ros = RandomOverSampler(random_state=23)
x_ros, y_ros = ros.fit_resample(predictor_df, target_df)
# y_ros.value_counts()

enc=LabelEncoder()
y_ros['status_binary']=enc.fit_transform(y_ros['status'])

enc1=LabelEncoder()
x_ros['workex_binary']=enc1.fit_transform(x_ros['workex'])
x_ros.drop('workex',axis=1,inplace=True)

ordinal_list=['Central','Others']
ct=ColumnTransformer([('ohe',OneHotEncoder(drop='first'),['gender', 'hsc_s', 'degree_t', 'specialisation']),
                ('oe',OrdinalEncoder(categories=[ordinal_list,ordinal_list]),['ssc_b','hsc_b']),
                ],remainder='passthrough')

x_encoded=ct.fit_transform(x_ros)

X_train,X_test,y_train,y_test=train_test_split(x_encoded,y_ros[['status_binary']],test_size=0.30,random_state=15)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test=sc.transform(X_test)

def pre_process_data(predictor_data):
    predictor_data['workex_binary']=enc1.transform(predictor_data['workex'])
    predictor_data_new=predictor_data.drop('workex',axis=1)
    predictor_data_encoded=ct.transform(predictor_data_new)
    X_data_final = sc.transform(predictor_data_encoded)
    return X_data_final
```

```
#sc=StandardScaler()
#X_train=sc.fit_transform(X_train)
```

```
y_train
```

	status_binary
158	0
189	0
67	1
78	1
175	0
...	...
199	1
155	0
156	1
133	1
245	0

207 rows × 1 columns

Next steps: Generate code with y_train View recommended plots

```
X_train
```

```
array([[ 0.69943451, -1.07529066,  1.16287262, ..., -0.95685724,
        -0.05465445, -0.69178857],
       [-1.42972642,  0.92998111, -0.85993942, ...,  1.75481825,
         0.83921027, -0.69178857],
       [ 0.69943451,  0.92998111, -0.85993942, ..., -0.2943413 ,
        -0.20980257, -0.69178857],
       ...,
       [ 0.69943451, -1.07529066,  1.16287262, ...,  0.56441933,
        -1.66784227,  1.44552836],
       [ 0.69943451,  0.92998111, -0.85993942, ..., -0.5765381 ,
        -0.21156562,  1.44552836],
       [ 0.69943451, -1.07529066,  1.16287262, ...,  1.24899379,
        -0.34555717, -0.69178857]])
```

```
#X_test=sc.transform(X_test)
```

```
X_test
```

```
array([[ 0.69943451,  0.92998111, -0.85993942, ..., -0.42441044,
        -0.9079691 , -0.69178857],
       [-1.42972642,  0.92998111, -0.85993942, ...,  0.10803636,
         0.84978855, -0.69178857],
       [-1.42972642, -1.07529066,  1.16287262, ..., -1.29914447,
        -0.66643169, -0.69178857],
       ...,
       [ 0.69943451,  0.92998111, -0.85993942, ..., -1.71749553,
        -0.47778568, -0.69178857],
       [-1.42972642,  0.92998111, -0.85993942, ..., -0.95685724,
        -0.86212988, -0.69178857],
       [ 0.69943451, -1.07529066,  1.16287262, ..., -0.50047427,
        -0.48483787, -0.69178857]])
```

```
lr_model=LogisticRegression(solver='newton-cholesky')
lr_model.fit(X_train,y_train['status_binary'])
```

```
LogisticRegression
LogisticRegression(solver='newton-cholesky')
```

```
lr_testscore=lr_model.score(X_test,y_test['status_binary'])
lr_testscore
```

```
0.8426966292134831
```

```
y_test
```

	status_binary
86	1
75	0
127	1
47	1
97	0
...	...
249	0
251	0
99	0
280	0
295	0

89 rows × 1 columns

Next steps: Generate code with y_test View recommended plots

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

```
array([1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0])
```

```
#Create Classifier Object
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=2)
```

```
#Train the classifier using the training data
knn.fit(X_train,y_train['status_binary'])
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=2)
```

```
#Estimate the accuracy of the classifier on future data, using the test data
knn.score(X_test,y_test['status_binary'])
```

```
0.7415730337078652
```

```
y_test
```

	status_binary
86	1
75	0
127	1
47	1
97	0
...	...
249	0
251	0
99	0
280	0
295	0

89 rows × 1 columns

Next steps: Generate code with y_test View recommended plots

```
y_pred_knn=knn.predict(X_test)
y_pred_knn
```

```
array([1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       0])
```

```
rfc=RandomForestClassifier(n_estimators=300,random_state=100)
rfc.fit(X_train,y_train['status_binary'])
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=300, random_state=100)
```

```
rfc.score(X_test,y_test['status_binary'])
```

```
0.9325842696629213
```

```
#score_rfc=rfc.score(X_train,y_train['status_binary'])
#score_rfc
```

```
rfc_predict=lr_model.predict(X_test)
rfc_predicted_dict={'predicted_values':rfc_predict,'original_values':y_test['status_binary']}
#Creating a new dataframe
op=pd.DataFrame(rfc_predicted_dict)
op[op['predicted_values']!=op['original_values']]
```

	predicted_values	original_values
92	0	1
219	1	0
165	1	0
131	0	1
135	0	1
124	0	1
202	0	1
153	0	1
94	0	1
154	0	1
193	0	1
121	0	1
150	0	1
178	0	1

```
nbc=GaussianNB()
nbc.fit(X_train,y_train['status_binary'])
```

```
GaussianNB
GaussianNB()
```

```
nbc.score(X_test,y_test['status_binary'])
```

```
0.449438202247191
```

```
X_test[0]
```

```
array([ 0.69943451,  0.92998111, -0.85993942, -0.24806947, -0.63138629,
       -0.96673649,  1.05463022,  0.79367579, -0.29859554, -0.121278  ,
       -0.14028192, -0.42441044, -0.9079691 , -0.69178857])
```

```
#Dumping the model object
pickle.dump(rfc,open('model.pkl','wb'))
pickle.dump(lr_model,open('lr_model.pkl','wb'))
print(confusion_matrix(y_test['status_binary'],rfc_predict))
```

```
[[44  2]
 [12 31]]
```

```
#requests.py
#import requests
#ulr='http://127.0.0.1:5000/upload'
#files={'file':open('your_csv_file.csv','rb')}
#response=requests.post(url,files=files)

#print(response.json())
```

```
#requirements.txt
'''Flask==2.0.2
Python==3.9
pandas==1.3.3
numpy==1.21.2
seaborn==0.11.2'''
```

```
'Flask==2.0.2\nPython==3.9\npandas==1.3.3\nnumpy==1.21.2\nseaborn==0.11.2'
```

```
#data_analysis.py
#import numpy as np
#import pandas as pd
#import matplotlib.pyplot as plt
#import seaborn as sns

#placement_df=pd.read_csv("Campus_Selection.csv")
# placement_df
#placement_df_required=placement_df.drop("sl_no",axis=1)
# placement_df_required

#main.py
#from flask import Flask, request, jsonify,render_template
#import pandas as pd
#import pickle

#def pre_process_data(predictor_data):
    #predictor_data['workex_binary']=enc1.transform(predictor_data['workex'])
    #predictor_data_new=predictor_data.drop('workex',axis=1)
    #predictor_data_encoded=ct.transform(predictor_data_new)
    #X_data_final = sc.transform(predictor_data_encoded)
    #return X_data_final

#app = Flask(__name__)
#placement_model1 = pickle.load(open('model.pkl', 'rb'))
#placement_model = pickle.load(open('lr_model.pkl', 'rb'))
#Defines a route for the home page
#@app.route('/')
#def home():
    #return render_template('input.html')

# Route for uploading CSV file
#@app.route('/upload', methods=['POST'])
#def upload_csv():
    #csv_file = request.files['file']
    #if csv_file:
        #data = pd.read_csv(csv_file)
        #X_data = pre_process_data(data)
        #response_final_lr=placement_model.predict(X_data)
        #response_final_rfc=placement_model1.predict(X_data)
        #p_status_lr='Placed' if response_final_lr[0]==1 else 'Not Placed'
        #p_status_rfc='Placed' if response_final_rfc[0]==1 else 'Not Placed'
        #return render_template('input.html',prediction_text_lr='Placement Status = {}'.format(p_status_lr),prediction_text_rfc='Placement Status = {}'.format(p_status
    # else:
    #     return jsonify({'error': 'No file uploaded'})

#if __name__ == '__main__':
    #app.run(debug=True)
```

```
#main.py
#from flask import Flask, request, jsonify,render_template
#import pandas as pd
#from processing.pre_processing import pre_process_data
#import pickle

#app = Flask(__name__)
#placement_model1 = pickle.load(open('model.pkl', 'rb'))
#placement_model = pickle.load(open('lr_model.pkl', 'rb'))
#Defines a route for the home page
```

```
#@app.route('/')
#def home():
    #return render_template('input.html')

# Route for uploading CSV file
#@app.route('/upload', methods=['POST'])
#def upload_csv():
   #csv_file = request.files['file']
    #if csv_file:
        #data = pd.read_csv(csv_file)
        #X_data = pre_process_data(data)
        #response_final_lr=placement_model.predict(X_data)
        #response_final_rfc=placement_model1.predict(X_data)
        #p_status_lr='Placed' if response_final_lr[0]==1 else 'Not Placed'
        #p_status_rfc='Placed' if response_final_rfc[0]==1 else 'Not Placed'
        #return render_template('input.html',prediction_text_lr='Placement Status = {}'.format(p_status_lr),prediction_text_rfc='Placement Status = {}'.format(p_status_
    # else:
    #     return jsonify({'error': 'No file uploaded'})

#if __name__ == '__main__':
    #app.run(debug=True)
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
##
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```

```
#
```