Import Libraries

In [1]:

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

Import Dataset

In [2]:

```
dataset=pd.read_csv(r"C:\Users\admin\Desktop\Machine learning\All datasets\winequalityN.csv")
dataset
```

Out[2]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	
...	...	...	...	...	...	...	...	...	...	...	...	
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN	
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

6497 rows × 13 columns

In [3]:

```
dataset.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: 660.0+ KB
```

Data Preprocessing

1) Object Feature Transform Into Integer

In [4]:

```
dataset["type"].unique()
```

Out[4]:

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

In [5]:

```
dataset["quality"].unique()
```

Out[5]:

```
array([6, 5, 7, 8, 4, 3, 9], dtype=int64)
```

In [6]:

```
dataset["type"]= dataset["type"].map({'white':1,'red':2})
```

In [7]:

```
dataset.head()
```

Out[7]:

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

2) Fill Null Values

In [8]:

```
dataset.isnull().sum()
```

Out[8]:

```
type                     0
fixed acidity           10
volatile acidity         8
citric acid              3
residual sugar           2
chlorides                2
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       9
sulphates                4
alcohol                  0
quality                  0
dtype: int64
```

In [9]:

```
dataset["fixed acidity"]=dataset["fixed acidity"].ffill()
dataset["volatile acidity"]=dataset["volatile acidity"].ffill()
dataset["citric acid"]=dataset["citric acid"].ffill()
dataset["residual sugar"]=dataset["residual sugar"].ffill()
dataset["chlorides"]=dataset["chlorides"].ffill()
dataset["fixed acidity"]=dataset["fixed acidity"].ffill()
dataset["volatile acidity"]=dataset["volatile acidity"].ffill()
dataset["citric acid"]=dataset["citric acid"].ffill()
dataset["residual sugar"]=dataset["residual sugar"].ffill()
dataset["chlorides"]=dataset["chlorides"].ffill()
dataset["pH"]=dataset["pH"].ffill()
dataset["sulphates"]=dataset["sulphates"].ffill()
```

In [10]:

```
dataset.isnull().sum()
```

Out[10]:

```
type                    0
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
```

In [11]:

```
dataset.head()
```

Out[11]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	1	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8
1	1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9
2	1	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10
3	1	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9
4	1	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9

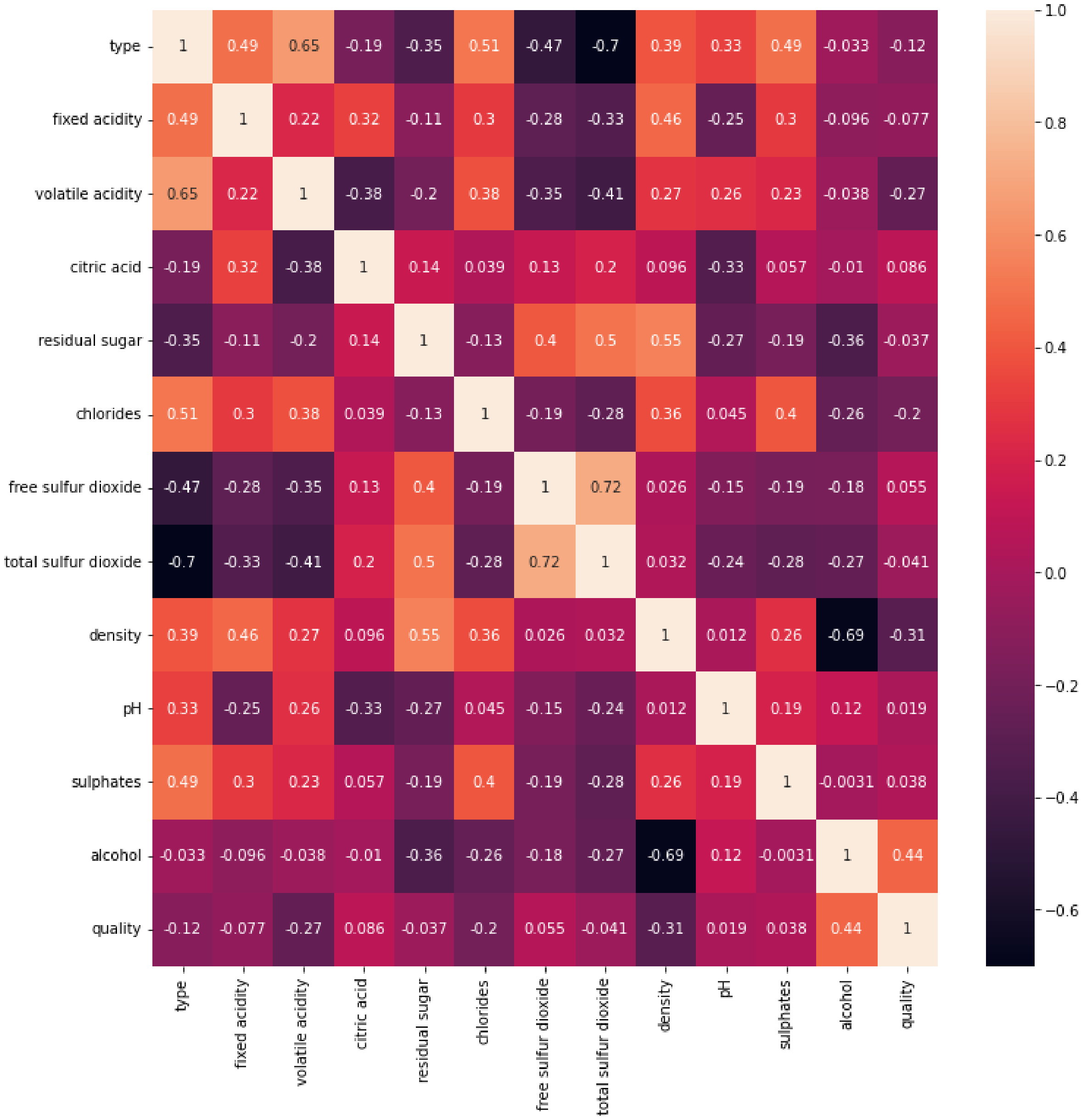
3) To checks Correlation between Features

In [12]:

```
plt.figure(figsize=[12,12])
sns.heatmap(dataset.corr(),annot=True)
```

Out[12]:

<AxesSubplot:>



4) Data Slicing

In [13]:

```
x=dataset.iloc[:,:-1]  # Independent # slicing independent and deppendent variable
y=dataset.iloc[:,-1]
```

Standardizing the Data

Standardizing the numerical columns in X dataset.

StandardScaler() adjusts the mean of the features as 0 and standard deviation of features as 1.

Formula that StandardScaler() uses is as follows

In [14]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)
x=pd.DataFrame(x)
```

As you can see, standardization is done successfully

In [15]:

```
x.head()
```

Out[15]:

	0	1	2	3	4	5	6	7	8	
0	-0.571367	-0.166455	-0.423490	0.284515	3.206914	-0.315004	0.815565	0.959976	2.102214	-1
1	-0.571367	-0.706585	-0.241243	0.146835	-0.807819	-0.200816	-0.931107	0.287618	-0.232332	(
2	-0.571367	0.682322	-0.362741	0.559874	0.306217	-0.172270	-0.029599	-0.331660	0.134525	(
3	-0.571367	-0.012132	-0.666486	0.009155	0.642529	0.056105	0.928254	1.243074	0.301278	-(
4	-0.571367	-0.012132	-0.666486	0.009155	0.642529	0.056105	0.928254	1.243074	0.301278	-(

Train-Test Split

Splitting the data into Train and Test chunks for better evaluation

In [16]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=20)
```

Defining several evaluation functions for convenience

In [17]:

```
def rmse_cv(model):
    rmse = np.sqrt(-cross_val_score(model, x, y, scoring="neg_mean_squared_error", cv=5)).mean
    return rmse



def evaluation(y, predictions):
    mae = mean_absolute_error(y, predictions)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mean_squared_error(y, predictions))
    r_squared = r2_score(y, predictions)
    return mae, mse, rmse, r_squared
```

Machine Learning Models

In [18]:

```
models= pd.DataFrame(columns=["Model","MAE","MSE","RMSE","R2 Score","RMSE (Cross-Validation)"]
```

Linear Regression

In [19]:

```
from sklearn.linear_model import LinearRegression
regressor=LinearRegression()
regressor.fit(x_train,y_train)
```

Out[19]:

```
▾ LinearRegression
LinearRegression()
```

In [20]:

```
predictions=regressor.predict(x_test)
```

In [21]:

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error,accuracy_score
from sklearn.model_selection import cross_val_score
mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(regressor)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "LinearRegression","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squ
models = models.append(new_row, ignore_index=True)
```

```
MAE: 0.576883810366818
MSE: 0.5586391892609275
RMSE: 0.7474216944007763
R2 Score: 0.30185592704949527
------------------------------
RMSE Cross-Validation: 0.7440270843189294

C:\Users\admin\AppData\Local\Temp\ipykernel_7032\4060312762.py:13: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
  models = models.append(new_row, ignore_index=True)
```

Lasso Regression

In [22]:

```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso.fit(x_train, y_train)
predictions = lasso.predict(x_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(lasso)
print("RMSE Cross-Validation:", rmse_cross_val)


new_row = {"Model": "Lasso","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE
models = models.append(new_row, ignore_index=True)
```

```
MAE: 0.7112138659877741
MSE: 0.8021436098012432
RMSE: 0.8956247036573094
R2 Score: -0.0024570735518016917
------------------------------
RMSE Cross-Validation: 0.8757635277961425

C:\Users\admin\AppData\Local\Temp\ipykernel_7032\3016036326.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
  models = models.append(new_row, ignore_index=True)
```

Ridge Regression

In [23]:

```
from sklearn.linear_model import Ridge
ridge = Ridge()
ridge.fit(x_train, y_train)
predictions = ridge.predict(x_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(ridge)
print("RMSE Cross-Validation:", rmse_cross_val)


new_row = {"Model": "Ridge","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared, "RMSE
models = models.append(new_row, ignore_index=True)
```

```
MAE: 0.5768917366312186
MSE: 0.5586539106115783
RMSE: 0.747431542424842
R2 Score: 0.3018375294434906
------------------------------
RMSE Cross-Validation: 0.7439916752609381
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_7032\2241856080.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
  models = models.append(new_row, ignore_index=True)
```

XGBoost Regressor

In [24]:

```
from xgboost import XGBRegressor
xgb = XGBRegressor(n_estimators=1000, learning_rate=0.01)
xgb.fit(x_train, y_train)
predictions = xgb.predict(x_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(xgb)
print("RMSE Cross-Validation:", rmse_cross_val)

new_row = {"Model": "XGBRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score": r_squared
models = models.append(new_row, ignore_index=True)
```

```
MAE: 0.4936669111251831
MSE: 0.42661811231372543
RMSE: 0.6531600970005175
R2 Score: 0.4668456630849692
------------------------------
RMSE Cross-Validation: 0.7295386396144149

C:\Users\admin\AppData\Local\Temp\ipykernel_7032\77520318.py:16: FutureWarning:
The frame.append method is deprecated and will be removed from pandas in a futur
e version. Use pandas.concat instead.
  models = models.append(new_row, ignore_index=True)
```

Random Forest Regressor

In [25]:

```
from sklearn.ensemble import RandomForestRegressor
random_forest = RandomForestRegressor(n_estimators=100)
random_forest.fit(x_train, y_train)
predictions = random_forest.predict(x_test)

mae, mse, rmse, r_squared = evaluation(y_test, predictions)
print("MAE:", mae)
print("MSE:", mse)
print("RMSE:", rmse)
print("R2 Score:", r_squared)
print("-"*30)
rmse_cross_val = rmse_cv(random_forest)
print("RMSE Cross-Validation:", rmse_cross_val)


new_row = {"Model": "RandomForestRegressor","MAE": mae, "MSE": mse, "RMSE": rmse, "R2 Score":
models = models.append(new_row, ignore_index=True)
```

```
MAE: 0.4323461538461538
MSE: 0.38803899999999997
RMSE: 0.6229277646725982
R2 Score: 0.5150588552835909
------------------------------
RMSE Cross-Validation: 0.7423697337553591

C:\Users\admin\AppData\Local\Temp\ipykernel_7032\1061377017.py:17: FutureWarnin
g: The frame.append method is deprecated and will be removed from pandas in a fu
ture version. Use pandas.concat instead.
  models = models.append(new_row, ignore_index=True)
```

Model Comparison

The less the Root Mean Squared Error (RMSE), The better the model is.

In [26]:

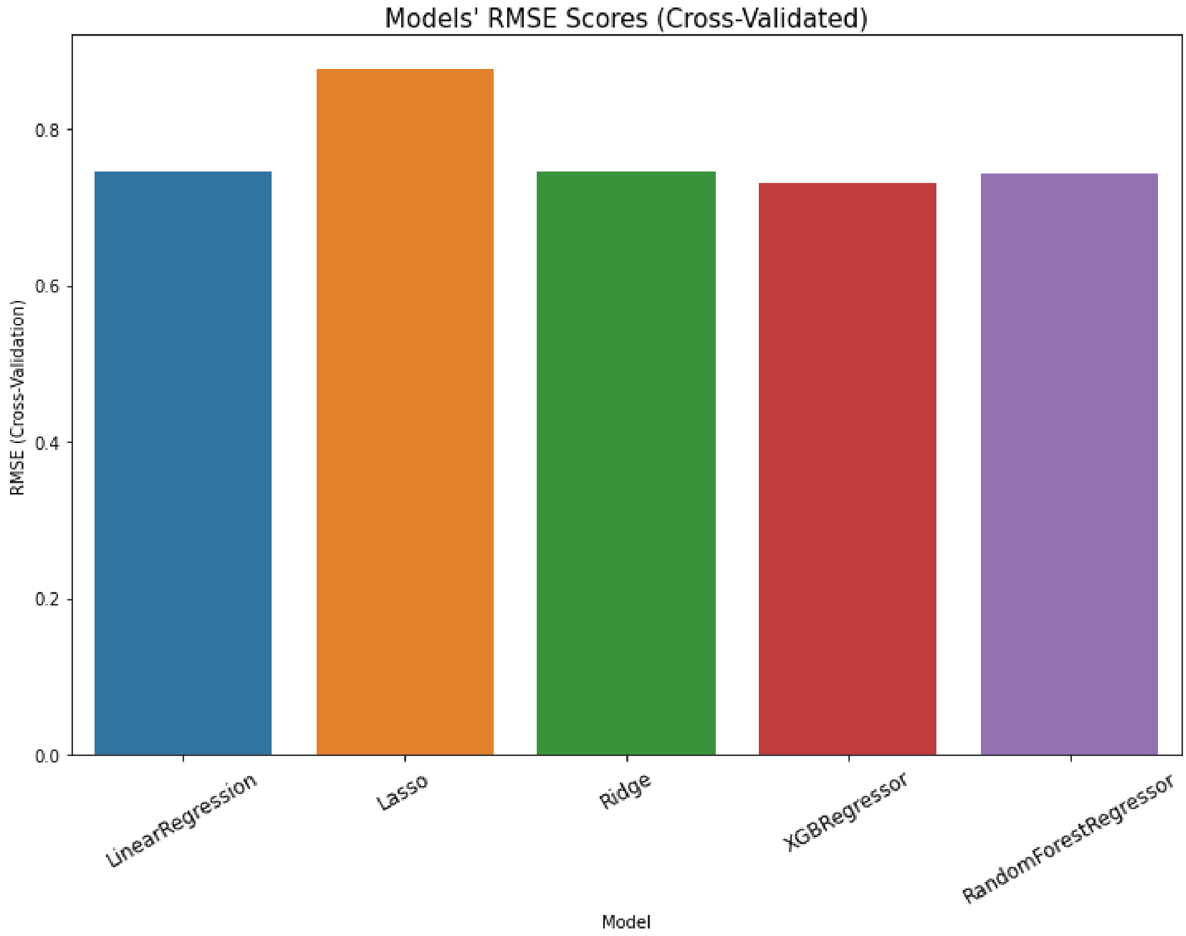
```
models.sort_values(by="RMSE (Cross-Validation)")
```

Out[26]:

	Model	MAE	MSE	RMSE	R2 Score	RMSE (Cross-Validation)
3	XGBRegressor	0.493667	0.426618	0.65316	0.466846	0.729539
4	RandomForestRegressor	0.432346	0.388039	0.622928	0.515059	0.74237
2	Ridge	0.576892	0.558654	0.747432	0.301838	0.743992
0	LinearRegression	0.576884	0.558639	0.747422	0.301856	0.744027
1	Lasso	0.711214	0.802144	0.895625	-0.002457	0.875764

In [27]:

```
plt.figure(figsize=(12,8))
sns.barplot(x=models["Model"], y=models["RMSE (Cross-Validation)"])
plt.title("Models' RMSE Scores (Cross-Validated)", size=15)
plt.xticks(rotation=30, size=12)
plt.show()
```



___Thank You_____

In [ ]: