SPANISH WINE PREDICTION

PROJECT 4

By:
Priyanka
Chandramohan

1. Introduction

The Spanish Wine are popular all across the globe , so to make the most out of it both wine buyers and sellers need to make appropriate decisions which will require them to do some analysis and visualization so they can understand which types of trends have followed in the past years concerning prices and other factors if someone is new in such type of thing they will get benefited from this analysis.

Business Problems we are trying to Solve:

Problems we generally face during buying a wine (especially Spanish) like:

- We are not aware of general factors that influence the house prices.
- Transportation cost increases the price.
- Here, we are making predictions of the selling price of the wine on the basis of the previous sold wines.
- We are also making conclusion what are the factors affecting the prices of the wine.

Main Goal:

➔ Create an analytical framework to understand
Key factors impacting Wine prices

➔ Develop a modeling framework
To estimate the price of a wine that is up for sale

About Dataset

This dataset is related to red variants of Spanish wines. The dataset describes several popularity and description metrics their effect on its quality. The dataset contains 7500 different types of red wines from Spain with 11 features that describe their price, rating, and even some flavour description. The datasets can be used for classification or regression tasks.

This is what the dataset looks like,

	winery	wine	year	rating	num_reviews	country	region	price	type	body	acidity
0	Teso La Monja	Tinto	2013	4.9	58	Espana	Toro	995.00	Toro Red	5.0	3.0
1	Artadi	Vina El Pison	2018	4.9	31	Espana	Vino de Espana	313.50	Tempranillo	4.0	2.0
2	Vega Sicilia	Unico	2009	4.8	1793	Espana	Ribera del Duero	324.95	Ribera Del Duero Red	5.0	3.0
3	Vega Sicilia	Unico	1999	4.8	1705	Espana	Ribera del Duero	692.96	Ribera Del Duero Red	5.0	3.0
4	Vega Sicilia	Unico	1996	4.8	1309	Espana	Ribera del Duero	778.06	Ribera Del Duero Red	5.0	3.0
5	Vega Sicilia	Unico	1998	4.8	1209	Espana	Ribera del Duero	490.00	Ribera Del Duero Red	5.0	3.0
6	Vega Sicilia	Unico	2010	4.8	1201	Espana	Ribera del Duero	349.00	Ribera Del Duero Red	5.0	3.0
7	Vega Sicilia	Unico	1995	4.8	926	Espana	Ribera del Duero	810.89	Ribera Del Duero Red	5.0	3.0
8	Vega Sicilia	Unico Reserva Especial Edicion	2015	4.8	643	Espana	Ribera del Duero	345.00	Ribera Del Duero Red	5.0	3.0
9	Vega Sicilia	Unico	2011	4.8	630	Espana	Ribera del Duero	315.00	Ribera Del Duero Red	5.0	3.0

Table1 - Dataset

It has lots of rows and columns , which is 7500 rows spread across 11 columns. Here is the list of columns this dataset has,

```
1 data_wine.columns
```

```
Index(['winery', 'wine', 'year', 'rating', 'num_reviews', 'country', 'region',
       'price', 'type', 'body', 'acidity'],
      dtype='object')
```

Table2 - Columns in

2. EDA & Business Implication

EDA stands for exploratory data analysis where we explore our data and grab insights from it. EDA helps us in getting knowledge in form of various plots and diagrams where we can easily understand the data and its features.

Analysis of the Data

```
1 data_wine.describe().T
```

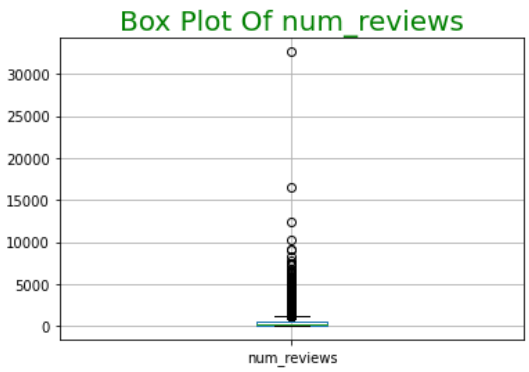
	count	mean	std	min	25%	50%	75%	max
year	2033.0	2011.272996	11.044993	1910.00	2010.0	2015.0	2017.0	2021.00
rating	2033.0	4.402607	0.146542	4.20	4.3	4.4	4.5	4.90
num_reviews	2033.0	575.239547	1381.146095	25.00	57.0	140.0	490.0	32624.00
price	2033.0	136.004267	273.029742	4.99	32.0	53.9	110.0	3119.08
body	2033.0	4.226267	0.616345	2.00	4.0	4.0	5.0	5.00
acidity	2033.0	2.931136	0.314009	1.00	3.0	3.0	3.0	3.00

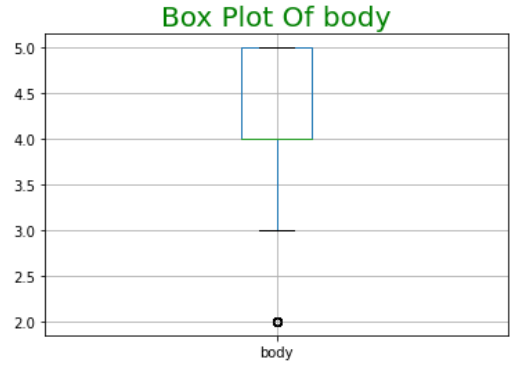
Table3 - Checking of the Skewness

Observation:

- **'year'** value ranges from 1910 to 2021.As mean < median, we can say that it is **slightly left skewed.**
- **'rating'** ranges from 4.2 to 4.9.As mean and median are almost equal, we can say that it is **almost Normal Distributed.**
- **'num_reviews'** ranges from 25 to 32624.As mean is almost 4 times as of median , we can say that it is **Highly rightly skewed.**
- Also in this column we have very big difference between the 3rd quartile and maximum value , there is very high chances of having outliers.
- **'price'** ranges from 4.99 to 3119.mean is more than twice as that of median , it is **Highly rightly skewed.**
- Also in this column we have very big difference between the 3rd quartile and maximum value , there is very high chances of having outliers.
- **'body'** value ranges from 2 to 5 . Mean is slightly greater than median , it is **slightly right skewed.**
- Also in this column we can observe big difference between the 1st quartile and minimum value , their is very high chances of having outliers.
- '**acidity**' ranges from 1 to 3 . Mean ~ Median , we can say that it is **almost Normal Distributed.**

Uni-variate Analysis - By BoxPlot





Plot1 - Boxplot

Observation:

We can clearly see that there is outliers present in all these 3 columns

Uni-Variate Analysis of Each Column

```
1 data_wine.winery.value_counts()
```

```
Vega Sicilia              58
Artadi                    43
Marques de Murrieta       31
Alvaro Palacios           30
Martinet                  30
                          ..
Vins Miquel Gelabert       1
El Grillo y La Luna        1
Jean Leon                  1
Bodegas Ateca              1
Binigrau                   1
Name: winery, Length: 479, dtype: int64
```

```
1 data_wine.wine.value_counts()
```

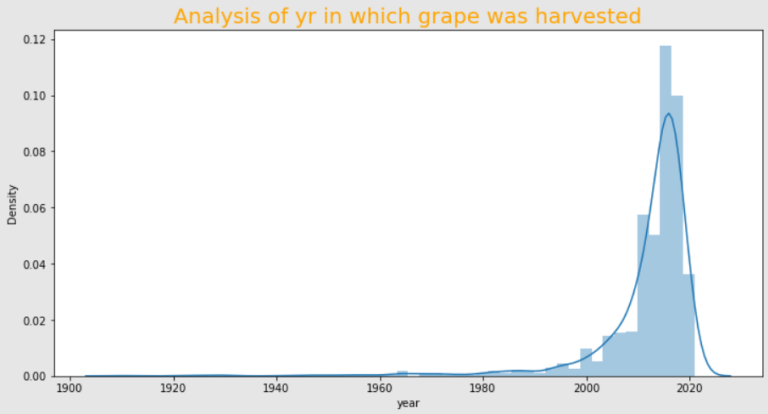
```
Tinto                                45
Reserva                              28
Priorat                              24
Valbuena 5o                          24
Gran Reserva                         22
                                     ..
Finca Helena                          1
Solera PAP Palo Cortado Rare Sherry   1
Capricho Crianza                      1
Savinat Sauvignon Blanc               1
Les Brugueres                         1
Name: wine, Length: 841, dtype: int64
```

Analysis on winery

- **Observation:** wines are coming from 479 winery

Analysis on wine

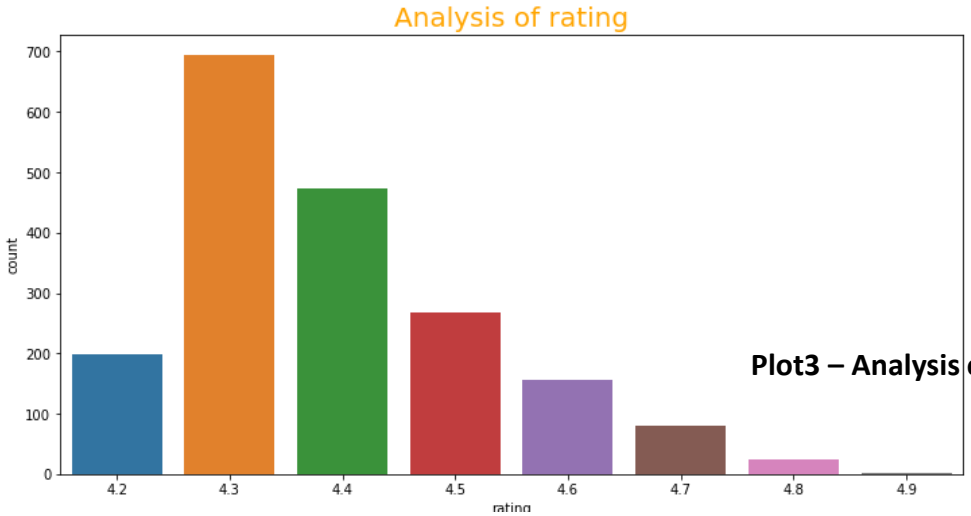
- **Observation:** There are 836 wine names.



Analysis on year

Observation: Most of the wines are made of grapes which are harvested after year 2000

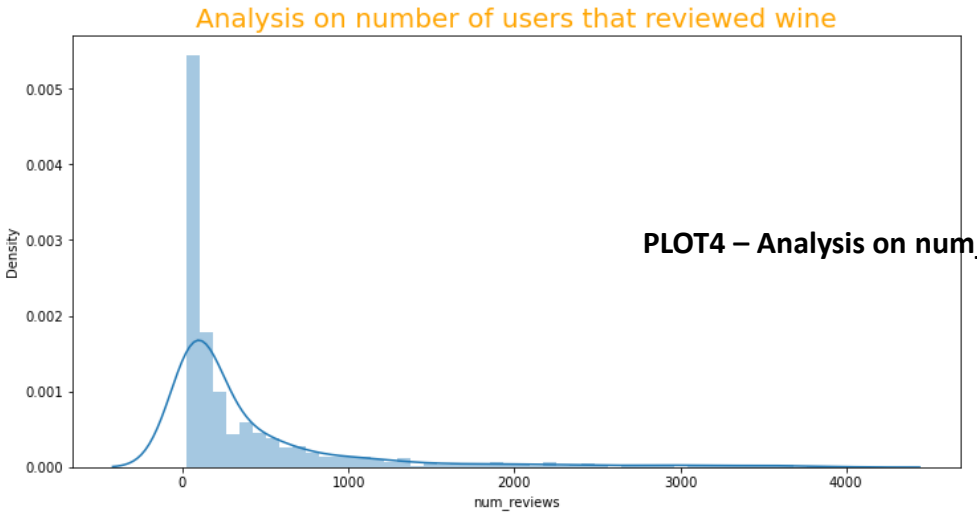
Plot2 – Analysis on Year



Plot3 – Analysis on Rating

ANALYSIS ON RATING

- **Observation**: we can clearly see most of wine are rated between 4.3 and 4.5.



PLOT4 – Analysis on num_reviews

ANALYSIS ON num_reviews

- **Observation:** Most of the wines are reviwed by 100 users or less than 100 as we can clearly see a very big spike there.

ANALYSIS ON Country

- As the dataset is of wines from spain ,so there is no need to review country column for all the records it will be Spain.
- We will drop the country column later in the preprocessing phase.

```
1  data_wine.region.value_counts().head(10)
```

```
Ribera del Duero      464
Rioja                 454
Priorato              228
Toro                   68
Castilla y Leon        51
Vino de Espana         44
Cava                   34
Rias Baixas            33
Jerez-Xeres-Sherry     28
Montilla-Moriles       26
Name: region, dtype: int64
```

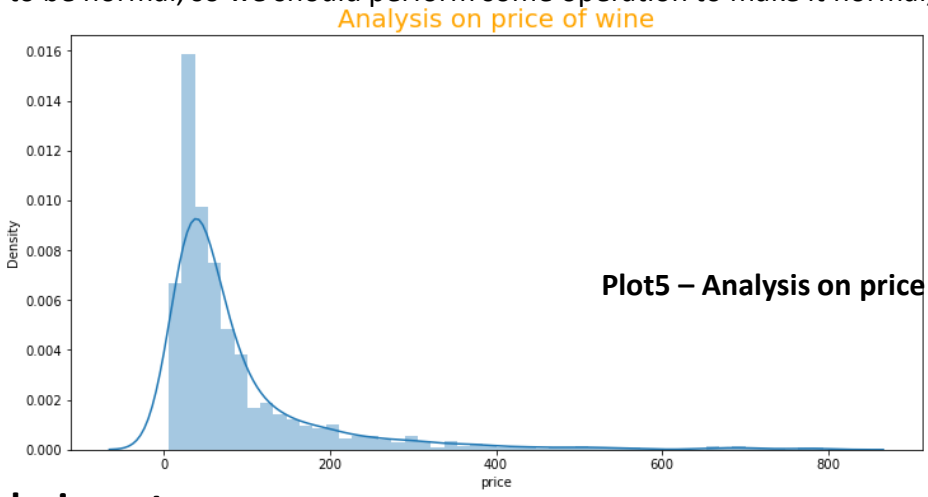
ANALYSIS ON Region

- **Observation:** Majority of wines are coming from the 3 regions : Ribera del Duero,Rioja,Rioja.

Analysis on price (Target variable)

Observation:

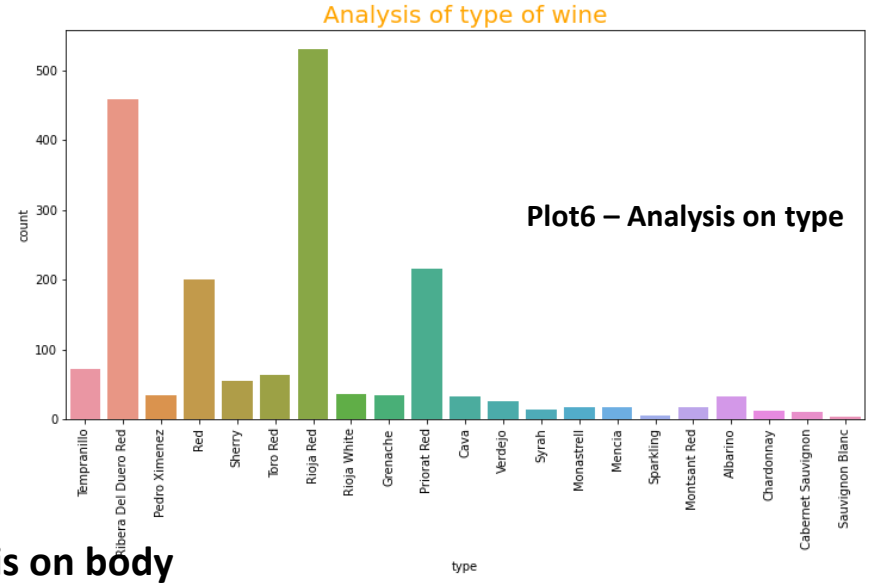
- Price of most of the wines less than 200.
- The above graph shows that price has right skewness. And we know that the assumption of linear regression tells us that the distribution of dependent variable has to be normal, so we should perform some operation to make it normal, in later part.



Plot5 – Analysis on price

Analysis on type

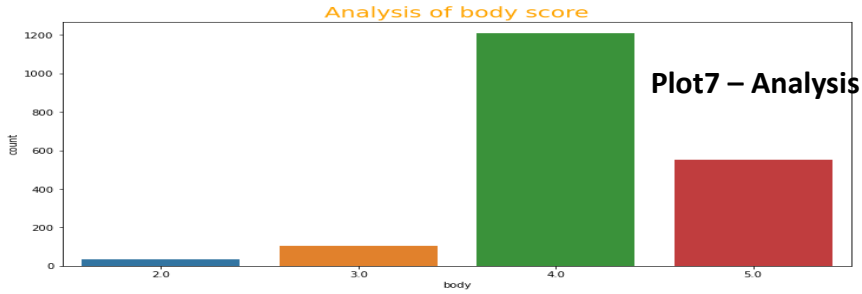
- **Observation:** 'Ribera Del Duero Red','Red','Rioja Red','Priorat Red' are some of the most availiable wine varities.
- In which 'Rioja Red' has most sales followed by 'Ribera Del Duero Red'.



Plot6 – Analysis on type

Analysis on body

- **Observation:** Majority of the wine are rated 4 or 5 , which says wine from spain has good body score.



Plot7 – Analysis on Body Score

Analysis on acidity

- **Observation**: More than 90 % of the wine has high acidity rating.

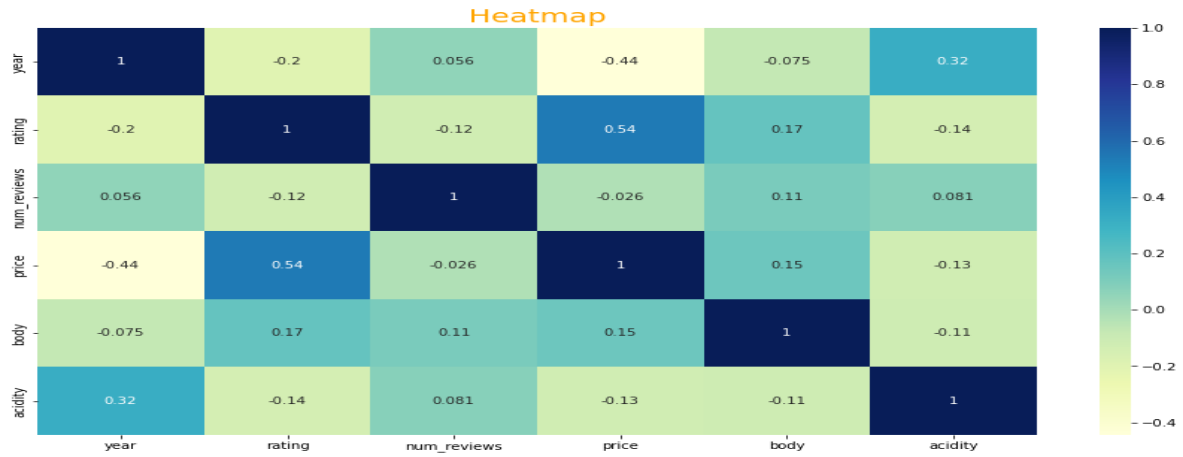
```
1  data_wine.acidity.value_counts()
```

```
3.0    1792
2.0      70
1.0      35
Name: acidity, dtype: int64
```

Bivariate Analysis

Heatmap

Here we are looking which feature can be dropped according to correlation between them. On left the name of the column is written and on right the columns which are highly correlated with it is written.



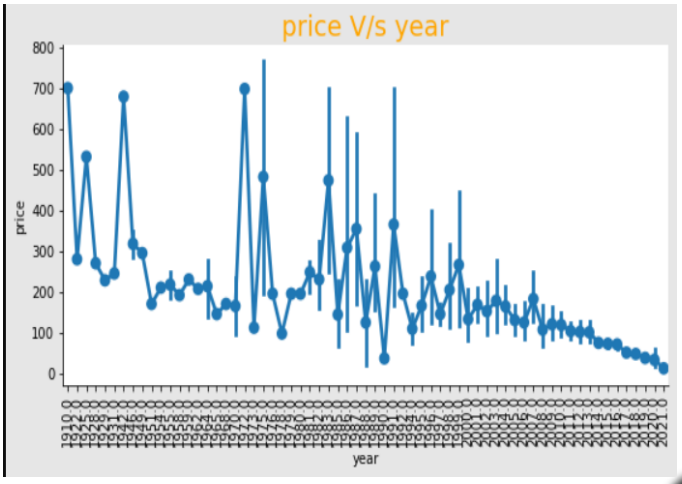
Plot8 – Heatmap

We have linear relationships in below features as we got to know from above heatmap

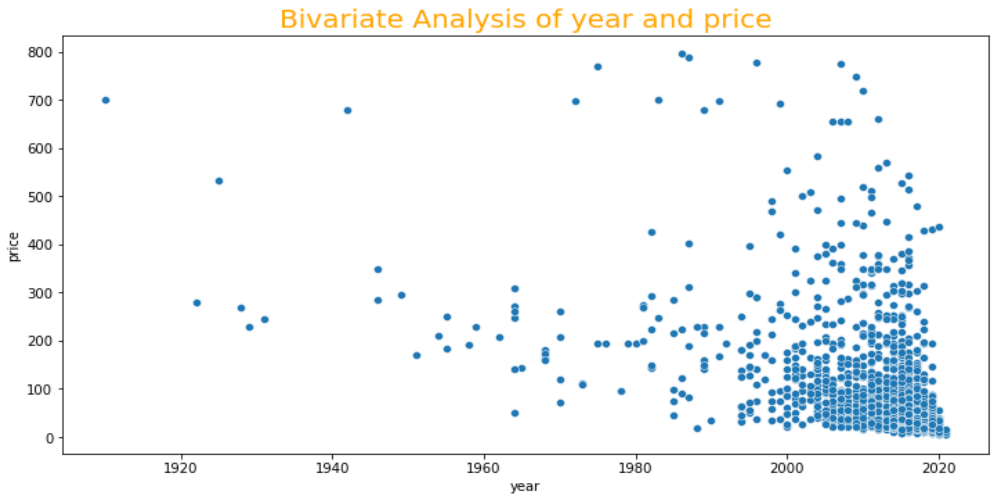
- There is high correlation between price and rating as already discussed.
- We also have moderate correlation between year and acidity.
- There is also moderate relation between rating and year.
- There is high - ve correlation between price and year.

Bivariate Analysis of Each Column with Price

Bivariate Analysis of Price and year

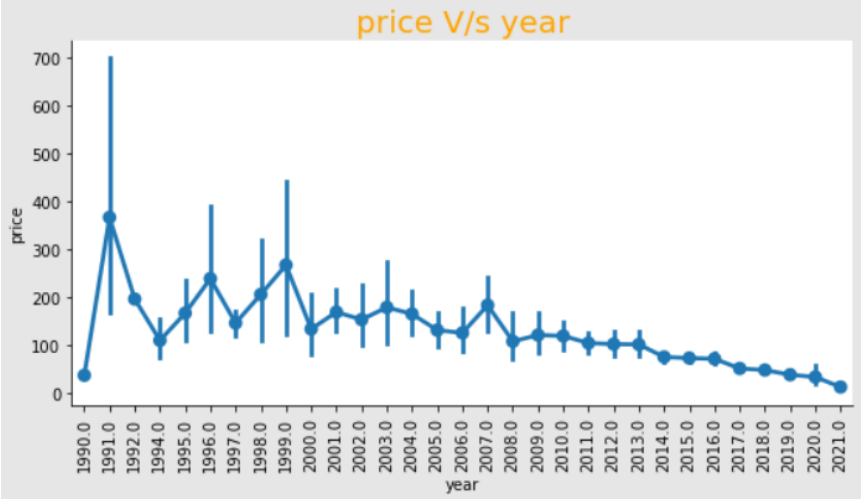


Plot9 – Analysis of Price and Year



Plot10 – Analysis of year and Price

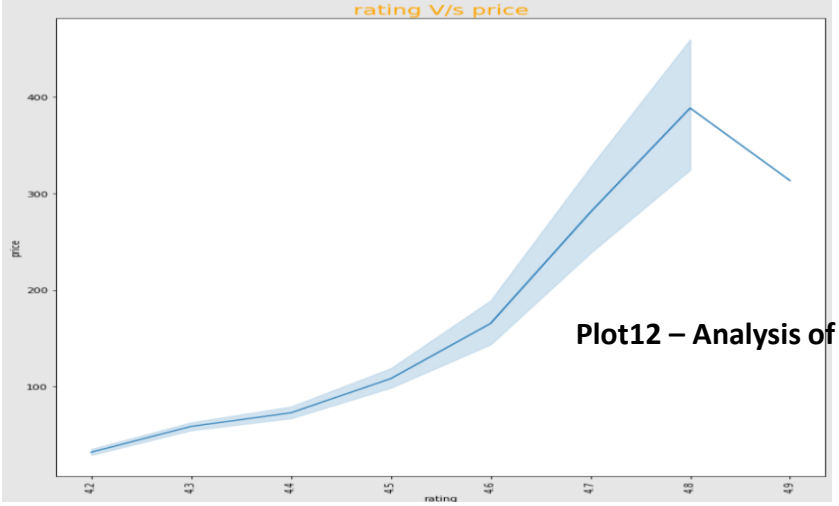
- **Observation:** By looking at it we can say that prices of wines is low for wines which have grapes harvested about 20-30 years.
- But as the data range values after 1990 would be interesting to see.



Plot11 – Analysis of Price and Year

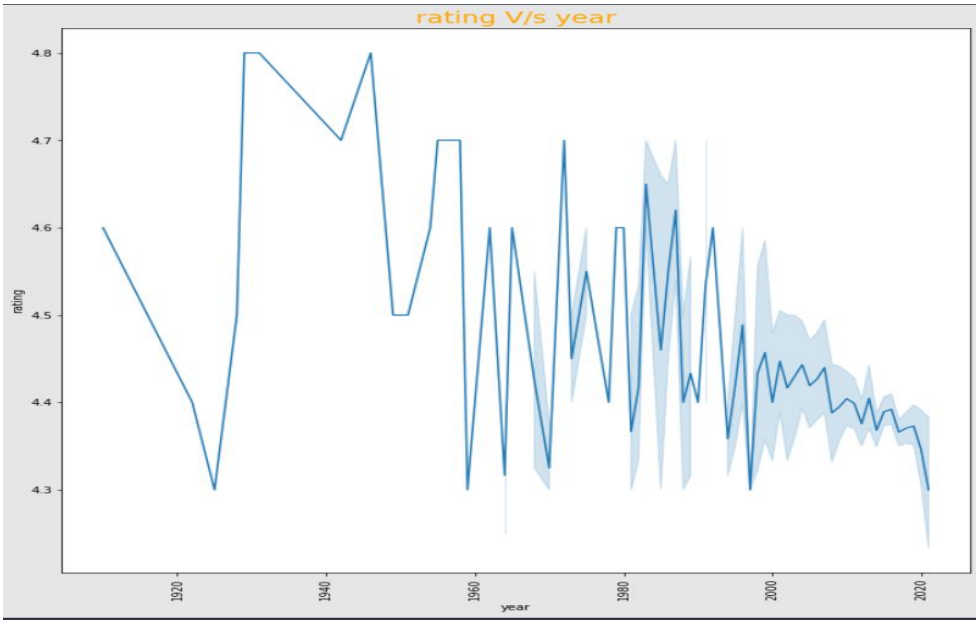
- There is clearly a downward trend , as time period is increasing price is decreasing.
- Old is Gold.

Bivariate Analysis of rating and price



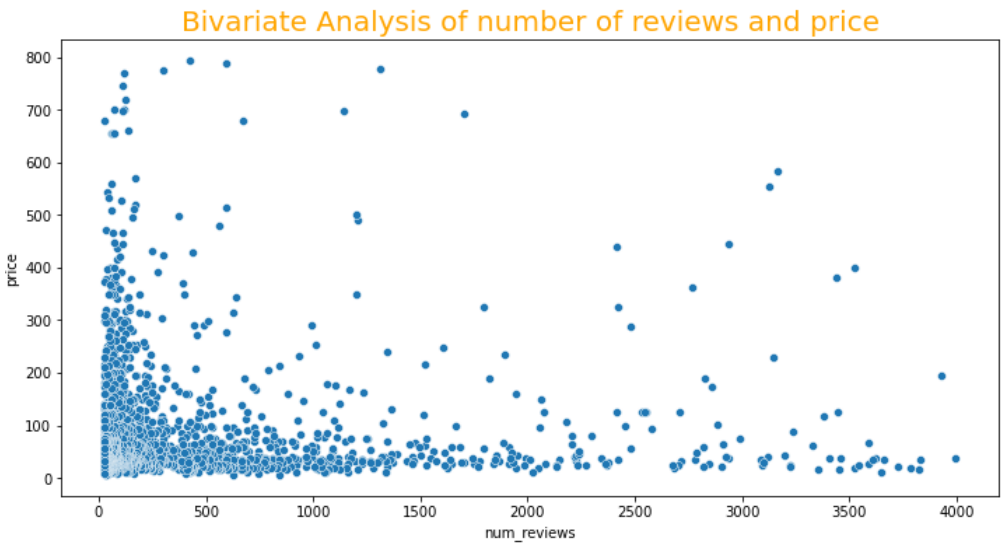
Plot12 – Analysis of rating and price

- **Observation:** we can say that their is gradual increase in the price of wine with increase in ratings.(+ ve linear relation)
- we should also analyse the relation between number of years and ratings
- Older Wine is more valuable than recent graped ones.



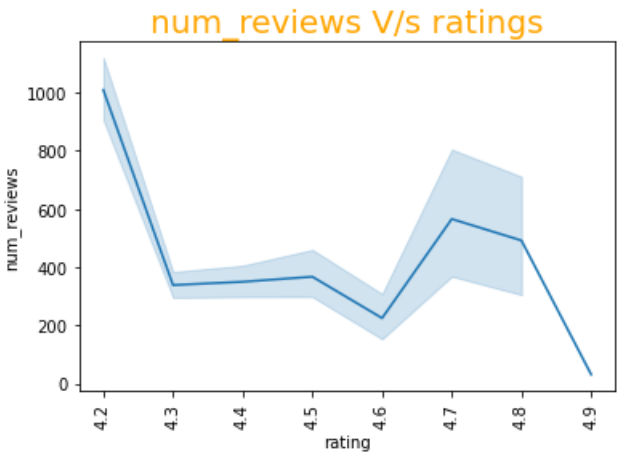
Plot13 – Analysis of year and rating

Analysis of price and num_reviews



Plot14 – Analysis of price and num_reviews

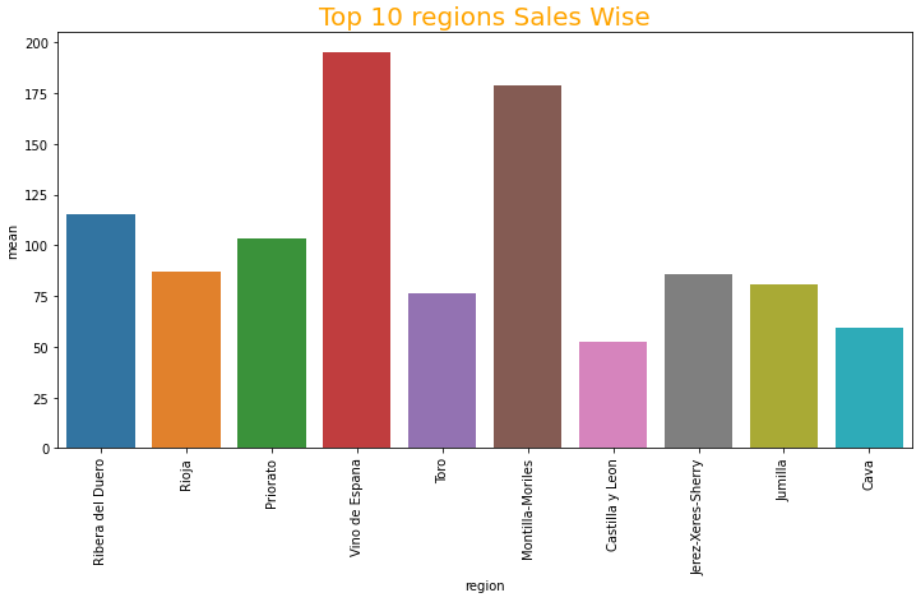
- **Observations:** prices of wine are high, when the numbers of reviews are less.
- As price is positively linearly related with rating ,we should check the relation between ratings and number of reviews.



- The more people review a wine, the less rating this wine get.

Plot15 – Analysis of rating and num_reviews

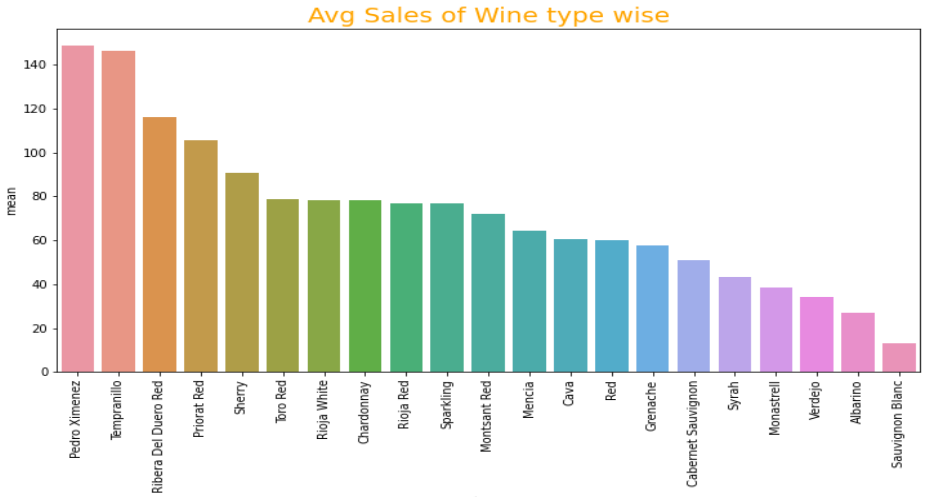
Bivariate Analysis of price and region



- These are the top 10 regions with highest sales of wine.

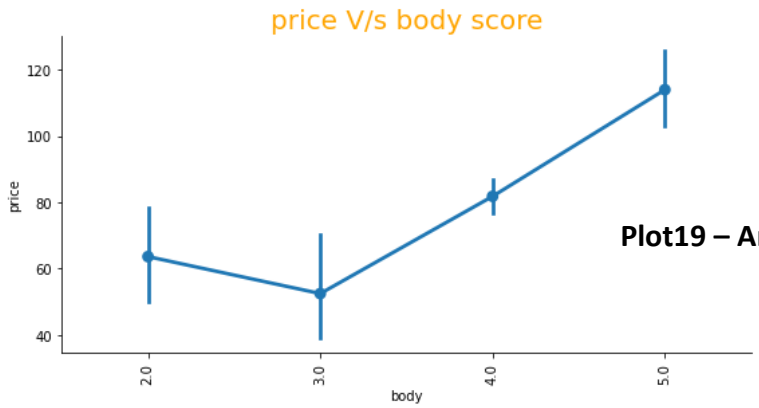
Plot17 – Analysis of price and region

Bivariate Analysis of price and type

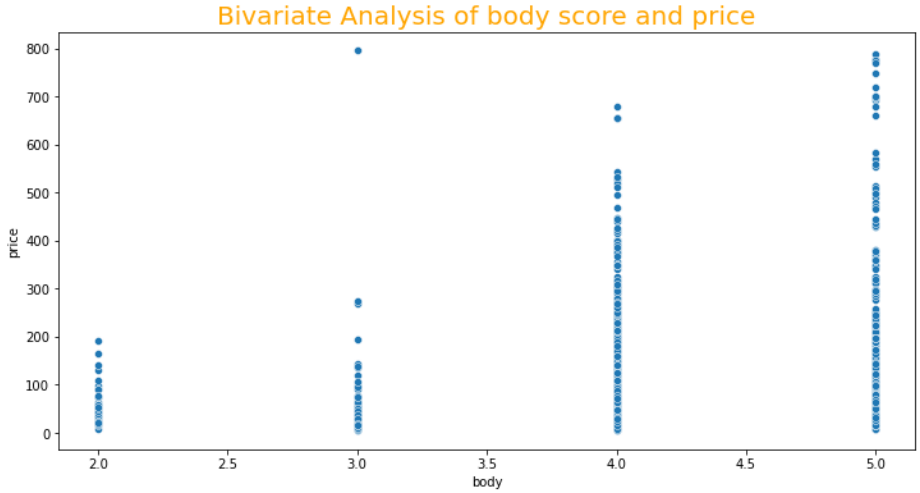


PLOT18 – Analysis of Type and price

Bivariate Analysis of price and body Score



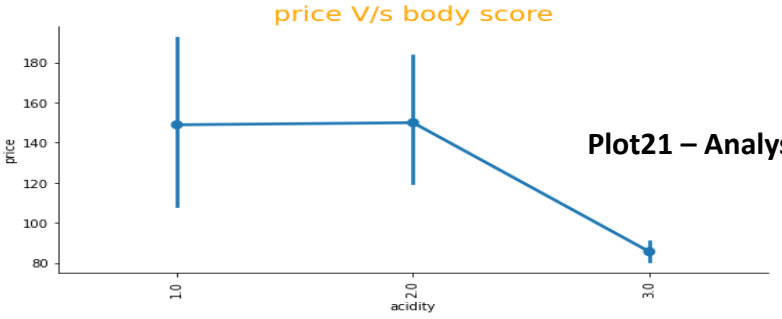
Plot19 – Analysis of Price and Body score



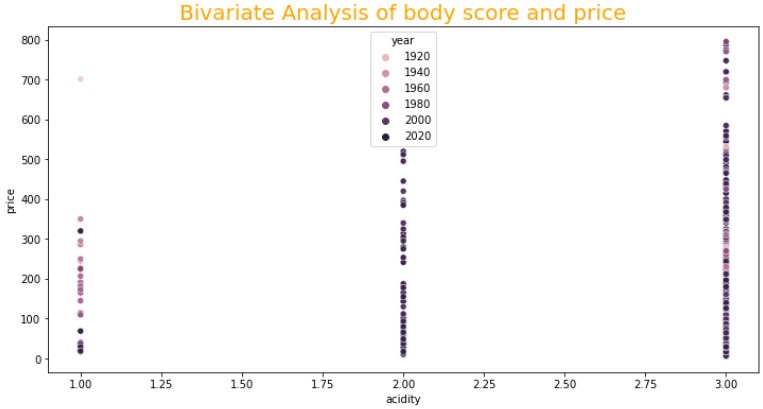
Plot20 – Analysis of body score and price

- **Observation**: Their is linear positive relation between body score and price.

Bivariate Analysis of price and acidity



Plot21 – Analysis of price and acidity score score



- **Observation:** Prices of wines having high acidity values is low because most of high acidity score wines are made of grapes that are harvested in recent past, as we have already concluded Older wine has higher price.

Plot22 – Analysis of price and Acidity Score

3. Data Cleaning & Pre-processing

Data Cleaning is an important phase in any data science project, if our data is clean then only we can provide it to our machine learning model. Un-cleaned Data can further lead our model with low accuracy. And, if data is incorrect, outcomes and algorithms are unreliable, even though they may look correct. There is no one absolute way to prescribe the exact steps in the data cleaning process because the processes will vary from dataset to dataset.

The approach used for identifying and treating missing values and outlier treatment:-

```
1 print(start+"Checking for null values count column-wise"+end,'\n',data_wine.isnull().sum(),'\n')
```

```
Checking for null values count column-wise
 winery            0
wine               0
year             290
rating             0
num_reviews        0
country            0
region             0
price              0
type             545
body            1169
acidity         1169
dtype: int64
```

Code Sample-1: Null Values

To identify any missing values in our data set we have used Pandas pre built function isnull() to detect any missing values in our datasets. As we can see that column body and acidity has a high number of missing values. So our next step is how to handle a large number of missing values. One approach is , that we will delete the column if we don't need that column for further analysis. And, what if we need that column for further analysis then we have use an approach will is a predefined function in Pandas called fillna().

Dealing with Missing values

Filling the misssing value in the type column with the mode

```
]: 1 data_wine["type"] = data_wine["type"].fillna(data_wine["type"].mode()[0])
```

Filling the missing value in numerical variable using Median

```
]: 1 for i in ['year','body','acidity']:
2     data_wine[i] = data_wine[i].fillna(data_wine[i].median())
```

Code Sample-2: Missing Values

As you can have a look, How we have filled the missing values in a categorical variable using mode. And, How we have filled the missing values in a numerical variable using Median. This is how we have an approach for identifying and handling missing values.

While performing Pre-processing and Data cleaning we have to also deal with outliers. Dealing with outliers is also a necessary step to be taken for further analysis and model building. Outliers are data points in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset.

Handling outliers with z-score

In [838]:

```python
# function for dropping outliers

def drop_outliers_zscore(dfcopy:pd.DataFrame, cols, threshold:int=3, inplace:bool=False):
    '''
    input:  dfcopy            ==> the dataframe that contains outliers
            cols  list / str  ==> list of strings (names of columns that have outliers)
            inplace           ==> if True,  method will edit the original dataframe
                                  if False, method will return the new dataframe
            threshold         ==> maximum and minimun threshold of zscore

    output: df:        ==>clean dataframe
    this method drops outliers from data using zscore method
    '''
    if inplace:
        global data_wine
    else:
        data_wine = dfcopy.copy()

    def drop_col(df_, col):

        mean, std = np.mean(df_[col]), np.std(df_[col])
        df_['is_outlier'] = df_[col].apply(lambda x : np.abs((x - mean) / std) > threshold)
        outliers_idx = df_.loc[df_['is_outlier']].index
        df_ = df_.drop(outliers_idx, axis=0)

        data_wine = df_.drop('is_outlier', axis=1)
        return data_wine


    if type(cols) == str:
        data_wine = drop_col(data_wine, cols)
    elif type(cols) == list:
        for col in cols:
            data_wine = drop_col(data_wine, col)
    else :
        raise ValueError('Pass neither list nor string in {Cols}')

    if inplace:

        dfcopy = data_wine
    else:
        return data_wine

THRESHOLD = 4
num_outlier_records =  data_wine.shape[0] - drop_outliers_zscore(data_wine, columns, threshold=THRESHOLD).shape[0]
print(f'Number of Outliers {num_outlier_records}, with threshold, with threshold {THRESHOLD}')

drop_outliers_zscore(data_wine, columns, threshold=THRESHOLD, inplace=True)
```

```
Number of Outliers 58, with threshold, with threshold 4
```

Code Sample-3: Outlier removal

After performing Uni-variate, Bi-variate, and Multi-variate analysis we can get an idea about outliers present in our dataset. We have seen outliers for columns num_reviews,price and body. We have created a function named drop_outliers_zscore this method drops outliers from data using zscore method. We got 58 outliers records present in our dataset.

<u>Need for Variable Transformation</u>:-

- Variable transformation is a way to make the data work better in your model. Here specifically data type of year column is object , we need to convert it in numerical type. The need for this is because we need our model to have a good score and accuracy which will make good predictions. So to feed the data to our model we must ensure to

Defining a function to convert datatype of year column

]:

```python
def func(x):
    try :
        return int(x)
    except ValueError:
        return x
l = data_wine['year'].apply(func)
l = [*map(lambda x : type(x) == str, l.tolist())]

print(data_wine.loc[l, 'year'])

data_wine.loc[l, 'year'] = np.NaN
data_wine['year'] = data_wine.year.values.astype(float)
```

Code Sample4 – Variable Transformation

Variables removed or added and why?

Variables are removed in such a scenario where we have a large number of null values in any columns particularly or else in our dataset, to make our data clean and ready for modeling. Whereas, variables are added in such a scenario where Consider an example where we have a column as start-date and another column as end-date so we can create a column named 'difference' where we subtract the start-date and end-date and have a number of days between them in a column named 'difference' and later drop start-date and end-date as if they are of no use.

We have dropped the Country column as it has only one unique value in entire column i.e Spain.

Feature Scaling

Feature scaling is important for every algorithm where distance matter. Two famous techniques for Feature Scaling are:

- Normalization
- Standardization

Standardization is useful when the feature distribution is Normal or Gaussian, otherwise we do Normalization.

```
# we will perform standardization

data_wine = (data_wine-data_wine.mean())/data_wine.std()
data_wine.head()
```

	winery	wine	year	rating	num_reviews	region	price	type	body	acidity
1	-1.430706	1.470907	0.590331	3.736375	-0.574791	1.426853	2.082914	1.907574	-0.323380	-2.854102
2	1.399291	1.303015	-0.269775	3.001725	2.047669	0.386740	2.189228	-0.014766	1.303814	0.227418
3	1.399291	1.303015	-1.225447	3.001725	1.916695	0.386740	5.606215	-0.014766	1.303814	0.227418
4	1.399291	1.303015	-1.512149	3.001725	1.327311	0.386740	6.396372	-0.014766	1.303814	0.227418
5	1.399291	1.303015	-1.321015	3.001725	1.178477	0.386740	3.721723	-0.014766	1.303814	0.227418

4. Modeling Building

Model Selection and Why?

After cleaning and processing the data then comes the modelling part which includes building Machine Learning models, let's first understand in brief what Machine Learning is?
Machine Learning is a technique that analyses past data and tries to extract meaningful insights and patterns from them which can be further used to perform predictions in future. For example, classifying whether a tumor is benign or malignant, predicting stock prices, etc. One such application which we're using right here is predicting house prices. Before making predictions first we need to build a model and train it using past data.
First, we need to separate the dataset into two parts: features (property attributes) and labels (prices) which is the required format for any model to be trained on.

Then the data needs to be split into 3 sets
1. Training set - This will be the part of the dataset which the model will be using to train itself, the size should be at least 60-70% of the total data we've.
2. Validation set - This set is used for validating our model's performance for a different set of hyperparameters. After taking out the train set, the remaining set can be split into validation and test set.
3. Testing set - To evaluate how the model is performing on the unseen data on which the model will be doing future predictions on, test set is used. It helps to understand how much error is there between actual and predicted values.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(1327, 9)
(570, 9)
```

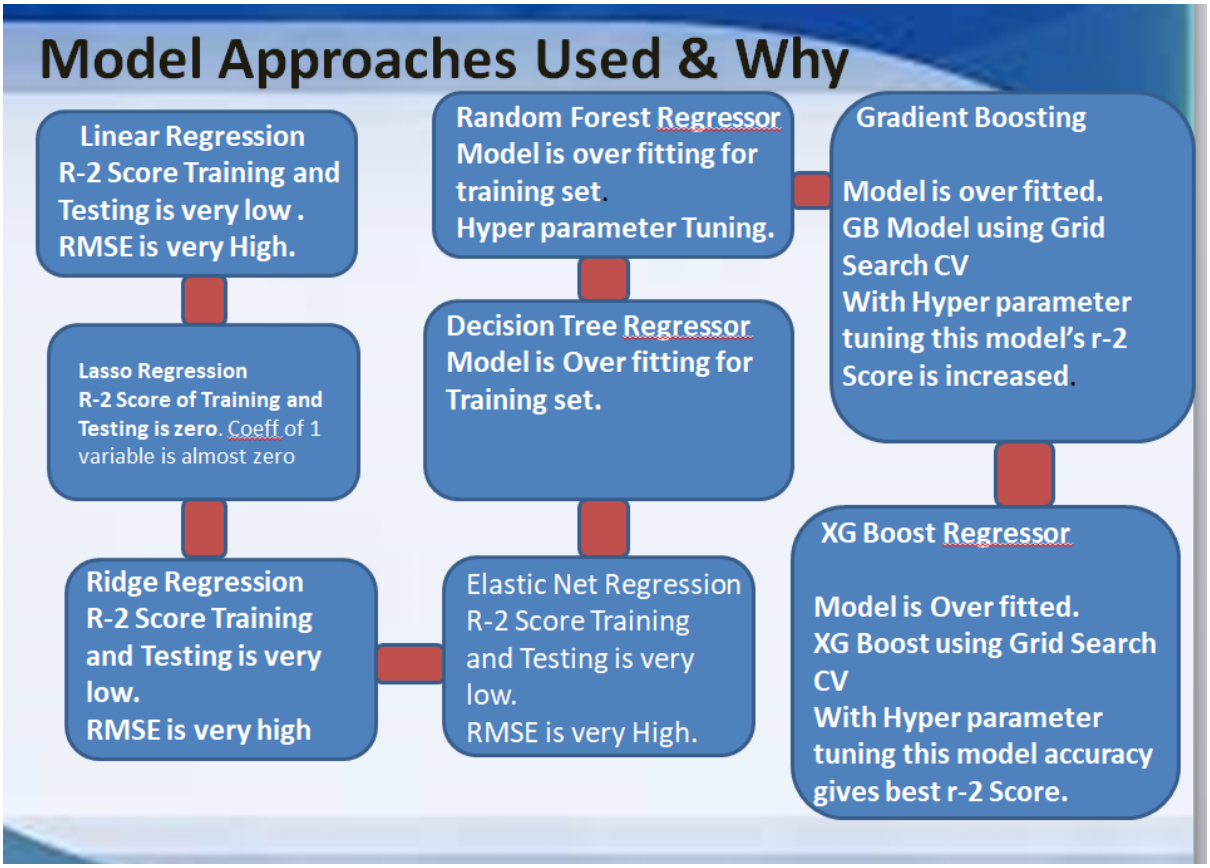
Code Sample 5 : Splitting Data

We need to build different regression algorithms and using the testing set we can determine which model to keep for making final predictions.

Here is the list of all the algorithms we've to build and evaluated:

1. Simple Linear Regression
2. Lasso Regression
3. Ridge Regression
4. Elasticnet Regression
5. Decision Tree Regression
6. Random Forest Regression
7. Gradient Boosting Regression
8. Xgboost Regression

Initially, we've tried Linear Regression and its variants Lasso (l1 norm) , Ridge (l2 norm) Regression and Elastic net Regression , but they are performing quite similarly on the 2 sets (train & test).
They are not giving good result on train , neither test set , with lasso giving 0 r-2 score for both train test set which is very terrible.
Let's look at some other algorithms, and how they are performing as compared to linear regression
So , **Decision Tree Regression** is giving quite good score of 0.82 on the train set, but not that good on the valid and test set, looks like it is overfitting on the training data.
If we look at the Random Forest Regressor is giving quite good score of 0.95 on the train set, but not that good on the valid and test set, looks like it is overfitting on the training data.



Flow Diagram 1- Flow of the Model Used and Why

Finally, now we can try those boosting algorithms and see where they are getting us? Generally boosting algorithms give a very good performance, and they are giving on the training set but there isn't any significant improvement on the test set as compared to Tree-based models.
Same Case with the Gradient boosting regressora and Xgboost regression , Over fitting is seen in both of these as well.

Efforts to improve model performance

Hyperparameter Tuning

Hyperparameter tuning is the process of trying out a different set of model parameters, actually, they are algorithm's parameter for example theta1 and theta2 in the hypothesis function for Linear Regression is model parameter, but lambda which is a factor that decides the amount of regularization is algorithm's parameter called as a hyperparameter. Tuning hyper parameters helps us to find out the optimal parameter values for which the model is giving less error and a better overall score. We've performed tuning for Gradient Boosting Regressor with both the methods RandomizedSearchCV as well as GridSearchCV. What random search does is from the given set of parameter values, it tries out n number of combinations whereas grid search builds a model with all the possible combinations from the given set of parameters and gives us the optimal values.

Grid Search CV

These are the parameters that we've given to Grid Search to find out the best one for Gradient Boosting.

Gradient Boosting Regressor with GridSearchCV

```
# Number of trees
n_estimators = [50,80,100]

# Maximum depth of trees
max_depth = [4,6,8]

# Minimum number of samples required to split a node
min_samples_split = [50,100,150]

# Minimum number of samples required at each leaf node
min_samples_leaf = [40,50]

# HYperparameter Grid
parameter_dict = {'n_estimators' : n_estimators,
              'max_depth' : max_depth,
              'min_samples_split' : min_samples_split,
              'min_samples_leaf' : min_samples_leaf}
```

Code Sample6 – Best Parameter Grid

Then we need to provide it with the estimator and specify other things such as how many cross-validations sets to evaluate upon.

```
#importing package
from sklearn.model_selection import GridSearchCV
# Create an instance of the GradientBoostingRegressor
gb_model = GradientBoostingRegressor()

# Grid search
gb_grid = GridSearchCV(estimator=gb_model,
                       param_grid = parameter_dict,
                       cv = 5, verbose=2)

gb_grid.fit(X_train,y_train)
```

After fitting GridSearchCV it returns those params with which it got the best score.

```
gb_grid.best_params_
```

```
{'max_depth': 8,
 'min_samples_leaf': 40,
 'min_samples_split': 100,
 'n_estimators': 100}
```

Similarly, we've done it using RandomSearchCV with Random Forest Regression , even if they have improved the final test score by a little difference, they are performing the best when compared with all the other algorithms. So this tuned Random Forest Regression Model can be used to make the final predictions.

Random Search with Cross Validation

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
```

```
In [974]:
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor(random_state = 42)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                               n_iter = 100, scoring='neg_mean_absolute_error',
                               cv = 3, verbose=2, random_state=42, n_jobs=-1,
                               return_train_score=True)

In [975]:
# Fit the random search model
rf_random.fit(X_train,y_train);

Fitting 3 folds for each of 100 candidates, totalling 300 fits

In [976]:
rf_random.best_params_

Out[976]:
{'n_estimators': 800,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 50,
 'bootstrap': False}
```

Code Sample8 – Hyperparameter Tuning

Performance Metrics

Just building is not enough, as we need to evaluate it using different metrics based on the problem we're solving. Model Validation helps us to understand how well the model is generalizing on the real-world data, the data which it has not seen during the training phase.

For regression problems the evaluation metrics we've used are:

- RMSE (Root Mean Squared Error)
- MSE (Mean Squared Error)
- MAE (Mean Absolute Error)
- MAPE (Mean Absolute Percentage Error)
- Adjusted R2

The R2 score is between 0 and 1 (it can also get -ve when the model is performing worse). The closer the value to 1 the better the performance of the model will be and a model which always predicts constant value irrespective of the input values gets an R2 score of 0.

MAE is on average how far those predicted values are from actual values, whereas MSE is the average of squared differences between actual and predicted values.

Let's look at these metrics for the best-performing algorithms on the test set, Here are the top 10 algorithms based on the test score, MAE, MSE, MAPE, and Adjusted R2. Depending on increasing or decreasing which metric helps solve the business problem, we can pick the appropriate model for final deployment.

Note: Other than Test scores all scores are sorted in ascending order

		Model	MAE	MSE	RMSE	R2_score	Adjusted R2
Training set	0	Linear regression	0.463	0.566	0.752	0.440	0.43
	1	Lasso regression	0.640	1.011	1.006	0.000	-0.02
	2	Ridge regression	0.463	0.566	0.752	0.440	0.43
	3	Elastic net regression	0.455	0.584	0.764	0.423	0.41
	4	Dicision tree regression	0.255	0.182	0.427	0.820	0.82
	5	Random forest regression	0.138	0.060	0.244	0.941	0.94
	6	Gradient boosting regression	0.284	0.218	0.466	0.785	0.78
	7	Xtreme Gradient boosting regression	0.051	0.006	0.076	0.994	0.99
	8	Gradient Boosting gridsearchcv	0.283	0.240	0.490	0.763	0.76
	9	Random forest regression gridsearchcv	0.358	0.006	0.076	0.584	0.58
	10	Xtreme Gradient Boosting gridsearchcv	0.096	0.020	0.142	0.980	0.98
Test set	0	Linear regression	0.477	0.599	0.774	0.383	0.37
	1	Lasso regression	0.626	0.972	0.986	-0.000	-0.02
	2	Ridge regression	0.477	0.599	0.774	0.383	0.37
	3	Elastic net regression Test	0.456	0.602	0.776	0.381	0.37
	4	Dicision tree regression	0.458	0.695	0.834	0.285	0.27
	5	Random forest regression	0.366	0.392	0.626	0.597	0.59
	6	Gradient boosting regression	0.363	0.218	0.466	0.586	0.58
	7	Xtreme Gradient boosting regression	0.358	0.006	0.076	0.584	0.58
	8	Gradient Boosting gridsearchcv	0.365	0.370	0.608	0.619	0.61
	9	Random Forest gridsearchcv	0.116	0.057	0.240	0.943	0.94
	10	Xtreme Gradient Boosting gridsearchcv	0.328	0.308	0.555	0.683	0.68

Table4 - Final Algorithm Performance Table

5. Final Interpretation/Recommendation

In our analysis, we initially did EDA on all the features of our datset. One of the important insight was the relation between number of year and price(target variable). Next we analysed categorical variable and dropped off the irrelevant columns. we also analysed numerical variable, check out the correlation, distribution and their relationship with the dependent variable. We also removed some numerical features and label encoded the categorical variables.

Next we implemented 8 machine learning algorithms Linear Regression ,lasso , ridge , elasticnet , decission tree, Random Forest , GBoost and XGBoost. We did hyperparameter tuning to improve our model performance.

- overfitting is seen for random forest regression,gradient boosting,xtreme gradient boosting.
- Xtreme Gradient Boosting gridsearchcv gives the highest R2 score of 98% for Train Set and Gradient Boosting gridsearchcv gives the highest R2 score of 68% for Test set,which is decent.
- We can deploy this model.

Final Conclusions:

- The ensemble models have performed well compared to that of linear, Decision Tree.
- The best performance is given by the Xtreme Gradient boosting model.
- The top key features that drive the price of the wine are:
  rating,year,wine,acidity,region etc.
- We can conclude from above that xtreme gradient boosting GridSearchCV is giving better results compared to that of normal gradient boosting.
- The above data is also reinforced by the analysis done during bivariate analysis.