YASH RANDIVE

pandas

TensorFlow

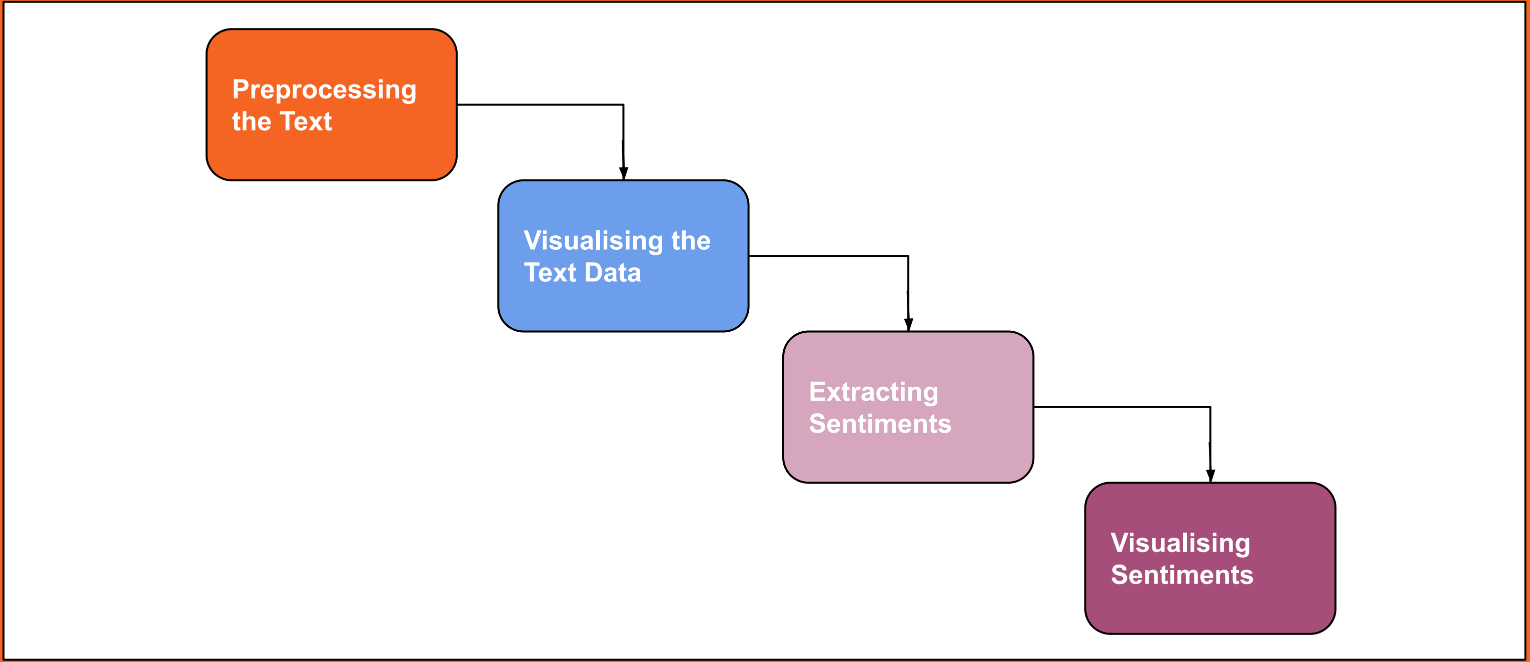
scikit learn

Restaurant Rating Prediction Using Sentiment Analysis and Machine Learning

Part III-a

YASH RANDIVE

QUICK RECAP



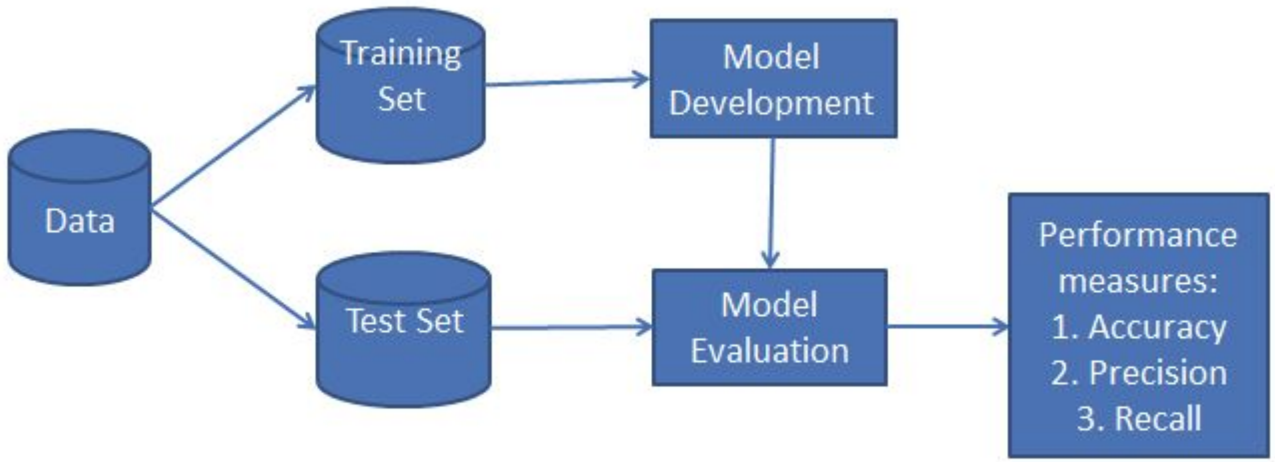
YASH
RANDIVE

Predictive Modelling

- ➔ Logistic Regression
- ➔ K-Nearest Neighbour Classifier
- ➔ Naive Bayes Classification
  - ◆ Gaussian NB
  - ◆ Bernoulli NB
  - ◆ Multinomial NB
- ➔ Random Forest Classifier

YASH
RANDIVE

Basic Modelling Process



YASH
RANDIVE

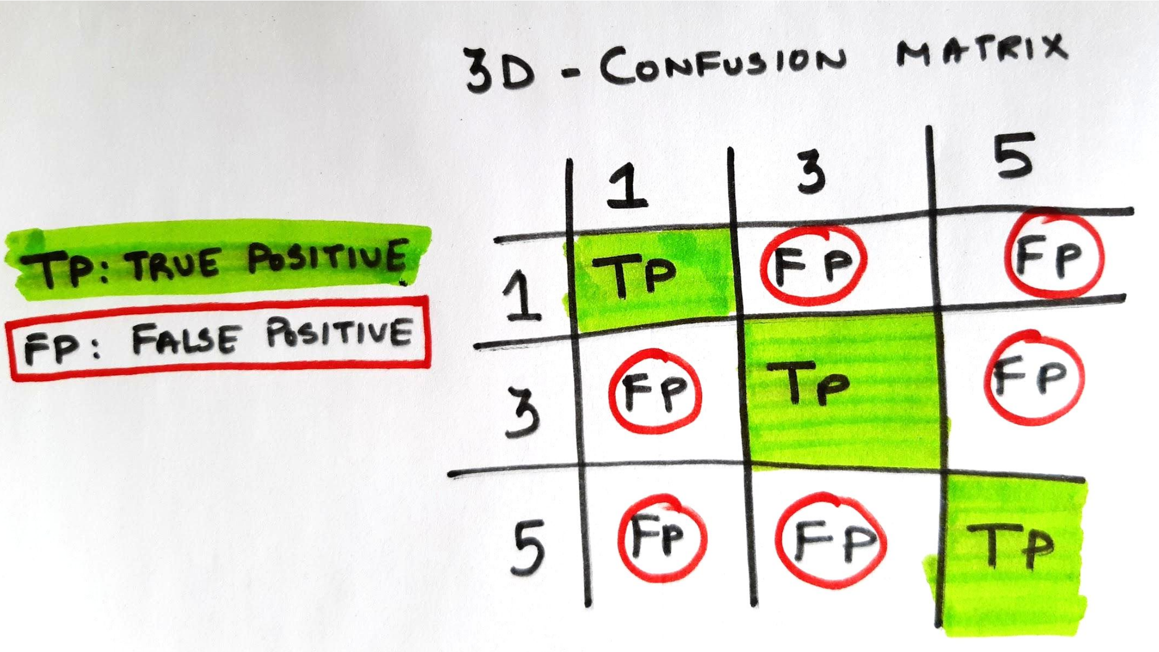
Splitting the train Data into Training and Testing Sets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(inputs, target
s, test_size = 0.1, random_state = 365)
```

What is a Confusion Matrix?

In the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.

What is a Confusion Matrix?



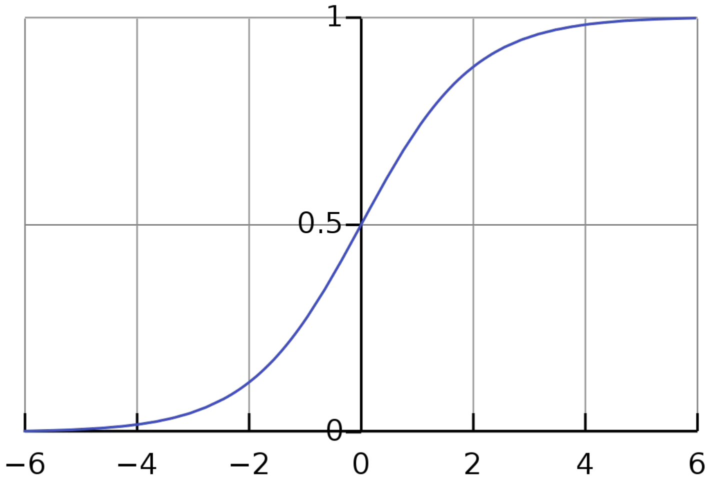
The Logistic Regression Model

1. Import relevant Libraries for the Regression
2. Create the Logistic Regression Model and fit the training data(x_train) to it
3. Evaluate the model by making predictions on the trained model using the x_test model testing data
4. Examine the Confusion Matrix and calculate the accuracy
5. Save the predictions in a .csv file

YASH RANDIVE

The Logistic Regression Model

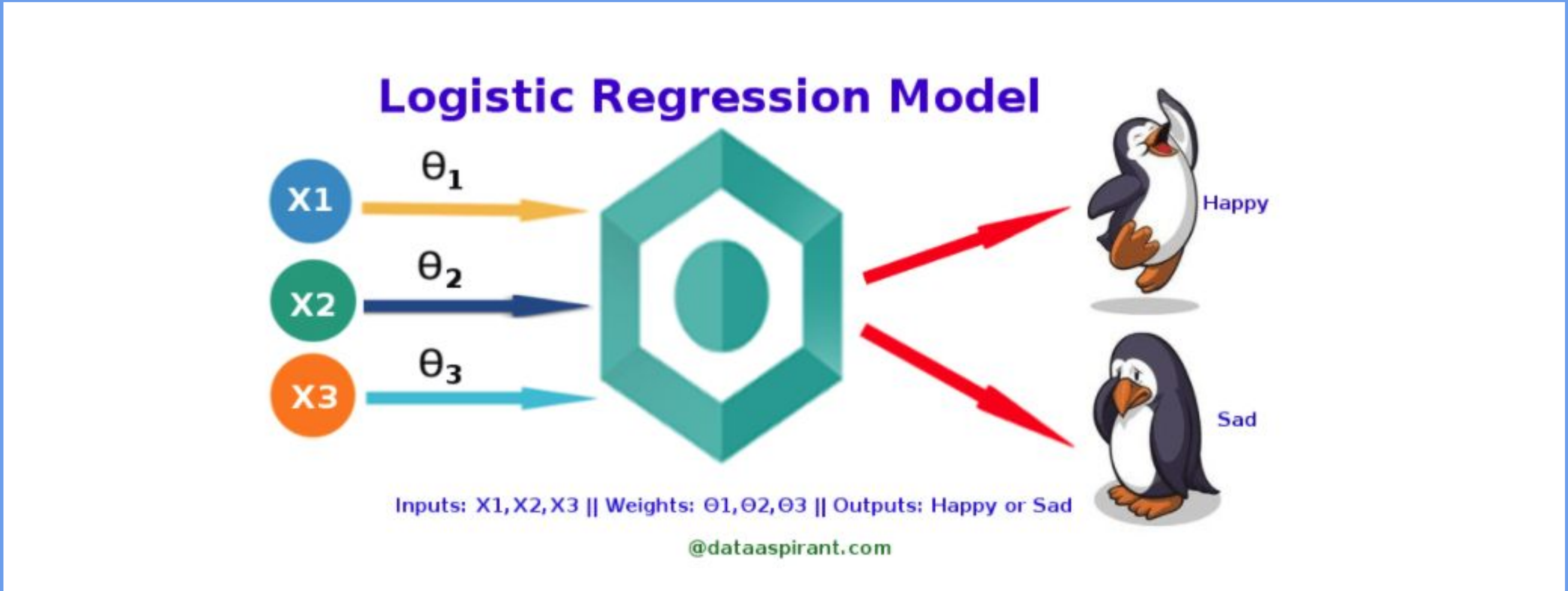
The Sigmoid Function



Transforms and represents all values between 0 and 1

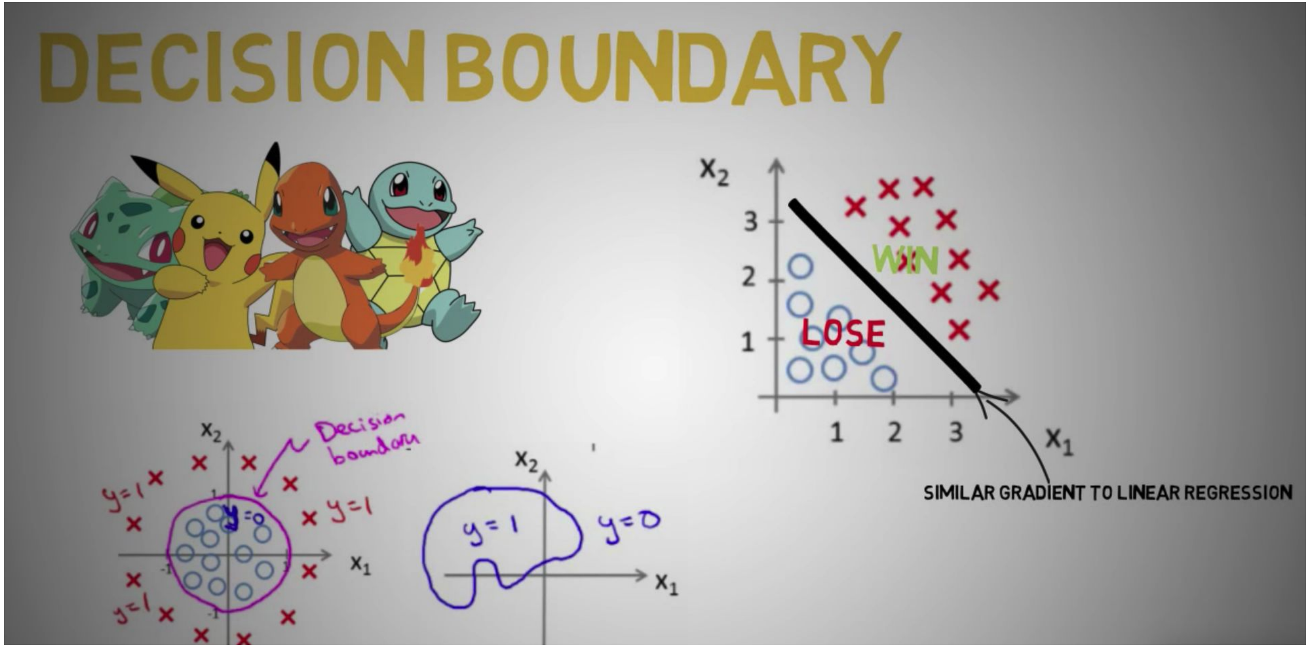
YASH RANDIVE

The Logistic Regression Model



YASH
RANDIVE

The Logistic Regression Model



Defining the Inputs and Targets (or Features and Targets)

YASH RANDIVE

```
targets = df_train['rating']
inputs = df_train.drop(['review','rating'], axis = 1)
```

Targets = rating
Inputs = vote_funny, vote_cool, vote_useful, polarity, subjectivity

Visualising the Distribution of Features against the Targets

YASH RANDIVE

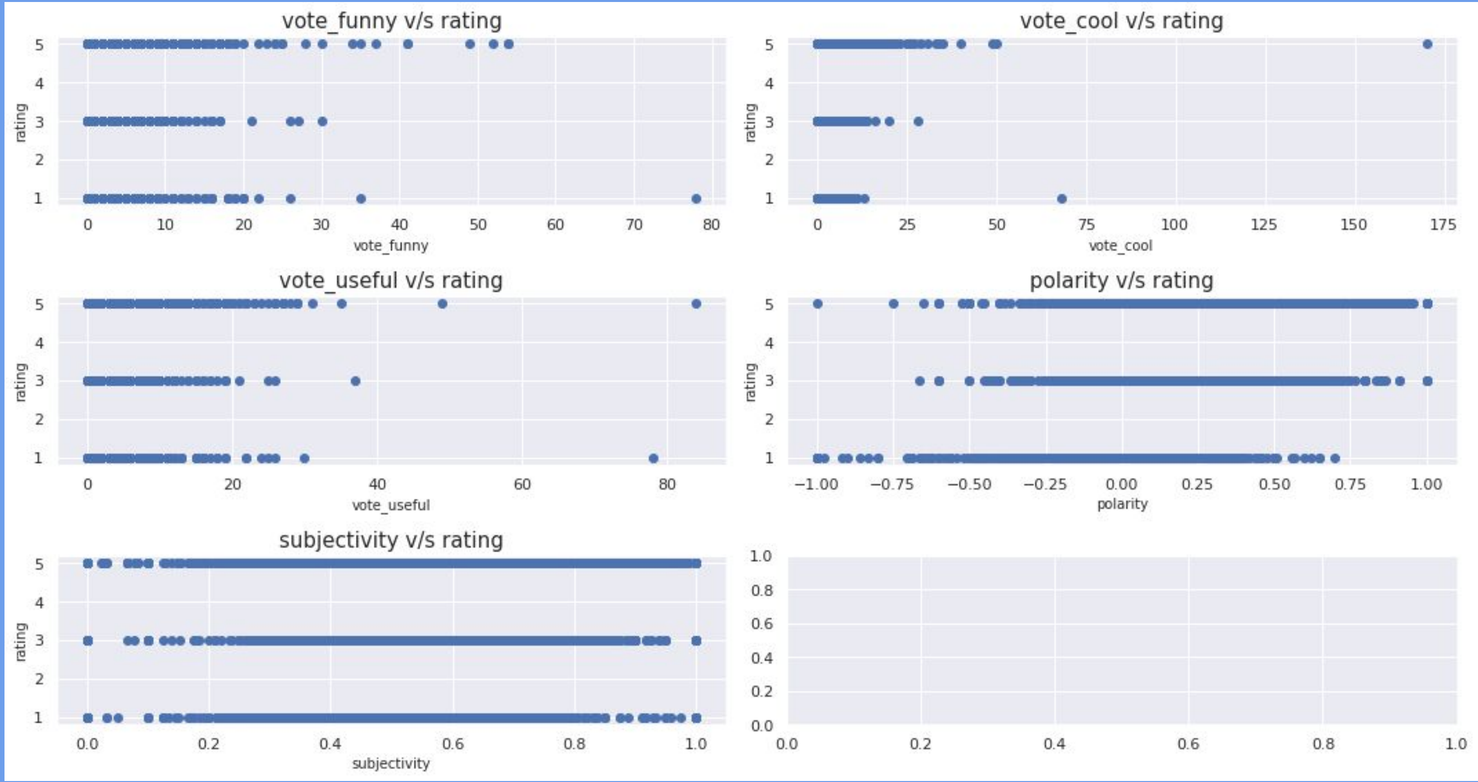
```
fig, axes = plt.subplots(nrows=3, ncols=2)
fig.set_figheight(8)
fig.set_figwidth(15)

axes[0,0].scatter(inputs['vote_funny'], targets)
axes[0,0].set_title('vote_funny v/s rating', size = 15)
axes[0,0].set_xlabel('vote_funny', size = 10)
axes[0,0].set_ylabel('rating', size = 10)

axes[0,1].scatter(inputs['vote_cool'], targets)
axes[0,1].set_title('vote_cool v/s rating', size = 15)
axes[0,1].set_xlabel('vote_cool', size = 10)
axes[0,1].set_ylabel('rating', size = 10)
```

```
axes[1,0].scatter(inputs['vote_useful'], targets)
axes[1,0].set_title('vote_useful v/s rating', size = 15)
axes[1,0].set_xlabel('vote_useful', size = 10)
axes[1,0].set_ylabel('rating', size = 10)

axes[1,1].scatter(inputs['polarity'], targets)
axes[1,1].set_title('polarity v/s rating', size = 15)
axes[1,1].set_xlabel('polarity', size = 10)
axes[1,1].set_ylabel('rating', size = 10)

axes[2,0].scatter(inputs['subjectivity'], targets)
axes[2,0].set_title('subjectivity v/s rating', size = 15)
axes[2,0].set_xlabel('subjectivity', size = 10)
axes[2,0].set_ylabel('rating', size = 10)

fig.tight_layout()

plt.savefig('logistic_regression_data_distribution_scatter_plot
s.png', dpi = 150)
```

Note :All these lines of code are meant to be typed and executed in the same cell i.e. a single cell

Visualising the Distribution of Features against the Targets

YASH RANDIVE



Importing relevant libraries for the Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Training the Logistic Regression Model on x_train and Predicting the Outputs on x_test

```
log_reg = LogisticRegression()
log_results  = log_reg.fit(x_train, y_train)
```

```
y_pred = log_reg.predict(x_test)
```

Examining the Confusion Matrix to derive the Accuracy of the Model

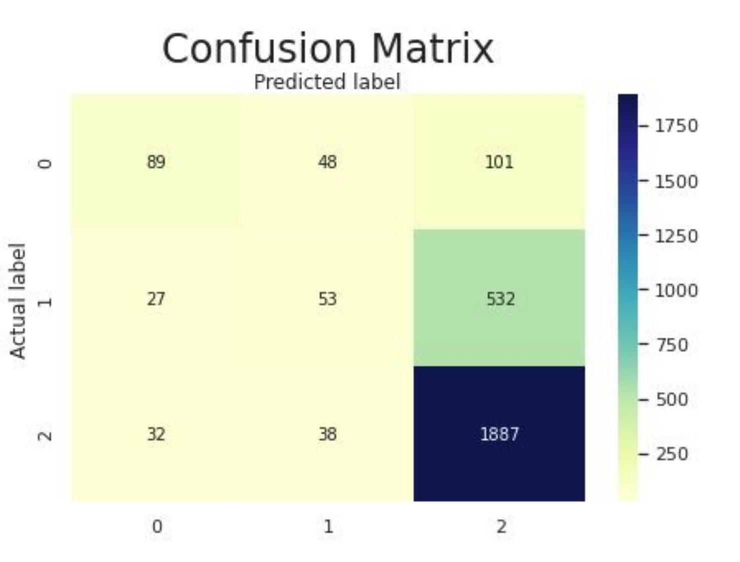
```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
```

```
array([[  89,   48,  101],
       [  27,   53,  532],
       [  32,   38, 1887]])
```

YASH
RANDIVE

Plotting the Confusion Matrix

```
class_names = [1,3,5]
fig, ax = plt.subplots()
tick_marks = np.arange(2)
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu"
,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1, size = 24)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')

fig.savefig('logistic_regression_confusion_matrix.png', dpi = 15
0)
```



Accuracy =
(89+53+1887)/(48+101+27+38+32+5
32+89+53+1887) = 0.722 ≈ 72.2%

a. Printing the testing accuracy of the model
b. Making predictions on the df_test data
c. Saving the predictions to a .csv file

YASH RANDIVE

a

```
print("Accuracy: "+str((metrics.accuracy_score(y_test, y_pred)*100).round(3))+"%")
```

```
Accuracy: 72.284%
```

b

```
y_pred_logistic_regression = log_reg.predict(df_test.drop(['review'], axis = 1))
```

c

```
pd.DataFrame(y_pred_logistic_regression).to_csv("logistic_regression_predictions.csv")
```

The K-Nearest Neighbour Classifier (KNN)

1. Import relevant Libraries for the Classification
2. Create the KNN Classifier
3. Train the classifier on x_train
4. Make predictions from the model using the x_test data and evaluate the model on the basis of the predictions
5. Make predictions on the df_test dataset
6. Save the predictions in a .csv file

Fundamentals of the KNN Algorithm

The KNN Classifier works on 3 basic fundamentals,

1. WHO are my neighbours?
2. WHAT class do they belong to?
3. I will be the SAME CLASS as the majority in my proximity

Note : the 'K' in KNN stands for the Number of Surrounding neighbours

YASH RANDIVE

The KNN Algorithm is based on calculating the 'Distance' between neighbours

> EUCLIDEAN DISTANCE,
>
> HAMMING DISTANCE,
>
> MANHATTAN DISTANCE (CITY BLOCK)
>
> MINKOWSKY
>
> CHEBYCHEV DISTANCE

Some frequently used distance functions.

Camberra:

$$d(x, y) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|} \quad (2)$$

Minkowsky:

$$d(x, y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r} \quad (3)$$

Chebychev:

$$d(x, y) = \max_{i=1}^{m} |x_i - y_i| \quad (4)$$

Euclidean:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \quad (5)$$

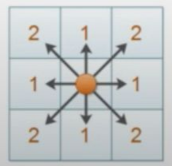
Manhattan / city - block:

$$d(x, y) = \sum_{i=1}^{m} |x_i - y_i| \quad (6)$$

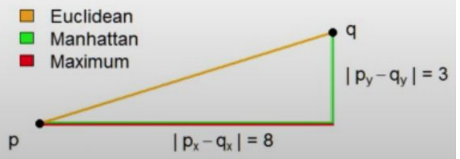
Euclidean:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan Distance

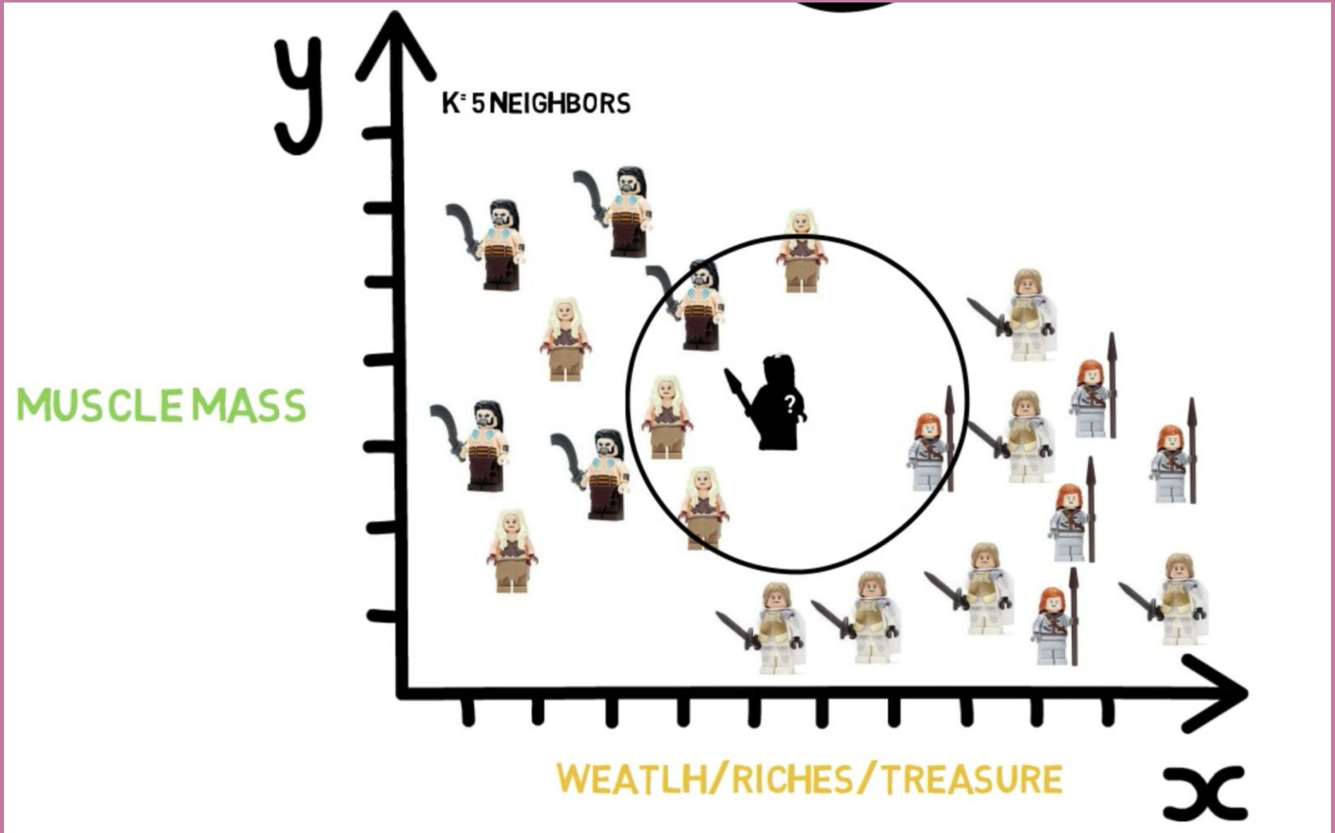


$$|x_1 - x_2| + |y_1 - y_2|$$



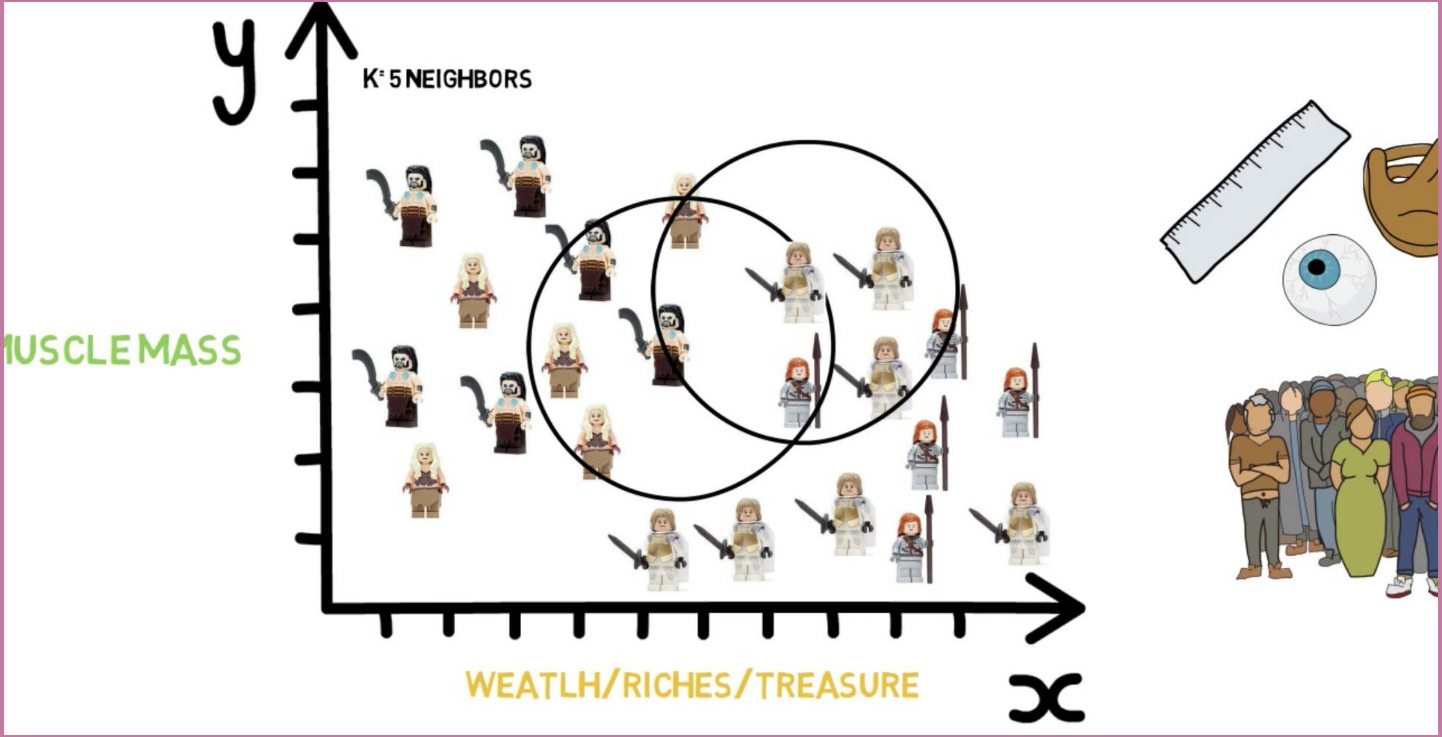
The KNN Classifier

YASH
RANDIVE



The KNN Classifier

YASH RANDIVE



The KNN Classifier

YASH
RANDIVE

```python
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier

#Create KNN Classifier
knn = KNeighborsClassifier(n_neighbors=125)

#Train the model using the training sets
knn.fit(x_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(x_test)
```

Calculating the Accuracy of the KNN Classifier

YASH
RANDIVE

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Calculating the Model Accuracy
print("Accuracy:" + str((metrics.accuracy_score(y_test, y_pred)*
100).round(3))+"%")
```

```
Accuracy:73.424%
```

Making Predictions using our model on df_test and saving the predictions to a .csv file

YASH RANDIVE

```
y_pred_KNN_predictions = knn.predict(df_test.drop(['review'], ax
is = 1))
```

Saving the KNN Model Predictions to a .csv file

```
pd.DataFrame(y_pred_KNN_predictions).to_csv("KNN_predictions.cs
v")
```

YASH
RANDIVE

The Naive Bayes Classifier(s)

1. Import relevant Libraries for the Classification
2. Create the Naive Bayes Classifier
3. Train the classifier on x_train
4. Make predictions from the model using the x_test data and evaluate the model on the basis of the predictions
5. Make predictions on the df_test dataset
6. Save the predictions in a .csv file

Fundamentals of the Naive Bayes Algorithm

The Naive Bayes Classifier works on 4 basic steps,

1. Calculate the prior probability of class tables
2. Find the likelihood probability for each attribute of each class
3. Put these values in the Bayes formula and calculate the posterior probability

Note : the 'Naive Bayes Classifier' generally performs better than Logistic Regression and requires lesser training data

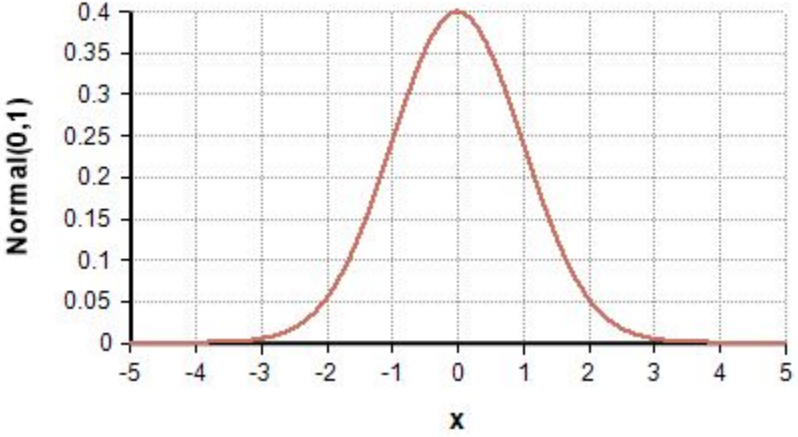
Bayes Formula

$$\underbrace{P(c \mid x)}_{\text{Posterior Probability}} = \frac{\overbrace{P(x \mid c)}^{\text{Likelihood}}\;\overbrace{P(c)}^{\text{Class Prior Probability}}}{\underbrace{P(x)}_{\text{Predictor Prior Probability}}}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Types of the Naive Bayes Algorithm

1. **Gaussian NB : It is used in classification tasks and assumes that the features follow a normal distribution**



YASH
RANDIVE

Types of the Naive Bayes Algorithm

2. **Multinomial NB** : It is used for discrete counts i.e. number of times ‘x’ is observed in n_trials

3. **Bernoulli NB** : This type of Naive Bayes is used for Binary Classification(0,1)

Applications of the Naive Bayes Algorithm

1. Real Time Predictions
2. Multiclass Predictions
3. Text Classification/ Spam Filtering and Sentiment Analysis
4. Recommendation System

Implementing Gaussian NB

YASH
RANDIVE

Gaussian Naive Bayes

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(x_train, y_train);
```

```
pred = model.predict(x_test)
```

```
acc = model.score(x_test,y_test)
print("Accuracy(Gaussian Naive Bayes) = " + str((acc*100).round
(3))+"%")
```

```
Accuracy(Gaussian Naive Bayes) = 62.95%
```

Implementing Multinomial NB

YASH
RANDIVE

Multinomial Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

model_mn = MultinomialNB()
model_mn.fit(x_train.drop(['polarity'], axis = 1), y_train)


accuracy_multinomial_nb = model_mn.score(x_test.drop(['polarit
y'], axis = 1), y_test)
print("Accuracy (Multinomial Naive Bayes): "+ str((accuracy_mult
inomial_nb*100).round(3)) + "%")
```

```
Accuracy (Multinomial Naive Bayes): 70.039%
```

Implementing Bernoulli NB

YASH
RANDIVE

Bernoulli Naive Bayes

```
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics
from sklearn.metrics import accuracy_score

BernNB = BernoulliNB(binarize = False)
BernNB.fit(x_train, y_train)

y_expected = y_test
y_pred = BernNB.predict(x_test)

print("Accuracy (Bernoulli Naive Bayes): "+ str(((accuracy_score
(y_expected, y_pred))*100).round(3)) + "%")
```

```
Accuracy (Bernoulli Naive Bayes): 71.25%
```

Since the Bernoulli NB model has the best accuracy we predict df_test using the Bernoulli NB model and save the predictions to a .csv file

YASH RANDIVE

```
y_pred = BernNB.predict(df_test.drop(['review'], axis = 1))
```

```
pd.DataFrame(y_pred).to_csv("Bernoulli_Naive_Bayes_Predictions.csv")
```

The Random Forest Classifier

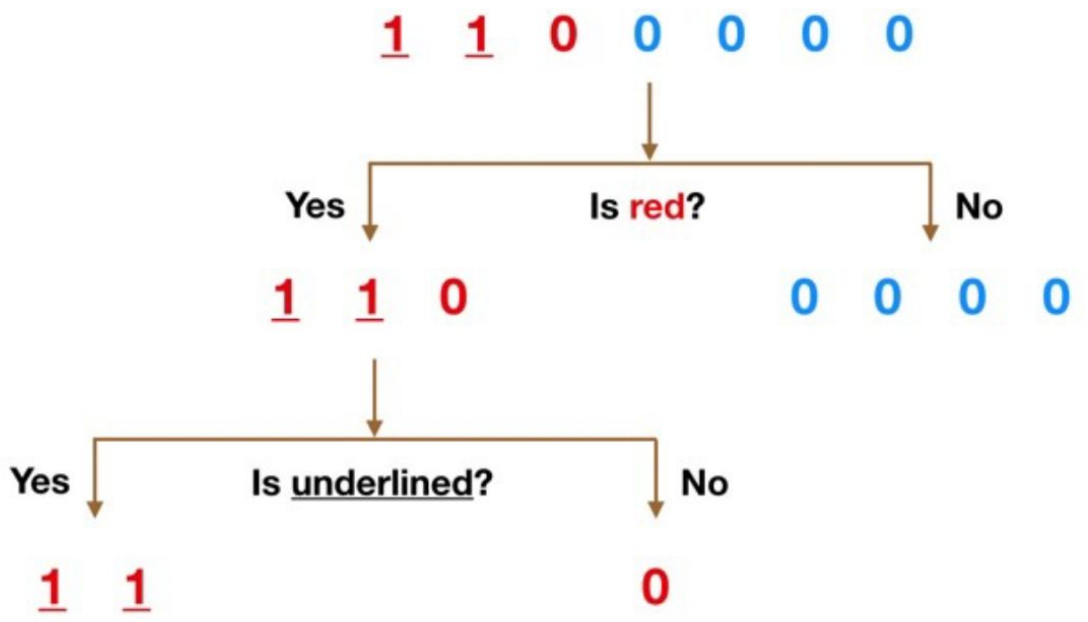
1. Import relevant Libraries for the Classification
2. Create the Random Forest Classifier
3. Train the classifier on x_train
4. Make predictions from the model using the x_test data and evaluate the model on the basis of the predictions
5. Make predictions on the df_test dataset
6. Save the predictions in a .csv file

Fundamentals of the Random Forest Classifier

1. As its name implies it consists of a number of decision trees that operate as an ensemble
2. Each individual decision tree in a random forest outputs a class prediction and the class with the "Most Votes" becomes our model's prediction

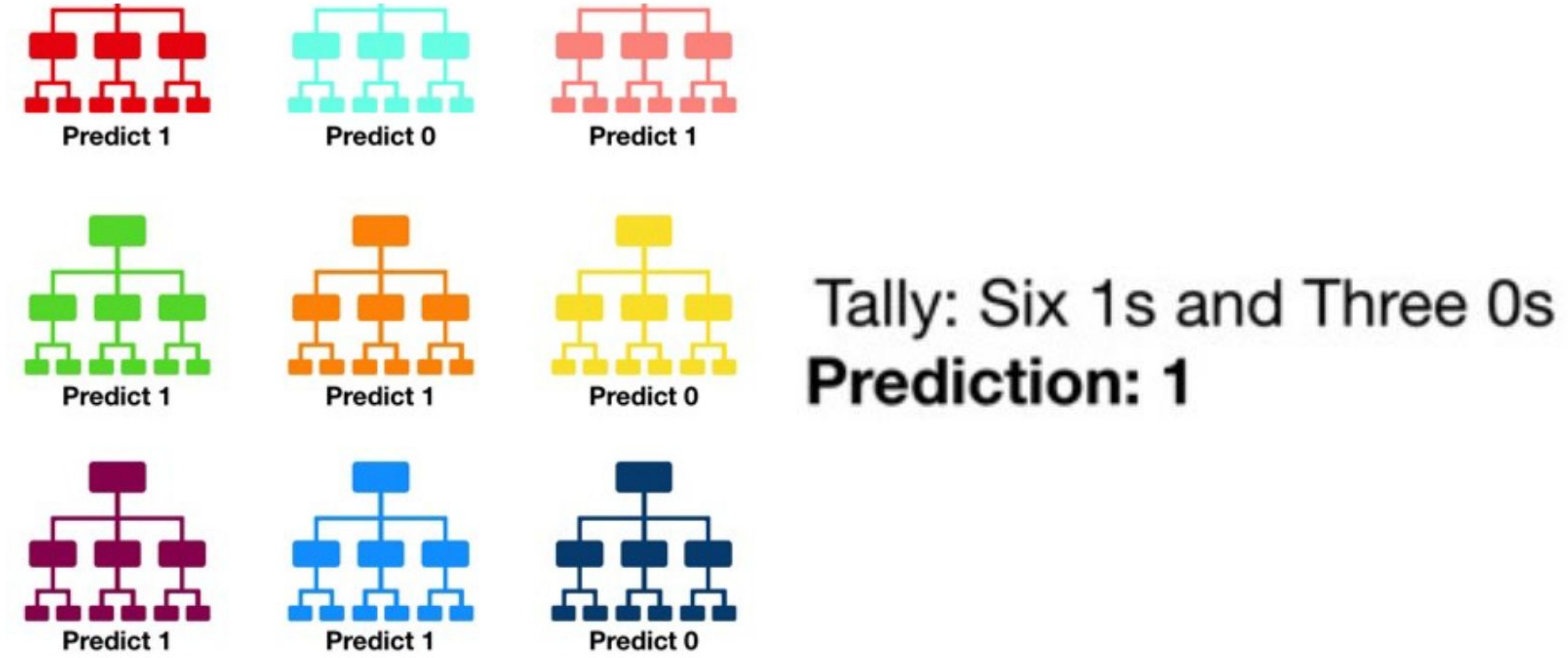
Note : A large number of relatively uncorrelated trees(models) operating as a committee will outperform any individual constituent models

Decision Tree Model



Simple Decision Tree Example

Random Forest Model



Note : Random Forest Classifier

1. The underlying decision trees should have low correlation - this allows them to protect each-other from their individual errors

2. Bagging(Bootstrap Aggregation) :
   a. Decision trees can change completely if the smallest of features of the underlying data changes
   b. Random Forests take advantage of this by allowing each individual tree to randomly sample data/features from the dataset thereby resulting different trees.
   c. This process is called Bagging.

YASH
RANDIVE

Note : Random Forest Classifier

1. The underlying decision trees should have low correlation - this allows them to protect each-other from their individual errors

2. Bagging(Bootstrap Aggregation) :
   a. Decision trees can change completely if the smallest of features of the underlying data changes
   b. Random Forests take advantage of this by allowing each individual tree to randomly sample data/features from the dataset thereby resulting different trees.
   c. This process is called Bagging.

YASH
RANDIVE

Importing relevant Libraries and Fitting the Model

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier(n_estimators = 1515, criterion
= 'gini', random_state = 45)
model.fit(x_train,y_train)
```

Importing relevant Libraries and Fitting the Model

```
y_pred = model.predict(x_test)
```

Validating Performance of the Random Forest Model

```
from sklearn.metrics import confusion_matrix
```

Confusion Matrix for the Random Forest Model

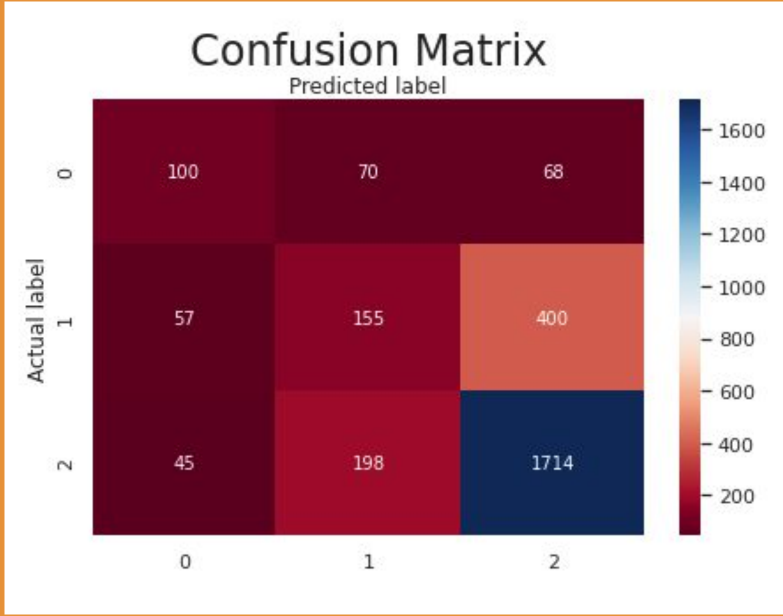
```
cm = confusion_matrix(y_test,y_pred)
cm
```

Plotting the Confusion Matrix and Printing the Accuracy

YASH RANDIVE

```
class_names = [1,3,5]
fig, ax = plt.subplots()
tick_marks = np.arange(1)
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cm), annot=True, cmap="RdBu" ,fmt
='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion Matrix', y=1.1, size = 24)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

```
Text(0.5, 257.44, 'Predicted label')
```




```
print("Accuracy = "+ str(((model.score(x_test,y_test))*100).r
ound(3))+"%")
```

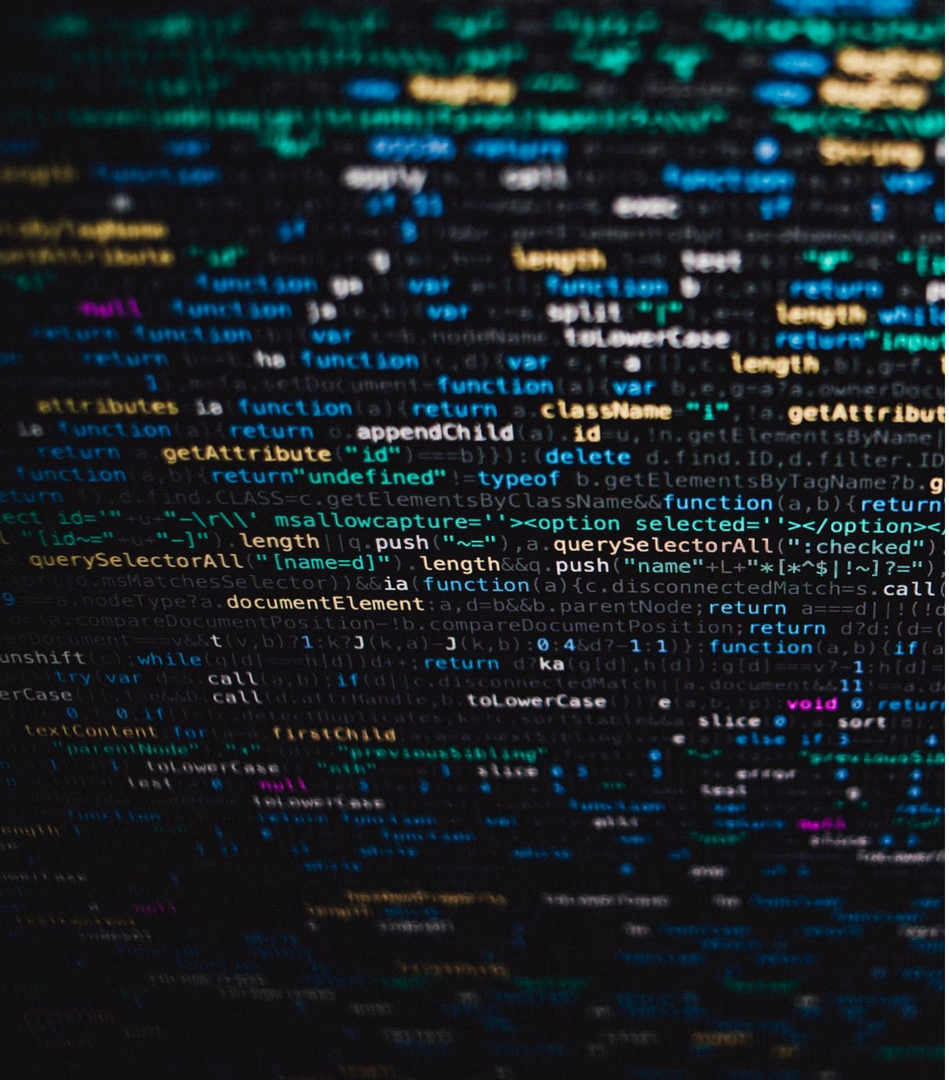
```
Accuracy = 70.146%
```

Making predictions on df_test data set and saving the predictions as a .csv file


```
y_pred = model.predict(df_test.drop(['review'], axis = 1))
```

Saving the Predictions of the Random Forest Classifier to a .csv file

```
pd.DataFrame(y_pred).to_csv("Random_Forest_Classifier_predictions.csv")
```



YASH
RANDIVE

Next Up

Predictive Analysis

1. Support Vector Machine(SVM) Classifier
2. Deep Neural Network (Tensorflow 2.0)

Congratulations!

You have reached the end of this part of the project and are all set to move ahead!

For queries or questions in the code reach me at :

yashrandive.datascience@gmail.com

All the Best!