Que 1. Review Game

Data preparation, cleaning and visualisation process

Data has been imported in the form of Json Lines. Then data fame was created from the 3 arrays. Those 3 arrays are Reviews, Voted Up and Early Access. Since data is given in the form of text in different languages.

I have used **TfidfVectorizer,** in each model to convert the text data set into a vector space where each term of vector is indexed as my index vocabulary.

Once data as created in the form of data frame, I have created the function for data cleaning where reviews **data was cleaned by converting all text to lower case and by removing brackets, punctuations, number and other symbols**. As you can see in the fig 1 and fig 2.

```
data = {'Review': X, 'Voted_Up': y, 'Early_Access':z}
df = pandas.DataFrame(data=data)
print(df)
```

```
                                                 Review  Voted_Up  Early_Access
0     Единственная серия игр в которой мне понравили...      True         False
1                                       kruta vashe imba      True         False
2                                Je recommande fortement =)      True         False
3     W chuj dobra grafika nie porównywalna do call ...      True         False
4                                     Świetna gra!!! :)      True         False
...                                                 ...       ...           ...
4995  In Racer 8 geht es schlicht und einfach darum ...     False         False
4996                              Странная игра не советую!     False         False
4997  It's super lame. Thankfully it was on sale, bu...     False         False
4998  It's a fidget spinner game. You, uh, spin the ...     False         False
4999  Even though the gameplay is clunky as hell, I ...     False          True
```

Fig 1: Raw reviews text data.

```
df['Review']=pandas.DataFrame(df.Review.apply(cleaned))
X=np.array(df.Review)
y=np.array(df.Voted_Up)
z=np.array(df.Early_Access)
print(df)
```

```
                                                 Review  Voted_Up  Early_Access
0     единственная серия игр в которой мне понравили...      True         False
1                                       kruta vashe imba      True         False
2                                   je recommande fortement      True         False
3     w chuj dobra grafika nie porównywalna do call ...      True         False
4                                          świetna gra      True         False
...                                                 ...       ...           ...
4995  in racer  geht es schlicht und einfach darum e...     False         False
4996                              странная игра не советую     False         False
4997  its super lame thankfully it was on sale but t...     False         False
4998  its a fidget spinner game you uh spin the thin...     False         False
4999  even though the gameplay is clunky as hell i s...     False          True
```

Fig 2: Cleaned reviews text data.

After text cleaning, I have **visualised** the data response based on the reviews.

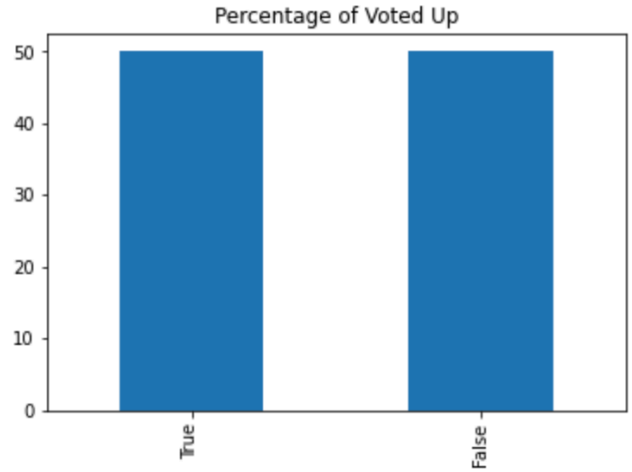


Fig 3: Response for Voted Up

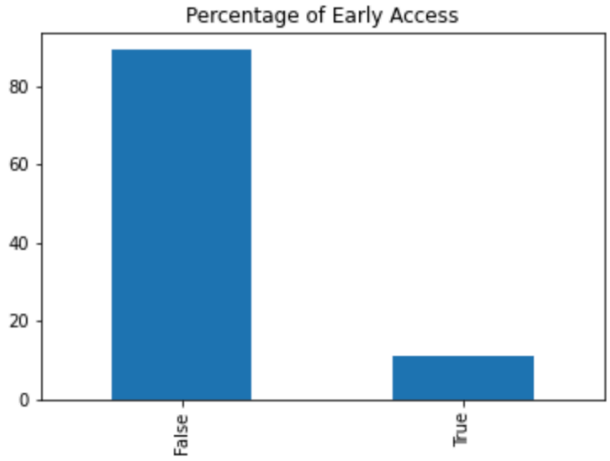


Fig 4: Response for Early Access.

As the above bar graph of fig 3, 50% users gave vote up for the game and other 50% users did not like the game. The data is fairly even and good for prediction.

But in the fig 4, only 10% of reviews are giving true value and rest of 90% reviews have false value. Hence the data set for this scenario is not good. It will give **bad prediction as the target data set has 1:9 ratio of data which might leads to under-fitting because high biased data.**

Data set was split into 75:25 where 75% was selected to be training and remaining was testing data. Since for the Early Access data, the true and false ration is only 1:9. So I took 25% of testing data instead of 20% of testing data.

1.1 Review Polarity (Voted Up Prediction)

Basically I used **3 different machine learning approaches** to create the model for predicting whether the review is for voted up or not.

Those approaches are:

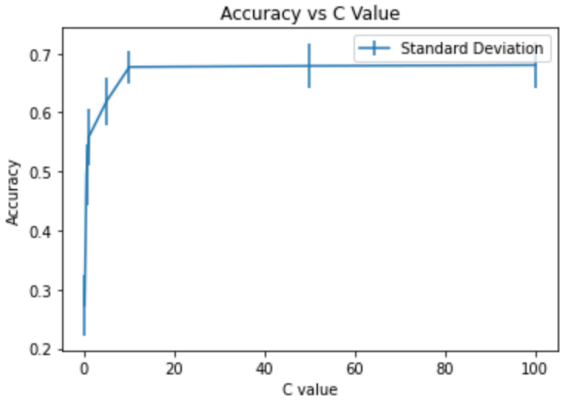
1.1.1 Logistic Regression Approach for review polarity:

First I performed the K Fold cross validations to check the best value of inverse of regularisation strength (C) in order to reduce overfitting and get best accuracy.

While performing the K fold cross validation, **I have used C in the range of 0.1 to 100 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of C.

Since the value of C was is high. So I set **500 maximum number of iterations** taken for the solvers to converge.

```
when C = 0.1 Mean Accuracy = 0.2732 Std = 0.051343548767104126
when C = 0.5 Mean Accuracy = 0.49420000000000003 Std = 0.05056
when C = 1 Mean Accuracy = 0.5586 Std = 0.04884301383002485
when C = 5 Mean Accuracy = 0.6196 Std = 0.040425734377992455
when C = 10 Mean Accuracy = 0.6772 Std = 0.027337885799746824
when C = 50 Mean Accuracy = 0.679 Std = 0.03783384727991589
when C = 100 Mean Accuracy = 0.6804 Std = 0.03838541389642684
```



As you can see the value of accuracy and standard deviation of each value of C.

For C=100, accuracy is highest but the standard deviation is also highest. Hence this is not a good option to choose.

But for C=10, accuracy is 3rd highest along with least value of standard deviation. So **I have used the parameter C=10 for the Logistic regression model.**

Once I got the best value of inverse of regularisation strength (C) parameter, I have used TfidfVectorizer and Logistic regression model into a **pipeline** and then trained the model. Also in **Logistic regression, two parameters are being used. C=10 and since value of C was high, Max iteration set to 500.**

Then I have created the confusion matrix along with the heap map as you can see in the fig 6.1.

Fig 6.1: Heat map of confusion matrix.

Fig 6.2: Confusion Matrix value and other values.

```
Predicted  False  True
Actual
False        465    166
True         150    469

Accuracy:  0.7472
Precision:  0.7474241905127713
Recall:  0.7472
F1 Score:  0.7480063795853269
```

In fig 6.2, for the Logistic model: **Number of True Negatives: 465 False Positives: 166 False Negatives: 150 True Positives : 469.** Hence, the number of true negative and true positive is fairly high so **model is performing good**.

Also, fig 6.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.7471 as an accuracy** is good having almost same f1 score which shows this model is well-fitted.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'worst game'.  Vote Up Result: False
Review:'happy with this game'.  Vote Up Result: True
```

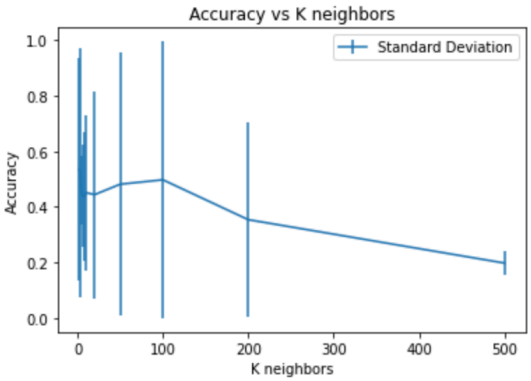
Here, I wrote 2 reviews, first one is bad review and second one is good review. Hence the voted up result is same as expected from the model sentiment. So model is working perfectly.

1.1.2 KNN Regression Approach for review polarity:

Again I performed the K Fold cross validations to check the best N-neighbours in order to reduce overfitting and get best accuracy.

While performing the K fold cross validation, **I have used N-neighbors in the range of 2 to 500 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of N-neighbours.

```
when n = 2 Mean Accuracy = 0.5356 Std = 0.39870068974106376
when n = 3 Mean Accuracy = 0.5229999999999999 Std = 0.447448097
when n = 4 Mean Accuracy = 0.45920000000000005 Std = 0.12292501
when n = 6 Mean Accuracy = 0.43979999999999997 Std = 0.18279048
when n = 8 Mean Accuracy = 0.4366 Std = 0.22994964666204643
when n = 10 Mean Accuracy = 0.4514 Std = 0.2789954121486588
when n = 20 Mean Accuracy = 0.44400000000000006 Std = 0.3717429
when n = 50 Mean Accuracy = 0.4816 Std = 0.4741137416274706
when n = 100 Mean Accuracy = 0.4974 Std = 0.49740289504585716
when n = 200 Mean Accuracy = 0.354 Std = 0.348003448258778
when n = 500 Mean Accuracy = 0.19799999999999998 Std = 0.042350
```



In this given fig 7, the X-axis displaying the different values of N-neighbors and Y axis with accuracy.

For the value N=2, accuracy is highest but standard deviation is also highest which is bad for the model.

Hence, I have selected N-neighbors = 4. As it has 3rd highest accuracy along with significantly low standard deviation.

Fig 7: Accuracy vs K Neighbors.

Once I got the best value of N-neighbors parameter for KNN model, I have used TfidfVectorizer and KNN regression model into a **pipeline** and then trained the model. Also **in KNN regression, one parameters are being used other are default that is N-neighbors=4.**

Then I have created the confusion matrix along with the heap map as you can see in the fig 8.1.

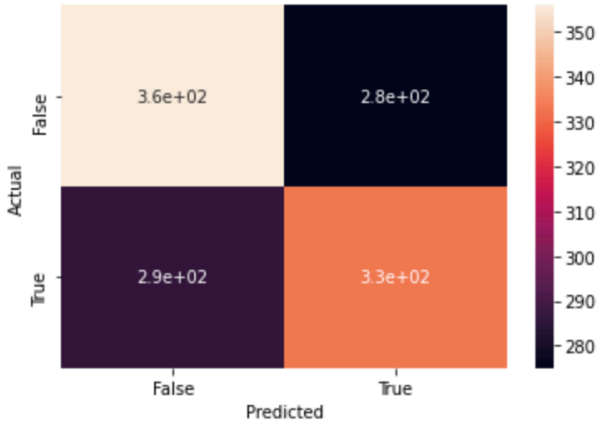


Fig 8.1: Heat map of confusion matrix.

```
Predicted  False  True
Actual
False        356    275
True         286    333

Accuracy:  0.5512
Precision:  0.5511399860632891
Recall:  0.5511999999999999
F1 Score:  0.5427872860635696
```

Fig 8.2: Confusion matrix and other values.

In fig 8.2, for the Logistic model: **Number of True Negatives: 356 False Positives: 275 False Negatives: 286 True Positives : 333.** The number of true negative and true positive is low, also false negative and false positivity is high. So **this model is not good as it has failed to give good number of true or real prediction** as you can see from confusion matrix value.

Again, fig 8.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.5512 as an accuracy is bad for any model**. So KNN is not working good for this data set.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'worst game'.  Vote Up Result: True
Review:'happy with this game'.  Vote Up Result: True
```

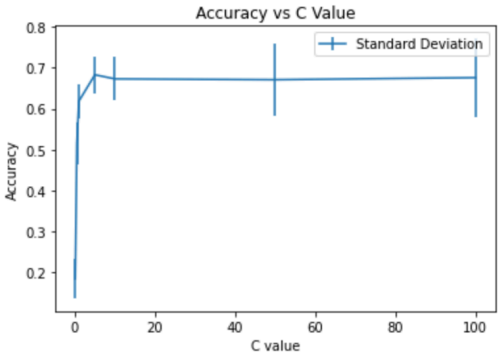
Here, I wrote 2 reviews, first one is bad review and second one is good review. But this KNN model is not giving good accuracy. So, the **1st review voted up result is not same as expected from the model sentiment.** Because you can see in the 1st review sentiment was not for voted up as it was a bad review but still its's giving true value for the voted up.

1.1.3 SVM (Support Vector Machine) Regression Approach for review polarity:

First I performed the K Fold cross validations to check the best value of inverse of regularisation strength (C) in order to reduce overfitting and get best accuracy.

While performing the K fold cross validation, **I have used C in the range of 0.1 to 100 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of C.

```
when C = 0.1 Mean Accuracy = 0.185 Std = 0.04741518744031283
when C = 0.5 Mean Accuracy = 0.5149999999999999 Std = 0.05167
when C = 1 Mean Accuracy = 0.6182 Std = 0.04185164273956282
when C = 5 Mean Accuracy = 0.6826000000000001 Std = 0.0450958
when C = 10 Mean Accuracy = 0.6726 Std = 0.052794317876074505
when C = 50 Mean Accuracy = 0.6706 Std = 0.08949659211389002
when C = 100 Mean Accuracy = 0.6756 Std = 0.09505703551026617
```



As you can see the value of accuracy and standard deviation of each value of C.

For C=5, accuracy is highest but the standard deviation is also high. Hence this is not a good option to choose.

But for C=1, accuracy is average but it has lowest standard deviation. So **I have used the parameter C=1 for the SVM regression model.**

Once I got the best value of inverse of regularisation strength (C) parameter, I have used TfidfVectorizer and SVM regression model into a **pipeline** and then trained the model. Also **in SVM regression, three parameters are being used. C=1, set Kernel as Linear because of large dataset and degree as 3.**

Then I have created the confusion matrix along with the heap map as you can see in the fig 6.1.

Fig9.1: Heat map of confusion matrix.

```
Predicted  False  True
Actual
False        471    160
True         153    466

Accuracy:  0.7496
Precision:  0.7496582329810765
Recall:  0.7496
F1 Score:  0.7485943775100402
```

Fig 9.2: Confusion Matrix value and other values.

In fig 9.2, for the Logistic model: **Number of True Negatives: 471 False Positives: 160 False Negatives: 153 True Positives : 466 .** Hence, the number of true negative and true positive is fairly high even **higher than logistic regression**.

Also, fig 9.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.7496 as an accuracy** is good having almost same f1 score which shows this model is well-fitted.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'worst game'.  Vote Up Result: False
Review:'happy with this game'.  Vote Up Result: True
```

Here, I wrote 2 reviews, first one is bad review and second one is good review. Hence the voted up result is same as expected from the model sentiment. So model is working perfectly.

1.1.4 Conclusion of all three approaches for review polarity.

Also I have add a base line model predicts only 1 that's is true value of voted up. As you can see the confusion matrix value of base line model. True Negative is 0 and True positive is 619, that is all the voted up numbers.

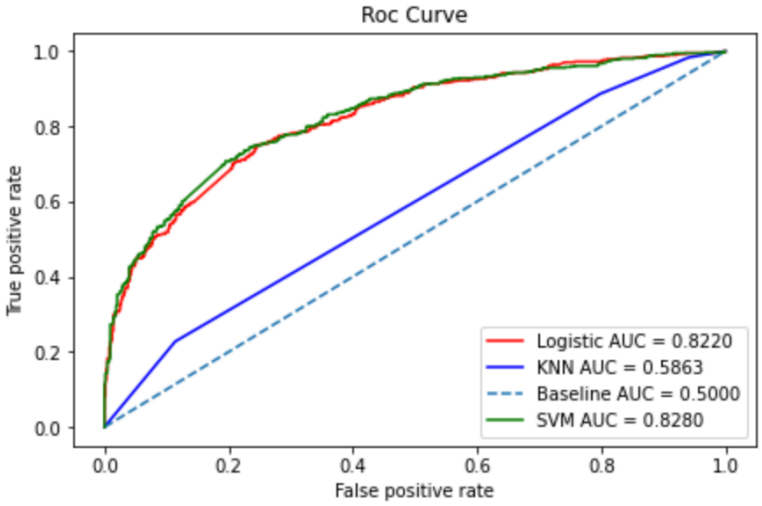
Baseline: Number of True Negatives: 0 False Positives: 631 False Negatives: 0 True Positives : 619

Confusion matrix for KNN, SVM and Logistic Regression:

Logistic: Number of True Negatives: 465 False Positives: 166 False Negatives: 150 True Positives : 469
KNN: Number of True Negatives: 356 False Positives: 275 False Negatives: 286 True Positives : 333
SVM: Number of True Negatives: 471 False Positives: 160 False Negatives: 153 True Positives : 466

Let's compare the confusion matrixes value of all 3 model together. Eliminating the KNN model first because it has lowest number of true negative and true positive value. Now, in SVM and Logistic regression. **SVM is better as value of true negative is significantly greater that Logistic true negative.**

To analyse the performance of all four [Logistic, KNN, SVM and Baseline] approaches, I have drawn ROC Curve.



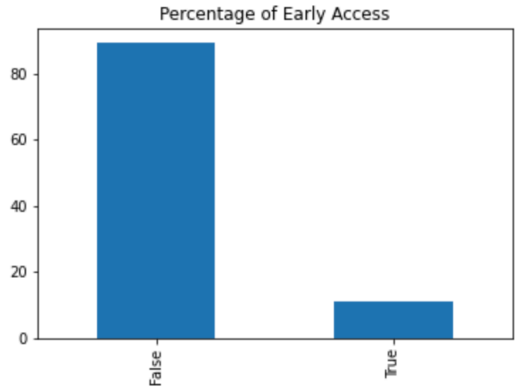
As you can see in left side graph of ROC. The AUC of **SVM (Auc: 0.8280) is best followed by Logistic Regression (Auc: 0.8220)** which is almost same but slightly lesser than SVM.

Also KNN is giving **worst AUC** for this given data set if we exclude the baseline accuracy.

Hence, I would go for SVM regression model because it is giving best AUC result for distinguishing b/w -ve and +ve classes.

SVM is giving best accuracy too if we compare confusion matrix. Since SVM model is creating best decision boundary for this data set for the identification of voted up or not for the given review.

1.2 Review for an Early Access (Version Prediction)



Dataset of review for early access prediction is not even, 9:1 response will cause **Under fitting in the model.**

Data given for the early access are unevenly distributed. There are large number of reviews are based on false early access. Only 10% of reviews are reflecting the response for 'early access version'. **Hence any model will not perform well as most of the model will biased towards for false early access as the ration of false early access is 9:1.**

Again I used 3 different machine learning approaches to create the model for predicting whether the review is for an Early Access version or not.

Those approaches are:

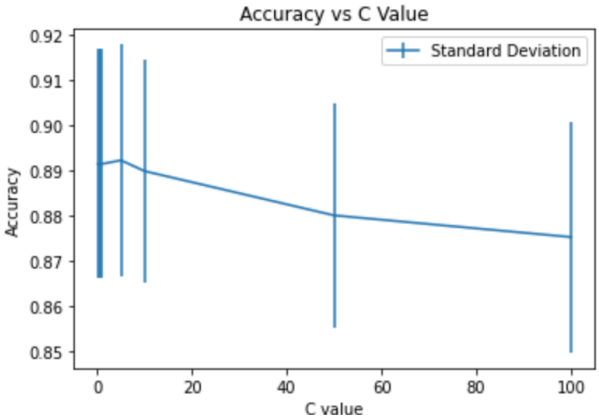
1.2.1 Logistic Regression Approach for version prediction:

First I performed the K Fold cross validations to check the best value of inverse of regularisation strength (C) in order to reduce overfitting and get best accuracy.

While performing the K fold cross validation, **I have used C in the range of 0.1 to 100 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of C.

Since the value of C was is high. So I set **500 maximum number of iterations** taken for the solvers to converge.

```
when C = 0.1 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when C = 0.5 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when C = 1 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when C = 5 Mean Accuracy = 0.8922000000000001 Std = 0.02565073098373614
when C = 10 Mean Accuracy = 0.8897999999999999 Std = 0.02481048165594536
when C = 50 Mean Accuracy = 0.8799999999999999 Std = 0.024915858403835922
when C = 100 Mean Accuracy = 0.8752000000000001 Std = 0.025584370228715837
```



As you can see the value of accuracy and standard deviation of each value of C.

For C=5, accuracy is highest but the standard deviation is also highest. Hence this is not a good option to choose.

But for C=10, accuracy is average but it has least value of standard deviation. So **I have used the parameter C=10 for the Logistic regression model.**

Once I got the best value of inverse of regularisation strength (C) parameter, I have used TfidfVectorizer and Logistic regression model into a **pipeline** and then trained the model. Also **in Logistic regression, two parameters are being used. C=10 and since value of C was high, Max iteration set to 500.**

Then I have created the confusion matrix along with the heap map as you can see in the fig 6.1.

Fig 10.1: Heat map of confusion matrix.

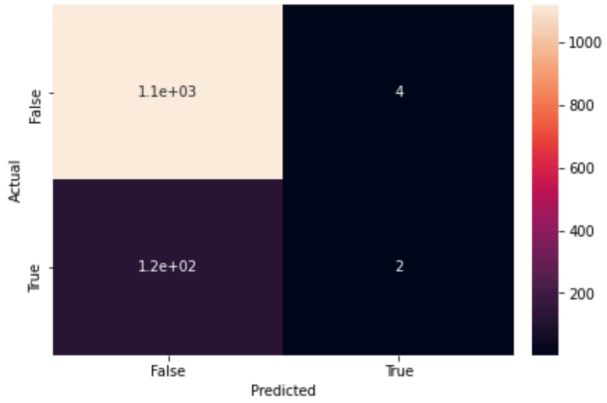


Fig 10.2: Confusion Matrix value and other values.

```
Predicted  False  True
Actual
False       1121     4
True         123     2

Accuracy:  0.8984
Precision:  0.8443461950696678
Recall:  0.8984
F1 Score:  0.030534351145038167
```

In fig 10.2, for the Logistic model: **Number of True Negatives: 1121 False Positives: 4 False Negatives: 123 True Positives : 2.** Hence, the number of true negative and false negative is covering most of the data. As most 90% of reviews are showing that game is not early access. So, **Model is predicting most of the data as not early access because of data imbalance**.

Also, fig 10.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.8984 as an accuracy but this model is not good as it is biased towards 'not early access' prediction.** That's why the **F1 score is significantly less** as balance between precision and recall is not good.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'old game'.  Early Access Result: False
Review:'new upcoming game'.  Early Access Result: False
```

Here, I wrote 2 reviews, first one is for not early access review and second one is early access review. But for **2nd review too, I was getting False value means predicting as an older version** because 90% of data sets representing the value of 'not early access'.

1.2.2 KNN Regression Approach for version prediction:

While performing the K fold cross validation, **I have used N-neighbors in the range of 2 to 500 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of N-neighbours.

But the data is not evenly distributed, It has the review ratio for 'early access' to 'not early access' is 1:9 which will lead to overfitting.

```
when n = 2 Mean Accuracy = 0.8904 Std = 0.025531157435572736
when n = 4 Mean Accuracy = 0.8896000000000001 Std = 0.024426
when n = 6 Mean Accuracy = 0.8911999999999999 Std = 0.025079
when n = 8 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 10 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 20 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 50 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 100 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 200 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when n = 500 Mean Accuracy = 0.8914 Std = 0.0253937000061039
```

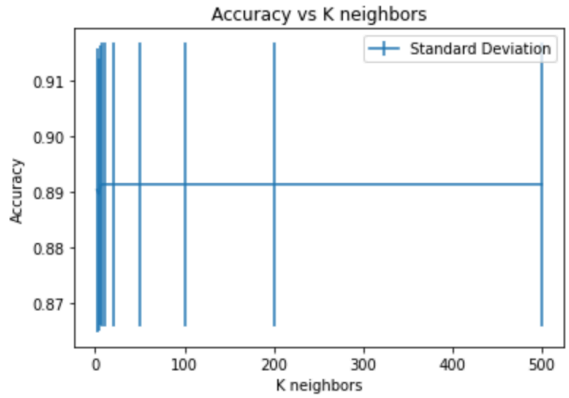


Fig 11: Accuracy vs K Neighbors.

In this given fig 11, the X-axis displaying the different values of N-neighbors and Y axis with accuracy.

For the value N=8, accuracy is highest but standard deviation is also high which is bad for the model. After n=8, this graph is flattened.

Hence, I have selected N-neighbors = 4. As it has 3rd highest accuracy along with lowest standard deviation.

Once I got the best value of N-neighbors parameter for KNN model, I have used TfidfVectorizer and KNN regression model into a **pipeline** and then trained the model. Also **in KNN regression, one parameters are being used other are default that is N-neighbors=4.**

Then I have created the confusion matrix along with the heap map as you can see in the fig 12.1.

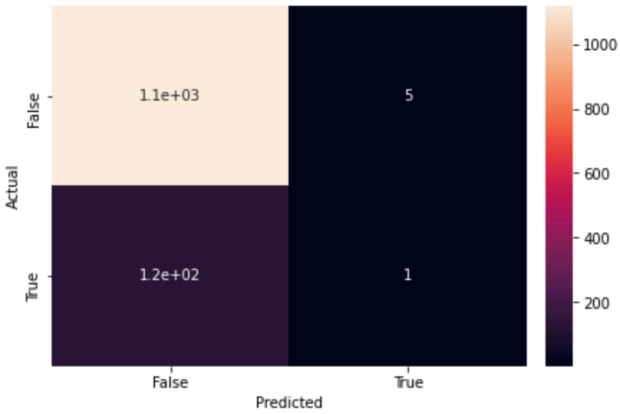


Fig 12.1: Heat map of confusion matrix.

```
Predicted  False  True
Actual
False       1120      5
True         124      1

Accuracy:  0.8968
Precision:  0.8269560557341907
Recall:  0.8968
F1 Score:  0.015267175572519083
```

Fig 12.2: Confusion matrix and other values.

In fig 12.2, for the KNN model: **Number of True Negatives: 1120 False Positives: 5 False Negatives: 124 True Positives : 1.** Hence, the number of true negative and false negative is covering most of the data. As most 90% of reviews are showing that game is not early access. So, **Model is predicting most of the data as 'not early access' because kNN is creating bigger cluster of 'Not Early Access' and predicting all new test points under it**.

Also, fig 12.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.8968 as an accuracy but this model is not good as it is biased towards not early access prediction.** That's why the **F1 score is significantly less** as balance between precision and recall is not good.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'old game'.  Early Access Result: False
Review:'new upcoming game'.  Early Access Result: False
```

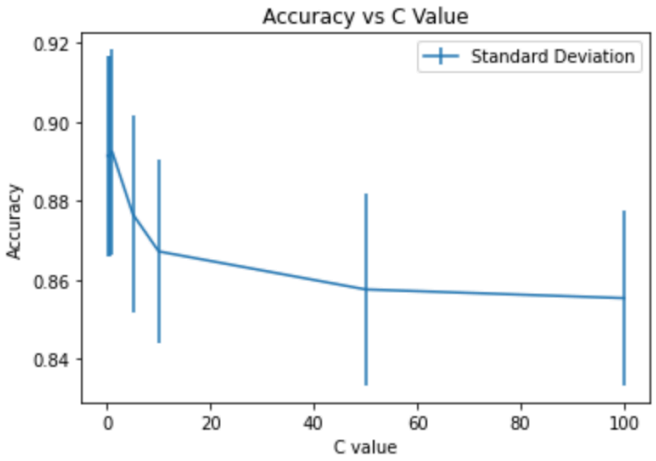
Here, I wrote 2 reviews, first one is for not early access review and second one is early access review. But for **2nd review too, I was getting False value means predicting as an older version** because 90% of data sets representing the 'not early access' means it formed bigger cluster of 'Not Early Access' and predicting this review under it .

1.2.3 SVM (Support Vector Machine) Regression Approach for version prediction:

First I performed the K Fold cross validations to check the best value of inverse of regularisation strength (C) in order to reduce overfitting and get best accuracy.

While performing the K fold cross validation, **I have used C in the range of 0.1 to 100 with 10 number of folds.** Hence calculated the accuracy and standard deviation for each value of C.

```
when C = 0.1 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when C = 0.5 Mean Accuracy = 0.8914 Std = 0.0253937000061039
when C = 1 Mean Accuracy = 0.8924 Std = 0.025827117531772 71
when C = 5 Mean Accuracy = 0.8766 Std = 0.02493270943960967
when C = 10 Mean Accuracy = 0.8672000000000001 Std = 0.0232929
when C = 50 Mean Accuracy = 0.8576 Std = 0.0242619042945932
when C = 100 Mean Accuracy = 0.8554 Std = 0.021964516839666673
```



As you can see the value of accuracy and standard deviation of each value of C.

For C=5, accuracy is highest but the standard deviation is also highest. Hence this is not a good option to choose.

But for C=10, accuracy is average but it has 2nd lowest standard deviation. So **I have used the parameter C=10 for the SVM regression model.**

Once I got the best value of inverse of regularisation strength (C) parameter, I have used TfidfVectorizer and SVM regression model into a **pipeline** and then trained the model. Also **in SVM regression, three parameters are being used. C=10, set Kernel as Linear because of large dataset and degree as 3.**

Fig13.1: Heat map of confusion matrix.

```
Predicted  False  True
Actual
False       1091     34
True         114     11

Accuracy:  0.8816
Precision:  0.8392992162286768
Recall:  0.8816
F1 Score:  0.12941176470588234
```

Fig 13.2: Confusion Matrix value and other values.

Then I have created the confusion matrix along with the heap map as you can see in the fig 13.1.

In fig 13.2, for the Logistic model: **Number of True Negatives: 1091 False Positives: 34 False Negatives: 114 True Positives : 11 .** Again, the number of true negative and false negative is covering most of the data. As most 90% of reviews are showing that game is not early access. But this time **number true positive and false positive is better compared to KNN and Logistic**.

Also, fig 13.2 is showing Accuracy, precision, recall and F1 Score (Balance between precision and recall). **0.8816 as an accuracy but this model is not good as it is biased towards not early access prediction.** That's why the **F1 score is significantly less but still more than KNN and Logistic** as balance between precision and recall is not that much good.

I have tied **2 sample review example** on this model to predict the voted up.

```
Review:'old game'.  Early Access Result: False
Review:'new upcoming game'.  Early Access Result: False
```

Here, I wrote 2 reviews, first one is for not early access review and second one is early access review. But for **2nd review too, I was getting False value means predicting as an older version** because 90% of data sets representing the false value of early access.

1.2.4 Conclusion of all three approaches for review polarity.

Also I have add a base line model predicts only 1 that's is true value for early access. As you can see the confusion matrix value of base line model. True Negative is 0 and True positive is 125, that is all the early access reviews. Other are for older version of game.

Baseline: Number of True Negatives: 0 False Positives: 1125 False Negatives: 0 True Positives : 125

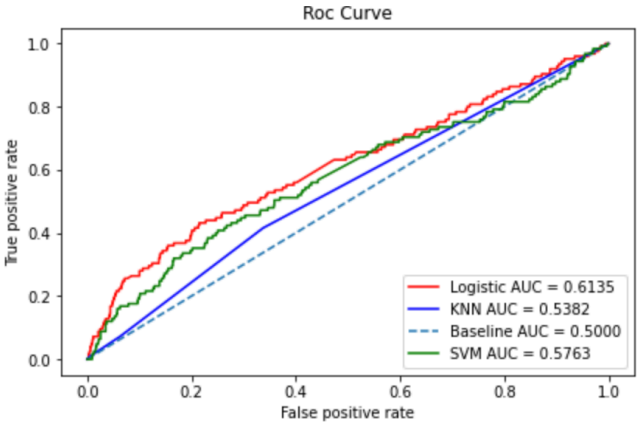
Confusion matrix for KNN, SVM and Logistic Regression:

Logistic: Number of True Negatives: 1121 False Positives: 4 False Negatives: 123 True Positives : 2
KNN: Number of True Negatives: 1120 False Positives: 5 False Negatives: 124 True Positives : 1
SVM: Number of True Negatives: 1091 False Positives: 34 False Negatives: 114 True Positives : 11

Let's compare the confusion matrixes value of all 3 model together. **SVM is better as value of true positive is highest as model is already biased towards 'not early access' value still SVM managed to get 11 True positive value which far more compared to KNN and Logistic model.**

Also if we compare the F1 Score, SVM has the highest F1 score among all of the model because F1 score indicated the balance between precision and recall.

To analyse the performance of all four [Logistic, KNN, SVM and Baseline] approaches, I have drawn ROC Curve.



As you can see in left side graph of ROC. The area under the curve of **Logistic Regression (Auc: 0.6135) is best followed by SVM (Auc: 0.5763)** which is almost same but slightly lesser than Logistic.

Also KNN is giving **worst AUC** for this given data set if we exclude the baseline accuracy.

Here Logistic regression is giving the best AUC results followed by SVM. But all the models giving the bad AUC results as models are failing in distinguishing between the positive and negative classes.

If we **compare the f1 score of both SVM and Logistic regression. SVM is the winner.**

F1 Score for SVM: 0.1294

F1 Score for Logistic Regression: 0.03053

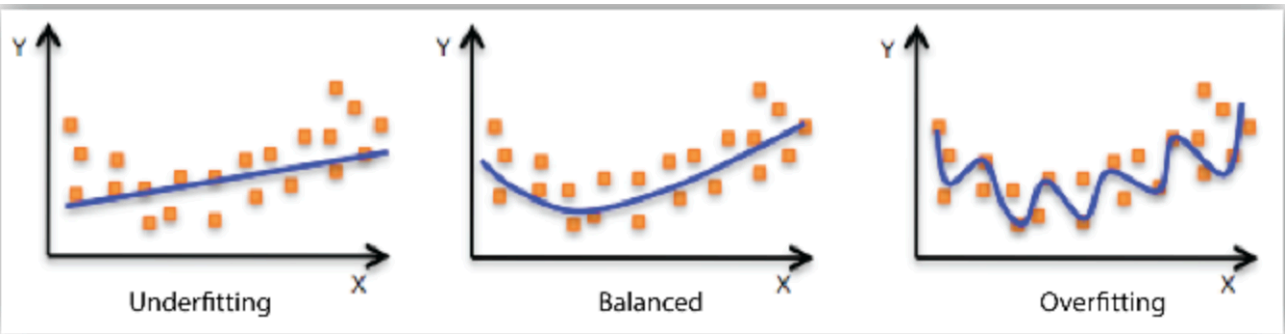
F1 Score for KNN: 0.0152

(Note: For F1 Score of each model, please refer fig 10.2, 12.2, 13.2 in the above pages.)

Balance between recall and precision is better in SVM. So I would go for **SVM regression model as data set is not proper as it has significantly highest F1- Score. It was important to measure the F1 score because of data imbalance (9:1).**

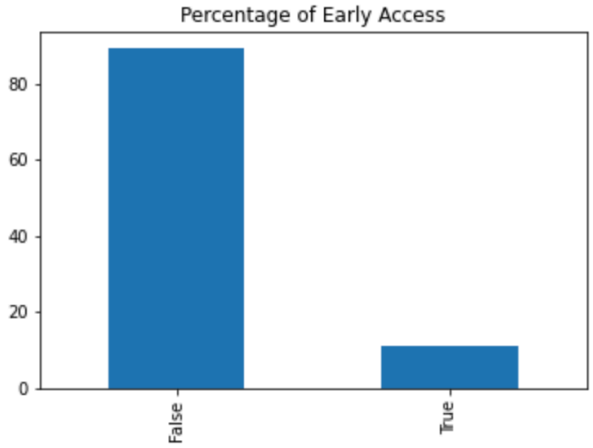
But **none of the model was good for prediction as all the models are under fitted** because it was biased towards 'not early access' reviews because reviews for 'not early access' was 90%.

2.1 Under-fitting and Over-fitting



Under-fitted model means that we do not have enough parameters to capture the trends in the underlying system for the prediction. For example that we have data that is quadratic in nature, but we try to fit this with a linear function, with just one parameter as being major in the given data set. As the function does not have the required complexity to fit the data (two parameters), we end up with a poor predictor. In this case the model will have **high bias**. This means that we will get same answers as it will biased, but biased is a wrong answers.So, if model is *under-fitting* when the model performs badly on the training data. As the model is failed to understand the relationship between the input data (X) and the target data (Y).

For Example:



In previous question 1.2, for game review prediction whether the review is for early access version or not. The review data parameter was unevenly distributed. Only 10% of review was showing the early access version. Other 90% of the reviews was showing 'not early access'. Hence the **function highly biased towards 'not early access'** prediction and the **complexity of the model is almost linear** as model is unable to capture the relationship between reviews to early access version prediction.

Fig: Response for Early Access.

Over-fitted means there are a lot of parameters to be justified by the actual working of data and hence, this cause a creation of complex model. If we imagine that the true system is a quadratic, but used an upper order polynomial degree to fit into it. As we have natural noise in the data used to fit (deviations from the perfect quadratic), the overly complex model treats these complexity and noise as if they were intrinsic properties of the model and attempts to fit to them. Hence, it results to **high variance** for this model. This means that we will not get consistent predictions of future results.

Model will overfitting only when the model working well on the training data but fails to perform good on the evaluation data. This is because the model is memorising the data it has seen and is unable to generalise or predict to unseen examples.

For Example:
If a person is good in remembering the answers, words and paragraphs of a given book (Dataset) but bad in understanding the concept and reason behind the answers, then that person will sure get good marks in small test asked from the same book. But if the test is bigger and questions has been asked from other books having same concept of solution. Then person will fail to score marks. As person has memorised every thing from one book but been able to generalise the concept.

2.2 Pseudo-Code Implementing K-Fold Cross-Validation.

1. Array of well defined hyper-parameter combinations, C, for current model.

2. Data divided into number of K folds.

3. (Main for loop) For fold-X, in the K folds:

   3.1. Each fold-X = test dataset
   3.2. Selection of features on the K-1 folds, remove one fold (k1) for computing accuracy.
   3.3. (Loop for Array C) For parameter c in C:

      3.3.1. (inner loop) For fold-Y in the remaining K-1 folds:

         3.3.1.1. Use fold fold-Y as a validation dataset
         3.3.1.2. Training of model on remaining K-2 folds
         3.3.3.3. Compute the accuracy using fold k1

      3.3.2. Calculate average performance over K-2 folds for parameter c

   3.4. Training of model on K-1 folds using hyper-parameter combination that have best average performance combined.
   3.5. Evaluation of model using fold -X.

4. Calculate and print overall performance of K folds combined together.

2.3 Cross-Validation To Strike A Balance Between Over/Under-Fitting.

Since K fold cross validation divides the data into k small parts and uses one part as a testing data and other k-1 part as a training data. This process gives use k numbers of J(θ) which can be use to estimate the accuracy and the spread of the values for each J(θ). Hence we are mean accuracy of different combination of data set (Training and Testing data).

Here we get **to check the accuracy of different combination of data set (Training and Testing data) then we are overcoming the problem of overfitting**. As we are calculating the performance of model using multiple data set combinations also Cross-validation allows you to tune hyper-parameters with only your original training set. This allows us to keep your test set as a truly unseen dataset for selecting your final model.

As I have already implement the K-fold cross validation on Logistic regression for question 1:

```
when C = 0.1 Mean Accuracy = 0.2732 Std = 0.051343548767104126
when C = 0.5 Mean Accuracy = 0.49420000000000003 Std = 0.05056
when C = 1 Mean Accuracy = 0.5586 Std = 0.04884301383002485
when C = 5 Mean Accuracy = 0.6196 Std = 0.040425734377992455
when C = 10 Mean Accuracy = 0.6772 Std = 0.027337885799746824
when C = 50 Mean Accuracy = 0.679 Std = 0.03783384727991589
when C = 100 Mean Accuracy = 0.6804 Std = 0.03838541389642684
```

For each and every value of C (inverse of regularisation strength) parameter in logistic regression, K fold was set to 10. So data was divided into 10 equal parts and 9 was used for training and 1 was used testing. Then for each C value mean accuracy and standard deviation was calculated.

Now, now we need to select the minimum C value having best accuracy and minimum standard deviation **as smaller the C value, lower the risk of over fitting**.

Under-fitting is when a model does not estimate the variable well in either the original data or new data. Your model is missing some variables that are necessary to better estimate and predict the behaviour of your dependent variable.

So K fold uses different training and testing data set in different round of k fold number to explore new combination of data set. **This will create enough data for the model to learn the population pattern to overcome the under fitting.**

2.4 Pros and Cons Of A Logistic Regression Vs A kNN Classifier.

KNN classifier is based on distance techniques while Logistic regression is totally based on probability.
As KNN (K-nearest Neighbours) plots the data points (your training data) into a vector space and while prediction it plots your test data-point and finds the k-nearest neighbours.
But Logistic regression is a discriminative algorithm, meaning it tries to find beast decision boundaries between two classes. Thus the logistic models work on probabilities of one class to another.

1. Logic Regression:

Pros:

1. Chances of overfitting are less in logistic regression as it's a linear model. Especially with the C regularisation parameter can easily take control of any overfitting problem.

2. Logistic Regression is not a data set hungry model unlike kNN and this makes it suitable for some simple applications.

3. It is very easy to implement and fast at classifying unknown records.

Cons:

1. Logistic Regression inherently runs on a linear model. This means even more restriction when it comes to implementing logistic regression.

2. Logistic Regression is not immune to missing data. Hence data cleaning is very important part for this model.

3. It constructs linear boundaries also it is important to have more number of observation else it may leads to overfitting.

2. kNN Classifier:

Pros:

1. No training period is needed for the kNN as the data itself is a model which will be the reference for future prediction.

2. It is a non-parametric algorithm which means there are no assumptions to be met to implement K-NN.

3. This model can be used both for classification and regression problems.

Cons:

1. K-NN inherently has no capability of dealing with missing value problem.

2. As it chose the neighbors based on distance criteria, so this model would be more sensitive and critical for decision boundary outliner.

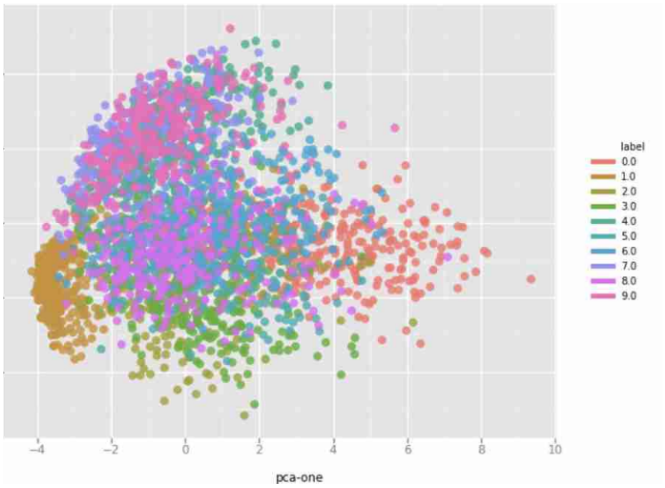
3. It will not work well in high dimensional problem as distance calculating process will get more complicated with increase in dimension.

2.5 Two Failure Example For kNN Model.

Case 1:

When data is randomly jumbled.

As you can see in left image, data having multiple level are randomly jumbled. If we want to predict the sample data for this dataset using kNN, then kNN will fail to give correct answer as there is no proper cluster are formed. All the data points are scattered and prediction will not work as using kNN, model will look for nearest data or belonging cluster.

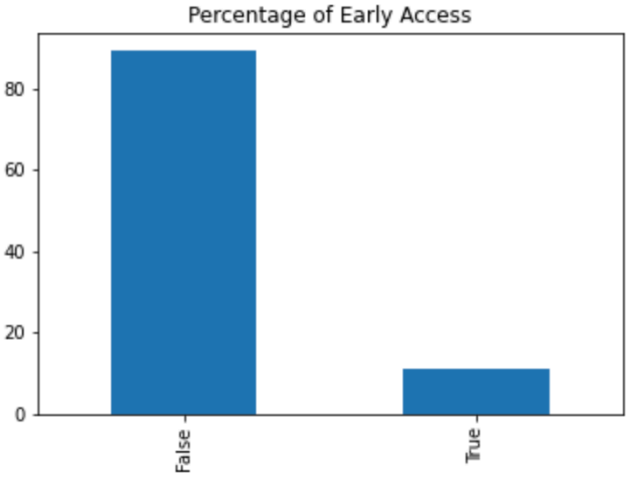


But here, it is hard to find correct nearest data set. Even if model will predict the proper cluster of the sample data, **it will be always questionable.**

Case 2:

When the the data is imbalanced.

As already I have discussed in the first game questions that 90% of the reviews was showing ‘not early access’.



So kNN will not work properly as **it will create huge cluster for the class ‘not early access’** and when a new data will look for nearest cluster, it will find only biggest cluster. Hence, predicting the class of a point always give the majority class of that huge cluster.

Appendix:

You can also view the code in c-lab directly using link: https://colab.research.google.com/drive/1uMHSgqMydSvXFMSPiyGpKs1t-gRCYLeW?usp=sharing

Code:

```
# -*- coding: utf-8 -*-
"""final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1uMHSgqMydSvXFMSPiyGpKs1t-gRCYLeW

### Libraries
"""

pip install json_lines

# Commented out IPython magic to ensure Python compatibility.
import json_lines
import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
import pandas
import pandas as pd
import numpy as np
from sklearn import metrics, svm
import seaborn as sn
from sklearn.linear_model import Lasso
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
# %matplotlib inline

"""### Data Processing and Cleaning """

X=[]; y=[]; z=[]
with open('reviews_111.jl', 'rb') as f:
  for item in json_lines.reader(f):
```

```
    X.append(item['text'])
    y.append(item['voted_up'])
    z.append(item['early_access'])

data = {'Review': X, 'Voted_Up': y, 'Early_Access':z}
df = pandas.DataFrame(data=data)
print(df)

def cleaning(text):
  text=text.lower()
  text=re.sub('\[.*?\],!',", text)
  text=re.sub('[%s]'% re.escape(string.punctuation),",text)
  text=re.sub('\w*\d\w*',",text)
  return text

cleaned = lambda x: cleaning(x)

df['Review']=pandas.DataFrame(df.Review.apply(cleaned))
X=np.array(df.Review)
y=np.array(df.Voted_Up)
z=np.array(df.Early_Access)
print(df)

"""### Data Visualisation """

round(df.Voted_Up.value_counts(normalize=True)*100,2).plot(kind='bar')
plt.title("Percentage of Voted Up")
plt.show()

round(df.Early_Access.value_counts(normalize=True)*100,2).plot(kind='bar')
plt.title("Percentage of Early Access")
plt.show()

cleaned_reviews=df.Review
voted_up=df.Voted_Up
early=df.Early_Access

re_train,re_test,vo_train,vo_test,ea_train,ea_test =
train_test_split(cleaned_reviews,voted_up,early,test_size=0.25,random_state=255)

print("re_train: ",len(re_train))
print("re_test: ",len(re_test))
print("vo_train: ",len(vo_train))
print("vo_test: ",len(vo_test))
print("ea_train: ",len(ea_train))
print("ea_test: ",len(ea_test))

"""### Logistic Regression With Cross Validation For Question 1.1 and 1.2"""
```

```
tvec= TfidfVectorizer()
clf1= LogisticRegression(C=10, max_iter=500)
logic_1= Pipeline([('vectorizer',tvec),('classifier',clf1)])
logic_1.fit(re_train,vo_train)

predictions= logic_1.predict(re_test)

confusion_matrix = pd.crosstab(vo_test,predictions, rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)

print("\nAccuracy: ", accuracy_score(vo_test,predictions))
print("Precision: ", precision_score(vo_test,predictions, average= 'weighted'))
print("Recall: ", recall_score(vo_test,predictions,average= 'weighted'))
print("F1 Score: ", f1_score(vo_test,predictions))
#print("Mean Square Error: ", mean_squared_error(vo_test,predictions))

sample=['worst game','happy with this game']
example=logic_1.predict(sample)

print("\nReview:'worst game'.  Vote Up Result:",example[0])
print("Review:'happy with this game'.  Vote Up Result:",example[1],'\n')
plt.show()

importance = clf1.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
  print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()

X=np.array(df.Review)
y=np.array(df.Voted_Up)
mean_array=[]; std_array=[]

Ci_range = [0.1, 0.5, 1, 5, 10, 50, 100]

for Ci in Ci_range:
  #model = LogisticRegression(C=Ci)
  clf1= LogisticRegression(C=Ci,max_iter=500)
  model= Pipeline([('vectorizer',tvec),('classifier',clf1)])
  small_array=[]

  kf = KFold(n_splits=10)
```

```
  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when C = "+ str(Ci)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(Ci_range,mean_array,yerr= std_array)
plt.xlabel('C value')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
plt.title('Accuracy vs C Value')
plt.show()

tvec= TfidfVectorizer()
clf1= LogisticRegression(C=10, max_iter=500)
logic_2= Pipeline([('vectorizer',tvec),('classifier',clf1)])
logic_2.fit(re_train,ea_train)

predictions= logic_2.predict(re_test)

confusion_matrix = pd.crosstab(ea_test,predictions, rownames=['Actual'], colnames=['Predicted'])

sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)

print("\nAccuracy: ", accuracy_score(ea_test,predictions))
print("Precision: ", precision_score(ea_test,predictions, average= 'weighted'))
print("Recall: ", recall_score(ea_test,predictions,average= 'weighted'))
print("F1 Score: ", f1_score(ea_test,predictions))

sample=['old game','new upcoming game']
example=logic_2.predict(sample)

print("\nReview:'old game'.  Early Access Result:",example[0])
print("Review:'new upcoming game'.  Early Access Result:",example[1],'\n')

plt.show()

X=np.array(df.Review)
y=np.array(df.Early_Access)
mean_array=[]; std_array=[]

Ci_range = [0.1, 0.5, 1, 5, 10, 50, 100]
```

```
for Ci in Ci_range:
  #model = LogisticRegression(C=Ci)
  clf1= LogisticRegression(C=Ci,max_iter=500)
  model= Pipeline([('vectorizer',tvec),('classifier',clf1)])
  small_array=[]

  kf = KFold(n_splits=10)

  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when C = "+ str(Ci)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(Ci_range,mean_array,yerr= std_array)
plt.xlabel('C value')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
plt.title('Accuracy vs C Value')
plt.show()

"""### KNN With Cross Validation For Question 1.1 and 1.2"""

tvec= TfidfVectorizer()
clf2= KNeighborsClassifier(n_neighbors=4)
kNeighborsClassifier_1=Pipeline([('vectorizer',tvec),('classifier',clf2)])
kNeighborsClassifier_1.fit(re_train,vo_train)
y_pred=kNeighborsClassifier_1.predict(re_test)
confusion_matrix = pd.crosstab(vo_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)
print('\nAccuracy: ',metrics.accuracy_score(vo_test, y_pred))
print("Precision: ", precision_score(vo_test, y_pred, average= 'weighted'))
print("Recall: ", recall_score(vo_test, y_pred,average= 'weighted'))
print("F1 Score: ", f1_score(vo_test, y_pred))

sample=['worst game','happy with this game']
example=kNeighborsClassifier_1.predict(sample)

print("\nReview:'worst game'.  Vote Up Result:",example[0])
print("Review:'happy with this game'.  Vote Up Result:",example[1],'\n')
```

```
plt.show()

X=np.array(df.Review)
y=np.array(df.Voted_Up)
mean_array=[]; std_array=[]

N = [2,3,4,6,8,10,20,50,100,200,500]

for n in N:

  clf2= KNeighborsClassifier(n_neighbors=n)
  model= Pipeline([('vectorizer',tvec),('classifier',clf2)])
  small_array=[]

  kf = KFold(n_splits=10)

  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when n = "+ str(n)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(N,mean_array,yerr= std_array)
plt.xlabel('K neighbors')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
plt.title('Accuracy vs K neighbours')
plt.show()

tvec= TfidfVectorizer()
clf2= KNeighborsClassifier(n_neighbors=4)
kNeighborsClassifier_2=Pipeline([('vectorizer',tvec),('classifier',clf2)])
kNeighborsClassifier_2.fit(re_train,ea_train)
y_pred=kNeighborsClassifier_2.predict(re_test)
confusion_matrix = pd.crosstab(ea_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)
print('\nAccuracy: ',accuracy_score(ea_test, y_pred))
print("Precision: ", precision_score(ea_test, y_pred, average= 'weighted'))
print("Recall: ", recall_score(ea_test, y_pred,average= 'weighted'))
print("F1 Score: ", f1_score(ea_test, y_pred))

sample=['old game','new upcoming game']
```

```
example=kNeighborsClassifier_2.predict(sample)

print("\nReview:'old game'.  Early Access Result:",example[0])
print("Review:'new upcoming game'.  Early Access Result:",example[1],'\n')
plt.show()

X=np.array(df.Review)
y=np.array(df.Early_Access)
mean_array=[]; std_array=[]

N = [2,4,6,8,10,20,50,100,200,500]

for n in N:

  clf2= KNeighborsClassifier(n_neighbors=n)
  model= Pipeline([('vectorizer',tvec),('classifier',clf2)])
  small_array=[]

  kf = KFold(n_splits=10)

  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when n = "+ str(n)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(N,mean_array,yerr= std_array)
plt.xlabel('K neighbors')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
plt.title('Accuracy vs K neighbors')
plt.show()

"""### SVM With Cross Validation For Question 1.1 and 1.2

"""

tvec= TfidfVectorizer()
clf3 = svm.SVC(kernel='linear',C = 1)
svm_1=Pipeline([('vectorizer',tvec),('classifier',clf3)])
svm_1.fit(re_train,vo_train)
y_pred=svm_1.predict(re_test)
confusion_matrix = pd.crosstab(vo_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
```

```
sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)
print('\nAccuracy: ', accuracy_score(vo_test, y_pred))
print("Precision: ", precision_score(vo_test, y_pred, average= 'weighted'))
print("Recall: ", recall_score(vo_test, y_pred,average= 'weighted'))
print("F1 Score: ", f1_score(vo_test, y_pred))

sample=['worst game','happy with this game']
example=svm_1.predict(sample)

print("\nReview:'worst game'.  Vote Up Result:",example[0])
print("Review:'happy with this game'.  Vote Up Result:",example[1],'\n')

plt.show()

X=np.array(df.Review)
y=np.array(df.Voted_Up)
mean_array=[]; std_array=[]

Ci_range = [0.1, 0.5, 1, 5, 10, 50, 100]

for Ci in Ci_range:
  #model = LogisticRegression(C=Ci)
  clf3 = svm.SVC(kernel='linear',C = Ci)
  model= Pipeline([('vectorizer',tvec),('classifier',clf3)])
  small_array=[]

  kf = KFold(n_splits=10)

  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when C = "+ str(Ci)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(Ci_range,mean_array,yerr= std_array)
plt.xlabel('C value')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
plt.title('Accuracy vs C Value')
plt.show()

tvec= TfidfVectorizer()
```

```
clf3 = svm.SVC(kernel='linear',C =10)
svm_2=Pipeline([('vectorizer',tvec),('classifier',clf3)])
svm_2.fit(re_train,ea_train)
y_pred=svm_2.predict(re_test)
confusion_matrix = pd.crosstab(ea_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print(confusion_matrix)
print('\nAccuracy: ',accuracy_score(ea_test, y_pred))
print("Precision: ", precision_score(ea_test, y_pred, average= 'weighted'))
print("Recall: ", recall_score(ea_test, y_pred,average= 'weighted'))
print("F1 Score: ", f1_score(ea_test, y_pred))

sample=['old game','new upcoming game']
example=svm_2.predict(sample)

print("\nReview:'old game'.  Early Access Result:",example[0])
print("Review:'new upcoming game'.  Early Access Result:",example[1],'\n')
plt.show()

X=np.array(df.Review)
y=np.array(df.Early_Access)
mean_array=[]; std_array=[]

Ci_range = [0.1, 0.5, 1, 5, 10, 50, 100]

for Ci in Ci_range:
  #model = LogisticRegression(C=Ci)
  clf3 = svm.SVC(kernel='linear',C = Ci)
  model= Pipeline([('vectorizer',tvec),('classifier',clf3)])
  small_array=[]

  kf = KFold(n_splits=10)

  for train, test in kf.split(X):
    model.fit(X[train], y[train])
    ypred = model.predict(X[test])
    scores = model.score(X[test],y[test])
    small_array.append(scores)

  mean_array.append(np.array(small_array).mean())
  std_array.append(np.array(small_array).std())
  print("when C = "+ str(Ci)+" Mean Accuracy = "+str(np.array(small_array).mean())+" Std =
"+str(np.array(small_array).std()))

plt.errorbar(Ci_range,mean_array,yerr= std_array)
plt.xlabel('C value')
plt.ylabel('Accuracy')
plt.legend(['Standard Deviation'])
```

```
plt.title('Accuracy vs C Value')
plt.show()

"""### ROC Curve For Question 1.1 and Question 1.2"""

def baseline_model(X):
  print(np.ones(X.shape[0]))
  return np.ones(X.shape[0])

baseline_y_pred = baseline_model(re_test)

from sklearn.metrics import roc_curve,auc

Logi_y_pred = logic_1.predict(re_test)
Knn_y_pred = kNeighborsClassifier_1.predict(re_test)
svm_y_pred = svm_1.predict(re_test)
baseline_y_pred = baseline_model(re_test)

confusion_matrix2 = confusion_matrix(vo_test,Knn_y_pred)
confusion_matrix3 = confusion_matrix(vo_test,baseline_y_pred)
confusion_matrix1 = confusion_matrix(vo_test,Logi_y_pred)
confusion_matrix4 = confusion_matrix(vo_test,svm_y_pred)

tn, fp, fn, tp = confusion_matrix3.ravel()
print('For Baseline Model: Number of True Negatives: ',tn,' False Positives: ',fp,' False Negatives:
',fn,' True Positives : ',tp)
tn1, fp1, fn1, tp1 = confusion_matrix1.ravel()
print('\nFor Logistic Regression: Number of True Negatives: ',tn1,' False Positives: ',fp1,' False
Negatives: ',fn1,' True Positives : ',tp1)
tn2, fp2, fn2, tp2 = confusion_matrix2.ravel()
print('\nFor Knn: Number of True Negatives: ',tn2,' False Positives: ',fp2,' False Negatives: ',fn2,'
True Positives : ',tp2)
tn2, fp2, fn2, tp2 = confusion_matrix4.ravel()
print('\nFor SVM: Number of True Negatives: ',tn2,' False Positives: ',fp2,' False Negatives: ',fn2,'
True Positives : ',tp2,'\n')


False_Positive_rate,True_Positive_rate,_ = roc_curve(vo_test,logic_1.decision_function(re_test))
False_Positive_rate2,True_Positive_rate2,_ =
roc_curve(vo_test,kNeighborsClassifier_1.predict_proba(re_test)[:,1])
False_Positive_rate3,True_Positive_rate3,_ = roc_curve(vo_test,baseline_y_pred)
False_Positive_rate4,True_Positive_rate4,_ = roc_curve(vo_test,svm_1.decision_function(re_test))

roc_auc = auc(False_Positive_rate, True_Positive_rate)
roc_auc2 = auc(False_Positive_rate2, True_Positive_rate2)
roc_auc3 = auc(False_Positive_rate3, True_Positive_rate3)
roc_auc4 = auc(False_Positive_rate4, True_Positive_rate4)
```

```
plt.plot(False_Positive_rate,True_Positive_rate,'r', label = 'Logistic AUC = %0.4f' % roc_auc)
plt.plot(False_Positive_rate2,True_Positive_rate2,'b', label = 'KNN AUC = %0.4f' % roc_auc2)
plt.plot(False_Positive_rate3,True_Positive_rate3,linestyle='--',label = 'Baseline AUC = %0.4f' %
roc_auc3)
plt.plot(False_Positive_rate4,True_Positive_rate4,'g', label = 'SVM AUC = %0.4f' % roc_auc4)
plt.legend(loc = 'lower right')
plt.rc('font', size=10); plt.rcParams['figure.constrained_layout.use'] = True
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
# plt.legend(['Logistic Regression','KNN Classifier', 'Baseline Model'])
plt.title('Roc Curve')

from sklearn.metrics import roc_curve,auc

Logi_y_pred = logic_2.predict(re_test)
Knn_y_pred = kNeighborsClassifier_2.predict(re_test)
svm_y_pred = svm_2.predict(re_test)
baseline_y_pred = baseline_model(re_test)

confusion_matrix2 = confusion_matrix(ea_test,Knn_y_pred)
confusion_matrix3 = confusion_matrix(ea_test,baseline_y_pred)
confusion_matrix1 = confusion_matrix(ea_test,Logi_y_pred)
confusion_matrix4 = confusion_matrix(ea_test,svm_y_pred)

tn, fp, fn, tp = confusion_matrix3.ravel()
print('For Baseline Model: Number of True Negatives: ',tn,' False Positives: ',fp,' False Negatives:
',fn,' True Positives : ',tp)
tn1, fp1, fn1, tp1 = confusion_matrix1.ravel()
print('\nFor Logistic Regression: Number of True Negatives: ',tn1,' False Positives: ',fp1,' False
Negatives: ',fn1,' True Positives : ',tp1)
tn2, fp2, fn2, tp2 = confusion_matrix2.ravel()
print('\nFor Knn: Number of True Negatives: ',tn2,' False Positives: ',fp2,' False Negatives: ',fn2,'
True Positives : ',tp2)
tn2, fp2, fn2, tp2 = confusion_matrix4.ravel()
print('\nFor SVM: Number of True Negatives: ',tn2,' False Positives: ',fp2,' False Negatives: ',fn2,'
True Positives : ',tp2,'\n')


False_Positive_rate,True_Positive_rate,_ = roc_curve(ea_test,logic_2.decision_function(re_test))
False_Positive_rate2,True_Positive_rate2,_ =
roc_curve(ea_test,kNeighborsClassifier_2.predict_proba(re_test)[:,1])
False_Positive_rate3,True_Positive_rate3,_ = roc_curve(ea_test,baseline_y_pred)
False_Positive_rate4,True_Positive_rate4,_ = roc_curve(ea_test,svm_2.decision_function(re_test))

roc_auc = auc(False_Positive_rate, True_Positive_rate)
roc_auc2 = auc(False_Positive_rate2, True_Positive_rate2)
roc_auc3 = auc(False_Positive_rate3, True_Positive_rate3)
roc_auc4 = auc(False_Positive_rate4, True_Positive_rate4)
```

```
plt.plot(False_Positive_rate,True_Positive_rate,'r', label = 'Logistic AUC = %0.4f' % roc_auc)
plt.plot(False_Positive_rate2,True_Positive_rate2,'b', label = 'KNN AUC = %0.4f' % roc_auc2)
plt.plot(False_Positive_rate3,True_Positive_rate3,linestyle='--',label = 'Baseline AUC = %0.4f' %
roc_auc3)
plt.plot(False_Positive_rate4,True_Positive_rate4,'g', label = 'SVM AUC = %0.4f' % roc_auc4)
plt.legend(loc = 'lower right')
plt.rc('font', size=10); plt.rcParams['figure.constrained_layout.use'] = True
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
# plt.legend(['Logistic Regression','KNN Classifier', 'Baseline Model'])
plt.title('Roc Curve')



#END
```