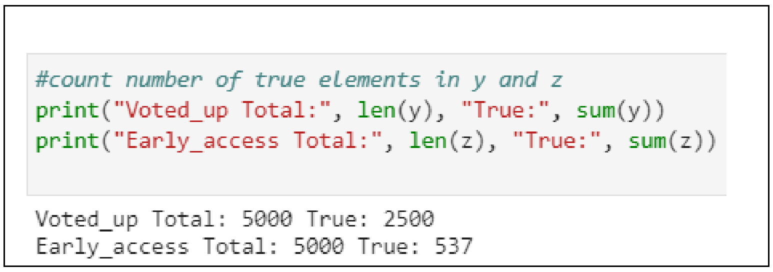
Machine Learning Final Assignment

1. Report

The machine learning algorithms that were applied to the dataset (renamed as '*reviews.jl'*) were logistic regression and k-nearest neighbours (kNN). The classification performances of these algorithms were also evaluated using a confusion matrix and a classification report. Non-ascii characters were removed from the 'text' dataset.
Out of 5000, there were 2500 who voted up recommending the game, and 537 said that they had the early access or the beta version of the game.

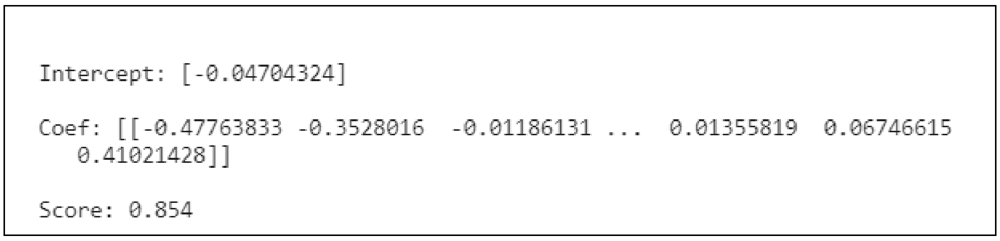


```
#count number of true elements in y and z
print("Voted_up Total:", len(y), "True:", sum(y))
print("Early_access Total:", len(z), "True:", sum(z))

Voted_up Total: 5000 True: 2500
Early_access Total: 5000 True: 537
```

(i) Predict Review Polarity:

Logistic Regression

Training a logistic regression on the 'text' and 'voted_up' data, the following results were extracted:

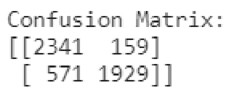


```
Intercept: [-0.04704324]

Coef: [[-0.47763833 -0.3528016  -0.01186131 ...  0.01355819  0.06746615
   0.41021428]]

Score: 0.854
```

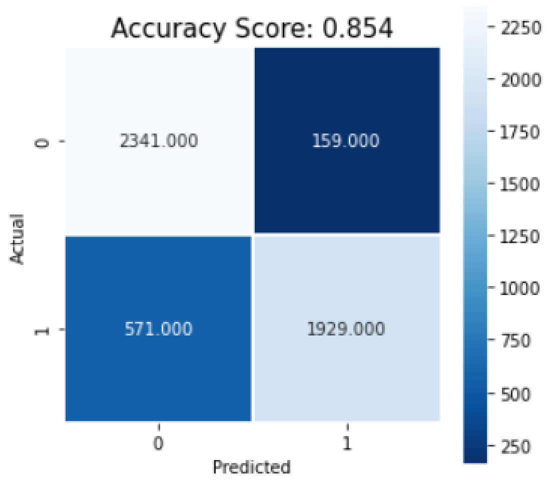
The score function was used to find the accuracy of the model and we can see that it is 85.4% accurate.

Confusion Matrix:

To evaluate the performance of the logistic regression model, a confusion matrix using the metrics function from the sklearn library was found and this was plotted using the seaborn library:



```
Confusion Matrix:
[[2341  159]
 [ 571 1929]]
```



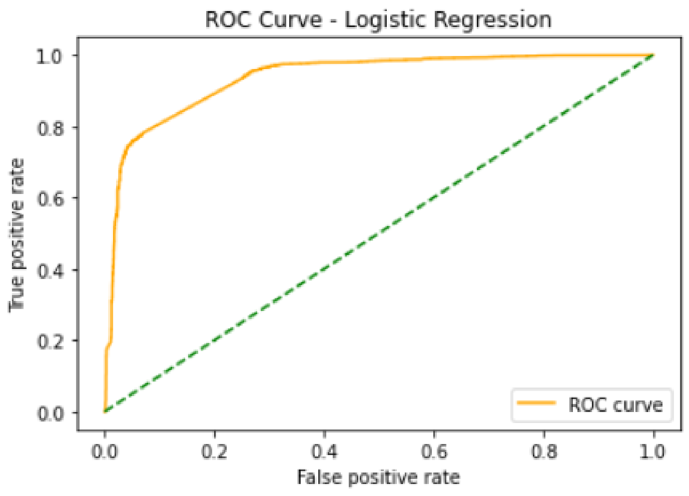
Classification Report:

The classification_report function from the sklearn.metrics library was used to visualise the precision, recall, f1-scores, support scores and the accuracy for the logistic regression model. From the graph below, *False* had a slightly higher f1-score than *True* which means that more reviews were not in favour of recommending the game as the f1-score takes both false positives and false negatives into account.

```
              precision    recall  f1-score   support

       False       0.80      0.94      0.87      2500
        True       0.92      0.77      0.84      2500

    accuracy                           0.85      5000
   macro avg       0.86      0.85      0.85      5000
weighted avg       0.86      0.85      0.85      5000
```

ROC Curve:

Using the roc_curve function from sklearn.metrics library, an ROC on the logistic regression model is plotted.



Using the roc_auc_score function, the AUC score for this model was:

```
AUC Score: 0.93632064
```

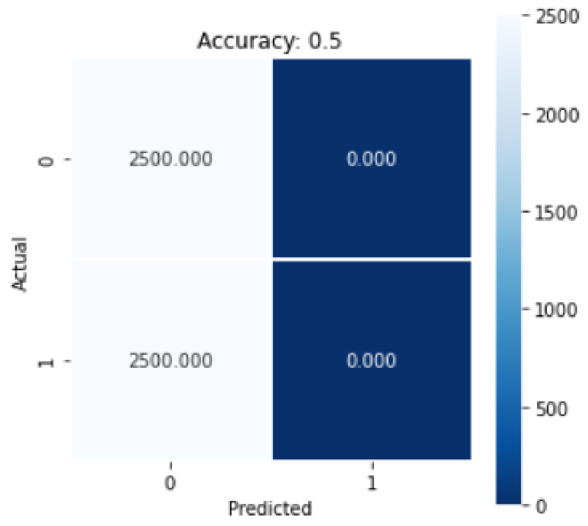
This indicates that there is as 93% chance that the model will be able to distinguish between positive and negative class. The AUC score is close to 1 which is an ideal classifier.

Comparison against the baseline:

The DummyClassifier function from the sklearn.dummy library was used to compare the logistic regression model to a baseline classifier model.

The baseline model's confusion matrix was as follows:

```
Confusion Matrix:
 [[2500    0]
 [2500    0]]
```



The classification report for the baseline model was:

```
Classification Report:
              precision    recall  f1-score   support

       False       0.50      1.00      0.67      2500
        True       0.00      0.00      0.00      2500

    accuracy                           0.50      5000
   macro avg       0.25      0.50      0.33      5000
weighted avg       0.25      0.50      0.33      5000
```

Ease of Comparison:

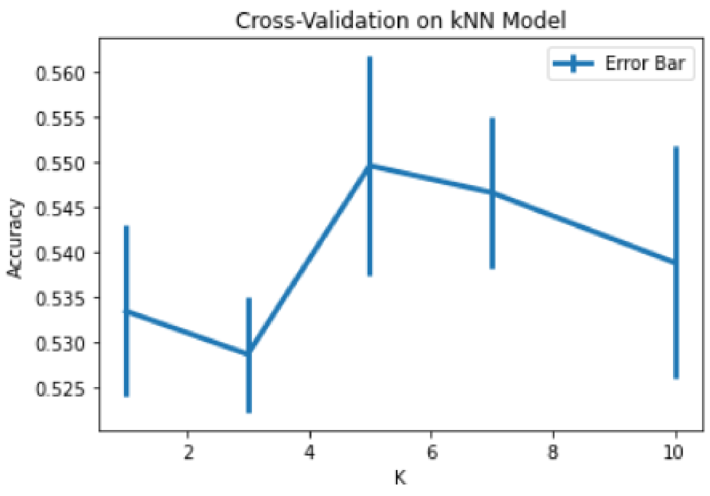
For ease of comparison, I displayed the accuracy and f1-scores of the logistic regression model and the baseline model in a table below:

	Logistic Regression	*Baseline Model*
Accuracy	**0.85**	0.50
F1-score (False)	**0.87**	0.67

Comparing against the baseline model, the logistic regression model was much better as it had a higher accuracy score than the baseline model and its f1-score was also higher. Thus, the logistic regression model is the better approach in this case.

k-Nearest Neighbours (kNN)

Before training the 'text' and 'voted_up' data using a kNN classifier model, cross-validation was used to select the best value of k. With a range of values of 1, 3, 5, 7 and 10, the cross-validation plot was as follows:

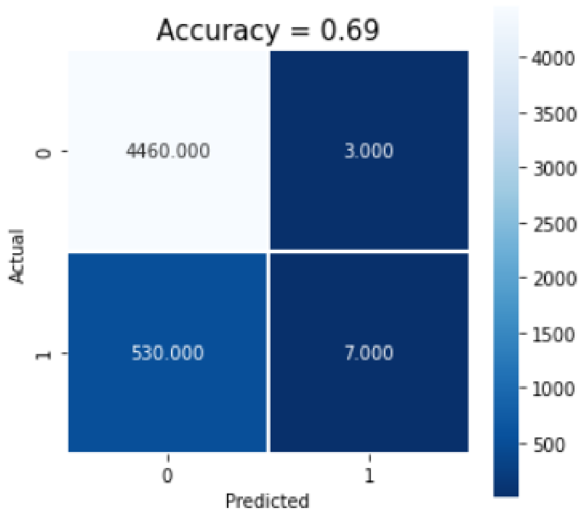


From the error bar, a value of 5 was chosen for the kNN model as it had a higher accuracy score compared to the other values.

Confusion Matrix:

The performance on the kNN model was evaluated once again using a confusion matrix and plotted using the seaborn library:

```
Confusion Matrix:
[[1847  653]
 [ 917 1583]]
```



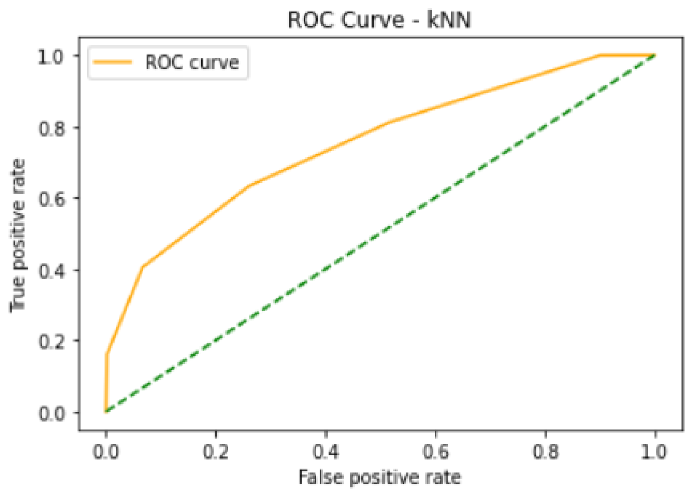
Classification Report:

The classification report for the kNN model is shown below. Again, the f1-score on *False* is 0.03 higher than *True,* indicating that more reviews were not in favour of recommending the game.

```
              precision    recall  f1-score   support

       False       0.67      0.74      0.70      2500
        True       0.71      0.63      0.67      2500

    accuracy                           0.69      5000
   macro avg       0.69      0.69      0.69      5000
weighted avg       0.69      0.69      0.69      5000
```

ROC Curve:

The ROC Curve on the kNN model is plotted below.



The AUC score on the kNN model was:

```
AUC Score: 0.75105672
```

This suggests that there is a 75.1% chance that the kNN model will be able to distinguish between positive and negative classes. The AUC score on the kNN model is much lower than the AUC of the logistic regression model (75.1% vs 93.6%). Though the AUC score on the kNN model is not very close to 1, it is still good enough as it is bigger than 0.5.

Comparison against the baseline:

For ease of comparison again, I displayed the accuracy and f1-score on the kNN and baseline model below:

Ease of Comparison:

	kNN	*Baseline Model*
Accuracy	**0.69**	0.50
F1-score (False)	**0.70**	0.67

From the table, the kNN model again displayed a much higher accuracy and f1-score indicating that the kNN model is much more attractive than the baseline model.

(ii) Predict whether review is for an early access version of a game or not

Logistic Regression

Similar from part *(i)*, the following results were found when a logistic regression model was trained on the *'text'* data and the *'early_access'* data.

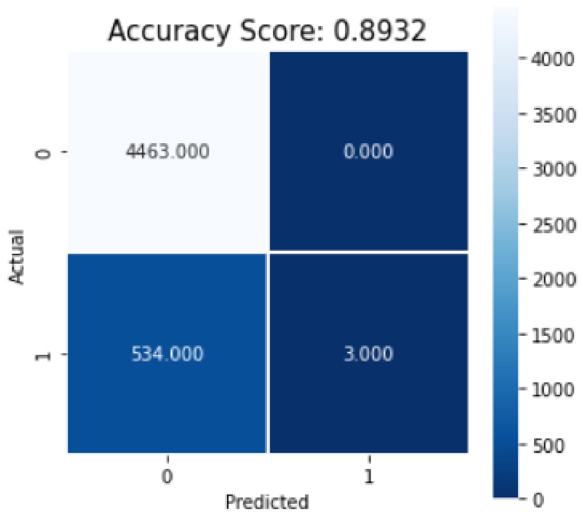
```
Intercept: [-2.22475922]

Coef: [[ 0.21193807 -0.12174736 -0.00300896 ... -0.0021769  -0.01431211
  -0.08992057]]

Score: 0.8932
```

The score function on this data generated an accuracy of 89.32%.

Confusion Matrix:

The evaluation of this model's performance is once again displayed in a confusion matrix plot using the seaborn library.

```
Confusion Matrix:
[[4463    0]
 [ 534    3]]
```



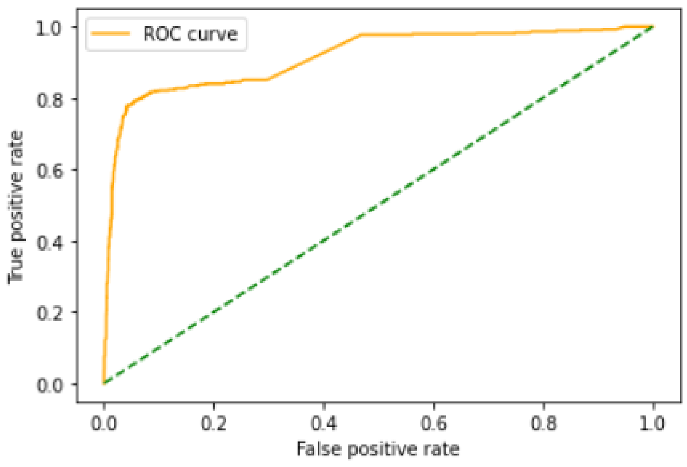
Classification Report:

The classification report of this model is displayed below, showing a 94% f1-score on the *False* data against a 1% f1-score on *True.*

```
              precision    recall  f1-score   support

       False       0.89      1.00      0.94      4463
        True       1.00      0.01      0.01       537

    accuracy                           0.89      5000
   macro avg       0.95      0.50      0.48      5000
weighted avg       0.90      0.89      0.84      5000
```

ROC Curve:

The ROC curve on this model was:



The AUC score was:

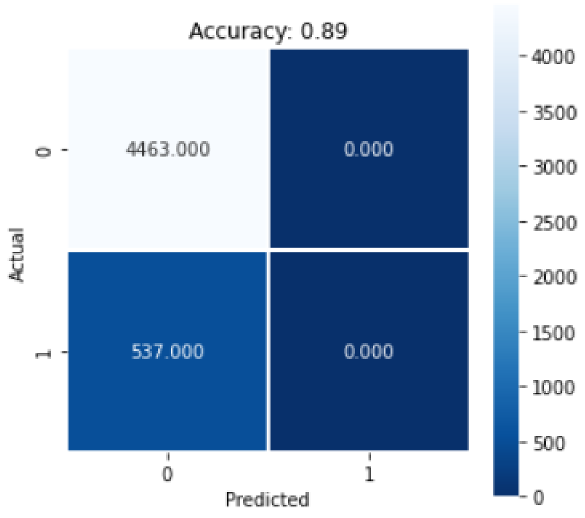
```
AUC Score: 0.9151325339612147
```

From this AUC score, there is as 91.5% chance that the model will be able to distinguish between positive and negative class. The score is also close to 1 which is very ideal.

Comparison against the baseline:

The baseline model's confusion matrix on this data was as follows:

```
Confusion Matrix:
 [[4463    0]
 [ 537    0]]
```



The classification report for the baseline model was:

```
Classification Report:
               precision    recall  f1-score   support

       False       0.89      1.00      0.94      4463
        True       0.00      0.00      0.00       537

    accuracy                           0.89      5000
   macro avg       0.45      0.50      0.47      5000
weighted avg       0.80      0.89      0.84      5000
```

Ease of Comparison:

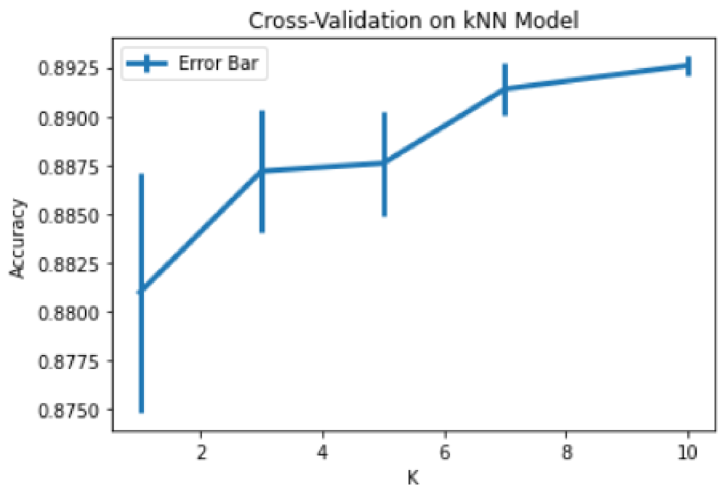
	Logistic Regression	*Baseline Model*
Accuracy	**0.8932**	0.89
F1-score (False)	0.94	0.94

Comparing against the baseline model and the logistic regression model, the models were almost close to each other. The f1-scores *(False)* on both the logistic regression model and the baseline model were the same. However, the accuracy score on the logistic regression model was 0.0032 bigger than the baseline model's accuracy score. Therefore, the logistic regression model is more attractive in this case.

---

k-Nearest Neighbours (kNN)

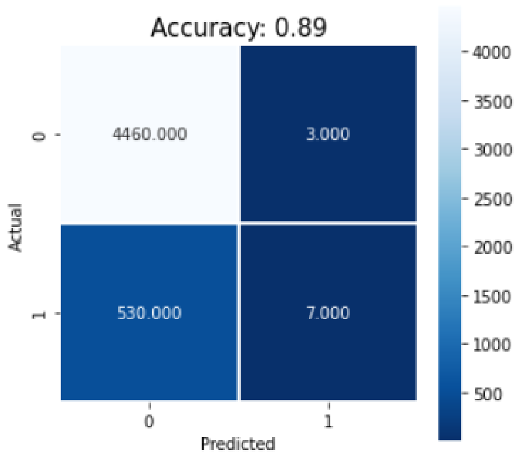
From the cross-validation plot, a value of 7 was chosen this time instead of 5 for the kNN model despite having a low accuracy score than 10. However, it had a higher accuracy score compared to the other values.



Confusion Matrix:

The confusion matrix for the kNN model are displayed below:

```
Confusion Matrix:
[[4460    3]
 [ 530    7]]
```



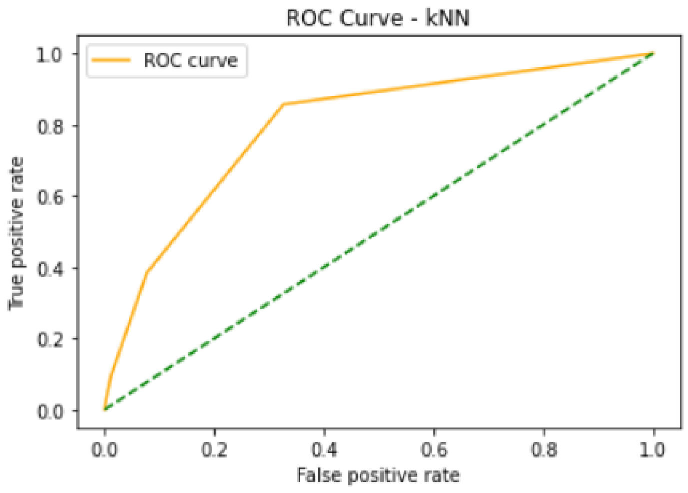
Classification Report:

The classification report of the kNN model shows a 0.94 f1-score on *False* against a 0.03 f1-score on *True* and a 0.89 accuracy score.

```
              precision    recall  f1-score   support

       False       0.89      1.00      0.94      4463
        True       0.70      0.01      0.03       537

    accuracy                           0.89      5000
   macro avg       0.80      0.51      0.48      5000
weighted avg       0.87      0.89      0.85      5000
```

ROC Curve:

The ROC curve on the kNN model is shown below:



The AUC Score on the model was:

```
AUC Score: 0.7963837570322674
```

The AUC score on the kNN model (79.6%) is much lower than the Logistic Regression model's AUC score (91.5%).

Comparison against the baseline

Ease of Comparison:

	kNN	*Baseline Model*
Accuracy	0.89	0.89
F1-score (False)	0.94	0.94

From the table above, the kNN model and the baseline model had a similar accuracy and f1-scores. Either of these models can be used to predict whether the review is for a beta version of the game.

Conclusion

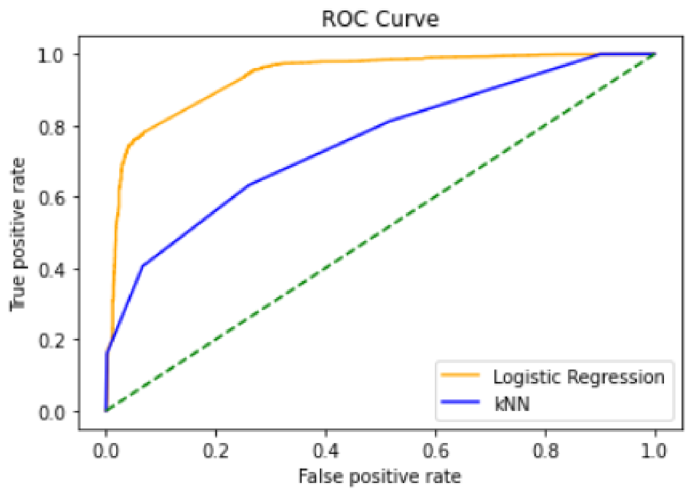
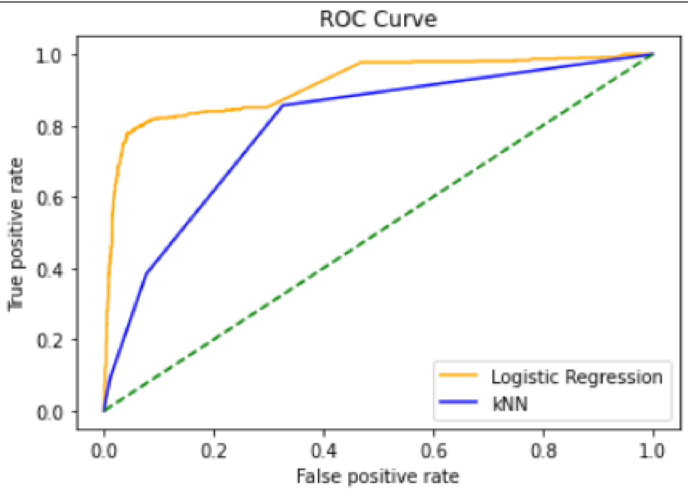
Comparison: Review Polarity

	Logistic Regression	*kNN*	*Baseline*
Accuracy	**0.854**	0.69	0.50
F1-score (False)	**0.87**	0.70	0.67
F1-score (True)	**0.84**	0.67	0.0

Comparison: Beta version of the game

	Logistic Regression	*kNN*	*Baseline*
Accuracy	**0.8932**	0.89	0.89
F1-score (False)	**0.94**	0.94	0.94
F1-score (True)	0.01	**0.03**	0.00

In conclusion, the 'text' review can be used to predict the review polarity and whether the review is for a beta version of the game. Whether the reviewer recommends the game is a close prediction but from the models above, it seems that the predictions show that the reviewer does not recommend the game in the future (F1-score False 0.87 vs F1-score True 0.84). From the models above, they all agree that the majority of the review is not for a beta version of the game.
From the comparison tables above, the logistic regression model seems to be the most attractive approach compared to the kNN model and baseline model as it has the highest accuracy score and f-1 scores compared to the other models. The logistic regression model should be used to predict the review polarity of the game as well as predicting whether the review is for a beta version of the game.

Review Polarity	**Beta version of the game**
<u>Logistic Regression vs kNN</u>	*<u>Logistic Regression vs kNN</u>*
	

Furthermore, the ROC curve comparisons from the table above show that the logistic regression model is a better approach than the kNN model.

2. Questions:

(i) *Underfitting:* Underfitting occurs when the predictions become poor because the model is too simple. It will have poor performance on the training data. E.g. use of q = 1 (purely linear model) when data is quadratic.

Overfitting: Overfitting occurs when the predictions also become poor because the model is too complicated. E.g. use of polynomial features with q = 6 when data is quadratic.

(ii) Pseudo-code implementing k-fold cross-validation:

#example: k-fold of 5

```
Split the training data into 5 parts
For train, test in kfold.split(X):
    Train model on training data
    Predict on test data
    Calculate train and test scores
    Calculate mean squared error
    Calculate standard error
    End for loop
Plot error bar plot
```

(iii) K-fold cross-validation provides a way to select a model hyperparameter 'C' to strike a balance between overfitting and underfitting to ensure that the model is not too simple or too complicated. Also, k-fold cross-validation is an example of a resampling technique to estimate the accuracy of a model that can be used to prevent overfitting and underfitting. K-fold cross-validation allows for training and testing the model k-times on the training data. After the model hyperparameters are tuned, the model can be evaluated to identify how the model might perform on unseen data.

(iv)

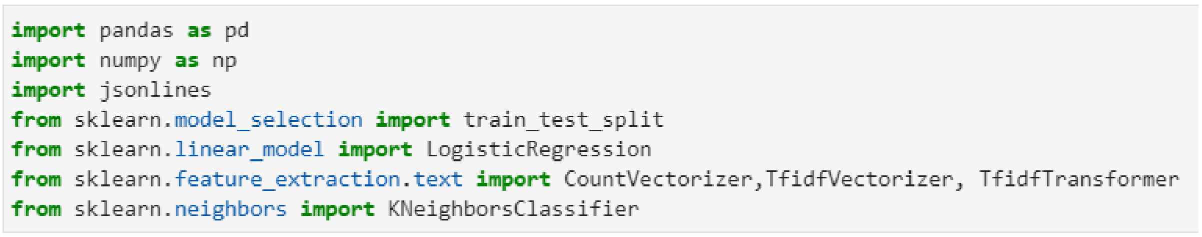
Logistic Regression (Pros)	**kNN (Cons)**
• can derive confidence level about its prediction	• can only output the labels about its prediction
• faster than kNN, efficient computation as it is not a resource hungry model	• slower than logistic regression especially when dealing with large datasets
• scales well with large datasets	• require high memory, need to store all the training data which can be computationally expensive

Logistic Regression (Cons)	**kNN (Pros)**
• requires some training on the dataset	• does not require training on the dataset
• cannot be used for regression problems	• can be used for both classification and regression problems
• only supports linear solutions	• supports non-linear solutions

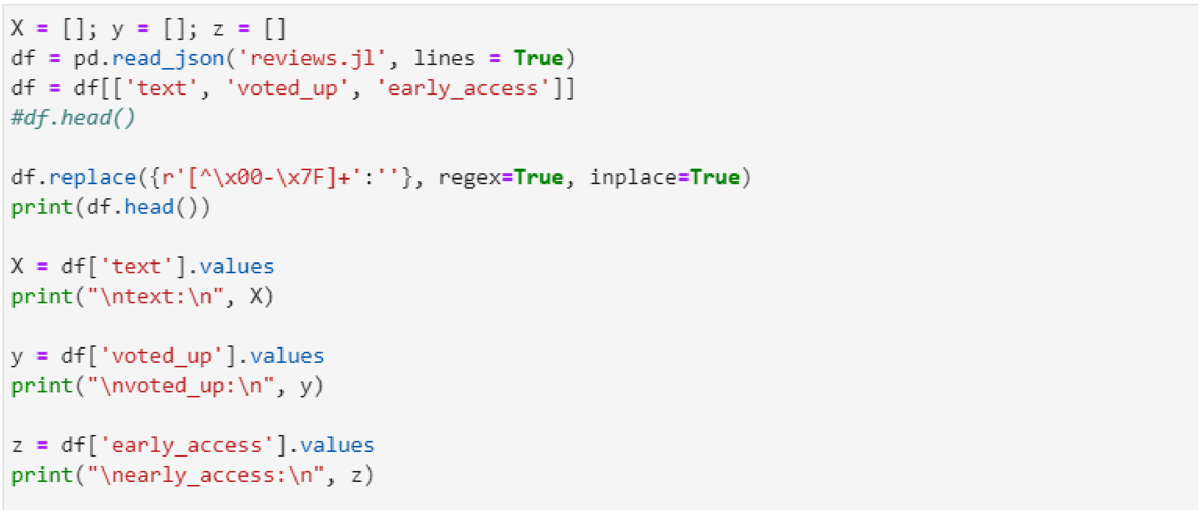
(v) An example of a situation when a kNN classifier would give inaccurate predictions is when the value of *K* is decreased. As *K* decreases, the predictions become less stable and inaccurate. For example, if the *K* value is decreased to 1 and if there are many blue points and 1 orange point being the nearest neighbour surrounding a query point, kNN would incorrectly predict that the query point is orange when in fact it is blue.

Appendix

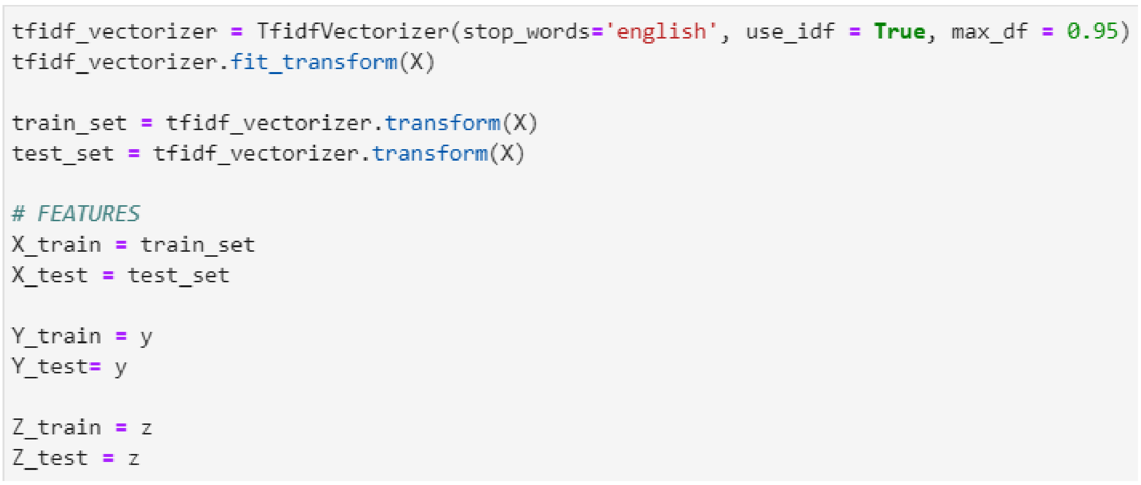
Logistic Regression – (i) Review Polarity



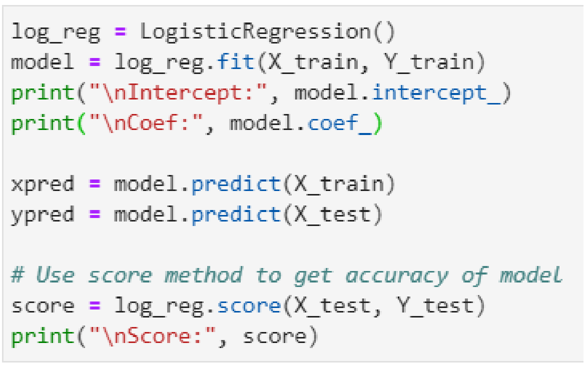
```
import pandas as pd
import numpy as np
import jsonlines
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer, TfidfTransformer
from sklearn.neighbors import KNeighborsClassifier
```



```
X = []; y = []; z = []
df = pd.read_json('reviews.jl', lines = True)
df = df[['text', 'voted_up', 'early_access']]
#df.head()

df.replace({r'[^\x00-\x7F]+':''}, regex=True, inplace=True)
print(df.head())

X = df['text'].values
print("\ntext:\n", X)

y = df['voted_up'].values
print("\nvoted_up:\n", y)

z = df['early_access'].values
print("\nearly_access:\n", z)
```



```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', use_idf = True, max_df = 0.95)
tfidf_vectorizer.fit_transform(X)

train_set = tfidf_vectorizer.transform(X)
test_set = tfidf_vectorizer.transform(X)

# FEATURES
X_train = train_set
X_test = test_set

Y_train = y
Y_test= y

Z_train = z
Z_test = z
```



```
log_reg = LogisticRegression()
model = log_reg.fit(X_train, Y_train)
print("\nIntercept:", model.intercept_)
print("\nCoef:", model.coef_)

xpred = model.predict(X_train)
ypred = model.predict(X_test)

# Use score method to get accuracy of model
score = log_reg.score(X_test, Y_test)
print("\nScore:", score)
```

Confusion Matrix

```
from sklearn import metrics
cm = metrics.confusion_matrix(Y_test, ypred)
print("Confusion Matrix:")
print(cm)
```

Confusion Matrix using seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15)
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, ypred))
```

ROC Curve

```
from sklearn.metrics import roc_curve
preds = log_reg.predict_proba(X_test)
print(model.classes_)
fpr, tpr, _ = roc_curve(Y_test, preds[:,1])
plt.plot(fpr, tpr)

auc_score = metrics.roc_auc_score(Y_test, preds[:,1])
print("AUC Score:", auc_score)
```

```
fpr, tpr, _ = roc_curve(Y_test, log_reg.decision_function(X_test))
plt.plot(fpr, tpr, color='orange')
plt.legend(['ROC curve','Logistic Regression'])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.title('ROC Curve - Logistic Regression')
plt.show()
```

Baseline

```
from sklearn import metrics
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy = "most_frequent").fit(X_train, Y_train)
ydummy = dummy.predict(X_test)

print("Classification Report:\n", classification_report(Y_test, ydummy))
dummycm = metrics.confusion_matrix(Y_test, ydummy)
print("Confusion Matrix:\n", dummycm)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(dummycm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual');
plt.xlabel('Predicted')
plt.title("Accuracy: 0.5")
plt.show()
```

<u>Logistic Regression – (ii) Beta version of the game</u>

```
log_reg = LogisticRegression()
model = log_reg.fit(X_train, Z_train)
print("\nIntercept:", model.intercept_)
print("\nCoef:", model.coef_)

xpred = model.predict(X_train)
zpred = model.predict(X_test)

# Use score method to get accuracy of model
score = log_reg.score(X_test, Z_test)
print("\nScore:", score)
```

Confusion Matrix

```
from sklearn import metrics
cm = metrics.confusion_matrix(Z_test, zpred)
print("Confusion Matrix:")
print(cm)
```

Confusion Matrix using seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15)
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(Z_test, zpred))
```

ROC Curve

```
from sklearn.metrics import roc_curve
preds = log_reg.predict_proba(X_test)
print(model.classes_)
fpr, tpr, _ = roc_curve(Z_test, preds[:,1])
plt.plot(fpr, tpr)

auc_score = metrics.roc_auc_score(Z_test, preds[:,1])
print("AUC Score:", auc_score)
```

```
fpr, tpr, _ = roc_curve(Z_test, log_reg.decision_function(X_test))
plt.plot(fpr, tpr, color='orange')
plt.legend(['ROC curve','Logistic Regression'])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.show()
```

Baseline

```
from sklearn import metrics
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy = "most_frequent").fit(X_train, Z_train)
zdummy = dummy.predict(X_test)

print("Classification Report:\n", classification_report(Z_test, zdummy))
dummycm = metrics.confusion_matrix(Z_test, zdummy)
print("Confusion Matrix:\n", dummycm)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(dummycm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Accuracy: 0.89')
plt.show()
```

k-Nearest Neighbours – (i) Review Polarity

```
import pandas as pd
import numpy as np
import jsonlines
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer, TfidfTransformer
from sklearn.neighbors import KNeighborsClassifier
```

```
X = []; y = []; z = []
df = pd.read_json('reviews.jl', lines = True)

#remove non-ascii characters
df.replace({r'[^\x00-\x7F]+':''}, regex=True, inplace=True)
print(df.head())

X = df['text'].values
print("text:\n", X)

y = df['voted_up'].values
print("text:\n", y)

z = df['early_access'].values
print("text:\n", z)
```

```
tfidf_vectorizer = TfidfVectorizer(stop_words='english', use_idf = True, max_df = 0.95)
tfidf_vectorizer.fit_transform(X)

train_set = tfidf_vectorizer.transform(X)
test_set = tfidf_vectorizer.transform(X)

# GET FEATURES
X_train = train_set
X_test = test_set

Y_train = y
Y_test= y

Z_train = z
Z_test = z
```

Cross-Validation

```
import matplotlib.pyplot as plt

mean_error=[]; std_error=[]
k_range = [1,3,5,7,10]
for Ki in k_range:
    print("K %d\n"%Ki)
    from sklearn.neighbors import KNeighborsClassifier
    model = KNeighborsClassifier(n_neighbors = Ki, weights = 'uniform')
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(model, X_train, Y_train, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())

plt.errorbar(k_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('K'); plt.ylabel('Accuracy')
plt.title('Cross-Validation on kNN Model')
plt.legend(['Error Bar'])
plt.show()
```

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
model = KNeighborsClassifier(n_neighbors = 5, weights = 'uniform').fit(X_train, Y_train)
xpred = model.predict(X_train)
ypred = model.predict(X_test)
#plt.scatter(X_train, Y_train, color = 'red', marker = '+')
plt.plot(xpred, ypred, color = 'blue')
plt.xlabel("input x"); plt.ylabel("output y"); plt.legend(["predict", "train"])
plt.show()
```

```
from sklearn.metrics import roc_curve
preds = model.predict_proba(X_test)
print(model.classes_)
fpr, tpr, _ = roc_curve(Y_test, preds[:,1])
plt.plot(fpr, tpr)

from sklearn import metrics
auc_score = metrics.roc_auc_score(Y_test, preds[:,1])
print("AUC Score:", auc_score)
```

```
fpr, tpr, _ = roc_curve(Y_test, preds[:,1])
plt.plot(fpr, tpr, color='orange')
plt.legend(['ROC curve','Logistic Regression'])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.title('ROC Curve - kNN')
plt.show()
```

```
from sklearn import metrics
cm = metrics.confusion_matrix(Y_test, ypred)
print("Confusion Matrix:")
print(cm)

metrics.accuracy_score(Y_test, ypred)
```

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Accuracy = 0.69', size = 15)
```

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, ypred))
```

Baseline

```
from sklearn import metrics
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy = "most_frequent").fit(X_train, Y_train)
ydummy = dummy.predict(X_test)
dummycm = metrics.confusion_matrix(Y_test, ydummy)
print(dummycm)

print(classification_report(Y_test, ydummy))

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

k-Nearest Neighbours – (ii) Beta version of the game

Cross-Validation

```
import matplotlib.pyplot as plt

mean_error=[]; std_error=[]
k_range = [1,3,5,7,10]
for Ki in k_range:
    print("K %d\n"%Ki)
    from sklearn.neighbors import KNeighborsClassifier
    model = KNeighborsClassifier(n_neighbors = Ki, weights = 'uniform')
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(model, X_train, Z_train, cv=5, scoring='accuracy')
    mean_error.append(np.array(scores).mean())
    std_error.append(np.array(scores).std())

plt.errorbar(k_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('K'); plt.ylabel('Accuracy')
plt.title('Cross-Validation on kNN Model')
plt.legend(['Error Bar'])
plt.show()
```

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
model = KNeighborsClassifier(n_neighbors = 7, weights = 'uniform').fit(X_train, Z_train)
xpred = model.predict(X_train)
zpred = model.predict(X_test)
#plt.scatter(X_train, Y_train, color = 'red', marker = '+')
plt.plot(xpred, zpred, color = 'blue')
plt.xlabel("input x"); plt.ylabel("output z"); plt.legend(["predict", "train"])
plt.show()
```

ROC Curve

```
from sklearn.metrics import roc_curve
zpreds = model.predict_proba(X_test)
print(model.classes_)
fpr, tpr, _ = roc_curve(Z_test, zpreds[:,1])
plt.plot(fpr, tpr)

from sklearn import metrics
auc_score = metrics.roc_auc_score(Z_test, zpreds[:,1])
print("AUC Score:", auc_score)
```

```
fpr, tpr, _ = roc_curve(Z_test, zpreds[:,1])
plt.plot(fpr, tpr, color='orange')
plt.legend(['ROC curve','kNN'])
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green', linestyle='--')
plt.title('ROC Curve - kNN')
plt.show()
```

Confusion Matrix

```
from sklearn import metrics
cm = metrics.confusion_matrix(Z_test, zpred)
print("Confusion Matrix:")
print(cm)
```

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Accuracy: 0.89', size = 15)
```

Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(Z_test, zpred))
```

Baseline

```
from sklearn import metrics
from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy = "most_frequent").fit(X_train, Z_train)
zdummy = dummy.predict(X_test)
dummycm = metrics.confusion_matrix(Z_test, zdummy)
print("Confusion Matrix:\n", dummycm)

print("\nClassfication Report:\n", classification_report(Z_test, zdummy))

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(5,5))
sns.heatmap(cm, annot=True, fmt="0.3f", linewidths=.5, square = True, cmap = 'Blues_r')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Accuracy = 0.89')
plt.show()
```