In [ ]:
```
pip install numpy
```

In [ ]:
```
pip install scipy==1.6.3 --user
```

In [ ]:
```
pip install scikit-learn
```

In [ ]:
```
pip install tensorflow==2.12 --user
```

In [ ]:
```
pip install --upgrade tensorflow
```

In [ ]:
```
import sys
!{sys.executable} -m pip install seaborn
```

In [ ]:
```
pip check
```

Random Forest Review Classifier

Introduction

Text analysis is a key area of interest, hence the selection of this type of analysis. The application of algorithms and AI to text analytics has the potential to massively improve the accuracy of the analysis, as it can study not just each individual word and its associations (single-token analysis), but can establish the sequence of the words, and how that is likely to impact the sentiment of the whole sentence. This offers the potential for much greater insight and accuracy when analysing texts. A good first step in developing text analysis tools and algorithms, is to create a sentiment identifier. The objective of this project is to create a random forest classifier that can read and analyse text reviews and establish whether each review is either positive or negative - this should be more accurate than the baseline for the data. For example, if a dataset has an equal number of positive and negative reviews, then the baseline accuracy would be 50%, as assigning all reviews as positive would be right 50% of the time. A random forest classifier was chosen for this project as it is one of the simpler algorithms to build and test. Therefore, one of the goals of this project is to establish whether implementing a simple model can be substantially more effective than the baseline, or if a more complex model - such as an artificial neural network - would be required. It is important to clarify that this project will test the effectiveness and accuracy of a random forest classifier, to establish if it sufficient, and not to test multiple different styles of algorithms. This is due to hardware and time limitations of the project. It is believed a random forest classifier will be reasonably effective due to its ability to combine the predictions of many differing decision trees, offering a greater level of accuracy, while reducing overfitting. Additionally, the ability to determine feature importance should help in determining which sequences of words are more likely to be associated with a positive or negative sentiment. This algorithm will be tested on two datasets. The first dataset is from Steam, and contains over 400,000 text reviews, organised by game title, and with information on whether the review recommends the game or not [1] - this provides all the information we require to test our algorithm. The second dataset [2] - although much shorter, at just over 17,000 reviews - also contains a list of Steam reviews, organised by game title, with a text review and a positive or negative recommendation value. This should mean the algorithm can work with both datasets without significant adjustment, as they both contain the core information required, even though they are of differing lengths and contain differing game titles.

In [1]:
```
#import the majority of packages we will be using
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.model_selection import train_test_split
```

Dataset One

In [2]:
```
reviews = pd.read_csv('m:\\pc\\desktop\\AI\\steam_reviews.csv')
reviews.head()
#Load in the dataset and ensure it is working
```

Out[2]:

	date_posted	funny	helpful	hour_played	is_early_access_review	recommendation	review	title
0	2019-02-10	2	4	578	False	Recommended	&gt Played as German Reich&gt Declare war on B...	Expansion - Hearts of Iron IV: Man the Guns
1	2019-02-10	0	0	184	False	Recommended	yes.	Expansion - Hearts of Iron IV: Man the Guns
2	2019-02-07	0	0	892	False	Recommended	Very good game although a bit overpriced in my...	Expansion - Hearts of Iron IV: Man the Guns
3	2018-06-14	126	1086	676	False	Recommended	Out of all the reviews I wrote This one is pro...	Dead by Daylight
4	2017-06-20	85	2139	612	False	Recommended	Disclaimer I survivor main. I play games for f...	Dead by Daylight

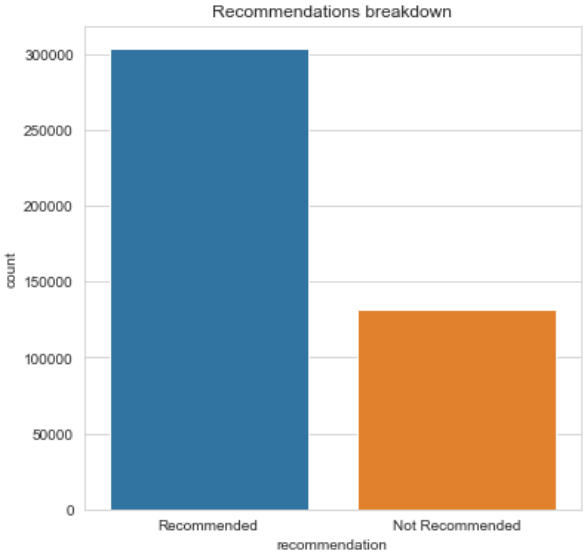
Exploritory Data Analysis

Getting to know more about the dataset is important before attempting any analysis. It ensures there are no obvious errors, or things we need to change before our analysis, and helps establish what needs to be done during data cleaning and pre-processing.

In [3]:
```
#How many reviews are we working with in this dataset?
print("There are", len(reviews["review"]), "reviews")
```

```
There are 434891 reviews
```

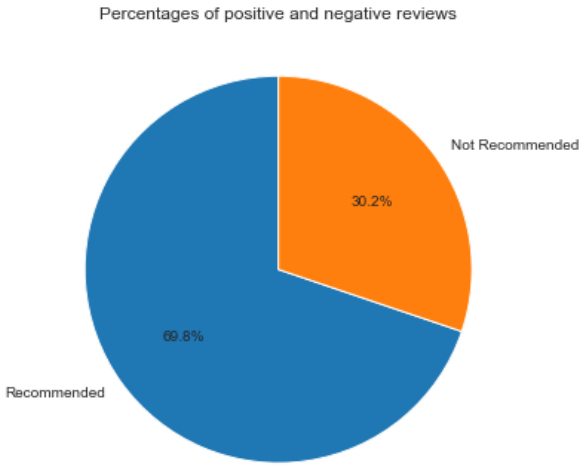
In [4]:
```
#Lets visualise how these reviews are broken down
sns.set_style("whitegrid")
plt.figure(figsize=(6,6))
sns.countplot(x="recommendation", data=reviews)
plt.title("Recommendations breakdown")
plt.show()
```



In [5]:
```
#Lets see how the values compare as percentages
reviews["recommendation"].value_counts(normalize=True)
```

Out[5]:
```
Recommended        0.69809
Not Recommended    0.30191
Name: recommendation, dtype: float64
```

In [6]:
```
#Creating a pie chart makes it easier to visualise the percentage breakdown of the reviews
rec_percent = reviews["recommendation"].value_counts(normalize=True)
plt.figure(figsize=(6,6))
plt.pie(rec_percent, labels=rec_percent.index, autopct='%1.1f%%', startangle=90)
plt.title("Percentages of positive and negative reviews")
plt.show()
```



By simply assigning all reviews as Recommended, the result would be about 70% accurate. This is the baseline score to beat with the algorithm.

Know lets see how the reviews are broken down between the games. This is not essential for the algorithm to work, but gives a better understanding of the dataset being worked with.

In [7]:
```
print("The", len(reviews["review"]), "reviews are split between", len(reviews["title"].unique()), "games")
```

```
The 434891 reviews are split between 48 games
```

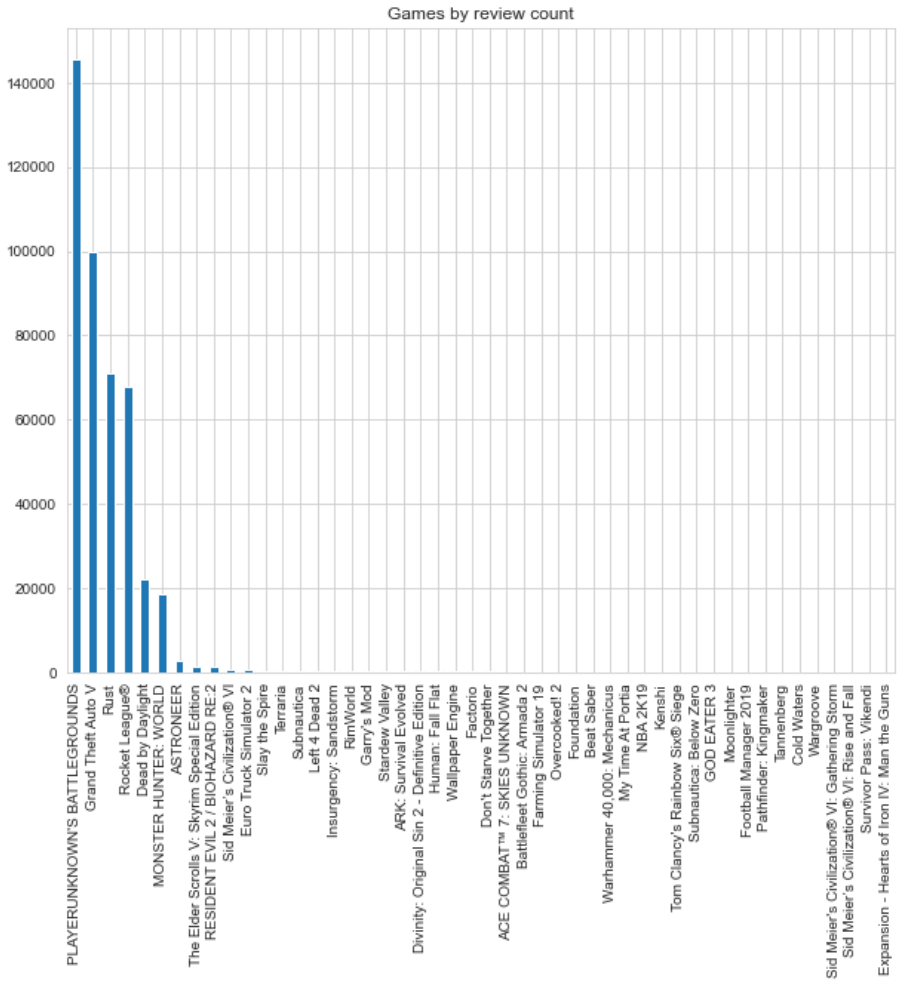
In [8]:
```
#Bring up the top 10 games by number of reviews
reviews.title.value_counts().head(10)
```

Out[8]:
```
PLAYERUNKNOWN'S BATTLEGROUNDS                145685
Grand Theft Auto V                            99956
Rust                                          71088
Rocket League®                                67907
Dead by Daylight                              22221
MONSTER HUNTER: WORLD                         18412
ASTRONEER                                      2661
The Elder Scrolls V: Skyrim Special Edition    1473
RESIDENT EVIL 2 / BIOHAZARD RE:2               1385
Sid Meier’s Civilization® VI                    522
Name: title, dtype: int64
```

It is clear the number of reviews per game drops off dramatically. By creating a graph the distribution of reviews per game is more easily visualised.

In [9]:
```
#Take the number of reviews per game title and plot them to a bar graph
user_reviews = reviews['title'].value_counts()
user_reviews.plot(kind='bar',figsize=(10,8))
plt.title('Games by review count')
```

Out[9]:
```
Text(0.5, 1.0, 'Games by review count')
```



Here it is obvious how heavily skewed the number of reviews are per game. Most of the reviews belong to a small number of games, with only the top 25 games having over 100 reviews, and only 1 game having over 100,000 reviews. However, as the objective of this project is to build a model to determine whether a review is positive or negative, the low number of reviews for some games does not negatively effect the ability of the algorithm to achieve our objectives.

Data Cleaning

In [10]:
```
reviews = reviews.drop('date_posted', axis=1)
#Drop the date posted column as it is not essential to our goals here.
```

As it is the text reviews being analysed by the algorithm, it is essential that there are no empty reviews, as this may skew the results of the classifier.

In [11]:
```
#This while show the number of empty reviews
reviews.isnull().sum()
```

Out[11]:
```
funny                     0
helpful                   0
hour_played               0
is_early_access_review    0
recommendation            0
review                 1516
title                     0
dtype: int64
```

In [12]:
```
#Dropping the empty reviews
reviews = reviews.dropna()
```

In [13]:
```
#Rechecking to make sure there are no empty reviews left
reviews.isnull().sum()
```

Out[13]:
```
funny                     0
helpful                   0
hour_played               0
is_early_access_review    0
recommendation            0
review                    0
title                     0
dtype: int64
```

Now to analyse the text reviews to see if there are any accidental duplicates within the dataset.

In [14]:
```
#In preparation for later analysis, convert all reveiws to lower case and then check for duplicates
reviews['review'] = reviews['review'].str.lower()
print("There are",reviews['review'].duplicated().sum(),"reviews with the same text")
```

```
There are 60704 reviews with the same text
```

Based on personal judgement and knowledge of gaming reviews, this project hypothesise that many of these duplicated reviews will likely be due to use of simple language common amongst gamers - for example, "gg" or "good game". Thus, by displaying a section of the reviews this hypothesis can be tested in a very rudimentary fashion.

In [15]:
```
#Taking all the reviews with the same text, and putting them into a new dataframe to make viewing the duplicate text r
duplicate_texts = reviews['review'].duplicated(keep=False)
duplicates = reviews[duplicate_texts]
duplicates.head(25)
```

Out[15]:

	funny	helpful	hour_played	is_early_access_review	recommendation	review	title
1	0	0	184	False	Recommended	yes.	Expansion - Hearts of Iron IV: Man the Guns
3	126	1086	676	False	Recommended	out of all the reviews i wrote this one is pro...	Dead by Daylight
6	12	228	48	False	Recommended	out of all the reviews i wrote this one is pro...	Dead by Daylight
14	0	0	654	False	Recommended	fun game to play with friends	Dead by Daylight
17	0	0	1449	False	Recommended	reeeeeeeee	Dead by Daylight
19	2	1	185	False	Recommended	i think they did a pretty good job so far.	Dead by Daylight
25	0	0	12	False	Recommended	good	Dead by Daylight
27	0	0	8	False	Recommended	fun game	Dead by Daylight
28	1	1	263	False	Recommended	its okay.	Dead by Daylight
51	1	1	274	False	Recommended	gg	Dead by Daylight
56	0	0	1	False	Recommended	yes	Dead by Daylight
57	0	0	342	False	Recommended	graphics you forget what reality is beautiful ...	Dead by Daylight
64	0	0	131	False	Recommended	gud	Dead by Daylight
106	1	1	515	False	Recommended	this game is amazing	Dead by Daylight
116	0	0	261	False	Recommended	product received for free. pretty good job so ...	Dead by Daylight
120	0	0	756	False	Recommended	pretty funny	Dead by Daylight
127	0	0	1186	False	Recommended	it's fun!	Dead by Daylight
138	0	0	548	False	Recommended	best game ever made.	Dead by Daylight
142	0	0	396	False	Not Recommended	too many bugs.	Dead by Daylight
145	1	1	128	False	Recommended	best game i love it	Dead by Daylight
153	0	0	0	False	Recommended	product received for free. pretty good game	Dead by Daylight
158	0	0	19	False	Recommended	best game)	Dead by Daylight
173	0	0	89	False	Recommended	one of the best	Dead by Daylight
180	0	0	233	False	Recommended	love this game	Dead by Daylight
191	0	0	63	False	Recommended	nice	Dead by Daylight

Although many of these reviews have the same text - therefore being counted as duplicates - due to the nature of this text the duplicates will be kept in the dataset. This is becuase, for the most part, this text is simple and likely to be copied between reviews, ie. "fun game" or "good"; therefore it could be argued the majority of these duplicates are not actually duplicated reviews, but instead common phrases expected to be found within reviews. There are definately some reviews which are clear duplicates (see number 3 and 6). However, given the number of seemingly legitimate reviews which are not accidental duplicates in the dataset, their inclusion is acceptable - it is judged that removing all duplicates would lose more legitimate reviews than accidental duplicates.

Creating and testing the algorithm

In [16]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import math
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn import metrics
from sklearn import tree
```

In [17]:
```
#Remove any punctuation for our analysis, ensure reviews are in lower case, and find the 1000 most used words.
text_filter = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True,split=' ',num_words = 1000)
#Create a new column showing the sequences the most used words appear in each review
sequence = text_filter.texts_to_sequences(reviews["review"].astype(str))
reviews['sequence'] = sequence
#Convert all the text reviews to a list
reviewtext = reviews["review"].astype(str).to_list()
#Take the words from the above list and convert them to a new dataframe showing the most used words and the frequency
text_filter.fit_on_texts(reviewtext)
texts = pd.DataFrame.from_dict(text_filter.word_counts, orient="index",columns=["frequency"])
texts.sort_values(by="frequency", ascending=False).head()
```

Out[17]:

	frequency
the	680652
game	505748
and	463968
to	442708
a	424284

In [18]:
```
#takes the text reviews, converts these into a sequence of the top 1000 words that are used in each review, and finall
sequences = text_filter.texts_to_sequences(reviews["review"].astype(str).to_list())
padded = pad_sequences(sequences, maxlen=65, padding='post', truncating='post')
reviews["sequences"] = list(padded)
reviews.head()
```

Out[18]:

	funny	helpful	hour_played	is_early_access_review	recommendation	review	title	sequence	sequences
0	2	4	578	False	Recommended	&gt played as german reich&gt declare war on b...	Expansion - Hearts of Iron IV: Man the Guns	[]	[117, 85, 31, 117, 18, 117, 138, 898, 25, 120,...
1	0	0	184	False	Recommended	yes.	Expansion - Hearts of Iron IV: Man the Guns	[]	[240, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
2	0	0	892	False	Recommended	very good game although a bit overpriced in my...	Expansion - Hearts of Iron IV: Man the Guns	[]	[47, 27, 2, 553, 5, 254, 13, 26, 566, 484, 69,...
3	126	1086	676	False	Recommended	out of all the reviews i wrote this one is pro...	Dead by Daylight	[]	[58, 9, 36, 1, 436, 6, 11, 49, 7, 336, 1, 100,...
4	85	2139	612	False	Recommended	disclaimer i survivor main. i play games for f...	Dead by Daylight	[]	[6, 535, 508, 6, 23, 61, 12, 21, 20, 12, 25, 1...

In [19]:
```
#Assign values to X and y based on the sequences column created earlier, and whether a review is Recommended or Not Re
X = np.array(reviews["sequences"].to_list())
y = reviews['recommendation'].values
#Split the dataset into test and train sets, using a 80/20 split for train and test sets. This is standard for this ty
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1000)
```

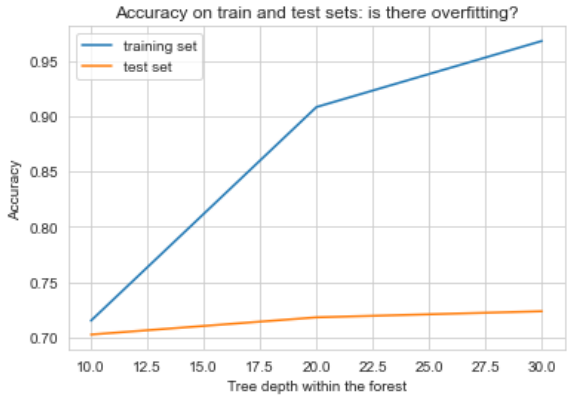
In [20]:
```
#Create the random forest classifier and set it's hyper-parameters
forest = RandomForestClassifier(n_estimators=50,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=None,rar
#Train the classifier on the training data
forest.fit(X_train,y_train)
#Generate predications for the training data
y_train_pred = forest.predict(X_train)
#Calculate the accuracy for the training data
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
#Generate predications for the test data
y_test_pred = forest.predict(X_test)
#Calculate the accuracy for the test data
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
#Calculate the OOB score (Out of the bag score - a method of estimating the accuracy of the model on unseen data)
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 0.98
Test set accuracy: 0.73
OOB score: 0.72
```

The test accuracy and OOB score (Out of the bag score - a method of estimating the accuracy of the model on unseen data) highlight that this model is more effective than the baseline. However, this is by a limited amount.

The high accuracy on the training set compared with the noticably lower accuracy on the test set is a clear sign of overfitting to the test set. By plotting the accuracy over several differing maximum tree depths, any overfitting will be clear, and it will help in finding the optimal depth for future testing.

In [21]:
```
#Set the variable tested, namely maximum tree depth
max_depth_vals = [10,20,30]
accuracytrain_list=[]
accuracytest_list=[]
#Create a loop to run random forest classifiers of varying maximum depths and calculate how this changes the accuracy
for i in range(0,len(max_depth_vals)):
    forest = RandomForestClassifier(n_estimators=35, bootstrap=True, max_features="sqrt",
                                    criterion='gini', max_depth=max_depth_vals[i], random_state=i, oob_score=Fals
    forest.fit(X_train,y_train)
    ytrain_pred = forest.predict(X_train)
    accuracy_train = metrics.accuracy_score(y_train, ytrain_pred)
    accuracytrain_list.append(accuracy_train)
    ytest_pred = forest.predict(X_test)
    accuracy_test = metrics.accuracy_score(y_test, ytest_pred)
    accuracytest_list.append(accuracy_test)
#Plot the above to a graph
fig = plt.figure()
ax = plt.axes()
line1, = ax.plot(max_depth_vals,accuracytrain_list,label='training set')
line2, = ax.plot(max_depth_vals,accuracytest_list,label='test set')
plt.legend(handles=[line1, line2])
plt.title("Accuracy on train and test sets: is there overfitting?")
plt.xlabel("Tree depth within the forest")
plt.ylabel("Accuracy")
plt.show()
```

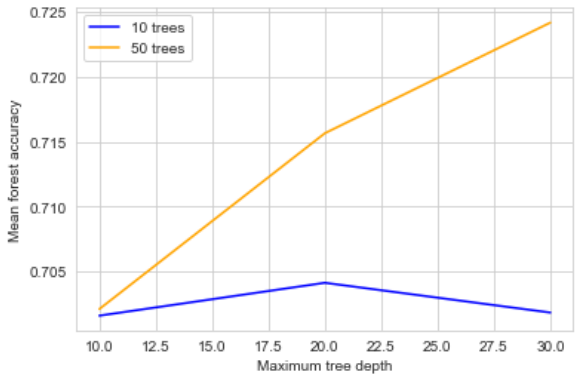


Clearly increasing the available tree depth leads to a large amount of overfitting - this is a common problem when using decision trees and random forest classifiers and is therefore expected. Although, increasing the tree depth does also increase the accuracy in the test set. Comparing the accuracy on the graph to the accuracay on the previous test - with no max depth - it can be seen that the accuracy is highly similar. While further testing of hyper-perameters is required before declaring the optimal settings for this model, it can likely be said that increasing the maximum tree depth beyond 30 will likely not result in a noticable increase in accuracy.

Now to test how the number of trees used in each model affects the accuracy, and how this is impacted by the depth of the trees.

In [22]:
```
#Create a graph to visualize the accuracy of the random forest classifier model with differing combinations of hyperpa
max_depth_vals = [10, 20, 30]
#Above, differing maximum tree depth. Below, differing number of decision trees per model.
n_estimators_vals = [10, 50]
mean_accuracy_store = []
sd_accuracy_store = []
k = 5 #number of folds for cross-validation
#Loop through each combination of hyperparameters, creating a model for each, cross-validate with the dataset, getting
#These are then appended to the corresponding lists.
for i, n_estimators in enumerate(n_estimators_vals):
    mean_accuracy_cv = []
    sd_cv = []
    for max_depth in max_depth_vals:
        forest = RandomForestClassifier(n_estimators=n_estimators, bootstrap=True, max_features="sqrt",
                                        criterion='gini', max_depth=max_depth, random_state=i, oob_score=False)
        cv_scores = cross_val_score(forest, X, y, cv=k)
        avg = sum(cv_scores) / len(cv_scores)
        sd = math.sqrt(sum((cv_scores - avg) ** 2) / (len(cv_scores) - 1))
        mean_accuracy_cv.append(avg)
        sd_cv.append(sd)
    mean_accuracy_store.append(mean_accuracy_cv)
    sd_accuracy_store.append(sd_cv)
#Plot the above to a graph
```

```
fig = plt.figure()
ax = plt.axes()
line1, = ax.plot(max_depth_vals, mean_accuracy_store[0], color='blue', label='10 trees')
line2, = ax.plot(max_depth_vals, mean_accuracy_store[1], color='orange', label='50 trees')
plt.legend(handles=[line1, line2])
plt.xlabel("Maximum tree depth")
plt.ylabel("Mean forest accuracy")
plt.show()
```



Here the deeper the tree the greater the average accuracy; however, given the linear nature of the increase after 10 levels, this could be down to overfitting - as otherwise some variation should be expected. An alternative could be that the maximum accuracy has not been reached, however due to hardware constraints of all available computers it is not possible to test this further.

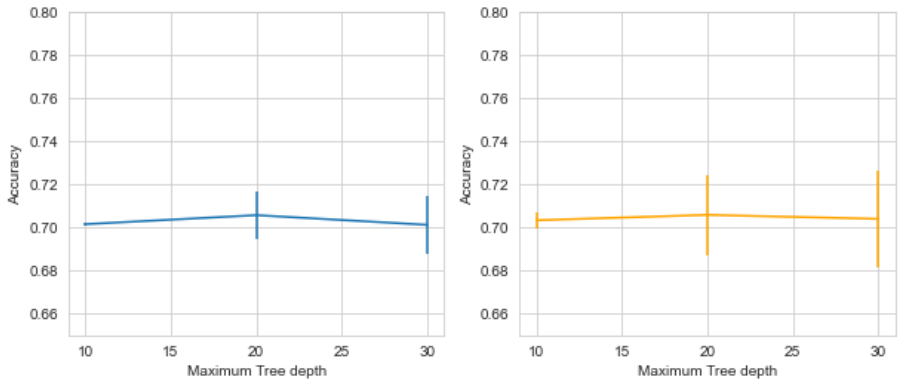
Adjusting the maximum depth of the trees used in the model may help increase the accuracy. To test this lets run several models with varying max tree depths, using the lower n_estimator (number of trees per model) value to keep this consistent between the differing models.

In [23]:
```
#Set the maximum tree depths to cycle through
max_depth_vals = [10,20,30]

k=5
mean_accuracy_cv_k5 = []
sd_cv_k5 = []
#Above are the lists for storing the mean accuracy and standard deviation scores over the k-folds for each of the tree
#Create a loop to perform a k-fold cross-validation using random forests with varying depths. Calculates the mean accu
for i in range(0,len(max_depth_vals)):
    forest = RandomForestClassifier(n_estimators=10, bootstrap=True, max_features="sqrt",
                                    criterion='gini', max_depth=max_depth_vals[i], random_state=i, oob_score=Fals
    cv_scores = cross_val_score(forest, X, y, cv=k)
    avg = sum(cv_scores)/len(cv_scores)
    sd = math.sqrt(sum((cv_scores-avg)**2)/(len(cv_scores)-1))
    mean_accuracy_cv_k5.append(avg)
    sd_cv_k5.append(sd)
```

In [24]:
```
#This does the same as explained above, but with a different number of k-folds.
k=10
mean_accuracy_cv_k10 = []
sd_cv_k10 = []
for i in range(0,len(max_depth_vals)):
    forest = RandomForestClassifier(n_estimators=10, bootstrap=True, max_features="sqrt",
                                    criterion='gini', max_depth=max_depth_vals[i], random_state=i, oob_score=Fals
    cv_scores = cross_val_score(forest, X, y, cv=k)
    avg = sum(cv_scores)/len(cv_scores)
    sd = math.sqrt(sum((cv_scores-avg)**2)/(len(cv_scores)-1))
    mean_accuracy_cv_k10.append(avg)
    sd_cv_k10.append(sd)
```

In [25]:
```
#Plot the above
fig, ax = plt.subplots(1,2, figsize=(10, 4))
ax[0].errorbar(max_depth_vals,mean_accuracy_cv_k5,yerr=sd_cv_k5)
ax[0].set_xlabel("Maximum Tree depth")
ax[0].set_ylabel("Accuracy")
ax[0].set_ylim([0.65,0.8])
ax[1].errorbar(max_depth_vals,mean_accuracy_cv_k10,yerr=sd_cv_k10, color="orange")
ax[1].set_xlabel("Maximum Tree depth")
ax[1].set_ylabel("Accuracy")
ax[1].set_ylim([0.65,0.8])
plt.show()
```



This shows that beyond a depth of 20 the average accuracy of the decision trees starts to drop off. Additionally, while the average accuracy does decline the standard deviation increases. However, this is based on a low level n_estimators - the number of decision trees used in the model. This is due to hardware limitations. By increasing this, the accuracy may be able to be increased. Lets re-test with some final adjustments to confirm if this is the case - using max depths of 20, and 30 respectively.

In [26]:
```
#max_depth set to 20
forest = RandomForestClassifier(n_estimators=50,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=20,randc
forest.fit(X_train,y_train)
y_train_pred = forest.predict(X_train)
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
y_test_pred = forest.predict(X_test)
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 0.91
Test set accuracy: 0.72
OOB score: 0.71
```

In [27]:
```
#max_depth set to 30
forest = RandomForestClassifier(n_estimators=50,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=30,randc
forest.fit(X_train,y_train)
y_train_pred = forest.predict(X_train)
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
y_test_pred = forest.predict(X_test)
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 0.97
Test set accuracy: 0.73
OOB score: 0.72
```

This confirms what was established earlier: increasing the maximum tree depth to 30 increases accuracy but not any more than having no maximum depth. Further tinkering with this hyper-perameter will likely not yeild anymore accuracy.

Earlier testing showed that using more trees per model increased the accuracy. Lets push this beyond the 50 tested earlier - while maintaining the optimal depth of 30 - to see if this will noticably increase the model accuracy. Now the number of decision trees per model will be increased to 75 to determine if more accuracy can be achieved by adjusting this hyper-perameter.

In [28]:
```
#n_estimators set to 75, max_depth set to 30
forest = RandomForestClassifier(n_estimators=75,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=30,randc
forest.fit(X_train,y_train)
y_train_pred = forest.predict(X_train)
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
y_test_pred = forest.predict(X_test)
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 0.97
Test set accuracy: 0.73
OOB score: 0.72
```

It is clear that adjusting the model to use more than 50 decision trees per model does not effect the results by any noticable amount. Thus, it can be assumed that - in its current form - the model is as accurate as it can be.

Dataset Two

Having tested this algorithm on one dataset, it is important to test it on another to ensure that it works consistently between varying data. Additionally, taking the lessons learnt from the testing the model on dataset one, the model can be fine-tuned for dataset two to be as accurate as possible. The second dataset is still based on Steam reviews and contains all the required information for the algorithm to work in its current form.

In [29]:
```
steam_reviews = pd.read_csv('m:\\pc\\desktop\\AI\\train.csv')
steam_reviews.head()
```

Out[29]:

	review_id	title	year	user_review	user_suggestion
0	1	Spooky's Jump Scare Mansion	2016.0	I'm scared and hearing creepy voices. So I'll...	1
1	2	Spooky's Jump Scare Mansion	2016.0	Best game, more better than Sam Pepper's YouTu...	1
2	3	Spooky's Jump Scare Mansion	2016.0	A littly iffy on the controls, but once you kn...	1
3	4	Spooky's Jump Scare Mansion	2015.0	Great game, fun and colorful and all that.A si...	1
4	5	Spooky's Jump Scare Mansion	2015.0	Not many games have the cute tag right next to...	1

In [30]:
```
#Renaming the columns of the new dataframe to ensure they align with those of the previous dataset
steam_reviews = steam_reviews.rename(columns={"user_review": "review", "user_suggestion": "recommendation"})
steam_reviews.head()
```

Out[30]:

	review_id	title	year	review	recommendation
0	1	Spooky's Jump Scare Mansion	2016.0	I'm scared and hearing creepy voices. So I'll...	1
1	2	Spooky's Jump Scare Mansion	2016.0	Best game, more better than Sam Pepper's YouTu...	1
2	3	Spooky's Jump Scare Mansion	2016.0	A littly iffy on the controls, but once you kn...	1
3	4	Spooky's Jump Scare Mansion	2015.0	Great game, fun and colorful and all that.A si...	1
4	5	Spooky's Jump Scare Mansion	2015.0	Not many games have the cute tag right next to...	1

In this dataset, a positive recommendation is denoted by the value "1", and a negative review by "0"; this is different to the preivous dataset where a positive review was denoted by "Recommended", and a negative by "Not Recommended". However, as the values were changed to binary for the last dataset, this difference saves a step in later lines of code.

Exploritory Data Analysis

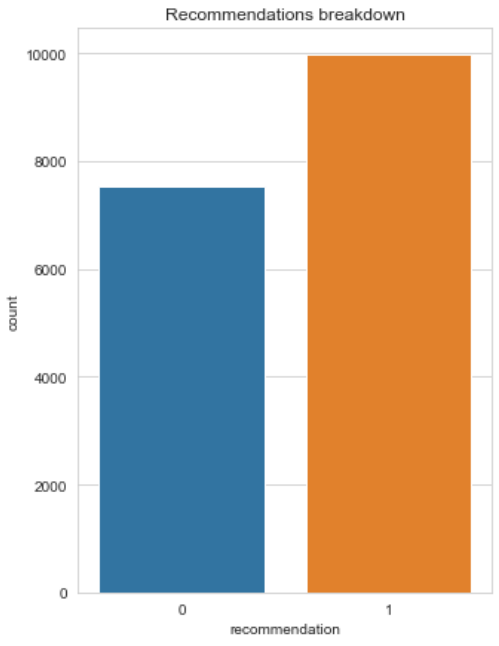
Many of the steps seen for dataset one are repeated here, thus many will not have in-code comments to explain them. This covers both the EDA and Data Cleaning sections of the project.

In [31]:
```
print("There are", len(steam_reviews["review"]), "reviews")
```

```
There are 17494 reviews
```

Noticably less reviews than the first dataset, but still enough to ensure a large sample for analysis.

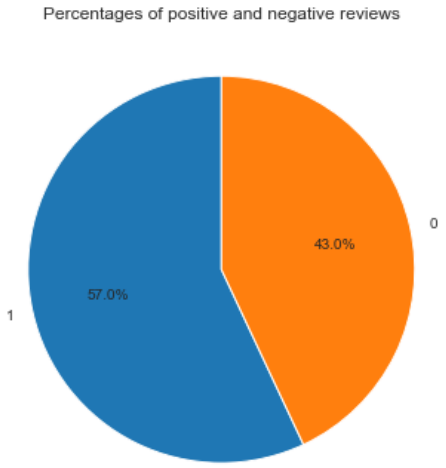
In [32]:
```
sns.set_style("whitegrid")
plt.figure(figsize=(5,7))
sns.countplot(x="recommendation", data=steam_reviews)
plt.title("Recommendations breakdown")
plt.show()
```



In [33]:
```
steam_reviews["recommendation"].value_counts(normalize=True)
```

Out[33]:
```
1    0.569795
0    0.430205
Name: recommendation, dtype: float64
```

In [34]:
```
rec_percent2 = steam_reviews["recommendation"].value_counts(normalize=True)
plt.figure(figsize=(6,6))
plt.pie(rec_percent2, labels=rec_percent2.index, autopct='%1.1f%%', startangle=90)
plt.title("Percentages of positive and negative reviews")
plt.show()
```



Dataset two contains a much more balanced level of reviews, compared with dataset one. This lowers the baseline for analysing the effectiveness of this algorithm, from 70% in dataset one to 57% in dataset two.

In [35]:
```
print("The", len(steam_reviews["review"]), "reviews are split between", len(steam_reviews["title"].unique()), "games")
```

```
The 17494 reviews are split between 44 games
```

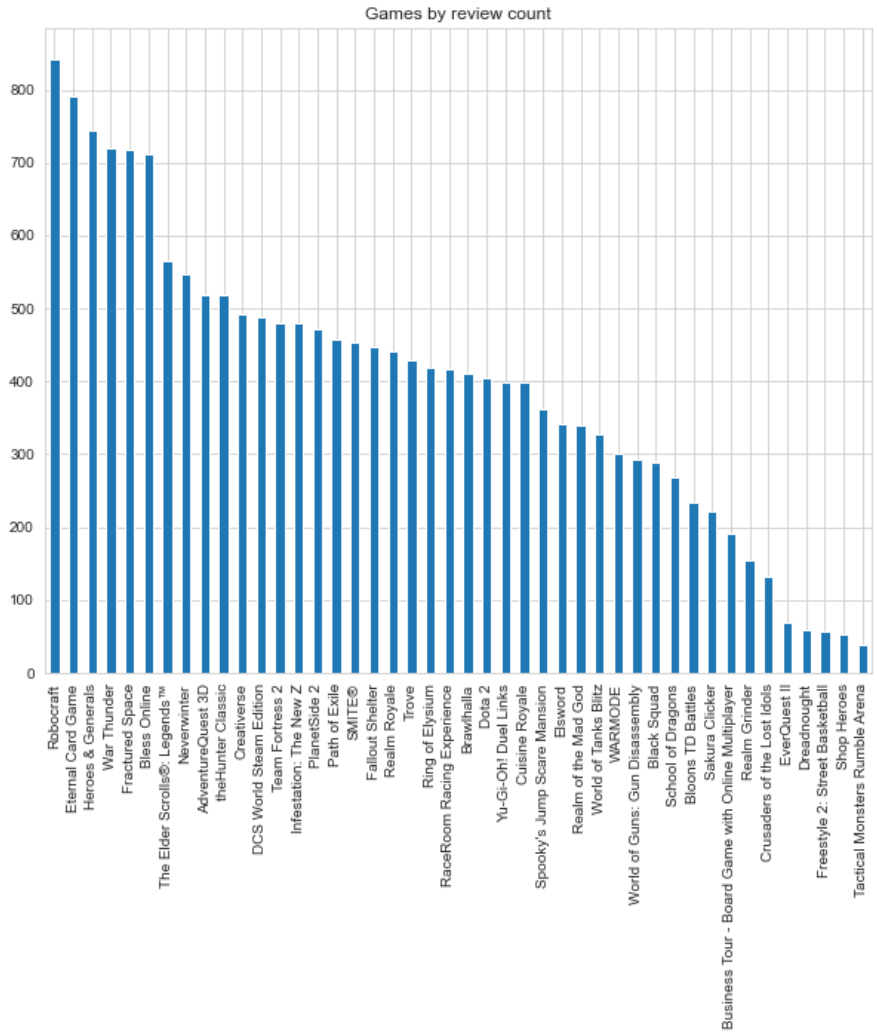
While there are noticably less reviews in dataset two compared with dataset one, the number of games remains quite similar. This should not noticably effect the algorithm's effectiveness however, as it studies the text reviews, not the games they are attatched to.

In [36]:
```
steam_reviews.title.value_counts().head(10)
```

Out[36]:
```
Robocraft                      842
Eternal Card Game              791
Heroes & Generals              745
War Thunder                    720
Fractured Space                718
Bless Online                   712
The Elder Scrolls®: Legends™   565
Neverwinter                    546
AdventureQuest 3D              519
theHunter Classic              518
Name: title, dtype: int64
```

In [37]:
```
user_reviews2 = steam_reviews['title'].value_counts()
user_reviews2.plot(kind='bar',figsize=(10,8))
plt.title('Games by review count')
```

Out[37]:
```
Text(0.5, 1.0, 'Games by review count')
```



Given the lower number of reviews, it would have been unexpected to see as noticable a difference in reviews per game as in dataset one. However, there is still a bias in the data towards some games having noticably more reviews, with the number of reviews per game dropping off rapidly - although nowhere near as rapidly as in dataset one.

Data Cleaning

In [38]:
```
steam_reviews = steam_reviews.drop('year', axis=1)
```

In [39]:
```
steam_reviews.isnull().sum()
```

Out[39]:
```
review_id         0
title             0
review            0
recommendation    0
dtype: int64
```

In [40]:
```
steam_reviews['review'] = steam_reviews['review'].str.lower()
print("There are",steam_reviews['review'].duplicated().sum(),"reviews with the same text")
```

```
There are 5 reviews with the same text
```

In [41]:
```
duplicate_texts2 = steam_reviews['review'].duplicated(keep=False)
duplicates2 = steam_reviews[duplicate_texts2]
duplicates2.head(5)
```

Out[41]:

	review_id	title	review	recommendation
77	78	Spooky's Jump Scare Mansion	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	1
449	450	Sakura Clicker	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa...	1
992	993	Fractured Space	#name?	1
2702	3166	War Thunder	#name?	0
6513	8888	Heroes & Generals	#name?	0

In [42]:
```
#Drop the duplicate reviews to help remove mistakes from later analysis
steam_reviews = steam_reviews.drop_duplicates(subset=['review'])
```

In [43]:
```
#Check to ensure the duplicates have been dropped
steam_reviews.duplicated().sum()
```

Out[43]:
```
0
```

Applying the algorithm to dataset two

As the model built ealier will now be applied to dataset two, and the steps below mirror those of earlier in the project, the code here will not have comments explain how the code functions.

In [44]:
```
#Using 1000 words as it contributed to the better score in earlier testing
text_filter = Tokenizer(filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',lower=True,split=' ',num_words = 1000)
sequence = text_filter.texts_to_sequences(steam_reviews["review"].astype(str))
steam_reviews['sequence'] = sequence
steam_reviewtext = steam_reviews["review"].astype(str).to_list()
text_filter.fit_on_texts(steam_reviewtext)
dftext = pd.DataFrame.from_dict(text_filter.word_counts, orient="index",columns=["frequency"])
dftext.sort_values(by="frequency", ascending=False).head()
```

Out[44]:

	frequency
the	99657
to	71165
and	66188
a	59987
game	52448

In [45]:
```
#Using a maximum sequence length of 65
sequences = text_filter.texts_to_sequences(steam_reviews["review"].astype(str).to_list())
padded = pad_sequences(sequences, maxlen=65, padding='post', truncating='post')
steam_reviews["sequences"] = list(padded)
steam_reviews
```

Out[45]:

	review_id	title	review	recommendation	sequence	sequences
0	1	Spooky's Jump Scare Mansion	i'm scared and hearing creepy voices. so i'll...	1	[]	[139, 3, 27, 496, 13, 4, 824, 3, 4, 181, 117, ...
1	2	Spooky's Jump Scare Mansion	best game, more better than sam pepper's youtu...	1	[]	[125, 5, 36, 98, 80, 902, 441, 91, 259, 114, 2...
2	3	Spooky's Jump Scare Mansion	a littly iffy on the controls, but once you kn...	1	[]	[4, 19, 1, 524, 15, 218, 6, 131, 92, 2, 26, 51...
3	4	Spooky's Jump Scare Mansion	great game, fun and colorful and all that.a si...	1	[]	[83, 5, 45, 3, 3, 32, 14, 4, 450, 851, 188, 59...
4	5	Spooky's Jump Scare Mansion	not many games have the cute tag right next to...	1	[]	[20, 102, 63, 18, 1, 174, 363, 2, 1, 19, 167, ...
...	...	...	...	...	...	...
17489	25535	EverQuest II	arguably the single greatest mmorp that exists...	1	[]	[1, 312, 14, 3, 71, 52, 62, 41, 9, 186, 2, 50,...
17490	25536	EverQuest II	an older game, to be sure, but has its own cha...	1	[]	[61, 5, 2, 22, 249, 15, 47, 52, 232, 3, 4, 689...
17491	25537	EverQuest II	when i frist started playing everquest 2 it wa...	1	[]	[59, 7, 307, 73, 87, 10, 38, 238, 52, 84, 83, ...
17492	25538	EverQuest II	cool game. the only thing that really pisses m...	1	[]	[317, 5, 1, 57, 140, 14, 58, 55, 162, 9, 1, 10...
17493	25539	EverQuest II	this game since i was a little kid, always hav...	1	[]	[11, 5, 143, 7, 38, 4, 186, 225, 18, 143, 7, 8...

17489 rows × 6 columns

In [46]:
```
X = np.array(steam_reviews["sequences"].to_list())
y = steam_reviews['recommendation'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1000)
```

Using previous testing, the number of trees used per model will be set to the earlier optimal of 50, and the maximum tree depth will be set to the earlier optimal of 30

In [47]:
```
#Using n_estimators of 50 and max_depth of 30
n_estimator_list = []
forest = RandomForestClassifier(n_estimators=50,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=30,randc
```

```
forest.fit(X_train,y_train)
y_train_pred = forest.predict(X_train)
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
y_test_pred = forest.predict(X_test)
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 1.0
Test set accuracy: 0.56
OOB score: 0.54
```

Clearly the best hyper-perameters which were established for dataset one, are not optimal for dataset two. This could be due to a number of factors, but a noticable difference between the two datasets is the sample size - over 400,00 compared to roughly 17,000. It was previously tested to establish whether increasing the number of trees used per model, and increasing the maximum tree depth, would increase the accuracy of the model. In dataset one it did not, however given the differences in dataset two, it may be possible that increasing these hyper-perameters may increase the accuracy of the model. Lets use the forest size of 75 and remove the maximum tree depth, as these were tested earlier on dataset one.

In [48]:
```
#Increased n_estimators to 75, and removed limits on the maximum tree depth.
forest = RandomForestClassifier(n_estimators=75,bootstrap=True,max_features='sqrt',criterion='gini',max_depth=None,rar
forest.fit(X_train,y_train)
y_train_pred = forest.predict(X_train)
print("Training set accuracy:",round(metrics.accuracy_score(y_train, y_train_pred),2))
y_test_pred = forest.predict(X_test)
print("Test set accuracy:",round(metrics.accuracy_score(y_test, y_test_pred),2))
print("OOB score:",round(forest.oob_score_,2))
```

```
Training set accuracy: 1.0
Test set accuracy: 0.58
OOB score: 0.56
```

For dataset two an increased number of trees per model and maximum tree depth does result in a greater model accuracy. However, the model is still not much more accurate tham baseline, even after all the adjusting and testing done in this report.

Conclusion

In conclusion, the algorithm designed has achieved the stated objective of creating a random forest review sentiment classifier which is more effective than the baseline. However, this has only barely been done with a 3% and 1% increase in accuracy for datasets one and two respectively over the baseline. Even with the adjustments to the model seen, the accuracy of the model has not improved by the desired amount. Therefore, it is concluded that despite the advantages explained earlier that a random forest classifier has, using it for the purpose of text analysis could be considered sub-par, especially when compared with other potential models. As the purpose of this project was to assess whether a simple model such as a random forest classifier was sufficiently accurate enough to serve as a text sentiment analyser, we can conclude that this project has served its purpose well, showing that - in this current format - random forest classifiers lack the ability to determine full text sentiment to a good enough degree for implementation in other projects. Future analysis of this conclusion should be considered, potentially comparing the use of a random forest classifier against other, more advanced methods such as artificial neural networks. However, due to the limitations of this project, this is currently beyond the scope of this report.

Bibliography:

[1] https://www.kaggle.com/datasets/luthfim/steam-reviews-dataset [2] https://www.kaggle.com/datasets/arashnic/game-review-dataset?resource=download

In [ ]:
```
```