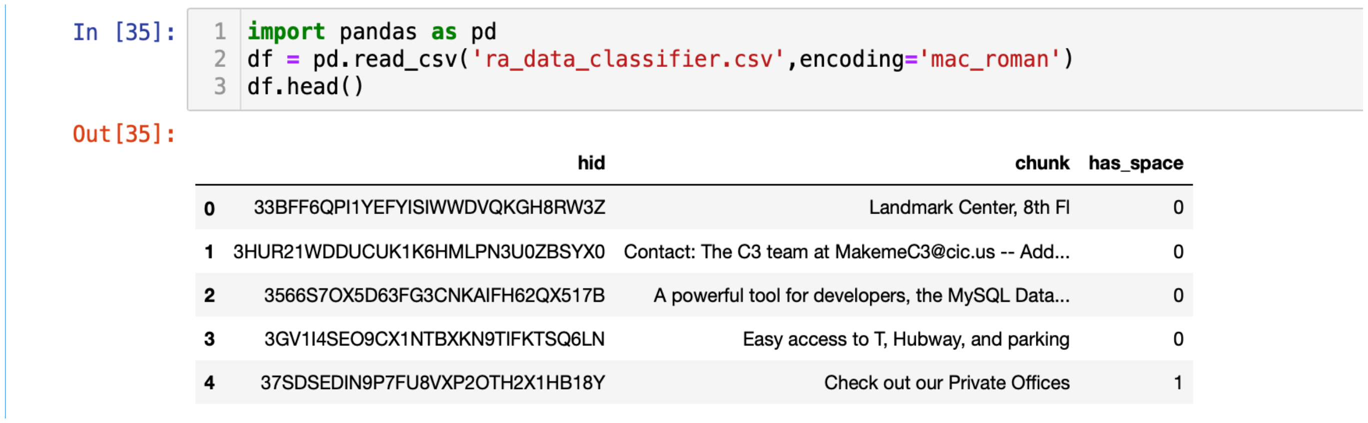
Marshall: Task 2

Text classification to describe whether a commercial establishment that offers office space for rent.

Problem Formulation:

The problem is to perform text classification on a given chunk of data and categorize whether a commercial establishment offers office space for rent. Given a new a new chunk of data, we want to assign it to one of 2 categories

Data Exploration:



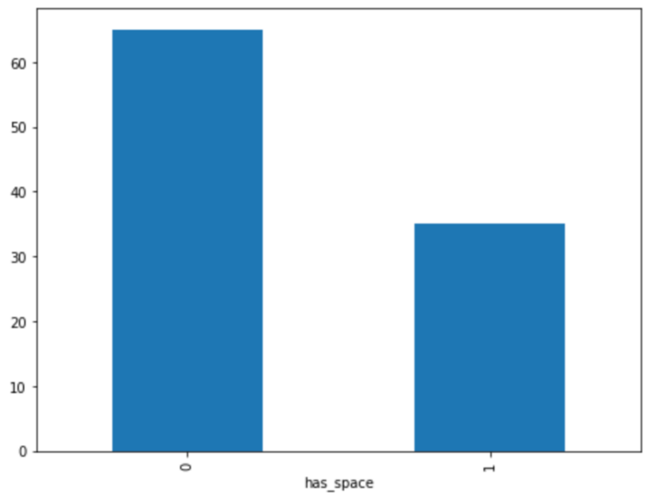
```
In [35]:
import pandas as pd
df = pd.read_csv('ra_data_classifier.csv',encoding='mac_roman')
df.head()
```

Out[35]:

	hid	chunk	has_space
0	33BFF6QPI1YEFYISIWWDVQKGH8RW3Z	Landmark Center, 8th Fl	0
1	3HUR21WDDUCUK1K6HMLPN3U0ZBSYX0	Contact: The C3 team at MakemeC3@cic.us -- Add...	0
2	3566S7OX5D63FG3CNKAIFH62QX517B	A powerful tool for developers, the MySQL Data...	0
3	3GV1I4SEO9CX1NTBXKN9TIFKTSQ6LN	Easy access to T, Hubway, and parking	0
4	37SDSEDIN9P7FU8VXP2OTH2X1HB18Y	Check out our Private Offices	1

For this project we only need two columns, chunk and has_space. The hid column maybe removed or kept without using. The last columns define the labels saying for the given chunk of data, if the office space is available or not.

Class Imbalance:



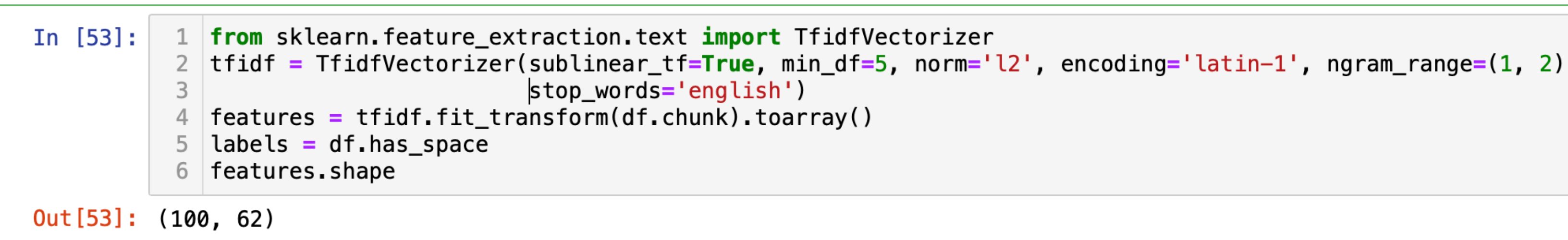
Conventional algorithms are often biased towards the majority class, not taking the data distribution into consideration. In the worst case, minority classes are treated as outliers and ignored. For some cases, such as fraud detection or cancer prediction, we would need to carefully configure our model or artificially balance the dataset, for example by under sampling or oversampling each class.

However, in our case of learning imbalanced data, the majority classes might be of our great interest. It is desirable to have a classifier that gives high prediction accuracy over the majority class, while maintaining reasonable accuracy for the minority classes. Therefore, we will leave it as it is.

Text Representation:

The classifiers and learning algorithms cannot directly process the text documents in their original form, as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length. Therefore, during the preprocessing step, the texts are converted to a more manageable representation.

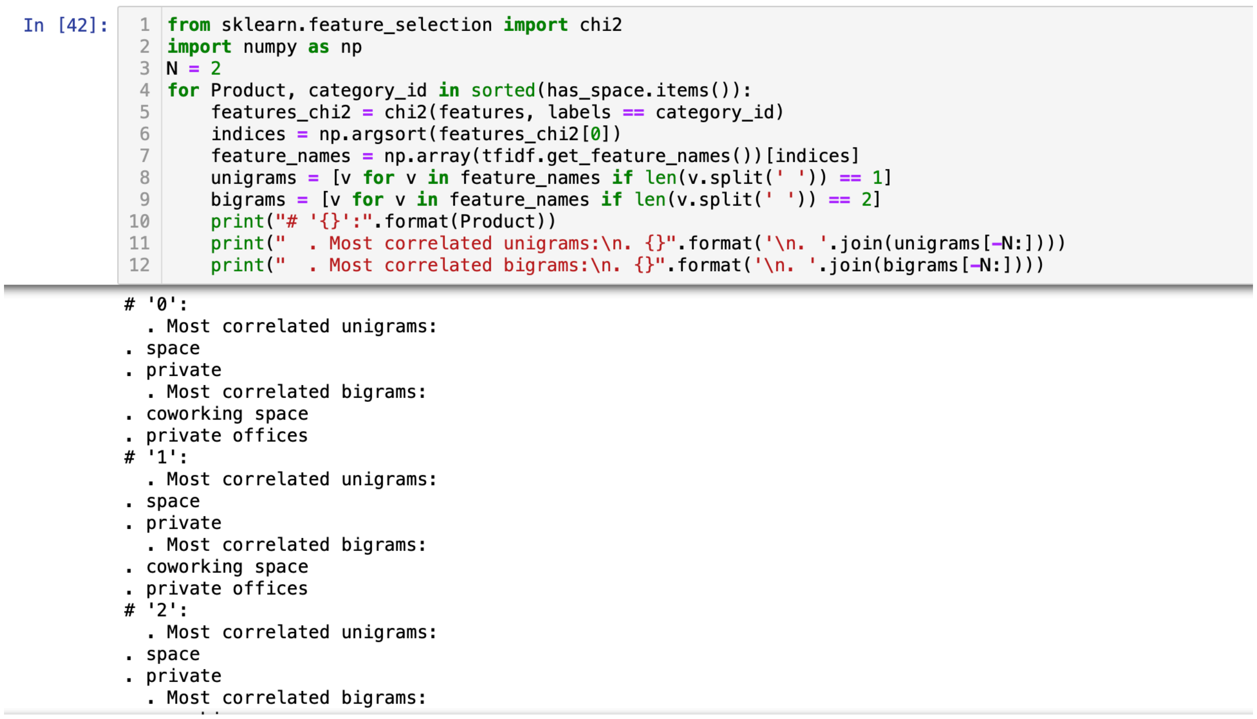
So, inorder to do this, we extract features from the text using the tf-idf feature extractor from sklearn. For each term in the data, it calculates the Term-Frequency, Inverse Document Frequency, i.e tf-idf.



```
In [53]: from sklearn.feature_extraction.text import TfidfVectorizer
         tfidf = TfidfVectorizer(sublinear_tf=True, min_df=5, norm='l2', encoding='latin-1', ngram_range=(1, 2)
                                 stop_words='english')
         features = tfidf.fit_transform(df.chunk).toarray()
         labels = df.has_space
         features.shape
Out[53]: (100, 62)
```

Now, each of 100 chunks of text is represented by 62 features, representing the tf-idf score for different unigrams and bigrams.

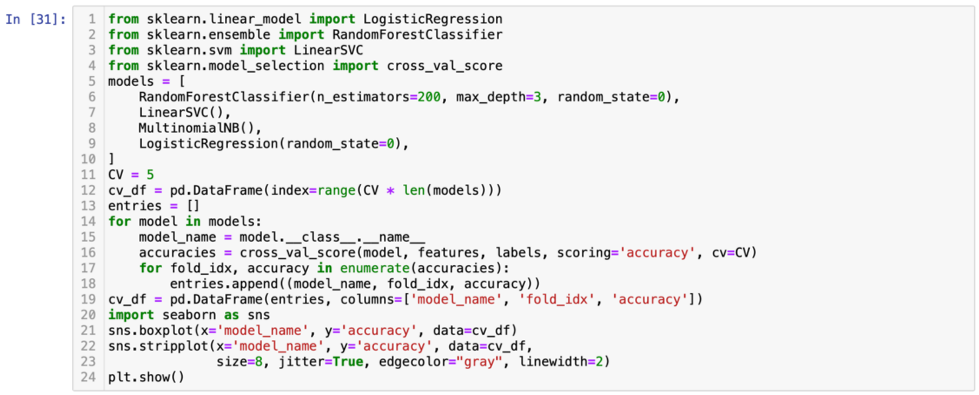
We can use sklearn.feature_selection.chi2 to find the terms that are the most correlated with each of the chunk of texts:



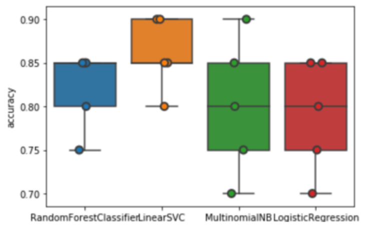
In [42]:
```python
from sklearn.feature_selection import chi2
import numpy as np
N = 2
for Product, category_id in sorted(has_space.items()):
    features_chi2 = chi2(features, labels == category_id)
    indices = np.argsort(features_chi2[0])
    feature_names = np.array(tfidf.get_feature_names())[indices]
    unigrams = [v for v in feature_names if len(v.split(' ')) == 1]
    bigrams = [v for v in feature_names if len(v.split(' ')) == 2]
    print("# '{}':".format(Product))
    print("  . Most correlated unigrams:\n. {}".format('\n. '.join(unigrams[-N:])))
    print("  . Most correlated bigrams:\n. {}".format('\n. '.join(bigrams[-N:])))
```

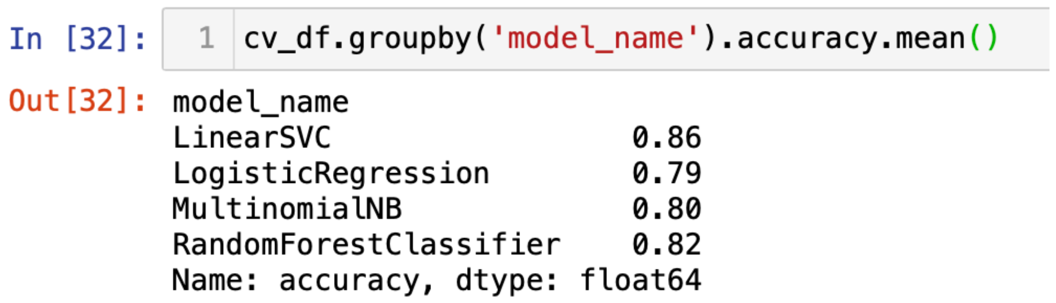
```
# '0':
  . Most correlated unigrams:
. space
. private
  . Most correlated bigrams:
. coworking space
. private offices
# '1':
  . Most correlated unigrams:
. space
. private
  . Most correlated bigrams:
. coworking space
. private offices
# '2':
  . Most correlated unigrams:
. space
. private
  . Most correlated bigrams:
```

Now, it is time to choose a Classifier. There are a number of algorithms we can use for this type of problem. I have tried to experiment various machine learning models to predict and compare the accuracy obtained by each of them with the following four models:
* Logistic Regression
* (Multinomial) Naive Bayes
* Linear Support Vector Machine
* Random Forest



In [31]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_val_score
models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0),
]
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, features, labels, scoring='accuracy', cv=CV)
    for fold_idx, accuracy in enumerate(accuracies):
        entries.append((model_name, fold_idx, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])
import seaborn as sns
sns.boxplot(x='model_name', y='accuracy', data=cv_df)
sns.stripplot(x='model_name', y='accuracy', data=cv_df,
              size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.show()
```





```
In [32]: cv_df.groupby('model_name').accuracy.mean()

Out[32]: model_name
         LinearSVC                 0.86
         LogisticRegression        0.79
         MultinomialNB             0.80
         RandomForestClassifier    0.82
         Name: accuracy, dtype: float64
```

Linear SVC and Random Forest Classifier performs the better than the other two classifiers, and Linear SVC performs slightly better than Random Forest with an accuracy of 86%.