**Introdução**

Bitcoins e outras criptomoedas estão sendo muito exploradas no mundo atualmente e sua variação tem um valor elevado.

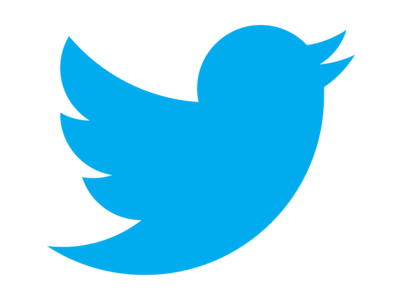
Juntamente com o mundo conectado que vivemos, podemos acompanhar em redes sócias informações on-line sobre bitcoins, informações essas que nos ajuda tomar decisões se está na hora de entrar nesse mercado ou não.

Por esse motivo, me senti motivado a minerar o Twitter em busca uma ajuda para identificar se o que estão falando no momento sobre bitcoin é bom ou ruim e possivelmente utilizar essa informação para acompanhar o mercado de Bitcoins para poder tomar uma decisão de quando será o melhor momento para a compra dessa criptomoeda.

**Definição do Problema**

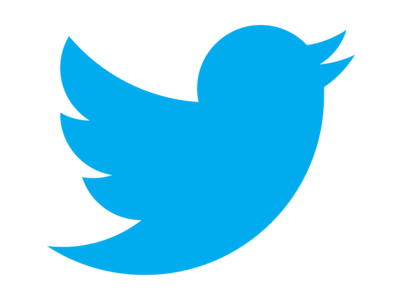
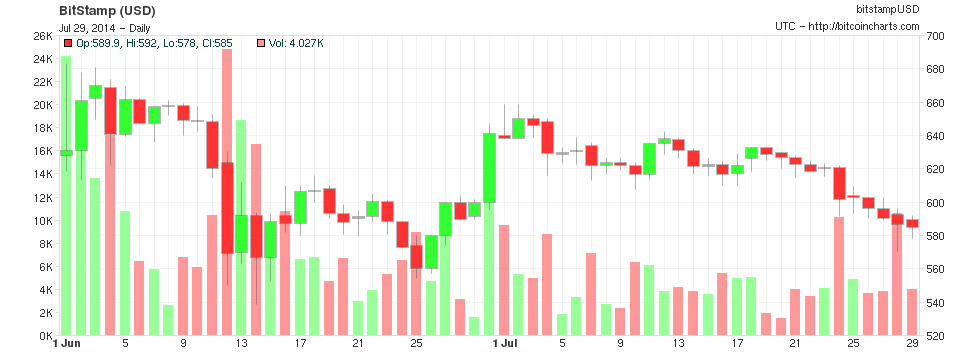
Primeiramente capturei em dias diferentes uma quantidade razoável de Tweets, o qual são textos publicados no Twitter, que contenham a palavra bitcoin. Após a captura desses Tweets, analisei e classifiquei eles manualmente como Positivo, Negativo ou Neutro. Após essa classificação usei esses textos (Tweets) como entrada para um modelo de classificação.





Treinamento e Teste

Tweets com a palavra Bitcoin Classificados previamente



Previsão

Tweets Novos

Correlação entre a classificação prevista e o valor do Bitcoin

**Conjunto de dados**

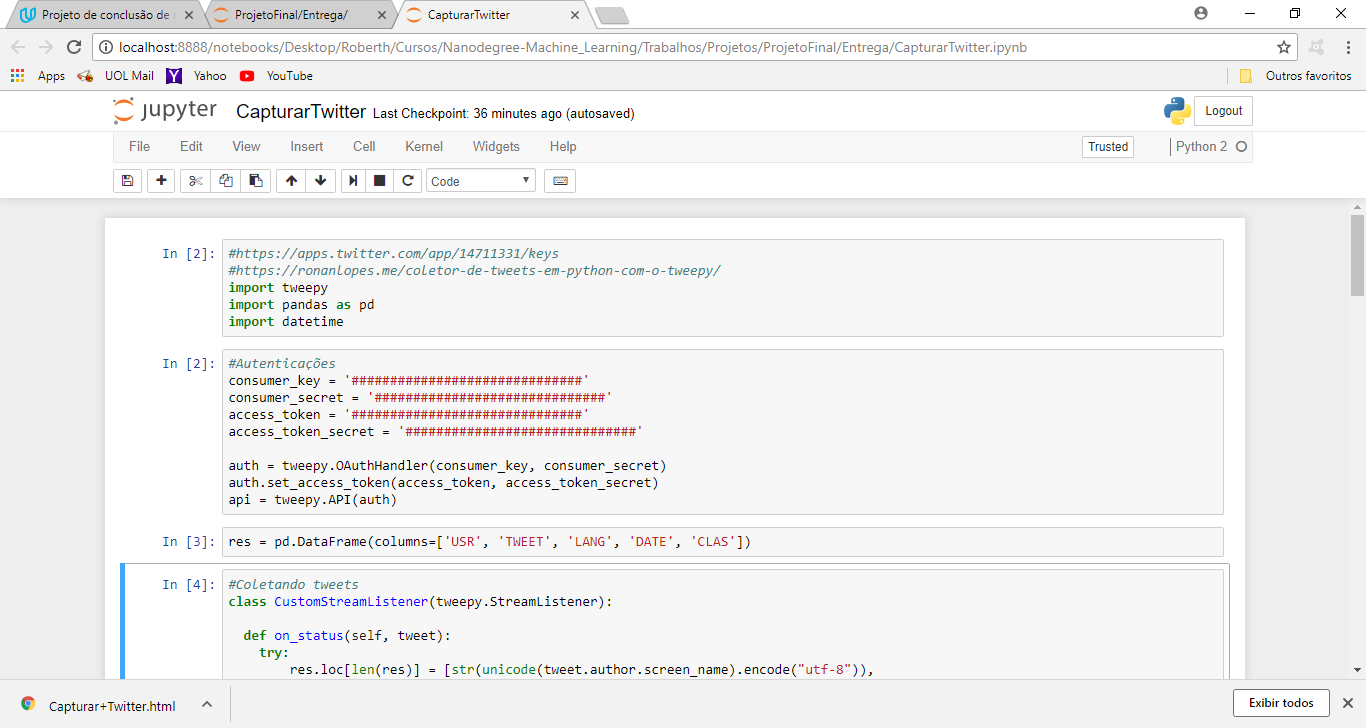
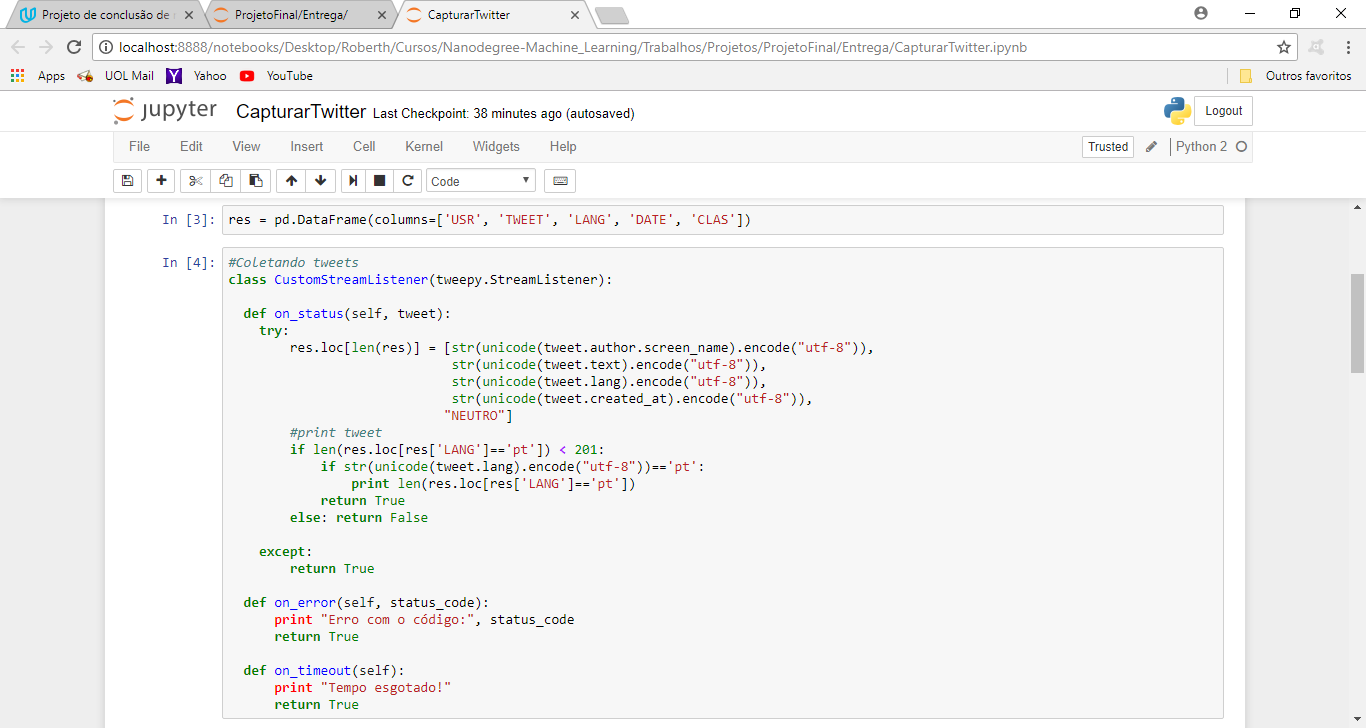
Para realização desse projeto usei os dados coletados do Twitter, no caso aproximadamente 270.000 Tweets.

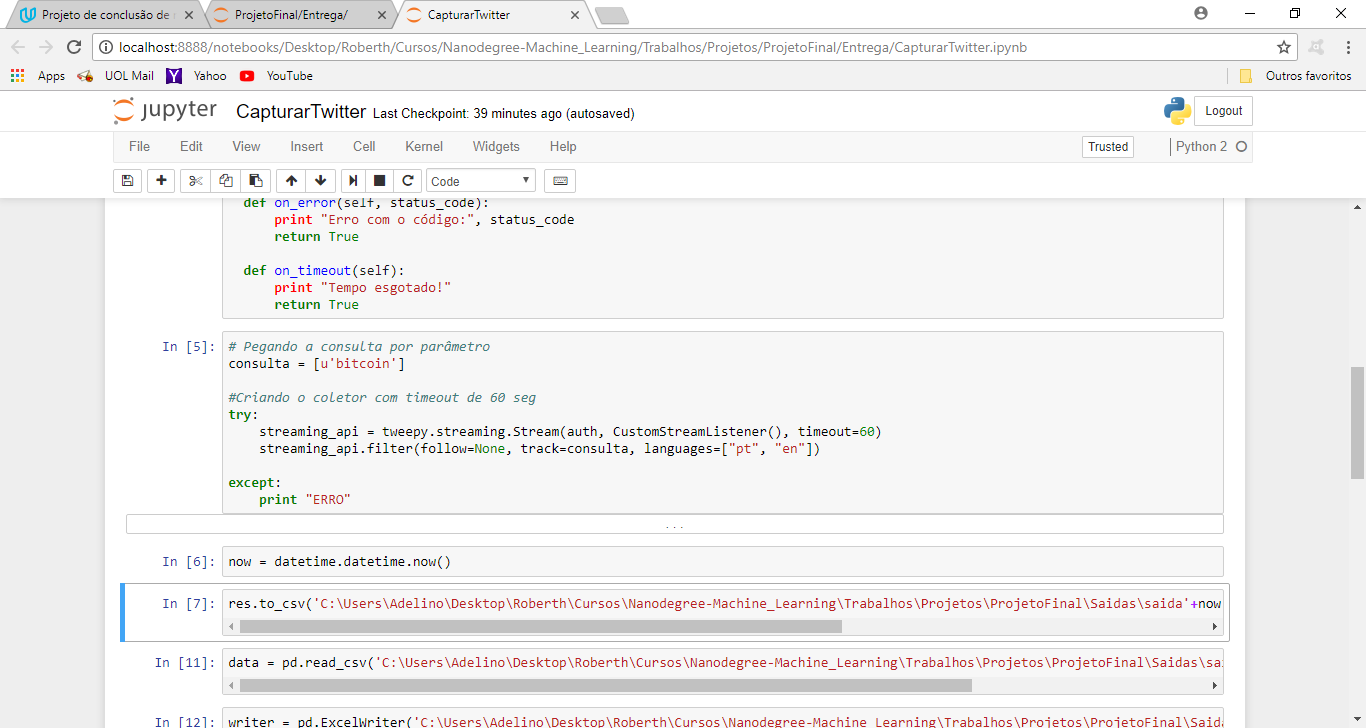
As informações coletadas foram:

* + **USR**
    - Nome do usuário do Twitter que publicou esse Tweet
  + **TWEET**
    - O texto publicado pelo usuário, o qual sempre terá a palavra bitcoin.
  + **LANG**
    - Linguagem em que o Tweet foi publicado
  + **DATE**
    - Data de publicação do Tweet
  + **CLAS**
    - No ato da captura dos tweets todos vieram como Neutro, mas em seguida eu classifiquei alguns como veremos na seção seguinte, sendo: Positiva, para as publicações que me influenciariam a comprar bitcoin: Negativa, para as publicações que me influenciariam a não comprar bitcoin: Neutro, para as publicações que não tem influência na compra de bitcoin.

Para captura dos dados do Twitter utilizei um código em Python criado por mim, mas usando como referência:

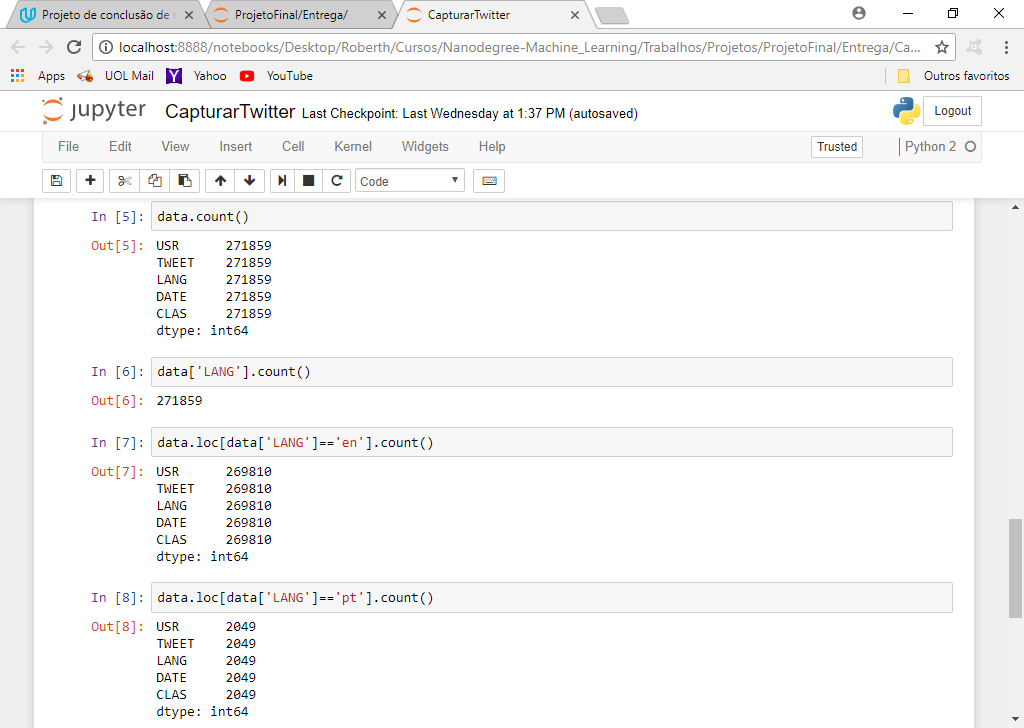
<https://apps.twitter.com/app/14711331/keys>

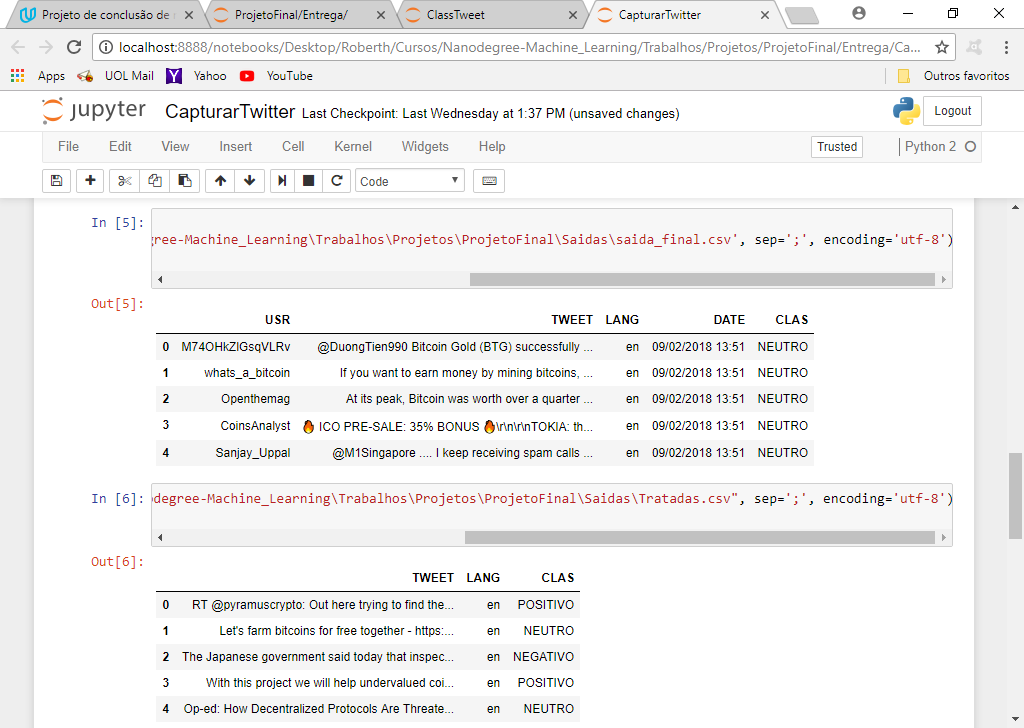
<https://ronanlopes.me/coletor-de-tweets-em-python-com-o-tweepy/>



**Exploração de dados**

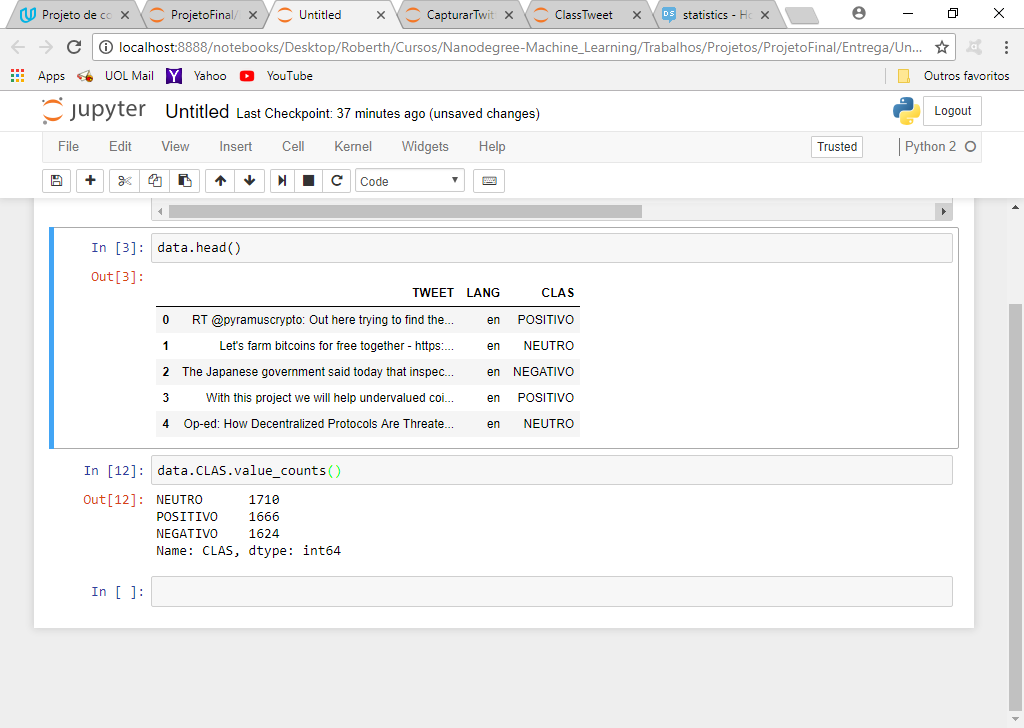
Com do tweets adquiridos passei para a analise do conteúdo. Conforme imagem abaixo, menos de 1% dos tweets eram do idioma português, então decidi abandar os tweets em português e manter apenas os que estavam na língua inglesa.



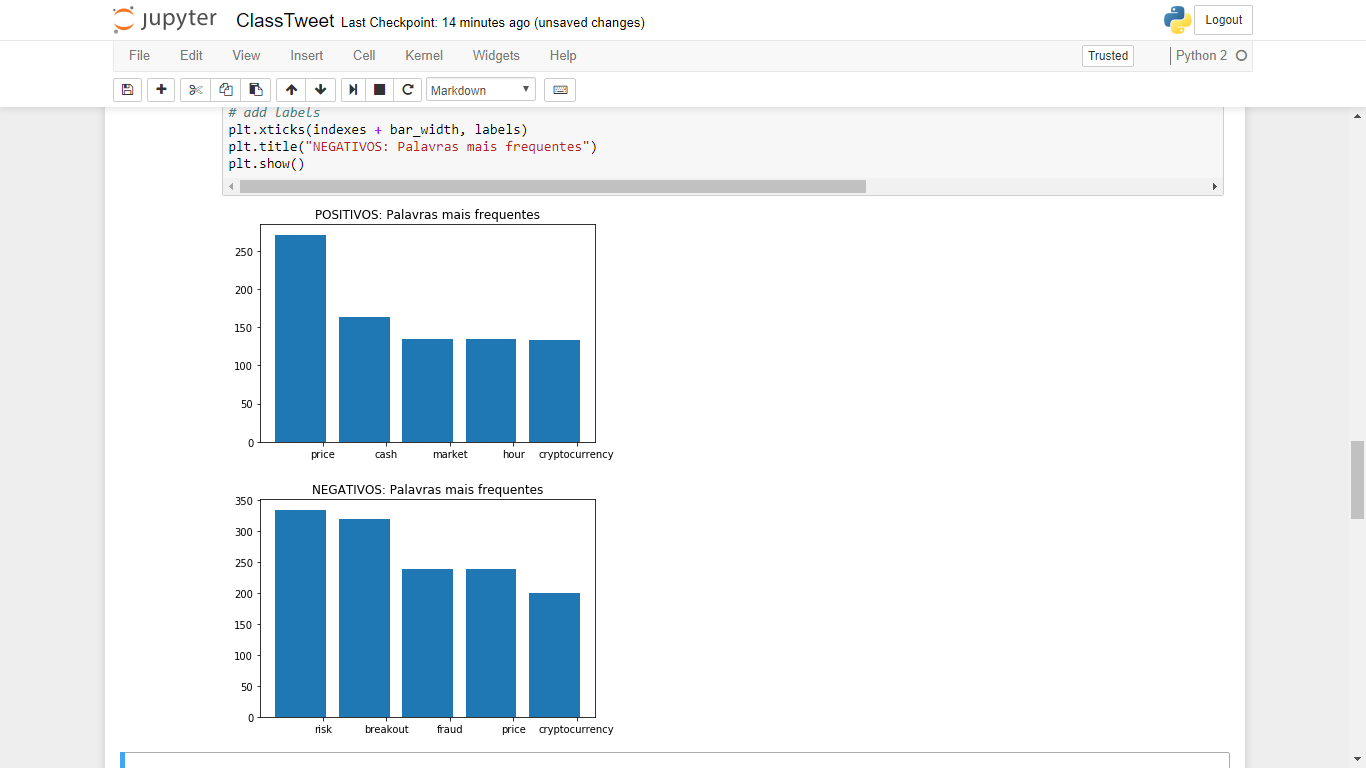
Restando apenas os tweets em inglês passei para a fase de classificação manual dos tweets, classificando-os como Positivo, Negativo e Neutro conforme visto na seção anterior. Entretanto, classificar aproximadamente 200.000 tweets não era viável, então ordenei os tweets de forma aleatória, abandonando assim a data em que o tweet foi postado. Nessa limpeza já removi o usuário que escreveu o tweet, pois para esse projeto o usuário não era necessário. Feito isso fiquei com uma base com apenas tweets em inglês contendo apenas as colunas TWEET, LANG (que não foi utilizada) e CLAS, como podem ser vistas na imagem abaixo:

Após essa alteração no arquivo dos tweets iniciei minha classificação manual seguindo os critérios preestabelecidos, porém após mais de 15.000 tweets classificados identifiquei que a maioria era neutro, tendo uma proporção de 0,6% Negativos, 7% Positivos e 92,4% Neutros. Então decidi continuar a classificação buscando apenas os tweets Negativos e Positivos para chegar em um dataset equilibrado entre os Positivos, Negativos e Neutros.

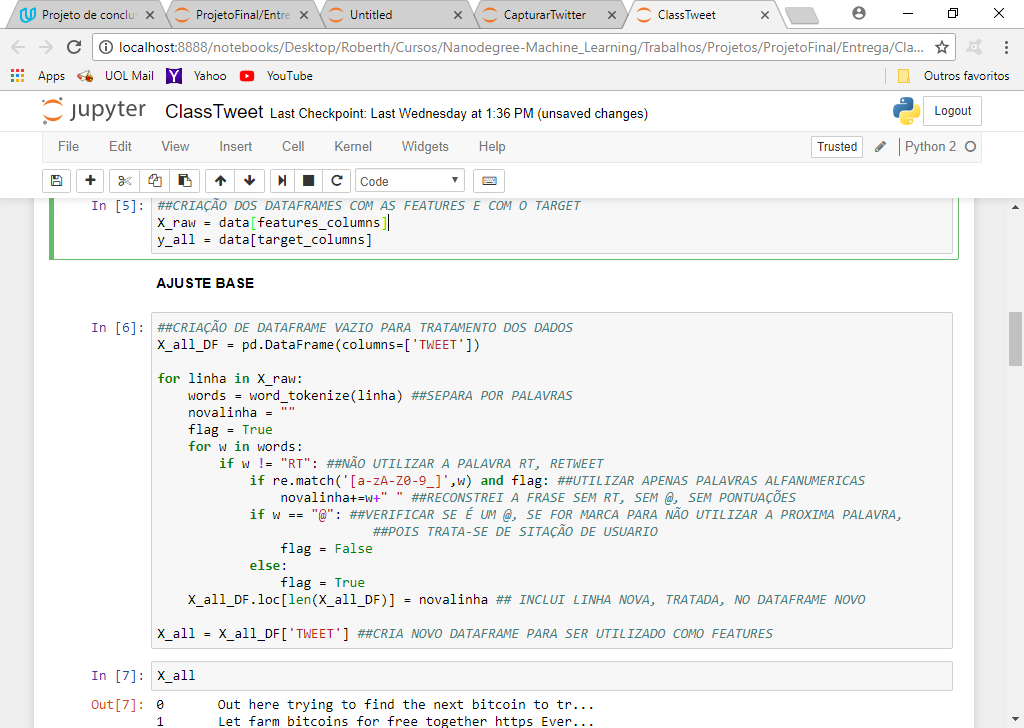
Feito isso cheguei em novo dataset com 5.000 tweets classificados conforme abaixo:



Para efeito exploratório, inclui um histograma de frequência, abaixo, para identificar quais eram as 5 palavras mais utilizadas para os tweets classificados como POSITIVO e as 5 palavras mais utilizadas para os tweets classificados como NEGATIVO.



Dessa forma tenho um dataset balanceado para poder iniciar o processo de criação de uma modelo, porém, antes do inicio da modelagem identifiquei que muitos tweets possuíam palavras que não deveriam ser analisadas, como “RT” (retweet), “emoji”, “@” (antecede o nome de um usuário). Para esses casos eu criei o código abaixo para fazer uma limpeza do texto dos tweets.



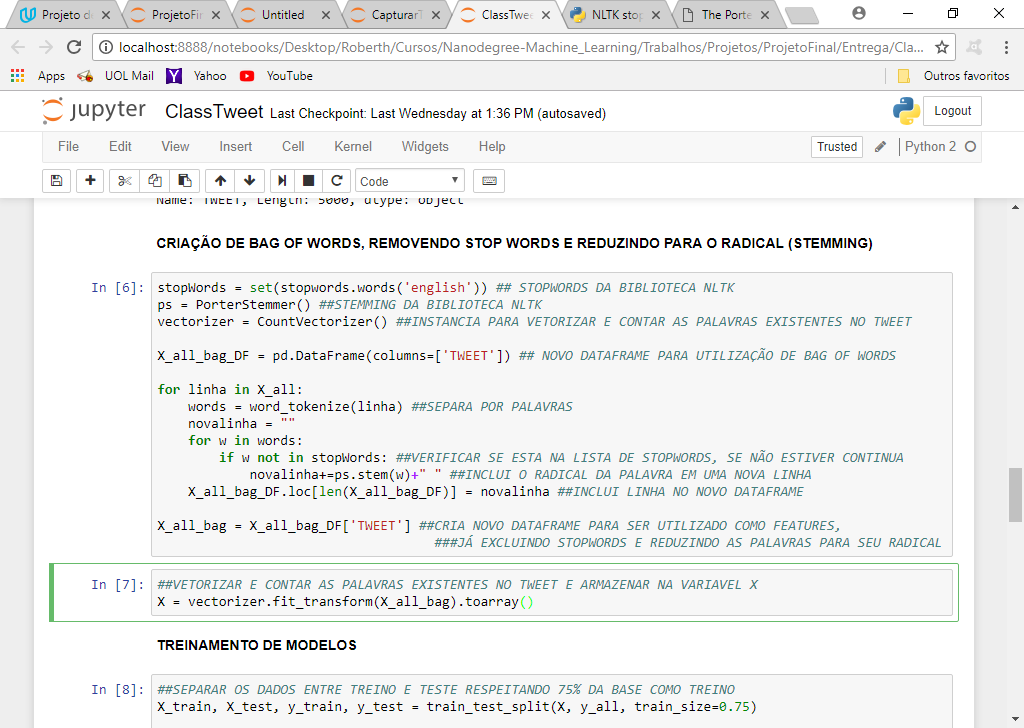
**Algoritmos e Técnicas**

Pela natureza desse projeto, onde gostaria de identificar o sentimento de cada tweet mencionando a palavra bitcoin, algumas técnicas recomendadas são Bag of Words, remoção de Stop Words, Stemming, criação de um Corpus e Word2Vec.

Decidi utilizar a criação de bag of words, onde conteve todas as palavras da minha base de treino vetorizadas para analise de quantidade de palavras em cada tweet. Além de considerar o set de stopwords da biblioteca NLTK (<https://pythonspot.com/nltk-stop-words/>) para filtrar palavras que não possuem peso para a analise de texto, como: the, is, are....ente outras palavras da língua inglesa.

Além de considerar stop words, decidi utilizar a técnica de Stemming, também da biblioteca NLTK (<http://www.nltk.org/howto/stem.html>), para reduzir as palavras para seu radical. Essa técnica permite trabalhar com palavras que possuem o mesmo radical como se fossem a mesma palavra, dessa forma podemos dar mais peso para as palavras, pois estamos considerando todas as formas que essa palavra possa ter aparecido como apenas uma forma.

Para vetorização, remoção de stop words e stemming das palavras eu criei o código abaixo:



Com o dataframe pronto, vetorizado, utilizei um holdout com 75% da base para treino e os outros 25% para teste. Separados dessa forma apliquei 3 algoritmos que são indicados para analise de texto, 2 tipos diferentes de Naive Bayes pois ele trabalha bem com analise de texto exatamente por considerar que cada palavra é um componente independente, além de um Random Forest, pois assim como arvore de decisão o RF é um ótimo classificador.

Para entender melhor a diferença entre os modelos escolhido, seguem algumas referências:

**NAIVE BAYES**

(ref: <https://www.vooo.pro/insights/6-passos-faceis-para-aprender-o-algoritmo-naive-bayes-com-o-codigo-em-python/>

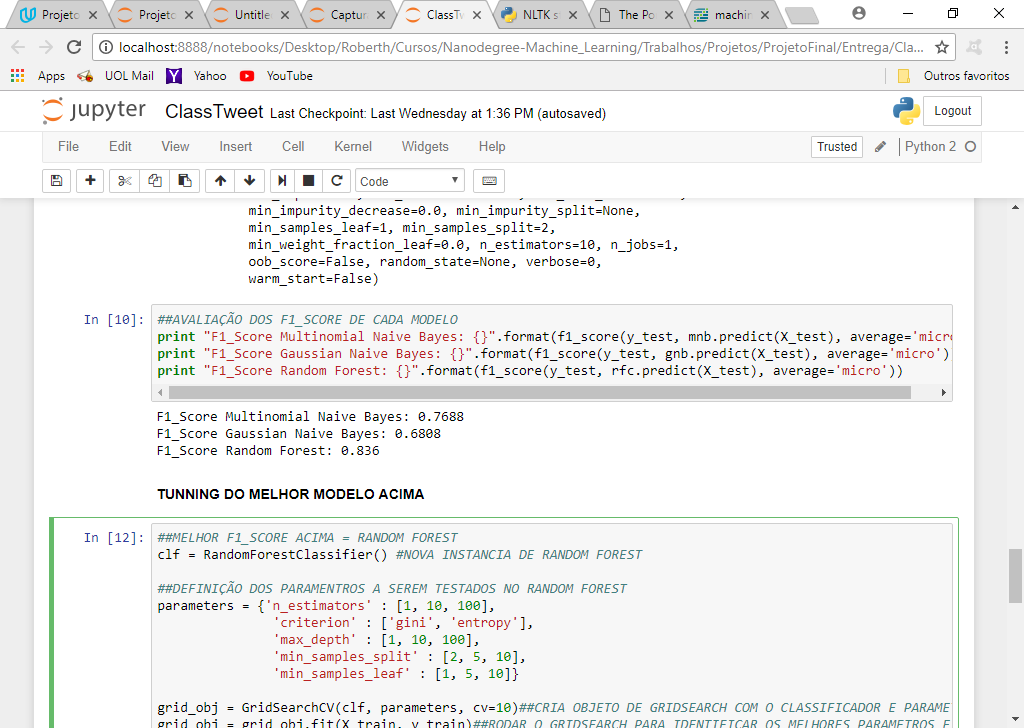
Ref2: <https://stats.stackexchange.com/questions/33185/difference-between-naive-bayes-multinomial-naive-bayes?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>).

* **APLICAÇÃO REAL**: Classificação de textos/Filtragem de spam/Análise de sentimento: classificadores Naive Bayes utilizados principalmente em classificação de textos (devido a um melhor resultado em problemas de classes múltiplas e regra de independência) têm maior taxa de sucesso em comparação com outros algoritmos. Como resultado, é amplamente utilizado na filtragem de spam (identificar spam) e Análise de Sentimento (em análise de mídia social, para identificar sentimentos positivos e negativos dos clientes).
* **PROS**: Quando a suposição de independência prevalece, um classificador Naive Bayes tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento. O desempenho é bom em caso de variáveis categóricas de entrada comparada com a variáveis numéricas. Para variáveis numéricas, assume-se a distribuição normal (curva de sino, que é uma suposição forte).
* **CONTRAS**: Uma limitação do Naive Bayes é a suposição de preditores independentes. Na vida real, é quase impossível que ter um conjunto de indicadores que sejam completamente independentes. Se a variável categórica tem uma categoria (no conjunto de dados de teste) que não foi observada no conjunto de dados de treinamento, então o modelo irá atribuir uma probabilidade de 0 (zero) e não será capaz de fazer uma previsão. Isso é muitas vezes conhecido como “Zero Frequency”. Para resolver isso, podemos usar a técnica de alisamento. Uma das técnicas mais simples de alisamento é a chamada estimativa de Laplace.
* **Diferença em Gausian NB e Multnomial NB**: O termo Multinomial Naive Bayes simplesmente nos permite saber que cada é uma distribuição multinomial, ao invés de alguma outra distribuição. Isso funciona bem para dados que podem ser facilmente convertidos em contagens, como contagens de palavras no texto, o que foi feito nesse projeto.

**RANDOM FOREST**

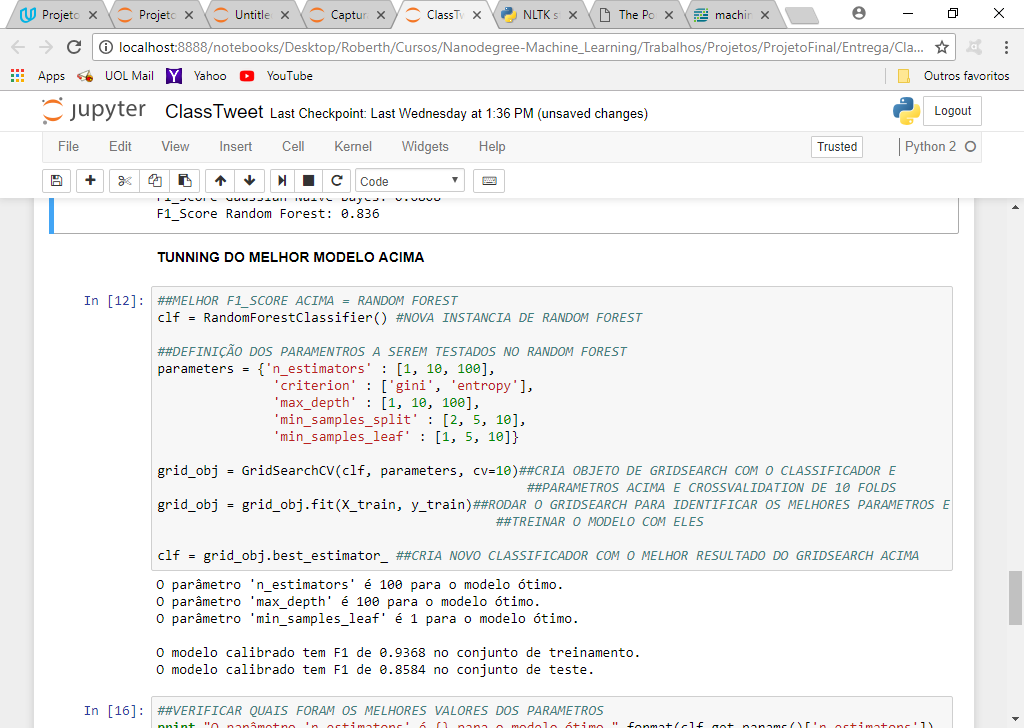
(ref: <https://dimensionless.in/introduction-to-random-forest/>)

* **APLICAÇÃO REAL**: Random Forest são como varias arvores de decisão trabalhando juntas para atingirem o melhor resultado. A escolha do RF para esse projeto vem principalmente por um dos pontos fortes dele, que é ser considerado um dos melhores modelos de decisão existentes. Várias áreas utilizam RF, como: reconhecimento de objetos, analise de sequência de aminoácidos e classificação de galáxias em astronomia.
* **PROS**: É um dos modelos de decisão mais preciso, funciona bem em grandes conjuntos de dados, pode ser usado para extrair as variáveis mais importantes e não requer feature engineering (scaling e normalização).
* **CONTRAS**: Pode dar overfitting em casos de dados ruidosos, ao contrario das arvores de decisão os resultados do RF são difíceis de interpretar e os parâmetros (hiperparametros) precisão de um bom ajuste para ter uma boa precisão, por isso utilizamos com Grid Search

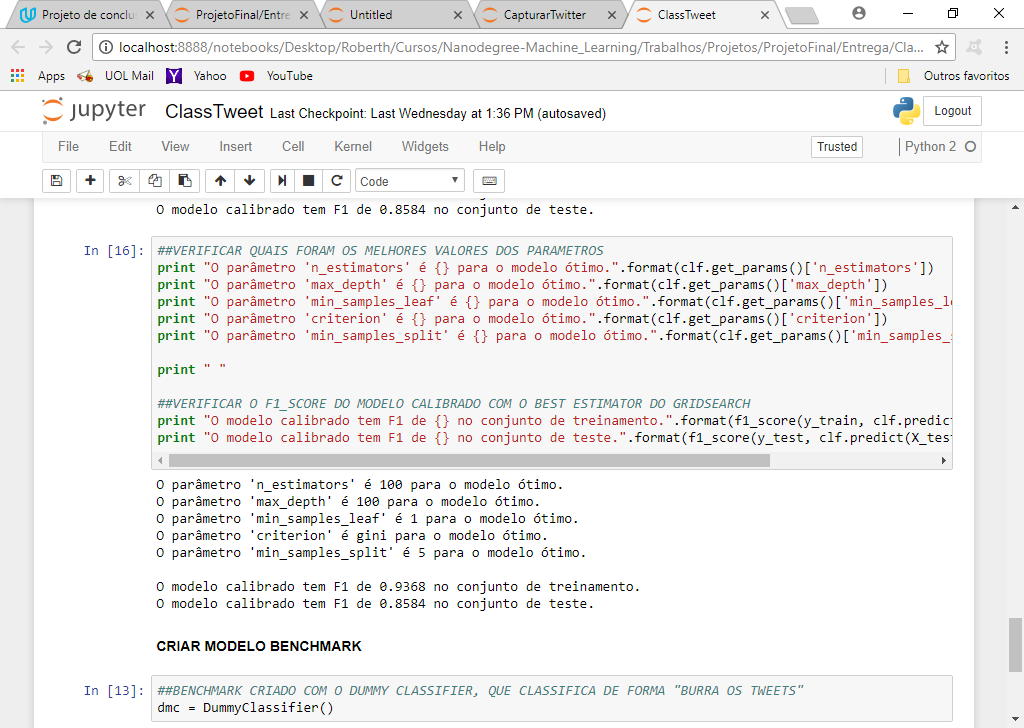
Para analisar qual dos modelos teve um melhor desempenho utilizei o F1\_Score, pois assim consigo mensurar a precisão e o recall do modelo em um indicador único. Tendo como resultado para o melhor modelo o Random Forest, conforme abaixo:

Dado o melhor modelo, considerando o F1\_Score dos dados de teste, decidi tentar melhorar o modelo buscando um melhor F1\_Score atrás do Grid Search, o qual executa o modelo alterando os parâmetros previamente definidos buscando quais são os parâmetros necessários para se alcançar o melhor modelo.

Para o Random Forest escolhi os parâmetros “n\_estimators” (numero de arvores na floresta), “criterion” (forma e qualidade de como será dividido os dados nas arvores), “max\_depth” (profundidade máxima de cada arvore), “min\_samples\_split” (número mínimo de amostras em cada separação), “min\_samples\_leaf” (número mínimo de amostrar para compor uma folha da arvore). Abaixo segue codificação:



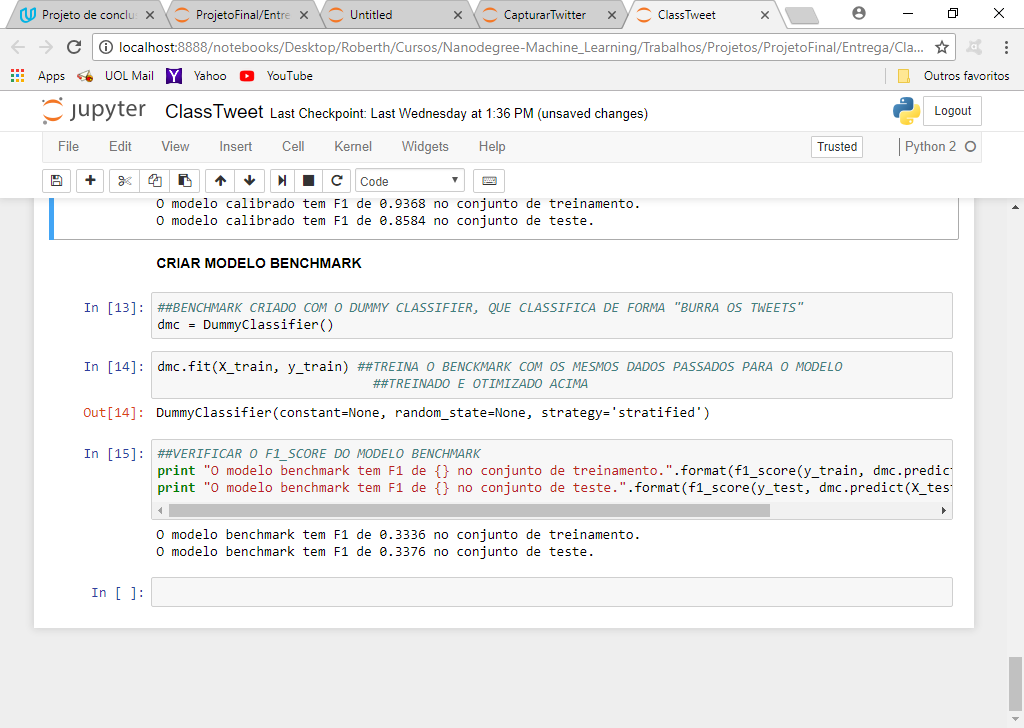
Obtive os seguintes resultados:



Com esses ajustes obtidos no GridSearch temos um modelo com F1\_Score 2% melhor que o modelo anterior. O ganho não foi muito significante, mas como não houve um aumento considerável de tempo de execução considero que esse modelo seja melhor que o anterior. Podemos notar também que o modelo está fit, pois tanto o treino quando o teste estão com F1\_Score próximos e altos, mostrando que o modelo acerta bem para os dados novos e para os dados de treino.

**Modelo de referência**

Após o desenvolvimento desse modelo um modelo de benchmark foi criado para aferir melhor a qualidade do meu modelo. Como benchmark para validar a sanidade do meu modelo final, utilizei uma classificação aleatória, o [DummyClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html) do sklearn (<http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html> ), afim de medir se meu modelo tem resultados melhores ou piores que a classificação aleatória.

 Como podemos ver abaixo, a implementação do modelo benchmark aleatório, os resultados do F1\_Score são bem piores que meu modelo, garantindo assim mais uma vez de que a classificação feita pelo meu modelo está boa.

**Conclusão**

O desenvolvimento desse projeto me fez entender melhor como criar uma analise de sentimentos, mas mesmo tendo bons resultados eu ainda não utilizaria esse modelo no mundo real, pois o mercado de bitcoins é muito volátil. Gostaria de aperfeiçoar esse modelo em um momento futuro aplicando word2vec e outras formas mais sofisticadas de analise de sentimento como redes neurais.