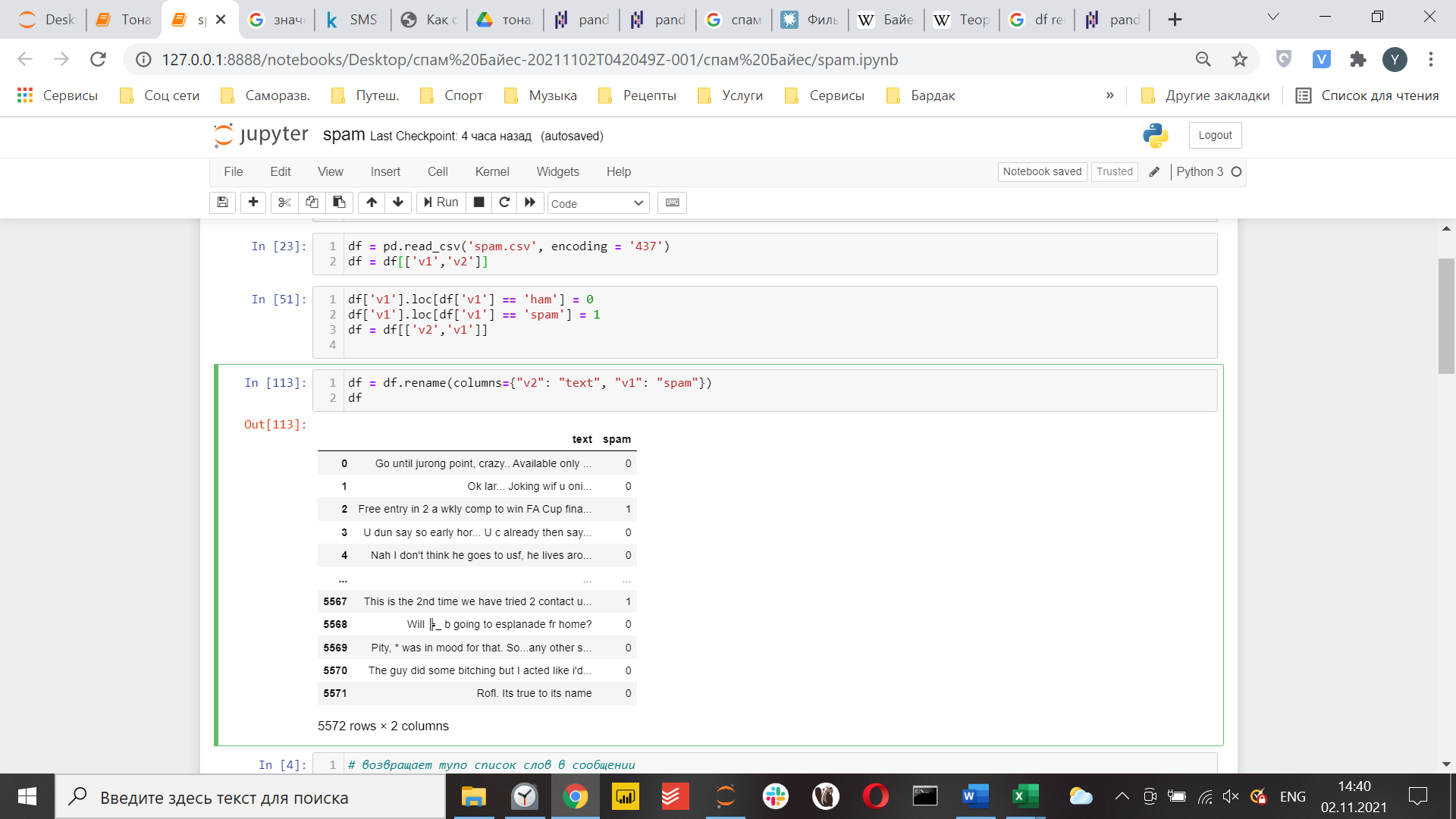
**Наивный Байесовский классификатор для фильтрации спама**

Использование наивного Баейсовского классификатора для фильтрации спама довольно старый и известный способ фильтрации сообщений. В его основе лежит простая идея – на обучающих данных для каждого слова в сообщении вычисляется оценка вероятности того, что письмо с этим словом является спамом и, исходя их этого, для вновь пришедшего письма на основании всех слов, вычисляется его вероятность оказаться спамом. Это модель предполагает серьезное допущение, что вероятность встретить одно слово не зависит от вероятности встретить другое слово в спамном или неспамном сообщении (отсюда и наивность в названии), однако до сих пор такой классификатор может использоваться и показывать хорошие результаты, даже по сравнению с более продвинутыми методами.

Рассмотрим подробнее как данный алгоритм можно пошагово реализовать на python. Для этого возьмем данные с сайта Kaggle (SMS Spam Collection Dataset). В датасете содержится информация о 5572 сообщений, среди которых 13% спама.



Разделим данные на обучающую и тестовую выборки. Сохраним обучающий датасет в переменную df. Для начала напишем простую вспомогательную функцию для того, что бы получить множество всех слов:

import re

def tokenize(message):

message = message.lower()

all\_words = re.findall("[a-z0-9]+", message)

return set(all\_words)

Теперь перейдем непосредственно к алгоритму. Для начала для каждого слова нужно найти оценку вероятности встретить его в спамном или неспамном сообщении. Для этого просто посчитаем сколько раз каждое слово встречается в каждом из типов сообщений. Также напишем функцию под это:

from collections import defaultdict

import pandas as pd

def count\_words(data):

counts = defaultdict(lambda: [0,0])

for index, row in data.iterrows():

for word in tokenize(row['text']):

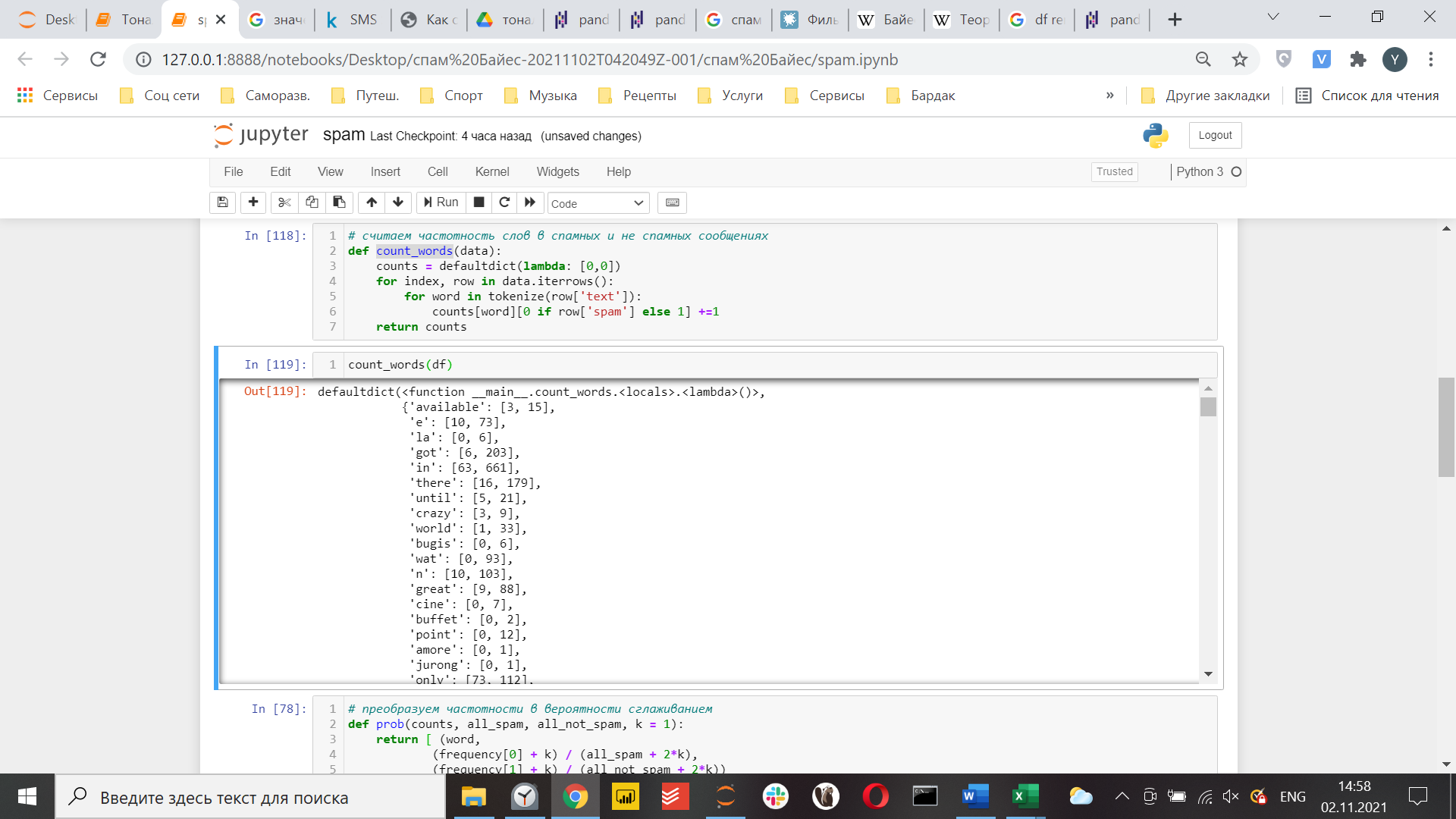
counts[word][0 if row['spam'] else 1] +=1

return counts

Воспользуемся этой функцией и сохраним результат в переменную freq:

freq = count\_words(df)

В результате получаем словарь, следующего вида:



Мы можем видеть каждое слово и сколько раз оно встречается в каждом типе сообщений. Теперь переходим к вычислению веса, то есть нам необходимо поделить конкретную частоту на общее количество сообщений каждого типа. Но здесь есть еще одно допущение. Если какое-то слово присутствует только в сообщениях без спама, то сообщение с этим словом классификатор всегда будет относить к неспамному. Что бы этого избежать при расчете вероятности добавим коэффициент сглаживания равный единицы. Это позволит избежать нулевые вероятности.

Итак идем дальше. Сохраним в переменных общее число сообщений со спамом и без него:

all\_spam = df['spam'].loc[df['spam'] == 1].count()

all\_not\_spam = df['spam'].loc[df['spam'] == 0].count()

Напишем очередную функцию. Она будет возвращать для каждого слова уже посчитанную вероятность:

def prob(freq, all\_spam, all\_not\_spam, k = 1):

return [ (word,

(frequency[0] + k) / (all\_spam + 2\*k),

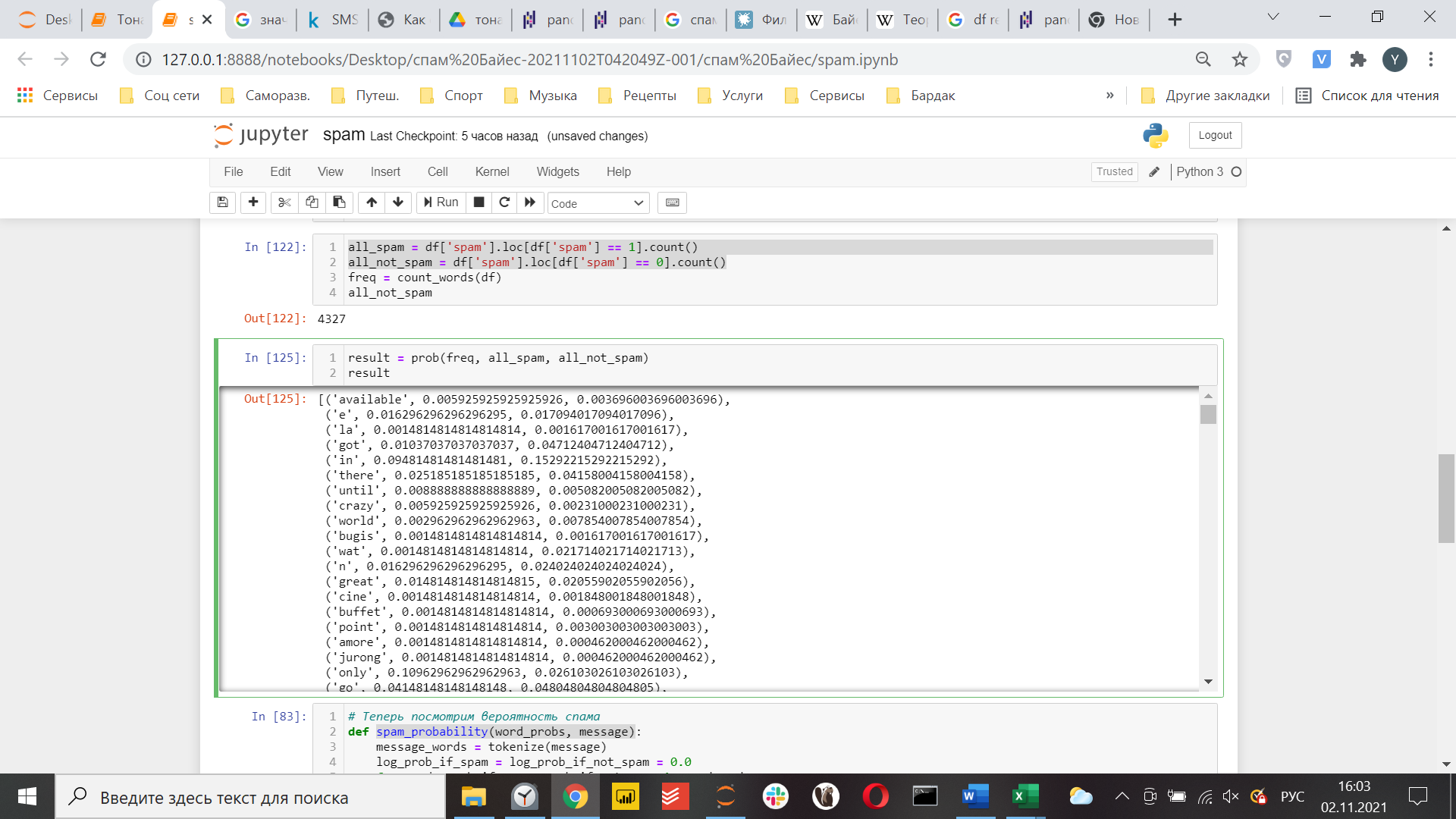
(frequency[1] + k) / (all\_not\_spam + 2\*k))

for word, frequency in counts.items()]

Воспользуемся функцией и сохраним результат в переменной result:

result = prob(freq, all\_spam, all\_not\_spam)

Получим данные следующего вида:



Теперь мы можем посчитать вероятности для сообщений. Также сделаем это с помощью функции. По идее мы должны перемножить частные вероятностные друг на друга. Но так как вероятности могут представлять собой очень маленькие числа, их перемножение может привести к исчезновению разрядов. Поэтому воспользуемся алгебраическими свойствами и заменим действия на сумму логарифмов:

import math

def final(word\_probs, message):

*# Также обрабатываются сообщения*

message\_words = tokenize(message)

*# изначально вероятности спама и неспама равны нулю*

spam\_prob = not\_spam\_prob = 0.0

*# наращиваем вероятности в циклах*

for word, prob\_if\_spam, prob\_if\_not\_spam in word\_probs:

*# если слово в сообщении используем посчитанную вероятность*

if word in message\_words:

spam\_prob += math.log(prob\_if\_spam)

not\_spam\_prob += math.log(prob\_if\_not\_spam)

*# если слова нет в сообщении используем вероятность противоположного события, т.е 1 - посчитанная вероятность*

else:

spam\_prob += math.log(1.0 - prob\_if\_spam)

not\_spam\_prob += math.log(1.0 - prob\_if\_not\_spam)

e\_spam\_prob = math.exp(spam\_prob)

e\_not\_spam\_prob = math.exp(not\_spam\_prob)

*# непосредственно формула вероятность по теореме Баейса*

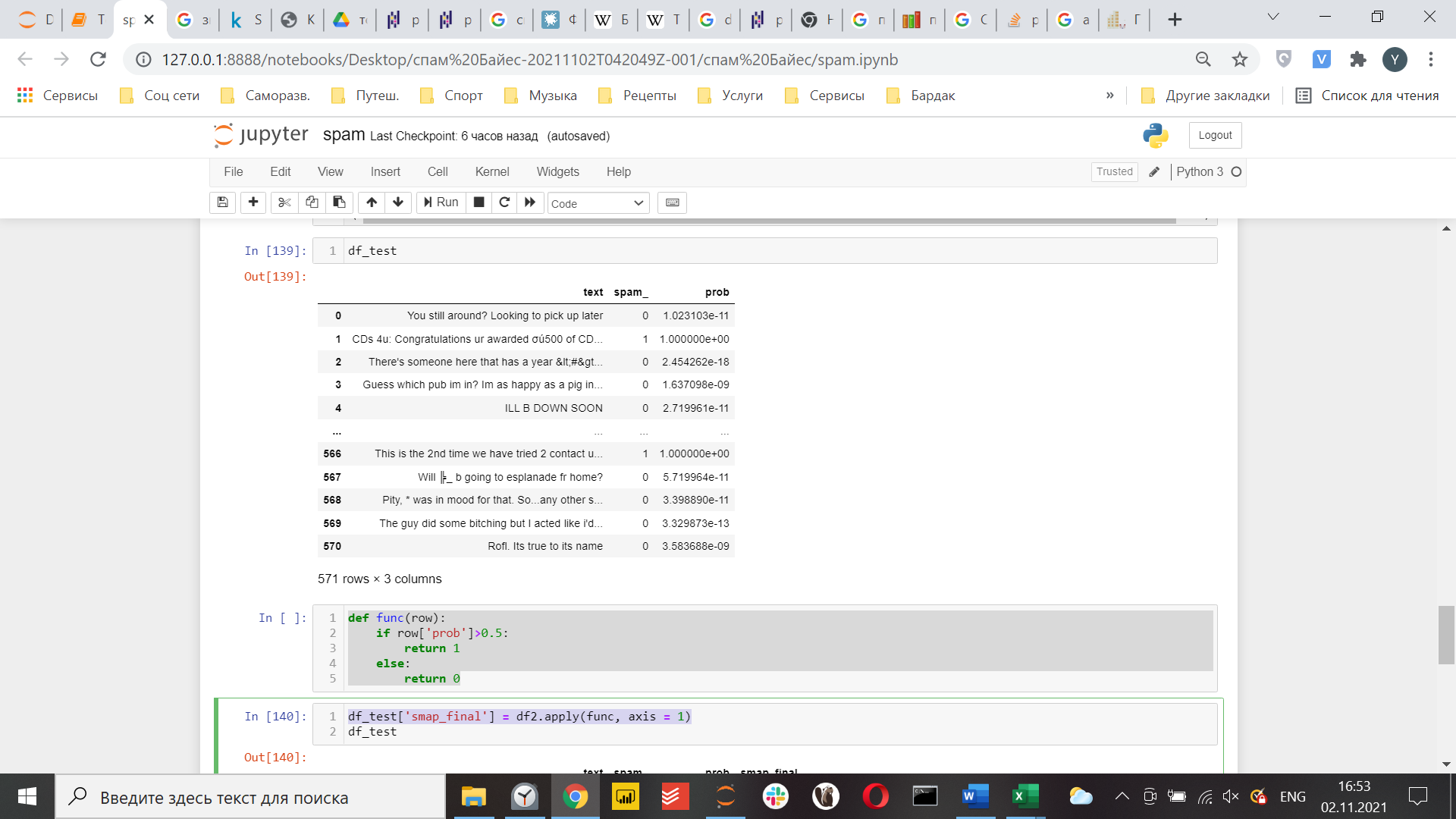
return e\_spam\_prob/(e\_spam\_prob+e\_not\_spam\_prob)

Из тестового набора данных собираем датасет с вероятностями:

for index, row in test.iterrows():

df\_test = df\_test.append({'text': row['text'],'prob': final(result, row['text']), 'spam\_': row['spam']}, ignore\_index=True)

Получаем датасет вида:



Осталось только поставить порог отнесения к классу на вероятность и проверить работу алгоритма на тестовых данных.

def func(row):

if row['prob']>0.5:

return 1

else:

return 0

df\_test['spam\_final'] = df\_test.apply(func, axis = 1)

accuracy\_score(df['spam'], df['spam\_final'])

Выбираем порог произвольно. Для порога в 50% точность составила 76,88%. Попробуем немного сдвинуть. Повторим те же действия для порога в 60% и получим точность уже 98,42%. Получаем очень хороший результат для такой простой модели.