**Steps Involved:**



**1:- Identify Misspelled Word** — A word is misspelled if the text is not found on the vocabulary of the corpus (dictionary), then the autocorrect system flags out for correction.

**2:- Find ‘n’ Strings Edit distance away** — An edit is one of the operations which is performed on a string in order to transform it into another String, and **n** is nothing but the edit distance that is an edit distance like- 1, 2, 3, so on… which will count the number of edit operations that to be performed. Hence, the edit distance n tells us that how many operations are away from one string to another. Following are the different types of edits:-

* + - Insert (will add a letter)
    - Delete (will remove a letter)
    - Switch (it will swap two nearby letters)
    - Replace (exchange one letter to another one)

With these four edits, we are proficient in modifying any string. So the combination of edits allows us to find a list of all possible strings that are n edits to perform.

**IMPORTANT Note**: For autocorrect, we take n  usually between 1 to 3 edits.

**3:- Filtering of Candidates**— Here we want to consider only correctly spelled real words from our generated candidate list so we can compare the words to a known dictionary (like we did in the first step**)** and then filter out the words in our generated candidate list that do not appear in the known “dictionary”.

**4:- Calculate Probabilities of Words**— We can calculate the probabilities of words and then find the most likely word from our generated candidates with our list of actual words. This requires word frequencies that we know and the total number of words in the corpus (also known as dictionary).

**Source Code:**



import re

from collections import Counter

import numpy as np

import pandas as pd

w = []

with open('sample.txt','r',encoding="utf8") as f:

file\_name\_data = f.read()

file\_name\_data = file\_name\_data.lower()

w = re.findall('\w+', file\_name\_data)

v = set(w)

print(f"The first 10 words in our dictionary are: \n{w[0:10]}")

print(f"The dictionary has {len(v)} words ")

def get\_count(words):

word\_count = {}

for word in words:

if word in word\_count:

word\_count[word] += 1

else:

word\_count[word] = 1

return word\_count

word\_count = get\_count(w)

print(f"The dictionary has {len(word\_count)} key values pairs")

def get\_probs(word\_count\_dict):

probs = {}

m = sum(word\_count\_dict.values())

for key in word\_count\_dict.keys():

probs[key] = word\_count\_dict[key] / m

return probs

def DeleteLetter(word):

delete\_list = []

split\_list = []

for i in range(len(word)):

split\_list.append((word[0:i], word[i:]))

for a, b in split\_list:

delete\_list.append(a + b[1:])

return delete\_list

delete\_word\_l = DeleteLetter(word="cans")

def SwitchLetter(word):

split\_l = []

switch\_l = []

for i in range(len(word)):

split\_l.append((word[0:i], word[i:]))

switch\_l = [a + b[1] + b[0] + b[2:] for a, b in split\_l if len(b) >= 2]

return switch\_l



switch\_word\_l = SwitchLetter(word="eta")

def replace\_letter(word):

split\_l = []

replace\_list = []

for i in range(len(word)):

split\_l.append((word[0:i], word[i:]))

alphabets = 'abcdefghijklmnopqrstuvwxyz'

replace\_list = [a + l + (b[1:] if len(b) > 1 else '') for a, b in split\_l if b for l in alphabets]

return replace\_list

replace\_l = replace\_letter(word='can')

def insert\_letter(word):

split\_l = []

insert\_list = []

for i in range(len(word) + 1):

split\_l.append((word[0:i], word[i:]))

letters = 'abcdefghijklmnopqrstuvwxyz'

insert\_list = [a + l + b for a, b in split\_l for l in letters]

return insert\_list

def edit\_one\_letter(word, allow\_switches=True):

edit\_set1 = set()

edit\_set1.update(DeleteLetter(word))

if allow\_switches:

edit\_set1.update(SwitchLetter(word))

edit\_set1.update(replace\_letter(word))

edit\_set1.update(insert\_letter(word))

return edit\_set1

def edit\_two\_letters(word, allow\_switches=True):

edit\_set2 = set()

edit\_one = edit\_one\_letter(word, allow\_switches=allow\_switches)

for w in edit\_one:

if w:

edit\_two = edit\_one\_letter(w, allow\_switches=allow\_switches)

edit\_set2.update(edit\_two)

return edit\_set2

def get\_corrections(word, probs, vocab, n=2):

suggested\_word = []

best\_suggestion = []

suggested\_word = list(

(word in vocab and word) or edit\_one\_letter(word).intersection(vocab) or edit\_two\_letters(word).intersection(

vocab))

best\_suggestion = [[s, probs[s]] for s in list(reversed(suggested\_word))]

return best\_suggestion

my\_word = input("Enter any word: ")

probs = get\_probs(word\_count)

if my\_word in v:

print('Your word seems to be correct')

else:

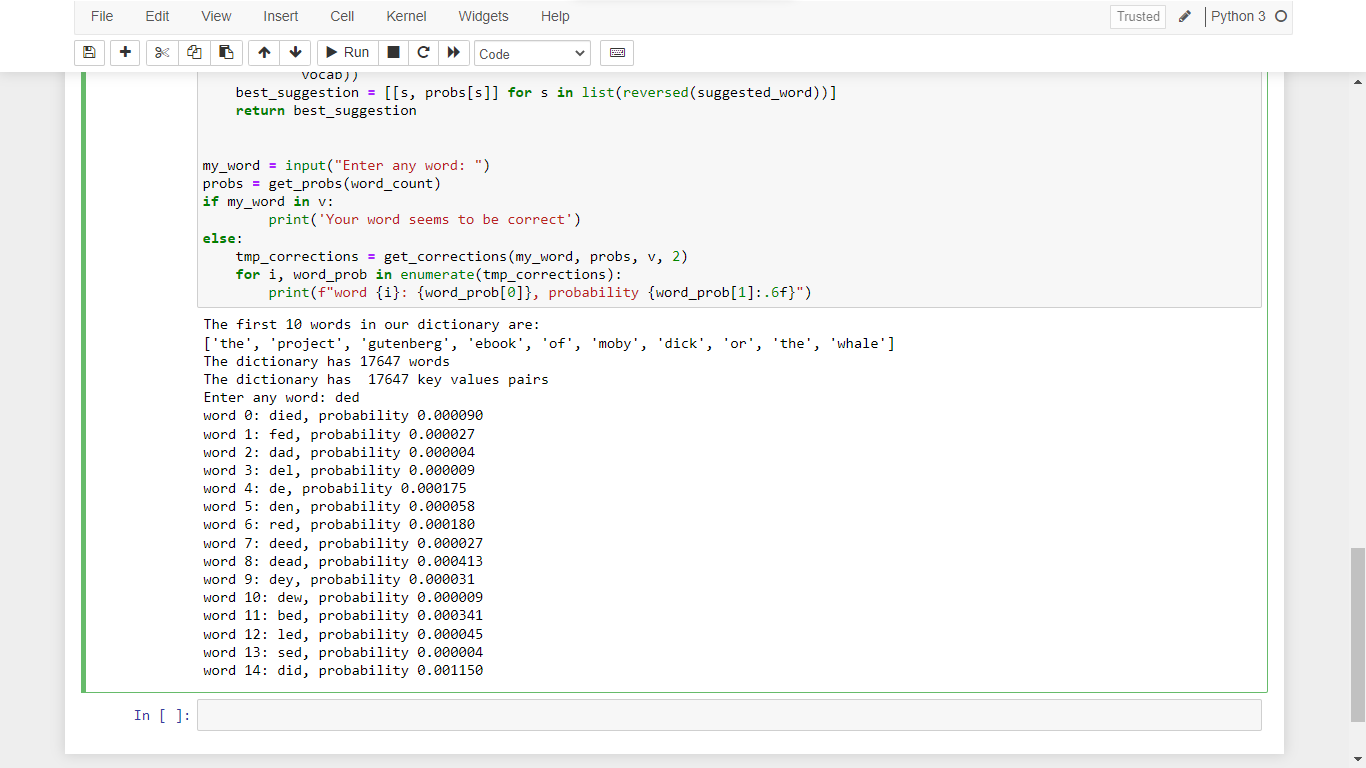
tmp\_corrections = get\_corrections(my\_word, probs, v, 2)

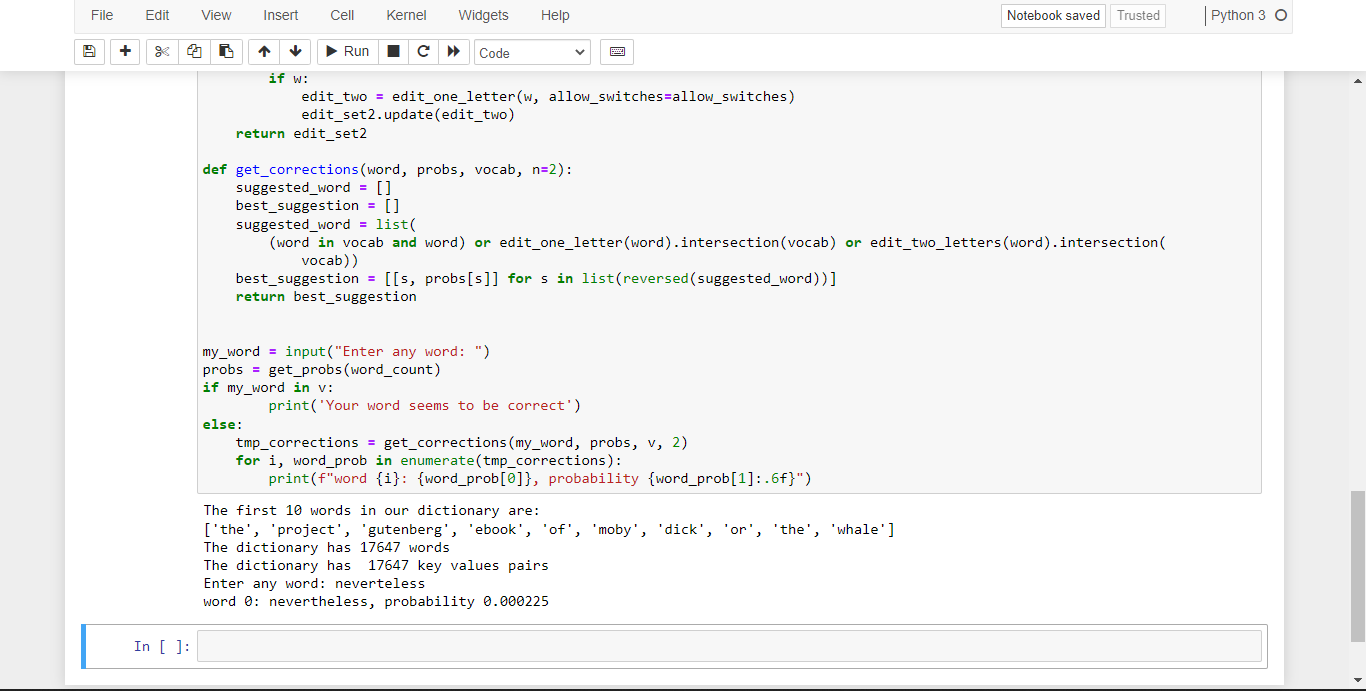
for i, word\_prob in enumerate(tmp\_corrections):

print(f"word {i}: {word\_prob[0]}, probability {word\_prob[1]:.6f}")

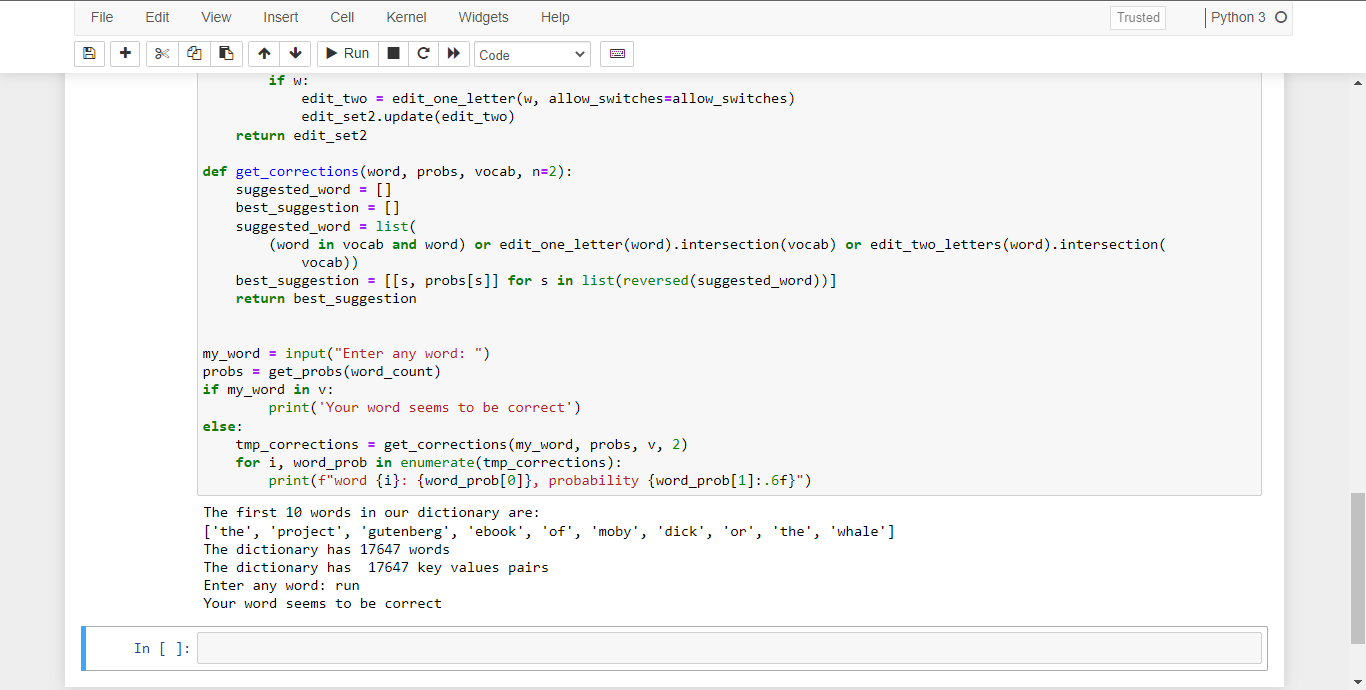
**Screenshots:**











**Conclusion:**

This is how the autocorrect feature using NLP with python works. NLP plays a crucial role in enabling computers to understand and process natural human language. This is as implemented above using the autocorrect system. Here, we have taken words from a book. In the same way, there are some words that are already present in the vocabulary of the smartphone/pc and then some words it records while the user starts typing using the keyboard. This feature can be used to implement in real-time.