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.

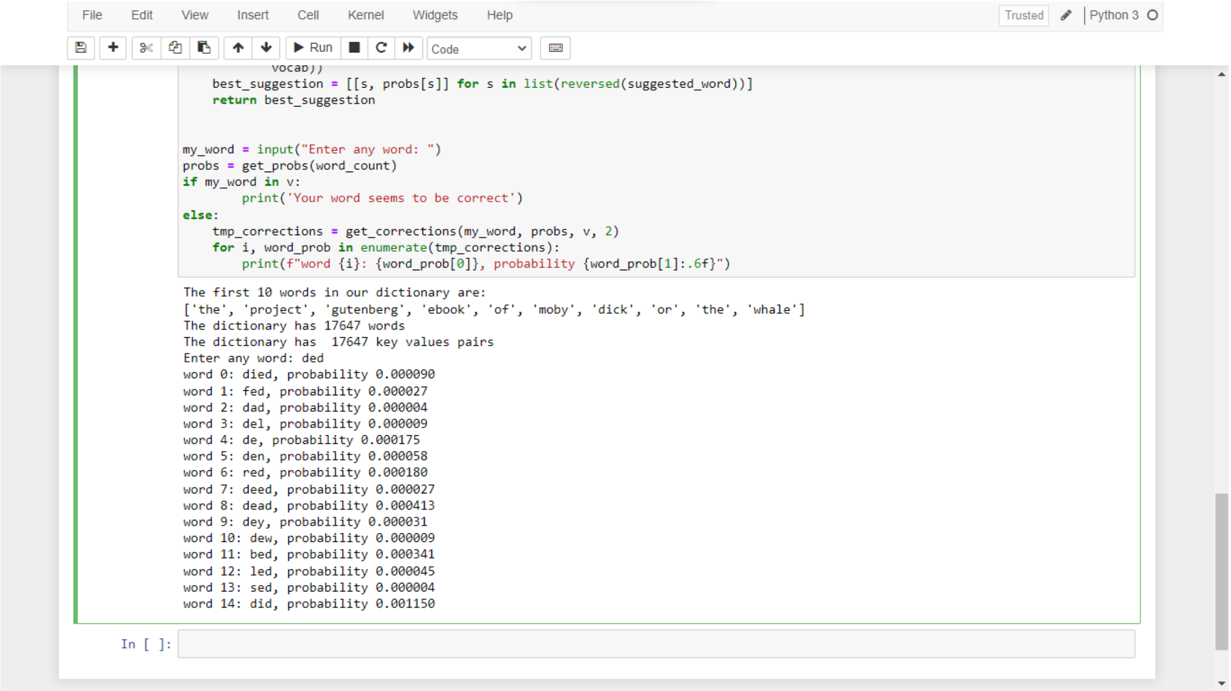
3:- Filtering of Candidates — Here we want to consider only correctly spelled real words from our generated candidate list so we can compare the words to a known dictionary (like we did in the first step**)** and then filter out the words in our generated candidate list that do not appear in the known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

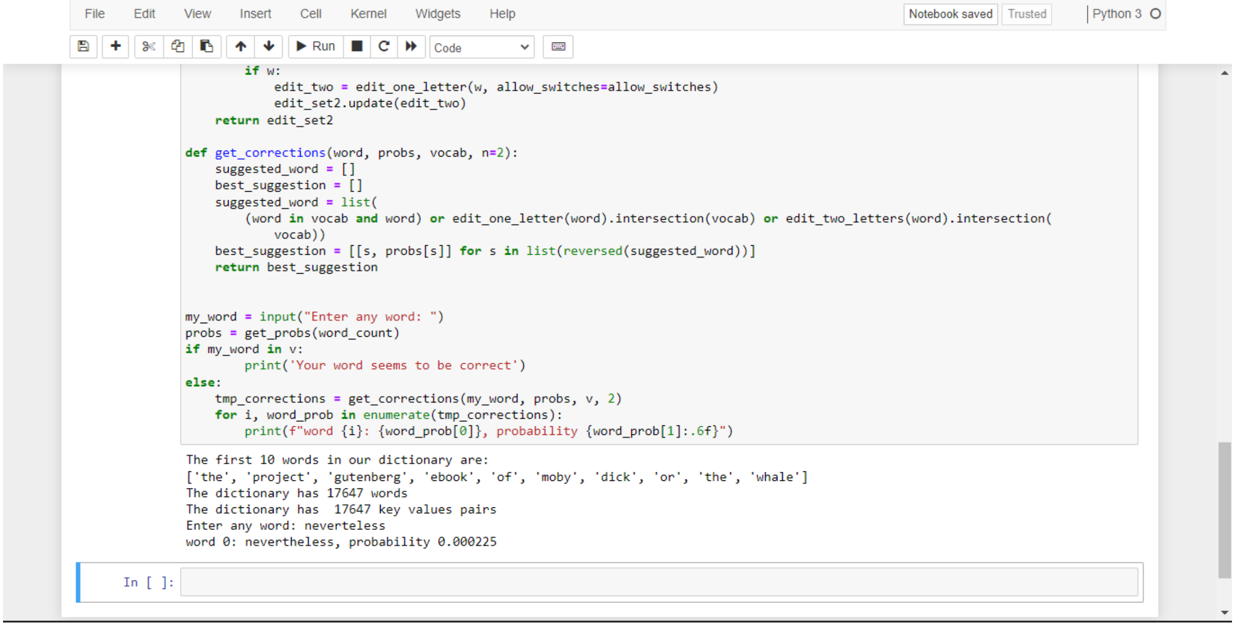
Screenshots:



```
            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}")
```

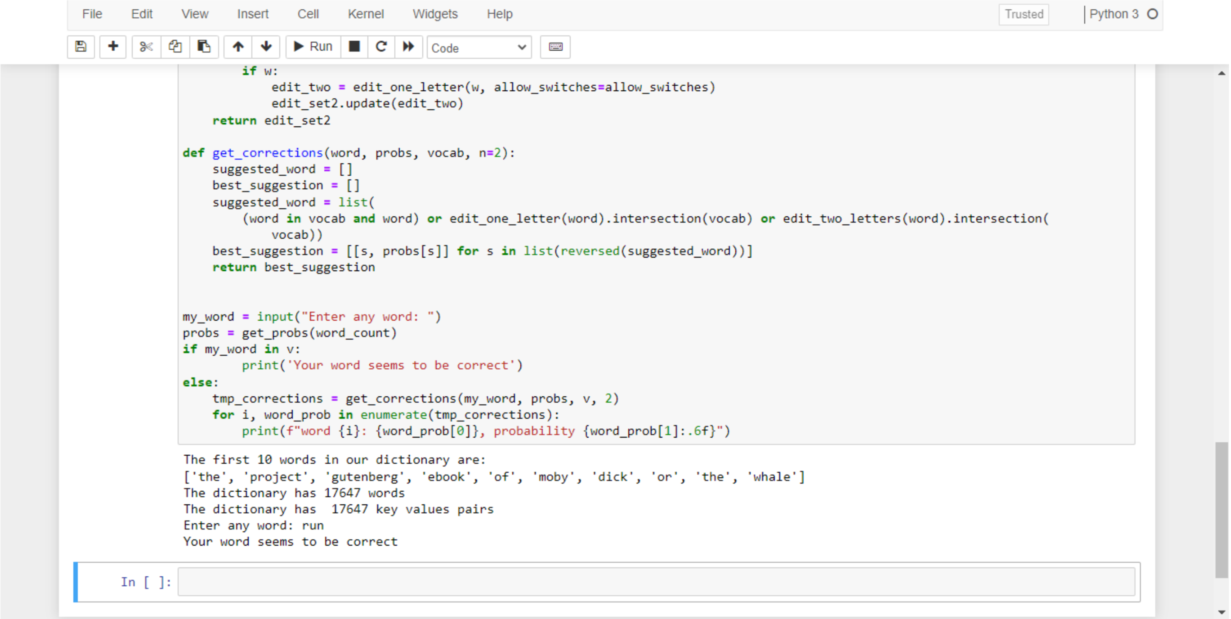
```
The first 10 words in our dictionary are:
['the', 'project', 'gutenberg', 'ebook', 'of', 'moby', 'dick', 'or', 'the', 'whale']
The dictionary has 17647 words
The dictionary has  17647 key values pairs
Enter any word: ded
word 0: died, probability 0.000090
word 1: fed, probability 0.000027
word 2: dad, probability 0.000004
word 3: del, probability 0.000009
word 4: de, probability 0.000175
word 5: den, probability 0.000058
word 6: red, probability 0.000180
word 7: deed, probability 0.000027
word 8: dead, probability 0.000413
word 9: dey, probability 0.000031
word 10: dew, probability 0.000009
word 11: bed, probability 0.000341
word 12: led, probability 0.000045
word 13: sed, probability 0.000004
word 14: did, probability 0.001150
```

```
In [ ]:
```



```
        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}")
```

```
The first 10 words in our dictionary are:
['the', 'project', 'gutenberg', 'ebook', 'of', 'moby', 'dick', 'or', 'the', 'whale']
The dictionary has 17647 words
The dictionary has  17647 key values pairs
Enter any word: neverteless
word 0: nevertheless, probability 0.000225
```



```
        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}")
```

```
The first 10 words in our dictionary are:
['the', 'project', 'gutenberg', 'ebook', 'of', 'moby', 'dick', 'or', 'the', 'whale']
The dictionary has 17647 words
The dictionary has  17647 key values pairs
Enter any word: run
Your word seems to be correct
```

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.