Members: Andrei Shchapaniak (shchaand), Maksim Spiridonov (spirimak)

Members: Andrei Shchapaniak (shchaand), Maksim Spiridonov (spirimak)

`Preparation.`

```
# Representative member: Andrei Shchapaniak, 14.05.2002
'''
K = 14
L = len('Shchapaniak')
X = ((K*L*23) % 20) + 1
Y = ((X + ((K*5 + L*7) % 19)) % 20) + 1
'''
file1 = 003.txt
file2 = 018.txt
'''
```

1. `From both data files, load the texts for analysis.For each text separately, determine the absolute frequencies of individual characters (symbols including space) that occur in the texts. Furthermore, assume that the first text is generated from a homogeneous Markov chain with discrete time.`

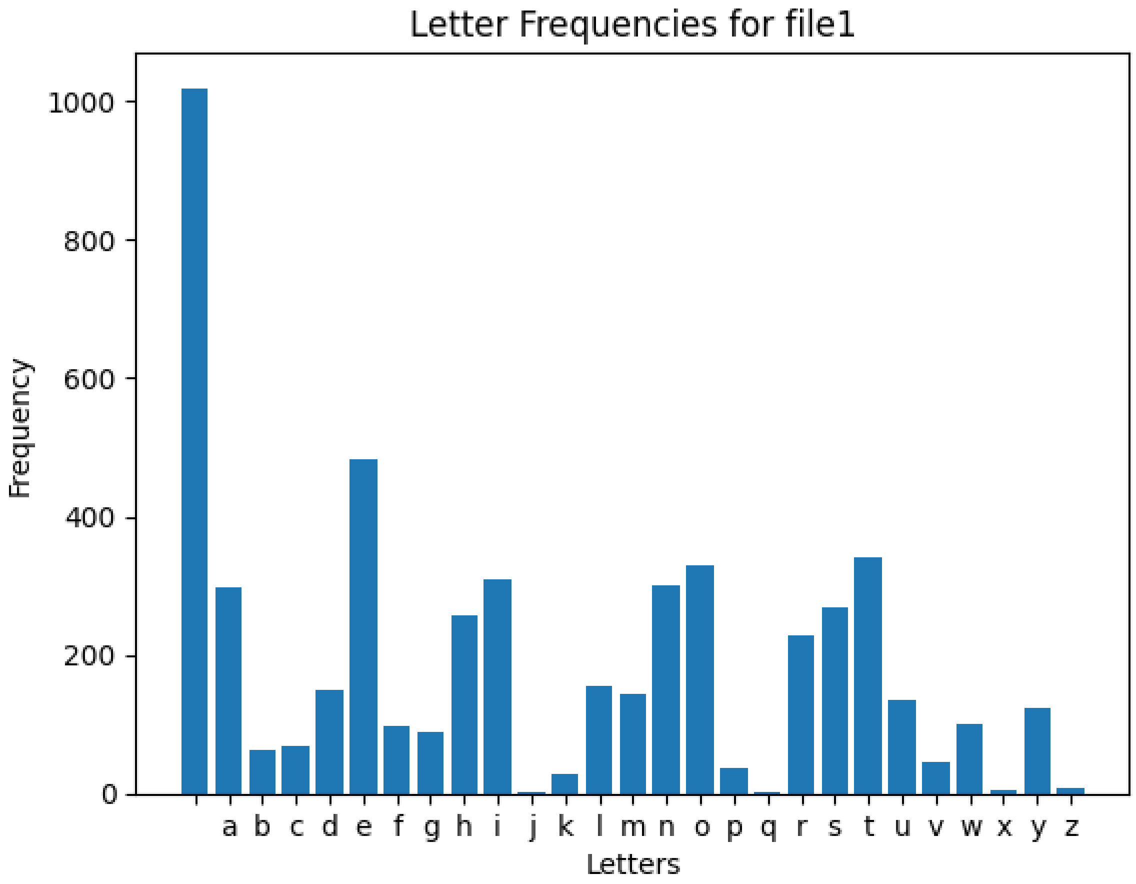
Calculate the frequencies

The function **calc_frequencies** is used to calculate the absolute frequencies of each individual character in each text.

```
def calc_freq(filename):
  with open(os.path.join('files', filename), 'r') as f:
    text = f.readlines()[1].strip()
    letters, letters_counts = np.unique(list(text), return_counts=True)

  return text, dict(zip(letters, letters_counts))
```

The following frequencies have been calculated for **file1**:
' ': 1019,
'a': 296,
'b': 62,
'c': 70,
'd': 151,
'e': 484,

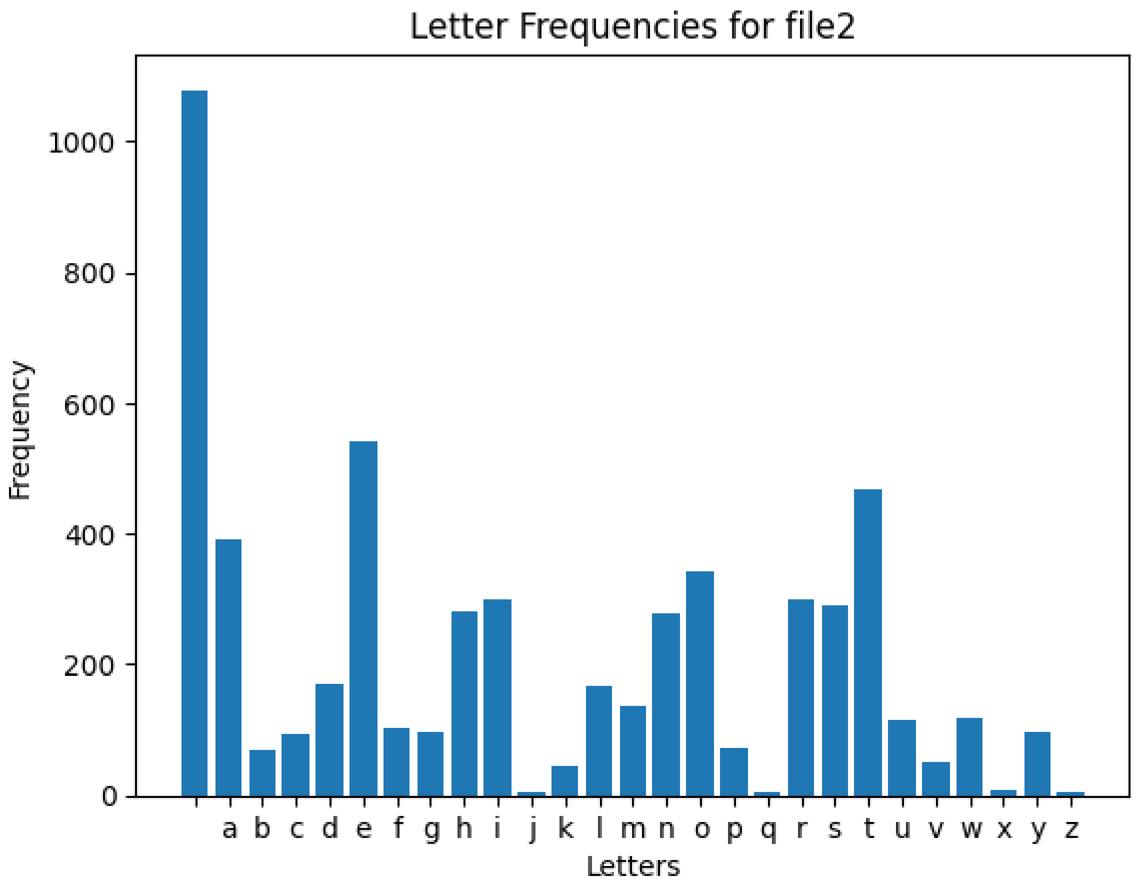
```
'f': 97,
'g': 89,
'h': 258,
'i': 309,
'j': 3,
'k': 29,
'l': 156,
'm': 145,
'n': 299,
'o': 329,
'p': 36,
'q': 2,
'r': 227,
's': 268,
't': 340,
'u': 134,
'v': 45,
'w': 100,
'x': 6,
'y': 123,
'z': 9
```



The following frequencies have been calculated for **file2**:

```
' ': 1079,
'a': 391,
'b': 69,
'c': 94,
'd': 171,
'e': 542,
'f': 104,
'g': 96,
'h': 281,
'i': 300,
'j': 6,
'k': 45,
'l': 169,
'm': 136,
'n': 278,
'o': 341,
'p': 72,
'q': 4,
'r': 299,
's': 290,
't': 468,
'u': 117,
'v': 52,
'w': 118,
'x': 8,
'y': 96,
```

'z': 5



Assuming the above, estimate the transition matrix of the Markov chain for the first text. For the estimation of the transition matrix, see lecture 17. Suitably illustrate the estimated transition probabilities graphically, for example, by using a heatmap. Describe the method by which you estimate the parameters!

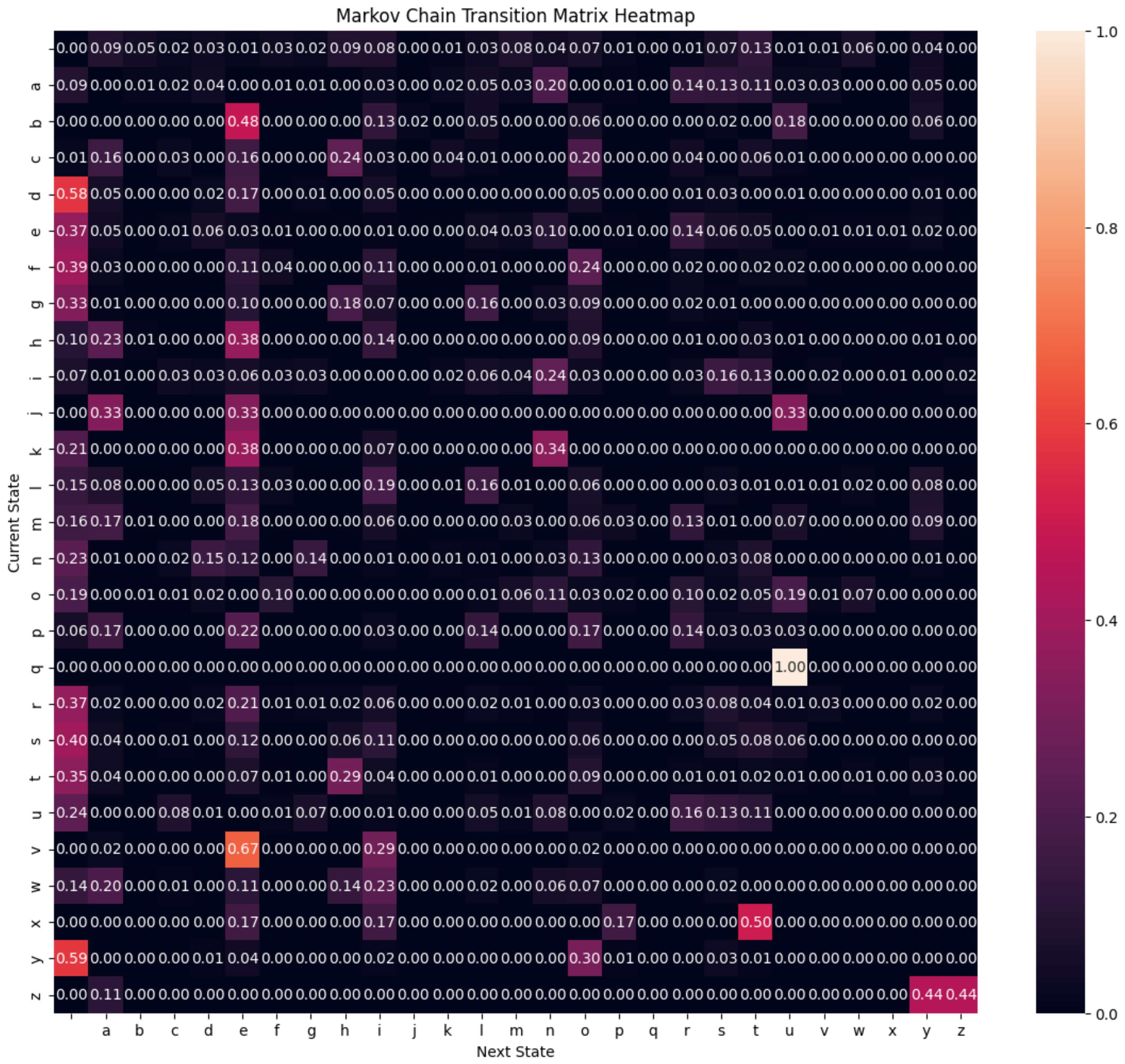
Class **MarkovChain**
Inside the class, the following functions are utilized to perform the needed calculations.

The function **calculate_transition_matrix** is used to estimate the transition matrix.

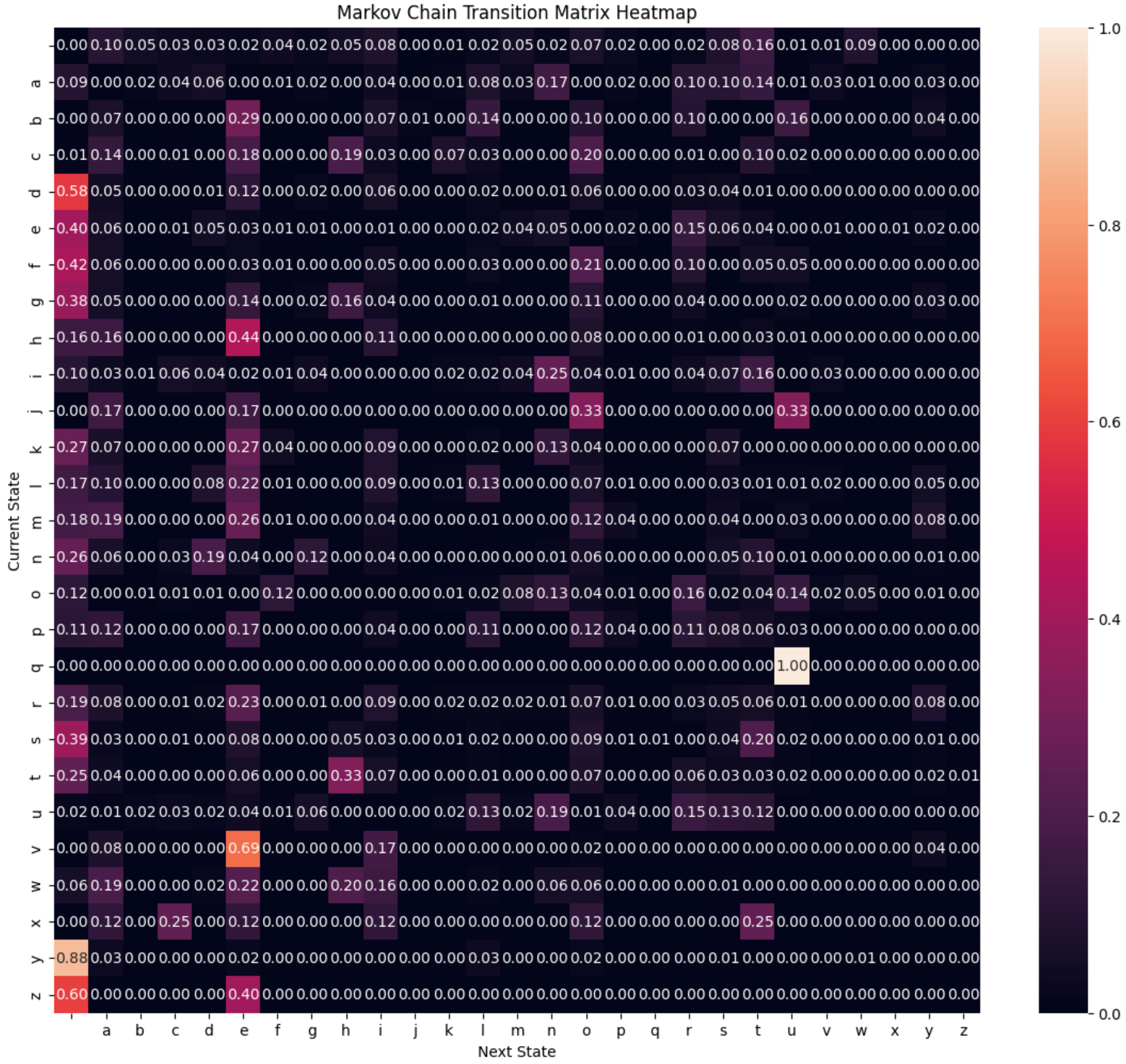
- Transition Matrix Initialization: An empty transition matrix of size (number of states) x (number of states) is created using numpy zeros.
- Transition Counting: For each pair of consecutive characters in the text (excluding the last character), the transition count from the current state to the next state is incremented in the transition matrix. The states are converted to their corresponding indices using the state_index dictionary.
- Normalization: After counting all transitions, each row of the transition matrix is normalized to obtain transition probabilities. This is done by dividing each row by the sum of its elements.
- Heatmap Plotting: Finally, a heatmap plot is generated using seaborn to visually represent the transition matrix. The heatmap provides a graphical representation of the estimated transition probabilities between states.

```
class MarkovChain:
  def __init__(self, text, text_freq):
    self.text = text
    self.text_freq = text_freq
    self.transition_matrix = None
    self.states = None

  def calculate_transition_matrix(self):
    self.states = self.text_freq.keys()
    state_index = {state: i for i, state in enumerate(self.states)}
    self.transition_matrix = np.zeros((len(self.states), len(self.states)),
dtype=float)

    for i in range(len(self.text) - 2):
      current_state = state_index[self.text[i]]
      next_state = state_index[self.text[i+1]]
      self.transition_matrix[current_state, next_state] += 1

    self.transition_matrix = self.transition_matrix /
np.sum(self.transition_matrix, axis=1).reshape(-1, 1)

  def get_transition_matrix(self):
    return self.transition_matrix

  def plot_heatmap(self):
    plt.figure(figsize=(14, 12))
    sns.heatmap(self.transition_matrix, annot=True, fmt=".2f",
                xticklabels=self.states, yticklabels=self.states)
    plt.title('Markov Chain Transition Matrix Heatmap')
    plt.xlabel('Next State')
    plt.ylabel('Current State')
    plt.show()
```

The resuling heatmap for file 1



The resuling heatmap for file 2



2. Based on the matrix from the previous point, find the stationary distribution π of this chain for the first text.Verify that it is indeed a stationary distribution!

Let $\{X_n$ | {n} $\in \mathbb{N}_0\}$ be a homogenous Markov chain with transition matrix P. If there exists a vector π such that: (i) $\forall i \in S$: $\pi_i \geq$ 0, (ii) $\sum_{i\in S} \pi_i$ = 1, we call it the stationary distribution of the chain.
To find π, we need to solve the following equation: $\pi^T * P - \pi^T => \pi^T\ (P - E)$ = 0, where E is the identity matrix and P is the transition matrix $=> (P-E)^T\pi = 0$
In order to find π we need to solve that following equation.

For that we use a function **find_stationary_distribution** that completes the task by finding the null space of a transformed transition matrix, normalizing the resulting distribution, and returning it. This distribution represents the long-term probabilities of being in each state, ensuring that it remains unchanged over successive steps according to the stationary equation. The function also returns the test result of $\pi * P$

```
def find_stationary_distribution(transition_matrix, el_count):
  pi = sp.linalg.null_space((transition_matrix - np.eye(el_count)).T).T[0]
  pi = pi / np.sum(pi)
  return pi, (pi.T@transition_matrix)
```

Resulting stationary distribution π and the test $\pi * P$

```
[0.20043822 0.05821905 0.01219736 0.013766 0.02970201 0.09519371
0.01907474 0.01750049 0.05074636 0.06058887 0.00059013 0.00570491
0.03068281 0.02852009 0.05880308 0.06470688 0.00708284 0.0003934
0.04467082 0.0527118 0.06667719 0.02636176 0.00885396 0.01966934
0.00117889 0.02420055 0.00176472]
```

```
[0.20043822 0.05821905 0.01219736 0.013766 0.02970201 0.09519371
0.01907474 0.01750049 0.05074636 0.06058887 0.00059013 0.00570491
0.03068281 0.02852009 0.05880308 0.06470688 0.00708284 0.0003934
0.04467082 0.0527118 0.06667719 0.02636176 0.00885396 0.01966934
0.00117889 0.02420055 0.00176472]
```

The test confirms, that these are equal.

3. **`Compare the character distribution of the second text with the stationary distribution π, i.e., at a 5% significance level, test the hypothesis that the character distribution of the second text is equal to the distribution π from the previous point. Properly describe the hypothesis you are testing!`**

For that we utilize class **PearsonsTest**.
Inside of it we have the following functions:

- **normalize_table** method is implemented to ensure that all expected frequencies in the chi-squared test are greater than or equal to 5. This is a requirement for the validity of the test.
- **test** method conducts the chi-squared test. It calculates the expected frequencies based on the stationary distribution π and compares them with the observed frequencies in the second text. The test χ^2 is asymptotic and this is why we can only use it for a sufficiently large sample size n. It is usually stated that it must hold true for every $n * p_i \geq 5$ for each i

```
# Pearson's chi-squared test
class PearsonsTest:
    def __init__(self, text2_freq, stat_distr1, text2_len):
      self.stat_distr1 = stat_distr1
      self.text2_freq = text2_freq
      self.text2_len = text2_len
      self.sets_array = []

    def normalize_table(self):
      while any(expected_freq < 5 for _, expected_freq, _ in self.sets_array):
        sorted_array = sorted(self.sets_array, key=lambda x: x[1])
        smallest = sorted_array[:2]

        combined_letter = smallest[0][0] + smallest[1][0]
        combined_expected = smallest[0][1] + smallest[1][1]
        combined_actual = smallest[0][2] + smallest[1][2]

        self.sets_array.append((combined_letter, combined_expected, combined_actual))

        self.sets_array.remove(smallest[0])
        self.sets_array.remove(smallest[1])

      print(self.sets_array)

    def test(self, alpha, normalize_flag):
      for (letter, freq), distr_prob in zip(self.text2_freq.items(), self.stat_distr1):
        expected_freq = self.text2_len * distr_prob
        self.sets_array.append((letter, expected_freq, freq))

      print(self.sets_array)
      if normalize_flag:
        self.normalize_table()

      test_val = 0.0
      for _, np, N in self.sets_array:
        test_val += (N - np)**2 / np

      ddf = len(self.sets_array) - 1
      critical_value= sp.stats.chi2.isf(alpha, ddf)
      p_value = sp.stats.chi2.sf(test_val, ddf)
      print(f'Manual values: ddf: {ddf}, p_value: {p_value}, test_val: {test_val}, critical_value: {critical_value}')

      print(sp.stats.chisquare([s[2] for s in self.sets_array], [s[1] for s in self.sets_array]))
```

The obtained data is the following:

```
(' ', 1128.6676345399371, 1079)
('a', 327.83147237840427, 391)
('b', 68.68331555302264, 69)
```

```
('c', 77.51637107570592, 94)
('d', 167.25202502680017, 171)
('e', 536.0357637001385, 542)
('f', 107.4098593758746, 104)
('g', 98.54527108361391, 96)
('h', 285.75274243912924, 281)
('i', 341.1759353832927, 300)
('j', 3.3230410147753924, 6)
('k', 32.12437357285711, 45)
('l', 172.77487813375745, 169)
('m', 160.59665396197138, 136)
('n', 331.120148887828, 278)
('o', 364.3644567841671, 341)
('p', 39.8834802385281, 72)
('q', 2.2152456026299707, 4)
('r', 251.5413882478122, 299)
('s', 296.82012751482995, 290)
('t', 375.459242780203, 468)
('u', 148.44306543265208, 117)
('v', 49.85666011677329, 52)
('w', 110.75807271359079, 118)
('x', 6.638306115330696, 8)
('y', 136.2733051598708, 96)
('z', 9.937163166503735, 5)
```

As the result contains values, which are less than 5, we need to normalize it as following:

```
(' ', 1128.6676345399371, 1079)
('a', 327.83147237840427, 391)
('b', 68.68331555302264, 69)
('c', 77.51637107570592, 94)
('d', 167.25202502680017, 171)
('e', 536.0357637001385, 542)
('f', 107.4098593758746, 104)
('g', 98.54527108361391, 96)
('h', 285.75274243912924, 281)
('i', 341.1759353832927, 300)
('k', 32.12437357285711, 45)
('l', 172.77487813375745, 169)
('m', 160.59665396197138, 136)
('n', 331.120148887828, 278)
('o', 364.3644567841671, 341)
('p', 39.8834802385281, 72)
('r', 251.5413882478122, 299)
```

```
('s', 296.82012751482995, 290)
('t', 375.459242780203, 468)
('u', 148.44306543265208, 117)
('v', 49.85666011677329, 52)
('w', 110.75807271359079, 118)
('x', 6.638306115330696, 8)
('y', 136.2733051598708, 96)
('z', 9.937163166503735, 5)
('qj', 5.5382866174053635, 10)
```

Manual values:

- Degrees of freedom: 25
- p_value: 2.343262721024455e-15
- Test statistic: 125.5035984454735
- Critical value: 37.65248413348277

As a result, **125.50** $\geq$ **37.65**, so null hypothesis H_0 (The character distribution of the second text equals the stationary distribution π) is hereby rejected in favor of the alternative hypothesis H_A (The character distribution of the second text does not equal the stationary distribution π) at the at the 5% significance level.

- Test statistic=125.50359844547351
- p_value=2.343262721024438e-15

As the resulting $p-value$ is less, than 0.05, we can thereby reject the null hypothesis H_0 in favor of the alternative hypothesis H_A at the at the 5% significance level.