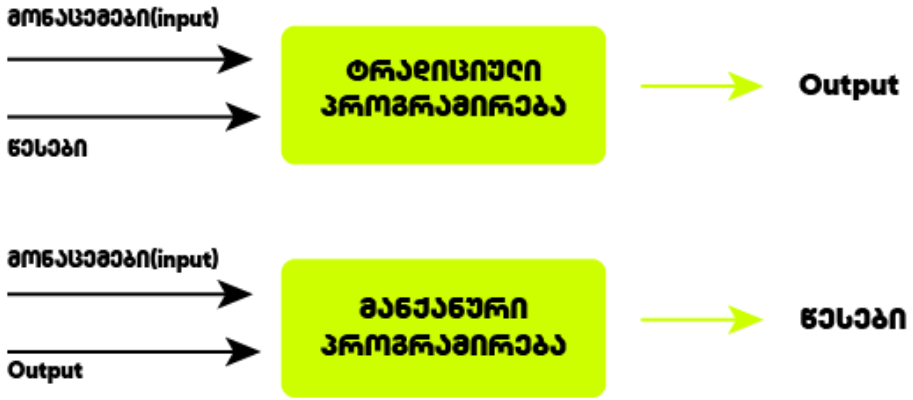
შეხვედრა 8: შესავალი მანქანურ სწავლებაში (Machine Learning)

წინა შეხვედრების განმავლობაში ჩვენ ვისწავლეთ, თუ როგორ დავწეროთ წესები ჩვენი პროგრამისთვის: „თუ მომხმარებელმა გვითხრა ეს, მაშინ ასეთი პასუხი დაუბრუნე“. დღეს კი ახალ, ბევრად უფრო ძლიერ კონცეფციას, **მანქანურ სწავლებას (Machine Learning)**, გავეცნობით. მანქანური სწავლება ხელოვნური ინტელექტის ის ნაწილია, რომელიც პროგრამებს შესაძლებლობას აძლევს, დამოუკიდებლად ისწავლონ მონაცემების ანალიზის შედეგად. ეს არის პროცესი, სადაც კომპიუტერი იღებს დიდი რაოდენობის მონაცემს და თავად პოულობს მათში კანონზომიერებებს. კანონზომიერებებს, რომლის საფუძველზეც თავად გამოაქვს დასკვნები. თავად იღებს გადაწყვეტილებებს ჩვენ მიერ გაწერილი წესების გარეშე. ამ შეხვედრის განმავლობაში შენ მიიღებ ინფორმაციას მანქანური სწავლების ძირითადი ტიპების შესახებ. დაწერ პროგრამას, რომელიც დაალაგებს და დაამუშავებს მონაცემებს მანქანური სწავლების შესაბამისად.

1. რა არის მანქანური სწავლება?

ტრადიციულ პროგრამირებაში ჩვენ, პროგრამისტები, კომპიუტერს ვწერდით მკაფიო და დეტალურ ინსტრუქციებს (წესებს). თავის მხრივ, კომპიუტერი ამ წესების შესაბამისად მოქმედებდა. მანქანურ სწავლებაში კი მიდგომა იცვლება: ჩვენ უკვე აღარ ვეუბნებით კომპიუტერს „როგორ“ შეასრულოს ესა თუ ის სამუშაო. ამის ნაცვლად, ვაწვდით უამრავ მონაცემს და ვეუბნებით „რა“ შედეგს ველოდებით. კომპიუტერი კი თავად სწავლობს, როგორ მიაღწიოს ამ შედეგს. მაგალითად, მასწავლებელმა რომ არ ანახოს მოსწავლეს დავალების შესრულების გზა, მხოლოდ დავალების პირობა გაუზიაროს და სთხოვოს, რომ მოსწავლემ თავად მოიძიოს საჭირო ინფორმაცია, თავისი ცოდნის გამოყენებით, თავად შეძლოს დავალების ამოხსნა.



1.1. განსხვავება ტრადიციულ პროგრამირებასა და ML-ს შორის

ტრადიციული პროგრამირება არის დედუქციური აზროვნება: ზოგადი წესებიდან კონკრეტული დასკვნების გამოტანა. მანქანური სწავლება კი ინდუქციურია: კონკრეტული მაგალითებიდან ზოგადი წესების აღმოჩენა.

1.1.1. ტრადიციული პროგრამირება

წარმოიდგინე, რომ პროგრამა უნდა დაწერო, რომელიც ფოტოსურათზე ძაღლს ამოიცნობს. ტრადიციულ პროგრამირებაში შენ უნდა დაწერო უამრავი კონკრეტული წესი:

თუ ოთხი ფეხი აქვს,

თუ თვალის ფორმა არის ასეთი,

თუ კუდის სიგრძე არის ისეთი,

თუ ცხვირის ფორმა წააგავს ამას,

მაშინ ეს ძაღლია.

ასეთი წესების დაწერა ძალიან რთულია, განსაკუთრებით მაშინ, როცა საქმე ბევრ განსხვავებულ ნიუანსს ეხება.

1.1.2. მანქანური სწავლება (Machine Learning)

მანქანური სწავლებისას შენ არ წერ წესებს. შენ აწვდი პროგრამას უამრავ მონაცემს (ფოტოსურათებს) ეტიკეტთან (Label) ერთად, სადაც თითოეულ სურათს აქვს ეტიკეტი რომ „ეს ძაღლია“ ან „ეს ძაღლი არ არის“. პროგრამა დამოუკიდებლად სწავლობს ამ მონაცემებიდან და პოულობს ისეთ კანონზომიერებებს (patterns), რასაც შენ ვერ დაინახავდი. ამის შემდეგ, მას უკვე შეუძლია ამოიცნოს ახალი ფოტოები, რომლებიც აქამდე არ უნახავს.

1.2. როგორ სწავლობს ბავშვი ცხოველების ამოცნობას სურათებიდან

წარმოიდგინე, რომ ბავშვს გინდა ასწავლო ძაღლის ამოცნობა. შენ არ უხსნი ყველა წესს: „ძაღლს აქვს ოთხი ფეხი, ყეფს, აქვს ბეწვი...“. ამის ნაცვლად შენ უჩვენებ უამრავ სურათს, სადაც ძაღლები არიან, და ეუბნები: „ეს ძაღლია“. რამდენიმე სურათის ნახვის შემდეგ, ბავშვს უკვე შეუძლია, დამოუკიდებლად, ახალ სურათზე თავად ამოიცნოს ძაღლი. ეს არის ზუსტად ის, რასაც მანქანური სწავლების მოდელი აკეთებს.

2. მანქანური სწავლების ტიპები

წარმოიდგინეთ, რომ მანქანა არის მოსწავლე. ისევე როგორც ადამიანები, მანქანებიც სხვადასხვა მეთოდით სწავლობენ. მანქანური სწავლების სამყაროში სამი ძირითადი „პედაგოგიური“ მიდგომა არსებობს.

2.1. ზედამხედველობითი სწავლება (Supervised Learning)

ეს ყველაზე გავრცელებული მეთოდია. წარმოიდგინეთ, რომ ბავშვს სურათებს აჩვენებთ: ერთზე კატაა დახატული და ეუბნებით – „ეს კატაა“, მეორეზე – ძაღლი და ეუბნებით – „ეს ძაღლია“. რამდენიმე ასეთი მაგალითის შემდეგ, ბავშვს შეუძლია დამოუკიდებლად ამოიცნოს, რომელ სურათზეა კატა და რომელზე – ძაღლი.

- **ანალოგია:** როდესაც სკოლაში ხარ, მასწავლებელი გაძლევს დავალებას და შემდეგ ამოწმებს შენს ნამუშევარს. თუ შეცდომა გაქვს, მასწავლებელი გეხმარება მის გამოსწორებაში. მოდელიც ასე სწავლობს: მას აძლევენ მონაცემებს და სწორ პასუხებს, შემდეგ ის აანალიზებს შეცდომას და ცდილობს, შეცდომა მინიმუმამდე დაიყვანოს.
- **გამოყენების მაგალითი:** სახლის ფასის პროგნოზირება. მოდელს აწვდი უამრავი სახლის მონაცემებს (ზომა, ოთახების რაოდენობა, ადგილმდებარეობა) და მათ რეალურ ფასებს. მოდელი სწავლობს, თუ როგორ არის ეს მახასიათებლები დაკავშირებული ფასთან.

2.2. ზედამხედველობის გარეშე სწავლება (Unsupervised Learning)

ახლა წარმოიდგინეთ, რომ ბავშვს აძლევთ ფერადი კუბიკების ნაკრებს, მაგრამ არაფერს ეუბნებით. ის, სავარაუდოდ, თავისით დაიწყებს მათ დახარისხებას: წითლებს ერთად დააწყობს, ლურჯებს – ერთად. ან იქნებ ფორმის მიხედვით დააჯგუფოს.

მანქანაც ასე იქცევა – ჩვენ ვაძლევთ მას უამრავ მონაცემს ეტიკეტის (label) გარეშე და მისი ამოცანაა, თავად იპოვოს მათში ფარული სტრუქტურა ან მსგავსების მქონე ჯგუფები (**კლასტერები**).

- **ანალოგია:** წარმოიდგინე, რომ შედიხარ უზარმაზარ ბიბლიოთეკაში, სადაც წიგნები ქაოტურად არის მიმოფანტული. შენი ამოცანაა, დაალაგო ისინი, მაგრამ არ გაქვს არანაირი მითითება. შენ, ალბათ, დაიწყებ წიგნების დაჯგუფებას ყდის ფერის, ზომის ან თემატიკის მიხედვით. ეს არის დამოუკიდებლად სწავლა – ორგანიზების პრინციპების აღმოჩენა ყოველგვარი წინასწარი წესების გარეშე.
- **გამოყენების მაგალითი:** მომხმარებლების დაჯგუფება. კომპანიას შეუძლია, დააჯგუფოს თავისი მომხმარებლები იმის მიხედვით, თუ რას ყიდულობენ ისინი, რათა შემდეგ მათ კონკრეტული სარეკლამო შეთავაზებები გაუგზავნოს.

2.3. გამოწრთობით სწავლება (Reinforcement Learning)

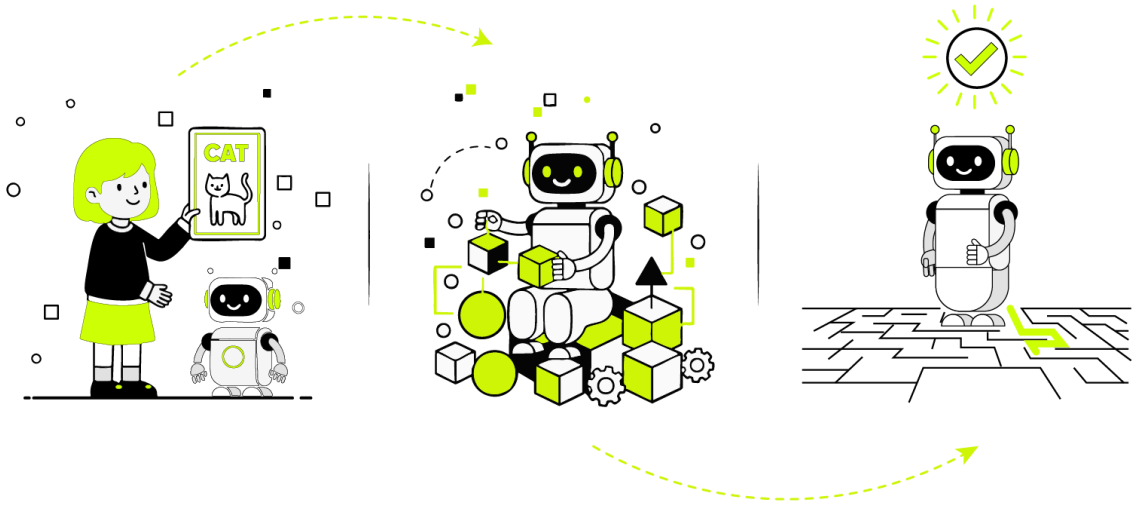
ეს მეთოდი ძალიან ჰგავს ცხოველის გაწვრთნას. როდესაც თქვენს ძაღლს ასწავლით ბრძანებას „დაჯექი“ და ის ამას სწორად აკეთებს, თქვენ მას ჯილდოს (მაგ., სასუსნავს) აძლევთ. თუ არასწორად აკეთებს – არაფერს. დროთა განმავლობაში ძაღლი ხვდება, თუ რომელი ქმედება იწვევს წახალისებას.

ამ შემთხვევაში, ხელოვნური ინტელექტიც ასე მოქმედებს: ის გარკვეულ გარემოში ცდის სხვადასხვა მოქმედებას. სწორი მოქმედებისთვის იღებს „ქულას“ (ჯილდოს), არასწორისთვის კი – არაფერს. მისი მიზანია, მაქსიმალურად ბევრი ქულა დააგროვოს. ამ მეთოდით სწავლობენ პროგრამები, რომლებიც თამაშობენ ჭადრაკს ან მართავენ ავტომობილს.

- **ანალოგია:** წარმოიდგინე ვიდეოთამაშის პერსონაჟი, რომელიც ლაბირინთიდან გამოსვლას ცდილობს. ყოველ სწორ ნაბიჯზე (მაგალითად, წინ წასვლა), ის იღებს ჯილდოს (+10 ქულა), ხოლო თუკი კედელს შეეჯახება (არასწორი მოქმედება), ის იღებს სასჯელს (-5 ქულა). პროგრამა მილიონობით ასეთ ცდას ატარებს და სწავლობს, როგორ

მიიღოს ყველაზე მაღალი ქულა, ანუ როგორ გამოვიდეს ლაბირინთიდან ყველაზე ეფექტურად.

- **გამოყენების მაგალითი:** რობოტი, რომელიც კუბიკის აღებას სწავლობს. თავდაპირველად, რობოტი შემთხვევით მოძრაობებს აკეთებს. თუ მისი ხელი კუბიკს შეეხო, ის იღებს ჯილდოს. თუ კუბიკი აიღო და მაგიდაზე დადო, იღებს დიდ ჯილდოს. თუ მოძრაობისას ხელით მაგიდას შეეხო, იღებს სასჯელს. ამ მეთოდით, ათასობით ცდის შემდეგ, რობოტი თავად სწავლობს, როგორ შეასრულოს დავალება საუკეთესოდ.



3. მონაცემების როლი

მანქანური სწავლების მთავარი ინგრედიენტი მონაცემებია. მოდელის წარმატება პირდაპირ არის დამოკიდებული მონაცემების ხარისხზე.

3.1. სავარჯიშო (Training) და სატესტო (Test) მონაცემების მნიშვნელობა

მოდელის შესაქმნელად, ჩვენ მონაცემებს ვყოფთ ორ ნაწილად:

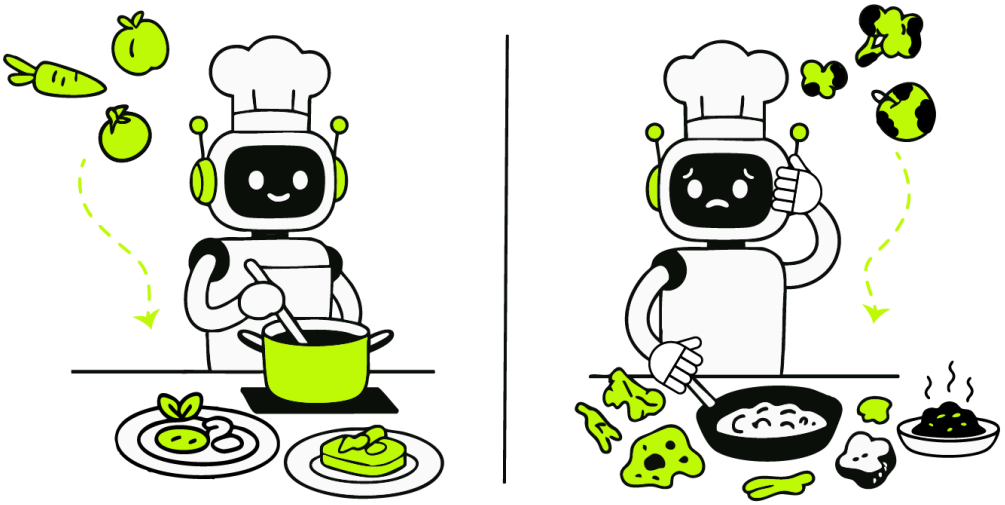
- **სავარჯიშო მონაცემები (Training Data):** ამ მონაცემებზე მოდელი სწავლობს.
- **სატესტო მონაცემები (Test Data):** ამ მონაცემებით ვამოწმებთ, რამდენად კარგად ისწავლა მოდელმა. ეს ჰგავს სკოლაში ნასწავლი მასალის გამოცდაზე შემოწმებას.

3.2. "ნაგავი შედის, ნაგავი გამოდის" (Garbage In, Garbage Out) პრინციპი

ეს არის ერთ-ერთი ყველაზე მნიშვნელოვანი პრინციპი. თუ მოდელს ცუდ ან არასწორ მონაცემებს მივაწვდით, ის აუცილებლად ცუდ შედეგს დაგვიბრუნებს. წარმოიდგინეთ, რომ ცდილობთ, მსოფლიო კლასის შეფ-მზარეული გახდეთ, მაგრამ მხოლოდ გაფუჭებულ, უხარისხო

პროდუქტებს იყენებთ. თქვენი კულინარიული ნიჭის მიუხედავად შედეგი კატასტროფული იქნება. .

ხელოვნური ინტელექტის შემთხვევაშიც ასეა: თუ მას ვასწავლით არასწორი, არასრული ან მიკერძოებული მონაცემებით, მისი პასუხებიც არასწორი, არასრული და მიკერძოებული იქნება. ამ პრინციპს კომპიუტერულ მეცნიერებაში „ნაგავი შედის, ნაგავი გამოდის“ (Garbage In, Garbage Out) ჰქვია.



3.3. მონაცემთა მიკერძოების (Bias) პრობლემა და მისი საფრთხეები

მონაცემები შეიძლება მიკერძოებული იყოს. მაგალითად, თუ პროგრამას, რომელიც ადამიანების სახეებს აანალიზებს, ვასწავლით მხოლოდ ღია ფერის კანის მქონე ადამიანების სურათებზე, ის შეიძლება ნაკლებად ზუსტი იყოს მუქი ფერის კანის მქონე ადამიანების ამოცნობაში. ეს არის სერიოზული ეთიკური პრობლემა, რომელსაც ყოველთვის უნდა მივაქციოთ ყურადღება.

დავალება 8: "შემდეგი სიტყვის პროგნოზირება"

ამ დავალების მიზანია, შექმნა ენობრივი მოდელი უფრო დიდ ტექსტზე დაყრდნობით. პროგრამამ უნდა დაამუშაოს ტექსტი, შექმნას სიტყვების წყვილების ლექსიკონი, დათვალოს, რამდენი უნიკალური სიტყვა ისწავლა მოდელმა, და იწინასწარმეტყველოს მომდევნო სიტყვა.v

დავალება 8.1: შეცდომის პოვნა და გასწორება

მოცემულ კოდში დაშვებულია შეცდომა. შენი ამოცანაა იპოვო და გაასწორო ის, რათა პროგრამამ პირობის შესაბამისად იმუშაოს.

```
# შეცდომით დაწერილი კოდი

# 1. სასწავლო ტექსტი (გაფართოებული)
text = """
ხელოვნური ინტელექტი არის კომპიუტერული მეცნიერების მიმართულება.
ის სწავლობს მონაცემების ანალიზს და ამ მონაცემების საფუძველზე იღებს
გადაწყვეტილებებს.
მანქანური სწავლება არის ხელოვნური ინტელექტის ქვედარგი.
გამართული მოდელის შესაქმნელად საჭიროა ბევრი და ხარისხიანი მონაცემი.
მონაცემების დამუშავება მნიშვნელოვანი ეტაპია.
საქართველოში ტექნოლოგიების განვითარება სწრაფად მიმდინარეობს.
ახალგაზრდები ინტერესდებიან ისეთი სფეროებით, როგორიც არის მანქანური
სწავლება.
"""

# 2. ტექსტის დამუშავება
words = text.lower().replace('.', '').replace(',', '').split()

# 3. მოდელის შექმნა (შეცდომა აქ არის)
model = {}
for i in range(len(words) - 1):
    current_word = words[i]
    next_word = words[i+1]

    # ეს ხაზი იწვევს KeyError-ს, რადგან current_word-ისთვის სია ჯერ
არ არის შექმნილი.
    model[current_word].append(next_word)

# 4. ნასწავლი სიტყვების რაოდენობის განსაზღვრა
vocabulary_size = len(model)
print(f"მოდელმა ისწავლა {vocabulary_size} უნიკალური სიტყვა.")

# 5. პროგნოზის ფუნქცია
def predict_next_word(word):
    word = word.lower()
    if word not in model:
        return "ეს სიტყვა მოდელში არ არის."
```

```
    possible_next_words = model[word]
    most_common_word = max(set(possible_next_words),
key=possible_next_words.count)
    return most_common_word

# 6. მოდელის ტესტირება
input_word = "მონაცემების"
predicted_word = predict_next_word(input_word)

print(f"სიტყვისთვის '{input_word}' შემდეგი სავარაუდო სიტყვაა:
'{predicted_word}'")
```

დავალება 8.2: კოდის დასრულება
მოცემულ კოდს აკლია მნიშვნელოვანი ლოგიკური ელემენტი – for ციკლი, რომელიც ტექსტისგან ქმნის სიტყვების მოდელს. შენი ამოცანაა, შეავსო ეს გამოტოვებული ნაწილი, რათა პროგრამამ შეძლოს მოდელის აწყობა. უნიკალური სიტყვების დათვლა და პროგნოზის გაკეთება.

```
# კოდი, რომელიც უნდა დაასრულო

# 1. სასწავლო ტექსტი (გაფართოებული)
text = """
ხელოვნური ინტელექტი არის კომპიუტერული მეცნიერების მიმართულება.
ის სწავლობს მონაცემების ანალიზს და ამ მონაცემების საფუძველზე იღებს
გადაწყვეტილებებს.
მანქანური სწავლება არის ხელოვნური ინტელექტის ქვედარგი.
ტექნოლოგიურად დახვეწილი მოდელის შესაქმნელად საჭიროა ბევრი და
ხარისხიანი მონაცემი.
მონაცემების დამუშავება მნიშვნელოვანი ეტაპია.
საქართველოში ტექნოლოგიების განვითარება სწრაფად მიმდინარეობს.
ახალგაზრდები ინტერესდებიან ისეთი სფეროებით, როგორიც არის მანქანური
სწავლება.
"""

# 2. ტექსტის დამუშავება
words = text.lower().replace('.', '').replace(',', '').split()

# 3. მოდელის შექმნა
model = {}

# შენი კოდი აქ:
```

```
# დაწერე for ციკლი, რომელიც გადაუვლის words სიას და შეავსებს model ლექსიკონს.
# არ დაგავიწყდეს გასაღების არსებობის შემოწმება, სანამ მას მნიშვნელობას დაამატებ.


# 4. ნასწავლი სიტყვების რაოდენობის განსაზღვრა
vocabulary_size = len(model)
print(f"მოდელმა ისწავლა {vocabulary_size} უნიკალური სიტყვა.")

# 5. პროგნოზის ფუნქცია (ეს ნაწილი გამართულია)
def predict_next_word(word):
    word = word.lower()
    if word not in model:
        return "ეს სიტყვა მოდელში არ არის."

    possible_next_words = model[word]
    most_common_word = max(set(possible_next_words), key=possible_next_words.count)
    return most_common_word

# 6. მოდელის ტესტირება
input_word = "მონაცემების"
predicted_word = predict_next_word(input_word)

print(f"სიტყვისთვის '{input_word}' შემდეგი სავარაუდო სიტყვაა: '{predicted_word}'")
```

დავალება 8.3: კოდის დაწერა ნულიდან

დაწერე პროგრამული კოდი, შექმენი პროგრამა, რომელიც:

1. გამოიყენებს მოცემულ გრძელ ტექსტს.
2. დაამუშავებს ტექსტს (სიტყვებად დაშლა, პატარა ასოებზე გადაყვანა).
3. ააწყობს `model` ლექსიკონს სიტყვების წყვილების დასამახსოვრებლად.
4. დათვლის და დაბეჭდავს, რამდენი უნიკალური სიტყვაა მოდელში.
5. შექმნის `predict_next_word` ფუნქციას.
6. გამოიძახებს ფუნქციას სიტყვა "ხელოვნური"-სთვის და დაბეჭდავს შედეგს.

```
# დაწერე შენი კოდი აქ:
```

სწორი პასუხი (პროგრამული კოდი სრულად):

```
# ამ კოდის საშუალებით შეგიძლია გადაამოწმო შენი ნამუშევარი

# 1. სასწავლო ტექსტი (გაფართოებული)
text = """
ხელოვნური ინტელექტი არის კომპიუტერული მეცნიერების მიმართულება.
ის სწავლობს მონაცემების ანალიზს და ამ მონაცემების საფუძველზე იღებს
გადაწყვეტილებებს.
მანქანური სწავლება არის ხელოვნური ინტელექტის ქვედარგი.
ტექნოლოგიურად დახვეწილი მოდელის შესაქმნელად საჭიროა ბევრი და
ხარისხიანი მონაცემი.
მონაცემების დამუშავება მნიშვნელოვანი ეტაპია.
საქართველოში ტექნოლოგიების განვითარება სწრაფად მიმდინარეობს.
ახალგაზრდები ინტერესდებიან ისეთი სფეროებით, როგორიც არის მანქანური
სწავლება.
"""

# 2. ტექსტის დამუშავება
# ვასუფთავებთ ტექსტს სასვენი ნიშნებისგან და ვშლით სიტყვებად
words = text.lower().replace('.', '').replace(',', '').split()

# 3. მოდელის შექმნა
model = {}
for i in range(len(words) - 1):
    current_word = words[i]
    next_word = words[i+1]

    # ვამოწმებთ, თუ სიტყვა (გასაღები) უკვე არსებობს ლექსიკონში.
    # თუ არა, ვქმნით ახალ ცარიელ სიას.
    if current_word not in model:
        model[current_word] = []

    # ვამატებთ მომდევნო სიტყვას შესაბამის სიაში.
    model[current_word].append(next_word)

# 4. ნასწავლი სიტყვების რაოდენობის განსაზღვრა
# ლექსიკონში გასაღებების რაოდენობა უნიკალური სიტყვების რაოდენობის
ტოლია
vocabulary_size = len(model)
```

```
print(f"მოდელმა ისწავლა {vocabulary_size} უნიკალური სიტყვა.")

# 5. პროგნოზის ფუნქციის შექმნა
def predict_next_word(word):
    """
    პოულობს და აბრუნებს მოცემული სიტყვის ყველაზე სავარაუდო მომდევნო
სიტყვას.
    """
    word = word.lower()
    if word not in model:
        return "ეს სიტყვა მოდელში არ არის."

    possible_next_words = model[word]
    # პოულობს ყველაზე ხშირ ელემენტს სიაში
    most_common_word = max(set(possible_next_words),
key=possible_next_words.count)
    return most_common_word

# 6. მოდელის ტესტირება
input_word = "მონაცემების"
predicted_word = predict_next_word(input_word)

print(f"სიტყვისთვის '{input_word}' ნაპროგნოზები სიტყვაა:
'{predicted_word}'")
```