Vigenere Cipher

July 6, 2023

[1]:
```
# Verschlüsselt einen Text message mit Schlüssel key nach Vigenère
cipher = ""
message =␣
 ↪"AUFDIESEWEISEBEKOMMTMANRECHTSCHNELLDIESCHLUESSELLAENGEDESVERSCHLUESSELTENTEXTESHERAUSJETZT
 ↪#message in Grossbuchstaben, kann mit Lücken sein
key = "vigenere" #Schlüssel in Kleinbuchstaben
keylength = len(key)

l=0
for k in range(len(message)):
    if message[k] == " ":
        cipher += " "
    else:
        kshift = l % keylength
        l += 1
        shift = ord(key[kshift]) - 97
        cipher += chr((ord(message[k]) - 65 + shift) % 26 + 65)
print(cipher)
```

```
VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJL
ZZGYFNVXUBSYFWEYMVUGUHVVBMNIVQKISBYTNPKIIEKMFIQIMTKKGAVVYMTHVIJTVTZIAAVPXPKQVXUI
HAKPOIEFPKNWGESIIDKVFGYPPMYWRPKAPZJIAAVVYMTDHWRQHMTKRJRWNBJMRIEXNXXIPLVRYMGPCLRF
ZBBIEWTLDMHYAKUIMMORMICRZVZIVPKISBKPBIJXHITRHRDMOBKPFLRIPNOKXIZXNITEYCJIRMOWGHVV
OMDXXIZRZWJIERLVRMTMTIIIYCTHNRQIIIAJOIZWKQKPFAVMNMCIVPVVFCXDVWKPVMYWGWZGCLOIFGYP
PMYWRPCEZVMIAMTLOUOXQIDOVAOWXMKINBNIEELWAQTHRRLROMXHRVMSMIAWFIKDPVMHNWJINAOGUFVM
YMSWPLCYZAYIYYDIDVCSEXRYNMORRQNSZZZIEFLGCPGRQICXPVJEHGYHZZZIKXDMOMORRQNSMBHITMER
OTGIFWKWDKNNRHFGCLAVPLXINKNMPOKINIAWFSIXDMXIAYEAVPXWPLVMITOGUIIRBZGQZTREMMORIMVP
ZVLERPCIILKVFGYPPMYWRPWMILKRQEQYRMXHRRQYZZYXGIOXNKNPHIJWZTQSZFZRVBOSAIEFZEKVGIKS
CVKHNWJINMORRVFPGMYTVICXRIYHNZFRFTGVGIOXWHCWPLCYZAYIYMJXYQKEADRLGLKVZSVKGQILXIZX
ZVCMEHJSWMXIVXJMHMXWGIEWXPXMGXMSIAKGUYEWUEGRMMXEPNJVRMQICVNEYFZIMBGPYIUEIVTSPLMI
MJRIVFVRYMTRTVRQHXGEEINIMLKRAYEKZUGIFWZLMMXANLIWXPKMAPZGCSKMGEDEINGRTIZRZACSEXVW
UCYXRLVRBMCMPLKIOQYXZMEHZAZIAWVMIMYHRVEKMISQREVYNAKVFXLRRINVFGYIDVRMPLRRZQTIZAFV
OITJNRXADZJHNWXEIHKTNEIZZZCSEJVR
```

[2]:
```
# Berechnet den Koizidenzindex eines Texts
def coincidence(text):
    length = len(text) # Text halbieren
    half_length = length // 2
```

```
    first_half = text[:half_length]
    second_half = text[half_length:]

    count = 0

    for i in range(half_length): # Koinzidenzen zählen
        if first_half[i] == second_half[i]:
            count += 1
    coinc = count/half_length
    return count, coinc

text =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
equals_count, coindex = coincidence(text)
half_length = len(text) // 2
cipher_kappa = coindex
friedmann_guess = (0.0762-0.03846153846)/(cipher_kappa-0.03846153846) # kappa␣
 ↪Deutsch ist 0.0762

print(equals_count, "Übereinstimmungen auf", half_length, "Pärchen")
print("Koinzidenzindex \N{GREEK SMALL LETTER KAPPA} =", cipher_kappa)
print("Schlüssellängen-Schätzung nach Friedmann: n =", friedmann_guess)
```

```
41 Übereinstimmungen auf 536 Pärchen
Koinzidenzindex   = 0.07649253731343283
Schlüssellängen-Schätzung nach Friedmann: n = 0.9923079245286132
```

[3]:
```
# Cipher in Anzahl-vermutete-Schlüsselwortlänge Spalten schreiben
# Cipher auf zweitletzte Zeile trimmen, falls die letzte Zeile nicht␣
 ↪vollständig ist
def split_text(text, k):
    rows = [text[i:i+k] for i in range(0, len(text), k)]
    return rows


text =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
k = int(input("Vermutete Schlüsselwortlänge: "))

rows = split_text(text, k)

print("Splited Text:")
for row in rows:
    print(row)
```

```
Vermutete Schlüsselwortlänge: 50
Splited Text:
```

```
VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGI
KRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUBSYFWEYMVUG
UHVVBMNIVQKISBYTNPKIIEKMFIQIMTKKGAVVYMTHVIJTVTZIAA
VPXPKQVXUIHAKPOIEFPKNWGESIIDKVFGYPPMYWRPKAPZJIAAVV
YMTDHWRQHMTKRJRWNBJMRIEXNXXIPLVRYMGPCLRFZBBIEWTLDM
HYAKUIMMORMICRZVZIVPKISBKPBIJXHITRHRDMOBKPFLRIPNOK
XIZXNITEYCJIRMOWGHVVOMDXXIZRZWJIERLVRMTMTIIIYCTHNR
QIIIAJOIZWKQKPFAVMNMCIVPVVFCXDVWKPVMYWGWZGCLOIFGYP
PMYWRPCEZVMIAMTLOUOXQIDOVAOWXMKINBNIEELWAQTHRRLROM
XHRVMSMIAWFIKDPVMHNWJINAOGUFVMYMSWPLCYZAYIYYDIDVCS
EXRYNMORRQNSZZZIEFLGCPGRQICXPVJEHGYHZZZIKXDMOMORRQ
NSMBHITMEROTGIFWKWDKNNRHFGCLAVPLXINKNMPOKINIAWFSIX
DMXIAYEAVPXWPLVMITOGUIIRBZGQZTREMMORIMVPZVLERPCIIL
KVFGYPPMYWRPWMILKRQEQYRMXHRRQYZZYXGIOXNKNPHIJWZTQS
ZFZRVBOSAIEFZEKVGIKSCVKHNWJINMORRVFPGMYTVICXRIYHNZ
FRFTGVGIOXWHCWPLCYZAYIYMJXYQKEADRLGLKVZSVKGQILXIZX
ZVCMEHJSWMXIVXJMHMXWGIEWXPXMGXMSIAKGUYEWUEGRMMXEPN
JVRMQICVNEYFZIMBGPYIUEIVTSPLMIMJRIVFVRYMTRTVRQHXGE
EINIMLKRAYEKZUGIFWZLMMXANLIWXPKMAPZGCSKMGEDEINGRTI
ZRZACSEXVWUCYXRLVRBMCMPLKIOQYXZMEHZAZIAWVMIMYHRVEK
MISQREVYNAKVFXLRRINVFGYIDVRMPLRRZQTIZAFVOITJNRXADZ
JHNWXEIHKTNEIZZZCSEJVR
```

[4]:
```python
# Wiederholungen der Länge 3 finden
from collections import Counter

def find_repeated_sequences(text):
    repeated_sequences = []

    for i in range(len(text) - 2):
        sequence = text[i:i+3]
        remaining_text = text[i+3:]
        if sequence in remaining_text:
            repeated_sequences.append(sequence)

    return Counter(repeated_sequences)


text =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
repeated_sequences = find_repeated_sequences(text)

print("Wiederholungen von Schnippsel:")
for sequence, count in repeated_sequences.items():
    if count >=3:
        print(f"Schnippsel: {sequence}, Anzahl: {count}")
```

```
Wiederholungen von Schnippsel:
```

```
Schnippsel: FGY, Anzahl: 5
Schnippsel: KVF, Anzahl: 3
Schnippsel: VFG, Anzahl: 3
Schnippsel: GYP, Anzahl: 3
Schnippsel: YPP, Anzahl: 3
Schnippsel: PPM, Anzahl: 3
Schnippsel: PMY, Anzahl: 3
Schnippsel: MYW, Anzahl: 4
Schnippsel: YWR, Anzahl: 3
Schnippsel: WRP, Anzahl: 3
Schnippsel: MOR, Anzahl: 4
```

[5]:
```python
# Wiederholungen vorgegebener Länge finden und auch deren Abstände printen
from collections import Counter
import re


def find_repeated_sequences(text, min_length, max_length):
    repeated_sequences = []

    for length in range(min_length, max_length + 1):
        for i in range(len(text) - length + 1):
            sequence = text[i:i+length]
            remaining_text = text[i+length:]
            if sequence in remaining_text:
                repeated_sequences.append(sequence)

    return Counter(repeated_sequences)

def measure_gap(sequence, text):
    gaps = []
    length = len(sequence)
    gap = 0
    for i in range(len(text) - length + 1):
        remaining_text = text[i+length:i+2*length]
        gap += 1
        if sequence == remaining_text:
            gaps.append(gap)
            gap = 0
    return gaps


text =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
min_length = int(input("Schnippsel-Minimallänge: "))
max_length = int(input("Schnippselmaximallänge: "))
```

```
repeated_sequences = find_repeated_sequences(text, min_length, max_length)

for sequence, count in repeated_sequences.items():
    if count >= 2:
        abstand = measure_gap(sequence, text)
        print(f"Schnippsel: {sequence}, Anzahl: {count}, Abstand: {abstand}" )
```

```
Schnippsel-Minimallänge: 8
Schnippselmaximallänge: 12
Schnippsel: KVFGYPPM, Anzahl: 2, Abstand: [51, 120, 472]
Schnippsel: VFGYPPMY, Anzahl: 2, Abstand: [52, 120, 472]
Schnippsel: FGYPPMYW, Anzahl: 3, Abstand: [53, 120, 216, 256]
Schnippsel: GYPPMYWR, Anzahl: 3, Abstand: [54, 120, 216, 256]
Schnippsel: YPPMYWRP, Anzahl: 3, Abstand: [55, 120, 216, 256]
Schnippsel: KVFGYPPMY, Anzahl: 2, Abstand: [50, 120, 472]
Schnippsel: VFGYPPMYW, Anzahl: 2, Abstand: [51, 120, 472]
Schnippsel: FGYPPMYWR, Anzahl: 3, Abstand: [52, 120, 216, 256]
Schnippsel: GYPPMYWRP, Anzahl: 3, Abstand: [53, 120, 216, 256]
Schnippsel: KVFGYPPMYW, Anzahl: 2, Abstand: [49, 120, 472]
Schnippsel: VFGYPPMYWR, Anzahl: 2, Abstand: [50, 120, 472]
Schnippsel: FGYPPMYWRP, Anzahl: 3, Abstand: [51, 120, 216, 256]
Schnippsel: KVFGYPPMYWR, Anzahl: 2, Abstand: [48, 120, 472]
Schnippsel: VFGYPPMYWRP, Anzahl: 2, Abstand: [49, 120, 472]
Schnippsel: KVFGYPPMYWRP, Anzahl: 2, Abstand: [47, 120, 472]
```

[6]:
```
import math

gcd = math.gcd(120,216,256,472)
print(gcd)
```

```
8
```

[7]:
```
def split_text(text, k): # Text mit vermuteter Schlüsselwortlänge splitten und␣
 ↪die monoalphabetischen Spalten auswerten
    rows = [text[i:i+k] for i in range(0, len(text), k)]
    return rows


def count_most_frequent_letters(rows): # Häufigste Buchstaben zählen
    column_counts = []

    for column in zip(*rows):
        counts = {}
        for letter in column:
            if letter not in counts:
                counts[letter] = 0
            counts[letter] += 1
```

```
        most_frequent_letter = max(counts, key=counts.get)
        column_counts.append((most_frequent_letter,␣
 ↪counts[most_frequent_letter]))

    return column_counts


text =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
k = int(input("Vermutete Schlüsselwortlänge: "))

rows = split_text(text, k)
column_counts = count_most_frequent_letters(rows)

print("\nHäufigster Buchstabe pro Spalte:")
i = 1
for column_count in column_counts:
    letter, count = column_count
    print(f"Spalte {i}: {letter}, {count} Mal")
    i += 1

print("\nMögliches Keyword: ", end="") # end empty string verhindert den␣
 ↪Zeilenumbruch
for column_count in column_counts: # Shift des jeweiligen Es berechnen und␣
 ↪daraus keyword bestimmen
    letter, count = column_count
    count = ord(letter)-ord("E")
    if count>0:
        guess = chr(65+count)
    else:
        guess = chr(69+count)
    print(f"{guess} ", end="")
print()

print("\nSplit Text:")
for row in rows:
    print(row)
```

```
Vermutete Schlüsselwortlänge: 8

Häufigster Buchstabe pro Spalte:
Spalte 1: Z, 21 Mal
Spalte 2: M, 32 Mal
Spalte 3: K, 23 Mal
Spalte 4: I, 24 Mal
Spalte 5: R, 19 Mal
Spalte 6: I, 29 Mal
```

Spalte 7: V, 21 Mal
Spalte 8: I, 27 Mal

Mögliches Keyword: V I G E N E R E

Split Text:
VCLHVIJI
RMOWRFVO
JUSXZEEV
ZKNXFGYR
ZTRHVIJG
CTAIFWVP
GIKRTIUI
NDKVFGYP
PMYWRPKI
IBKBGIJL
ZZGYFNVX
UBSYFWEY
MVUGUHVV
BMNIVQKI
SBYTNPKI
IEKMFIQI
MTKKGAVV
YMTHVIJT
VTZIAAVP
XPKQVXUI
HAKPOIEF
PKNWGESI
IDKVFGYP
PMYWRPKA
PZJIAAVV
YMTDHWRQ
HMTKRJRW
NBJMRIEX
NXXIPLVR
YMGPCLRF
ZBBIEWTL
DMHYAKUI
MMORMICR
ZVZIVPKI
SBKPBIJX
HITRHRDM
OBKPFLRI
PNOKXIZX
NITEYCJI
RMOWGHVV
OMDXXIZR
ZWJIERLV

RMTMTIII
YCTHNRQI
IIAJOIZW
KQKPFAVM
NMCIVPVV
FCXDVWKP
VMYWGWZG
CLOIFGYP
PMYWRPCE
ZVMIAMTL
OUOXQIDO
VAOWXMKI
NBNIEELW
AQTHRRLR
OMXHRVMS
MIAWFIKD
PVMHNWJI
NAOGUFVM
YMSWPLCY
ZAYIYYDI
DVCSEXRY
NMORRQNS
ZZZIEFLG
CPGRQICX
PVJEHGYH
ZZZIKXDM
OMORRQNS
MBHITMER
OTGIFWKW
DKNNRHFG
CLAVPLXI
NKNMPOKI
NIAWFSIX
DMXIAYEA
VPXWPLVM
ITOGUIIR
BZGQZTRE
MMORIMVP
ZVLERPCI
ILKVFGYP
PMYWRPWM
ILKRQEQY
RMXHRRQY
ZZYXGIOX
NKNPHIJW
ZTQSZFZR
VBOSAIEF
ZEKVGIKS

```
CVKHNWJI
NMORRVFP
GMYTVICX
RIYHNZFR
FTGVGIOX
WHCWPLCY
ZAYIYMJX
YQKEADRL
GLKVZSVK
GQILXIZX
ZVCMEHJS
WMXIVXJM
HMXWGIEW
XPXMGXMS
IAKGUYEW
UEGRMMXE
PNJVRMQI
CVNEYFZI
MBGPYIUE
IVTSPLMI
MJRIVFVR
YMTRTVRQ
HXGEEINI
MLKRAYEK
ZUGIFWZL
MMXANLIW
XPKMAPZG
CSKMGEDE
INGRTIZR
ZACSEXVW
UCYXRLVR
BMCMPLKI
OQYXZMEH
ZAZIAWVM
IMYHRVEK
MISQREVY
NAKVFXLR
RINVFGYI
DVRMPLRR
ZQTIZAFV
OITJNRXA
DZJHNWXE
IHKTNEIZ
ZZCSEJVR
```

[8]:
```
# Entschlüsselt eine Vigenère-Cipher cipher mit Schlüsselwort key
message = ""
```

```
cipher =␣
 ↪"VCLHVIJIRMOWRFVOJUSXZEEVZKNXFGYRZTRHVIJGCTAIFWVPGIKRTIUINDKVFGYPPMYWRPKIIBKBGIJLZZGYFNVXUB
 ↪#message in Grossbuchstaben, kann mit Lücken sein
key = "vigenere" #Schlüssel in Kleinbuchstaben
keylength = len(key)

l=0
for k in range(len(cipher)):
    if cipher[k] == " ":
        message += " "
    else:
        kshift = l % keylength
        l += 1
        shift = ord(cipher[k]) - (ord(key[kshift])-32)
        if shift>=0:
            message += chr(65+shift)
        else:
            message += chr(91+shift)
print(message)
```

```
AUFDIESEWEISEBEKOMMTMANRECHTSCHNELLDIESCHLUESSELLAENGEDESVERSCHLUESSELTENTEXTESH
ERAUSJETZTMUSSNURNOCHDERGEHEIMTEXTSPALTENWEISEZERLEGTWERDENDIESPALTENWELCHEMITDE
MSELBENBUCHSTABENVERSCHLUESSELTWURDENWERDENZUSAMMENGEFASSTDIEENTSPRECHENDEALPHAB
ETVERSCHIEBUNGDEREINZELNENTEILTEXTELOESTMANNUNMITTELSHAEUFIGKEITSANALYSEWEISTDER
TEXTKEINEODERNURWENIGEREDUNDANZENAUFBEISPIELSWEISEWEILERKURZISTLAESSTSICHDIESCHL
UESSELLAENGENICHTMITDEMKASISKITESTHERAUSFINDENUNTERDERVORAUSSETZUNGDASSESSICHBEI
DEMSCHLUESSELUMEINWORTAUSEINEMWOERTERBUCHHANDELTUNDAUCHDERTEXTMITEINEMWORTBEGINN
TLAESSTSICHJEDOCHDURCHGESCHICKTESAUSSORTIERENUNWAHRSCHEINLICHERNGRAMMPAAREINVIEL
ENFAELLENDERSCHLUESSELFINDENDAZUWERDENZUERSTTEXTSCHLUESSELKOMBINATIONENBEWERTETO
HNEDASSESEINEROLLESPIELTWASDAVONKLARTEXTBZWSCHLUESSELISTDIEANZAHLDERMOEGLICHKEIT
ENWIRDSOBEREITSIMERSTENSCHRITTVONSECHUNSZWANZIGAUFDREIZEHNHALBIERTALLEDANNNOCHVE
RBLEIBENDENNGRAMMPAAREWERDENNUNGEMAESSIHRERWAHRSCHEINLICHKEITAMANFANGEINESWORTES
ZUSTEHENGEWICHTETISTMINDESTENSEINESDERNGRAMMEAEUSSERSTUNWAHRSCHEINLICHANEINEMWOR
TANFANGWIRDDASGANZEPAARVERWORFEN
```

[ ]: