CS 528 – Spring 2024
Project Report

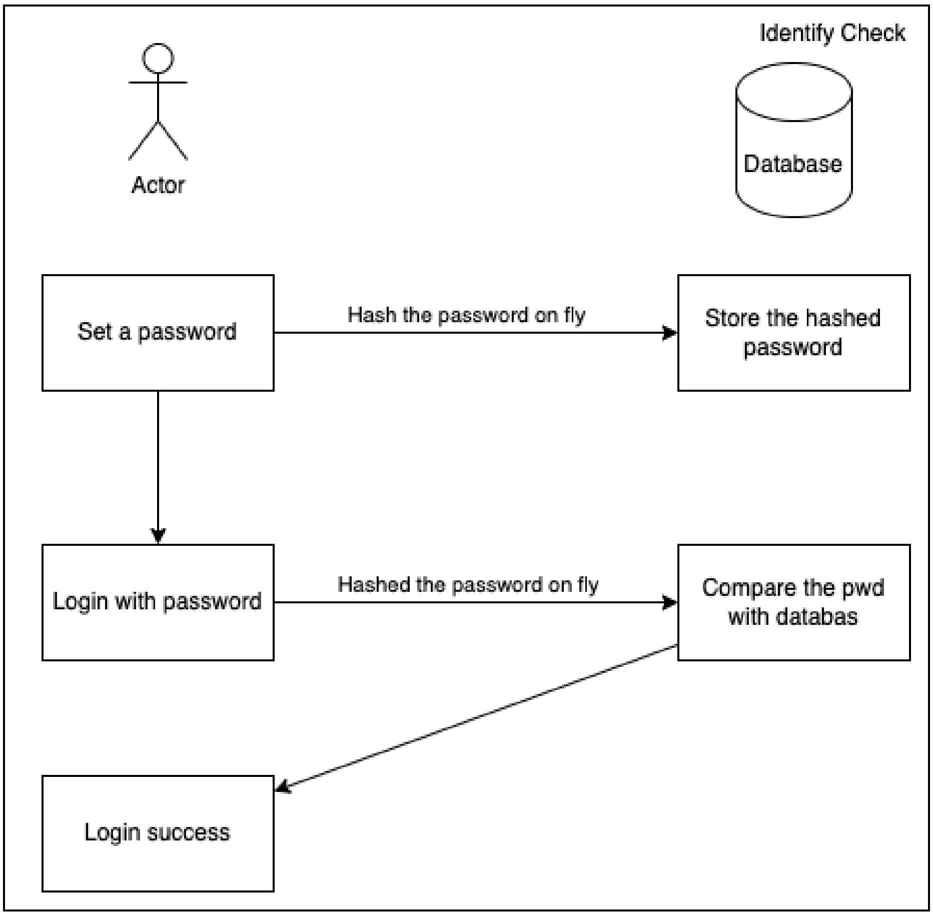
Group Member:
Jianqi Jin(jjin19@hawk.iit.edu)
Hsin-I Lo(hlo2@hawk.iit.edu)

Application

Our application in this project aims to improve hotel management practices by leveraging advanced data analytics techniques while prioritizing the protection of guests' privacy. We focus on two key aspects: data preprocessing and user data retrieval.

In data preprocessing, we employ k-anonymity techniques to anonymize sensitive attributes, particularly guest visitation dates. This ensures that each guest's information remains indistinguishable among a group of at least 10 other individuals in the dataset, mitigating the risk of re-identification attacks while preserving data utility.

For user data retrieval, we utilize cryptographic techniques such as MD5 and RSA to securely retrieve and confirm user identities. By generating unique identifiers for each user and encrypting them for secure transmission, we authenticate user identities without compromising privacy or exposing sensitive information.



Overall, our application seamlessly integrates with hotel management systems to enhance operational efficiency and guest experience. By unlocking insights from anonymized datasets and securely managing user identities, hotels can optimize resource allocation, personalize guest services, and improve overall customer satisfaction.

Datasets

Our dataset encompasses a rich array of information pertinent to hotel bookings, providing insights into guests' reservation patterns and preferences. It includes a wide range of attributes such as hotel type, booking status, lead time, arrival date specifics, duration of stays, guest demographics, meal preferences, country of origin, market segment, distribution channel, guest status, history of cancellations and previous bookings, room type specifications, booking modifications, deposit type, agent and company associations, waiting list duration, customer type, average daily rate, parking space requirements, special requests, reservation status, and relevant dates.

Additionally, personal identification details such as name, email, phone number, and credit card information are collected for reservation purposes. Age is recorded as a string value, among other potentially sensitive information.

Furthermore, it's important to note that we have extracted 1000 user profiles from this dataset, which originally contains over one hundred thousand entries, for our subsequent analysis.

Technique

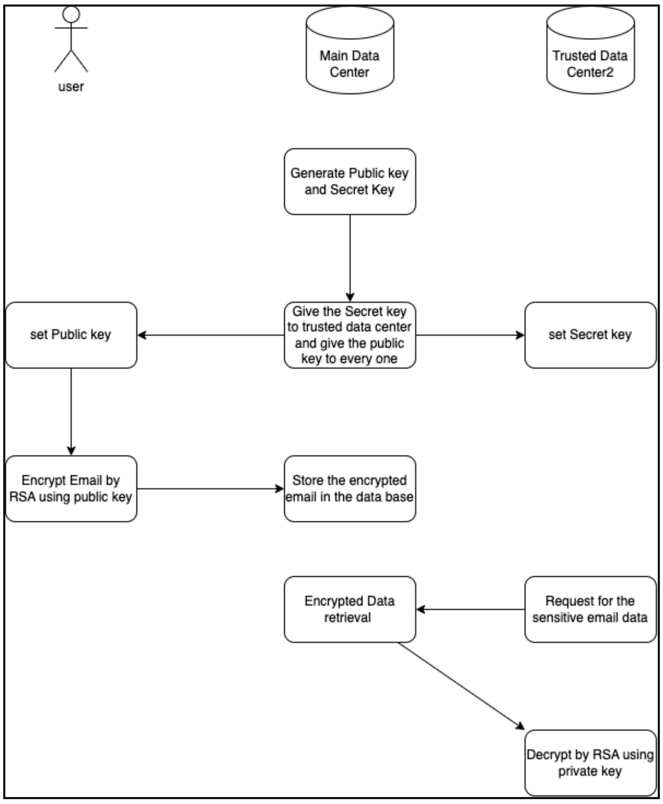
In our project part 1, we will implement k-anonymity techniques to safeguard the privacy of our guests' information, particularly concerning sensitive attributes such as visitation dates. We recognize the potential risks associated with disclosing guests' accommodation dates, as well as the significance of quasi-identifiers like age, nationality, and the total number of overnight stays. These factors collectively contribute to the complexity of our dataset and underscore the importance of privacy preservation.

By employing k-anonymity with a parameter set to 10, we aim to ensure that each individual's information remains indistinguishable among at least 9 other individuals in the dataset. This approach aligns with best practices in data privacy and reflects our commitment to maintaining the trust and confidentiality of our guests' data, while still allowing for meaningful analysis of the dataset's insights.

Part 2 is focusing on the user identity check and sensitive data retrieval. And it will be separated into two different sub parts.

Part 2.1 utilizes MD5 hashing to enhance the security of user password storage during the registration and login processes. Upon user registration, passwords are hashed using the MD5 algorithm and stored as tokens in the database, ensuring that actual passwords are not stored or transmitted in plaintext. During login, the same MD5 hashing is applied to the submitted password for authentication by comparing it to the stored hash. Although MD5 helps prevent direct password exposure, its vulnerability to collision attacks and advancements in computing power necessitate considering stronger algorithms like SHA-256 for increased security, especially given the modern requirements for cryptographic strength and data integrity.

Part 2.2 implements RSA encryption to secure user email data across two trusted data centers, highlighting the importance of robust key management and secure data transfer protocols. Each user's email is encrypted using their public RSA key at Data Center 1 and stored encrypted to ensure privacy and security. When required, the encrypted emails are transferred to and decrypted at Data Center 2 using the corresponding private keys, which are managed under strict access controls to prevent unauthorized access. This method not only secures the email content during transmission and storage but also ensures that decryption is only possible within the secure environment of the trusted data centers, significantly reducing the risk of data breaches and enhancing compliance with data protection regulations.


user
Main Data Center
Trusted Data Center2
Generate Public key and Secret Key
set Public key
Give the Secret key to trusted data center and give the public key to every one
set Secret key
Encrypt Email by RSA using public key
Store the encrypted email in the data base
Encrypted Data retrieval
Request for the sensitive email data
Decrypt by RSA using private key

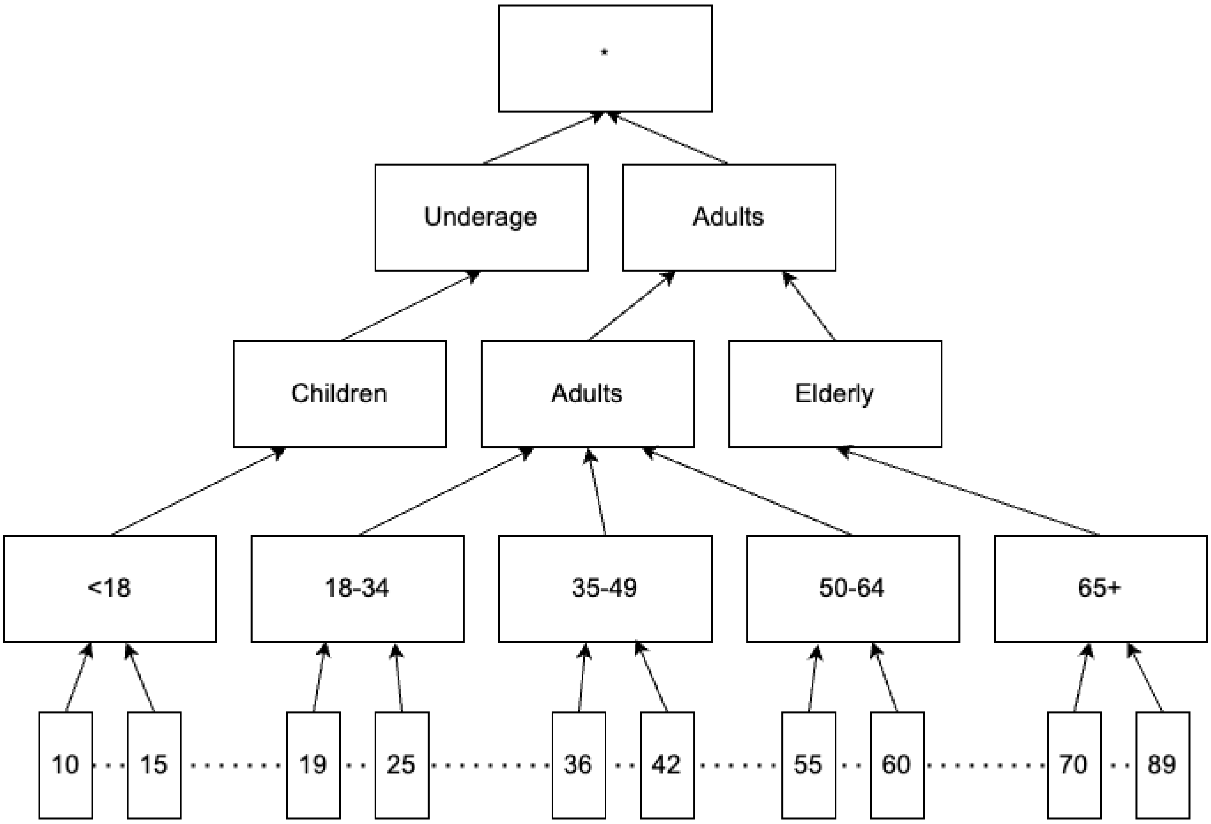
Code Implementation

Part 1: k-anonymity

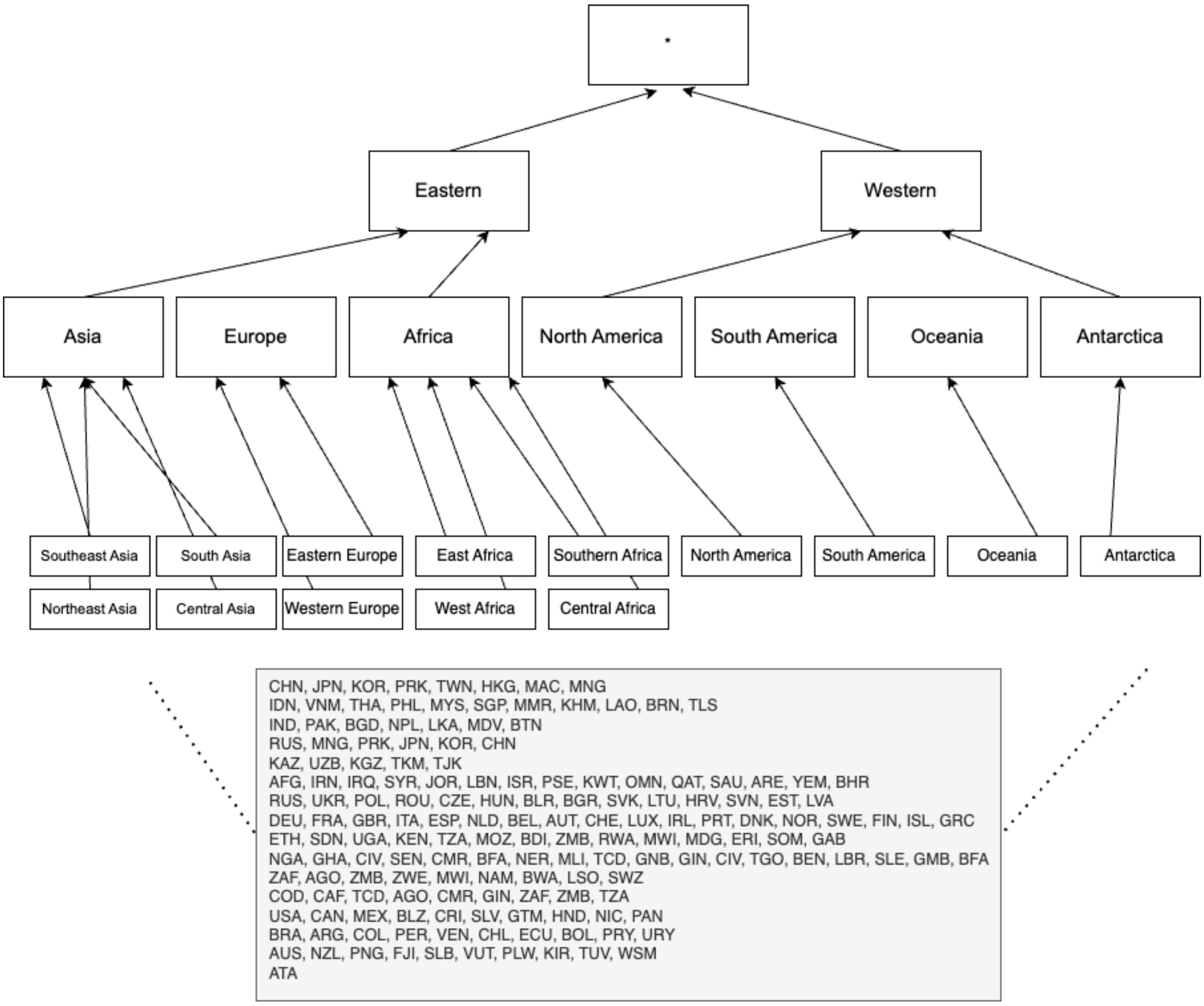
Part 1.1: Design

Quasi-identifiers (QIs) generalization and/or suppression and the Domain Generalization Hierarchy (DGH) and Value Generalization Hierarchy (VGH)

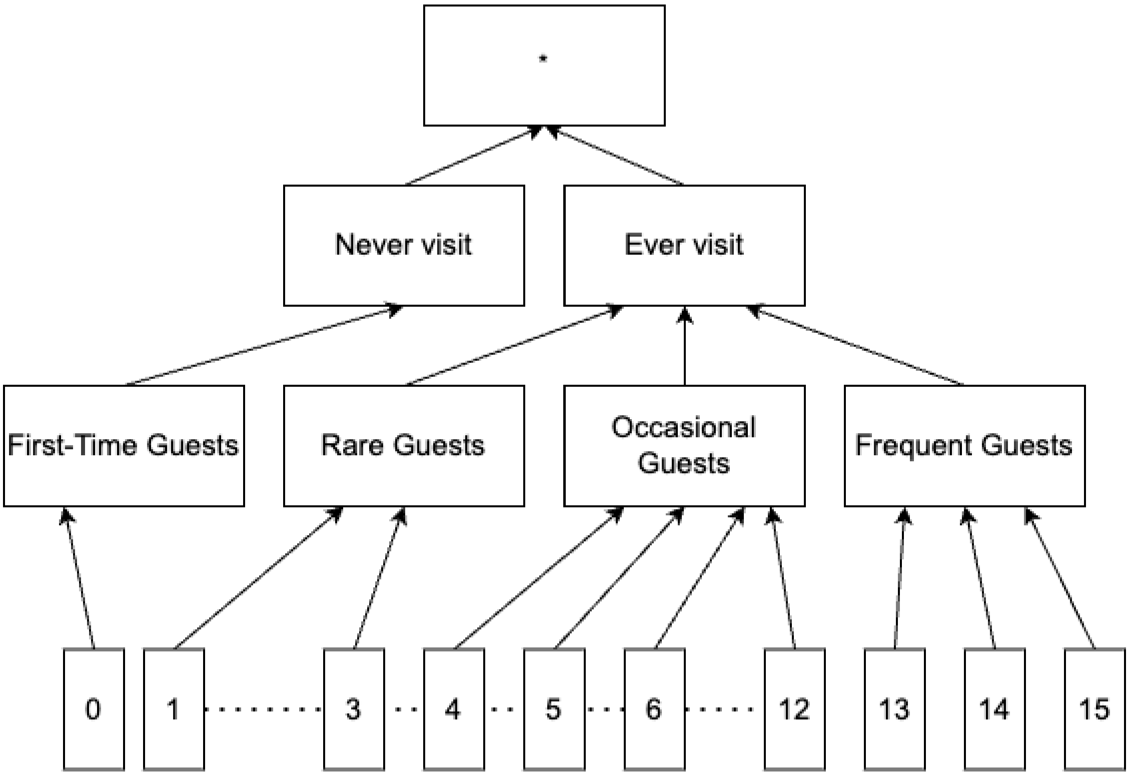
1. Age:
   D4 = {*}
   D3 = {Underage / Adults}
   D2 = {Children / Adults / Elderly}
   D1 = {<18 / 18-34 / 35-49 / 50-64 / 65+}
   D0 = {Real Age}



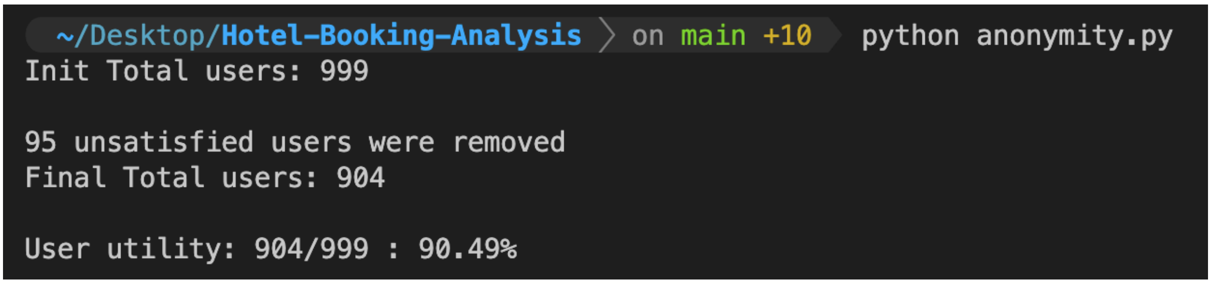
2. Country:
   D4 = {*}
   D3 = {Eastern / Western}
   D2 = {Asia, Europe, Africa, North America, South America, Oceania, Antarctica}
   D1 = {East Asia, Southeast Asia, South Asia, Northeast Asia, Central Asia, West Asia, Eastern Europe, Western Europe, East Africa, West Africa, Southern Africa, Central Africa, North America, South America, Oceania, Antarctica}
   D0 = {Real country code}


*
Eastern
Western
Asia
Europe
Africa
North America
South America
Oceania
Antarctica
Southeast Asia
South Asia
Eastern Europe
East Africa
Southern Africa
North America
South America
Oceania
Antarctica
Northeast Asia
Central Asia
Western Europe
West Africa
Central Africa
CHN, JPN, KOR, PRK, TWN, HKG, MAC, MNG
IDN, VNM, THA, PHL, MYS, SGP, MMR, KHM, LAO, BRN, TLS
IND, PAK, BGD, NPL, LKA, MDV, BTN
RUS, MNG, PRK, JPN, KOR, CHN
KAZ, UZB, KGZ, TKM, TJK
AFG, IRN, IRQ, SYR, JOR, LBN, ISR, PSE, KWT, OMN, QAT, SAU, ARE, YEM, BHR
RUS, UKR, POL, ROU, CZE, HUN, BLR, BGR, SVK, LTU, HRV, SVN, EST, LVA
DEU, FRA, GBR, ITA, ESP, NLD, BEL, AUT, CHE, LUX, IRL, PRT, DNK, NOR, SWE, FIN, ISL, GRC
ETH, SDN, UGA, KEN, TZA, MOZ, BDI, ZMB, RWA, MWI, MDG, ERI, SOM, GAB
NGA, GHA, CIV, SEN, CMR, BFA, NER, MLI, TCD, GNB, GIN, CIV, TGO, BEN, LBR, SLE, GMB, BFA
ZAF, AGO, ZMB, ZWE, MWI, NAM, BWA, LSO, SWZ
COD, CAF, TCD, AGO, CMR, GIN, ZAF, ZMB, TZA
USA, CAN, MEX, BLZ, CRI, SLV, GTM, HND, NIC, PAN
BRA, ARG, COL, PER, VEN, CHL, ECU, BOL, PRY, URY
AUS, NZL, PNG, FJI, SLB, VUT, PLW, KIR, TUV, WSM
ATA

3. StayNight:
   D3 = {*}
   D2 = {Never visit / Ever visit}
   D1 = {First-Time Guests / Rare Guests / Occasional Guests / Frequent Guests}
   D0 = {Actual number of overnight stays}



Part 1.2: Code
When initiating the code, simply type 'python anonymity.py' in the command line interface. Upon execution, you will observe the following result:

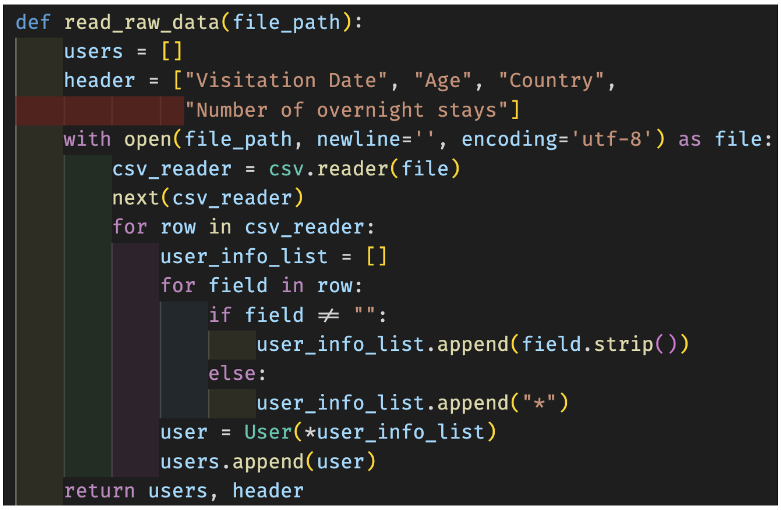


```
 ~/Desktop/Hotel-Booking-Analysis 〉 on main +10   python anonymity.py
Init Total users: 999

95 unsatisfied users were removed
Final Total users: 904

User utility: 904/999 : 90.49%
```

In our dataset, there are 999 users. After applying generalization with a parameter k set to 10, we find that there are 904 valid user records, indicating a user utility of 904. This means that for each record, there are at least 9 other tuples with the same value for the quasi-identifier.

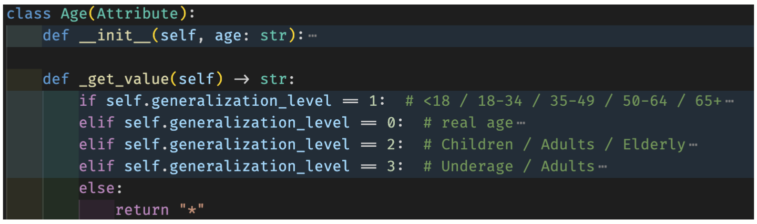
First, we read the raw data from a CSV file. This function parses the file, omitting the header, and iterates through each row to extract user information. Each record consists of 38 fields.

However, for our project, we only utilize specific data fields, which are predefined in the header: 'Visitation Date', 'Age', 'Country', and 'Number of overnight stays'. Once the user data is collected, it is organized into a list of User objects, which align with the provided header. This function returns both the list of users and the header for the resulting table.

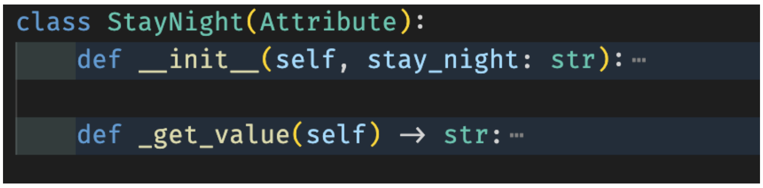


```
def read_raw_data(file_path):
    users = []
    header = ["Visitation Date", "Age", "Country",
              "Number of overnight stays"]
    with open(file_path, newline='', encoding='utf-8') as file:
        csv_reader = csv.reader(file)
        next(csv_reader)
        for row in csv_reader:
            user_info_list = []
            for field in row:
                if field ≠ "":
                    user_info_list.append(field.strip())
                else:
                    user_info_list.append("*")
            user = User(*user_info_list)
            users.append(user)
    return users, header
```

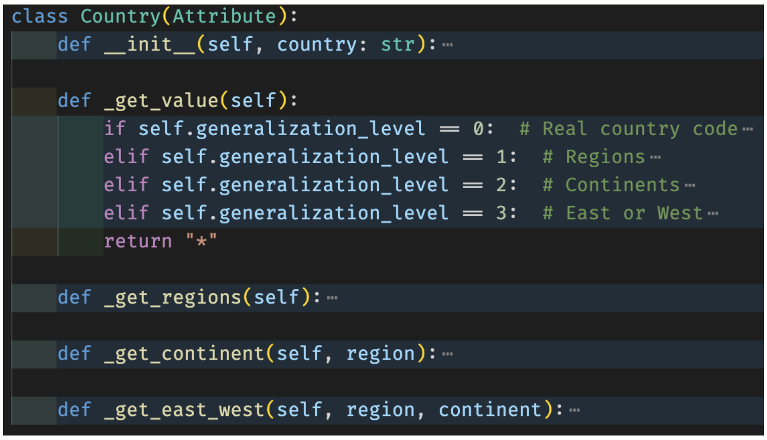
Second, we define classes for attributes such as Age, Country, and StayNight. These classes handle the generalization and distortion of attribute values based on predefined rules and levels. The attributes are generalized to different levels of detail to protect user privacy while still providing useful information for analysis. Each attribute class contains methods to set the generalization level, add or reduce generalization levels, and retrieve the generalized value based on the current level. Additionally, specific logic is implemented for each attribute type to determine the generalized value based on the specified criteria, such as age ranges, geographical regions, and frequency of stays.



```
class Age(Attribute):
    def __init__(self, age: str):⋯

    def _get_value(self) → str:
        if self.generalization_level == 1:  # <18 / 18-34 / 35-49 / 50-64 / 65+⋯
        elif self.generalization_level == 0:  # real age⋯
        elif self.generalization_level == 2:  # Children / Adults / Elderly⋯
        elif self.generalization_level == 3:  # Underage / Adults⋯
        else:
            return "*"
```

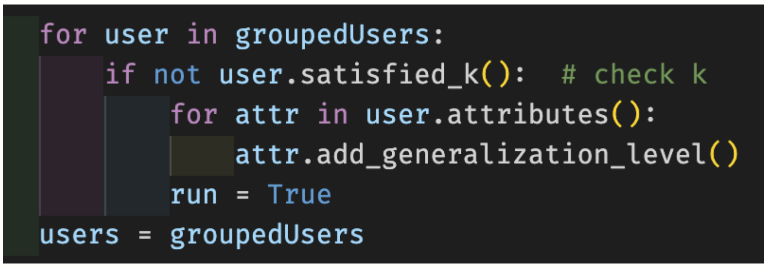


```
class StayNight(Attribute):
    def __init__(self, stay_night: str):⋯

    def _get_value(self) → str:⋯
```



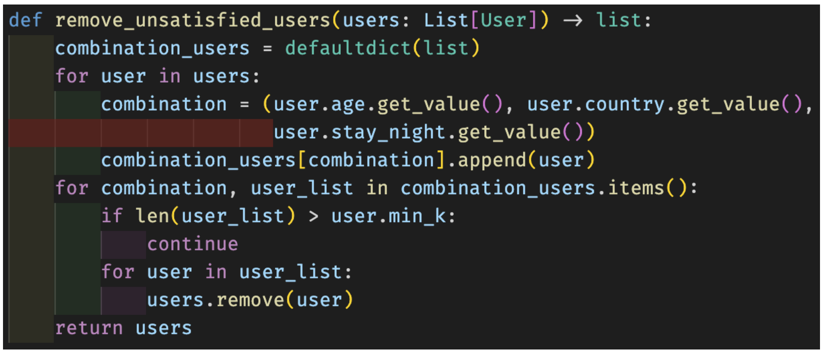
```
class Country(Attribute):
    def __init__(self, country: str):⋯

    def _get_value(self):
        if self.generalization_level == 0:  # Real country code⋯
        elif self.generalization_level == 1:  # Regions⋯
        elif self.generalization_level == 2:  # Continents⋯
        elif self.generalization_level == 3:  # East or West⋯
        return "*"

    def _get_regions(self):⋯

    def _get_continent(self, region):⋯

    def _get_east_west(self, region, continent):⋯
```

A crucial aspect of our process involves generalization, facilitated by the generalizeUsers() function. This function iterates through each user to identify correlations and ensure that the count of similar instances meets the threshold set by parameter k. It also determines whether further generalization is necessary. If so, it advances to the next level of generalization. This iterative process continues until the desired level of generalization is achieved or until no further changes occur. The function operates by merging users with similar attributes, incrementally increasing the generalization level if necessary to satisfy the k-anonymity requirement. This iterative refinement ensures that sensitive information is adequately protected while maintaining data utility.



```
for user in groupedUsers:
    if not user.satisfied_k():  # check k
        for attr in user.attributes():
            attr.add_generalization_level()
        run = True
users = groupedUsers
```

The final step involves removing unsatisfactory users through the remove_unsatisfied_users() function. Here, users are grouped based on their attributes, such as age, country, and number of stay nights. If the count of users within a particular attribute combination falls below the minimum k threshold, indicating insufficient anonymity, those users are eliminated. This ensures that the dataset complies with the k-anonymity requirement, safeguarding individual privacy while preserving data utility.

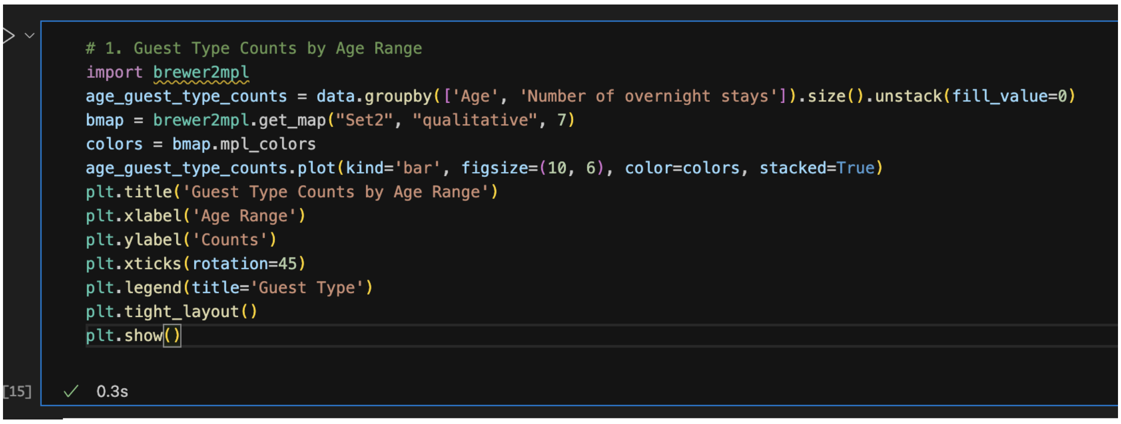


```python
def remove_unsatisfied_users(users: List[User]) -> list:
    combination_users = defaultdict(list)
    for user in users:
        combination = (user.age.get_value(), user.country.get_value(),
                       user.stay_night.get_value())
        combination_users[combination].append(user)
    for combination, user_list in combination_users.items():
        if len(user_list) > user.min_k:
            continue
        for user in user_list:
            users.remove(user)
    return users
```

Part 1.3: Data analysis

We've done some different types of data analysis:

- Guest Type Counts by Age Range



```python
# 1. Guest Type Counts by Age Range
import brewer2mpl
age_guest_type_counts = data.groupby(['Age', 'Number of overnight stays']).size().unstack(fill_value=0)
bmap = brewer2mpl.get_map("Set2", "qualitative", 7)
colors = bmap.mpl_colors
age_guest_type_counts.plot(kind='bar', figsize=(10, 6), color=colors, stacked=True)
plt.title('Guest Type Counts by Age Range')
plt.xlabel('Age Range')
plt.ylabel('Counts')
plt.xticks(rotation=45)
plt.legend(title='Guest Type')
plt.tight_layout()
plt.show()
```

- Time Trend Analysis: Examine visit trends over different periods (e.g., monthly or yearly) to see if there are specific times when visitor numbers increase.
- Regional Visitor Distribution: Analyze the distribution of visitors from different countries/regions to understand which areas are the most popular destinations.
- Relationship Between Time and Guest Type: Analyze the relationship between visit dates and guest types to see if certain times attract specific types of visitors.

And so on.

Part 2: Identity

In the identity part we have two different scenarios, the first one is "MD5 for User Password Hashing" and the second one is "RSA for Email Encryption and Decryption".

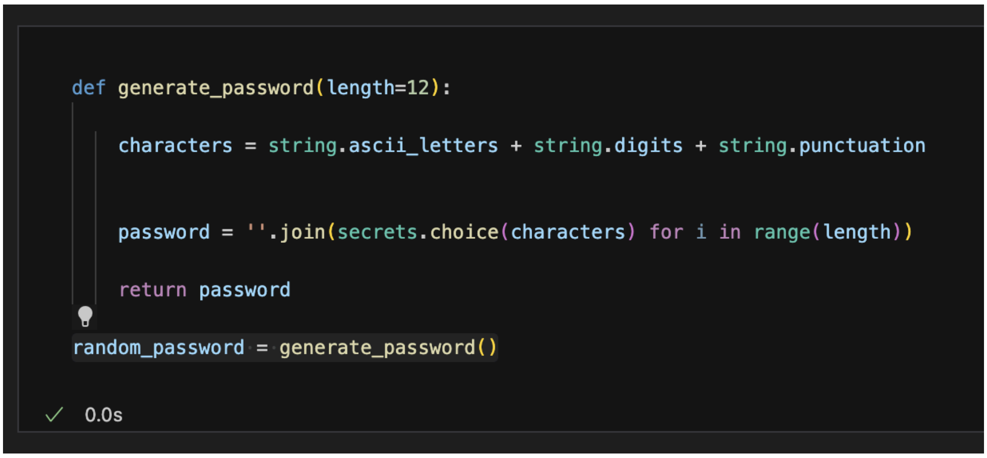
Part 2.1: User Identity Check MD5 for User Password Hashing

We are going to securely store user passwords in the database using MD5 hashing to enhance security during the login process.

Process Flow:

1. User Registration:
    - When a user registers, they provide a password. And for the test data, we just generate a bunch of pwd based on different rules.

This is the core function to generate the password:

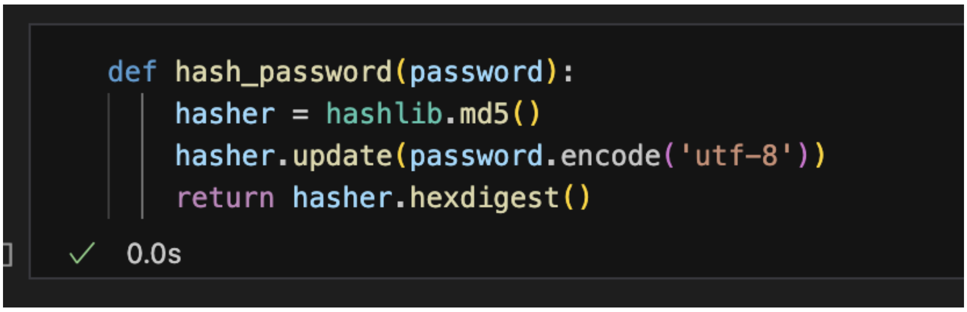


```
def generate_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation

    password = ''.join(secrets.choice(characters) for i in range(length))
    return password
random_password = generate_password()
```

✓ 0.0s

You should be able to see the whole code in the file “psw-generate.ipynb”. This file will generate passwords for each row of the data and write them to the local file.

- The password is hashed using the MD5 algorithm.
  This is the core function:

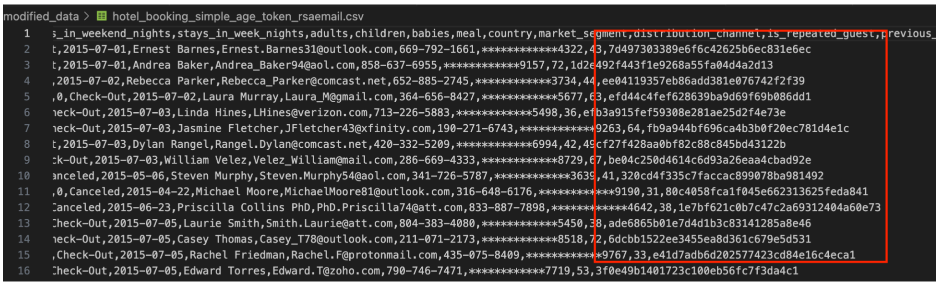


```
def hash_password(password):
    hasher = hashlib.md5()
    hasher.update(password.encode('utf-8'))
    return hasher.hexdigest()
```

✓ 0.0s

This is using MD5 to hash passwords. In the meanwhile, we need to keep in mind MD5 is not safe anymore if we directly use it to encrypt sensitive information. But here we just want to make sure we do the core implementation and explain the logic.

- The MD5 hash of the password (now referred to as a token) is stored in the database alongside the user's other details (e.g., phonenumber).
  You could see the file:



modified_data > hotel_booking_simple_age_token_rsaemail.csv

```
s_in_weekend_nights,stays_in_week_nights,adults,children,babies,meal,country,market_segment,distribution_channel,is_repeated_guest,previous_
t,2015-07-01,Ernest Barnes,Ernest.Barnes31@outlook.com,669-792-1661,************4322,43,7d497303389e6f6c42625b6ec831e6ec
t,2015-07-01,Andrea Baker,Andrea_Baker94@aol.com,858-637-6955,************9157,72,1d2e492f443f1e9268a55fa04d4a2d13
,2015-07-02,Rebecca Parker,Rebecca_Parker@comcast.net,652-885-2745,************3734,44,ee04119357eb86add381e076742f2f39
,0,Check-Out,2015-07-02,Laura Murray,Laura_M@gmail.com,364-656-8427,************5677,63,efd44c4fef628639ba9d69f69b086dd1
neck-Out,2015-07-03,Linda Hines,LHines@verizon.com,713-226-5883,************5498,36,efb3a915fef59308e281ae25d2f4e73e
neck-Out,2015-07-03,Jasmine Fletcher,JFletcher43@xfinity.com,190-271-6743,************9263,64,fb9a944bf696ca4b3b0f20ec781d4e1c
t,2015-07-03,Dylan Rangel,Rangel.Dylan@comcast.net,420-332-5209,************6994,42,49cf27f428aa0bf82c88c845bd43122b
ck-Out,2015-07-03,William Velez,Velez_William@mail.com,286-669-4333,************8729,67,be04c250d4614c6d93a26eaa4cbad92e
anceled,2015-05-06,Steven Murphy,Steven.Murphy54@aol.com,341-726-5787,************3639,41,320cd4f335c7faccac899078ba981492
,0,Canceled,2015-04-22,Michael Moore,MichaelMoore81@outlook.com,316-648-6176,************9190,31,80c4058fca1f045e662313625feda841
Canceled,2015-06-23,Priscilla Collins PhD,PhD.Priscilla74@att.com,833-887-7898,************4642,38,1e7bf621c0b7c47c2a69312404a60e73
Check-Out,2015-07-05,Laurie Smith,Smith.Laurie@att.com,804-383-4080,************5450,38,ade6865b01e7d4d1b3c83141285a8e46
neck-Out,2015-07-05,Casey Thomas,Casey_T78@outlook.com,211-071-2173,************8518,72,6dcbb1522ee3455ea8d361c679e5d531
,Check-Out,2015-07-05,Rachel Friedman,Rachel.F@protonmail.com,435-075-8409,************9767,33,e41d7adb6d202577423cd84e16c4eca1
Check-Out,2015-07-05,Edward Torres,Edward.T@zoho.com,790-746-7471,************7719,53,3f0e49b1401723c100eb56fc7f3da4c1
```

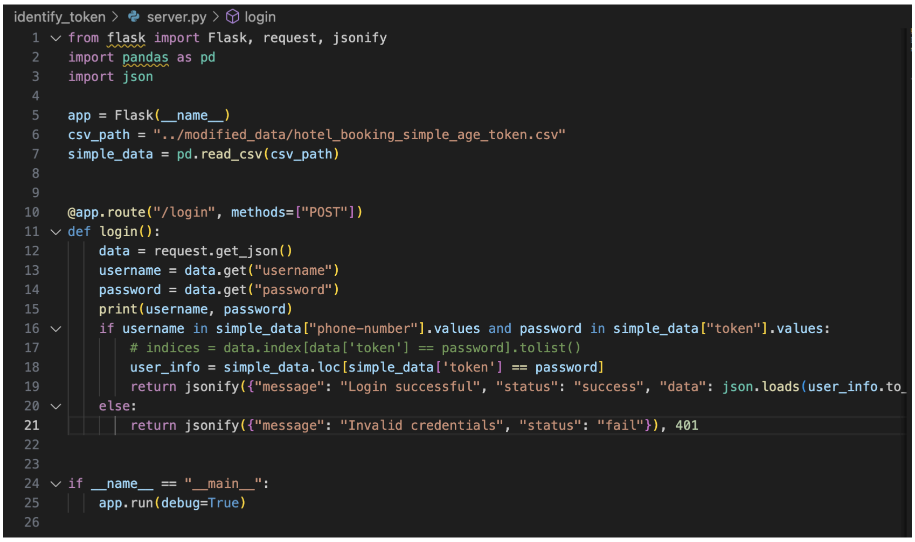
After running the “identify_token_generate.ipynb” file, we added a new column called “token” to the CSV file.

2. User Identity Check:
   - During the login process, the user submits their phone number and password.

Use this to send the request:
We can use this to do the identity Check:

```
curl --location 'http://127.0.0.1:5000/login' \
--header 'Content-Type: application/json' \
--data '{
  "username": "669-792-1661",
  "password": "3c5678493dd6298f1ffc178bed9905ae"
}'
```

- The password provided by the user is hashed on the server using the same MD5 algorithm. And also you can find the original password in the “random_passwords.txt” just for test cases, for real cases, we don’t have this file and the one who needs to remember the password is the user.
- The server retrieves the stored MD5 hash associated with the username from the database. In this picture below, we start the server and receive the data from the user and retrieve the data of a certain user and return it back.



```python
from flask import Flask, request, jsonify
import pandas as pd
import json

app = Flask(__name__)
csv_path = "../modified_data/hotel_booking_simple_age_token.csv"
simple_data = pd.read_csv(csv_path)


@app.route("/login", methods=["POST"])
def login():
    data = request.get_json()
    username = data.get("username")
    password = data.get("password")
    print(username, password)
    if username in simple_data["phone-number"].values and password in simple_data["token"].values:
        # indices = data.index[data['token'] == password].tolist()
        user_info = simple_data.loc[simple_data['token'] == password]
        return jsonify({"message": "Login successful", "status": "success", "data": json.loads(user_info.to_
    else:
        return jsonify({"message": "Invalid credentials", "status": "fail"}), 401


if __name__ == "__main__":
    app.run(debug=True)

```

- The server compares the submitted hash with the stored hash.
    - If they match, authentication is successful.
    - If they do not match, the login is rejected, and the user is informed of an authentication failure.

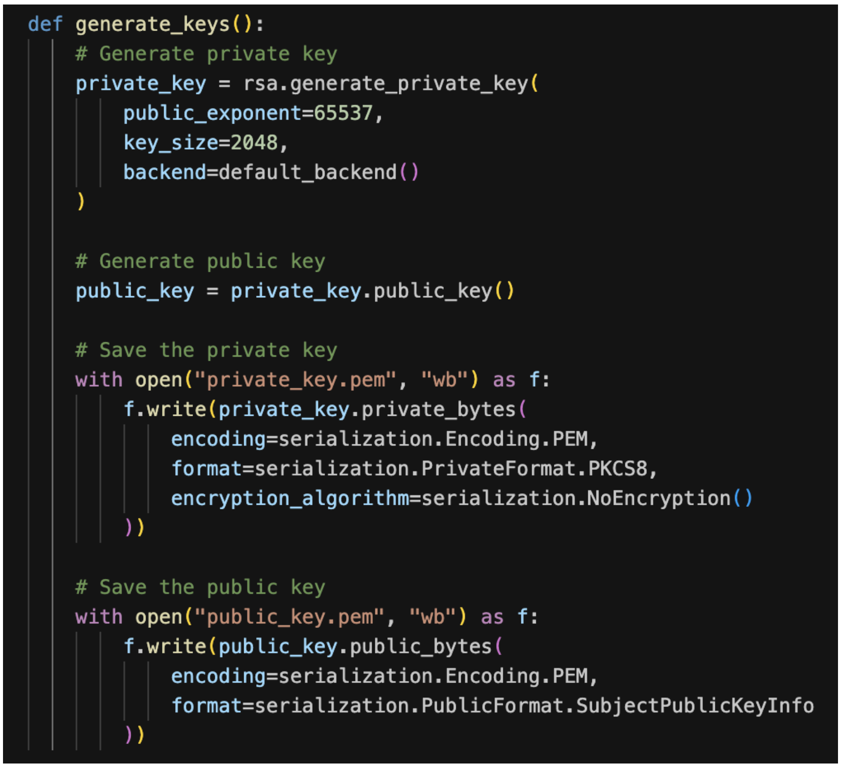
3. Security Considerations(the real world usage):
    - Employ salting: To enhance security, a unique salt should be added to each password before it is hashed. This prevents the use of precomputed hash databases (rainbow tables) to reverse the hashes.
    - Limitations: It’s important to note that MD5 is no longer recommended for security-critical applications due to its vulnerabilities and susceptibility to collision attacks. Consider using more secure hashing algorithms like SHA-256.

Part 2.2: Two Trusted Data Centers Share Data Using RSA for Email Encryption and Decryption

Protect user emails by encrypting them using RSA encryption, allowing secure data exchange between two trusted data centers.

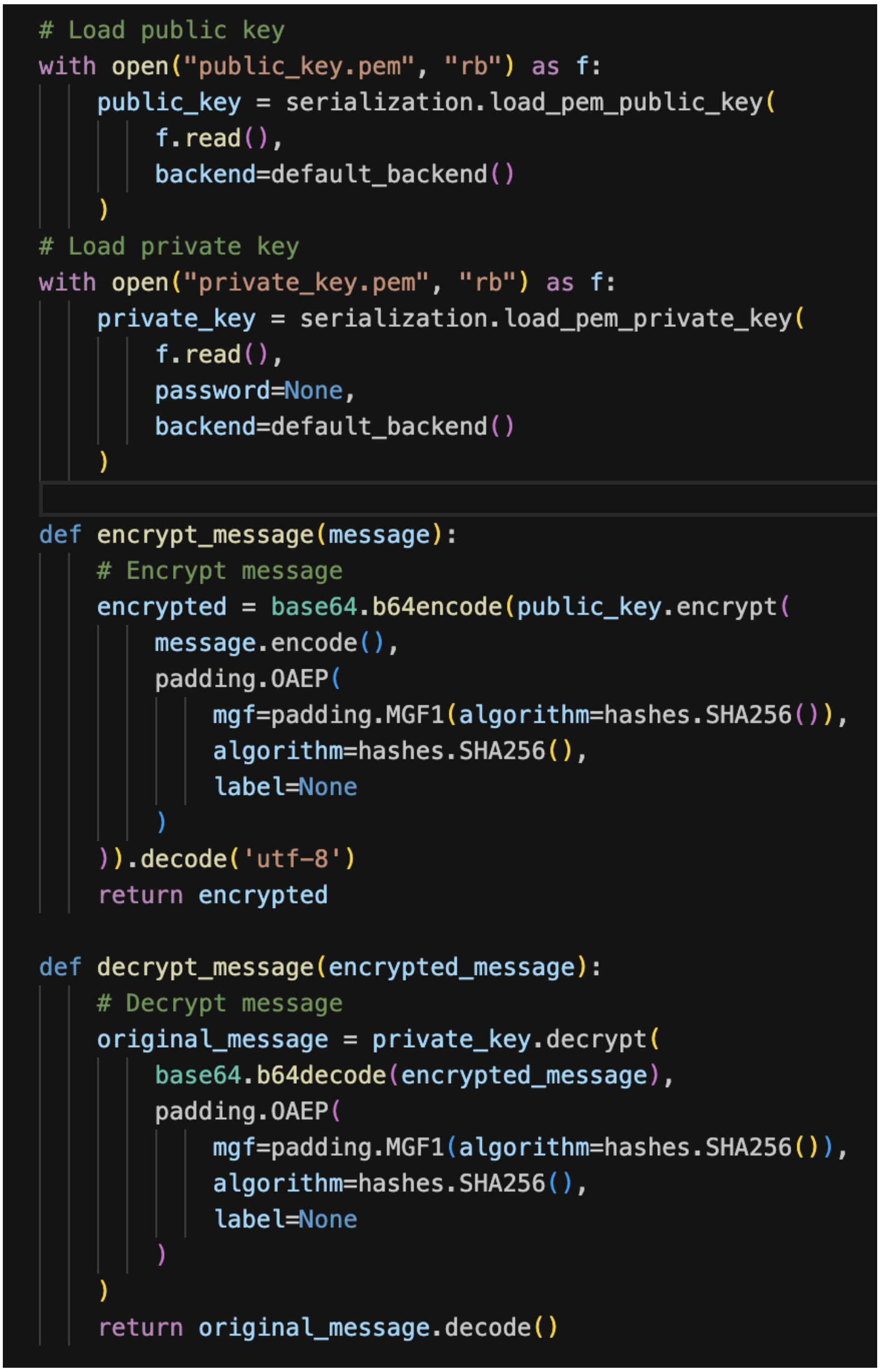
Process Flow:

1. Email Encryption at Data Center 1:
   - Each user has a public and private RSA key pair. We can generate the PK and SK using this function:



```python
def generate_keys():
    # Generate private key
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )

    # Generate public key
    public_key = private_key.public_key()

    # Save the private key
    with open("private_key.pem", "wb") as f:
        f.write(private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption()
        ))

    # Save the public key
    with open("public_key.pem", "wb") as f:
        f.write(public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        ))
```

   - When a user’s email is to be stored in Data Center 1, it is encrypted using the user’s public key. We modify the origin data for our test case, this is the core encrypt function:



```python
# Load public key
with open("public_key.pem", "rb") as f:
    public_key = serialization.load_pem_public_key(
        f.read(),
        backend=default_backend()
    )
# Load private key
with open("private_key.pem", "rb") as f:
    private_key = serialization.load_pem_private_key(
        f.read(),
        password=None,
        backend=default_backend()
    )

def encrypt_message(message):
    # Encrypt message
    encrypted = base64.b64encode(public_key.encrypt(
        message.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )).decode('utf-8')
    return encrypted

def decrypt_message(encrypted_message):
    # Decrypt message
    original_message = private_key.decrypt(
        base64.b64decode(encrypted_message),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return original_message.decode()
```

- The encrypted email is stored in Data Center 1 along with other user data.

2. Data Transfer to Data Center 2:
    - Data Center 2, being a trusted entity, can request and retrieve user data from Data Center 1.
    - The retrieved data includes the RSA encrypted emails.
3. Email Decryption at Data Center 2:
    - Data Center 2 uses the stored private RSA keys (secured with the necessary permissions and protocols) to decrypt the user emails.

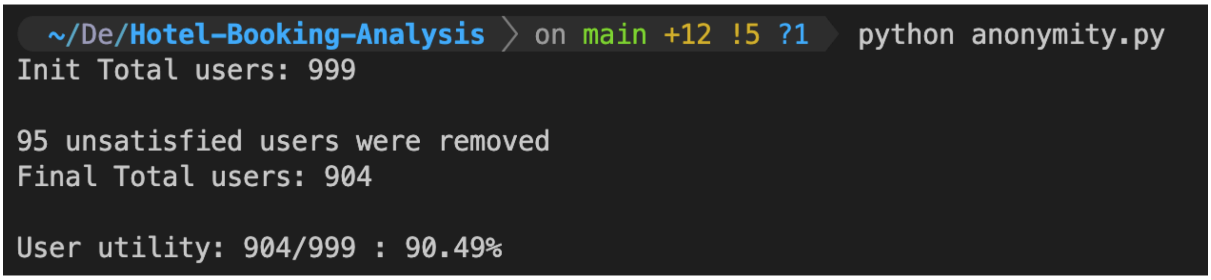
- Emails are decrypted only when needed, ensuring that data remains secure during storage and transfer.

4. Security and Access Management:
    - All private keys are stored securely, ensuring they are accessible only to authorized systems and personnel.
    - Implement strict access controls and audit logs to monitor access and decryption activities in Data Center 2.
    - Ensure all transmitted data between data centers is secured using SSL/TLS to prevent interception.
5. Key Management:
    - Periodically update and rotate RSA keys to mitigate risks associated with key exposure.
    - Employ a robust key management system to handle the generation, storage, and destruction of keys in a secure manner.

Testing

Part 1: k-anonymity

To test the code, simply type 'python anonymity.py' in the command line interface. Upon execution, you will observe the result below. In our dataset, there are 999 users. After applying generalization with a parameter k set to 10, we find that there are 904 valid user records, indicating a user utility of 904. This means that for each record, there are at least 9 other tuples with the same value for the quasi-identifier.



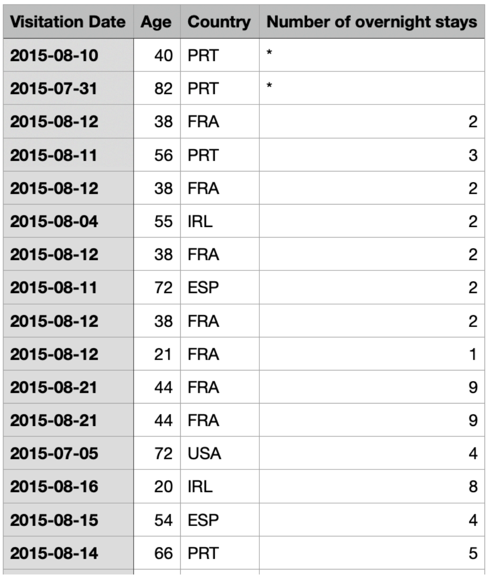
```
 ~/De/Hotel-Booking-Analysis  on main +12 !5 ?1   python anonymity.py
Init Total users: 999

95 unsatisfied users were removed
Final Total users: 904

User utility: 904/999 : 90.49%
```

After this process, you can navigate to the result folder to examine both the input dataset before generalization and the output dataset after generalization. This allows for comparison and verification of the anonymization process.

```
result
  input.csv
  output.csv
```

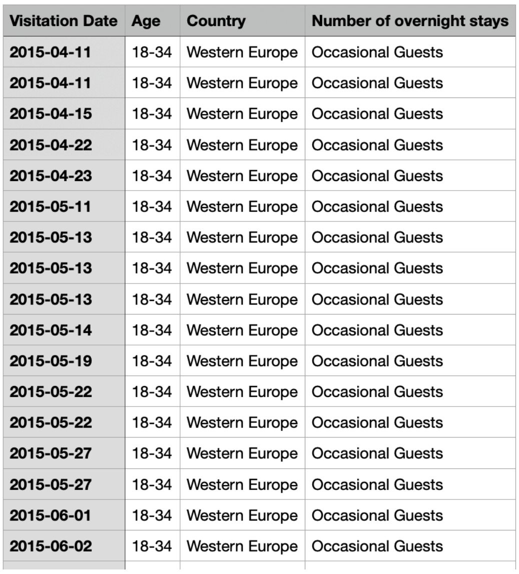
Displayed below are excerpts from both the input and output datasets. The input dataset reflects the data prior to generalization, while the output dataset guarantees that each entry has at least k-1 identical entries, excluding the sensitive attribute. It's noteworthy that the first column, 'Visitation Date,' pertains to our sensitive attribute.

input



Visitation Date	Age	Country	Number of overnight stays
2015-08-10	40	PRT	*
2015-07-31	82	PRT	*
2015-08-12	38	FRA	2
2015-08-11	56	PRT	3
2015-08-12	38	FRA	2
2015-08-04	55	IRL	2
2015-08-12	38	FRA	2
2015-08-11	72	ESP	2
2015-08-12	38	FRA	2
2015-08-12	21	FRA	1
2015-08-21	44	FRA	9
2015-08-21	44	FRA	9
2015-07-05	72	USA	4
2015-08-16	20	IRL	8
2015-08-15	54	ESP	4
2015-08-14	66	PRT	5

output

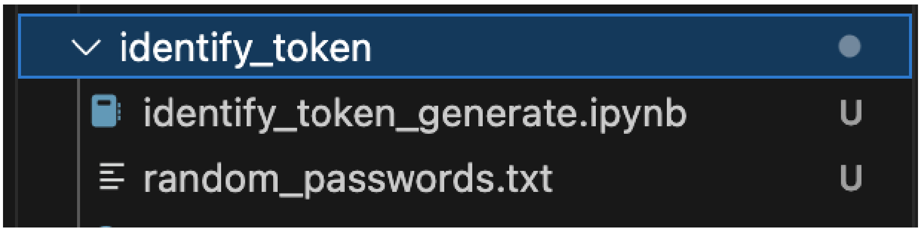


Visitation Date	Age	Country	Number of overnight stays
2015-04-11	18-34	Western Europe	Occasional Guests
2015-04-11	18-34	Western Europe	Occasional Guests
2015-04-15	18-34	Western Europe	Occasional Guests
2015-04-22	18-34	Western Europe	Occasional Guests
2015-04-23	18-34	Western Europe	Occasional Guests
2015-05-11	18-34	Western Europe	Occasional Guests
2015-05-13	18-34	Western Europe	Occasional Guests
2015-05-13	18-34	Western Europe	Occasional Guests
2015-05-13	18-34	Western Europe	Occasional Guests
2015-05-14	18-34	Western Europe	Occasional Guests
2015-05-19	18-34	Western Europe	Occasional Guests
2015-05-22	18-34	Western Europe	Occasional Guests
2015-05-22	18-34	Western Europe	Occasional Guests
2015-05-27	18-34	Western Europe	Occasional Guests
2015-05-27	18-34	Western Europe	Occasional Guests
2015-06-01	18-34	Western Europe	Occasional Guests
2015-06-02	18-34	Western Europe	Occasional Guests

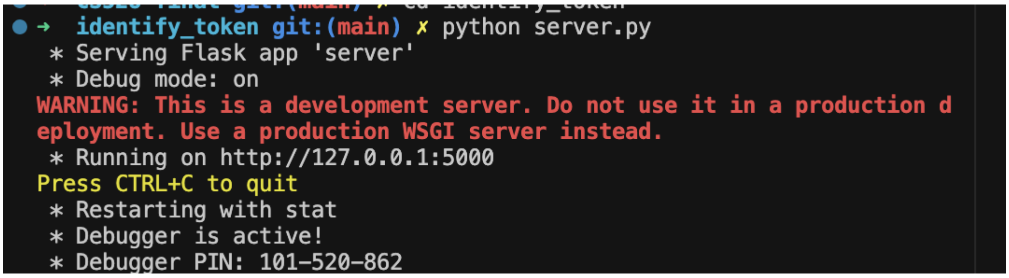
Part 2: Identity

Part 2.1 User Identity Check MD5 for User Password Hashing:

- Go to the “identify_token” folder:



- First of all, we need to generate random passwords and tokens using “identify_token_generate.ipynb”. It will generate the “random_password” file and add the “token” column to every user and save it as the csv to the output.
- Run the server using “python ./server.py ”:



```
● → identify_token git:(main) ✗ python server.py
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production d
eployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 101-520-862
```

- And after that we can use this to do the identity Check:

  curl --location 'http://127.0.0.1:5000/login' \

  --header 'Content-Type: application/json' \

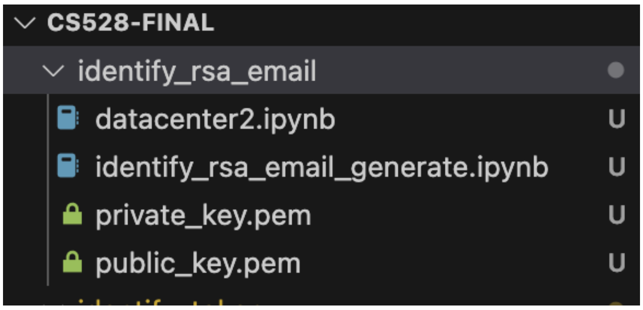
  --data '{

```
  "username": "669-792-1661",
  "password": "3c5678493dd6298f1ffc178bed9905ae"
}'
```

The user needs to use md5 to hash the password once send the request to the server.

Part 2.2 Two Trusted Data Centers Share Data Using RSA for Email Encryption and Decryption:

- Go to the “identify_rsa_email” folder:

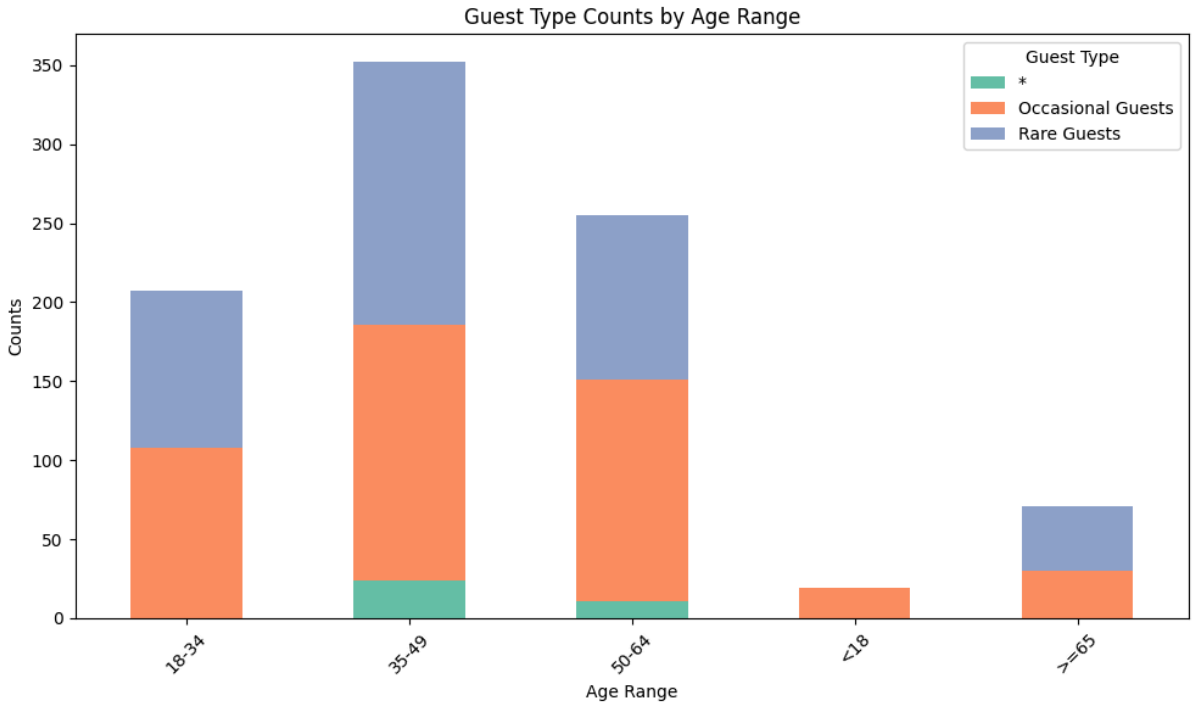


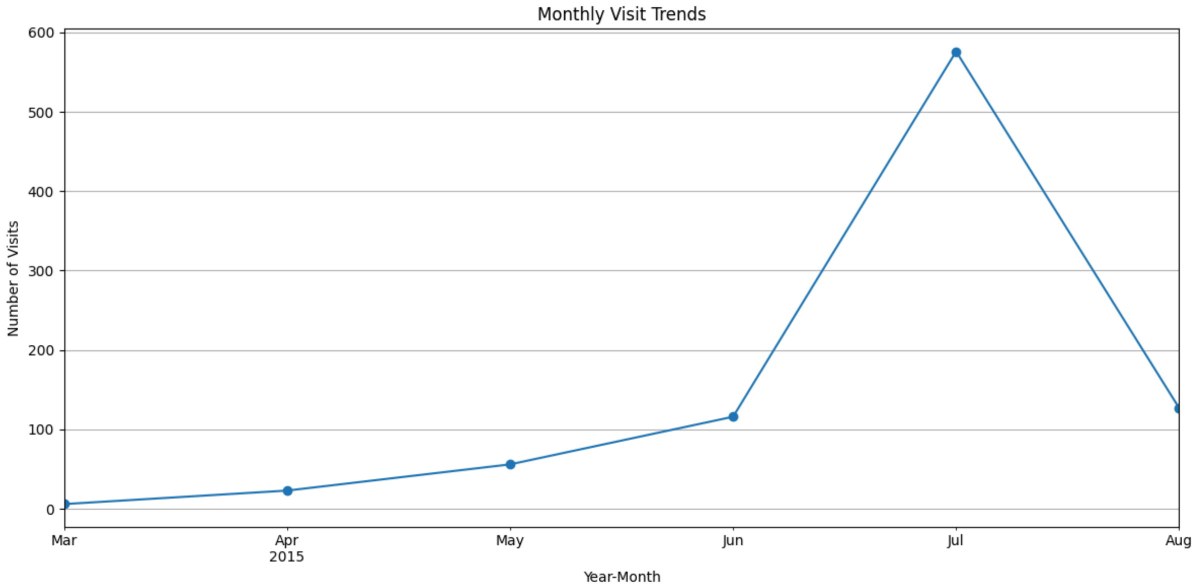
```
∨ CS528-FINAL
  ∨ identify_rsa_email
      datacenter2.ipynb                     U
      identify_rsa_email_generate.ipynb     U
      private_key.pem                       U
      public_key.pem                        U
```

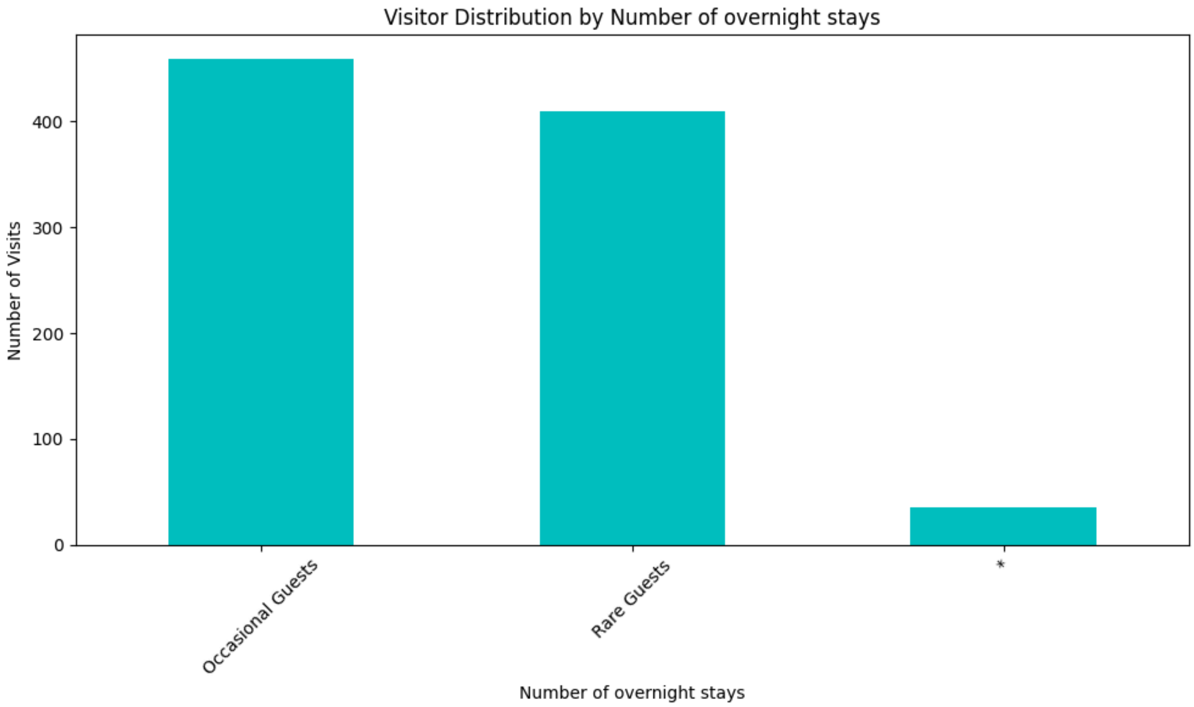
- Run the “identify_rsa_email_generate.ipynb” file, this script will generate public key and secret key and save them to the local files. And also it will use the public key to encrypt the email of users using the public key to simulate the users’ encrypt behavior. It will generate a new csv which the email column will be replaced by an encrypted one.
- Run the “datacenter2.ipynb” file. This script will retrieve the data which is generated by datacenter1 and try to decrypt it by RSA using the secret key.

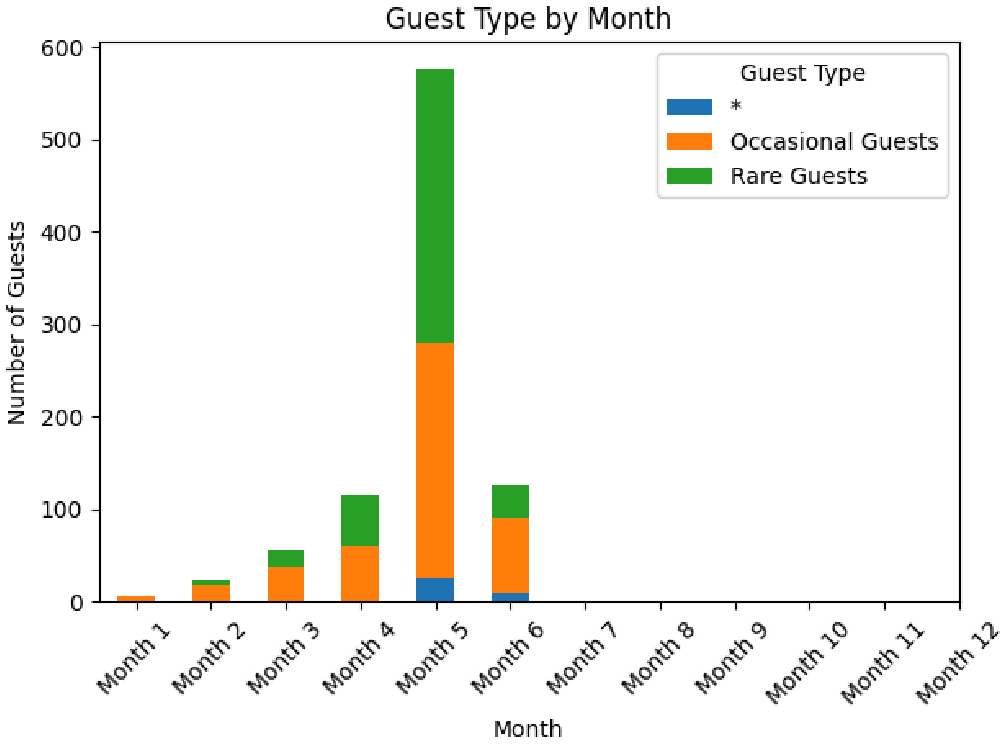
Results

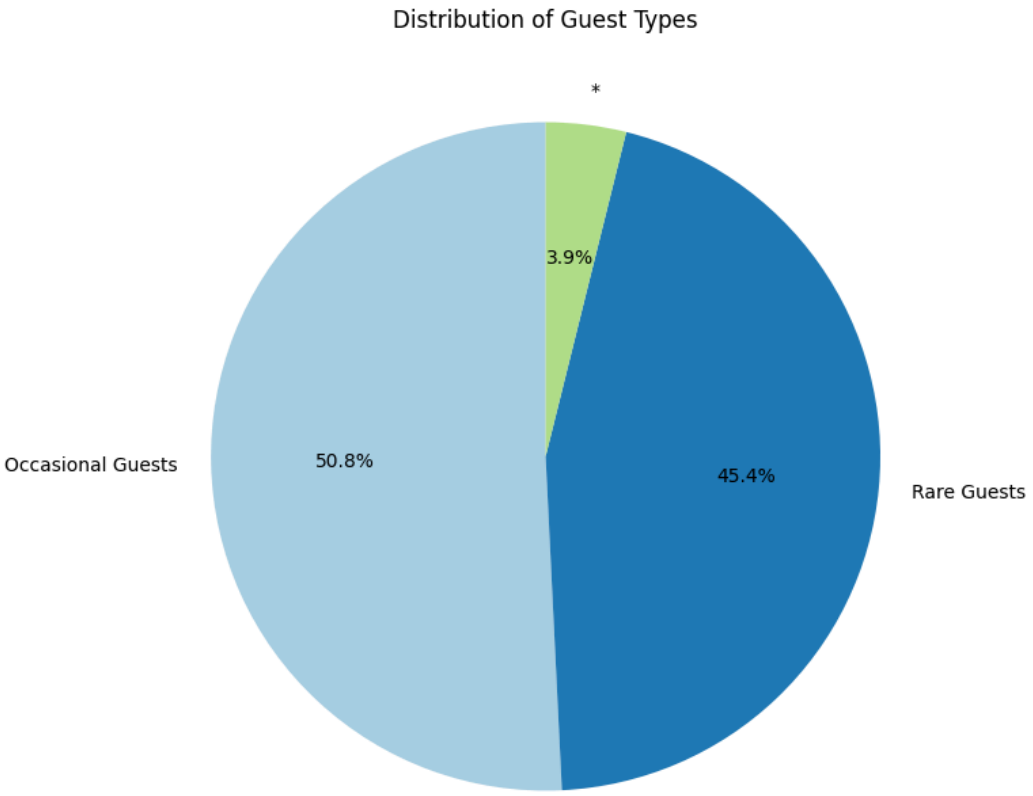
Part 1: Data analysis based on k-anonymity:

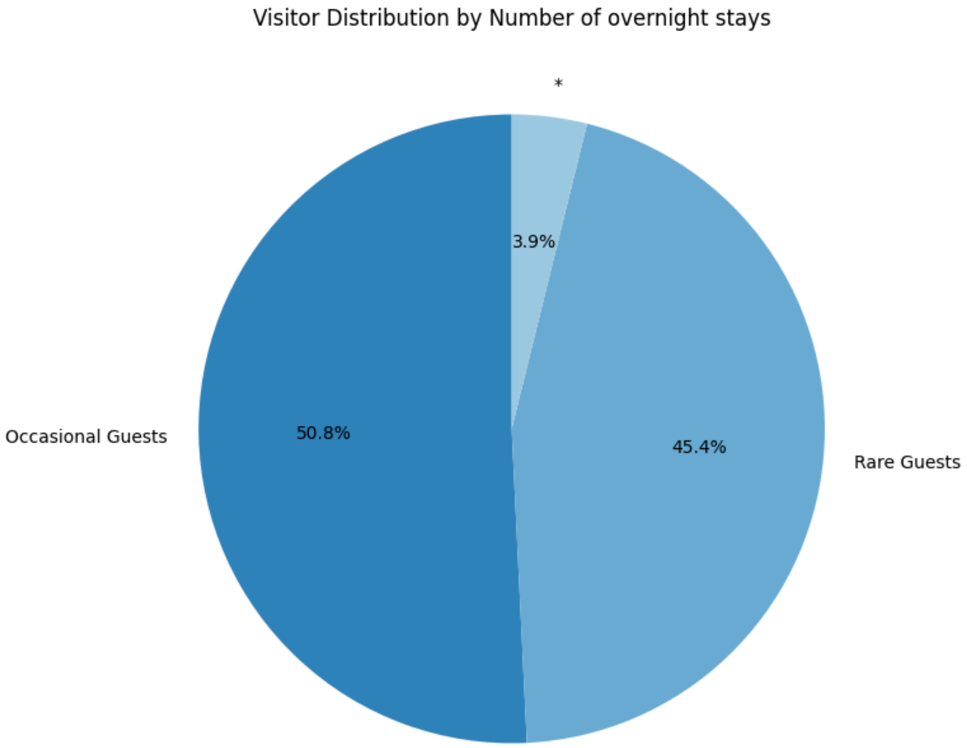


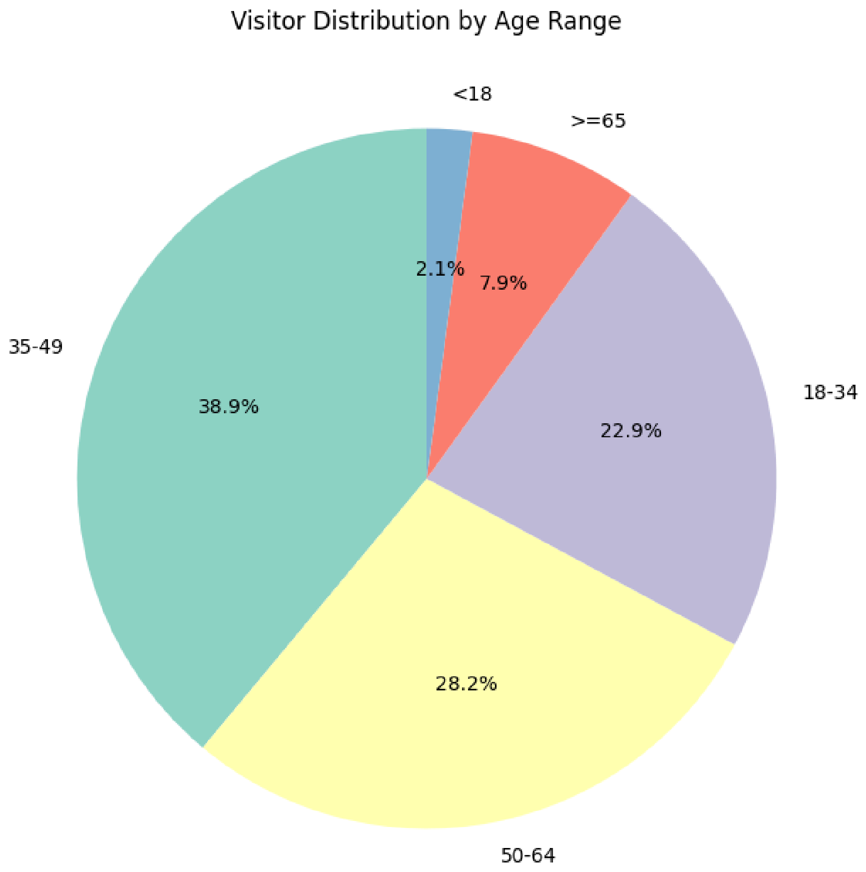



Visitor Distribution by Number of overnight stays
Number of Visits
0
100
200
300
400
Occasional Guests
Rare Guests
*
Number of overnight stays


Guest Type by Month
Number of Guests
0
100
200
300
400
500
600
Guest Type
*
Occasional Guests
Rare Guests
Month 1
Month 2
Month 3
Month 4
Month 5
Month 6
Month 7
Month 8
Month 9
Month 10
Month 11
Month 12
Month


Distribution of Guest Types
*
3.9%
Occasional Guests
50.8%
45.4%
Rare Guests

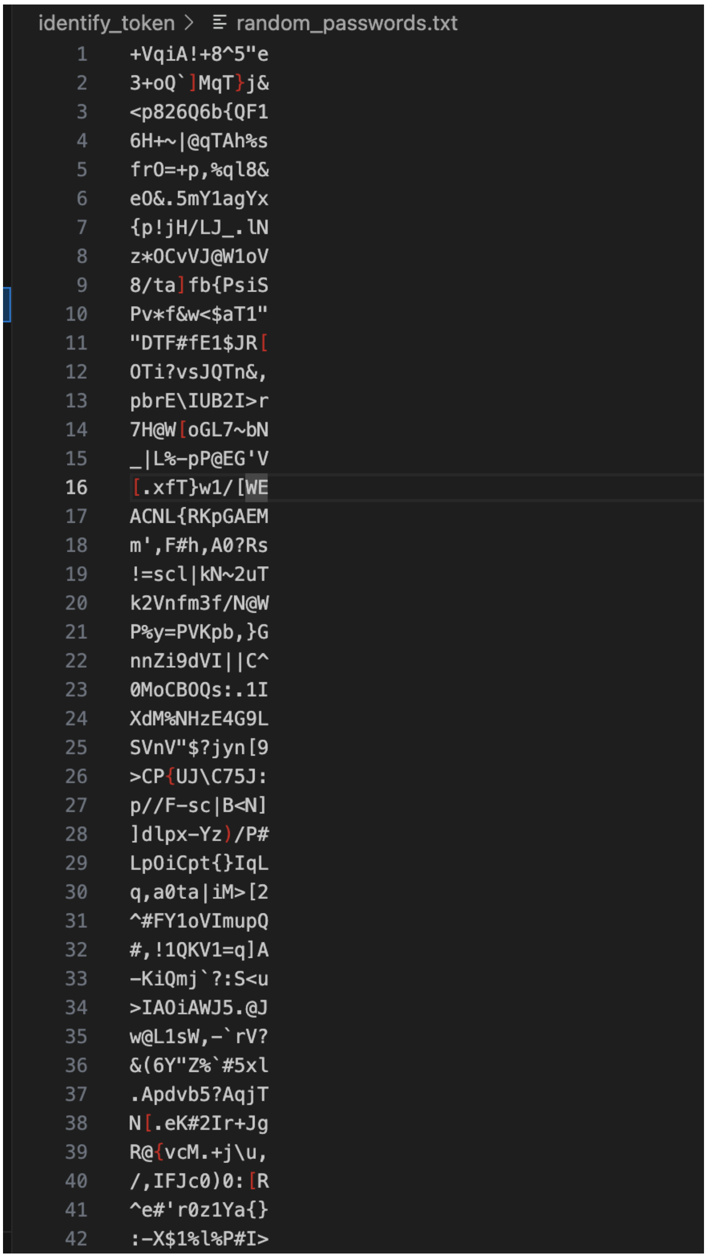

Visitor Distribution by Number of overnight stays
*
3.9%
Occasional Guests
50.8%
45.4%
Rare Guests


Visitor Distribution by Age Range
<18
>=65
2.1%
7.9%
35-49
38.9%
18-34
22.9%
28.2%
50-64

Part 2: Identity

Part 2.1:

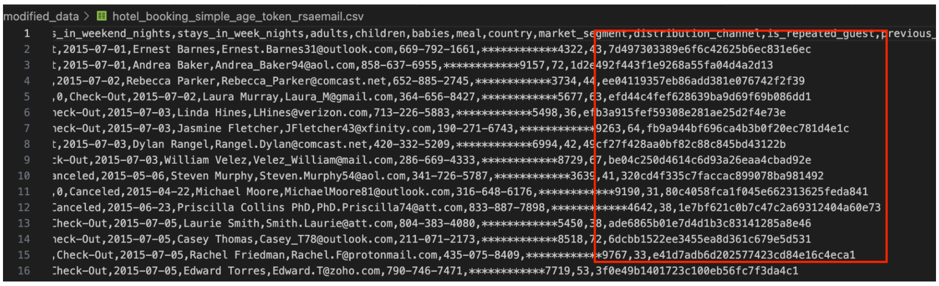
- Generated original password:



identify_token > random_passwords.txt

```
+VqiA!+8^5"e
3+oQ`]MqT}j&
<p826Q6b{QF1
6H+~|@qTAh%s
fr0=+p,%ql8&
e0&.5mY1agYx
{p!jH/LJ_.lN
z*0CvVJ@W1oV
8/ta]fb{PsiS
Pv*f&w<$aT1"
"DTF#fE1$JR[
0Ti?vsJQTn&,
pbrE\IUB2I>r
7H@W[oGL7~bN
_|L%-pP@EG'V
[.xfT}w1/[WE
ACNL{RKpGAEM
m',F#h,A0?Rs
!=scl|kN~2uT
k2Vnfm3f/N@W
P%y=PVKpb,}G
nnZi9dVI||C^
0MoCBOQs:.1I
XdM%NHzE4G9L
SVnV"$?jyn[9
>CP{UJ\C75J:
p//F-sc|B<N]
]dlpx-Yz)/P#
Lp0iCpt{}IqL
q,a0ta|iM>[2
^#FY1oVImupQ
#,!1QKV1=q]A
-KiQmj`?:S<u
>IAOiAWJ5.@J
w@L1sW,-`rV?
&(6Y"Z%`#5xl
.Apdvb5?AqjT
N[.eK#2Ir+Jg
R@{vcM.+j\u,
/,IFJc0)0:[R
^e#'r0z1Ya{}
:-X$1%l%P#I>
```

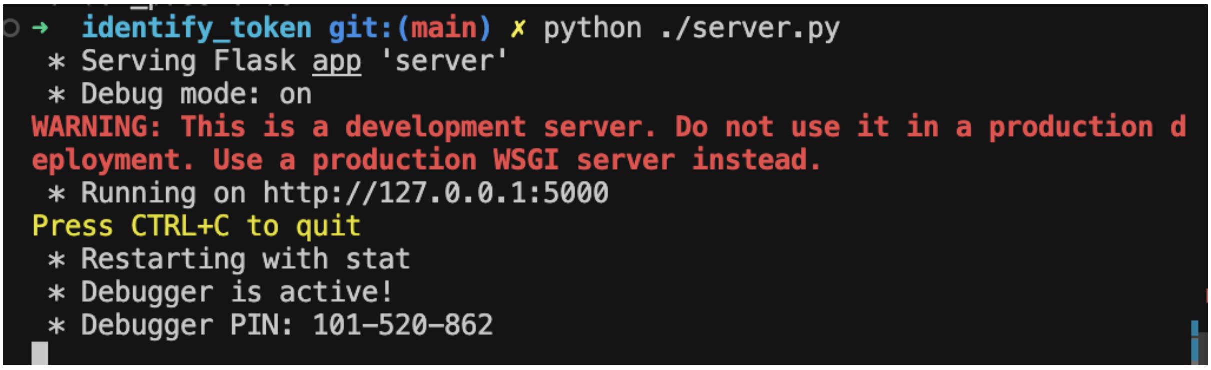
- Csv Data with token:



modified_data > hotel_booking_simple_age_token_rsaemail.csv

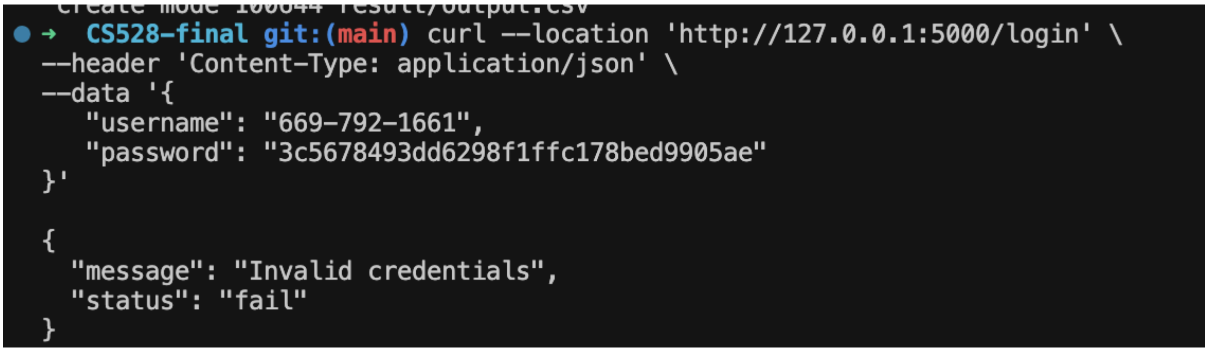
```
s_in_weekend_nights,stays_in_week_nights,adults,children,babies,meal,country,market_segment,distribution_channel,is_repeated_guest,previous_
t,2015-07-01,Ernest Barnes,Ernest.Barnes31@outlook.com,669-792-1661,************4322,43,7d497303389e6f6c42625b6ec831e6ec
t,2015-07-01,Andrea Baker,Andrea_Baker94@aol.com,858-637-6955,************9157,72,1d2e492f443f1e9268a55fa04d4a2d13
,2015-07-02,Rebecca Parker,Rebecca_Parker@comcast.net,652-885-2745,************3734,44,ee04119357eb86add381e076742f2f39
,0,Check-Out,2015-07-02,Laura Murray,Laura_M@gmail.com,364-656-8427,************5677,63,efd44c4fef628639ba9d69f69b086dd1
neck-Out,2015-07-03,Linda Hines,LHines@verizon.com,713-226-5883,************5498,36,efb3a915fef59308e281ae25d2f4e73e
neck-Out,2015-07-03,Jasmine Fletcher,JFletcher43@xfinity.com,190-271-6743,************9263,64,fb9a944bf696ca4b3b0f20ec781d4e1c
t,2015-07-03,Dylan Rangel,Rangel.Dylan@comcast.net,420-332-5209,************6994,42,49cf27f428aa0bf82c88c845bd43122b
ck-Out,2015-07-03,William Velez,Velez_William@mail.com,286-669-4333,************8729,67,be04c250d4614c6d93a26eaa4cbad92e
anceled,2015-05-06,Steven Murphy,Steven.Murphy54@aol.com,341-726-5787,************3639,41,320cd4f335c7faccac899078ba981492
,0,Canceled,2015-04-22,Michael Moore,MichaelMoore81@outlook.com,316-648-6176,************9190,31,80c4058fca1f045e662313625feda841
Canceled,2015-06-23,Priscilla Collins PhD,PhD.Priscilla74@att.com,833-887-7898,************4642,38,1e7bf621c0b7c47c2a69312404a60e73
Check-Out,2015-07-05,Laurie Smith,Smith.Laurie@att.com,804-383-4080,************5450,38,ade6865b01e7d4d1b3c83141285a8e46
neck-Out,2015-07-05,Casey Thomas,Casey_T78@outlook.com,211-071-2173,************8518,72,6dcbb1522ee3455ea8d361c679e5d531
,Check-Out,2015-07-05,Rachel Friedman,Rachel.F@protonmail.com,435-075-8409,************9767,33,e41d7adb6d202577423cd84e16c4eca1
Check-Out,2015-07-05,Edward Torres,Edward.T@zoho.com,790-746-7471,************7719,53,3f0e49b1401723c100eb56fc7f3da4c1
```

- Server running:



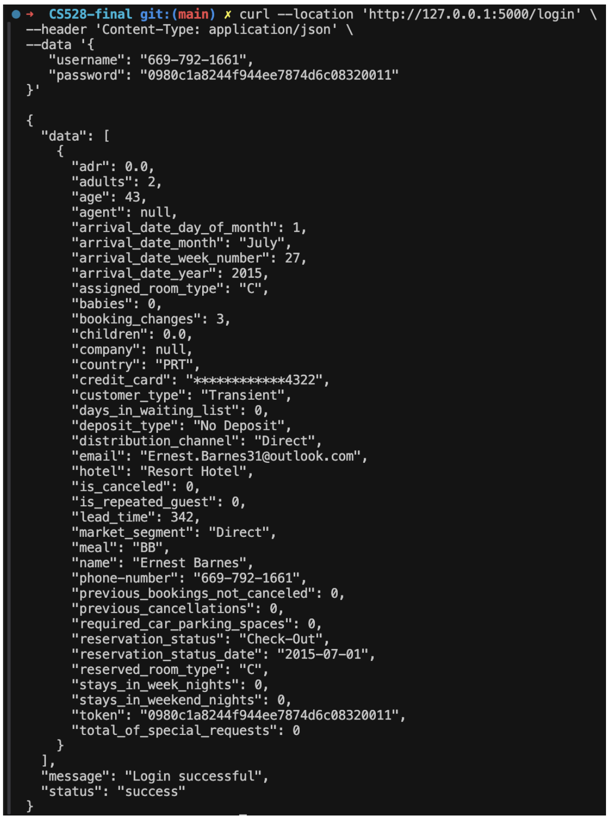
```
○ ➜  identify_token git:(main) ✗ python ./server.py
 * Serving Flask app 'server'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production d
eployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 101-520-862
```

- User Identity Check:
  - Failed Check:



```
  create mode 100644 result/output.csv
● ➜  CS528-final git:(main) curl --location 'http://127.0.0.1:5000/login' \
  --header 'Content-Type: application/json' \
  --data '{
     "username": "669-792-1661",
     "password": "3c5678493dd6298f1ffc178bed9905ae"
  }'

  {
    "message": "Invalid credentials",
    "status": "fail"
  }
```

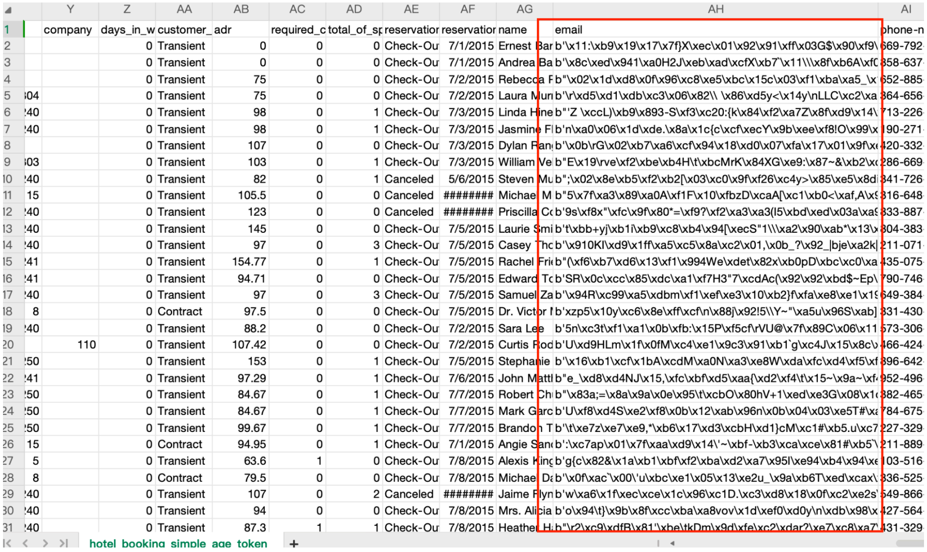
  - Success Check:



```
● ➜  CS528-final git:(main) ✗ curl --location 'http://127.0.0.1:5000/login' \
  --header 'Content-Type: application/json' \
  --data '{
     "username": "669-792-1661",
     "password": "0980c1a8244f944ee7874d6c08320011"
  }'

  {
    "data": [
      {
        "adr": 0.0,
        "adults": 2,
        "age": 43,
        "agent": null,
        "arrival_date_day_of_month": 1,
        "arrival_date_month": "July",
        "arrival_date_week_number": 27,
        "arrival_date_year": 2015,
        "assigned_room_type": "C",
        "babies": 0,
        "booking_changes": 3,
        "children": 0.0,
        "company": null,
        "country": "PRT",
        "credit_card": "************4322",
        "customer_type": "Transient",
        "days_in_waiting_list": 0,
        "deposit_type": "No Deposit",
        "distribution_channel": "Direct",
        "email": "Ernest.Barnes31@outlook.com",
        "hotel": "Resort Hotel",
        "is_canceled": 0,
        "is_repeated_guest": 0,
        "lead_time": 342,
        "market_segment": "Direct",
        "meal": "BB",
        "name": "Ernest Barnes",
        "phone-number": "669-792-1661",
        "previous_bookings_not_canceled": 0,
        "previous_cancellations": 0,
        "required_car_parking_spaces": 0,
        "reservation_status": "Check-Out",
        "reservation_status_date": "2015-07-01",
        "reserved_room_type": "C",
        "stays_in_week_nights": 0,
        "stays_in_weekend_nights": 0,
        "token": "0980c1a8244f944ee7874d6c08320011",
        "total_of_special_requests": 0
      }
    ],
    "message": "Login successful",
    "status": "success"
  }
```

Part 2.2:

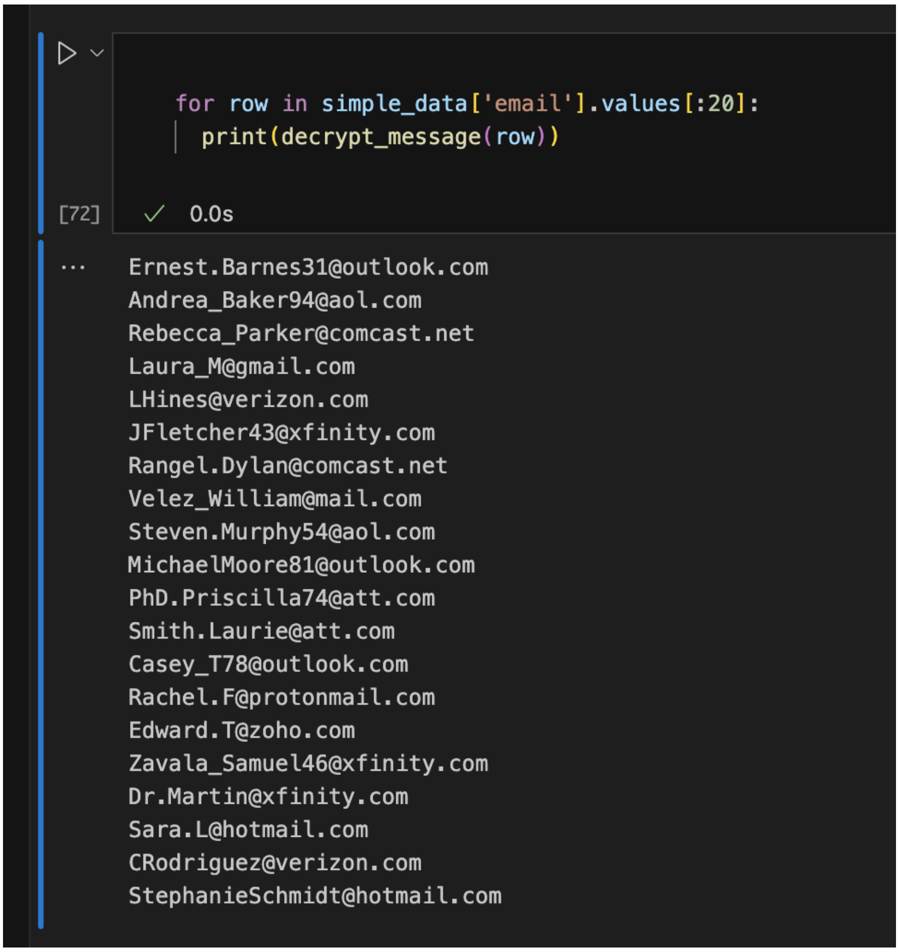
As you can see, we use RSA to encrypt email using the public key for in the data center 1.



	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
1		company	days_in_w	customer_	adr	required_c	total_of_sp	reservation	reservation	name	email	phone-n
2			0	Transient	0	0	0	Check-Ou	7/1/2015	Ernest Bar	b'\x11:\xb9\x19\x17\x7f}X\xec\x01\x92\x91\xff\x03G$\x90\xf9\	669-792
3			0	Transient	0	0	0	Check-Ou	7/1/2015	Andrea Ba	b'\x8c\xed\x941\xa0H2J\xeb\xad\xcfX\xb7`\x11\\\x8f\xb6A\xf0	858-637
4			0	Transient	75	0	0	Check-Ou	7/2/2015	Rebecca F	b"\x02\x1d\xd8\x0f\x96\xc8\xe5\xbc\x15c\x03\xf1\xba\xa5_\x	652-885
5	304		0	Transient	75	0	0	Check-Ou	7/2/2015	Laura Mur	b'\r\xd5\xd1\xdb\xc3\x06\x82\\ \x86\xd5y<\x14y\nLLC\xc2\xa	364-656
6	240		0	Transient	98	0	1	Check-Ou	7/3/2015	Linda Hine	b"'Z \xccL)\xb9\x893-S\xf3\xc20:{k\x84\xf2\xa7Z\x8f\xd9\x14\	713-226
7	240		0	Transient	98	0	1	Check-Ou	7/3/2015	Jasmine Fl	b'n\xa0\x06\x1d\xde.\x8a\x1c{c\xcf\xecY\x9b\xee\xf8!O\x99\x	190-271
8			0	Transient	107	0	0	Check-Ou	7/3/2015	Dylan Ran	b'\x0b\rG\x02\xb7\xa6\xcf\x94\x18\xd0\x07\xfa\x17\x01\x9f\xe	420-332
9	303		0	Transient	103	0	1	Check-Ou	7/3/2015	William Ve	b"E\x19\rve\xf2\xbe\xb4H\t\xbcMrK\x84XG\xe9:\x87~&\xb2\xc	286-669
10	240		0	Transient	82	0	1	Canceled	5/6/2015	Steven Mu	b";\x02\x8e\xb5\xf2\xb2[\x03\xc0\x9f\xf26\xc4y>\x85\xe5\x8di	841-726
11	15		0	Transient	105.5	0	0	Canceled	########	Michael M	b"5\x7f\xa3\x89\xa0A\xf1F\x10\xfbzD\xcaA[\xc1\xb0<\xaf,A\x9	816-648
12	240		0	Transient	123	0	0	Canceled	########	Priscilla Cc	b'9s\xf8x"\xfc\x9f\x80*=\xf9?\xf2\xa3\xa3(I5\xbd\xed\x03a\xa9	833-887
13	240		0	Transient	145	0	0	Check-Ou	7/5/2015	Laurie Smi	b't\xbb+yj\xb1i\xb9\xc8\xb4\x94[\xecS"1\\\xa2\x90\xab*\x13\x	804-383
14	240		0	Transient	97	0	3	Check-Ou	7/5/2015	Casey Thc	b'\x910KI\xd9\x1ff\xa5\xc5\x8a\xc2\x01,\x0b_?\x92_|bje\xa2k|	211-071
15	241		0	Transient	154.77	0	1	Check-Ou	7/5/2015	Rachel Frie	b"(\xf6\xb7\xd6\x13\xf1\x994We\xdet\x82x\xb0pD\xbc\xc0\xa	435-075
16	241		0	Transient	94.71	0	0	Check-Ou	7/5/2015	Edward Tc	b'SR\x0c\xcc\x85\xdc\xa1\xf7H3"7\xcdAc(\x92\x92\xbd$~Ep\	790-746
17	240		0	Transient	97	0	3	Check-Ou	7/5/2015	Samuel Za	b'\x94R\xc99\xa5\xdbm\xf1\xef\xe3\x10\xb2}f\xfa\xe8\xe1\x19	649-384
18	8		0	Contract	97.5	0	0	Check-Ou	7/5/2015	Dr. Victor I	b'xzp5\x10y\xc6\x8e\xff\xcf\n\x88j\x92!5\\Y~"\xa5u\x96S\xab]	831-430
19	240		0	Transient	88.2	0	0	Check-Ou	7/2/2015	Sara Lee	b'5n\xc3t\xf1\xa1\x0b\xfb:\x15P\xf5cf\rVU@\x7f\x89C\x06\x11	573-306
20		110	0	Transient	107.42	0	0	Check-Ou	7/2/2015	Curtis Rod	b'U\xd9HLm\x1f\x0fM\xc4\xe1\x9c3\x91\xb1`g\xc4J\x15\x8c\x	466-424
21	250		0	Transient	153	0	1	Check-Ou	7/5/2015	Stephanie	b'\x16\xb1\xcf\x1bA\xcdM\xa0N\xa3\xe8W\xda\xfc\xd4\xf5\xf	896-642
22	241		0	Transient	97.29	0	1	Check-Ou	7/6/2015	John Mattl	b"e_\xd8\xd4NJ\x15,\xfc\xbf\xd5\xaa{\xd2\xf4\t\x15~\x9a~\xf4	952-496
23	250		0	Transient	84.67	0	1	Check-Ou	7/7/2015	Robert Ch	b"\x83a;=\x8a\x9a\x0e\x95\t\xcbO\x80hV+1\xed\xe3G\x08\x1	882-465
24	250		0	Transient	84.67	0	1	Check-Ou	7/7/2015	Mark Garc	b'U\xf8\xd4S\xe2\xf8\x0b\x12\xab\x96n\x0b\x04\x03\xe5T#\xa	784-675
25	250		0	Transient	99.67	0	1	Check-Ou	7/7/2015	Brandon T	b'\t\xe7z\xe7\xe9,*\xb6\x17\xd3\xcbH\xd1}cM\xc1#\xb5.u\xc7	227-329
26	15		0	Contract	94.95	0	1	Check-Ou	7/1/2015	Angie San	b':\xc7ap\x01\x7f\xaa\xd9\x14\'~\xbf-\xb3\xca\xce\x81#\xb5`\	211-889
27	5		0	Transient	63.6	1	0	Check-Ou	7/8/2015	Alexis King	b'g{c\x82&\x1a\xb1\xbf\xf2\xba\xd2\xa7\x95l\xe94\xb4\x94\xe	103-516
28	8		0	Contract	79.5	0	0	Check-Ou	7/8/2015	Michael Da	b'\x0f\xac`\x00\'u\xbc\xe1\x05\x13\xe2u_\x9a\xb6T\xed\xcax\	836-525
29	240		0	Transient	107	0	2	Canceled	########	Jaime Flyn	b'w\xa6\x1f\xec\xce\x1c\x96\xc1D.\xc3\xd8\x18\x0f\xc2\xe2s'	549-866
30	240		0	Transient	94	0	0	Check-Ou	7/8/2015	Mrs. Alicia	b'o\x94\t}\x9b\x8f\xcc\xba\xa8vov\x1d\xef0\xd0y\n\xdb\x98\x	427-564
31	240		0	Transient	87.3	1	1	Check-Ou	7/8/2015	Heather H	b"\r2\xc9\xdfB\x81'\xbe\tkDm\x9d\xfe\xc2\xdar?\xe7\xc8\xa7\	431-329

hotel booking simple age token

After that, in the data center 2, we can use paired secret key to decrypt the email:



```
for row in simple_data['email'].values[:20]:
    print(decrypt_message(row))
```

[72] ✓ 0.0s

```
Ernest.Barnes31@outlook.com
Andrea_Baker94@aol.com
Rebecca_Parker@comcast.net
Laura_M@gmail.com
LHines@verizon.com
JFletcher43@xfinity.com
Rangel.Dylan@comcast.net
Velez_William@mail.com
Steven.Murphy54@aol.com
MichaelMoore81@outlook.com
PhD.Priscilla74@att.com
Smith.Laurie@att.com
Casey_T78@outlook.com
Rachel.F@protonmail.com
Edward.T@zoho.com
Zavala_Samuel46@xfinity.com
Dr.Martin@xfinity.com
Sara.L@hotmail.com
CRodriguez@verizon.com
StephanieSchmidt@hotmail.com
```

Summary

The project implements comprehensive data privacy measures using k-anonymity and encryption techniques to protect sensitive guest information. The primary goal is to ensure privacy preservation by making each guest's data indistinguishable among at least k-1 others, particularly focusing on sensitive attributes such as visitation dates. This approach safeguards against the risks posed by disclosing accommodation dates and leverages quasi-identifiers like age, nationality, and stay duration to enhance dataset complexity.

In the second part of the project, there are two crucial initiatives to ensure data security during user interactions and data transfers:

1. Part 2.1: Utilizes MD5 hashing for securing user passwords during registration and login processes. This method involves hashing passwords and storing them as tokens in the database to prevent the exposure of plaintext passwords. However, due to known vulnerabilities in the MD5 algorithm, such as susceptibility to collision attacks, there is a consideration to shift towards more robust algorithms like SHA-256 to meet contemporary security demands.
2. Part 2.2: Implements RSA encryption to protect user email data across two trusted data centers. This segment emphasizes strong key management and secure data transfer protocols. Emails are encrypted with the user's public RSA key and decrypted only within the secure environment of these data centers using private keys under strict access control.

Together, these strategies underscore the project's commitment to maintaining guest confidentiality and trust while enabling meaningful data analysis and ensuring compliance with data protection standards.

References:

Dataset: https://www.kaggle.com/datasets/mojtaba142/hotel-booking/data