About Dataset:

The data set consists of New York City Taxi trip reports in the Year 2013, which was released under the FOIL (The Freedom of Information Law) and made public by Chris Whong (https://chriswhong.com/open-data/foil_nyc_taxi/).

The data set is 93 MB compressed, uncompressed 384 MB, namely taxi-data-sortedsmall.csv.bz2.

Setting up the Environment:

Step 1: Import Necessary Libraries

pandas: A powerful Python data analysis toolkit used for working with data structures and performing data analysis.

numpy: A fundamental package for scientific computing with Python, often used alongside pandas for numerical operations.

Code:

```
import pandas as pd
import numpy as np
```

Step 2: Read the CSV File

The script assumes the presence of a CSV file named taxi-data-sorted-small.csv located at a specific path (/content/). This file presumably contains taxi trip data.

pd.read_csv(file_path): This function reads the CSV file from the given file path into a pandas DataFrame. A DataFrame is a 2-dimensional labeled data structure with columns of potentially different types.

```
file_path = "/content/taxi-data-sorted-small.csv"
df = pd.read_csv(file_path)
```

Step 3: Set Column Names

The script specifies a list of column names that correspond to various attributes of taxi trips, such as pickup and dropoff locations, times, distances, fares, and payments.

Assigning df.columns = columns replaces the existing column names in the DataFrame df with the ones listed in the columns array. This step is important for ensuring the data is correctly labeled for further analysis.

code:

```
columns = [
    "medallion", "hack_license", "pickup_datetime", "dropoff_datetime",
    "trip_time_in_secs", "trip_distance", "pickup_longitude", "pickup_latitude",
    "dropoff_longitude", "dropoff_latitude", "payment_type", "fare_amount",
    "surcharge", "mta_tax", "tip_amount", "tolls_amount", "total_amount"
]
df.columns = columns
```

Step 4: Display the DataFrame
df.head(): This method returns the first five rows of the DataFrame df. It's a useful function for quickly testing if your data has been loaded correctly or to get a quick snapshot of the dataset.
code:
print(df.head())

Understanding the Data
The dataset appears to represent detailed information about taxi trips, including:

Identifiers: medallion (a unique identifier for the taxi), hack_license (a unique identifier for the taxi driver).
Timestamps: pickup_datetime, dropoff_datetime.
Trip Details: trip_time_in_secs, trip_distance, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude.
Payment Information: payment_type, fare_amount, surcharge, mta_tax, tip_amount, tolls_amount, total_amount.

Example 1:
Encryption and Decryption with RSA on a Large Dataset

Step 1: Setup RSA Encryption Environment
Key Generation
The first part of the setup involves generating a pair of RSA keys—a private key and a public key. These keys are essential for the encryption and decryption processes.
The generate_rsa_key_pair function creates a new RSA private key using a specified key size (2048 bits in this example) and public exponent (65537, a common choice for RSA). It then generates the corresponding public key from the private key.
The private key is used for decryption, while the public key is used for encryption. It's crucial to securely store the private key since anyone with access to it can decrypt the encrypted data.
Encryption and Decryption Functions
The encrypt_data function takes the public key and a piece of data (as a string), encrypts the data using the RSA public key with OAEP padding and SHA-256 as the hash algorithm, and returns the encrypted data.
The decrypt_data function takes the private key and encrypted data, decrypts the data using the corresponding private key with the same padding and hash algorithm, and returns the decrypted data as a string.

Step 2: Encrypt Sensitive Data in the DataFrame
With the encryption setup complete, the next step is to apply encryption to sensitive data within a pandas DataFrame.
The DataFrame df contains several columns with sensitive information such as fare_amount, tip_amount, and total_amount.
The encrypt_column function applies the encrypt_data function to each value in a specified column of the DataFrame, effectively encrypting the data. This function converts each value to a string before encryption, as the encryption function operates on byte strings.
For demonstration purposes, only the first five records of the DataFrame are encrypted to avoid the computational expense of encrypting a large dataset. This is done by selecting the first five rows using df.head(5) and applying encryption to the selected columns.

Step 3: Decrypt an Encrypted Value
Finally, to access the original value of the encrypted data:
The decrypt_data function is used with the private key to decrypt a value from one of the encrypted columns.
As an example, the first encrypted value in the encrypted_fare_amount column is decrypted, demonstrating how to retrieve the original fare amount from the encrypted data.
Important Considerations
Security: RSA encryption provides a secure way to protect sensitive information. However, managing and protecting the private key is crucial, as anyone with access to it can decrypt the encrypted data.
Performance: RSA encryption can be computationally intensive, especially for large datasets. Encrypting only necessary data and considering the use of more efficient encryption schemes or hardware acceleration for large-scale applications is important.
Compliance: Encrypting sensitive data helps comply with data protection regulations. Still, it's essential to ensure that the entire data handling process, from collection to storage and transfer, meets regulatory requirements.
This demonstration provides a basic understanding of using RSA encryption and decryption with pandas DataFrames, showcasing a way to protect sensitive data within large datasets.

Results:

The result "Decrypted fare amount: 27.0" indicates the outcome of decrypting an encrypted value in the dataset, specifically for a taxi trip's fare amount. Here's the context and explanation of what this result means in the process you've been working through:

Encryption Phase: Initially, you encrypted certain pieces of data in the NYC Taxi dataset, including fare amounts, to protect sensitive information. This was done using RSA encryption, which ensures that the data can only be read by someone with the appropriate private key. The fare amount for at least one of the taxi trips was encrypted, transforming it from a readable value (e.g., "27.0") into an encrypted format that's not human-readable.

Transmission/Storage (Implicit): Although not explicitly shown in the code snippets, the encrypted data represents a state where it could be safely transmitted over public networks or

stored in a database without exposing the sensitive fare information to unauthorized access. This step is crucial for data privacy and security, especially when handling personal or sensitive data.

Decryption Phase: The final step involves decrypting the previously encrypted fare amount using the corresponding private key of the RSA pair that was used for encryption. The process reverses the encryption, converting the data back into its original, readable form.

The result "Decrypted fare amount: 27.0" showcases the successful decryption of an encrypted fare amount, confirming that:

The encryption process preserved the original value without alteration.
The decryption process correctly reverted the encrypted data back to its original state, using the appropriate private key.
The integrity of sensitive data was maintained throughout the process, demonstrating a secure method for handling such information.

Example 2:
Anomaly Detection with Isolation Forest

Initializing the Isolation Forest Model
Isolation Forest is a popular machine learning algorithm for anomaly detection. It isolates anomalies instead of profiling normal data points, making it efficient and scalable, especially for datasets with many features.

Step 1: Feature Selection
The first step involves selecting the features that will be used to detect anomalies. In this case, fare_amount and trip_distance are chosen because they are likely to highlight unusual or irregular taxi trips, such as those that are too costly for the distance traveled or vice versa.

Code:
features = df[['fare_amount', 'trip_distance']]

This line creates a new DataFrame features containing only the columns fare_amount and trip_distance from the original DataFrame df.

Step 2: Anomaly Detection with Isolation Forest
Initializing the Isolation Forest Model
n_estimators specifies the number of base estimators (trees) in the ensemble. More trees can improve the performance but also increase computation time.
contamination is an estimate of the proportion of outliers in the dataset. Setting it to 0.01 indicates that 1% of the data is expected to be anomalous.

random_state ensures reproducibility by setting a seed for the random number generator used by the model.Fitting the Model and Predicting Anomalies. Next train the Isolation Forest model on the selected features (fare_amount and trip_distance). After fitting, the model is used to predict anomalies in the data. The prediction results are stored in a new column anomaly in the original DataFrame df. In this column, -1 represents an outlier (anomalous trip), and 1 indicates an inlier (normal trip).

code:

```
from sklearn.ensemble import IsolationForest

model = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)

model.fit(features)

df['anomaly'] = model.predict(features)
```

Step 3:
Analyzing Detected Anomalies

Once anomalies have been identified, the next step is to examine these unusual data points to understand their characteristics and possibly the reason they were flagged as anomalies.
The code filters the DataFrame to include only the records flagged as anomalies (where anomaly == -1) and prints the number of detected anomalies.
Finally, the script displays a few of the detected anomalies by printing the fare_amount, trip_distance, and the pickup and dropoff timestamps. This can help in understanding why these particular trips were considered unusual, whether it be due to discrepancies in fare amount relative to trip distance or other factors.

Code:

```
anomalies = df[df['anomaly'] == -1]
print(f"Detected {len(anomalies)} anomalies out of {len(df)} records.")

print(anomalies[['fare_amount', 'trip_distance', 'pickup_datetime', 'dropoff_datetime']].head())
```

Result:

The result indicates that the Isolation Forest algorithm detected 19,911 anomalies out of a total of 1,999,998 records in the dataset. This represents approximately 1% of the data, which aligns with the contamination parameter set during the model initialization. The contamination parameter is an estimate of the proportion of outliers in the dataset, and here it seems to have been set to capture a small fraction of the data as anomalies.

The anomalies detected, as shown in the provided subset, have peculiar characteristics that justify their identification as outliers:

Zero or Near-Zero Trip Distance: Each of the listed anomalies involves a trip where the trip_distance is either zero or nearly zero (0.02). In the context of taxi trips, it's unusual for a fare to be charged for no distance traveled, suggesting that these records might represent canceled trips, data entry errors, or other irregular situations.

Identical Pickup and Dropoff Timestamps: The pickup_datetime and dropoff_datetime are the same for several of these records, indicating that these trips have a duration of zero minutes. This further supports the notion that these trips are anomalies, as it's highly unlikely for a taxi trip to begin and end at the exact same time under normal circumstances.

High Fares for No Distance: Some of the anomalies involve high fare amounts (e.g., 259.0, 75.0) for trips with no distance traveled. This is an unusual occurrence since taxi fares are typically correlated with the distance of the trip or the duration in cases of slow traffic. These records could indicate situations where a minimum fare is charged despite the trip not taking place, or they might be errors or fraudulent entries.

Overall, the anomalies detected by the Isolation Forest algorithm highlight records that deviate significantly from typical taxi trip patterns. Identifying such outliers is crucial for multiple reasons:

Data Cleaning: Outliers may indicate errors in data collection or entry that need to be corrected.
Fraud Detection: Anomalies could signify fraudulent activities that require further investigation.
Operational Insights: Analyzing outliers can provide insights into unusual but legitimate scenarios, helping to understand customer behavior and operational challenges.