Task A – Data Preparation and Database Integration

GD612 Assessment 2 supporting files: https://github.com/AIgulKZNZ/GD612.git

a. For my Assessment 2, I will utilize the dataset titled *Bitcoin_Prices_Minutes_2024* from Kaggle. Brief info about dataset:

Unix Timestamp - This is the unix timestamp or also known as "Epoch Time". Use this to convert to your local timezone
Date - This timestamp is in UTC datetime
Symbol - The symbol for which the timeseries data refers
Open - This is the opening price of the time period
High - This is the highest price of the time period
Low - This is the lowest price of the time period
Close - This is the closing price of the time period
Volume (Crypto) - This is the volume in the transacted Ccy. Ie. For BTC/USDT, this is in BTC amount
Volume Base Ccy - This is the volume in the base/converted ccy. Ie. For BTC/USDT, this is in USDT amount
Trade Count - This is the unique number of trades for the given time period

To begin, I will import the Pandas library, which is an essential tool in Python for data manipulation and analysis. Pandas offers robust data structures and functions that are ideal for working with structured data, enabling us to efficiently load, manipulate, and analyze data from various sources, including CSV files, Excel spreadsheets, and SQL databases. I will use Pandas' read_csv function to load the dataset from a CSV file into a DataFrame:

```
[2]: import pandas as pd
```

```
[6]: df = pd.read_csv("C:/Users/aseks/Downloads/archive/Binance_BTCUSDT_2024_minute.csv")
     print(df.head())
```

```
           Unix                 Date   Symbol      Open      High      Low  \
0  1722383940000  2024-07-30 23:59:00  BTCUSDT  66196.00  66196.00  66188.0
1  1722383880000  2024-07-30 23:58:00  BTCUSDT  66224.00  66224.01  66196.0
2  1722383820000  2024-07-30 23:57:00  BTCUSDT  66224.01  66224.01  66224.0
3  1722383760000  2024-07-30 23:56:00  BTCUSDT  66236.01  66240.01  66224.0
4  1722383700000  2024-07-30 23:55:00  BTCUSDT  66243.99  66244.00  66236.0

      Close  Volume BTC    Volume USDT  tradecount
0  66188.00     1.15863   76691.239749         201
1  66196.00     1.95432  129392.697341         420
2  66224.01     2.04784  135616.169898         139
3  66224.01     1.92365  127407.013204         369
4  66236.01     1.76886  117163.812182         165
```

To obtain more detailed information, I will use the Pandas command df.describe():

```
df.describe()
```

	Unix	High
count	2.719710e+05	271921.000000
mean	1.712990e+12	59672.951717
std	5.278070e+09	9682.157932
min	1.704067e+12	38578.610000
25%	1.708495e+12	51741.180000
50%	1.712578e+12	63304.650000
75%	1.717956e+12	67121.160000
max	1.722384e+12	73777.000000

b. Before I can start analyzing the data, it's crucial to identify any missing or inconsistent data. Missing values can lead to inaccurate analysis, so detecting them early allows me to make informed decisions on how to handle them. I use isnull().sum() to count the missing values in each column. This helps me understand the extent of the missing data problem. If there are rows with missing values, I display them to see the specific cases we need to address.

```
missing_values = df.isnull().sum()
print("Missing values in each column:")
print(missing_values[missing_values > 0])
```

```
Missing values in each column:
Open           49
High           50
Low            50
Close          50
Volume BTC     50
Volume USDT    50
tradecount     50
dtype: int64
```

The results from Step 2 indicate that there are missing values in several columns of your dataset, specifically in the Open, High, Low, Close, Volume BTC, Volume USDT, and tradecount columns. Each of these columns has 49 or 50 missing values, which is a significant portion of the data. To transform the messy dataset into a tidy data format I will fill the missing values with the mean of the respective column. It will help to avoid losing data and if the missing values are assumed to be random. But before I will identify which entries in the Open column are causing the issue. By using the apply() function along with a lambda function, we can filter out the rows where the values in Open are not integers or floats. This helps see exactly which values are problematic. The

output gives me a list of rows where non-numeric data is present, which is causing the error when we try to calculate the mean.

```
non_numeric_open = df[~df['Open'].apply(lambda x: isinstance(x, (int, float)))]
print("Non-numeric values in 'Open' column:")
print(non_numeric_open)
```

```
Non-numeric values in 'Open' column:
               Unix                 Date   Symbol      Open      High  \
0      1722383940000  2024-07-30 23:59:00  BTCUSDT   66196.0  66196.00
1      1722383880000  2024-07-30 23:58:00  BTCUSDT   66224.0  66224.01
2      1722383820000  2024-07-30 23:57:00  BTCUSDT  66224.01  66224.01
3      1722383760000  2024-07-30 23:56:00  BTCUSDT  66236.01  66240.01
4      1722383700000  2024-07-30 23:55:00  BTCUSDT  66243.99  66244.00
...              ...                  ...      ...       ...       ...
65531  1718103780000  2024-06-11 11:03:00  BTCUSDT  66901.45  66918.49
65532  1718103720000  2024-06-11 11:02:00  BTCUSDT  66902.02  66923.27
65533  1718103660000  2024-06-11 11:01:00  BTCUSDT  66938.57  66945.94
65534  1718103600000  2024-06-11 11:00:00  BTCUSDT  66961.99  66961.99
65535  1718103540000  2024-06-11 10:59:00  BTCUSDT   66924.0  66962.00

            Low     Close Volume BTC      Volume USDT tradecount
0       66188.0   66188.0    1.15863    76691.2397491        201
1       66196.0   66196.0    1.95432   129392.6973409        420
2       66224.0  66224.01    2.04784   135616.1698976        139
3       66224.0  66224.01    1.92365   127407.0132041        369
4       66236.0  66236.01    1.76886   117163.8121815        165
...         ...       ...        ...              ...        ...
65531  66890.98  66895.99   24.65696  1649546.0352647       1078
65532  66901.45  66901.45    9.37358   627186.504254        793
65533  66899.99  66902.01  100.48282   6723302.124066       1493
65534   66932.0  66938.57   26.63191  1782846.7553396       1106
65535  66906.69  66961.99    14.3411   959857.2424306       1135
```

After identifying the non-numeric values, the next step is to convert the entire column to a numeric type. I will use pd.to_numeric() to attempt this conversion. The parameter errors='coerce' is particularly useful because it converts any values that cannot be interpreted as numbers into NaN (Not a Number). This allows us to handle the problematic entries uniformly, ensuring that the column is now entirely numeric and ready for further processing.

```
df['Open'] = pd.to_numeric(df['Open'], errors='coerce')
df['High'] = pd.to_numeric(df['High'], errors='coerce')
df['Low'] = pd.to_numeric(df['Low'], errors='coerce')
df['Close'] = pd.to_numeric(df['Close'], errors='coerce')
df['Volume BTC'] = pd.to_numeric(df['Volume BTC'], errors='coerce')
df['Volume USDT'] = pd.to_numeric(df['Volume USDT'], errors='coerce')
df['tradecount'] = pd.to_numeric(df['tradecount'], errors='coerce')
```

Now we can fill missing numerical values with the column's mean:

```
df['Open'] = df['Open'].fillna(df['Open'].mean())
df['High'] = df['High'].fillna(df['High'].mean())
df['Low'] = df['Low'].fillna(df['Low'].mean())
df['Close'] = df['Close'].fillna(df['Close'].mean())
df['Volume BTC'] = df['Volume BTC'].fillna(df['Volume BTC'].mean())
df['Volume USDT'] = df['Volume USDT'].fillna(df['Volume USDT'].mean())
df['tradecount'] = df['tradecount'].fillna(df['tradecount'].mean())
```

I resolved the TypeError caused by mixed data types in a column expected to be numeric. I identified non-numeric entries, converted the entire column to a

numeric format (coercing any errors to NaN), and then handled these missing values by filling them with the column mean.
Final check before continue:

```
print("Remaining non-numeric values after conversion:")
print(df[['Open', 'High', 'Low', 'Close', 'Volume BTC', 'Volume USDT', 'tradecount']].isnull().sum())
```

```
Remaining non-numeric values after conversion:
Open           0
High           0
Low            0
Close          0
Volume BTC     0
Volume USDT    0
tradecount     0
dtype: int64
```

c. After cleaning and tidying the dataset, the next logical step is to verify the structure and content of the dataset by displaying its initial rows.

```
6]: import pandas as pd
    df = pd.read_csv("C:\\Users\\aseks\\Downloads\\BTCUSDT_2024_minute - Binance_BTCUSDT_2024_minute.csv")
```

```
C:\Users\aseks\AppData\Local\Temp\ipykernel_8416\3804753410.py:2: DtypeWarning: Columns (3,5,6,7,8,9) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("C:\\Users\\aseks\\Downloads\\BTCUSDT_2024_minute - Binance_BTCUSDT_2024_minute.csv")
```

```
0]: df['Open'] = pd.to_numeric(df['Open'], errors='coerce')
    df['High'] = pd.to_numeric(df['High'], errors='coerce')
    df['Low'] = pd.to_numeric(df['Low'], errors='coerce')
    df['Close'] = pd.to_numeric(df['Close'], errors='coerce')
    df['Volume BTC'] = pd.to_numeric(df['Volume BTC'], errors='coerce')
    df['Volume USDT'] = pd.to_numeric(df['Volume USDT'], errors='coerce')
    df['tradecount'] = pd.to_numeric(df['tradecount'], errors='coerce')

    # Fill missing values with the mean of each column
    df['Open'] = df['Open'].fillna(df['Open'].mean())
    df['High'] = df['High'].fillna(df['High'].mean())
    df['Low'] = df['Low'].fillna(df['Low'].mean())
    df['Close'] = df['Close'].fillna(df['Close'].mean())
    df['Volume BTC'] = df['Volume BTC'].fillna(df['Volume BTC'].mean())
    df['Volume USDT'] = df['Volume USDT'].fillna(df['Volume USDT'].mean())
    df['tradecount'] = df['tradecount'].fillna(df['tradecount'].mean())
```

```
2]: df_tidy = df.pivot(index='Date', columns='Symbol', values=['Open', 'High', 'Low', 'Close'])
```

```
4]: print(df_tidy.head())
```

```
                        Open      High       Low     Close
Symbol                BTCUSDT   BTCUSDT   BTCUSDT   BTCUSDT
Date
2024-01-01 0:01:00  42298.62  42320.00  42298.61  42320.00
2024-01-01 0:02:00  42319.99  42331.54  42319.99  42325.50
2024-01-01 0:03:00  42325.50  42368.00  42325.49  42367.99
2024-01-01 0:04:00  42368.00  42397.23  42367.99  42397.23
2024-01-01 0:05:00  42397.22  42409.20  42385.26  42409.20
```

d. Filtering data based on specific criteria allows us to narrow down the dataset to only the relevant rows that meet certain conditions, making the analysis more focused and manageable.

In this example, I will use conditional statements to filter the dataset. I define my filtering criteria by specifying that I only want rows where the Open price is greater than 42,000 and the High price is greater than 42,500. This means I'm interested in instances where Bitcoin was traded at a high opening price with a significant peak during the period. Filtering Process: The filtering is done using the & operator, which ensures that both conditions must be true for a row to be included in the resulting dataset. This method is efficient and allows for complex filtering logic by combining multiple conditions.After applying the filter, I use head() to display the first few rows of the filtered dataset. This step

is essential to verify that the filtering has been applied correctly and that the resulting dataset meets my specified criteria. By employing these filtering techniques, I can focus my analysis on specific segments of the data, which is particularly useful when dealing with large datasets. This approach helps me draw more targeted insights and make data-driven decisions.

```
]: filtered_df = df_tidy[(df_tidy[('Open', 'BTCUSDT')] > 42000) & (df_tidy[('High', 'BTCUSDT')] > 42500)]
print(filtered_df.head())
```

	Open	High	Low	Close
Symbol	BTCUSDT	BTCUSDT	BTCUSDT	BTCUSDT
Date				
2024-01-01 0:15:00	42488.00	42516.46	42482.47	42510.00
2024-01-01 0:16:00	42510.01	42554.57	42510.00	42541.27
2024-01-01 0:17:00	42541.27	42541.27	42503.36	42503.36
2024-01-01 0:18:00	42503.36	42503.37	42467.28	42474.03
2024-01-01 10:00:00	42649.69	42653.57	42649.68	42653.56

e. As a data analyst, it's essential to have the ability to connect to databases, whether they are SQL or NoSQL, to retrieve and manipulate data efficiently. In this demonstration, I will show how to establish a connection to NoSQL databases using Python.

I need to ensure that I have the **pymongo** package installed to work with MongoDB:

```
from pymongo import MongoClient
client = MongoClient('mongodb+srv://aseksenbayeva88([redacted]luster0.gabnicq.mongodb.net/')
db = client['crypto_data']
collection = db['btc_trading']
```

I connect to a MongoDB server running on my local machine and access the crypto_data database and the btc_trading collection. This setup is equivalent to connecting to a table in an SQL database.

f.I import the Bitcoin trading dataset into the btc_trading collection within the crypto_data database. Each document represents a trading record with fields for date, open, high, low, and close.

```
btc_trading_data = [
    {"date": "2024-01-01 0:01:00", "open": 42298.62, "high": 42320.00, "low": 42298.61, "close": 42320.00},
    {"date": "2024-01-01 0:02:00", "open": 42319.99, "high": 42331.54, "low": 42319.99, "close": 42325.50},
    {"date": "2024-01-01 0:03:00", "open": 42325.50, "high": 42368.00, "low": 42325.49, "close": 42367.99},
    {"date": "2024-01-01 0:04:00", "open": 42368.00, "high": 42397.23, "low": 42367.99, "close": 42397.23},
    {"date": "2024-01-01 0:05:00", "open": 42397.22, "high": 42409.20, "low": 42385.26, "close": 42409.20}
]

collection.insert_many(btc_trading_data)
```

```
InsertManyResult([ObjectId('66c00ee7d501d88c5e0d57bb'), ObjectId('66c00ee7d501d88c5e0d57bc'), ObjectId('66c00ee7d501d88c5e0d57bd'), ObjectId('66c00ee7d501d88c5e0d57be'), ObjectId('66c00ee7d501d88c5e0d57bf')], acknowledged=True)
```

g. I use the find() method to retrieve all documents from the btc_trading collection and display them. This allows me to verify that the data has been correctly imported and is accessible.

```
results = collection.find()
for document in results:
    print(document)
```

```
{'_id': ObjectId('66c00e4fd501d88c5e0d57b6'), 'date': '2024-01-01 0:01:00', 'open': 42298.62, 'high': 42320.0, 'low': 42298.61, 'close': 42320.0}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b7'), 'date': '2024-01-01 0:02:00', 'open': 42319.99, 'high': 42331.54, 'low': 42319.99, 'close': 42325.5}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b8'), 'date': '2024-01-01 0:03:00', 'open': 42325.5, 'high': 42368.0, 'low': 42325.49, 'close': 42367.99}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b9'), 'date': '2024-01-01 0:04:00', 'open': 42368.0, 'high': 42397.23, 'low': 42367.99, 'close': 42397.23}
{'_id': ObjectId('66c00e4fd501d88c5e0d57ba'), 'date': '2024-01-01 0:05:00', 'open': 42397.22, 'high': 42409.2, 'low': 42385.26, 'close': 42409.2}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bb'), 'date': '2024-01-01 0:01:00', 'open': 42298.62, 'high': 42320.0, 'low': 42298.61, 'close': 42320.0}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bc'), 'date': '2024-01-01 0:02:00', 'open': 42319.99, 'high': 42331.54, 'low': 42319.99, 'close': 42325.5}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bd'), 'date': '2024-01-01 0:03:00', 'open': 42325.5, 'high': 42368.0, 'low': 42325.49, 'close': 42367.99}
{'_id': ObjectId('66c00ee7d501d88c5e0d57be'), 'date': '2024-01-01 0:04:00', 'open': 42368.0, 'high': 42397.23, 'low': 42367.99, 'close': 42397.23}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bf'), 'date': '2024-01-01 0:05:00', 'open': 42397.22, 'high': 42409.2, 'low': 42385.26, 'close': 42409.2}
```

h. I sort the documents in the btc_trading collection by the open price in descending order. The sort() method is used to specify the sorting field and the order (-1 for descending).

```
sorted_results = collection.find().sort("open", -1)
for document in sorted_results:
    print(document)
```

```
{'_id': ObjectId('66c00e4fd501d88c5e0d57ba'), 'date': '2024-01-01 0:05:00', 'open': 42397.22, 'high': 42409.2, 'low': 42385.26, 'close': 42409.2}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bf'), 'date': '2024-01-01 0:05:00', 'open': 42397.22, 'high': 42409.2, 'low': 42385.26, 'close': 42409.2}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b9'), 'date': '2024-01-01 0:04:00', 'open': 42368.0, 'high': 42397.23, 'low': 42367.99, 'close': 42397.23}
{'_id': ObjectId('66c00ee7d501d88c5e0d57be'), 'date': '2024-01-01 0:04:00', 'open': 42368.0, 'high': 42397.23, 'low': 42367.99, 'close': 42397.23}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b8'), 'date': '2024-01-01 0:03:00', 'open': 42325.5, 'high': 42368.0, 'low': 42325.49, 'close': 42367.99}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bd'), 'date': '2024-01-01 0:03:00', 'open': 42325.5, 'high': 42368.0, 'low': 42325.49, 'close': 42367.99}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b7'), 'date': '2024-01-01 0:02:00', 'open': 42319.99, 'high': 42331.54, 'low': 42319.99, 'close': 42325.5}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bc'), 'date': '2024-01-01 0:02:00', 'open': 42319.99, 'high': 42331.54, 'low': 42319.99, 'close': 42325.5}
{'_id': ObjectId('66c00e4fd501d88c5e0d57b6'), 'date': '2024-01-01 0:01:00', 'open': 42298.62, 'high': 42320.0, 'low': 42298.61, 'close': 42320.0}
{'_id': ObjectId('66c00ee7d501d88c5e0d57bb'), 'date': '2024-01-01 0:01:00', 'open': 42298.62, 'high': 42320.0, 'low': 42298.61, 'close': 42320.0}
```

i. I use the count_documents() method to count the total number of documents in the btc_trading collection. This is useful for understanding the size of the dataset.

```
count = collection.count_documents({})
print(f"Number of documents in the collection: {count}")
```

```
Number of documents in the collection: 10
```

j. I perform a grouping operation using the aggregate() method with a pipeline that groups documents by their close price and counts the number of occurrences for each unique price. This is similar to the GROUP BY operation in SQL.

```
pipeline = [
    {"$group": {"_id": "$close", "count": {"$sum": 1}}}
]

grouped_results = collection.aggregate(pipeline)
for result in grouped_results:
    print(result)
```

```
{'_id': 42397.23, 'count': 2}
{'_id': 42409.2, 'count': 2}
{'_id': 42320.0, 'count': 2}
{'_id': 42325.5, 'count': 2}
{'_id': 42367.99, 'count': 2}
```

k. I use the update_one() method to modify the open price for the document corresponding to the date 2024-01-01 0:01:00. The find_one() method is then used to verify that the update was successful.

```
collection.update_one({"date": "2024-01-01 0:01:00"}, {"$set": {"open": 42300.00}})
updated_document = collection.find_one({"date": "2024-01-01 0:01:00"})
print(updated_document)
```

```
{'_id': ObjectId('66c00e4fd501d88c5e0d57b6'), 'date': '2024-01-01 0:01:00', 'open': 42300.0, 'high': 42320.0, 'low': 42298.61, 'close': 42320.0}
```

Task B – Data Export, Migration and Backup

a) Export Data from a Specified Table or Collection in a Database to a Specified Format File (10 Marks)

I need to export data from a database to share it with others or to use it in other tools. In this task, I'm exporting data from a MongoDB collection to a CSV file format. This is a common practice when working with datasets, as CSV files are widely used and easily imported into various applications like Excel, Python, and R.

Here's how I did it:

1. **Connected to the MongoDB Database**: I used the `pymongo` library to establish a connection to the MongoDB server.
2. **Queried the Data**: I retrieved all documents from the `btc_trading` collection, which contains Bitcoin trading data.
3. **Exported Data to CSV**: I utilized Python's `csv` module to write the queried data into a CSV file, ensuring that each document was correctly formatted with headers and values.

This approach allows me to create a portable and shareable version of the dataset, which can be easily accessed by others or used for further analysis.

```
import csv
from pymongo import MongoClient
client = MongoClient('mongodb+srv://aseksenbayeva88:Alisha2810@cluster0.gabnicq.mongodb.net/')
db = client['crypto_data']
collection = db['btc_trading']
documents = collection.find({}, {'_id': False})
csv_file_path = 'btc_trading_data.csv'
header = ["date", "open", "high", "low", "close"]

with open(csv_file_path, 'w', newline='') as file:
    writer = csv.DictWriter(file, fieldnames=header)
    writer.writeheader()
    for document in documents:
        writer.writerow(document)

print(f"Data exported successfully to {csv_file_path}")
```

```
Data exported successfully to btc_trading_data.csv
```

b) Establish a Connection to the Cloud Storage Service and Upload the Locally Stored File to a Specified Cloud Storage Bucket Using a Cloud Service (15 Marks)

It's crucial for me to securely manage and store data, especially when working with cloud services like Azure. In this step, I established a connection to Azure Blob Storage and uploaded a locally stored file to a specified container within the storage account. This process is essential for ensuring that data is not only stored securely but is also easily accessible for analysis and sharing across the team.

First, I made sure that the necessary Python package, azure-storage-blob, was installed in my environment. This package allows me to interact programmatically with Azure Blob Storage.

I used the Anaconda environment to manage the dependencies and ensure that the necessary tools were available for my script:

```
: pip install azure-storage-blob

Collecting azure-storage-blob
  Using cached azure_storage_blob-12.22.0-py3-none-any.whl.metadata (26 kB)
Collecting azure-core>=1.28.0 (from azure-storage-blob)
  Using cached azure_core-1.30.2-py3-none-any.whl.metadata (37 kB)
Requirement already satisfied: cryptography>=2.1.4 in c:\users\aseks\anaconda3\lib\site-packages (from azure-storage-blob) (42.0.5)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\aseks\anaconda3\lib\site-packages (from azure-storage-blob) (4.11.0)
Collecting isodate>=0.6.1 (from azure-storage-blob)
  Using cached isodate-0.6.1-py2.py3-none-any.whl.metadata (9.6 kB)
Requirement already satisfied: requests>=2.21.0 in c:\users\aseks\anaconda3\lib\site-packages (from azure-core>=1.28.0->azure-storage-blob) (2.32.2)
Requirement already satisfied: six>=1.11.0 in c:\users\aseks\anaconda3\lib\site-packages (from azure-core>=1.28.0->azure-storage-blob) (1.16.0)
Requirement already satisfied: cffi>=1.12 in c:\users\aseks\anaconda3\lib\site-packages (from cryptography>=2.1.4->azure-storage-blob) (1.16.0)
Requirement already satisfied: pycparser in c:\users\aseks\anaconda3\lib\site-packages (from cffi>=1.12->cryptography>=2.1.4->azure-storage-blob) (2.21)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\aseks\anaconda3\lib\site-packages (from requests>=2.21.0->azure-core>=1.28.0->azure-storage-blob) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\aseks\anaconda3\lib\site-packages (from requests>=2.21.0->azure-core>=1.28.0->azure-storage-blob) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\aseks\anaconda3\lib\site-packages (from requests>=2.21.0->azure-core>=1.28.0->azure-storage-blob) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\aseks\anaconda3\lib\site-packages (from requests>=2.21.0->azure-core>=1.28.0->azure-storage-blob) (2024.7.4)
Using cached azure_storage_blob-12.22.0-py3-none-any.whl (404 kB)
Using cached azure_core-1.30.2-py3-none-any.whl (194 kB)
Using cached isodate-0.6.1-py2.py3-none-any.whl (41 kB)
Installing collected packages: isodate, azure-core, azure-storage-blob
Successfully installed azure-core-1.30.2 azure-storage-blob-12.22.0 isodate-0.6.1
Note: you may need to restart the kernel to use updated packages.
```

```
: pip show azure-storage-blob

Name: azure-storage-blob
Version: 12.22.0
Summary: Microsoft Azure Blob Storage Client Library for Python
Home-page: https://github.com/Azure/azure-sdk-for-python/tree/main/sdk/storage/azure-storage-blob
Author: Microsoft Corporation
Author-email: ascl@microsoft.com
License: MIT License
Location: C:\Users\aseks\anaconda3\Lib\site-packages
Requires: azure-core, cryptography, isodate, typing-extensions
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

I logged into the Azure Portal and navigated to the Storage Account I had set up earlier.

From the Access keys section, I retrieved the connection string, which is a crucial credential that allows my Python script to authenticate and connect to Azure Blob Storage. This connection string includes the account name and access key, which I kept secure and used in my script to establish a connection.

I wrote a Python script to upload my data file (btc_trading_data.csv) to Azure Blob Storage. Here's what the script did:

Establish Connection: Using the BlobServiceClient from the azure-storage-blob package, I established a connection to Azure Blob Storage by passing the connection string. I specified the name of the container where I wanted to store my file and created a BlobClient object that represents the specific file (or blob) I wanted to upload. I then used the upload_blob method to upload the CSV file from my local machine to the specified container in Azure Blob Storage. After executing the script, I verified the upload by checking the Azure Portal. I navigated to the container in my Storage Account and confirmed that the file btc_trading_data.csv was successfully uploaded and accessible. This step ensured that the data was securely stored in the cloud, making it accessible for further analysis or sharing with others.

```
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient

connection_string = "DefaultEndpointsProtocol=https;AccountName=[redacted]F/cTkkcx89WMXLnCk8+AStyAElJQ==;EndpointSuffix=core.windows.net"

blob_service_client = BlobServiceClient.from_connection_string(connection_string)

container_name = '612assessment2'
blob_name = 'btc_trading_data.csv'
local_file_name = 'btc_trading_data.csv'

blob_client = blob_service_client.get_blob_client(container="612assessment2", blob="btc_trading_data.csv")

with open(local_file_name, "rb") as data:
    blob_client.upload_blob(data)

print(f"File {local_file_name} uploaded to Azure Blob Storage container {container_name} successfully.")
```

```
File btc_trading_data.csv uploaded to Azure Blob Storage container 612assessment2 successfully.
```

c) Schedule automated backups of a specified directory to a cloud storage service.

I need to ensure that my data is securely backed up, especially when working on important projects. To do this, I wrote a Python script that automatically uploads files from a specified directory to Azure Blob Storage. I start by configuring logging to keep track of successful uploads and any errors during the process. I then set up the connection to my Azure Storage Account using the connection string, which I retrieved from the Azure Portal. I specify the directory I want to back up and the container in Azure where the files will be stored. The script uses the os.walk() function to navigate through the directory and uploads each file to Azure Blob Storage using the upload_blob() method.

By writing this script, I can automate the backup process, making sure that my important files are always backed up without manual intervention.

```python
import os
from azure.storage.blob import BlobServiceClient
import logging
logging.basicConfig(filename='backup_log.log', level=logging.INFO, format='%(asctime)s %(message)s')

connection_string = "DefaultEndpointsProtocol=https;AccountName=assessments22024;AccountKey=ti9TZya6auqc0bY2ytx054i5lYcPlAG/aCjcrQU+DCmJvqUiroiFJhbGoZf/cTkkcx89WNXLnCk8+AStyAElJQ==;EndpointSuffix=core.windows.nett"
blob_service_client = BlobServiceClient.from_connection_string(connection_string)

directory_to_backup = r'C:\Users\aseks\OneDrive\Рабочий стол'
container_name = "612assessment2"

def upload_directory_to_azure(directory_path, container_name):
    for root, dirs, files in os.walk(directory_path):
        for file_name in files:
            file_path = os.path.join(root, file_name)
            blob_name = os.path.relpath(file_path, directory_path).replace("\\", "/")
            try:
                blob_client = blob_service_client.get_blob_client(container="612assessment2", blob="btc_trading_data.csv")
                with open(file_path, "rb") as data:
                    blob_client.upload_blob(data, overwrite=True)
                logging.info(f"Uploaded {Backup} to Azure Blob Storage as {btc_trading_data.csv}")
            except Exception as e:
                logging.error(f"Failed to upload {file_name}: {e}")

upload_directory_to_azure(directory_to_backup, container_name)
```

Results: 2024-08-17 16:39:23,985 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '146540'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'ad827298-5c52-11ef-923f-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:39:54,489 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '146540'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'bfb0fb78-5c52-11ef-b856-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:40:30,766 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '146540'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'd5507791-5c52-11ef-a85a-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:41:21,894 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '146540'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'f3c9f4ad-5c52-11ef-964c-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:41:31,961 Failed to upload acer nitro 3.png: <urllib3.connection.HTTPSConnection object at 0x0000027890F52810>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:41:32,015 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '258119'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'f9d22199-5c52-11ef-b946-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:41:48,688 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '258119'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '03c24ac7-5c53-11ef-bab0-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:42:21,599 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '258119'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '176024d9-5c53-11ef-a581-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:43:14,646 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '258119'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '36fe7105-5c53-11ef-a439-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:43:24,721 Failed to upload Acer Nitro.png: <urllib3.connection.HTTPSConnection object at 0x0000027890F8A570>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:43:27,144 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '9887620'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '3e718aa4-5c53-11ef-af3b-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:43:42,287 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '9887620'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '47780397-5c53-11ef-a984-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:44:14,946 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '9887620'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '5aef5fef-5c53-11ef-9bb3-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:45:08,128 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '9887620'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '7aa25e01-5c53-11ef-845b-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:45:18,191 Failed to upload Addison_Wesley_Object_Oriented_Analysis_and_Design_with_Applications_3rd_Edition_May_2007.pdf: <urllib3.connection.HTTPSConnection object at 0x0000027890F51B80>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:45:18,193 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2407'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '80a2488e-5c53-11ef-8f74-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:45:34,484 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2407'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '8a581294-5c53-11ef-a318-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:46:08,160 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2407'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '9e6aa500-5c53-11ef-8bc9-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:47:01,938 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2407'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'be785dd7-5c53-11ef-973c-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:47:12,010 Failed to upload Aigul - Chrome.lnk: <urllib3.connection.HTTPSConnection object at 0x0000027890F50080>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:47:12,038 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2086163'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'c47d72a2-5c53-11ef-9892-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:47:28,286 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2086163'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'ce2cd1fb-5c53-11ef-a932-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:48:04,873 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2086163'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': 'e3fb7dcb-5c53-11ef-b92a-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:48:56,435 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '2086163'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '02b76d2d-5c54-11ef-8cfe-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:49:06,499 Failed to upload Alisha print1.pdf: <urllib3.connection.HTTPSConnection object at 0x0000027890F8A1E0>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:49:06,524 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '1765378'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '08bab8f9-5c54-11ef-8f41-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:49:26,508 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '1765378'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '14a4030e-5c54-11ef-b054-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:49:59,046 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '1765378'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '28090166-5c54-11ef-a7a6-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:50:53,894 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '1765378'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '48ba0ea4-5c54-11ef-9537-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:51:03,963 Failed to upload Alisha Print2.pdf: <urllib3.connection.HTTPSConnection object at 0x0000027890F67110>: Failed to resolve 'assessments22024.blob.core.windows.nett' ([Errno 11002] getaddrinfo failed)

2024-08-17 16:51:03,968 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '289'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '4ebb62cd-5c54-11ef-9744-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:51:21,030 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '289'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '58e6cf5e-5c54-11ef-9509-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

2024-08-17 16:51:57,455 Request URL: 'https://assessments22024.blob.core.windows.nett/612assessment2/btc_trading_data.csv'

Request method: 'PUT'

Request headers:

'Content-Length': '289'

'x-ms-blob-type': 'REDACTED'

'x-ms-version': 'REDACTED'

'Content-Type': 'application/octet-stream'

'Accept': 'application/xml'

'User-Agent': 'azsdk-python-storage-blob/12.22.0 Python/3.12.4 (Windows-11-10.0.22631-SP0)'

'x-ms-date': 'REDACTED'

'x-ms-client-request-id': '6e9cbd21-5c54-11ef-9735-6cf6da11534f'

'Authorization': 'REDACTED'

A body is sent with the request

d. Include error handling to handle backup failures gracefully, such as connectivity issues or file upload errors.

To handle backup failures gracefully, such as connectivity issues or file upload errors, you can incorporate error handling into your Python script. This involves using try, except, and finally blocks to catch exceptions and manage them appropriately. Additionally, logging the errors is important for diagnosing issues later.

```python
import os
from azure.storage.blob import BlobServiceClient, BlobClient, ContainerClient
import logging

logging.basicConfig(filename='backup_log.log', level=logging.INFO, format='%(asctime)s %(message)s')

connection_string = "DefaultEndpointsProtocol=https;AccountName=assessments22024;AccountKey=ti9TZya6auqc0bY2ytx054i5lYcPlAG/aCjcrQU+DCmJvqUiroiFJhbGoZf/cTkkcx89WMXLnCk8+AStyAElJQ==;EndpointSuffix=core.windows.nett"
blob_service_client = BlobServiceClient.from_connection_string(connection_string)

directory_to_backup = r'C:\Users\aseks\OneDrive\Рабочий стол'
container_name = "612assessment2"

def upload_directory_to_azure(directory_path, container_name):
    try:
        for root, dirs, files in os.walk(directory_path):
            for file_name in files:
                file_path = os.path.join(root, file_name)
                blob_name = os.path.relpath(file_path, directory_path).replace("\\", "/")
                try:
                    blob_client = blob_service_client.get_blob_client(container=container_name, blob=blob_name)
                    with open(file_path, "rb") as data:
                        blob_client.upload_blob(data, overwrite=True)
                    logging.info(f"Uploaded {file_name} to Azure Blob Storage as {blob_name}")
                except Exception as e:
                    logging.error(f"Failed to upload {file_name}: {e}")
                    print(f"Error uploading {file_name}: {e}")

    except Exception as e:
        logging.critical(f"Critical error during backup: {e}")
        print(f"Critical error: {e}")

    finally:
        logging.info("Backup process completed.")


upload_directory_to_azure(directory_to_backup, container_name)
```