EEET2574 | Big Data for Engineering

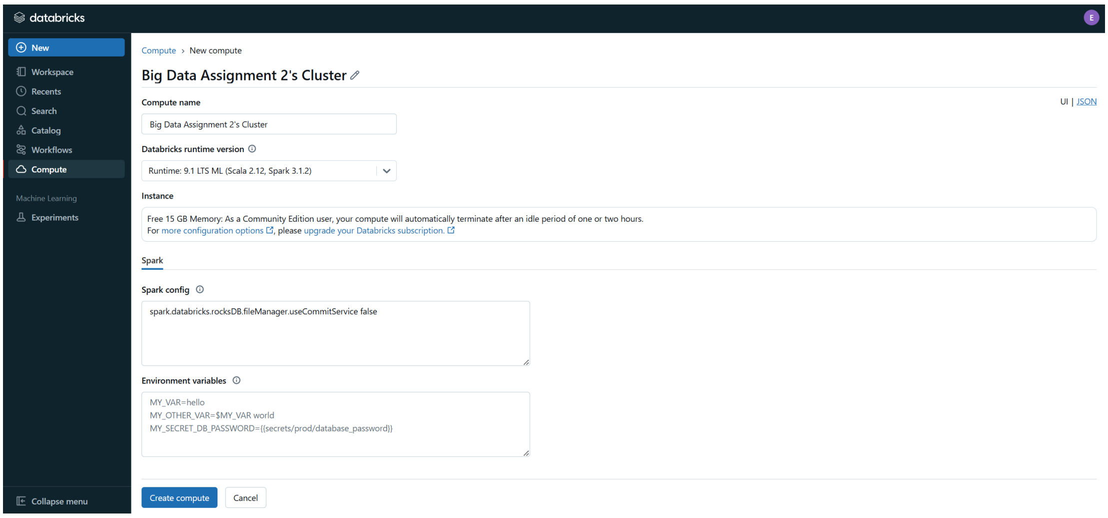
Assignment 2: MongoDB and Spark

- **Author**: Tran Manh Cuong - s3974735
- **Instructor**: Dr. Arthur Tang

Task 0: Instruction on how to run the notebook

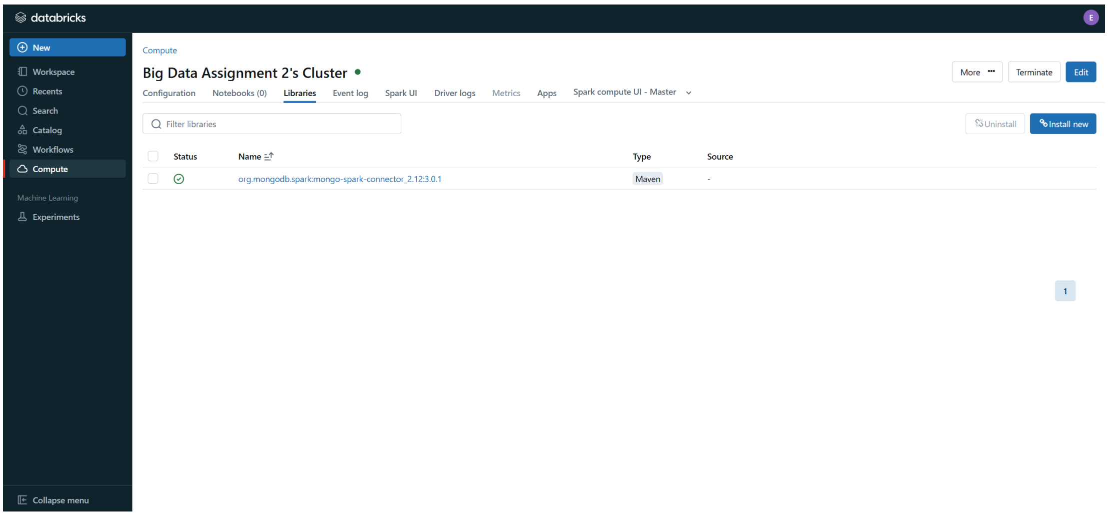
Step 1: Create a compute cluster:

- On the sidebar, select **Compute**.
- Select **Create compute**.
- Enter compute name: **Big Data Assignment 2's Cluster**.
- Choose the databricks runtime version: **9.1 LTS ML (Scala 2.12, Spark 3.1.2)**.
- Select **Create compute**.



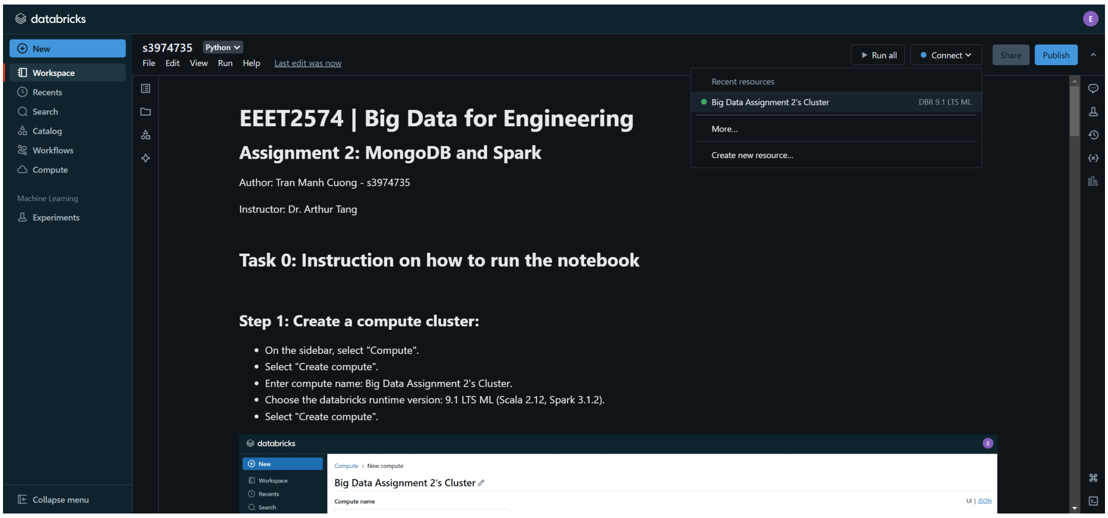
Step 2: Install mongodb spark connector library

- On the navigation bar, select **Libraries**.
- Select **Install new**.
- For Library Source, select **Maven**.
- For Coordinates, select **Select Packages**.
- Select **Spark Packages**.
- Search and select **mongo-spark** with version **3.0.1**.
- Select **Install**.



Step 3: Attach cluster to notebook

- Import the notebook to databricks using .dbc or .ipynb file.
- Select the notebook.
- For the Connect, select the created cluster.



Task 1: MongoDB

Load Datasets to MongoDB Collections

The datasets had already been loaded into MongoDB, so you don't need to run this code.

In [0]:
```
import pymongo
import pandas as pd
import os

# You can replace the data directory with your own path
ELECTRICITY_DIR = "./data/Electricity"
GAS_DIR = "./data/Gas"
```

```python
# You can replace the URI with your own MongoDB URI
MONGO_URI = "mongodb+srv://cuongtran:cuongtran123@cluster0.5zjcy.mongodb.net/?re

# MongoDB connection
client = pymongo.MongoClient(MONGO_URI)
db = client["main"]


# Data mapping for collections
collections_mapping = {
    "electricity": {
        "2018": ["coteq_electricity_2018.csv", "stedin_electricity_2018.csv", "w
        "2019": ["coteq_electricity_2019.csv", "stedin_electricity_2019.csv", "w
        "2020": ["coteq_electricity_2020.csv", "stedin_electricity_2020.csv", "w
    },
    "gas": {
        "2018": ["coteq_gas_2018.csv", "stedin_gas_2018.csv", "westland-infra_ga
        "2019": ["coteq_gas_2019.csv", "stedin_gas_2019.csv", "westland-infra_ga
        "2020": ["coteq_gas_2020.csv", "stedin_gas_2020.csv", "westland-infra_ga
    }
}

# Check if a collection exists in MongoDB
def is_collection_exist(collection_name):
    return collection_name in db.list_collection_names()

# Load data into MongoDB
def load_data_to_mongodb(database, directory, year, file_list):
    collection_name = f"{database}_{year}"
    if is_collection_exist(collection_name):
        print(f"Collection {collection_name} already exists. Skipping data load
        return

    collection = db[collection_name]
    for file_name in file_list:
        file_path = os.path.join(directory, file_name)
        df = pd.read_csv(file_path)
        records = df.to_dict(orient="records")
        collection.insert_many(records)
        print(f"Data from {file_name} has been loaded to collection {collection_

# Main function
def main():
    for database, years in collections_mapping.items():
        for year, file_list in years.items():
            if database == "electricity":
                load_data_to_mongodb(database, ELECTRICITY_DIR, year, file_list)
            elif database == "gas":
                load_data_to_mongodb(database, GAS_DIR, year, file_list)

if __name__ == "__main__":
    main()
```

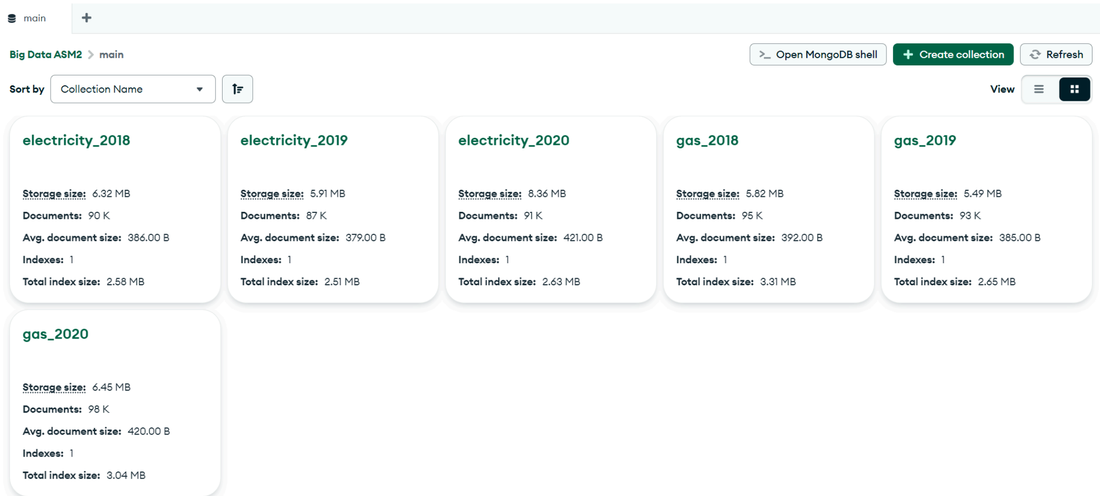
Question 1A: How many collections do you have? Why?

The MongoDB database contains a total of 6 collections, categorized based on type of data (Electricity, Gas) and year (2018, 2019, 2020):

- Electricity collections: electricity_2018, electricity_2019, electricity_2020
- Gas collections: gas_2018, gas_2019, gas_2020

This design choice allows targeted querying and manipulation during data ingestion and cleaning in Task 2. By having distinct collections for each category and year, we can streamline the transformation pipeline, as each collection can be processed individually or combined as needed. Furthermore, this structure supports scalable data management. If new data for additional years or companies becomes available, it can easily be incorporated into separate collections without affecting the existing ones.

Another reason for this design is to simplify model training and tracking in Task 3. Since the task requires using 2018 and 2019 data for training and 2020 for testing, the separate collections make it straightforward to load and partition data for machine learning pipelines. Additionally, visualizations in Task 4 benefit from this schema as it allows flexibility to aggregate data across years or focus on specific time frames and data types.



Task 2: Data ingestion and data cleaning/transformation

Data Ingestion

Load electricity and gas consumption data from MongoDB into PySpark DataFrames

In [0]:
```
from pyspark.sql import SparkSession

# MongoDB URI
MONGO_URI = "mongodb+srv://cuongtran:cuongtran123@cluster0.5zjcy.mongodb.net/?re

# Initialize Spark session
```

```
spark = SparkSession.builder \
    .appName("Big_Data_ASM2") \
    .config("spark.mongodb.input.uri", MONGO_URI) \
    .config("spark.mongodb.output.uri", MONGO_URI) \
    .getOrCreate()

# Electricity collections
electricity_2018_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "electricity_2018") \
    .load()

electricity_2019_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "electricity_2019") \
    .load()

electricity_2020_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "electricity_2020") \
    .load()

# Gas collections
gas_2018_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "gas_2018") \
    .load()

gas_2019_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "gas_2019") \
    .load()

gas_2020_df = spark.read.format("com.mongodb.spark.sql.DefaultSource") \
    .option("uri", MONGO_URI) \
    .option("database", "main") \
    .option("collection", "gas_2020") \
    .load()
```

Concatenate the 2018 and 2019 dataframes to create train datasets and assigns the 2020 dataframes as test datasets for Task 3.

In [0]:
```
# Train datasets
electricity_train_df = electricity_2018_df.union(electricity_2019_df)
gas_train_df = gas_2018_df.union(gas_2019_df)

# Test datasets
electricity_test_df = electricity_2020_df
gas_test_df = gas_2020_df
```

Data Exploration

Convert the Spark DataFrames for electricity and gas train datasets into Pandas DataFrames using the `toPandas()` method. The purpose of this conversion is to facilitate data exploration using Pandas' versatile and intuitive tools, which are better suited for detailed analysis, such as summarizing data, checking for missing values, and visualizing distributions

In [0]:
```
electricity_pandas_df = electricity_train_df.toPandas()
gas_pandas_df = gas_train_df.toPandas()
```

Display the number of rows and columns to understand the overall size of the dataset.

In [0]:
```
electricity_pandas_df.shape
```

Out[69]: (176807, 16)

In [0]:
```
gas_pandas_df.shape
```

Out[70]: (187606, 16)

Preview the first five rows of the electricity and gas train datasets to understand the structure, column names, and sample data.

In [0]:
```
electricity_pandas_df.head()
```

	%Defintieve aansl (NRM)	_id	annual_consume	annual_consume_lowtarif_p
0	NaN	{'oid': '676d0e00999375116ce42b08'}	4122	89
1	NaN	{'oid': '676d0e00999375116ce42b09'}	1800	94
2	NaN	{'oid': '676d0e00999375116ce42b0a'}	1315	100
3	NaN	{'oid': '676d0e00999375116ce42b0b'}	6379	92
4	NaN	{'oid': '676d0e00999375116ce42b0c'}	4404	92

In [0]:
```
gas_pandas_df.head()
```

	%Defintieve aansl (NRM)	_id	annual_consume	annual_consume_lowtarif_p
0	NaN	{'oid': '676d0e57999375116ce842fd'}	3457	
1	NaN	{'oid': '676d0e57999375116ce842fe'}	4036	
2	NaN	{'oid': '676d0e57999375116ce842ff'}	3695	
3	NaN	{'oid': '676d0e57999375116ce84300'}	3307	
4	NaN	{'oid': '676d0e57999375116ce84301'}	2306	

Display the data types of each column to understand which features are numerical, categorical, or require further transformation. The result indicates that most numerical columns are suitable for scaling, while categorical columns will need encoding during the data transformation process.

In [0]:
```
electricity_pandas_df.dtypes
```

```
Out[12]: %Defintieve aansl (NRM)        float64
_id                               object
annual_consume                     int32
annual_consume_lowtarif_perc     float64
city                              object
delivery_perc                    float64
net_manager                       object
num_connections                    int32
perc_of_active_connections       float64
purchase_area                     object
smartmeter_perc                  float64
street                            object
type_conn_perc                   float64
type_of_connection                object
zipcode_from                      object
zipcode_to                        object
dtype: object
```

In [0]:
```
gas_pandas_df.dtypes
```

```
Out[13]: %Defintieve aansl (NRM)         float64
_id                              object
annual_consume                    int32
annual_consume_lowtarif_perc    float64
city                             object
delivery_perc                   float64
net_manager                      object
num_connections                   int32
perc_of_active_connections      float64
purchase_area                    object
smartmeter_perc                 float64
street                           object
type_conn_perc                  float64
type_of_connection               object
zipcode_from                     object
zipcode_to                       object
dtype: object
```

Check for missing values in each column to identify potential data quality issues. The results reveal that the **%Defintieve aansl (NRM)** column has a significant number of missing values (171,965 in electricity and 182,994 in gas datasets), indicating it may require removal or imputation. All other columns have no missing values, suggesting they are ready for further analysis and transformation.

In [0]:
```
electricity_pandas_df.isnull().sum()
```

```
Out[47]: %Defintieve aansl (NRM)         171965
_id                                 0
annual_consume                      0
annual_consume_lowtarif_perc        0
city                                0
delivery_perc                       0
net_manager                         0
num_connections                     0
perc_of_active_connections          0
purchase_area                       0
smartmeter_perc                     0
street                              0
type_conn_perc                      0
type_of_connection                  0
zipcode_from                        0
zipcode_to                          0
dtype: int64
```

In [0]:
```
gas_pandas_df.isnull().sum()
```

```
Out[15]: %Defintieve aansl (NRM)        182994
_id                              0
annual_consume                   0
annual_consume_lowtarif_perc     0
city                             0
delivery_perc                    0
net_manager                      0
num_connections                  0
perc_of_active_connections       0
purchase_area                    0
smartmeter_perc                  0
street                           0
type_conn_perc                   0
type_of_connection               0
zipcode_from                     0
zipcode_to                       0
dtype: int64
```

Identify and categorize columns into numerical and categorical features for targeted exploration and transformation.

In [0]:
```
numerical_columns = [
    "annual_consume",
    "annual_consume_lowtarif_perc",
    "delivery_perc",
    "num_connections",
    "perc_of_active_connections",
    "smartmeter_perc",
    "type_conn_perc"
]

categorical_columns = [
    "city",
    "net_manager",
    "purchase_area",
    "street",
    "type_of_connection",
    "zipcode_from",
    "zipcode_to"
]
```

Generate summary statistics for numerical columns to understand their distribution, central tendency, and spread. The electricity dataset shows notable outliers in **annual_consume** (max 110,857 vs. mean 3,913) and **num_connections** (max 1,146 vs. mean 25), indicating skewness that may require log transformation. Similarly, the gas dataset reveals significant skewness in **annual_consume** (max 27,917 vs. mean 1,559) and **num_connections** (max 1,065 vs. mean 22). These findings highlight the need for outlier handling, scaling, and possible transformation during the data cleaning and transformation phases. Additionally, the constant value in **annual_consume_lowtarif_perc** for gas indicates it may not provide significant variability for modeling.

In [0]:
```
electricity_pandas_df[numerical_columns].describe()
```

	annual_consume	annual_consume_lowtarif_perc	delivery_perc	num_connections
count	176807.000000	176807.000000	176807.000000	176807.000000
mean	3913.422755	87.088556	94.702266	25.088967
std	3281.350003	18.725204	10.108968	18.562825
min	0.000000	0.000000	0.000000	10.000000
25%	2387.000000	81.480000	92.860000	16.000000
50%	3114.000000	95.240000	100.000000	21.000000
75%	4055.000000	100.000000	100.000000	28.000000
max	110857.000000	100.000000	100.000000	1146.000000

In [0]:
```
gas_pandas_df[numerical_columns].describe()
```

	annual_consume	annual_consume_lowtarif_perc	delivery_perc	num_connections
count	187606.000000	187606.0	187606.000000	187606.000000
mean	1559.704295	0.0	99.999916	22.961163
std	959.909904	0.0	0.022761	14.325173
min	0.000000	0.0	90.910000	10.000000
25%	1065.000000	0.0	100.000000	15.000000
50%	1374.000000	0.0	100.000000	20.000000
75%	1825.000000	0.0	100.000000	26.000000
max	27917.000000	0.0	100.000000	1065.000000

Analyze the unique value counts for categorical columns to understand their diversity and identify potential encoding challenges. Both datasets have a high cardinality in **street**, **zipcode_from**, and **zipcode_to**, which may require dimensionality reduction or exclusion if they do not provide meaningful insights. Columns like **net_manager**, **purchase_area**, and **type_of_connection** have relatively fewer unique values, making them suitable for encoding during the transformation process.

In [0]:
```
electricity_pandas_df[categorical_columns].nunique()
```

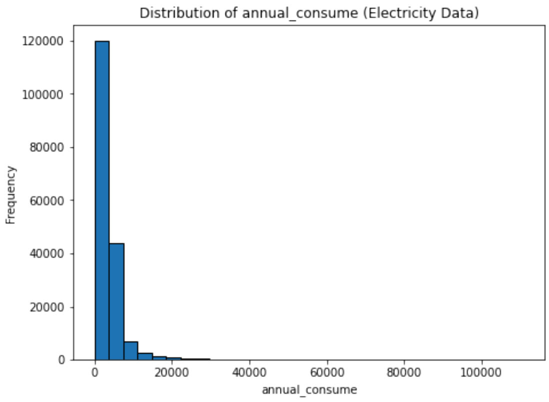
```
Out[19]: city                  287
net_manager            10
purchase_area           9
street              29426
type_of_connection     10
zipcode_from        89513
zipcode_to          88945
dtype: int64
```

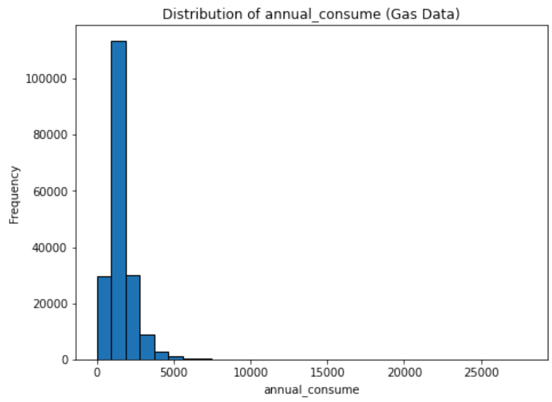
In [0]:
```
gas_pandas_df[categorical_columns].nunique()
```

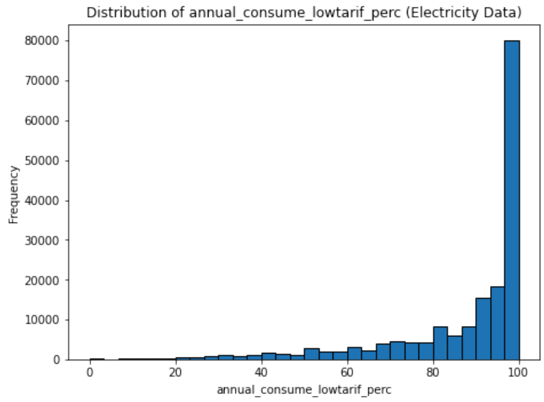
```
Out[20]: city                    491
net_manager             14
purchase_area           39
street               31598
type_of_connection       6
zipcode_from         95617
zipcode_to           95087
dtype: int64
```

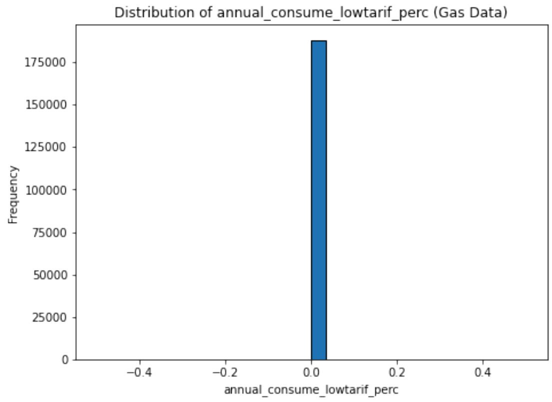
Analyze the distribution of numerical columns in both datasets using histograms to identify data spread, skewness, and potential outliers. The visualizations reveal that columns like **annual_consume**, **num_connections**, and **type_conn_perc** exhibit highly skewed distributions, indicating the need for transformation techniques like log scaling to reduce skewness. Additionally, **smartmeter_perc** and **delivery_perc** show a more uniform distribution, which can be directly utilized in model training without transformations.

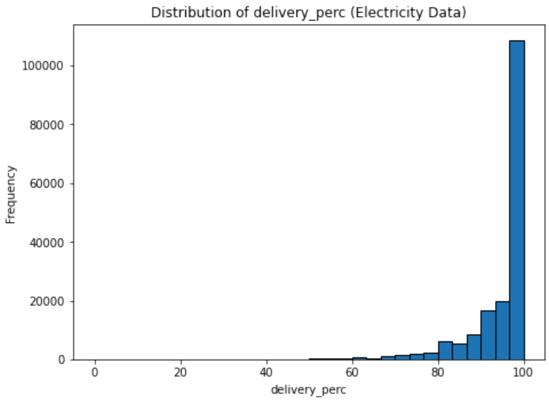
In [0]:
```python
import matplotlib.pyplot as plt

fig, axes = plt.subplots(len(numerical_columns), 2, figsize=(14, 5 * len(numeric
fig.tight_layout(w_pad=10.0, h_pad=5.0)

for i, col in enumerate(numerical_columns):
    # Electricity histogram
    axes[i, 0].hist(electricity_pandas_df[col], bins=30, edgecolor='black')
    axes[i, 0].set_title(f"Distribution of {col} (Electricity Data)")
    axes[i, 0].set_xlabel(col)
    axes[i, 0].set_ylabel("Frequency")

    # Gas histogram
    axes[i, 1].hist(gas_pandas_df[col], bins=30, edgecolor='black')
    axes[i, 1].set_title(f"Distribution of {col} (Gas Data)")
    axes[i, 1].set_xlabel(col)
    axes[i, 1].set_ylabel("Frequency")

plt.show()
```

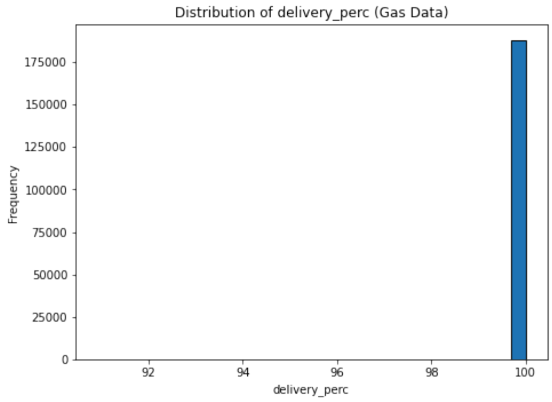

Distribution of annual_consume (Electricity Data)
Frequency
annual_consume

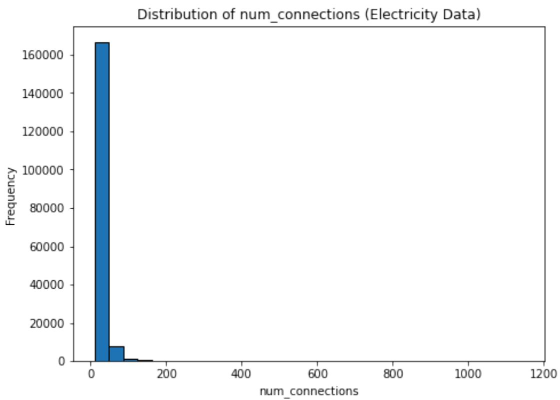

Distribution of annual_consume (Gas Data)
Frequency
annual_consume

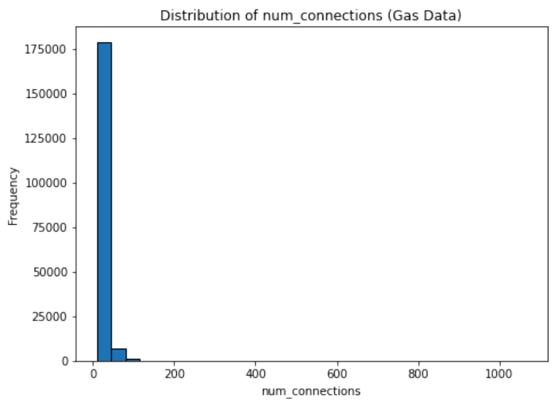

Distribution of annual_consume_lowtarif_perc (Electricity Data)
Frequency
annual_consume_lowtarif_perc

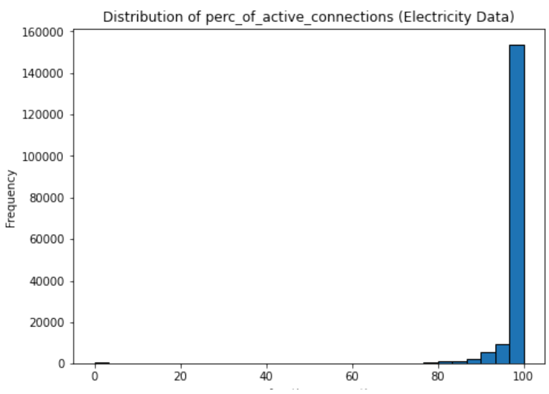

Distribution of annual_consume_lowtarif_perc (Gas Data)
Frequency
annual_consume_lowtarif_perc

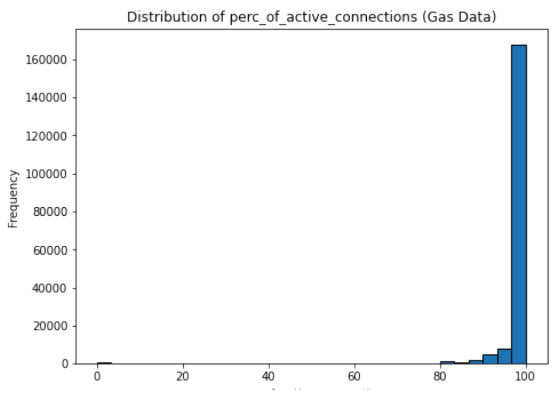

Distribution of delivery_perc (Electricity Data)
Frequency
delivery_perc

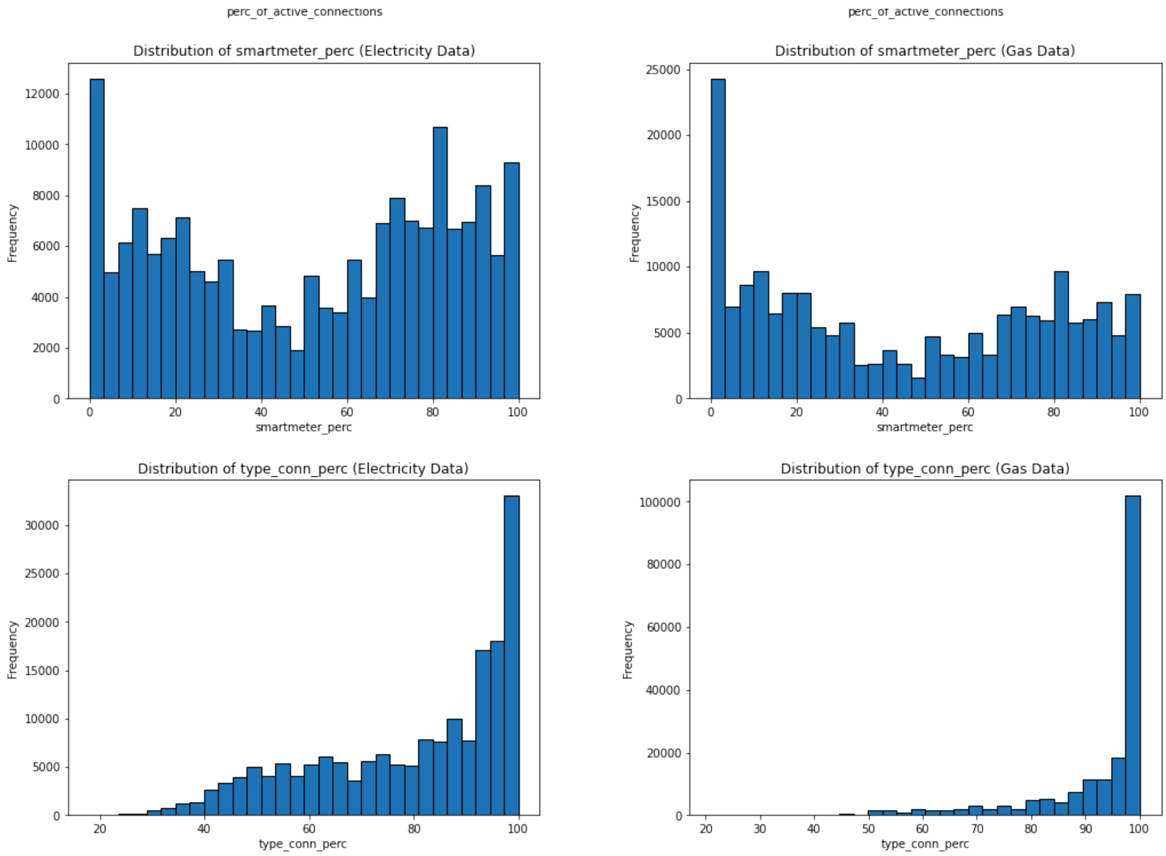

Distribution of delivery_perc (Gas Data)
Frequency
delivery_perc


Distribution of num_connections (Electricity Data)
Frequency
num_connections


Distribution of num_connections (Gas Data)
Frequency
num_connections


Distribution of perc_of_active_connections (Electricity Data)
Frequency


Distribution of perc_of_active_connections (Gas Data)
Frequency



Analyze the frequency distributions of key categorical columns to understand the diversity and dominance of specific categories. In the **city** column, both electricity and gas datasets have a significant number of unique values, indicating a high geographical coverage. The **net_manager** and **purchase_area** columns show skewed distributions, with a few dominant categories, such as **8716892000005** for **net_manager** and **Stedin** for **purchase_area**. Similarly, **type_of_connection** is dominated by **1x35** in electricity and **G4** in gas, indicating standardized types of connections.

In [0]:
```
electricity_pandas_df["city"].value_counts()
```

```
Out[22]: 'S-GRAVENHAGE      23069
ROTTERDAM          20644
UTRECHT            11949
AMERSFOORT          5182
ZOETERMEER          5038
                    ...
NIEUWGEIN              1
ZEVENHOVEN             1
'Sâ��GRAVENHAGE         1
MAURIK                 1
AMSTELVEEN             1
Name: city, Length: 287, dtype: int64
```

In [0]:
```
gas_pandas_df["city"].value_counts()
```

```
Out[23]: 'S-GRAVENHAGE          20702
ROTTERDAM             17994
UTRECHT                9712
AMERSFOORT             4792
ZOETERMEER             4107
                       ...
ZWAAGWESTEINDE            1
DAMWOUDE                  1
NEDERHORST DEN BERG       1
'S GRAVENDEEL             1
NIGTEVECHT                1
Name: city, Length: 491, dtype: int64
```

In [0]:
```
electricity_pandas_df["net_manager"].value_counts()
```

```
Out[24]: 8716892000005            89266
8716874000009            51435
8716886000004            13073
8716921000006             8628
westland-infra            4842
8716925000002             3150
Cogas Infra & Beheer BV   2563
Coteq Netbeheer BV        2503
8716946000005             1331
8716924000003               16
Name: net_manager, dtype: int64
```

In [0]:
```
gas_pandas_df["net_manager"].value_counts()
```

```
Out[25]: 8716892000005            81232
8716874000009            39433
8716886000004            11291
8716921000006             8015
Cogas Infra & Beheer BV   6868
Coteq Netbeheer BV        6774
8716892700004             6098
8716892750009             6032
8716892740000             5825
8716892710003             5134
westland-infra            4612
8716892720002             3285
8716925000002             3002
8716924000003                5
Name: net_manager, dtype: int64
```

In [0]:
```
electricity_pandas_df["purchase_area"].value_counts()
```

```
Out[26]: Stedin                                  88881
Stedin Utrecht                          51669
Stedin Delfland                         13330
Stedin Midden-Holland                    8524
Netbeheerder Centraal Overijssel B.V.    5066
871687800090000015                       4842
Stedin Schiedam                          3149
Stedin Elektriciteit Zuid-Kennemerland   1330
Stedin Weert                               16
Name: purchase_area, dtype: int64
```

In [0]:
```
gas_pandas_df["purchase_area"].value_counts()
```

```
Out[27]: NG Den Haag                          26217
Pseudo Gos Houten ENBU                23313
Pseudo-GOS Dordrecht                  17420
GAS Gastransport Services (GASUNIE)   13642
Pseudo-GOS Rotterdam                  13100
Pseudo Gos Hoogland ENBU               9550
NG Leerdam                             8520
Pseudo-GOS Zoetermeer                  6606
Pseudo Gos Veenendaal ENBU             6602
NG Gouda                               6313
Pseudo-GOS Zeist                       6032
Pseudo-GOS Amstelland                  5580
NG Noord-Oost Friesland                5551
NG Hoekse waard                        5267
Pseudo-GOS Midden Kennemerland         5134
871718518003006694                     4612
Pseudo-GOS Delft                       3805
NG Heemstede                           3032
Schiedam Kethel                        3002
Pseudo-GOS Vlaardingen                 2624
NG Brielle                             2313
NG Krimpen                             1664
NG Waddinxveen                         1662
Achterweg                              1610
Maassluis                              1077
Pseudo-GOS Bleiswijk                    891
Oranjelaan                              460
Hoek van Holland                        386
Ouderkerk ad Amstel                     366
Ameland                                 272
Halfweg                                 252
Ruigendijk                              228
Moerseweg                               217
Duivendrecht                            153
Graswalseweg                            115
Wildersekade                             12
Pseudo-GOS Weert                          4
NaN                                       1
Weert Trancheeweg                         1
Name: purchase_area, dtype: int64
```

In [0]:
```python
electricity_pandas_df["type_of_connection"].value_counts()
```

```
Out[28]: 1x35    126663
3x25     29887
1x25     14674
1x50      4416
3x35       769
3x63       189
3x80        98
3x50        85
OBK         19
1x6          7
Name: type_of_connection, dtype: int64
```

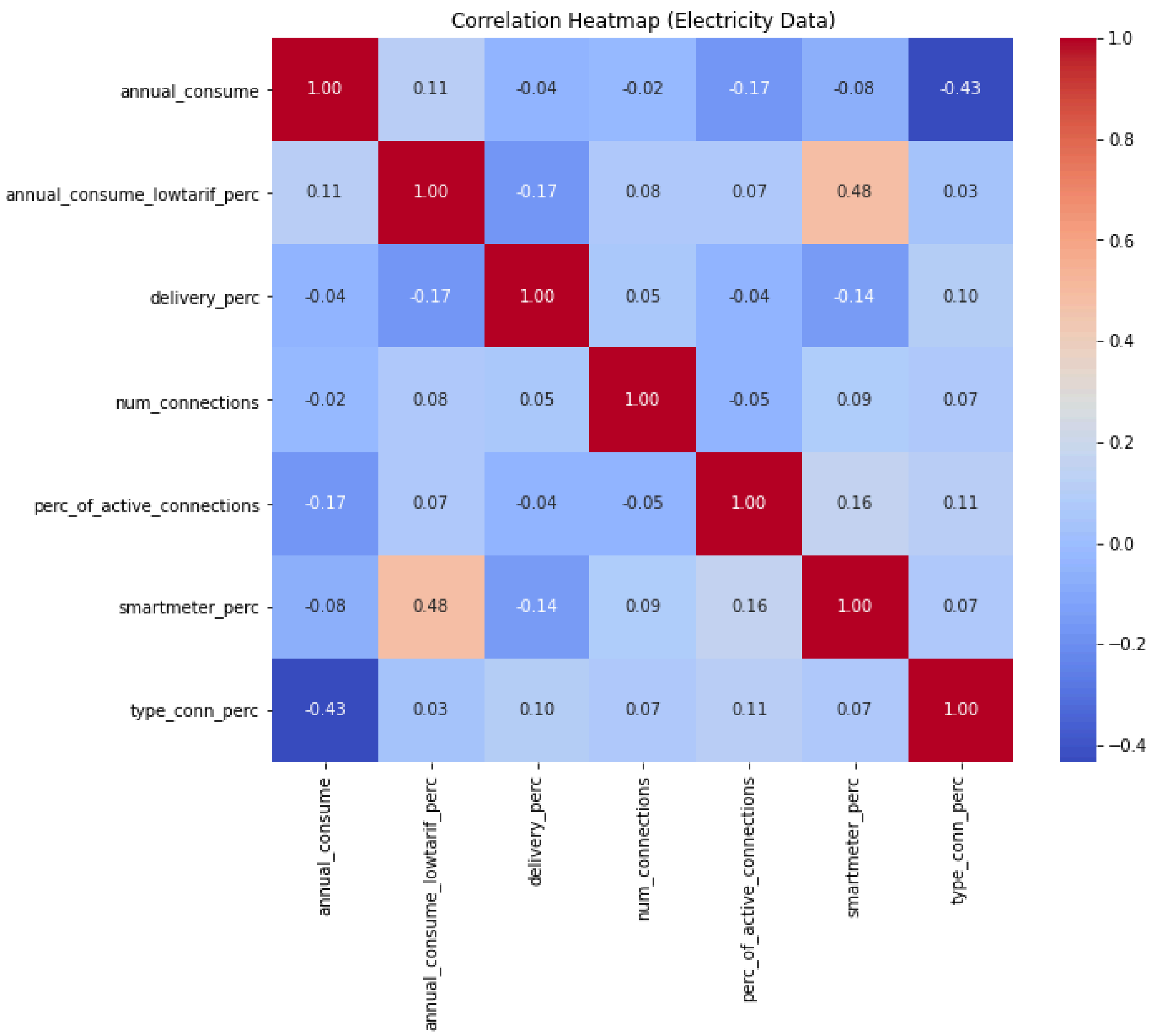
In [0]:
```python
gas_pandas_df["type_of_connection"].value_counts()
```

```
Out[29]: G4      182367
G6        4243
OBK        677
G16        216
G10         57
G25         46
Name: type_of_connection, dtype: int64
```

Generate correlation heatmaps for numerical columns to identify potential linear relationships between features. The result shows that most features have weak or no correlations.

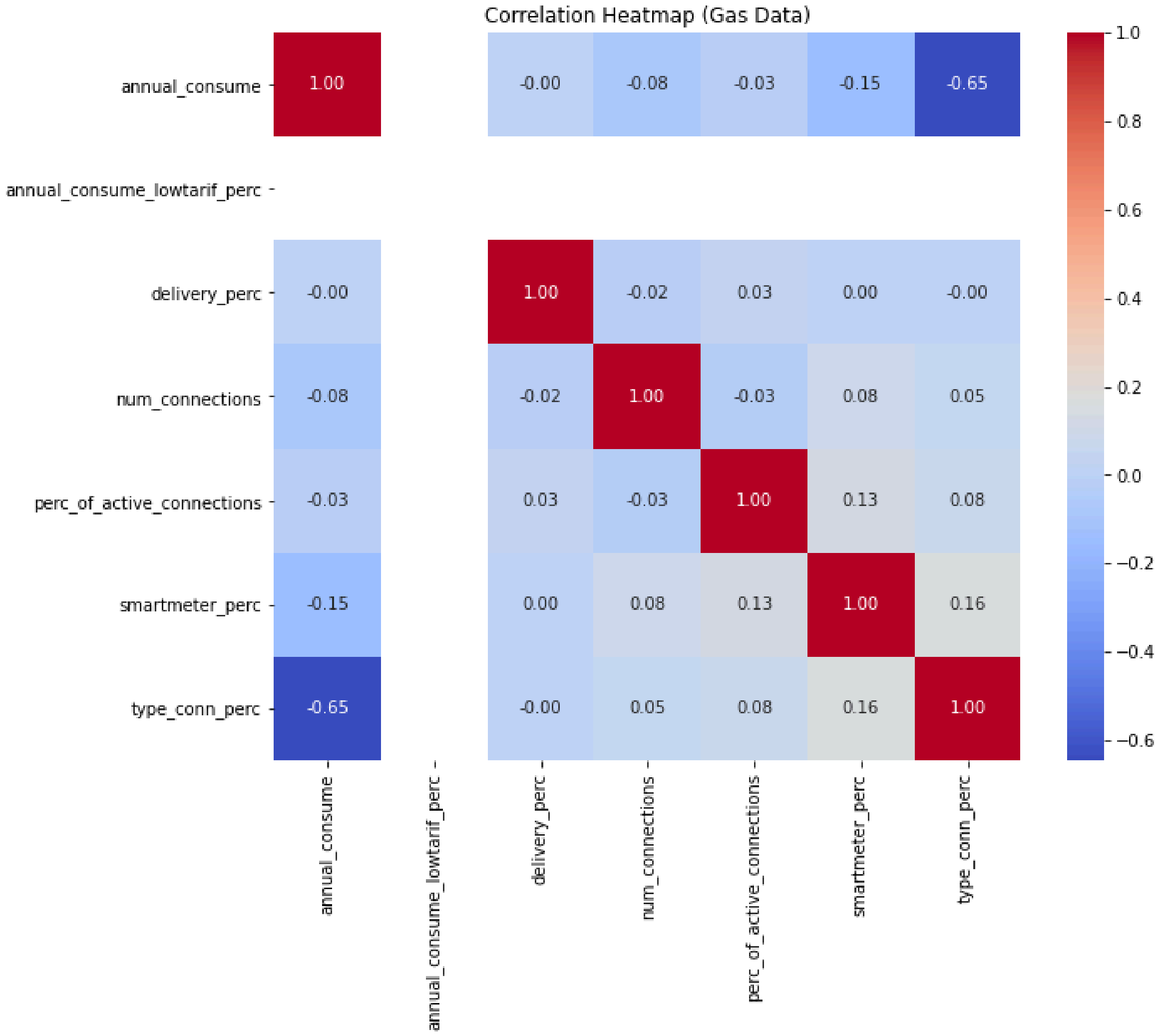
In [0]:

```python
import seaborn as sns

# Heatmap for Electricity Data
plt.figure(figsize=(10, 8))
sns.heatmap(electricity_pandas_df[numerical_columns].corr(), annot=True, fmt=".2
plt.title("Correlation Heatmap (Electricity Data)")
plt.show()
```



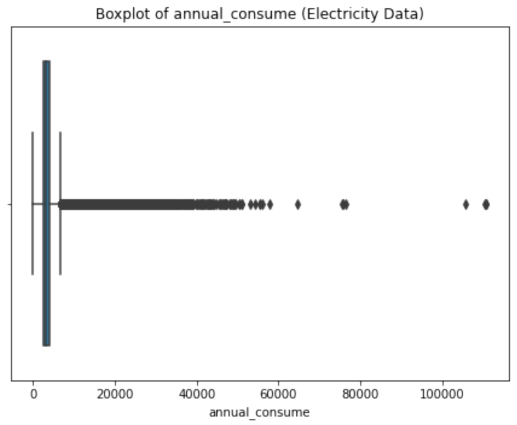
In [0]:

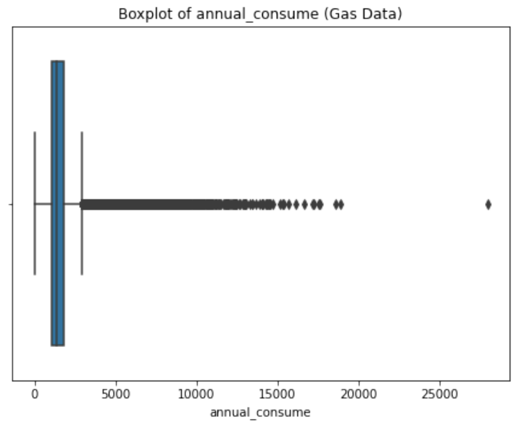
```python
# Heatmap for Gas Data
plt.figure(figsize=(10, 8))
sns.heatmap(gas_pandas_df[numerical_columns].corr(), annot=True, fmt=".2f", cmap
plt.title("Correlation Heatmap (Gas Data)")
plt.show()
```

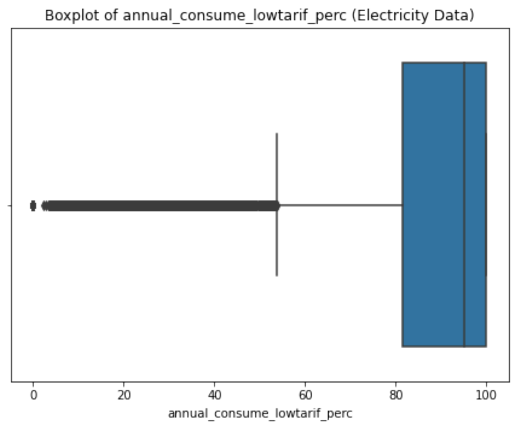


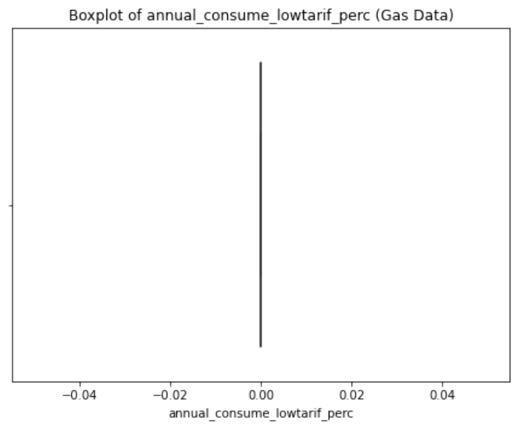
Generates boxplots for numerical columns to visualize the distributions and identify potential outliers. The boxplots reveal that several features, such as **annual_consume** and **num_connections**, have significant outliers in both datasets, indicating highly skewed distributions. For instance, **annual_consume** shows a long tail with extreme values, suggesting that these outliers may need to be handled during data cleaning to prevent them from negatively impacting model performance. Additionally, features like **delivery_perc** in the gas dataset appear to have no variability, as the values are constant, which might make the feature redundant for modeling and should be considered for removal.

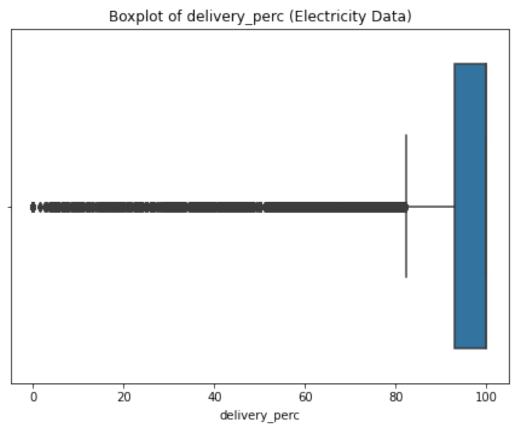
In [0]:
```
fig, axes = plt.subplots(len(numerical_columns), 2, figsize=(14, 5 * len(numeric
fig.tight_layout(w_pad=10.0, h_pad=5.0)

for i, col in enumerate(numerical_columns):
    sns.boxplot(x=electricity_pandas_df[col], ax=axes[i, 0])
    axes[i, 0].set_title(f"Boxplot of {col} (Electricity Data)")
    axes[i, 0].set_xlabel(col)

    sns.boxplot(x=gas_pandas_df[col], ax=axes[i, 1])
    axes[i, 1].set_title(f"Boxplot of {col} (Gas Data)")
    axes[i, 1].set_xlabel(col)

plt.show()
```

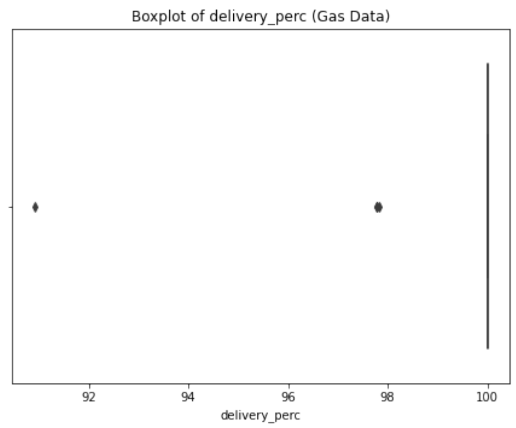

Boxplot of annual_consume (Electricity Data)
0
20000
40000
60000
80000
100000
annual_consume

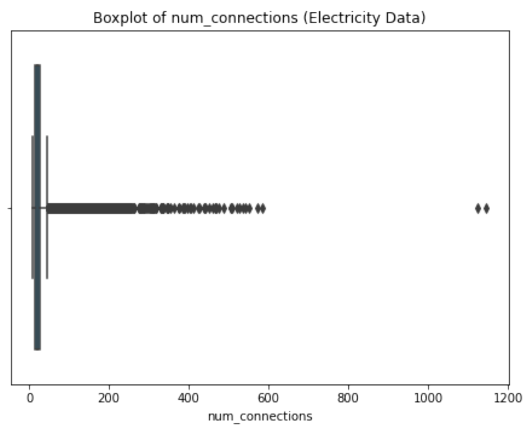

Boxplot of annual_consume (Gas Data)
0
5000
10000
15000
20000
25000
annual_consume

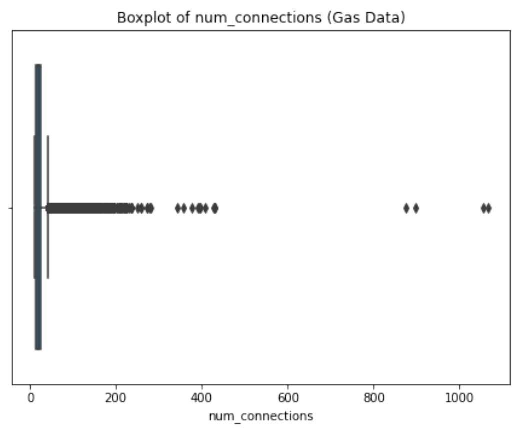

Boxplot of annual_consume_lowtarif_perc (Electricity Data)
0
20
40
60
80
100
annual_consume_lowtarif_perc

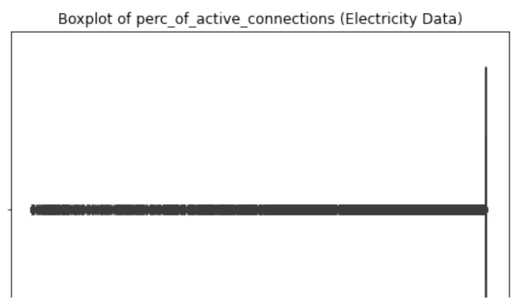

Boxplot of annual_consume_lowtarif_perc (Gas Data)
−0.04
−0.02
0.00
0.02
0.04
annual_consume_lowtarif_perc

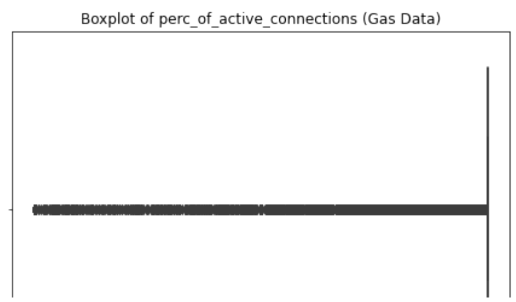

Boxplot of delivery_perc (Electricity Data)
0
20
40
60
80
100
delivery_perc

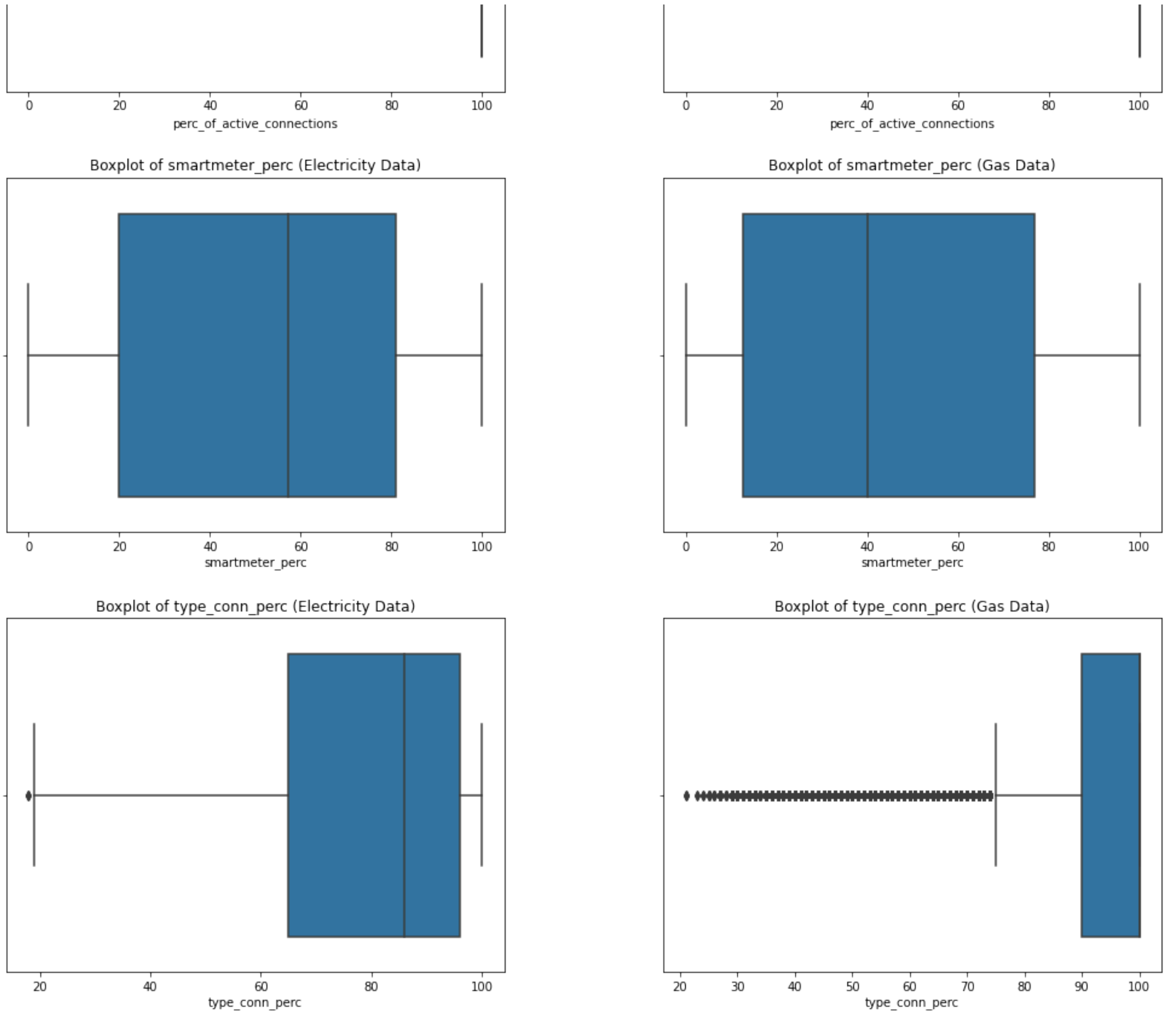

Boxplot of delivery_perc (Gas Data)
92
94
96
98
100
delivery_perc


Boxplot of num_connections (Electricity Data)
0
200
400
600
800
1000
1200
num_connections


Boxplot of num_connections (Gas Data)
0
200
400
600
800
1000
num_connections


Boxplot of perc_of_active_connections (Electricity Data)


Boxplot of perc_of_active_connections (Gas Data)



Question 2A: What are the chosen data cleaning steps? Why?

Step 1: Drop Columns with Missing Values

The **%Defintieve aansl (NRM)** column has an overwhelming number of missing values (171,965 in the electricity dataset and 182,994 in the gas dataset), accounting for the majority of rows. Imputation for such a large percentage of missing values would be impractical and unreliable, while retaining the column may introduce noise to the analysis. Since it does not provide meaningful information for modeling, this column is dropped entirely from all datasets.

In [0]:
```
electricity_train_df = electricity_train_df.drop("%Defintieve aansl (NRM)")
gas_train_df = gas_train_df.drop("%Defintieve aansl (NRM)")
electricity_test_df = electricity_test_df.drop("%Defintieve aansl (NRM)")
gas_test_df = gas_test_df.drop("%Defintieve aansl (NRM)")
```

Step 2: Drop Columns with Low Analytical Values

Columns such as **_id**, **street**, **zipcode_from**, and **zipcode_to** are identifiers or high-cardinality features that do not provide meaningful patterns for analysis. Encoding them would lead to sparse datasets, adding computational overhead without improving model accuracy. These columns are removed to focus on features with higher analytical value.

In [0]:
```
columns_to_drop = ["_id", "street", "zipcode_from", "zipcode_to"]

electricity_train_df = electricity_train_df.drop(*columns_to_drop)
gas_train_df = gas_train_df.drop(*columns_to_drop)
electricity_test_df = electricity_test_df.drop(*columns_to_drop)
gas_test_df = gas_test_df.drop(*columns_to_drop)
```

Step 3: Drop Columns with Constant Values

The **annual_consume_lowtarif_perc** column in the gas dataset contains a constant value of 0 across all rows, offering no variability or predictive power for the model. Retaining this column would introduce redundancy without contributing to the model's performance. Thus, this column is removed to simplify the dataset and improve efficiency.

In [0]:
```
gas_train_df = gas_train_df.drop("annual_consume_lowtarif_perc")
gas_test_df = gas_test_df.drop("annual_consume_lowtarif_perc")
```

Step 4: Handle Outliers in Numerical Columns

Exploratory analysis revealed extreme outliers in numerical columns such as **annual_consume**, **num_connections**, and **type_conn_perc**, with maximum values significantly exceeding the mean. These outliers distort statistical summaries and can negatively affect model training. To address this, log transformation is applied to these columns, reducing skewness and stabilizing variance, ensuring the data is more suitable for predictive modeling.

In [0]:
```
from pyspark.sql.functions import log1p

columns_to_transform = ["annual_consume", "num_connections", "type_conn_perc"]
for col in columns_to_transform:
    electricity_train_df = electricity_train_df.withColumn(col, log1p(electricit
    gas_train_df = gas_train_df.withColumn(col, log1p(gas_train_df[col]))
    electricity_test_df = electricity_test_df.withColumn(col, log1p(electricity_
    gas_test_df = gas_test_df.withColumn(col, log1p(gas_test_df[col]))
```

Step 5: Verify Data Integrity

After cleaning, it is essential to ensure data integrity by checking for duplicate rows, missing values, or anomalies in the datasets. Duplicate rows are removed to prevent biases during modeling, and missing value checks confirm that no additional imputation or removal is required. This step ensures the datasets are fully prepared for transformation and analysis.

In [0]:
```
# Remove duplicates
electricity_train_df = electricity_train_df.dropDuplicates()
gas_train_df = gas_train_df.dropDuplicates()
```

```
electricity_test_df = electricity_test_df.dropDuplicates()
gas_test_df = gas_test_df.dropDuplicates()
```

In [0]:
```
from pyspark.sql.functions import col, sum as _sum

# Verify missing values
display(electricity_train_df.select([_sum(col(c).isNull().cast("int")).alias(c)
```

annual_consume	annual_consume_lowtarif_perc	city	delivery_perc	net_manager	num
0	0	0	0	0	

In [0]:
```
display(gas_train_df.select([_sum(col(c).isNull().cast("int")).alias(c) for c in
```

annual_consume	city	delivery_perc	net_manager	num_connections	perc_of_active_co
0	0	0	0	0	

Question 2B: What are the chosen data transformation steps? Why?

Step 1: Encode Categorical Variables

Categorical variables, such as **city**, **net_manager**, **purchase_area**, and **type_of_connection**, are non-numeric and cannot be directly used by most machine learning algorithms. Using `StringIndexer`, each category is assigned a unique numerical index, capturing categorical distinctions. To avoid introducing ordinal bias (where numerical values imply ranking), these indices are converted into binary vectors using `OneHotEncoder`. This step ensures that all categorical variables are represented in a way that preserves their relationships without misinterpreting their values as numeric magnitudes.

In [0]:
```
from pyspark.ml.feature import StringIndexer, OneHotEncoder

categorical_columns = ["city", "net_manager", "purchase_area", "type_of_connecti

# String Indexing and One-Hot Encoding for each categorical column
indexers = [StringIndexer(inputCol=col, outputCol=f"{col}_index", handleInvalid=
encoders = [OneHotEncoder(inputCol=f"{col}_index", outputCol=f"{col}_onehot") fo

# Combine all transformations into a list
categorical_transformations = indexers + encoders
```

Step 2: Scale Numerical Features

Numerical features, such as **num_connections** and **type_conn_perc**, have varying scales and units, as observed during data exploration. Features with larger magnitudes may disproportionately influence model training, especially for algorithms like decision trees.

To mitigate this, `MinMaxScaler` is applied to normalize all numerical features to a range of [0, 1], ensuring equal contribution of all variables during training.

In [0]:
```
from pyspark.ml.feature import MinMaxScaler, VectorAssembler

numerical_columns = ["num_connections", "type_conn_perc"]

# Assemble numerical columns into a single vector for scaling
assembler = VectorAssembler(inputCols=numerical_columns, outputCol="numerical_fe
scaler = MinMaxScaler(inputCol="numerical_features", outputCol="scaled_numerical

# Combine assembler and scaler into a list
numerical_transformations = [assembler, scaler]
```

Step 3: Assemble Features

Machine learning models typically require a single vector column containing all the features. After encoding categorical variables and scaling numerical features, the `VectorAssembler` consolidates all transformed columns (scaled_numerical_features and one-hot encoded categorical columns) into a unified features column. This step ensures that the data is structured consistently and efficiently for input into machine learning pipelines.

In [0]:
```
all_features = [f"{col}_onehot" for col in categorical_columns] + ["scaled_numer

# Assemble all features into a single vector
feature_assembler = VectorAssembler(inputCols=all_features, outputCol="features"
```

Task 3: Model training and tracking with data pipeline and MLflow

Data Preparation

In [0]:
```
# Target column
target_column = "annual_consume"

# Prepare datasets with labeled target column
electricity_train_df = electricity_train_df.withColumnRenamed(target_column, "la
electricity_test_df = electricity_test_df.withColumnRenamed(target_column, "labe
gas_train_df = gas_train_df.withColumnRenamed(target_column, "label")
gas_test_df = gas_test_df.withColumnRenamed(target_column, "label")
```

Define Reusable Training and Logging Pipeline

In [0]:
```
from pyspark.ml import Pipeline
from pyspark.ml.regression import RandomForestRegressor, DecisionTreeRegressor,
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.functions import col
import mlflow
import mlflow.spark
```

```python
def train_and_log_model(train_data, test_data, algorithm, params):
    with mlflow.start_run():
        # Initialize the model
        if algorithm == "RandomForestRegressor":
            model = RandomForestRegressor(featuresCol="features", labelCol="labe
        elif algorithm == "DecisionTreeRegressor":
            model = DecisionTreeRegressor(featuresCol="features", labelCol="labe

        # Create pipeline
        pipeline = Pipeline(stages=categorical_transformations + numerical_trans

        # Train the model
        trained_pipeline = pipeline.fit(train_data)

        # Predict on test data
        predictions = trained_pipeline.transform(test_data)

        # Evaluate the model
        evaluator_mae = RegressionEvaluator(labelCol="label", predictionCol="pre
        evaluator_r2 = RegressionEvaluator(labelCol="label", predictionCol="pred
        evaluator_rmse = RegressionEvaluator(labelCol="label", predictionCol="pr

        mae = evaluator_mae.evaluate(predictions)
        r2 = evaluator_r2.evaluate(predictions)
        rmse = evaluator_rmse.evaluate(predictions)

        # Log parameters, metrics, and the model
        mlflow.log_param("algorithm", algorithm)
        mlflow.log_params(params)
        mlflow.log_metric("MAE", mae)
        mlflow.log_metric("R2", r2)
        mlflow.log_metric("RMSE", rmse)
        mlflow.spark.log_model(trained_pipeline, "model")

        print(f"Logged {algorithm} with params {params}")
        print("Evaluation metrics:")
        print(f"- MAE: {mae}")
        print(f"- R2: {r2}")
        print(f"- RMSE: {rmse}")
```

Define Different Parameter Settings for Each Algorithm.

In [0]:

```python
# Random Forest parameters
random_forest_params = [
    {"numTrees": 10, "maxDepth": 3},  # Small number of trees, shallow depth
    {"numTrees": 20, "maxDepth": 5},  # Slightly deeper with more trees
    {"numTrees": 30, "maxDepth": 7}   # Moderate depth and tree count
]

# Decision Tree parameters
decision_tree_params = [
    {"maxDepth": 3, "minInstancesPerNode": 1},  # Shallow tree
    {"maxDepth": 5, "minInstancesPerNode": 2},  # Slightly deeper
    {"maxDepth": 7, "minInstancesPerNode": 2}   # Moderate depth
]
```

Electricity Model Training

Random Forest Regressor Model

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "RandomForestRegr
```

```
Logged RandomForestRegressor with params {'numTrees': 10, 'maxDepth': 3}
Evaluation metrics:
- MAE: 0.3245205459858661
- R2: 0.3055551751803406
- RMSE: 0.43719801389464497
```

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "RandomForestRegr
```

```
Logged RandomForestRegressor with params {'numTrees': 20, 'maxDepth': 5}
Evaluation metrics:
- MAE: 0.30309765928996096
- R2: 0.4100154729258382
- RMSE: 0.4029764052916728
```

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "RandomForestRegr
```

```
Logged RandomForestRegressor with params {'numTrees': 30, 'maxDepth': 7}
Evaluation metrics:
- MAE: 0.2924566271932784
- R2: 0.45467513836524165
- RMSE: 0.38742438864546014
```

Decision Tree Regressor Model

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "DecisionTreeRegr
```

```
Logged DecisionTreeRegressor with params {'maxDepth': 3, 'minInstancesPerNode':
1}
Evaluation metrics:
- MAE: 0.3198013444238246
- R2: 0.33819237191648355
- RMSE: 0.4268007675394118
```

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "DecisionTreeRegr
```

```
Logged DecisionTreeRegressor with params {'maxDepth': 5, 'minInstancesPerNode':
2}
Evaluation metrics:
- MAE: 0.3005998752347698
- R2: 0.41813901313332635
- RMSE: 0.40019248323442663
```

In [0]:
```
train_and_log_model(electricity_train_df, electricity_test_df, "DecisionTreeRegr
```

```
Logged DecisionTreeRegressor with params {'maxDepth': 7, 'minInstancesPerNode':
2}
Evaluation metrics:
- MAE: 0.2919387583533915
- R2: 0.4505358781081866
- RMSE: 0.3888919714346681
```

Gas Model Training

Random Forest Regressor Model

In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "RandomForestRegressor", random_f
```

```
Logged RandomForestRegressor with params {'numTrees': 10, 'maxDepth': 3}
Evaluation metrics:
- MAE: 0.4311770564844025
- R2: 0.1673514212579711
- RMSE: 0.7598437611829207
```

In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "RandomForestRegressor", random_f
```

```
Logged RandomForestRegressor with params {'numTrees': 20, 'maxDepth': 5}
Evaluation metrics:
- MAE: 0.4384877355674457
- R2: 0.06385954112566983
- RMSE: 0.8056825121370964
```

In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "RandomForestRegressor", random_f
```

```
Logged RandomForestRegressor with params {'numTrees': 30, 'maxDepth': 7}
Evaluation metrics:
- MAE: 0.5128250100815277
- R2: 0.12755182278567023
- RMSE: 0.7777916012613559
```

Decision Tree Regressor Model

In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "DecisionTreeRegressor", decision
```

```
Logged DecisionTreeRegressor with params {'maxDepth': 3, 'minInstancesPerNode':
1}
Evaluation metrics:
- MAE: 0.4292077809910484
- R2: 0.17396052800107087
- RMSE: 0.7568221421899121
```

In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "DecisionTreeRegressor", decision
```

```
Logged DecisionTreeRegressor with params {'maxDepth': 5, 'minInstancesPerNode':
2}
Evaluation metrics:
- MAE: 0.4210915936830995
- R2: 0.19771574365696465
- RMSE: 0.745860425127864
```

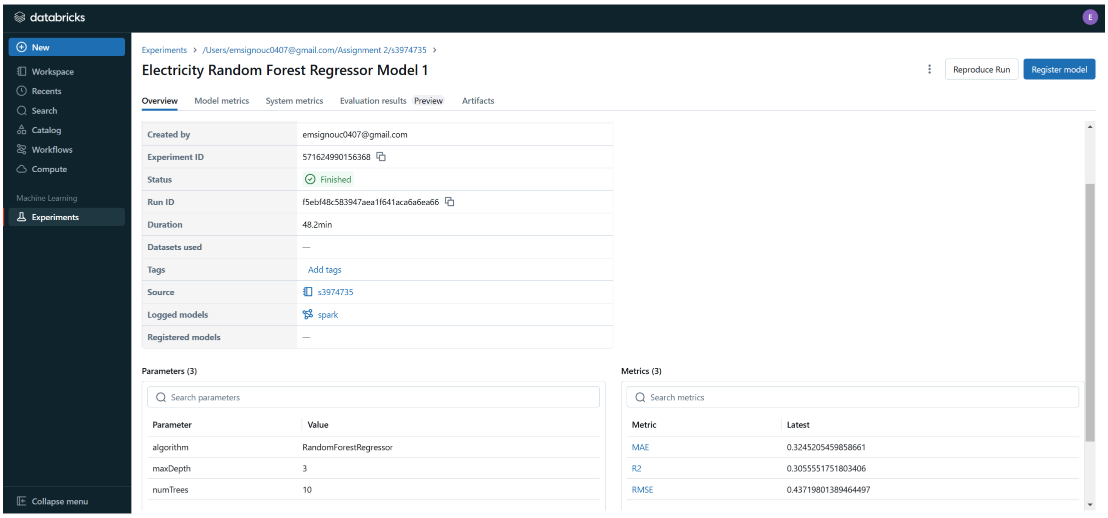
In [0]:
```
train_and_log_model(gas_train_df, gas_test_df, "DecisionTreeRegressor", decision
```

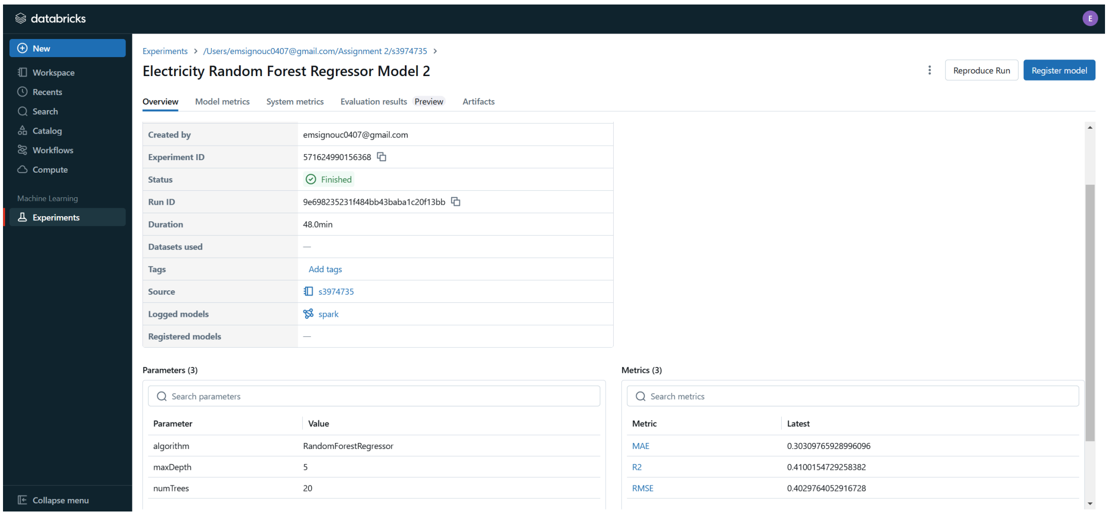
```
Logged DecisionTreeRegressor with params {'maxDepth': 7, 'minInstancesPerNode':
2}
Evaluation metrics:
- MAE: 0.4177651838941951
- R2: 0.20934962369445542
- RMSE: 0.7404328364065984
```

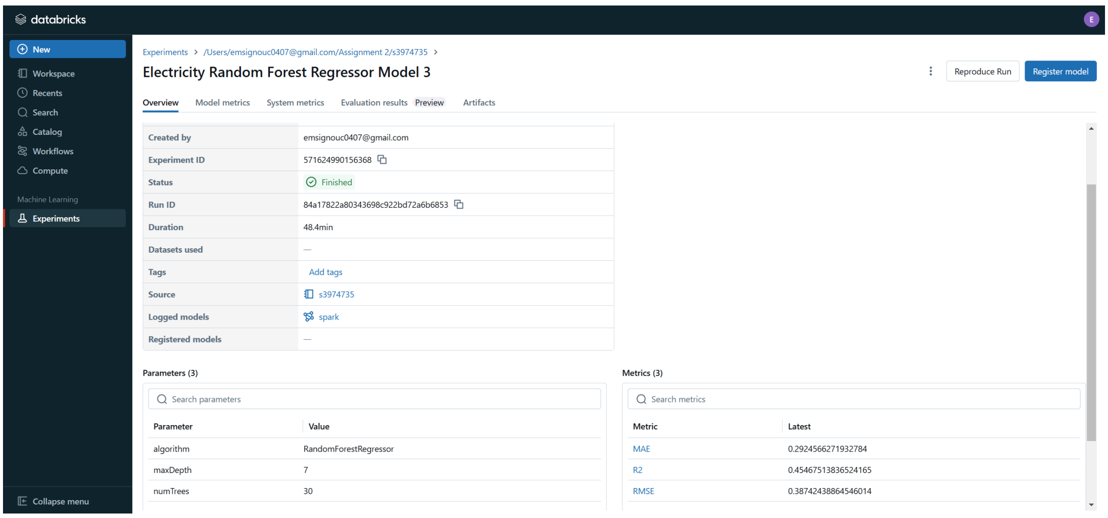
Question 3A: What is/are your final model(s) based on the evaluation metrics?

Electricity Model

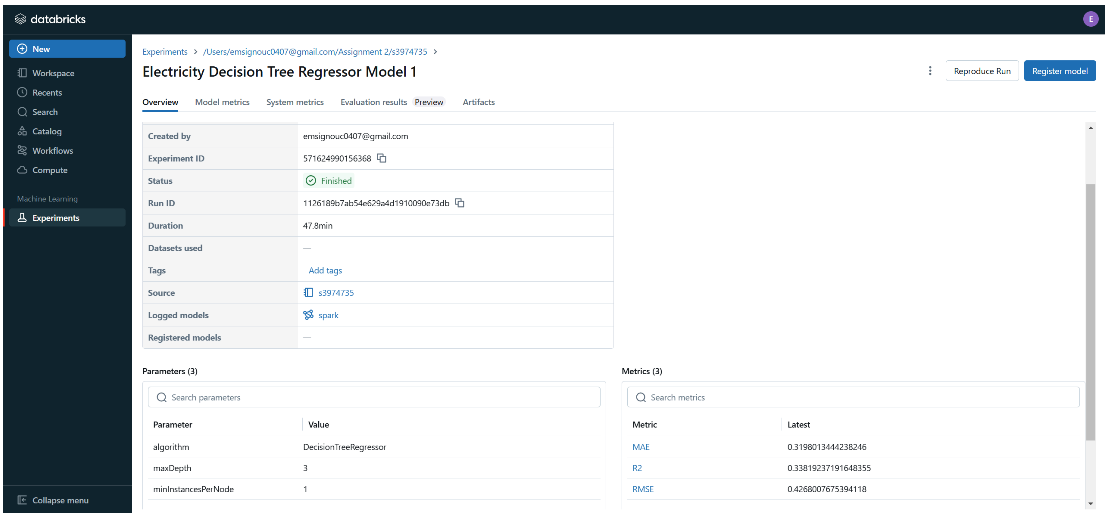
MLflow UI for Electricity Random Forest Regressor Models:

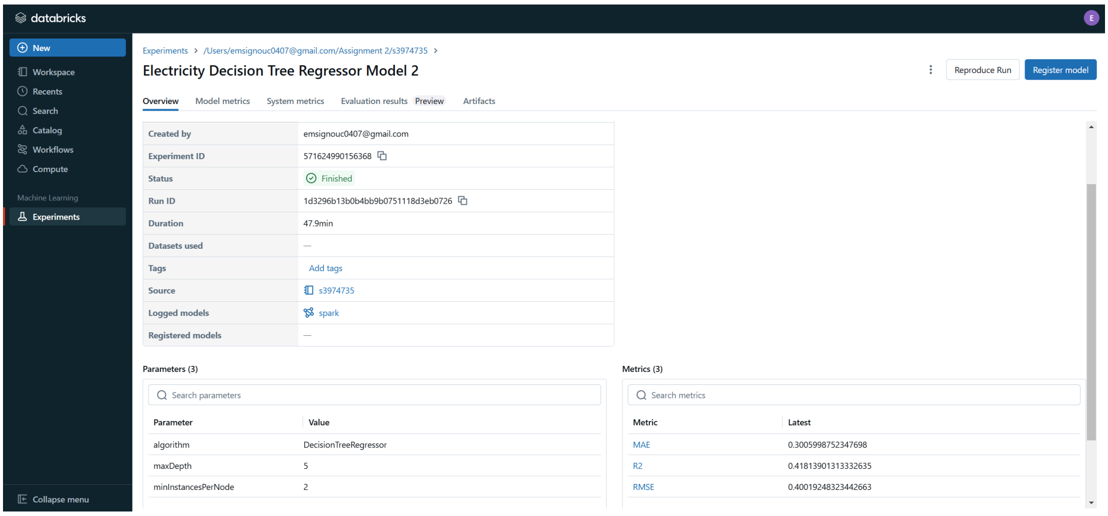


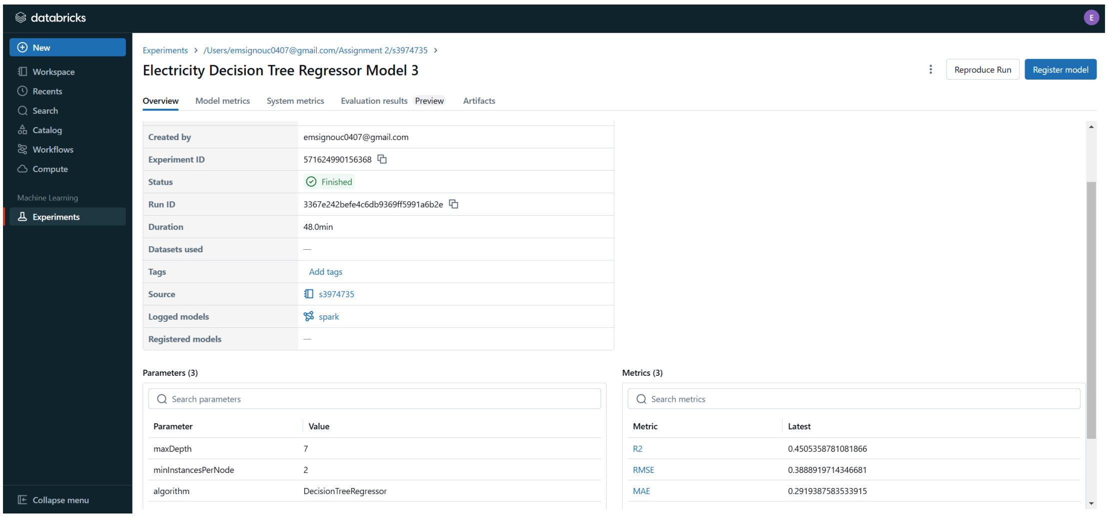




MLflow UI for Electricity Decision Tree Regressor Models:







Based on the evaluation metrics, the best-performing model for the electricity dataset is the **RandomForestRegressor** with parameters `{'numTrees': 30, 'maxDepth': 7}`. This model achieved the lowest MAE of 0.2925, the highest R2 of 0.4547, and the lowest RMSE of 0.3874 among all tested models and parameter settings. These metrics indicate that this configuration provides the most accurate predictions with the least error and the best explained variance.

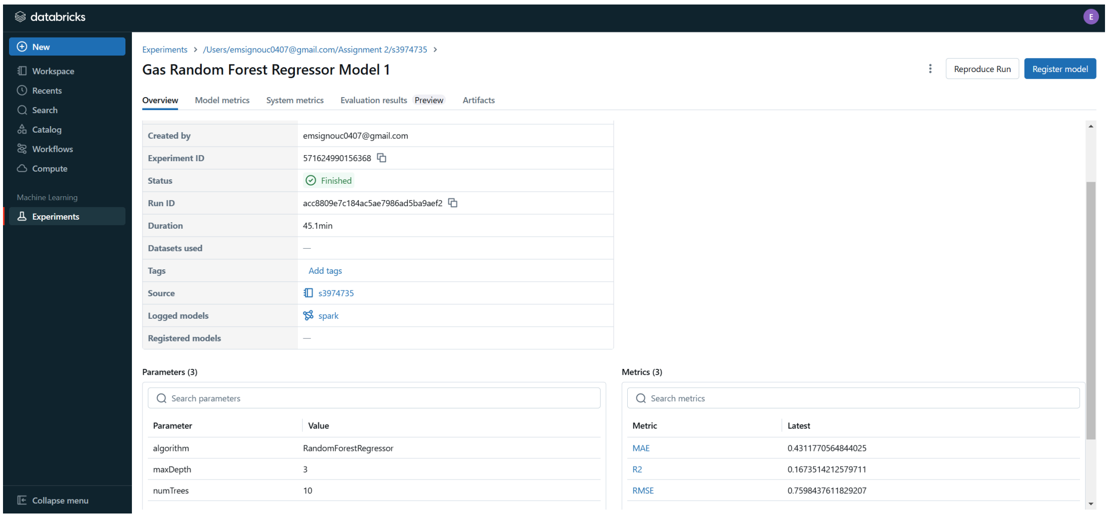
The **DecisionTreeRegressor** with parameters `{'maxDepth': 7, 'minInstancesPerNode': 2}` performed closely, achieving an MAE of 0.2919, an R2 of

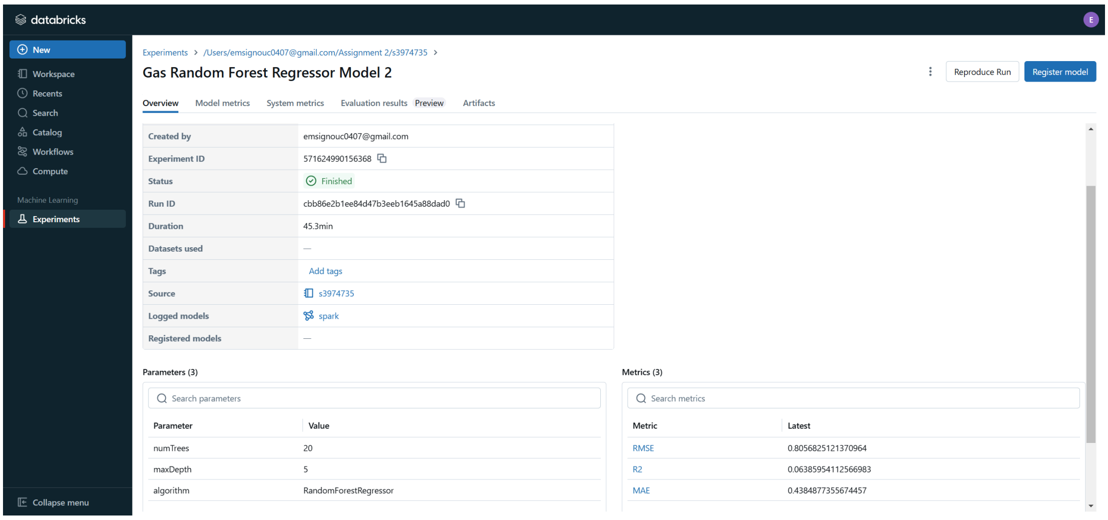
0.4505, and an RMSE of 0.3889. However, the RandomForestRegressor slightly outperformed it, likely due to its ensemble nature, which reduces variance and enhances prediction stability.

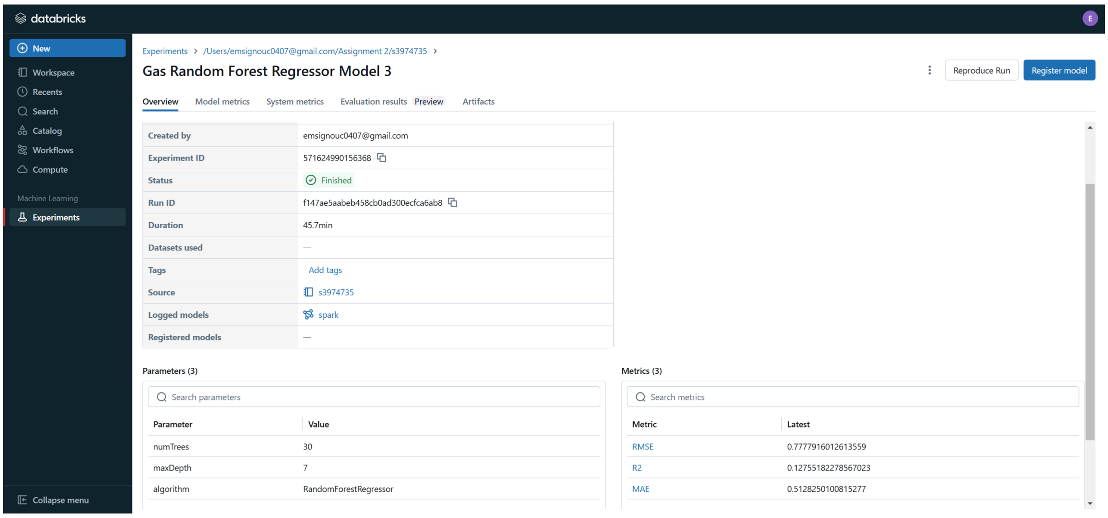
Thus, the final model for the electricity dataset is the **RandomForestRegressor** with `{'numTrees': 30, 'maxDepth': 7}` due to its superior performance across all evaluation metrics.

Gas Model

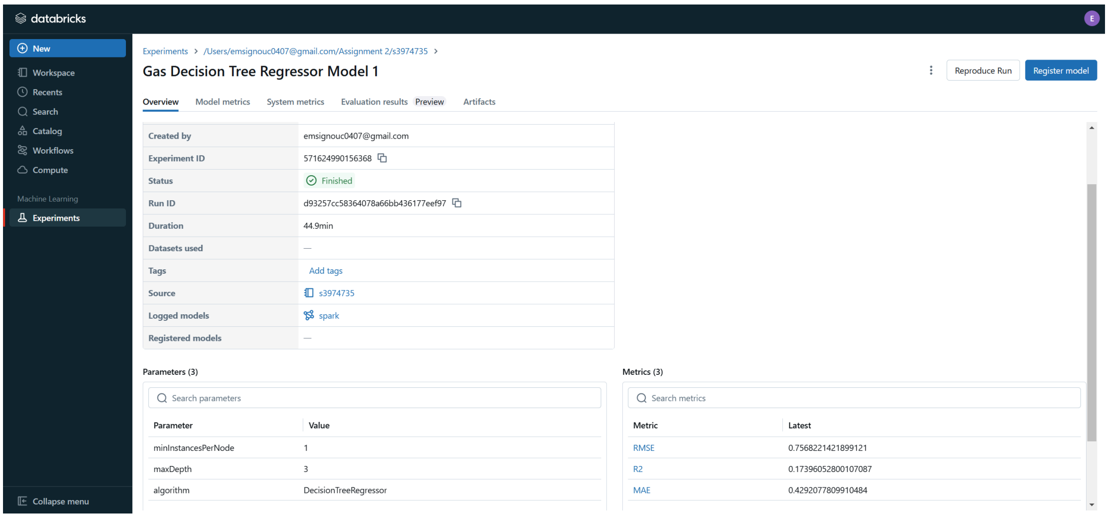
MLflow UI for Gas Random Forest Regressor Models:

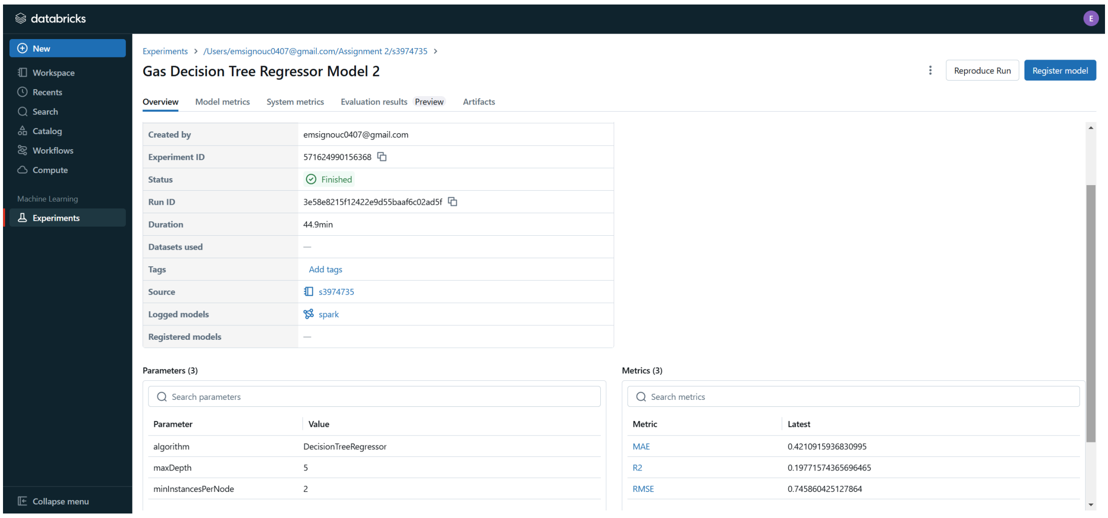


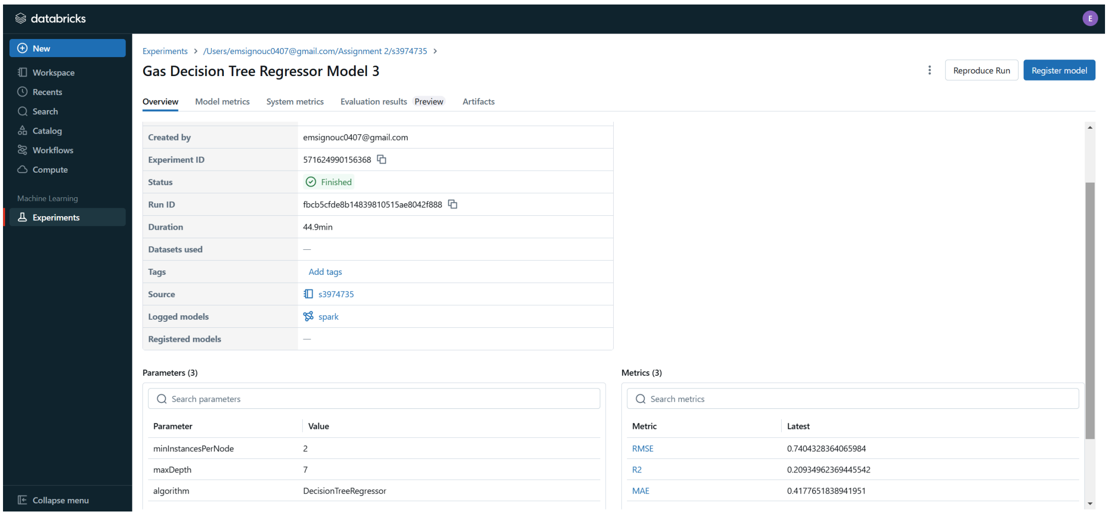




MLflow UI for Gas Decision Tree Regressor Models:







Based on the evaluation metrics, the best-performing model for the gas dataset is the **DecisionTreeRegressor** with parameters `{'maxDepth': 7, 'minInstancesPerNode': 2}`. This model achieved the lowest MAE of 0.4178, the highest R2 of 0.2093, and the lowest RMSE of 0.7404 among all tested configurations. These metrics suggest that this model has the best balance of accuracy, explained variance, and minimal error compared to other tested models.

Although the **RandomForestRegressor** was evaluated with multiple configurations, its performance was consistently inferior to the **DecisionTreeRegressor**. The highest R2 achieved by the **RandomForestRegressor** was only 0.1674, with an RMSE of 0.7598 and an MAE of 0.4312, indicating lower predictive accuracy and higher error rates.

Thus, the final model for the gas dataset is the **DecisionTreeRegressor** with parameters `{'maxDepth': 7, 'minInstancesPerNode': 2}` due to its superior performance across all evaluation metrics.

Question 3B: Did you build one model for both electricity and gas or separate models? Why?

Separate models were built for the electricity and gas datasets. This decision stems from the insights gained during data exploration, which revealed significant differences between the two datasets:

- The **annual_consume** column in the electricity dataset has a much higher mean of 3913 and maximum value of 110,857 compared to the gas dataset, where the mean is 1559 and the maximum is 27,917. This indicates fundamentally different consumption behaviors between the two datasets.
- Numerical columns such as **num_connections** and **type_conn_perc** exhibit distinct patterns in the two datasets. Electricity data shows higher variability in **num_connections**, with values reaching 1146, while gas data has a maximum of 1065. Similarly, **type_conn_perc** is generally higher in the gas dataset.
- Categorical columns like **city**, **net_manager**, and **purchase_area** have varying levels of diversity and dominant categories. For instance, the electricity dataset has 287

unique cities, while the gas dataset has 491. Additionally, the **purchase_area** column features entirely different dominant categories for each dataset.
- Correlation heatmaps showed weak or no relationships between most numerical features in both datasets. The lack of shared strong correlations indicates that the datasets would likely require different feature importance considerations during model training.
- The **annual_consume_lowtarif_perc** column in the gas dataset is constant with a value of 0.0 and adds no value to model predictions, whereas it has variability in the electricity dataset.

These differences demonstrate that the datasets represent distinct consumption patterns, influenced by their respective geographical, infrastructural, and operational contexts. Building separate models ensures that each model is tailored to the unique characteristics of its dataset, thereby improving the accuracy, robustness, and interpretability of the predictions.

Question 3C: Should we build a separate model for each company or not? Why?

Although for this project we are working with the datasets of three different companies, it is unnecessary to create separate models for each company. This decision is supported by several observations:

- During data exploration, it was found that while the **net_manager** and **purchase_area** columns influence data distribution, they do not exhibit substantial variability in their effect on the target variable, **annual_consume**. The target variable is more strongly influenced by features like **num_connections**, **delivery_perc**, and **type_conn_perc**, which are not tied to specific companies.
- **net_manager** and **purchase_area** columns also have high cardinality and sparse representation for some companies. Building separate models would result in less-represented companies having sparse datasets, potentially causing overfitting and reducing the generalizability of the models.
- Key features such as **annual_consume_lowtarif_perc** and **smartmeter_perc** display similar predictive relationships across companies. This indicates that a single model can effectively generalize across companies without requiring specific adjustments.
- Creating separate models for each company would increase computational complexity, development time, and maintenance effort. A single model for the electricity dataset and another for the gas dataset ensures a scalable and efficient solution while maintaining strong predictive accuracy.
- By including company-specific features like **net_manager** as categorical inputs in the models, the differences between companies can still be accounted for without needing separate models.

For these reasons, building one model for each dataset is sufficient to achieve accurate predictions while keeping the solution simple, scalable, and easy to manage.

Task 4: Visualisation

Link to MongoDB Charts Dashboard: https://charts.mongodb.com/charts-bigdataasm2-szigrao/public/dashboards/67724c59-0c78-4054-8e6b-1061df46332b

Chart 1: Top 10 Cities by Electricity Annual Consumption in 2018

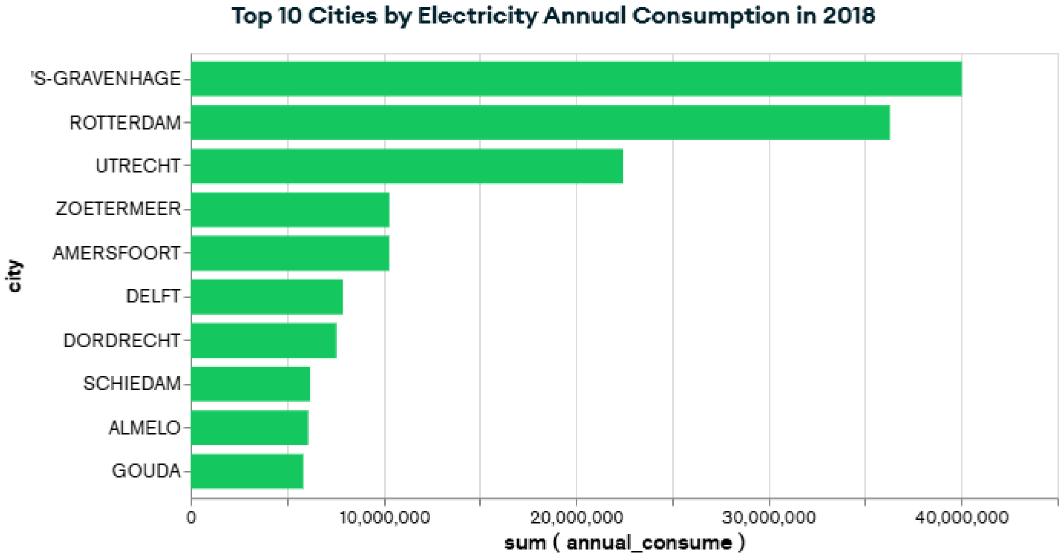


Chart 2: Electricity Distribution of Connection Types in 2019

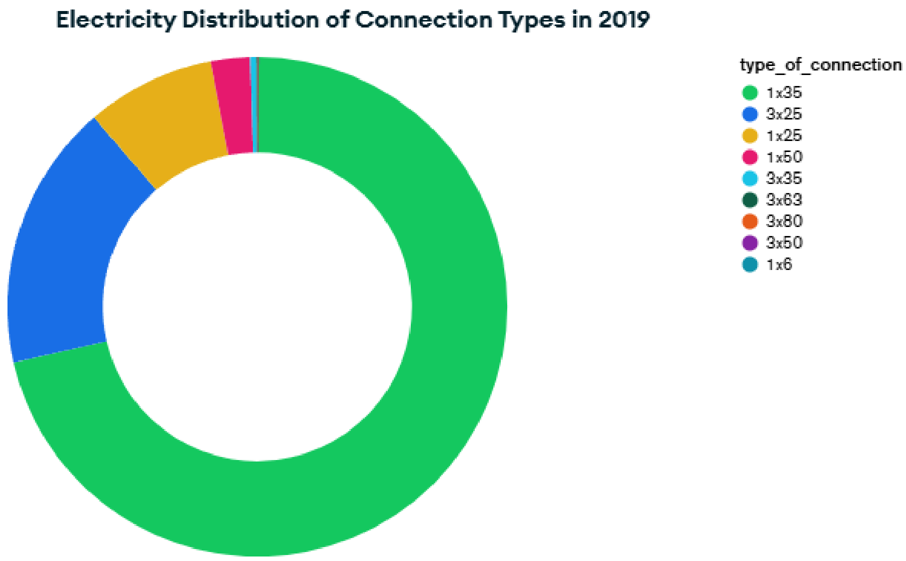


Chart 3: Top 10 Cities by Gas Annual Consumption in 2018

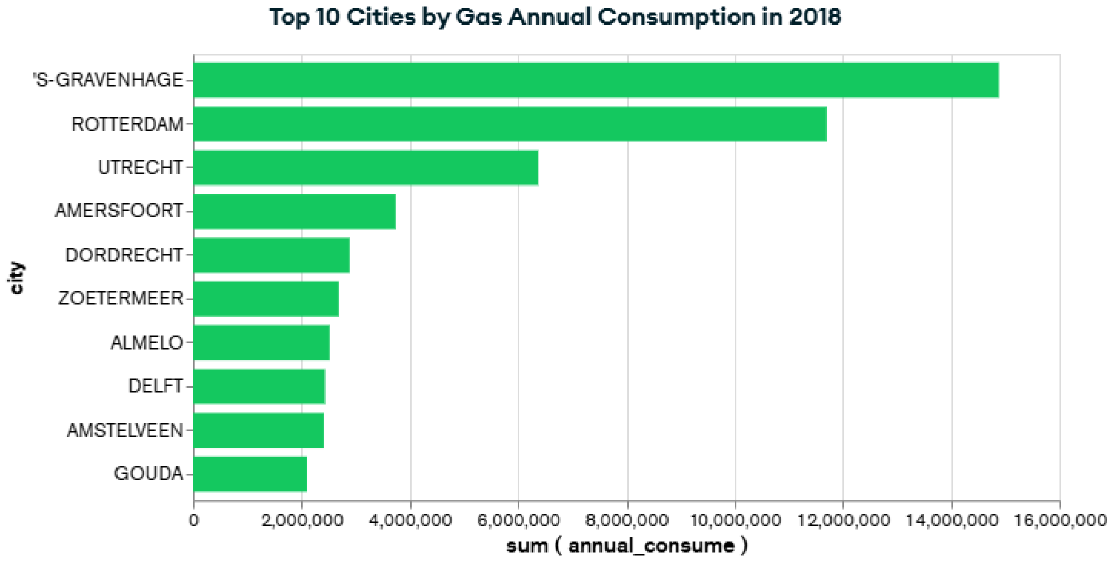


Chart 4: Gas Distribution of Connection Types in 2019