Workshop #1 - Documentation

Martín García Chagüezá

Contents

About the project

In this workshop we use randomly generated data describing each candidate registered in a selection process. The dataset used (candidates.csv) has 50,000 rows and 10 columns, we put it through loading, cleaning and transformation processes to find interesting insights about the candidates hired using BI tools.

The tools used are:

- Python 3.12 ➔ Download site

- Jupyter Notebook ➜ VS Code tool for using notebooks
- PostgreSQL ➜ Download site
- Power BI (Desktop version) ➜ Download site

The libraries needed for Python are:

- Pandas
- Matplotlib
- Seaborn
- SQLAlchemy
- Dotenv

However, these libraries are included in the Poetry project config file (*pyproject.toml*). The step-by-step installation will be described later.

Goals of the project

Obtain a clean dataset for the creation of an analytical report using BI tools such as Power BI, which will be connected through a PostgreSQL database that will contain the transformed data set.

The analytical report will contain visualizations such as:

- ***Hires by technology*** (*pie chart*).
- ***Hires by year*** (*horizontal bar chart*).
- ***Hires by seniority*** (*bar chart*).
- ***Hires by country over years*** (USA, Brazil, Colombia, and Ecuador only) (*multiline chart*).

Process

Creating virtualenv and installing the dependencies with *Poetry*

> *For a tutorial on how to install and configure Poetry, follow the next page → Poetry*

Poetry works as a dependency manager primarily, but it also let us create a virtual enviroment to add or install the dependencies we need for our project. Through `poetry add <dependency>` we can add the libraries to our project: those libraries are going to be registered in the Poetry config file, named ***pyproject.toml***.

If the ***pyproject.toml*** is already in our directory, but we're working in a different virtualenv, we can use the `poetry install` command, as showed in the image, to create the new virtual enviroment and install the registered dependencies in the Poetry configuration file.

```
PS C:\Users\marti\OneDrive\Escritorio - PC\Ingenieria de Datos e IA - UAO\Semestre 4\ETL\Semana #1 - #6\Workshop #1> poetry install
Creating virtualenv workshop-1-Ih9GMGpq-py3.12 in C:\Users\marti\AppData\Local\pypoetry\Cache\virtualenvs
Installing dependencies from lock file

Package operations: 45 installs, 0 updates, 0 removals

  - Installing six (1.16.0)
  - Installing asttokens (2.4.1)
  - Installing executing (2.0.1)
  - Installing numpy (2.1.0)
  - Installing parso (0.8.4)
  - Installing platformdirs (4.2.2)
  - Installing pure-eval (0.2.3)
  - Installing pywin32 (306)
  - Installing traitlets (5.14.3)
  - Installing wcwidth (0.2.13)
  - Installing colorama (0.4.6): Installing...
  - Installing contourpy (1.2.1)
  - Installing cycler (0.12.1)
  - Installing decorator (5.1.1)
  - Installing contourpy (1.2.1)
  - Installing cycler (0.12.1)
  - Installing decorator (5.1.1)
```

Establishing the connection with the DB and loading the raw data

> ***Files used*** → *connection.py* and *001_rawDataLoad.ipynb*

connection.py - Creating the connection engine

The creation of the connection engine it's realized in the ***connection.py*** module.

It uses the SQLAlchemy library alongside the *psycopg2* driver. To establish the connection, it's required to import a SQLA function named `create_engine()`, which in turn requires in its parameters a URL with the database credentials: these credentials are included in environment variables contained in an ***.env*** file; these are fetched from the file before making the connection.

```python
import os

from dotenv import load_dotenv
from sqlalchemy import create_engine


# Reading the environment variables
load_dotenv("../env/.env")

driver = os.getenv("PG_DRIVER")

user = os.getenv("PG_USER")
password = os.getenv("PG_PASSWORD")

host = os.getenv("PG_HOST")
port = os.getenv("PG_PORT")

database = os.getenv("PG_DATABASE")

# Creating the connection engine from the URL made up of the environment variables
def creating_engine():
    url = f"{driver}://{user}:{password}@{host}:{port}/{database}"
    engine = create_engine(url)

    return engine
```

001_rawDataLoad.ipynb - Loading the raw data

We import the connection module for the creation of the engine and Pandas to process the CSV file and manage the DB operations through the engine. Once we imported the needed libraries, we read the dataset with the Pandas function `read_csv()`.

Importing libraries and modules

```python
import pandas as pd
import connection

from sqlalchemy import text
```

Python

Reading the dataset

```python
df = pd.read_csv('../data/candidates.csv', sep=';')
```

Python

After processing the raw data set through Pandas, we create the connection engine using the `creating_engine` function. To load the raw data to our database in PostgreSQL we use the `to_sql()` function of Pandas: the table in which our data is going to be uploaded is named *"candidates_raw"*.

Transfering the data to the database in PostgreSQL

```python
engine = connection.creating_engine()
```

[23] Python

```python
df.to_sql('candidates_raw', engine, if_exists='replace', index=False)
```

[26] Python

```
1000
```

We verify in *pgAdmin* if the data uploaded correctly.

	First Name text	Last Name text	Email text	Application Date text	Country text	YOE bigint	Seniority text	Technology text
1	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer
2	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer
3	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success
4	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual
5	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management
6	Alec	Abbott	juanita_hansen@gmail.com	2019-08-17	Zimbabwe	8	Junior	Adobe Experience Manager
7	Allison	Jacobs	alba_rolfson27@yahoo.com	2018-05-18	Wallis and Futuna	19	Trainee	Sales
8	Nya	Skiles	madisen.zulauf@gmail.com	2021-12-09	Myanmar	1	Lead	Mulesoft
9	Mose	Lakin	dale_murazik@hotmail.com	2018-03-13	Italy	18	Lead	Social Media Community Management
10	Terrance	Zieme	dustin31@hotmail.com	2022-04-08	Timor-Leste	25	Lead	DevOps
11	Aiyana	Goodwin	vallie.damore@yahoo.com	2019-09-22	Armenia	24	Intern	Development - CMS Backend
12	Emilia	Waelchi	peter.grady@gmail.com	2020-07-15	French Southern Territories	28	Lead	DevOps
13	Terrell	Streich	meta92@yahoo.com	2021-12-27	Chad	3	Mid-Level	Salesforce
14	Hilda	Rodriguez	jordan.hyatt@hotmail.com	2020-05-09	El Salvador	16	Junior	System Administration
15	Hope	Hansen	clemmie.bruen@hotmail.com	2019-10-12	Mozambique	18	Architect	Security
16	Arno	Altenwerth	cheyenne_rau2@gmail.com	2018-10-18	Brunei Darussalam	21	Mid-Level	Game Development
17	Betty	Crona	judd.wisozk55@gmail.com	2020-03-25	Morocco	28	Architect	Social Media Community Management
18	Clint	Oberbrunner	dwight_jacobson@gmail.com	2021-05-23	Saint Helena	19	Senior	Social Media Community Management
19	Donny	Boehm	lucile97@hotmail.com	2018-05-02	Portugal	20	Trainee	Development - CMS Frontend
20	Adah	Pouros	derick1@gmail.com	2021-04-13	Mozambique	3	Architect	Adobe Experience Manager
21	Winona	Zboncak	doyle78@yahoo.com	2019-03-21	Central African Republic	29	Architect	Game Development
22	Eulalia	Wiza	lauriane42@gmail.com	2020-07-12	Seychelles	11	Lead	Security Compliance
23	Crawford	Ullrich	bruce.koch7@yahoo.com	2021-01-09	Dominica	14	Junior	Game Development
24	Daphney	Price	einar.schmidt@yahoo.com	2021-07-27	Finland	24	Mid-Level	Development - Backend
25	Shana	Kuphal	sydnie_hoppe@yahoo.com	2019-11-18	Belgium	3	Architect	DevOps

Exploring the Data (EDA notebook)

> ***Files used*** → *connection.py* and *002_candidatesEDA.ipynb*

To perform a better analysis of the dataset, we parse the ***Application Date*** column as a *datetime* type.

Loading the dataset

We read the dataset from a table that stores our data: this table is located in a PostgreSQL database that is linked to the engine created with SQLAlchemy.

```python
df = pd.read_sql_table("candidates_raw", engine, parse_dates=["Application Date"])
df
```

✓ 0.6s Abrir "df" en Data Wrangler Python

	First Name	Last Name	Email	Application Date	Country	YOE	Seniority	Technology	Code Challenge Score	Technical Interview Score
0	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	3	3
1	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	2	10
2	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success	10	9
3	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual	7	1
4	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management	9	7
...	...	...	...	...	...	...	...	...	...	...
49995	Bethany	Shields	rocky_mitchell@hotmail.com	2022-01-09	Dominican Republic	27	Trainee	Security	2	1
49996	Era	Swaniawski	dolores.roob@hotmail.com	2020-06-02	Morocco	21	Lead	Game Development	1	2
49997	Martin	Lakin	savanah.stracke@gmail.com	2018-12-15	Uganda	20	Trainee	System Administration	6	1
49998	Aliya	Abernathy	vivienne.fritsch@yahoo.com	2020-05-30	Czech Republic	20	Senior	Database Administration	0	0
49999	Coleman	Wisozk	abigayle.crooks@yahoo.com	2022-06-13	Palau	15	Intern	Mulesoft	3	1

50000 rows × 10 columns

Now, we can analyze the Dtype and the count of non-null values of our dataset. We can see that Pandas automatically converts numeric values to **int64**. Also, we see that there are absolutely no null values in our dataset, which simplifies our ETL process a bit.

```
df.info()
[5] ✓ 0.0s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   first_name                 50000 non-null  object
 1   last_name                  50000 non-null  object
 2   email                      50000 non-null  object
 3   application_date           50000 non-null  datetime64[ns]
 4   country                    50000 non-null  object
 5   yoe                        50000 non-null  int64
 6   seniority                  50000 non-null  object
 7   technology                 50000 non-null  object
 8   code_challenge_score       50000 non-null  int64
 9   technical_interview_score  50000 non-null  int64
dtypes: datetime64[ns](1), int64(3), object(6)
memory usage: 3.8+ MB
```

There are 5 main points addressed in the EDA, which will be explained below:

- Are there reapplicants in the dataset? How many are there?
- What happened in 2022? Why the amount of applicants is so small?
- What is the comparison between the hired and the non-hired applicants?
- What were the technologies with which most employees were hired?
- How is the relationship between the seniority and YOE of the hired candidates?

Let’s review the number of reapplicants by their e-mail

Let's consider that the total number of records is 50.000. If each of these were unique, then each email would also be unique, however, we find that there are approximately 167 emails that seem to be repeated. Let's see if this estimate is true.

```
[8]  df.nunique()
✓ 0.0s

first_name                 3007
last_name                   474
email                     49833
application_date           1646
country                     244
yoe                          31
seniority                     7
technology                   24
code_challenge_score         11
technical_interview_score    11
dtype: int64
```

```
duplicated_emails = df.loc[df.duplicated(subset=['email'], keep=False)]
duplicated_emails["email"].value_counts()
✓ 0.0s
email
fern70@gmail.com               3
marianne31@yahoo.com           3
matilda17@gmail.com            2
candelario19@hotmail.com       2
furman49@gmail.com             2
                              ..
jasper81@gmail.com             2
desmond85@yahoo.com            2
bertrand65@hotmail.com         2
hildegard_prohaska@yahoo.com   2
easter75@gmail.com             2
Name: count, Length: 165, dtype: int64
```

As we can see in the image on the right, our calculation misses by a very small margin: 2 emails are repeated 3 times. Thus, we have that **165 emails are repeated 2 or even 3 times**.

What happened in 2022?

The minimum of application_date dates from January 1, 2018; the maximum, July 4, 2022. This brings an affectation in the 2022 cumulative data as we will observe.

```
[10]  df.describe()
✓ 0.0s
```

	application_date	yoe	code_challenge_score	technical_interview_score
count	50000	50000.000000	50000.000000	50000.000000
mean	2020-04-03 23:04:14.592000	15.286980	4.996400	5.003880
min	2018-01-01 00:00:00	0.000000	0.000000	0.000000
25%	2019-02-17 00:00:00	8.000000	2.000000	2.000000
50%	2020-04-06 00:00:00	15.000000	5.000000	5.000000
75%	2021-05-21 00:00:00	23.000000	8.000000	8.000000
max	2022-07-04 00:00:00	30.000000	10.000000	10.000000
std	NaN	8.830652	3.166896	3.165082

For the years spanning 2018 through 2021 there is a steady trend ranging from 11.000 to 11.200. However, in 2022 this cumulative drops considerably to 5.642, given the fact that the maximum column data only reaches July 4, 2022.

We will extract **the year and month** from the application date field. With these data, we will analyze the **frequency of enrollees per year** and the **availability of the data** in these time periods.

```python
df['year'] = df['application_date'].dt.year
df['month'] = df['application_date'].dt.month_name()
```

✓ 0.0s Python

From 2018 to 2021 there is a **steady and persistent** trend of data, however, it is necessary to explore in depth the values of year 2022 given **its low frequency** compared to the other years.

```python
year_counts = (df['year'].value_counts()
               .sort_index())
year_counts
```

✓ 0.0s Abrir "year_counts" en Data Wrangler Python

```
year
2018    11061
2019    11009
2020    11237
2021    11051
2022     5642
Name: count, dtype: int64
```

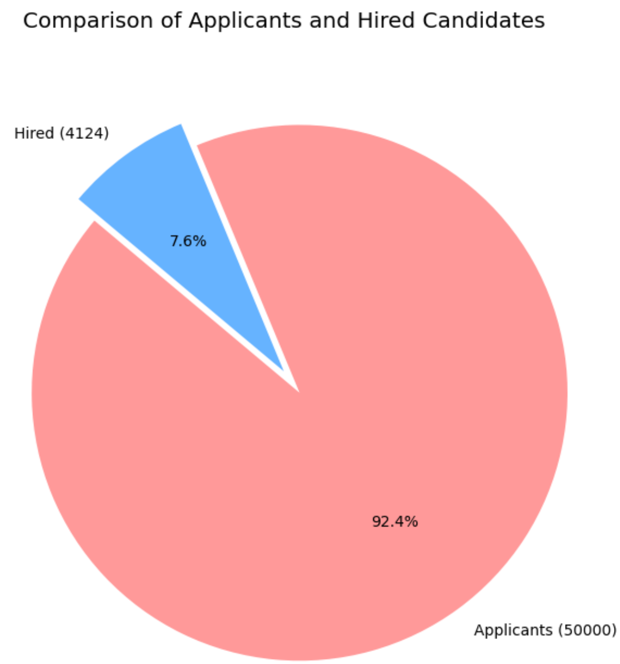
```python
month_order = ["January", "February", "March", "April", "May", "June",
               "July", "August", "September", "October", "November", "December"]

monthly_counts = (df.query("year == 2022")
                  .groupby("month")
                  .size()
                  .reindex(month_order))

monthly_counts
```

✓ 0.0s Abrir "monthly_counts" en Data Wrangler Python

```
month
January      912.0
February     844.0
March        962.0
April        923.0
May          979.0
June         910.0
July         112.0
August         NaN
September      NaN
October        NaN
November       NaN
December       NaN
dtype: float64
```

How many applicants do we hired?

Of the total number of applicants (50.000 candidates), only 7.6% of them (4.124 candidates) were able to meet the necessary scores to be hired.



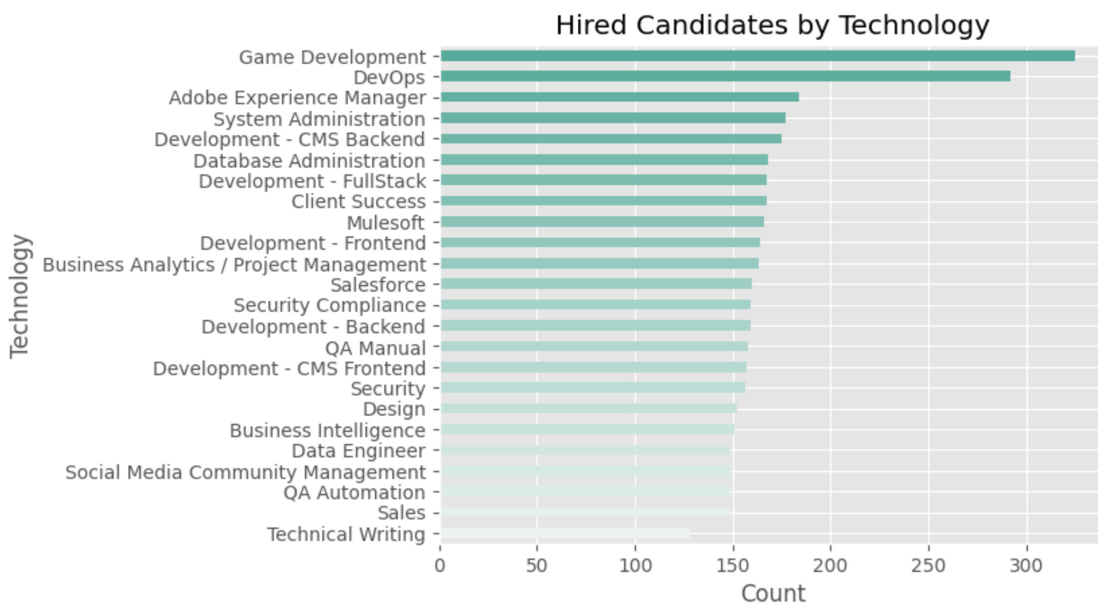
What technologies do these recruited candidates use?

There is an important part of recruits that are in charge of the Game Development and DevOps area, as can be seen in the following graphs, where it is better represented how these two positions are the only ones that exceed the barrier of 250 candidates hired.

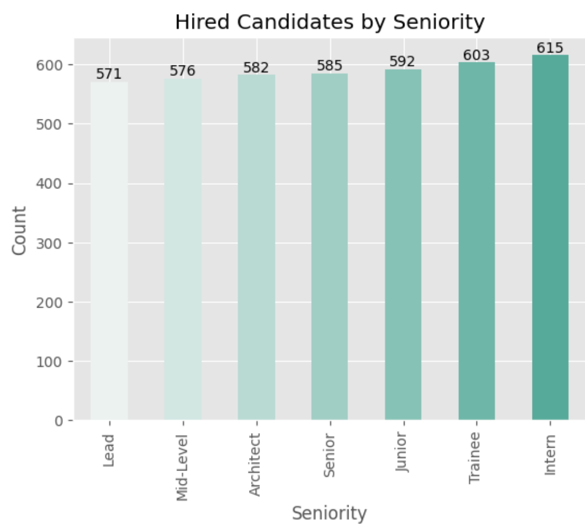
```
df_technology_count = (df_hired.groupby('technology')
                       .size()
                       .sort_values(ascending=False))

df_technology_count
```

✓ 0.0s Abrir "df_technology_count" en Data Wrangler

```
technology
Game Development                             325
DevOps                                       292
Adobe Experience Manager                     184
System Administration                        177
Development - CMS Backend                    175
Database Administration                      168
Development - FullStack                      167
Client Success                               167
Mulesoft                                     166
Development - Frontend                       164
Business Analytics / Project Management      163
Salesforce                                   160
Security Compliance                          159
Development - Backend                        159
QA Manual                                    158
Development - CMS Frontend                   157
Security                                     156
Design                                       152
Business Intelligence                        151
Social Media Community Management            149
QA Automation                                149
Sales                                        149
Data Engineer                                149
Technical Writing                            128
dtype: int64
```



Why is the YOE of interns so high? Why don't they replace the Leader?



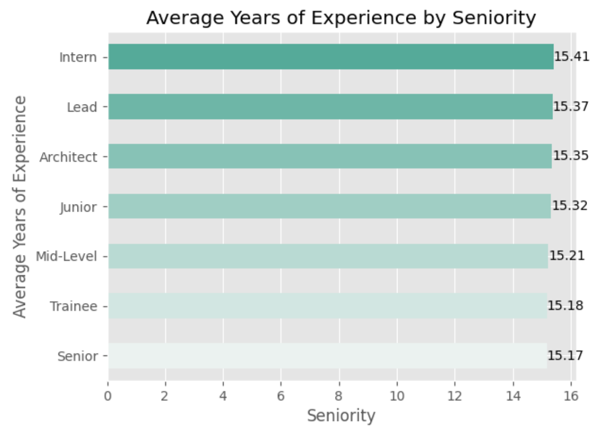
For a brief background, the seniority of a large part of the candidates hired varies between Intern, Trainee and Junior: together they count 1.810 hired candidates.

However, there is a little situation.

```
seniority_avg_yoe = (df.groupby('seniority')['yoe']
                     .mean()
                     .sort_values(ascending=False))
seniority_avg_yoe
✓ 0.0s  Abrir "seniority_avg_yoe" en Data Wrangler

seniority
Intern       15.406892
Lead         15.365578
Architect    15.345105
Junior       15.324930
Mid-Level    15.213291
Trainee      15.178616
Senior       15.174529
Name: yoe, dtype: float64
```

The average years of experience (YOE) of these three seniority roles is 15 years of experience: this amount is similar to more experienced positions such as Leader or Senior.

Keep in mind that these incomprehensible and illogical insights are given by the randomness of the data, but they would be useful relationships to know in case of an exploratory data analysis.



Transforming the raw data and loading the clean one

> ***Files used*** → *connection.py* and *003_cleanDataLoad.ipynb*

Renaming of the columns

For a better experience when interacting with the data, the column names are standardized to a snake_case style.

```
renamed_columns = {
    'First Name': 'first_name',
    'Last Name': 'last_name',
    'Email': 'email',
    'Application Date': 'application_date',
    'Country': 'country',
    'YOE': 'yoe',
    'Seniority': 'seniority',
    'Technology': 'technology',
    'Code Challenge Score': 'code_challenge_score',
    'Technical Interview Score': 'technical_interview_score'
}

df = df.rename(columns=renamed_columns)
df.columns
```

```
Index(['first_name', 'last_name', 'email', 'application_date', 'country',
       'yoe', 'seniority', 'technology', 'code_challenge_score',
       'technical_interview_score'],
      dtype='object')
```

Grouping technologies by categories

For a better visualization of further statistics with the column *"technology"* we group these technologies by broader categories, as can be seen in the image on the left.

```python
df["technology"].unique()
```

```
array(['Data Engineer', 'Client Success', 'QA Manual',
       'Social Media Community Management', 'Adobe Experience Manager',
       'Sales', 'Mulesoft', 'DevOps', 'Development - CMS Backend',
       'Salesforce', 'System Administration', 'Security',
       'Game Development', 'Development - CMS Frontend',
       'Security Compliance', 'Development - Backend', 'Design',
       'Business Analytics / Project Management',
       'Development - Frontend', 'Development - FullStack',
       'Business Intelligence', 'Database Administration',
       'QA Automation', 'Technical Writing'], dtype=object)
```

```python
renamed_technologies = {
    'Development - Backend': 'Development and Programming',
    'Development - Frontend': 'Development and Programming',
    'Development - FullStack': 'Development and Programming',
    'Development - CMS Backend': 'Development and Programming',
    'Development - CMS Frontend': 'Development and Programming',
    'Game Development': 'Development and Programming',

    'Business Analytics / Project Management': 'Analytics and Business Intelligence',
    'Business Intelligence': 'Analytics and Business Intelligence',

    'QA Manual': 'QA and Testing',
    'QA Automation': 'QA and Testing',

    'System Administration': 'Systems Administration and Management',
    'Database Administration': 'Systems Administration and Management',
    'Security': 'Systems Administration and Management',
    'Security Compliance': 'Systems Administration and Management',

    'Client Success': 'Project and Client Management',
    'Sales': 'Project and Client Management',

    'Design': 'Design and Creativity',
    'Social Media Community Management': 'Design and Creativity',

    'DevOps': 'Automation and DevOps',
    'Mulesoft': 'Automation and DevOps',
    'Salesforce': 'Automation and DevOps',
}
```

```python
df['technology'] = df['technology'].replace(renamed_technologies)
```

```python
df["technology"].unique()
```

```
array(['Data Engineer', 'Project and Client Management', 'QA and Testing',
       'Design and Creativity', 'Adobe Experience Manager',
       'Automation and DevOps', 'Development and Programming',
       'Systems Administration and Management',
       'Analytics and Business Intelligence', 'Technical Writing'],
      dtype=object)
```

Creating an "*is_hired*" column

Given the need to determine whether the candidate is hired or not, a column is created that bases its values on a logical statement: if that statement is true for the record, then the value is True; otherwise, it is False.

Creating an "*is_hired*" column

This column contains a boolean value that follows the required logical statement that code_challenge_score and technical_interview_score must be greater than or equal to 7.

```python
df['is_hired'] = (df['code_challenge_score'] >= 7) & (df['technical_interview_score'] >= 7)
df
```

Python

	first_name	last_name	email	application_date	country	yoe	seniority	technology	code_challenge_score	technical_interview_score	is_hired
0	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	3	3	False
1	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	2	10	False
2	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Project and Client Management	10	9	True
3	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA and Testing	7	1	False
4	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Design and Creativity	9	7	True
...	...	...	...	...	...	...	...	...	...	...	...
49995	Bethany	Shields	rocky_mitchell@hotmail.com	2022-01-09	Dominican Republic	27	Trainee	Systems Administration and Management	2	1	False
49996	Era	Swaniawski	dolores.roob@hotmail.com	2020-06-02	Morocco	21	Lead	Development and Programming	1	2	False
49997	Martin	Lakin	savanah.stracke@gmail.com	2018-12-15	Uganda	20	Trainee	Systems Administration and Management	6	1	False
49998	Aliya	Abernathy	vivienne.fritsch@yahoo.com	2020-05-30	Czech Republic	20	Senior	Systems Administration and Management	0	0	False

Loading the clean data to PostgreSQL

Similar to the case of the raw data, we load the transformed data into a table called *candidates_hired*. For the rest of the parameters, the process is similar to the previous one.

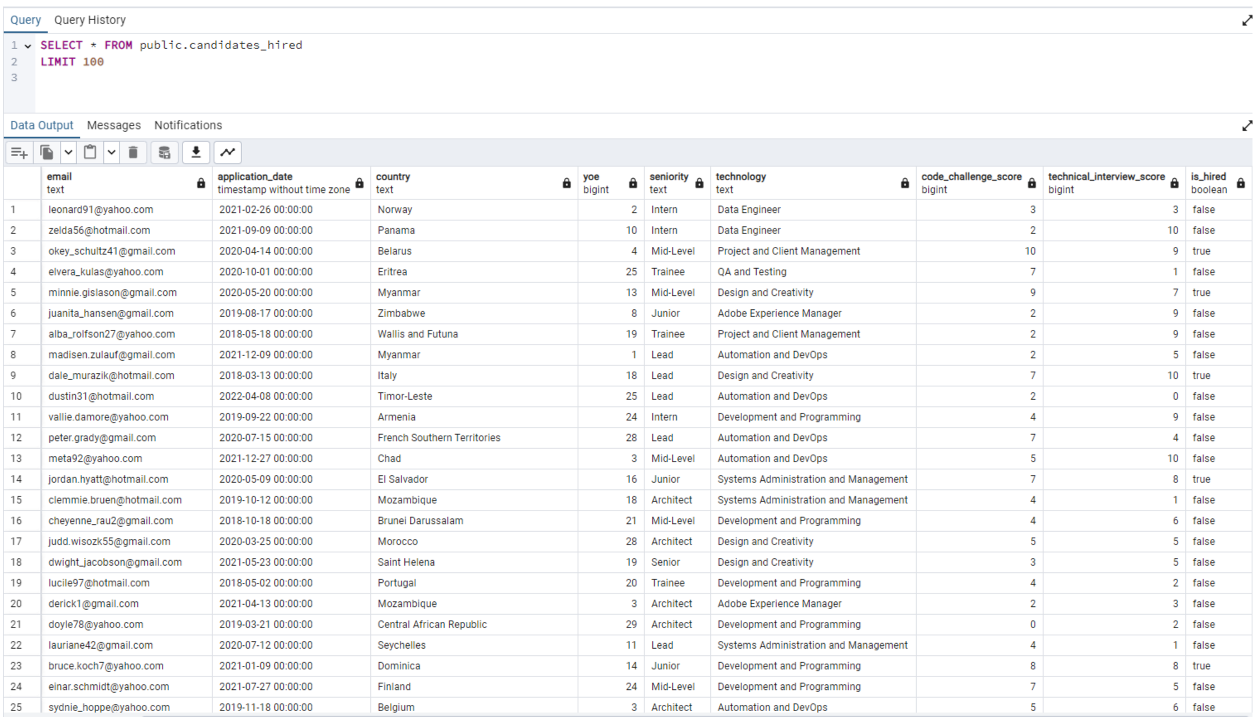
Loading the clean data

The table is created using the engine and the Pandas `to_sql()` function. The data types are shown below the code.

```
df.to_sql('candidates_hired', engine, if_exists='replace', index=False)
```

```
1000
```

We verify again in *pgAdmin* if the data uploaded correctly.



Query Query History

```
SELECT * FROM public.candidates_hired
LIMIT 100
```

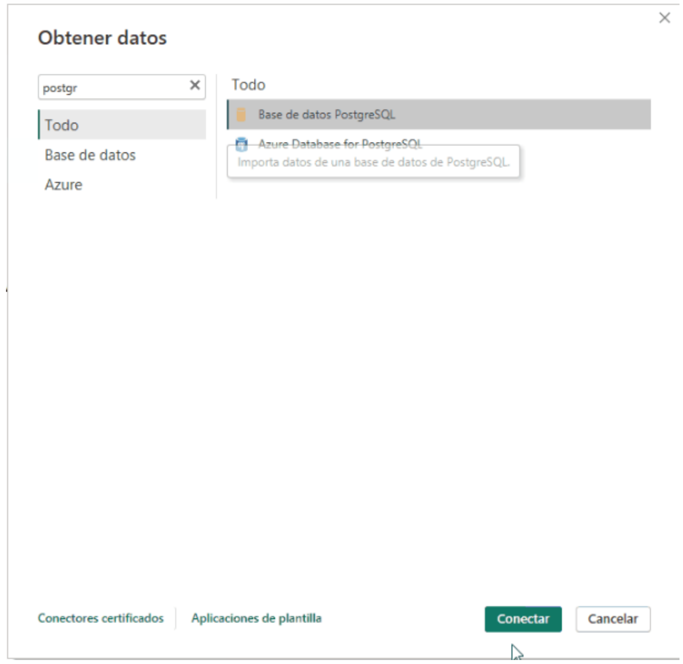
Data Output Messages Notifications

	email text	application_date timestamp without time zone	country text	yoe bigint	seniority text	technology text	code_challenge_score bigint	technical_interview_score bigint	is_hired boolean
1	leonard91@yahoo.com	2021-02-26 00:00:00	Norway	2	Intern	Data Engineer	3	3	false
2	zelda56@hotmail.com	2021-09-09 00:00:00	Panama	10	Intern	Data Engineer	2	10	false
3	okey_schultz41@gmail.com	2020-04-14 00:00:00	Belarus	4	Mid-Level	Project and Client Management	10	9	true
4	elvera_kulas@yahoo.com	2020-10-01 00:00:00	Eritrea	25	Trainee	QA and Testing	7	1	false
5	minnie.gislason@gmail.com	2020-05-20 00:00:00	Myanmar	13	Mid-Level	Design and Creativity	9	7	true
6	juanita_hansen@gmail.com	2019-08-17 00:00:00	Zimbabwe	8	Junior	Adobe Experience Manager	2	9	false
7	alba_rolfson27@yahoo.com	2018-05-18 00:00:00	Wallis and Futuna	19	Trainee	Project and Client Management	2	9	false
8	madisen.zulauf@gmail.com	2021-12-09 00:00:00	Myanmar	1	Lead	Automation and DevOps	2	5	false
9	dale_murazik@hotmail.com	2018-03-13 00:00:00	Italy	18	Lead	Design and Creativity	7	10	true
10	dustin31@hotmail.com	2022-04-08 00:00:00	Timor-Leste	25	Lead	Automation and DevOps	2	0	false
11	vallie.damore@yahoo.com	2019-09-22 00:00:00	Armenia	24	Intern	Development and Programming	4	9	false
12	peter.grady@gmail.com	2020-07-15 00:00:00	French Southern Territories	28	Lead	Automation and DevOps	7	4	false
13	meta92@yahoo.com	2021-12-27 00:00:00	Chad	3	Mid-Level	Automation and DevOps	5	10	false
14	jordan.hyatt@hotmail.com	2020-05-09 00:00:00	El Salvador	16	Junior	Systems Administration and Management	7	8	true
15	clemmie.bruen@hotmail.com	2019-10-12 00:00:00	Mozambique	18	Architect	Systems Administration and Management	4	1	false
16	cheyenne_rau2@gmail.com	2018-10-18 00:00:00	Brunei Darussalam	21	Mid-Level	Development and Programming	4	6	false
17	judd.wisozk55@gmail.com	2020-03-25 00:00:00	Morocco	28	Architect	Design and Creativity	5	5	false
18	dwight_jacobson@gmail.com	2021-05-23 00:00:00	Saint Helena	19	Senior	Design and Creativity	3	5	false
19	lucile97@hotmail.com	2018-05-02 00:00:00	Portugal	20	Trainee	Development and Programming	4	2	false
20	derick1@gmail.com	2021-04-13 00:00:00	Mozambique	3	Architect	Adobe Experience Manager	2	3	false
21	doyle78@yahoo.com	2019-03-21 00:00:00	Central African Republic	29	Architect	Development and Programming	0	2	false
22	lauriane42@gmail.com	2020-07-12 00:00:00	Seychelles	11	Lead	Systems Administration and Management	4	1	false
23	bruce.koch7@yahoo.com	2021-01-09 00:00:00	Dominica	14	Junior	Development and Programming	8	8	true
24	einar.schmidt@yahoo.com	2021-07-27 00:00:00	Finland	24	Mid-Level	Development and Programming	7	5	false
25	sydnie_hoppe@yahoo.com	2019-11-18 00:00:00	Belgium	3	Architect	Automation and DevOps	5	6	false

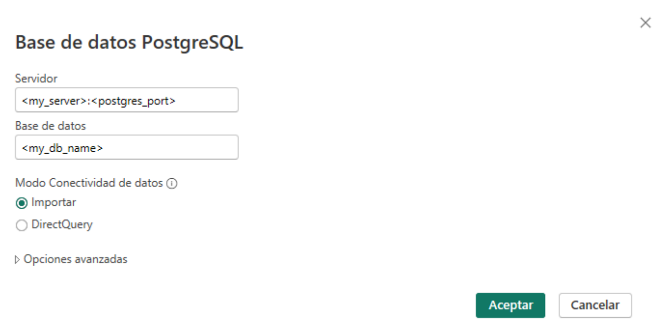
Visualizing the Data

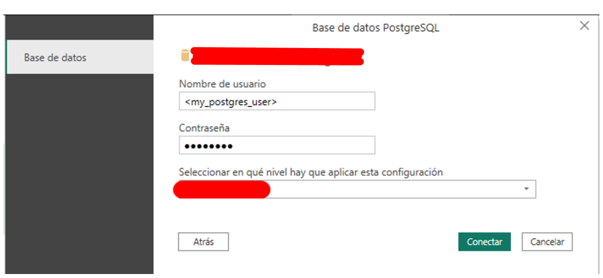
Connecting the database to Power BI

1. Open Power BI Desktop and create a new dashboard. Select the *Get data* option - be sure you choose the "PostgreSQL Database" option.



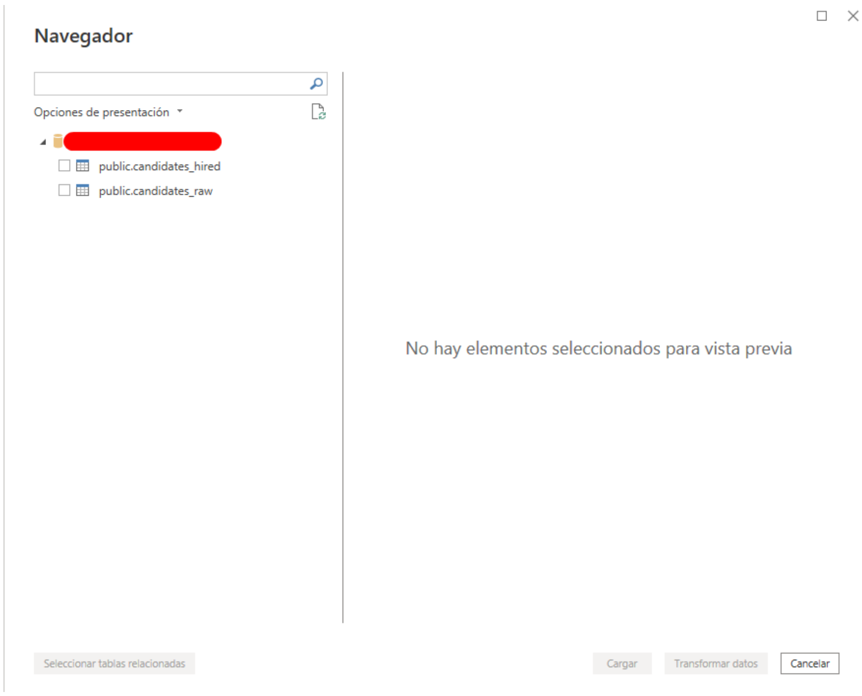
2. Insert the PostgreSQL Server and Database Name.



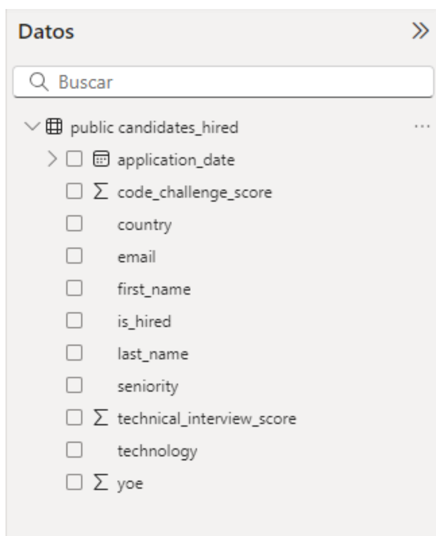


3. Fill in the following fields with your credentials.

4. If you manage to connect to the database the following tables will appear:



5. Choose the candidates_hired table and start making your own visualizations!



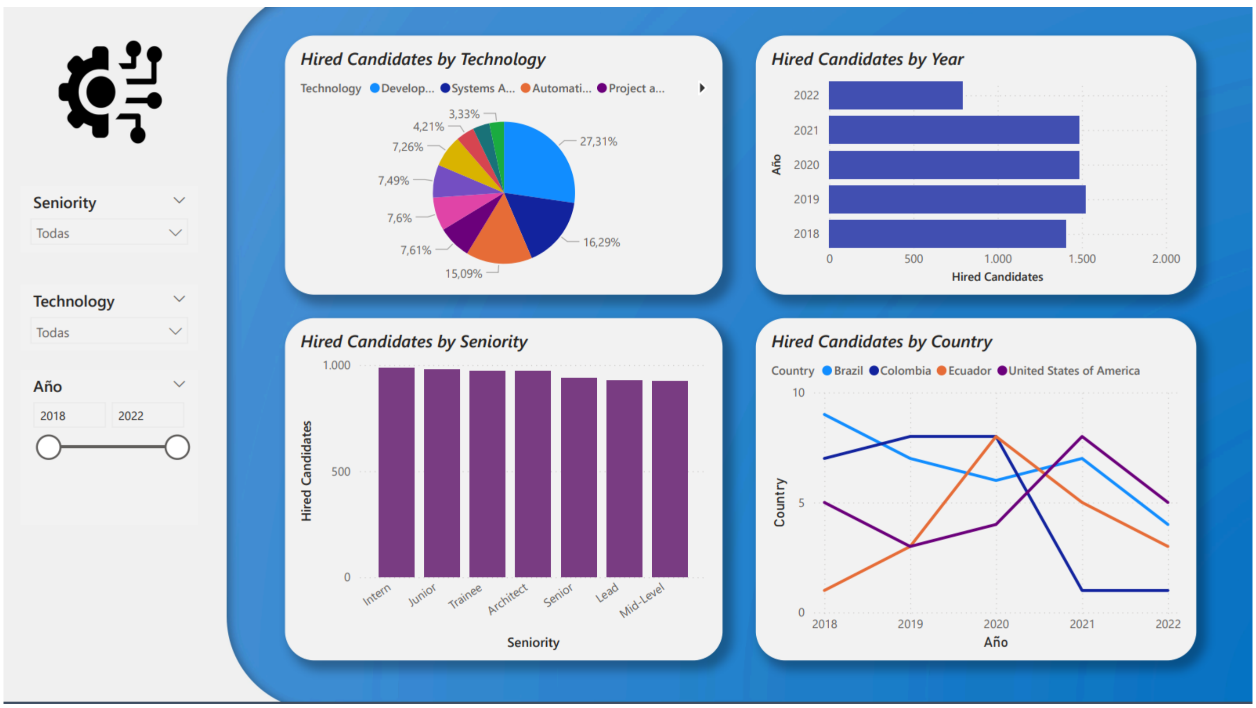
Dashboard Results

The following details about the dashboard can be highlighted:

- Filters help to dynamically segment data tailored to whoever visits the dashboard.
- The pie chart (*Hired Candidates by Technology*) applies the grouping of categories made in the previous transformation, resulting in a better distribution of information on a

visual level.

- The vertical line chart (*Hired Candidates by Seniority*) shows a distribution where the quantity or magnitude of each role is highlighted.
- The horizontal line chart (*Hired Candidates by Year*) shows a constant evolution as each year passes. This evolution, it should be noted, is interrupted in 2022 for the reasons mentioned above in .
- The multi-line graph is effective in showing the distribution of candidates hired by selected countries. It should be noted that a selection of certain countries had to be made from the hundreds of countries included in the dataset in order to make the graph look cleaner, but still provide valuable insights.



Conclusions

1. In this workshop we started to approaching to the basics of data engineering, from raw data ingestion to insightful analysis. The use of tools like Python, PostgreSQL, and Power BI showcased the importance of a well-structured ETL process in deriving meaningful insights from large datasets.
2. The project highlighted the critical role of data cleaning and transformation in preparing data for analysis. By standardizing column names, grouping technologies, and creating derived columns like "*is_hired*", the dataset became more manageable and informative.

3. The EDA process revealed several interesting findings, such as the presence of reapplicants, anomalies in hiring patterns, and unexpected relationships between seniority and years of experience. These insights underscore the value of thorough data exploration in uncovering hidden patterns and potential data quality issues.
6. The use of various chart types (pie charts, bar charts, multiline charts) in the final Power BI report emphasized the power of visual representation in communicating complex data patterns and trends effectively.