Workshop #1 - Documentation

Luz Ángela Carabalí Mulato - 2230652

About the workshop

This workshop involves analyzing randomly generated data of candidates participating in a selection process. The dataset (candidates.csv) contains 50,000 rows and 10 columns. It undergoes various loading, cleaning, and transformation processes to extract meaningful insights 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
- Seaborn
- Numpy
- Scikit-learn
- Sqlalchemy
- Psycopg2
- Psycopg2-binary
- Jupyter

These libraries are included in the Poetry project configuration file (pyproject.toml), making dependency management easier.

Project Goals

The objective is to obtain a clean dataset to create an analytical report using BI tools like Power BI. The transformed dataset is stored in a PostgreSQL database and visualized through various graphical representations:

- **Hires by Technology** (Pie Chart)
- **Hires by Year** (Horizontal Bar Chart)
- **Hires by Seniority** (Bar Chart)
- **Hires by Country Over the Years** (Multiline Chart for USA, Brazil, Colombia, and Ecuador)

Process Overview

1. Setting Up the Environment

- Installed dependencies using Poetry.
- Configured a virtual environment.

installation poetry

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

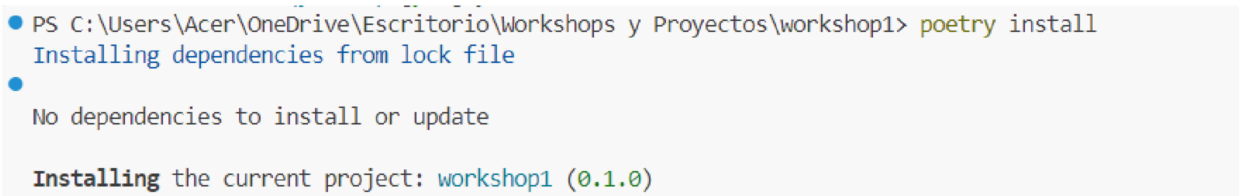
Using Poetry for Dependency Management and Virtual Environments

Poetry registers these libraries in the **pyproject.toml** configuration file.

If `pyproject.toml` is already in our directory but we are working in a **different virtual environment**, we can use the command:

```
poetry install
```

This will **create a new virtual environment** and install the dependencies specified in the Poetry configuration file, as shown in the following image:



Terminal Output:

(Shows the installation of dependencies and project initialization)

2. Poetry Lock Files

After installing dependencies, two important files are created:

- **pyproject.toml** – Stores project dependencies and settings.
- **poetry.lock** – Ensures consistent dependency versions across environments.

The following image illustrates these files in the project directory:

Project Files:

(Shows `pyproject.toml` and `poetry.lock` in the file explorer)

3. Running Jupyter Notebook with Poetry

Once dependencies are installed, we can run Jupyter Notebook inside our Poetry-managed virtual environment with:

```
poetry run jupyter notebook
```

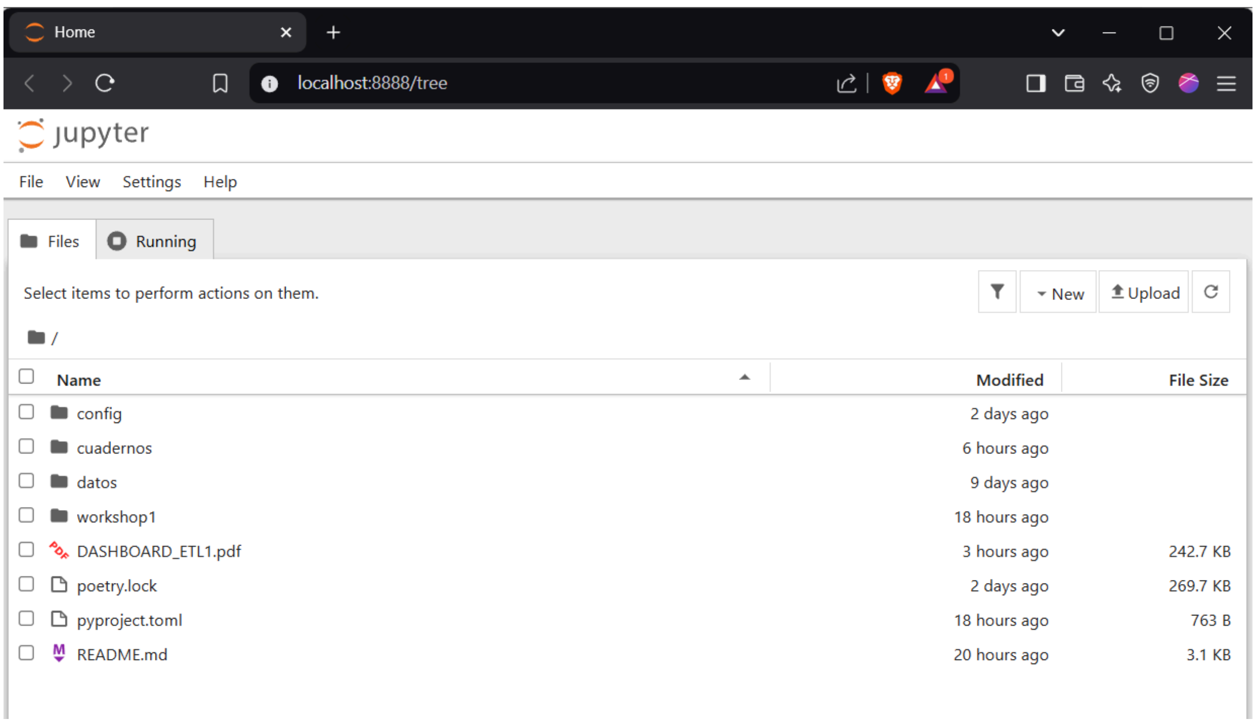
The following output confirms that Jupyter Notebook and its extensions have been successfully loaded:

Terminal Output:

(Displays the successful launch of Jupyter Notebook using Poetry)

This setup ensures that all dependencies remain well-organized and isolated within the Poetry environment, making project management more efficient.

```
PS C:\Users\Acer\OneDrive\Escritorio\Workshops y Proyectos\workshop1> poetry run jupyter notebook
>>
[I 2025-02-28 20:27:52.348 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2025-02-28 20:27:52.354 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2025-02-28 20:27:52.371 ServerApp] jupyterlab | extension was successfully linked.
[I 2025-02-28 20:27:52.384 ServerApp] notebook | extension was successfully linked.
[I 2025-02-28 20:27:52.854 ServerApp] notebook_shim | extension was successfully linked.
[I 2025-02-28 20:27:53.138 ServerApp] notebook_shim | extension was successfully loaded.
[I 2025-02-28 20:27:53.145 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2025-02-28 20:27:53.145 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2025-02-28 20:27:53.152 LabApp] JupyterLab extension loaded from C:\Users\Acer\OneDrive\Escritorio\Work
shops y Proyectos\workshop1\.venv\Lib\site-packages\jupyterlab
```



Establishing the connection with the DB and loading the raw data

1. Project Structure and Required Files

The required files for database connection and data extraction are stored in the **config** directory:

- `conexion_db.py` – Handles the database connection.
- `001_extract.ipynb` – Loads and processes raw data.

2. Creating the Connection Engine (`conexion_db.py`)

The **connection engine** is created in the `conexion_db.py` module using the **SQLAlchemy** library with the **psycopg2** driver. To establish the connection, we use the `create_engine()` function, which requires a **database URL** containing credentials stored in an **.env** file.

Code to Load Environment Variables and Create the Connection:

```
import os
from sqlalchemy import create_engine
from dotenv import load_dotenv

# Cargar variables de entorno desde .env
load_dotenv("../env/.env")

# Obtener las credenciales de la base de datos
DB_USER = os.getenv("DB_USER")
DB_PASSWORD = os.getenv("DB_PASSWORD")
DB_HOST = os.getenv("DB_HOST")
DB_PORT = os.getenv("DB_PORT")
DB_NAME = os.getenv("DB_NAME")

# Construir la URL de conexión
DATABASE_URL = f"postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"

def get_engine():
    """Devuelve un objeto SQLAlchemy Engine para la conexión a la base de datos."""
    return create_engine(DATABASE_URL)
```

3. Loading and Processing Raw Data (`001_extract.ipynb`)

Once the connection is established, we use **Pandas** to process a CSV file and perform database operations through the engine.

Steps to Load Data:

1. Import the necessary libraries.
2. Read the dataset using `pandas.read_csv()`.
3. Use `to_sql()` to upload the data to PostgreSQL.
4. The data is stored in a table named **"candidates"**.

Setting the environment

```
[1]: import sys
     sys.path.append('../config')
```

Importing libraries and modules ¶

```
[9]: from sqlalchemy import text
     from conexion_db import get_engine
     import pandas as pd
     from sqlalchemy import Column, Integer, String,Date,inspect
     from sqlalchemy.orm import declarative_base
```

Reading the dataset

```
[ ]: df = pd.read_csv(r"C:/Users/Acer/OneDrive/Escritorio/Workshops y Proyectos/workshop1/datos/candidates.csv", sep=";")
```

Connection with PostgreSQL

```
engine = get_engine()

try:
    with engine.connect() as connection:
        print("✅ Conexión exitosa a PostgreSQL")
except Exception as e:
    print("❌ Error de conexión:", e)
```

```
✅ Conexión exitosa a PostgreSQL
```

Table creation

```
Base = declarative_base()

class Candidates(Base):
    __tablename__ = 'candidates'
    id = Column(Integer, primary_key=True, autoincrement=True)
    first_name = Column(String(50))
    last_name = Column(String(50))
    email = Column(String(100))
    application_date = Column(Date)
    country = Column(String(200))
    yoe = Column(Integer)
    seniority = Column(String(200))
    technology = Column(String(200))
    code_challenge_score = Column(Integer)
    technical_interview_score = Column(Integer)

try:
    with engine.connect() as connection:
        print("✅ Conexión exitosa a PostgreSQL")
except Exception as e:
    print("❌ Error de conexión:", e)


inspector = inspect(engine)
print("Tablas en la base de datos:", inspector.get_table_names())
```

```
✅ Conexión exitosa a PostgreSQL
Tablas en la base de datos: ['candidates']
```

```
Tablas en la base de datos: ['candidates']
```

Transfering the data to the database in PostgreSQL

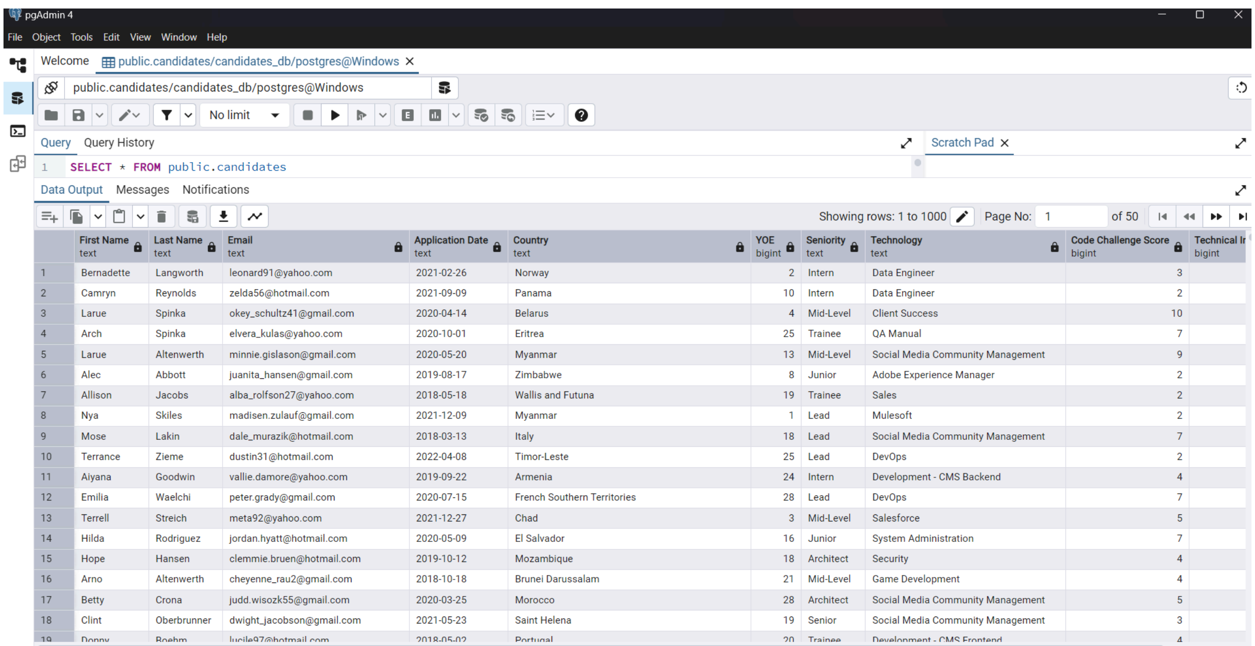
```
[ ]: df.to_sql("candidates", engine, if_exists="replace", index=False)

print("✅ Datos insertados en PostgreSQL correctamente")
```

```
✅ Datos insertados en PostgreSQL correctamente
```

This structured approach ensures a **clean, reproducible, and efficient** process for database management and data extraction.

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	Code Challenge Score bigint	Technical Ir bigint
1	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	3	
2	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	2	
3	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success	10	
4	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual	7	
5	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management	9	
6	Alec	Abbott	juanita_hansen@gmail.com	2019-08-17	Zimbabwe	8	Junior	Adobe Experience Manager	2	
7	Allison	Jacobs	alba_rolfson27@yahoo.com	2018-05-18	Wallis and Futuna	19	Trainee	Sales	2	
8	Nya	Skiles	madisen.zulauf@gmail.com	2021-12-09	Myanmar	1	Lead	Mulesoft	2	
9	Mose	Lakin	dale_murazik@hotmail.com	2018-03-13	Italy	18	Lead	Social Media Community Management	7	
10	Terrance	Zieme	dustin31@hotmail.com	2022-04-08	Timor-Leste	25	Lead	DevOps	2	
11	Aiyana	Goodwin	vallie.damore@yahoo.com	2019-09-22	Armenia	24	Intern	Development - CMS Backend	4	
12	Emilia	Waelchi	peter.grady@gmail.com	2020-07-15	French Southern Territories	28	Lead	DevOps	7	
13	Terrell	Streich	meta92@yahoo.com	2021-12-27	Chad	3	Mid-Level	Salesforce	5	
14	Hilda	Rodriguez	jordan.hyatt@hotmail.com	2020-05-09	El Salvador	16	Junior	System Administration	7	
15	Hope	Hansen	clemmie.bruen@hotmail.com	2019-10-12	Mozambique	18	Architect	Security	4	
16	Arno	Altenwerth	cheyenne_rau2@gmail.com	2018-10-18	Brunei Darussalam	21	Mid-Level	Game Development	4	
17	Betty	Crona	judd.wisozk55@gmail.com	2020-03-25	Morocco	28	Architect	Social Media Community Management	5	
18	Clint	Oberbrunner	dwight_jacobson@gmail.com	2021-05-23	Saint Helena	19	Senior	Social Media Community Management	3	
19	Donny	Boehm	lucile97@hotmail.com	2018-05-02	Portugal	20	Trainee	Development - CMS Frontend	4	

Exploring the Data (EDA notebook)

1. Files Used

The files required for this exploratory data analysis (EDA) process are:

- `conexion_db.py` – Handles database connection.
- `002_candidatosEDA.ipynb` – Contains the data exploration steps.

2. Loading Data from PostgreSQL

To perform a thorough analysis of the dataset, we first extract the data from **PostgreSQL** using an SQL query and load it into a **Pandas DataFrame**. This step ensures that all retrieved data meets the expected criteria and does not contain duplicates.

Read data from PostgreSQL ¶

We ran a SQL query to fetch information from the candidates table and loaded it into a pandas DataFrame for further analysis. To ensure data quality, we checked that the extracted information met the expected criteria and didn't contain any duplicates.

```
[4]: df = pd.read_sql("SELECT * FROM candidates", engine)
df
```

	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

3. Data Type Conversion

To ensure consistency in our analysis, we perform type conversions where necessary. In this case, the **Application Date** column is converted into a **datetime** format for accurate time-based analysis.

```
[6]: df["application_date"] = pd.to_datetime(df["application_date"], errors="coerce")
df
```

	first_name	last_name	email	application_date	country	yoe	seniority	technology	code_challeng
0	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	
1	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	
2	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success	
3	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual	
4	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management	
...	...	...	...	...	...	...	...	...	
49995	Bethany	Shields	rocky_mitchell@hotmail.com	2022-01-09	Dominican Republic	27	Trainee	Security	

4. Dataset Overview and Data Quality Check

Before proceeding with further analysis, we inspect the structure of the dataset using `df.info()`. This helps us understand:

- The number of non-null values per column.
- Automatic conversion of numeric values to `int64`.

```
[7]: df.info()

<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
```

- Confirmation that there are no missing values, simplifying the ETL process.

Applicants & Reapplications

Topics Covered:

- Applicants & Reapplications
- Hiring Criteria & Candidate Classification
- Technology & Experience Analysis
- Clean Data Storage

Analyzing Reapplications by Email

1. Dataset Overview

Our dataset consists of **50,000 records**. Ideally, if each record represented a unique applicant, we would expect **50,000 unique email addresses**.

```
valores_unicos = df.nunique()
valores_unicos
```

```
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
```

📌 However, after analyzing the dataset, we find that the number of unique emails is **49,833**, indicating that some applicants have submitted multiple applications.

2. Identifying Reapplicants

By computing the difference between total records and unique emails, we estimate that approximately **167 applicants** have reapplied using the same email address.

```
Total number of records: 50000
Number of unique emails: 49833
Estimated number of reapplicants: 167
email
marianne31@yahoo.com        3
fern70@gmail.com            3
sandra83@gmail.com          2
dewayne50@gmail.com         2
matilda17@gmail.com         2
                           ..
marjolaine91@hotmail.com    2
jazmin54@gmail.com          2
reyna2@hotmail.com          2
kasandra68@hotmail.com      2
easter75@gmail.com          2
Name: count, Length: 165, dtype: int64
```

What happened in 2022?

To obtain an overview, we used `df.describe()`, which provides central tendency and dispersion values for numerical fields.

For the years from 2018 to 2021, there is a steady trend ranging between 11,000 and 11,200. However, in 2022, this total drops significantly to 5,642, considering that the maximum recorded data only extends until July 4, 2022.

```
[11]: df.describe()
```

[11]:

	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

Extracting Year and Month from Application Date

To identify trends over time, we extracted the year and month from the `application_date` column.

📊 Evolution of Applications by Year

We extracted the year and month from the `application_date` column to facilitate the analysis:

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

To analyze the number of applications submitted per year, we performed the following steps:

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

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

Observations:

- From 2018 to 2021, applications remained stable between **11,000 and 11,800 per year**.
- In **2022**, applications dropped significantly (**5,642 records**), likely because the dataset only includes data up to **July 2022**.
- It would be beneficial to obtain data from 2023 to complete the analysis.

Monthly Application Analysis for 2022

Monthly Application Count (2022)

To analyze the number of applications submitted per month in 2022, we performed the following steps:

```
[14]: 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, fill_value=0))

      monthly_counts
```

```
[14]: month
      January      912
      February     844
      March        962
      April        923
      May          979
      June         910
      July         112
      August         0
      September      0
      October        0
      November       0
      December       0
      dtype: int64
```

Key Insights:

- Application activity was consistent from **January to June**, with between

844 and 979 applications per month.

- In **July, applications dropped sharply to 112** , and from **August to December, there are no records** (probably due to a dataset cutoff).

Analysis of Technical Interview Scores vs. Years of Experience (YoE)

The table shows the distribution of technical interview scores across different years of experience (`yoe`). The goal is to analyze how experience correlates with interview performance and identify notable trends.

```
score_counts = df[['yoe', 'technical_interview_score']].value_counts().reset_index(name='count')
score_counts = score_counts.sort_values(by='technical_interview_score')
score_counts
```

	yoe	technical_interview_score	count
18	7	0	167
21	16	0	166
25	3	0	165
308	22	0	132
298	10	0	134
...	...	...	...
53	16	10	161
229	28	10	143
307	8	10	132
319	24	10	129
0	7	10	178

341 rows × 3 columns

Graphical Analysis of the Dataset

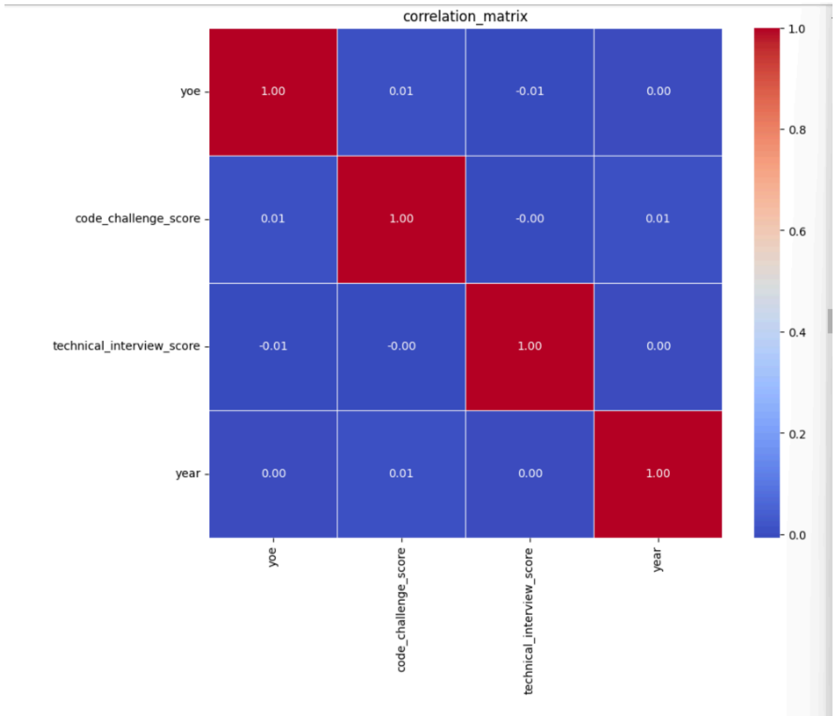
Objective

This section presents a graphical analysis of the dataset, using correlation matrices and bar charts to identify key insights regarding experience, interview performance, and technology trends among candidates.

📊 Correlation Matrix Analysis

- Companies should not assume that experienced candidates will automatically perform better in technical evaluations.

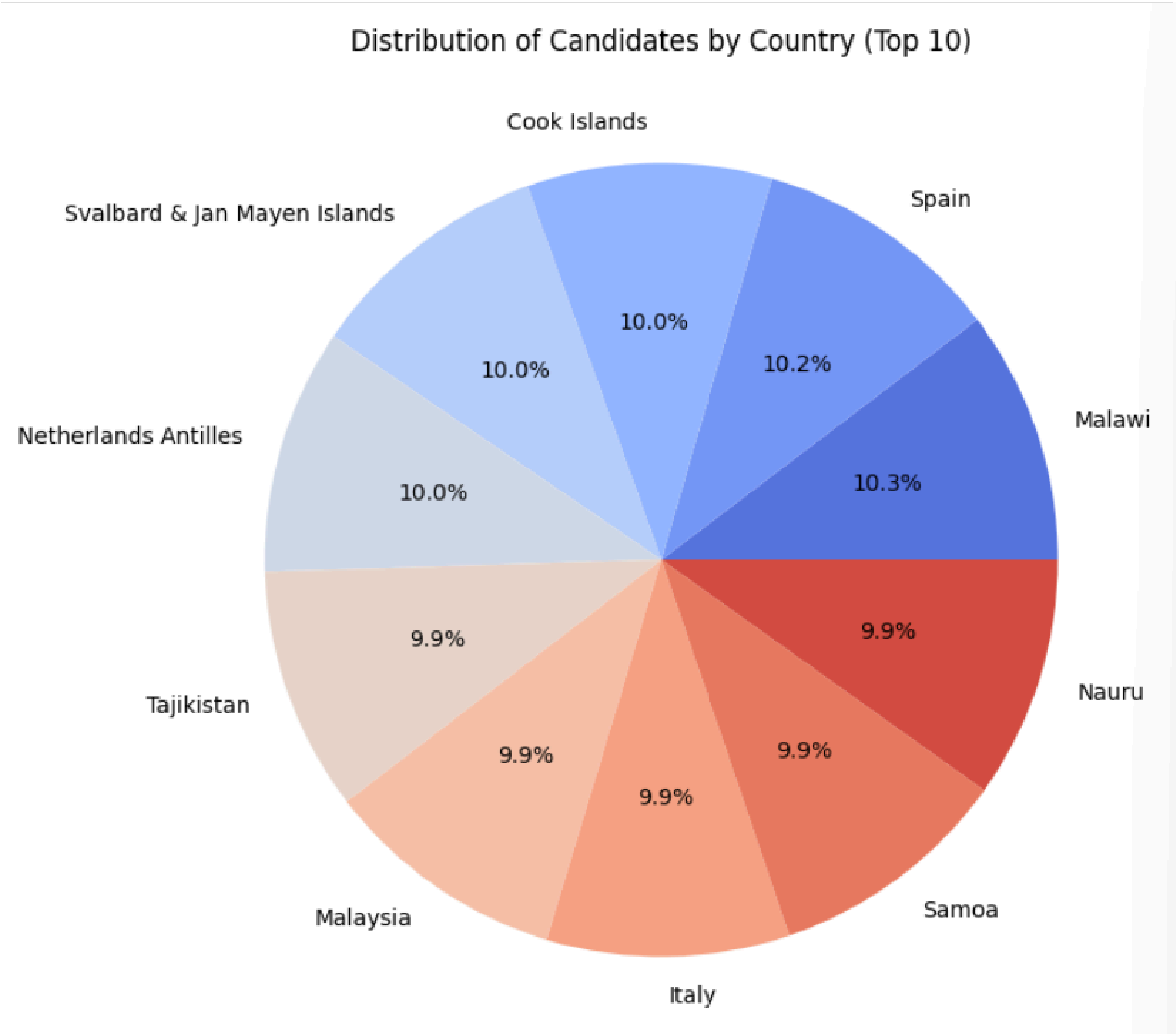
- Interview processes might need to balance assessments between skills-based testing and experience-based evaluation.

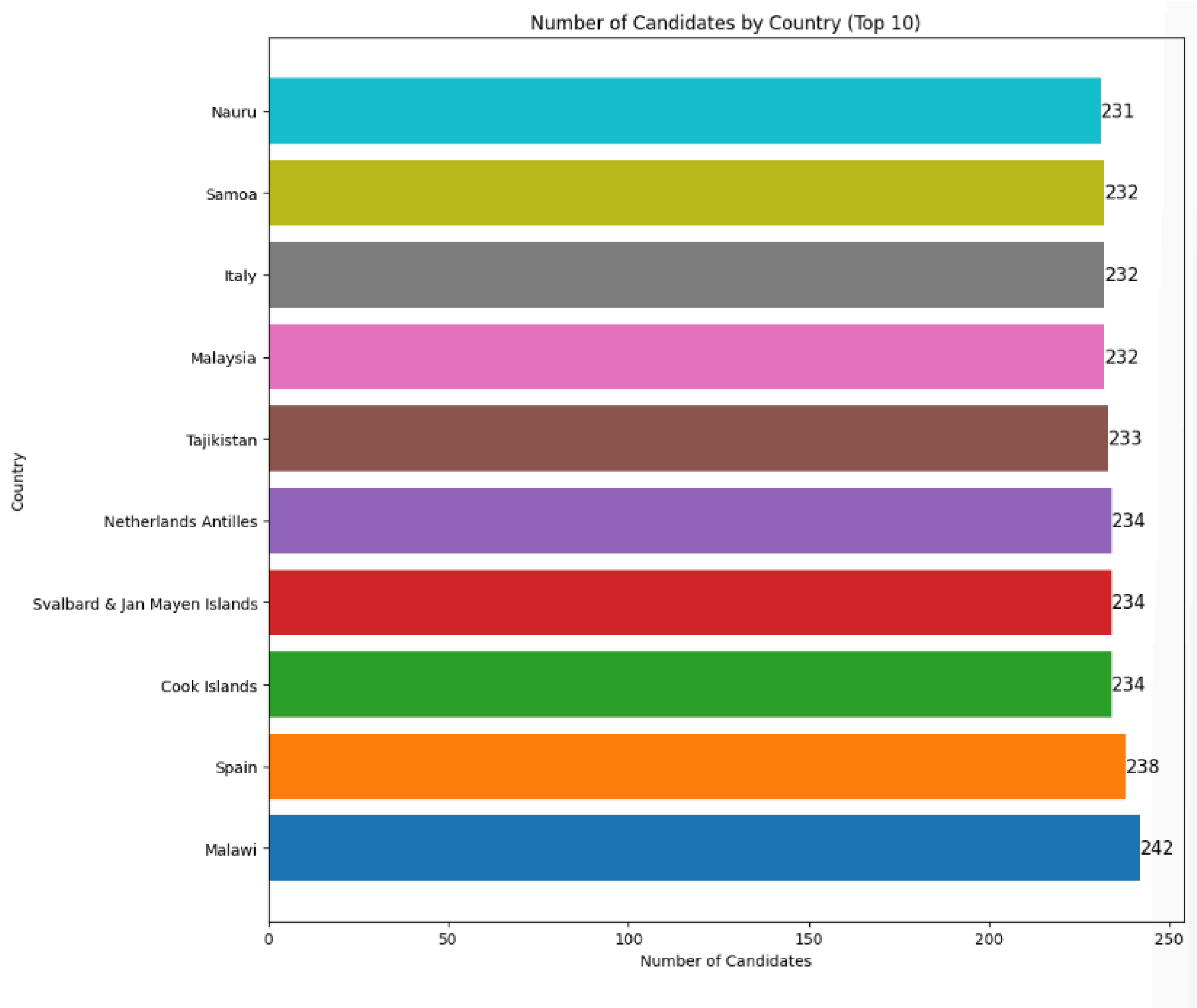


Candidate Distribution by Country

Understanding the geographical distribution of candidates is crucial for assessing whether recruitment is concentrated in specific regions or if it follows a more global trend.

This analysis provides valuable insights into the global nature of recruitment, confirming that hiring efforts are not overly concentrated in a single country. The even spread of candidates suggests a well-balanced recruitment strategy, promoting international diversity in hiring processes.


Distribution of Candidates by Country (Top 10)
Cook Islands
Spain
Svalbard & Jan Mayen Islands
Malawi
Netherlands Antilles
Nauru
Tajikistan
Samoa
Malaysia
Italy
10.0%
10.2%
10.0%
10.3%
10.0%
9.9%
9.9%
9.9%
9.9%
9.9%



Hiring Status Analysis: Hired vs. Not Hired Candidates

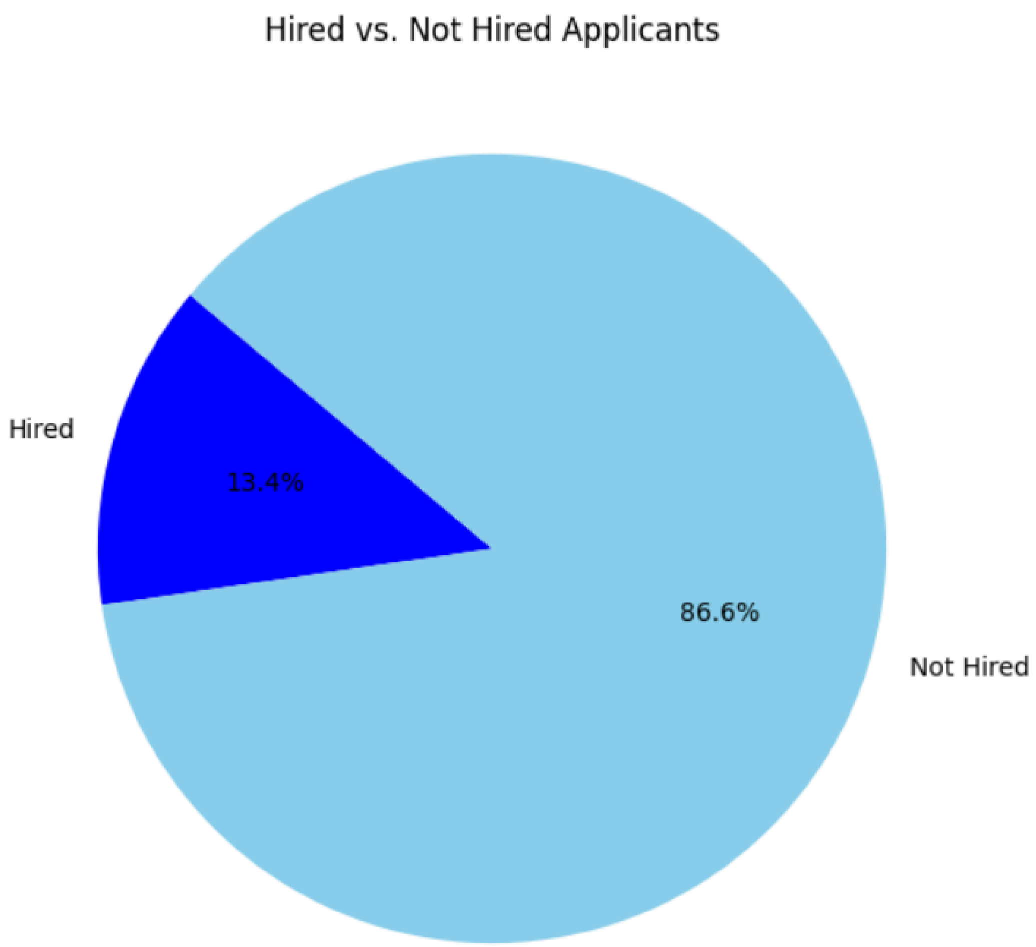
Overview

To efficiently classify candidates based on their hiring status, a new column **"is_hired"** was introduced. This column is determined based on logical conditions where:

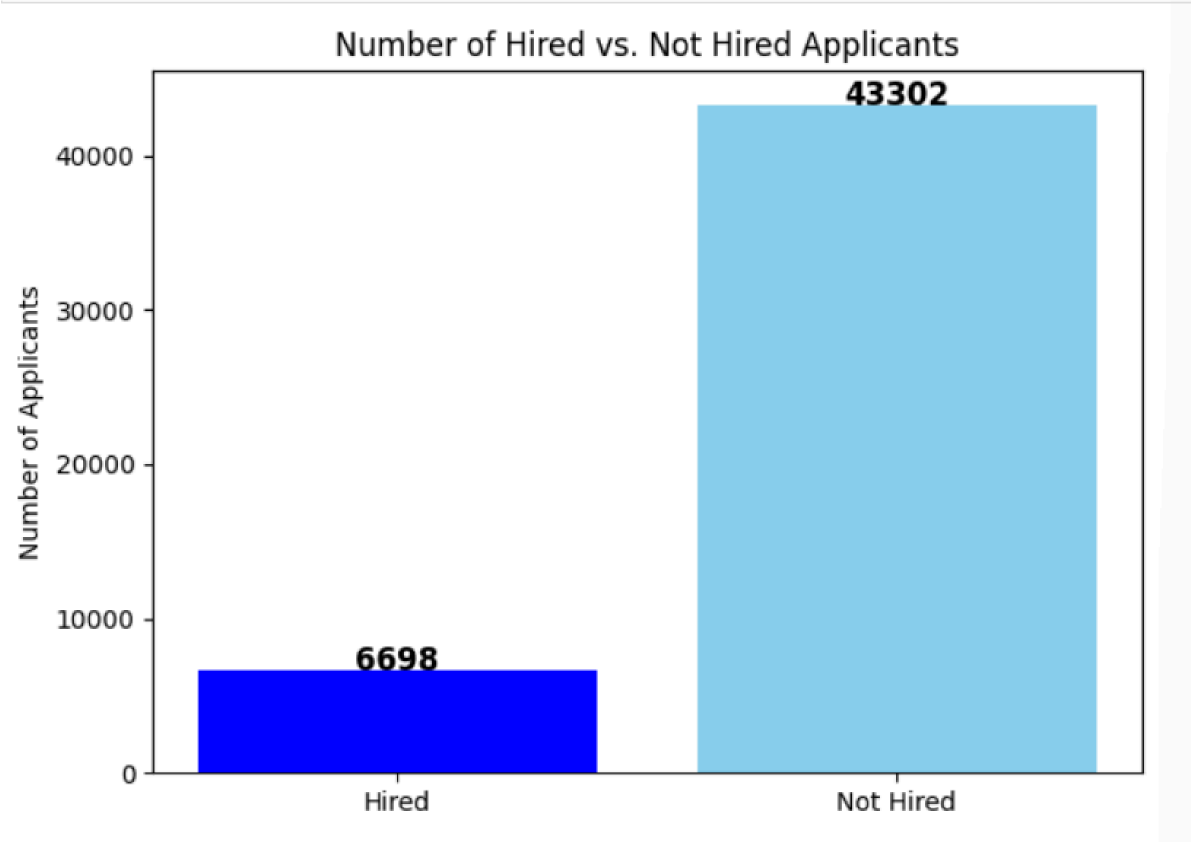
- **True** → The candidate was hired.
- **False** → The candidate was not hired.

t_name	email	application_date	country	yoe	seniority	technology	code_challenge_score	technical_interview_score	year	month	is_hired
gworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	3	3	2021	February	False
eynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	2	10	2021	September	False
Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success	10	9	2020	April	True
Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual	7	1	2020	October	False
nwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management	9	7	2020	May	True
...	...	...	...	...	...	...	...	...	...	...	...
Shields	rocky_mitchell@hotmail.com	2022-01-09	Dominican Republic	27	Trainee	Security	2	1	2022	January	False
niawski	dolores.roob@hotmail.com	2020-06-02	Morocco	21	Lead	Game Development	1	2	2020	June	False
Lakin	savanah.stracke@gmail.com	2018-12-15	Uganda	20	Trainee	System Administration	6	1	2018	December	False
ernathy	vivienne.fritsch@yahoo.com	2020-05-30	Czech Republic	20	Senior	Database Administration	0	0	2020	May	False
Wisozk	abigayle.crooks@yahoo.com	2022-06-13	Palau	15	Intern	Mulesoft	3	1	2022	June	False

Out of all candidates, 86.6% were not hired, while only 13.4% successfully secured a position.



With only
13.4% of applicants being hired, the selection process appears to be highly competitive.



The hiring process is highly selective, with a low percentage of hired candidates compared to applicants. Future recruitment strategies should focus on efficiency—either by improving applicant screening, increasing job availability, or refining hiring criteria to balance the number of hires with the available talent pool.

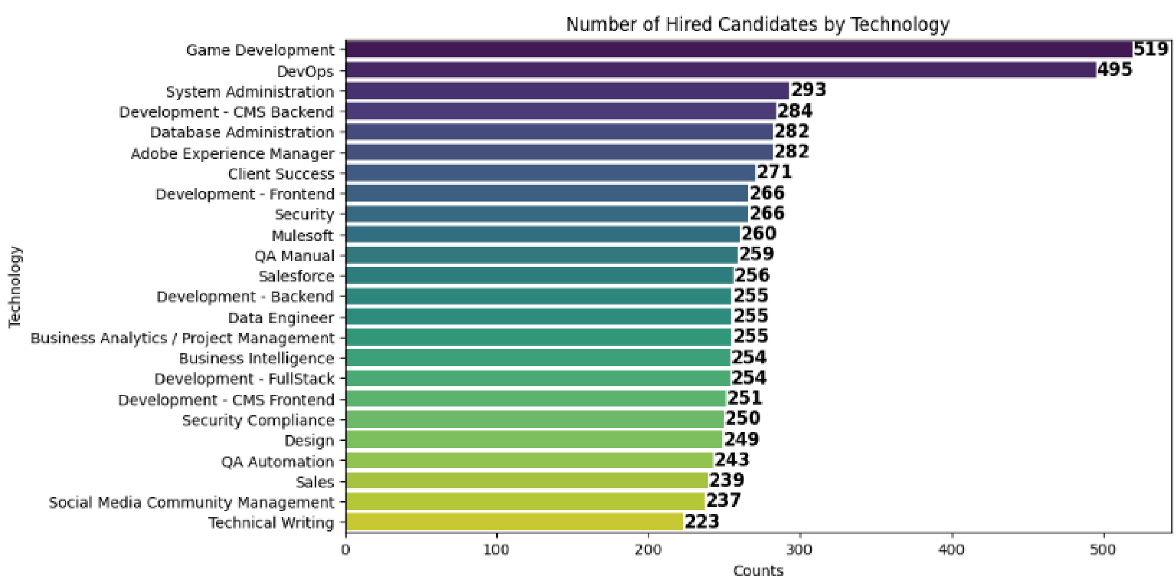
Technology Trends Among Candidates (Bar Chart Analysis)

Overview

This analysis explores the technologies used by recruited candidates, highlighting key areas where most hires are concentrated.

This analysis emphasizes the dominance of Game Development and DevOps in recruitment, alongside the steady rise of data and security-related roles. Companies should adjust their hiring strategies accordingly to attract and retain top talent in these key areas.

```
technology
Game Development                         519
DevOps                                   495
System Administration                    293
Development - CMS Backend                284
Database Administration                  282
Adobe Experience Manager                 282
Client Success                           271
Development - Frontend                   266
Security                                 266
Mulesoft                                 260
QA Manual                                259
Salesforce                               256
Development - Backend                    255
Data Engineer                            255
Business Analytics / Project Management  255
Business Intelligence                    254
Development - FullStack                  254
Development - CMS Frontend               251
Security Compliance                      250
Design                                   249
QA Automation                            243
Sales                                    239
Social Media Community Management        237
Technical Writing                        223
dtype: int64
```



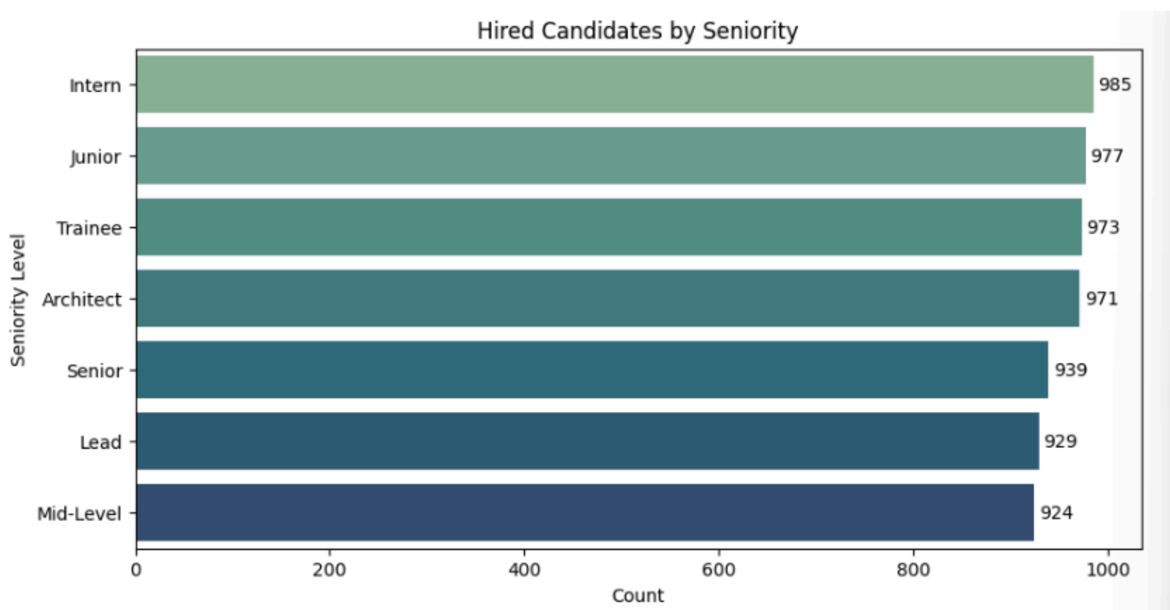
Hiring by Seniority Level

Overview

To understand hiring trends, we analyzed the number of hires across different seniority levels. This helps identify which experience levels are prioritized during recruitment.

```
df_seniority_count = (df[df["is_hired"] == 1]
                      .groupby("seniority")
                      .size()
                      .sort_values(ascending=False))
df_seniority_count
```

```
seniority
Intern       985
Junior       977
Trainee      973
Architect    971
Senior       939
Lead         929
Mid-Level    924
dtype: int64
```



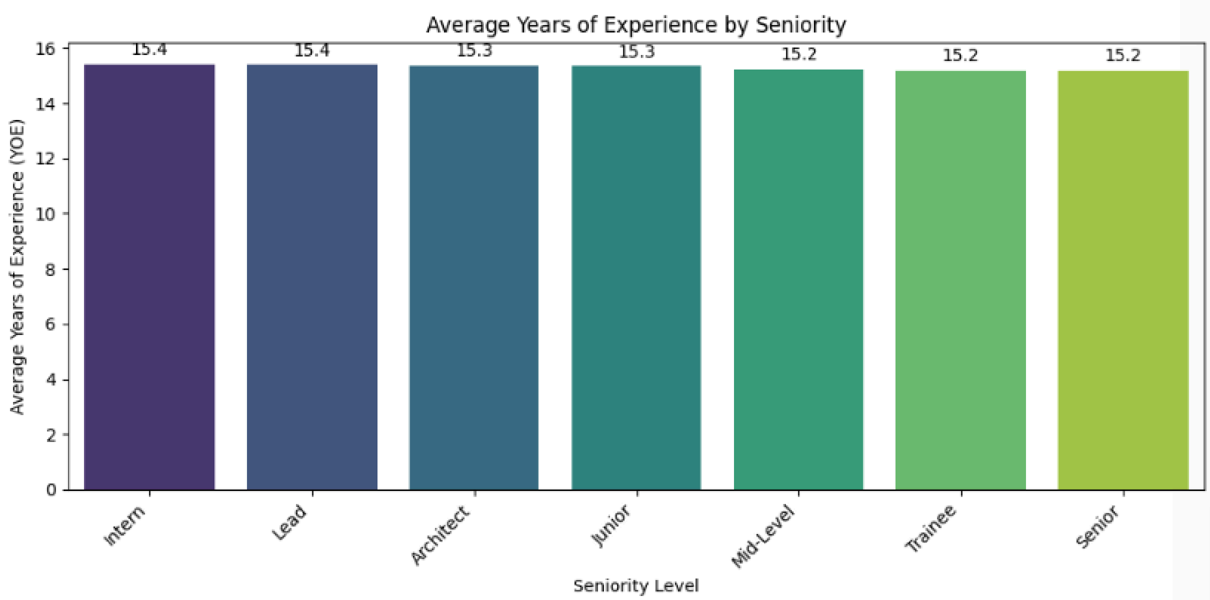
Average Years of Experience (YOE) per Seniority Level

Overview

We analyzed the **average years of experience (YOE)** for each seniority level to see if hiring patterns align with expected experience levels.

```
seniority_avg_yoe = df.groupby("seniority")["yoe"].mean().sort_values(ascending=False)
seniority_avg_yoe
```

```
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
```



Loading the clean data to PostgreSQL

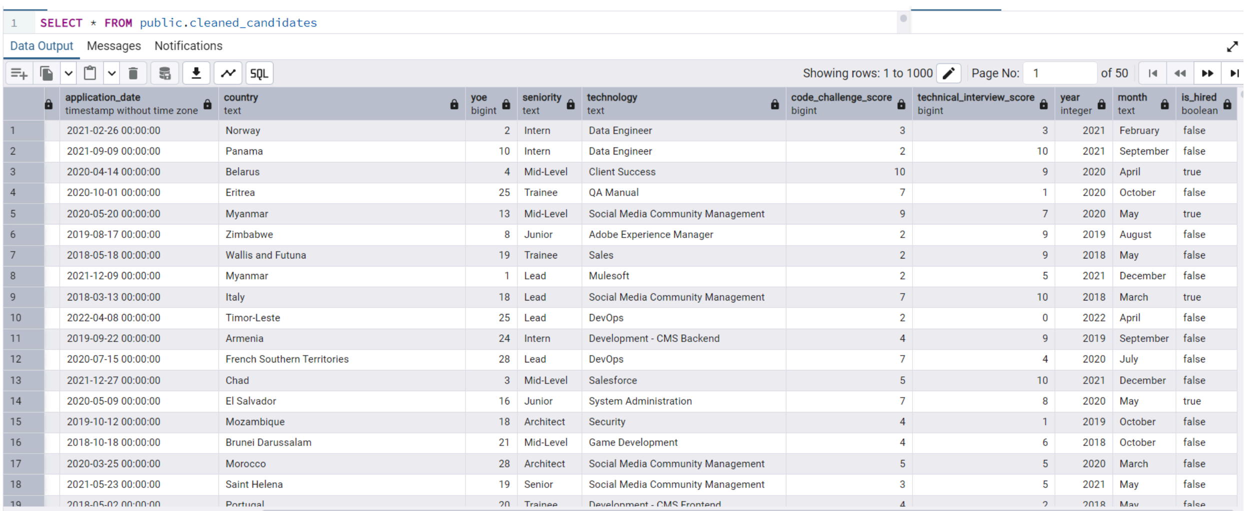
We processed and cleaned the raw dataset using the files `connection_db.py` and `003_cleanDataLoad.ipynb`. During this process, we verified that the `is_hire` column was present to ensure the accuracy of the hiring data.

The dataset has been successfully
cleaned and stored in the PostgreSQL database under the table `cleaned_candidates`.
The process ensures that only refined data is used for further analysis, eliminating inconsistencies and enhancing data reliability.

```
[ ]: df.to_sql("cleaned_candidates", engine, if_exists="replace", index=False)

print("✅ Datos limpios insertados en 'cleaned_candidates'")
```

```
✅ Datos limpios insertados en 'cleaned_candidates'
```



```
1  SELECT * FROM public.cleaned_candidates
```

Data Output Messages Notifications

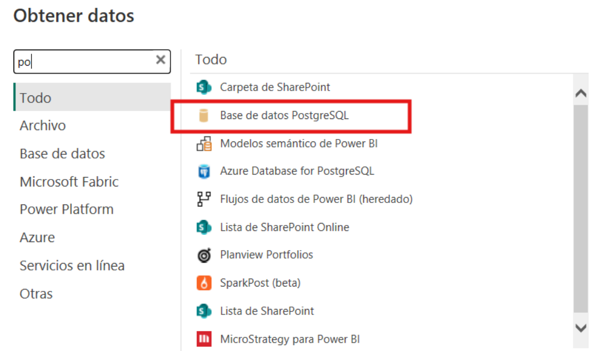
Showing rows: 1 to 1000 Page No: 1 of 50

	application_date timestamp without time zone	country text	yoe bigint	seniority text	technology text	code_challenge_score bigint	technical_interview_score bigint	year integer	month text	is_hired boolean
1	2021-02-26 00:00:00	Norway	2	Intern	Data Engineer	3	3	2021	February	false
2	2021-09-09 00:00:00	Panama	10	Intern	Data Engineer	2	10	2021	September	false
3	2020-04-14 00:00:00	Belarus	4	Mid-Level	Client Success	10	9	2020	April	true
4	2020-10-01 00:00:00	Eritrea	25	Trainee	QA Manual	7	1	2020	October	false
5	2020-05-20 00:00:00	Myanmar	13	Mid-Level	Social Media Community Management	9	7	2020	May	true
6	2019-08-17 00:00:00	Zimbabwe	8	Junior	Adobe Experience Manager	2	9	2019	August	false
7	2018-05-18 00:00:00	Wallis and Futuna	19	Trainee	Sales	2	9	2018	May	false
8	2021-12-09 00:00:00	Myanmar	1	Lead	Mulesoft	2	5	2021	December	false
9	2018-03-13 00:00:00	Italy	18	Lead	Social Media Community Management	7	10	2018	March	true
10	2022-04-08 00:00:00	Timor-Leste	25	Lead	DevOps	2	0	2022	April	false
11	2019-09-22 00:00:00	Armenia	24	Intern	Development - CMS Backend	4	9	2019	September	false
12	2020-07-15 00:00:00	French Southern Territories	28	Lead	DevOps	7	4	2020	July	false
13	2021-12-27 00:00:00	Chad	3	Mid-Level	Salesforce	5	10	2021	December	false
14	2020-05-09 00:00:00	El Salvador	16	Junior	System Administration	7	8	2020	May	true
15	2019-10-12 00:00:00	Mozambique	18	Architect	Security	4	1	2019	October	false
16	2018-10-18 00:00:00	Brunei Darussalam	21	Mid-Level	Game Development	4	6	2018	October	false
17	2020-03-25 00:00:00	Morocco	28	Architect	Social Media Community Management	5	5	2020	March	false
18	2021-05-23 00:00:00	Saint Helena	19	Senior	Social Media Community Management	3	5	2021	May	false
19	2018-05-02 00:00:00	Portugal	20	Trainee	Development - CMS Frontend	4	2	2018	May	false

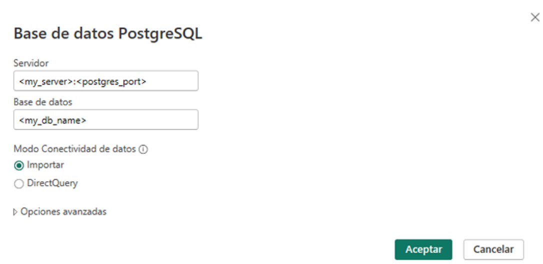
Visualizing the Data

Connecting the database to Power BI

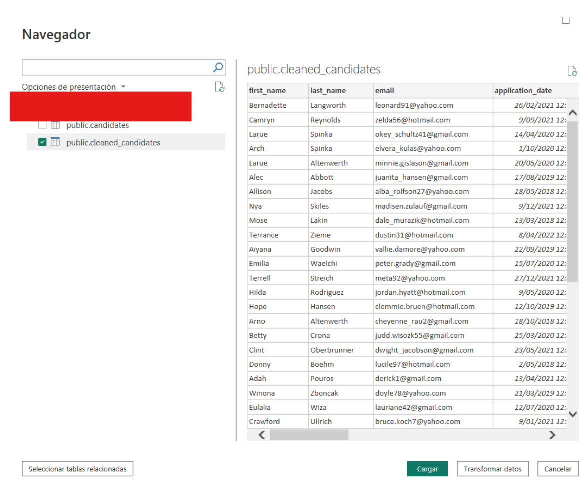
1. Open Power BI Desktop and create a new dashboard. Select the Get data option - be sure you choose the "PostgreSQL Database" option.



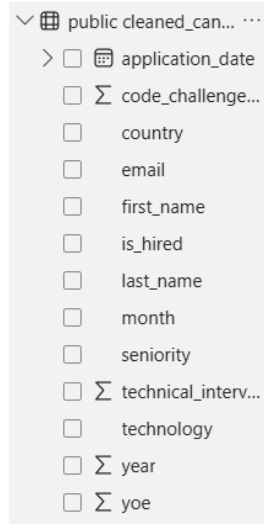
1. Insert the PostgreSQL Server and Database Name



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



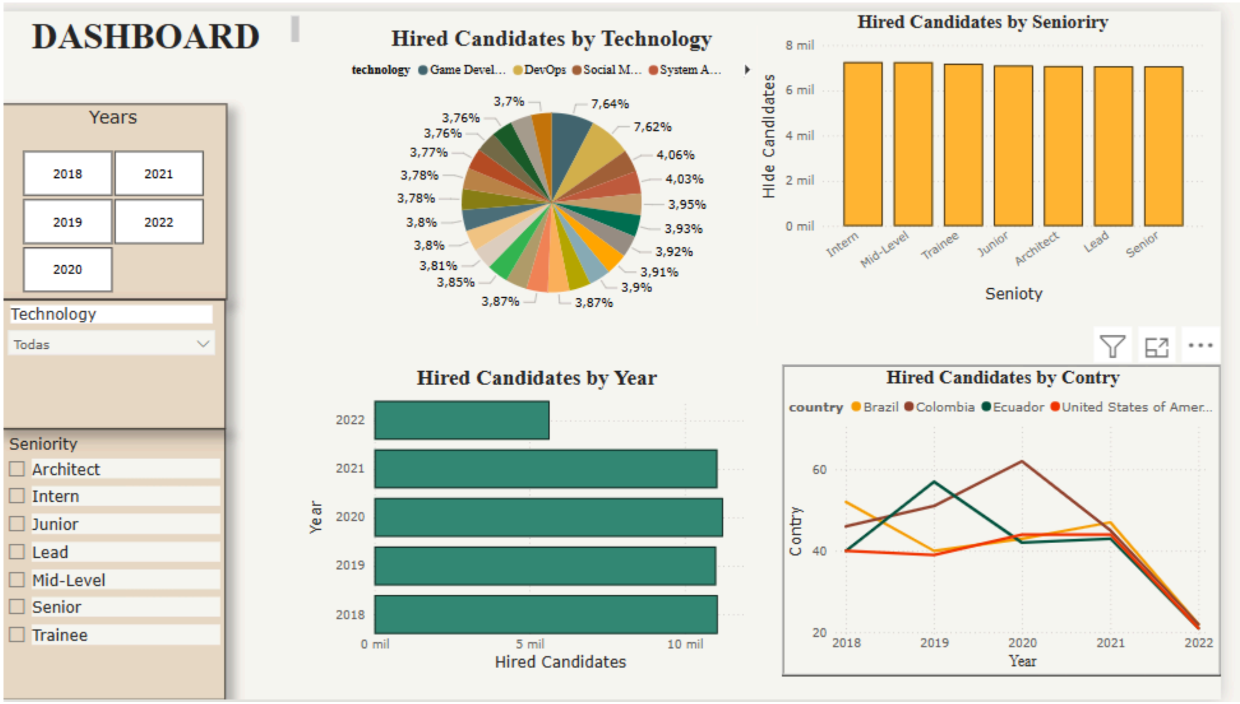
5. Choose the candidates_hired table and start making your own visualizations!



📊 Dashboard Analysis

Here are some key takeaways from the dashboard:

- The **filters** make it easy to explore the data by year, technology, and seniority level, allowing users to focus on specific trends based on their needs.
- The **pie chart (Hired Candidates by Technology)** helps break down the different technology areas where hires were made, making it easier to see which fields dominate recruitment.
- The **bar chart (Hired Candidates by Seniority)** gives a clear picture of how hiring is distributed across different experience levels, showing which roles had the highest number of recruits.
- The **hiring trend by year** shows a steady increase in recruitment, though there's a noticeable drop in 2022, which aligns with previous observations in the dataset.
- The **multi-line graph (Hired Candidates by Country)** provides insight into hiring trends across different regions. Since the dataset includes a large number of countries, only a selected few are shown to keep the visualization clear while still capturing key patterns.



Conclusions

1. **Transforming raw data into valuable insights**

   Throughout this process, we followed a structured workflow to turn raw data into clean, usable information. Using tools like Python and Jupyter Notebook, along with the `connection_db.py` script and the `003_cleanDataLoad.ipynb` notebook, we ensured a smooth **extraction, transformation, and loading (ETL) process** for the dataset.

2. **Data cleaning and preparation**

   Several key transformations were applied to improve data quality, such as **standardizing column names, grouping categories, and creating derived variables like `is_hire`**. These refinements made the dataset more structured and meaningful, allowing for better decision-making.

3. **Exploring data and uncovering trends**

   The exploratory data analysis (EDA) revealed valuable patterns, such as the **distribution of candidates by country, seniority, and technology**. We also identified hiring trends over the years and detected some anomalies, leading to deeper insights into the selection process.

4. **Making data easier to understand through visualization**

   The **dashboard played a crucial role in presenting the findings effectively**. Using pie charts, bar graphs, and line charts, we visualized hiring trends by technology, seniority, country, and year, making it easier to interpret and communicate the results.

5. **Driving smarter hiring decisions**

   By structuring and analyzing the data, we provided essential insights to **optimize recruitment strategies**. We identified high-demand talent areas, analyzed experience trends, and gained a clearer understanding of what influences hiring success. These findings can help refine future talent acquisition processes.

6. **Final thoughts**: This analysis not only helped clean and organize data but also transformed it into a valuable resource for decision-making. Leveraging structured data and clear visualizations enables companies to make more informed, strategic hiring choices.