Workshop - 001: Data engineer

Por: Juan Pablo Gómez Veira

Carrera: Ingeniería de Datos e Inteligencia Artificial

Materia: ETL

Introducción

Este documento describe el desarrollo del Workshop 001 , un ejercicio práctico que simula un desafío real de una entrevista de trabajo. El objetivo fue implementar un proceso completo de Extracción, Transformación y Carga (ETL) utilizando datos de candidatos generados aleatoriamente, almacenarlos en una base de datos relacional y finalmente crear visualizaciones específicas en un dashboard interactivo.

Para llevar a cabo el workshop, utilicé las siguientes tecnologías:

- **Python**: Para manipular y transformar los datos.
- **Jupyter Notebook**: Para documentar y ejecutar el código de manera organizada.
- **PostgreSQL**: Como base de datos relacional para almacenar la información.
- **Power BI**: Para diseñar el dashboard final con las visualizaciones solicitadas.

La consigna del workshop pedía migrar los datos desde un archivo CSV (candidates.csv) con 50,000 filas y 10 columnas a una base de datos, aplicar transformaciones —como determinar qué candidatos fueron contratados según una regla específica— y mostrar métricas en cuatro tipos de gráficos: un pie chart, un bar chart horizontal, un bar chart vertical y un multiline chart.

Todo el proceso está disponible en el siguiente repositorio de GitHub: https://github.com/jpgomezv/ETL_Workshop_001.

A continuación, detallo cada etapa del proyecto, desde la configuración inicial hasta las conclusiones finales.

Configuración del entorno

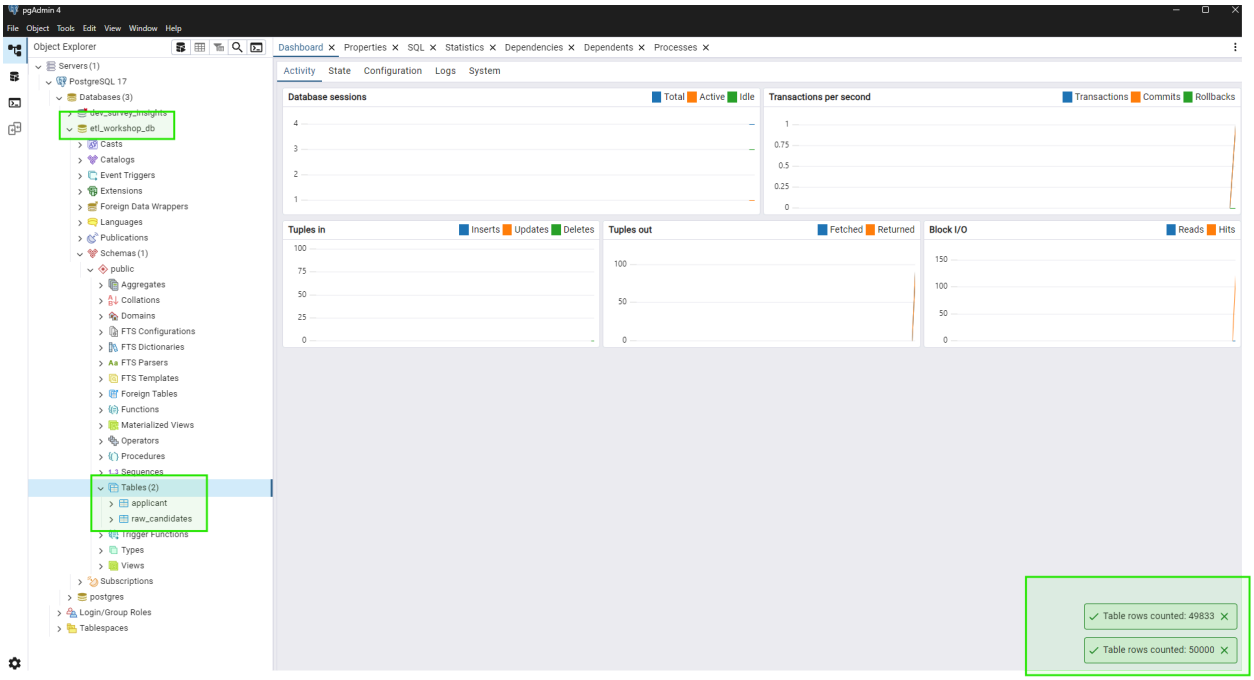
Antes de empezar con el ETL, preparé un entorno de desarrollo limpio y reproducible. Esto fue clave para asegurarme de que todo funcionara correctamente y de que el proyecto fuera fácil de replicar.

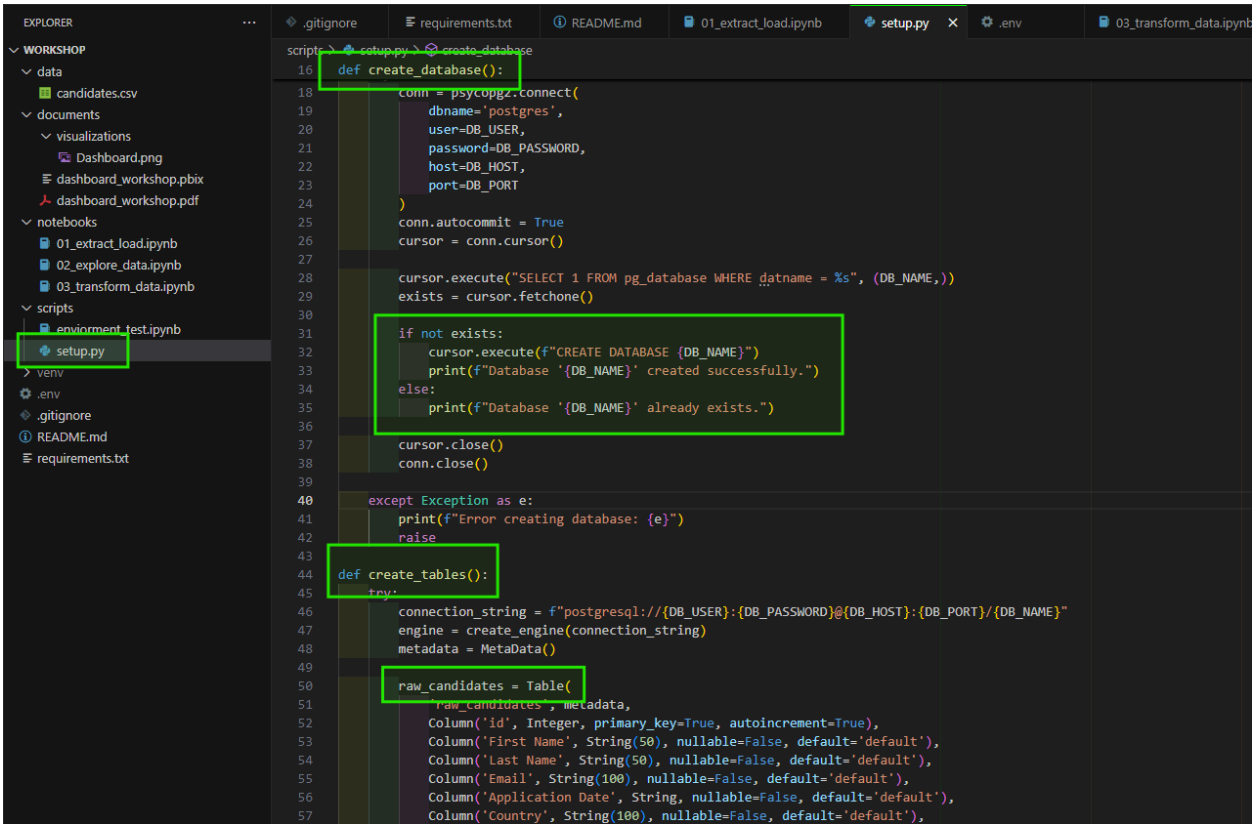
Entorno Virtual y Dependencias

Primero, creé un entorno virtual en Python “venv” para aislar las dependencias del proyecto. Luego, instalé las librerías necesarias usando el archivo requirements.txt, que incluye herramientas como pandas para manipular datos, sqlalchemy para conectar con la base de datos y psycopg2 para trabajar con PostgreSQL.

Configuración de la Base de Datos

Elegí PostgreSQL como base de datos relacional por su robustez y facilidad de integración con Python. Creé una base de datos llamada etl_workshop_db y dos tablas: raw_candidates para los datos crudos y applicant para los datos transformados. Para automatizar este paso, escribí un script en setup.py que configura todo desde cero.


pgAdmin 4
Object Explorer
Dashboard
Properties
SQL
Statistics
Dependencies
Dependents
Processes
Activity
State
Configuration
Logs
System
Database sessions
Total
Active
Idle
Transactions per second
Transactions
Commits
Rollbacks
Tuples in
Inserts
Updates
Deletes
Tuples out
Fetched
Returned
Block I/O
Reads
Hits
Servers (1)
PostgreSQL 17
Databases (3)
etl_workshop_db
Schemas (1)
public
Tables (2)
applicant
raw_candidates
Table rows counted: 49833
Table rows counted: 50000

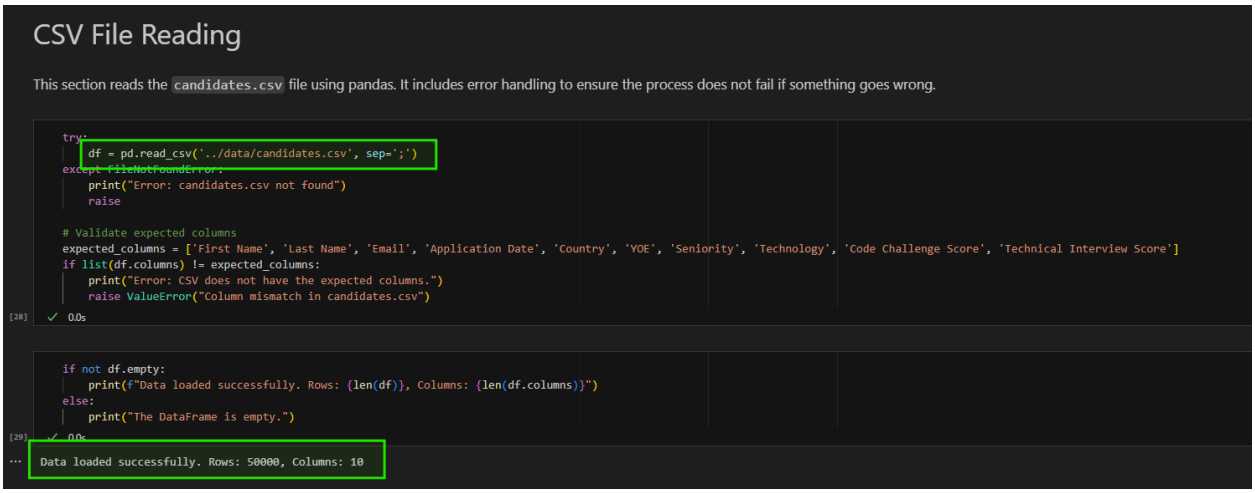

EXPLORER
WORKSHOP
data
candidates.csv
documents
visualizations
Dashboard.png
dashboard_workshop.pbix
dashboard_workshop.pdf
notebooks
01_extract_load.ipynb
02_explore_data.ipynb
03_transform_data.ipynb
scripts
enviorment_test.ipynb
setup.py
venv
.env
.gitignore
README.md
requirements.txt
.gitignore
requirements.txt
README.md
01_extract_load.ipynb
setup.py
.env
03_transform_data.ipynb
def create_database():
conn = psycopg2.connect(
dbname='postgres',
user=DB_USER,
password=DB_PASSWORD,
host=DB_HOST,
port=DB_PORT
)
conn.autocommit = True
cursor = conn.cursor()
cursor.execute("SELECT 1 FROM pg_database WHERE datname = %s", (DB_NAME,))
exists = cursor.fetchone()
if not exists:
cursor.execute(f"CREATE DATABASE {DB_NAME}")
print(f"Database '{DB_NAME}' created successfully.")
else:
print(f"Database '{DB_NAME}' already exists.")
cursor.close()
conn.close()
except Exception as e:
print(f"Error creating database: {e}")
raise
def create_tables():
try:
connection_string = f"postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
engine = create_engine(connection_string)
metadata = MetaData()
raw_candidates = Table(
'raw_candidates', metadata,
Column('id', Integer, primary_key=True, autoincrement=True),
Column('First Name', String(50), nullable=False, default='default'),
Column('Last Name', String(50), nullable=False, default='default'),
Column('Email', String(100), nullable=False, default='default'),
Column('Application Date', String, nullable=False, default='default'),
Column('Country', String(100), nullable=False, default='default'),

Proceso ETL

Dividí el proceso ETL entre los notebooks `01_extract_load.ipynb` y `03_transform_data.ipynb`

Extracción

En esta etapa, cargué los datos del archivo candidates.csv a la tabla raw_candidates en PostgreSQL. Usé pandas para leer el CSV y sqlalchemy para establecer la conexión con la base de datos y subir los datos. El archivo contenía 50,000 filas con columnas como first_name, email, country, application_date, technology, entre otras.



CSV File Reading

This section reads the `candidates.csv` file using pandas. It includes error handling to ensure the process does not fail if something goes wrong.

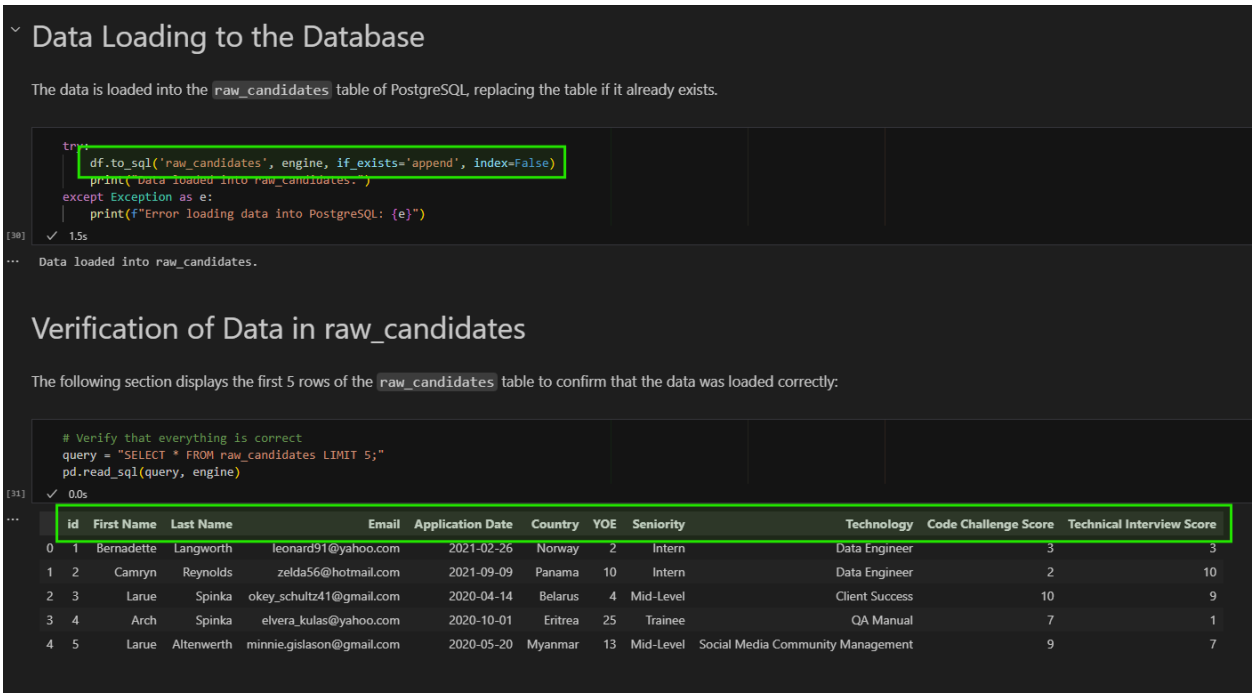
```
try:
    df = pd.read_csv('../data/candidates.csv', sep=';')
except FileNotFoundError:
    print("Error: candidates.csv not found")
    raise

# Validate expected columns
expected_columns = ['First Name', 'Last Name', 'Email', 'Application Date', 'Country', 'YOE', 'Seniority', 'Technology', 'Code Challenge Score', 'Technical Interview Score']
if list(df.columns) != expected_columns:
    print("Error: CSV does not have the expected columns.")
    raise ValueError("Column mismatch in candidates.csv")
```

[28] ✓ 0.0s

```
if not df.empty:
    print(f"Data loaded successfully. Rows: {len(df)}, Columns: {len(df.columns)}")
else:
    print("The DataFrame is empty.")
```

[29] ✓ 0.0s

```
Data loaded successfully. Rows: 50000, Columns: 10
```



Data Loading to the Database

The data is loaded into the `raw_candidates` table of PostgreSQL, replacing the table if it already exists.

```
try:
    df.to_sql('raw_candidates', engine, if_exists='append', index=False)
    print("Data loaded into raw_candidates.")
except Exception as e:
    print(f"Error loading data into PostgreSQL: {e}")
```

[30] ✓ 1.5s

```
Data loaded into raw_candidates.
```

Verification of Data in raw_candidates

The following section displays the first 5 rows of the `raw_candidates` table to confirm that the data was loaded correctly:

```
# Verify that everything is correct
query = "SELECT * FROM raw_candidates LIMIT 5;"
pd.read_sql(query, engine)
```

[31] ✓ 0.0s

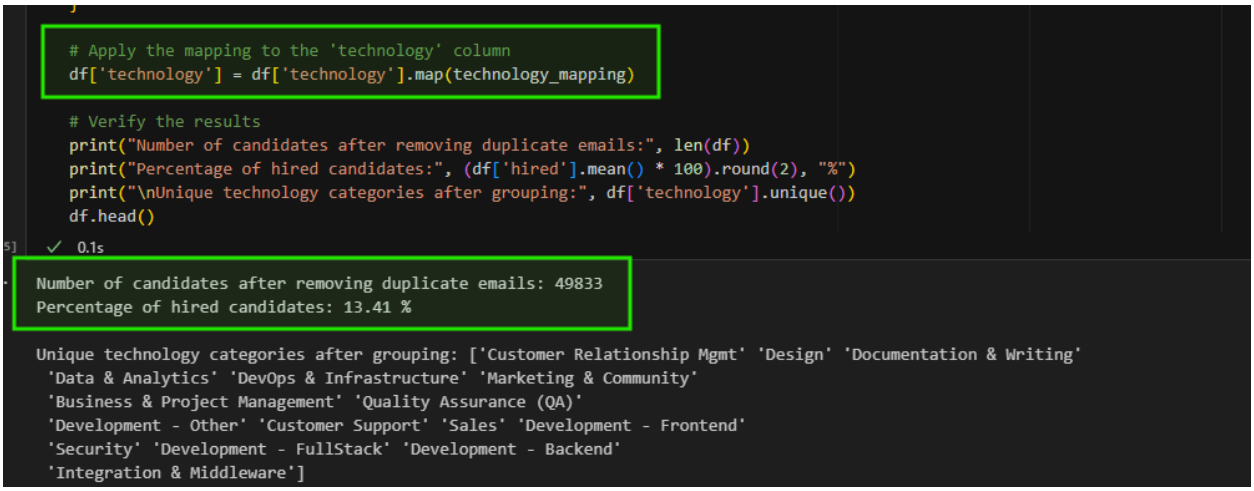
	id	First Name	Last Name	Email	Application Date	Country	YOE	Seniority	Technology	Code Challenge Score	Technical Interview Score
0	1	Bernadette	Langworth	leonard91@yahoo.com	2021-02-26	Norway	2	Intern	Data Engineer	3	3
1	2	Camryn	Reynolds	zelda56@hotmail.com	2021-09-09	Panama	10	Intern	Data Engineer	2	10
2	3	Larue	Spinka	okey_schultz41@gmail.com	2020-04-14	Belarus	4	Mid-Level	Client Success	10	9
3	4	Arch	Spinka	elvera_kulas@yahoo.com	2020-10-01	Eritrea	25	Trainee	QA Manual	7	1
4	5	Larue	Altenwerth	minnie.gislason@gmail.com	2020-05-20	Myanmar	13	Mid-Level	Social Media Community Management	9	7

(Nota: en esa captura en “if_exists='append'”, va es **replace**, aclaro por que cuando lo tenía en append se estaban cargando los datos varias veces y esto claramente es inconveniente)

Transformación

Las transformaciones las realicé en el notebook `03_transform_data.ipynb`. Aquí apliqué varias operaciones para limpiar y preparar los datos:

- Creé la columna hired, que indica si un candidato fue contratado. Según la consigna, un candidato es contratado si su code_challenge_score y technical_interview_score son ambos mayores o iguales a 7.
- Eliminé duplicados en la columna email, quedándome con el registro más reciente basado en application_date.
- Normalicé las columnas technology y country para corregir inconsistencias (por ejemplo, mayúsculas o espacios).
- Agrupé las tecnologías en categorías más amplias para simplificar el análisis.

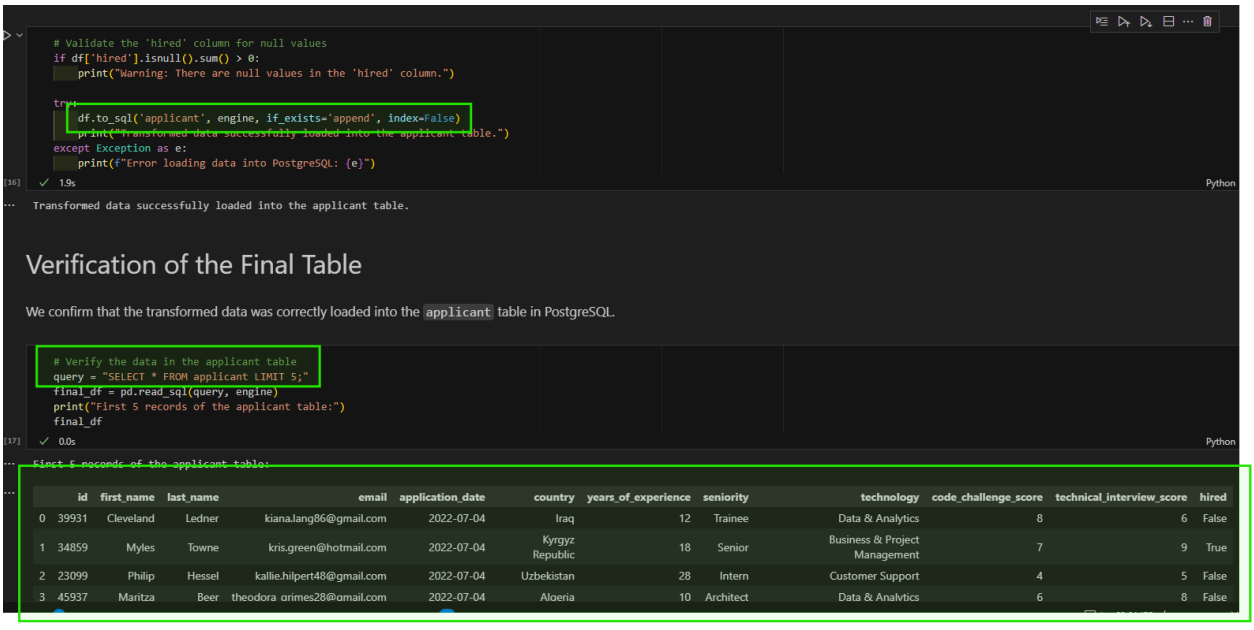


```
# Apply the mapping to the 'technology' column
df['technology'] = df['technology'].map(technology_mapping)

# Verify the results
print("Number of candidates after removing duplicate emails:", len(df))
print("Percentage of hired candidates:", (df['hired'].mean() * 100).round(2), "%")
print("\nUnique technology categories after grouping:", df['technology'].unique())
df.head()
```

```
✓ 0.1s

Number of candidates after removing duplicate emails: 49833
Percentage of hired candidates: 13.41 %

Unique technology categories after grouping: ['Customer Relationship Mgmt' 'Design' 'Documentation & Writing'
 'Data & Analytics' 'DevOps & Infrastructure' 'Marketing & Community'
 'Business & Project Management' 'Quality Assurance (QA)'
 'Development - Other' 'Customer Support' 'Sales' 'Development - Frontend'
 'Security' 'Development - FullStack' 'Development - Backend'
 'Integration & Middleware']
```

Carga

Una vez transformados, cargué los datos finales en la tabla applicant de PostgreSQL. Esta tabla contiene la versión limpia y lista para análisis, con la columna hired y los ajustes necesarios.



```
# Validate the 'hired' column for null values
if df['hired'].isnull().sum() > 0:
    print("Warning: There are null values in the 'hired' column.")

try:
    df.to_sql('applicant', engine, if_exists='append', index=False)
    print("Transformed data successfully loaded into the applicant table.")
except Exception as e:
    print(f"Error loading data into PostgreSQL: {e}")
```

```
✓ 1.9s
Python

Transformed data successfully loaded into the applicant table.
```

Verification of the Final Table

We confirm that the transformed data was correctly loaded into the `applicant` table in PostgreSQL.

```
# Verify the data in the applicant table
query = "SELECT * FROM applicant LIMIT 5;"
final_df = pd.read_sql(query, engine)
print("First 5 records of the applicant table:")
final_df
```

```
✓ 0.0s
Python

First 5 records of the applicant table:
```

	id	first_name	last_name	email	application_date	country	years_of_experience	seniority	technology	code_challenge_score	technical_interview_score	hired
0	39931	Cleveland	Ledner	kiana.lang86@gmail.com	2022-07-04	Iraq	12	Trainee	Data & Analytics	8	6	False
1	34859	Myles	Towne	kris.green@hotmail.com	2022-07-04	Kyrgyz Republic	18	Senior	Business & Project Management	7	9	True
2	23099	Philip	Hessel	kallie.hilpert48@gmail.com	2022-07-04	Uzbekistan	28	Intern	Customer Support	4	5	False
3	45937	Maritza	Beer	theodora.grimes28@gmail.com	2022-07-04	Algeria	10	Architect	Data & Analytics	6	8	False

Análisis Exploratorio de Datos (EDA)

Antes de pasar a las visualizaciones finales, hice un análisis exploratorio en el notebook `02_explore_data.ipynb` para entender mejor los datos y detectar patrones o problemas.

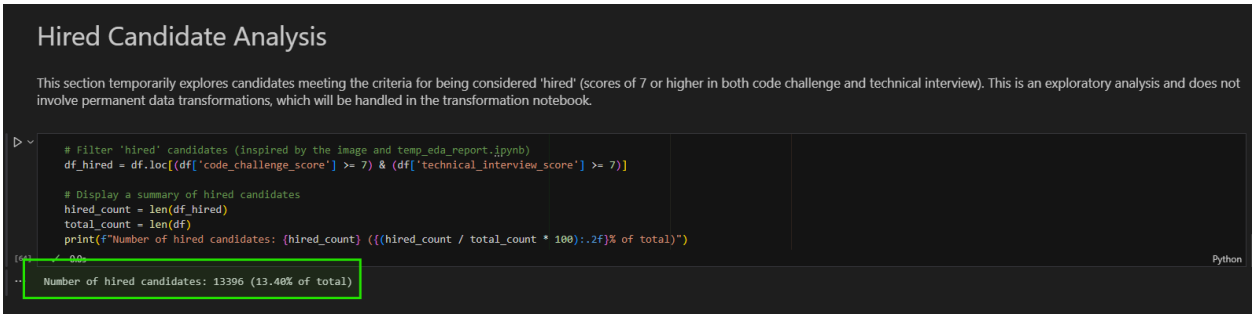
4.1. Hallazgos Clave

- No encontré valores nulos en el dataset, lo cual facilitó el proceso.
- Había duplicados en email, pero los manejé en la etapa de transformación.
- Las variables categóricas como seniority y technology mostraron distribuciones variadas; por ejemplo, algunas tecnologías eran más comunes que otras.
- Las puntuaciones code_challenge_score y technical_interview_score tenían una distribución uniforme, típico de datos generados aleatoriamente. No hay una relación explicada entre ambos puntajes
- Al aplicar la regla de contratación, descubrí que aproximadamente el 13.4% de los candidatos fueron contratados.

4.2. Visualizaciones del EDA

Generé varias gráficas para explorar los datos, como por ejemplo:

- Un histograma de la distribución de seniority.
- Un conteo de las tecnologías más frecuentes.
- Un heatmap para ver la relación entre code_challenge_score y technical_interview_score.

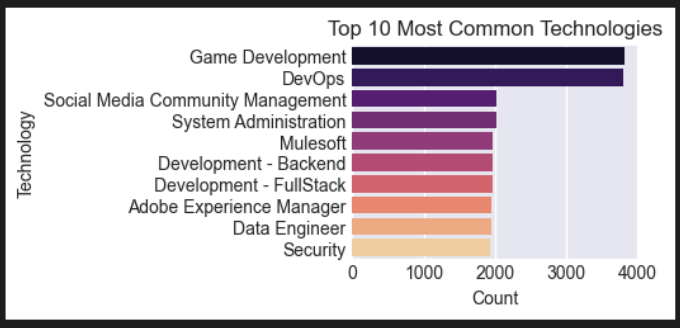


```python
# Check for null values
null_counts = df.isnull().sum()
print("Null values by column:\n", null_counts)

# Check for duplicate rows
duplicates = df.duplicated().sum()
print(f"Number of duplicate rows: {duplicates}")

# Check for duplicate emails
email_duplicates = df['email'].duplicated().sum()
print(f"Number of duplicate emails (excluding the first): {email_duplicates}")

# Review duplicate records
email_duplicates = df.duplicated(subset='email', keep=False).sum()
print("Records sharing the same email:", email_duplicates)
# Optionally display some duplicate email records
if email_duplicates > 0:
    duplicated_emails_df = df[df.duplicated(subset='email', keep=False)].sort_values('email')
    display(duplicated_emails_df.head(10))

# Note on duplicate emails (given the fictional context)
print("""
Note: Duplicate emails might suggest re-applications in a real context, but since the data is randomly generated, \nthese repetitions are likely artifacts of the generation process and may not i
""")
```

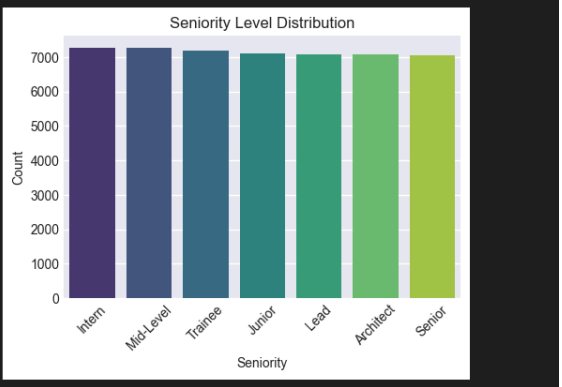
```
✓ 0.0s
Null values by column:
 first_name                0
last_name                  0
email                      0
application_date           0
country                    0
years_of_experience        0
seniority                  0
technology                 0
code_challenge_score       0
technical_interview_score  0
dtype: int64
Number of duplicate rows: 0
Number of duplicate emails (excluding the first): 167
Records sharing the same email: 332
```

```python
# 2. Technology Distribution (top 10, bar chart)
plt.subplot(2, 2, 2)
tech_counts = df['technology'].value_counts().head(10)
sns.barplot(x=tech_counts.values, y=tech_counts.index, hue=tech_counts.index, palette='magma', legend=False)
plt.title('Top 10 Most Common Technologies')
plt.xlabel('Count')
plt.ylabel('Technology')
```

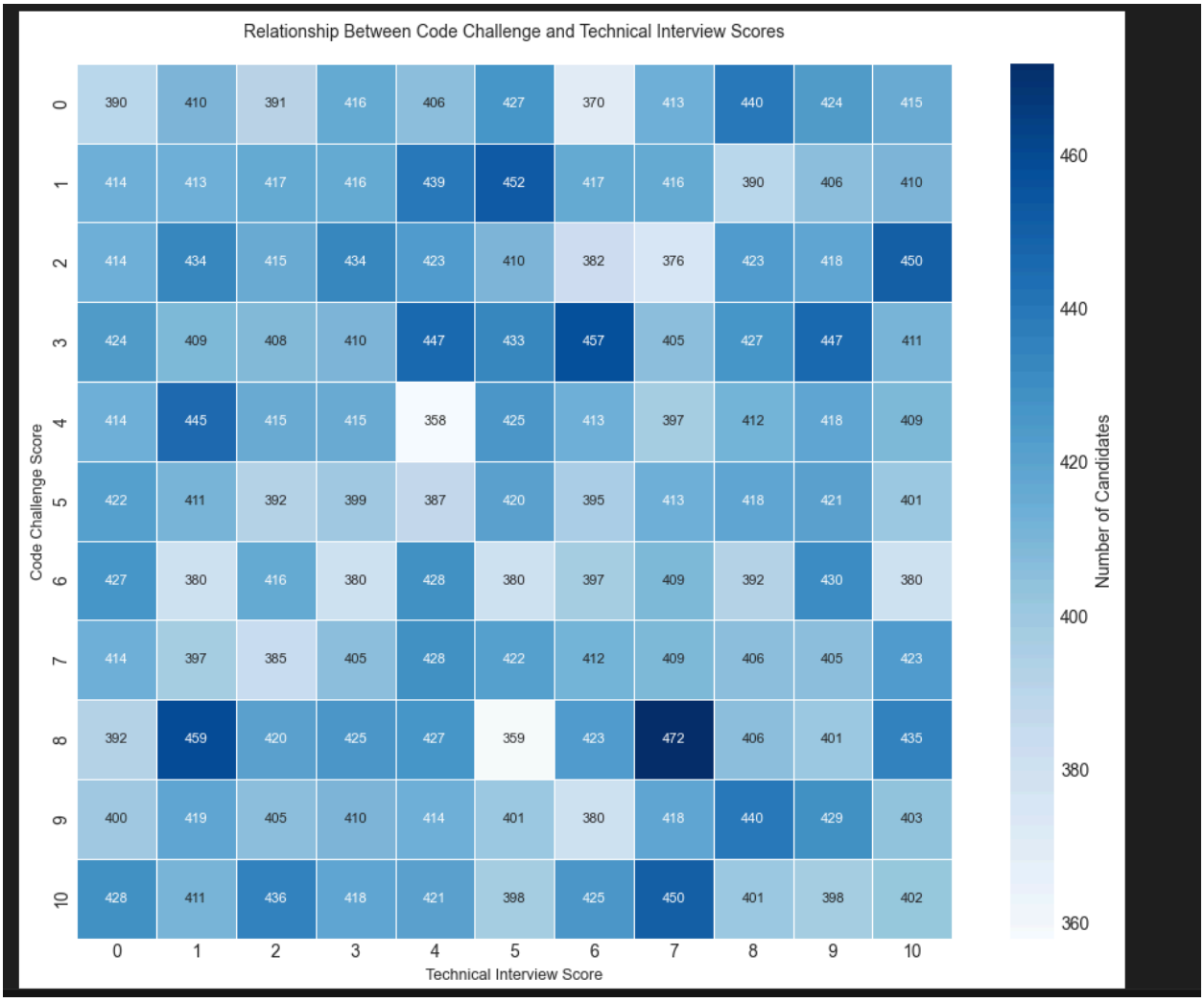
```
✓ 0.1s
Text(0, 0.5, 'Technology')
```



```
Text(0, 0.5, 'Count')
```




```
Distribution of seniority:
 seniority
Intern       7255
Mid-Level    7253
Trainee      7183
Junior       7100
Architect    7079
Lead         7071
Senior       7059
Name: count, dtype: int64
```



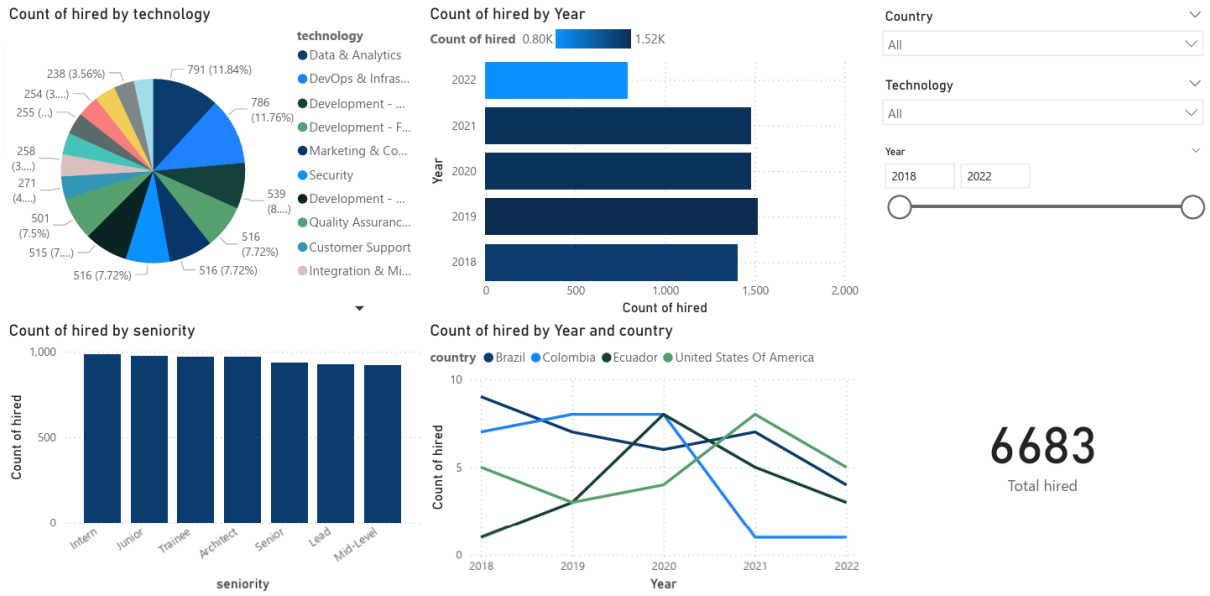
(Dentro del 02_explore_data.ipynb se encuentran más graficas, pero con estas ya se entienden los hallazgos encontrados)

Visualizaciones Finales

Para las visualizaciones finales, usé Power BI conectado directamente a la tabla applicant en PostgreSQL. Diseñé un dashboard con los cuatro gráficos solicitados en la consigna:

1. **Contrataciones por Tecnología (Pie Chart):** Muestra cómo se distribuyen los candidatos contratados entre las categorías de tecnología.
2. **Contrataciones por Año (Horizontal Bar Chart):** Presenta el número de contrataciones por año entre 2018 y 2022.
3. **Contrataciones por Seniority (Bar Chart):** Indica cuántos candidatos contratados hay por nivel de experiencia.
4. **Contrataciones por País a lo Largo de los Años (Multiline Chart):** Muestra las tendencias de contrataciones en USA, Brasil, Colombia y Ecuador a lo largo del tiempo.

Añadí filtros interactivos para Country, Technology y Year, lo que permite explorar los datos de forma dinámica.



Conclusiones

Este workshop fue una gran oportunidad para poner a prueba practicas que se deben tener en proyectos de ETL. Desde configurar el entorno hasta transformar datos y crear visualizaciones, cada paso me enseñó algo nuevo. Algunos puntos clave que rescato:

- Automatizar la configuración del entorno con scripts como setup.py ahorra tiempo y asegura consistencia.
- El EDA es esencial para conocer los datos antes de trabajar con ellos.
- Aplicar reglas de negocio, como la de contratación, requiere pensar bien cómo traducirlas a código.
- Power BI es una herramienta poderosa para presentar resultados de manera clara y profesional (en lo personal me gustó la simplicidad y solidés de uso).

Si tuviera que mejorar el proyecto, intentaría:

- Agregar pruebas para validar las transformaciones.
- Optimizar el código para manejar datasets más grandes.
- Explorar análisis adicionales, como predecir tendencias de contratación.