Example Workspace Code

In [10]:

```
# Get any  patients with Acromegaly + icd9 and 10 codes
query = """
SELECT NFER_PID,
       MIN(NFER_DTM) AS DIAGNOSIS_DTM
FROM   fact_diagnosis fd
       LEFT JOIN diagnosis_dim_diagnosis_code dd
              ON dd.diagnosis_code_dk = fd.diagnosis_code_dk
WHERE  ( Lower(dd.diagnosis_code) LIKE "e22.0"
         AND Lower(diagnosis_method_code) LIKE Any ('icd-10%', 'icd10%') )
        OR ( dd.diagnosis_code LIKE "253.0"
             AND Lower(diagnosis_method_code) LIKE Any ('icd-9%', 'icd9%') )
GROUP  BY NFER_PID
HAVING DIAGNOSIS_DTM IS NOT NULL
"""

# Execute the query
result = sql_client.query(query)

# Fetch the data
diagnosis_cohort_df = result.fetch_df()
print(diagnosis_cohort_df.shape)

print(diagnosis_cohort_df.head())
```

In [10]:

```
# Let's take the ICD codes for acromegaly and see the corresponding diagnosis_code_dk
method = "ICD9"
codes = [
    "253.0",
]

# method = "ICD10"
# codes = [
#     "e22.0",
# ]
# Join the codes list into a properly formatted string for SQL
codes_str = ", ".join(f"'{code}'" for code in codes)

query = f"""
SELECT DISTINCT d.diagnosis_code_dk,
                d.diagnosis_method_code,
                d.diagnosis_code
FROM diagnosis_dim_diagnosis_code AS d
WHERE d.diagnosis_code IN ({codes_str})
AND d.diagnosis_method_code = '{method}'
"""

# Execute the query
result = sql_client.query(query)

# Fetch the data
df = result.fetch_df()

# Get the list of diagnosis_code_dk
asthma_codes_dk = df["diagnosis_code_dk"].to_list()
# Let's see the first 10 diagnosis_code_dk
print(asthma_codes_dk[:10])
print(asthma_codes_dk)
print(f"Total number of IDs in acromegaly: {len(asthma_codes_dk)}")
```

Example Workspace Code

In [7]:

```
## add dosage to the main dataframe
## Which medications are most effective at normalizing IGF-1 levels?
# Are IGF-1 levels influenced by overlapping medications?

# Investigate whether patients on multiple overlapping medications have different IGF-1 levels compared to those on mono

import pandas as pd
import os

# Load the main dataset
main_data = pd.read_csv("subset_filtered_data.tsv", sep="\t", low_memory=False)

# List of medications
medications = ["Pegvisomant", "Octreotide", "Pasireotide", "Cabergoline", "Lanreotide"]

# Path template for medication files
file_format = "{}_medication_data.tsv"  # Update with the actual file location if needed

# Initialize an empty DataFrame to store doses
dose_dataframes = []

# Load each medication file and extract ADMINISTERED_DOSE
for med in medications:
    file_name = file_format.format(med)

    if os.path.exists(file_name):
        # Read medication data
        med_data = pd.read_csv(file_name, sep="\t", low_memory=False)

        # Keep only NFER_PID and ADMINISTERED_DOSE
        if "ADMINISTERED_DOSE" in med_data.columns:
            med_data = med_data[["NFER_PID", "ADMINISTERED_DOSE"]].drop_duplicates()
            med_data = med_data.rename(columns={"ADMINISTERED_DOSE": f"{med}_Dose"})

            # Append to the list
            dose_dataframes.append(med_data)
        else:
            print(f"Skipping {file_name} - 'ADMINISTERED_DOSE' column missing.")
    else:
        print(f"File {file_name} not found. Skipping.")

# Merge all extracted dose data with main dataset
for dose_df in dose_dataframes:
    main_data = main_data.merge(dose_df, on="NFER_PID", how="left")

# Save the updated dataset
```

```
plt.figure(figsize=(12, 8))
sns.barplot(
    data=category_surgery_breakdown,
    x='Category',
    y='Unique_Patient_Count',
    hue=surgery_status_column,
    palette='Set2'
)
plt.title('Breakdown of Patients by Category and Surgery Status', fontsize=16)
plt.xlabel('Category', fontsize=12)
plt.ylabel('Unique Patient Count', fontsize=12)
plt.xticks(rotation=45)
plt.legend(title='Surgery Status')
plt.tight_layout()
plt.show()
```

In [9]:

```
# List of medications to analyze
medications = ["pegvisomant", "somatropin", "octreotide", "pasireotide",
               "cabergoline", "lanreotide", "sandostatin", "hydrocortisone", "levothyroxine"]

# Get NFER_PID of patients with Acromegaly
acromegaly_ids = patient_list  # Ensure this is defined earlier
acromegaly_ids_str = ", ".join(map(str, acromegaly_ids))  # Convert IDs to a string for SQL query

# Format the IDs for SQL
def format_ids_for_sql(ids):
    return ", ".join([f"'{id}'" for id in ids])
```

Example Workspace Code

In [44]:
```
# List of medications to analyze
medications = ["pegvisomant", "somatropin", "octreotide", "pasireotide",
               "cabergoline", "lanreotide", "sandostatin", "hydrocortisone", "levothyroxine"]

# Get NFER_PID of patients with Acromegaly
acromegaly_ids = patient_list  # Ensure this is defined earlier
acromegaly_ids_str = ", ".join(map(str, acromegaly_ids))  # Convert IDs to a string for SQL query

# Format the IDs for SQL
def format_ids_for_sql(ids):
    return ", ".join([f"'{id}'" for id in ids])

sql_formatted_ids = format_ids_for_sql(acromegaly_ids)

# Dictionary to store results for each medication
results = {}

# Initialize a list to store concatenated data for all medications
all_adverse_events = []

for drug in medications:
    # Query for adverse events filtered by Acromegaly patient IDs
    query_ae = f"""
    SELECT entity_2_pref_name AS Adverse_Event
    FROM fact_syn_augmented_curation
    WHERE prediction_type = 'Adverse_Event'
      AND entity_1_pref_name = '{drug}'
      AND NFER_PID IN ({sql_formatted_ids})
    """

    # Execute the query
    result_ae = sql_client.query(query_ae)
    ae_df = result_ae.fetch_df()

    # Count unique adverse events and their occurrences
    events, counts = np.unique(ae_df["Adverse_Event"], return_counts=True)
    ae_counts = pd.DataFrame({
        "Adverse Event": events,
        "Number of Occurrences": counts
    }).sort_values("Number of Occurrences", ascending=False)

    # Add drug information to the dataframe
    ae_counts["Drug"] = drug
```

In [24]:
```
events, counts = np.unique(nivolumab_df.entity_2_pref_name, return_counts=True)

print(
    "BERT model found {} unique adverse events associated with {}".format(
        len(events), theraputic
    )
)

AE_counts = pd.DataFrame(
    {"Adverse Event": events, "Number of occurences": counts}
).sort_values("Number of occurences", ascending=False)

display(AE_counts.head())
```

Example Workspace Code

In [35]:

```
## trying to get around the 100M limit error for radiosurgery data
def format_ids_for_sql(ids):
    return ", ".join([f"'{id}'" for id in ids])

def get_surgery_data(sql_client, acromegaly_ids):
    sql_formatted_ids = format_ids_for_sql(acromegaly_ids)

    # Query for FACT_SYN_AUGMENTED_CURATION
    syn_query = f"""
    SELECT
        'FACT_SYN_AUGMENTED_CURATION' as source_table,
        NFER_PID as patient_id,
        ENTITY_1_ORIGINAL_PHRASE as procedure_name,
        ENTITY_2_PREF_NAME as procedure_name2,
        NFER_DTM as record_date,
        PREDICTION_TYPE as performed,
        FACT_GUID,
        META_INFORMATION
    FROM FACT_SYN_AUGMENTED_CURATION
    WHERE NFER_PID IN ({sql_formatted_ids})
    AND LOWER(ENTITY_1_ORIGINAL_PHRASE) LIKE '%radiosurgery%'
    AND LOWER(PREDICTION_TYPE) LIKE '%Procedure%Performed%'
    """
# Query for FACT_AUGMENTED_CURATION
    aug_query = f"""
    SELECT
        'FACT_AUGMENTED_CURATION' as source_table,
        NFER_PID as patient_id,
        ENTITY_1_ORIGINAL_PHRASE as procedure_name,
        ENTITY_2_PREF_NAME as procedure_date,
        NFER_DTM as record_date,
        META_INFORMATION
    FROM FACT_AUGMENTED_CURATION
    WHERE NFER_PID IN ({sql_formatted_ids})
    AND LOWER(ENTITY_1_ORIGINAL_PHRASE) LIKE '%radiosurgery%'
    """

    print("Executing queries...")
```

In [32]:

```
# get transsphenoidal surgery information

# Get NFER_PID of patients with Acromegaly
acromegaly_ids = patient_list  # Ensure this is defined earlier
acromegaly_ids_str = ", ".join(map(str, acromegaly_ids))  # Convert IDs to a string for SQL query

# Format the IDs for SQL
def format_ids_for_sql(ids):
    return ", ".join([f"'{id}'" for id in ids])

sql_formatted_ids = format_ids_for_sql(acromegaly_ids)

# Query for both tables
syn_query = f"""
SELECT
    'FACT_SYN_AUGMENTED_CURATION' as source_table,
    NFER_PID as patient_id,
    ENTITY_1_ORIGINAL_PHRASE as procedure_name,
    ENTITY_2_PREF_NAME as procedure_name2,
    NFER_DTM as record_date,
    PREDICTION_TYPE as performed,
    FACT_GUID,
    META_INFORMATION
FROM FACT_SYN_AUGMENTED_CURATION
WHERE NFER_PID IN ({sql_formatted_ids})
AND LOWER(ENTITY_1_ORIGINAL_PHRASE) LIKE '%transsphenoidal%'
AND LOWER(PREDICTION_TYPE) LIKE '%ProcedureAPerformed%'

"""
# Query for FACT_AUGMENTED_CURATION
aug_query = f"""
SELECT
    'FACT_AUGMENTED_CURATION' as source_table,
    NFER_PID as patient_id,
    ENTITY_1_ORIGINAL_PHRASE as procedure_name,
    ENTITY_2_PREF_NAME as procedure_date,
    NFER_DTM as record_date,
    META_INFORMATION
FROM FACT_AUGMENTED_CURATION
WHERE NFER_PID IN ({sql_formatted_ids})
AND LOWER(ENTITY_1_ORIGINAL_PHRASE) LIKE '%transsphenoidal%'
"""

print("Executing query...")
# Execute first query
syn_result = sql_client.query(syn_query, fetch_max=True)
if not syn_result.success:
    raise Exception(f"FACT_SYN_AUGMENTED_CURATION query failed: {syn_result.error}")
syn_data = syn_result.fetch_df()

    # Execute second query
aug_result = sql_client.query(aug_query, fetch_max=True)
if not aug_result.success:
    raise Exception(f"FACT_AUGMENTED_CURATION query failed: {aug_result.error}")
aug_data = aug_result.fetch_df()

# Combine the results using pandas concat
surgery_data = pd.concat([syn_data, aug_data], ignore_index=True)

    # Sort the combined data
surgery_data = surgery_data.sort_values(['patient_id', 'record_date'])

    # Print summary statistics
print(f"\nFound {len(surgery_data)} total surgery records")
print(f"Number of unique patients with surgery: {surgery_data['patient_id'].nunique()}")

print("\nRecords by source table:")
print(surgery_data['source_table'].value_counts())

print("\nUnique procedure descriptions found:")
print(surgery_data['procedure_name'].unique())

    # Save the data
surgery_data.to_csv("transsphenoidal_surgery_data.tsv", sep="\t", index=False)
print("\nData saved to 'transsphenoidal_surgery_data.tsv'")

    # Show sample of the data
print("\nSample of retrieved data:")
print(surgery_data[['patient_id', 'procedure_name', 'procedure_date', 'record_date']].head())
```