BMI6016 Data Quality Assessment

Name: Liam Wilson

Framework: The framework selected for this data quality assessment was that described by Weiskopf and Weng.

Domains: The domains selected are the demographic data found in “Patient”, and the ICD-code-based disease data in the “Diagnosis” domain.

Elements: The elements within the `patient` table that were examined were sex, race, and briefly age_at_deaht and postal code. The elements within the `diagnosis` table are code, code system, date, and encounter ID. Patient ID is used briefly in both in the concoordance analysis.

In [ ]:
```
#load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime

plt.style.use('seaborn')
sns.set_palette('husl')
%matplotlib inline
```

Load and Preview Data

In [ ]:
```
# read the CSV files containing the domain data (from the XLSX file provided)
patients = pd.read_csv('patient.csv')
diagnoses = pd.read_csv('diagnosis.csv')

# preview the data
display(patients.head())
display(diagnoses.head())
```

	patient_id	sex	race	ethnicity	year_of_birth	age_at
0	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	F	Unknown	Not Hispanic or Latino	1980	
1	c45f42c418ebd65d25d10276a3f2a8ba33d7d6c5	F	Unknown	Hispanic or Latino	1997	
2	9cab24d7325547465b6439a5aea68b6efebace47	F	White	Hispanic or Latino	1970	
3	22e3b3b96da711bdf73d24fe6886c083f12ca3e2	F	White	Not Hispanic or Latino	1984	
4	580043e2b64b6e170ad284101375bdedd71ec231	M	White	Not Hispanic or Latino	1991	

	patient_id	encounter_id	code
0	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	7ec55977affd4f27497ae22f4723686bb7c76d59	IC
1	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	7ec55977affd4f27497ae22f4723686bb7c76d59	IC
2	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	7ec55977affd4f27497ae22f4723686bb7c76d59	IC
3	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	7ec55977affd4f27497ae22f4723686bb7c76d59	IC
4	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96bc34a	7ec55977affd4f27497ae22f4723686bb7c76d59	IC

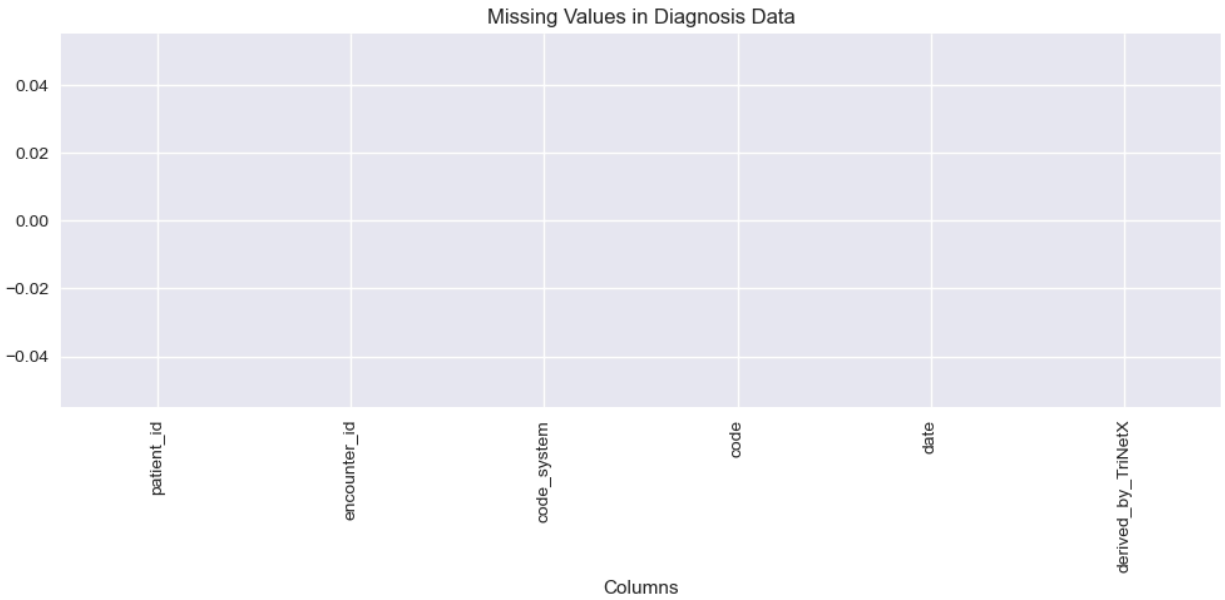
1. Completeness Assessment

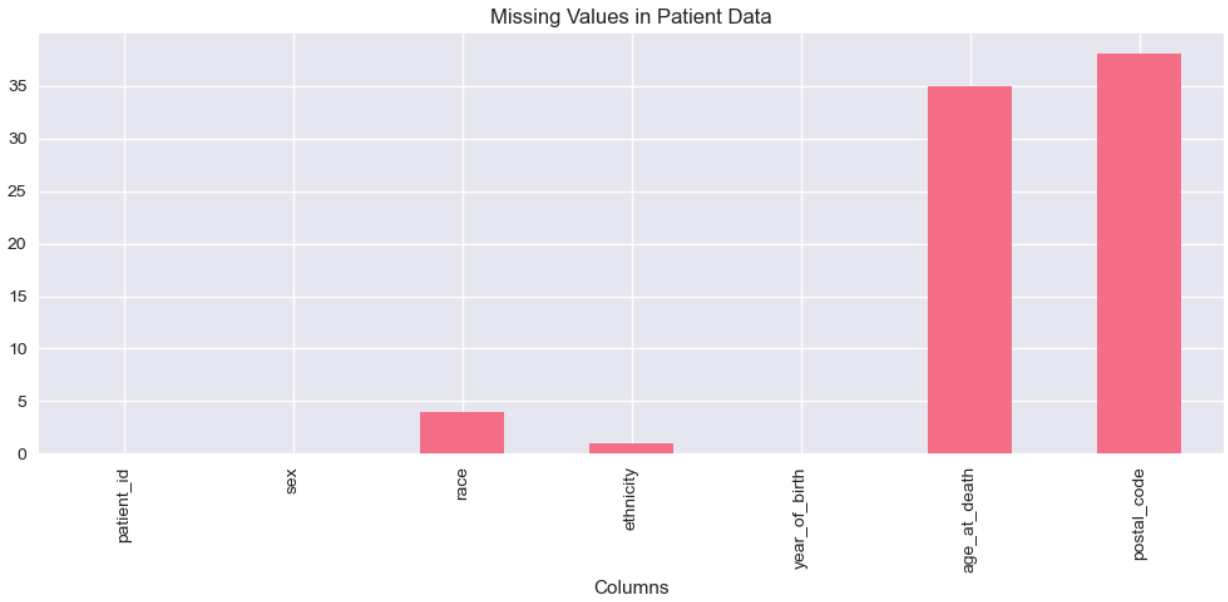
In [ ]:
```
def plot_missing_values(df, title):
    # there is an argument about whether unknown should be treated this way, bu
    # it is also potentially prudent to come up with a more robust list (N/A, r
    missing = (df.isna() | (df == "Unknown")).sum()
    plt.figure(figsize=(10, 5))
    missing.plot(kind='bar')
    plt.title(f'Missing Values in {title}')
    plt.xlabel('Columns')
    plt.tight_layout()
    plt.show()
    return missing

display(diagnoses.info())
display(patients.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31404 entries, 0 to 31403
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   patient_id          31404 non-null  object
 1   encounter_id        31404 non-null  object
 2   code_system         31404 non-null  object
 3   code                31404 non-null  object
 4   date                31404 non-null  int64 
 5   derived_by_TriNetX  31404 non-null  object
dtypes: int64(1), object(5)
memory usage: 1.4+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38 entries, 0 to 37
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   patient_id     38 non-null     object 
 1   sex            38 non-null     object 
 2   race           38 non-null     object 
 3   ethnicity      38 non-null     object 
 4   year_of_birth  38 non-null     int64  
 5   age_at_death   3 non-null      float64
 6   postal_code    0 non-null      float64
dtypes: float64(2), int64(1), object(4)
memory usage: 2.2+ KB
None
```

In [ ]:
```
missing_diagnoses = plot_missing_values(diagnoses, 'Diagnosis Data')
missing_patients = plot_missing_values(patients, 'Patient Data')
```





Based on these two graphs, we know that there are no missing values present in the diagnosis table across any columns. However, age_at_death and postal_code columns in the patient domain are almost always missing (specifically, always missing in the case of postal code). Additionally, there are some values in the ethnicity and race fields that are not known or null.

2. Correctness Assessment

In [ ]:
```
import re

# look for invalid patient IDs
invalid_patient_ids = patients[patients['patient_id'].isna() ]
print(f"Number of invalid patient IDs: {len(invalid_patient_ids)}")

# invalid ICD codes
invalid_icds = diagnoses['code'].str.strip().eq('')
print(f"Number of empty ICD codes: {invalid_icds.sum()}")

# check if ICD codes are in appropriate format:
# regex taken from: https://stackoverflow.com/questions/32720294/check-icd10-v
def is_valid_icd10(code):
    icd10_format = r'^([a-tA-T]|[v-zV-Z])\d[a-zA-Z0-9](\.[a-zA-Z0-9]{1,4})?$'
    return bool(re.match(icd10_format, code))


def check_code_system_validity(code_system):
    if code_system == "icd10":
        icd10_only = diagnoses['code_system'] == 'ICD-10-CM'
        invalid_codes = diagnoses[icd10_only][~diagnoses[icd10_only]['code'].a
    elif code_system == 'icd9':
        icd9_only = diagnoses['code_system'] == 'ICD-9-CM'
        invalid_codes = diagnoses[icd9_only][~diagnoses[icd9_only]['code'].app

    print(f"{len(invalid_codes)} incorrectly formatted {code_system} codes")
```

```
check_code_system_validity('icd10')

# now, look at the codes that are not icd10:
non_icd10 = diagnoses[diagnoses['code_system'] != 'ICD-10-CM']
print(non_icd10['code_system'].unique())

def is_valid_icd9(code):
    icd9_format = "^(V\d{2}(\.\d{1,2})?|\d{3}(\.\d{1,2})?|E\d{3}(\.\d)?)$"
    return bool(re.match(icd9_format, code))

check_code_system_validity('icd9')
```

```
Number of invalid patient IDs: 0
Number of empty ICD codes: 0
0 incorrectly formatted icd10 codes
['ICD-9-CM']
107 incorrectly formatted icd9 codes
```

Generally, the ICD domain is "correct" in that the codes are formatted correctly: a small percentage of ICD9 codes may not be formatted correctly, but that could also be a fault of the regex. A more robust analysis could involve looking up the exact ICD codes through an API.

In [ ]:

```
def analyze_column_variation(df):
    results = {}

    for col in df.columns:
        total_count = df[col].count()
        unique_count = df[col].nunique()

        variation_ratio = unique_count / total_count if total_count > 0 else 0

        value_counts = df[col].value_counts()
        mode = value_counts.index[0] if len(value_counts) > 0 else None
        mode_count = value_counts.iloc[0] if len(value_counts) > 0 else 0

        results[col] = {
            'total_values': total_count,
            'unique_values': unique_count,
            'variation_ratio': f"{variation_ratio:.2%}",
            'mode': mode,
            'mode_count': mode_count
        }

    return pd.DataFrame(results).T

print("Patient Data Variability:")
display(analyze_column_variation(patients))
print("\nDiagnosis Data Variability:")
display(analyze_column_variation(diagnoses))
```

```
Patient Data Variability:
```

	total_values	unique_values	variation_ratio	
patient_id	38	38	100.00%	fb5fe7dfa1a03f7d823c62e51d3f6a9ca96
sex	38	2	5.26%	
race	38	3	7.89%	
ethnicity	38	3	7.89%	Not Hispanic or
year_of_birth	38	28	73.68%	
age_at_death	3	3	100.00%	
postal_code	0	0	0.00%	

```
Diagnosis Data Variability:
```

	total_values	unique_values	variation_ratio	
patient_id	31404	38	0.12%	00433ff44e321af809a89ad0fac
encounter_id	31404	2745	8.74%	7983358e67c0b9926d3cac1f67e9
code_system	31404	2	0.01%	
code	31404	2389	7.61%	
date	31404	1397	4.45%	2018-
derived_by_TriNetX	31404	1	0.00%	

We can use the variation tables to make some further notes about correctness: there are two possible gender fields (which can be seen as reductively accurate but potentially inaccurate). We also noted earlier the variation in ICD code system, but this is not strictly an issue with correctness. There are some surprisingly frequent patients (~2400 ICD codes for one subject?) that warrants further examination. In a similar (and to some degree more concerning) vein, one encounter id maps to 235 rows, which seems implausibly high. The expressed White/Not-Hispanic match up in mode helps to validate those columns to some degree.

In [ ]:
```
suspicious_encounter = diagnoses[diagnoses['encounter_id'] == '7983358e67c0b99
print("Codes:")
print(suspicious_encounter['code'].value_counts())

print("Dates in encounter")
print(suspicious_encounter['date'].value_counts())


print("Potentially different patients in encounter")
print(suspicious_encounter['patient_id'].value_counts())
```

```
Codes:
code
B20       6
A41.89    3
R65.21    3
G93.41    3
G99.8     3
          ..
J96.90    2
J90       2
I50.1     2
415.19    1
518.81    1
Name: count, Length: 102, dtype: int64
Dates in encounter
date
2018-08-21    235
Name: count, dtype: int64
Potentially different patients in encounter
patient_id
282cfd8256b0df98543f4b06111248ab20a42e4e    235
Name: count, dtype: int64
```

There appears to be a high likelihood of duplication: the patient ID and date are the same, but there are multiple instances of the same ICD codes, which is somewhat confusing. This may mean some deduplication is in order.

3. Concordance Assessment

In [ ]:
```
dx_patients = set(diagnoses['patient_id'].dropna())
demo_patients = set(patients['patient_id'].dropna())

patients_without_dx = demo_patients - dx_patients
dx_without_patients = dx_patients - demo_patients

print(f"Patients without diagnosis: {len(patients_without_dx)}")
print(f"Diagnoses missing patient : {len(dx_without_patients)}")

plt.figure(figsize=(10, 6))
venn_data = [
    len(demo_patients - dx_patients),
    len(demo_patients & dx_patients),
    len(dx_patients - demo_patients)
]
plt.bar(['Only in Patients', 'In Both', 'Only in Diagnoses'], venn_data)
plt.title('Patient ID Overlap')
```

Out[ ]:
```
Patients without diagnosis: 0
Diagnoses missing patient : 0
Text(0.5, 1.0, 'Patient ID Overlap')
```

In [ ]:
```
male_icd10 = ['C60.0','Q55.21', 'N52.2', 'N50.0'] #random male-specific ICD10 c
female_icd10 = ['C53.0','B37.32', 'D07.2', 'N76.0'] # random female-specific IC

male_code_count = diagnoses[diagnoses['code'].isin(male_icd10)].shape[0]
female_code_count = diagnoses[diagnoses['code'].isin(female_icd10)].shape[0]

print(f"Total sex-specific ICDs: {male_code_count + female_code_count}")

def check_sex_diagnosis_concordance(patients_df, diagnoses_df, male_codes, fema
    merged = diagnoses_df.merge(patients_df[['patient_id', 'sex']], on='patien

    male_discrepancies = merged[(merged['code'].isin(male_codes)) & (merged['s
    female_discrepancies = merged[(merged['code'].isin(female_codes)) & (merge

    print(f" {len(male_discrepancies)} male ICD10 for non-male patients")
    print(f" {len(female_discrepancies)} female-specific ICD10 for non-female p


discrepancies = check_sex_diagnosis_concordance(patients, diagnoses, male_icd1
```

```
Total sex-specific ICDs: 6
 0 male ICD10 for non-male patients
 0 female-specific ICD10 for non-female patients
```

There are no lonesome patients in either set. A very brief and cursory view of a handful of gender-specific ICD codes (from icd10data.com) did not yield anything surprising or discordant, but including many more ICD10s could be more fruitful.

4. Plausibility Assessment

In [ ]:
```
patients['age'] = 2025 - pd.to_numeric(patients['year_of_birth'])
```

```
plt.figure(figsize=(10, 6))
plt.hist(patients['age'].dropna(), bins=30)
plt.title('Age Distribution')

implausible_old = patients[patients['age'] > 120] # arbitrary
implausible_future = patients[patients['age'] < 0]

print(f"Number of implausibly old patients: {len(implausible_old)}")
print(f"Number of patients born in the future {len(implausible_future)}")
```

```
Number of implausibly old patients: 0
Number of patients born in the future 0
```

There don't seem to be any plausibility issues with respect to age; other columns are difficult to review plausibility (e.g. encounter IDs are hashes). Some date plausibility is reviewed in the Currency section below as well. Reviewing data present prior to this section, I suspect some plausibility issues with the sheer number of ICD codes for a single encounter (as >100 comorbidities would seem relatively abnormal to me), but this is at least partially explained by the duplication we noticed with a high degree of multiplicity for the same ICD code for the one encounter with 250 rows.

5. Currency Assessment

Analyze the temporal aspects of the data.

In [ ]:
```
diagnoses['date'] = pd.to_datetime(diagnoses['date'], format='%Y%m%d')

latest_record = diagnoses['date'].max()
earliest_record = diagnoses['date'].min()
data_span = latest_record - earliest_record
```

```
# check recency
today = pd.Timestamp.now()
days_since_last_record = (today - latest_record).days
records_last_30d = diagnoses[diagnoses['date'] > (today - pd.Timedelta(days=30
records_last_90d = diagnoses[diagnoses['date'] > (today - pd.Timedelta(days=90

# look for gaps
daily_records = diagnoses['date'].dt.date.value_counts().sort_index()
gaps = daily_records[daily_records == 0]
max_gap = max((gaps.index[i+1] - gaps.index[i]).days for i in range(len(gaps)-

print(f"Time Span:")
print(f"Earliest : {earliest_record.date()}")
print(f"Latest : {latest_record.date()}")
print(f"Span {data_span.days} days")

print(f"Days since last record: {days_since_last_record}")


unique_visits = diagnoses.drop_duplicates(['patient_id', 'date'])

patient_spans = (unique_visits.groupby('patient_id').agg({'date': lambda x: x.

print(f"\nAverage time between unique visits: {patient_spans.days:.1f} days")
```

```
Time Span:
Earliest : 2011-01-04
Latest : 2019-11-30
Span 3252 days
Days since last record: 1891

Average time between unique visits: 51.0 days
```

The data is not particularly recent with the last visit being over 5 years ago. However, the data is quite continuous: the average subject seems to have an appointment more frequently than every other month. Over the course of ~9 years, this is likely pretty powerful despite the age/recency of the data for understanding long-term disease progression, healthcare outcomes, and general analysis of temporal health trends. There are no dates absurdly far in the past and no dates absurdly far in the future to the point of implausibility as well.