VA Diabetes Overview

Problem:

Veterans who receive care in Veterans Health Administration experience and suffer from a higher burden of obesity and diabetes compared to non-Veteran populations. The human and economic costs are staggering as reflected in the recent 2020 Obesity VA/DoD Clinical Practice Guideline Management of Adult Overweight and Obesity and the newly revised 2023 VA/DoD Clinical Practice Guideline Management of Type 2 Diabetes Mellitus. Most patients with diabetes mellitus are asymptomatic, which leads to delayed and more complex treatment.

Research Question:

Will empowering the veteran with knowledge and simple assessment promote healthier lifestyle choices that address and mitigate risk factors associated with T2DM?

Research Demographic - Veterans with BMI >= 25

Outcomes we hope to achieve over a 6 - 12 month follow up:

1. Weight loss: Evaluate percent of patients that achieved 5%; Evaluate percent of patients that achieved 10%

2. HbA1c reduction: Evaluate percent of patients that achieved HbA1c less than 6.5%; Evaluate percent of patients that reduced HbA1c by more than 1%

Objective:

The prevalence of Type 2 Diabetes Mellitus (T2DM) among veterans is a multifaceted issue. Environmental, genetic and lifestyle interactions have not been fully explored as predeterminants of T2DM. Glycated Hemoglobin (HbA1c) is one of the primary tools to diagnose T2DM. However, often times this is only monitored effectively after the onset of the disease. This research will develop a classification model to proactively identify veterans at high risk for T2DM, providing risk scores that enable earlier interventions and potentially prevent or delay disease development.

Imports

In [ ]:
```
# Package imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
```

```
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

# Set display options to show all rows
#pd.set_option('display.max_rows', 20)
```

In [ ]:
```
import os

# Retrieve the 'Projects' environment variable
base_dir = os.environ.get('Projects', '/Users/ben/Projects')

# Construct the path to the Excel file
file_path = os.path.join(base_dir, 'VA_diab_review', 'update.xlsx')
```

In [ ]:
```
# Loading xlsx file into a dataframe
df = (
    pd.read_excel(file_path, sheet_name = 'Sheet1')
    # Dropping columns with 0 non-null count
    .dropna(axis=1, how='all')
    # Lowercasing and replacing spaces with underscores and remove leading s
    .rename(columns=lambda x: x.strip().lower().replace(' ', '_').replace('-
)
```

In [ ]:
```
df.info()
df.tail()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1770 entries, 0 to 1769
Data columns (total 28 columns):
 #   Column                               Non-Null Count  Dtype
---  ------                               --------------  -----
 0   subject                              574 non-null    float64
 1   date_of_visit                        1181 non-null   object
 2   age                                  1225 non-null   float64
 3   sex                                  549 non-null    object
 4   wt                                   1175 non-null   float64
 5   sbp                                  1161 non-null   float64
 6   dbp                                  1160 non-null   object
 7   bmi                                  1105 non-null   object
 8   obese                                572 non-null    object
 9   pre_dm                               553 non-null    object
 10  dm_t2                                555 non-null    object
 11  a1c                                  832 non-null    object
 12  trig                                 679 non-null    float64
 13  hdl                                  679 non-null    object
 14  total_chol                           679 non-null    float64
 15  ldl                                  675 non-null    object
 16  medications_(gluc,_bp,_lipid,_statin) 502 non-null    object
 17  cgm                                  21 non-null     object
 18  mh_dx                                330 non-null    object
 19  wo_mh_dx                             217 non-null    object
 20  metf                                 2 non-null      object
 21  sglt2i                               2 non-null      object
 22  dpp4                                 3 non-null      object
 23  su                                   2 non-null      object
 24  ccb                                  3 non-null      object
 25  ace/arb                              4 non-null      object
 26  diur                                 3 non-null      object
 27  statin                               4 non-null      object
dtypes: float64(6), object(22)
memory usage: 387.3+ KB
```

Out[ ]:

	subject	date_of_visit	age	sex	wt	sbp	dbp	bmi	obese	pre_dm	...
1765	NaN	2024-05-30 00:00:00	NaN	NaN	196.0	125.0	81	28.2	NaN	NaN	...
1766	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1767	573.0	2024-05-31 00:00:00	NaN	NaN	190.0	135.0	79	27.3	n	n	...
1768	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
1769	574.0	2024-06-13 00:00:00	NaN	NaN	243.0	116.0	67	33	y	n	...

5 rows × 28 columns

Cleaning & Transformation

Research Audit

```
1. drop any nulls subsetting for ['date_of_visit', 'wt',
'a1c']
2. create visit category for initial and follow up (address
research questions)
3. non-numeric character check necessary for expected numeric
dtypes that imported as object
4. laboratory nulls will not be imputed / outlier check
(missing bmi impute by creating ht)
5. a1c standard for diagnoses; create function to label
encode for binary, multi-class / pre_dm and dm_t2 can be
dropped
6. create blood_pressure feature as ordinal EDA / potential
drop
7. create body_type instead of obese from bmi for EDA /
potential drop
8. drop med columns due to NULLS / drop age column due to
error / drop sex due to balance
 HbA1c Target Labels

    Healthy: Below 5.7%
    Prediabetes: 5.7% to 6.4%
    Diabetes: 6.5% or higher

blood_pressure

    Normal: SBP is less than 120 and DBP is less than 80.
    Elevated: SBP is between 120 and 129 and DBP is less than
80.
    Hypertension Stage 1: SBP is between 130 and 139 or DBP
is between 80 and 89.
    Hypertension Stage 2: SBP is 140 or higher or DBP is 90
or higher.
    Hypertensive Crisis: SBP is above 180 or DBP is above
120.
    UNDETERMINED: Any blood pressure reading that does not
fit the above categories

 body_type
    Underweight: BMI less than 18.5
    Normal weight: BMI 18.5 to 24.9
    Overweight: BMI 25 to 29.9
    Obesity: BMI 30 and over
```

Pre-process and Model

```
8. date_of_visit, subject can be dropped before modeling
9. split dataframe into train/test
10. standarize and select classification models for eval
```

```
In [ ]: df = df.dropna(subset=['date_of_visit', 'wt', 'a1c'])  # Drop rows with any
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 819 entries, 0 to 1769
Data columns (total 28 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   subject                                 548 non-null    float64
 1   date_of_visit                           819 non-null    object 
 2   age                                     721 non-null    float64
 3   sex                                     524 non-null    object 
 4   wt                                      819 non-null    float64
 5   sbp                                     814 non-null    float64
 6   dbp                                     814 non-null    object 
 7   bmi                                     797 non-null    object 
 8   obese                                   547 non-null    object 
 9   pre_dm                                  540 non-null    object 
 10  dm_t2                                   541 non-null    object 
 11  a1c                                     819 non-null    object 
 12  trig                                    620 non-null    float64
 13  hdl                                     620 non-null    object 
 14  total_chol                              620 non-null    float64
 15  ldl                                     616 non-null    object 
 16  medications_(gluc,_bp,_lipid,_statin)  429 non-null    object 
 17  cgm                                     19 non-null     object 
 18  mh_dx                                   313 non-null    object 
 19  wo_mh_dx                                204 non-null    object 
 20  metf                                    2 non-null      object 
 21  sglt2i                                  2 non-null      object 
 22  dpp4                                    3 non-null      object 
 23  su                                      2 non-null      object 
 24  ccb                                     3 non-null      object 
 25  ace/arb                                 4 non-null      object 
 26  diur                                    3 non-null      object 
 27  statin                                  4 non-null      object 
dtypes: float64(6), object(22)
memory usage: 185.6+ KB
```

```
In [ ]: # Address data types of a1c, hdl, and ldl by removing non-numeric characters
        columns_of_interest = ['bmi','a1c', 'hdl', 'ldl']
        unique_values = {col: df[col].unique() for col in columns_of_interest}

        unique_values
```

Out[ ]:
```
{'bmi': array([39.07, nan, 34.5, 31.67, 29.9, 29.59, 30.02, 24.7, 32.5, 38.
7,
        34.12, 39.3, 30.5, 28, 42.9, 29, 33.8, 31.1, 41, 37.5, 37.7, 29.7,
        33.1, 31.4, 30.7, 29.6, 26.7, 26, 45.8, 40.6, 55.3, 31.5, 28.6, 27,
        33.09, 33.5, 43.1, 45.6, 41.5, 38.4, 36.2, 35.2, 36.6, 44.6, 46,
        31.7, 33.4, 32.8, 35, 35.6, 34.6, 37.6, 30, 25, 42.7, 34.8, 32.1,
        34.1, 38, 33.6, 30.2, 36, 36.9, 35.1, 38.2, 28.7, 35.8, 27.4, 50.5,
        36.4, 36.8, 33.7, 30.3, 29.5, 28.8, 32.6, 40, 34.2, 30.8, 35.5,
        26.8, 34.3, 25.5, 24.3, 27.7, 38.5, 32.4, 34.9, 31.2, 24, 23.5,
        34.7, 41.1, 39.1, 35.3, 36.5, 31, 28.3, 40.5, 41.8, 40.2, 39.9,
        31.8, 43.6, 33.3, 36.1, 46.5, 40.1, 39.7, 50.8, 35.9, 34.4, 45.3,
        42.8, 36.7, 37.2, 32.3, 31.3, 32.9, 29.2, 27.9, 25.2, 28.9, 27.5,
        27.6, 30.9, 33, 30.1, 57.5, 25.6, 30.6, 29.1, 35.4, 42.5, 29.3,
        33.9, 31.6, 39.6, 47.5, 41.2, 37, 33.2, 30.4, 41.7, 41.6, 42, 28.4,
        26.4, 36.3, 40.8, 43, 40.7, 41.9, 32, 45.5, 46.4, 23, 21, 23.6,
        29.8, 25.1, 26.6, 25.7, 23.8, 39.2, '37.6q', 31.9, 45.9, 37.4,
        26.2, 54, 24.4, 39, 40.3, '37?', 26.9, 24.6, 42.3, 32.2, 38.1,
        43.4, 23.4, 42.4, 39.8, 37.1, 28.5, 43.3, 27.8, 28.2, 25.9, 27.3,
        44, 47, 37.3, 28.1, 29.4, 37.9, 35.7, 37.8, 26.3, 32.7, 34, 38.8,
        3, 27.2, 41.3, 43.8, 24.9, 42.1, 38.9, 38.3, 43.5, 55.6, 22.6,
        49.3, 27.1, 54.7, 54.9, 23.2, 41.4, 48.4, 45.7, 44.4, 25.8, 47.1,
        40.4, 42.6, 52.2, 25.3, 24.2, 44.8, 57.2, 38.6, 33.16],
       dtype=object),
 'a1c': array([6.4, 6.3, 5.8, 6.7, 5.3, 5.6, 9, 6.5, 6.1, 5.7, 6.9, 5.5, 5,
        '5.7(2021)', 6, 5.4, 7.2, 5.1, 8.5, 7, 6.8, 5.9, 5.2, 6.6, 10.4,
        9.2, 4.7, 8.6, 7.7, 7.9, 8.2, 4.4, 9.4, 4.9, 7.3, 7.5, 4.8, 7.1,
        8.8, 11, 6.2, 9.5, 10.6, 10.3, 4.6, 8.3, 9.1, 8.1, 7.8, 8,
        '7.1 (dex)', 4.5, 7.6, 10.9, '5.6 (5.8)', 9.7, '5.6 (5.7)',
        '5.4 (5.7)', 9.8, 8.4, 10.7, '5.5 (5.7)', '8.5 (CGM 7.2)',
        '6.6 (CGM)', '5.5(5.9)', '5.3(5.7)', 0.52, '5.3 (5.7)', 10, 10.5,
        '5.6(5.8)', 9.9, 13.5, 11.9, 4, 14.1, 11.4, 7.4], dtype=object),
 'hdl': array([54, nan, 52, 49, 33, 23, 28, 46, 27, 40, 36, 31, 34, 20, 50,
53,
        44, 56, 68, 58, 55, 45, 29, 51, 35, 38, 41, 42, 32, 47, 39, 37,
        'yy', 26, 30, 48, 43, 25, 76, 59, 61, 60, 57, 22, 62, 77, 73, 78,
        69, 66, 63, 64, 75, 91, 71, 74, 18, 275, 101, 87, 24, 428, 98],
       dtype=object),
 'ldl': array([22.1, nan, 74, 84.6, 120.3, 53, 25, 130.9, 19, 96, 69, 116,
95,
        100, 97, 108, 104, 115, 119, 137, 35, 152, 121, 101, 66, 157, 94,
        165, 153, 144, 80, 164, 125, 149, 128, 129, 'nc', 113, 93, 61, 114,
        33, 102, 134, 198, 175, 136, 130, 64, 23, 73, 213, 158, 172, 44,
        162, 150, 29, 21, 87, 120, 109, 78, 138, 91, 65, 161, 103, 43, 51,
        111, 156, 63, 107, 126, 90, 83, 173, 187, 123, 147, 140, 146, 88,
        191, 151, 79, 76, 39, 77, 176, 181, 177, 99, 59, 31, 106, 131, 145,
        81, 84, 194, 82, 143, 141, 124, 122, 110, 159, 118, 174, 127, 193,
        155, 47, 42, 56, 75, 142, 112, 89, 0, 163, 62, 224, 105, 206, 86,
        98, 168, 67, 132, 68, 50, 41, 117, 170, 133, 184, 72, 28, 221, 154,
        135, 71, 186, 265, 234, 3, 92, 55, 7.2, 54, 85, 180, 232, 139, 27,
        160, 273, 60, 208, 183, 167, 202, 207, 182, 169, 148, 220, 'NC',
        200, 233, 197, 203, 32, 179, 188, 8, 58, 199, 166], dtype=object)}
```

In [ ]:
```python
# Define the function to clean and convert bmi, a1c, ldl, hdl dtypes
def clean_and_convert_columns(df, columns):
    for column in columns:
        # Handle text within parentheses and any textual values for hdl and
        df[column] = df[column].apply(lambda x: np.nan if isinstance(x, str)
        # Remove anything within parentheses and convert to float
        df[column] = df[column].astype(str).str.replace(r"\(.*\)", "", regex
    return df

# Apply the function to clean and convert the columns
columns_to_clean = ['bmi','a1c', 'hdl', 'ldl']
clean_and_convert_columns(df, columns_to_clean)
```

Out[ ]:

	subject	date_of_visit	age	sex	wt	sbp	dbp	bmi	obese	pre_dm	.
0	1.0	2023-03-27 00:00:00	124.0	Male	264.0	129.0	78	39.07	YES	NO	.
1	NaN	2023-06-08 00:00:00	124.0	NaN	258.0	116.0	62	NaN	NaN	NaN	.
4	NaN	2024-05-16 00:00:00	NaN	NaN	233.0	104.0	79	34.50	NaN	NaN	.
6	2.0	2023-03-28 00:00:00	124.0	Male	233.0	130.0	89	31.67	YES	NO	.
7	NaN	2023-06-27 00:00:00	124.0	NaN	210.0	139.0	83	NaN	NaN	NaN	.
...	...	...	...	...	...	...	...	...	...	...	.
1758	NaN	2024-05-23 00:00:00	NaN	NaN	231.0	142.0	77	29.00	NaN	NaN	.
1760	570.0	2024-05-31 00:00:00	NaN	NaN	200.0	132.0	75	32.80	y	n	.
1762	571.0	2024-05-30 00:00:00	NaN	NaN	267.0	129.0	97	34.30	y	n	.
1767	573.0	2024-05-31 00:00:00	NaN	NaN	190.0	135.0	79	27.30	n	n	.
1769	574.0	2024-06-13 00:00:00	NaN	NaN	243.0	116.0	67	33.00	y	n	.

819 rows × 28 columns

In [ ]:
```
# convert date to datetime and forwardfill subject
df['date_of_visit'] = pd.to_datetime(df['date_of_visit'], errors='coerce')
df['subject'] = df['subject'].ffill()
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 819 entries, 0 to 1769
Data columns (total 28 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   subject                                819 non-null    float64
 1   date_of_visit                          819 non-null    datetime64[ns]
 2   age                                    721 non-null    float64
 3   sex                                    524 non-null    object
 4   wt                                     819 non-null    float64
 5   sbp                                    814 non-null    float64
 6   dbp                                    814 non-null    object
 7   bmi                                    795 non-null    float64
 8   obese                                  547 non-null    object
 9   pre_dm                                 540 non-null    object
 10  dm_t2                                  541 non-null    object
 11  a1c                                    802 non-null    float64
 12  trig                                   620 non-null    float64
 13  hdl                                    619 non-null    float64
 14  total_chol                             620 non-null    float64
 15  ldl                                    574 non-null    float64
 16  medications_(gluc,_bp,_lipid,_statin)  429 non-null    object
 17  cgm                                    19 non-null     object
 18  mh_dx                                  313 non-null    object
 19  wo_mh_dx                               204 non-null    object
 20  metf                                   2 non-null      object
 21  sglt2i                                 2 non-null      object
 22  dpp4                                   3 non-null      object
 23  su                                     2 non-null      object
 24  ccb                                    3 non-null      object
 25  ace/arb                                4 non-null      object
 26  diur                                   3 non-null      object
 27  statin                                 4 non-null      object
dtypes: datetime64[ns](1), float64(10), object(17)
memory usage: 185.6+ KB
```

In [ ]:

```python
# all visits table

df_full = df.copy()
```

In [ ]:

```python
# Create new column for visit type (initial or followup)

df_sorted = df.sort_values(by=['subject', 'date_of_visit'])

# Function to label visits as 'initial' or 'follow_up', and drop middle visi
def label_visits(group):
    if len(group) > 2:  # If there are more than 2 visits, drop the middle c
        return pd.concat([group.head(1).assign(visit='initial'), group.tail(
    elif len(group) == 2:  # If there are exactly 2 visits
        return pd.concat([group.head(1).assign(visit='initial'), group.tail(
    else:  # If there is only 1 visit
        return group.assign(visit='initial')

# Apply the function to each group and re-assign to df
df = df_sorted.groupby('subject', as_index=False).apply(label_visits).reset_
```

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['subject', 'date_of_visit', 'age', 'sex', 'wt', 'sbp', 'dbp', 'bmi',
       'obese', 'pre_dm', 'dm_t2', 'a1c', 'trig', 'hdl', 'total_chol', 'ld
l',
       'medications_(gluc,_bp,_lipid,_statin)', 'cgm', 'mh_dx', 'wo_mh_dx',
       'metf', 'sglt2i', 'dpp4', 'su', 'ccb', 'ace/arb', 'diur', 'statin',
       'visit'],
      dtype='object')
```

In [ ]:
```
df.drop(columns=['age','sex','obese','medications_(gluc,_bp,_lipid,_statin)'
```

In [ ]:
```
# Function to create diabetes status
def categorize_diabetes_and_clean(df, a1c_column):
    """
    Categorizes HbA1c levels, adds binary and multi-class labels to the Data
    and drops 'db_stat' and 'status' columns if they exist. Categories are t

    Parameters:
    - df: DataFrame containing a1c data.
    - a1c: The column name in the DataFrame that contains a1c values.

    The function modifies the DataFrame in-place and adds two new categorica
    - db_bin: Binary categorization (0 for Healthy, 1 for Diabetes).
    - db_multi: Multi-class categorization (0 for Healthy, 1 for Pre-Diabete

    Drops 'db_stat' and 'status' columns if present.
    """
    # Binary categorization
    db_bin_cats = pd.Categorical(df[a1c_column].apply(lambda x: 0 if x < 6.5
    df['db_bin'] = db_bin_cats.rename_categories(['Non-Diabetes', 'Diabetes'

    # Multi-class categorization
    db_multi_cats = pd.Categorical(df[a1c_column].apply(lambda x: 0 if x < 5
    df['db_multi'] = db_multi_cats.rename_categories(['Healthy', 'Pre-Diabet

    # Drop 'db_stat' and 'status' columns if they exist
    df.drop(columns=['pre_dm', 'dm_t2'], errors='ignore', inplace=True)

# Apply the function to create a new column for binary and multi-class categ
categorize_diabetes_and_clean(df, 'a1c')
```

In [ ]:
```
# Function for blood pressure categorization
def categorize_blood_pressure_and_clean(df, sbp_column, dbp_column):
    """
    Categorizes blood pressure readings into multi-class labels in the DataF
    and removes any redundant or irrelevant columns. The categories are trea

    Parameters:
    - df: DataFrame containing blood pressure data.
    - sbp: The column name in the DataFrame that contains systolic blood pre
    - dbp: The column name in the DataFrame that contains diastolic blood pr

    The function modifies the DataFrame in-place and adds a new categorical
    - blood_pressure: Multi-class categorization (Normal, Elevated, Hyperter
    Optionally drops any columns named 'old_bp' and 'previous_bp' if present
    """
    # Define multi-class categorization logic
    def multi_class_bp(sbp, dbp):
        if sbp < 120 and dbp < 80:
            return 'Normal'
        elif 120 <= sbp <= 129 and dbp < 80:
            return 'Elevated'
        elif (130 <= sbp <= 139) or (80 <= dbp <= 89):
            return 'Hypertension Stage 1'
        elif sbp >= 140 or dbp >= 90:
            return 'Hypertension Stage 2'
        elif sbp > 180 or dbp > 120:
            return 'Hypertensive Crisis'
        else:
            return 'Undetermined'

    # Apply categorization
    df['blood_pressure'] = pd.Categorical(df.apply(lambda row: multi_class_b
                                      categories=['Normal', 'Elevated',
                                      ordered=True)

    # Drop columns if they exist
    df.drop(columns=['old_bp', 'previous_bp'], errors='ignore', inplace=True

categorize_blood_pressure_and_clean(df, 'sbp', 'dbp')
```

In [ ]:

```
# Function for BMI categorization
def categorize_bmi_and_clean(df, bmi_column):
    """
    Categorizes BMI values into multi-class labels in the DataFrame,
    and removes any redundant or irrelevant columns. The categories are trea

    Parameters:
    - df: DataFrame containing BMI data.
    - bmi_column: The column name in the DataFrame that contains BMI values.

    The function modifies the DataFrame in-place and adds a new categorical
    - body_type: Multi-class categorization (Underweight, Normal weight, Ove

    Optionally drops any columns named 'old_bmi' and 'previous_bmi' if prese
    """
    # Define multi-class categorization logic
    def multi_class_bmi(bmi):
        if bmi < 18.5:
            return 'Underweight'
        elif 18.5 <= bmi <= 24.9:
            return 'Normal weight'
        elif 25 <= bmi <= 29.9:
            return 'Overweight'
        else:  # BMI of 30 and over
            return 'Obesity'

    # Apply categorization
    df['body_type'] = pd.Categorical(df[bmi_column].apply(multi_class_bmi),
                                          categories=['Underweight', 'Normal weig
                                          ordered=True)

    # Drop columns if they exist
    df.drop(columns=['old_bmi', 'previous_bmi'], errors='ignore', inplace=Tr

categorize_bmi_and_clean(df, 'bmi')
```

In [ ]:

```
# mapping categorical values as ordinal for interpretability

db_multi_mapping = {'Healthy': 0, 'Pre-Diabetes': 1, 'Diabetes': 2}

db_bin_mapping = {'Non-Diabetes': 0, 'Diabetes': 1}

blood_pressure_mapping = {
    'Normal': 0, 'Elevated': 1, 'Hypertension Stage 1': 2,
    'Hypertension Stage 2': 3, 'Hypertensive Crisis': 4, 'Undetermined': 5
}
body_type_mapping = {
    'Underweight': 0, 'Normal weight': 1, 'Overweight': 2, 'Obesity': 3
}

# Apply the mappings
df['blood_pressure_c'] = df['blood_pressure'].map(blood_pressure_mapping)
df['body_type_c'] = df['body_type'].map(body_type_mapping)
df['db_multi_c'] = df['db_multi'].map(db_multi_mapping)
df['db_bin_c'] = df['db_bin'].map(db_bin_mapping)

# Convert these columns to numeric to ensure they are treated as such
df['blood_pressure_c'] = pd.to_numeric(df['blood_pressure_c'], errors='coerc
df['body_type_c'] = pd.to_numeric(df['body_type_c'], errors='coerce')
df['db_multi_c'] = pd.to_numeric(df['db_multi_c'], errors='coerce')
df['db_bin_c'] = pd.to_numeric(df['db_bin_c'], errors='coerce')
```

In [ ]:

```
df_follow = df[df['visit'] == 'follow_up']
df_init = df[df['visit'] == 'initial']
df_init.subject.nunique(), df_follow.subject.nunique()
```

Out[ ]:

```
(547, 212)
```

In [ ]:

```
# check how many subjects will be removed due to bmi < 25
df_init.query('bmi < 25').subject.nunique()
```

Out[ ]:

```
7
```

In [ ]:

```
# create function for research questions

def create_research_df(df):
    # Filter for initial visits with BMI >= 25
    initial_visits = df[(df['bmi'] >= 25) & (df['visit'] == 'initial')]

    # Get the subject IDs of those with initial BMI >= 25
    subjects_with_high_bmi = initial_visits['subject'].unique()

    # Filter for follow-up visits for those subjects
    follow_up_visits = df[(df['visit'] == 'follow_up') & (df['subject'].isir

    # Ensure that each subject with an initial visit also has a follow-up vi
    subjects_with_follow_up = follow_up_visits['subject'].unique()

    # Filter the initial visits to include only those subjects who have foll
    initial_visits_with_follow_up = initial_visits[initial_visits['subject']

    # Combine the initial visits and the selected follow-up visits
    research_df = pd.concat([initial_visits_with_follow_up, follow_up_visits

    return research_df

# Apply the function to the original data
df_research = create_research_df(df)
```

In [ ]:

```
df_research.value_counts('visit')
```

Out[ ]:

```
visit
follow_up    206
initial      206
Name: count, dtype: int64
```

Exploratory Data Analysis

```
1. df_init = all initial visits
2. df_research = research questions on pre-post
3. df = use for feature importance / cor matrix / preprocess
for modeling
```

In [ ]:

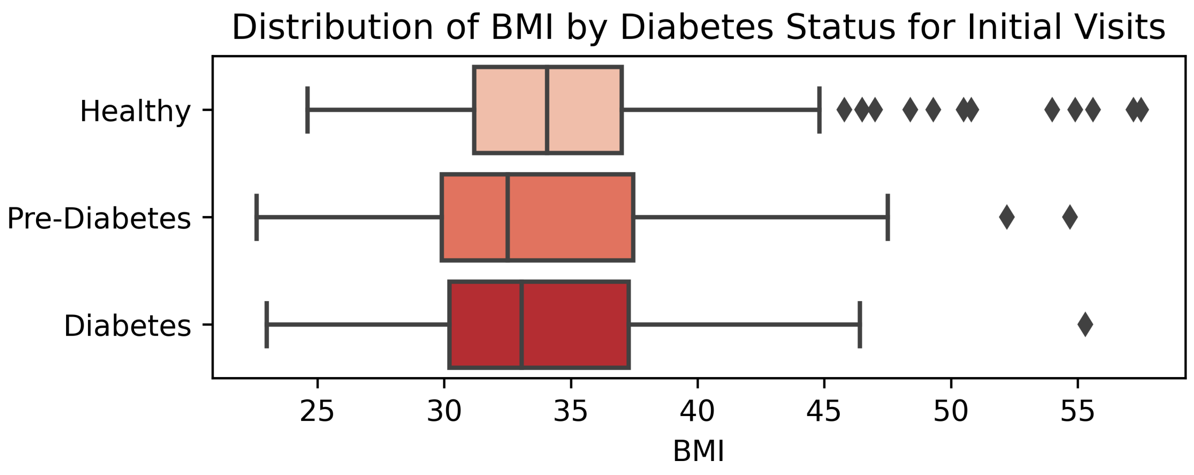
```
df_init.drop(['date_of_visit', 'db_multi_c', 'db_bin_c'], axis=1).describe()
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
subject	547.0	282.334552	163.726419	1.00	141.5	282.0	421.5	574.0
wt	547.0	241.524680	40.885716	153.00	213.5	237.0	264.0	397.0
sbp	546.0	128.679487	13.184012	15.00	121.0	129.0	136.0	200.0
bmi	541.0	34.299815	5.565896	22.60	30.6	33.3	37.2	57.5
a1c	533.0	5.855197	1.194021	0.52	5.3	5.6	6.0	14.1
trig	531.0	199.990584	145.131411	25.00	107.0	159.0	243.5	1272.0
hdl	530.0	42.920755	19.847275	18.00	35.0	41.0	48.0	428.0
total_chol	531.0	197.531073	46.991734	52.00	165.0	197.0	224.0	419.0
ldl	488.0	116.963320	40.785945	0.00	91.0	116.5	143.0	273.0
blood_pressure_c	546.0	1.815018	0.863444	0.00	2.0	2.0	2.0	3.0
body_type_c	547.0	2.797075	0.433225	1.00	3.0	3.0	3.0	3.0

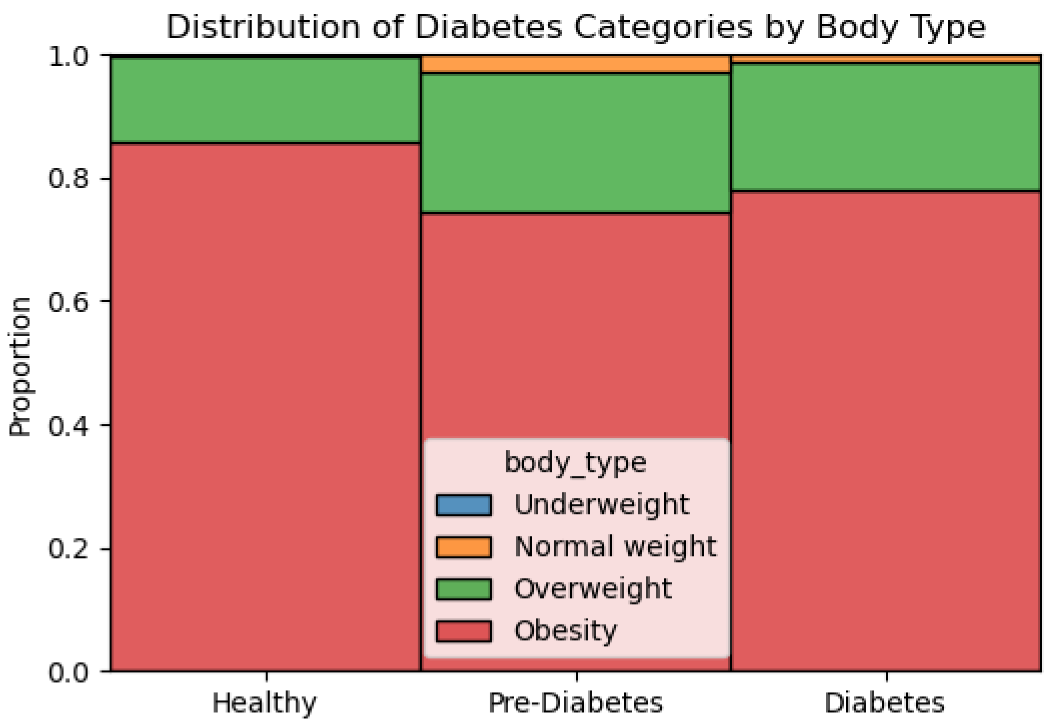
In [ ]:

```
# Create the horizontal box plot using seaborn
plt.figure(figsize=(6, 2), dpi=600)
sns.boxplot(y="db_multi", x="bmi", data=df_init, orient="h", palette="Reds")

# Set the plot title and labels
plt.title("Distribution of BMI by Diabetes Status for Initial Visits")
plt.xlabel("BMI")
plt.ylabel("")
plt.show()
```



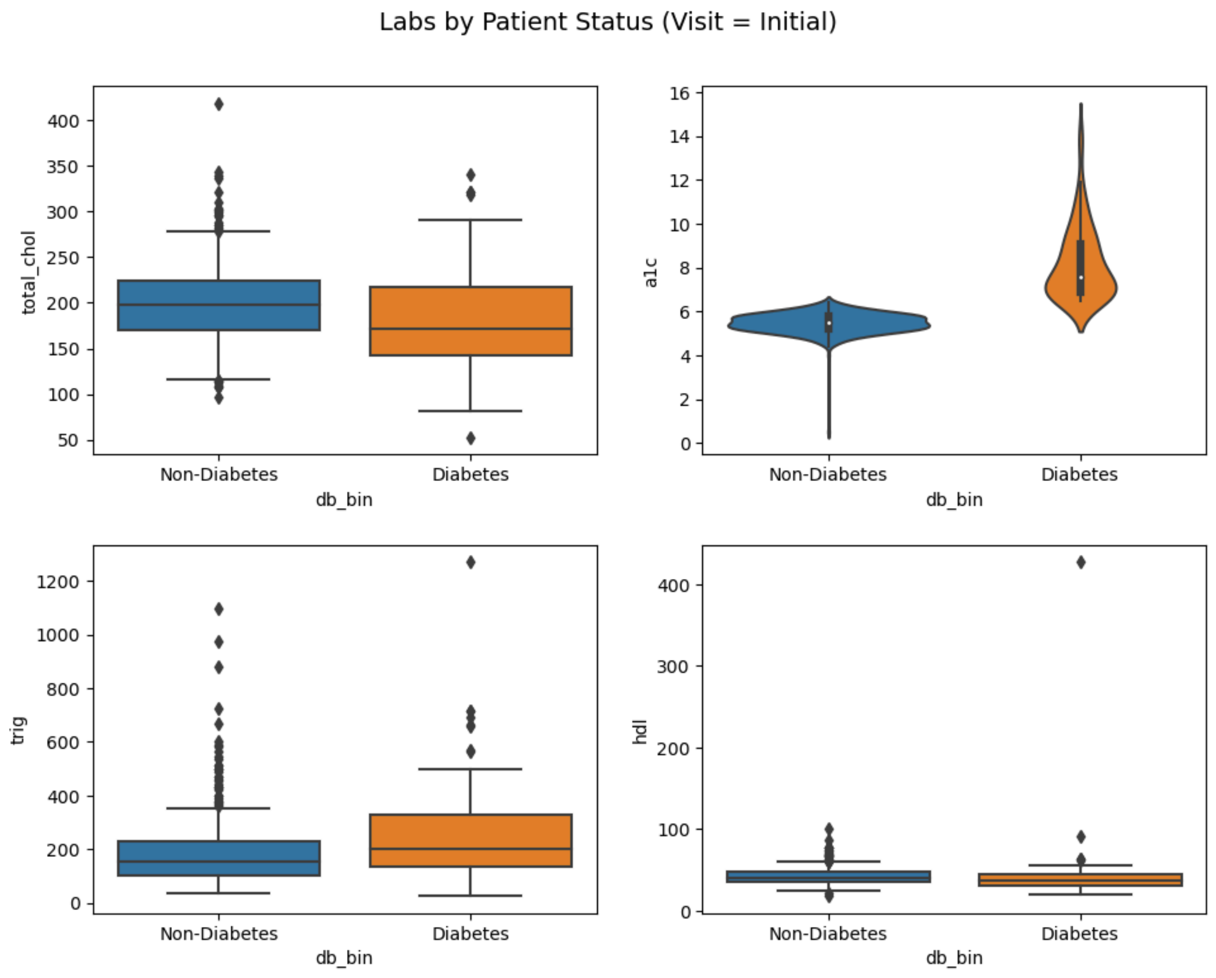
In [ ]:

```
# Create the histogram with frequencies
plt.figure(figsize=(6, 4))
sns.histplot(data=df_init, x="db_multi", hue="body_type", multiple="fill", s

# Set the plot title and labels
plt.title("Distribution of Diabetes Categories by Body Type",fontsize=12)
plt.xlabel("")
plt.ylabel("Proportion")
plt.show()
```

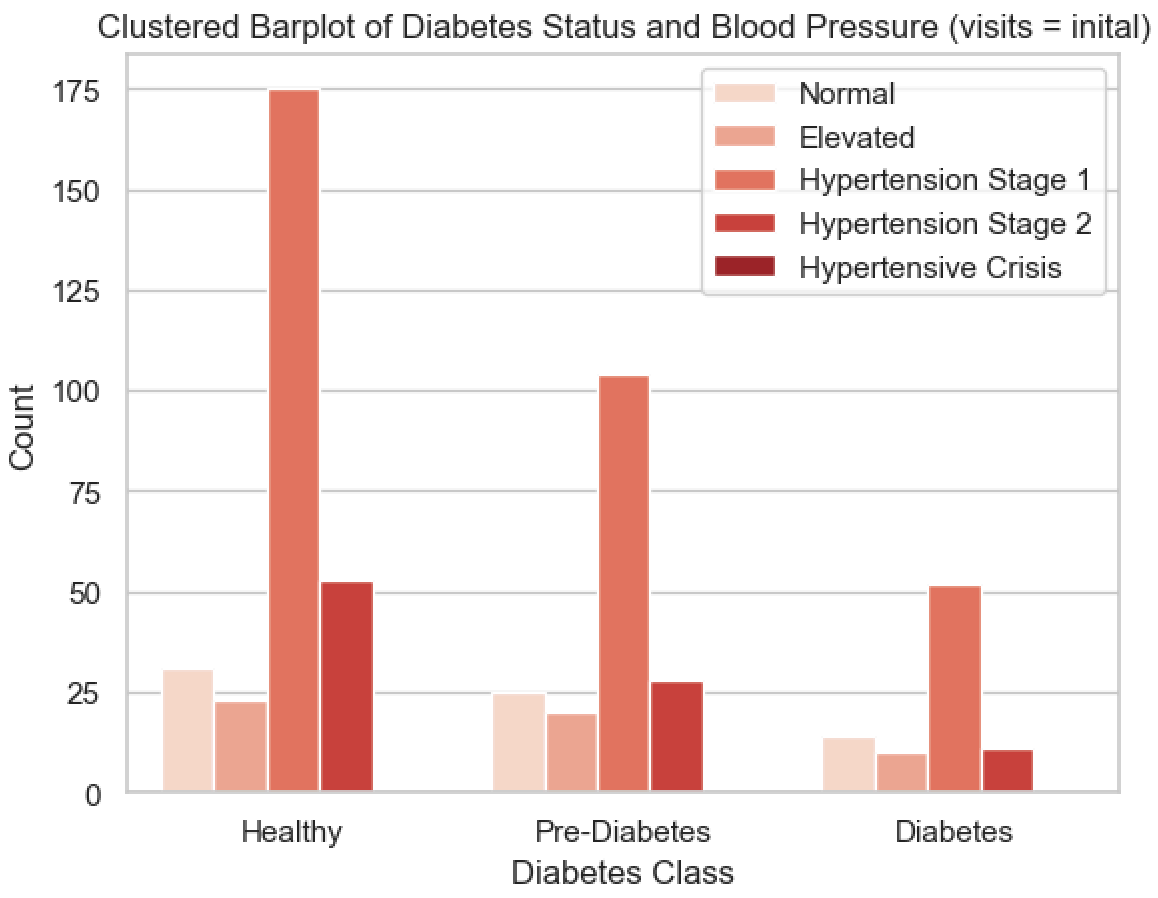


In [ ]:

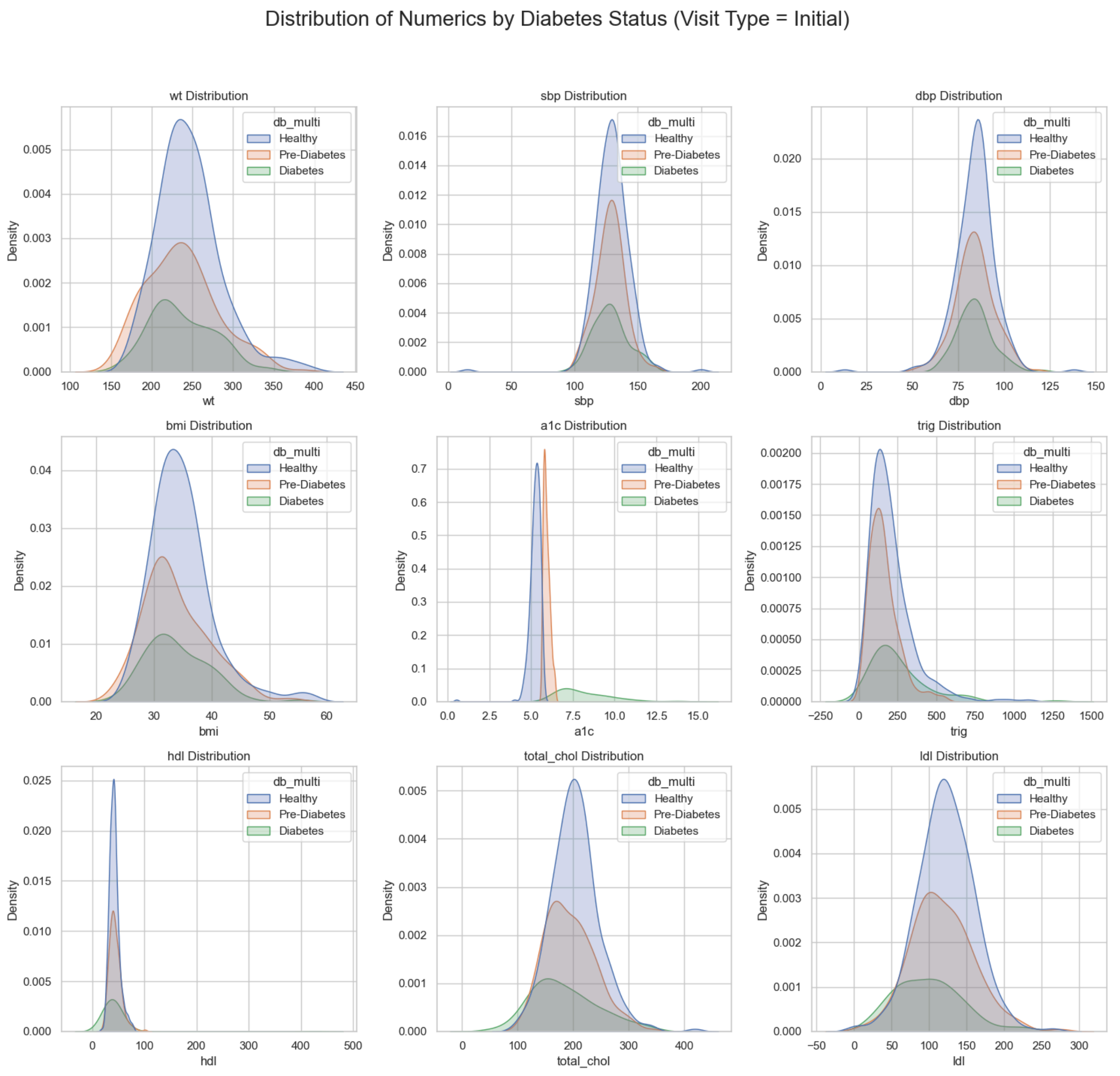
```
# Inital Visit Overview

fig, axes = plt.subplots(2, 2, figsize= (10, 8))

fig.suptitle('Labs by Patient Status (Visit = Initial)', fontsize=14)
sns.boxplot(ax=axes[0, 0], data=df_init, x='db_bin', y='total_chol', whis=1)
sns.violinplot(ax=axes[0, 1], data=df_init, x='db_bin', y='a1c', whis=1)
sns.boxplot(ax=axes[1, 0], data=df_init, x='db_bin', y='trig', whis=1)
sns.boxplot(ax=axes[1, 1], data=df_init, x='db_bin', y='hdl', whis=1)


plt.tight_layout(pad=2)
plt.show()
```



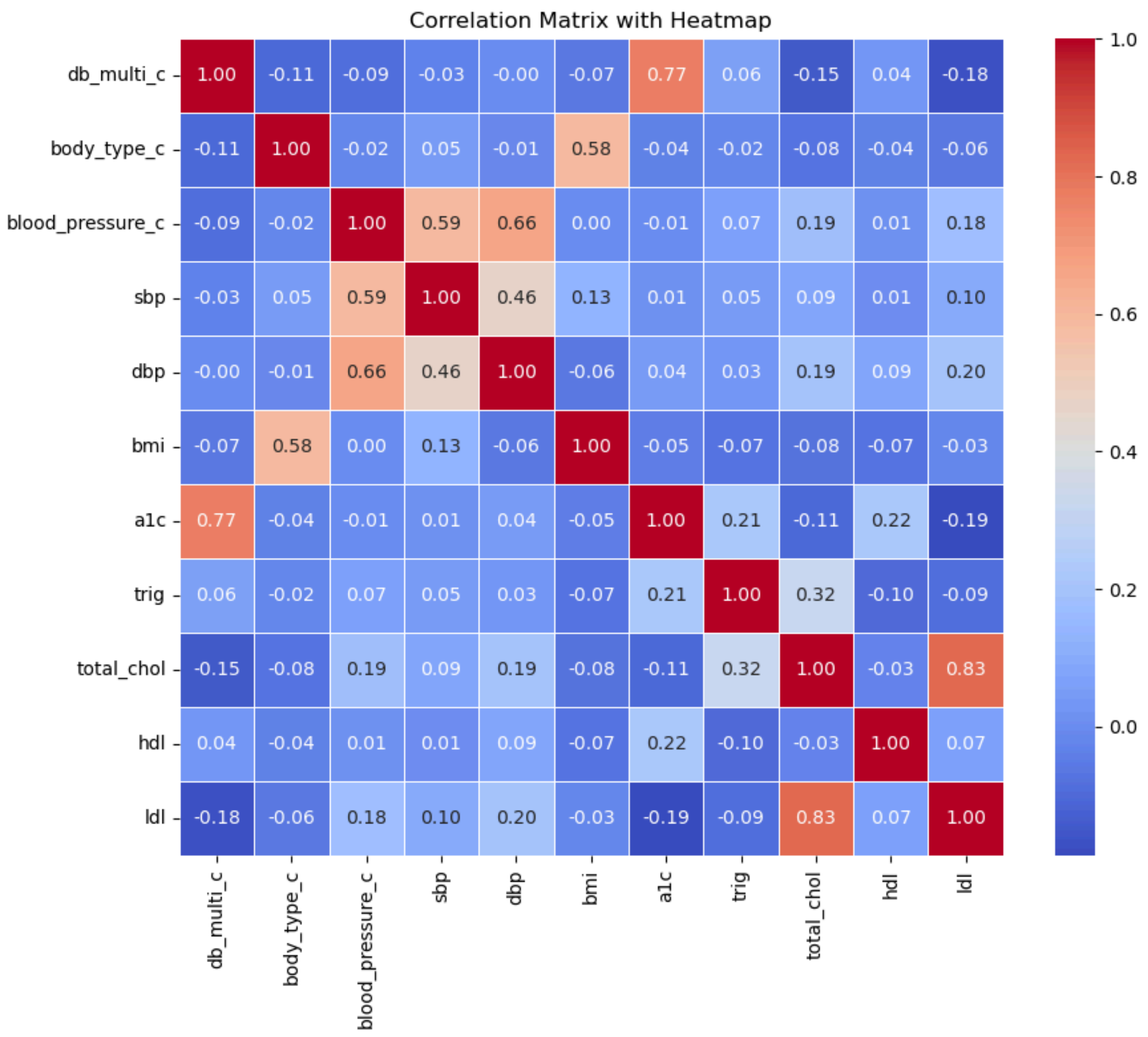
```
In [ ]: # Create the count plot
        sns.countplot(data=df_init, x='db_multi', hue='blood_pressure',
                      palette='Reds', hue_order=['Normal', 'Elevated', 'Hypertensior

        # Set plot title and labels
        plt.title('Clustered Barplot of Diabetes Status and Blood Pressure (visits =
        plt.legend(loc='upper right')
        plt.xlabel('Diabetes Class')
        plt.ylabel('Count')

        # Show the plot
        plt.show()
```



```
In [ ]: # Define numeric columns and subplot grid dimensions
        numeric_cols = ['wt', 'sbp', 'dbp', 'bmi', 'a1c', 'trig', 'hdl', 'total_chol
        n_cols = 3
        n_rows = (len(numeric_cols) + n_cols - 1) // n_cols

        # Create figure and axes for subplots
        fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 5 * n_rows))
        fig.suptitle('Distribution of Numerics by Diabetes Status (Visit Type = Init

        # Flatten axes array for easy iterating
        axes = axes.flatten()

        # Plot KDE for each numeric column with a hue
        for i, col in enumerate(numeric_cols):
            sns.kdeplot(data=df_init, x=col, hue="db_multi", ax=axes[i], fill=True)
            axes[i].set_title(f'{col} Distribution')
            axes[i].set_xlabel(col)
            axes[i].set_ylabel('Density')

        # Hide any unused axes if the number of numeric columns isn't a perfect mult
        for ax in axes[len(numeric_cols):]:
            ax.set_visible(False)

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to make room for
        plt.show()
```



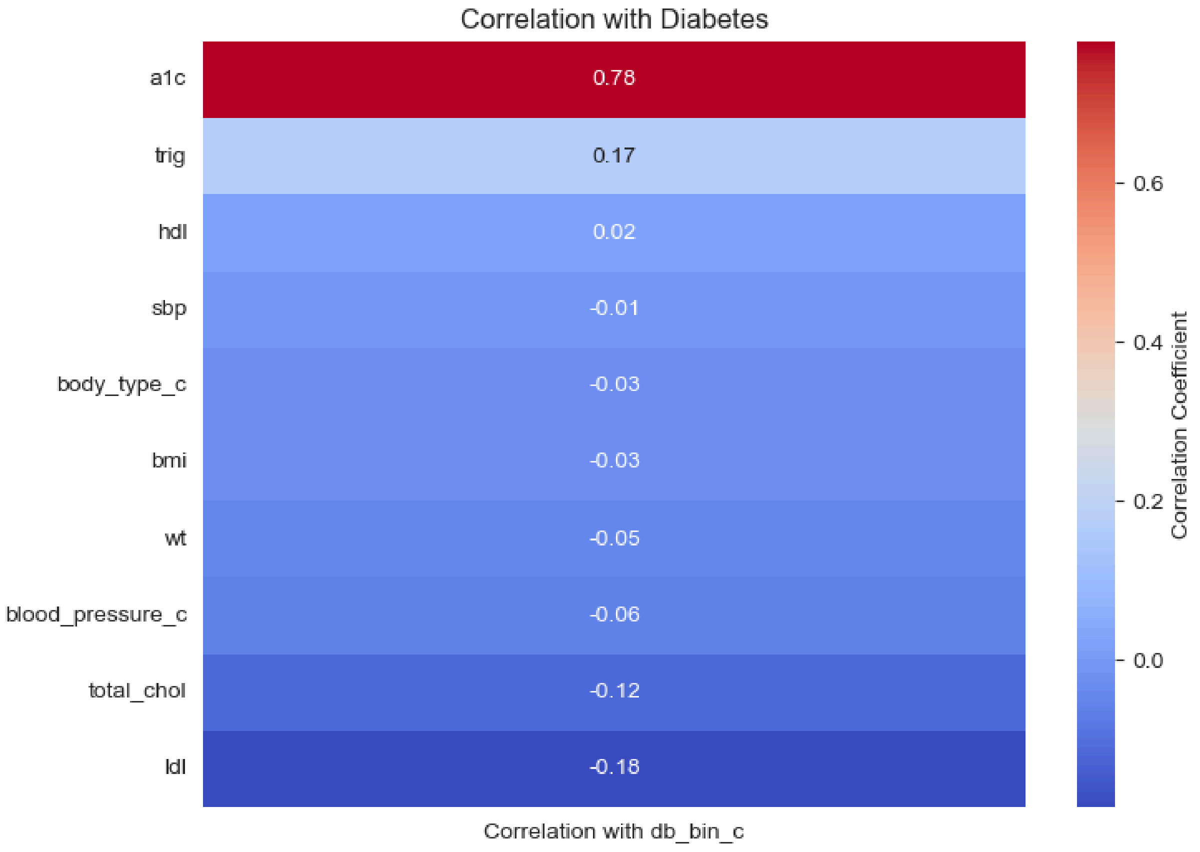
```
In [ ]: # Select numeric columns for correlation matrix (including the newly coded c
        numeric_cols = ['db_multi_c','body_type_c','blood_pressure_c', 'sbp', 'dbp',

        # Calculate the correlation matrix
        corr = df_init[numeric_cols].corr()


        # Plot the heatmap of the correlation matrix using seaborn
        plt.figure(figsize=(10, 8))
        sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
        plt.title('Correlation Matrix with Heatmap')
        plt.show()
```



	db_multi_c	body_type_c	blood_pressure_c	sbp	dbp	bmi	a1c	trig	total_chol	hdl	ldl
db_multi_c	1.00	-0.11	-0.09	-0.03	-0.00	-0.07	0.77	0.06	-0.15	0.04	-0.18
body_type_c	-0.11	1.00	-0.02	0.05	-0.01	0.58	-0.04	-0.02	-0.08	-0.04	-0.06
blood_pressure_c	-0.09	-0.02	1.00	0.59	0.66	0.00	-0.01	0.07	0.19	0.01	0.18
sbp	-0.03	0.05	0.59	1.00	0.46	0.13	0.01	0.05	0.09	0.01	0.10
dbp	-0.00	-0.01	0.66	0.46	1.00	-0.06	0.04	0.03	0.19	0.09	0.20
bmi	-0.07	0.58	0.00	0.13	-0.06	1.00	-0.05	-0.07	-0.08	-0.07	-0.03
a1c	0.77	-0.04	-0.01	0.01	0.04	-0.05	1.00	0.21	-0.11	0.22	-0.19
trig	0.06	-0.02	0.07	0.05	0.03	-0.07	0.21	1.00	0.32	-0.10	-0.09
total_chol	-0.15	-0.08	0.19	0.09	0.19	-0.08	-0.11	0.32	1.00	-0.03	0.83
hdl	0.04	-0.04	0.01	0.01	0.09	-0.07	0.22	-0.10	-0.03	1.00	0.07
ldl	-0.18	-0.06	0.18	0.10	0.20	-0.03	-0.19	-0.09	0.83	0.07	1.00

In [ ]:
```python
# Select only numeric columns from df_init
df_numeric = df_init.select_dtypes(include=[np.number])

# Remove 'db_multi_c' column if it exists in df_numeric to ensure it's not i
if 'db_multi_c' in df_numeric.columns:
    df_numeric = df_numeric.drop(columns=['db_multi_c','subject'])

# Compute the correlation matrix
correlation_matrix = df_numeric.corr()

# Extract the correlation of 'db_bin_c' and drop its self-correlation
target_corr = correlation_matrix['db_bin_c'].drop('db_bin_c')

# Convert to DataFrame for heatmap compatibility and rename column for clari
target_corr_df = target_corr.to_frame(name='Correlation with db_bin_c')

# Sort correlation values in descending order for better visualization
target_corr_sorted_df = target_corr_df.sort_values(by='Correlation with db_b

# Plotting setup
sns.set_style("white")
sns.set_palette("PuBuGn_d")
```

```
plt.figure(figsize=(8, 6))  # Adjust size to accommodate all variables if ne
sns.heatmap(target_corr_sorted_df, cmap="coolwarm", annot=True, fmt='.2f', c
plt.title('Correlation with Diabetes')
plt.show()
```



Research Outcomes

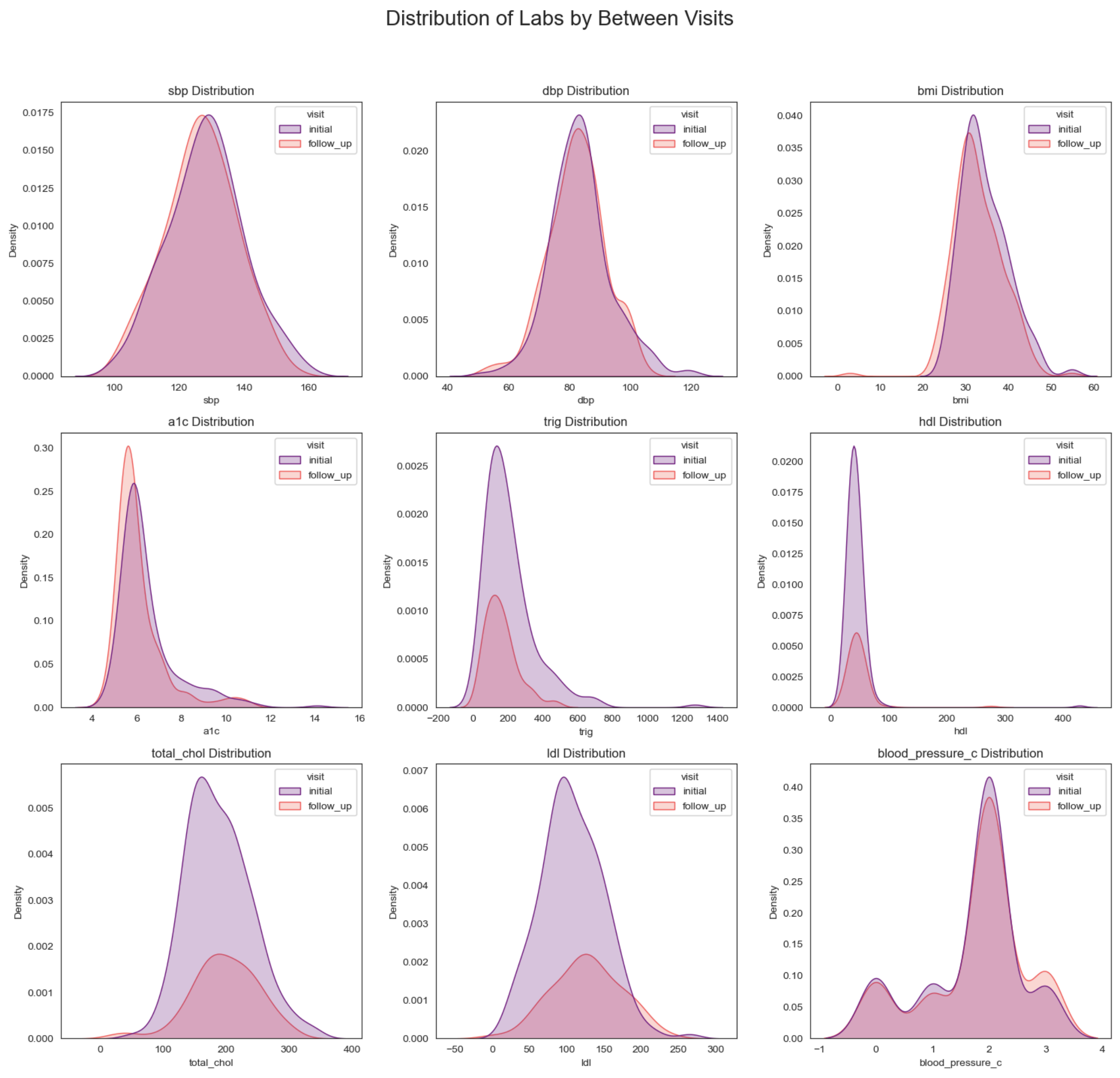
Outcomes we hope to achieve after intervention over a 6 - 12 month follow up

Weight loss: Evaluate percent of patients that achieved 5%; Evaluate percent of patients that achieved 10%

A1c reduction: Evaluate percent of patients that achieved HbA1c less than 6.5%; Evaluate percent of patients that reduced HbA1c by more than 1%

```
In [ ]: # Define numeric columns and subplot grid dimensions
numeric_cols = ['sbp', 'dbp', 'bmi', 'a1c', 'trig', 'hdl', 'total_chol', 'lc
n_cols = 3
n_rows = (len(numeric_cols) + n_cols - 1) // n_cols

# Create figure and axes for subplots
fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 5 * n_rows))
fig.suptitle('Distribution of Labs by Between Visits', fontsize=20)

# Flatten axes array for easy iterating
axes = axes.flatten()

# Define a color palette
```

```
palette = "magma"

# Plot KDE for each numeric column with a hue
for i, col in enumerate(numeric_cols):
    sns.kdeplot(data=df_research, x=col, hue="visit", ax=axes[i], fill=True,
    axes[i].set_title(f'{col} Distribution')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Density')

# Hide any unused axes if the number of numeric columns isn't a perfect mult
for ax in axes[len(numeric_cols):]:
    ax.set_visible(False)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])  # Adjust layout to make room for
plt.show()
```



```
In [ ]: from scipy.stats import ttest_rel

        # Perform the paired t-test between initial and follow-up bmi

        # Filter DataFrame for initial and follow-up visits
```

```
df_initial = df_research[df_research['visit'] == 'initial'][['subject', 'bmi
df_follow_up = df_research[df_research['visit'] == 'follow_up'][['subject',

# Merge the initial and follow-up DataFrames on subject
df_paired = pd.merge(df_initial, df_follow_up, on='subject', suffixes=('_ini

# Drop any rows with NaN values in 'bmi'
df_paired.dropna(subset=['bmi_initial', 'bmi_follow_up'], inplace=True)

# Perform the paired t-test
t_stat, p_value = ttest_rel(df_paired['bmi_initial'], df_paired['bmi_follow_

# Calculate Cohen's d
mean_diff = np.mean(df_paired['bmi_initial'] - df_paired['bmi_follow_up'])
std_diff = np.std(df_paired['bmi_initial'] - df_paired['bmi_follow_up'], ddo
cohen_d = mean_diff / std_diff

t_stat, p_value, cohen_d
```

Out[ ]:

```
(6.678290304440181, 2.4848629314932356e-10, 0.47824230201629464)
```

In [ ]:

```
# Filter DataFrame for initial and follow-up visits for A1C similarly to BMI
df_initial_a1c = df_research[df_research['visit'] == 'initial'][['subject',
df_follow_up_a1c = df_research[df_research['visit'] == 'follow_up'][['subjec

# Merge the initial and follow-up A1C DataFrames on subject
df_paired_a1c = pd.merge(df_initial_a1c, df_follow_up_a1c, on='subject', suf

# Drop any rows with NaN values in 'a1c'
df_paired_a1c.dropna(subset=['a1c_initial', 'a1c_follow_up'], inplace=True)

# Merge the BMI and A1C change DataFrames to analyze together
df_changes = pd.merge(df_paired[['subject', 'bmi_initial', 'bmi_follow_up']]
                      df_paired_a1c[['subject', 'a1c_initial', 'a1c_follow_u

# Calculate changes in BMI and A1C
df_changes['bmi_change'] = df_changes['bmi_follow_up'] - df_changes['bmi_ini
df_changes['a1c_change'] = df_changes['a1c_follow_up'] - df_changes['a1c_ini

# Calculate the Pearson correlation between changes in BMI and changes in A1
correlation_result = df_changes[['bmi_change', 'a1c_change']].corr().iloc[0,
correlation_result
```

Out[ ]:

```
0.0909956793465215
```

In [ ]:

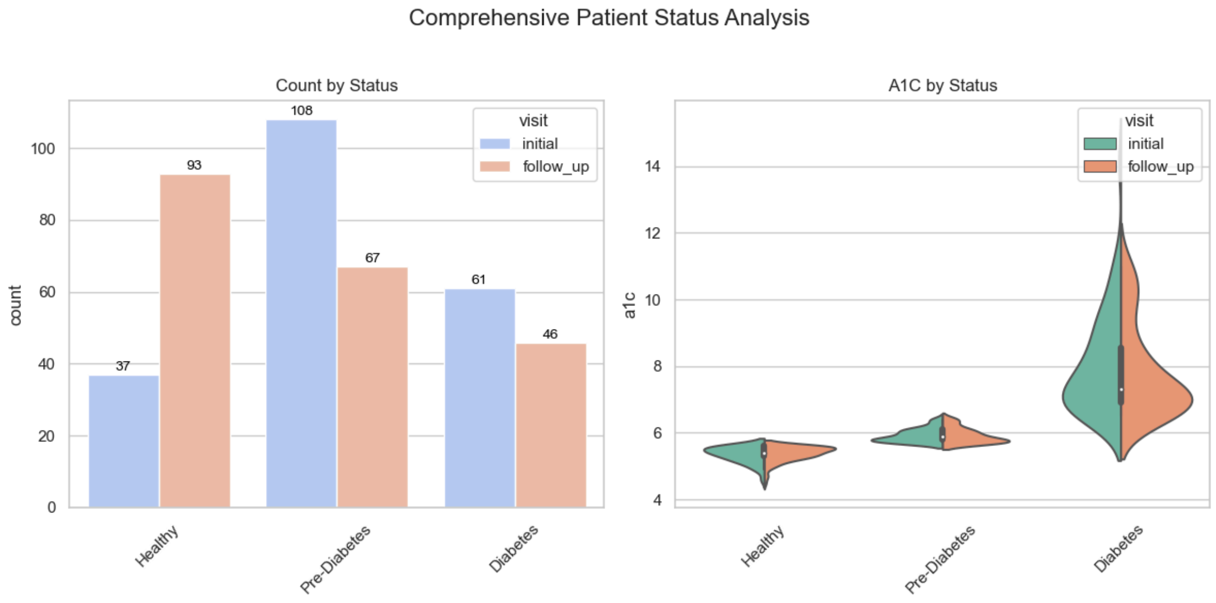
```
# Perform the paired t-test for A1C between initial and follow-up visits
t_stat_a1c, p_value_a1c = ttest_rel(df_paired_a1c['a1c_initial'], df_paired_

# Calculate the mean difference and standard deviation for Cohen's d calcula
mean_diff_a1c = np.mean(df_paired_a1c['a1c_initial'] - df_paired_a1c['a1c_fo
std_diff_a1c = np.std(df_paired_a1c['a1c_initial'] - df_paired_a1c['a1c_foll
cohen_d_a1c = mean_diff_a1c / std_diff_a1c

t_stat_a1c, p_value_a1c, cohen_d_a1c
```

Out[ ]:
```
(4.335726192019578, 2.2961707994308436e-05, 0.3050606303327793)
```

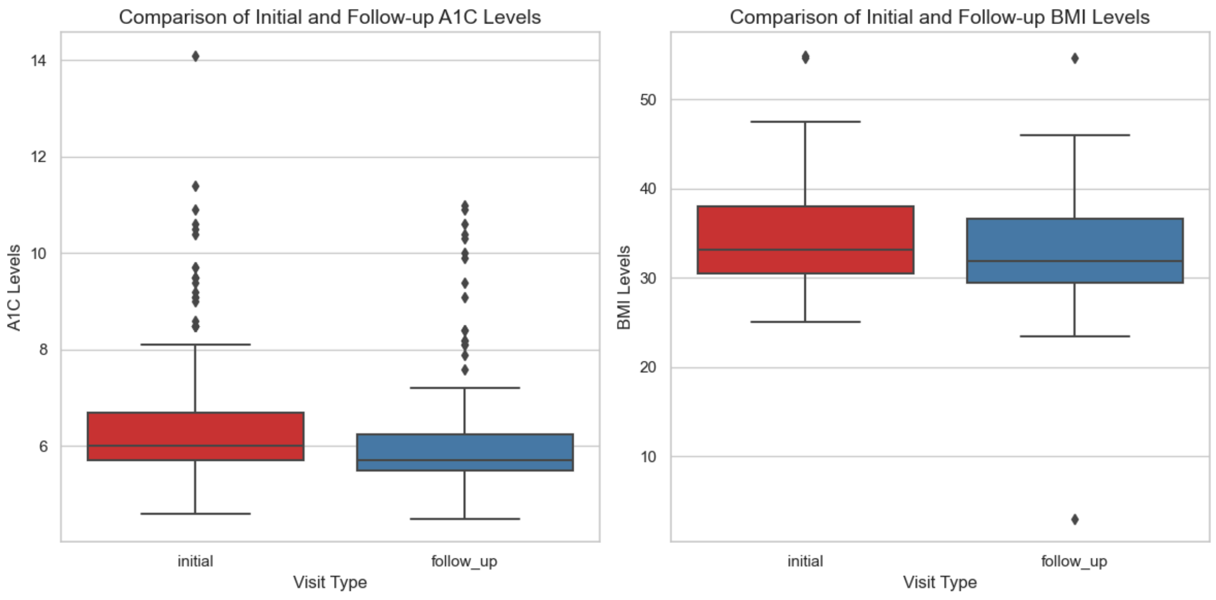
In [ ]:
```
#calculating the percentage of patients achieving significant changes

# Filter and merge data for initial and follow-up visits directly
df_initial = df_research[df_research['visit'] == 'initial'][['subject', 'wt'
df_follow_up = df_research[df_research['visit'] == 'follow_up'][['subject',
df_paired = pd.merge(df_initial, df_follow_up, on='subject')

# Calculate changes in weight and A1C
df_paired['wt_change'] = df_paired['wt_follow_up'] - df_paired['wt_initial']
df_paired['a1c_change'] = pd.to_numeric(df_paired['a1c_follow_up'], errors='

# Determine significant weight loss and A1C reduction
df_paired['significant_wt_loss'] = (df_paired['wt_change'] / df_paired['wt_i
df_paired['significant_10_wt_loss'] = (df_paired['wt_change'] / df_paired['w
df_paired['significant_a1c_reduction'] = df_paired['a1c_change'] <= -0.5
df_paired['significant_1_a1c_reduction'] = df_paired['a1c_change'] <= -1.0

# Calculate the percentage of patients achieving significant changes
percent_significant_wt_loss = 100 * df_paired['significant_wt_loss'].mean()
percent_significant_10_wt_loss = 100 * df_paired['significant_10_wt_loss'].m
percent_significant_a1c_reduction = 100 * df_paired['significant_a1c_reducti
percent_significant_1_a1c_reduction = 100 * df_paired['significant_1_a1c_red

(percent_significant_wt_loss, percent_significant_10_wt_loss, percent_signif
```

Out[ ]:
```
(41.262135922330096,
 13.106796116504855,
 32.038834951456316,
 13.592233009708737)
```

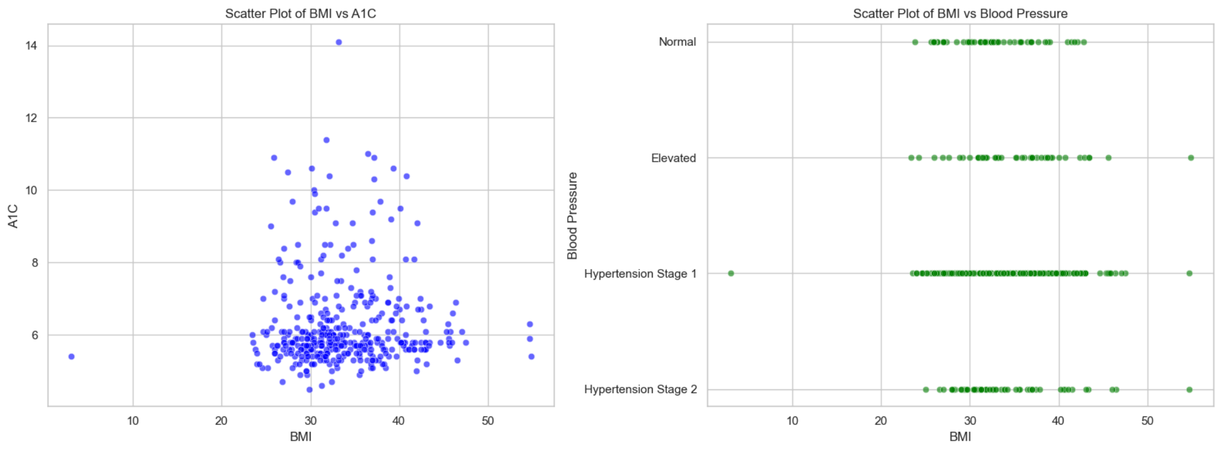
In [ ]:
```
# Setup for the figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
fig.suptitle('Comprehensive Patient Status Analysis', fontsize=16)

# First subplot - Count plot for 'db_multi'
countplot1 = sns.countplot(ax=axes[0], data=df_research, x='db_multi', hue='
axes[0].set_title('Count by Status')
axes[0].set_xlabel('')
axes[0].tick_params(axis='x', rotation=45)
for p in countplot1.patches:
    countplot1.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width()
                        ha='center', va='center', fontsize=10, color='black'
                        textcoords='offset points')

# Second subplot - Violin plot for 'a1c' by 'db_multi'
sns.violinplot(ax=axes[1], data=df_research, x='db_multi', y='a1c', hue='vis
axes[1].set_title('A1C by Status')
axes[1].set_xlabel('')
axes[1].tick_params(axis='x', rotation=45)

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.96])  # Adjust the layout to make room for
plt.show()
```



```
In [ ]: # Plotting the boxplot for A1C and BMI
        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))  # Use the same

        # First subplot for A1C
        sns.boxplot(x='visit', y='a1c', data=df_research, palette="Set1", ax=axes[0]
        axes[0].set_title('Comparison of Initial and Follow-up A1C Levels', fontsize
        axes[0].set_xlabel('Visit Type')
        axes[0].set_ylabel('A1C Levels')

        # Second subplot for BMI
        sns.boxplot(x='visit', y='bmi', data=df_research, palette="Set1", ax=axes[1]
        axes[1].set_title('Comparison of Initial and Follow-up BMI Levels', fontsize
        axes[1].set_xlabel('Visit Type')
        axes[1].set_ylabel('BMI Levels')

        # Adjust layout for better spacing
        plt.tight_layout()
        plt.show()
```



```
In [ ]: import seaborn as sns
        import matplotlib.pyplot as plt

        # Set the aesthetic style of the plots
        sns.set(style="whitegrid")

        # Create a figure with subplots
        fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))

        # Plot BMI vs A1C on the first subplot
        sns.scatterplot(ax=axes[0], x='bmi', y='a1c', data=df_research, color='blue'
        axes[0].set_title('Scatter Plot of BMI vs A1C')
        axes[0].set_xlabel('BMI')
        axes[0].set_ylabel('A1C')

        # Plot BMI vs Blood Pressure on the second subplot
        sns.scatterplot(ax=axes[1], x='bmi', y='blood_pressure', data=df_research, c
        axes[1].set_title('Scatter Plot of BMI vs Blood Pressure')
        axes[1].set_xlabel('BMI')
        axes[1].set_ylabel('Blood Pressure')

        # Adjust layout
        plt.tight_layout()

        # Show the plots
        plt.show()
```



Hypothesis

Empowering the veteran with knowledge and simple assessments allows clinicians to guide veterans in making scientifically based nutritional and behavioral changes, which will greatly improve their overall health and outcomes in the future.

Results BMI Reduction Analysis:

The paired t-test analysis comparing initial and follow-up BMI values of subjects revealed a significant difference, with a t-statistic of 6.678 and a p-value less than 0.05. This low p-value indicates that the observed difference is statistically significant, allowing us to reject the null hypothesis confidently. Additionally, Cohen's d, a measure of effect size, was calculated to be 0.478, suggesting a medium effect size. This implies that the change in BMI over the observed period is not only statistically significant but also practically meaningful.

A1C Level Analysis:

The paired t-test analysis comparing initial and follow-up A1C values of subjects revealed a significant difference, with a t-statistic of 4.336 and a p-value less than 0.05. This low p-value indicates that the observed difference is statistically significant, allowing us to reject the null hypothesis confidently. Cohen's d, was calculated to be 0.305, suggesting a small to medium effect size.

Significant Weight Loss (at least 5%): Percentage of Patients: 41.26% This indicates that 41.26% of the patients achieved a weight loss of 5% or more between their initial and follow-up visits.

Significant Weight Loss (at least 10%): Percentage of Patients: 13.11% This indicates that 13.11% of the patients achieved a weight loss of 10% or more between their initial and follow-up visits.

Significant A1C Reduction (at least 0.5 units): Percentage of Patients: 32.04% This indicates that 32.04% of the patients achieved an A1C reduction of 0.5 units or more between their initial and follow-up visits.

Significant A1C Reduction (at least 1.0 unit): Percentage of Patients: 13.59% This indicates that 13.59% of the patients achieved an A1C reduction of 1.0 unit or more between their initial and follow-up visits.

These results highlight that a considerable portion of the patients experienced significant improvements in both weight and A1C levels over the observed period. Notably, over 40% of the patients achieved a meaningful weight loss of at least 5%, and approximately one-third saw a reduction of 0.5 units or more in their A1C levels. However, fewer patients achieved the more substantial targets of 10% weight loss and 1.0 unit A1C reduction.

Classifcation Model and Evaluation

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold
from sklearn.metrics import accuracy_score, classification_report, roc_curve

# List of columns to keep
```

```
columns = ['sbp', 'dbp', 'bmi', 'trig', 'hdl', 'total_chol', 'ldl','db_bin_c

# Selecting the columns from the DataFrame
df_mod = df[columns]
```

Regression Model and Evaluation