PA1_Classification

February 4, 2026

1 INITIALIZERS

[1]:
```
### IMPORTS
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

[2]:
```
### A HELPER FUNCTION TO DISPLAY SECTION TITLE
def print_section(title):
    print(f'{"="*60}\n{title}\n{"="*60}')

### SEE ALL COLUMNS
pd.set_option('display.max_columns', None)
```

2 DATA LOADING

[3]:
```
### LOAD DATA
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

3 INITIAL DATA INSPECTION

3.1 GET OVERVIEW

[4]:
```
### TRAIN DATASET
print_section('TRAIN')
train.info()
display(train.head())
display(train.describe())

### TEST DATASET
print_section('TEST')
test.info()
display(test.head())
display(test.describe())
```

```
============================================================
TRAIN
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   p_id                    614 non-null    int64
 1   no_times_pregnant       614 non-null    int64
 2   glucose_concentration   614 non-null    int64
 3   blood_pressure          614 non-null    int64
 4   skin_fold_thickness     614 non-null    int64
 5   serum_insulin           614 non-null    int64
 6   bmi                     614 non-null    float64
 7   diabetes pedigree       614 non-null    float64
 8   age                     614 non-null    int64
 9   diabetes                614 non-null    int64
dtypes: float64(2), int64(8)
memory usage: 48.1 KB

   p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
0   316                  2                    112              68
1    25                 11                    143              94
2   710                  2                     93              64
3   658                  1                    120              80
4   542                  3                    128              72

   skin_fold_thickness  serum_insulin   bmi  diabetes pedigree  age  diabetes
0                   22             94  34.1              0.315   26         0
1                   33            146  36.6              0.254   51         1
2                   32            160  38.0              0.674   23         1
3                   48            200  38.9              1.162   41         0
```

```
4                   25            190  32.4             0.549   27         1

             p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
count  614.000000         614.000000             614.000000      614.000000
mean   385.773616           3.853420             120.542345       68.765472
std    223.603024           3.358126              31.252286       19.914836
min      1.000000           0.000000               0.000000        0.000000
25%    191.250000           1.000000              99.000000       62.000000
50%    387.000000           3.000000             117.000000       72.000000
75%    572.750000           6.000000             139.000000       80.000000
max    768.000000          17.000000             197.000000      114.000000

       skin_fold_thickness  serum_insulin         bmi  diabetes pedigree  \
count           614.000000     614.000000  614.000000         614.000000
mean             20.244300      79.355049   31.909935           0.466342
std              15.886083     117.709950    8.007699           0.331090
min               0.000000       0.000000    0.000000           0.078000
25%               0.000000       0.000000   27.300000           0.240250
50%              23.000000      17.000000   32.000000           0.361000
75%              32.000000     126.000000   36.600000           0.613500
max              63.000000     846.000000   59.400000           2.420000

              age    diabetes
count  614.000000  614.000000
mean    33.325733    0.348534
std     11.929569    0.476895
min     21.000000    0.000000
25%     24.000000    0.000000
50%     29.000000    0.000000
75%     41.000000    1.000000
max     81.000000    1.000000
============================================================
TEST
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 154 entries, 0 to 153
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   p_id                   154 non-null    int64
 1   no_times_pregnant      154 non-null    int64
 2   glucose_concentration  154 non-null    int64
 3   blood_pressure         154 non-null    int64
 4   skin_fold_thickness    154 non-null    int64
 5   serum_insulin          154 non-null    int64
 6   bmi                    154 non-null    float64
 7   diabetes pedigree      154 non-null    float64
 8   age                    154 non-null    int64
```

```
dtypes: float64(2), int64(7)
memory usage: 11.0 KB
```

```
   p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
0   437                 12                    140              85   
1   411                  6                    102              90   
2   639                  7                     97              76   
3   213                  7                    179              95   
4   181                  6                     87              80   

   skin_fold_thickness  serum_insulin   bmi  diabetes pedigree  age  
0                   33              0  37.4              0.244   41  
1                   39              0  35.7              0.674   28  
2                   32             91  40.9              0.871   32  
3                   31              0  34.2              0.164   60  
4                    0              0  23.2              0.084   32  
```

```
             p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
count  154.000000         154.000000             154.000000      154.000000   
mean   379.422078           3.811688             122.298701       70.461039   
std    215.338912           3.425719              34.769480       16.935917   
min      4.000000           0.000000               0.000000        0.000000   
25%    194.250000           1.000000              97.000000       64.000000   
50%    383.000000           3.000000             115.000000       72.000000   
75%    583.250000           6.000000             144.750000       78.000000   
max    738.000000          14.000000             199.000000      122.000000   

       skin_fold_thickness  serum_insulin         bmi  diabetes pedigree  \
count           154.000000     154.000000  154.000000         154.000000   
mean             21.701299      81.571429   32.322078           0.493942   
std              16.213095     105.178271    7.386724           0.332439   
min               0.000000       0.000000    0.000000           0.084000   
25%               0.000000       0.000000   28.150000           0.248000   
50%              24.500000      52.000000   32.900000           0.411500   
75%              33.000000     129.500000   36.200000           0.654750   
max              99.000000     474.000000   67.100000           2.137000   

              age  
count  154.000000  
mean    32.902597  
std     11.090106  
min     21.000000  
25%     24.000000  
50%     29.500000  
75%     40.000000  
max     66.000000  
```

3.1.1 REMARKS

- 5 non-zero numeric features (`glucose_concentration`, `blood_pressure`, `skin_fold_thickness`, `serum_insulin`, `bmi`) have invalid zeros.
- No column has any missing values.

3.2 IDENTIFY ISSUES FOR DATA CLEANING

A systematic check on potential data issues.

3.2.1 CHECK DATA TYPES

- Are all the columns in both datasets numeric (`int64` or `float64`)?
- Only numeric columns are used in building ML models.

[5]:
```
print_section('CHECK DATA TYPES')

def check_data_types(df, name):
    if df.columns.equals(df.select_dtypes(include=['float64', 'int64']).
 ↪columns):
        print(f'  {name}: All data types are numeric (float64 or int64).')
    else:
        print(f'  {name}: There are non-numeric data types.')

check_data_types(train, 'train')
check_data_types(test, 'test')
```

```
============================================================
CHECK DATA TYPES
============================================================
  train: All data types are numeric (float64 or int64).
  test: All data types are numeric (float64 or int64).
```

3.2.2 CHECK DUPLICATES

- Are there any duplicated rows?
- Are there any duplicated id?

[6]:
```
print_section('CHECK ROW DUPLICATES')

def check_row_dup(df, name):
    row_dup = df.duplicated().sum()
    if row_dup == 0:
        print(f'  {name}: {row_dup} row duplicate found.')
    else:
        print(f'  {name}: {row_dup} row duplicates found.')

check_row_dup(train, 'train')
check_row_dup(test, 'test')
```

```
============================================================
CHECK ROW DUPLICATES
============================================================
  train: 0 row duplicate found.
  test: 0 row duplicate found.
```

[7]:
```python
print_section('CHECK ID DUPLICATES')

def check_id_dup(df, name):
    total_entries = len(df)
    unique_id = df['p_id'].nunique()
    print(f"Total entries: {total_entries}")
    print(f"Unique p_id: {unique_id}")
    if total_entries == unique_id:
        print(f'  {name}: No id duplicate found. 1 unique p_id per patient.\n')
    else:
        print(f'  {name}: Multiple entries per patient detected!\n')

check_id_dup(train, 'train')
check_id_dup(test, 'test')
```

```
============================================================
CHECK ID DUPLICATES
============================================================
Total entries: 614
Unique p_id: 614
  train: No id duplicate found. 1 unique p_id per patient.

Total entries: 154
Unique p_id: 154
  test: No id duplicate found. 1 unique p_id per patient.
```

3.2.3 CHECK MISSING VALUES

- Are there any columns with missing values?
- If yes, how much do the missing values account for?

[8]:
```python
print_section('CHECK MISSING VALUES')

def check_missing(df, name):
    ### COUNT OF MISSING VALUES
    miss_cnt = df.isnull().sum()

    ### PERCENTAGE OF MISSING VALUES ROUNDED TO 2 DP
    miss_pct = (miss_cnt / len(df) * 100).round(2)

    ### DF CONTAINING COLUMNS WITH MISSING VALUES
    miss_df = pd.DataFrame({'missing_count': miss_cnt[miss_cnt>0],
                            'missing_percentage': miss_pct[miss_pct>0]}) \
                .sort_values('missing_count', ascending=False)

    if len(miss_df) > 0:
        print('\nColumns with missing values:')
        print(miss_df)
        print(f"\n  {name}: Missing values detected.")
        print('Remove columns with % missing more than threshold in DATA␣
 ↪CLEANING.')
    else:
        print(miss_cnt)
        print(f"  {name}: No missing values detected.\n")

check_missing(train, 'train')
check_missing(test, 'test')
```

```
============================================================
CHECK MISSING VALUES
============================================================
p_id                     0
no_times_pregnant        0
glucose_concentration    0
blood_pressure           0
skin_fold_thickness      0
serum_insulin            0
bmi                      0
diabetes pedigree        0
age                      0
diabetes                 0
dtype: int64
  train: No missing values detected.

p_id                     0
no_times_pregnant        0
glucose_concentration    0
blood_pressure           0
skin_fold_thickness      0
serum_insulin            0
bmi                      0
diabetes pedigree        0
age                      0
dtype: int64
  test: No missing values detected.
```

3.2.4 CHECK INVALID VALUES

- Are there any columns with invalid values, i.e. values that are unexpected based on domain understanding?
- From the overview earlier, 5 numerical non-zero columns have invalid zeros.

[9]:
```python
### 1. DEFINE COLUMNS THAT CAN BE ZERO BUT NOT NEGATIVE
test_zero_cols = ['no_times_pregnant', 'diabetes pedigree']
train_zero_cols = test_zero_cols + ['diabetes']

### 2. DEFINE COLUMNS THAT MUST BE POSITIVE (NOT ZERO/NEGATIVE)
pos_cols = [col for col in train.columns if col not in train_zero_cols]

### 3. UDF TO IDENTIFY INVALID (ZERO/NEGATIVE) VALUES
def check_invalid(df, zero_cols, pos_cols, name):
    print_section(f"CHECK INVALID VALUES - {name}")

    ### COLUMNS THAT CAN BE ZERO BUT NOT NEGATIVE
    neg_only_cnt = (df[zero_cols] < 0).sum()
    neg_only_pct = (neg_only_cnt / len(df) * 100).round(1)
    zero_df = pd.DataFrame({
        "negatives_count": neg_only_cnt,
        "negatives_percent": neg_only_pct
    })
    print("1. COLUMNS THAT CAN BE ZERO BUT NOT NEGATIVE (CHECK NEGATIVE)")
    display(zero_df)

    ### COLUMNS THAT MUST BE POSITIVE (NOT ZERO/NEGATIVE)
    zero_cnt = (df[pos_cols] == 0).sum()
    zero_pct = (zero_cnt / len(df) * 100).round(1)
    neg_cnt = (df[pos_cols] < 0).sum()
    neg_pct = (neg_cnt / len(df) * 100).round(1)
    pos_df = pd.DataFrame({
        "zeros_count": zero_cnt,
        "zeros_percent": zero_pct,
        "negatives_count": neg_cnt,
        "negatives_percent": neg_pct
    })
    print("2. COLUMNS THAT MUST BE POSITIVE (CHECK ZERO/NEGATIVE)")
    display(pos_df)

    return zero_df, pos_df

### 4. OUTPUT
train_zero, train_pos = check_invalid(train, train_zero_cols, pos_cols, 'TRAIN')
test_zero, test_pos = check_invalid(test, test_zero_cols, pos_cols, 'TEST')
```

```
============================================================
CHECK INVALID VALUES - TRAIN
```

```
============================================================
1. COLUMNS THAT CAN BE ZERO BUT NOT NEGATIVE (CHECK NEGATIVE)

                   negatives_count  negatives_percent
no_times_pregnant                0                0.0
diabetes pedigree                0                0.0
diabetes                         0                0.0

2. COLUMNS THAT MUST BE POSITIVE (CHECK ZERO/NEGATIVE)

                       zeros_count  zeros_percent  negatives_count  \
p_id                             0            0.0                0
glucose_concentration            4            0.7                0
blood_pressure                  31            5.0                0
skin_fold_thickness            187           30.5                0
serum_insulin                  304           49.5                0
bmi                             10            1.6                0
age                              0            0.0                0

                       negatives_percent
p_id                                 0.0
glucose_concentration                0.0
blood_pressure                       0.0
skin_fold_thickness                  0.0
serum_insulin                        0.0
bmi                                  0.0
age                                  0.0
============================================================
CHECK INVALID VALUES - TEST
============================================================
1. COLUMNS THAT CAN BE ZERO BUT NOT NEGATIVE (CHECK NEGATIVE)

                   negatives_count  negatives_percent
no_times_pregnant                0                0.0
diabetes pedigree                0                0.0

2. COLUMNS THAT MUST BE POSITIVE (CHECK ZERO/NEGATIVE)

                       zeros_count  zeros_percent  negatives_count  \
p_id                             0            0.0                0
glucose_concentration            1            0.6                0
blood_pressure                   4            2.6                0
skin_fold_thickness             40           26.0                0
serum_insulin                   70           45.5                0
bmi                              1            0.6                0
age                              0            0.0                0

                       negatives_percent
p_id                                 0.0
glucose_concentration                0.0
```

```
blood_pressure                    0.0
skin_fold_thickness               0.0
serum_insulin                     0.0
bmi                               0.0
age                               0.0
```

REMARKS

- Both train and test datasets have 5 non-zero columns with invalid zero values.
- We shall treat the invalid values as "missing values".
- In `DATA CLEANING`, the columns with "missing values (invalid values)" above a percentage threshold will be dropped, whereas the others below the threshold will be imputed using median.
- Invalid value threshold: 30%
- Columns to impute: `glucose_concentration`, `blood_pressure`, `bmi`
- Columns to drop: `skin_fold_thickness`, `serum_insulin`
- `skin_fold_thickness` is decided to be dropped.
  - It has significant invalid percent (26% and 30.5%).
  - It is a proxy for body fat, which is sufficiently represented by `bmi`.
  - Imputing will likely introduce bias for marginal gain in model performance.

3.2.5 CHECK OUTLIERS

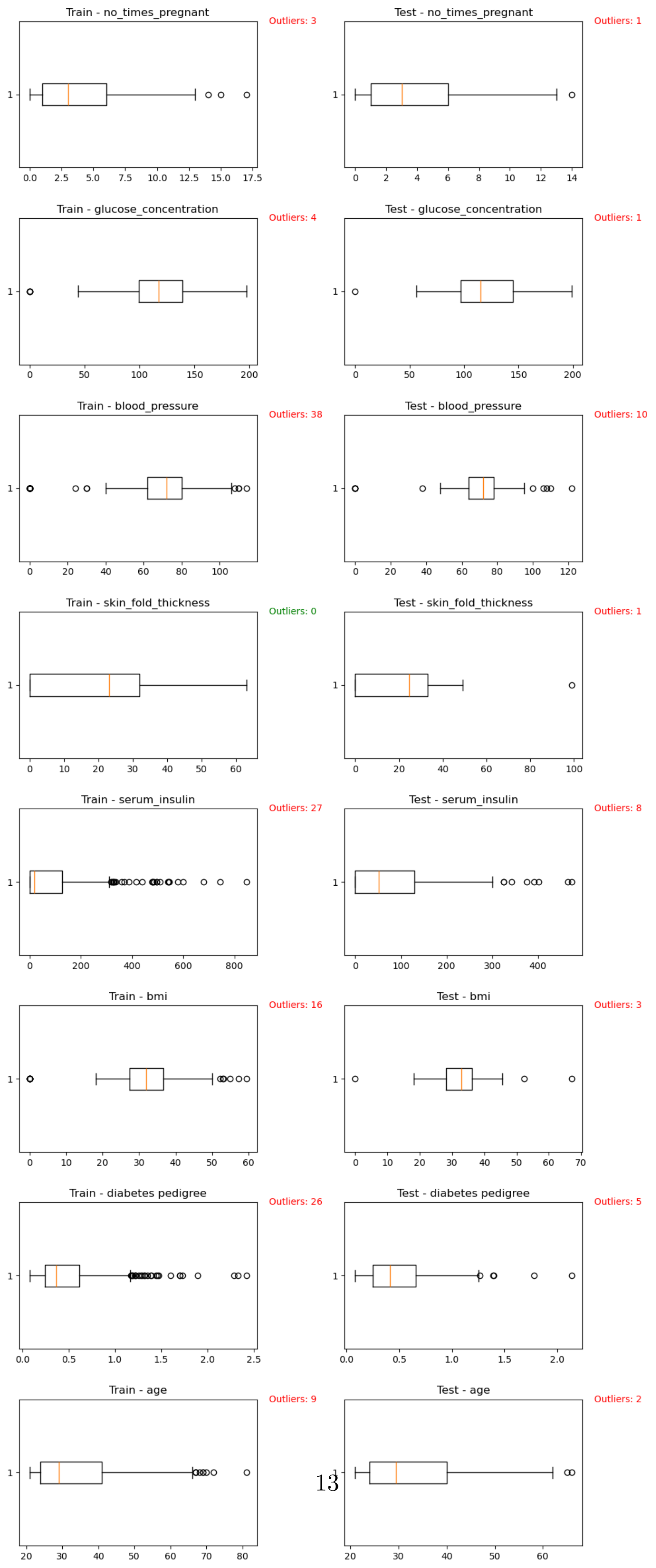
- Methods: IQR & box plot.
- TRAIN dataset is used for the bounds for both TRAIN dataset & TEST dataset.

[10]:
```python
### DEFINE FEATURES (EXCLUDE ID & TARGET)
feature_cols = [c for c in train.columns if c not in ['p_id', 'diabetes']]
print(f"Feature columns for outlier detection: {feature_cols}")
print(f"\nTotal: {len(feature_cols)}")

### UDF TO COMPUTE IQR BOUNDARIES
# For each column in col, compute IQR-based lower and upper bounds.
# Returns (lower, upper) where lower = Q1 - 1.5*IQR and upper = Q3 + 1.5*IQR.
def compute_iqr_bounds(df, col):
    s = df[col]
    Q1 = s.quantile(0.25)
    Q3 = s.quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return lower, upper

### UDF TO COUNT OUTLIERS BEFORE CAPPING
def outlier_counts_by_bounds(df, cols, bounds_source='train'):
    counts = {}
    ### FOR EACH COLUMN IN COLS
    for col in cols:
        s = df[col]
        ### COMPUTE IQR BOUNDS
        lower, upper = compute_iqr_bounds(df, col)
        ### COUNT OUTLIERS OUTSIDE THOSE BOUNDS
        counts[col] = int(((s < lower) | (s > upper)).sum())
    return pd.Series(counts)
```

```
Feature columns for outlier detection: ['no_times_pregnant',
'glucose_concentration', 'blood_pressure', 'skin_fold_thickness',
'serum_insulin', 'bmi', 'diabetes pedigree', 'age']

Total: 8
```

[11]:
```
### PREP THE SUBPLOT GRID
n_featureCols = len(feature_cols)
ncols = 2  # TRAIN | TEST
nrows = n_featureCols  # ONE ROW PER FEATURE
fig, axes = plt.subplots(nrows, ncols, figsize=(10, nrows*3))

print_section('CHECK OUTLIERS (OUTSIDE IQR BOUNDS OF TRAIN)')

### GET OUTLIER COUNTS
before_train_counts = outlier_counts_by_bounds(train, feature_cols)
before_test_counts = outlier_counts_by_bounds(test, feature_cols)

### FOR EACH COLUMN IN TRAIN/TEST, PLOT THE BOXPLOT & SHOW OUTLIER COUNT
for i, c in enumerate(feature_cols):
    ### TRAIN BOXPLOT (LEFT COLUMN)
    axes[i, 0].boxplot(train[c], vert=False)
    axes[i, 0].set_title(f'\nTrain - {c}')
    ### RED IF OUTLIERS; GREEN IF NO OUTLIERS
    train_color = 'green' if before_train_counts[c] == 0 else 'red'
    axes[i, 0].text(1.05, 1, f"Outliers: {before_train_counts[c]}",
                    transform=axes[i, 0].transAxes, fontsize=10,␣
 ↪color=train_color, va='center')

    axes[i, 1].boxplot(test[c], vert=False)
    axes[i, 1].set_title(f'\nTest - {c}')
    ### RED IF OUTLIERS; GREEN IF NO OUTLIERS
    test_color = 'green' if before_test_counts[c] == 0 else 'red'
    axes[i, 1].text(1.05, 1, f"Outliers: {before_test_counts[c]}",
                    transform=axes[i, 1].transAxes, fontsize=10,␣
 ↪color=test_color, va='center')

plt.tight_layout()
plt.show()
```

```
============================================================
CHECK OUTLIERS (OUTSIDE IQR BOUNDS OF TRAIN)
============================================================
```


Train - no_times_pregnant
Outliers: 3
Test - no_times_pregnant
Outliers: 1
Train - glucose_concentration
Outliers: 4
Test - glucose_concentration
Outliers: 1
Train - blood_pressure
Outliers: 38
Test - blood_pressure
Outliers: 10
Train - skin_fold_thickness
Outliers: 0
Test - skin_fold_thickness
Outliers: 1
Train - serum_insulin
Outliers: 27
Test - serum_insulin
Outliers: 8
Train - bmi
Outliers: 16
Test - bmi
Outliers: 3
Train - diabetes pedigree
Outliers: 26
Test - diabetes pedigree
Outliers: 5
Train - age
Outliers: 9
Test - age
Outliers: 2

REMARKS

- Both datasets have outliers, except `skin_fold_thickness` in train.
- We shall ignore the outlier plots in `skin_fold_thickness` and `insulin_serum` which will be dropped later in `DATA CLEANING`.
- For other outliers, they will be **intentionally left unmodified** during `DATA CLEANING` as they represent legitimate, extreme patient medical characteristics, rather than systematic data errors, due to the small percentage they account for.

4 DATA CLEANING

- From `INITIAL DATA INSPECTION`, there are no invalid data types, duplicates, or missing values.
- Columns with invalid values will be imputed or dropped.
- Outliers will be left unmodified.

4.1 HANDLE INVALID VALUES

From `INITIAL DATA INSPECTION`, - Invalid value threshold: 30% - Columns to impute: `glucose_concentration`, `blood_pressure`, `bmi` - Columns to drop: `skin_fold_thickness`, `serum_insulin` - `skin_fold_thickness` is decided to be dropped. - It has significant invalid percent (26% and 30.5%). - It is a proxy for body fat, which is sufficiently represented by `bmi`. - Imputing will likely introduce bias for marginal gain in model performance.

[12]:
```
print_section('HANDLE INVALID VALUES')

### DEFINE COLUMNS TO DROP/IMPUTE
cols_drop = ['skin_fold_thickness', 'serum_insulin']
cols_imp = ['glucose_concentration', 'blood_pressure', 'bmi']

### DROP COLUMNS & CREATE COPIES FOR IMPUTATION
print(f"Dropping columns {cols_drop}...\n")
train_imp = train.drop(cols_drop, axis=1).copy()
test_imp = test.drop(cols_drop, axis=1).copy()

### SANITY CHECK
print(f"Train: {train.shape} -> {train_imp.shape}")
print(f"Test: {test.shape} -> {test_imp.shape}")

### IMPUTE ZEROS WITH TRAIN'S MEDIAN VALUES
print(f"\nImputing zeros in {cols_imp}...\n")
for col in cols_imp:
    ### IMPUTE
    median_val = train_imp[col].median()
    train_imp[col] = train_imp[col].replace(0, median_val)
    test_imp[col] = test_imp[col].replace(0, median_val)

    ### SANITY CHECK
    zeros_train = (train_imp[col] == 0).sum()
    zeros_test = (test_imp[col] == 0).sum()
    print(f"{col}: train zeros = {zeros_train}, test zeros = {zeros_test}")

### LAST CHECK
print("\nColumn Stats")
display(train_imp.describe())
display(test_imp.describe())
```

```
============================================================
HANDLE INVALID VALUES
============================================================
Dropping columns ['skin_fold_thickness', 'serum_insulin']…

Train: (614, 10) -> (614, 8)
Test: (154, 9) -> (154, 7)

Imputing zeros in ['glucose_concentration', 'blood_pressure', 'bmi']…

glucose_concentration: train zeros = 0, test zeros = 0
blood_pressure: train zeros = 0, test zeros = 0
bmi: train zeros = 0, test zeros = 0

Column Stats

             p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
count  614.000000         614.000000             614.000000      614.000000
mean   385.773616           3.853420             121.304560       72.400651
std    223.603024           3.358126              29.688213       12.031595
min      1.000000           0.000000              44.000000       24.000000
25%    191.250000           1.000000             100.000000       64.000000
50%    387.000000           3.000000             117.000000       72.000000
75%    572.750000           6.000000             139.000000       80.000000
max    768.000000          17.000000             197.000000      114.000000

              bmi  diabetes pedigree         age    diabetes
count  614.000000         614.000000  614.000000  614.000000
mean    32.431107           0.466342   33.325733    0.348534
std      6.873171           0.331090   11.929569    0.476895
min     18.200000           0.078000   21.000000    0.000000
25%     27.500000           0.240250   24.000000    0.000000
50%     32.000000           0.361000   29.000000    0.000000
75%     36.600000           0.613500   41.000000    1.000000
max     59.400000           2.420000   81.000000    1.000000
             p_id  no_times_pregnant  glucose_concentration  blood_pressure  \
count   154.000000          154.000000              154.000000       154.000000
mean    379.422078            3.811688              123.058442        72.331169
std     215.338912            3.425719               33.328085        12.392310
min       4.000000            0.000000               56.000000        38.000000
25%     194.250000            1.000000               97.500000        66.000000
50%     383.000000            3.000000              115.500000        72.000000
75%     583.250000            6.000000              144.750000        78.000000
max     738.000000           14.000000              199.000000       122.000000

             bmi  diabetes pedigree         age
count  154.00000         154.000000  154.000000
mean    32.52987           0.493942   32.902597
std      6.90599           0.332439   11.090106
min     18.20000           0.084000   21.000000
25%     28.37500           0.248000   24.000000
50%     32.90000           0.411500   29.500000
75%     36.20000           0.654750   40.000000
max     67.10000           2.137000   66.000000
```

4.2 LATEST CLEANED DATA

[13]:
```
train_clean = train_imp.copy()
test_clean = test_imp.copy()
feature_cols_clean = [c for c in train_clean.columns if c not in ['p_id',␣
↪'diabetes']]
print(f"Feature Columns After Cleaning: {feature_cols_clean}\n")

print_section('TRAIN_CLEAN')
train_clean.info()
display(train_clean.head())
print_section('TEST_CLEAN')
test_clean.info()
display(test_clean.head())

print('''
 DATA CLEANING COMPLETED!
- TRAIN DATASET IS READY FOR EDA.
- TEST DATASET IS READY FOR FEATURE ENGINEERING.
''')
```

```
Feature Columns After Cleaning: ['no_times_pregnant', 'glucose_concentration',
'blood_pressure', 'bmi', 'diabetes pedigree', 'age']

============================================================
TRAIN_CLEAN
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
```

```
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   p_id                   614 non-null    int64  
 1   no_times_pregnant      614 non-null    int64  
 2   glucose_concentration  614 non-null    int64  
 3   blood_pressure         614 non-null    int64  
 4   bmi                    614 non-null    float64
 5   diabetes pedigree      614 non-null    float64
 6   age                    614 non-null    int64  
 7   diabetes               614 non-null    int64  
dtypes: float64(2), int64(6)
memory usage: 38.5 KB
```

```
   p_id  no_times_pregnant  glucose_concentration  blood_pressure   bmi  \
0   316                  2                    112              68  34.1   
1    25                 11                    143              94  36.6   
2   710                  2                     93              64  38.0   
3   658                  1                    120              80  38.9   
4   542                  3                    128              72  32.4   

   diabetes pedigree  age  diabetes  
0              0.315   26         0  
1              0.254   51         1  
2              0.674   23         1  
3              1.162   41         0  
4              0.549   27         1  
```

```
============================================================
TEST_CLEAN
============================================================
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 154 entries, 0 to 153
Data columns (total 7 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   p_id                   154 non-null    int64  
 1   no_times_pregnant      154 non-null    int64  
 2   glucose_concentration  154 non-null    int64  
 3   blood_pressure         154 non-null    int64  
 4   bmi                    154 non-null    float64
 5   diabetes pedigree      154 non-null    float64
 6   age                    154 non-null    int64  
dtypes: float64(2), int64(5)
memory usage: 8.6 KB
```

```
   p_id  no_times_pregnant  glucose_concentration  blood_pressure   bmi  \
0   437                 12                    140              85  37.4   
1   411                  6                    102              90  35.7   
```

```
2   639                   7                        97                76   40.9
3   213                   7                       179                95   34.2
4   181                   6                        87                80   23.2

   diabetes pedigree  age
0              0.244   41
1              0.674   28
2              0.871   32
3              0.164   60
4              0.084   32


 DATA CLEANING COMPLETED!
- TRAIN DATASET IS READY FOR EDA.
- TEST DATASET IS READY FOR FEATURE ENGINEERING.
```

5 EDA

- EDA is done only on the TRAIN dataset without touching the TEST dataset.
- This is to prevent leaking information and artificially improving ML model performance.

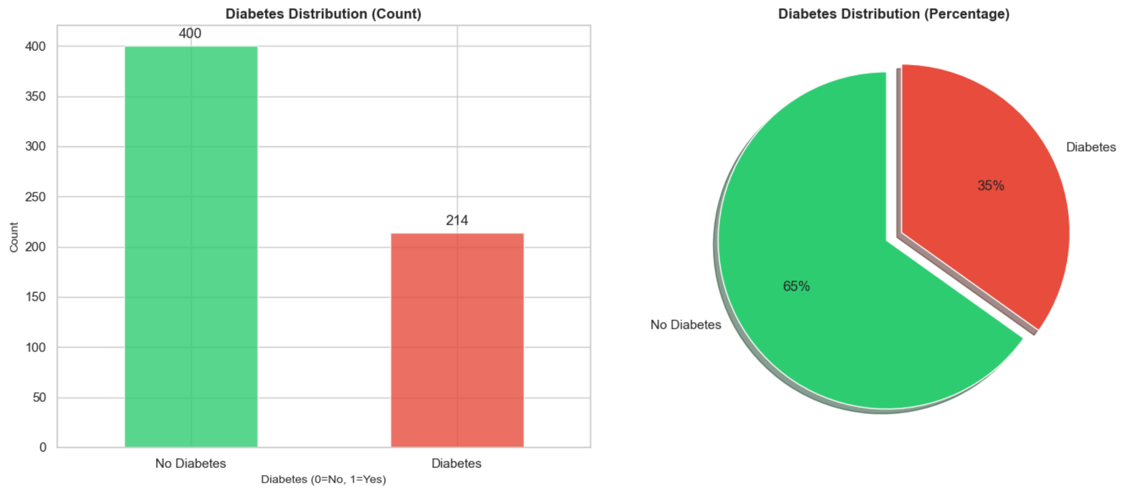
[14]:
```
### SET PLOTTING STYLE
# white background, visible grey gridlines
# all figure sizes to 12 inches wide x 6 inches tall unless specified
sns.set_theme(style='whitegrid', rc={'figure.figsize': (14, 6)})
```

5.1 TARGET VARIABLE ANALYSIS

[15]:
```
print_section('TARGET VARIABLE ANALYSIS')

### SHOW COUNT AND PERCENTAGE
diabetes_count = train_clean['diabetes'].value_counts()
diabetes_pct = train_clean['diabetes'].value_counts(normalize=True) * 100

# :.1f and :.2f to round to 1 and 2 decimal place(s) respectively
print('Diabetes Distribution:')
print(f"Non-Diabetic (0): {diabetes_count[0]} ({diabetes_pct[0]:.1f}%)")
print(f"Diabetic (1): {diabetes_count[1]} ({diabetes_pct[1]:.1f}%)")
print(f"\nNon-Diabetic-to-Diabetic Ratio: {diabetes_count[0]/diabetes_count[1]:.
 ↪2f}:1")

### VISUALIZE TARGET DISTRIBUTION
# Create a figure of two subplots (1 row x 2 cols)
# -------------------------------
# |   Axes[0]   |   Axes[1]     |
# -------------------------------
fig, axes = plt.subplots(1, 2)
```

```
colors = ['#2ecc71', '#e74c3c']

# Bar Chart for Axes[0]
diabetes_count.plot(kind='bar', ax=axes[0], color=colors, alpha=0.8)
axes[0].set_title('Diabetes Distribution (Count)', fontsize=12,␣
 ↪fontweight='bold')
axes[0].set_xlabel('Diabetes (0=No, 1=Yes)', fontsize=10)
axes[0].set_ylabel('Count', fontsize=10)
axes[0].set_xticklabels(['No Diabetes', 'Diabetes'], rotation=0)

# Add count labels on bars (5 counts above current count y)
# va='bottom': the bottom of text box sits above y-coordinate = y + 5
for x, y in enumerate(diabetes_count):
    axes[0].text(x, y + 5, str(y), ha='center', va='bottom')

# Pie Chart for Axes[1]
axes[1].pie(diabetes_count, labels=['No Diabetes', 'Diabetes'],
            colors=colors, autopct='%1.0f%%', startangle=90,
            explode=(0.05, 0.05), shadow=True)
axes[1].set_title('Diabetes Distribution (Percentage)', fontsize=12,␣
 ↪fontweight='bold')

plt.tight_layout()
plt.show()
```

```
============================================================
TARGET VARIABLE ANALYSIS
============================================================
Diabetes Distribution:
Non-Diabetic (0): 400 (65.1%)
Diabetic (1): 214 (34.9%)

Non-Diabetic-to-Diabetic Ratio: 1.87:1
```



5.1.1 REMARKS

- The train dataset has class imbalance - 1.87 times more non-diabetic than diabetic patients.
- This can make the ML models more biased towards the majority class (non-diabetic) and predict it more frequently.
- Hence, the "accuracy" metric becomes unreliable, and cannot be the sole evaluation metric.
- During model evaluation, **use other metrics (precision, recall, F1-score) alongside accuracy** to give a better picture of a model's performance.
- https://www.geeksforgeeks.org/machine-learning/handling-imbalanced-data-for-classification/

5.2 STATISTICAL SUMMARY BY DIABETES STATUS

[16]:
```
print_section('STATISTICAL SUMMARY BY DIABETES')

### SUMMARY STATISTICS GROUPED BY DIABETES STATUS
print('\n--- Non-Diabetic (diabetes=0) ---')
non_diabetic_stat = train_clean[train_clean['diabetes']==0][feature_cols_clean].
  ↪describe()
display(non_diabetic_stat.round(2))

print('\n--- Diabetic (diabetes=1) ---')
diabetic_stat = train_clean[train_clean['diabetes']==1][feature_cols_clean].
  ↪describe()
display(diabetic_stat.round(2))

### COMPARE MEDIANS BETWEEN GROUPS
print('\n--- Median Comparison: Diabetic vs Non-Diabetic ---')
median_compare = pd.DataFrame({
    'Non-Diabetic': train_clean[train_clean['diabetes']==0][feature_cols_clean].
  ↪median(),
    'Diabetic': train_clean[train_clean['diabetes']==1][feature_cols_clean].
  ↪median(),
})
median_compare['Difference'] = median_compare['Diabetic'] -␣
  ↪median_compare['Non-Diabetic']
median_compare['Pct_Change'] = (median_compare['Difference'] /␣
  ↪median_compare['Non-Diabetic'] * 100).round(0)
display(median_compare.round(2))
```

```
============================================================
STATISTICAL SUMMARY BY DIABETES
============================================================


--- Non-Diabetic (diabetes=0) ---
```

```
       no_times_pregnant  glucose_concentration  blood_pressure     bmi  \
count             400.00                 400.00          400.00  400.00
mean                3.34                 110.41           70.78   30.94
std                 3.01                  23.78           11.61    6.58
min                 0.00                  44.00           24.00   18.20
25%                 1.00                  94.00           62.00   25.90
50%                 2.00                 108.00           72.00   30.10
75%                 5.00                 124.25           78.00   35.30
max                13.00                 197.00          110.00   57.30

       diabetes pedigree     age
count             400.00  400.00
mean                0.42   31.39
std                 0.30   12.00
min                 0.08   21.00
25%                 0.23   23.00
50%                 0.33   27.00
75%                 0.55   37.00
max                 2.33   81.00


--- Diabetic (diabetes=1) ---
       no_times_pregnant  glucose_concentration  blood_pressure     bmi  \
count             214.00                 214.00          214.00  214.00
mean                4.81                 141.67           75.43   35.22
std                 3.74                  28.93           12.25    6.54
min                 0.00                  78.00           30.00   22.90
25%                 1.25                 119.00           68.00   30.50
50%                 4.00                 140.00           74.00   34.05
75%                 8.00                 164.75           84.00   38.92
max                17.00                 197.00          114.00   59.40

       diabetes pedigree     age
count             214.00  214.00
mean                0.54   36.94
std                 0.37   10.95
min                 0.12   21.00
25%                 0.26   28.00
50%                 0.43   36.00
75%                 0.73   44.75
max                 2.42   70.00


--- Median Comparison: Diabetic vs Non-Diabetic ---
                       Non-Diabetic  Diabetic  Difference  Pct_Change
no_times_pregnant              2.00      4.00        2.00       100.0
glucose_concentration        108.00    140.00       32.00        30.0
blood_pressure                72.00     74.00        2.00         3.0
```

```
bmi                           30.10     34.05        3.95        13.0
diabetes pedigree              0.33      0.43        0.11        32.0
age                           27.00     36.00        9.00        33.0
```

5.2.1 REMARKS

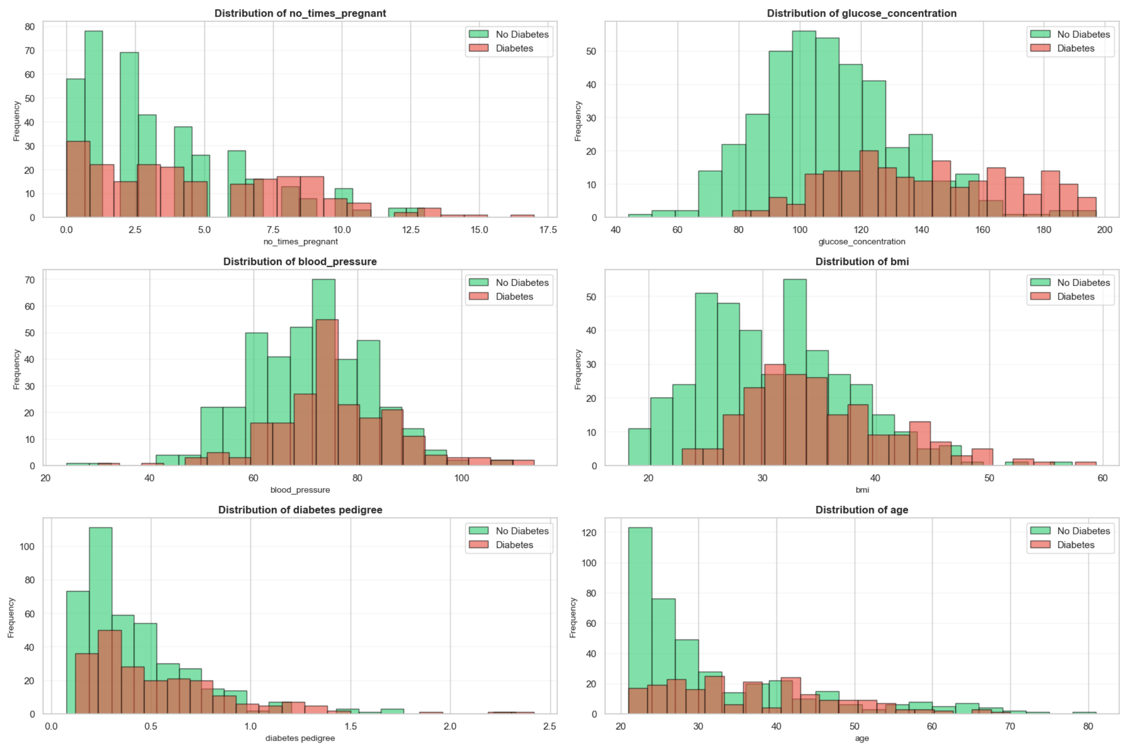
- **Median values are consistently higher in diabetic patients across all six numerical features**, indicating a systematic shift in central tendency rather than effects driven by outliers.
- The most pronounced differences are observed in glucose concentration (+30%), age (+33%), number of pregnancies (median increase from 2 to 4), and diabetes pedigree score (+32%). BMI shows a moderate increase (+13%), while blood pressure exhibits only a small median difference (+3%).
- These results are descriptive but do not imply statistical significance.

5.3 FEATURE DISTRIBUTIONS BY DIABETES

[17]:
```python
print_section('FEATURE DISTRIBUTIONS BY DIABETES')

### SET THE SUBPLOT GRID SHAPE
n_cols = 2
# Count no. of rows of subplots based on feature cols
n_rows = int(np.ceil(len(feature_cols_clean) / n_cols))
# Create the grid 18-inch wide with 4 inches per row
fig, axes = plt.subplots(n_rows, n_cols, figsize=(18, n_rows*4))
# Flatten the 2D array of axes (n_rows, n_cols) to 1D for easy indexing of each␣
 ↪subplot as axes(idx)
axes = axes.ravel()

### FOR EVERY COLUMN AND ITS ENUMERATED INDEX IN feature_cols_clean
for idx, col in enumerate(feature_cols_clean):
    ### PLOT FOR NON-DIABETIC
    train_clean[train_clean['diabetes']==0][col].hist(
        bins=20, alpha=0.6, label='No Diabetes',
        color=colors[0], ax=axes[idx], edgecolor='black'
    )
    ### PLOT FOR DIABETIC
    train_clean[train_clean['diabetes']==1][col].hist(
        bins=20, alpha=0.6, label='Diabetes',
        color=colors[1], ax=axes[idx], edgecolor='black'
    )
    ### SET PLOT ANNOTATIONS
    axes[idx].set_title(f'Distribution of {col}', fontsize=12,␣
 ↪fontweight='bold')
    axes[idx].set_xlabel(col, fontsize=10)
    axes[idx].set_ylabel('Frequency', fontsize=10)
    axes[idx].legend()
    axes[idx].grid(axis='y', alpha=0.2)
```

```python
### HIDE UNUSED SUBPLOT (IF ANY)
for idx in range(len(feature_cols_clean), len(axes)):
    axes[idx].axis('off')

plt.tight_layout()
plt.show()
```

```
============================================================
FEATURE DISTRIBUTIONS BY DIABETES
============================================================
```



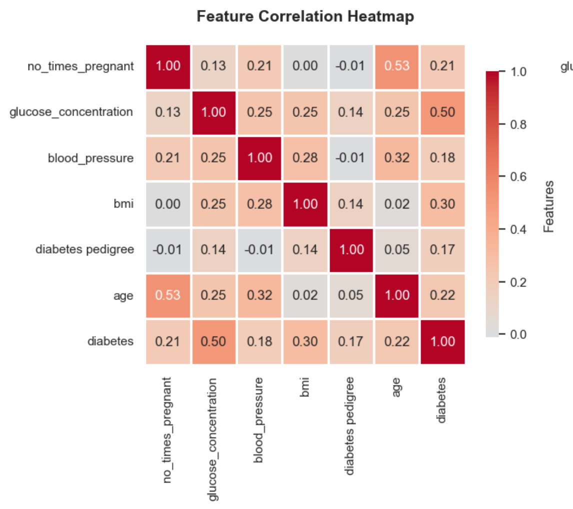
5.3.1 REMARKS

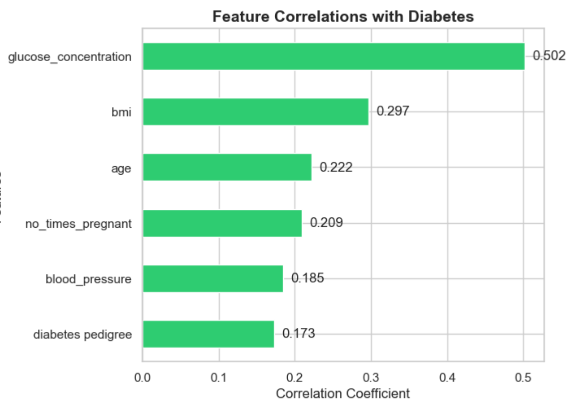
- Obvious difference in distributions of `glucose_concentration`, `bmi`, and `age` between diabetic and non-diabetic patients.
- Diabetic patients tend to have **higher glucose concentration, BMI, and age**.
- Other features (`no_times_pregnant`, `blood_pressure`, `diabetes pedigree`) have similar, overlapping distributions.

5.4 CORRELATION ANALYSIS

[18]:
```
print_section('CORRELATION ANALYSIS')

### CALCULATE CORRELATION MATRIX
corr_matrix = train_clean[feature_cols_clean + ['diabetes']].corr()

### CREATE A FIGURE OF TWO SUBPLOTS (1 ROW X 2 COLS)
fig, axes = plt.subplots(1, 2)

### 1. CORRELATION HEATMAP IN LEFT SUBPLOT
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            center=0, square=True, linewidths=1, cbar_kws={"shrink": 0.8},␣
 ↪ax=axes[0])
axes[0].set_title('Feature Correlation Heatmap', fontsize=14,␣
 ↪fontweight='bold', pad=20)

### 2. HORIZONAL BAR PLOT IN RIGHT SUBPLOT
diabetes_corr = corr_matrix['diabetes'].round(3).drop('diabetes').sort_values()
# Dynamic color based on corr sign
colorsCorr = [colors[0] if x > 0 else colors[1] for x in diabetes_corr.values]
ax = diabetes_corr.plot(kind='barh', color=colorsCorr, ax=axes[1])

# Add corr values as text labels at the end of each bar
for i, (value, feature) in enumerate(zip(diabetes_corr.values, diabetes_corr.
 ↪index)):
    # x-position = end of bar (value)
    # y-position = bar center (i)
    ax.text(value + 0.01 if value > 0 else value - 0.01,  # offset outside the␣
 ↪bar; dynamic based on corr sign
            i, # y-coordinate of horizontal bar
            f"{value:.3f}",  # formatted corr value to 3 dp
            va='center',
            ha='left' if value > 0 else 'right') # outside bar; dynamic based␣
 ↪on corr sign

axes[1].set_title('Feature Correlations with Diabetes', fontsize=14,␣
 ↪fontweight='bold')
axes[1].set_xlabel('Correlation Coefficient', fontsize=12)
axes[1].set_ylabel('Features', fontsize=12)

plt.tight_layout()
plt.show()
```

```
============================================================
CORRELATION ANALYSIS
============================================================
```





5.4.1 REMARKS

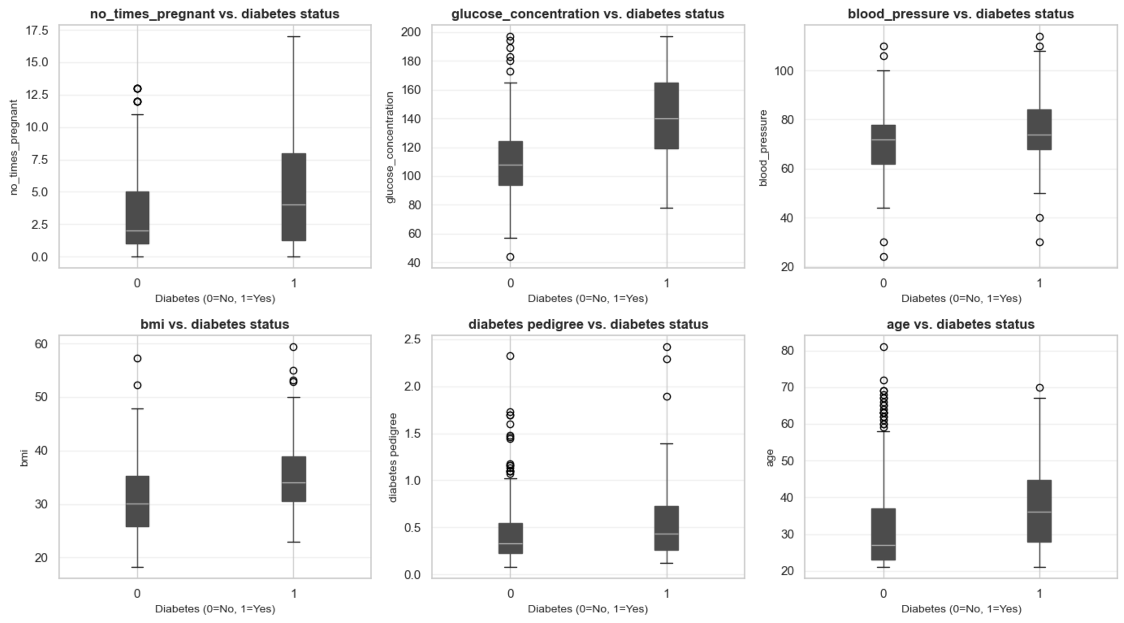
- Glucose concentration has highest linear correlation with diabetes (0.502).
- BMI is second most correlated with diabetes (0.297).
- Other features show moderate-to-low correlation.
- Some features are correlated with each other (multicollinearity), such as no_times_pregnant & age (0.53).

5.5 BOX PLOTS BY DIABETES

[19]:
```
print_section('BOX PLOTS BY DIABETES STATUS')

### BUILD BOX PLOTS FOR ALL FEATURES BY DIABETES STATUS
n_cols = 3
n_rows = int(np.ceil(len(feature_cols_clean) / n_cols))
fig, axes = plt.subplots(n_rows, n_cols, figsize=(14, n_rows*4))
axes = axes.ravel()

for idx, col in enumerate(feature_cols_clean):
    train_clean.boxplot(column=col, by='diabetes', ax=axes[idx],␣
 ↪patch_artist=True, return_type='dict')
    axes[idx].set_title(f'{col} vs. diabetes status', fontsize=12,␣
 ↪fontweight='bold')
    axes[idx].set_xlabel('Diabetes (0=No, 1=Yes)', fontsize=10)
    axes[idx].set_ylabel(col, fontsize=10)
    axes[idx].grid(axis='y', alpha=0.3)

### HIDE UNUSED SUBPLOT
for idx in range(len(feature_cols_clean), len(axes)):
    axes[idx].axis('off')
```

```
plt.suptitle('')  # Remove default parent title
plt.tight_layout()
plt.show()
```

```
============================================================
BOX PLOTS BY DIABETES STATUS
============================================================
```



5.5.1 REMARKS

1. **Glucose concentration** shows the strongest separation between diabetic and non-diabetic groups.
   - Diabetic individuals have much higher glucose (higher median, higher quartiles, more extreme values).
   - This aligns with medical expectations and indicates glucose is the most discriminative feature.
2. Several features shift upward for diabetics (**age**, **BMI**, **pregnancies**).
   - **Age:** Diabetic individuals tend to be older on average.
   - **BMI:** Higher BMI among diabetics suggests a connection with obesity-related risk.
   - **Number of pregnancies:** The diabetic group shows higher median values & upper bound, but largely overlapping.
   - These features show moderate separation but not as strong as **glucose concentration**.
3. High overlap in **blood pressure** and **pedigree**.
   - Although the distributions overlap, diabetics tend to have:
     - Slightly higher blood pressure

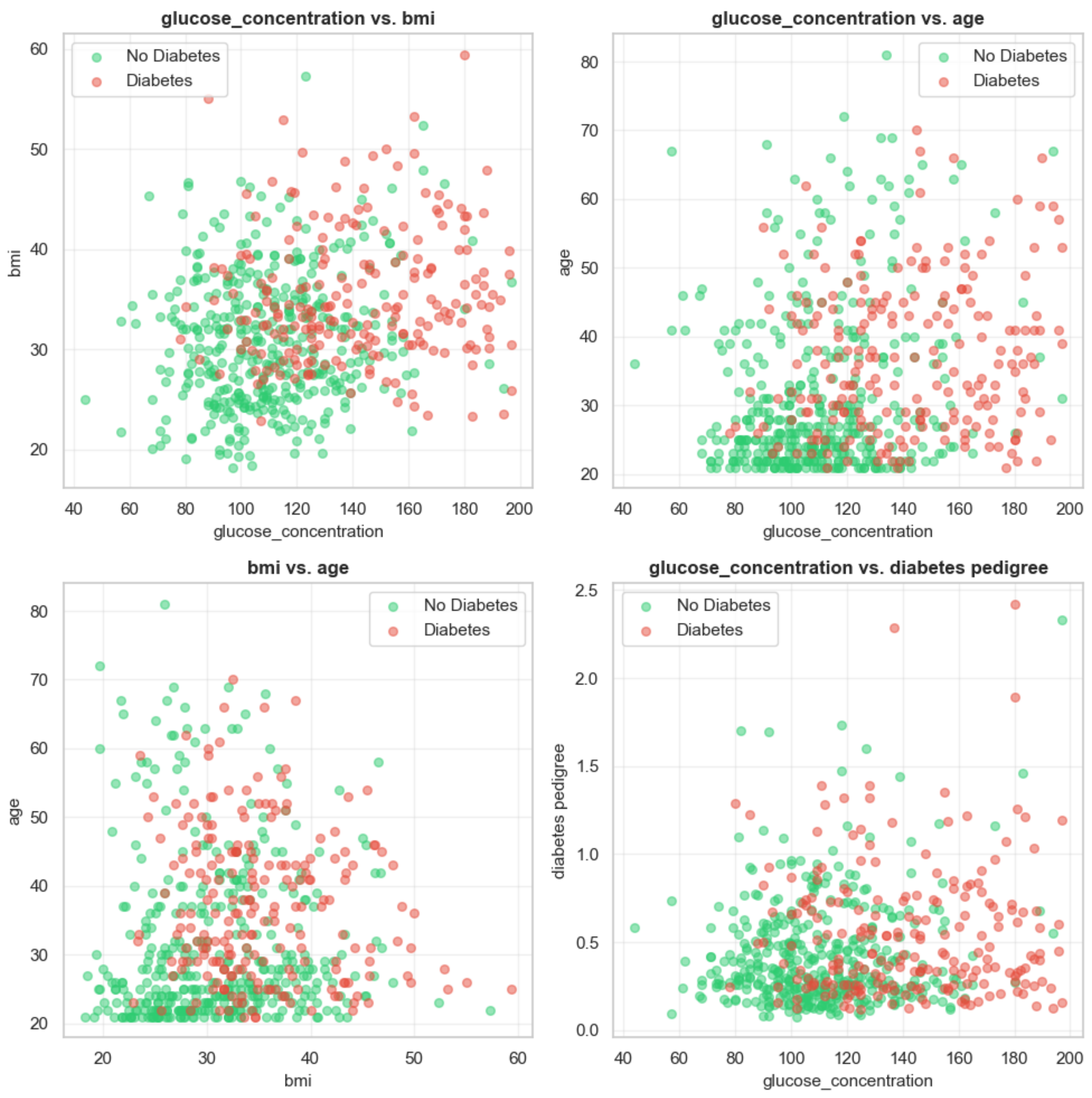
  - Higher diabetes pedigree function (genetic risk indicator)
- These features contribute information but *individually* are not determining factors to separate between statuses.

5.6 SCATTER PLOTS

[20]:
```
print_section('SCATTER PLOTS')

### CHOOSE MOST IMPORTANT FEATURES ONLY
# glucose_concentration, bmi, age, diabetes pedigree

### CREATE A FIGURE OF 2x2 SUBPLOTS
fig, axes = plt.subplots(2, 2, figsize=(10, 10)) # Square plots for easier␣
 ↪visualization
axes = axes.ravel()

### DEFINE THE SCATTER PLOT COMBINATIONS
combinations = [
    ('glucose_concentration', 'bmi'),
    ('glucose_concentration', 'age'),
    ('bmi', 'age'),
    ('glucose_concentration', 'diabetes pedigree')
]
for idx, (feat1, feat2) in enumerate(combinations):
    ### PLOT NON-DIABETIC
    axes[idx].scatter(
        train_clean[train_clean['diabetes']==0][feat1],
        train_clean[train_clean['diabetes']==0][feat2],
        alpha=0.5, c='#2ecc71', label='No Diabetes', s=30
    )
    ### PLOT DIABETIC
    axes[idx].scatter(
        train_clean[train_clean['diabetes']==1][feat1],
        train_clean[train_clean['diabetes']==1][feat2],
        alpha=0.5, c='#e74c3c', label='Diabetes', s=30
    )
    ### SET PLOT ANNOTATIONS
    axes[idx].set_xlabel(feat1, fontsize=11)
    axes[idx].set_ylabel(feat2, fontsize=11)
    axes[idx].set_title(f'{feat1} vs. {feat2}', fontsize=12, fontweight='bold')
    axes[idx].legend()
    axes[idx].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

```
============================================================
SCATTER PLOTS
```

```
================================================================
```



5.6.1 REMARKS

1. **Glucose concentration is strongly associated with diabetes**
    - Across all plots involving glucose concentration (vs BMI, age, and diabetes pedigree), individuals with diabetes generally have higher glucose levels compared to those without diabetes.
    - This indicates that glucose concentration is a strong indicator of diabetes risk.
2. **BMI and age show moderate separation**
    - In the BMI vs age plot, individuals with diabetes tend to cluster at slightly higher BMI and older ages *compared to* those without diabetes, but there is considerable overlap.
    - This suggests BMI and age contribute to diabetes risk but are **not as definitive alone**.

3. **Diabetes pedigree contributes to risk prediction**
    - In the glucose concentration vs diabetes pedigree plot, individuals with diabetes often have higher diabetes pedigree values, especially when combined with higher glucose levels.
    - This shows that a family history of diabetes (captured by the diabetes pedigree) is an important factor when assessing diabetes risk.

Overall, glucose concentration appears to be the strongest single indicator, while BMI, age, and diabetes pedigree provide additional context that can improve risk assessment.

6 FEATURE ENGINEERING

[21]:
```
train_before_FE = train_clean.copy()
test_before_FE = test_clean.copy()
```

6.1 CREATE CATEGORICAL FEATURES

6.1.1 BMI CATEGORIES

- WHO classifications: https://www.who.int/data/gho/data/themes/topics/topic-details/GHO/body-mass-index

[22]:
```
print_section('CREATE BMI CATEGORIES BASED ON WHO CLASSIFICATIONS')
### DEFINE CATEGORICAL BINS AND LABELS BASED ON WHO CLASSIFICATIONS
bins_bmi = [0, 18.5, 25, 30, float('inf')] # infinite upper bound
labels_bmi = [0, 1, 2, 3]  # 0=Underweight, 1=Normal, 2=Overweight, 3=Obese

### CUT WITH LEFT-INCLUSIVE (DEFAULT MODE IS RIGHT INCLUSIVE right=True)
train_clean['bmi_category'] = pd.cut(train_clean['bmi'], bins=bins_bmi,␣
 ↪labels=labels_bmi, right=False)
test_clean['bmi_category'] = pd.cut(test_clean['bmi'], bins=bins_bmi,␣
 ↪labels=labels_bmi, right=False)

print('BMI CATEGORIES CREATED:')
print('0 = Underweight (<18.5)')
print('1 = Normal (18.5-24.9)')
print('2 = Overweight (25-29.9)')
print('3 = Obese ( 30)')

### SHOW CATEGORIES FOR TRAIN DATA
print('\nDISTRIBUTION IN TRAIN DATA:')
print(train_clean['bmi_category'].value_counts().sort_index())
```

```
============================================================
CREATE BMI CATEGORIES BASED ON WHO CLASSIFICATIONS
============================================================
BMI CATEGORIES CREATED:
0 = Underweight (<18.5)
1 = Normal (18.5-24.9)
```

```
2 = Overweight (25-29.9)
3 = Obese ( 30)

DISTRIBUTION IN TRAIN DATA:
bmi_category
0      2
1     78
2    155
3    379
Name: count, dtype: int64
```

6.1.2 AGE GROUPS

[23]:
```
print_section('CREATE AGE GROUPS')
### DEFINE DESIRED CATEGORICAL BINS & LABELS
bins_age = [20, 30, 40, 50, float('inf')] # infinite upper bound
labels_age = [0, 1, 2, 3]  # 0=20-29, 1=30-39, 2=40-49, 3=50+

### CUT WITH LEFT-INCLUSIVE (DEFAULT MODE IS RIGHT INCLUSIVE right=True)
train_clean['age_group'] = pd.cut(train_clean['age'], bins=bins_age,␣
 ↪labels=labels_age, right=False)
test_clean['age_group'] = pd.cut(test_clean['age'], bins=bins_age,␣
 ↪labels=labels_age, right=False)

print('AGE GROUPS CREATED:')
print('0 = 20-29 years')
print('1 = 30-39 years')
print('2 = 40-49 years')
print('3 = 50+ years')

### SHOW CATEGORIES FOR TRAIN DATA
print('\nDISTRIBUTION IN TRAIN DATA:')
print(train_clean['age_group'].value_counts().sort_index())
```

```
============================================================
CREATE AGE GROUPS
============================================================
AGE GROUPS CREATED:
0 = 20-29 years
1 = 30-39 years
2 = 40-49 years
3 = 50+ years

DISTRIBUTION IN TRAIN DATA:
age_group
0    319
1    128
2     95
```

```
3     72
Name: count, dtype: int64
```

6.1.3 GLUCOSE RISK CATEGORIES

- Bins are based on https://diabetesjournals.org/care/article/29/suppl_1/s43/23313/Diagnosis-and-Classification-of-Diabetes-Mellitus

[24]:
```
print_section('CREATE GLUCOSE RISK CATEGORIES')
### DEFINE CATEGORICAL BINS & LABELS
bins_glucose = [0, 100, 126, float('inf')]
labels_glucose = [0, 1, 2]  # 0=Normal, 1=Prediabetic, 2=Diabetic

### CUT WITH LEFT-INCLUSIVE (DEFAULT MODE IS RIGHT INCLUSIVE right=True)
train_clean['glucose_category'] = pd.cut(train_clean['glucose_concentration'],␣
 ↪bins=bins_glucose, labels=labels_glucose, right=False)
test_clean['glucose_category'] = pd.cut(test_clean['glucose_concentration'],␣
 ↪bins=bins_glucose, labels=labels_glucose, right=False)

print('GLUCOSE RISK CATEGORIES CREATED:')
print('0 = Normal (<100 mg/dL)')
print('1 = Prediabetic (100-125.9 mg/dL)')
print('2 = Diabetic range ( 126 mg/dL)')

### SHOW CATEGORIES FOR TRAIN DATA
print('\nDISTRIBUTION IN TRAIN DATA:')
print(train_clean['glucose_category'].value_counts().sort_index())
```

```
============================================================
CREATE GLUCOSE RISK CATEGORIES
============================================================
GLUCOSE RISK CATEGORIES CREATED:
0 = Normal (<100 mg/dL)
1 = Prediabetic (100-125.9 mg/dL)
2 = Diabetic range ( 126 mg/dL)

DISTRIBUTION IN TRAIN DATA:
glucose_category
0    151
1    229
2    234
Name: count, dtype: int64
```

6.1.4 CONVERT CATEGORICAL FEATURES TO NUMERIC

[25]:
```
train_clean['bmi_category'] = train_clean['bmi_category'].astype(int)
test_clean['bmi_category'] = test_clean['bmi_category'].astype(int)
```

```
train_clean['age_group'] = train_clean['age_group'].astype(int)
test_clean['age_group'] = test_clean['age_group'].astype(int)

train_clean['glucose_category'] = train_clean['glucose_category'].astype(int)
test_clean['glucose_category'] = test_clean['glucose_category'].astype(int)

print('  CATEGORICAL FEATURES (CONVERTED TO NUMERIC) CREATED!')
```

```
 CATEGORICAL FEATURES (CONVERTED TO NUMERIC) CREATED!
```

6.2 CREATE RATIO & INTERACTION FEATURES

6.2.1 GLUCOSE-BMI RATIO

[26]:
```
### CREATE
train_clean['glucose_bmi_ratio'] = train_clean['glucose_concentration']/␣
 ↪(train_clean['bmi'] + 1e-6)
test_clean['glucose_bmi_ratio'] = test_clean['glucose_concentration']/␣
 ↪(test_clean['bmi'] + 1e-6)

### SHOW MEAN & RANGE FOR TRAIN DATA
print_section('CREATE GLUCOSE-BMI RATIO')
print(f"Mean: {train_clean['glucose_bmi_ratio'].mean():.2f}")
print(f"Range: {train_clean['glucose_bmi_ratio'].min():.2f} to␣
 ↪{train_clean['glucose_bmi_ratio'].max():.2f}")
```

```
============================================================
CREATE GLUCOSE-BMI RATIO
============================================================
Mean: 3.86
Range: 1.48 to 8.26
```

6.2.2 BMI-AGE INTERACTION

[27]:
```
### CREATE
train_clean['bmi_age_interaction'] = train_clean['bmi'] * train_clean['age']
test_clean['bmi_age_interaction'] = test_clean['bmi'] * test_clean['age']

### SHOW MEAN FOR TRAIN DATA
print_section('CREATE BMI-AGE INTERACTION')
print(f"Mean: {train_clean['bmi_age_interaction'].mean():.2f}")
print(f"Range: {train_clean['bmi_age_interaction'].min():.2f} to␣
 ↪{train_clean['bmi_age_interaction'].max():.2f}")
```

```
============================================================
CREATE BMI-AGE INTERACTION
============================================================
Mean: 1082.14
Range: 382.20 to 2697.00
```

6.2.3 PREGNANCY-AGE INTERACTION

[28]:
```
### CREATE
train_clean['pregnancy_age_interaction'] = train_clean['no_times_pregnant'] *␣
  ↪train_clean['age']
test_clean['pregnancy_age_interaction'] = test_clean['no_times_pregnant'] *␣
  ↪test_clean['age']

### SHOW MEAN FOR TRAIN DATA
print_section('CREATE PREGNANCY-AGE INTERACTION')
print(f"Mean: {train_clean['pregnancy_age_interaction'].mean():.2f}")
print(f"Range: {train_clean['pregnancy_age_interaction'].min():.2f} to␣
  ↪{train_clean['pregnancy_age_interaction'].max():.2f}")

print('\n Ratio and interaction features created successfully!')
```

```
============================================================
CREATE PREGNANCY-AGE INTERACTION
============================================================
Mean: 149.79
Range: 0.00 to 799.00

 Ratio and interaction features created successfully!
```

6.3 CREATE HIGH-RISK FLAGS

6.3.1 HIGH GLUCOSE FLAG

[29]:
```
# DEFINE THRESHOLD
glucose_threshold = 126  # mg/dL (diabetic range)

### CREATE FLAGS
train_clean['high_glucose_flag'] = (train_clean['glucose_concentration'] >=␣
  ↪glucose_threshold).astype(int)
test_clean['high_glucose_flag'] = (test_clean['glucose_concentration'] >=␣
  ↪glucose_threshold).astype(int)

### SHOW FLAG LOGIC & COUNT FOR TRAIN DATA
print_section('CREATE HGH GLUCOSE FLAG')
print(f'high_glucose_flag = 1 if glucose_concentration >= {glucose_threshold}␣
  ↪mg/dL; 0 otherwise')
print(f'{train_clean["high_glucose_flag"].sum()} patients␣
  ↪({train_clean["high_glucose_flag"].mean()*100:.1f}%) flagged in TRAIN data')
```

```
============================================================
CREATE HGH GLUCOSE FLAG
============================================================
high_glucose_flag = 1 if glucose_concentration >= 126 mg/dL; 0 otherwise
234 patients (38.1%) flagged in TRAIN data
```

6.3.2 HIGH BMI FLAG

[30]:
```
### DEFINE THRESHOLD
bmi_threshold = 30  # Obese category

### CREATE FLAGS
train_clean['high_bmi_flag'] = (train_clean['bmi'] >= bmi_threshold).astype(int)
test_clean['high_bmi_flag'] = (test_clean['bmi'] >= bmi_threshold).astype(int)

### SHOW FLAG LOGIC & COUNT FOR TRAIN DATA
print_section('CREATE HIGH BMI FLAG')
print(f'high_bmi_flag = 1 if BMI >= {bmi_threshold}; 0 otherwise')
print(f'{train_clean["high_bmi_flag"].sum()} patients␣
  ↪({train_clean["high_bmi_flag"].mean()*100:.1f}%) flagged in TRAIN data')
```

```
============================================================
CREATE HIGH BMI FLAG
============================================================
high_bmi_flag = 1 if BMI >= 30; 0 otherwise
379 patients (61.7%) flagged in TRAIN data
```

6.3.3 HIGH BLOOD PRESSURE FLAG

- Based on https://www.mayoclinic.org/diseases-conditions/high-blood-pressure/in-depth/blood-pressure/art-20050982

[31]:
```
### DEFINE THRESHOLD
bp_threshold = 80  # mmHg (hypertension stage 1)

### CREATE FLAGS
train_clean['high_bp_flag'] = (train_clean['blood_pressure'] >= bp_threshold).
  ↪astype(int)
test_clean['high_bp_flag'] = (test_clean['blood_pressure'] >= bp_threshold).
  ↪astype(int)

### SHOW FLAG LOGIC & COUNT FOR TRAIN DATA
print_section('CREATE HIGH BLOOD PRESSURE FLAG')
print(f'high_bp_flag = 1 if BP >= {bp_threshold} mmHg; 0 otherwise')
print(f'{train_clean["high_bp_flag"].sum()} patients␣
  ↪({train_clean["high_bp_flag"].mean()*100:.1f}%) flagged in TRAIN data')
```

```
============================================================
CREATE HIGH BLOOD PRESSURE FLAG
============================================================
high_bp_flag = 1 if BP >= 80 mmHg; 0 otherwise
168 patients (27.4%) flagged in TRAIN data
```

6.3.4 HIGH AGE FLAG

[32]:
```
### DEFINE THRESHOLD
age_threshold = 40

### CREATE FLAGS
train_clean['high_age_flag'] = (train_clean['age'] >= age_threshold).astype(int)
test_clean['high_age_flag'] = (test_clean['age'] >= age_threshold).astype(int)

### SHOW FLAG LOGIC & COUNT FOR TRAIN DATA
print_section('CREATE HIGH AGE FLAG')
print(f'high_age_flag = 1 if age >= {age_threshold}; 0 otherwise')
print(f'{train_clean["high_age_flag"].sum()} patients␣
  ↪({train_clean["high_age_flag"].mean()*100:.1f}%) flagged in TRAIN data')
```

```
============================================================
CREATE HIGH AGE FLAG
============================================================
high_age_flag = 1 if age >= 40; 0 otherwise
167 patients (27.2%) flagged in TRAIN data
```

6.3.5 COMBINED RISK SCORE

[33]:
```
### SUM THE HIGH-RISK FLAGS INTO RISK SCORES COLUMNS
train_clean['risk_score'] = (train_clean['high_glucose_flag'] +
                             train_clean['high_bmi_flag'] +
                             train_clean['high_bp_flag'] +
                             train_clean['high_age_flag'])

test_clean['risk_score'] = (test_clean['high_glucose_flag'] +
                            test_clean['high_bmi_flag'] +
                            test_clean['high_bp_flag'] +
                            test_clean['high_age_flag'])

### SHOW COMBINED RISK SCORE FOR TRAIN DATA
print_section('CREATE COMBINED RISK SCORE')
print('risk_score: Sum of all high-risk flags (0-4)')
print(f'Mean risk score for train: {train_clean["risk_score"].mean():.2f}')

print(f'\nDISTRIBUTION IN TRAIN DATA:')
print(train_clean['risk_score'].value_counts().sort_index())

print('\n  High-risk flags created successfully!')
```

```
============================================================
CREATE COMBINED RISK SCORE
============================================================
risk_score: Sum of all high-risk flags (0-4)
Mean risk score for train: 1.54
```

```
DISTRIBUTION IN TRAIN DATA:
risk_score
0    127
1    187
2    172
3     95
4     33
Name: count, dtype: int64

 High-risk flags created successfully!
```

6.4 CREATE POLYNOMIAL FEATURES

[34]:
```python
print_section('CREATE SQUARED FEATURES FOR NON-LINEAR RELATIONSHIPS')

### GLUCOSE SQUARED
train_clean['glucose_squared'] = train_clean['glucose_concentration'] ** 2
test_clean['glucose_squared'] = test_clean['glucose_concentration'] ** 2
print('glucose_squared: Glucose concentration squared')

### BMI SQUARED
train_clean['bmi_squared'] = train_clean['bmi'] ** 2
test_clean['bmi_squared'] = test_clean['bmi'] ** 2
print('bmi_squared: BMI squared')

### AGE SQUARED
train_clean['age_squared'] = train_clean['age'] ** 2
test_clean['age_squared'] = test_clean['age'] ** 2
print('age_squared: Age squared')

print('\n  Polynomial features created successfully!')
```

```
============================================================
CREATE SQUARED FEATURES FOR NON-LINEAR RELATIONSHIPS
============================================================
glucose_squared: Glucose concentration squared
bmi_squared: BMI squared
age_squared: Age squared

 Polynomial features created successfully!
```

6.5 CREATE LOG-TRANSFORMED FEATURES

[35]:
```python
print_section('CREATE LOG-TRANSFORMED FEATURES FOR SKEWED FEATURES')

### DIABETES PEDIGREE
train_clean['pedigree_log'] = np.log1p(train_clean['diabetes pedigree'])
```

```
test_clean['pedigree_log'] = np.log1p(test_clean['diabetes pedigree'])
print('pedigree_log: Log-transformed diabetes pedigree function')

print('\n  Log-transformed features created successfully!')
```

```
============================================================
CREATE LOG-TRANSFORMED FEATURES FOR SKEWED FEATURES
============================================================
pedigree_log: Log-transformed diabetes pedigree function

 Log-transformed features created successfully!
```

6.6 FEATURE ENGINEERING SUMMARY

[36]:
```
print_section('FEATURE ENGINEERING SUMMARY')

### FEATURE BEFORE AND AFTER FEATURE ENGINEERING
original_features = list(train_before_FE.columns)
new_features = [col for col in train_clean.columns if col not in␣
  ↪original_features]

print(f'\n--- FEATURE COUNTS ---')
print(f'Original feature count: {len(original_features)}')
print(f'New features created: {len(new_features)}')
print(f'Total feature count: {len(train_clean.columns)}')

### NEW FEATURES
print(f'\n--- NEW FEATURES CREATED ---')
for i, feat in enumerate(new_features, 1):
    print(f'{i:2d}. {feat}')

### DISPLAY SAMPLE VALUES OF ENGINEERED FEATURES
print(f'\n--- SAMPLE VALUES OF ENGINEERED FEATURES ---')
display(train_clean[new_features].head())

print('\n  FEATURE ENGINEERING COMPLETED!')
print('Ready to proceed with model training.\n')
```

```
============================================================
FEATURE ENGINEERING SUMMARY
============================================================

--- FEATURE COUNTS ---
Original feature count: 8
New features created: 15
Total feature count: 23

--- NEW FEATURES CREATED ---
```

```
 1. bmi_category
 2. age_group
 3. glucose_category
 4. glucose_bmi_ratio
 5. bmi_age_interaction
 6. pregnancy_age_interaction
 7. high_glucose_flag
 8. high_bmi_flag
 9. high_bp_flag
10. high_age_flag
11. risk_score
12. glucose_squared
13. bmi_squared
14. age_squared
15. pedigree_log

--- SAMPLE VALUES OF ENGINEERED FEATURES ---

   bmi_category  age_group  glucose_category  glucose_bmi_ratio  \
0             3          0                 1           3.284457
1             3          3                 2           3.907104
2             3          0                 0           2.447368
3             3          2                 1           3.084833
4             3          0                 2           3.950617

   bmi_age_interaction  pregnancy_age_interaction  high_glucose_flag  \
0                886.6                         52                  0
1               1866.6                        561                  1
2                874.0                         46                  0
3               1594.9                         41                  0
4                874.8                         81                  1

   high_bmi_flag  high_bp_flag  high_age_flag  risk_score  glucose_squared  \
0              1             0              0           1            12544
1              1             1              1           4            20449
2              1             0              0           1             8649
3              1             1              1           3            14400
4              1             0              0           2            16384

   bmi_squared  age_squared  pedigree_log
0      1162.81          676      0.273837
1      1339.56         2601      0.226338
2      1444.00          529      0.515216
3      1513.21         1681      0.771034
4      1049.76          729      0.437610


  FEATURE ENGINEERING COMPLETED!
Ready to proceed with model training.
```

[37]:
```python
### DISPLAY FINAL DATASETS READY FOR ML
print_section('FINAL DATASETS READY FOR ML')
print(f'Train shape: {train_clean.shape}')
print(f'Test shape: {test_clean.shape}')
print(f'\nFirst 5 rows of TRAIN data:')
display(train_clean.head())
```

```
============================================================
FINAL DATASETS READY FOR ML
============================================================
Train shape: (614, 23)
Test shape: (154, 22)

First 5 rows of TRAIN data:

   p_id  no_times_pregnant  glucose_concentration  blood_pressure   bmi  \
0   316                  2                    112              68  34.1
1    25                 11                    143              94  36.6
2   710                  2                     93              64  38.0
3   658                  1                    120              80  38.9
4   542                  3                    128              72  32.4

   diabetes pedigree  age  diabetes  bmi_category  age_group  \
0              0.315   26         0             3          0
1              0.254   51         1             3          3
2              0.674   23         1             3          0
3              1.162   41         0             3          2
4              0.549   27         1             3          0

   glucose_category  glucose_bmi_ratio  bmi_age_interaction  \
0                 1           3.284457                886.6
1                 2           3.907104               1866.6
2                 0           2.447368                874.0
3                 1           3.084833               1594.9
4                 2           3.950617                874.8

   pregnancy_age_interaction  high_glucose_flag  high_bmi_flag  high_bp_flag  \
0                         52                  0              1             0
1                        561                  1              1             1
2                         46                  0              1             0
3                         41                  0              1             1
4                         81                  1              1             0

   high_age_flag  risk_score  glucose_squared  bmi_squared  age_squared  \
0              0           1            12544      1162.81          676
1              1           4            20449      1339.56         2601
2              0           1             8649      1444.00          529
```

```
3                 1             3             14400       1513.21          1681
4                 0             2             16384       1049.76           729

   pedigree_log
0      0.273837
1      0.226338
2      0.515216
3      0.771034
4      0.437610
```

7 MACHINE LEARNING

- Goal: Train classification models and select the best one.
- Algorithms: k-Nearest Neighbors, Decision Tree, Random Forest.

[38]:
```
### MAKE COPIES TO PRESERVE DATA
train_before_ML = train_clean.copy()
test_before_ML = test_clean.copy()
```

7.1 PREP DATA FOR MODELING

[39]:
```
print_section('PREP DATA FOR MODELING')

### DROP NON-PREDICTIVE IDENTIFIER (p_id) AND SEPARATE FEATURES (X) FROM TARGET␣
↪(y)
X = train_clean.drop(['p_id', 'diabetes'], axis=1)
y = train_clean['diabetes']
X_test_final = test_clean.drop(['p_id'], axis=1) # Reserve for final predictions

### SPLIT TRAIN DATA INTO 80% TRAINING & 20% VALIDATION
# Set the random_state seed (42) to ensure same random split for every run
# stratify=y to have equal proportion of each class in train & val during␣
↪split, based on y
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f'Train set: {X_train.shape}')
print(f'Validation set: {X_val.shape}')

### FEATURE SCALING
scaler = StandardScaler()
# FIT THE SCALER ON TRAIN SETS ONLY AND TRANSFORM IT
X_train_scaled = scaler.fit_transform(X_train)
# TRANSFORM THE VALIDATION AND TEST SETS USING THE SAME SCALER
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test_final)
```

```
print('\n  Data prepared for modeling!')
```

```
============================================================
PREP DATA FOR MODELING
============================================================
Train set: (491, 21)
Validation set: (123, 21)

  Data prepared for modeling!
```

7.2 TRAIN CLASSIFICATION MODELS

[40]:
```
### INITIALIZE MODELS IN A DICTIONARY
models = {
    'k-Nearest Neighbors': KNeighborsClassifier(n_neighbors=5),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42, n_estimators=100)
}

### DEFINE AN EMPTY LIST TO STORE THE TRAINING RESULTS
results = []

### LOOP THROUGH EACH MODEL
for name, model in models.items():
    ### SHOW WHAT MODEL IS BEING TRAINED
    print(f'\nTraining {name}...')

    ### TRAIN MODEL
    model.fit(X_train_scaled, y_train)

    ### USE TRAINED MODEL TO PREDICT THE TARGET (y) IN THE VALIDATION SET
    y_pred = model.predict(X_val_scaled)

    ### CALCULATE METRICS
    accuracy = accuracy_score(y_val, y_pred) # % of correct predictions overall
    precision = precision_score(y_val, y_pred) # % of predicted positives that␣
↪are actually positive
    recall = recall_score(y_val, y_pred) # % of actual positives correctly␣
↪predicted
    f1 = f1_score(y_val, y_pred) # Harmonic mean of precision & recall␣
↪(balances both)

    ### STORE RESULTS
    results.append({
        'Model': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1
    })

    ### SHOW REPORT
    print(f'  {name} trained')
    print(f'- Accuracy: {accuracy:.3f}')
    print(f'- Precision: {precision:.3f}')
    print(f'- Recall: {recall:.3f}')
    print(f'- F1-Score: {f1:.3f}')

print('\n  All models trained successfully!')
```

```
Training k-Nearest Neighbors…
  k-Nearest Neighbors trained
- Accuracy: 0.756
- Precision: 0.659
- Recall: 0.628
- F1-Score: 0.643

Training Decision Tree…
  Decision Tree trained
- Accuracy: 0.748
- Precision: 0.625
- Recall: 0.698
- F1-Score: 0.659

Training Random Forest…
  Random Forest trained
- Accuracy: 0.821
- Precision: 0.733
- Recall: 0.767
- F1-Score: 0.750

  All models trained successfully!
```

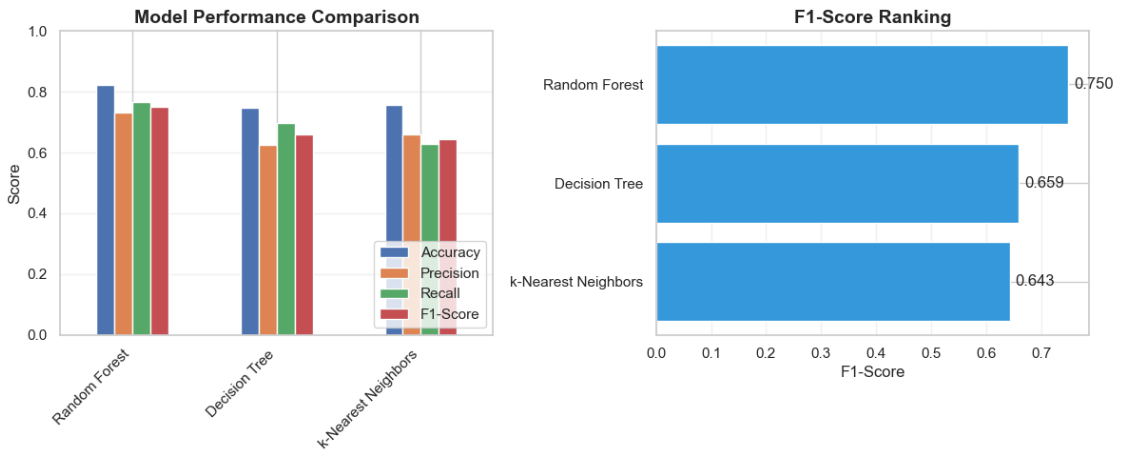
7.3 COMPARE MODEL PERFORMANCE

[41]:
```
### CREATE COMPARISON DF FROM RESULTS, SORTED BY DESCENDING F1-Score
results_df = pd.DataFrame(results)
results_df = results_df.sort_values('F1-Score', ascending=False)

print('--- MODEL PERFORMANCE METRICS COMPARISON ---')
display(results_df)

### VISUALIZE COMPARISON IN BAR CHART
```

```python
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

### LEFT SUBPLOT axes[0] TO COMPARE ALL MODEL PERFORMANCE METRICS
# Plot on the left
ax1 = axes[0]
results_df.plot(x='Model', y=['Accuracy', 'Precision', 'Recall', 'F1-Score'],
                kind='bar', ax=ax1)

# Set plot annotations
ax1.set_title('Model Performance Comparison', fontsize=14, fontweight='bold')
ax1.set_ylabel('Score', fontsize=12)
ax1.set_xlabel('')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=45, ha='right')
ax1.legend(loc='lower right')
ax1.grid(axis='y', alpha=0.3)
ax1.set_ylim([0, 1])

### RIGHT SUBPLOT axes[1] TO COMPARE F1-Score ONLY
# Plot on the right
ax2 = axes[1]
results_df_sorted = results_df.sort_values('F1-Score')

# Set plot annotations
ax2.barh(results_df_sorted['Model'], results_df_sorted['F1-Score'],␣
 ↪color='#3498db')
ax2.set_title('F1-Score Ranking', fontsize=14, fontweight='bold')
ax2.set_xlabel('F1-Score', fontsize=12)
ax2.grid(axis='x', alpha=0.3)

# Add F1-Score value labels
for i, v in enumerate(results_df_sorted['F1-Score']):
    ax2.text(v + 0.01, i, f'{v:.3f}', va='center')

plt.tight_layout()
plt.show()

# Identify best model
best_model_name = results_df.iloc[0]['Model']
best_f1 = results_df.iloc[0]['F1-Score']

print(f'\n  BEST MODEL: {best_model_name}')
print(f'   F1-Score: {best_f1:.3f}')
```

```
--- MODEL PERFORMANCE METRICS COMPARISON ---

                Model  Accuracy  Precision    Recall  F1-Score
2        Random Forest  0.821138   0.733333  0.767442  0.750000
1        Decision Tree  0.747967   0.625000  0.697674  0.659341
```

```
0  k-Nearest Neighbors  0.756098   0.658537  0.627907  0.642857
```



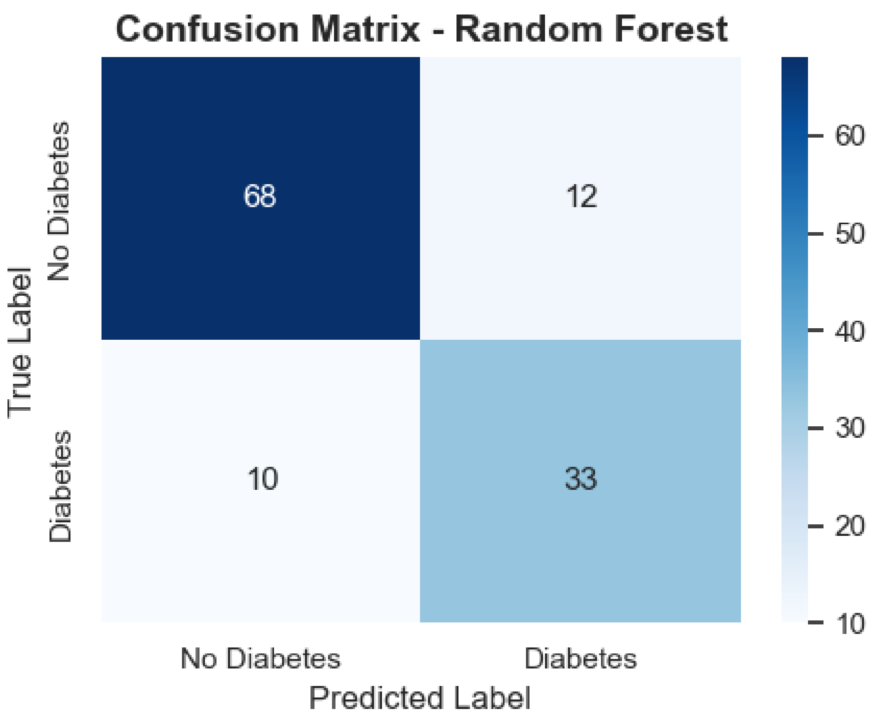
```
 BEST MODEL: Random Forest
  F1-Score: 0.750
```

7.4 EVALUATE BEST MODEL

[42]:
```
### GET BEST MODEL AND ITS PREDICTIONS ON y IN THE VALIDATION SET
best_model = models[best_model_name]
y_pred_best = best_model.predict(X_val_scaled)

### CONFUSION MATRIX TO SEE WHERE THE MODEL IS MAKING MISTAKES
print(f'--- CONFUSION MATRIX: {best_model_name} ---')
cm = confusion_matrix(y_val, y_pred_best)

### VISUALIZE CONFUSION MATRIX
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', # annot=True adds the values;
 ↪ fmt='d' present them as int
            xticklabels=['No Diabetes', 'Diabetes'],
            yticklabels=['No Diabetes', 'Diabetes'])

# Set plot annotations
plt.title(f'Confusion Matrix - {best_model_name}', fontsize=14,␣
 ↪fontweight='bold')
plt.ylabel('True Label', fontsize=12)
plt.xlabel('Predicted Label', fontsize=12)
plt.tight_layout()
plt.show()

### CLASSIFICATION REPORT
```

```python
# Precision: Of all patients predicted as diabetics, how many actually are?
# Recall: Of all actual diabetics, how many were correctly identified?
# F1-Score: Balances precision and recall - important when false negatives are␣
↪costly.
# Support: Number of true samples for each class, gives context to the metrics
print(f'\n--- Classification Report: {best_model_name} ---')
print(classification_report(y_val, y_pred_best, target_names=['No Diabetes',␣
↪'Diabetes']))
```

```
--- CONFUSION MATRIX: Random Forest ---
```



```
--- Classification Report: Random Forest ---
              precision    recall  f1-score   support

 No Diabetes       0.87      0.85      0.86        80
    Diabetes       0.73      0.77      0.75        43

    accuracy                           0.82       123
   macro avg       0.80      0.81      0.81       123
weighted avg       0.82      0.82      0.82       123
```

CONFUSION MATRIX - Most non-diabetic patients are correctly identified (68/80). - Some diabetic patients are missed (10/43).

CLASSIFICATION REPORT - Accuracy: 0.81 => 82% of predictions are correct overall. - Macro avg (unweighted): 0.81 => average of F1 across classes (diabetics & non-diabetics), treats both classes equally. - Weighted avg: 0.82 => average of F1 weighted by class size, slightly favors the majority class (non-diabetics). - F1-Score for diabetic patients (0.75) is lower than for non-diabetic (0.86) => the model struggles more with detecting diabetics.

7.5 MAKE PREDICTIONS ON TEST DATASET USING BEST MODEL

[43]:
```
### MAKE PREDICTIONS ON TEST DATASET USING BEST MODEL
test_predictions = best_model.predict(X_test_scaled)

### CREATE A DF OF PREDICTION RESULTS
predictions_df = pd.DataFrame({
    'p_id': test_clean['p_id'],
    'predicted_diabetes': test_predictions
})

print(f"  Predictions completed for {len(predictions_df)} test samples")
print('Prediction distribution:')
print(predictions_df['predicted_diabetes'].value_counts())

### SHOW SAMPLE PREDICTIONS
print('\nSample predictions:')
display(predictions_df.head())
```

```
  Predictions completed for 154 test samples
Prediction distribution:
predicted_diabetes
0    97
1    57
Name: count, dtype: int64

Sample predictions:
```

```
   p_id  predicted_diabetes
0   437                   1
1   411                   0
2   639                   0
3   213                   1
4   181                   0
```

7.6 ML SUMMARY

[44]:
```
print_section('ML RESULTS SUMMARY')
print(f'''- ALGORITHMS TESTED: {', '.join(models)}
- BEST MODEL: {best_model_name}
- PERFORMANCE (VALIDATION SET):
    - Accuracy:  {results_df.iloc[0]["Accuracy"]:.4f}
    - Precision: {results_df.iloc[0]["Precision"]:.4f}
    - Recall:    {results_df.iloc[0]["Recall"]:.4f}
    - F1-Score:  {results_df.iloc[0]["F1-Score"]:.4f}
- TEST PREDICTIONS:
   - Total samples: {len(predictions_df)}
   - Predictions saved in: predictions_df''')

print('\n  ML COMPLETED!')
print('  PROJECT COMPLETED!')
```

```
============================================================
ML RESULTS SUMMARY
============================================================
- ALGORITHMS TESTED: k-Nearest Neighbors, Decision Tree, Random Forest
- BEST MODEL: Random Forest
- PERFORMANCE (VALIDATION SET):
    - Accuracy:  0.8211
    - Precision: 0.7333
    - Recall:    0.7674
    - F1-Score:  0.7500
- TEST PREDICTIONS:
   - Total samples: 154
   - Predictions saved in: predictions_df

  ML COMPLETED!
  PROJECT COMPLETED!
```