In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.patches as patches
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [14]:
```
# To show all columns when printing
pd.set_option("display.max_columns", None)

df = pd.read_csv(r"/Users/maniisshhhh/Downloads/Healthcare.csv")
df
```

Out[14]:

	Age	Gender	BMI	Glucose	BloodPressure	Insulin	Ski
0	56	Female	27.0	76.5	68.90	16.40	8.9
1	69	Male	28.9	43.5	66.20	31.90	20.
2	46	Female	27.4	82.1	75.25	79.10	16.
3	32	Female	28.9	105.5	75.10	189.80	26.
4	60	Male	24.0	84.2	79.00	100.40	24.
...	...	...	...	...	...	...	...
17995	46	Female	32.3	84.3	86.10	59.50	15.
17996	31	Male	28.9	137.2	79.80	87.30	20.
17997	21	Male	26.9	108.4	69.90	79.85	32.
17998	61	Male	28.6	139.4	66.30	92.00	20.
17999	73	Male	28.0	135.4	77.80	115.60	21.

18000 rows × 30 columns

Basic Dataset Overview

In [15]:
```
print("Shape of dataset:", df.shape)
```

```
Shape of dataset: (18000, 30)
```

In [16]:
```
print("\nFirst 5 rows:")
print(df.head())
```

```
First 5 rows:
   Age  Gender   BMI  Glucose  BloodPressure  Insulin
SkinThickness  \
0   56  Female  27.0     76.5          68.90     16.4
8.9
1   69    Male  28.9     43.5          66.20     31.9
20.5
2   46  Female  27.4     82.1          75.25     79.1
16.1
3   32  Female  28.9    105.5          75.10    189.8
26.2
4   60    Male  24.0     84.2          79.00    100.4
24.2

   Pregnancies  DiabetesPedigreeFunction PhysicalActivityLevel
Outcome  \
0            2                     1.181              Moderate
0
1            2                     0.339              Moderate
0
2            2                     1.659                  High
1
3            7                     2.497              Moderate
0
4            8                     1.342                   Low
1

        Country  Cholesterol Smoking_Status Family_History
Diet_Type  \
0     Australia        189.8            Yes             No
High-Fat
1            UK        235.4            Yes            Yes
High-Carb
2  South Africa        143.7             No             No
Balanced
3        Canada        219.5            Yes             No
Balanced
4            UK        219.0             No            Yes
High-Fat

   Sleep_Hours  Daily_Steps  Water_Intake_Liters
Mental_Health_Status  \
0          8.5       4878.0                  3.7
Good
1          4.9       5604.0                  3.1
Good
2          7.0       7533.0                  2.6
Good
3          6.2       5963.0                  2.6
```

```
Poor
4          8.0       3201.0                  2.6
Average

  Sleep_Quality Chronic_Disease Work_Stress_Level
Alcohol_Consumption  \
0          Fair         Thyroid              High
Regular
1          Fair   Heart Disease          Moderate
Occasional
2          Good         Thyroid          Moderate
Regular
3          Fair    Hypertension               Low
Occasional
4          Good         Thyroid          Moderate
Regular

  Diabetes_Status Diabetes_Type Fast_Food_Intake Prediabetes  \
0     No Diabetes        Type 2           Rarely          No
1     No Diabetes        Type 2           Rarely         Yes
2        Diabetes        Type 2        Sometimes          No
3     No Diabetes        Type 2       Frequently          No
4        Diabetes        Type 2       Frequently          No

   Fasting_Blood_Sugar  Screen_Time
0                185.4          8.9
1                231.7          1.9
2                 64.8         10.7
3                 89.1          6.3
4                137.7          4.7
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['Age', 'Gender', 'BMI', 'Glucose', 'BloodPressure',
'Insulin',
       'SkinThickness', 'Pregnancies',
'DiabetesPedigreeFunction',
       'PhysicalActivityLevel', 'Outcome', 'Country',
'Cholesterol',
       'Smoking_Status', 'Family_History', 'Diet_Type',
'Sleep_Hours',
       'Daily_Steps', 'Water_Intake_Liters',
'Mental_Health_Status',
       'Sleep_Quality', 'Chronic_Disease', 'Work_Stress_Level',
       'Alcohol_Consumption', 'Diabetes_Status',
'Diabetes_Type',
       'Fast_Food_Intake', 'Prediabetes',
'Fasting_Blood_Sugar',
       'Screen_Time'],
      dtype='object')
```

In [18]:
```
df.dtypes
```

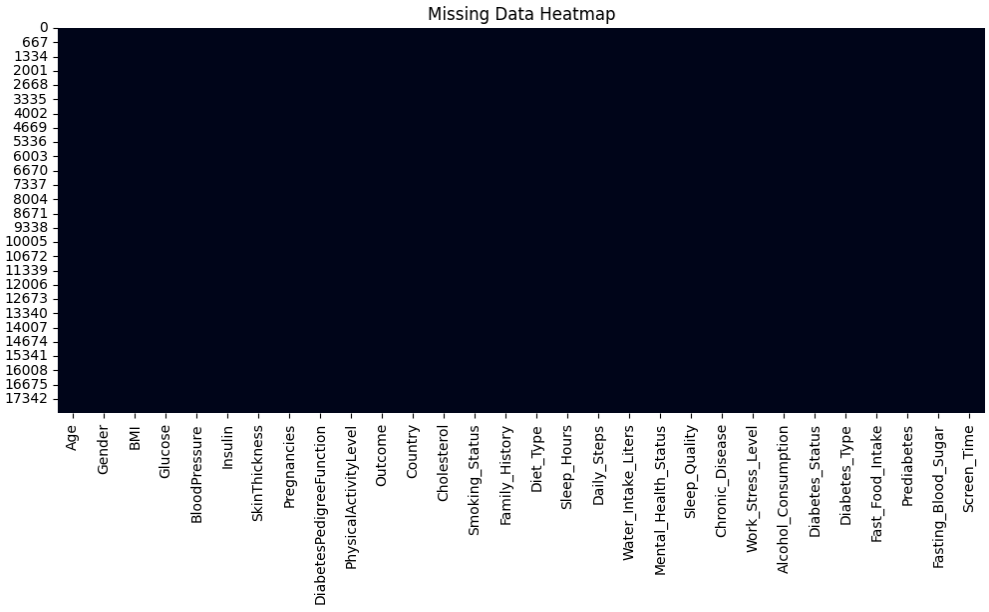
Out[18]:
```
Age                          int64
Gender                      object
BMI                        float64
Glucose                    float64
BloodPressure              float64
Insulin                    float64
SkinThickness              float64
Pregnancies                  int64
DiabetesPedigreeFunction   float64
PhysicalActivityLevel       object
Outcome                      int64
Country                     object
Cholesterol                float64
Smoking_Status              object
Family_History              object
Diet_Type                   object
Sleep_Hours                float64
Daily_Steps                float64
Water_Intake_Liters        float64
Mental_Health_Status        object
Sleep_Quality               object
Chronic_Disease             object
Work_Stress_Level           object
Alcohol_Consumption         object
Diabetes_Status             object
Diabetes_Type               object
Fast_Food_Intake            object
Prediabetes                 object
Fasting_Blood_Sugar        float64
Screen_Time                float64
dtype: object
```

Check Missing Values

In [19]:
```
missing = df.isnull().sum()
missing_percent = (missing / len(df)) * 100

missing_summary = pd.DataFrame({
    "Missing Values": missing,
    "Missing %": missing_percent.round(2)
})

print("\nMissing Values Summary:")
print(
    missing_summary[missing_summary["Missing Values"] > 0]
    .sort_values("Missing %", ascending=False)
)
```

```
Missing Values Summary:
                     Missing Values  Missing %
BMI                               5       0.03
Glucose                           5       0.03
Sleep_Hours                       5       0.03
Daily_Steps                       5       0.03
Water_Intake_Liters               5       0.03
```

In [20]:
```
plt.figure(figsize=(12,5))
sns.heatmap(df.isnull(), cbar=False)
plt.title("Missing Data Heatmap")
plt.show()
```



In [21]:
```
print("\nSummary Statistics (Numeric Columns):")
df.describe()
```

```
Summary Statistics (Numeric Columns):
```

Out[21]:

	Age	BMI	Glucose	BloodPressure
count	18000.000000	17995.000000	17995.000000	18000.000000
mean	48.591056	28.015793	109.903351	75.112711
std	17.629407	5.899675	34.562562	11.836675
min	18.000000	4.900000	-46.300000	25.100000
25%	34.000000	24.100000	87.100000	67.200000
50%	48.000000	28.200000	108.700000	75.250000
75%	64.000000	32.000000	132.900000	83.000000
max	79.000000	54.900000	240.500000	119.900000

Data Type Cleaning Summary

In [22]:
```
num_cols = df.select_dtypes(include=["int64","float64"]).columns
cat_cols = df.select_dtypes(include="object").columns

# Fill numeric with median
for col in num_cols:
    df[col].fillna(df[col].median(), inplace=True)

# Fill categorical with mode
for col in cat_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)

print("Missing values after filling:\n", df.isnull().sum().sum())
```

```
Missing values after filling:
 0
```

In [23]:
```
categorical_cols = df.select_dtypes(include="object").columns
print("\nCategorical Columns:", list(categorical_cols))

# Explore categorical columns
for col in categorical_cols:
    print(f"\nColumn: {col}")
    print("Unique values:", df[col].nunique())
    print(df[col].value_counts().head())
```

```
Categorical Columns: ['Gender', 'PhysicalActivityLevel',
'Country', 'Smoking_Status', 'Family_History', 'Diet_Type',
'Mental_Health_Status', 'Sleep_Quality', 'Chronic_Disease',
'Work_Stress_Level', 'Alcohol_Consumption', 'Diabetes_Status',
'Diabetes_Type', 'Fast_Food_Intake', 'Prediabetes']

Column: Gender
Unique values: 3
Gender
Female    9374
Male      8621
Other        5
Name: count, dtype: int64

Column: PhysicalActivityLevel
Unique values: 3
PhysicalActivityLevel
High        6383
Low         5811
Moderate    5806
Name: count, dtype: int64

Column: Country
Unique values: 8
Country
South Africa    2630
Brazil          2259
UK              2253
India           2205
Australia       2182
Name: count, dtype: int64

Column: Smoking_Status
Unique values: 3
Smoking_Status
No         13546
Yes         4449
unknown        5
Name: count, dtype: int64

Column: Family_History
Unique values: 2
Family_History
```

```
No     10891
Yes     7109
Name: count, dtype: int64

Column: Diet_Type
Unique values: 3
Diet_Type
High-Carb    6296
Balanced     5865
High-Fat     5839
Name: count, dtype: int64

Column: Mental_Health_Status
Unique values: 3
Mental_Health_Status
Good       9190
Average    5268
Poor       3542
Name: count, dtype: int64

Column: Sleep_Quality
Unique values: 4
Sleep_Quality
Good         4883
Excellent    4426
Poor         4368
Fair         4323
Name: count, dtype: int64

Column: Chronic_Disease
Unique values: 3
Chronic_Disease
Thyroid          9256
Heart Disease    4477
Hypertension     4267
Name: count, dtype: int64

Column: Work_Stress_Level
Unique values: 3
Work_Stress_Level
Low         6319
High        5879
Moderate    5802
Name: count, dtype: int64

Column: Alcohol_Consumption
Unique values: 2
Alcohol_Consumption
Regular       12129
Occasional     5871
Name: count, dtype: int64

Column: Diabetes_Status
```

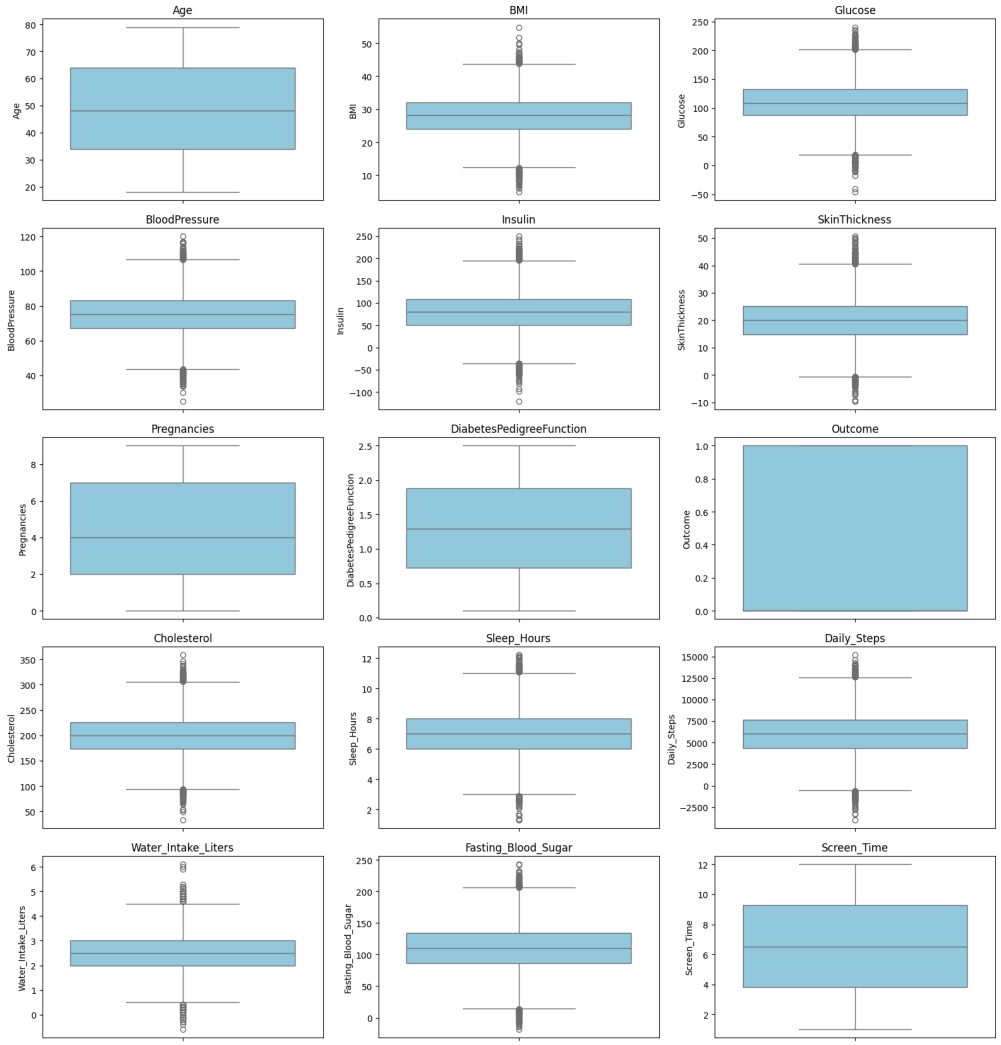
```
Unique values: 2
Diabetes_Status
No Diabetes    12845
Diabetes        5155
Name: count, dtype: int64

Column: Diabetes_Type
Unique values: 2
Diabetes_Type
Type 2    17215
Type 1      785
Name: count, dtype: int64

Column: Fast_Food_Intake
Unique values: 4
Fast_Food_Intake
Rarely        7192
Sometimes     6341
Frequently    3570
Daily          897
Name: count, dtype: int64

Column: Prediabetes
Unique values: 2
Prediabetes
No     12236
Yes     5764
Name: count, dtype: int64
```

In [24]:

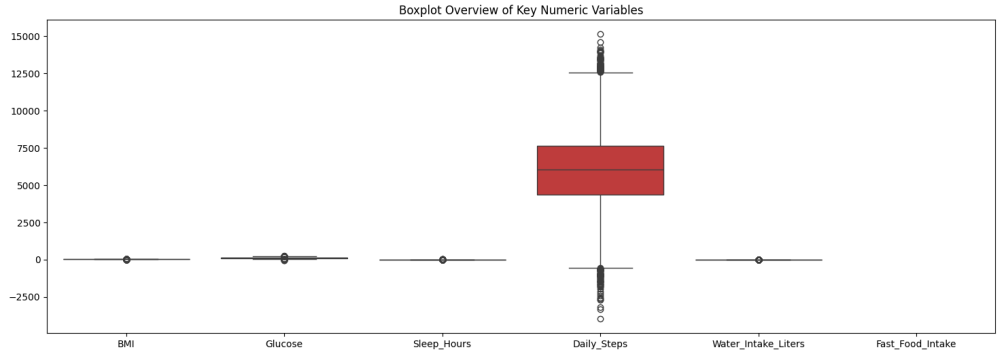
```
# Convert numeric-looking object columns to numeric
numeric_like_cols =
["BMI","Glucose","Sleep_Hours","Daily_Steps","Water_Intake_Liters","
Fast_Food_Intake"]

for col in numeric_like_cols:
    df[col] = pd.to_numeric(df[col], errors="coerce")
```

Outlier Check (Summary)

In [25]:
```
plt.figure(figsize=(16, 20))

for i, col in enumerate(num_cols, 1):
    plt.subplot(len(num_cols)//3 + 1, 3, i)
    sns.boxplot(data=df, y=col, color="skyblue")
    plt.title(col)

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

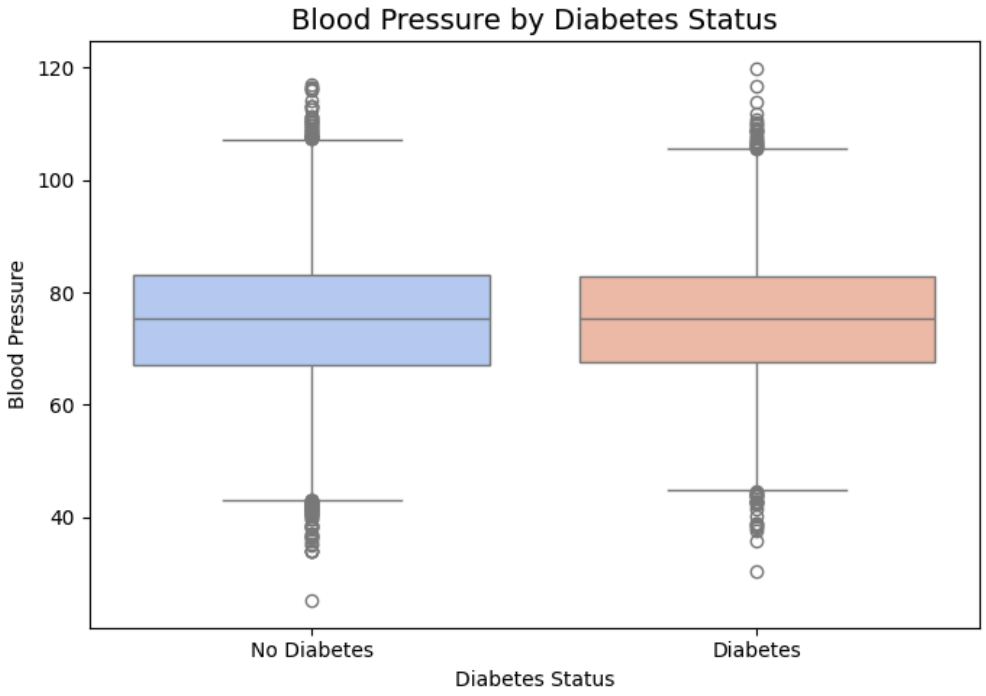


In [26]:
```
plt.figure(figsize=(18, 6))

cols =
["BMI","Glucose","Sleep_Hours","Daily_Steps","Water_Intake_Liters","
Fast_Food_Intake"]
sns.boxplot(data=df[cols])
plt.title("Boxplot Overview of Key Numeric Variables")
plt.show()
```



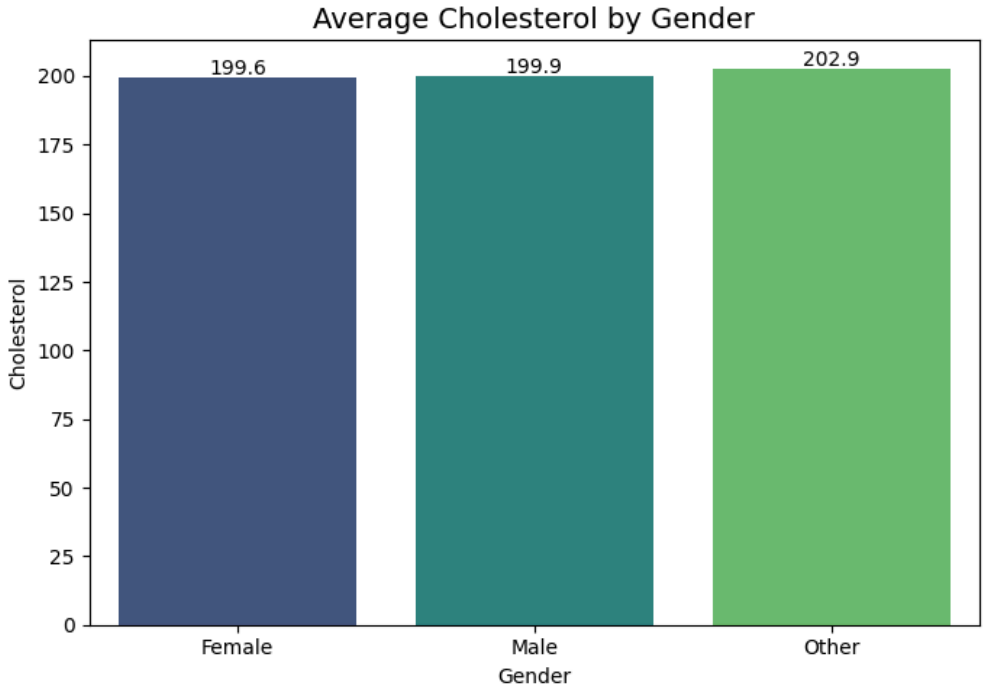
EDA

In [27]:
```
plt.figure(figsize=(7, 5))
sns.boxplot(data=df, x="Diabetes_Status", y="BloodPressure",
palette="coolwarm")
plt.title("Blood Pressure by Diabetes Status", fontsize=14)
plt.xlabel("Diabetes Status")
plt.ylabel("Blood Pressure")
plt.tight_layout()
plt.show()
```



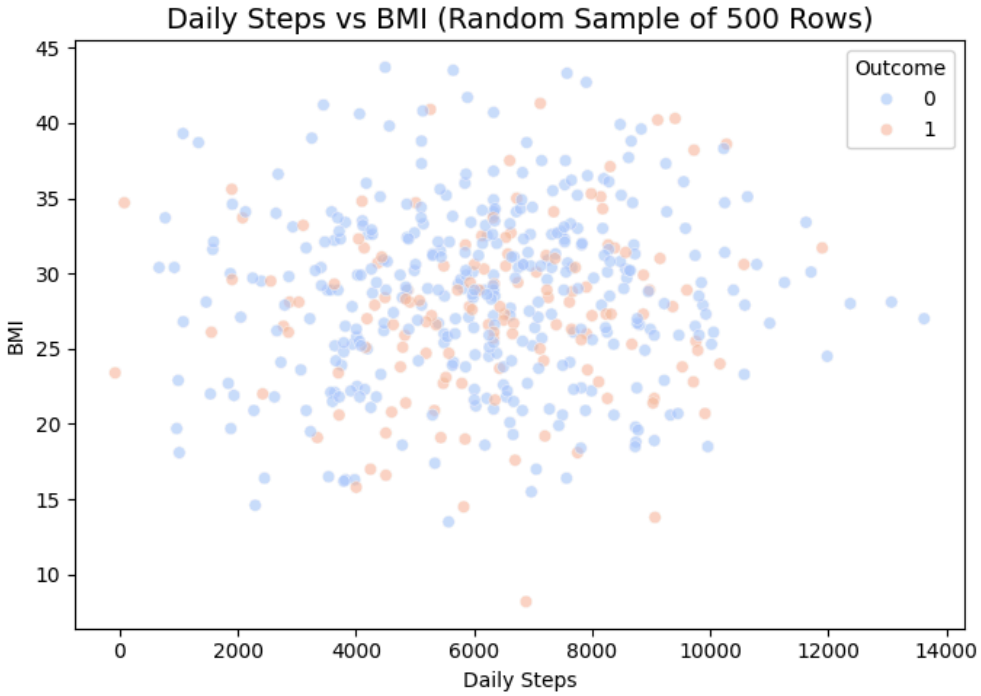
Average Cholesterol by Gender (Bar Plot)

In [28]:
```
chol_gender = df.groupby("Gender", as_index=False)
["Cholesterol"].mean()

plt.figure(figsize=(7, 5))
sns.barplot(data=chol_gender, x="Gender", y="Cholesterol",
palette="viridis")
plt.title("Average Cholesterol by Gender", fontsize=14)
plt.ylabel("Cholesterol")

for i, v in enumerate(chol_gender["Cholesterol"]):
    plt.text(i, v + 1, f"{v:.1f}", ha="center")

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



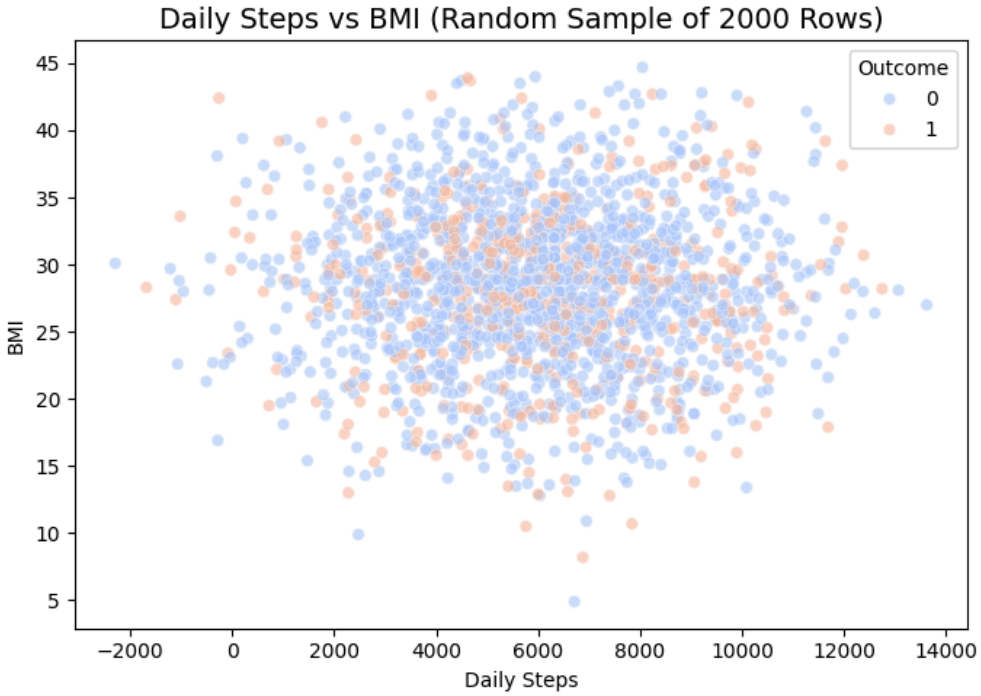
Daily Steps vs BMI (Scatter Plot)

In [29]:
```
sample_df = df.sample(500, random_state=42)

plt.figure(figsize=(7, 5))
sns.scatterplot(data=sample_df,x="Daily_Steps",y="BMI",hue="Outcome"
,palette="coolwarm",alpha=0.6)
plt.title("Daily Steps vs BMI (Random Sample of 500 Rows)",
fontsize=14)
plt.xlabel("Daily Steps")
plt.ylabel("BMI")
plt.tight_layout()
plt.show()
```



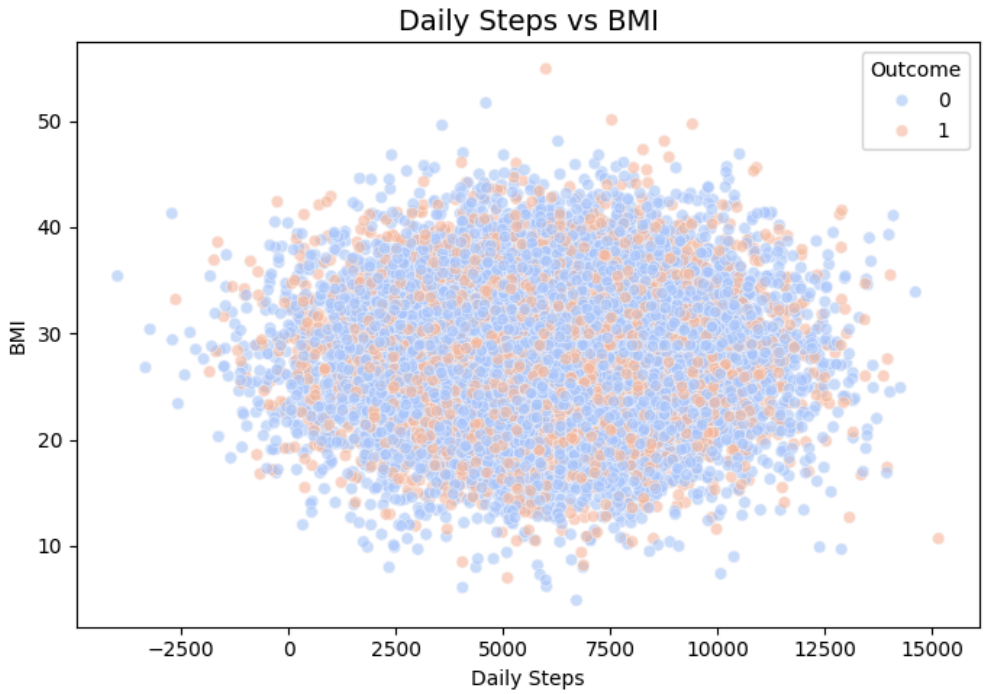
Random Sample (2000 rows)

In [30]:
```
sample_df = df.sample(2000, random_state=42)

plt.figure(figsize=(7, 5))
sns.scatterplot(data=sample_df,x="Daily_Steps",y="BMI",hue="Outcome"
,palette="coolwarm",alpha=0.6)
plt.title("Daily Steps vs BMI (Random Sample of 2000 Rows)",
fontsize=14)
plt.xlabel("Daily Steps")
plt.ylabel("BMI")
plt.tight_layout()
plt.show()
```



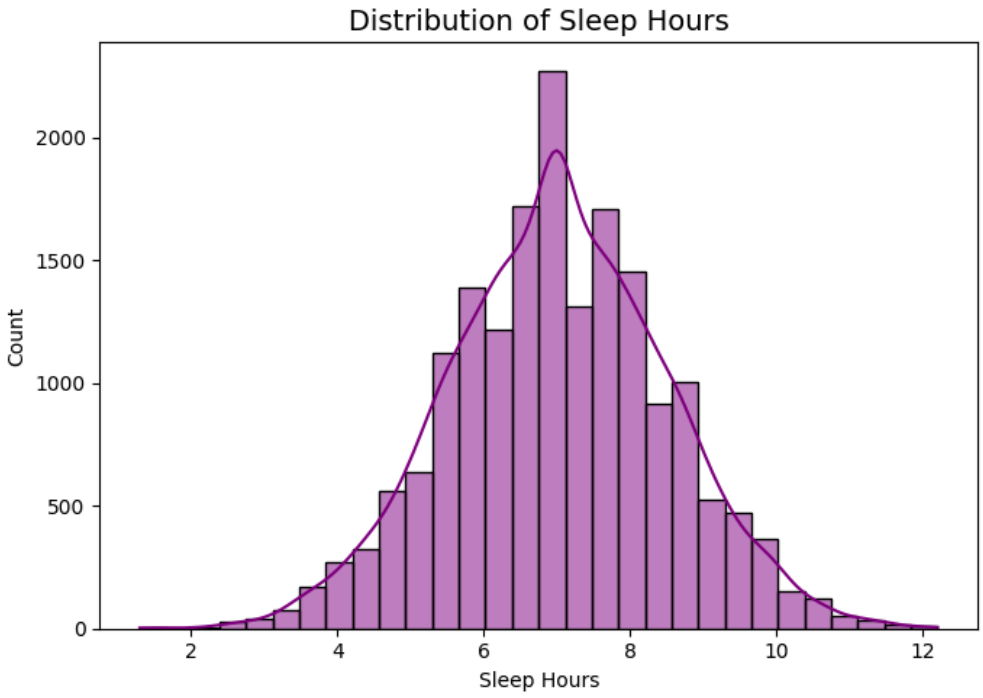
More Detailed

In [31]:
```
plt.figure(figsize=(7, 5))
sns.scatterplot(data=df, x="Daily_Steps", y="BMI", hue="Outcome",
palette="coolwarm", alpha=0.6)
plt.title("Daily Steps vs BMI", fontsize=14)
plt.xlabel("Daily Steps")
plt.ylabel("BMI")
plt.tight_layout()
plt.show()
```



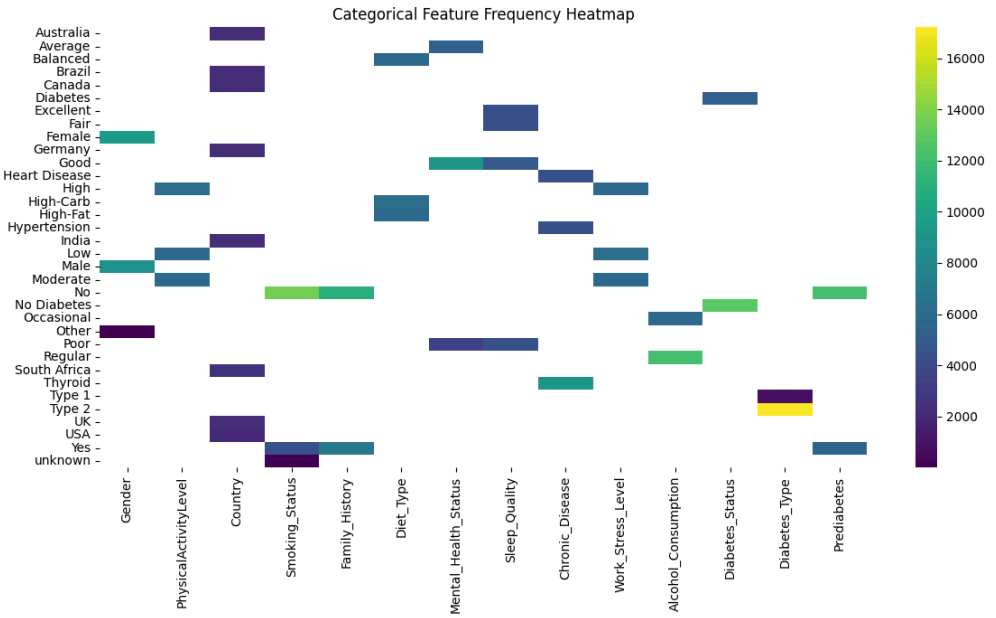
Sleep Hours Distribution (Histogram)

In [32]:
```
plt.figure(figsize=(7, 5))
sns.histplot(df["Sleep_Hours"], kde=True, bins=30, color="purple")
plt.title("Distribution of Sleep Hours", fontsize=14)
plt.xlabel("Sleep Hours")
plt.tight_layout()
plt.show()
```



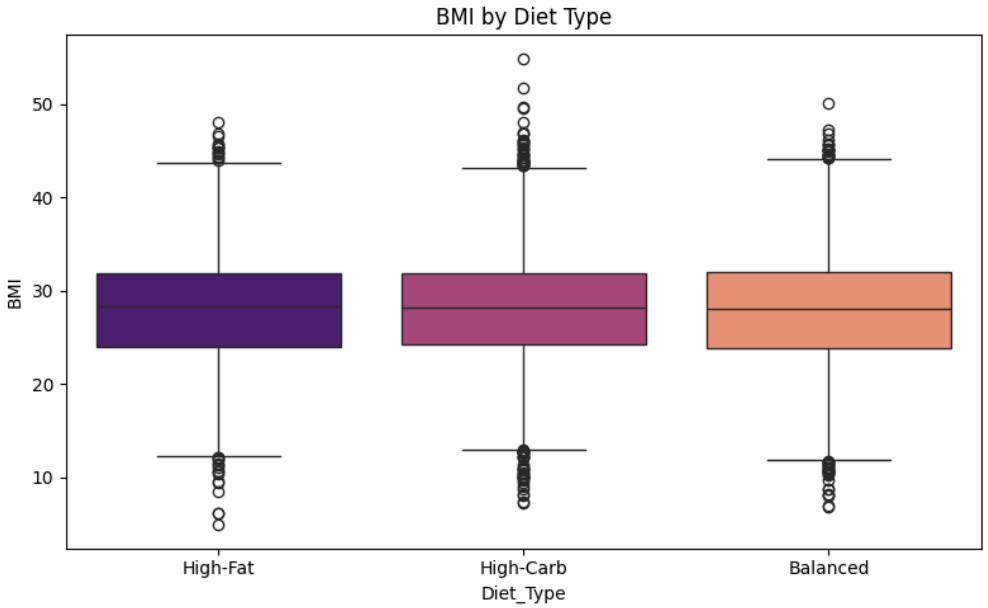
Categorical Frequency Heatmap

In [33]:
```
cat_cols = df.select_dtypes(include="object").columns

freq_df = df[cat_cols].apply(lambda x: x.value_counts())
plt.figure(figsize=(12,7))
sns.heatmap(freq_df, cmap="viridis")
plt.title("Categorical Feature Frequency Heatmap")
plt.tight_layout()
plt.show()
```



BMI by Diet Type (Box Plot)

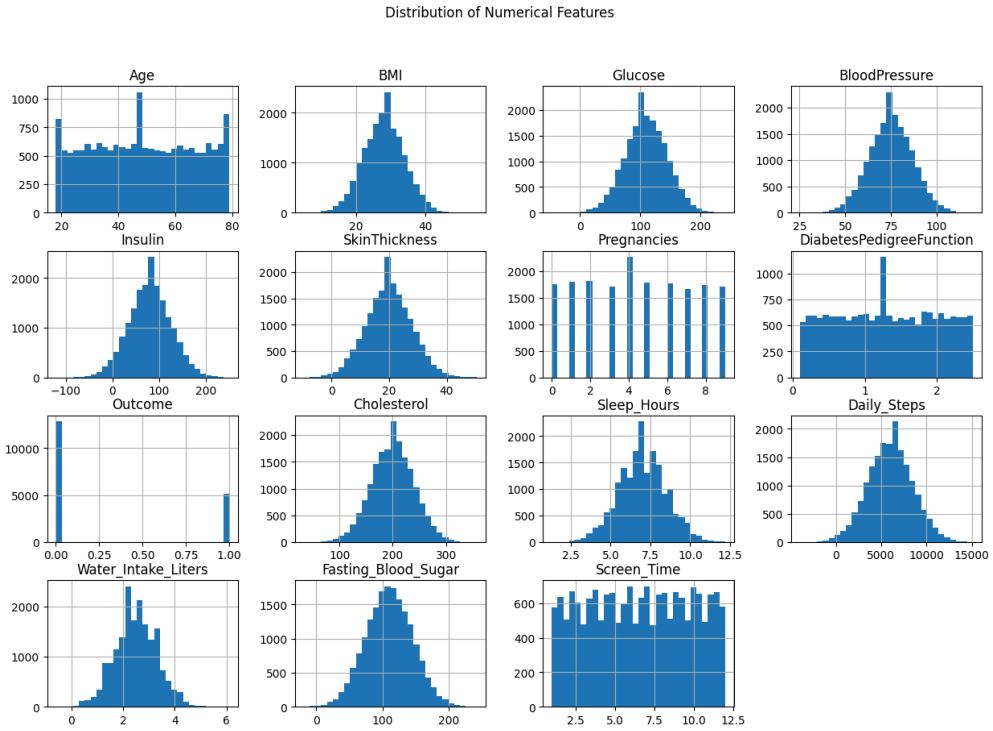
In [34]:
```
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x="Diet_Type", y="BMI", palette="magma")
plt.title("BMI by Diet Type")
plt.tight_layout()
plt.show()
```



UNIVARIATE ANALYSIS (Single variable)

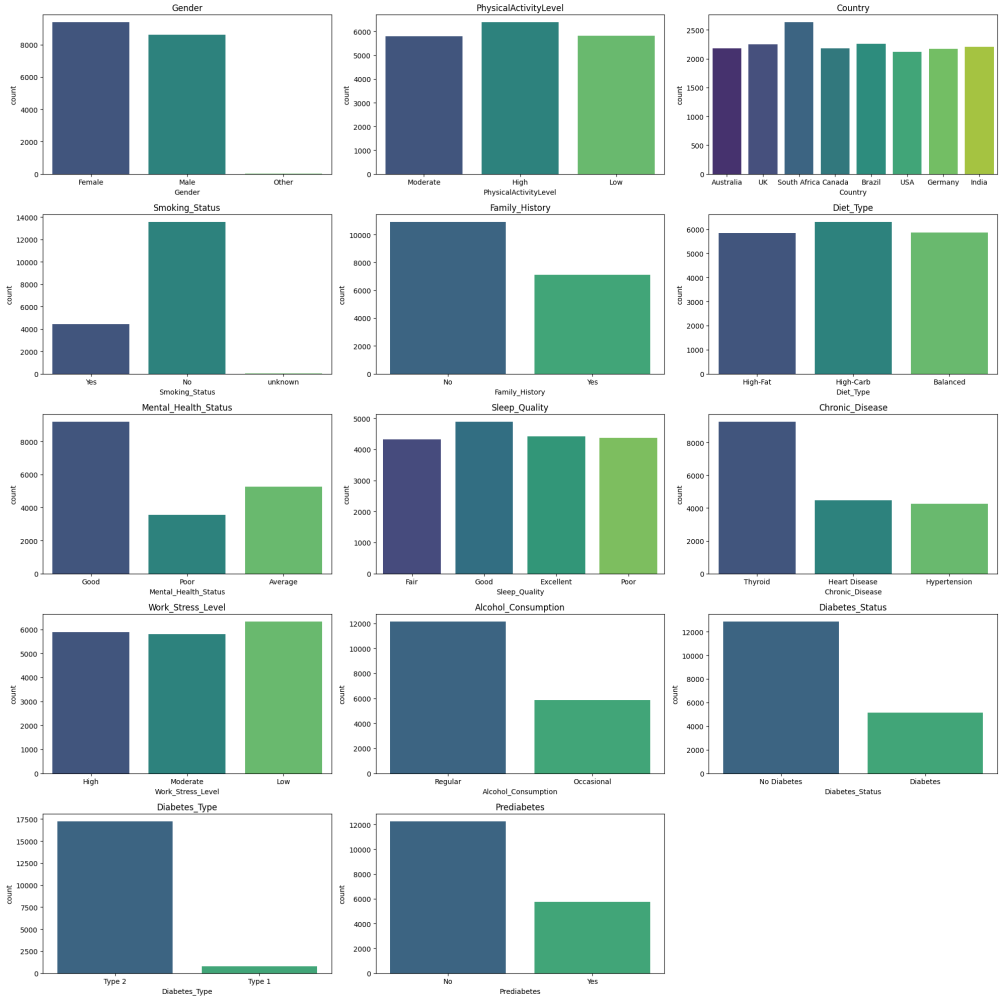
Numeric Variables (Histograms)

In [35]:
```
df[num_cols].hist(figsize=(15,10), bins=30)
plt.suptitle("Distribution of Numerical Features")
plt.show()
```



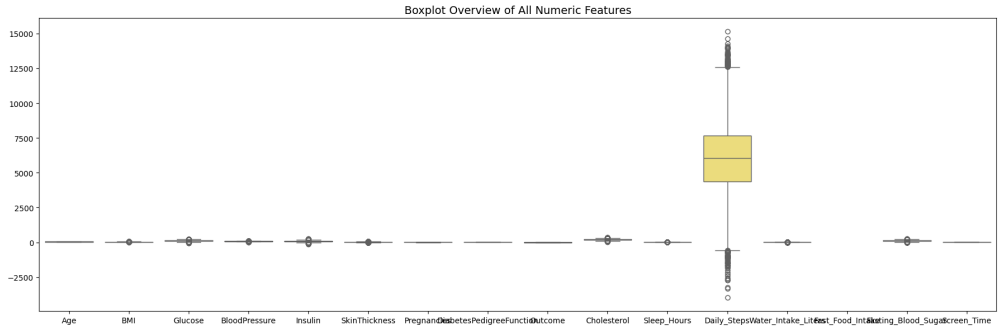
Categorical Variables (Countplots)

In [36]:
```
import math

n = len(cat_cols)
rows = math.ceil(n / 3)

plt.figure(figsize=(20,20))
for i, col in enumerate(cat_cols, 1):
    plt.subplot(rows, 3, i)
    sns.countplot(data=df, x=col, palette="viridis")
    plt.title(col)

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



Boxplots for Outlier Detection

In [37]:
```
# Select all numeric columns
num_cols = df.select_dtypes(include=["float64", "int64"]).columns

plt.figure(figsize=(18, 6))
sns.boxplot(data=df[num_cols], palette="Set3")
plt.title("Boxplot Overview of All Numeric Features", fontsize=14)
plt.tight_layout()
plt.show()
```

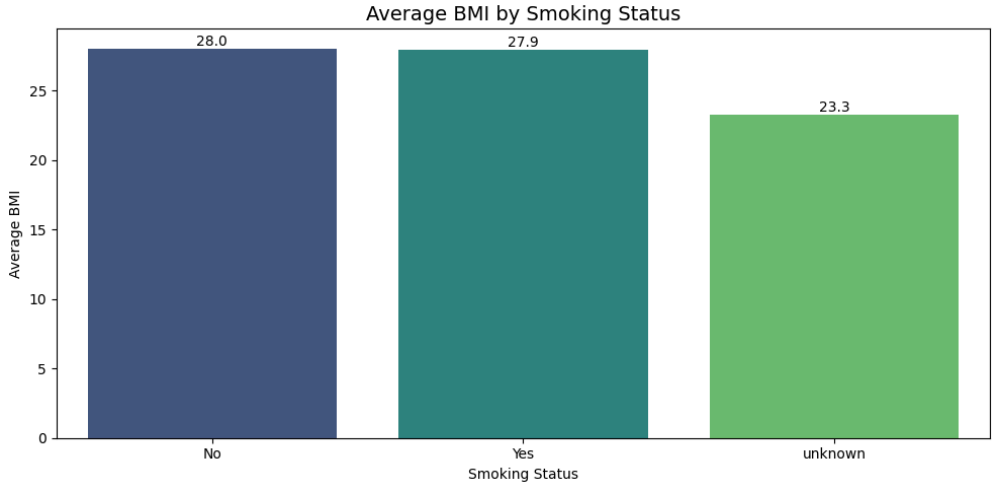


BIVARIATE ANALYSIS (Two variables)

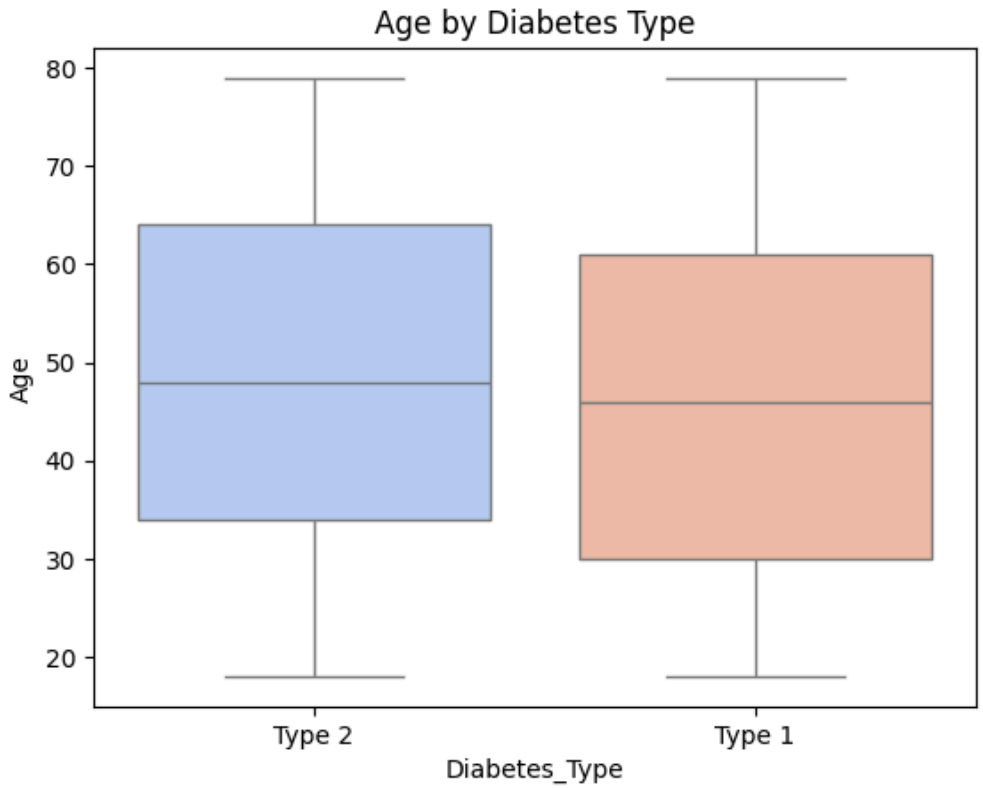
In [38]:
```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x="Mental_Health_Status", y="Sleep_Hours",
palette="viridis")

plt.title("Sleep Hours by Mental Health Status", fontsize=14)
plt.xlabel("Mental Health Status", fontsize=12)
plt.ylabel("Sleep Hours", fontsize=12)
plt.tight_layout()
plt.show()
```

Average BMI by Smoking Status

In [39]:
```
bmi_smoking = (df.groupby("Smoking_Status", as_index=False)
["BMI"].mean().sort_values("BMI", ascending=False))

plt.figure(figsize=(10, 5))
sns.barplot(data=bmi_smoking, x="Smoking_Status", y="BMI",
palette="viridis")

for index, row in bmi_smoking.iterrows():
    plt.text(index, row["BMI"] + 0.2, round(row["BMI"],
1),ha='center', fontsize=10)

plt.title("Average BMI by Smoking Status", fontsize=14)
plt.xlabel("Smoking Status")
plt.ylabel("Average BMI")
plt.tight_layout()
plt.show()
```

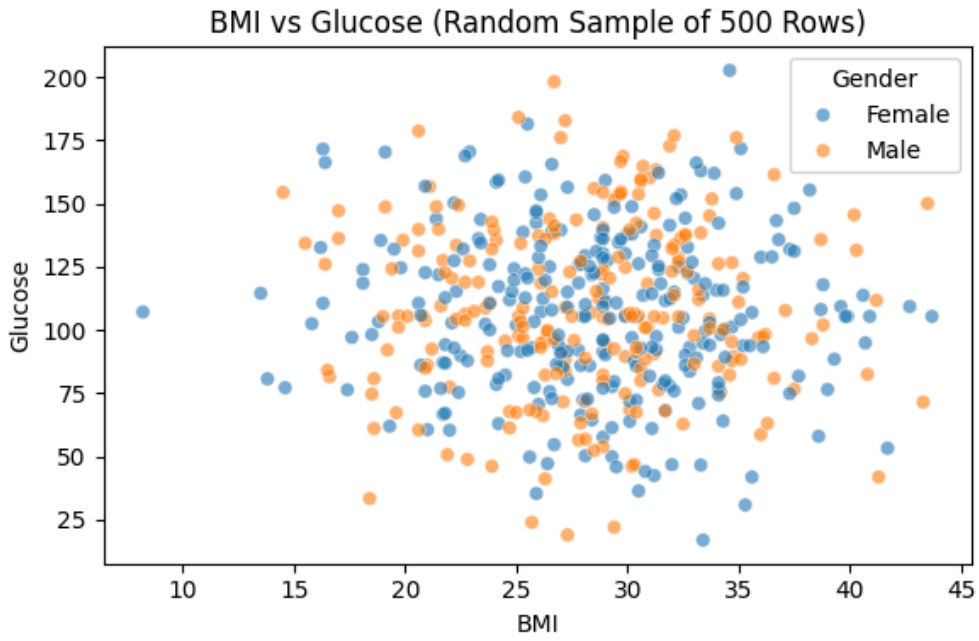


In [40]:
```
sns.boxplot(data=df, x="Diabetes_Type", y="Age", palette="coolwarm")
plt.title("Age by Diabetes Type")
plt.show()
```



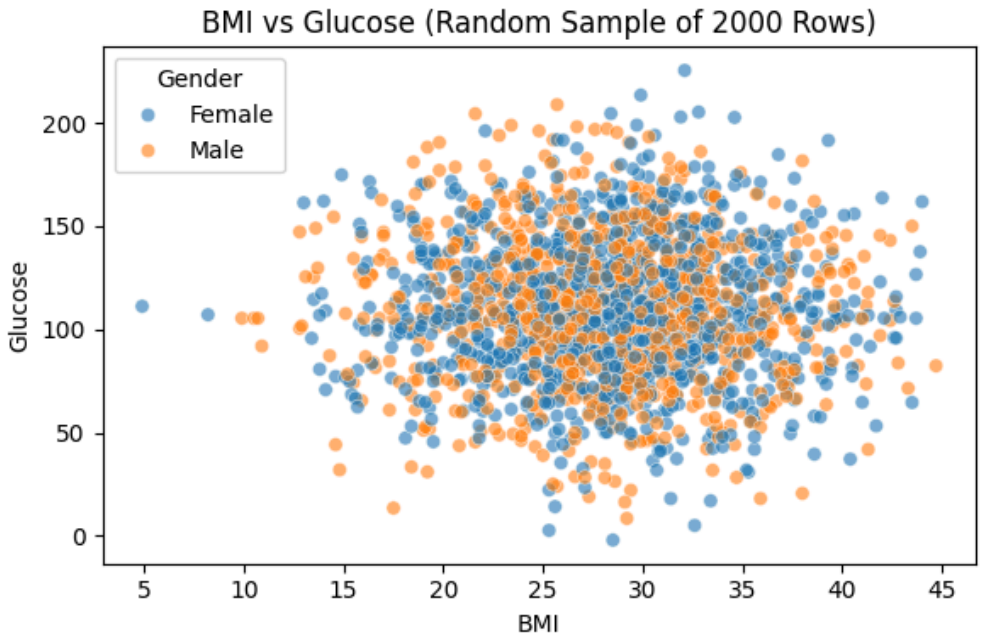
Random Sample of 500 Rows

In [41]:
```
sample_df = df.sample(500, random_state=42)

plt.figure(figsize=(6,4))
sns.scatterplot(data=sample_df,x="BMI",y="Glucose",hue="Gender",alpha=0.6)
plt.title("BMI vs Glucose (Random Sample of 500 Rows)")
plt.tight_layout()
plt.show()
```



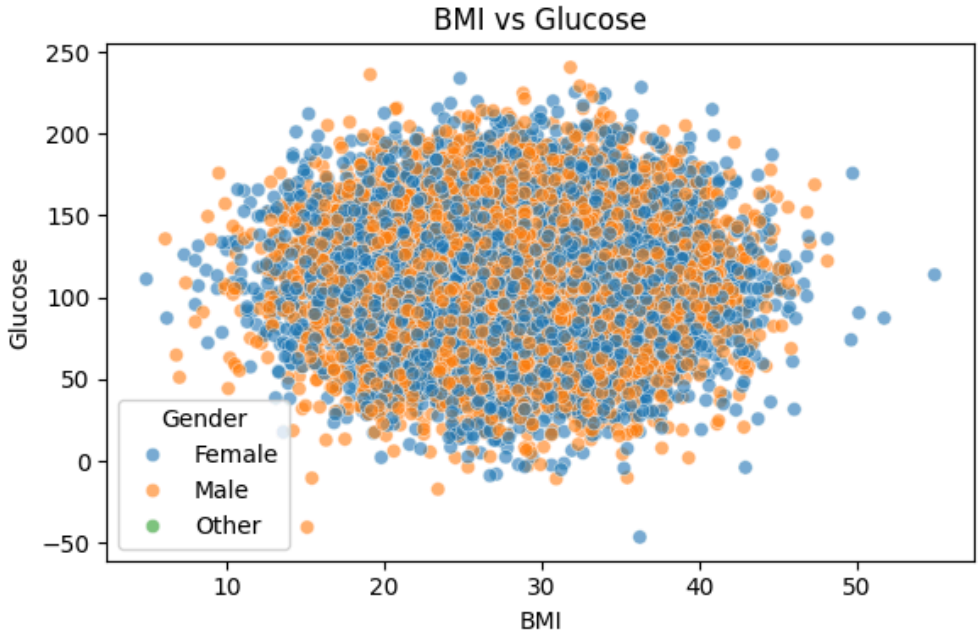
Random Sample of 2000 Rows

In [42]:
```
sample_df = df.sample(2000, random_state=42)

plt.figure(figsize=(6,4))
sns.scatterplot(data=sample_df,x="BMI",y="Glucose",hue="Gender",alpha=0.6)
plt.title("BMI vs Glucose (Random Sample of 2000 Rows)")
plt.tight_layout()
plt.show()
```



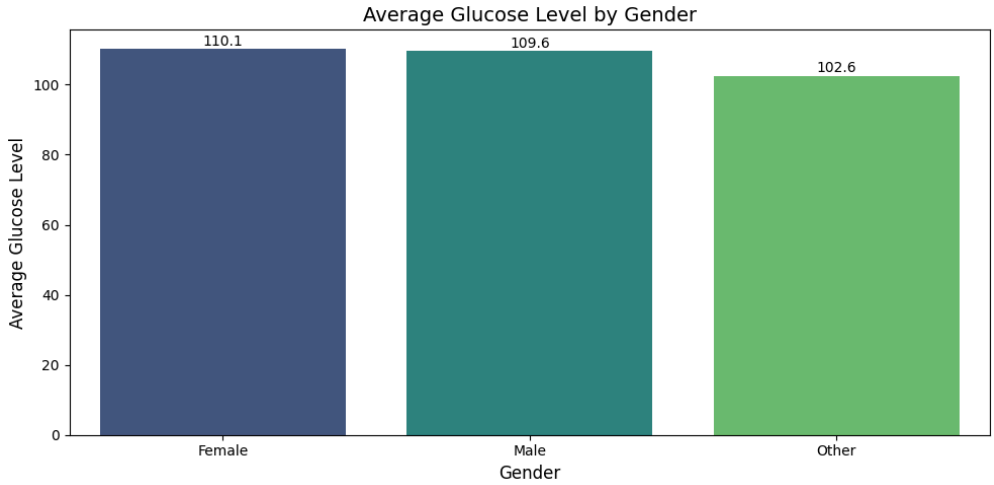
More Detailed

In [43]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(data=df, x="BMI", y="Glucose", hue="Gender",
alpha=0.6)
plt.title("BMI vs Glucose")
plt.tight_layout()
plt.show()
```

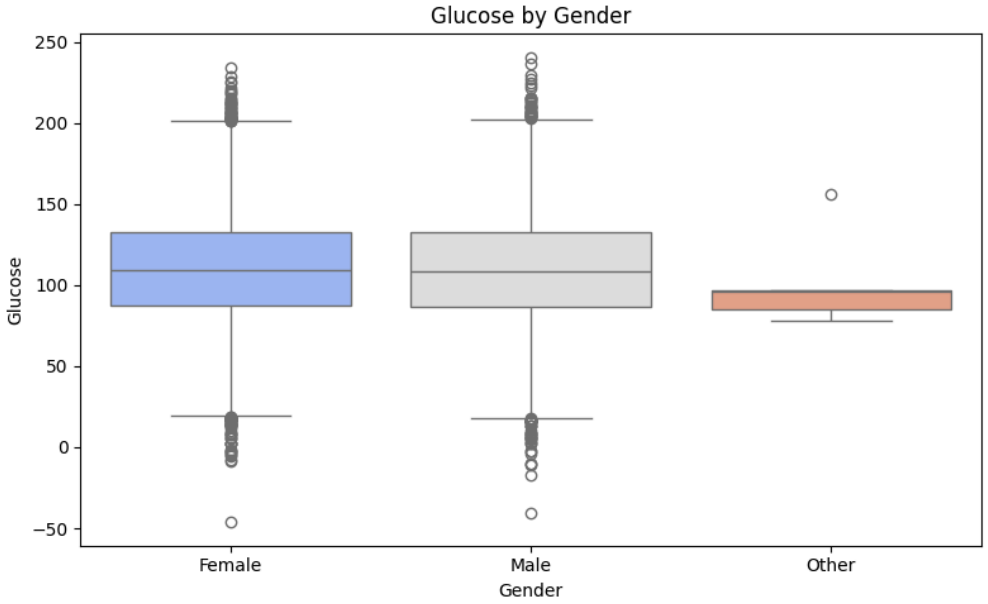


Glucose Levels by Gender

In [44]:
```
# Average Glucose Level by Gender
glucose_by_gender = (df.groupby("Gender", as_index=False)
["Glucose"].mean().sort_values("Glucose", ascending=False))

plt.figure(figsize=(10, 5))
sns.barplot(data=glucose_by_gender, x="Gender", y="Glucose",
palette="viridis")
plt.title("Average Glucose Level by Gender", fontsize=14)
plt.xlabel("Gender", fontsize=12)
plt.ylabel("Average Glucose Level", fontsize=12)

for index, row in glucose_by_gender.iterrows():
    plt.text(index, row["Glucose"] + 1, round(row["Glucose"],
1),ha='center', fontsize=10)

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

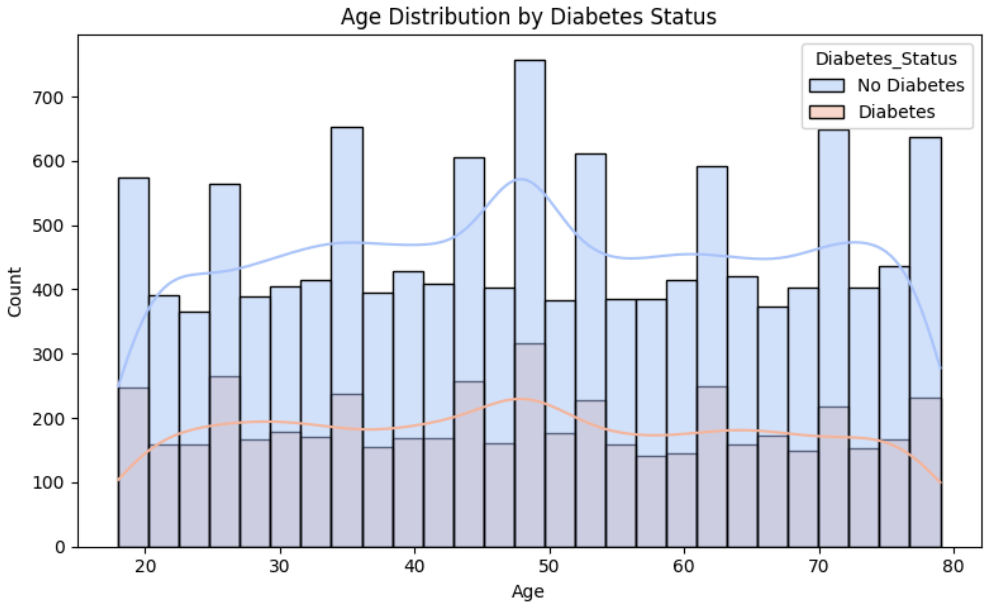


In [45]:
```
plt.figure(figsize=(8,5))
sns.boxplot(data=df, x="Gender", y="Glucose", palette="coolwarm")
plt.title("Glucose by Gender")
plt.tight_layout()
plt.show()
```



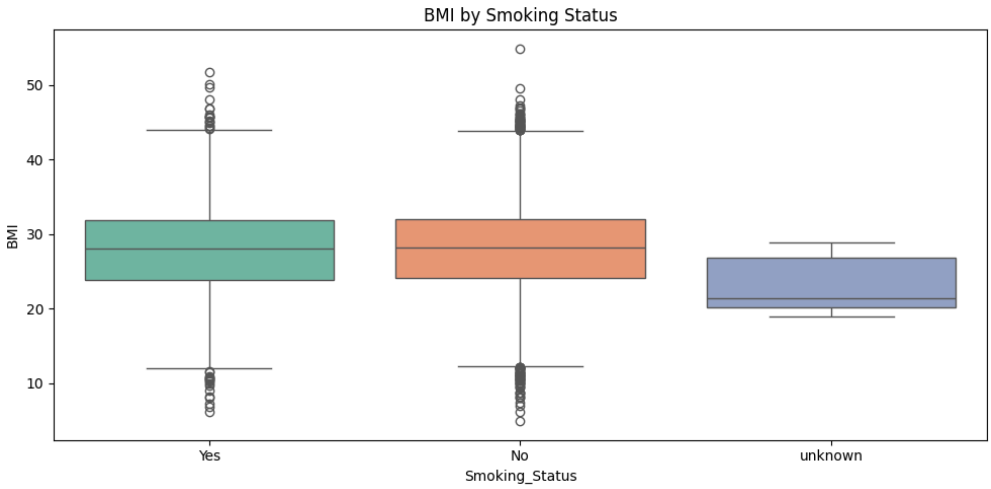
Diabetes Outcome vs Age

In [46]:
```
plt.figure(figsize=(8,5))
sns.histplot(data=df, x="Age", hue="Diabetes_Status", kde=True,
palette="coolwarm")
plt.title("Age Distribution by Diabetes Status")
plt.xlabel("Age")
plt.tight_layout()
plt.show()
```



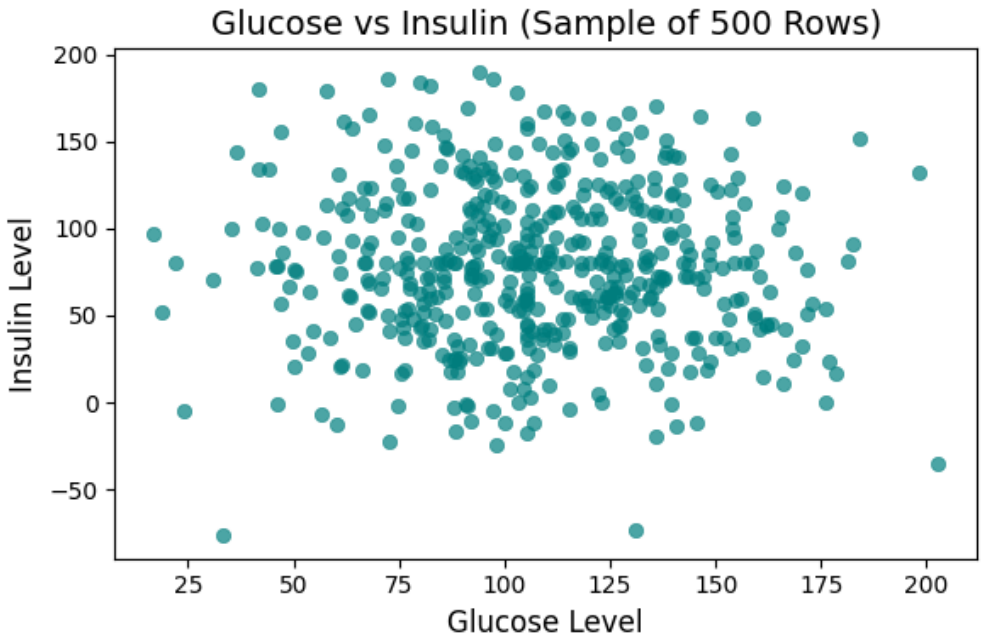
BMI by Smoking Status

In [47]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data=df, x="Smoking_Status", y="BMI", palette="Set2")
plt.title("BMI by Smoking Status")
plt.tight_layout()
plt.show()
```



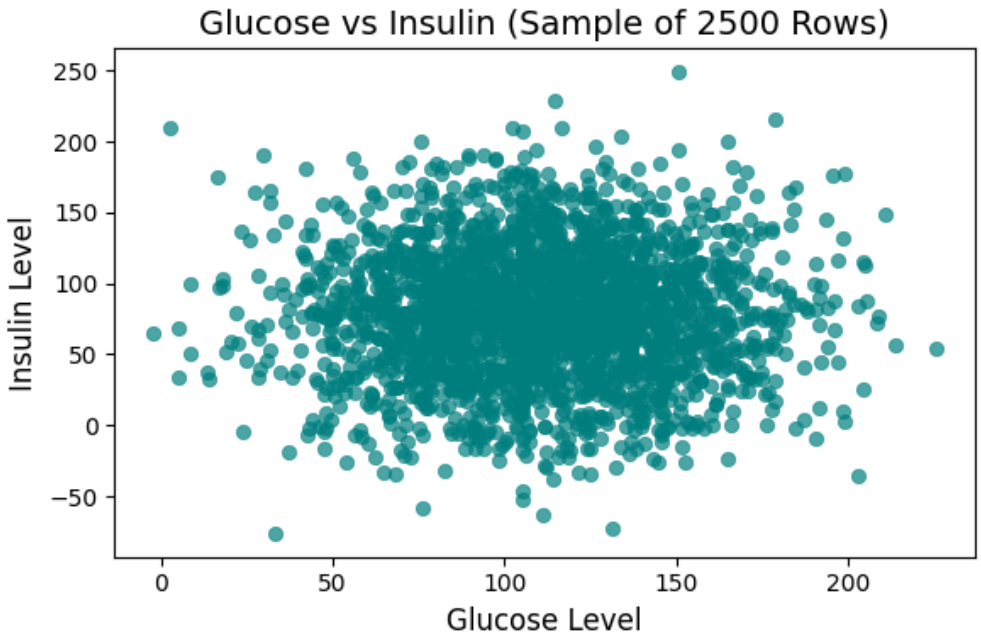
Random Sample of 500 Rows

In [48]:
```
sample_df = df.sample(500, random_state=42)

plt.figure(figsize=(6, 4))
sns.scatterplot(data=sample_df,x="Glucose", y="Insulin",alpha=0.7,
edgecolor=None, color="teal")
plt.title("Glucose vs Insulin (Sample of 500 Rows)", fontsize=14)
plt.xlabel("Glucose Level", fontsize=12)
plt.ylabel("Insulin Level", fontsize=12)
plt.tight_layout()
plt.show()
```



Random Sample of 2500 Rows

In [49]:
```
sample_df = df.sample(2500, random_state=42)

plt.figure(figsize=(6, 4))
sns.scatterplot(data=sample_df,x="Glucose", y="Insulin",alpha=0.7,
edgecolor=None, color="teal")
plt.title("Glucose vs Insulin (Sample of 2500 Rows)", fontsize=14)
plt.xlabel("Glucose Level", fontsize=12)
plt.ylabel("Insulin Level", fontsize=12)
plt.tight_layout()
plt.show()
```

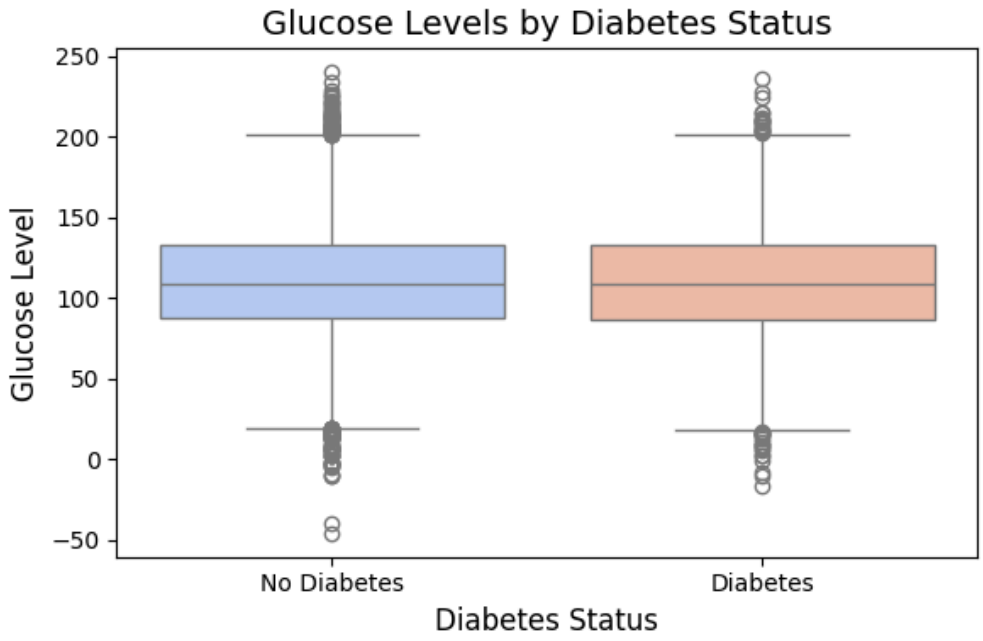


More Detailed

In [50]:
```
plt.figure(figsize=(6, 4))
sns.scatterplot(data=df,x="Glucose", y="Insulin",alpha=0.7,
edgecolor=None, color="teal")

plt.title("Glucose vs Insulin", fontsize=14)
plt.xlabel("Glucose Level", fontsize=12)
plt.ylabel("Insulin Level", fontsize=12)
plt.tight_layout()
plt.show()
```

Compares Glucose vs Diabetes Outcome

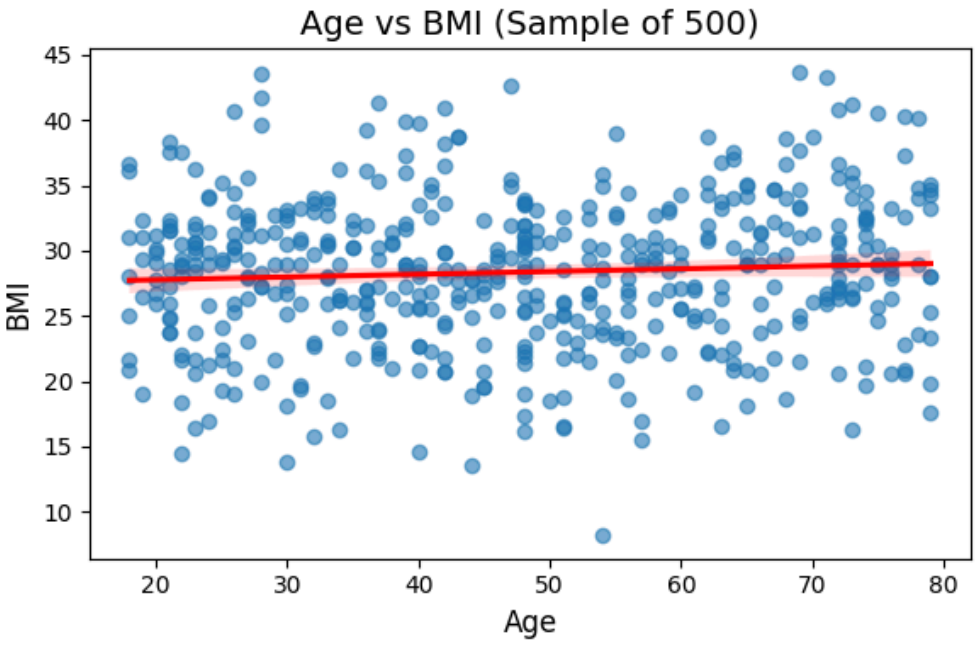
In [51]:
```
plt.figure(figsize=(6, 4))
sns.boxplot(data=df, x="Diabetes_Status", y="Glucose",
palette="coolwarm")
plt.title("Glucose Levels by Diabetes Status", fontsize=14)
plt.xlabel("Diabetes Status", fontsize=12)
plt.ylabel("Glucose Level", fontsize=12)
plt.tight_layout()
plt.show()
```



Relationship between Age and BMI (Regplot)

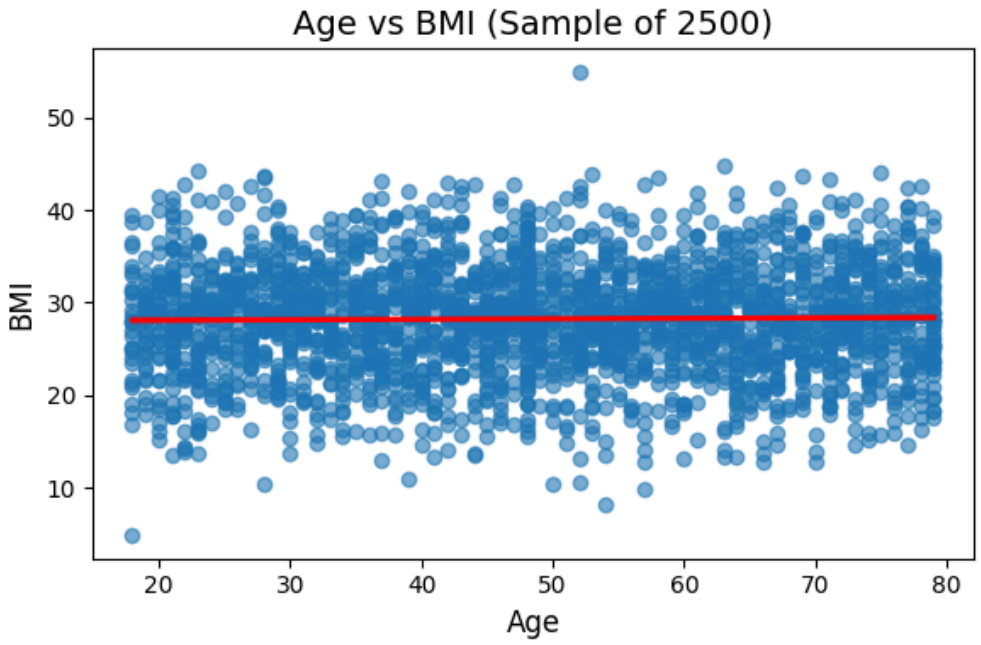
Random Sample of 500

In [52]:
```
sample_500 = df.sample(500, random_state=42)

plt.figure(figsize=(6, 4))
sns.regplot(data=sample_500, x="Age", y="BMI", scatter_kws=
{"alpha":0.6}, line_kws={"color":"red"})
plt.title("Age vs BMI (Sample of 500)", fontsize=14)
plt.xlabel("Age", fontsize=12)
plt.ylabel("BMI", fontsize=12)
plt.tight_layout()
plt.show()
```



Random Sample of 2500

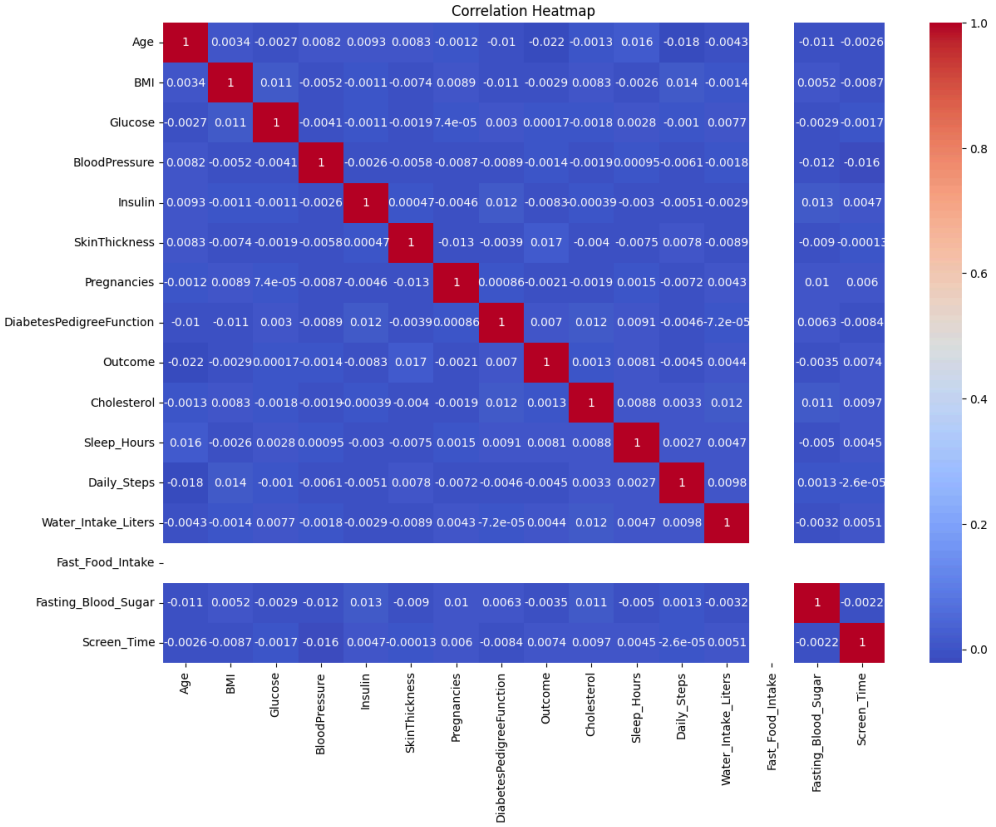
In [53]:
```
sample_2500 = df.sample(2500, random_state=42)

plt.figure(figsize=(6, 4))
sns.regplot(data=sample_2500, x="Age", y="BMI",scatter_kws=
{"alpha":0.6}, line_kws={"color":"red"})
plt.title("Age vs BMI (Sample of 2500)", fontsize=14)
plt.xlabel("Age", fontsize=12)
plt.ylabel("BMI", fontsize=12)
plt.tight_layout()
plt.show()
```



MULTIVARIATE ANALYSIS (Three or more variables)

Correlation Heatmap (All numeric variables)

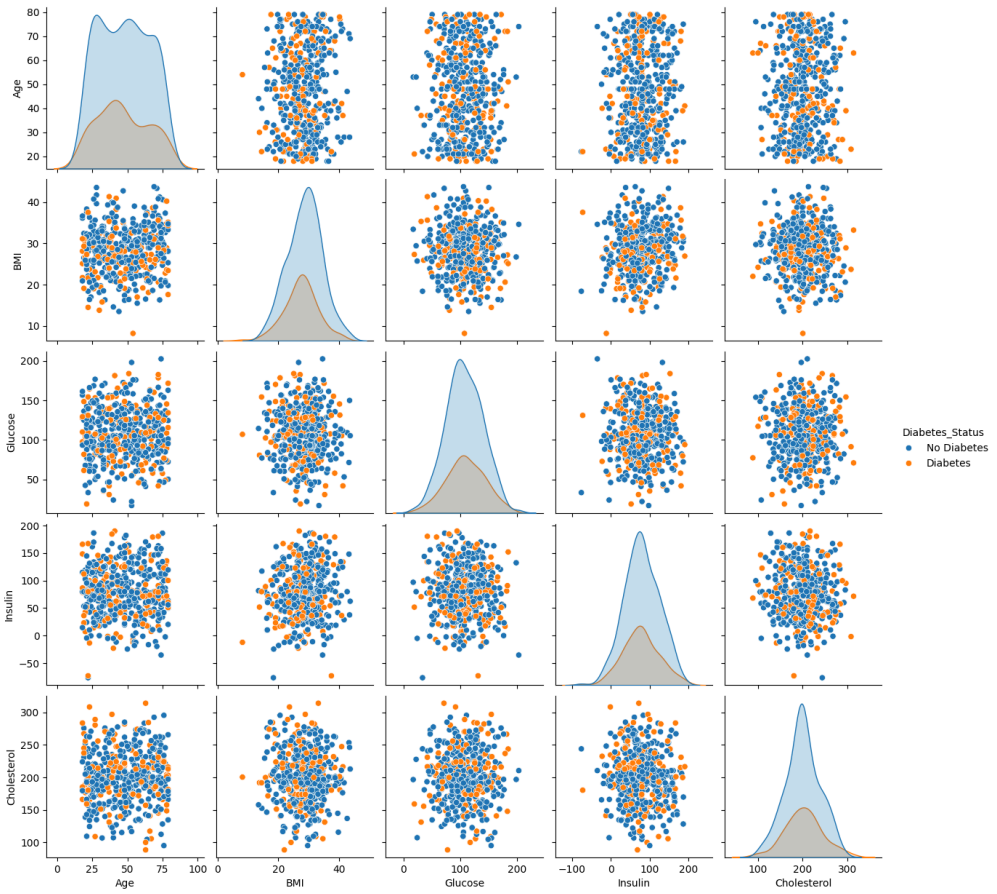
In [54]:
```
plt.figure(figsize=(14,10))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



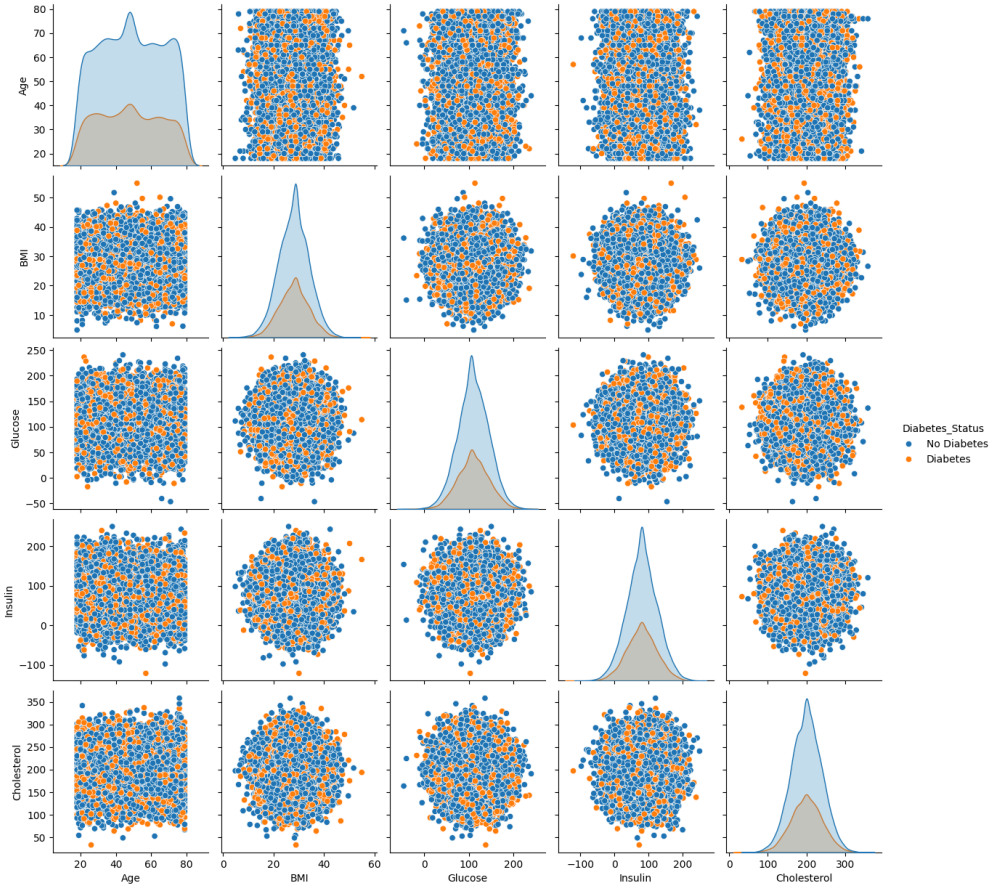
Pairplot of Top Health Indicators

with Random Sample of 500 Rows

In [55]:
```
sample_500 = df.sample(500, random_state=42)
sns.pairplot(sample_500[["Age", "BMI", "Glucose", "Insulin",
"Cholesterol",
"Diabetes_Status"]],hue="Diabetes_Status",diag_kind="kde")
plt.show()
```



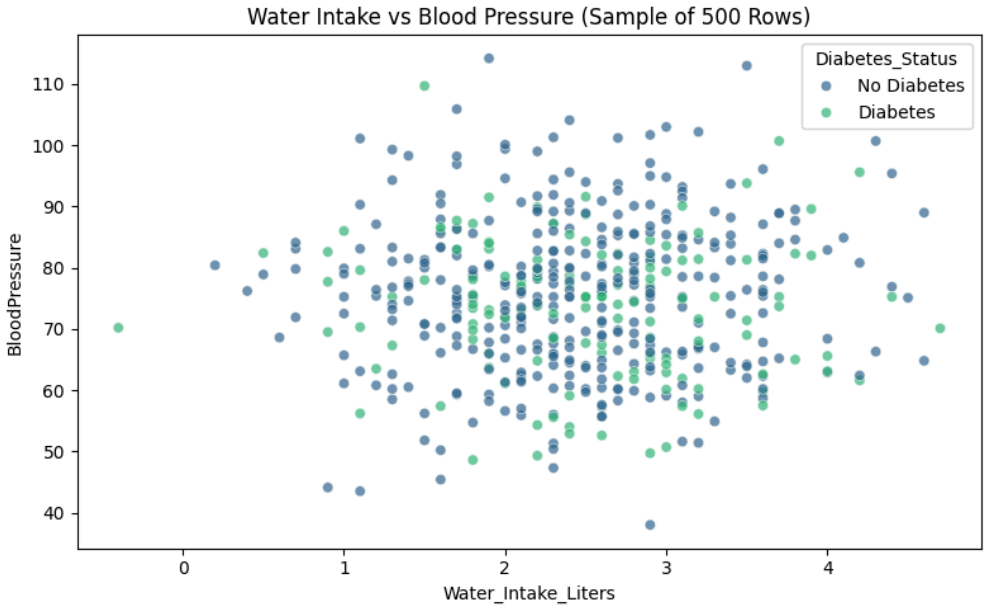
In [56]:
```
sns.pairplot(df[["Age", "BMI", "Glucose", "Insulin", "Cholesterol",
"Diabetes_Status"]],
             hue="Diabetes_Status", diag_kind="kde")
plt.show()
```



Water Intake vs BP colored by Diabetes Outcome

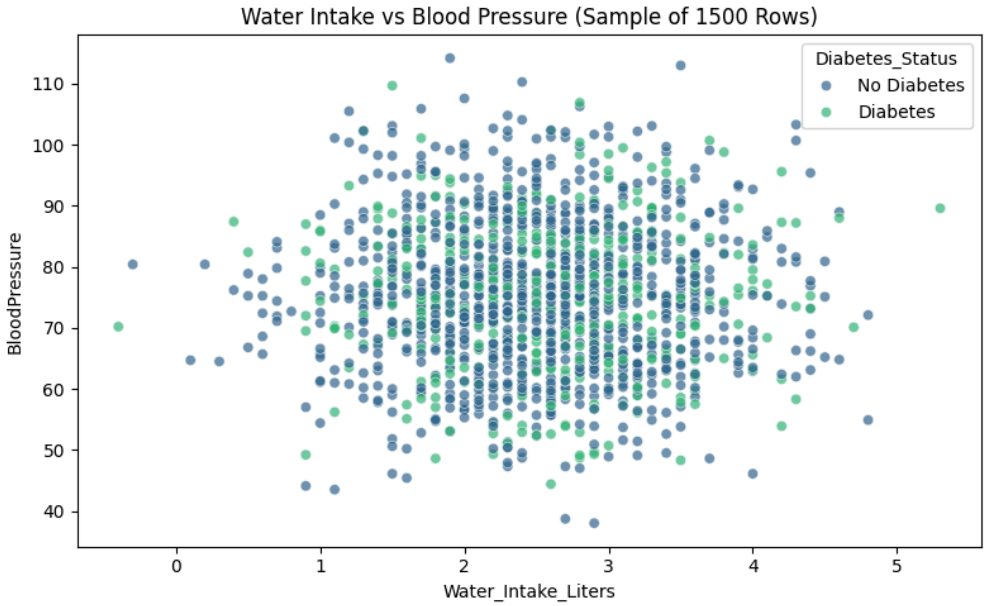
Random Sample of 500 Rows

In [57]:
```
sample_df = df.sample(500, random_state=42)

plt.figure(figsize=(8,5))
sns.scatterplot(data=sample_df,x="Water_Intake_Liters",y="BloodPress
ure",hue="Diabetes_Status",alpha=0.7,palette="viridis")
plt.title("Water Intake vs Blood Pressure (Sample of 500 Rows)")
plt.tight_layout()
plt.show()
```



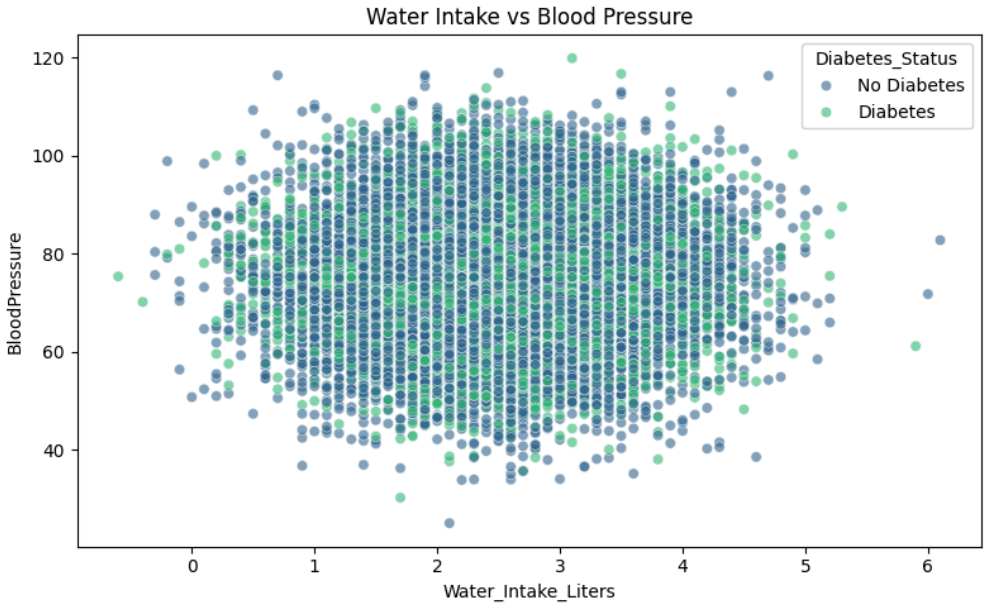
Random Sample of 1500 Rows

In [58]:
```
sample_df = df.sample(1500, random_state=42)

plt.figure(figsize=(8,5))
sns.scatterplot(data=sample_df,x="Water_Intake_Liters",y="BloodPress
ure",hue="Diabetes_Status",alpha=0.7,palette="viridis")
plt.title("Water Intake vs Blood Pressure (Sample of 1500 Rows)")
plt.tight_layout()
plt.show()
```



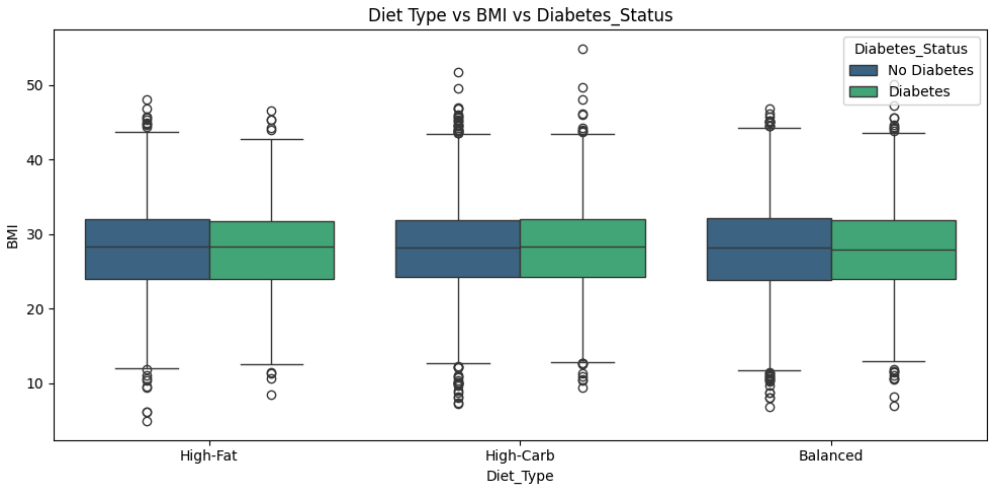
More Detailed

In [59]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(data=df, x="Water_Intake_Liters",
y="BloodPressure",hue="Diabetes_Status", alpha=0.6,
palette="viridis")
plt.title("Water Intake vs Blood Pressure")
plt.tight_layout()
plt.show()
```



Diet Type + BMI + Diabetes Outcome (Boxplot)

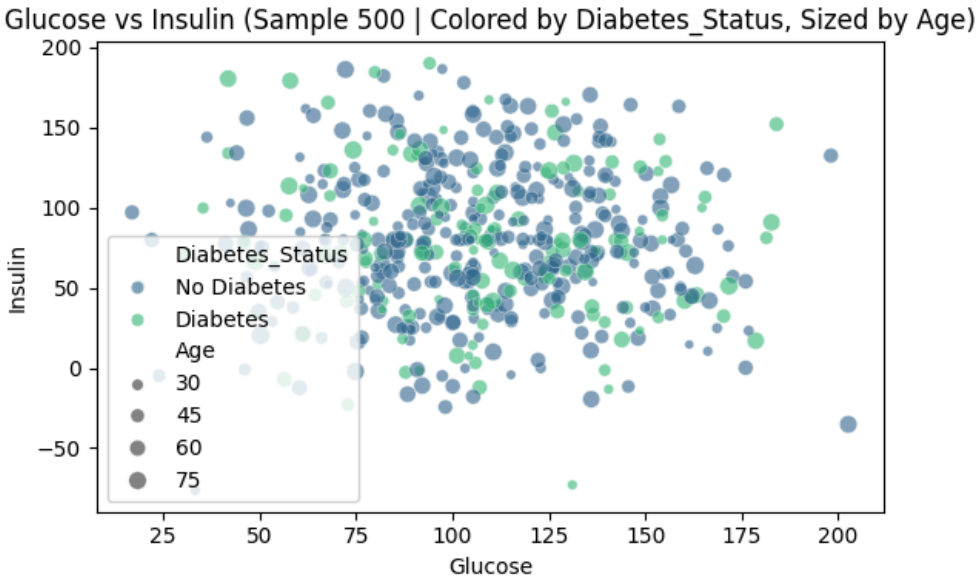
In [60]:
```
plt.figure(figsize=(10,5))
sns.boxplot(data=df, x="Diet_Type", y="BMI", hue="Diabetes_Status",
palette="viridis")
plt.title("Diet Type vs BMI vs Diabetes_Status")
plt.tight_layout()
plt.show()
```



Glucose vs Insulin

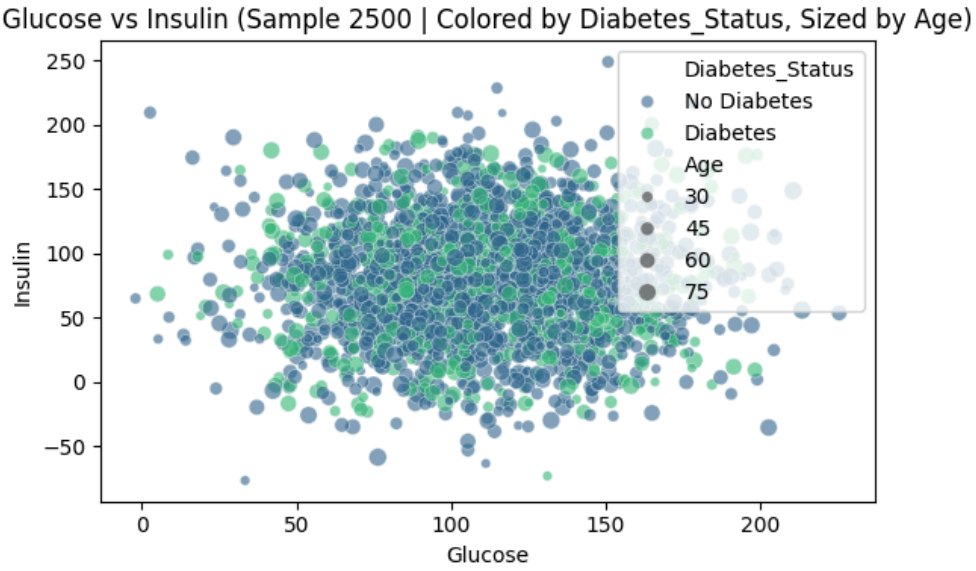
Random Sample of 500 Rows

In [61]:
```
sample_500 = df.sample(500, random_state=42)

plt.figure(figsize=(6,4))
sns.scatterplot(data=sample_500,x="Glucose", y="Insulin",
hue="Diabetes_Status", size="Age", alpha=0.6, palette="viridis")
plt.title("Glucose vs Insulin (Sample 500 | Colored by
Diabetes_Status, Sized by Age)")
plt.tight_layout()
plt.show()
```



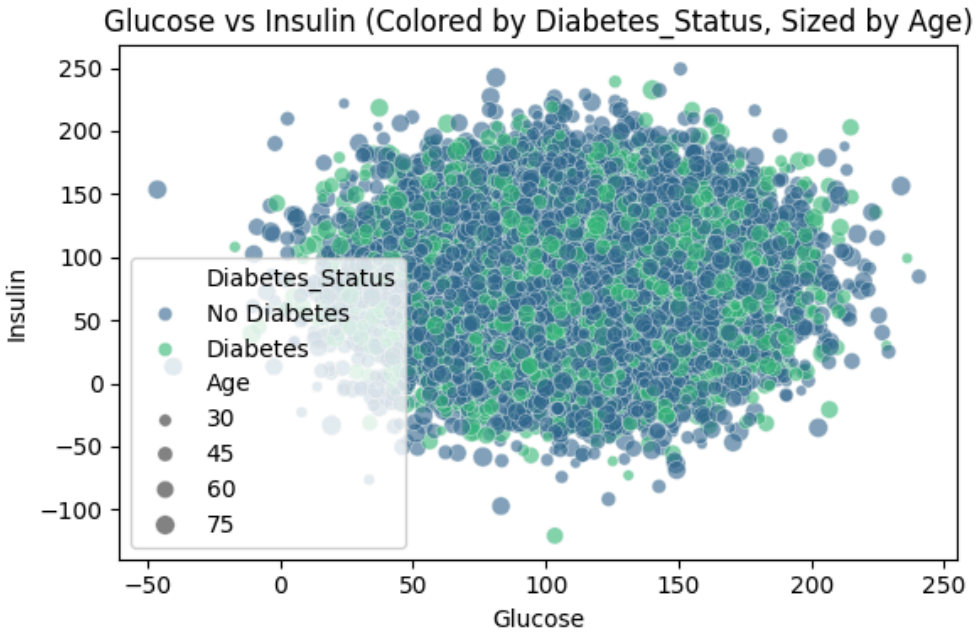
Random Sample of 2500 Rows

In [62]:
```
sample_500 = df.sample(2500, random_state=42)

plt.figure(figsize=(6,4))
sns.scatterplot(data=sample_500,x="Glucose", y="Insulin",
hue="Diabetes_Status", size="Age", alpha=0.6, palette="viridis")
plt.title("Glucose vs Insulin (Sample 2500 | Colored by
Diabetes_Status, Sized by Age)")
plt.tight_layout()
plt.show()
```



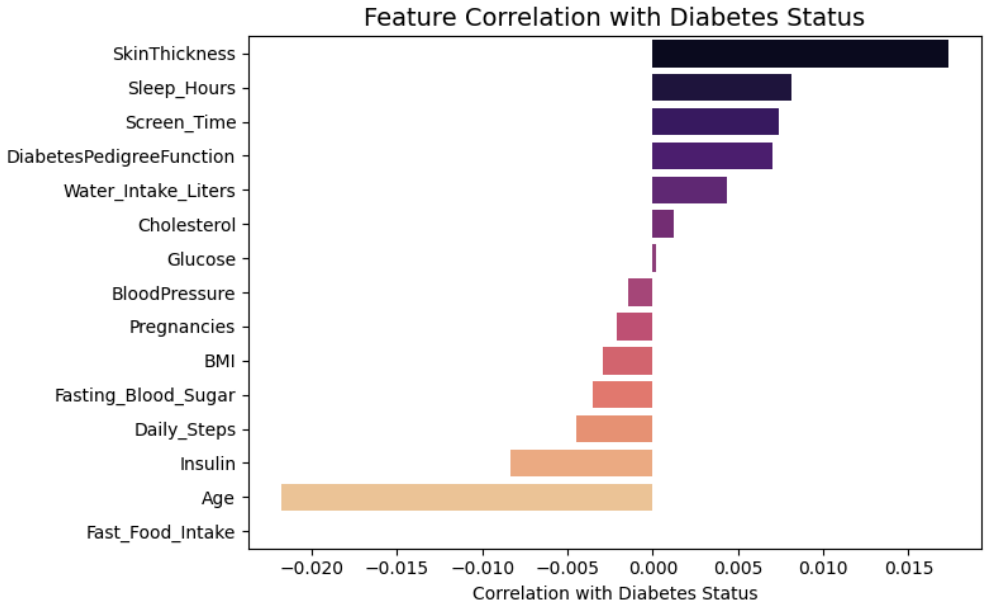
More Detailed

In [63]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(data=df, x="Glucose", y="Insulin",
hue="Diabetes_Status", size="Age", alpha=0.6, palette="viridis")
plt.title("Glucose vs Insulin (Colored by Diabetes_Status, Sized by
Age)")
plt.tight_layout()
plt.show()
```



Feature Correlation with Diabetes Outcome

In [64]:
```
corr_with_outcome = (df.corr(numeric_only=True)
["Outcome"].drop("Outcome").sort_values(ascending=False))
corr_with_outcome.index.name = ""

plt.figure(figsize=(8, 5))
sns.barplot(x=corr_with_outcome.values,y=corr_with_outcome.index,pal
ette="magma")
plt.title("Feature Correlation with Diabetes Status", fontsize=14)
plt.xlabel("Correlation with Diabetes Status")
plt.tight_layout()
plt.show()
```



In [92]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (confusion_matrix,
classification_report,
                             accuracy_score, precision_score,
recall_score,
                             f1_score, roc_auc_score, roc_curve)

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Select Target Variable

In [65]:
```
df["Diabetes_Status"] = df["Diabetes_Status"].map({"No Diabetes": 0,
"Diabetes": 1})
y = df["Diabetes_Status"]
```

Feature Selection

In [66]:
```
X = df.drop(columns=["Diabetes_Status", "Outcome", "Country"])
```

Encode Categorical Variables

In [67]:
```
from sklearn.preprocessing import OneHotEncoder

X = pd.get_dummies(X, drop_first=True)
```

Train-Test Split

In [97]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Feature Scaling

In [98]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

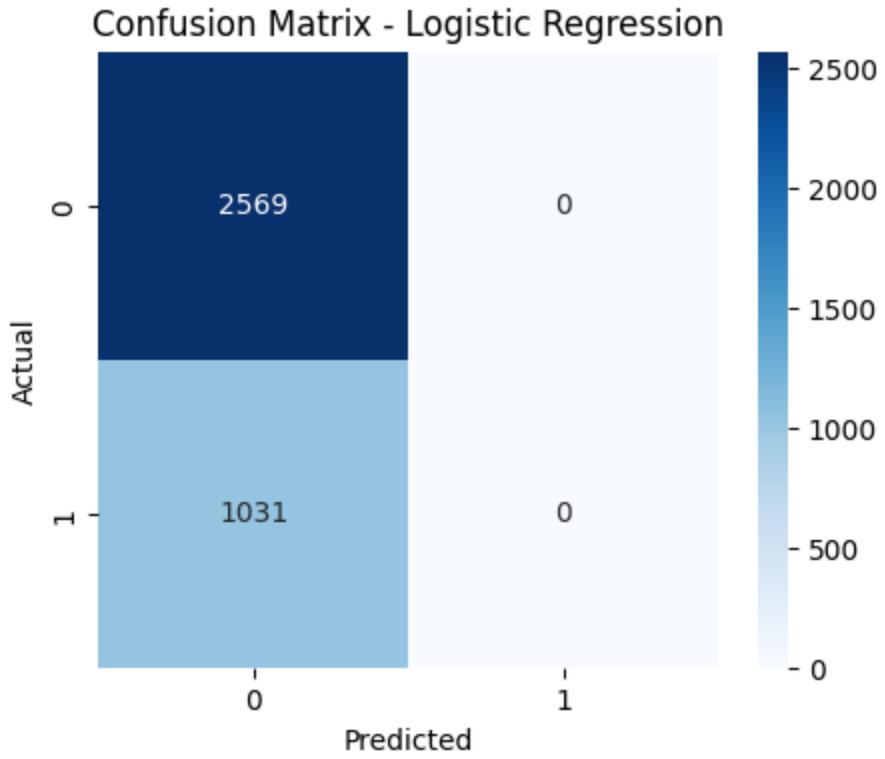
In [101]:
```
from sklearn.impute import SimpleImputer
```

In [102]:
```
imputer = SimpleImputer(strategy="median")

X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)
```

In [103]:
```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)
```

In [107]:
```
log_model = LogisticRegression(max_iter=1000)
log_model.fit(X_train_scaled, y_train)

y_pred_log = log_model.predict(X_test_scaled)
y_prob_log = log_model.predict_proba(X_test_scaled)[:, 1]
```

In [108]:
```
cm_log = confusion_matrix(y_test, y_pred_log)

plt.figure(figsize=(5,4))
sns.heatmap(cm_log, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [109]:
```
print("Logistic Regression Classification Report:\n")
print(classification_report(y_test, y_pred_log))

print("Accuracy:", accuracy_score(y_test, y_pred_log))
print("Precision:", precision_score(y_test, y_pred_log))
print("Recall:", recall_score(y_test, y_pred_log))
print("F1 Score:", f1_score(y_test, y_pred_log))
print("ROC AUC:", roc_auc_score(y_test, y_prob_log))
```

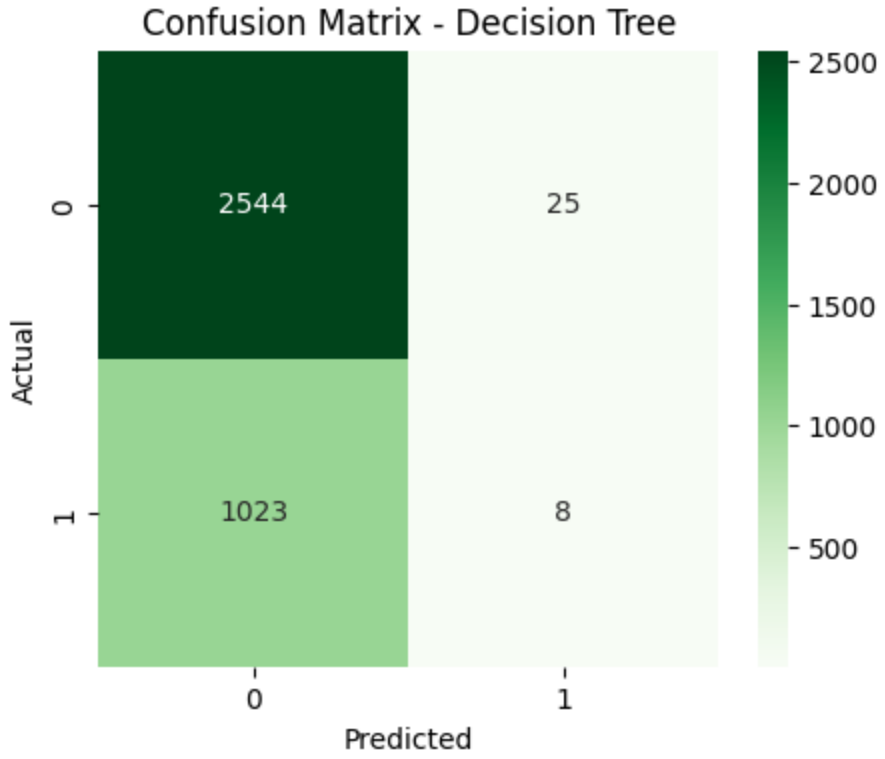
```
Logistic Regression Classification Report:

              precision    recall  f1-score   support

           0       0.71      1.00      0.83      2569
           1       0.00      0.00      0.00      1031

    accuracy                           0.71      3600
   macro avg       0.36      0.50      0.42      3600
weighted avg       0.51      0.71      0.59      3600

Accuracy: 0.7136111111111111
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
ROC AUC: 0.4927285296335212
```

DecisionTree Classifier

In [74]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=6, random_state=42)
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.71      0.99      0.83      2569
           1       0.24      0.01      0.02      1031

    accuracy                           0.71      3600
   macro avg       0.48      0.50      0.42      3600
weighted avg       0.58      0.71      0.60      3600
```

In [110]:
```
cm_dt = confusion_matrix(y_test, y_pred_dt)

plt.figure(figsize=(5,4))
sns.heatmap(cm_dt, annot=True, fmt="d", cmap="Greens")
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [113]:
```
print("Decision Tree Classification Report:\n")
print(classification_report(y_test, y_pred_dt))

print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt))
print("Recall:", recall_score(y_test, y_pred_dt))
print("F1 Score:", f1_score(y_test, y_pred_dt))
print("ROC AUC:", roc_auc_score(y_test, y_pred_dt))
```

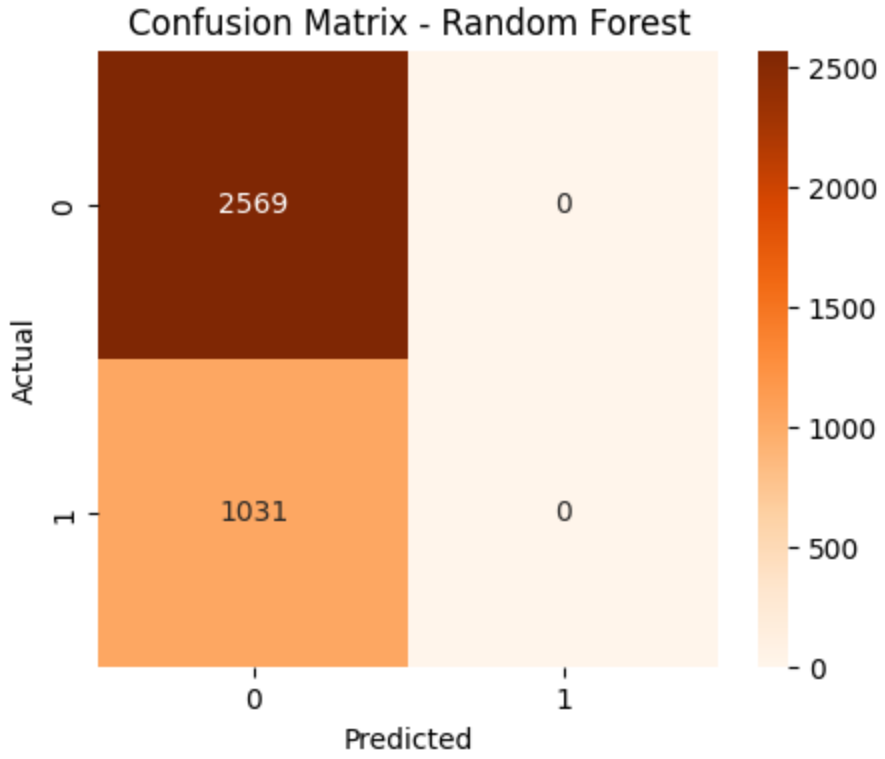
```
Decision Tree Classification Report:

              precision    recall  f1-score   support

           0       0.71      0.99      0.83      2569
           1       0.24      0.01      0.02      1031

    accuracy                           0.71      3600
   macro avg       0.48      0.50      0.42      3600
weighted avg       0.58      0.71      0.60      3600

Accuracy: 0.7088888888888889
Precision: 0.24242424242424243
Recall: 0.007759456838021339
F1 Score: 0.015037593984962405
ROC AUC: 0.49901402191842675
```

RandomForestClassifier

In [75]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=10,
    random_state=42
)

rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.71      1.00      0.83      2569
           1       0.00      0.00      0.00      1031

    accuracy                           0.71      3600
   macro avg       0.36      0.50      0.42      3600
weighted avg       0.51      0.71      0.59      3600
```

In [114]:
```
cm_rf = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(5,4))
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Oranges")
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [116]:
```
print("Random Forest Classification Report:\n")
print(classification_report(y_test, y_pred_rf))

print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1 Score:", f1_score(y_test, y_pred_rf))
print("ROC AUC:", roc_auc_score(y_test, y_pred_rf))
```

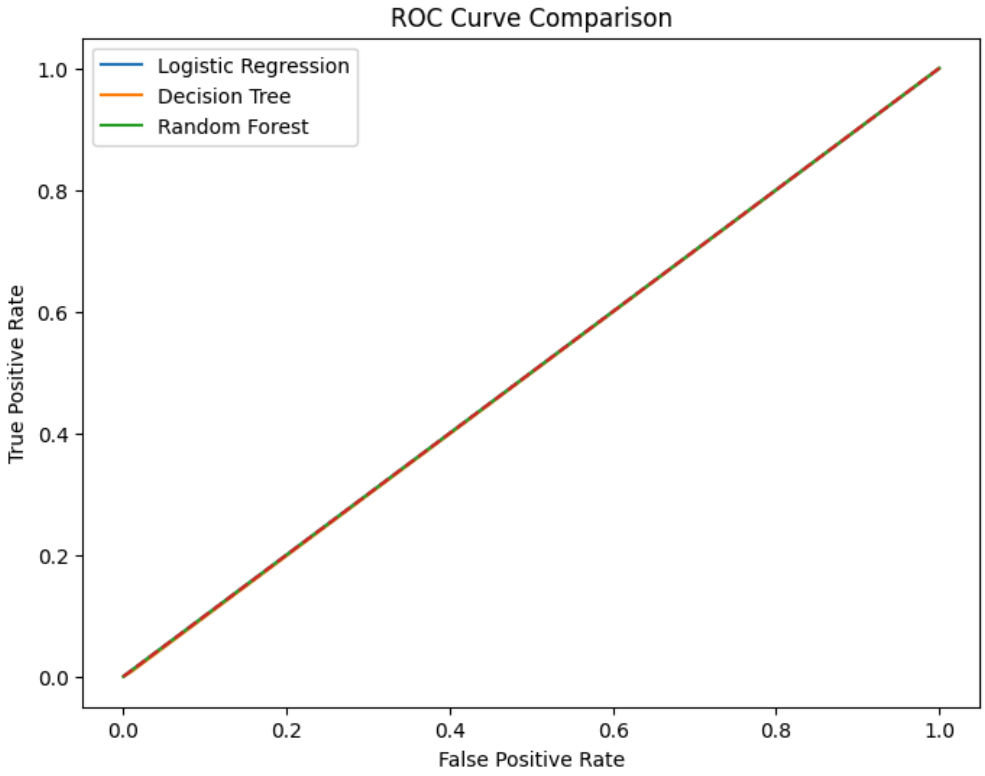
```
Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.71      1.00      0.83      2569
           1       0.00      0.00      0.00      1031

    accuracy                           0.71      3600
   macro avg       0.36      0.50      0.42      3600
weighted avg       0.51      0.71      0.59      3600

Accuracy: 0.7136111111111111
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
ROC AUC: 0.5
```

Model Evaluation Metrics

In [76]:
```
from sklearn.metrics import roc_auc_score

print("ROC AUC:", roc_auc_score(y_test, rf.predict_proba(X_test)
[:,1]))
```

```
ROC AUC: 0.4977764051650678
```

In [118]:
```
fpr_log, tpr_log, _ = roc_curve(y_test, y_pred_log)
fpr_dt, tpr_dt, _ = roc_curve(y_test, y_pred_dt)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)

plt.figure(figsize=(8,6))
plt.plot(fpr_log, tpr_log, label="Logistic Regression")
plt.plot(fpr_dt, tpr_dt, label="Decision Tree")
plt.plot(fpr_rf, tpr_rf, label="Random Forest")
plt.plot([0,1], [0,1], linestyle="--")

plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



In [120]:
```
results = pd.DataFrame({
    "Model": ["Logistic Regression", "Decision Tree", "Random
Forest"],
    "Accuracy": [
        accuracy_score(y_test, y_pred_log),
        accuracy_score(y_test, y_pred_dt),
        accuracy_score(y_test, y_pred_rf)
    ],
    "Precision": [
        precision_score(y_test, y_pred_log),
        precision_score(y_test, y_pred_dt),
        precision_score(y_test, y_pred_rf)
    ],
    "Recall": [
        recall_score(y_test, y_pred_log),
        recall_score(y_test, y_pred_dt),
        recall_score(y_test, y_pred_rf)
    ],
    "F1 Score": [
        f1_score(y_test, y_pred_log),
        f1_score(y_test, y_pred_dt),
        f1_score(y_test, y_pred_rf)
    ],
    "ROC AUC": [
        roc_auc_score(y_test, y_prob_log),
        roc_auc_score(y_test, y_pred_dt),
        roc_auc_score(y_test, y_pred_rf)
    ]
})

print(results.sort_values(by="ROC AUC", ascending=False))
```

```
                   Model  Accuracy  Precision    Recall  F1 Score
  ROC AUC
  2        Random Forest  0.713611   0.000000  0.000000  0.000000
  0.500000
  1        Decision Tree  0.708889   0.242424  0.007759  0.015038
  0.499014
  0  Logistic Regression  0.713611   0.000000  0.000000  0.000000
  0.492729
```

In [122]:
```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

Out[122]:
```
RandomForestClassifier(random_state=42)
```

In [123]:
```
importances = rf_model.feature_importances_
feature_names = X.columns

feat_imp = pd.DataFrame({
    "Feature": feature_names,
    "Importance": importances
}).sort_values(by="Importance", ascending=False).head(15)

plt.figure(figsize=(8,6))
sns.barplot(data=feat_imp, x="Importance", y="Feature")
plt.title("Top 15 Feature Importances - Random Forest")
plt.show()
```