Datathon FY'25

Team 3

IMPACT ANALYSIS OF DIABETES ON HUMANS

Python Imports

In [27]:
```
# Dataset Preprocessing
import pandas as pd
import numpy as np

# For visualizations
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import seaborn as sns

# Encoding
# from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder

# Data Scaling
from sklearn.preprocessing import StandardScaler

# KNN Imputation
from sklearn.impute import KNNImputer

import warnings
warnings.filterwarnings('ignore')
```

Exploring the Dataset

In [28]:
```
# Importing Dataset
raw_dataset = pd.read_csv('../data/raw_data/diabetes_data.csv')
diabetes_ds = raw_dataset
```

In [29]:
```
diabetes_ds.head(5)
```

Out[29]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	star_sign	BMI	weight	family_diabetes_history	social_media_usag
0	female	NaN	NaN	0.37	paleo	NaN	NaN	197.7	NaN	Occasionall
1	male	59.0	0.0	0.73	NaN	Leo	17.1	156.5	0.0	Occasionall
2	female	31.0	NaN	NaN	NaN	NaN	22.5	137.8	NaN	Occasionall
3	female	81.0	1.0	0.37	pescatarian	NaN	NaN	108.2	0.0	NaN
4	NaN	64.0	0.0	0.55	carnivore	NaN	NaN	179.7	NaN	Occasionall

In [30]:
```
diabetes_ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 16 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   gender                       79954 non-null  object 
 1   age                          80145 non-null  float64
 2   hypertension                 80169 non-null  float64
 3   diabetes_pedigree_function   80120 non-null  float64
 4   diet_type                    79939 non-null  object 
 5   star_sign                    79806 non-null  object 
 6   BMI                          79934 non-null  float64
 7   weight                       80126 non-null  float64
 8   family_diabetes_history      79863 non-null  float64
 9   social_media_usage           79968 non-null  object 
 10  physical_activity_level      80032 non-null  object 
 11  sleep_duration               80063 non-null  float64
 12  stress_level                 80024 non-null  object 
 13  pregnancies                  80033 non-null  float64
 14  alcohol_consumption          79896 non-null  object 
 15  diabetes                     80242 non-null  float64
dtypes: float64(9), object(7)
memory usage: 12.2+ MB
```

The dataset contains **100,000 rows and 16 columns**, including a target variable.

Approximately **20% of the records contain missing values (NAs)**, which must be addressed using appropriate imputation techniques. Additionally, some columns may require encoding to handle categorical data effectively.

The dataset consists of a mix of **float64 and object data types**. The object-type columns should be appropriately transformed into categorical variables using encoding methods to ensure compatibility with the desired analysis or modeling pipeline.

Let us now look at the descriptive statistics of the dataset:

In [31]:
```
diabetes_ds.describe()
```

Out[31]:

	age	hypertension	diabetes_pedigree_function	BMI	weight	family_diabetes_history	sleep_duration	pregr
count	80145.000000	80169.000000	80120.000000	79934.000000	80126.000000	79863.000000	80063.000000	80033.
mean	45.107306	0.202248	0.500877	26.978545	150.526618	0.302167	5.295149	0.
std	18.550434	0.401678	0.173783	6.005039	57.731539	0.459200	2.842133	1.
min	18.000000	0.000000	0.200000	1.800000	50.000000	0.000000	0.000000	0.
25%	27.000000	0.000000	0.350000	22.900000	100.300000	0.000000	3.300000	0.
50%	45.000000	0.000000	0.500000	27.000000	150.900000	0.000000	5.300000	0.
75%	60.000000	0.000000	0.650000	31.000000	200.400000	1.000000	7.000000	1.
max	91.000000	1.000000	0.800000	53.100000	250.000000	1.000000	12.000000	5.

We can see that the scale of each feature column is different and varied.

FEATURE DESCRIPTION:

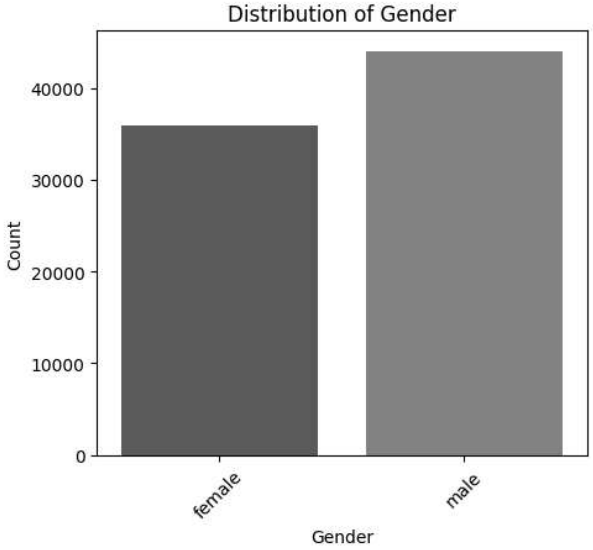
The description of each feature in the dataset is given below:

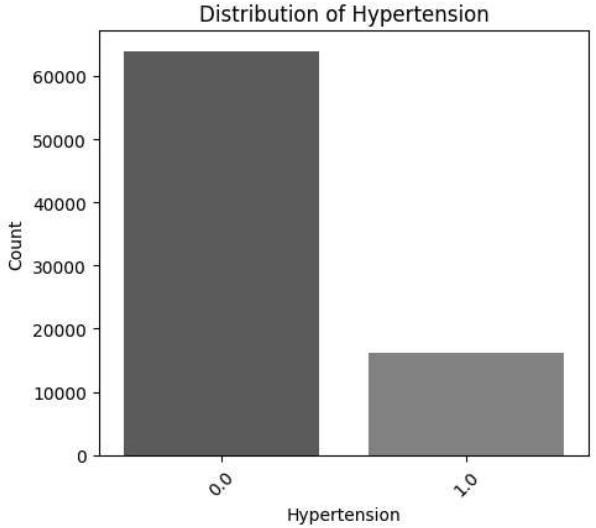
Types of Features:

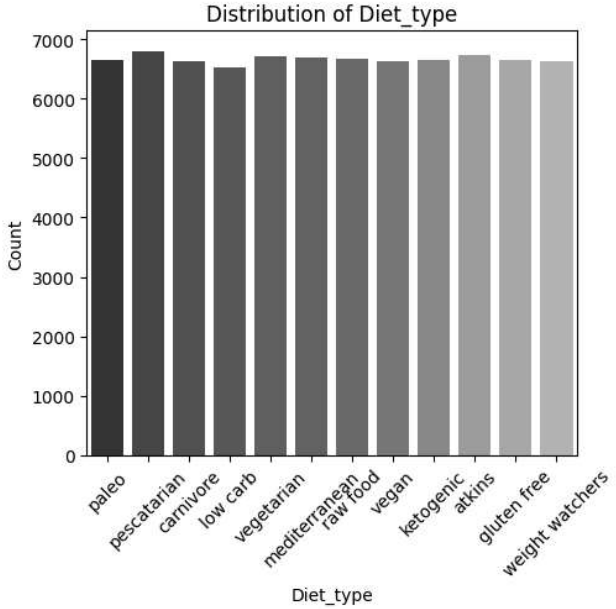
- **Categorical features** (Has two or more categories, and each value in that feature can be categorized by them)
  - **Ordinal features** (Variable having relative ordering or sorting between the values)
    - Examples: `hypertension`, `family_diabetes_history`, `pregnancies`, `social_media_usage`, `physical_activity_level`, `stress_level`, `alcohol_consumption`, `diabetes`
  - **Nominal features** (Variable with no inherent ordering or ranking among the values)
    - Examples: `gender`, `diet_type`, `star_sign`
- **Continuous features** (Variable taking values between any two points or between the minimum or maximum values in the feature column)
  - Examples: `age`, `weight`, `diabetes_pedigree_function`, `sleep_duration`, `BMI`

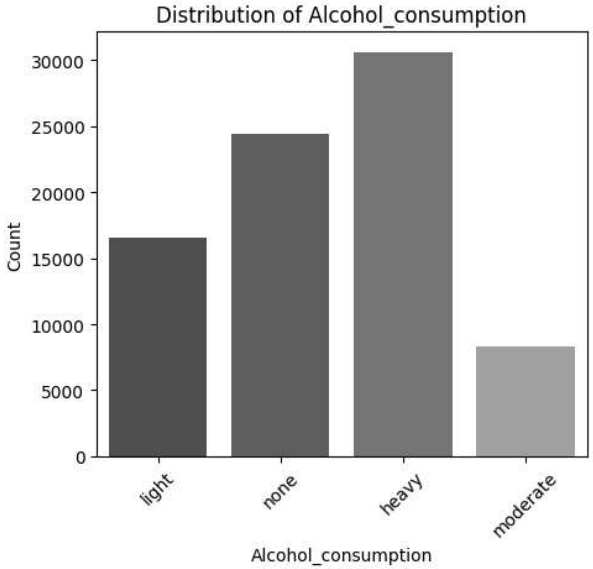
Let's now look at the pictorial representation of the distribution of categorical feature columns:

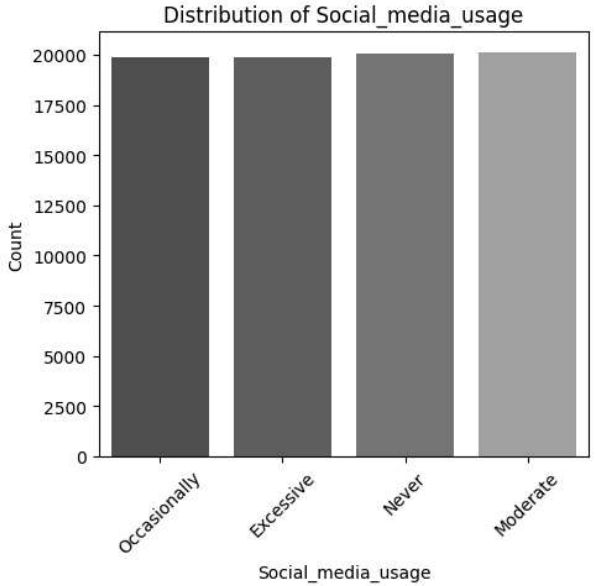
In [32]:
```
# List of 9 categorical features to plot
# We are ignoring the target column and star_sign in this representation
categorical_columns = [
    'gender', 'hypertension', 'diet_type', 'alcohol_consumption',
    'social_media_usage', 'physical_activity_level', 'stress_level',
    'pregnancies', 'family_diabetes_history'
]

# Sets up a 3x3 grid for the plots
fig, axes = plt.subplots(3, 3, figsize=(15, 15))
axes = axes.flatten()

# Generates a bar chart for each categorical variable
for i, column in enumerate(categorical_columns):
    sns.countplot(data=diabetes_ds, x=column, ax=axes[i], palette="viridis")
    axes[i].set_title(f"Distribution of {column.capitalize()}", fontsize=12)
    axes[i].set_xlabel(column.capitalize(), fontsize=10)
    axes[i].set_ylabel('Count', fontsize=10)
    axes[i].tick_params(axis='x', rotation=45)

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

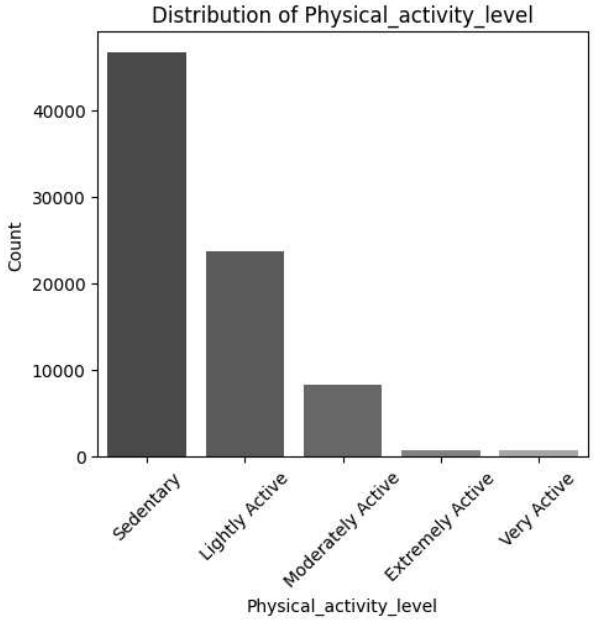

Distribution of Gender
Count
Gender
female
male
0
10000
20000
30000
40000

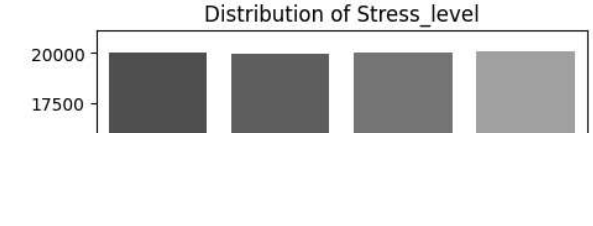

Distribution of Hypertension
Count
Hypertension
0.0
1.0
0
10000
20000
30000
40000
50000
60000

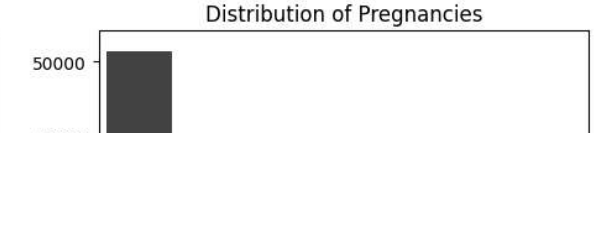

Distribution of Diet_type
Count
Diet_type
paleo
pescatarian
carnivore
low carb
vegetarian
mediterranean
raw food
vegan
ketogenic
atkins
gluten free
weight watchers
0
1000
2000
3000
4000
5000
6000
7000

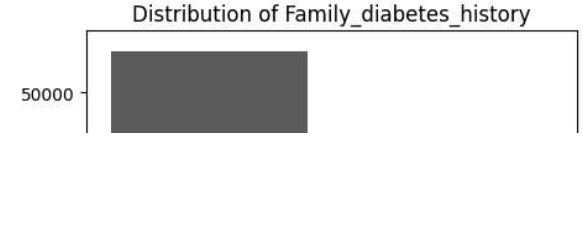

Distribution of Alcohol_consumption
Count
Alcohol_consumption
light
none
heavy
moderate
0
5000
10000
15000
20000
25000
30000

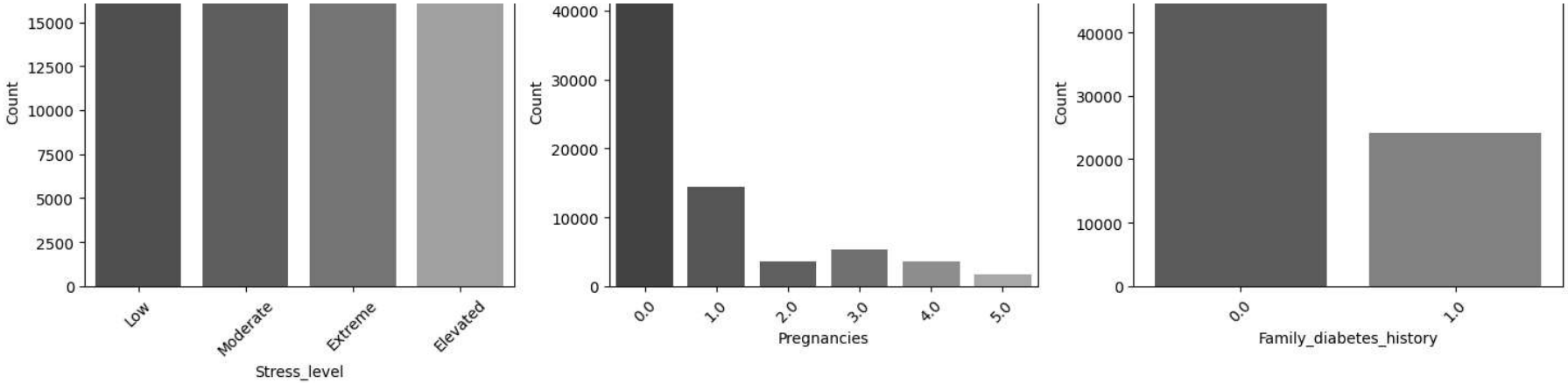

Distribution of Social_media_usage
Count
Social_media_usage
Occasionally
Excessive
Never
Moderate
0
2500
5000
7500
10000
12500
15000
17500
20000


Distribution of Physical_activity_level
Count
Physical_activity_level
Sedentary
Lightly Active
Moderately Active
Extremely Active
Very Active
0
10000
20000
30000
40000


Distribution of Stress_level
17500
20000


Distribution of Pregnancies
50000


Distribution of Family_diabetes_history
50000



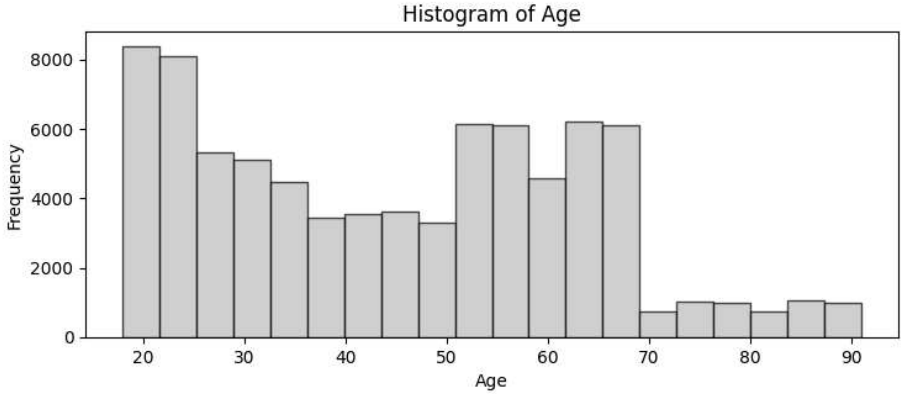
The bar charts indicate that categorical features such `as diet_type`, `social_media_usage`, and `stress_level` exhibit a relatively balanced distribution across their respective categories, with no single category dominating significantly.

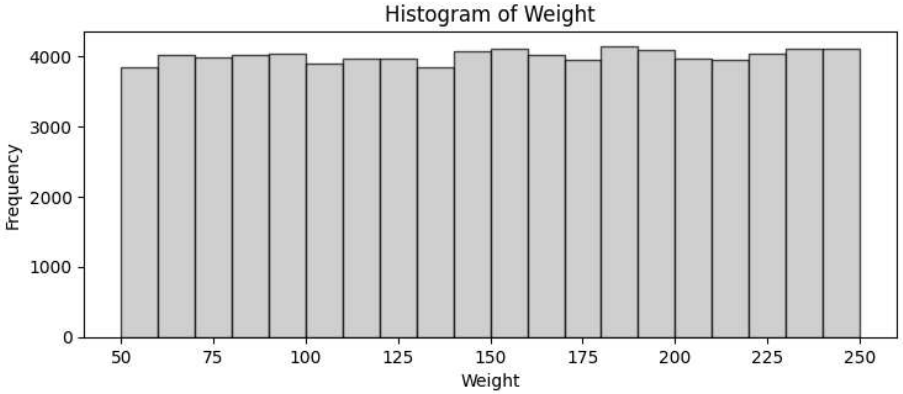
In contrast, features such as `hypertension, alcohol_consumption, physical_activity_level, pregnancies, and family_diabetes_history` demonstrate a skewed distribution, where one particular category constitutes a significantly larger proportion of the dataset compared to others. This suggests potential class imbalance in these features.

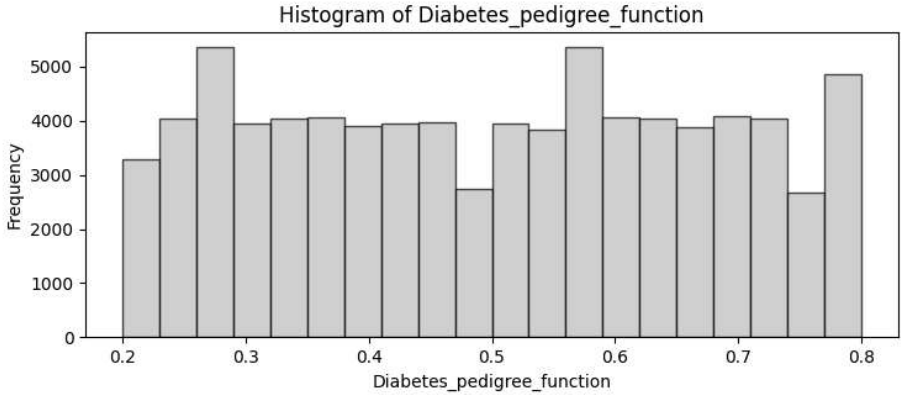
Let's now look at the pictorial representation of the distribution of continuous or numerical feature columns:

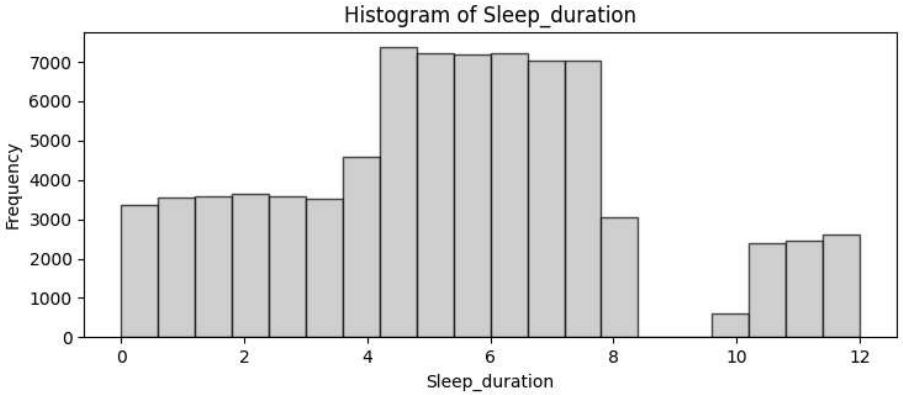
In [33]:
```
# List of 5 continuous features to plot
continuous_columns = ['age', 'weight', 'diabetes_pedigree_function', 'sleep_duration', 'BMI']

# Sets up a 3x2 grid for the plots
fig, axes = plt.subplots(3, 2, figsize=(15, 10))
axes = axes.flatten()

# Plots histograms for each continuous feature
for i, column in enumerate(continuous_columns):
    axes[i].hist(diabetes_ds[column], bins=20, color='skyblue', edgecolor='black', alpha=0.7)
    axes[i].set_title(f"Histogram of {column.capitalize()}", fontsize=12)
    axes[i].set_xlabel(column.capitalize(), fontsize=10)
    axes[i].set_ylabel("Frequency", fontsize=10)

# Hides unused subplots
for j in range(len(continuous_columns), len(axes)):
```

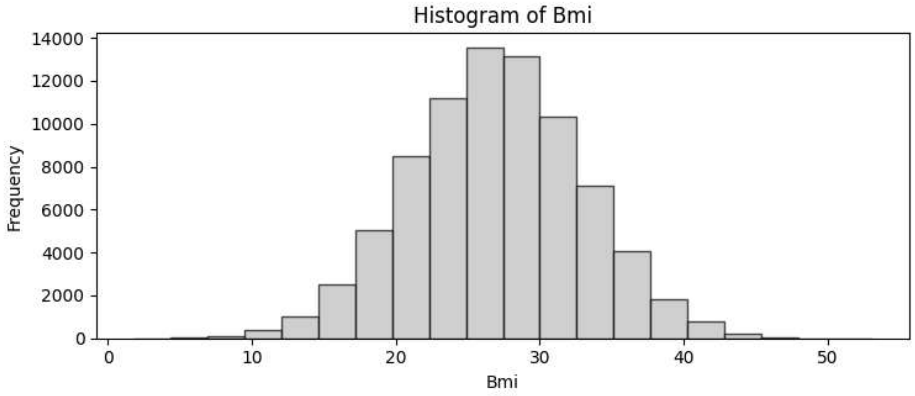
```
    fig.delaxes(axes[j])

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











From the above histograms plots, we can infer that the feature column `age` is right-skewed, columns `weight`, `diabetes_pedigree_function` and `sleep_duration` are uniformly distributed with sleep_duration looks having few outliers.

The `BMI` feature column is normally distributed.

Data Cleaning & PreProcessing

Handling NAs & Unwanted features

In this step, we will remove all rows where the target variable `diabetes` contains null or NaN values, as these entries cannot contribute to our analysis. Dropping rows with missing values in the target variable is more appropriate than imputing them, as imputations may introduce bias or inaccuracies in the target prediction.

Furthermore, the `star_sign` feature is deemed **irrelevant to the analysis** and does not provide any meaningful contribution to the model. Therefore, this feature will be **dropped** and excluded from further analysis.

In [34]:
```
diabetes_ds = diabetes_ds.dropna(subset=['diabetes'])

diabetes_ds = diabetes_ds.drop('star_sign', axis=1)
```

In [35]:
```
diabetes_ds.head(5)
```

Out[35]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usage	physical_
0	female	NaN	NaN	0.37	paleo	NaN	197.7	NaN	Occasionally	
1	male	59.0	0.0	0.73	NaN	17.1	156.5	0.0	Occasionally	
2	female	31.0	NaN	NaN	NaN	22.5	137.8	NaN	Occasionally	
4	NaN	64.0	0.0	0.55	carnivore	NaN	179.7	NaN	Occasionally	
5	male	50.0	0.0	0.43	low carb	NaN	NaN	0.0	Excessive	

In [36]:
```
diabetes_ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 80242 entries, 0 to 99999
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   gender                       64068 non-null  object 
 1   age                          64265 non-null  float64
 2   hypertension                 64333 non-null  float64
 3   diabetes_pedigree_function   64270 non-null  float64
 4   diet_type                    64163 non-null  object 
 5   BMI                          64080 non-null  float64
 6   weight                       64313 non-null  float64
 7   family_diabetes_history      64029 non-null  float64
 8   social_media_usage           64112 non-null  object 
 9   physical_activity_level      64175 non-null  object 
 10  sleep_duration               64259 non-null  float64
 11  stress_level                 64291 non-null  object 
 12  pregnancies                  64237 non-null  float64
 13  alcohol_consumption          64095 non-null  object 
 14  diabetes                     80242 non-null  float64
dtypes: float64(9), object(6)
memory usage: 9.8+ MB
```

Now the dataset is reduced to **80242** rows and **15** columns with no NaN values in the target column.

Handling Outliers

Previously, we observed potential outliers in the dataset based on the **histogram plots**. To validate this observation, we will utilize **box plots** and compute the **Interquartile Range (IQR)** for each individual categorical feature in the dataset.
This approach will help us accurately identify and confirm the presence of outliers.

In [37]:
```python
# Function to determine the outliers
def find_outliers_IQR(df):
    q1=df.quantile(0.25)
    q3=df.quantile(0.75)
    IQR=q3-q1
    #outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
    outlier_mask = (df < (q1 - 1.5 * IQR)) | (df > (q3 + 1.5 * IQR))
```

```
    outliers = df[outlier_mask]
    return outliers

for column in continuous_columns:
    outliers = find_outliers_IQR(diabetes_ds[column])
    print('Number of outliers in ' + column +' :'+ str(len(outliers)))
    print('Max outlier value: '+ str(outliers.max()))
    print('Min outlier value: '+ str(outliers.min()))
```

```
Number of outliers in age :0
Max outlier value: nan
Min outlier value: nan
Number of outliers in weight :0
Max outlier value: nan
Min outlier value: nan
Number of outliers in diabetes_pedigree_function :0
Max outlier value: nan
Min outlier value: nan
Number of outliers in sleep_duration :0
Max outlier value: nan
Min outlier value: nan
Number of outliers in BMI :469
Max outlier value: 52.9
Min outlier value: 1.8
```

Based on the analysis, we can conclude that the dataset contains outliers for `BMI` feature column that requires removal.

In [38]:
```
diabetes_ds.head(5)
```

Out[38]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usage	physical_
0	female	NaN	NaN	0.37	paleo	NaN	197.7	NaN	Occasionally	
1	male	59.0	0.0	0.73	NaN	17.1	156.5	0.0	Occasionally	
2	female	31.0	NaN	NaN	NaN	22.5	137.8	NaN	Occasionally	
4	NaN	64.0	0.0	0.55	carnivore	NaN	179.7	NaN	Occasionally	
5	male	50.0	0.0	0.43	low carb	NaN	NaN	0.0	Excessive	

In [39]:
```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = diabetes_ds['BMI'].quantile(0.25)
Q3 = diabetes_ds['BMI'].quantile(0.75)

# Compute IQR
IQR = Q3 - Q1

# Define bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

diabetes_ds_cleaned = diabetes_ds[(diabetes_ds['BMI'] >= lower_bound) & (diabetes_ds['BMI'] <= upper_bound)]
```

In [40]:
```
diabetes_ds_cleaned
```

Out[40]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usage
1	male	59.0	0.0	0.73	NaN	17.1	156.5	0.0	Occasionally
2	female	31.0	NaN	NaN	NaN	22.5	137.8	NaN	Occasionally
6	male	NaN	0.0	0.21	vegetarian	24.8	91.1	1.0	Never
7	female	67.0	0.0	0.27	mediterranean	21.2	NaN	0.0	Moderate
9	NaN	24.0	0.0	0.31	raw food	30.7	210.5	1.0	Never
...	...	...	...	...	...	...	...	...	...
99988	female	22.0	1.0	0.79	atkins	27.0	125.0	NaN	Occasionally
99993	female	52.0	0.0	0.72	vegetarian	17.4	54.5	0.0	Moderate
99995	male	24.0	NaN	0.29	vegetarian	18.9	56.0	0.0	Never
99996	male	53.0	1.0	0.64	weight watchers	28.4	140.8	NaN	Excessive
99998	male	NaN	0.0	0.71	mediterranean	28.8	123.6	0.0	Excessive

63611 rows × 15 columns

Handling NAs in feature columns

As previously discussed during the exploration of the diabetes dataset, approximately **20% of the data contains missing values (NaN)**. To address these missing values, the most suitable approach involves **encoding techniques** for categorical features and **K-Nearest Neighbors (KNN) imputation** for continuous features, given the moderate proportion of missing data.

For simplicity, and since most of the categorical features in the dataset are **ordinal**, we will utilize **Label Encoding** to handle missing values in the corresponding columns. This approach ensures consistency and appropriateness in imputing missing values for these feature types.

In [41]:
```
# Columns to label encode
columns_to_label_encode = ["hypertension", "family_diabetes_history", "social_media_usage","physical_activity_level","stress_l
```

```
# Apply LabelEncoder to each column
label_encoders = {}
for col in columns_to_label_encode:
    label_encoders[col] = LabelEncoder()
    diabetes_ds_cleaned[col] = label_encoders[col].fit_transform(diabetes_ds_cleaned[col])
```

In [42]:
```
diabetes_ds_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 63611 entries, 1 to 99998
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   gender                       63611 non-null  int64  
 1   age                          50899 non-null  float64
 2   hypertension                 63611 non-null  int64  
 3   diabetes_pedigree_function   50976 non-null  float64
 4   diet_type                    63611 non-null  int64  
 5   BMI                          63611 non-null  float64
 6   weight                       51041 non-null  float64
 7   family_diabetes_history      63611 non-null  int64  
 8   social_media_usage           63611 non-null  int64  
 9   physical_activity_level      63611 non-null  int64  
 10  sleep_duration               50932 non-null  float64
 11  stress_level                 63611 non-null  int64  
 12  pregnancies                  63611 non-null  int64  
 13  alcohol_consumption          63611 non-null  int64  
 14  diabetes                     63611 non-null  int64  
dtypes: float64(5), int64(10)
memory usage: 7.8 MB
```

In [43]:
```
diabetes_ds_cleaned.head(5)
```

Out[43]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usage	physical_a
1	1	59.0	0	0.73	12	17.1	156.5	0	3	
2	0	31.0	2	NaN	12	22.5	137.8	2	3	
6	1	NaN	0	0.21	10	24.8	91.1	1	2	
7	0	67.0	0	0.27	5	21.2	NaN	0	1	
9	2	24.0	0	0.31	8	30.7	210.5	1	2	

We can observe that there are no null values now in any categorical features, as they are imputed with a separate encoded value by the algorithm.

Now, let's proceed with addressing the missing values in continuous features. Since we have a moderately sized dataset, it has been decided that the best approach is to use the KNN Imputation method to fill the missing values in the continuous features.

However, before handling the missing values, it is crucial to standardize the data because KNN relies on distance metrics (e.g., Euclidean distance) to identify neighbors. Without standardization, features with larger scales may dominate the distance calculation, leading to biased neighbor selection and inaccurate imputations. Standardizing scales all features to a similar range (mean = 0, standard deviation = 1), ensuring fair contributions from all features and improving imputation accuracy.

Scaling Data

Feature scaling is an important step in pre-processing for Machine Learning. Most ML and optimization algorithms perform better when features are on the same scale.

For this dataset, we will scale the data using **Standardization**, as **Normalization** should be implemented on a case-by-case basis depending on the ML algorithms to be used for model building.

In [44]:
```
# Standardizing Data
standardScaler = StandardScaler()

diabetes_ds_cleaned[continuous_columns] = standardScaler.fit_transform(diabetes_ds_cleaned[continuous_columns])
```

In [45]:
```
diabetes_ds_cleaned.head(5)
```

Out[45]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usag
1	1	0.748635	0	1.311939	12	-1.696629	0.102709	0	
2	0	-0.762688	2	NaN	12	-0.769588	-0.221015	2	
6	1	NaN	0	-1.677502	10	-0.374737	-1.029460	1	
7	0	1.180442	0	-1.332566	5	-0.992764	NaN	0	
9	2	-1.140519	0	-1.102609	8	0.638142	1.037527	1	

After Standardization of the dataset, the selected features are transformed to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.

This scaling happened independently on each individual selected feature by computing the relevant statistics of the samples in the dataset.

In [46]:
```
# Initialize the KNNImputer
knn_imputer = KNNImputer(n_neighbors=10)

# Apply the imputer to the selected columns
imputed_columns = knn_imputer.fit_transform(diabetes_ds_cleaned[continuous_columns])

# Replace the original columns with the imputed values
diabetes_ds_cleaned[continuous_columns] = imputed_columns
```

The choice of **n_neighbors=10** for KNN Imputer in our scenario is based on a balance of theoretical reasoning, practical experience, and the dataset characteristics such as dataset size (~80000 rows) and missing value proprotion (20%).

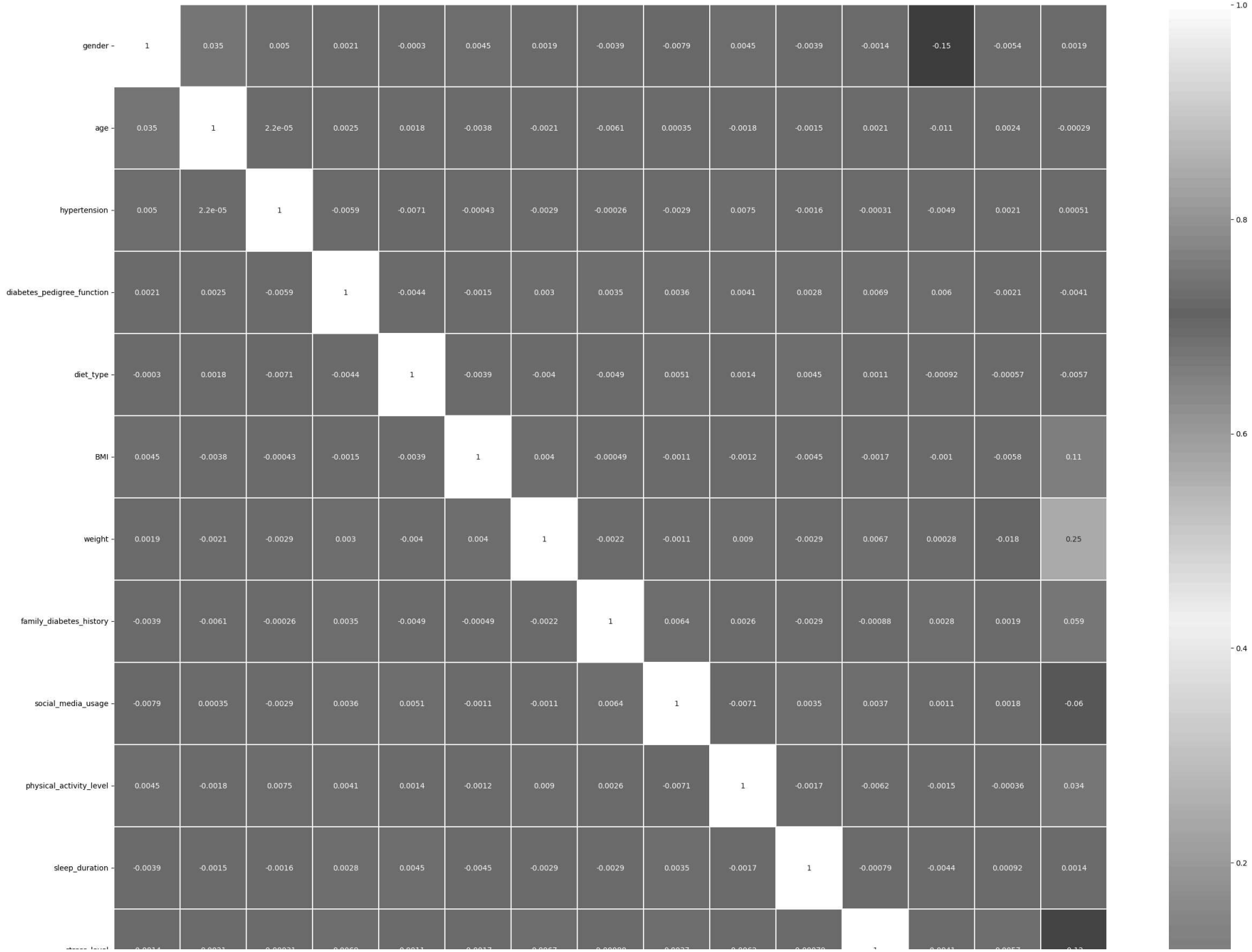
In [47]:
```
diabetes_ds_cleaned.info()
```

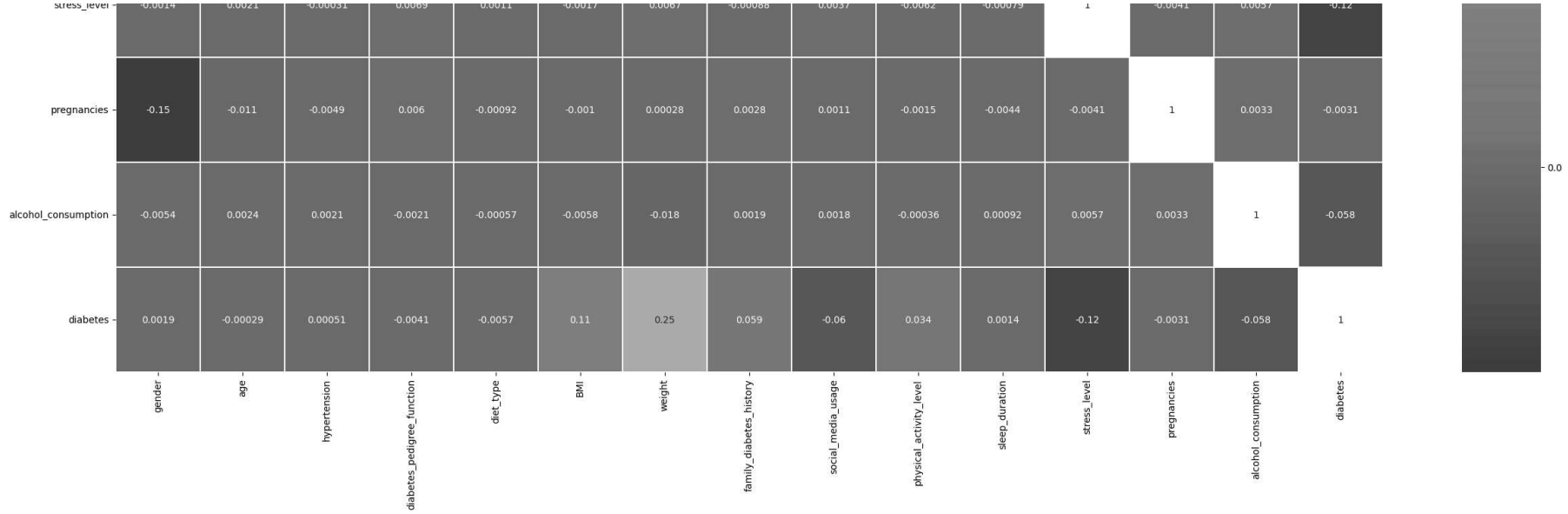
```
<class 'pandas.core.frame.DataFrame'>
Index: 63611 entries, 1 to 99998
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   gender                       63611 non-null  int64  
 1   age                          63611 non-null  float64
 2   hypertension                 63611 non-null  int64  
 3   diabetes_pedigree_function   63611 non-null  float64
 4   diet_type                    63611 non-null  int64  
 5   BMI                          63611 non-null  float64
 6   weight                       63611 non-null  float64
 7   family_diabetes_history      63611 non-null  int64  
 8   social_media_usage           63611 non-null  int64  
 9   physical_activity_level      63611 non-null  int64  
 10  sleep_duration               63611 non-null  float64
 11  stress_level                 63611 non-null  int64  
 12  pregnancies                  63611 non-null  int64  
 13  alcohol_consumption          63611 non-null  int64  
 14  diabetes                     63611 non-null  int64  
dtypes: float64(5), int64(10)
memory usage: 7.8 MB
```

All features in the dataset now have complete values, with no NaNs or missing entries. The dataset is fully prepared for further analysis.

Correlation Heatmap

In [48]:
```
# creating a correlation heatmap
sns.heatmap(diabetes_ds_cleaned.corr(),annot=True, cmap='terrain', linewidths=0.1)
fig=plt.gcf()
fig.set_size_inches(30,30)
plt.show()
```


gender 1 0.035 0.005 0.0021 -0.0003 0.0045 0.0019 -0.0039 -0.0079 0.0045 -0.0039 -0.0014 -0.15 -0.0054 0.0019
age 0.035 1 2.2e-05 0.0025 0.0018 -0.0038 -0.0021 -0.0061 0.00035 -0.0018 -0.0015 0.0021 -0.011 0.0024 -0.00029
hypertension 0.005 2.2e-05 1 -0.0059 -0.0071 -0.00043 -0.0029 -0.00026 -0.0029 0.0075 -0.0016 -0.00031 -0.0049 0.0021 0.00051
diabetes_pedigree_function 0.0021 0.0025 -0.0059 1 -0.0044 -0.0015 0.003 0.0035 0.0036 0.0041 0.0028 0.0069 0.006 -0.0021 -0.0041
diet_type -0.0003 0.0018 -0.0071 -0.0044 1 -0.0039 -0.004 -0.0049 0.0051 0.0014 0.0045 0.0011 -0.00092 -0.00057 -0.0057
BMI 0.0045 -0.0038 -0.00043 -0.0015 -0.0039 1 0.004 -0.00049 -0.0011 -0.0012 -0.0045 -0.0017 -0.001 -0.0058 0.11
weight 0.0019 -0.0021 -0.0029 0.003 -0.004 0.004 1 -0.0022 -0.0011 0.009 -0.0029 0.0067 0.00028 -0.018 0.25
family_diabetes_history -0.0039 -0.0061 -0.00026 0.0035 -0.0049 -0.00049 -0.0022 1 0.0064 0.0026 -0.0029 -0.00088 0.0028 0.0019 0.059
social_media_usage -0.0079 0.00035 -0.0029 0.0036 0.0051 -0.0011 -0.0011 0.0064 1 -0.0071 0.0035 0.0037 0.0011 0.0018 -0.06
physical_activity_level 0.0045 -0.0018 0.0075 0.0041 0.0014 -0.0012 0.009 0.0026 -0.0071 1 -0.0017 -0.0062 -0.0015 -0.00036 0.034
sleep_duration -0.0039 -0.0015 -0.0016 0.0028 0.0045 -0.0045 -0.0029 -0.0029 0.0035 -0.0017 1 -0.00079 -0.0044 0.00092 0.0014
1.0
0.8
0.6
0.4
0.2



The above correlation heatmap shows the relationship between various features and the target variable `diabetes` using correlation coefficients.

Key Observations:

- **Weight** has the highest positive correlation (**0.25**), suggesting it is an important predictor of diabetes.
- **BMI (Body Mass Index)** and **stress_level** show comparitively strong correlations (**0.099** and **0.12**, respectively).
- Features such as **social_media_usage**, **family_diabetes_history**, **alcohol_consumption** and **physical_activity** show moderate and weak correlations.
- Features like **gender**, **age**, **hypertension**, **diabetes_pedigree_function**, **diet_type**, **sleep_duration** and **pregnancies** have negligible correlations (values near 0), meaning they have limited or no direct linear impact on diabetes in this dataset.

Based on the analysis, we can conclude that the following seven factors have a significant impact on the likelihood of developing diabetes in humans.

- **weight**
- **BMI**

- **stress_level**
- **social_media_usage**
- **family_diabetes_history**
- **alcohol_consumption**
- **physical_activity**

Dumping the cleaned dataset to a CSV file

In [49]:
```
diabetes_ds_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 63611 entries, 1 to 99998
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   gender                       63611 non-null  int64
 1   age                          63611 non-null  float64
 2   hypertension                 63611 non-null  int64
 3   diabetes_pedigree_function   63611 non-null  float64
 4   diet_type                    63611 non-null  int64
 5   BMI                          63611 non-null  float64
 6   weight                       63611 non-null  float64
 7   family_diabetes_history      63611 non-null  int64
 8   social_media_usage           63611 non-null  int64
 9   physical_activity_level      63611 non-null  int64
 10  sleep_duration               63611 non-null  float64
 11  stress_level                 63611 non-null  int64
 12  pregnancies                  63611 non-null  int64
 13  alcohol_consumption          63611 non-null  int64
 14  diabetes                     63611 non-null  int64
dtypes: float64(5), int64(10)
memory usage: 7.8 MB
```

In [50]:
```
#Need to decode all the cateogorical and continous columns
diabetes_ds_cleaned.head(5)
```

Out[50]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usag
1	1	0.748635	0	1.311939	12	-1.696629	0.102709	0	
2	0	-0.762688	2	-0.211526	12	-0.769588	-0.221015	2	
6	1	0.160299	0	-1.677502	10	-0.374737	-1.029460	1	
7	0	1.180442	0	-1.332566	5	-0.992764	-0.255119	0	
9	2	-1.140519	0	-1.102609	8	0.638142	1.037527	1	

In [ ]:

```
from sklearn.preprocessing import LabelEncoder, StandardScaler

# Decode the label-encoded columns
for col in columns_to_label_encode:
    if col in label_encoders:
        diabetes_ds_cleaned[col] = label_encoders[col].inverse_transform(diabetes_ds_cleaned[col])

# Display a preview of the decoded dataset
print(diabetes_ds_cleaned.head())
```

```
   gender       age  hypertension  diabetes_pedigree_function      diet_type  \
1    male  0.748635           0.0                    1.311939            NaN
2  female -0.762688           NaN                   -0.211526            NaN
6    male  0.160299           0.0                   -1.677502     vegetarian
7  female  1.180442           0.0                   -1.332566  mediterranean
9     NaN -1.140519           0.0                   -1.102609       raw food

        BMI    weight  family_diabetes_history social_media_usage  \
1 -1.696629  0.102709                      0.0       Occasionally
2 -0.769588 -0.221015                      NaN       Occasionally
6 -0.374737 -1.029460                      1.0              Never
7 -0.992764 -0.255119                      0.0           Moderate
9  0.638142  1.037527                      1.0              Never

  physical_activity_level  sleep_duration stress_level  pregnancies  \
1          Lightly Active        0.039820     Moderate          0.0
2          Lightly Active        0.813489          Low          0.0
6              Sedentary       -1.472350     Moderate          0.0
7       Moderately Active        0.532155     Moderate          1.0
9              Sedentary        1.692657      Extreme          0.0

  alcohol_consumption  diabetes
1                none       1.0
2               light       1.0
6                 NaN       1.0
7                 NaN       1.0
9            moderate       1.0
```

In [52]:
```
diabetes_ds_cleaned[continuous_columns] = standardScaler.inverse_transform(diabetes_ds_cleaned[continuous_columns])
diabetes_ds_cleaned.head(5)
```

Out[52]:

	gender	age	hypertension	diabetes_pedigree_function	diet_type	BMI	weight	family_diabetes_history	social_media_usage	physi
1	male	59.0	0.0	0.730	NaN	17.1	156.50	0.0	Occasionally	
2	female	31.0	NaN	0.465	NaN	22.5	137.80	NaN	Occasionally	
6	male	48.1	0.0	0.210	vegetarian	24.8	91.10	1.0	Never	
7	female	67.0	0.0	0.270	mediterranean	21.2	135.83	0.0	Moderate	N
9	NaN	24.0	0.0	0.310	raw food	30.7	210.50	1.0	Never	

In [53]:

```
file_path = "../data/cleaned_data/diabetes_data_cleansed_AV.csv"

diabetes_ds_cleaned.to_csv(file_path, index=False)
```