AAI_510_Final_Project_Merged

June 22, 2024

[ ]:
```python
# supress warnings
import warnings
warnings.filterwarnings("ignore")

from scipy.stats import chi2_contingency
from scipy.stats import ttest_ind

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #to allow subplot creation

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.model_selection import train_test_split

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix

from dataprep.eda import *
from dataprep.datasets import load_dataset
from dataprep.eda import plot, plot_correlation, plot_missing, plot_diff,␣
↪create_report #

# Apply the seaborn theme
sns.set_theme() #overwrite default Matplotlib styling parameters
```

0.1 Problem Statement

0.1.1 Objective

The goal of this project is to create a model that can predict the presence of diabetes in individuals by analyzing data from the Behavioral Risk Factor Surveillance System (BRFSS) 2015 survey. The main objective is to accurately forecast whether someone has diabetes ($Outcome = 2$), prediabetes ($Outcome = 1$) or no diabetes ($Outcome = 0$) based on specific health measurements and indicators.

0.1.2 Background

Diabetes poses a significant challenge to public health, impacting numerous Americans and leading to severe health complications and considerable financial strains. Timely identification and intervention can greatly enhance patient outcomes by facilitating prompt adjustments in lifestyle and appropriate treatments.

This initiative intends to harness machine learning methods to bolster the early recognition of diabetes, potentially assisting healthcare professionals in making well informed choices and enhancing patient well being.

0.1.3 Health Indicators and Indicators for Diagnosis

The model will consider a range of health indicators, such as high blood pressure, elevated cholesterol levels, recent cholesterol screenings, BMI, smoking habits, history of stroke or heart disease/attack, level of physical activity, consumption of fruits and vegetables, excessive alcohol intake, access to healthcare services, financial obstacles hindering doctor visits, overall health condition perception, number of days with mental or physical health issues, mobility difficulties, gender identity, age groupings education attainment levels and income brackets.

These signs offer a detailed overview of a person's well being and habits, which are essential for predicting the likelihood of developing diabetes.

0.2 Data understanding (EDA)

[ ]:
```python
# Load data and print dataframe shape
file_path = "content/diabetes_012_health_indicators_BRFSS2015.csv"
df = pd.read_csv(file_path)

shape = df.shape
print("Shape of the dataframe (row, col):",shape,"\r\n")

# Show the dataframe
pd.set_option('display.max_columns', None)
df
```

```
Shape of the dataframe (row, col): (253680, 22)
```

[ ]:

```
        Diabetes_012  HighBP  HighChol  CholCheck   BMI  Smoker  Stroke  \
0                0.0     1.0       1.0        1.0  40.0     1.0     0.0
1                0.0     0.0       0.0        0.0  25.0     1.0     0.0
2                0.0     1.0       1.0        1.0  28.0     0.0     0.0
3                0.0     1.0       0.0        1.0  27.0     0.0     0.0
4                0.0     1.0       1.0        1.0  24.0     0.0     0.0
…                  …       …         …          …     …       …       …
253675           0.0     1.0       1.0        1.0  45.0     0.0     0.0
253676           2.0     1.0       1.0        1.0  18.0     0.0     0.0
253677           0.0     0.0       0.0        1.0  28.0     0.0     0.0
253678           0.0     1.0       0.0        1.0  23.0     0.0     0.0
253679           2.0     1.0       1.0        1.0  25.0     0.0     0.0

        HeartDiseaseorAttack  PhysActivity  Fruits  Veggies  \
0                        0.0           0.0     0.0      1.0
1                        0.0           1.0     0.0      0.0
2                        0.0           0.0     1.0      0.0
3                        0.0           1.0     1.0      1.0
4                        0.0           1.0     1.0      1.0
…                          …             …       …        …
253675                   0.0           0.0     1.0      1.0
253676                   0.0           0.0     0.0      0.0
253677                   0.0           1.0     1.0      0.0
253678                   0.0           0.0     1.0      1.0
253679                   1.0           1.0     1.0      0.0

        HvyAlcoholConsump  AnyHealthcare  NoDocbcCost  GenHlth  MentHlth  \
0                     0.0            1.0          0.0      5.0      18.0
1                     0.0            0.0          1.0      3.0       0.0
2                     0.0            1.0          1.0      5.0      30.0
3                     0.0            1.0          0.0      2.0       0.0
4                     0.0            1.0          0.0      2.0       3.0
…                       …              …            …        …         …
253675                0.0            1.0          0.0      3.0       0.0
253676                0.0            1.0          0.0      4.0       0.0
253677                0.0            1.0          0.0      1.0       0.0
253678                0.0            1.0          0.0      3.0       0.0
253679                0.0            1.0          0.0      2.0       0.0

        PhysHlth  DiffWalk  Sex   Age  Education  Income
0           15.0       1.0  0.0   9.0        4.0     3.0
1            0.0       0.0  0.0   7.0        6.0     1.0
2           30.0       1.0  0.0   9.0        4.0     8.0
3            0.0       0.0  0.0  11.0        3.0     6.0
4            0.0       0.0  0.0  11.0        5.0     4.0
…              …         …    …     …          …       …
253675       5.0       0.0  1.0   5.0        6.0     7.0
```

```
253676       0.0       1.0  0.0  11.0         2.0      4.0
253677       0.0       0.0  0.0   2.0         5.0      2.0
253678       0.0       0.0  1.0   7.0         5.0      1.0
253679       0.0       0.0  0.0   9.0         6.0      2.0

[253680 rows x 22 columns]
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253680 entries, 0 to 253679
Data columns (total 22 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Diabetes_012          253680 non-null  float64
 1   HighBP                253680 non-null  float64
 2   HighChol              253680 non-null  float64
 3   CholCheck             253680 non-null  float64
 4   BMI                   253680 non-null  float64
 5   Smoker                253680 non-null  float64
 6   Stroke                253680 non-null  float64
 7   HeartDiseaseorAttack  253680 non-null  float64
 8   PhysActivity          253680 non-null  float64
 9   Fruits                253680 non-null  float64
 10  Veggies               253680 non-null  float64
 11  HvyAlcoholConsump     253680 non-null  float64
 12  AnyHealthcare         253680 non-null  float64
 13  NoDocbcCost           253680 non-null  float64
 14  GenHlth               253680 non-null  float64
 15  MentHlth              253680 non-null  float64
 16  PhysHlth              253680 non-null  float64
 17  DiffWalk              253680 non-null  float64
 18  Sex                   253680 non-null  float64
 19  Age                   253680 non-null  float64
 20  Education             253680 non-null  float64
 21  Income                253680 non-null  float64
dtypes: float64(22)
memory usage: 42.6 MB
```

```
None
```

0.2.1 Initial Plot for EDA

[ ]:
```
plot(df)
```

```
  0%|          | 0/714 [00:00<?, ?it/s]
```

[ ]:
```
<dataprep.eda.container.Container at 0x177416410>
```

0.2.2 Dataset Overview

Total Variables: `22` Total Rows: `253,680` Missing Data: `None reported` Duplicate Rows: `23,899 (9.4%)` Memory Usage: `42.6 MB` Data Types: All variables are numeric but largely categorical by nature.

Variable Descriptions Categorical (Binary or Multi-Class): Most of the variables are categorical, with binary encoding (e.g., HighBP, Smoker, PhysActivity) except Diabetes_012, which is a multi-class categorical variable. Numerical: BMI is a true continuous variable and provides an actual measure rather than a category. Key Insights and Considerations No Missing Values: This indicates that the dataset is complete, which simplifies preprocessing but should be verified for correctness (e.g., no improper imputation).

Duplicates: The presence of about 9.4% duplicate rows should be addressed. Determine if these duplicates are due to data entry errors or if they represent legitimate repeated measurements.

Variable Characteristics: Diabetes Status (`Diabetes_012`): Captures the absence or presence of diabetes and its stage, useful for detailed health-related analyses. Lifestyle and Health Checks (`Smoker, CholCheck`): Reflect lifestyle choices and adherence to health monitoring, relevant for risk factor analysis. Physical Health Metrics (`BMI, PhysActivity`): Directly measures aspects of physical health and activity, crucial for studying correlations with health outcomes.

Data Skewness and Distribution: Skewed Variables (`MentHlth, PhysHlth`): High zero count suggests many participants report no issues, which might require special modeling techniques like zero-inflated models if predicting these conditions. BMI Distribution: Given its role as a continuous variable, the analysis of its distribution (e.g., normality, presence of outliers) is important for understanding population health metrics. Binary Variables: Many variables are binary, indicating conditions like high blood pressure, smoking status, or having had a stroke. These will be particularly useful in logistic regression models or similar statistical tests to determine risk factors for various health conditions.

0.2.3 Understanding Correlation

[ ]:
```
plot_correlation(df)
```

```
100%|##########| 4/4 [00:00<?, ?it/s]
```

[ ]:
```
<dataprep.eda.container.Container at 0x1774b60e0>
```

[ ]:
```
correlation = df.corr()
correlation
```

[ ]:
```
              Diabetes_012    HighBP  HighChol  CholCheck       BMI  \
Diabetes_012      1.000000  0.271596  0.209085   0.067546  0.224379
HighBP            0.271596  1.000000  0.298199   0.098508  0.213748
HighChol          0.209085  0.298199  1.000000   0.085642  0.106722
CholCheck         0.067546  0.098508  0.085642   1.000000  0.034495
BMI               0.224379  0.213748  0.106722   0.034495  1.000000
Smoker            0.062914  0.096991  0.091299  -0.009929  0.013804
```

Stroke	0.107179	0.129575	0.092620	0.024158	0.020153
HeartDiseaseorAttack	0.180272	0.209361	0.180765	0.044206	0.052904
PhysActivity	-0.121947	-0.125267	-0.078046	0.004190	-0.147294
Fruits	-0.042192	-0.040555	-0.040859	0.023849	-0.087518
Veggies	-0.058972	-0.061266	-0.039874	0.006121	-0.062275
HvyAlcoholConsump	-0.057882	-0.003972	-0.011543	-0.023730	-0.048736
AnyHealthcare	0.015410	0.038425	0.042230	0.117626	-0.018471
NoDocbcCost	0.035436	0.017358	0.013310	-0.058255	0.058206
GenHlth	0.302587	0.300530	0.208426	0.046589	0.239185
MentHlth	0.073507	0.056456	0.062069	-0.008366	0.085310
PhysHlth	0.176287	0.161212	0.121751	0.031775	0.121141
DiffWalk	0.224239	0.223618	0.144672	0.040585	0.197078
Sex	0.031040	0.052207	0.031205	-0.022115	0.042950
Age	0.185026	0.344452	0.272318	0.090321	-0.036618
Education	-0.130517	-0.141358	-0.070802	0.001510	-0.103932
Income	-0.171483	-0.171235	-0.085459	0.014259	-0.100069

	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity \
Diabetes_012	0.062914	0.107179	0.180272	-0.121947
HighBP	0.096991	0.129575	0.209361	-0.125267
HighChol	0.091299	0.092620	0.180765	-0.078046
CholCheck	-0.009929	0.024158	0.044206	0.004190
BMI	0.013804	0.020153	0.052904	-0.147294
Smoker	1.000000	0.061173	0.114441	-0.087401
Stroke	0.061173	1.000000	0.203002	-0.069151
HeartDiseaseorAttack	0.114441	0.203002	1.000000	-0.087299
PhysActivity	-0.087401	-0.069151	-0.087299	1.000000
Fruits	-0.077666	-0.013389	-0.019790	0.142756
Veggies	-0.030678	-0.041124	-0.039167	0.153150
HvyAlcoholConsump	0.101619	-0.016950	-0.028991	0.012392
AnyHealthcare	-0.023251	0.008776	0.018734	0.035505
NoDocbcCost	0.048946	0.034804	0.031000	-0.061638
GenHlth	0.163143	0.177942	0.258383	-0.266186
MentHlth	0.092196	0.070172	0.064621	-0.125587
PhysHlth	0.116460	0.148944	0.181698	-0.219230
DiffWalk	0.122463	0.176567	0.212709	-0.253174
Sex	0.093662	0.002978	0.086096	0.032482
Age	0.120641	0.126974	0.221618	-0.092511
Education	-0.161955	-0.076009	-0.099600	0.199658
Income	-0.123937	-0.128599	-0.141011	0.198539

	Fruits	Veggies	HvyAlcoholConsump	AnyHealthcare \
Diabetes_012	-0.042192	-0.058972	-0.057882	0.015410
HighBP	-0.040555	-0.061266	-0.003972	0.038425
HighChol	-0.040859	-0.039874	-0.011543	0.042230
CholCheck	0.023849	0.006121	-0.023730	0.117626
BMI	-0.087518	-0.062275	-0.048736	-0.018471

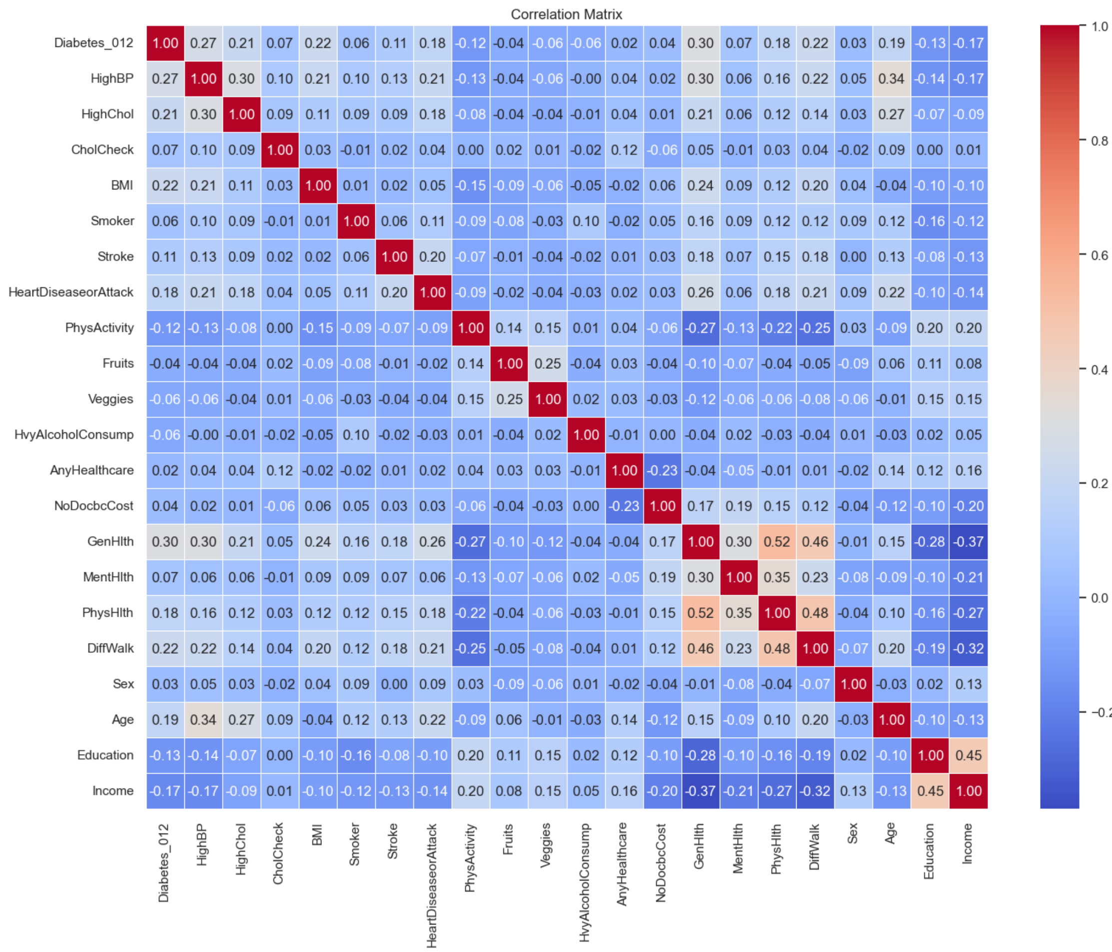
Smoker	-0.077666	-0.030678	0.101619	-0.023251
Stroke	-0.013389	-0.041124	-0.016950	0.008776
HeartDiseaseorAttack	-0.019790	-0.039167	-0.028991	0.018734
PhysActivity	0.142756	0.153150	0.012392	0.035505
Fruits	1.000000	0.254342	-0.035288	0.031544
Veggies	0.254342	1.000000	0.021064	0.029584
HvyAlcoholConsump	-0.035288	0.021064	1.000000	-0.010488
AnyHealthcare	0.031544	0.029584	-0.010488	1.000000
NoDocbcCost	-0.044243	-0.032232	0.004684	-0.232532
GenHlth	-0.103854	-0.123066	-0.036724	-0.040817
MentHlth	-0.068217	-0.058884	0.024716	-0.052707
PhysHlth	-0.044633	-0.064290	-0.026415	-0.008276
DiffWalk	-0.048352	-0.080506	-0.037668	0.007074
Sex	-0.091175	-0.064765	0.005740	-0.019405
Age	0.064547	-0.009771	-0.034578	0.138046
Education	0.110187	0.154329	0.023997	0.122514
Income	0.079929	0.151087	0.053619	0.157999

	NoDocbcCost	GenHlth	MentHlth	PhysHlth	DiffWalk \
Diabetes_012	0.035436	0.302587	0.073507	0.176287	0.224239
HighBP	0.017358	0.300530	0.056456	0.161212	0.223618
HighChol	0.013310	0.208426	0.062069	0.121751	0.144672
CholCheck	-0.058255	0.046589	-0.008366	0.031775	0.040585
BMI	0.058206	0.239185	0.085310	0.121141	0.197078
Smoker	0.048946	0.163143	0.092196	0.116460	0.122463
Stroke	0.034804	0.177942	0.070172	0.148944	0.176567
HeartDiseaseorAttack	0.031000	0.258383	0.064621	0.181698	0.212709
PhysActivity	-0.061638	-0.266186	-0.125587	-0.219230	-0.253174
Fruits	-0.044243	-0.103854	-0.068217	-0.044633	-0.048352
Veggies	-0.032232	-0.123066	-0.058884	-0.064290	-0.080506
HvyAlcoholConsump	0.004684	-0.036724	0.024716	-0.026415	-0.037668
AnyHealthcare	-0.232532	-0.040817	-0.052707	-0.008276	0.007074
NoDocbcCost	1.000000	0.166397	0.192107	0.148998	0.118447
GenHlth	0.166397	1.000000	0.301674	0.524364	0.456920
MentHlth	0.192107	0.301674	1.000000	0.353619	0.233688
PhysHlth	0.148998	0.524364	0.353619	1.000000	0.478417
DiffWalk	0.118447	0.456920	0.233688	0.478417	1.000000
Sex	-0.044931	-0.006091	-0.080705	-0.043137	-0.070299
Age	-0.119777	0.152450	-0.092068	0.099130	0.204450
Education	-0.100701	-0.284912	-0.101830	-0.155093	-0.192642
Income	-0.203182	-0.370014	-0.209806	-0.266799	-0.320124

	Sex	Age	Education	Income
Diabetes_012	0.031040	0.185026	-0.130517	-0.171483
HighBP	0.052207	0.344452	-0.141358	-0.171235
HighChol	0.031205	0.272318	-0.070802	-0.085459
CholCheck	-0.022115	0.090321	0.001510	0.014259

```
BMI                   0.042950 -0.036618  -0.103932 -0.100069
Smoker                0.093662  0.120641  -0.161955 -0.123937
Stroke                0.002978  0.126974  -0.076009 -0.128599
HeartDiseaseorAttack  0.086096  0.221618  -0.099600 -0.141011
PhysActivity          0.032482 -0.092511   0.199658  0.198539
Fruits               -0.091175  0.064547   0.110187  0.079929
Veggies              -0.064765 -0.009771   0.154329  0.151087
HvyAlcoholConsump     0.005740 -0.034578   0.023997  0.053619
AnyHealthcare        -0.019405  0.138046   0.122514  0.157999
NoDocbcCost          -0.044931 -0.119777  -0.100701 -0.203182
GenHlth              -0.006091  0.152450  -0.284912 -0.370014
MentHlth             -0.080705 -0.092068  -0.101830 -0.209806
PhysHlth             -0.043137  0.099130  -0.155093 -0.266799
DiffWalk             -0.070299  0.204450  -0.192642 -0.320124
Sex                   1.000000 -0.027340   0.019480  0.127141
Age                  -0.027340  1.000000  -0.101901 -0.127775
Education             0.019480 -0.101901   1.000000  0.449106
Income                0.127141 -0.127775   0.449106  1.000000
```

[ ]:
```
# use sns heatmap to plot the correlation matrix
plt.figure(figsize=(16, 12))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



0.2.4 Notable Positive Correlations

- **`Diabetes_012` and `GenHlth` (0.302587)**: Higher diabetes categorization correlates with worse general health ratings. This suggests that as diabetes severity increases, overall health perceptions tend to decline.
- **`HighBP` and `Age` (0.344452)**: Older age groups tend to have higher incidences of high blood pressure.
- **`PhysHlth` and `GenHlth` (0.524364)**: More days with poor physical health correlate strongly with poorer general health ratings.
- **`MentHlth` and `PhysHlth` (0.353619)**: A significant positive correlation indicating that more days with mental health issues are associated with more days of poor physical health.
- **`DiffWalk` and `PhysHlth` (0.478417)**: Difficulty walking correlates with more days of poor physical health, suggesting mobility issues are associated with worse physical conditions.

0.2.5 Notable Negative Correlations

- **`PhysActivity` and `GenHlth` (-0.266186)**: Higher levels of physical activity correlate with better general health ratings.

- **`Income` and `GenHlth` (-0.370014)**: Higher income levels correlate with better general health, highlighting possible socioeconomic impacts on health.
- **`Education` and `GenHlth` (-0.284912)**: Higher education levels are associated with better general health ratings.

0.2.6 Implications for Analysis

- **Health Outcomes**: Variables like `GenHlth`, `Diabetes_012`, `HighBP`, and `HighChol` can be used to model health outcomes, especially for understanding risk factors associated with chronic diseases.
- **Behavioral Factors**: The relationships between lifestyle choices (e.g., `Smoker`, `PhysActivity`, `Fruits`, `Veggies`) and health outcomes can inform public health interventions.
- **Socioeconomic Factors**: The strong correlations between `Income`, `Education`, and health variables suggest that socioeconomic factors are crucial determinants of health, which could be a focal point for deeper socioeconomic studies and policy-making.

0.3 Data Preparation

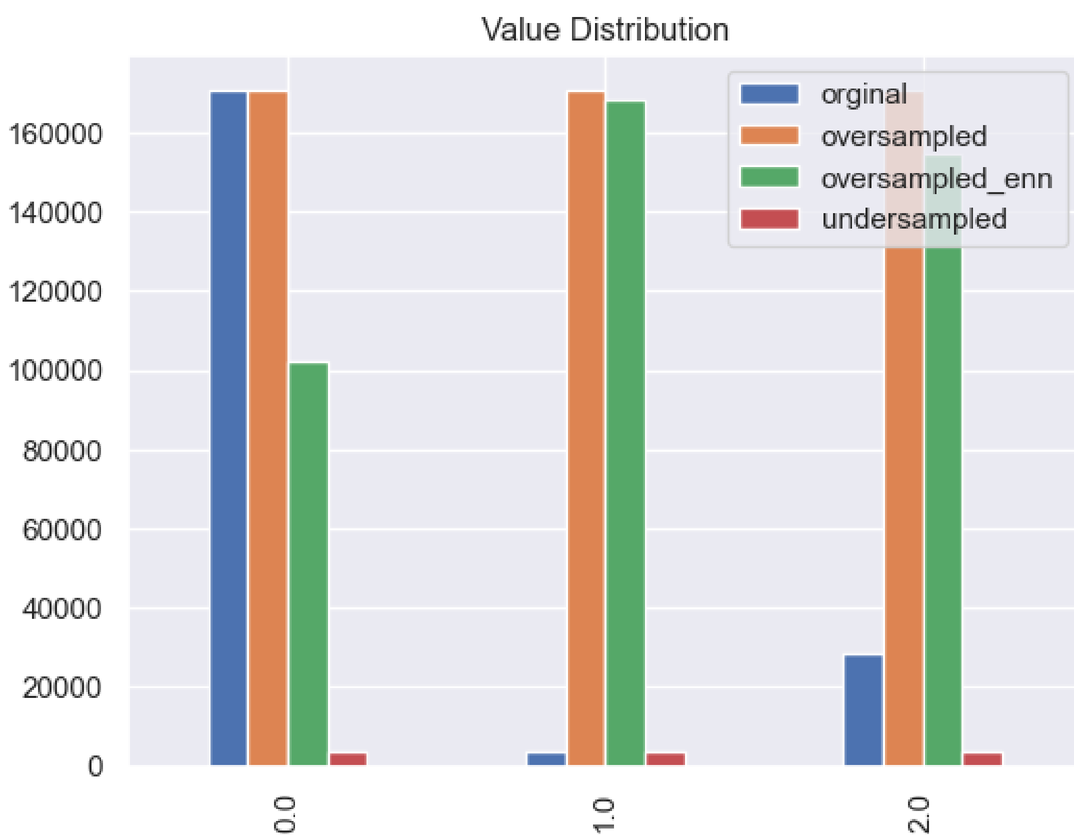
```
[ ]: # Classify variables
def classify_variables():
    numerical = ['BMI', 'MentHlth', 'PhysHlth']
    categorical = ['Diabetes_012', 'Age']
    binary = ['HighBP', 'HighChol', 'CholCheck', 'Smoker', 'Stroke',␣
↪'HeartDiseaseorAttack',
              'PhysActivity', 'Fruits', 'Veggies', 'HvyAlcoholConsump',␣
↪'AnyHealthcare',
              'NoDocbcCost', 'DiffWalk', 'Sex']
    ordinal = ['GenHlth', 'Education', 'Income']
    return numerical, categorical, binary, ordinal

numerical, categorical, binary, ordinal = classify_variables()

non_numerical = categorical + binary + ordinal  # All non-numerical variables

# Handle missing values for numerical variables
df[numerical] = df[numerical].fillna(df[numerical].median())

# Encode ordinal variables
le = LabelEncoder()
for col in ordinal:
    df[col] = le.fit_transform(df[col])

# Define the target and features
target_column = 'Diabetes_012'
features = df.columns.difference([target_column])
X = df[features]
y = df[target_column]
```

0.4 Feature Engineering

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Standardize the numerical features
scaler = StandardScaler()
X_train[numerical] = scaler.fit_transform(X_train[numerical])
X_test[numerical] = scaler.transform(X_test[numerical])

# Genereate oversample with SMOTE:
smote = SMOTE(random_state=42)
X_ovsampled, y_ovsampled = smote.fit_resample(X_train, y_train)

# Generate oversample with SMOTEENN:
smoteenn = SMOTEENN(random_state=42)
X_ovsampled_enn, y_ovsampled_enn = smoteenn.fit_resample(X_train, y_train)
```

```python
#  Generate undersample with RandomUnderSampler:
undersampler = RandomUnderSampler(random_state=42)
X_undersampled, y_undersampled = undersampler.fit_resample(X_train, y_train)

values_distribution = {
    'orginal': y_train.value_counts(),
    'oversampled': y_ovsampled.value_counts(),
    'oversampled_enn': y_ovsampled_enn.value_counts(),
    'undersampled': y_undersampled.value_counts()
}

# plot value distribution
plt.figure(figsize=(16, 8))
pd.DataFrame(values_distribution).plot(kind='bar')
plt.title('Value Distribution')
plt.show()
```

```
<Figure size 1600x800 with 0 Axes>
```



0.5 Feature Selection

0.5.1 Chi-Square Test

[ ]:
```
# Dataframe to store result of Chi2 test
results_df_cat = pd.DataFrame(columns=['Feature', 'Chi2', 'P-Value'])

# Copy to keep the original dataframe
cat_df = df.copy()

# for all_non numerical variables, convert to categorical
for var in non_numerical:
    cat_df[var] = cat_df[var].astype('category')
    print(f"Variable {var} converted to categorical")

# For each categorical ver perform Chi2 test
# Print results, create bar plot
for var in non_numerical:
  cat_df[var] = cat_df[var].astype('category')
  contingency_table = pd.crosstab(cat_df[var], cat_df["Diabetes_012"])
  chi2, p, _, _ = chi2_contingency(contingency_table)
  print(f"Chi-Squared Test for {var} and {target_column}")
  print(f"Chi2 value = {chi2}, p-value = {p}\n")
  # Add the results to the DataFrame
  results_df_cat = results_df_cat.append({'Feature': var, 'Chi2': chi2,␣
 ↪'P-Value': p}, ignore_index=True)

display(results_df_cat)
```

```
Variable Diabetes_012 converted to categorical
Variable Age converted to categorical
Variable HighBP converted to categorical
Variable HighChol converted to categorical
Variable CholCheck converted to categorical
Variable Smoker converted to categorical
Variable Stroke converted to categorical
Variable HeartDiseaseorAttack converted to categorical
Variable PhysActivity converted to categorical
Variable Fruits converted to categorical
Variable Veggies converted to categorical
Variable HvyAlcoholConsump converted to categorical
Variable AnyHealthcare converted to categorical
Variable NoDocbcCost converted to categorical
Variable DiffWalk converted to categorical
Variable Sex converted to categorical
Variable GenHlth converted to categorical
Variable Education converted to categorical
Variable Income converted to categorical
```

```
Chi-Squared Test for Diabetes_012 and Diabetes_012
Chi2 value = 507360.0, p-value = 0.0

Chi-Squared Test for Age and Diabetes_012
Chi2 value = 9641.376530679845, p-value = 0.0

Chi-Squared Test for HighBP and Diabetes_012
Chi2 value = 18794.644052016425, p-value = 0.0

Chi-Squared Test for HighChol and Diabetes_012
Chi2 value = 11258.920399414841, p-value = 0.0

Chi-Squared Test for CholCheck and Diabetes_012
Chi2 value = 1173.749357770035, p-value = 1.3291236675197173e-255

Chi-Squared Test for Smoker and Diabetes_012
Chi2 value = 1010.5117511111928, p-value = 3.7167324294119075e-220

Chi-Squared Test for Stroke and Diabetes_012
Chi2 value = 2916.75197962113, p-value = 0.0

Chi-Squared Test for HeartDiseaseorAttack and Diabetes_012
Chi2 value = 8244.88910662167, p-value = 0.0

Chi-Squared Test for PhysActivity and Diabetes_012
Chi2 value = 3789.3014625427313, p-value = 0.0

Chi-Squared Test for Fruits and Diabetes_012
Chi2 value = 454.3470587241542, p-value = 2.1867028126650155e-99

Chi-Squared Test for Veggies and Diabetes_012
Chi2 value = 893.8419053866104, p-value = 8.029645985781328e-195

Chi-Squared Test for HvyAlcoholConsump and Diabetes_012
Chi2 value = 850.3240478355594, p-value = 2.2619296719502035e-185

Chi-Squared Test for AnyHealthcare and Diabetes_012
Chi2 value = 69.07797672213422, p-value = 9.997880563068128e-16

Chi-Squared Test for NoDocbcCost and Diabetes_012
Chi2 value = 396.08182159008913, p-value = 9.815789822340756e-87

Chi-Squared Test for DiffWalk and Diabetes_012
Chi2 value = 12776.94188915485, p-value = 0.0

Chi-Squared Test for Sex and Diabetes_012
Chi2 value = 250.85057509520166, p-value = 3.376678611575899e-55
```

```
Chi-Squared Test for GenHlth and Diabetes_012
Chi2 value = 24248.10614736849, p-value = 0.0

Chi-Squared Test for Education and Diabetes_012
Chi2 value = 4560.6402794568585, p-value = 0.0

Chi-Squared Test for Income and Diabetes_012
Chi2 value = 7816.462905911266, p-value = 0.0
```

	Feature	Chi2	P-Value
0	Diabetes_012	507360.000000	0.000000e+00
1	Age	9641.376531	0.000000e+00
2	HighBP	18794.644052	0.000000e+00
3	HighChol	11258.920399	0.000000e+00
4	CholCheck	1173.749358	1.329124e-255
5	Smoker	1010.511751	3.716732e-220
6	Stroke	2916.751980	0.000000e+00
7	HeartDiseaseorAttack	8244.889107	0.000000e+00
8	PhysActivity	3789.301463	0.000000e+00
9	Fruits	454.347059	2.186703e-99
10	Veggies	893.841905	8.029646e-195
11	HvyAlcoholConsump	850.324048	2.261930e-185
12	AnyHealthcare	69.077977	9.997881e-16
13	NoDocbcCost	396.081822	9.815790e-87
14	DiffWalk	12776.941889	0.000000e+00
15	Sex	250.850575	3.376679e-55
16	GenHlth	24248.106147	0.000000e+00
17	Education	4560.640279	0.000000e+00
18	Income	7816.462906	0.000000e+00

[ ]:
```python
# List all the values with p-value less than 0.05
significant_cat = results_df_cat[results_df_cat['P-Value'] < 0.05]
display(significant_cat)
```

	Feature	Chi2	P-Value
0	Diabetes_012	507360.000000	0.000000e+00
1	Age	9641.376531	0.000000e+00
2	HighBP	18794.644052	0.000000e+00
3	HighChol	11258.920399	0.000000e+00
4	CholCheck	1173.749358	1.329124e-255
5	Smoker	1010.511751	3.716732e-220
6	Stroke	2916.751980	0.000000e+00
7	HeartDiseaseorAttack	8244.889107	0.000000e+00
8	PhysActivity	3789.301463	0.000000e+00
9	Fruits	454.347059	2.186703e-99
10	Veggies	893.841905	8.029646e-195
11	HvyAlcoholConsump	850.324048	2.261930e-185
12	AnyHealthcare	69.077977	9.997881e-16

```
13          NoDocbcCost     396.081822   9.815790e-87
14             DiffWalk   12776.941889   0.000000e+00
15                  Sex     250.850575   3.376679e-55
16              GenHlth   24248.106147   0.000000e+00
17            Education    4560.640279   0.000000e+00
18               Income    7816.462906   0.000000e+00
```

0.5.2 T-Test

[ ]:
```
results_df_nums = pd.DataFrame(columns=['Feature', 'Statistic', 'P-Value'])

for var in numerical:
  # Statistical Test (e.g., t-test) for significance
  for category in df[target_column].unique():
    group1 = df[df[target_column] == category][var]
    group2 = df[df[target_column] != category][var]

  stat, p = ttest_ind(group1, group2)

  # Add the results to the DataFrame
  results_df_nums = results_df_nums.append({'Feature': var, 'Statistic': stat,␣
 ↪'P-Value': p}, ignore_index=True)

display(results_df_nums)
```

```
    Feature  Statistic        P-Value
0       BMI  24.368855  5.183573e-131
1  MentHlth  12.466688   1.162108e-35
2  PhysHlth  16.602108   7.255186e-62
```

[ ]:
```
# List all the values with p-value less than 0.05
significant_nums = results_df_nums[results_df_nums['P-Value'] < 0.05]
display(significant_nums)
```

```
    Feature  Statistic        P-Value
0       BMI  24.368855  5.183573e-131
1  MentHlth  12.466688   1.162108e-35
2  PhysHlth  16.602108   7.255186e-62
```

[ ]:
```
# compare length of significant variables
print(f"Number of total categorical variables: {len(non_numerical)}")
print(f"Number of significant categorical variables: {len(significant_cat)}")
print(f"Number of total numerical variables: {len(numerical)}")
print(f"Number of significant numerical variables: {len(significant_nums)}")
```

```
Number of total categorical variables: 19
Number of significant categorical variables: 19
Number of total numerical variables: 3
Number of significant numerical variables: 3
```

Conculsion for the significance tests

0.6 Modeling

0.6.1 Helper Functions

[ ]:
```
def train_and_evaluate_svm(X_train, X_test, y_train, y_test, random_state=42):
    svm_classifier = SVC(random_state=random_state)
    svm_classifier.fit(X_train, y_train)
    y_pred = svm_classifier.predict(X_test)

    # return accuracy_score, confusion_matrix, classification_report
    report = classification_report(y_test, y_pred, output_dict=True)
    return accuracy_score(y_test, y_pred), pd.DataFrame(report).transpose(),␣
 ↪confusion_matrix(y_test, y_pred)
```

[ ]:
```
def train_and_evaluate_nb(X_train, X_test, y_train, y_test):
    # Training a Multinomial Naive Bayes classifier
    # normalize the data not to have negative values
    X_train = X_train - X_train.min()
    X_test = X_test - X_test.min()

    # Create and train the classifier
    nb_classifier = MultinomialNB()
    nb_classifier.fit(X_train, y_train)

    # Make predictions and evaluate
    y_pred = nb_classifier.predict(X_test)

    # return accuracy_score, confusion_matrix, classification_report
    report = classification_report(y_test, y_pred, output_dict=True)
    return accuracy_score(y_test, y_pred), pd.DataFrame(report).transpose(),␣
 ↪confusion_matrix(y_test, y_pred)
```

[ ]:
```
def train_and_evaluate_knn(X_train, X_test, y_train, y_test, n_neighbors=5):
    knn_classifier = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)

    # return accuracy_score, confusion_matrix, classification_report
    report = classification_report(y_test, y_pred, output_dict=True)
    return accuracy_score(y_test, y_pred), pd.DataFrame(report).transpose(),␣
 ↪confusion_matrix(y_test, y_pred)
```

[ ]:
```
def plot_confusion_matrix(conf_matrix, title):
    plt.figure(figsize=(10, 7))
```

```
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=['No Diabetes', 'Prediabetes', 'Diabetes'], yticklabels=['No␣
 ↪Diabetes', 'Prediabetes', 'Diabetes'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()
```

```
[ ]: def display_results(accuracy, report, conf_matrix, title):
         print(title)
         print(f"Accuracy: {accuracy}")
         print("\n")
         print(report)
         print("\n")
         plot_confusion_matrix(conf_matrix, title)
```

0.6.2 Random Forest

Hyperparameter Tuning for RandomForest

```
[ ]: # Define a reduced parameter grid for GridSearchCV
     param_grid_rf_reduced = {
         'n_estimators': [100, 150],
         'max_depth': [10, 20],
         'min_samples_split': [2, 5],
         'min_samples_leaf': [1, 2],
         'bootstrap': [True]
     }

     # Initialize the RandomForestClassifier
     rf_clf_reduced = RandomForestClassifier(random_state=42)

     # Perform GridSearchCV to find the best parameters for RandomForest using SMOTE␣
      ↪balanced data
     grid_search_rf_reduced = GridSearchCV(estimator=rf_clf_reduced,␣
      ↪param_grid=param_grid_rf_reduced,
                                           cv=3, n_jobs=-1, verbose=2)

     # Fit the grid search to the SMOTE balanced data
     grid_search_rf_reduced.fit(X_ovsampled, y_ovsampled)

     # Get the best parameters and the best model
     best_params_rf_reduced_smote = grid_search_rf_reduced.best_params_
     best_rf_clf_reduced_smote = grid_search_rf_reduced.best_estimator_

     print("Best parameters for RandomForest (SMOTE) found: ",␣
      ↪best_params_rf_reduced_smote)
```

```
# Perform GridSearchCV to find the best parameters for RandomForest using
↪SMOTEENN balanced data
grid_search_rf_reduced.fit(X_ovsampled_enn, y_ovsampled_enn)

# Get the best parameters and the best model
best_params_rf_reduced_enn = grid_search_rf_reduced.best_params_
best_rf_clf_reduced_enn = grid_search_rf_reduced.best_estimator_

print("Best parameters for RandomForest (SMOTEENN) found: ",
↪best_params_rf_reduced_enn)

# Fit the grid search to the undersampled data
grid_search_rf_reduced.fit(X_undersampled, y_undersampled)

# Get the best parameters and the best model
best_params_rf_reduced_undersampled = grid_search_rf_reduced.best_params_
best_rf_clf_reduced_undersampled = grid_search_rf_reduced.best_estimator_

print("Best parameters for RandomForest (undersampled) found: ",
↪best_params_rf_reduced_undersampled)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  46.6s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  46.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  47.6s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  49.4s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  49.4s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  45.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  48.6s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  45.2s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
```

```
n_estimators=100; total time=  44.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=  45.0s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=  45.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=  47.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.9min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time= 1.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time= 1.1min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
```

```
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 1.1min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 1.1min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 1.2min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.5min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 1.5min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 1.3min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=  59.5s
Best parameters for RandomForest (SMOTE) found:  {'bootstrap': True,
'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
150}
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  36.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  37.3s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  37.4s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=  39.1s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  39.0s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=  55.6s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=  55.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=  59.4s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  38.1s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  46.0s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
```

```
n_estimators=100; total time=  46.8s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  47.3s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.2min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 6.5min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 6.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 6.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time= 6.1min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 6.4min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 6.3min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time= 6.4min
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=19.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=14.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=14.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=14.5min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=14.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=14.8min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=14.9min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  54.1s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=14.9min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=  54.0s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
```

```
n_estimators=150; total time= 1.5min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  55.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  57.7s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=  56.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time= 1.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.5min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time= 1.4min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=16.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=16.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=16.7min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=17.1min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=17.0min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=16.9min
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=16.6min
Best parameters for RandomForest (SMOTEENN) found:  {'bootstrap': True,
'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
150}
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   0.4s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=   0.8s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
```

```
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   0.8s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.5s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.7s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   1.0s
[CV] END bootstrap=True, max_depth=10, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   1.0s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   1.1s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   0.8s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=100; total time=   0.8s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=   0.7s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=   0.7s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=   1.1s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
n_estimators=150; total time=   1.2s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=2,
```

```
n_estimators=150; total time=   1.1s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   1.0s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=1, min_samples_split=5,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   1.0s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=2,
n_estimators=150; total time=   0.9s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=100; total time=   0.6s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   0.8s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   0.8s
[CV] END bootstrap=True, max_depth=20, min_samples_leaf=2, min_samples_split=5,
n_estimators=150; total time=   0.7s
Best parameters for RandomForest (undersampled) found:  {'bootstrap': True,
'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
150}
```

0.6.3 SVM

[ ]:
```
# get a slice of the data - 10_000 rows for SVM
num_rows = 10_000


X_train_slice = X_train.sample(n=num_rows, random_state=42)
y_train_slice = y_train.loc[X_train_slice.index]

# oversampled data

X_ovsampled_slice= X_ovsampled.sample(n=num_rows, random_state=42)
```

```
y_ovsampled_slice = y_ovsampled.loc[X_ovsampled_slice.index]

# undersampled data
X_undersampled_slice = X_undersampled.sample(n=num_rows, random_state=42)
y_undersampled_slice = y_undersampled.loc[X_undersampled_slice.index]

# SMOTEENN data
X_ovsampled_enn_slice = X_ovsampled_enn.sample(n=num_rows, random_state=42)
y_ovsampled_enn_slice = y_ovsampled_enn.loc[X_ovsampled_enn_slice.index]


# # SVM + Original Data
svm_original = train_and_evaluate_svm(X_train_slice, X_test, y_train_slice,␣
 ↪y_test)

# # SVM + Oversampled Data
svm_oversampled = train_and_evaluate_svm(X_ovsampled_slice, X_test,␣
 ↪y_ovsampled_slice, y_test)

# # SVM + Undersampled Data
svm_undersampled = train_and_evaluate_svm(X_undersampled_slice, X_test,␣
 ↪y_undersampled_slice, y_test)

# SVM + SMOTEENN Data
svm_smoteenn = train_and_evaluate_svm(X_ovsampled_enn_slice, X_test,␣
 ↪y_ovsampled_enn_slice, y_test)
```

0.6.4 Naive Bayes

[ ]:
```
 # Naive Bayes + Original Data
nb_original = train_and_evaluate_nb(X_train, X_test, y_train, y_test)

# Naive Bayes + Oversampled Data
nb_oversampled = train_and_evaluate_nb(X_ovsampled, X_test, y_ovsampled, y_test)

# Naive Bayes + Undersampled Data
nb_undersampled = train_and_evaluate_nb(X_undersampled, X_test, y_undersampled,␣
 ↪y_test)

# Naive Bayes + SMOTEENN Data
nb_smoteenn = train_and_evaluate_nb(X_ovsampled_enn, X_test, y_ovsampled_enn,␣
 ↪y_test)
```

0.6.5 KNN

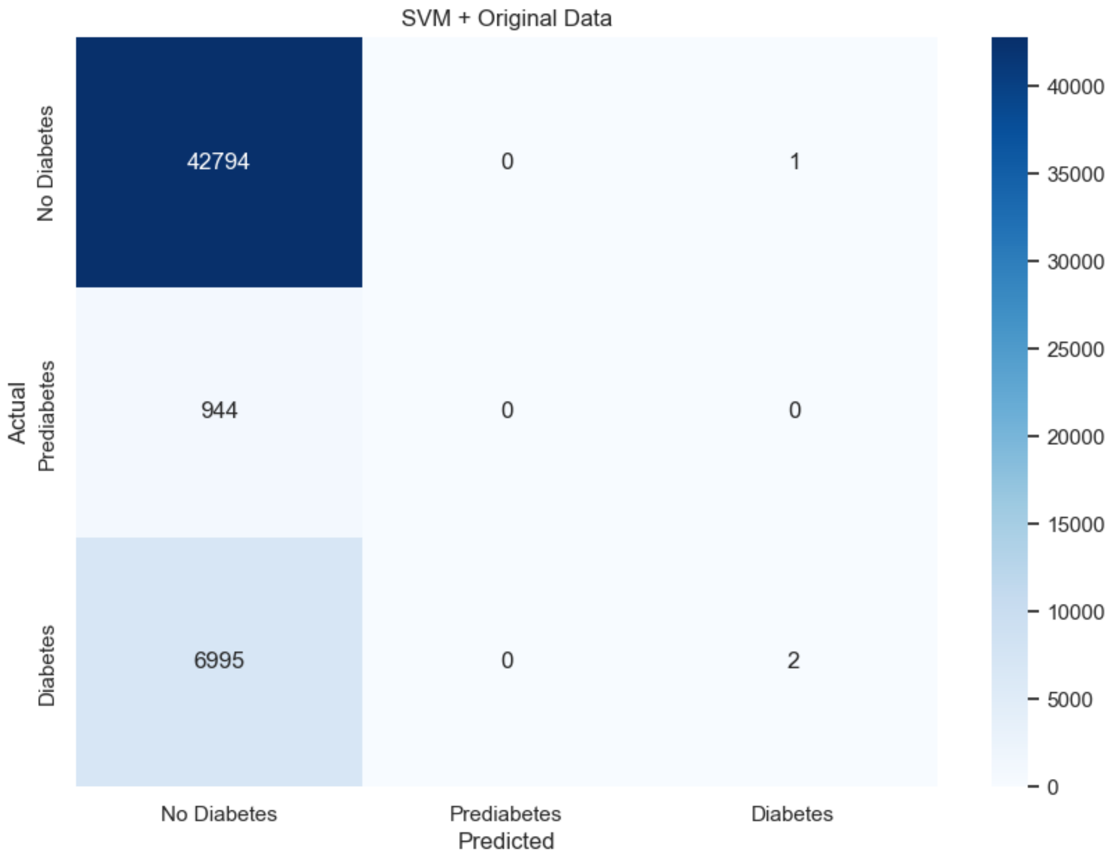
[ ]:
```
# KNN + Original Data
knn_original = train_and_evaluate_knn(X_train, X_test, y_train, y_test)

# KNN + Oversampled Data
knn_oversampled = train_and_evaluate_knn(X_ovsampled, X_test, y_ovsampled,␣
 ↪y_test)

# KNN + Undersampled Data
knn_undersampled = train_and_evaluate_knn(X_undersampled, X_test,␣
 ↪y_undersampled, y_test)

# KNN + SMOTEENN Data
knn_smoteenn = train_and_evaluate_knn(X_ovsampled_enn, X_test, y_ovsampled_enn,␣
 ↪y_test)
```

0.7 Evaluation

[ ]:
```
# Display the results
display_results(*svm_original, "SVM + Original Data")
display_results(*svm_oversampled, "SVM + Oversampled Data")
display_results(*svm_undersampled, "SVM + Undersampled Data")
display_results(*svm_smoteenn, "SVM + SMOTEENN Data")

display_results(*nb_original, "Naive Bayes + Original Data")
display_results(*nb_oversampled, "Naive Bayes + Oversampled Data")
display_results(*nb_undersampled, "Naive Bayes + Undersampled Data")
display_results(*nb_smoteenn, "Naive Bayes + SMOTEENN Data")

display_results(*knn_original, "KNN + Original Data")
display_results(*knn_oversampled, "KNN + Oversampled Data")
display_results(*knn_undersampled, "KNN + Undersampled Data")
display_results(*knn_smoteenn, "KNN + SMOTEENN Data")
```

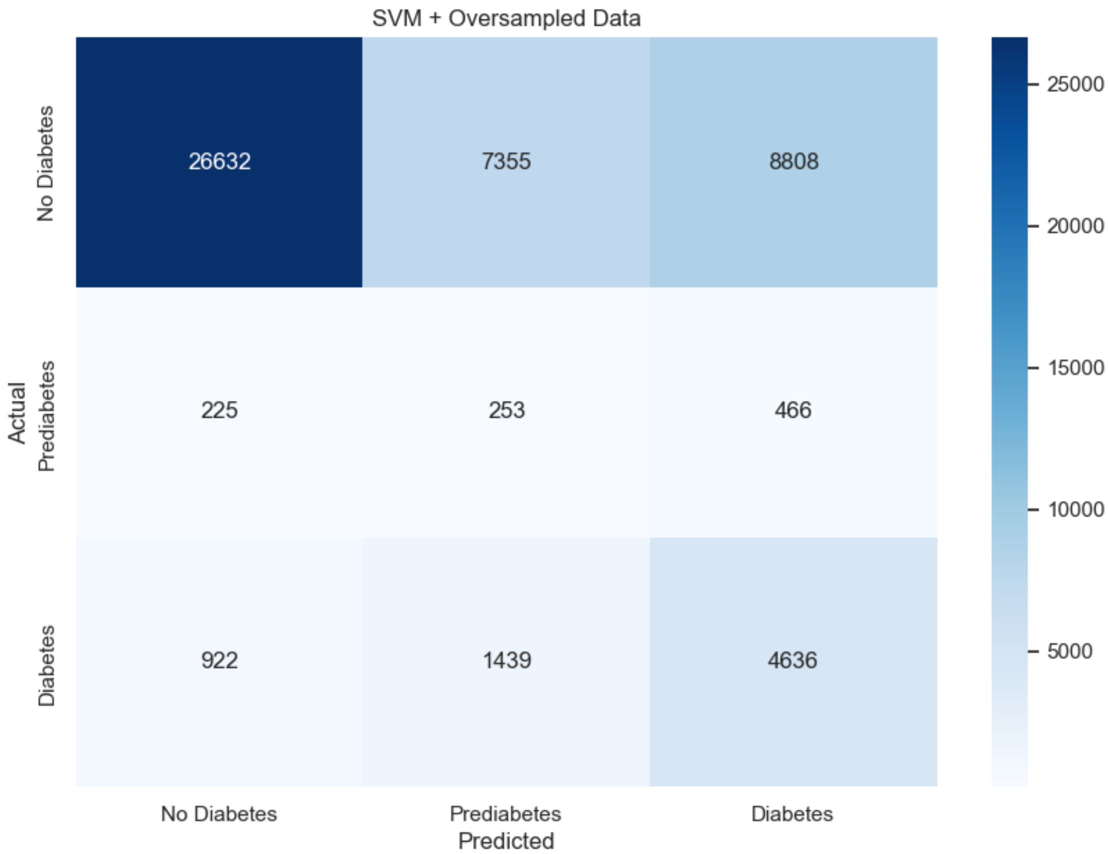
```
SVM + Original Data
Accuracy: 0.8435036266162094
```

	precision	recall	f1-score	support
0.0	0.843514	0.999977	0.915106	42795.000000
1.0	0.000000	0.000000	0.000000	944.000000
2.0	0.666667	0.000286	0.000571	6997.000000
accuracy	0.843504	0.843504	0.843504	0.843504
macro avg	0.503394	0.333421	0.305226	50736.000000
weighted avg	0.803431	0.843504	0.771956	50736.000000



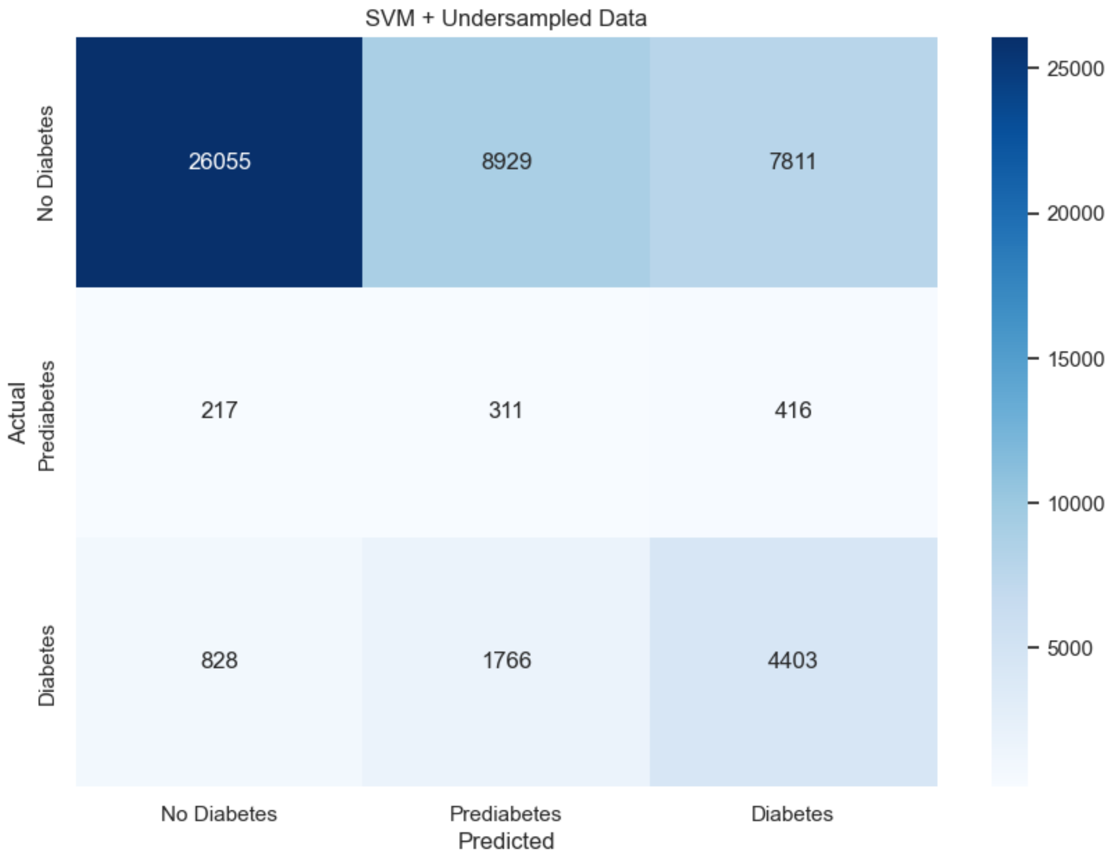
SVM + Oversampled Data
Accuracy: 0.6212748344370861

	precision	recall	f1-score	support
0.0	0.958710	0.622316	0.754726	42795.000000
1.0	0.027965	0.268008	0.050646	944.000000
2.0	0.333285	0.662570	0.443488	6997.000000
accuracy	0.621275	0.621275	0.621275	0.621275
macro avg	0.439987	0.517631	0.416286	50736.000000
weighted avg	0.855140	0.621275	0.698703	50736.000000



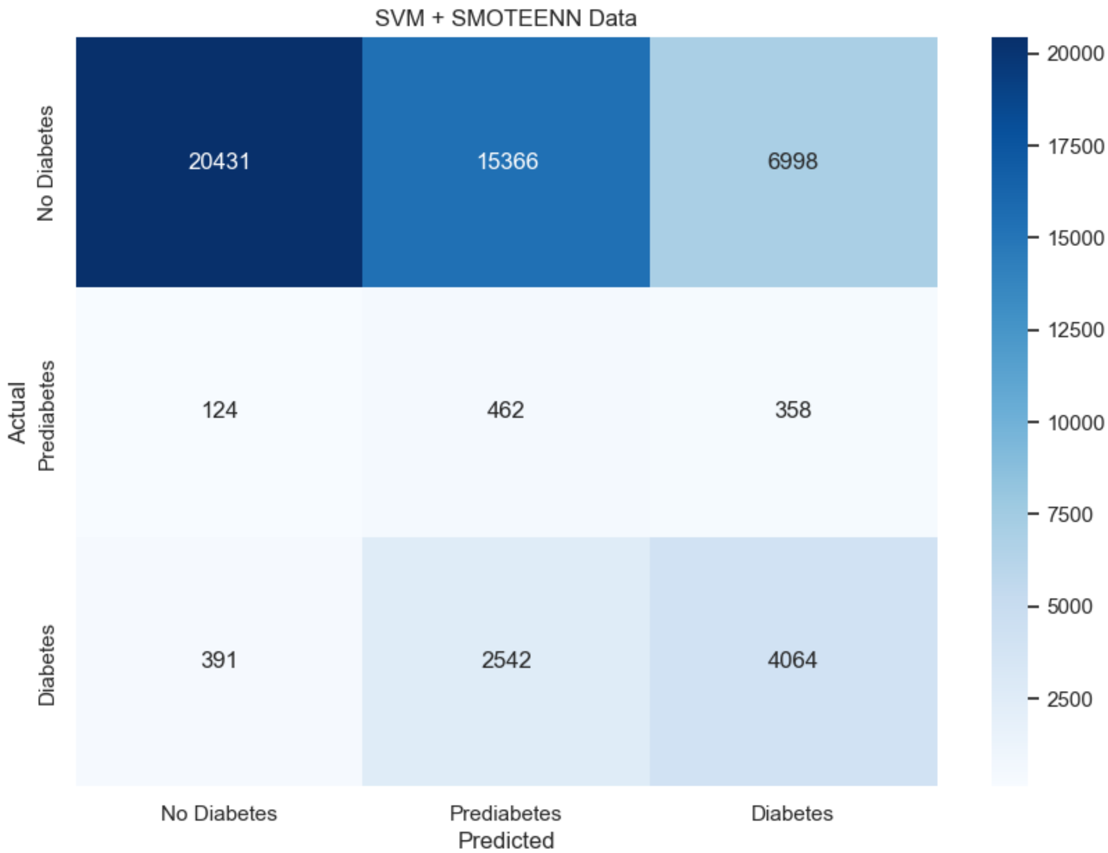

```
SVM + Undersampled Data
Accuracy: 0.6064530116682435
```

	precision	recall	f1-score	support
0.0	0.961439	0.608833	0.745547	42795.000000
1.0	0.028257	0.329449	0.052050	944.000000
2.0	0.348614	0.629270	0.448668	6997.000000
accuracy	0.606453	0.606453	0.606453	0.606453
macro avg	0.446104	0.522517	0.415422	50736.000000
weighted avg	0.859562	0.606453	0.691701	50736.000000



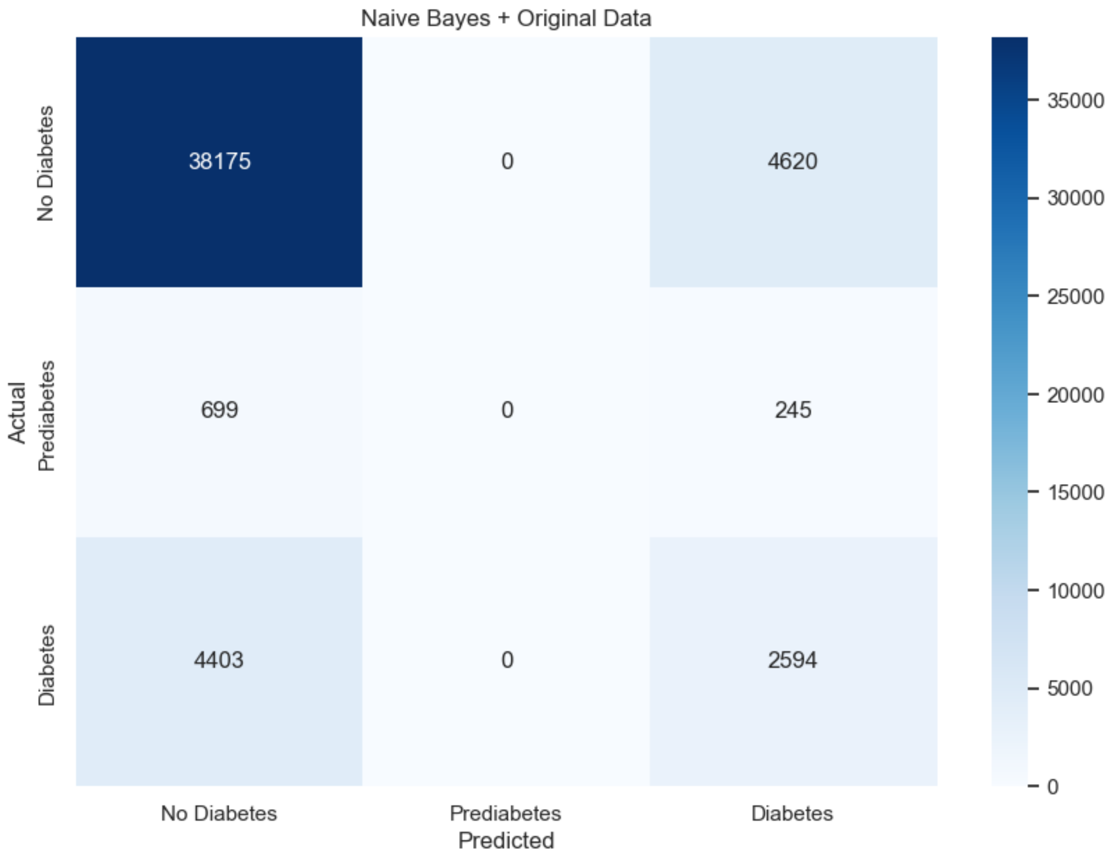
SVM + SMOTEENN Data
Accuracy: 0.491899243140965

	precision	recall	f1-score	support
0.0	0.975413	0.477416	0.641063	42795.000000
1.0	0.025150	0.489407	0.047841	944.000000
2.0	0.355867	0.580820	0.441331	6997.000000
accuracy	0.491899	0.491899	0.491899	0.491899
macro avg	0.452143	0.515881	0.376745	50736.000000
weighted avg	0.872291	0.491899	0.602481	50736.000000



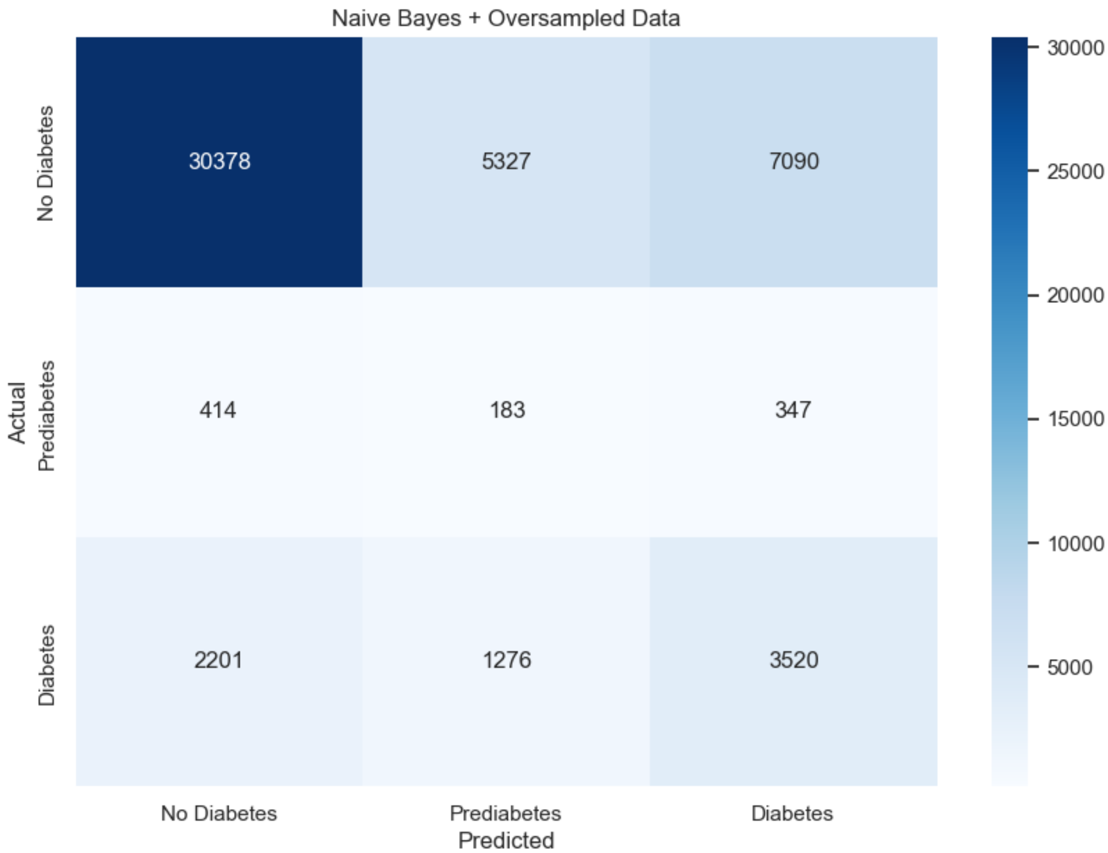
Naive Bayes + Original Data
Accuracy: 0.8035517187007253

	precision	recall	f1-score	support
0.0	0.882108	0.892043	0.887048	42795.000000
1.0	0.000000	0.000000	0.000000	944.000000
2.0	0.347768	0.370730	0.358882	6997.000000
accuracy	0.803552	0.803552	0.803552	0.803552
macro avg	0.409959	0.420925	0.415310	50736.000000
weighted avg	0.792005	0.803552	0.797704	50736.000000



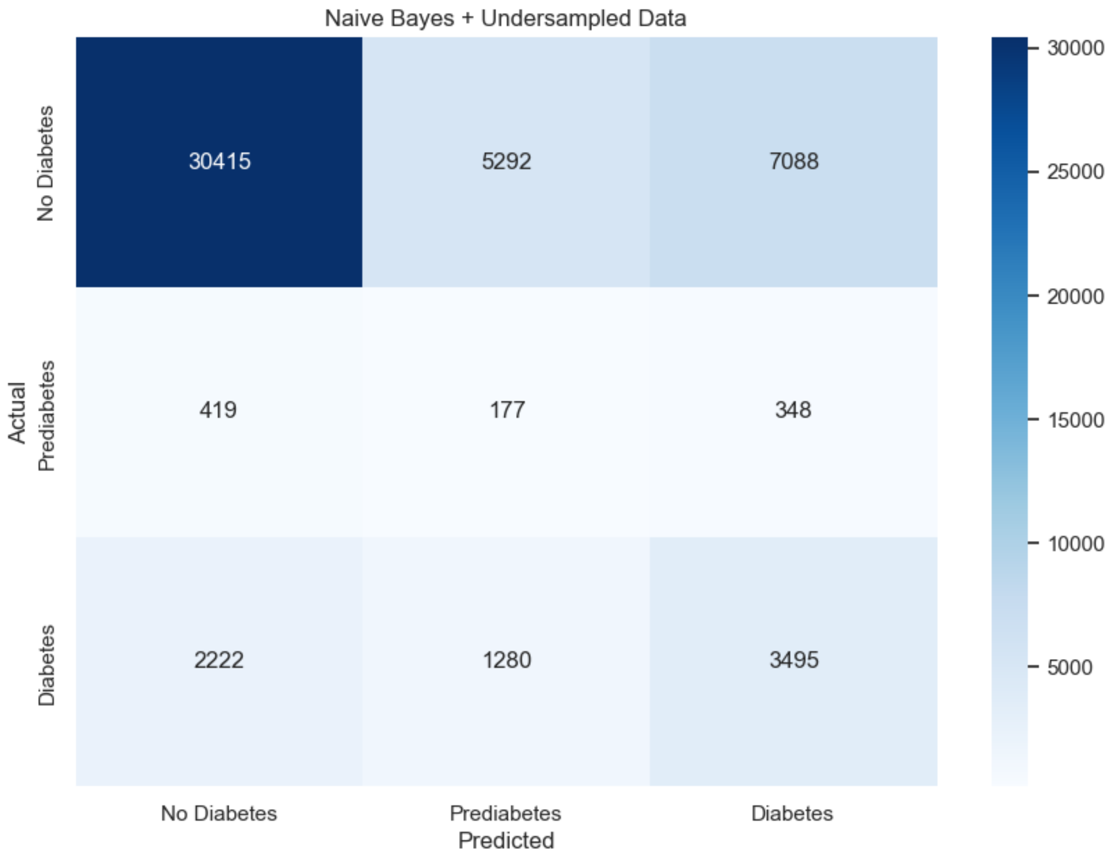

```
Naive Bayes + Oversampled Data
Accuracy: 0.6717321034374014
```

	precision	recall	f1-score	support
0.0	0.920741	0.709849	0.801657	42795.000000
1.0	0.026967	0.193856	0.047348	944.000000
2.0	0.321256	0.503073	0.392113	6997.000000
accuracy	0.671732	0.671732	0.671732	0.671732
macro avg	0.422988	0.468926	0.413706	50736.000000
weighted avg	0.821436	0.671732	0.731142	50736.000000



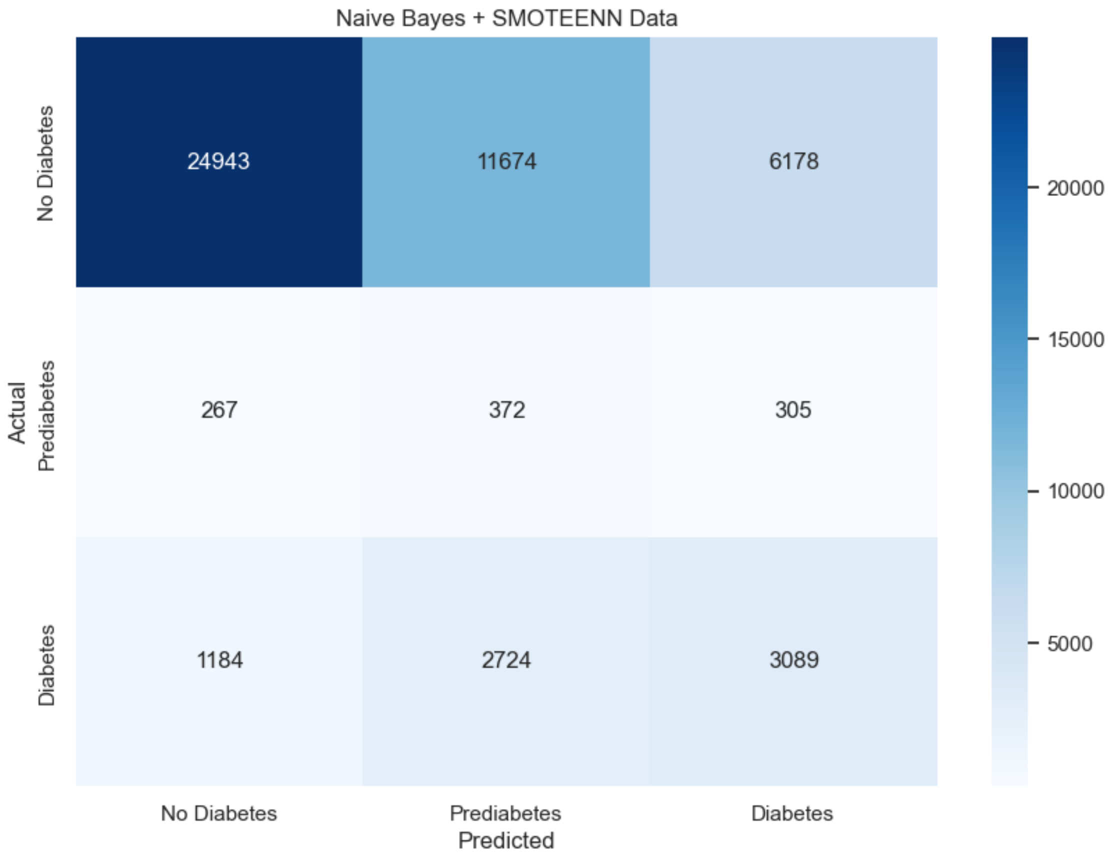
Naive Bayes + Undersampled Data
Accuracy: 0.6718503626616209

	precision	recall	f1-score	support
0.0	0.920105	0.710714	0.801967	42795.00000
1.0	0.026226	0.187500	0.046016	944.00000
2.0	0.319733	0.499500	0.389893	6997.00000
accuracy	0.671850	0.671850	0.671850	0.67185
macro avg	0.422021	0.465905	0.412625	50736.00000
weighted avg	0.820676	0.671850	0.731073	50736.00000



Naive Bayes + SMOTEENN Data
Accuracy: 0.5598391674550615

	precision	recall	f1-score	support
0.0	0.945025	0.582848	0.721011	42795.000000
1.0	0.025186	0.394068	0.047346	944.000000
2.0	0.322712	0.441475	0.372865	6997.000000
accuracy	0.559839	0.559839	0.559839	0.559839
macro avg	0.430975	0.472797	0.380407	50736.000000
weighted avg	0.842088	0.559839	0.660464	50736.000000

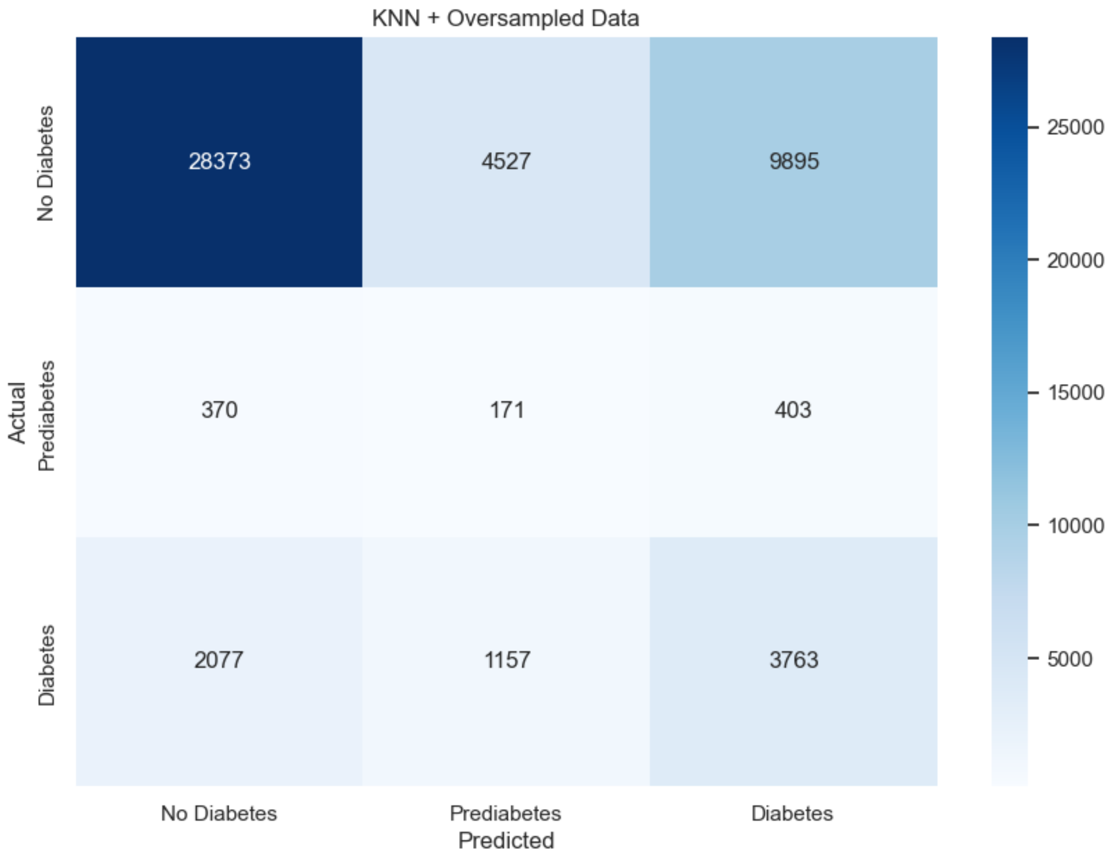


KNN + Original Data
Accuracy: 0.8331953642384106

	precision	recall	f1-score	support
0.0	0.864746	0.954808	0.907548	42795.000000
1.0	0.000000	0.000000	0.000000	944.000000
2.0	0.410943	0.201801	0.270680	6997.000000
accuracy	0.833195	0.833195	0.833195	0.833195
macro avg	0.425230	0.385536	0.392743	50736.000000
weighted avg	0.786073	0.833195	0.802832	50736.000000


```
KNN + Oversampled Data
Accuracy: 0.6367667928098392
```

	precision	recall	f1-score	support
0.0	0.920604	0.662998	0.770848	42795.000000
1.0	0.029206	0.181144	0.050302	944.000000
2.0	0.267620	0.537802	0.357394	6997.000000
accuracy	0.636767	0.636767	0.636767	0.636767
macro avg	0.405810	0.460648	0.392848	50736.000000
weighted avg	0.813965	0.636767	0.700422	50736.000000

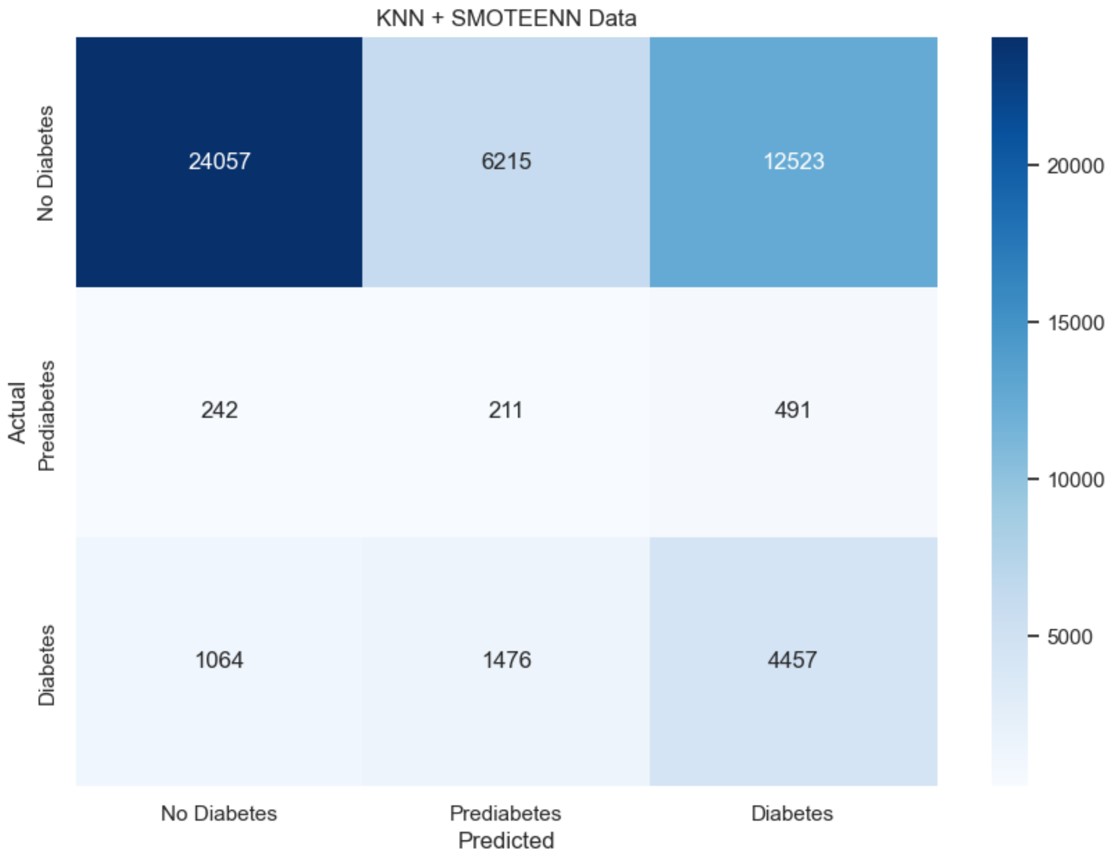


KNN + Undersampled Data
Accuracy: 0.5976624093345948

	precision	recall	f1-score	support
0.0	0.926225	0.639841	0.756848	42795.000000
1.0	0.027883	0.375000	0.051906	944.000000
2.0	0.305179	0.369730	0.334367	6997.000000
accuracy	0.597662	0.597662	0.597662	0.597662
macro avg	0.419762	0.461524	0.381040	50736.000000
weighted avg	0.823862	0.597662	0.685467	50736.000000

KNN + SMOTEENN Data
Accuracy: 0.5661660359508042

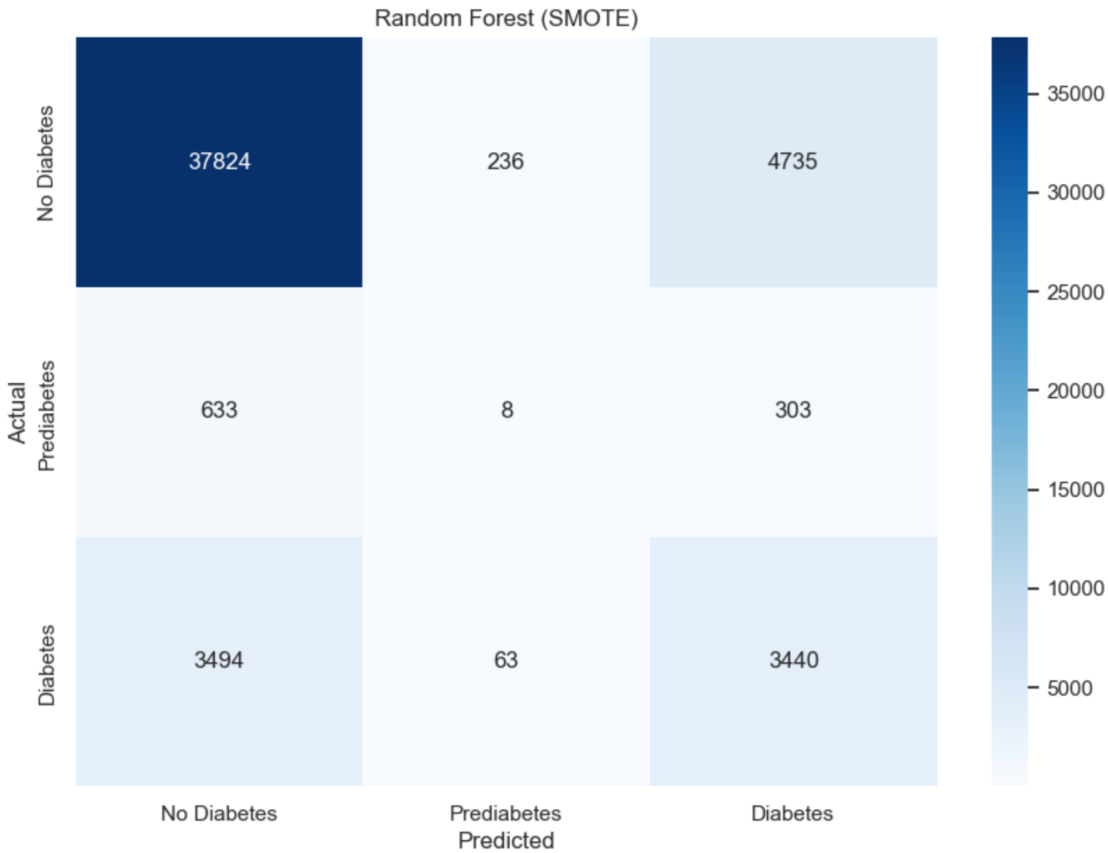
	precision	recall	f1-score	support
0.0	0.948508	0.562145	0.705919	42795.000000
1.0	0.026702	0.223517	0.047705	944.000000
2.0	0.255108	0.636987	0.364313	6997.000000
accuracy	0.566166	0.566166	0.566166	0.566166
macro avg	0.410106	0.474216	0.372645	50736.000000
weighted avg	0.835730	0.566166	0.646561	50736.000000



[ ]:
```
# Random Forest Evaluation
# Predict on the test set using the tuned RandomForest model (SMOTE)
y_pred_best_rf = best_rf_clf_reduced_smote.predict(X_test)

# Get the classification report
report_best_rf = classification_report(y_test, y_pred_best_rf, output_dict=True)
display_results(accuracy_score(y_test, y_pred_best_rf), pd.
 ↪DataFrame(report_best_rf).transpose(), confusion_matrix(y_test,␣
 ↪y_pred_best_rf), "Random Forest (SMOTE)")


# Predict on the test set using the tuned RandomForest model (SMOTEENN)
y_pred_best_rf_enn = best_rf_clf_reduced_enn.predict(X_test)

# Get the classification report
report_best_rf_enn = classification_report(y_test, y_pred_best_rf_enn,␣
 ↪output_dict=True)
```

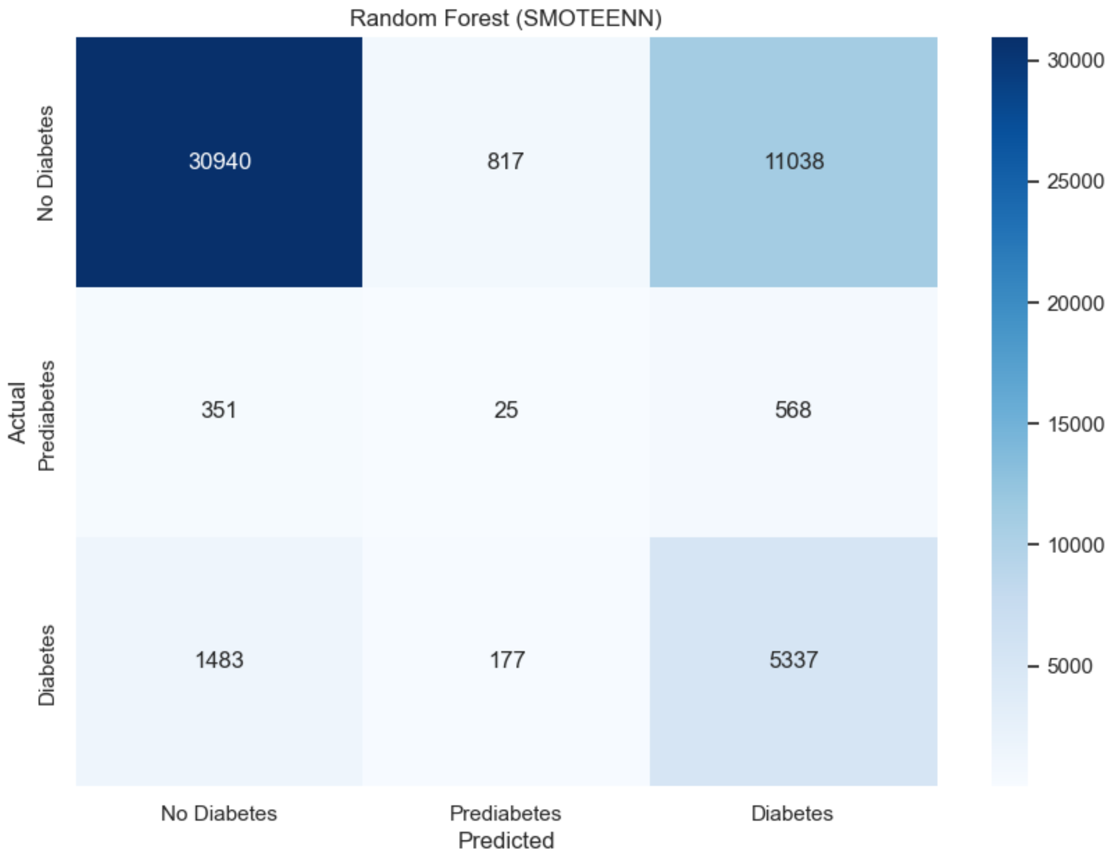
```
display_results(accuracy_score(y_test, y_pred_best_rf_enn), pd.
 ↪DataFrame(report_best_rf_enn).transpose(), confusion_matrix(y_test,␣
 ↪y_pred_best_rf_enn), "Random Forest (SMOTEENN)")


# Predict on the test set using the tuned RandomForest model (undersampled)
y_pred_best_rf_undersampled = best_rf_clf_reduced_undersampled.predict(X_test)
display_results(accuracy_score(y_test, y_pred_best_rf_undersampled), pd.
 ↪DataFrame(report_best_rf).transpose(), confusion_matrix(y_test,␣
 ↪y_pred_best_rf_undersampled), "Random Forest (undersampled)")
```

```
Random Forest (SMOTE)
Accuracy: 0.8134657836644592


              precision    recall  f1-score       support
0.0            0.901623  0.883842  0.892644  42795.000000
1.0            0.026059  0.008475  0.012790    944.000000
2.0            0.405756  0.491639  0.444588   6997.000000
accuracy       0.813466  0.813466  0.813466      0.813466
macro avg      0.444479  0.461318  0.450007  50736.000000
weighted avg   0.816947  0.813466  0.814482  50736.000000
```



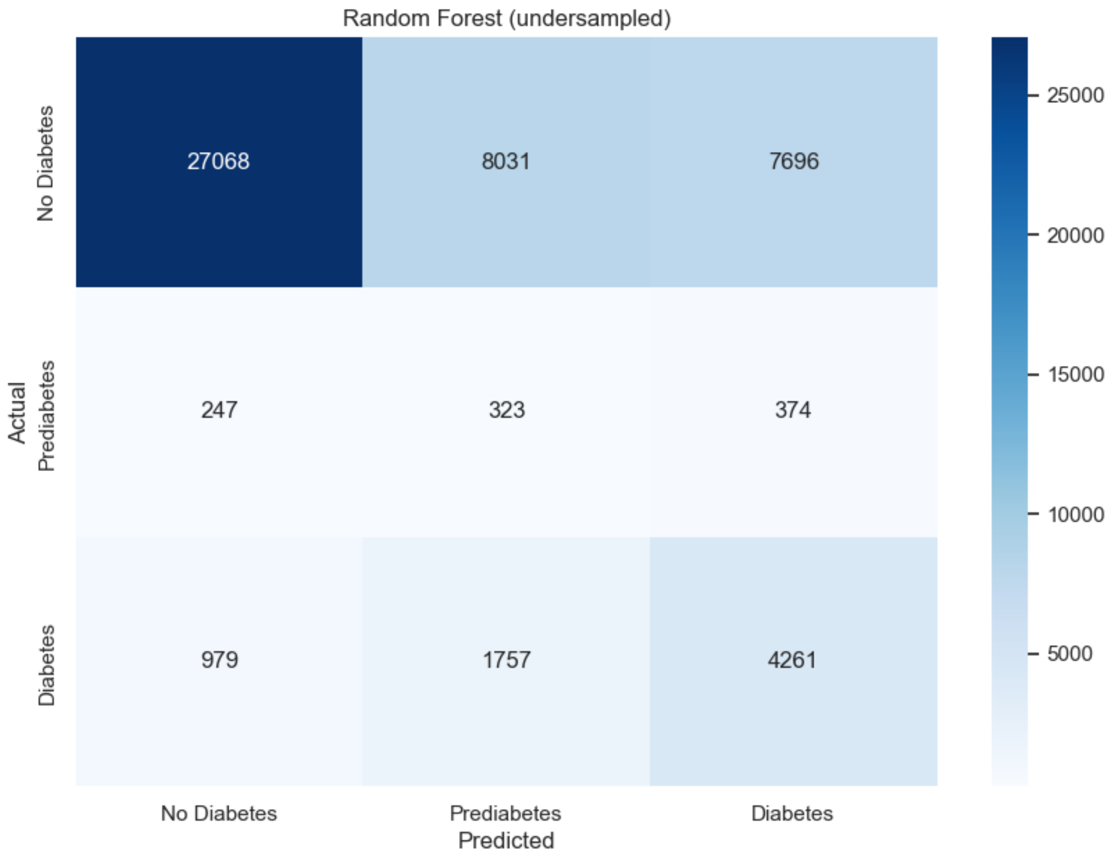

```
Random Forest (SMOTEENN)
Accuracy: 0.7155077262693157
```

	precision	recall	f1-score	support
0.0	0.944041	0.722982	0.818854	42795.000000
1.0	0.024534	0.026483	0.025471	944.000000
2.0	0.314997	0.762755	0.445865	6997.000000
accuracy	0.715508	0.715508	0.715508	0.715508
macro avg	0.427857	0.504073	0.430063	50736.000000
weighted avg	0.840181	0.715508	0.752654	50736.000000




```
Random Forest (undersampled)
Accuracy: 0.6238568274992116
```

	precision	recall	f1-score	support
0.0	0.901623	0.883842	0.892644	42795.000000
1.0	0.026059	0.008475	0.012790	944.000000
2.0	0.405756	0.491639	0.444588	6997.000000
accuracy	0.813466	0.813466	0.813466	0.813466
macro avg	0.444479	0.461318	0.450007	50736.000000
weighted avg	0.816947	0.813466	0.814482	50736.000000



0.8 Discussion and Conclusions