```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import os #used to access file
```

```
!pip install scikit-posthocs
```

```
Requirement already satisfied: scikit-posthocs in /opt/anaconda3/lib/python3.12/site-packages (0.11.4)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (1.26.4)
Requirement already satisfied: scipy>=1.9.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (1.13.
Requirement already satisfied: statsmodels in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (0.14.2
Requirement already satisfied: pandas>=0.20.0 in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (2.2
Requirement already satisfied: seaborn in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (0.13.2)
Requirement already satisfied: matplotlib in /opt/anaconda3/lib/python3.12/site-packages (from scikit-posthocs) (3.9.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.12/site-packages (from pandas>=0.20.
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.12/site-packages (from pandas>=0.20.0->scikit-
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/python3.12/site-packages (from pandas>=0.20.0->sciki
Requirement already satisfied: contourpy>=1.0.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit-
Requirement already satisfied: cycler>=0.10 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit-post
Requirement already satisfied: fonttools>=4.22.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit
Requirement already satisfied: kiwisolver>=1.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit
Requirement already satisfied: packaging>=20.0 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit-p
Requirement already satisfied: pillow>=8 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit-posthoc
Requirement already satisfied: pyparsing>=2.3.1 in /opt/anaconda3/lib/python3.12/site-packages (from matplotlib->scikit-
Requirement already satisfied: patsy>=0.5.6 in /opt/anaconda3/lib/python3.12/site-packages (from statsmodels->scikit-pos
Requirement already satisfied: six in /opt/anaconda3/lib/python3.12/site-packages (from patsy>=0.5.6->statsmodels->sciki
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("nguyenvy/nhanes-19882018")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version (installed: 0.3.10), please consider upgrading to the l
Path to dataset files: /Users/eshikajanbandhu/.cache/kagglehub/datasets/nguyenvy/nhanes-19882018/versions/9
```

```
#helen 4-23. create a new merged data set that merges in more columns from the kaggle data set

# Import Libraries
import seaborn as sns
from scipy.stats import kruskal
import pandas as pd
from scikit_posthocs import posthoc_dunn

# List All Files in the Dataset Directory
files = os.listdir(path)

# Load the Questionnaire Data (questionnaire_clean.csv contains DIQ010)
questionnaire_file = os.path.join(path, "questionnaire_clean.csv")
if os.path.exists(questionnaire_file):
    questionnaire_df = pd.read_csv(questionnaire_file)
    # Check if DIQ010 exists
    #check for required columns
    required_q_cols = ['SEQN','DIQ010','ALQ120Q','PAQ625','PAQ655','PAQ759V','RHQ131',
                       'ALQ130','BPQ020','BPQ080','HAD3','HAR1','HFC6E','HFC6E1','PAQ678',
                       'PAQ706']
    missing_cols_q = [col for col in required_q_cols if col not in questionnaire_df.columns]
    if missing_cols_q:
        raise KeyError(f"Missing required questionnaire columns: {missing_cols_q}")
    else:
        print("\nAll required questionnaire columns found in questionnaire_clean.csv")
else:
    raise FileNotFoundError("questionnaire_clean.csv not found in the dataset directory.")


# Load the Dietary Data (dietary_clean.csv contains dietary intake)
dietary_file = os.path.join(path, "dietary_clean.csv")
if os.path.exists(dietary_file):
    dietary_df = pd.read_csv(dietary_file)
    # Check for required dietary columns
    required_dietary_cols = ['SEQN','DRXTSUGR','DRXTFIBE', 'DRXTCARB', 'DRXTMFAT', 'DRXTPFAT', 'DRXTSFAT','DRXTTFAT','DRXTKC
                             'DRXTVAIU','DRXTCARO','DRXTVB1','DRXTVB2','DRXTNIAC','DRXTVB6','DRXTFOLA','DRXTVB12','DRXTVC','
                             'DRXTPHOS','DRXTMAGN','DRXTIRON','DRXTZINC','DRXTCOPP','DRXTSODI','DRXTPOTA','DRXTSELE','DRXTCA
                             'DRXTALCO','DRXTMOIS','DRXT_G_TOTAL','DRXT_V_TOTAL','DRXT_D_TOTAL','DRXT_PF_MPS_TOTAL','DRXT_A_
                             'DRXT_F_TOTAL','DRXT_ADD_SUGARS','DRXT_F_JUICE','DRX.320Z']
    missing_cols_diet = [col for col in required_dietary_cols if col not in dietary_df.columns]
    if missing_cols_diet:
        raise KeyError(f"Missing required dietary columns: {missing_cols_diet}")
```

```
    else:
        print("\nAll required dietary columns found in dietary_clean.csv")
else:
    raise FileNotFoundError("dietary_clean.csv not found in the dataset directory.")

#Load demographics file
demographics_file = os.path.join(path, "demographics_clean.csv")
if os.path.exists(demographics_file):
    demographics_df = pd.read_csv(demographics_file)
    #check for required columns
    required_dem_cols = ['SEQN','DMAETHNR','DMDEDUC2','DMDEDUC3','DMARACER','DMDMARTL','HFAGERR','RIAGENDR']
    missing_cols_dem = [col for col in required_dem_cols if col not in demographics_df.columns]
    if missing_cols_dem:
        raise KeyError(f"Missing required demographics columns: {missing_cols_dem}")
    else:
        print("\nAll required dietary columns found in demographics_clean.csv")
else:
    raise FileNotFoundError("demographics_clean.csv not found in the dataset directory.")

#Load demographics file
demographics_file = os.path.join(path, "demographics_clean.csv")
if os.path.exists(demographics_file):
    demographics_df = pd.read_csv(demographics_file)
    #check for required columns
    required_dem_cols = ['SEQN','DMAETHNR','DMDEDUC2','DMDEDUC3']
    missing_cols_dem = [col for col in required_dem_cols if col not in demographics_df.columns]
    if missing_cols_dem:
        raise KeyError(f"Missing required demographics columns: {missing_cols_dem}")
    else:
        print("\nAll required dietary columns found in demographics_clean.csv")
else:
    raise FileNotFoundError("demographics_clean.csv not found in the dataset directory.")


# Merge the Datasets on SEQN (unique identifier)
# First, merge questionnaire_df and dietary_df
merged_df = pd.merge(questionnaire_df[required_q_cols], dietary_df[required_dietary_cols], on='SEQN', how='inner')

# Then, merge the result with demographics_df
merged_df = pd.merge(merged_df, demographics_df[required_dem_cols], on='SEQN', how='inner')
```

```
All required questionnaire columns found in questionnaire_clean.csv

All required dietary columns found in dietary_clean.csv

All required dietary columns found in demographics_clean.csv

All required dietary columns found in demographics_clean.csv
```

```
merged_df.shape
merged_df.isnull().sum()
#determine if any rows have all null values
null_rows = merged_df[merged_df.isnull().all(axis=1)]
print(null_rows)
#determine if any columns have all null values
null_columns = merged_df.columns[merged_df.isnull().all()]
print(null_columns)
```

```
Empty DataFrame
Columns: [SEQN, DIQ010, ALQ120Q, PAQ625, PAQ655, PAQ759V, RHQ131, ALQ130, BPQ020, BPQ080, HAD3, HAR1, HFC6E, HFC6E1, PAQ
Index: []

[0 rows x 61 columns]
Index(['HFC6E', 'HFC6E1'], dtype='object')
```

```
#determine if any columns have all null values
null_columns = merged_df.columns[merged_df.isnull().all()]
print(null_columns)
```

```
Index(['HFC6E', 'HFC6E1'], dtype='object')
```

```
print("\nDataset Shape:", merged_df.shape)
print("\nNull value count per column:\n", merged_df.isnull().sum())
print("\nRows with all null values:\n", null_rows)
print("\nColumns with all null values:\n", null_columns)
```

```
Dataset Shape: (746800, 61)

Null value count per column:
 SEQN                 0
DIQ010          132728
ALQ120Q         551052
PAQ625          689193
PAQ655          703312
                 ...
DRXT_F_JUICE    445943
DRX.320Z         46408
DMAETHNR        464592
DMDEDUC2        330353
DMDEDUC3        493321
Length: 61, dtype: int64

Rows with all null values:
 Empty DataFrame
Columns: [SEQN, DIQ010, ALQ120Q, PAQ625, PAQ655, PAQ759V, RHQ131, ALQ130, BPQ020, BPQ080, HAD3, HAR1, HFC6E, HFC6E1, PAQ
Index: []

[0 rows x 61 columns]

Columns with all null values:
 Index(['HFC6E', 'HFC6E1'], dtype='object')
```

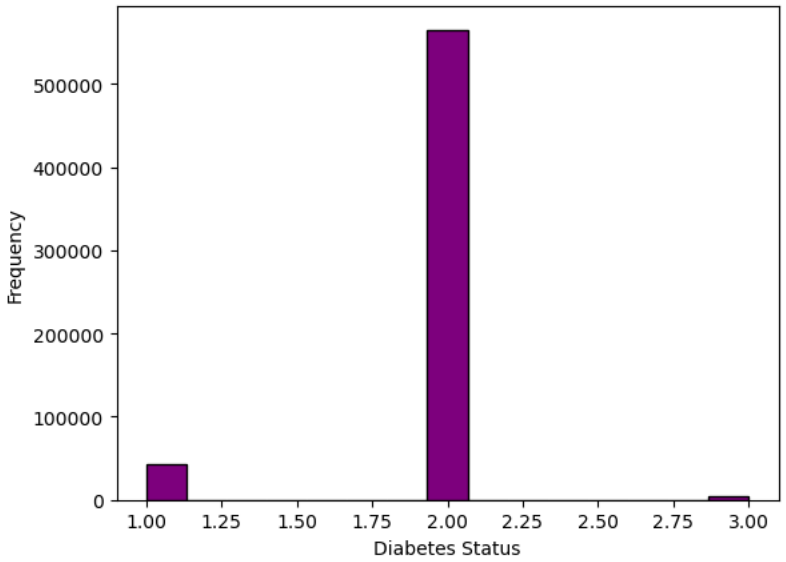
```
#data cleaning
# Filter for valid DIQ010 values (1.0 = Yes, 2.0 = No, 3.0 = Borderline)
merged_df = merged_df[merged_df['DIQ010'].isin([1.0, 2.0, 3.0])]
# Replace null values with 0 throughout merged_df
merged_df = merged_df.fillna(0)


# Remove rows with NaN in key columns
#merged_df = merged_df.dropna(subset=required_dietary_cols)
#merged_df = merged_df.dropna(subset=required_q_cols)
#merged_df = merged_df.dropna(subset=required_dem_cols)
# Remove rows where dietary values are zero or negative (invalid data)
#merged_df = merged_df[(merged_df['DRXTFIBE'] > 0) &
                     # (merged_df['DRXTCARB'] > 0) &
                     # (merged_df['DRXTMFAT'] >= 0) &
                      #(merged_df['DRXTPFAT'] >= 0) &
                      #(merged_df['DRXTSFAT'] > 0)]

  #consider adding in more cleaning



merged_df.shape
merged_df.columns = [col.replace('.','_') for col in merged_df.columns]
merged_df.columns = [col.replace(' ','_') for col in merged_df.columns]
merged_df.columns
merged_df.head()
```

	SEQN	DIQ010	ALQ120Q	PAQ625	PAQ655	PAQ759V	RHQ131	ALQ130	BPQ020	BPQ080	...	DRXT_D_TOTAL	DRXT_PF_MPS_TOTAL	DRX
0	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.744460	7.216010	
1	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	0.744460	7.216010	
2	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.327139	6.680265	
3	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.327139	6.680265	
4	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.000000	1.804000	

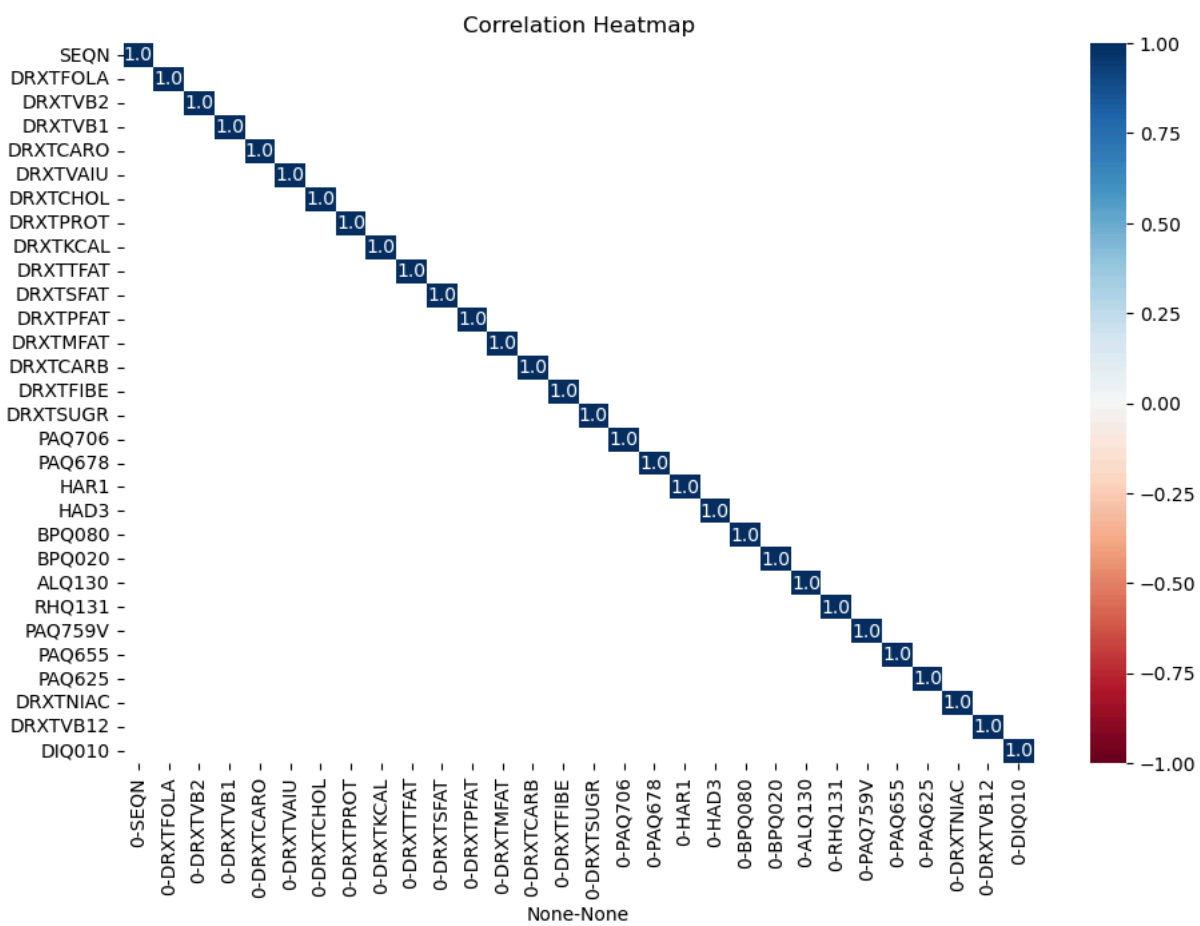
5 rows × 61 columns

```
plt.hist(merged_df['DIQ010'], bins=15, color='purple', edgecolor='black')
plt.xlabel('Diabetes Status')
plt.ylabel('Frequency')
#plt.axvline(x=merged_df['DIQ010'].median(), color='red', linestyle='dashed')
```

```
Text(0, 0.5, 'Frequency')
```

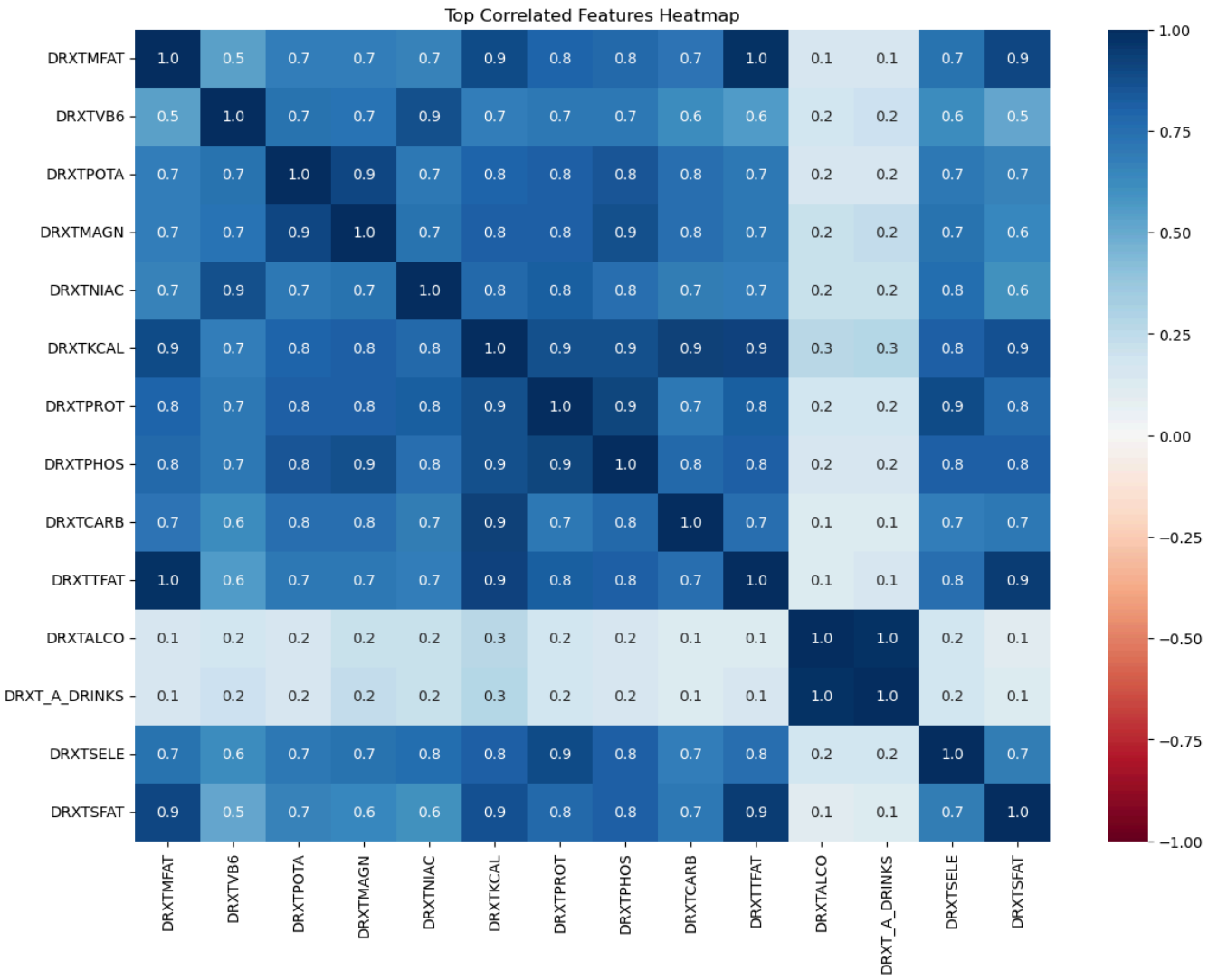



```
corr = merged_df.corr()
#determine top 30 correlations and only plot those
top_30_correlations = corr.unstack().sort_values(ascending=False).head(30)

#Convert the Series to a DataFrame to make it 2-dimensional
top_30_correlations = top_30_correlations.to_frame().unstack()


fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(top_30_correlations, annot=True, fmt=".1f", cmap="RdBu", center=0, ax=ax, vmin=-1, vmax=1)
ax.set_title("Correlation Heatmap")
plt.show()
```

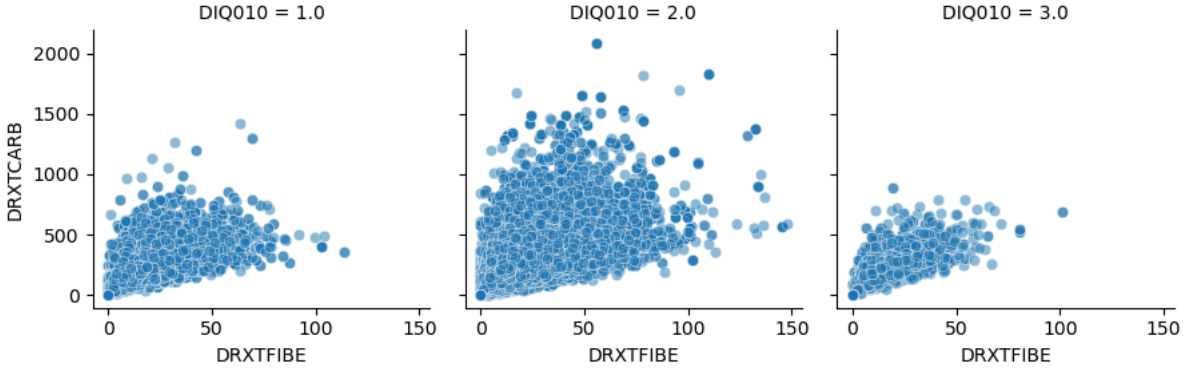



```
import seaborn as sns
import matplotlib.pyplot as plt

# Compute correlation matrix
```

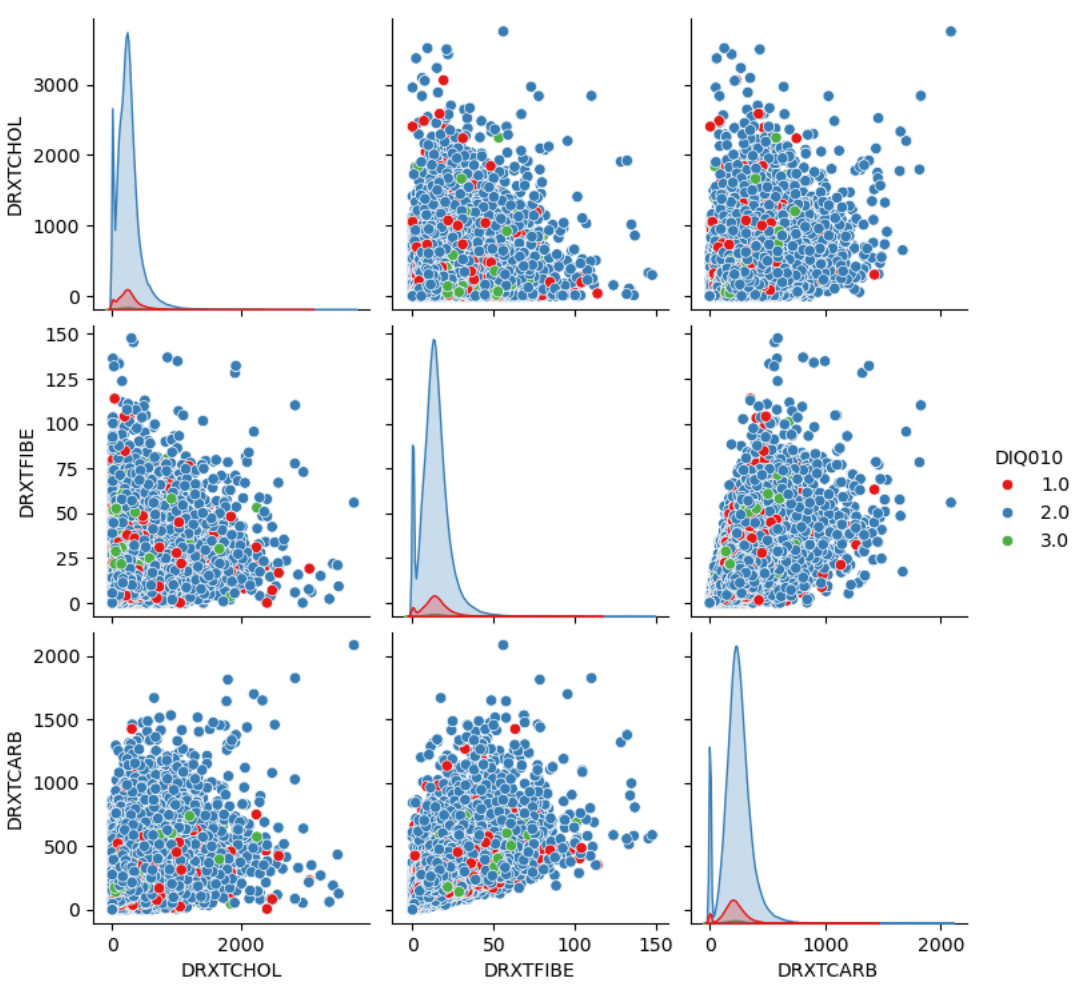
```
corr = merged_df.corr()

# Take the absolute correlations (ignore sign), remove self-correlation (diagonal)
corr_no_diag = corr.where(~np.eye(corr.shape[0], dtype=bool))
top_30_pairs = corr_no_diag.unstack().dropna().abs().sort_values(ascending=False).head(30)

# Now select unique variable names
features_to_plot = list(set([index[0] for index in top_30_pairs.index] + [index[1] for index in top_30_pairs.index]))

# Filter correlation matrix for these features
top_corr_matrix = merged_df[features_to_plot].corr()

# Plot
plt.figure(figsize=(14, 10))
sns.heatmap(top_corr_matrix, annot=True, fmt=".1f", cmap="RdBu", center=0, vmin=-1, vmax=1)
plt.title("Top Correlated Features Heatmap")
plt.show()
```




```
g = sns.FacetGrid(merged_df, col="DIQ010", col_wrap=3)   # 'col_wrap' controls the number of columns
g.map(sns.scatterplot, "DRXTFIBE","DRXTCARB", alpha=0.5)
g.add_legend()  # Add a legend if needed
plt.show()
```

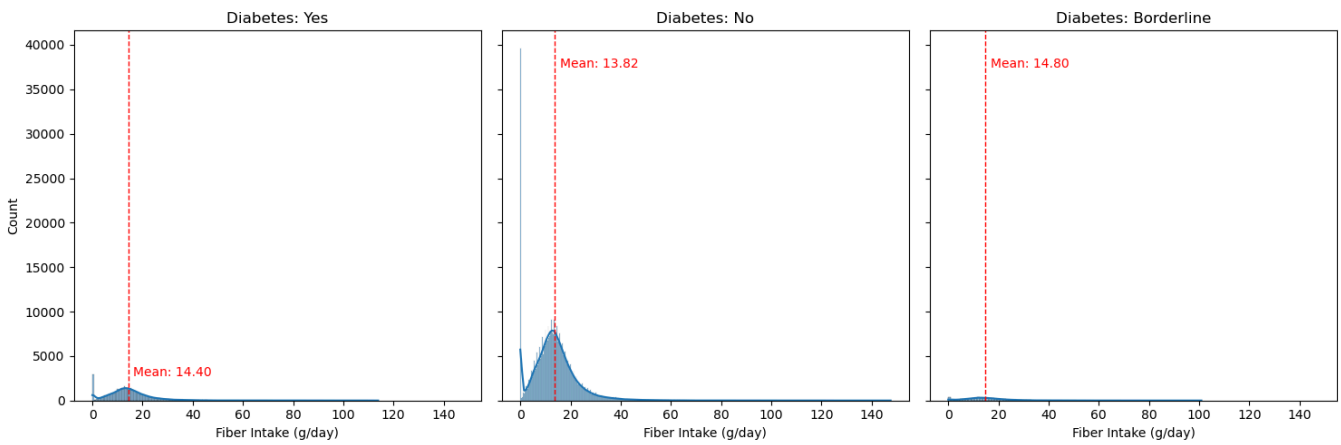



```
merged_df_new = merged_df[['DRXTCHOL','DRXTFIBE','DRXTCARB','DIQ010']]
sns.pairplot(merged_df_new, hue='DIQ010', palette='Set1')
plt.show()
```

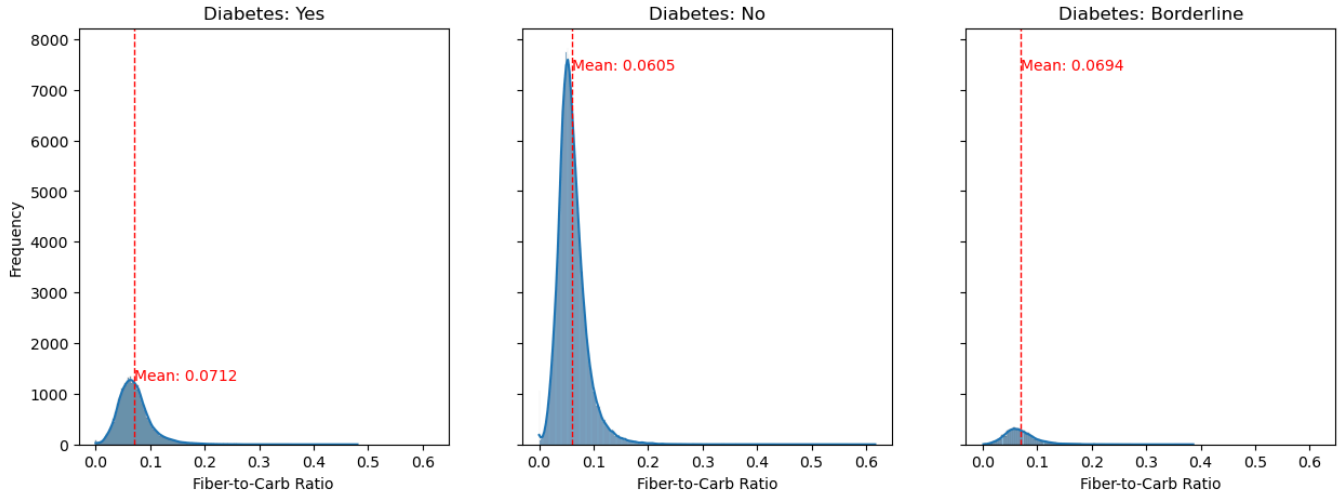



```
#helen code 4-22. creating a histogram of fiber intake for each category
#for each status of DIQ010, plot of a histogram of fiber intake

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Filter data for each DIQ010 status
diabetes_yes = merged_df[merged_df['DIQ010'] == 1.0]['DRXTFIBE']
diabetes_no = merged_df[merged_df['DIQ010'] == 2.0]['DRXTFIBE']
diabetes_borderline = merged_df[merged_df['DIQ010'] == 3.0]['DRXTFIBE']

# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True, sharex=True)  # 1 row, 3 columns

# Plot histograms for each status and show mean
for ax, data, title in zip(axes, [diabetes_yes, diabetes_no, diabetes_borderline],
                           ['Diabetes: Yes', 'Diabetes: No', 'Diabetes: Borderline']):
    sns.histplot(data, ax=ax, kde=True)
```

```
    ax.set_title(title)
    ax.set_xlabel('Fiber Intake (g/day)')

    # Calculate and display the mean
    mean_value = np.mean(data)
    ax.axvline(mean_value, color='red', linestyle='dashed', linewidth=1)
    ax.text(mean_value + 2, ax.get_ylim()[1] * 0.9, f'Mean: {mean_value:.2f}', color='red')


# Adjust layout and display
plt.tight_layout()
plt.show()
```

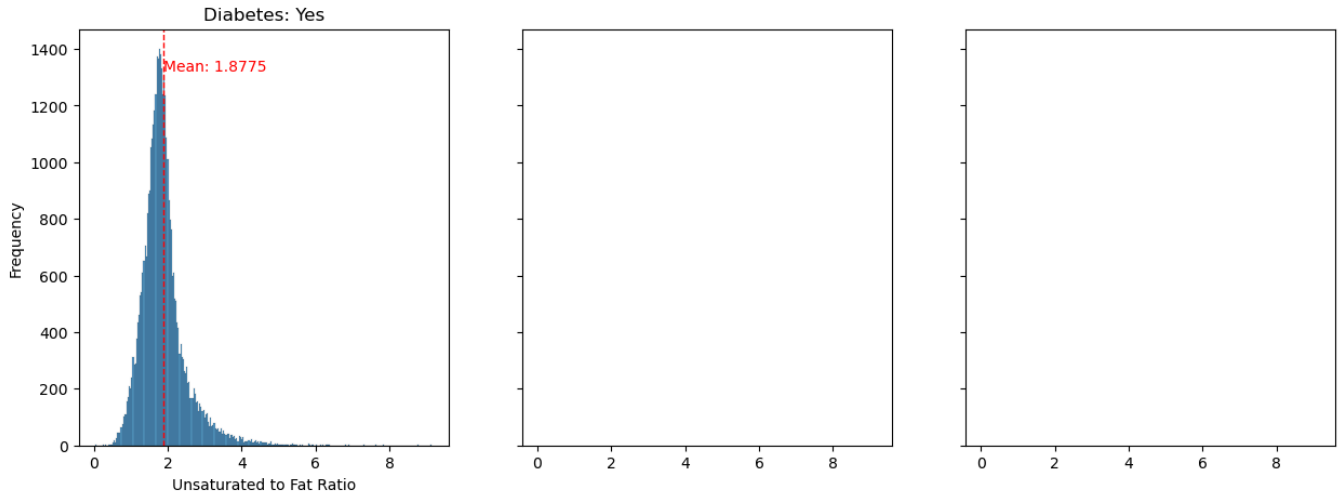



```
#helen 4-22. create histogram of fiber to carbohydrate ratio
if 'DRXTFIBE' in merged_df.columns and 'DRXTCARB' in merged_df.columns:
    # Calculate the fiber-to-carb ratio
    merged_df['Fiber_to_Carb_Ratio'] = merged_df['DRXTFIBE'] / merged_df['DRXTCARB']

    # Create subplots
    fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True, sharex=True)

    # Plot histograms for each status
    for ax, status, title in zip(axes, [1.0, 2.0, 3.0],
                         ['Diabetes: Yes', 'Diabetes: No', 'Diabetes: Borderline']):
      data = merged_df[merged_df['DIQ010'] == status]['Fiber_to_Carb_Ratio']
      sns.histplot(data, ax=ax, kde=True)
      ax.set_title(title)
      ax.set_xlabel('Fiber-to-Carb Ratio')
      ax.set_ylabel('Frequency')

      # Calculate and display the mean
      mean_value = np.mean(data)
      ax.axvline(mean_value, color='red', linestyle='dashed', linewidth=1)
      ax.text(mean_value, ax.get_ylim()[1] * 0.9, f'Mean: {mean_value:.4f}', color='red')



    # Adjust layout and display
    #plt.tight_layout()
    plt.show()


else:
    print("Cannot create Visualization 13: Missing 'DRXTFIBE' or 'DRXTCARB'.")
```




```
#helen 4-22 create histogram of unsaturated fat to saturated fat
import numpy as np

if 'DRXTMFAT' in merged_df.columns and 'DRXTSFAT' in merged_df.columns and 'DRXTPFAT' in merged_df.columns:
    # Calculate the unsaturated-to-saturated fat ratio
    merged_df['Unsaturated_to_Saturated_Fat_Ratio'] = (merged_df['DRXTMFAT'] + merged_df['DRXTPFAT']) / merged_df['DRXTSFAT']


    #create histogram of ratio for each diabetes status, and show the mean
    # Create subplots
    fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True, sharex=True)

    # Plot histograms for each status
    for ax, status, title in zip(axes, [1.0, 2.0, 3.0],
                         ['Diabetes: Yes', 'Diabetes: No', 'Diabetes: Borderline']):
      data = merged_df[merged_df['DIQ010'] == status]['Unsaturated_to_Saturated_Fat_Ratio']
      sns.histplot(data, ax=ax, kde=False)
      ax.set_title(title)
      ax.set_xlabel('Unsaturated to Fat Ratio')
      ax.set_ylabel('Frequency')

      # Calculate and display the mean
      mean_value = np.mean(data)
      ax.axvline(mean_value, color='red', linestyle='dashed', linewidth=1)
      ax.text(mean_value, ax.get_ylim()[1] * 0.9, f'Mean: {mean_value:.4f}', color='red')

      # Calculate percentiles to exclude outliers (e.g., 5th and 95th percentiles)
      #lower_bound = np.percentile(data, 5)  # 5th percentile
      upper_bound = np.percentile(data, 95)  # 95th percentile

      # Set x-axis limits for the current subplot
      ax.set_xlim(0, upper_bound)


    # Adjust layout and display
    #plt.tight_layout()
    plt.show()
else:
    print("Cannot create Visualization 14: Missing 'DRXTMFAT', 'DRXTPFAT', or 'DRXTSFAT'.")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[17], line 32
     29   upper_bound = np.percentile(data, 95)  # 95th percentile
     31   # Set x-axis limits for the current subplot
---> 32   ax.set_xlim(0, upper_bound)
     35 # Adjust layout and display
     36 #plt.tight_layout()
     37 plt.show()

File /opt/anaconda3/lib/python3.12/site-packages/matplotlib/axes/_base.py:3739, in _AxesBase.set_xlim(self, left, right, emit, auto, xmin, xmax)
   3737         raise TypeError("Cannot pass both 'right' and 'xmax'")
   3738     right = xmax
-> 3739 return self.xaxis._set_lim(left, right, emit=emit, auto=auto)

File /opt/anaconda3/lib/python3.12/site-packages/matplotlib/axis.py:1237, in Axis._set_lim(self, v0, v1, emit, auto)
   1235 self.axes._process_unit_info([(name, (v0, v1))], convert=False)
   1236 v0 = self.axes._validate_converted_limits(v0, self.convert_units)
-> 1237 v1 = self.axes._validate_converted_limits(v1, self.convert_units)
   1239 if v0 is None or v1 is None:
   1240     # Axes init calls set_xlim(0, 1) before get_xlim() can be called,
   1241     # so only grab the limits if we really need them.
   1242     old0, old1 = self.get_view_interval()

File /opt/anaconda3/lib/python3.12/site-packages/matplotlib/axes/_base.py:3660, in _AxesBase._validate_converted_limits(self, limit, convert)
   3657     converted_limit = converted_limit.squeeze()
   3658 if (isinstance(converted_limit, Real)
   3659         and not np.isfinite(converted_limit)):
-> 3660     raise ValueError("Axis limits cannot be NaN or Inf")
   3661 return converted_limit

ValueError: Axis limits cannot be NaN or Inf
```




```
import pandas as pd
import os

# Load dietary_clean.csv
dietary_file = os.path.join(path, "dietary_clean.csv")
dietary_df = pd.read_csv(dietary_file)

# Load demographics_clean.csv
demographics_file = os.path.join(path, "demographics_clean.csv")
demo_df = pd.read_csv(demographics_file)

# Load questionnaire_clean.csv (for DIQ010 - Diabetes indicator)
questionnaire_file = os.path.join(path, "questionnaire_clean.csv")
questionnaire_df = pd.read_csv(questionnaire_file)

print("✅ All datasets loaded.")
```

```
✅ All datasets loaded.
```

```
# Dietary Data: Select and Rename
dietary_df = dietary_df.rename(columns={
    'DRXTNIAC': 'Niacin',
    'DRXTSFAT': 'SatFat',
    'DRXTSELE': 'Selenium',
    'DRXTCARB': 'Carbs',
```

```
    'DRXTMAGN': 'Magnesium',
    'DRXTVB6': 'VitaminB6',
    'DRXTPOTA': 'Potassium',
    'DRXT_A_DRINKS': 'AlcoholicDrinks',
    'DRXTPHOS': 'Phosphorus',
    'DRXTTFAT': 'TotalFat',
    'DRXTPROT': 'Protein',
    'DRXTALCO': 'Alcohol',
    'DRXTMFAT': 'MonoFat',
    'DRXTKCAL': 'Calories'
})

# Keep only needed columns
dietary_df = dietary_df[['SEQN', 'Niacin', 'SatFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6', 'Potassium',
                         'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein', 'Alcohol', 'MonoFat', 'Calories']]

# Demographics: Select and Rename
demo_df = demo_df[['SEQN', 'RIDAGEYR', 'RIAGENDR']]
demo_df = demo_df.rename(columns={'RIDAGEYR': 'Age', 'RIAGENDR': 'Sex'})
demo_df['Sex'] = demo_df['Sex'].replace({1: 'Male', 2: 'Female'})
demo_df['Sex_encoded'] = demo_df['Sex'].map({'Male': 0, 'Female': 1})

# Questionnaire: Select Diabetes Label
questionnaire_df = questionnaire_df[['SEQN', 'DIQ010']]


# Merge dietary + demographics
merged_df = pd.merge(dietary_df, demo_df, on='SEQN', how='inner')

# Merge with questionnaire (for diabetes label)
merged_df = pd.merge(merged_df, questionnaire_df, on='SEQN', how='inner')

# Clean merged_df
merged_df.replace([np.inf, -np.inf], pd.NA, inplace=True)
merged_df = merged_df.dropna()

print(f"✅ Fully cleaned merged_df. Final shape: {merged_df.shape}")
```

```
✅ Fully cleaned merged_df. Final shape: (544264, 19)
```

```
# Only keep patients where DIQ010 is 1.0 (Diabetic) or 2.0 (Non-Diabetic)
merged_df = merged_df[merged_df['DIQ010'].isin([1.0, 2.0])]

# Create target column
merged_df['DiabetesRisk'] = merged_df['DIQ010'].map({1.0: 1, 2.0: 0})

print(f"✅ Diabetes filtering done. Shape: {merged_df.shape}")
```

```
✅ Diabetes filtering done. Shape: (539450, 20)
```

```
import pandas as pd
import numpy as np
import os

# Assume `path` is already defined where your NHANES CSVs are downloaded
# Example:
# path = "/path/to/your/directory/"

# Step 1: Load dietary_clean.csv
dietary_file = os.path.join(path, "dietary_clean.csv")
dietary_df = pd.read_csv(dietary_file)

# Step 2: Select relevant dietary features
diet_features = [
    'SEQN', 'DRXTNIAC', 'DRXTSFAT', 'DRXTSELE', 'DRXTCARB', 'DRXTMAGN', 'DRXTVB6',
    'DRXTPOTA', 'DRXT_A_DRINKS', 'DRXTPHOS', 'DRXTTFAT', 'DRXTPROT',
    'DRXTALCO', 'DRXTMFAT', 'DRXTKCAL'
]
dietary_df = dietary_df[diet_features].dropna()

# Step 3: Rename dietary columns
dietary_df = dietary_df.rename(columns={
    'DRXTNIAC': 'Niacin',
    'DRXTSFAT': 'SaturatedFat',
    'DRXTSELE': 'Selenium',
    'DRXTCARB': 'Carbs',
    'DRXTMAGN': 'Magnesium',
    'DRXTVB6': 'VitaminB6',
    'DRXTPOTA': 'Potassium',
    'DRXT_A_DRINKS': 'AlcoholicDrinks',
    'DRXTPHOS': 'Phosphorus',
```

```
    'DRXTTFAT': 'TotalFat',
    'DRXTPROT': 'Protein',
    'DRXTALCO': 'Alcohol',
    'DRXTMFAT': 'MonoFat',
    'DRXTKCAL': 'Calories'
})

# Step 4: Feature Engineering
dietary_df['Unsat_to_Sat_Fat_Ratio'] = (dietary_df['MonoFat']) / (dietary_df['SaturatedFat'] + 1e-6)
dietary_df['Fat_to_Calorie_Ratio'] = (dietary_df['TotalFat']) / (dietary_df['Calories'] + 1e-6)

# Step 5: Load demographics_clean.csv
demo_file = os.path.join(path, "demographics_clean.csv")
demo_df = pd.read_csv(demo_file)

demo_df = demo_df[['SEQN', 'RIDAGEYR', 'RIAGENDR']].dropna()
demo_df = demo_df.rename(columns={'RIDAGEYR': 'Age', 'RIAGENDR': 'Sex'})
demo_df['Sex'] = demo_df['Sex'].replace({1: 'Male', 2: 'Female'})
demo_df['Sex_encoded'] = demo_df['Sex'].map({'Male': 0, 'Female': 1})

# Step 6: Load questionnaire_clean.csv for Diabetes label
questionnaire_file = os.path.join(path, "questionnaire_clean.csv")
questionnaire_df = pd.read_csv(questionnaire_file)
questionnaire_df = questionnaire_df[['SEQN', 'DIQ010']].dropna()

# Step 7: Merge everything on SEQN
merged_df = pd.merge(dietary_df, demo_df, on='SEQN', how='inner')
merged_df = pd.merge(merged_df, questionnaire_df, on='SEQN', how='inner')

# Step 8: Final cleaning
merged_df = merged_df.dropna()
merged_df = merged_df.replace([np.inf, -np.inf], np.nan).dropna()

print(f"✅ Final merged_df shape: {merged_df.shape}")
print(merged_df.head())
```

```
✅ Final merged_df shape: (544264, 21)
   SEQN     Niacin  SaturatedFat    Selenium       Carbs   Magnesium  \
0   3.0  26.800000      39.30000   94.800000  371.500000  289.000000
1   3.0  26.800000      39.30000   94.800000  371.500000  289.000000
2   3.0  26.800000      39.30000   94.800000  371.500000  289.000000
3   3.0  26.800000      39.30000   94.800000  371.500000  289.000000
4   3.0  27.116752      35.32412  126.245063  336.643375  314.653095

   VitaminB6    Potassium  AlcoholicDrinks   Phosphorus  ...    Protein  \
0   2.720000  4350.000000          0.00000  1181.000000  ...  85.000000
1   2.720000  4350.000000          0.00000  1181.000000  ...  85.000000
2   2.720000  4350.000000          0.00000  1181.000000  ...  85.000000
3   2.720000  4350.000000          0.00000  1181.000000  ...  85.000000
4   2.354502  3541.321178          0.35381  1383.483857  ...  94.107539

   Alcohol    MonoFat     Calories  Unsat_to_Sat_Fat_Ratio  \
0     0.00  43.500000  2726.000000                1.106870
1     0.00  43.500000  2726.000000                1.106870
2     0.00  43.500000  2726.000000                1.106870
3     0.00  43.500000  2726.000000                1.106870
4     4.22  39.772387  2670.718424                1.125927

   Fat_to_Calorie_Ratio  Age     Sex Sex_encoded  DIQ010
0              0.039105   21    Male           0     2.0
1              0.039105   21    Male           0     2.0
2              0.039105   10  Female           1     2.0
3              0.039105   10  Female           1     2.0
4              0.038310   21    Male           0     2.0

[5 rows x 21 columns]
```

```
import statsmodels.formula.api as smf

# Step 1: Prepare model dataset
# Include only final engineered features + demographics
model_df = merged_df[[
    'DIQ010',
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein', 'Alcohol',
    'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]].dropna()

# Step 2: Keep only valid diabetes labels (1.0 = Diabetes, 2.0 = No Diabetes)
model_df = model_df[model_df['DIQ010'].isin([1.0, 2.0])]

# Step 3: Create new binary target
```

```
model_df['DiabetesRisk'] = model_df['DIQ010'].map({1.0: 1, 2.0: 0})

# Step 4: Build Logistic Regression formula
formula = (
    'DiabetesRisk ~ Niacin + SaturatedFat + Selenium + Carbs + Magnesium + VitaminB6 + '
    'Potassium + AlcoholicDrinks + Phosphorus + TotalFat + Protein + Alcohol + '
    'MonoFat + Calories + Unsat_to_Sat_Fat_Ratio + Fat_to_Calorie_Ratio + '
    'Age + Sex_encoded'
)

# Step 5: Fit the Logistic Regression model
model = smf.logit(formula=formula, data=model_df).fit()

# Step 6: Print the summary
print("📊 Logistic Regression Summary (Predicting Diabetes Risk):")
print(model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.237256
         Iterations 13
📊 Logistic Regression Summary (Predicting Diabetes Risk):
                           Logit Regression Results
==============================================================================
Dep. Variable:           DiabetesRisk   No. Observations:               539450
Model:                          Logit   Df Residuals:                   539431
Method:                           MLE   Df Model:                           18
Date:                Mon, 28 Apr 2025   Pseudo R-squ.:                 0.08405
Time:                        10:40:13   Log-Likelihood:            -1.2799e+05
converged:                       True   LL-Null:                   -1.3973e+05
Covariance Type:            nonrobust   LLR p-value:                     0.000
===============================================================================================
                              coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
Intercept                  -3.5713      0.060    -59.763      0.000      -3.688      -3.454
Niacin                     -0.0009      0.001     -0.690      0.490      -0.003       0.002
SaturatedFat               -0.0168      0.001    -13.065      0.000      -0.019      -0.014
Selenium                    0.0012      0.000      6.627      0.000       0.001       0.002
Carbs                      -0.0071      0.001    -10.254      0.000      -0.008      -0.006
Magnesium                   0.0007      0.000      6.439      0.000       0.000       0.001
VitaminB6                  -0.0026      0.012     -0.224      0.823      -0.025       0.020
Potassium                   0.0001   1.16e-05     10.222      0.000    9.56e-05       0.000
AlcoholicDrinks            -0.5028      0.072     -7.018      0.000      -0.643      -0.362
Phosphorus                  0.0002   3.04e-05      5.166      0.000    9.73e-05       0.000
TotalFat                   -0.0023      0.002     -1.246      0.213      -0.006       0.001
Protein                    -0.0040      0.001     -4.206      0.000      -0.006      -0.002
Alcohol                     0.0231      0.005      4.317      0.000       0.013       0.034
MonoFat                    -0.0041      0.002     -2.392      0.017      -0.007      -0.001
Calories                    0.0008      0.000      4.591      0.000       0.000       0.001
Unsat_to_Sat_Fat_Ratio -6.116e-05      0.000     -0.189      0.850      -0.001       0.001
Fat_to_Calorie_Ratio        8.9838      1.446      6.214      0.000       6.150      11.817
Age                         0.0277      0.000    123.778      0.000       0.027       0.028
Sex_encoded                -0.0821      0.011     -7.407      0.000      -0.104      -0.060
===============================================================================================
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

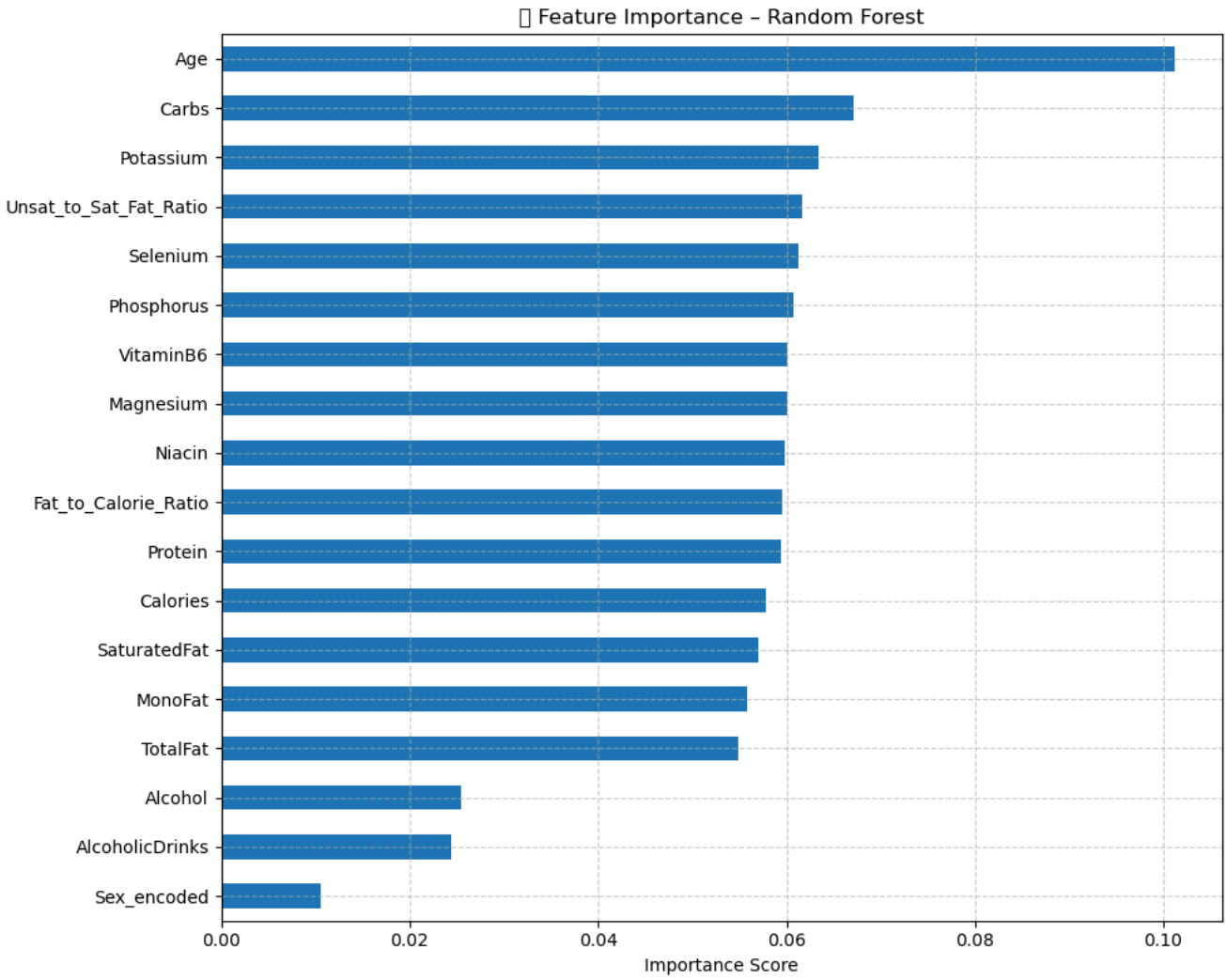
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Step 1: Prepare the dataset
features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein',
    'Alcohol', 'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]
target = 'DiabetesRisk'

# Step 2: Filter and clean model_df
model_df = merged_df[['DIQ010'] + features].dropna()
model_df = model_df[model_df['DIQ010'].isin([1.0, 2.0])]

# Step 3: Create binary target
model_df['DiabetesRisk'] = model_df['DIQ010'].map({1.0: 1, 2.0: 0})

# Step 4: Define X (features) and y (target)
X = model_df[features]
```

```
y = model_df[target]

# Step 5: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 6: Build and train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 7: Predict on test set
y_pred = rf_model.predict(X_test)

# Step 8: Evaluate the model
print("\n📋 Classification Report:")
print(classification_report(y_test, y_pred))

print("\n📊 Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
📋 Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95    100071
           1       0.11      0.05      0.07      7819

    accuracy                           0.90    107890
   macro avg       0.52      0.51      0.51    107890
weighted avg       0.87      0.90      0.89    107890


📊 Confusion Matrix:
[[97092  2979]
 [ 7447   372]]
```

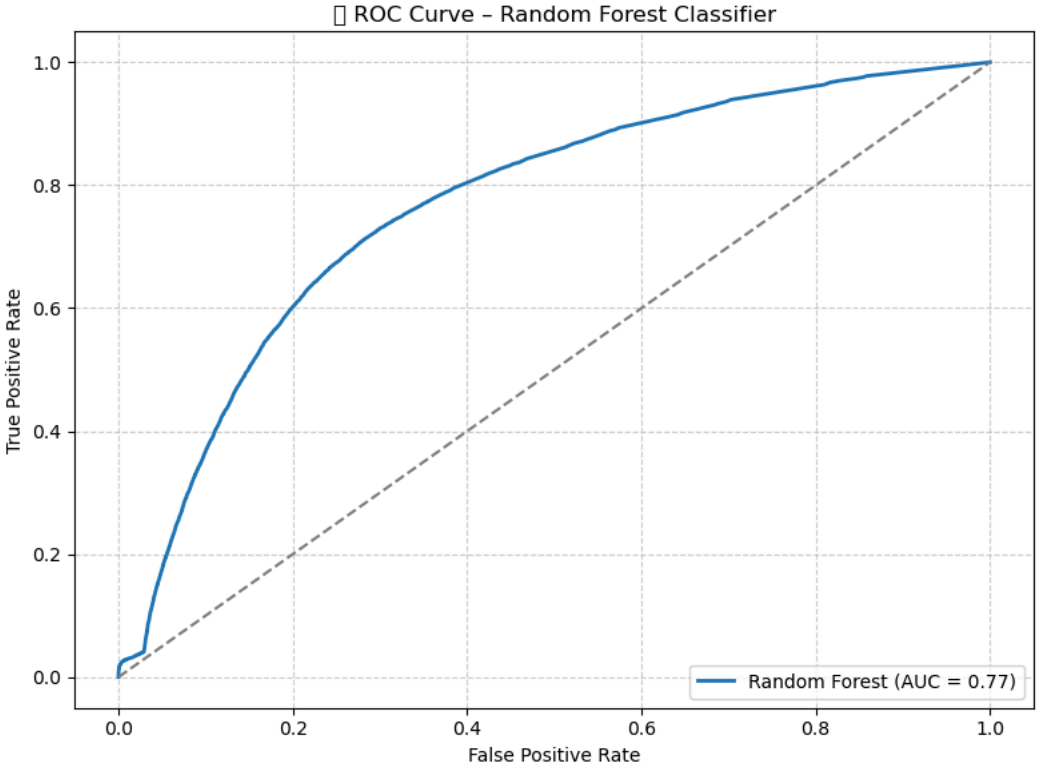
```
#This will show you which nutrient or feature is the most important in predicting diabetes risk.
importances = pd.Series(rf_model.feature_importances_, index=X_train.columns)
importances = importances.sort_values(ascending=True)  # Smallest to largest for nice horizontal plot

# Plot
plt.figure(figsize=(10, 8))
importances.plot(kind='barh')
plt.title('🌟 Feature Importance - Random Forest')
plt.xlabel('Importance Score')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
/var/folders/k7/5z9ffk3510dgxcdhs6bkjt0c0000gn/T/ipykernel_19295/3228279032.py:11: UserWarning: Glyph 127775 (\N{GLOWING
  plt.tight_layout()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 127775 (\N{GLOWING STAR})
  fig.canvas.print_figure(bytes_io, **kw)
```




```
from sklearn.metrics import roc_curve, roc_auc_score
#This will show how well the model separates diabetics from non-diabetics.
# Get prediction probabilities
y_proba = rf_model.predict_proba(X_test)[:, 1]  # Probability for class 1 (DiabetesRisk = 1)

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = roc_auc_score(y_test, y_proba)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal reference line
plt.title('📈 ROC Curve - Random Forest Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
/var/folders/k7/5z9ffk3510dgxcdhs6bkjt0c0000gn/T/ipykernel_19295/4198221392.py:19: UserWarning: Glyph 128200 (\N{CHART W
  plt.tight_layout()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 128200 (\N{CHART WITH UPW
  fig.canvas.print_figure(bytes_io, **kw)
```




```
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

# Step 1: Use the same X_train, X_test, y_train, y_test as before (already cleaned)

# Step 2: Build and train XGBoost Classifier
xgb_model = xgb.XGBClassifier(
    n_estimators=100,
    max_depth=4,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42,
    use_label_encoder=False,
    eval_metric='logloss'
)

xgb_model.fit(X_train, y_train)

# Step 3: Predict on test set
y_pred_xgb = xgb_model.predict(X_test)

# Step 4: Predict probabilities for ROC Curve
y_proba_xgb = xgb_model.predict_proba(X_test)[:, 1]

# Step 5: Evaluation
print("\n📋 Classification Report (XGBoost):")
print(classification_report(y_test, y_pred_xgb))

print("\n📊 Confusion Matrix (XGBoost):")
print(confusion_matrix(y_test, y_pred_xgb))

# Step 6: Calculate ROC AUC
auc_xgb = roc_auc_score(y_test, y_proba_xgb)
print(f"\n📈 ROC AUC (XGBoost): {auc_xgb:.4f}")
```

```
/opt/anaconda3/lib/python3.12/site-packages/xgboost/training.py:183: UserWarning: [10:44:25] WARNING: /Users/runner/work
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

📋 Classification Report (XGBoost):
              precision    recall  f1-score   support

           0       0.93      1.00      0.96    100071
           1       0.00      0.00      0.00      7819
```

```
           accuracy                           0.93    107890
          macro avg       0.46      0.50      0.48    107890
       weighted avg       0.86      0.93      0.89    107890


📊 Confusion Matrix (XGBoost):
[[100071      0]
 [  7819      0]]

📈 ROC AUC (XGBoost): 0.7510
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision i
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision i
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision i
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

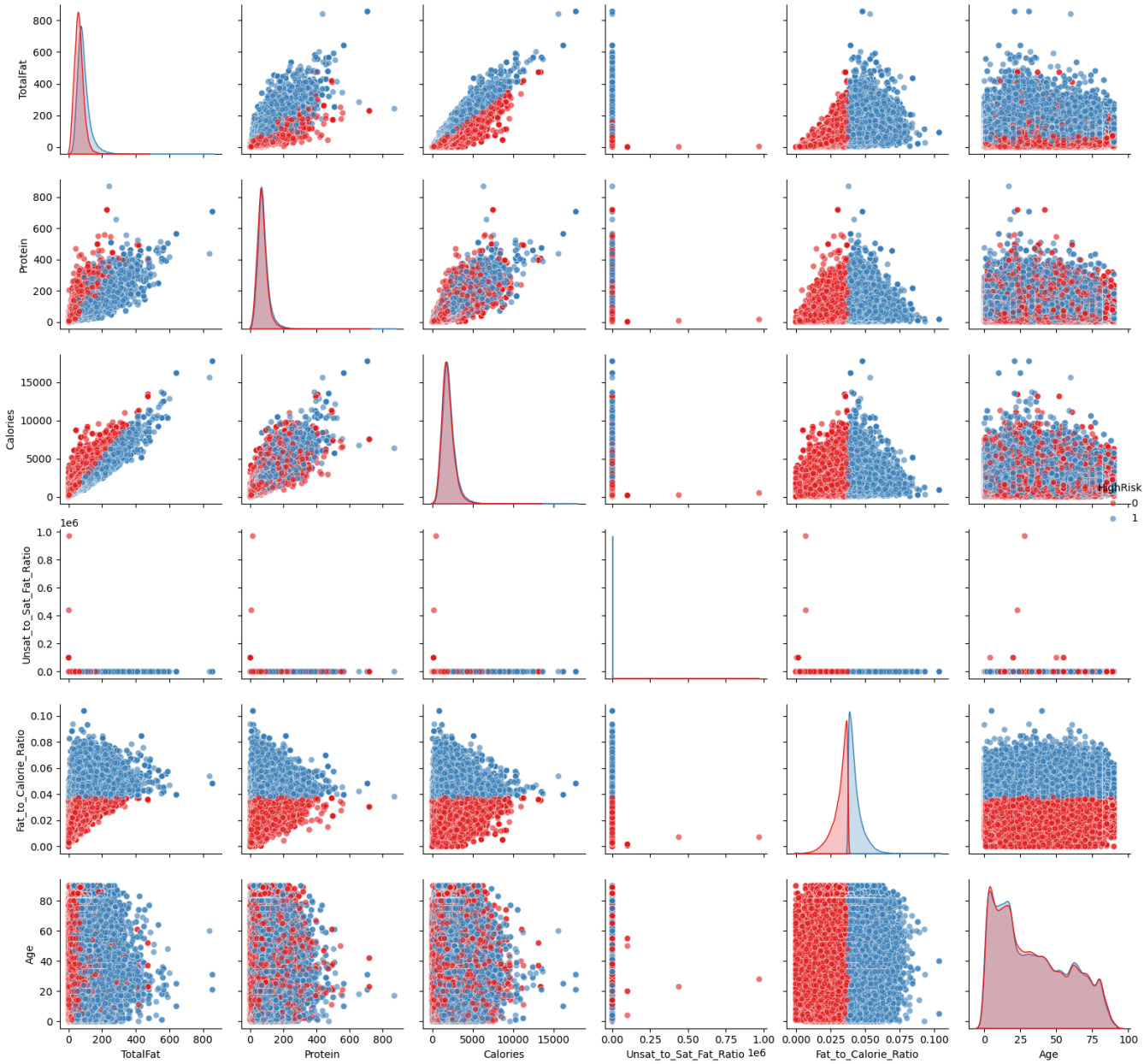
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

# Step 1: Filter for binary diabetes classification (only 1.0 = Yes, 2.0 = No)
df = merged_df[merged_df['DIQ010'].isin([1.0, 2.0])].copy()

# Step 2: Create binary target column
df['Diabetes'] = df['DIQ010'].replace({1.0: 1, 2.0: 0})

# Step 3: Define updated feature list
features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein',
    'Alcohol', 'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]

# Step 4: Drop any rows with missing data
df = df[features + ['Diabetes']].dropna()

# Step 5: Scale numeric features
scaler = StandardScaler()
df[features] = scaler.fit_transform(df[features])

# Step 6: Define feature matrix and target
X = df[features]
y = df['Diabetes']

# Step 7: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

print("✅ Data ready for modeling!")
print(f"🔢 Training samples: {X_train.shape[0]}, Test samples: {X_test.shape[0]}")
print(f"🧠 Features used: {features}")
```

```
✅ Data ready for modeling!
🔢 Training samples: 431560, Test samples: 107890
🧠 Features used: ['Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6', 'Potassium', 'AlcoholicDrink
```

```
# 1. Import required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# 2. Select relevant final features from merged_df and drop missing values
selected_features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein',
    'Alcohol', 'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]

model_df = merged_df[selected_features].dropna().copy()

# 3. Create pseudo high-risk label based on Fat_to_Calorie_Ratio
# (patients with higher fat-to-calorie ratios may be at higher diabetes risk)
fat_to_calorie_median = model_df['Fat_to_Calorie_Ratio'].median()
model_df['HighRisk'] = (model_df['Fat_to_Calorie_Ratio'] > fat_to_calorie_median).astype(int)

# 4. Plot pairplot to visualize feature distributions
```

```
# (choose a subset for readability, otherwise too crowded)
sns.pairplot(
    data=model_df,
    vars=['TotalFat', 'Protein', 'Calories', 'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio', 'Age'],
    hue='HighRisk',
    palette='Set1',
    diag_kind='kde',
    plot_kws={'alpha': 0.6}
)

# 5. Beautify the plot
plt.suptitle(
    "Pairplot of All Patients (Colored by High Fat-to-Calorie Ratio Risk)",
    fontsize=14,
    y=1.02
)
plt.tight_layout()
plt.show()
```

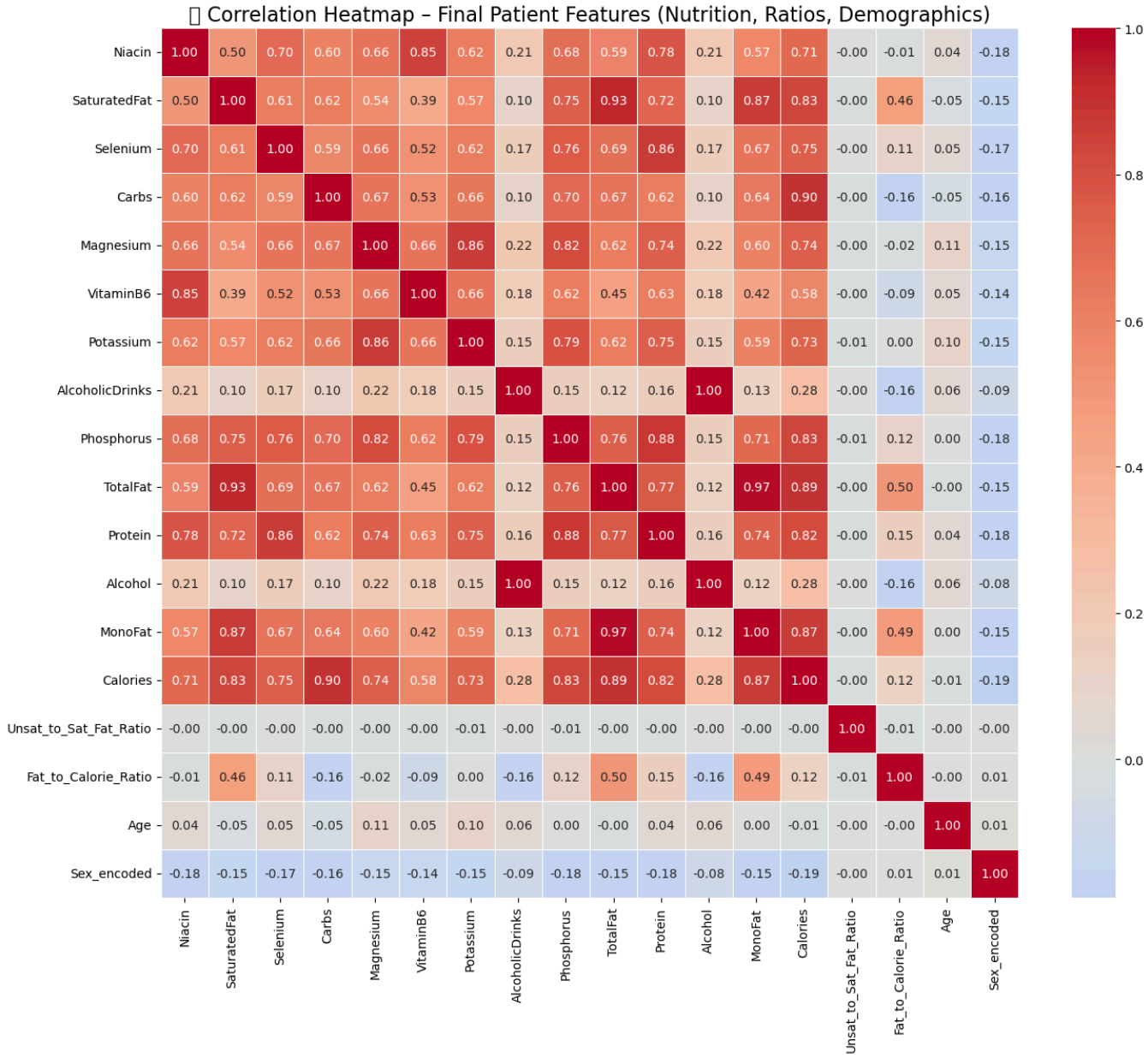
Pairplot of All Patients (Colored by High Fat-to-Calorie Ratio Risk)



- Your pairplot shows higher-dimensional relationships between nutrient balance and pseudo-risk.
- The color coding (HighRisk) depends on Fat-to-Calorie Ratio (stronger nutrition science meaning).

```
# 1. Import libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 2. Select final feature set
features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein',
    'Alcohol', 'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]

# 3. Drop missing values
model_df = merged_df[features].dropna().copy()

# 4. Create a pseudo HighRisk label based on Fat_to_Calorie_Ratio
fat_calorie_median = model_df['Fat_to_Calorie_Ratio'].median()
model_df['HighRisk'] = (model_df['Fat_to_Calorie_Ratio'] > fat_calorie_median).astype(int)

# 5. Compute correlation matrix (excluding HighRisk)
corr_matrix = model_df[features].corr()

# 6. Plot heatmap
plt.figure(figsize=(14, 12))
sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap='coolwarm',
    center=0,
    linewidths=0.5
)

# 7. Add title and format
plt.title("🔍 Correlation Heatmap — Final Patient Features (Nutrition, Ratios, Demographics)", fontsize=16)
plt.tight_layout()
plt.show()
```

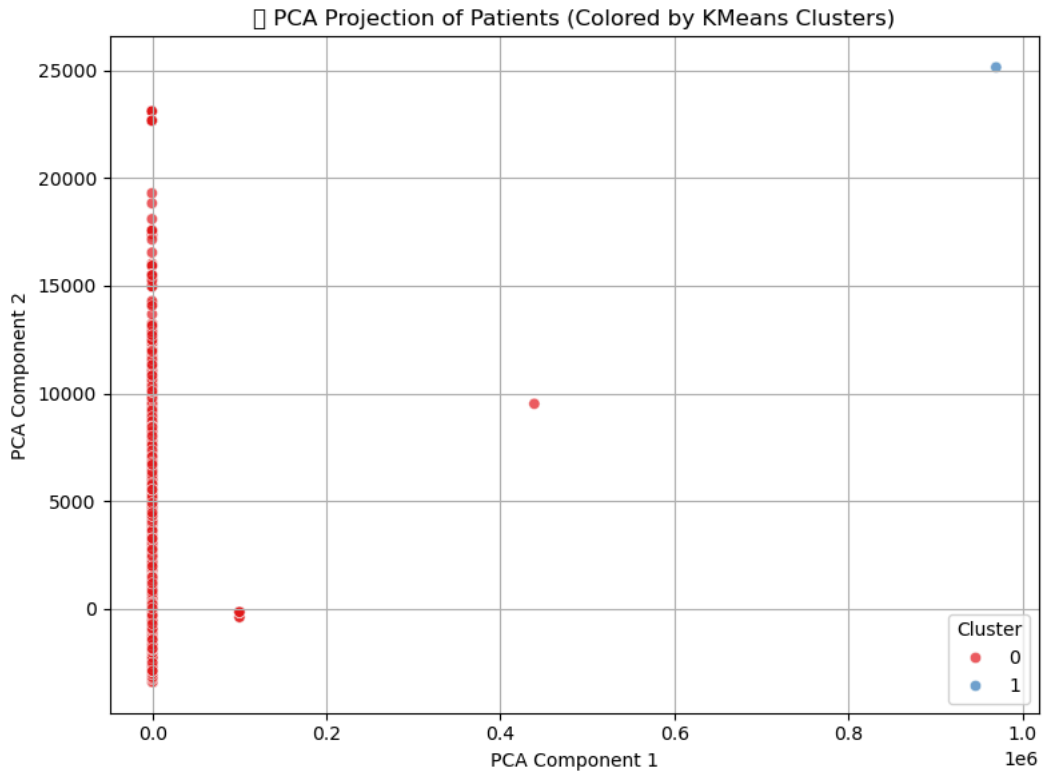
```
/var/folders/k7/5z9ffk3510dgxcdhs6bkjt0c0000gn/T/ipykernel_19295/3447378273.py:39: UserWarning: Glyph 128269 (\N{LEFT-PO
  plt.tight_layout()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 128269 (\N{LEFT-POINTING
  fig.canvas.print_figure(bytes_io, **kw)
```




```
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Prepare data for clustering (no need to drop HighRisk)
X_cluster = model_df[selected_features].copy()  # Only select your clean features

# Step 2: Fit KMeans
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_cluster)

# Step 3: Add clusters to the dataframe
model_df['Cluster'] = clusters
```

```
# Step 4: PCA Projection for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_cluster)

# Step 5: Plot the clusters in PCA space
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=model_df['Cluster'], palette='Set1', alpha=0.7)
plt.title("🧠 PCA Projection of Patients (Colored by KMeans Clusters)")
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/var/folders/k7/5z9ffk3510dgxcdhs6bkjt0c0000gn/T/ipykernel_19295/999126582.py:28: UserWarning: Glyph 129504 (\N{BRAIN})
  plt.tight_layout()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 129504 (\N{BRAIN}) missin
  fig.canvas.print_figure(bytes_io, **kw)
```




```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import (
    classification_report, confusion_matrix,
    roc_curve, roc_auc_score
)
import matplotlib.pyplot as plt
import joblib
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
rf_proba = rf.predict_proba(X_test)[:, 1]
```

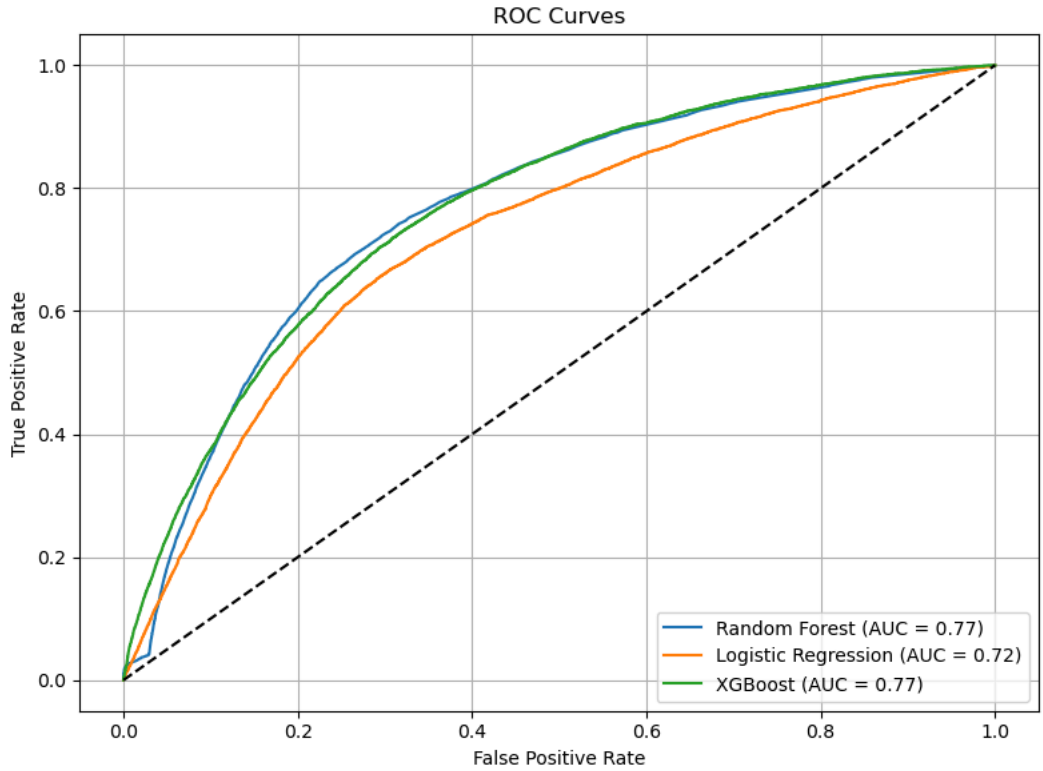
```
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
lr_proba = lr.predict_proba(X_test)[:, 1]
```

```
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train, y_train)
xgb_pred = xgb.predict(X_test)
xgb_proba = xgb.predict_proba(X_test)[:, 1]
```

```
/opt/anaconda3/lib/python3.12/site-packages/xgboost/training.py:183: UserWarning: [10:51:55] WARNING: /Users/runner/work
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
def evaluate_model(name, y_true, y_pred, y_proba):
    print(f"\n✅ {name} Results")
    print(classification_report(y_true, y_pred, target_names=["No Diabetes", "Diabetes"]))
    print("Confusion Matrix:")
    print(confusion_matrix(y_true, y_pred))
    auc = roc_auc_score(y_true, y_proba)
    print(f"ROC AUC: {auc:.4f}")
    fpr, tpr, _ = roc_curve(y_true, y_proba)
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc:.2f})')

plt.figure(figsize=(8,6))
evaluate_model("Random Forest", y_test, rf_pred, rf_proba)
evaluate_model("Logistic Regression", y_test, lr_pred, lr_proba)
evaluate_model("XGBoost", y_test, xgb_pred, xgb_proba)

plt.plot([0,1], [0,1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

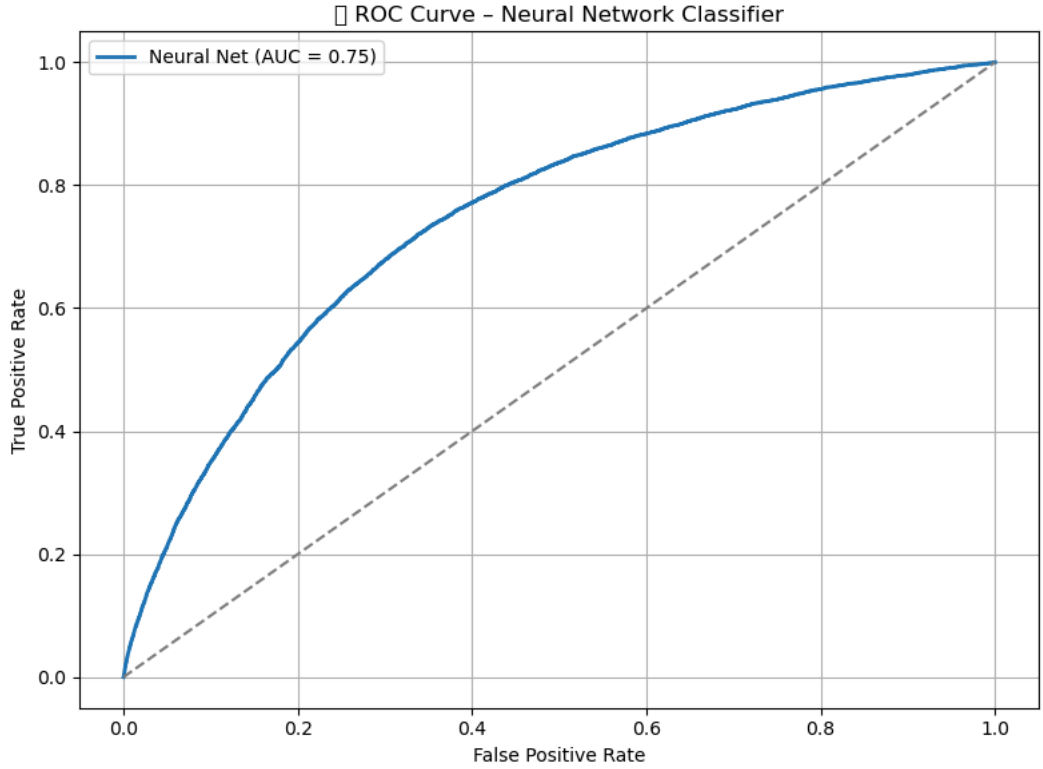
```
Random Forest Results
              precision    recall  f1-score   support

 No Diabetes       0.93      0.97      0.95    100071
    Diabetes       0.11      0.05      0.07      7819

    accuracy                           0.90    107890
   macro avg       0.52      0.51      0.51    107890
weighted avg       0.87      0.90      0.88    107890

Confusion Matrix:
[[97088  2983]
 [ 7454   365]]
ROC AUC: 0.7666

Logistic Regression Results
              precision    recall  f1-score   support

 No Diabetes       0.93      1.00      0.96    100071
    Diabetes       0.00      0.00      0.00      7819

    accuracy                           0.93    107890
   macro avg       0.46      0.50      0.48    107890
weighted avg       0.86      0.93      0.89    107890

Confusion Matrix:
[[100067      4]
 [  7819      0]]
ROC AUC: 0.7229

XGBoost Results
              precision    recall  f1-score   support

 No Diabetes       0.93      1.00      0.96    100071
    Diabetes       0.40      0.01      0.01      7819

    accuracy                           0.93    107890
   macro avg       0.66      0.50      0.49    107890
weighted avg       0.89      0.93      0.89    107890

Confusion Matrix:
[[99995    76]
 [ 7769    50]]
ROC AUC: 0.7679
```




```
# 1. Import necessary libraries
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt

# 2. Define final feature list
features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
```

```
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein',
    'Alcohol', 'MonoFat', 'Calories',
    'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio',
    'Age', 'Sex_encoded'
]

# 3. Prepare X and y
X = df[features]
y = df['Diabetes']

# 4. Split into training and testing (already done, but re-confirm)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 5. Build Neural Network (MLPClassifier)
mlp_model = MLPClassifier(
    hidden_layer_sizes=(64, 32),  # Two hidden layers
    activation='relu',
    solver='adam',
    learning_rate_init=0.001,
    max_iter=300,
    random_state=42
)

# 6. Train Neural Net
mlp_model.fit(X_train, y_train)

# 7. Predict
y_pred_mlp = mlp_model.predict(X_test)
y_proba_mlp = mlp_model.predict_proba(X_test)[:, 1]

# 8. Evaluation
print("\n📋 Classification Report (Neural Network):")
print(classification_report(y_test, y_pred_mlp))

print("\n📊 Confusion Matrix (Neural Network):")
print(confusion_matrix(y_test, y_pred_mlp))

# 9. ROC AUC
auc_mlp = roc_auc_score(y_test, y_proba_mlp)
print(f"\n📈 ROC AUC (Neural Network): {auc_mlp:.4f}")

# 10. Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba_mlp)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Neural Net (AUC = {auc_mlp:.2f})', linewidth=2)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('📈 ROC Curve — Neural Network Classifier')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
📋 Classification Report (Neural Network):
              precision    recall  f1-score   support

         0.0       0.93      1.00      0.96    100071
         1.0       0.25      0.00      0.00      7819

    accuracy                           0.93    107890
   macro avg       0.59      0.50      0.48    107890
weighted avg       0.88      0.93      0.89    107890


📊 Confusion Matrix (Neural Network):
[[100035     36]
 [  7807     12]]

📈 ROC AUC (Neural Network): 0.7471
/var/folders/k7/5z9ffk3510dgxcdhs6bkjt0c0000gn/T/ipykernel_19295/1243445586.py:63: UserWarning: Glyph 128200 (\N{CHART W
  plt.tight_layout()
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 128200 (\N{CHART WITH UPW
  fig.canvas.print_figure(bytes_io, **kw)
```



We'll use your 18 features: • 14 nutrient intake features • 2 engineered features (ratios) • 2 demographic features (Age, Sex_encoded)

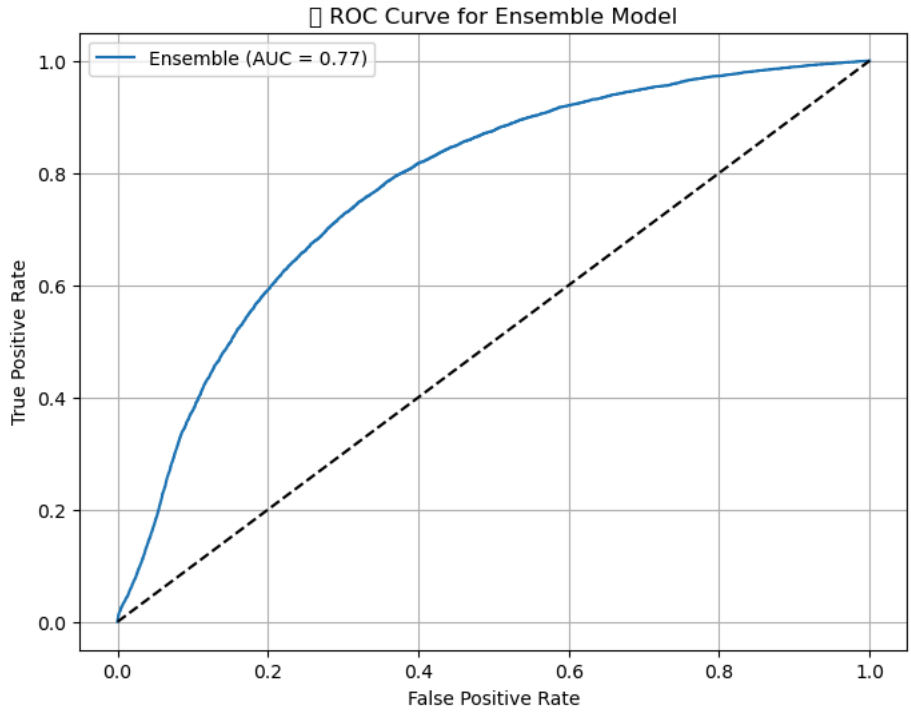
We'll set up a professional MLP neural network with: • 2 hidden layers (64, 32 neurons) • ReLU activation • Adam optimizer • AUC evaluation • Classification report and Confusion matrix

```
ENSEMBLE LEARNING MODEL – TRAINING
```

```
# 1. Import libraries
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# 2. Define final feature set
features = [
    'Niacin', 'SaturatedFat', 'Selenium', 'Carbs', 'Magnesium', 'VitaminB6',
    'Potassium', 'AlcoholicDrinks', 'Phosphorus', 'TotalFat', 'Protein', 'Alcohol',
    'MonoFat', 'Calories', 'Unsat_to_Sat_Fat_Ratio', 'Fat_to_Calorie_Ratio', 'Age', 'Sex_encoded'
]
target = 'DiabetesRisk'
```

```
# 3. Prepare dataset
model_df = merged_df[features + ['DIQ010']].dropna()
model_df = model_df[model_df['DIQ010'].isin([1.0, 2.0])]  # Only Yes/No
model_df['DiabetesRisk'] = model_df['DIQ010'].map({1.0: 1, 2.0: 0})

# 4. Define features and labels
X = model_df[features]
y = model_df[target]

# 5. Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# 7. Define base models
rf = RandomForestClassifier(n_estimators=100, random_state=42)
xgb = XGBClassifier(n_estimators=100, random_state=42, use_label_encoder=False, eval_metric='logloss')
lr = LogisticRegression(max_iter=1000, random_state=42)

# 8. Create the Ensemble (VotingClassifier)
ensemble_model = VotingClassifier(
    estimators=[('rf', rf), ('xgb', xgb), ('lr', lr)],
    voting='soft'  # 'soft' = use predicted probabilities (better usually)
)

# 9. Fit the ensemble
ensemble_model.fit(X_train, y_train)

# 10. Predict and evaluate
y_pred = ensemble_model.predict(X_test)
y_proba = ensemble_model.predict_proba(X_test)[:, 1]

print("\n📋 Ensemble Classification Report:")
print(classification_report(y_test, y_pred))

print("\n📊 Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print(f"🏆 ROC AUC Score: {roc_auc_score(y_test, y_proba):.4f}")

# 11. Plot ROC Curve
from sklearn.metrics import roc_curve

fpr, tpr, _ = roc_curve(y_test, y_proba)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f"Ensemble (AUC = {roc_auc_score(y_test, y_proba):.2f})")
plt.plot([0,1],[0,1],'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('🎯 ROC Curve for Ensemble Model')
plt.legend()
plt.grid(True)
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/xgboost/training.py:183: UserWarning: [10:57:03] WARNING: /Users/runner/work
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

📋 Ensemble Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96    100071
           1       0.51      0.00      0.01      7819

    accuracy                           0.93    107890
   macro avg       0.72      0.50      0.49    107890
weighted avg       0.90      0.93      0.89    107890


📊 Confusion Matrix:
[[100042     29]
 [  7789     30]]
🏆 ROC AUC Score: 0.7741
/opt/anaconda3/lib/python3.12/site-packages/IPython/core/pylabtools.py:170: UserWarning: Glyph 127919 (\N{DIRECT HIT}) m
  fig.canvas.print_figure(bytes_io, **kw)
```

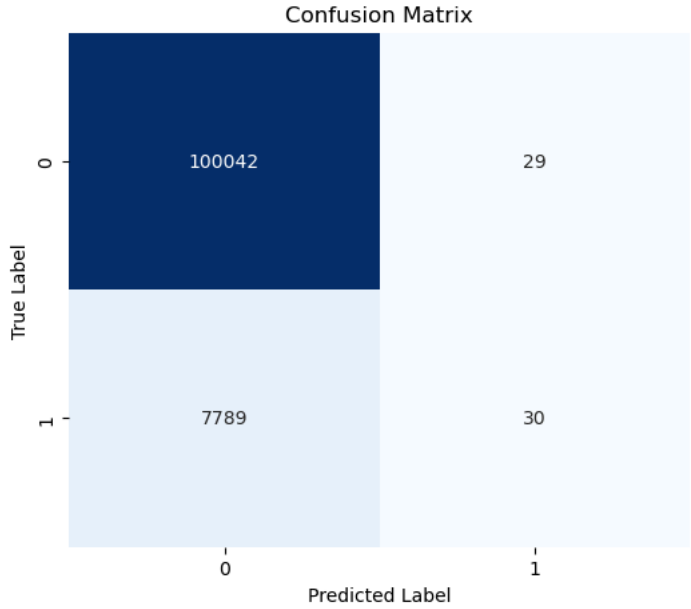


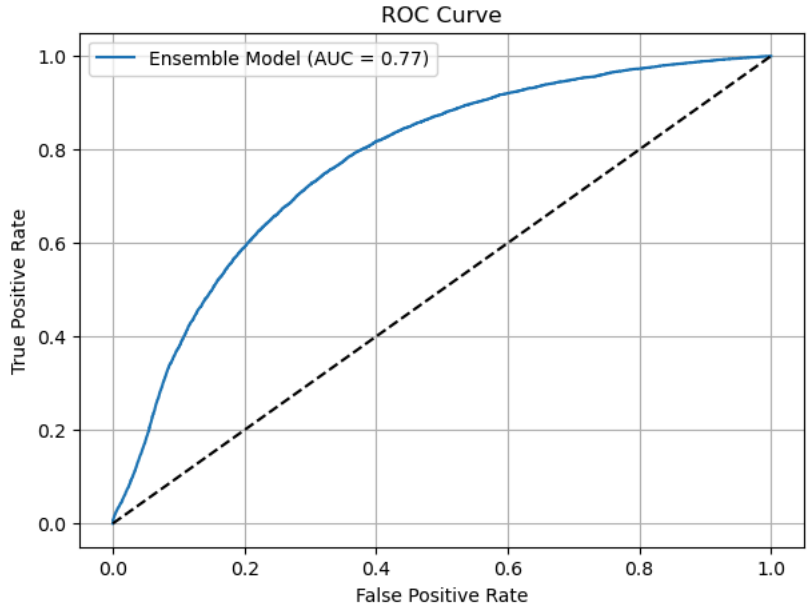
ENSEMBLE LEARNING MODEL - TESTING

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, classi
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Predict on the test set
y_pred = ensemble_model.predict(X_test)
y_proba = ensemble_model.predict_proba(X_test)[:, 1]  # Probabilities for ROC curve

# Step 2: Evaluate
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_proba)

# Step 3: Print all scores
print("✅ Model Test Results:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"ROC AUC Score: {auc:.4f}")

# Step 4: Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
```

```
plt.title('Confusion Matrix')
plt.show()

# Step 5: ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.figure(figsize=(7, 5))
plt.plot(fpr, tpr, label=f'Ensemble Model (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()

# Step 6: Classification Report
print("\n📋 Full Classification Report:")
print(classification_report(y_test, y_pred))
```

```
✅ Model Test Results:
Accuracy: 0.9275
Precision: 0.5085
Recall: 0.0038
F1-Score: 0.0076
ROC AUC Score: 0.7741
```






```
📋 Full Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.96    100071
           1       0.51      0.00      0.01      7819

    accuracy                           0.93    107890
   macro avg       0.72      0.50      0.49    107890
weighted avg       0.90      0.93      0.89    107890
```

```
# 1. Create a clean DataFrame for patients based on X
patients_risk_df = pd.DataFrame(X, columns=X.columns)  # Columns = same feature names

# 2. Predict risk probabilities
risk_proba = ensemble_model.predict_proba(X)[:, 1]

# 3. Attach diabetes risk score
patients_risk_df['DiabetesRiskScore'] = risk_proba

# 4. Sort by risk
top_risk_patients = patients_risk_df.sort_values(by='DiabetesRiskScore', ascending=False)

# 5. Display Top N patients
N = 20
print(f"🏥 Top {N} Highest-Risk Patients:")
```

```
display(top_risk_patients.head(N))
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but RandomForestClass
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but LogisticRegressio
  warnings.warn(
🏥 Top 20 Highest-Risk Patients:
```

	Niacin	SaturatedFat	Selenium	Carbs	Magnesium	VitaminB6	Potassium	AlcoholicDrinks	Phosphorus	TotalF
122869	0.000000	0.000000	0.200000	18.100000	3.000000	0.010000	39.000000	1.342880	3.000000	0.0000
122868	0.000000	0.000000	0.200000	18.100000	3.000000	0.010000	39.000000	1.342880	3.000000	0.0000
435944	1.135000	5.331000	1.200000	49.700000	100.000000	0.085000	197.000000	2.990000	205.000000	20.7600
282324	0.613000	0.005000	0.300000	0.890000	9.000000	0.000000	77.000000	0.000000	7.000000	0.0100
67305	8.700000	11.100000	61.500000	148.200000	59.000000	0.240000	512.000000	15.333640	304.000000	33.0000
67304	8.700000	11.100000	61.500000	148.200000	59.000000	0.240000	512.000000	15.333640	304.000000	33.0000
539684	0.090000	0.038000	0.400000	21.860000	24.000000	0.034000	94.000000	0.000000	8.000000	0.2100
252326	12.791000	31.993000	130.800000	365.760000	350.000000	2.355000	4014.000000	3.869000	1571.000000	98.4700
380347	30.189396	23.238773	119.719505	348.311748	321.095989	2.349387	2759.215225	4.534707	1442.049885	74.6914
13825	25.300000	24.500000	226.700000	274.300000	288.000000	2.280000	3205.000000	4.028640	1169.000000	92.5000
135303	25.060603	21.575887	129.281222	242.139896	326.150245	2.164670	2506.536752	4.187630	1209.605214	66.3461
210248	25.194854	27.779257	133.709513	252.631038	324.783853	2.118362	2808.084566	4.588544	1452.550089	80.1881
135302	25.060603	21.575887	129.281222	242.139896	326.150245	2.164670	2506.536752	4.187630	1209.605214	66.3461
329348	26.277608	26.542620	121.134687	279.721251	311.485014	2.118179	2639.252754	3.363820	1368.477518	84.7935
205306	26.924088	30.201030	112.827589	292.970632	304.366957	2.223063	2710.460310	3.558459	1545.727007	86.0010
13824	25.300000	24.500000	226.700000	274.300000	288.000000	2.280000	3205.000000	4.028640	1169.000000	92.5000
205304	24.090000	26.950000	81.310000	248.750000	287.000000	2.200000	2311.300000	4.670000	1530.250000	69.8800
312627	32.798489	38.334384	159.625596	303.205220	328.785615	2.411180	2830.946223	2.871514	1732.501989	111.0364
221390	29.225602	26.898192	121.355225	242.102889	342.439040	2.290539	3096.425449	4.314744	1423.458583	83.6497
322776	27.287081	29.493113	130.904945	236.351432	294.898871	2.337498	3159.988519	2.444872	1276.617827	94.6404

1.Predicts Diabetes Risk Score (probability) for all patients 2.Adds the risk score to their data 3.Sorts patients by highest risk first 4.Displays the top patients most likely to have diabetes

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Create patient DataFrame from X
patients_risk_df = pd.DataFrame(X, columns=features)  # feature_names = list of your model feature

# 2. Predict Risk Scores
risk_proba = ensemble_model.predict_proba(X)[:, 1]
patients_risk_df['DiabetesRiskScore'] = risk_proba

# 3. Sort patients by highest risk
top_risk_patients = patients_risk_df.sort_values(by='DiabetesRiskScore', ascending=False)

# 4. Display Top N
N = 10
print(f"🏥 Top {N} Highest-Risk Patients:")
display(top_risk_patients.head(N))


# 🎯 STEP 6: FLAG patients above 80% predicted diabetes risk
high_risk_patients = patients_risk_df[patients_risk_df['DiabetesRiskScore'] >= 0.8]

print(f"\n⚠️ Number of patients with >80% risk: {high_risk_patients.shape[0]}")
display(high_risk_patients)


# 📈 STEP 7: Plot Risk Score Distribution
plt.figure(figsize=(10,6))
sns.histplot(patients_risk_df['DiabetesRiskScore'], bins=30, kde=True, color='crimson')
plt.axvline(0.8, color='blue', linestyle='--', label='80% Risk Threshold')
plt.title("Distribution of Predicted Diabetes Risk Scores", fontsize=16)
plt.xlabel("Diabetes Risk Score (0 to 1)")
```

```
plt.ylabel("Number of Patients")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but RandomForestClass
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:486: UserWarning: X has feature names, but LogisticRegressio
  warnings.warn(
Top 10 Highest-Risk Patients:
```

	Niacin	SaturatedFat	Selenium	Carbs	Magnesium	VitaminB6	Potassium	Alcoholic	Drinks	Phosphorus	TotalFa
122869	0.000000	0.000000	0.200000	18.100000	3.000000	0.010000	39.000000	[illegible]	342880	3.000000	0.00000
122868	0.000000	0.000000	0.200000	18.100000	3.000000	0.010000	39.000000	[illegible]	342880	3.000000	0.00000