

```
In [52]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('diabetes.csv')
df.head()
```

```
Out[52]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeF
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
In [44]: df.dtypes
```

```
Out[44]: Pregnancies          int64
Glucose          int64
BloodPressure    int64
SkinThickness    int64
Insulin          int64
BMI              float64
DiabetesPedigreeFunction float64
Age              int64
Outcome          float64
dtype: object
```

```
In [45]: df.columns
```

```
Out[45]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='str')
```

```
In [46]: df.isnull().sum()
```

```
Out[46]: Pregnancies          0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
DiabetesPedigreeFunction    4
Age              0
Outcome          305
dtype: int64
```

```
In [47]: missing_percentage = df.isnull().mean() * 100
threshold = 30
# Filter columns with more than the threshold percentage of missing values
```

```

columns_to_drop = missing_percentage[missing_percentage > threshold].index
# Drop the columns
df = df.drop(columns=columns_to_drop)
# Display the resulting dataframe after removing columns with too many missing v
print(f"Columns dropped: {list(columns_to_drop)}")
print(f"Remaining columns: {df.columns}")

```

Columns dropped: ['Outcome']
 Remaining columns: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
 'BMI', 'DiabetesPedigreeFunction', 'Age'],
 dtype='str')

```

In [48]: #Pre-processing on dataset
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])
df.head()

```

Out[48]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
--	-------------	---------	---------------	---------------	---------	-----	--------------------------

0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	



```

In [49]: # Fill missing values in columns with mode
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean() # Get the mode (most frequent value)
    df[column] = df[column].fillna(mean_value)
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequent value)
    df[column] = df[column].fillna(mode_value)
df.isnull().sum()

```

Out[49]:

Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
dtype: int64	

```

In [50]: #Using IQR (Interquartile Range):
data=df.copy()
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR

```

```

upper_bound = Q3 + 1.5 * IQR
# Return a boolean series indicating if the value is an outlier
return (column < lower_bound) | (column > upper_bound)

# Apply the IQR method to each feature in the DataFrame
for col in data.columns:
    data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])

# Filter the DataFrame to display rows where any outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
# Display the filtered DataFrame with outliers
outliers_only

```

Out[50]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigree
4	0	137	40	35	168	43.1	
7	10	115	0	0	0	35.3	
8	2	197	70	45	543	30.5	
9	8	125	96	0	0	0.0	
12	10	139	80	0	0	27.1	
...	...	...	...	...	...	...	
706	10	115	0	0	0	0.0	
707	2	127	46	21	335	34.4	
710	3	158	64	13	387	31.2	
715	7	187	50	33	392	33.9	
753	0	181	88	44	510	43.3	

129 rows × 16 columns



In [64]:

```

data=df.copy()

def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <= upper_bound), np.nan)

# Replace outliers with NaN for each feature
for col in data.columns:
    data[col] = replace_outliers_with_nan(data[col])

null_counts = data.isnull().sum()
print("Sum of null values in each column:")
print(null_counts)

```

Sum of null values in each column:

Pregnancies	2
Glucose	2
BloodPressure	24
SkinThickness	1
Insulin	18
BMI	15
DiabetesPedigreeFunction	13
Age	12
Outcome	0

dtype: int64

```
In [54]: # Using the Interquartile Range (IQR) Method
data=df.copy()

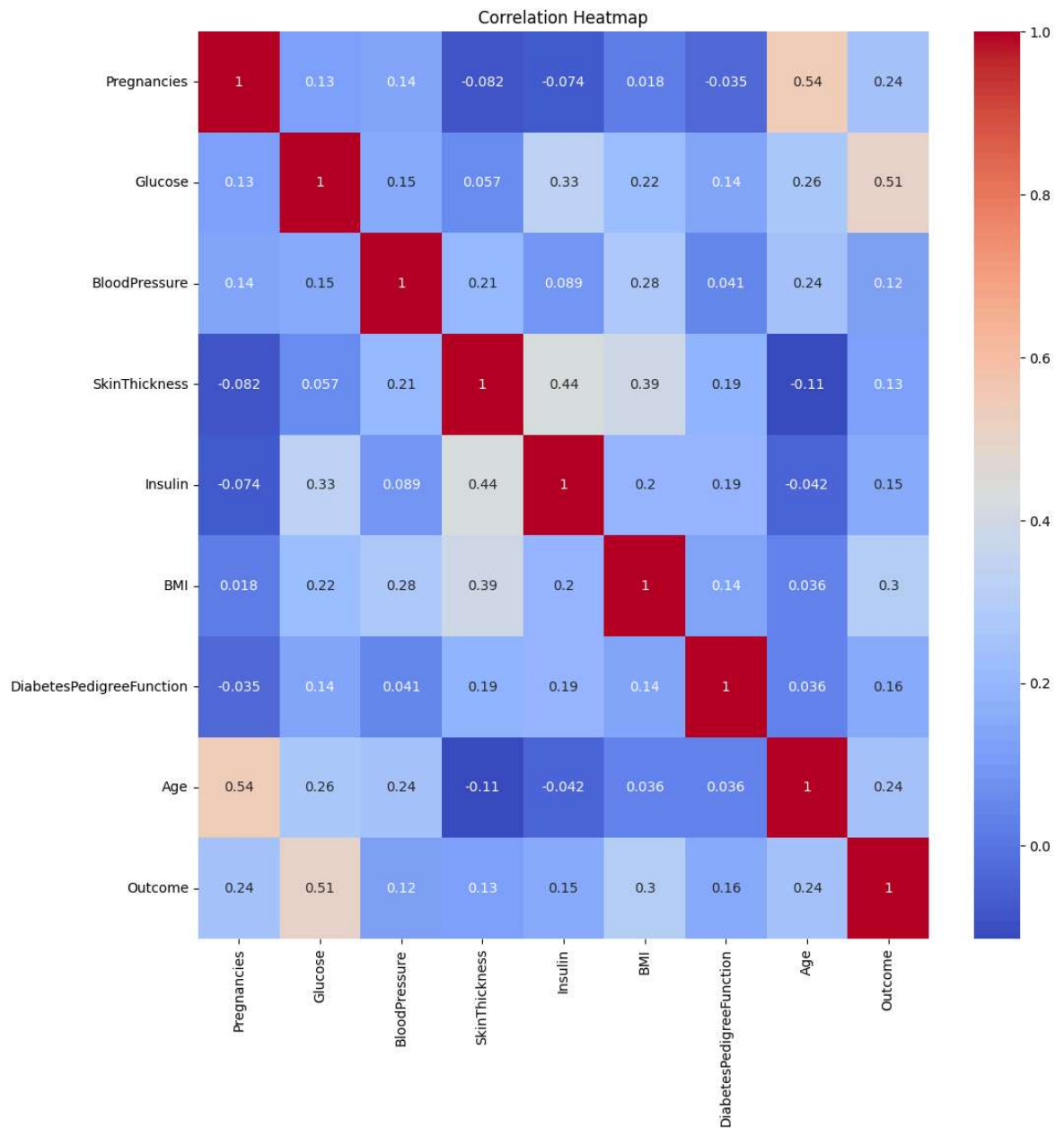
def remove_outliers_iqr(data):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each feature
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    # Define lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Filter the dataset to remove rows with outliers in any column
    data_cleaned = data[~((data < lower_bound) | (data > upper_bound)).any(axis=1)]
    return data_cleaned

cleaned_data = remove_outliers_iqr(data)
print("Original Data:")
print(len(data))
print("\nCleaned Data (without outliers):")
print(len(cleaned_data))
```

Original Data:
768

Cleaned Data (without outliers):
639

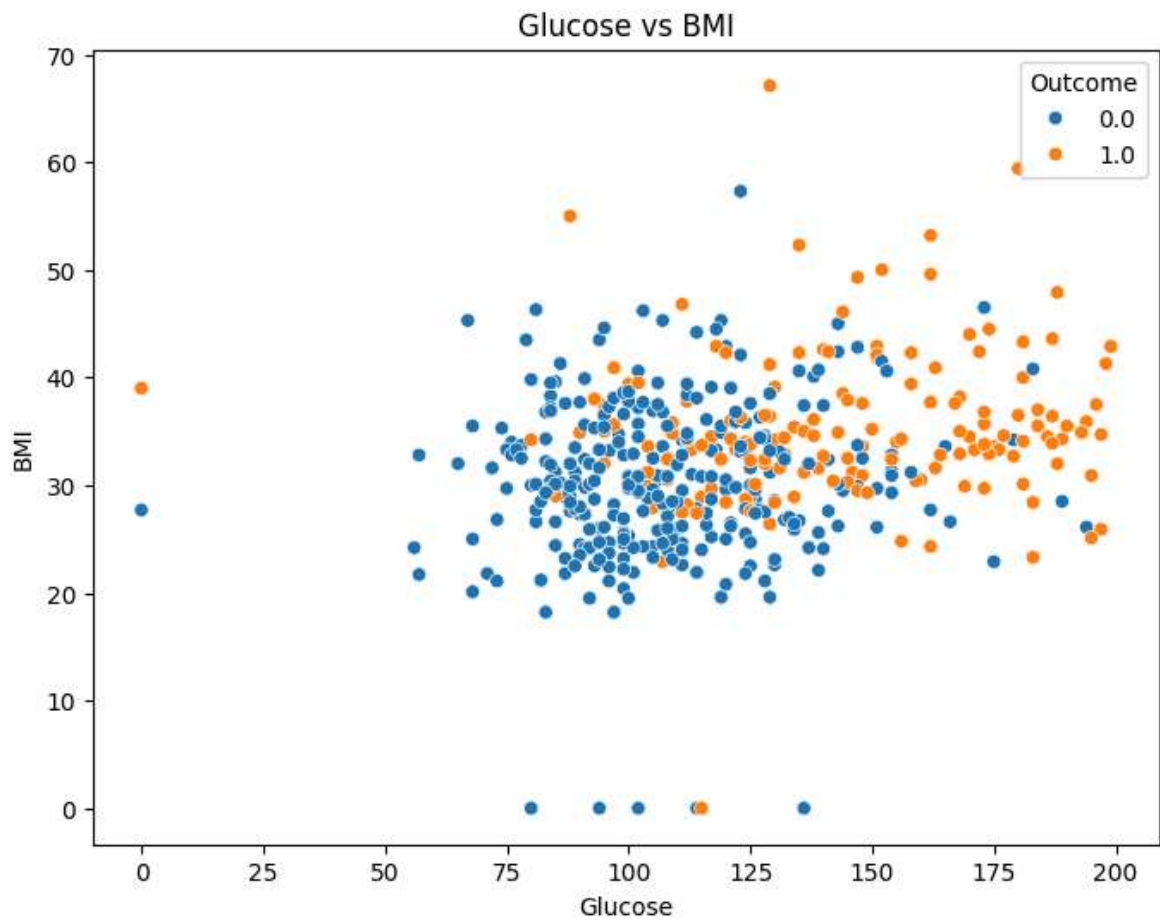
```
In [55]: plt.figure(figsize=(12, 12))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [58]: import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))
sns.scatterplot(x='Glucose', y='BMI', hue='Outcome', data=df)

plt.title("Glucose vs BMI")
plt.show()
```



```
In [65]: import matplotlib.pyplot as plt

df.hist(figsize=(12,10))
plt.show()
```

