Project : Diabetes Prediction from Health Indicators

Data Exploration

In [5]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import scipy.stats
from scipy.stats import chi2
from scipy import stats
```

In [2]:
```
# Load the dataset
df = pd.read_csv('diabetes_binary_health_indicators_BRFSS2015.csv')
df.head()
```

Out[2]:

	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0

5 rows × 22 columns

In [3]:
```
df.describe(include = "all").T
```

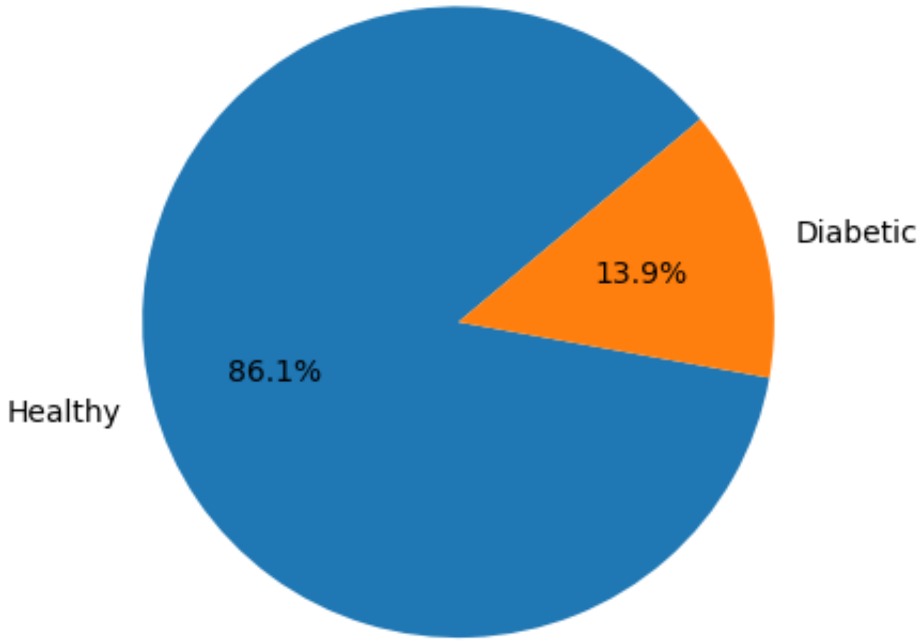
Out[3]:

	count	mean	std	min	25%	50%	75%	max
Diabetes_binary	253680.0	0.139333	0.346294	0.0	0.0	0.0	0.0	1.0
HighBP	253680.0	0.429001	0.494934	0.0	0.0	0.0	1.0	1.0
HighChol	253680.0	0.424121	0.494210	0.0	0.0	0.0	1.0	1.0
CholCheck	253680.0	0.962670	0.189571	0.0	1.0	1.0	1.0	1.0
BMI	253680.0	28.382364	6.608694	12.0	24.0	27.0	31.0	98.0
Smoker	253680.0	0.443169	0.496761	0.0	0.0	0.0	1.0	1.0
Stroke	253680.0	0.040571	0.197294	0.0	0.0	0.0	0.0	1.0
HeartDiseaseorAttack	253680.0	0.094186	0.292087	0.0	0.0	0.0	0.0	1.0
PhysActivity	253680.0	0.756544	0.429169	0.0	1.0	1.0	1.0	1.0
Fruits	253680.0	0.634256	0.481639	0.0	0.0	1.0	1.0	1.0
Veggies	253680.0	0.811420	0.391175	0.0	1.0	1.0	1.0	1.0
HvyAlcoholConsump	253680.0	0.056197	0.230302	0.0	0.0	0.0	0.0	1.0
AnyHealthcare	253680.0	0.951053	0.215759	0.0	1.0	1.0	1.0	1.0
NoDocbcCost	253680.0	0.084177	0.277654	0.0	0.0	0.0	0.0	1.0
GenHlth	253680.0	2.511392	1.068477	1.0	2.0	2.0	3.0	5.0
MentHlth	253680.0	3.184772	7.412847	0.0	0.0	0.0	2.0	30.0
PhysHlth	253680.0	4.242081	8.717951	0.0	0.0	0.0	3.0	30.0
DiffWalk	253680.0	0.168224	0.374066	0.0	0.0	0.0	0.0	1.0
Sex	253680.0	0.440342	0.496429	0.0	0.0	0.0	1.0	1.0
Age	253680.0	8.032119	3.054220	1.0	6.0	8.0	10.0	13.0
Education	253680.0	5.050434	0.985774	1.0	4.0	5.0	6.0	6.0
Income	253680.0	6.053875	2.071148	1.0	5.0	7.0	8.0	8.0

In [4]:
```
df.isnull().sum()
```

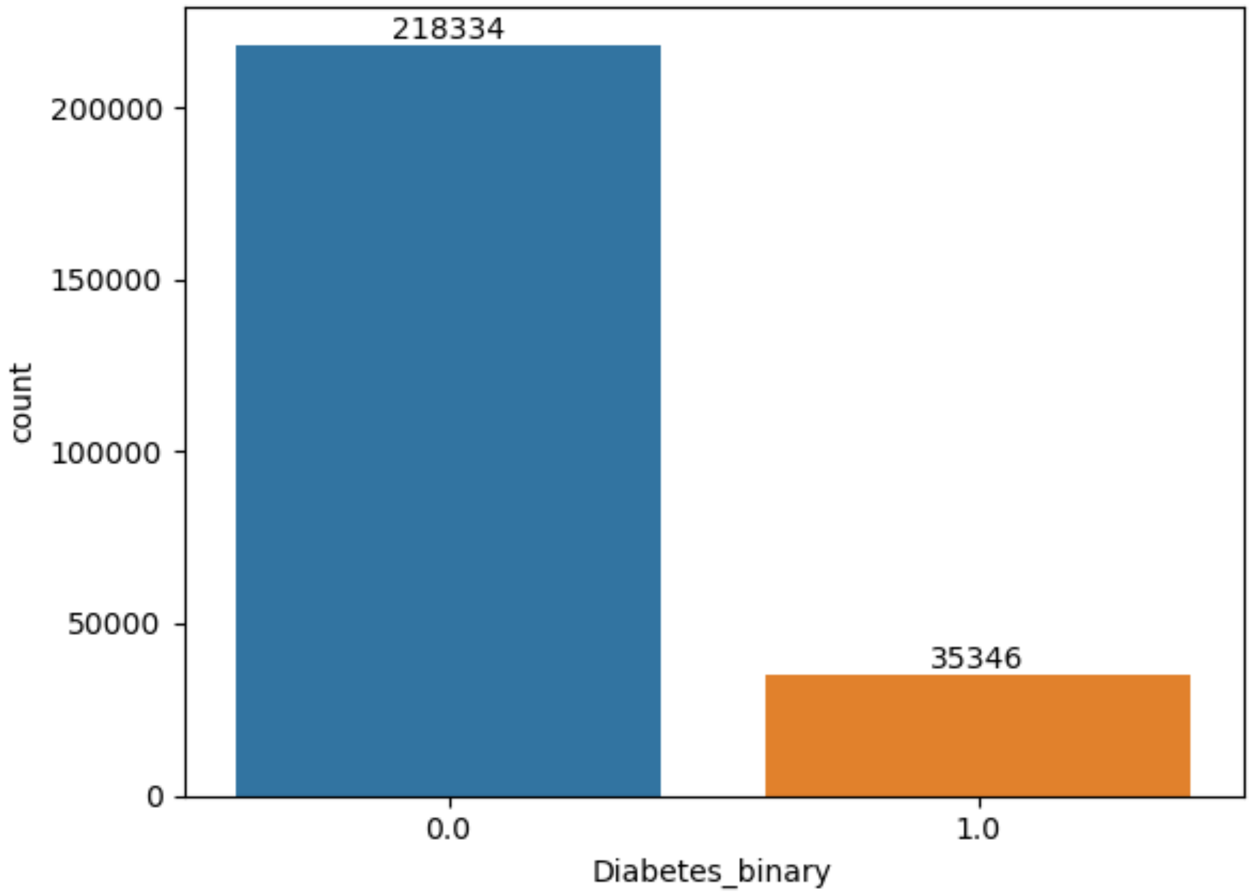
Out[4]:
```
Diabetes_binary         0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
HeartDiseaseorAttack    0
PhysActivity            0
Fruits                  0
Veggies                 0
HvyAlcoholConsump       0
AnyHealthcare           0
NoDocbcCost             0
GenHlth                 0
MentHlth                0
PhysHlth                0
DiffWalk                0
Sex                     0
Age                     0
Education               0
Income                  0
dtype: int64
```

In [5]:
```
# Pie chart for target variable 'Diabetes_012'
labels = 'Healthy','Diabetic'
df.Diabetes_binary.value_counts().plot.pie(labels=labels, autopct='%1.1f%%', startangle=40, label='');
```



In [6]:

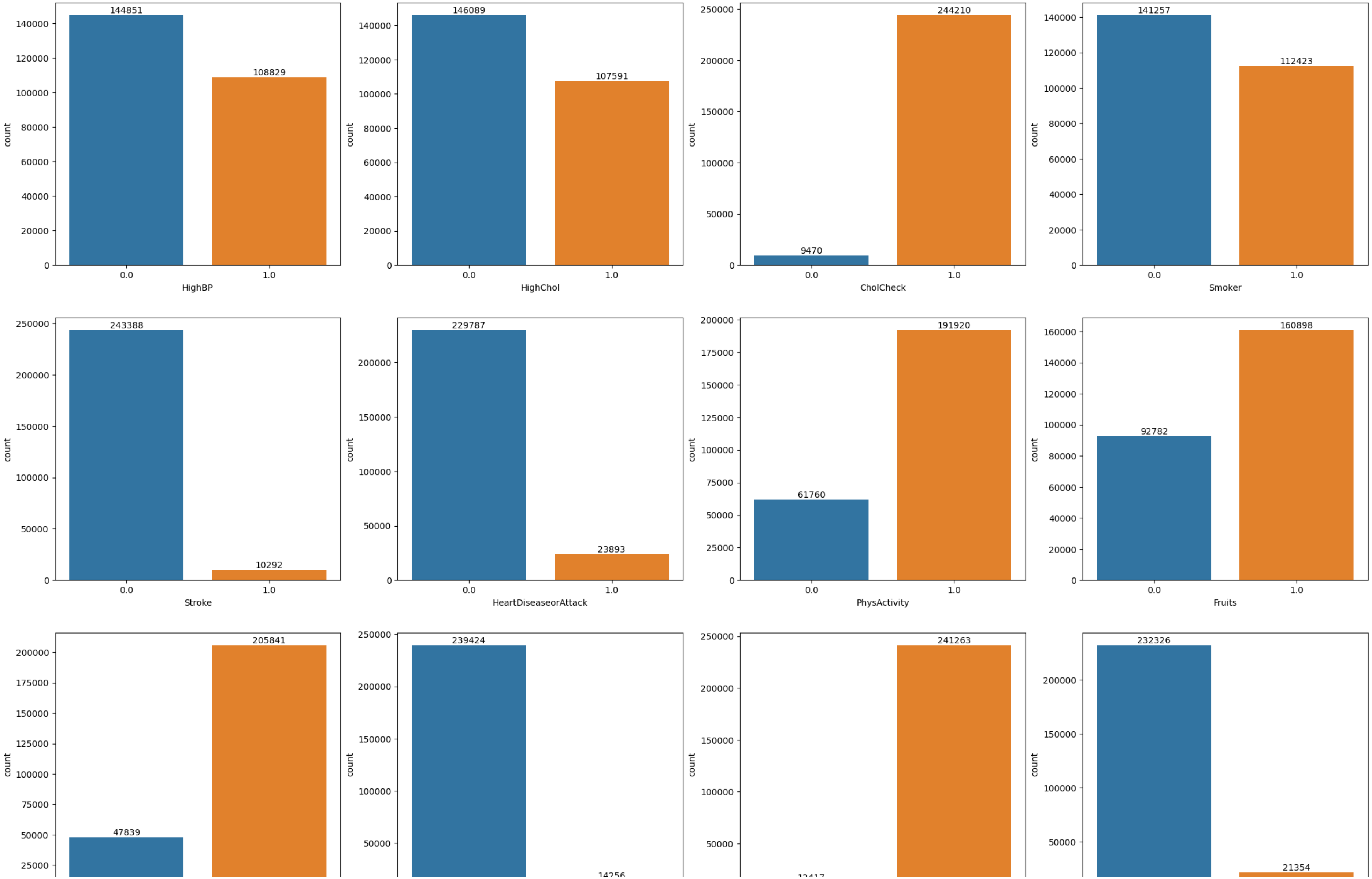
```
# Count plot for target variable
ax=sns.countplot(data=df, x='Diabetes_binary')
for i in ax.containers:
    ax.bar_label(i,)
```

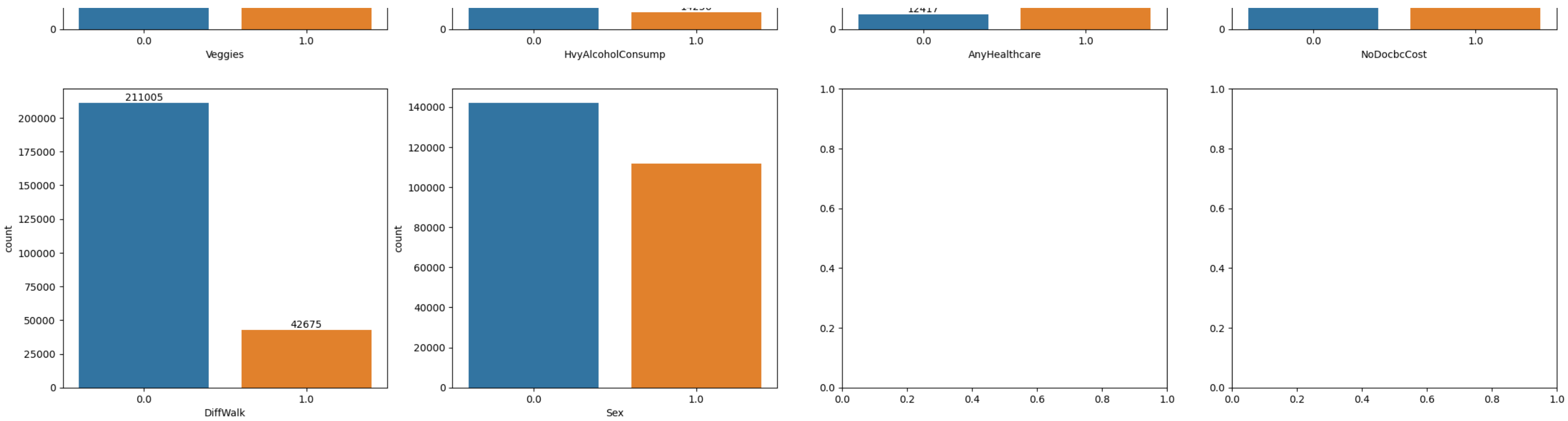


In [7]:
```
# Count plots for binary variables

fig, axes = plt.subplots(4, 4, figsize=(27,25))

ax1 = sns.countplot(ax=axes[0, 0], data=df, x='HighBP')
ax2 = sns.countplot(ax=axes[0, 1], data=df, x='HighChol')
ax3 = sns.countplot(ax=axes[0, 2], data=df, x='CholCheck')
ax4 = sns.countplot(ax=axes[0, 3], data=df, x='Smoker')
ax5 = sns.countplot(ax=axes[1, 0], data=df, x='Stroke')
ax6 = sns.countplot(ax=axes[1, 1], data=df, x='HeartDiseaseorAttack')
ax7 = sns.countplot(ax=axes[1, 2], data=df, x='PhysActivity')
ax8 = sns.countplot(ax=axes[1, 3], data=df, x='Fruits')
ax9 = sns.countplot(ax=axes[2, 0], data=df, x='Veggies')
```

```
ax10 = sns.countplot(ax=axes[2, 1], data=df, x='HvyAlcoholConsump')
ax11 = sns.countplot(ax=axes[2, 2], data=df, x='AnyHealthcare')
ax12 = sns.countplot(ax=axes[2, 3], data=df, x='NoDocbcCost')
ax13 = sns.countplot(ax=axes[3, 0], data=df, x='DiffWalk')
ax14 = sns.countplot(ax=axes[3, 1], data=df, x='Sex')

for i in ax1.containers:
    ax1.bar_label(i,)

for i in ax2.containers:
    ax2.bar_label(i,)

for i in ax3.containers:
    ax3.bar_label(i,)

for i in ax4.containers:
    ax4.bar_label(i,)

for i in ax5.containers:
    ax5.bar_label(i,)

for i in ax6.containers:
    ax6.bar_label(i,)

for i in ax7.containers:
    ax7.bar_label(i,)

for i in ax8.containers:
    ax8.bar_label(i,)

for i in ax9.containers:
    ax9.bar_label(i,)

for i in ax10.containers:
    ax10.bar_label(i,)

for i in ax11.containers:
    ax11.bar_label(i,)

for i in ax12.containers:
    ax12.bar_label(i,)
```

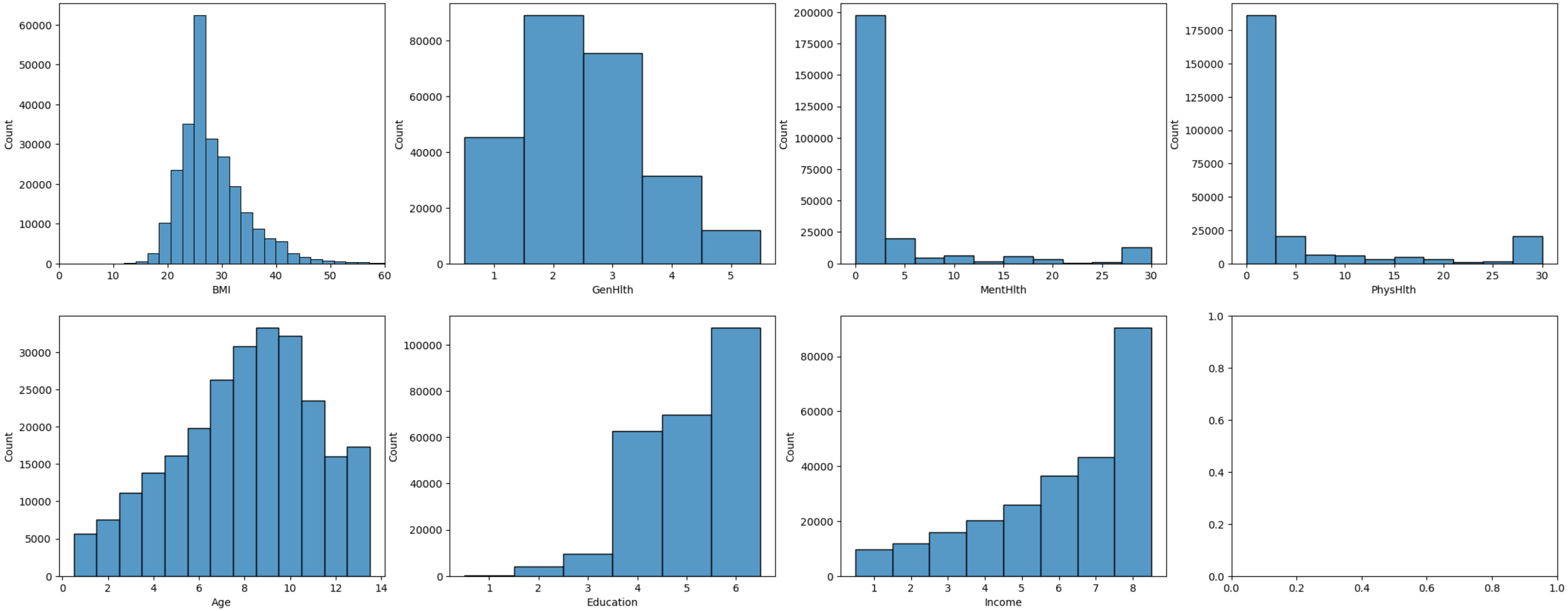
```python
for i in ax13.containers:
    ax13.bar_label(i,)

# for i in ax14.containers:
#     ax14.bar_label(i,)
```





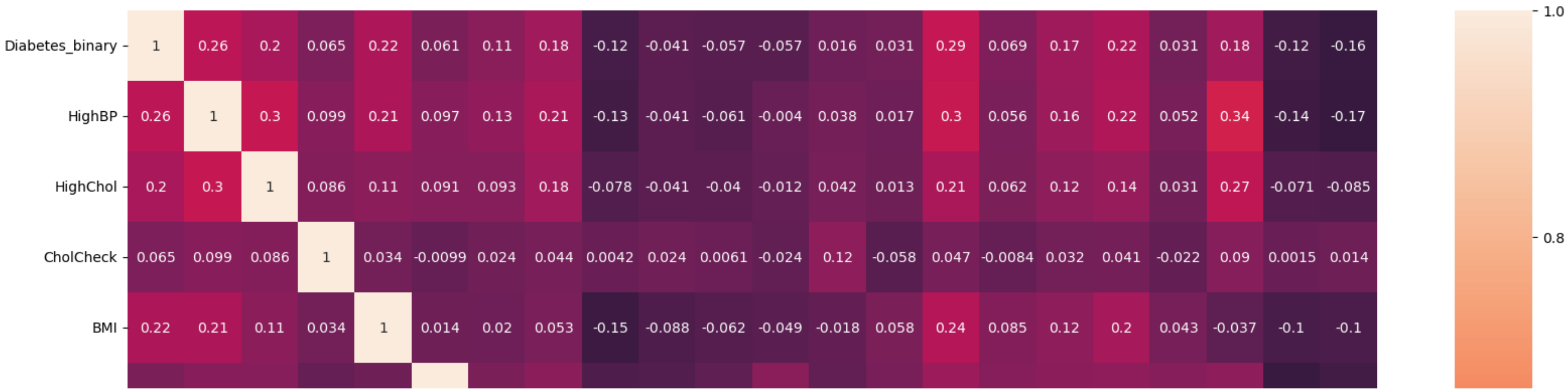
In [8]:
```python
# Histograms for numeric variables

fig, axes = plt.subplots(2, 4, figsize=(26,10))

sns.histplot(ax=axes[0, 0], data=df, x="BMI", bins=40)
axes[0, 0].set_xlim(0,60)
sns.histplot(ax=axes[0, 1], data=df, x="GenHlth", discrete=True)
sns.histplot(ax=axes[0, 2], data=df, x="MentHlth", bins=10)
sns.histplot(ax=axes[0, 3], data=df, x="PhysHlth", bins=10)
sns.histplot(ax=axes[1, 0], data=df, x="Age", discrete=True)
sns.histplot(ax=axes[1, 1], data=df, x="Education", discrete=True)
sns.histplot(ax=axes[1, 2], data=df, x="Income", discrete=True)
```

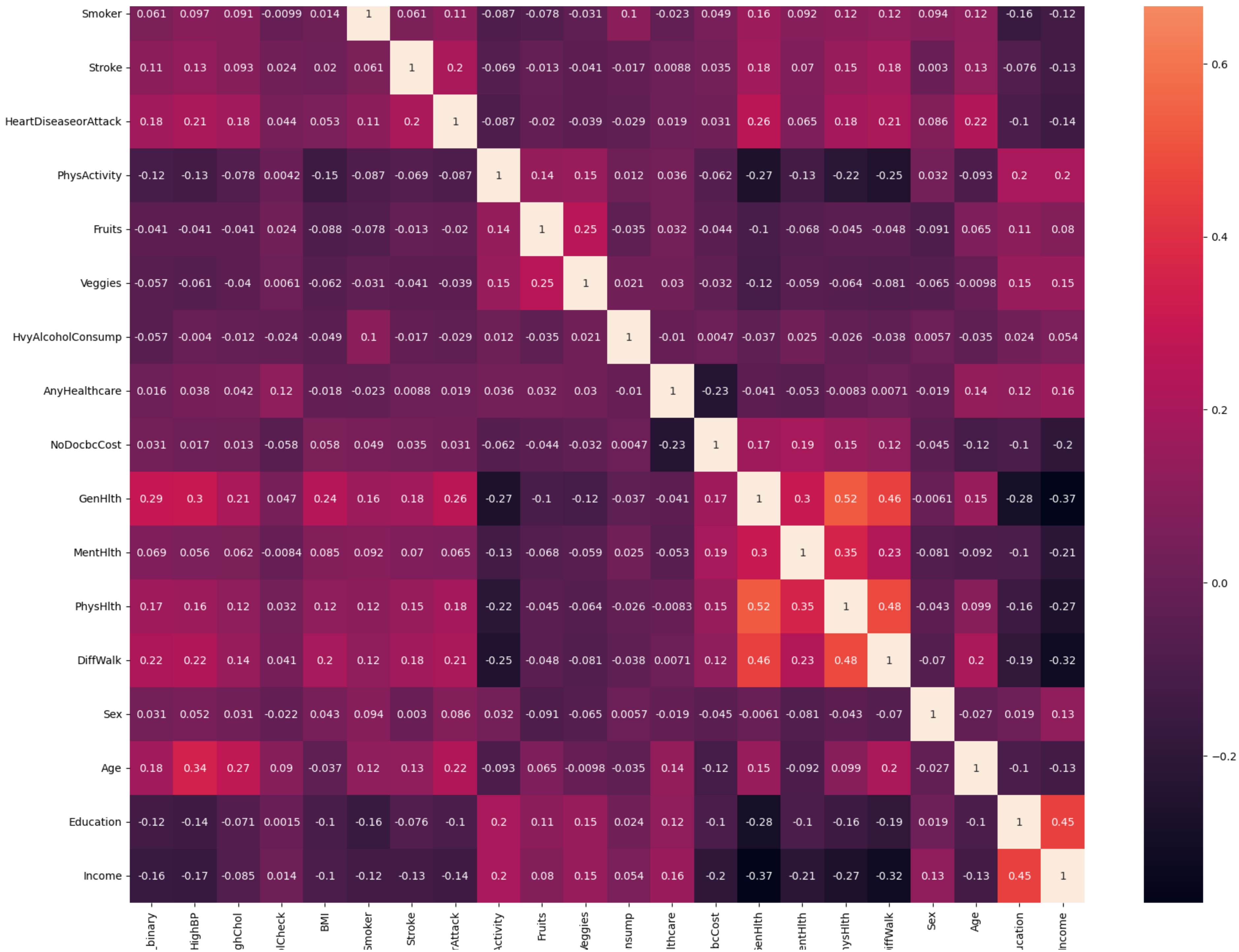
Out[8]:
```
<Axes: xlabel='Income', ylabel='Count'>
```



In [9]:
```python
# Correlation heatmap for all variables
plt.figure(figsize=(20, 20))
sns.heatmap(df.corr(), annot=True)
```

Out[9]: <Axes: >





Smoker 0.061 0.097 0.091 -0.0099 0.014 1 0.061 0.11 -0.087 -0.078 -0.031 0.1 -0.023 0.049 0.16 0.092 0.12 0.12 0.094 0.12 -0.16 -0.12
Stroke 0.11 0.13 0.093 0.024 0.02 0.061 1 0.2 -0.069 -0.013 -0.041 -0.017 0.0088 0.035 0.18 0.07 0.15 0.18 0.003 0.13 -0.076 -0.13
HeartDiseaseorAttack 0.18 0.21 0.18 0.044 0.053 0.11 0.2 1 -0.087 -0.02 -0.039 -0.029 0.019 0.031 0.26 0.065 0.18 0.21 0.086 0.22 -0.1 -0.14
PhysActivity -0.12 -0.13 -0.078 0.0042 -0.15 -0.087 -0.069 -0.087 1 0.14 0.15 0.012 0.036 -0.062 -0.27 -0.13 -0.22 -0.25 0.032 -0.093 0.2 0.2
Fruits -0.041 -0.041 -0.041 0.024 -0.088 -0.078 -0.013 -0.02 0.14 1 0.25 -0.035 0.032 -0.044 -0.1 -0.068 -0.045 -0.048 -0.091 0.065 0.11 0.08
Veggies -0.057 -0.061 -0.04 0.0061 -0.062 -0.031 -0.041 -0.039 0.15 0.25 1 0.021 0.03 -0.032 -0.12 -0.059 -0.064 -0.081 -0.065 -0.0098 0.15 0.15
HvyAlcoholConsump -0.057 -0.004 -0.012 -0.024 -0.049 0.1 -0.017 -0.029 0.012 -0.035 0.021 1 -0.01 0.0047 -0.037 0.025 -0.026 -0.038 0.0057 -0.035 0.024 0.054
AnyHealthcare 0.016 0.038 0.042 0.12 -0.018 -0.023 0.0088 0.019 0.036 0.032 0.03 -0.01 1 -0.23 -0.041 -0.053 -0.0083 0.0071 -0.019 0.14 0.12 0.16
NoDocbcCost 0.031 0.017 0.013 -0.058 0.058 0.049 0.035 0.031 -0.062 -0.044 -0.032 0.0047 -0.23 1 0.17 0.19 0.15 0.12 -0.045 -0.12 -0.1 -0.2
GenHlth 0.29 0.3 0.21 0.047 0.24 0.16 0.18 0.26 -0.27 -0.1 -0.12 -0.037 -0.041 0.17 1 0.3 0.52 0.46 -0.0061 0.15 -0.28 -0.37
MentHlth 0.069 0.056 0.062 -0.0084 0.085 0.092 0.07 0.065 -0.13 -0.068 -0.059 0.025 -0.053 0.19 0.3 1 0.35 0.23 -0.081 -0.092 -0.1 -0.21
PhysHlth 0.17 0.16 0.12 0.032 0.12 0.12 0.15 0.18 -0.22 -0.045 -0.064 -0.026 -0.0083 0.15 0.52 0.35 1 0.48 -0.043 0.099 -0.16 -0.27
DiffWalk 0.22 0.22 0.14 0.041 0.2 0.12 0.18 0.21 -0.25 -0.048 -0.081 -0.038 0.0071 0.12 0.46 0.23 0.48 1 -0.07 0.2 -0.19 -0.32
Sex 0.031 0.052 0.031 -0.022 0.043 0.094 0.003 0.086 0.032 -0.091 -0.065 0.0057 -0.019 -0.045 -0.0061 -0.081 -0.043 -0.07 1 -0.027 0.019 0.13
Age 0.18 0.34 0.27 0.09 -0.037 0.12 0.13 0.22 -0.093 0.065 -0.0098 -0.035 0.14 -0.12 0.15 -0.092 0.099 0.2 -0.027 1 -0.1 -0.13
Education -0.12 -0.14 -0.071 0.0015 -0.1 -0.16 -0.076 -0.1 0.2 0.11 0.15 0.024 0.12 -0.1 -0.28 -0.1 -0.16 -0.19 0.019 -0.1 1 0.45
Income -0.16 -0.17 -0.085 0.014 -0.1 -0.12 -0.13 -0.14 0.2 0.08 0.15 0.054 0.16 -0.2 -0.37 -0.21 -0.27 -0.32 0.13 -0.13 0.45 1
_binary
HighBP
ghChol
lCheck
BMI
Smoker
Stroke
rAttack
Activity
Fruits
Veggies
nsump
lthcare
bcCost
ienHlth
entHlth
ysHlth
iffWalk
Sex
Age
ucation
Income
0.6
0.4
0.2
0.0
−0.2

Dimension Reduction

In [10]:
```python
import sklearn
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from numpy import linalg as LA
from sklearn.preprocessing import StandardScaler
from scipy.stats import chi2_contingency

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from scipy.stats import pearsonr
```

In [11]:
```python
df.head()
```

Out[11]:

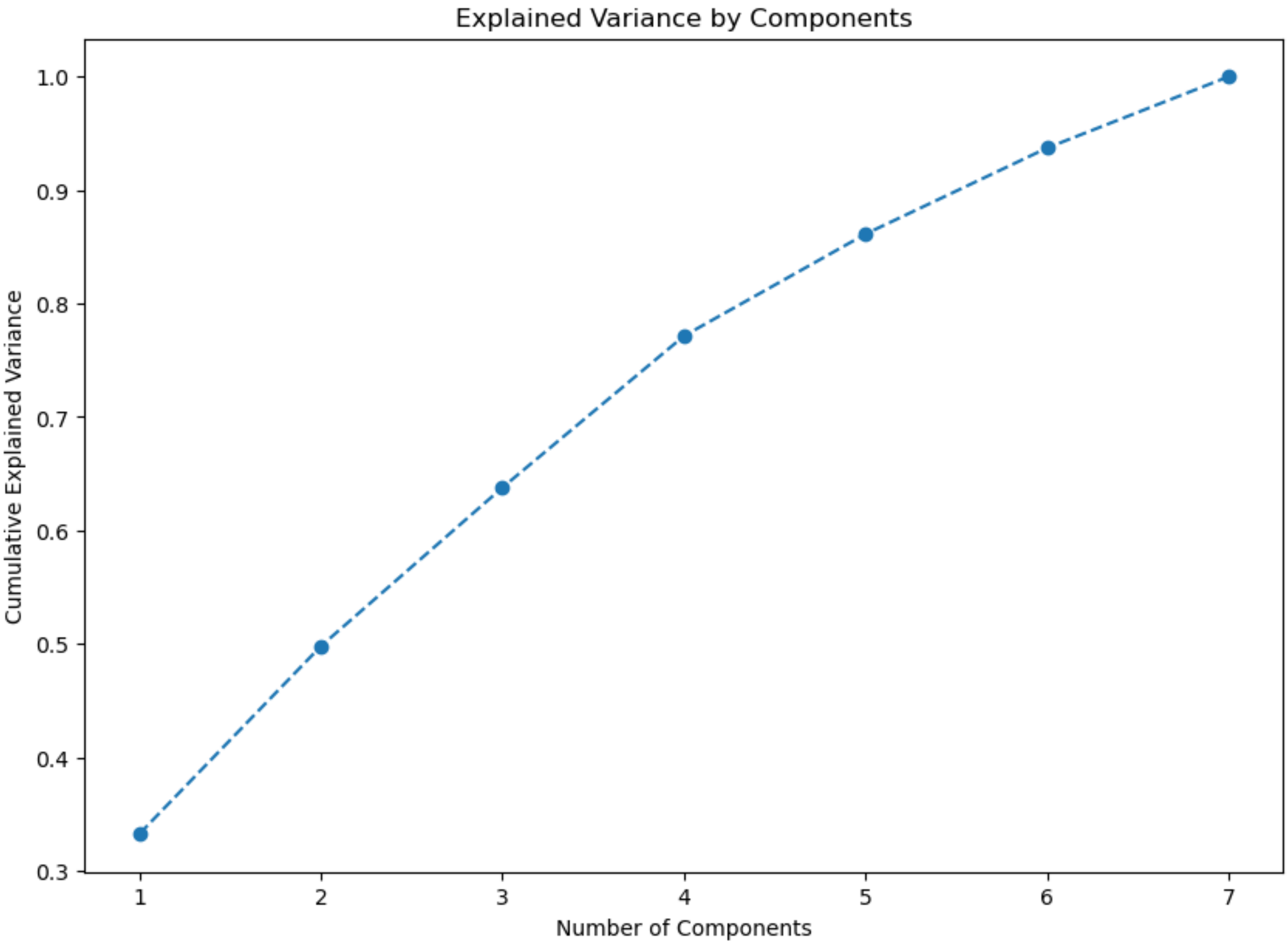
	Diabetes_binary	HighBP	HighChol	CholCheck	BMI	Smoker	Stroke	HeartDiseaseorAttack	PhysActivity	Fruits	...	AnyHealthcare
0	0.0	1.0	1.0	1.0	40.0	1.0	0.0	0.0	0.0	0.0	...	1.0
1	0.0	0.0	0.0	0.0	25.0	1.0	0.0	0.0	1.0	0.0	...	0.0
2	0.0	1.0	1.0	1.0	28.0	0.0	0.0	0.0	0.0	1.0	...	1.0
3	0.0	1.0	0.0	1.0	27.0	0.0	0.0	0.0	1.0	1.0	...	1.0
4	0.0	1.0	1.0	1.0	24.0	0.0	0.0	0.0	1.0	1.0	...	1.0

5 rows × 22 columns

In [12]:
```python
x = df[['BMI','GenHlth','MentHlth','PhysHlth','Age','Education','Income']]
```

In [13]:
```
# Fitting the scaler to the data 'X' and transforming 'X' to standardize the features
scaler = StandardScaler()
X_standard = scaler.fit_transform(x)

# Initializing PCA to reduce the dimensionality of the data to 2 principal components
pca = PCA()

# Fit the PCA model to the data 'X' and transform it to get the principal components
pca_scores_4 = pd.DataFrame(pca.fit_transform(X_standard))

# Displaying the PCA scores
print(f" The PCA scores : {pca_scores_4}")

# Calculate and print the explained variance of each principal component
explained_var  = pca.explained_variance_
print(f" Explained variance : {explained_var}")

# Calculate and print the proportion of variance explained by each principal component
proportion_var = pca.explained_variance_ratio_
print(f" Proportion Variance : {proportion_var}")

# Calculate and print the cumulative proportion of variance
cummulative_proportion_var = np.cumsum(proportion_var)
print(f"Cumulative proportion : {cummulative_proportion_var}")


# Plotting Explained Variance by Components and number of components
plt.figure(figsize=(10, 7))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_ratio_.cumsum(), marker='o', li
# plt.axhline(y=.95, linewidth=2, color = 'k')
# plt.axvline(x =18.7, color='k')

plt.title('Explained Variance by Components')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```

```
 The PCA scores :                 0         1         2         3         4         5         6
0       3.938752  0.797596  0.457037  0.549012  0.698055 -0.022165 -0.542490
1       0.441505 -0.529394  0.194478 -0.763594 -0.258081 -2.530424 -0.568373
2       3.816074  2.263866 -2.117000 -0.308808  0.483786  2.202473 -0.112776
3       0.248023 -1.862317  0.656276 -0.402573  0.355055  1.272845  0.311353
4      -0.030911 -1.235052 -0.349364 -0.502209  0.689224 -0.669848  0.110338
...          ...       ...       ...       ...       ...       ...       ...
253675 -0.021012  1.587194  1.408416  1.998868 -0.348543 -0.472571 -0.051610
253676  1.730174 -2.755133  0.266663 -1.580773 -0.510304  1.293905 -1.115803
253677 -0.492869  0.210255  1.803795 -1.694444 -0.322790 -1.697544  0.873030
253678  0.751364 -1.011319  0.453682 -1.291252 -0.363056 -1.872321 -0.516722
253679 -0.170585 -0.861304 -0.097272 -0.527703  0.282513 -2.149080  0.120548

[253680 rows x 7 columns]
 Explained variance : [2.32636026 1.1587944  0.97890314 0.93575313 0.62908012 0.53093537
 0.44020117]
 Proportion Variance : [0.33233587 0.1655414  0.13984275 0.13367849 0.08986823 0.07584761
 0.06288563]
Cumulative proportion : [0.33233587 0.49787728 0.63772003 0.77139852 0.86126676 0.93711437
 1.        ]
```


Explained Variance by Components
Cumulative Explained Variance
Number of Components
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
1
2
3
4
5
6
7

Applying the Dimension Reduction for Categorical Variable

In [14]:
```
df_columns = df.columns
df_ = df
for x in df_columns:
    contingency_table = pd.crosstab(df['Diabetes_binary'], df[x])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    if p == 0:
        print(f"There is a significant association between Diabetes_binary and {x} (reject H0).")
    else:
        print(f"There is no significant association between Diabetes_binary and {x} (fail to reject H0).")
        df_ = df_.drop([x], axis = 1)
```

```
There is a significant association between Diabetes_binary and Diabetes_binary (reject H0).
There is a significant association between Diabetes_binary and HighBP (reject H0).
There is a significant association between Diabetes_binary and HighChol (reject H0).
There is no significant association between Diabetes_binary and CholCheck (fail to reject H0).
There is a significant association between Diabetes_binary and BMI (reject H0).
There is no significant association between Diabetes_binary and Smoker (fail to reject H0).
There is a significant association between Diabetes_binary and Stroke (reject H0).
There is a significant association between Diabetes_binary and HeartDiseaseorAttack (reject H0).
There is a significant association between Diabetes_binary and PhysActivity (reject H0).
There is no significant association between Diabetes_binary and Fruits (fail to reject H0).
There is no significant association between Diabetes_binary and Veggies (fail to reject H0).
There is no significant association between Diabetes_binary and HvyAlcoholConsump (fail to reject H0).
There is no significant association between Diabetes_binary and AnyHealthcare (fail to reject H0).
There is no significant association between Diabetes_binary and NoDocbcCost (fail to reject H0).
There is a significant association between Diabetes_binary and GenHlth (reject H0).
There is no significant association between Diabetes_binary and MentHlth (fail to reject H0).
There is a significant association between Diabetes_binary and PhysHlth (reject H0).
There is a significant association between Diabetes_binary and DiffWalk (reject H0).
There is no significant association between Diabetes_binary and Sex (fail to reject H0).
There is a significant association between Diabetes_binary and Age (reject H0).
There is a significant association between Diabetes_binary and Education (reject H0).
There is a significant association between Diabetes_binary and Income (reject H0).
```

In [15]:
```
print("Dataset shape:", df_.shape)
```

```
Dataset shape: (253680, 13)
```

In [16]:
```
df_.head()
```

Out[16]:

	Diabetes_binary	HighBP	HighChol	BMI	Stroke	HeartDiseaseorAttack	PhysActivity	GenHlth	PhysHlth	DiffWalk	Age	Education	I
0	0.0	1.0	1.0	40.0	0.0	0.0	0.0	5.0	15.0	1.0	9.0	4.0	
1	0.0	0.0	0.0	25.0	0.0	0.0	1.0	3.0	0.0	0.0	7.0	6.0	
2	0.0	1.0	1.0	28.0	0.0	0.0	0.0	5.0	30.0	1.0	9.0	4.0	
3	0.0	1.0	0.0	27.0	0.0	0.0	1.0	2.0	0.0	0.0	11.0	3.0	
4	0.0	1.0	1.0	24.0	0.0	0.0	1.0	2.0	0.0	0.0	11.0	5.0	

Model Performance and Evaluation

In [17]:
```
# Import libraries
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pylab as plt
from tabulate import tabulate
```

In [18]:
```
# Split the target variable, y, and the predictor variables, x
x = df_.drop('Diabetes_binary', axis=1)
y = df_['Diabetes_binary']
```

In [19]:
```
# Standardise the dataset
scaler = preprocessing.MinMaxScaler()
x_minmax = scaler.fit_transform(x)
x_norm = pd.DataFrame(x_minmax, columns=['HighBP','HighChol','BMI','Stroke','HeartDiseaseorAttack','PhysActivity'
x_norm.head()
```

Out[19]:

	HighBP	HighChol	BMI	Stroke	HeartDiseaseorAttack	PhysActivity	GenHlth	PhysHlth	DiffWalk	Age	Education	Income
0	1.0	1.0	0.325581	0.0	0.0	0.0	1.00	0.5	1.0	0.666667	0.6	0.285714
1	0.0	0.0	0.151163	0.0	0.0	1.0	0.50	0.0	0.0	0.500000	1.0	0.000000
2	1.0	1.0	0.186047	0.0	0.0	0.0	1.00	1.0	1.0	0.666667	0.6	1.000000
3	1.0	0.0	0.174419	0.0	0.0	1.0	0.25	0.0	0.0	0.833333	0.4	0.714286
4	1.0	1.0	0.139535	0.0	0.0	1.0	0.25	0.0	0.0	0.833333	0.8	0.428571

In [20]:
```
from imblearn.over_sampling import SMOTE
# Initialize SMOTE
smote = SMOTE(random_state=42)

# Perform oversampling
X_resampled, y_resampled = smote.fit_resample(x_norm, y)
# split data into train test validation set

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)
```

K-NN Classifier

In [ ]:
```
# Measure accuracy of different k values
results = []
for k in range(1,20):
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train,y_train)
    y_predicted_knn_ = knn_classifier.predict(X_test)
    results.append({'k':k, 'accuracy':accuracy_score(y_test,y_predicted_knn_)})
results = pd.DataFrame(results)
print(results)
```

In [ ]:

```
# Plot accuracy vs k
results.plot.line('k', 'accuracy', ylabel='Accuracy', legend=False)

# Optimal k
optimal_k = results.loc[results['accuracy'].idxmax()]
print(f"\n","Optimal:",optimal_k,f"\n")
```

In [21]:

```
# Run kNN with optimal k = 1
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
y_pred_knn = knn.predict(X_test)
```

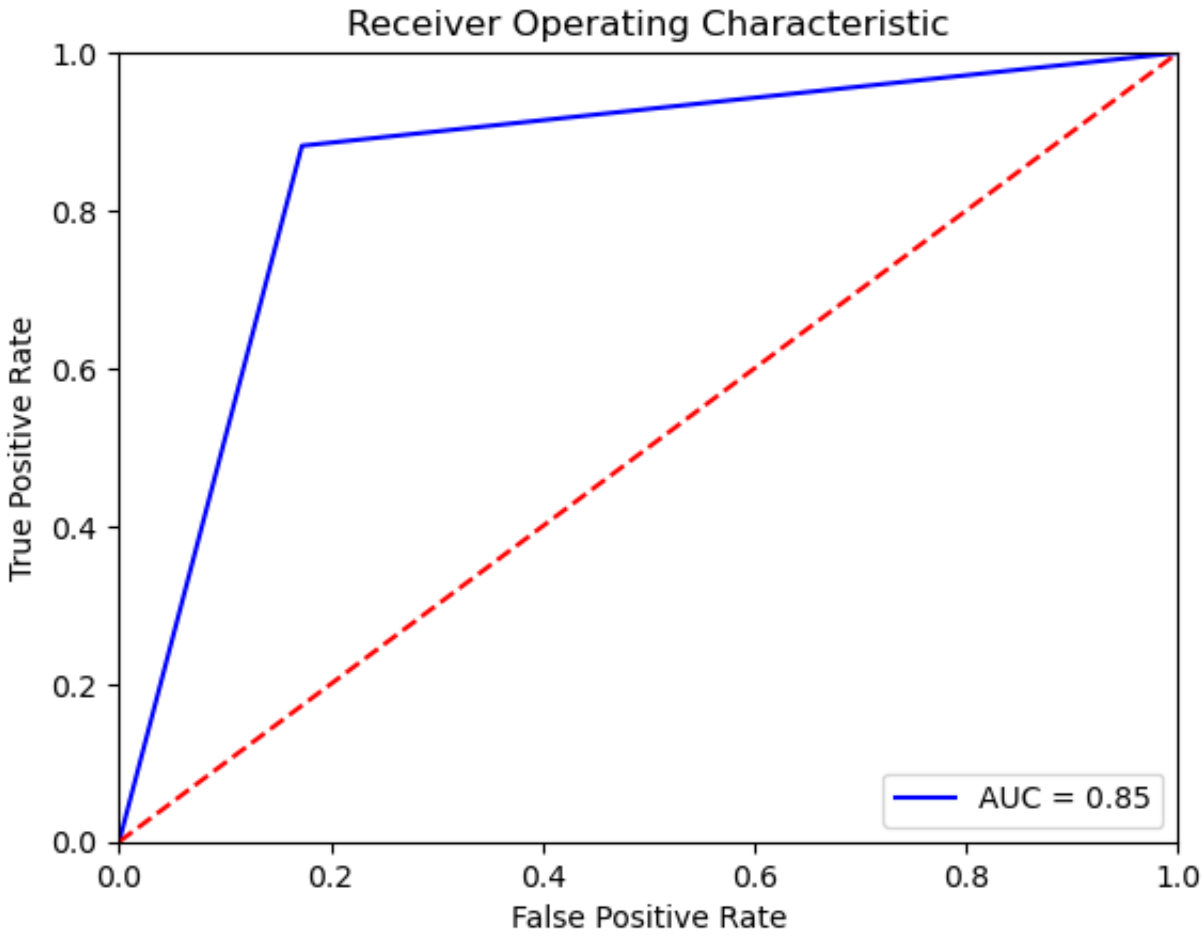
In [22]:

```
# Confusion matrix
c_matrix_knn = confusion_matrix(y_test, y_pred_knn)
cmatrix_list=c_matrix_knn.tolist()
cmatrix_list[0].insert(0,'True 0')
cmatrix_list[1].insert(0,'True 1')
print('Confusion matrix:',f"\n")
print(tabulate(cmatrix_list,headers=['Predicted 0','Predicted 1']),f"\n")

# Classification report
report_knn = classification_report(y_test, y_pred_knn, zero_division=0)
print('Classification report:',f"\n")
print(report_knn,f"\n")
```

```
Confusion matrix:

          Predicted 0    Predicted 1
------  -------------  -------------
True 0          54268          11326
True 1           7707          57700

Classification report:

              precision    recall  f1-score   support

         0.0       0.88      0.83      0.85     65594
         1.0       0.84      0.88      0.86     65407

    accuracy                           0.85    131001
   macro avg       0.86      0.85      0.85    131001
weighted avg       0.86      0.85      0.85    131001
```

In [24]:

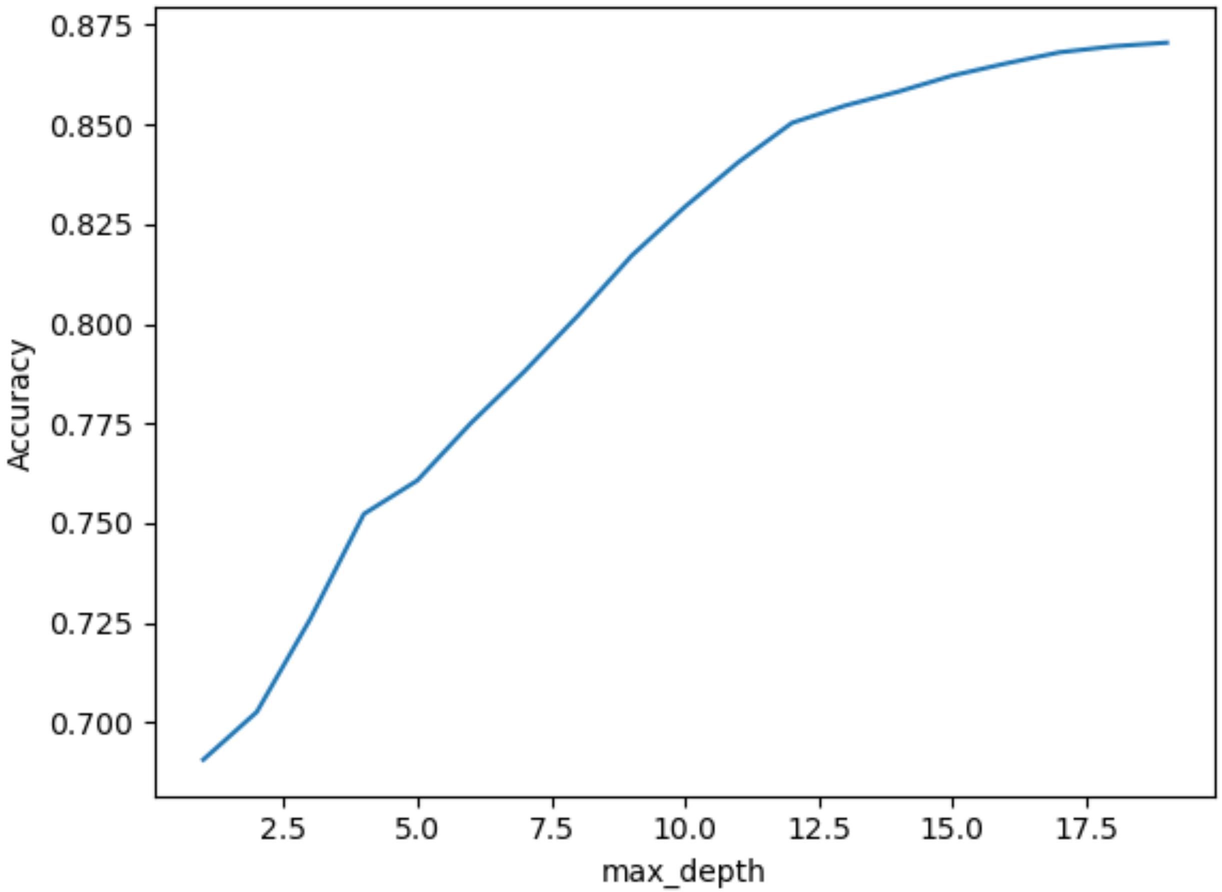
```python
# Plotting ROC
from sklearn.metrics import roc_curve, auc

fpr, tpr, threshold = roc_curve(y_test, y_pred_knn)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Classification Tree

In [25]:
```
# Measure accuracy with different max_depth values
results_tree = []
for k in range(1,20):
    tree_classifier = DecisionTreeClassifier(max_depth=k, random_state=0)
    tree_classifier.fit(X_train, y_train)
    y_predicted_tree_ = tree_classifier.predict(X_test)
    results_tree.append({'max_depth':k, 'accuracy':accuracy_score(y_test,y_predicted_tree_)})
results_tree = pd.DataFrame(results_tree)
print(results_tree)

# Plot accuracy vs max_depth
results_tree.plot.line('max_depth', 'accuracy', ylabel='Accuracy', legend=False)

# Optimal max_depth
optimal_max_depth = results_tree.loc[results_tree['accuracy'].idxmax()]
print(f"\n","Optimal:",optimal_max_depth,f"\n")
```

```
    max_depth  accuracy
0           1  0.690621
1           2  0.702590
2           3  0.725949
3           4  0.752277
4           5  0.760689
5           6  0.775070
6           7  0.788055
7           8  0.802070
8           9  0.817047
9          10  0.829375
10         11  0.840597
11         12  0.850436
12         13  0.854780
13         14  0.858314
14         15  0.862306
15         16  0.865345
16         17  0.868138
17         18  0.869627
18         19  0.870551

 Optimal: max_depth    19.000000
accuracy      0.870551
Name: 18, dtype: float64
```



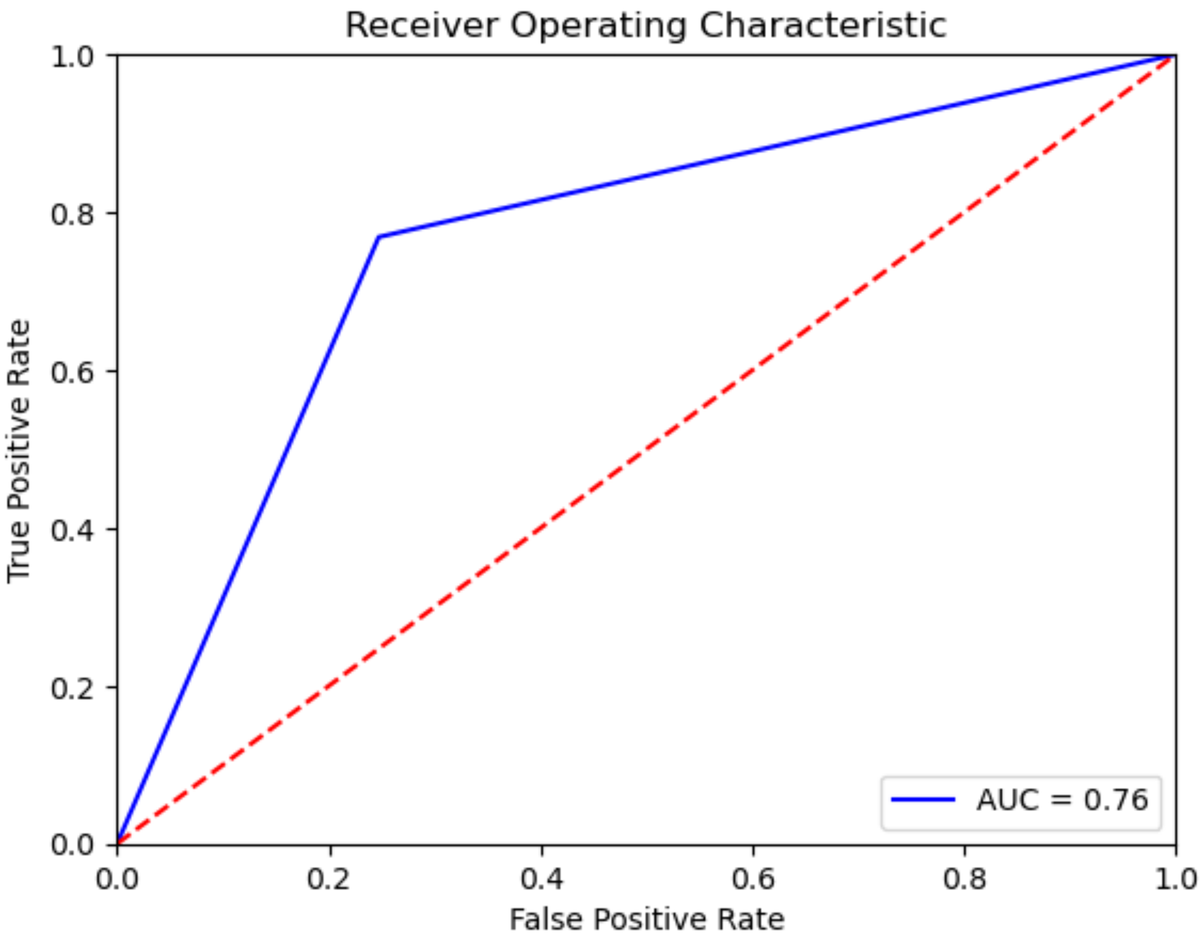
In [26]:
```
# Run tree classifier wtih optiaml max_depth = 5
tree = DecisionTreeClassifier(max_depth=5, random_state=0)
tree.fit(X_train, y_train)
y_pred_tree = tree.predict(X_test)
```

In [27]:
```python
# Confusion matrix
c_matrix_tree = confusion_matrix(y_test, y_pred_tree)
cmatrix_list=c_matrix_tree.tolist()
cmatrix_list[0].insert(0,'True 0')
cmatrix_list[1].insert(0,'True 1')

print('Confusion matrix:',f"\n")
print(tabulate(cmatrix_list,headers=['Predicted 0','Predicted 1']),f"\n")

# Classification report
report_tree = classification_report(y_test, y_pred_tree, zero_division=0)
print('Classification report:',f"\n")
print(report_tree,f"\n")
```

```
Confusion matrix:

          Predicted 0    Predicted 1
------  -------------  -------------
True 0          49385          16209
True 1          15141          50266

Classification report:

              precision    recall  f1-score   support

         0.0       0.77      0.75      0.76     65594
         1.0       0.76      0.77      0.76     65407

    accuracy                           0.76    131001
   macro avg       0.76      0.76      0.76    131001
weighted avg       0.76      0.76      0.76    131001
```

In [28]:

```
from sklearn.metrics import roc_curve, auc

# Plotting ROC

fpr, tpr, threshold = roc_curve(y_test, y_pred_tree)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Naive Bayes

In [29]:
```
# Run Naive Bayes
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
```

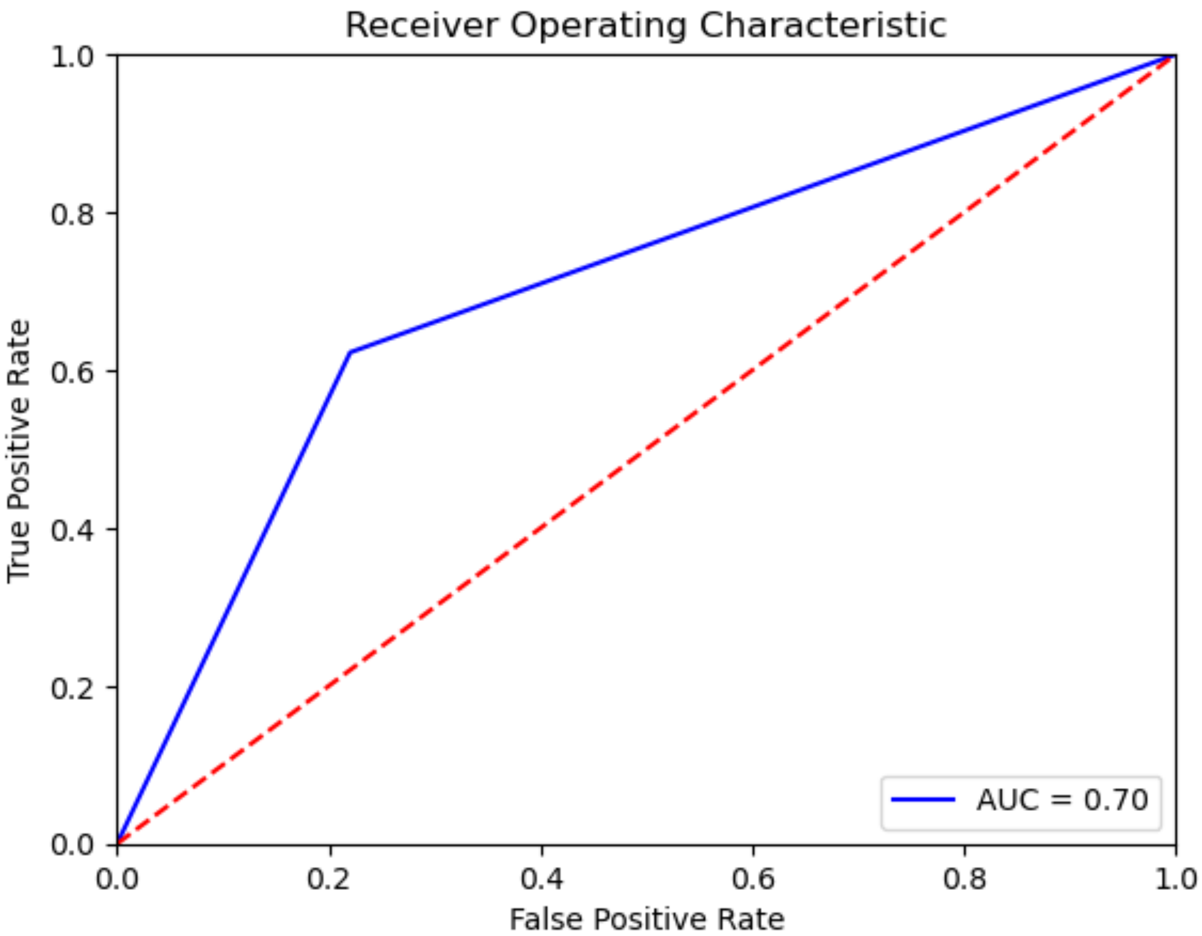
In [30]:

```python
# Confusion matrix
c_matrix_nb = confusion_matrix(y_test, y_pred_nb)
cmatrix_list=c_matrix_nb.tolist()
cmatrix_list[0].insert(0,'True 0')
cmatrix_list[1].insert(0,'True 1')

print('Confusion matrix:',f"\n")
print(tabulate(cmatrix_list,headers=['Predicted 0','Predicted 1']),f"\n")

# Classification report
report_nb = classification_report(y_test, y_pred_nb, zero_division=0)
print('Classification report:',f"\n")
print(report_nb,f"\n")
```

```
Confusion matrix:

          Predicted 0    Predicted 1
------  -------------  -------------
True 0          51155          14439
True 1          24699          40708

Classification report:

              precision    recall  f1-score   support

         0.0       0.67      0.78      0.72     65594
         1.0       0.74      0.62      0.68     65407

    accuracy                           0.70    131001
   macro avg       0.71      0.70      0.70    131001
weighted avg       0.71      0.70      0.70    131001
```

In [31]:

```
# Plotting ROC
fpr, tpr, threshold = roc_curve(y_test, y_pred_nb)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Logistic Regression

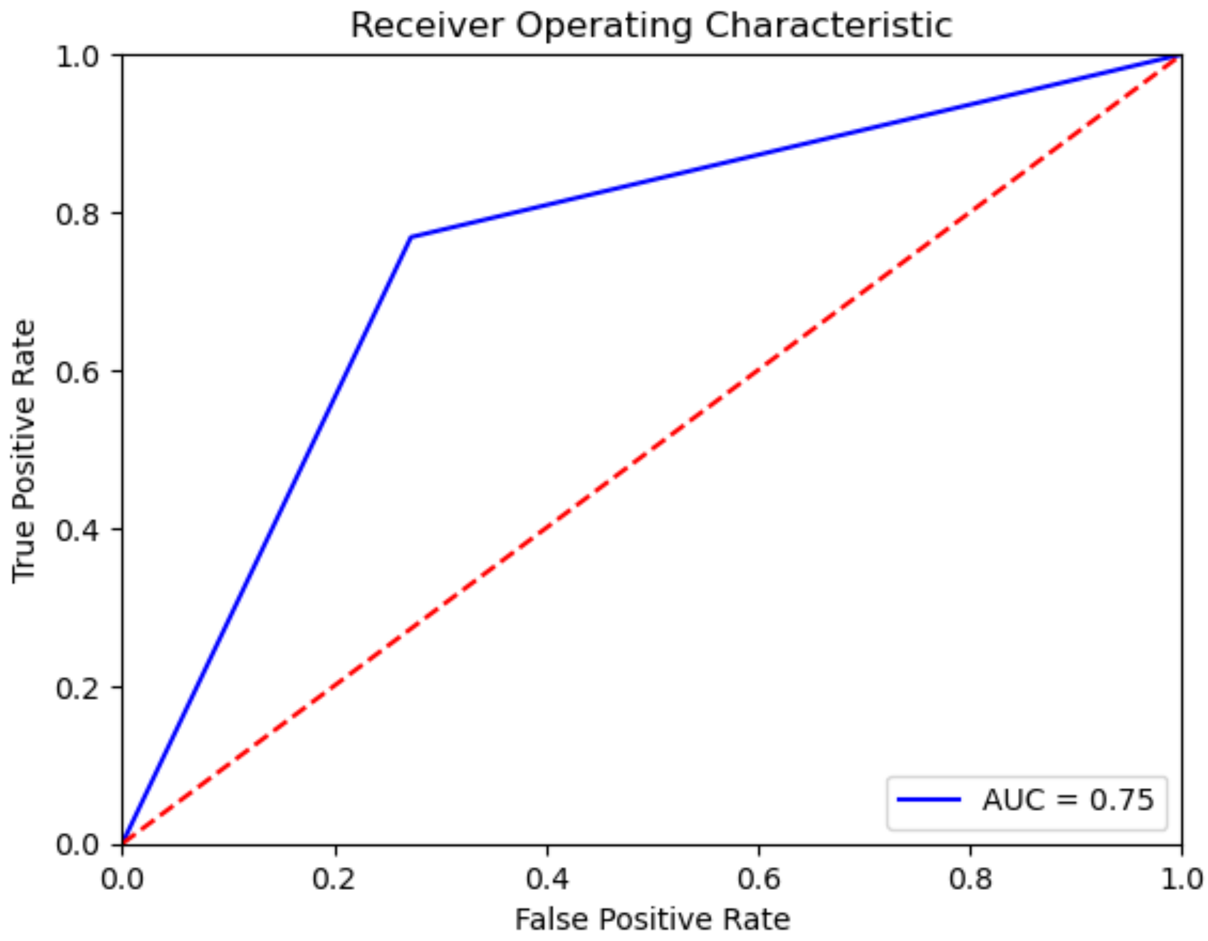
In [32]:
```
# Run logistic regression
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)
```

In [33]:
```python
# Confusion matrix
c_matrix_logreg = confusion_matrix(y_test, y_pred_logreg)
cmatrix_list=c_matrix_logreg.tolist()
cmatrix_list[0].insert(0,'True 0')
cmatrix_list[1].insert(0,'True 1')

print('Confusion matrix:',f"\n")
print(tabulate(cmatrix_list,headers=['Predicted 0','Predicted 1']),f"\n")

# Classification report
report_logreg = classification_report(y_test, y_pred_logreg, zero_division=0)
print('Classification report:',f"\n")
print(report_logreg,f"\n")
```

```
Confusion matrix:

          Predicted 0    Predicted 1
------  -------------  -------------
True 0          47721          17873
True 1          15148          50259

Classification report:

              precision    recall  f1-score   support

         0.0       0.76      0.73      0.74     65594
         1.0       0.74      0.77      0.75     65407

    accuracy                           0.75    131001
   macro avg       0.75      0.75      0.75    131001
weighted avg       0.75      0.75      0.75    131001
```

In [34]:

```
# Plotting ROC
fpr, tpr, threshold = roc_curve(y_test, y_pred_logreg)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

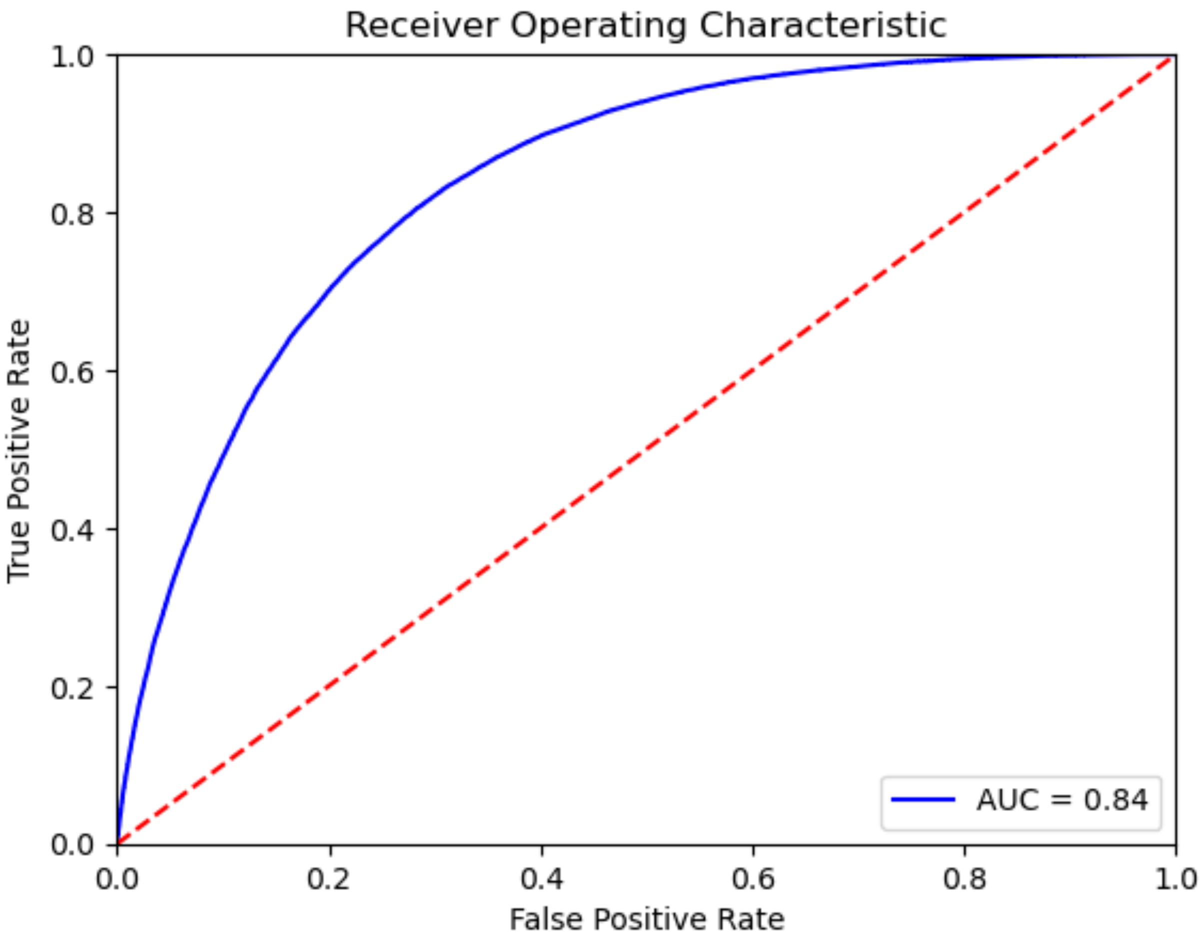


Neural Networks

In [35]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import roc_curve, auc


model = keras.Sequential([
    layers.Dense(64,activation='relu'),
    layers.Dense(64,activation='relu'),
```

```
    layers.Dense(1,activation='sigmoid')
])
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train,y_train,epochs=12,batch_size=32,verbose=1);

# Generate predictions and convert to binary predictions
y_pred = model.predict(X_test)
y_pred_binary = (y_pred > 0.5).astype("int32").flatten()

# Calculate the confusion matrix
conf_mat = confusion_matrix(y_test, y_pred_binary)

# Generate the classification report
class_report = classification_report(y_test, y_pred_binary, target_names=["0", "1"])

# Formatting the output
print("Confusion matrix:")
print(f"{'':<10}{'Predicted 0':<15}{'Predicted 1'}")
print(f"{'True 0':<10}{conf_mat[0, 0]:<15}{conf_mat[0, 1]}")
print(f"{'True 1':<10}{conf_mat[1, 0]:<15}{conf_mat[1, 1]}\n")
print("Classification report:")
print(class_report)


# Plotting ROC
fpr, tpr, threshold = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
Epoch 1/12
9553/9553 [==============================] - 8s 755us/step - loss: 0.5115 - accuracy: 0.7477
Epoch 2/12
9553/9553 [==============================] - 9s 899us/step - loss: 0.5016 - accuracy: 0.7534
Epoch 3/12
9553/9553 [==============================] - 6s 635us/step - loss: 0.4998 - accuracy: 0.7550
Epoch 4/12
9553/9553 [==============================] - 6s 626us/step - loss: 0.4985 - accuracy: 0.7559
Epoch 5/12
9553/9553 [==============================] - 6s 668us/step - loss: 0.4978 - accuracy: 0.7562
Epoch 6/12
9553/9553 [==============================] - 6s 642us/step - loss: 0.4969 - accuracy: 0.7571
Epoch 7/12
9553/9553 [==============================] - 6s 648us/step - loss: 0.4963 - accuracy: 0.7571
Epoch 8/12
9553/9553 [==============================] - 7s 748us/step - loss: 0.4956 - accuracy: 0.7574
Epoch 9/12
9553/9553 [==============================] - 6s 648us/step - loss: 0.4948 - accuracy: 0.7579
Epoch 10/12
9553/9553 [==============================] - 6s 669us/step - loss: 0.4941 - accuracy: 0.7580
Epoch 11/12
9553/9553 [==============================] - 6s 646us/step - loss: 0.4933 - accuracy: 0.7590
Epoch 12/12
9553/9553 [==============================] - 6s 661us/step - loss: 0.4925 - accuracy: 0.7591
4094/4094 [==============================] - 2s 423us/step
Confusion matrix:
          Predicted 0    Predicted 1
True 0    46296          19298
True 1    12076          53331

Classification report:
              precision    recall  f1-score   support

           0       0.79      0.71      0.75     65594
           1       0.73      0.82      0.77     65407

    accuracy                           0.76    131001
   macro avg       0.76      0.76      0.76    131001
weighted avg       0.76      0.76      0.76    131001
```



Random Forest Classifier

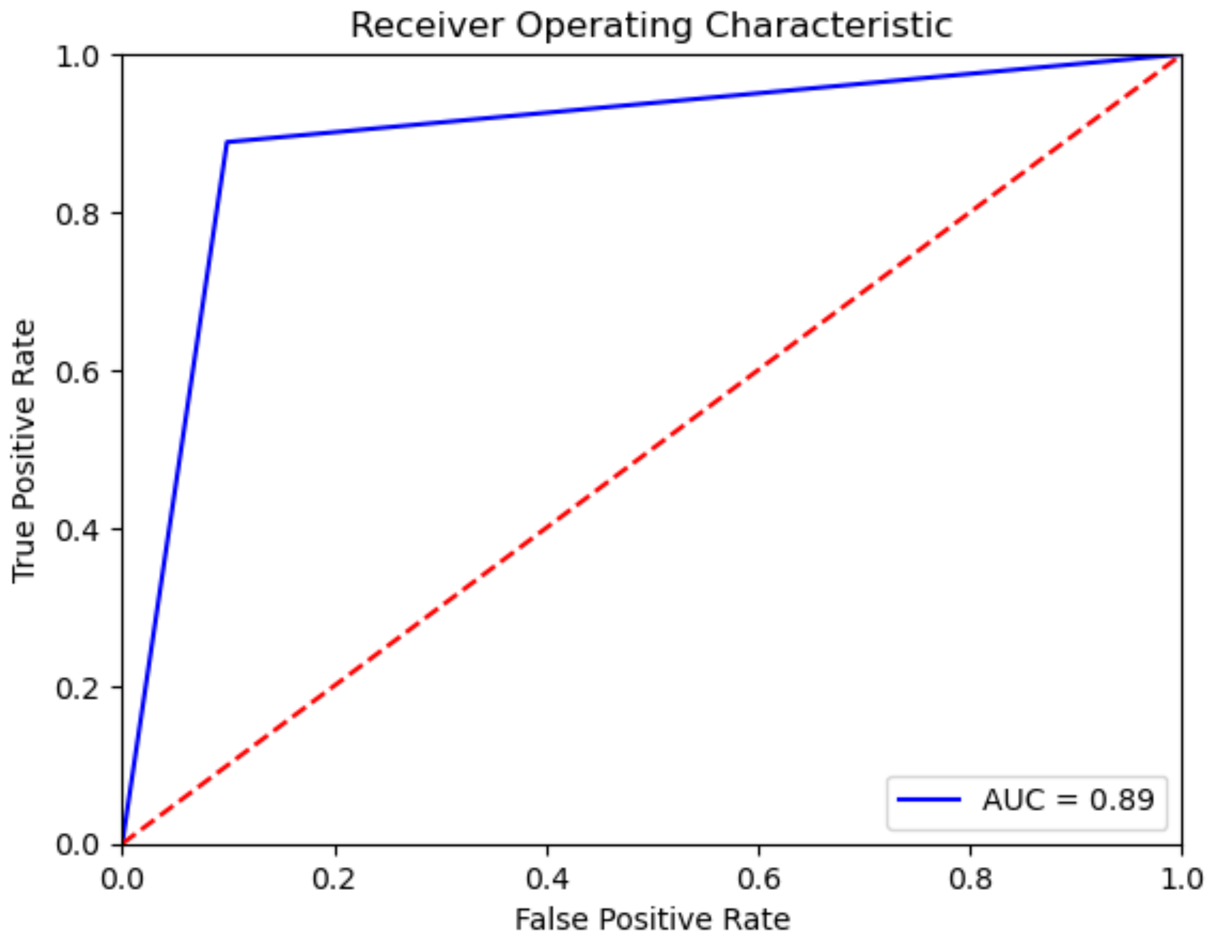
In [36]:
```
# Run random forest classifier
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

In [37]:
```python
# Confusion matrix
c_matrix_rf = confusion_matrix(y_test, y_pred_rf)
cmatrix_list=c_matrix_rf.tolist()
cmatrix_list[0].insert(0,'True 0')
cmatrix_list[1].insert(0,'True 1')

print('Confusion matrix:',f"\n")
print(tabulate(cmatrix_list,headers=['Predicted 0','Predicted 1']),f"\n")

# Classification report
report_rf = classification_report(y_test, y_pred_rf, zero_division=0)
print('Classification report:',f"\n")
print(report_rf,f"\n")
```

```
Confusion matrix:

          Predicted 0    Predicted 1
------  -------------  -------------
True 0          59107           6487
True 1           7270          58137

Classification report:

              precision    recall  f1-score   support

         0.0       0.89      0.90      0.90     65594
         1.0       0.90      0.89      0.89     65407

    accuracy                           0.89    131001
   macro avg       0.90      0.89      0.89    131001
weighted avg       0.90      0.89      0.89    131001
```

In [38]:

```
# Plotting ROC
fpr, tpr, threshold = roc_curve(y_test, y_pred_rf)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Summary of Model Performances

In [2]:
```
# Dictionary to hold model metrics
model_metrics = {
    'Model': ['k-NN', 'Naïve Bayes', 'Classification Tree', 'Logistic Regression', 'Neural Network', 'Random Fore
    'Accuracy': [],
    'Precision': [],
    'Recall': [],
    'F1 Score': [],
}
```

In [3]:
```python
def compute_metrics(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    return accuracy, precision, recall, f1
```

In [ ]:
```python
for model in [knn, tree, nb, logreg, model, rf]:
    y_pred = model.predict(X_test).ravel()
    # Compute metrics
    accuracy, precision, recall, f1 = compute_metrics(y_test, y_pred)

    # Store computed metrics
    model_metrics['Accuracy'].append(accuracy)
    model_metrics['Precision'].append(precision)
    model_metrics['Recall'].append(recall)
    model_metrics['F1 Score'].append(f1)
```

In [7]:
```python
# Get the maximum length of the lists in the dictionary
max_length = max(len(v) for v in model_metrics.values())

# Fill in the shorter lists with None or np.nan
for key in model_metrics:
    while len(model_metrics[key]) < max_length:
        model_metrics[key].append(None)  # or np.nan if you prefer

# Now create the DataFrame
metrics_df = pd.DataFrame(model_metrics)

print(metrics_df)
```

In [ ]: