CSM148 Final Project

Prithvi Kannan

UID: 405110096

In [1]:

```
# relevant imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_sco
re, accuracy_score
import sklearn.metrics.cluster as smc
from sklearn.model_selection import KFold
from sklearn.utils import resample
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.utils._testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from matplotlib import pyplot
import itertools

%matplotlib inline

import random

random.seed(42)
```

Load data

In [2]:

```
data = pd.read_csv(os.getcwd() + "/healthcare-dataset-stroke-data.csv")
```

Data Fields

- id: unique identifier
- gender: "Male", "Female" or "Other"
- age: age of the patient
- hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension 5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
- ever_married: "No" or "Yes"
- work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
- Residence_type: "Rural" or "Urban"
- avg_glucose_level: average glucose level in blood
- bmi: body mass index
- smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
- stroke: 1 if the patient had a stroke or 0 if not

Basic Statistics

In [3]:

```
# first 5 rows
data.head()
```

Out[3]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	a
0	9046	Male	67.0	0	1	Yes	Private	Urban	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	
2	31112	Male	80.0	0	1	Yes	Private	Rural	
3	60182	Female	49.0	0	0	Yes	Private	Urban	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	

In [4]:

```
# summary info to understand the different data types in the table
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5110 entries, 0 to 5109
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   id                 5110 non-null   int64  
 1   gender             5110 non-null   object 
 2   age                5110 non-null   float64
 3   hypertension       5110 non-null   int64  
 4   heart_disease      5110 non-null   int64  
 5   ever_married       5110 non-null   object 
 6   work_type          5110 non-null   object 
 7   Residence_type     5110 non-null   object 
 8   avg_glucose_level  5110 non-null   float64
 9   bmi                4909 non-null   float64
 10  smoking_status     5110 non-null   object 
 11  stroke             5110 non-null   int64  
dtypes: float64(3), int64(4), object(5)
memory usage: 479.2+ KB
```

- there are only 4909 non-null values for BMI, so we will need to impute missing data
- we will need to encode boolean data such as `gender` , `ever_married` , `residence_type`
- we will need to encode categorical data such as `work_type` , `smoking_status`

In [5]:

```
# statistics for the numerical features in the table
data.describe()
```

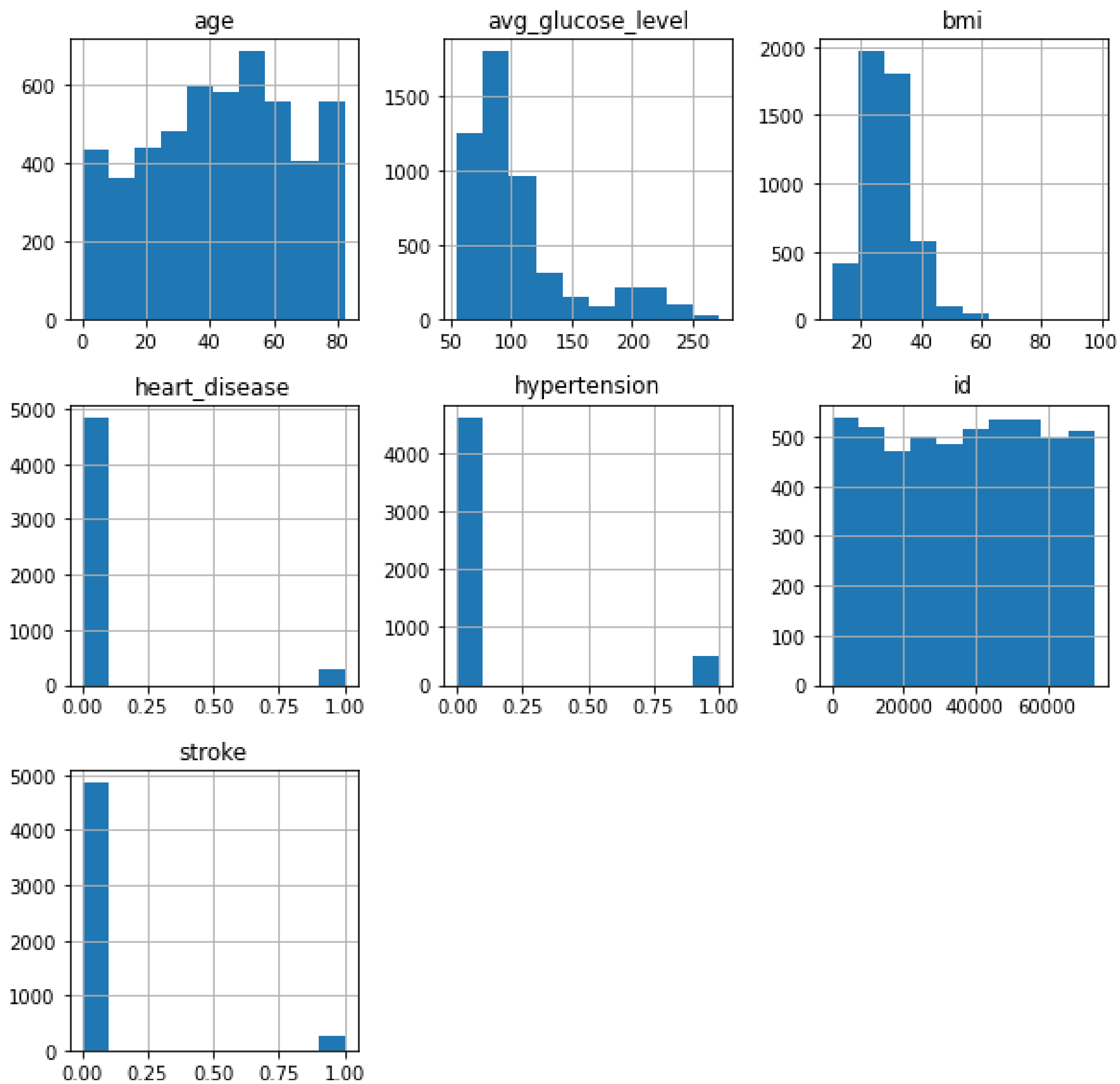
Out[5]:

	id	age	hypertension	heart_disease	avg_glucose_level	bmi	
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	

- numerical features are on very difference scales (glucose mean is 106 and BMI mean is 29)

In [6]:

```
# distributions of different features
data.hist(figsize = (10,10))
plt.show()
```



- `age` is normally distributed with mean around 43
- `avg_glucose_level` and `bmi` follow a normal curve but are heavily skewed to the right side with many large outliers
- binary features such as `heart_disease` and `hypertension` have significant class imbalance
- target variable `stroke` is imbalanced

In [7]:

```
data['work_type'].value_counts()
```

Out[7]:

```
Private          2925
Self-employed     819
children          687
Govt_job          657
Never_worked       22
Name: work_type, dtype: int64
```

- there is heavy class imbalance amongst 5 different categories

In [8]:

```
data['smoking_status'].value_counts()
```

Out[8]:

```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

- "Unknown" category may need to be imputed

In [9]:

```
data['stroke'].value_counts()
```

Out[9]:

```
0    4861
1     249
Name: stroke, dtype: int64
```

In [10]:

```
# analyze correlations with inputs and target output stroke
data.corr()['stroke']
```

Out[10]:

```
id                   0.006388
age                  0.245257
hypertension         0.127904
heart_disease        0.134914
avg_glucose_level    0.131945
bmi                  0.042374
stroke               1.000000
Name: stroke, dtype: float64
```

Data Pipeline

In [11]:

```
# data before pipeline
data.head()
```

Out[11]:

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	a
0	9046	Male	67.0	0	1	Yes	Private	Urban	
1	51676	Female	61.0	0	0	Yes	Self-employed	Rural	
2	31112	Male	80.0	0	1	Yes	Private	Rural	
3	60182	Female	49.0	0	0	Yes	Private	Urban	
4	1665	Female	79.0	1	0	Yes	Self-employed	Rural	

In [12]:

```
# upsample minority class
print("Before balancing: ")
print(data['stroke'].value_counts())

df_minority = data[data.stroke == 1]
df_majority = data[data.stroke == 0]
df_minority_upsampled = resample(df_minority, replace=True, n_samples=2000, random_
state=123)
balanced = pd.concat([df_majority, df_minority_upsampled])
print("After balancing: ")
print(balanced['stroke'].value_counts())

data = balanced
```

```
Before balancing: 
0    4861
1     249
Name: stroke, dtype: int64
After balancing: 
0    4861
1    2000
Name: stroke, dtype: int64
```

In [13]:

```
# run cleanup in pipeline
def run_pipeline(data, should_scale=True, verbose=False):
    processed_data = data

    # drop useless variables
    processed_data.drop(columns=['id'], inplace=True)

    # impute bad values
    imputer = SimpleImputer(missing_values="Other", strategy="most_frequent")
    processed_data['gender'] = imputer.fit_transform(processed_data[['gender']]).ra
vel()
    imputer = SimpleImputer(missing_values=np.nan, strategy="median")
    processed_data['bmi'] = imputer.fit_transform(processed_data[['bmi']]).ravel()
    imputer = SimpleImputer(missing_values="Unknown", strategy="most_frequent")
    processed_data['smoking_status'] = imputer.fit_transform(processed_data[['smoki
ng_status']]).ravel()

    # encode binary variables
    binary_replace = {
            "gender": {"Male": 1, "Female": 0},
            "ever_married": {"Yes": 1, "No": 0},
            "Residence_type": {"Rural": 1, "Urban": 0},
        }
    processed_data = processed_data.replace(binary_replace)

    # one hot encoding for categorical variables
    categoricals = ['work_type', 'smoking_status']
    one_hot_work = pd.get_dummies(processed_data['work_type'],prefix='work_type',dr
op_first=True)
    one_hot_smoking = pd.get_dummies(processed_data['smoking_status'],prefix='smoki
ng_status',drop_first=True)
    processed_data = pd.concat([processed_data, one_hot_work, one_hot_smoking], axi
s=1)
    processed_data.drop(columns=categoricals, inplace=True)

    # augment feature
    processed_data['prexisting'] = processed_data['heart_disease'] | processed_data
['hypertension']

    if verbose:
        print(processed_data.info())

    # scale data appropriately
    if should_scale:
        scaler = StandardScaler()
        scaler.fit(processed_data)
        processed_data = scaler.transform(processed_data)

    return processed_data
```

In [14]:

```
# train-test split
labels = data['stroke']
processed_data = data.drop(columns=['stroke'])

# with scaling
processed_data = run_pipeline(processed_data, should_scale=True, verbose=True)
x_train, x_test, y_train, y_test = train_test_split(processed_data, labels, test_si
ze = 0.2, stratify=labels)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6861 entries, 249 to 23
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   gender                       6861 non-null   int64  
 1   age                          6861 non-null   float64
 2   hypertension                 6861 non-null   int64  
 3   heart_disease                6861 non-null   int64  
 4   ever_married                 6861 non-null   int64  
 5   Residence_type               6861 non-null   int64  
 6   avg_glucose_level            6861 non-null   float64
 7   bmi                          6861 non-null   float64
 8   work_type_Never_worked       6861 non-null   uint8  
 9   work_type_Private            6861 non-null   uint8  
 10  work_type_Self-employed      6861 non-null   uint8  
 11  work_type_children           6861 non-null   uint8  
 12  smoking_status_never smoked  6861 non-null   uint8  
 13  smoking_status_smokes        6861 non-null   uint8  
 14  prexisting                   6861 non-null   int64  
dtypes: float64(3), int64(6), uint8(6)
memory usage: 576.2 KB
None
```

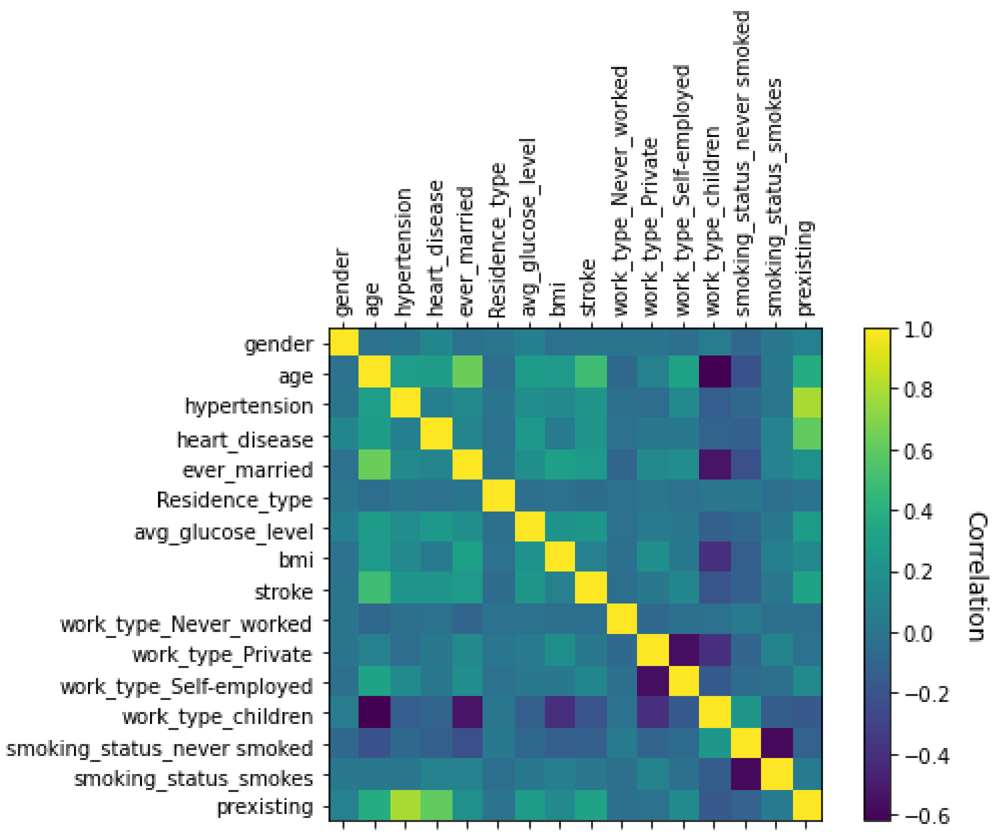
In [15]:

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(5488, 15)
(5488,)
(1373, 15)
(1373,)
```

In [16]:

```
corr_data = run_pipeline(data, should_scale=False, verbose=False)
fig, ax = plt.subplots(figsize=(8,6))
im = ax.matshow(corr_data.corr())
ax.set_xticks(np.arange(corr_data.shape[1]))
ax.set_yticks(np.arange(corr_data.shape[1]))
ax.set_xticklabels(corr_data.columns,rotation=90)
ax.set_yticklabels(corr_data.columns)
cbar = ax.figure.colorbar(im, ax=ax)
cbar.ax.set_ylabel("Correlation", rotation=-90, va="bottom", fontsize=12)
fig.tight_layout()
plt.show()
```



Logistic Regression

In [17]:

```
# run logreg
logreg = LogisticRegression(max_iter=1000)
logreg.fit(x_train, y_train)
y_predictions = logreg.predict(x_test)
```

In [18]:

```
print(f"Accuracy: {str(accuracy_score(y_test, y_predictions))}")
print(f"Precision: {str(precision_score(y_test, y_predictions))}")
print(f"Recall: {str(recall_score(y_test, y_predictions))}")
print(f"F1 Score: {str (f1_score(y_test, y_predictions))}")
print(f"Confusion matrix: {confusion_matrix(y_test, y_predictions)}")
```

```
Accuracy: 0.7705753823743627
Precision: 0.6115485564304461
Recall: 0.5825
F1 Score: 0.5966709346991037
Confusion matrix: [[825 148]
 [167 233]]
```

In [19]:

```
# calculate p-values
import statsmodels.api as sm
model = sm.Logit(y_train, x_train)
model_fit = model.fit()
for var, p in enumerate(model_fit.pvalues):
    if p > 0.05:
        print(f"x{str(var)}\tp-value: {p}")
```

```
Optimization terminated successfully.
         Current function value: 0.555540
         Iterations 6
x0      p-value: 0.05757598750158335
x2      p-value: 0.1957027739287156
x3      p-value: 0.7140493851769003
x7      p-value: 0.24018643390617855
x8      p-value: 0.18944597970736077
x10     p-value: 0.3892541830359826
x12     p-value: 0.058630039464205384
x13     p-value: 0.7948260375139753
```

The following features are significant to the logistic regression model

- gender
- hypertension
- heart_disease
- bmi
- work_type_Never_worked
- work_type_Self_employed
- smoking_status_smokes

PCA

In [25]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6861 entries, 249 to 23
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   gender             6861 non-null   object 
 1   age                6861 non-null   float64
 2   hypertension       6861 non-null   int64  
 3   heart_disease      6861 non-null   int64  
 4   ever_married       6861 non-null   object 
 5   work_type          6861 non-null   object 
 6   Residence_type     6861 non-null   object 
 7   avg_glucose_level  6861 non-null   float64
 8   bmi                6861 non-null   float64
 9   smoking_status     6861 non-null   object 
 10  stroke             6861 non-null   int64  
dtypes: float64(3), int64(3), object(5)
memory usage: 643.2+ KB
```

In [33]:

```
data = pd.read_csv(os.getcwd() + "/healthcare-dataset-stroke-data.csv")
labels = data['stroke']
processed_data = data.drop(columns=['stroke'])
processed_data = run_pipeline(processed_data, should_scale=True)
```

In [34]:

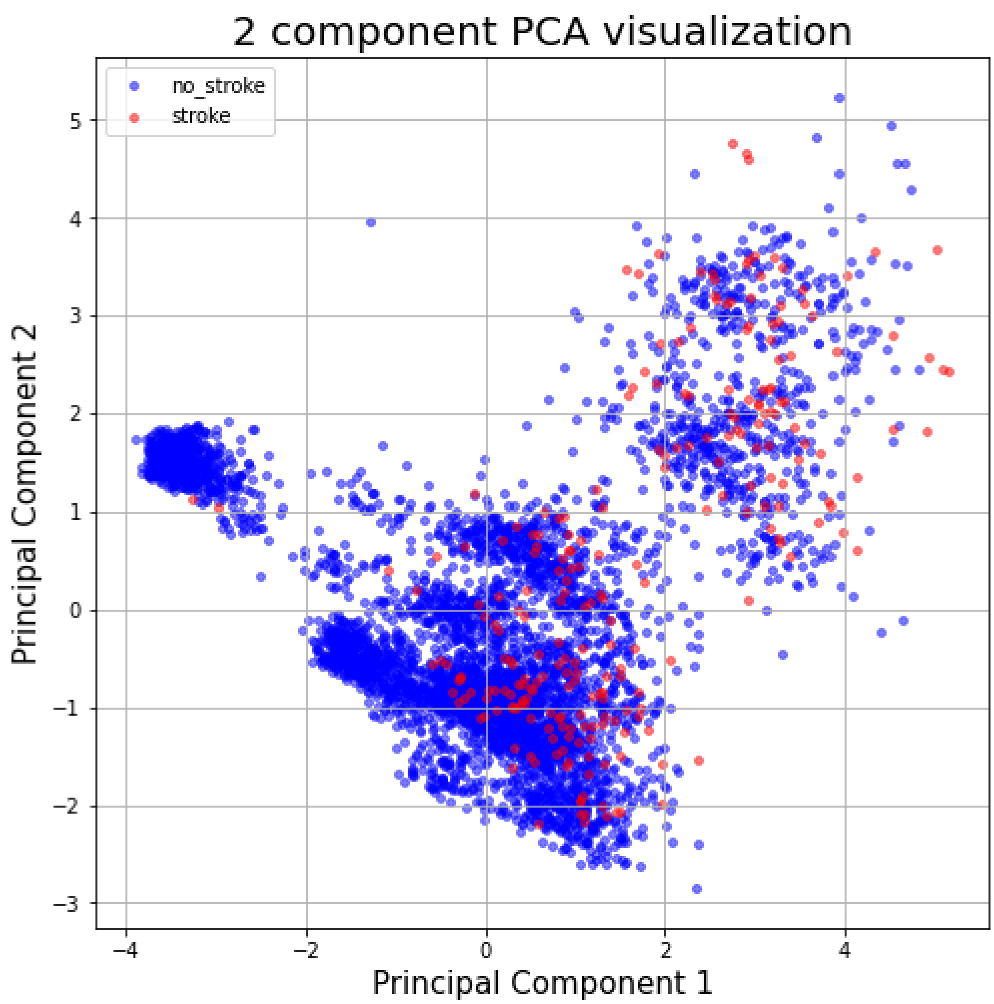
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(processed_data)
principalDf = pd.DataFrame(data = principalComponents, columns = ['principal compon
ent 1', 'principal component 2'])
labels.reset_index(drop=True, inplace=True)
principalDf.reset_index(drop=True, inplace=True)
finalDf = pd.concat([principalDf, labels], axis = 1)
print(finalDf)
```

```
      principal component 1  principal component 2  stroke
0                  3.723643               1.593191       1
1                  0.868349               0.957855       1
2                  2.800959               1.831385       1
3                  1.498661              -2.064928       1
4                  2.903718               3.531388       1
...                     ...                    ...     ...
5105               2.538604               1.540303       0
5106               1.398896               0.414359       0
5107              -0.058939               0.586210       0
5108               0.747184              -0.948240       0
5109              -0.459314              -0.060918       0

[5110 rows x 3 columns]
```

In [35]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA visualization', fontsize = 20)
targets = [0,1]
colors = ['b', 'r']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['stroke'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 15, alpha=0.5)
ax.legend(['no_stroke', 'stroke'])
ax.grid()
```



Ensemble

In [36]:

```
# train model and generate predictions
rfr = RandomForestClassifier(n_estimators=10)
rfr = rfr.fit(x_train, y_train)
y_predictions = rfr.predict(x_test)
```

In [37]:

```
print(f"Accuracy: {str(accuracy_score(y_test, y_predictions))}")
print(f"Precision: {str(precision_score(y_test, y_predictions))}")
print(f"Recall: {str(recall_score(y_test, y_predictions))}")
print(f"F1 Score: {str (f1_score(y_test, y_predictions))}")
print(f"Confusion matrix: {confusion_matrix(y_test, y_predictions)}")
```

```
Accuracy: 0.9796067006554989
Precision: 0.9428571428571428
Recall: 0.99
F1 Score: 0.9658536585365853
Confusion matrix: [[949  24]
 [  4 396]]
```

Neural Network

In [38]:

```
# train multi layer preceptron model
mlp = MLPClassifier(solver='lbfgs', alpha=1e-5, max_iter=1000)
mlp = mlp.fit(x_train, y_train)
y_predictions = mlp.predict(x_test)
```

In [39]:

```
print(f"Accuracy: {str(accuracy_score(y_test, y_predictions))}")
print(f"Precision: {str(precision_score(y_test, y_predictions))}")
print(f"Recall: {str(recall_score(y_test, y_predictions))}")
print(f"F1 Score: {str (f1_score(y_test, y_predictions))}")
print(f"Confusion matrix: {confusion_matrix(y_test, y_predictions)}")
```

```
Accuracy: 0.9519300801165331
Precision: 0.8583690987124464
Recall: 1.0
F1 Score: 0.9237875288683602
Confusion matrix: [[907  66]
 [  0 400]]
```

Cross Validation

In [40]:

```
# generate datasets
labels = data['stroke']
processed_data = data.drop(columns=['stroke'])

# run pipeline
processed_data = run_pipeline(processed_data, should_scale=True, verbose=False)
```

In [42]:

```
@ignore_warnings(category=ConvergenceWarning) # suppress convergence warning messag
es from NN models
def run_kfold(X_kfold, Y_kfold):
    # set up kfold
    kf = KFold(n_splits=10, shuffle=True)

    # prep data
    X = X_kfold
    y = Y_kfold

    # train logistic model
    rfr_scores = []
    nn_scores = []

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        model = RandomForestClassifier(n_estimators=10).fit(X_train, y_train)
        y_predictions = model.predict(X_test)
        rfr_scores.append(f1_score(y_test, y_predictions))
        model = MLPClassifier(solver='lbfgs', alpha=1e-5, max_iter=100).fit(X_train
, y_train)
        y_predictions = model.predict(X_test)
        nn_scores.append(f1_score(y_test, y_predictions))

    print(f"Average F1 RFR: {sum(rfr_scores)/len(rfr_scores)}")
    print(f"Average F1 NN: {sum(nn_scores)/len(nn_scores)}")

run_kfold(processed_data, labels)
```

```
Average F1 RFR: 0.045742798575898l5
Average F1 NN: 0.11008356525613774
```

SVM

In [43]:

```
# run SVC with large regularization param and rbf kernel
svc = SVC(C=10000, kernel='rbf').fit(x_train, y_train)
y_predictions = svc.predict(x_test)
```

In [44]:

```
print(f"Accuracy: {str(accuracy_score(y_test, y_predictions))}")
print(f"Precision: {str(precision_score(y_test, y_predictions))}")
print(f"Recall: {str(recall_score(y_test, y_predictions))}")
print(f"F1 Score: {str (f1_score(y_test, y_predictions))}")
print(f"Confusion matrix: {confusion_matrix(y_test, y_predictions)}")
```

```
Accuracy: 0.9147851420247632
Precision: 0.8029978586723768
Recall: 0.9375
F1 Score: 0.8650519031141869
Confusion matrix: [[881  92]
 [ 25 375]]
```

KNN

In [45]:

```
# run KNN
knn = KNeighborsClassifier(n_neighbors=5).fit(x_train, y_train)
y_predictions = knn.predict(x_test)
```

In [46]:

```
print(f"Accuracy: {str(accuracy_score(y_test, y_predictions))}")
print(f"Precision: {str(precision_score(y_test, y_predictions))}")
print(f"Recall: {str(recall_score(y_test, y_predictions))}")
print(f"F1 Score: {str (f1_score(y_test, y_predictions))}")
print(f"Confusion matrix: {confusion_matrix(y_test, y_predictions)}")
```

```
Accuracy: 0.8892935178441369
Precision: 0.7366412213740458
Recall: 0.965
F1 Score: 0.8354978354978355
Confusion matrix: [[835 138]
 [ 14 386]]
```

In [ ]:

```

```