RECENTManual_PCA_and_sklearn_PCA_1

September 24, 2021

[1]:
```
import numpy as np
import pandas as pd

#plotting
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, OneHotEncoder#to scale our␣
↪data
from sklearn.decomposition import PCA #for comparing our manual PCA

#used for classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import tree
from sklearn import metrics
import time
```

1 Data Pre-processing

[2]:
```
#load in data
heart_stroke_df = pd.read_csv("healthcare-dataset-stroke-data.csv")
```

[3]:
```
heart_stroke_df
```

[3]:
```
         id  gender   age  hypertension  heart_disease ever_married  \
0      9046    Male  67.0             0              1          Yes
1     51676  Female  61.0             0              0          Yes
2     31112    Male  80.0             0              1          Yes
3     60182  Female  49.0             0              0          Yes
4      1665  Female  79.0             1              0          Yes
…       …     …   …           …              …          …
5105  18234  Female  80.0             1              0          Yes
5106  44873  Female  81.0             0              0          Yes
5107  19723  Female  35.0             0              0          Yes
```

```
5108  37544    Male  51.0              0              0          Yes
5109  44679  Female  44.0              0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
0           Private          Urban             228.69  36.6  formerly smoked
1     Self-employed          Rural             202.21   NaN     never smoked
2           Private          Rural             105.92  32.5     never smoked
3           Private          Urban             171.23  34.4           smokes
4     Self-employed          Rural             174.12  24.0     never smoked
…               …            …                  …    …              …
5105        Private          Urban              83.75   NaN     never smoked
5106  Self-employed          Urban             125.20  40.0     never smoked
5107  Self-employed          Rural              82.99  30.6     never smoked
5108        Private          Rural             166.29  25.6  formerly smoked
5109       Govt_job          Urban              85.28  26.2          Unknown

      stroke
0          1
1          1
2          1
3          1
4          1
…        …
5105       0
5106       0
5107       0
5108       0
5109       0

[5110 rows x 12 columns]
```

[4]:
```python
#checking for any null values
heart_stroke_df.isnull().sum()
```

[4]:
```
id                     0
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

[5]:
```python
#looking over positive/negative classes
heart_stroke_df['stroke'].value_counts()
```

[5]:
```
0    4861
1     249
Name: stroke, dtype: int64
```

[6]:
```python
heart_stroke_df.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
```

[7]:
```python
#drop null rows
heart_stroke_df = heart_stroke_df.dropna()
heart_stroke_df
```

[7]:
```
         id  gender   age  hypertension  heart_disease ever_married  \
0      9046    Male  67.0             0              1          Yes
2     31112    Male  80.0             0              1          Yes
3     60182  Female  49.0             0              0          Yes
4      1665  Female  79.0             1              0          Yes
5     56669    Male  81.0             0              0          Yes
…       …      …    …             …              …            …
5104  14180  Female  13.0             0              0           No
5106  44873  Female  81.0             0              0          Yes
5107  19723  Female  35.0             0              0          Yes
5108  37544    Male  51.0             0              0          Yes
5109  44679  Female  44.0             0              0          Yes

          work_type Residence_type  avg_glucose_level   bmi   smoking_status  \
```

```
0          Private        Urban             228.69  36.6  formerly smoked
2          Private        Rural             105.92  32.5     never smoked
3          Private        Urban             171.23  34.4           smokes
4    Self-employed        Rural             174.12  24.0     never smoked
5          Private        Urban             186.21  29.0  formerly smoked
…              …           …            …    …           …
5104      children        Rural             103.08  18.6          Unknown
5106 Self-employed        Urban             125.20  40.0     never smoked
5107 Self-employed        Rural              82.99  30.6     never smoked
5108       Private        Rural             166.29  25.6  formerly smoked
5109      Govt_job        Urban              85.28  26.2          Unknown

      stroke
0          1
2          1
3          1
4          1
5          1
…        …
5104       0
5106       0
5107       0
5108       0
5109       0

[4909 rows x 12 columns]
```

[8]:

```python
#drop id and stroke columns:
strokeY = heart_stroke_df['stroke']
heart_stroke_df.drop(columns=['id', 'stroke'], axis=1, inplace=True)
heart_stroke_df.head()
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:4906:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[8]:

```
   gender   age  hypertension  heart_disease ever_married      work_type  \
0    Male  67.0             0              1          Yes        Private
2    Male  80.0             0              1          Yes        Private
3  Female  49.0             0              0          Yes        Private
4  Female  79.0             1              0          Yes  Self-employed
5    Male  81.0             0              0          Yes        Private
```

```
  Residence_type  avg_glucose_level   bmi   smoking_status
0          Urban             228.69  36.6  formerly smoked
2          Rural             105.92  32.5     never smoked
3          Urban             171.23  34.4           smokes
4          Rural             174.12  24.0     never smoked
5          Urban             186.21  29.0  formerly smoked
```

[9]:
```python
heart_stroke_df.head()
```

[9]:
```
   gender   age  hypertension  heart_disease ever_married      work_type  \
0    Male  67.0             0              1          Yes        Private
2    Male  80.0             0              1          Yes        Private
3  Female  49.0             0              0          Yes        Private
4  Female  79.0             1              0          Yes  Self-employed
5    Male  81.0             0              0          Yes        Private

  Residence_type  avg_glucose_level   bmi   smoking_status
0          Urban             228.69  36.6  formerly smoked
2          Rural             105.92  32.5     never smoked
3          Urban             171.23  34.4           smokes
4          Rural             174.12  24.0     never smoked
5          Urban             186.21  29.0  formerly smoked
```

[10]:
```python
categoric_features = ["gender", "work_type", "Residence_type", "ever_married",␣
 ↪"smoking_status"]
numeric_features = ["age", "hypertension", "heart_disease",␣
 ↪"avg_glucose_level", "bmi"]

#one hot encoder and scaler
scaler = StandardScaler()
ohe    = OneHotEncoder(sparse=False)

#scaling numeric columns
scaled_columns = pd.DataFrame(scaler.
 ↪fit_transform(heart_stroke_df[numeric_features]),
                                columns=numeric_features,
                                index=heart_stroke_df.index)
encoded_columns = ohe.fit_transform(heart_stroke_df[categoric_features])#turns␣
 ↪to dense matrix

# Concatenate them back together
for index, category in enumerate(np.concatenate(ohe.categories_)):
    scaled_columns[category] = encoded_columns[:, index]

scaled_columns
```

[10]:

```
           age  hypertension  heart_disease  avg_glucose_level       bmi  \
0     1.070138     -0.318067       4.381968           2.777698  0.981345
2     1.646563     -0.318067       4.381968           0.013842  0.459269
3     0.272012     -0.318067      -0.228208           1.484132  0.701207
4     1.602222      3.143994      -0.228208           1.549193 -0.623083
5     1.690903     -0.318067      -0.228208           1.821368  0.013595
…          …            …             …                 …        …
5104 -1.324241     -0.318067      -0.228208          -0.050094 -1.310695
5106  1.690903     -0.318067      -0.228208           0.447882  1.414286
5107 -0.348753     -0.318067      -0.228208          -0.502369  0.217332
5108  0.360692     -0.318067      -0.228208           1.372920 -0.419346
5109  0.050310     -0.318067      -0.228208          -0.450816 -0.342945

      Female  Male  Other  Govt_job  Never_worked  …  Self-employed  \
0        0.0   1.0    0.0       0.0           0.0  …            0.0
2        0.0   1.0    0.0       0.0           0.0  …            0.0
3        1.0   0.0    0.0       0.0           0.0  …            0.0
4        1.0   0.0    0.0       0.0           0.0  …            1.0
5        0.0   1.0    0.0       0.0           0.0  …            0.0
…         …     …      …         …             …  …              …
5104     1.0   0.0    0.0       0.0           0.0  …            0.0
5106     1.0   0.0    0.0       0.0           0.0  …            1.0
5107     1.0   0.0    0.0       0.0           0.0  …            1.0
5108     0.0   1.0    0.0       0.0           0.0  …            0.0
5109     1.0   0.0    0.0       1.0           0.0  …            0.0

      children  Rural  Urban   No  Yes  Unknown  formerly smoked  \
0          0.0    0.0    1.0  0.0  1.0      0.0              1.0
2          0.0    1.0    0.0  0.0  1.0      0.0              0.0
3          0.0    0.0    1.0  0.0  1.0      0.0              0.0
4          0.0    1.0    0.0  0.0  1.0      0.0              0.0
5          0.0    0.0    1.0  0.0  1.0      0.0              1.0
…            …      …      …    …    …        …                …
5104       1.0    1.0    0.0  1.0  0.0      1.0              0.0
5106       0.0    0.0    1.0  0.0  1.0      0.0              0.0
5107       0.0    1.0    0.0  0.0  1.0      0.0              0.0
5108       0.0    1.0    0.0  0.0  1.0      0.0              1.0
5109       0.0    0.0    1.0  0.0  1.0      1.0              0.0

      never smoked  smokes
0              0.0     0.0
2              1.0     0.0
3              0.0     1.0
4              1.0     0.0
5              0.0     0.0
…                …       …
5104           0.0     0.0
```

```
5106          1.0     0.0
5107          1.0     0.0
5108          0.0     0.0
5109          0.0     0.0

[4909 rows x 21 columns]
```

2 Manual PCA

[11]:
```
def eigsort(eigenValues, eigenVectors):
    # [Vsort,Dsort] = eigsort(V, eigvals)
    #
    # Sorts a matrix eigenvectors and a array of eigenvalues in order
    # of eigenvalue size, largest eigenvalue first and smallest eigenvalue
    # last.
    #
    # Example usage:
    # di, V = np.linarg.eig(L)
    # Vnew, Dnew = eigsort(V, di)
    #
    # Tim Marks 2002

    # Sort the eigenvalues from largest to smallest. Store the sorted
    # eigenvalues in the column vector lambd.
    idx = np.argsort(eigenValues)
    idx = np.flip(idx)
    eigenValues = eigenValues[idx]
    eigenVectors = eigenVectors[:,idx]
    return eigenValues, eigenVectors
```

[12]:
```
#features and labels
strokeY
strokeX = scaled_columns
```

[13]:
```
#caluculating covariance matrix
mean = np.mean(scaled_columns.to_numpy().T, axis=1) #calculating mean of each␣
 ↪column
stroke_data = scaled_columns.to_numpy() #converting to np array
centered_data = stroke_data - mean #subtracting the mean
n = len(stroke_data)
cov_matrix = (centered_data.T.dot(centered_data))*1/n
cov_matrix
```

[13]:
```
array([[ 1.00000000e+00,  2.74424873e-01,  2.57122776e-01,
         2.35838155e-01,  3.33397995e-01,  1.49788223e-02,
        -1.48264864e-02, -1.52335855e-04,  4.46819160e-02,
```

```
 -5.30240068e-03,  5.93742976e-02,  1.19357987e-01,
 -2.18111800e-01, -5.47348461e-03,  5.47348461e-03,
 -3.24133005e-01,  3.24133005e-01, -1.78880957e-01,
  9.11617718e-02,  6.02856141e-02,  2.74335710e-02],
[ 2.74424873e-01,  1.00000000e+00,  1.15990991e-01,
  1.80542699e-01,  1.67810584e-01, -1.06868951e-02,
  1.07516877e-02, -6.47925700e-05,  6.43227880e-03,
 -1.42543654e-03, -2.29374318e-03,  4.07627154e-02,
 -4.34758145e-02,  5.37016911e-04, -5.37016911e-04,
 -7.73246886e-02,  7.73246886e-02, -6.57617290e-02,
  2.33458692e-02,  3.23376700e-02,  1.00781898e-02],
[ 2.57122776e-01,  1.15990991e-01,  1.00000000e+00,
  1.54525119e-01,  4.13574429e-02, -4.07620304e-02,
  4.08085180e-02, -4.64876655e-05,  1.70397381e-03,
 -1.02272864e-03, -1.38123846e-04,  2.97109749e-02,
 -3.02540962e-02,  1.18074844e-03, -1.18074844e-03,
 -5.29659034e-02,  5.29659034e-02, -3.41934954e-02,
  2.68287396e-02, -1.00258399e-02,  1.73905957e-02],
[ 2.35838155e-01,  1.80542699e-01,  1.54525119e-01,
  1.00000000e+00,  1.75502176e-01, -2.62420470e-02,
  2.60676665e-02,  1.74380424e-04,  5.92760798e-03,
 -9.34496076e-04,  4.58726249e-03,  2.51774174e-02,
 -3.47577918e-02,  3.80787267e-03, -3.80787267e-03,
 -7.20736289e-02,  7.20736289e-02, -4.73975289e-02,
  2.79234474e-02,  1.55515973e-02,  3.92248423e-03],
[ 3.33397995e-01,  1.67810584e-01,  4.13574429e-02,
  1.75502176e-01,  1.00000000e+00,  1.29642232e-02,
 -1.27957938e-02, -1.68429387e-04,  2.66177914e-02,
 -1.91045371e-03,  1.02911479e-01,  2.65083448e-02,
 -1.54127161e-01,  6.12141930e-05, -6.12141930e-05,
 -1.62687279e-01,  1.62687279e-01, -1.24131284e-01,
  4.02514352e-02,  5.23304019e-02,  3.15494473e-02],
[ 1.49788223e-02, -1.06868951e-02, -4.07620304e-02,
 -2.62420470e-02,  1.29642232e-02,  2.41874680e-01,
 -2.41754464e-01, -1.20216044e-04,  2.48756332e-03,
 -4.03970723e-04,  9.39394806e-03,  4.00103227e-03,
 -1.54785729e-02, -1.13057857e-03,  1.13057857e-03,
 -8.59434746e-03,  8.59434746e-03, -1.30736296e-02,
 -7.32280448e-03,  2.24199809e-02, -2.02354682e-03],
[-1.48264864e-02,  1.07516877e-02,  4.08085180e-02,
  2.60676665e-02, -1.27957938e-02, -2.41754464e-01,
  2.41837914e-01, -8.34499357e-05, -2.46142038e-03,
  4.04883651e-04, -9.48100821e-03, -3.96887230e-03,
  1.55064172e-02,  1.02725169e-03, -1.02725169e-03,
  8.46139191e-03, -8.46139191e-03,  1.31351693e-02,
  7.15382977e-03, -2.23431290e-02,  2.05412992e-03],
[-1.52335855e-04, -6.47925700e-05, -4.64876655e-05,
```

```
  1.74380424e-04, -1.68429387e-04, -1.20216044e-04,
 -8.34499357e-05,  2.03665979e-04, -2.61429436e-05,
 -9.12928188e-07,  8.70601517e-05, -3.21599702e-05,
 -2.78443097e-05,  1.03326872e-04, -1.03326872e-04,
  1.32955542e-04, -1.32955542e-04, -6.15396592e-05,
  1.68974708e-04, -7.68519547e-05, -3.05830943e-05],
[ 4.46819160e-02,  6.43227880e-03,  1.70397381e-03,
  5.92760798e-03,  2.66177914e-02,  2.48756332e-03,
 -2.46142038e-03, -2.61429436e-05,  1.11865655e-01,
 -5.75144758e-04, -7.34878143e-02, -2.02607813e-02,
 -1.75419151e-02, -1.72012269e-03,  1.72012269e-03,
 -2.19621889e-02,  2.19621889e-02, -1.51399181e-02,
  3.78549823e-03,  7.60282445e-03,  3.75159539e-03],
[-5.30240068e-03, -1.42543654e-03, -1.02272864e-03,
 -9.34496076e-04, -1.91045371e-03, -4.03970723e-04,
  4.04883651e-04, -9.12928188e-07, -5.75144758e-04,
  4.46148005e-03, -2.56624114e-03, -7.07519345e-04,
 -6.12574814e-04, -7.82420954e-04,  7.82420954e-04,
  2.92502191e-03, -2.92502191e-03,  2.75787306e-04,
 -7.64120893e-04,  1.16116166e-03, -6.72828074e-04],
[ 5.93742976e-02, -2.29374318e-03, -1.38123846e-04,
  4.58726249e-03,  1.02911479e-01,  9.39394806e-03,
 -9.48100821e-03,  8.70601517e-05, -7.34878143e-02,
 -2.56624114e-03,  2.44726086e-01, -9.04016764e-02,
 -7.82703546e-02,  4.24283375e-03, -4.24283375e-03,
 -3.69362445e-02,  3.69362445e-02, -4.87264216e-02,
  4.62734251e-03,  2.65847593e-02,  1.75143198e-02],
[ 1.19357987e-01,  4.07627154e-02,  2.97109749e-02,
  2.51774174e-02,  2.65083448e-02,  4.00103227e-03,
 -3.96887230e-03, -3.21599702e-05, -2.02607813e-02,
 -7.07519345e-04, -9.04016764e-02,  1.32949317e-01,
 -2.15793400e-02, -2.21949441e-03,  2.21949441e-03,
 -3.32397568e-02,  3.32397568e-02, -1.79519444e-02,
  1.32124777e-02,  5.21871249e-03, -4.79245802e-04],
[-2.18111800e-01, -4.34758145e-02, -3.02540962e-02,
 -3.47577918e-02, -1.54127161e-01, -1.54785729e-02,
  1.55064172e-02, -2.78443097e-05, -1.75419151e-02,
 -6.12574814e-04, -7.82703546e-02, -2.15793400e-02,
  1.18004185e-01,  4.79204305e-04, -4.79204305e-04,
  8.92131684e-02, -8.92131684e-02,  8.15424967e-02,
 -2.08611975e-02, -4.05674579e-02, -2.01138413e-02],
[-5.47348461e-03,  5.37016911e-04,  1.18074844e-03,
  3.80787267e-03,  6.12141930e-05, -1.13057857e-03,
  1.02725169e-03,  1.03326872e-04, -1.72012269e-03,
 -7.82420954e-04,  4.24283375e-03, -2.21949441e-03,
  4.79204305e-04,  2.49947704e-01, -2.49947704e-01,
  1.18759508e-03, -1.18759508e-03,  1.47168174e-03,
```

```
 -1.10962272e-03,  5.17273410e-03, -5.53479312e-03],
[ 5.47348461e-03, -5.37016911e-04, -1.18074844e-03,
 -3.80787267e-03, -6.12141930e-05,  1.13057857e-03,
 -1.02725169e-03, -1.03326872e-04,  1.72012269e-03,
  7.82420954e-04, -4.24283375e-03,  2.21949441e-03,
 -4.79204305e-04, -2.49947704e-01,  2.49947704e-01,
 -1.18759508e-03,  1.18759508e-03, -1.47168174e-03,
  1.10962272e-03, -5.17273410e-03,  5.53479312e-03],
[-3.24133005e-01, -7.73246886e-02, -5.29659034e-02,
 -7.20736289e-02, -1.62687279e-01, -8.59434746e-03,
  8.46139191e-03,  1.32955542e-04, -2.19621889e-02,
  2.92502191e-03, -3.69362445e-02, -3.32397568e-02,
  8.92131684e-02,  1.18759508e-03, -1.18759508e-03,
  2.26689198e-01, -2.26689198e-01,  7.39300451e-02,
 -3.15151525e-02, -2.42898653e-02, -1.81250273e-02],
[ 3.24133005e-01,  7.73246886e-02,  5.29659034e-02,
  7.20736289e-02,  1.62687279e-01,  8.59434746e-03,
 -8.46139191e-03, -1.32955542e-04,  2.19621889e-02,
 -2.92502191e-03,  3.69362445e-02,  3.32397568e-02,
 -8.92131684e-02, -1.18759508e-03,  1.18759508e-03,
 -2.26689198e-01,  2.26689198e-01, -7.39300451e-02,
  3.15151525e-02,  2.42898653e-02,  1.81250273e-02],
[-1.78880957e-01, -6.57617290e-02, -3.41934954e-02,
 -4.73975289e-02, -1.24131284e-01, -1.30736296e-02,
  1.31351693e-02, -6.15396592e-05, -1.51399181e-02,
  2.75787306e-04, -4.87264216e-02, -1.79519444e-02,
  8.15424967e-02,  1.47168174e-03, -1.47168174e-03,
  7.39300451e-02, -7.39300451e-02,  2.10834872e-01,
 -5.15086947e-02, -1.13971449e-01, -4.53547288e-02],
[ 9.11617718e-02,  2.33458692e-02,  2.68287396e-02,
  2.79234474e-02,  4.02514352e-02, -7.32280448e-03,
  7.15382977e-03,  1.68974708e-04,  3.78549823e-03,
 -7.64120893e-04,  4.62734251e-03,  1.32124777e-02,
 -2.08611975e-02, -1.10962272e-03,  1.10962272e-03,
 -3.15151525e-02,  3.15151525e-02, -5.15086947e-02,
  1.41431831e-01, -6.43250861e-02, -2.55980499e-02],
[ 6.02856141e-02,  3.23376700e-02, -1.00258399e-02,
  1.55515973e-02,  5.23304019e-02,  2.24199809e-02,
 -2.23431290e-02, -7.68519547e-05,  7.60282445e-03,
  1.16116166e-03,  2.65847593e-02,  5.21871249e-03,
 -4.05674579e-02,  5.17273410e-03, -5.17273410e-03,
 -2.42898653e-02,  2.42898653e-02, -1.13971449e-01,
 -6.43250861e-02,  2.34936426e-01, -5.66398906e-02],
[ 2.74335710e-02,  1.00781898e-02,  1.73905957e-02,
  3.92248423e-03,  3.15494473e-02, -2.02354682e-03,
  2.05412992e-03, -3.05830943e-05,  3.75159539e-03,
 -6.72828074e-04,  1.75143198e-02, -4.79245802e-04,
```

```
         -2.01138413e-02, -5.53479312e-03,  5.53479312e-03,
         -1.81250273e-02,  1.81250273e-02, -4.53547288e-02,
         -2.55980499e-02, -5.66398906e-02,  1.27592669e-01]])
```

[14]:
```python
#perform eigendecomposition (getting eigenvalues and eigenvectors)
eigvalues_unsorted, eigvectors_unsorted = np.linalg.eig(cov_matrix)
eigvalues, eigvectors = eigsort(eigvalues_unsorted, eigvectors_unsorted)
eigvalues = np.real(eigvalues) #removing complex part
eigvalues
```

[14]:
```
array([ 2.06531636e+00,  1.01452851e+00,  8.80807780e-01,  8.24359680e-01,
        6.64450142e-01,  5.00459909e-01,  4.74789809e-01,  3.34209942e-01,
        2.84357431e-01,  1.89461132e-01,  1.81322133e-01,  1.54079952e-01,
        1.34637556e-01,  5.54344143e-02,  5.47318735e-03,  3.04639793e-04,
        1.01593749e-15, -9.64078022e-33, -6.36001403e-18, -2.82246261e-16,
       -6.25015612e-16])
```

[15]:
```python
eigvectors = np.real(eigvectors)
eigvectors
```

[15]:
```
array([[ 5.79693710e-01, -9.28574779e-02, -2.94480532e-01,
        -1.00475785e-01,  4.80178665e-01,  1.06385974e-02,
         1.07804353e-01, -1.07635683e-01, -1.23126931e-02,
        -3.82836399e-01,  2.56587348e-01,  1.67971436e-01,
        -1.67612954e-01, -1.88979530e-01,  7.23388364e-03,
        -3.15633167e-04,  5.38495185e-16,  2.40658650e-19,
        -2.04936453e-17, -1.11354475e-16,  1.62781541e-16],
       [ 3.58274805e-01,  1.38614717e-01,  5.84884859e-01,
        -6.97601961e-01, -1.28339399e-01, -6.56349088e-04,
        -3.52019433e-02,  2.58143449e-02, -5.10671717e-02,
         4.12821019e-02, -9.87177686e-03, -1.53690698e-02,
        -3.80550975e-03,  2.54780112e-02,  2.34747826e-05,
        -2.35679713e-05, -4.62717120e-17, -1.07098904e-16,
        -3.88425896e-17,  5.29431977e-17, -2.77049238e-16],
       [ 2.79768413e-01,  7.24901352e-01, -4.75082317e-01,
        -2.88541387e-02, -3.77012340e-01,  4.00589073e-04,
        -1.45338221e-01,  2.75247882e-02,  1.61072905e-02,
         6.26045434e-02, -1.47982847e-02, -2.89200558e-02,
         9.07955426e-03,  1.93851228e-02, -5.12538560e-04,
        -5.36511373e-05,  3.99039389e-16,  3.37749568e-17,
        -2.55378332e-17, -6.02942461e-17,  1.61440647e-16],
       [ 3.45410826e-01,  3.19019053e-01,  5.40139209e-01,
         6.70130922e-01,  1.73174957e-01, -7.94410283e-03,
        -7.52349056e-02,  2.84283410e-02, -1.75186361e-02,
         2.10881471e-02, -1.44231494e-02,  4.40440029e-03,
         4.66311330e-03,  2.38730606e-02, -6.64210161e-04,
         3.22228675e-04, -6.85523428e-16,  4.50917354e-17,
```

```
  5.64017538e-17, -2.06941130e-17, -4.20133312e-18],
[ 4.25412432e-01, -5.06941120e-01, -8.99533347e-03,
  2.20372856e-01, -6.86795459e-01, -2.82109192e-02,
  5.13190217e-03, -1.68044740e-01,  6.33271765e-02,
 -3.12117692e-02,  5.54108981e-02, -8.88171613e-04,
  4.51054565e-03, -6.75556008e-02,  8.67903504e-04,
 -3.19928580e-04,  7.17522064e-16,  3.95814122e-18,
 -2.82714990e-17,  9.39277818e-17,  1.99667490e-16],
[-3.16068346e-04, -1.03458430e-01, -2.64446327e-02,
 -2.37996899e-02,  8.59434518e-02,  2.94475576e-02,
 -6.77723237e-01, -1.21236394e-01, -7.23885355e-02,
  1.63943293e-03, -1.22333740e-02, -1.09183439e-02,
 -3.39156226e-03, -1.63306912e-02,  1.62015130e-03,
  4.08220796e-01,  5.76943137e-01,  3.27915687e-05,
 -3.08181016e-03, -3.84922947e-03,  5.02096236e-03],
[ 4.00615427e-04,  1.03325045e-01,  2.62761477e-02,
  2.36137417e-02, -8.60055548e-02, -2.97359646e-02,
  6.77708824e-01,  1.21001402e-01,  7.27616022e-02,
 -7.31894030e-04,  1.29522814e-02,  1.15953050e-02,
  3.47080978e-03,  1.52725991e-02, -1.01873725e-03,
  4.08273546e-01,  5.76943137e-01,  3.27915687e-05,
 -3.08181016e-03, -3.84922947e-03,  5.02096236e-03],
[-8.45470809e-05,  1.33385056e-04,  1.68485024e-04,
  1.85948130e-04,  6.21030252e-05,  2.88407034e-04,
  1.44134675e-05,  2.34992479e-04, -3.73066761e-04,
 -9.07538901e-04, -7.18907422e-04, -6.76961123e-04,
 -7.92475117e-05,  1.05809208e-03, -6.01414053e-04,
 -8.16494342e-01,  5.76943137e-01,  3.27915687e-05,
 -3.08181016e-03, -3.84922947e-03,  5.02096236e-03],
[ 2.64242024e-02, -1.73297310e-02, -1.48308938e-02,
 -3.37714356e-03,  3.02995190e-02, -1.01054381e-02,
  1.25506588e-02, -1.66306538e-01,  2.18563088e-01,
  1.70472496e-01, -2.03574554e-01,  3.37787137e-02,
 -7.08840166e-01,  3.06609294e-01, -2.32500295e-01,
  4.40975398e-04,  6.02774703e-03,  1.40533672e-03,
 -1.01355325e-03, -4.19215451e-01, -1.77824227e-01],
[-3.13638587e-03,  1.13425766e-03,  1.89336087e-03,
  6.25111900e-04, -2.78108468e-03, -2.35452393e-03,
 -6.79940514e-04,  1.36490250e-03,  9.29669202e-03,
 -4.65549615e-03, -1.67577712e-03,  7.75368628e-04,
 -6.97358012e-04,  2.02593269e-02,  8.94117700e-01,
 -6.30128051e-04,  6.02774703e-03,  1.40533672e-03,
 -1.01355325e-03, -4.19215451e-01, -1.77824227e-01],
[ 5.94497418e-02, -1.07012627e-01, -5.46563962e-02,
  3.85031433e-02, -6.85756121e-02,  3.60000354e-02,
 -4.63682385e-02,  6.09322300e-01, -5.48938364e-01,
 -9.21973376e-02,  1.55213682e-01, -2.53731839e-02,
```

```
  3.16283828e-02,  1.28115981e-01, -2.21672227e-01,
  7.19805237e-04,  6.02774703e-03,  1.40533672e-03,
 -1.01355325e-03, -4.19215451e-01, -1.77824227e-01],
[ 6.61219371e-02,  8.51098083e-03, -2.58103622e-02,
 -3.39159201e-02,  9.16671294e-02, -1.18575651e-02,
  2.00408300e-02, -2.81578330e-01,  2.73839527e-01,
 -1.40517508e-01, -2.44364270e-02,  5.80074611e-02,
  6.49148592e-01,  3.61443794e-01, -2.34043853e-01,
  4.85858490e-04,  6.02774703e-03,  1.40533672e-03,
 -1.01355325e-03, -4.19215451e-01, -1.77824227e-01],
[-1.48859495e-01,  1.14697120e-01,  9.34042912e-02,
 -1.83519153e-03, -5.06099516e-02, -1.16825084e-02,
  1.44566902e-02, -1.62802334e-01,  4.72390570e-02,
  6.68978458e-02,  7.44730760e-02, -6.71883596e-02,
  2.87605491e-02, -8.16428395e-01, -2.05901324e-01,
 -1.01651107e-03,  6.02774703e-03,  1.40533672e-03,
 -1.01355325e-03, -4.19215451e-01, -1.77824227e-01],
[-1.26977467e-03,  4.99116286e-03,  1.02886651e-02,
  9.18474494e-03, -2.18289505e-02,  7.04296406e-01,
  3.45079456e-02, -4.07011375e-02, -5.92850204e-03,
 -3.47340218e-03, -1.47459701e-02,  1.24781596e-02,
 -2.91436071e-04, -1.39099292e-03,  1.87270383e-03,
  2.47225897e-04,  6.12119098e-04, -1.35579494e-02,
 -7.07072658e-01,  2.55714970e-03,  2.47415982e-03],
[ 1.26977467e-03, -4.99116286e-03, -1.02886651e-02,
 -9.18474494e-03,  2.18289505e-02, -7.04296406e-01,
 -3.45079456e-02,  4.07011375e-02,  5.92850204e-03,
  3.47340218e-03,  1.47459701e-02, -1.24781596e-02,
  2.91436071e-04,  1.39099292e-03, -1.87270383e-03,
 -2.47225897e-04,  6.12119098e-04, -1.35579494e-02,
 -7.07072658e-01,  2.55714970e-03,  2.47415982e-03],
[-2.21562721e-01,  1.16898046e-01,  1.30818226e-01,
  1.43595696e-02, -1.87853197e-01, -9.07396507e-03,
 -6.37305716e-02,  9.97899853e-03,  8.82729756e-02,
 -5.68220772e-01,  6.47200818e-02,  1.65895512e-01,
 -1.31056070e-01,  2.82378001e-02, -6.52513325e-03,
  5.07395670e-04, -3.52823533e-15,  7.06971200e-01,
 -1.41491043e-14, -2.30605048e-17,  5.24359748e-16],
[ 2.21562721e-01, -1.16898046e-01, -1.30818226e-01,
 -1.43595696e-02,  1.87853197e-01,  9.07396507e-03,
  6.37305716e-02, -9.97899853e-03, -8.82729756e-02,
  5.68220772e-01, -6.47200818e-02, -1.65895512e-01,
  1.31056070e-01, -2.82378001e-02,  6.52513325e-03,
 -5.07395670e-04, -1.12354119e-15,  7.06971200e-01,
 -1.41055722e-14,  8.10818009e-16,  8.81488624e-16],
[-1.43464414e-01,  9.26711263e-02,  5.99847517e-02,
  1.43323011e-02, -5.62214316e-02, -1.51750647e-02,
```

```
         5.32382992e-02, -3.74005363e-01, -2.53509634e-01,
         1.60383764e-01,  6.65239480e-01,  8.09842203e-03,
        -7.44433989e-02,  1.81051519e-01,  1.88754403e-04,
        -4.83871654e-04, -1.75173700e-02,  1.21763085e-03,
        -3.96498723e-03, -1.74093726e-01,  4.58749618e-01],
       [ 6.07871463e-02, -6.56209500e-03, -3.23756316e-02,
         1.13151828e-03,  4.17833709e-02, -9.32906469e-03,
         6.79456305e-02, -1.28960700e-01, -1.82613311e-01,
        -3.17339699e-01, -4.16558814e-01, -6.43578537e-01,
        -1.33687070e-02, -1.35682593e-02,  9.64951020e-04,
         1.26971163e-03, -1.75173700e-02,  1.21763085e-03,
        -3.96498723e-03, -1.74093726e-01,  4.58749618e-01],
       [ 5.52592518e-02, -7.33827930e-02, -3.82372242e-03,
        -1.49611185e-02,  3.19588578e-02,  4.93059149e-02,
        -1.42793213e-01,  5.13560046e-01,  6.34804367e-01,
         7.26644508e-02,  1.94071332e-01, -6.25517470e-02,
         1.11992801e-02, -6.12800317e-02, -4.76500143e-03,
        -4.23595131e-04, -1.75173700e-02,  1.21763085e-03,
        -3.96498723e-03, -1.74093726e-01,  4.58749618e-01],
       [ 2.74180156e-02, -1.27262383e-02, -2.37853976e-02,
        -5.02700885e-04, -1.75207971e-02, -2.48017855e-02,
         2.16092836e-02, -1.05939833e-02, -1.98681422e-01,
         8.42914847e-02, -4.42751998e-01,  6.98031862e-01,
         7.66128258e-02, -1.06203228e-01,  3.61129601e-03,
        -3.62244844e-04, -1.75173700e-02,  1.21763085e-03,
        -3.96498723e-03, -1.74093726e-01,  4.58749618e-01]])
```

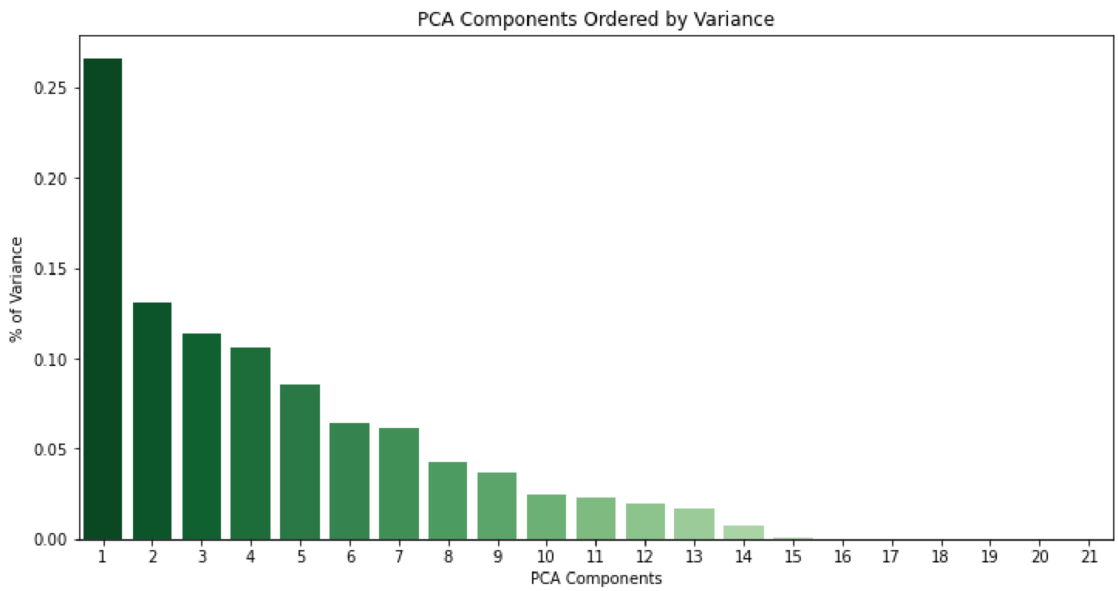
[16]:
```
#check variances to make sure they equal one and find the two that have the␣
↪most impact
variances = []
for i in range(len(eigvalues)):
    var = eigvalues[i] / np.sum(eigvalues)
    variances.append(var.real)

print(np.sum(variances), "\n", variances)
```

```
1.0000000000000002
 [0.2660121505972248, 0.13067097929913357, 0.1134477873885905,
0.10617728842104063, 0.08558098614894535, 0.06445909151464112,
0.06115279007559403, 0.04304614387806662, 0.036625154835014856,
0.024402538957112512, 0.023354238275503182, 0.019845453314561338,
0.017341278291473354, 0.007139936537188821, 0.0007049449486592293,
3.9237517250181174e-05, 1.3085245541479122e-16, -1.2417297046795426e-33,
-8.191679679962409e-19, -3.635323683473905e-17, -8.05018300374906e-17]
```

[17]:
```
#plotting variances
fig = plt.figure(figsize=(12,6))
```

```
sns.barplot(x=list(range(1,22)), y=variances, palette='Greens_r')
plt.ylabel("% of Variance")
plt.xlabel("PCA Components")
plt.title("PCA Components Ordered by Variance")
plt.show()
```

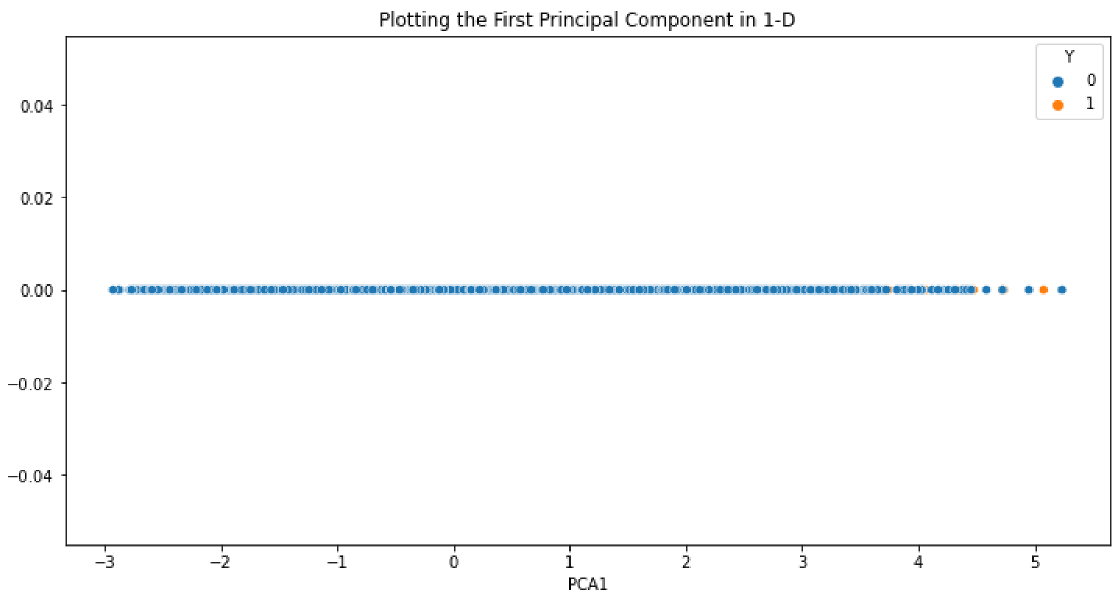


[18]:
```
#assign the two principal components and the target variable (stroke)
#transforming our data
PCA1 = strokeX.dot(eigvectors.T[0])
PCA2 = strokeX.dot(eigvectors.T[1])
res = pd.DataFrame(PCA1, columns=["PCA1"])
res["PCA2"] = PCA2
res["Y"] = strokeY
res.head()
```

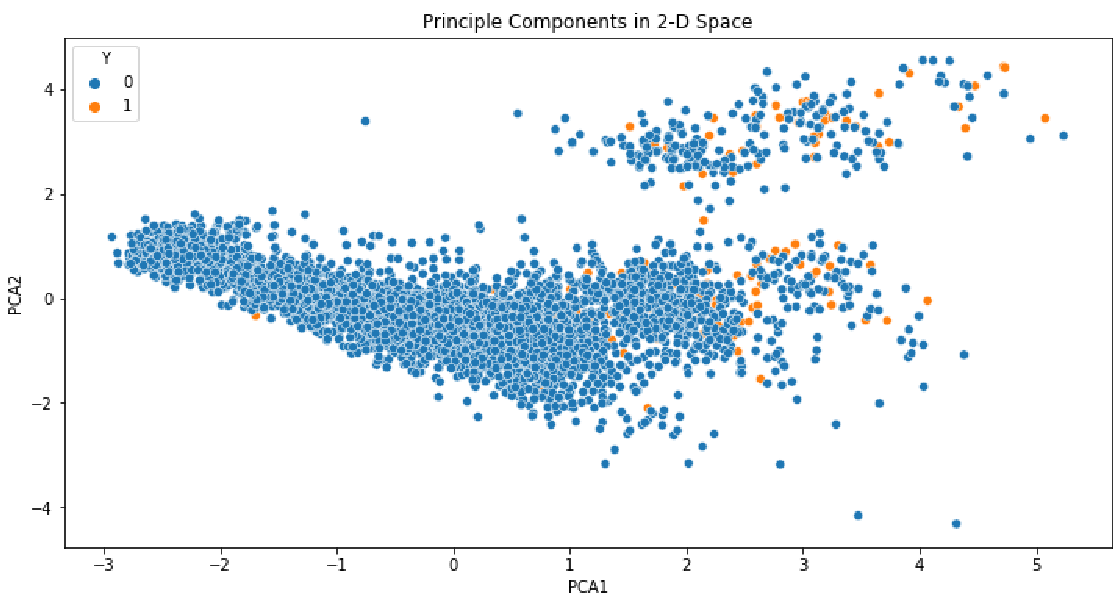
[18]:

	PCA1	PCA2	Y
0	3.452727	3.289551	1
2	2.602045	2.562126	1
3	1.100204	-0.461866	1
4	2.602765	0.651449	1
5	1.780779	0.075491	1

[19]:
```
#plotting in 1-D space
fig = plt.figure(figsize=(12,6))
sns.scatterplot(x=res["PCA1"], y=[0] * len(res), hue=res["Y"])
plt.title("Plotting the First Principal Component in 1-D")
plt.show()
```

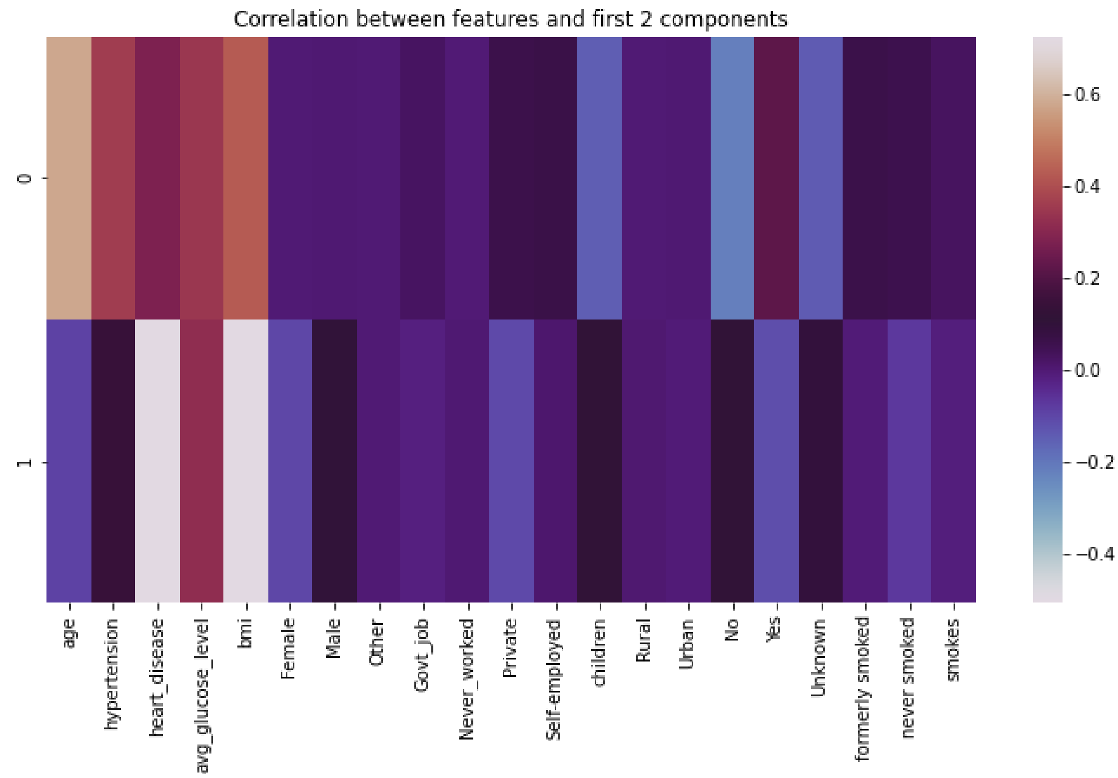


[20]:
```
#plotting in 2-D space
fig = plt.figure(figsize=(12,6))
sns.scatterplot(x=res["PCA1"], y=res["PCA2"], hue=res["Y"])
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.title("Principle Components in 2-D Space")
plt.show()
```



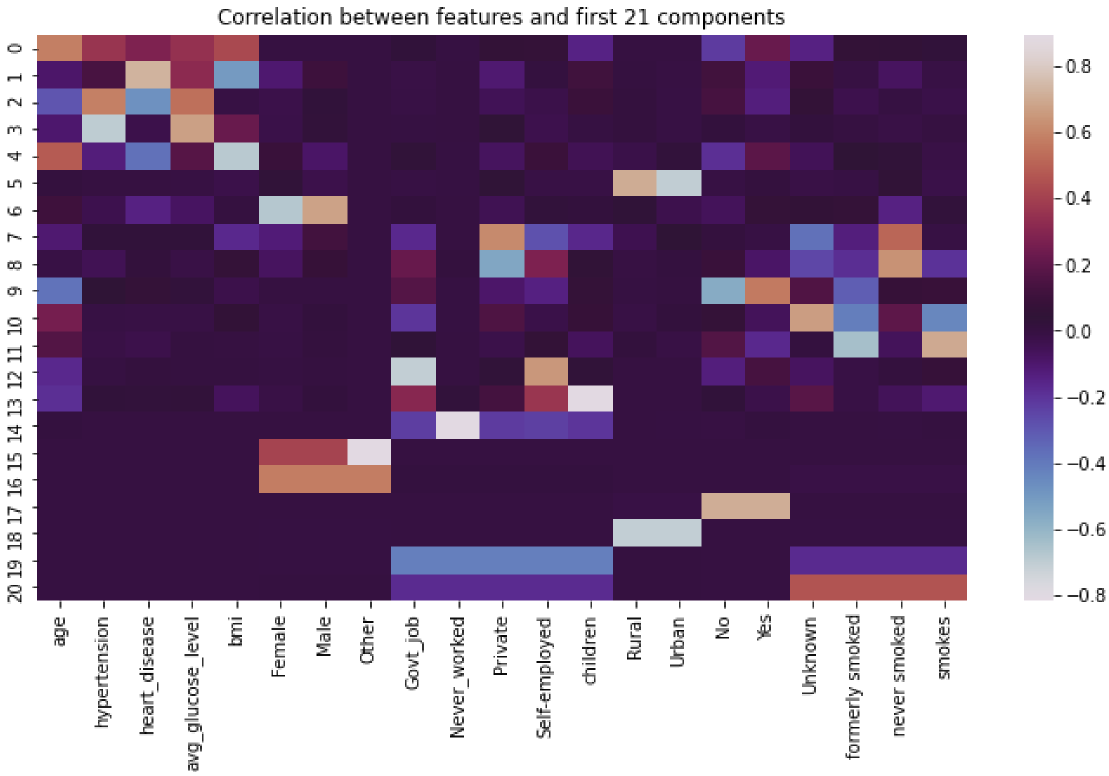
[21]:

```
#visualize components with heatmap for first 2 components
map = pd.DataFrame(eigvectors.T[:2], columns = scaled_columns.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
plt.title("Correlation between features and first 2 components")
plt.show()
```



[22]:

```
#visualize components with heatmap for first 21 components
map = pd.DataFrame(eigvectors.T, columns = scaled_columns.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
plt.title("Correlation between features and first 21 components")
plt.show()
```



3 Comparing manual PCA with scitkit learn library

[23]:
```
pca = PCA(n_components=21)
pca.fit(scaled_columns)
x_pca = pca.transform(scaled_columns)
```

[24]:
```
#turning to dataframe for easier plotting
sklearn_PCA1 = x_pca[:, 0]
sklearn_PCA2 = x_pca[:, 1]
sklearn_pd = pd.DataFrame(sklearn_PCA1, columns=["PCA1"])
sklearn_pd["PCA2"] = sklearn_PCA2
sklearn_pd["Y"] = strokeY
sklearn_pd.head()
```

[24]:

	PCA1	PCA2	Y
0	3.365576	3.393241	1.0
1	2.514895	2.665816	NaN
2	1.013053	-0.358177	1.0
3	2.515615	0.755138	1.0
4	1.693628	0.179180	1.0

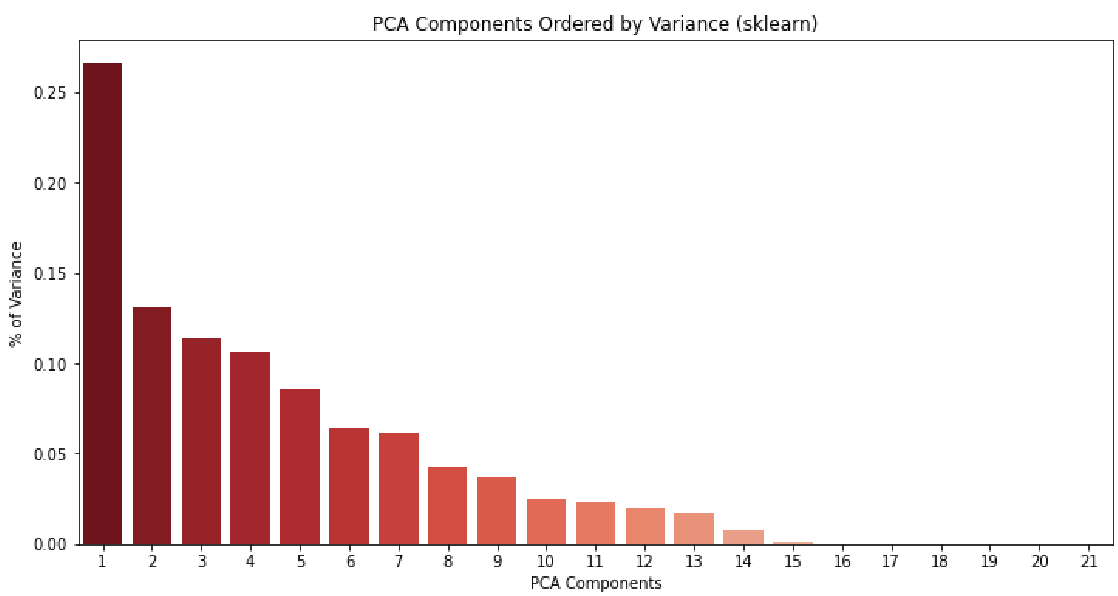
[25]:
```
#explained variance
pca.explained_variance_
```

[25]:
```
array([2.06573717e+00, 1.01473522e+00, 8.80987244e-01, 8.24527642e-01,
       6.64585523e-01, 5.00561877e-01, 4.74886547e-01, 3.34278037e-01,
       2.84415368e-01, 1.89499734e-01, 1.81359077e-01, 1.54111346e-01,
       1.34664988e-01, 5.54457090e-02, 5.47430251e-03, 3.04701863e-04,
       2.42827410e-32, 1.78921912e-32, 1.18645636e-32, 7.55940696e-33,
       3.70496490e-33])
```

[26]:
```
#percentage of explained variance
pca.explained_variance_ratio_
```

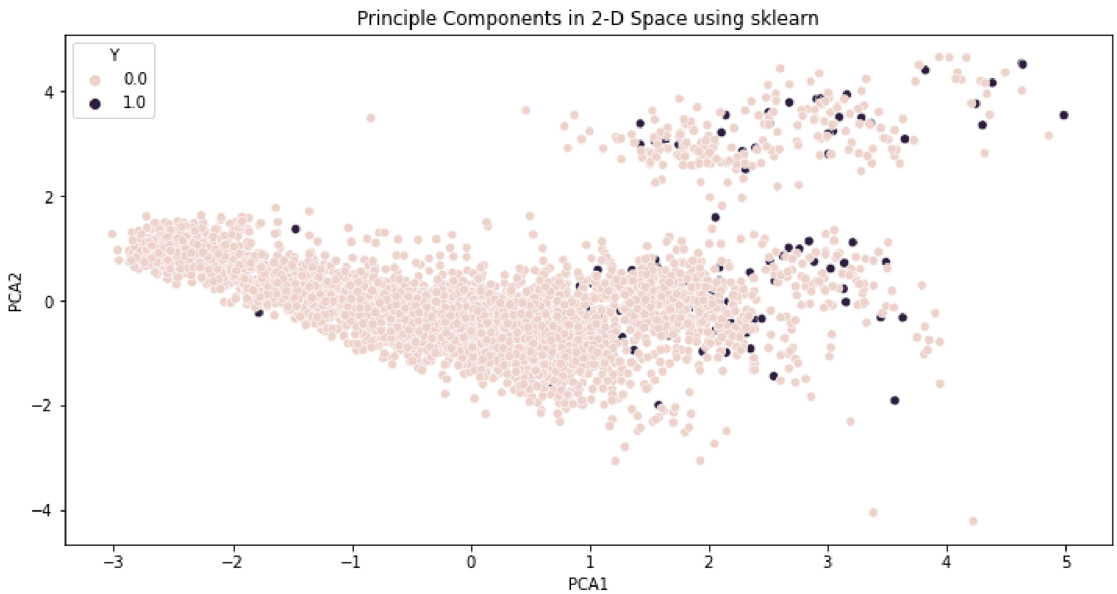
[26]:
```
array([2.66012151e-01, 1.30670979e-01, 1.13447787e-01, 1.06177288e-01,
       8.55809861e-02, 6.44590915e-02, 6.11527901e-02, 4.30461439e-02,
       3.66251548e-02, 2.44025390e-02, 2.33542383e-02, 1.98454533e-02,
       1.73412783e-02, 7.13993654e-03, 7.04944949e-04, 3.92375173e-05,
       3.12697290e-33, 2.30403961e-33, 1.52784107e-33, 9.73451091e-34,
       4.77101199e-34])
```

[27]:
```
#plotting variances
fig = plt.figure(figsize=(12,6))
sns.barplot(x=list(range(1,22)), y=pca.explained_variance_ratio_,␣
 ↪palette='Reds_r')
plt.ylabel("% of Variance")
plt.xlabel("PCA Components")
plt.title("PCA Components Ordered by Variance (sklearn)")
plt.show()
```



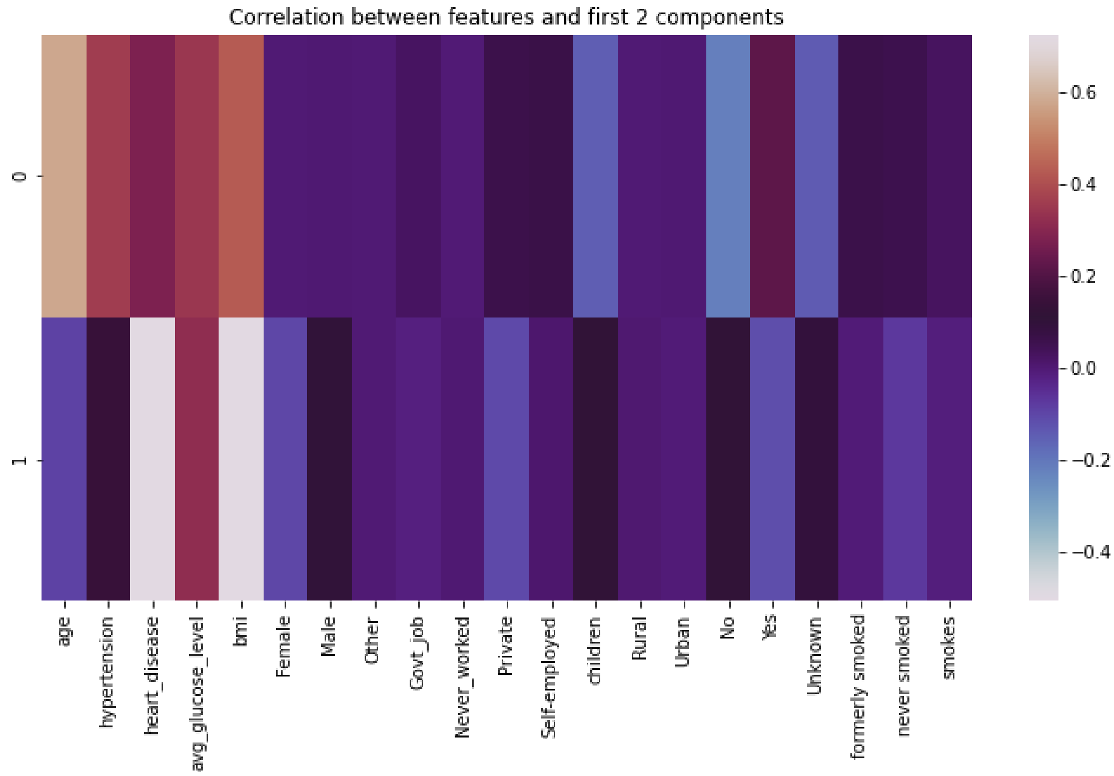
[28]:

```
#plotting in 2-D space
fig = plt.figure(figsize=(12,6))
sns.scatterplot(x=sklearn_pd["PCA1"], y=sklearn_pd["PCA2"], hue=sklearn_pd["Y"])
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.title("Principle Components in 2-D Space using sklearn")
plt.show()
```

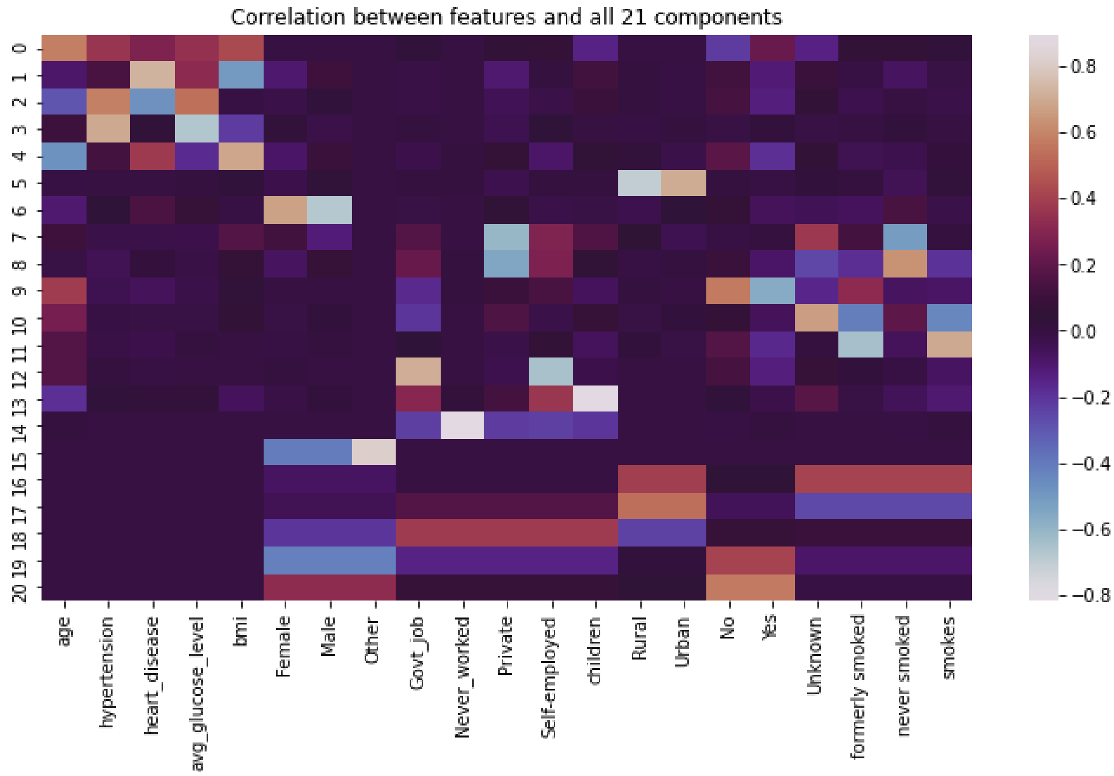


[29]:

```
#visualize components with heatmap for first 2 components
map= pd.DataFrame(pca.components_[:2], columns = scaled_columns.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
plt.title("Correlation between features and first 2 components")
plt.show()
```



[30]:
```
#visualize components with heatmap for all 21 components
map= pd.DataFrame(pca.components_, columns = scaled_columns.columns)
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')
plt.title("Correlation between features and all 21 components")
plt.show()
```



4 Classification with Pre-PCA and Post-PCA (first 2 components and first 9 components)

5 Pre-PCA and Post-PCA Data Comparison

In this section, we will run several supervised machine learning algorithms and analyze the effects of PCA dimension deduction on our dataset.

[31]:
```
sum(variances[0:9]) #variance covered by first 9 principle components
```

[31]: 0.9071723721582514

[32]:
```
#dataframe containing first 9 pricipal components, to cover 90% of variance
PCA3 = strokeX.dot(eigvectors.T[2])
PCA4 = strokeX.dot(eigvectors.T[3])
PCA5 = strokeX.dot(eigvectors.T[4])
PCA6 = strokeX.dot(eigvectors.T[5])
PCA7 = strokeX.dot(eigvectors.T[6])
PCA8 = strokeX.dot(eigvectors.T[7])
PCA9 = strokeX.dot(eigvectors.T[8])
```

```
res2 = res.copy()
res2["PCA3"] = PCA3
res2["PCA4"] = PCA4
res2["PCA5"] = PCA5
res2["PCA6"] = PCA6
res2["PCA7"] = PCA7
res2["PCA8"] = PCA8
res2["PCA9"] = PCA9
res2.head()
```

[32]:

```
       PCA1      PCA2  Y      PCA3      PCA4      PCA5      PCA6      PCA7  \
0  3.452727  3.289551  1 -1.293309  2.105311 -1.193449 -0.734689  0.014260
2  2.602045  2.562126  1 -2.902097  0.082474 -1.090216  0.775355  0.139937
3  1.100204 -0.461866  1  0.391620  1.340884  0.242430 -0.683137 -0.707632
4  2.602765  0.651449  1  2.141245 -1.524657  1.523697  0.800427 -0.726769
5  1.780779  0.075491  1  0.206255  1.321829  1.341756 -0.695034  0.818199

       PCA8      PCA9
0  0.543358 -0.654002
2  1.051593  0.159819
3  0.388801 -0.874730
4  0.111234  0.491498
5  0.485086 -0.780434
```

[33]:

```
# Seperate the dataset into train and test set
#training on original dataset
X_noPCA = strokeX#scaled data
y_noPCA = strokeY
#training on first 2 PCA components
X_PCA2 = res.drop(columns=['Y'])
y_PCA2 = strokeY
#training on first 9 PCA components
X_PCA9 = res2.drop(columns=['Y'])
y_PCA9 = strokeY


X_noPCA_train, X_noPCA_test, y_noPCA_train, y_noPCA_test =␣
 ↪train_test_split(X_noPCA, y_noPCA, test_size = 0.35)
X_PCA2_train, X_PCA2_test, y_PCA2_train, y_PCA2_test = train_test_split(X_PCA2,␣
 ↪y_PCA2, test_size = 0.35)
X_PCA9_train, X_PCA9_test, y_PCA9_train, y_PCA9_test = train_test_split(X_PCA9,␣
 ↪y_PCA9, test_size = 0.35)
```

5.1 Logistic Regression

[34]:
```
start_time = time.time()
clf = LogisticRegression().fit(X_noPCA_train, y_noPCA_train)
y_noPCA_pred = clf.predict(X_noPCA_test)

# Accuracy score for pre-PCA data
print(metrics.accuracy_score(y_noPCA_test,y_noPCA_pred))
print("Running time: %s seconds " % (time.time() - start_time))
pre_pca_log_reg_acc = metrics.accuracy_score(y_noPCA_test,y_noPCA_pred)
pre_pca_log_reg = (time.time() - start_time)
```

```
0.9598603839441536
Running time: 6.612985134124756 seconds
```

[35]:
```
start_time = time.time()
clf = LogisticRegression(random_state=0).fit(X_PCA2_train, y_PCA2_train)
y_PCA2_pred = clf.predict(X_PCA2_test)

# Accuracy score for post-PCA data using 2 principal components
print(metrics.accuracy_score(y_PCA2_test,y_PCA2_pred))
print("Running time: %s seconds " % (time.time() - start_time))
two_pca_log_reg_acc = metrics.accuracy_score(y_PCA2_test,y_PCA2_pred)
two_pca_log_reg = (time.time() - start_time)
```

```
0.9522978475858057
Running time: 0.01697397232055664 seconds
```

[36]:
```
start_time = time.time()
clf = LogisticRegression(random_state=0).fit(X_PCA9_train, y_PCA9_train)
y_PCA9_pred = clf.predict(X_PCA9_test)

# Accuracy score for post-PCA data using 9 principal components
print(metrics.accuracy_score(y_PCA9_test,y_PCA9_pred))
print("Running time: %s seconds " % (time.time() - start_time))
nine_pca_log_reg_acc = metrics.accuracy_score(y_PCA9_test,y_PCA9_pred)
nine_pca_log_reg = (time.time() - start_time)
```

```
0.9616055846422339
Running time: 4.56661581993103 seconds
```

5.2 Decision Trees

[37]:
```
start_time = time.time()
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_noPCA_train, y_noPCA_train)

y_noPCA_pred = clf.predict(X_noPCA_test)
```

```python

# Accuracy score for pre-PCA data
print(metrics.accuracy_score(y_noPCA_test,y_noPCA_pred))
print("Running time: %s seconds " % (time.time() - start_time))
pre_pca_dt_acc = metrics.accuracy_score(y_noPCA_test,y_noPCA_pred)
pre_pca_dt = (time.time() - start_time)
```

```
0.9075043630017452
Running time: 0.02106785774230957 seconds
```

[38]:
```python
start_time = time.time()
clf = clf.fit(X_PCA2_train, y_PCA2_train)

y_PCA2_pred = clf.predict(X_PCA2_test)

# Accuracy score for post-PCA data using 2 principal componenets
print(metrics.accuracy_score(y_PCA2_test,y_PCA2_pred))
print("Running time: %s seconds " % (time.time() - start_time))
two_pca_dt_acc = metrics.accuracy_score(y_PCA2_test,y_PCA2_pred)
two_pca_dt = (time.time() - start_time)
```

```
0.9162303664921466
Running time: 0.010604619979858398 seconds
```

[39]:
```python
clf = clf.fit(X_PCA9_train, y_PCA9_train)

y_PCA9_pred = clf.predict(X_PCA9_test)

# Accuracy score for post-PCA data using 9 principal components
print(metrics.accuracy_score(y_PCA9_test,y_PCA9_pred))
print("Running time: %s seconds " % (time.time() - start_time))
nine_pca_dt_acc = metrics.accuracy_score(y_PCA9_test,y_PCA9_pred)
nine_pca_dt = (time.time() - start_time)
```

```
0.9220477021524142
Running time: 0.047606468200683594 seconds
```

5.3 Linear SVM

[40]:
```python
svclassifier = SVC(kernel='linear')
```

[41]:
```python
start_time = time.time()
svclassifier.fit(X_noPCA_train, y_noPCA_train)

y_noPCA_pred = svclassifier.predict(X_noPCA_test)

# Accuracy score for pre-PCA data
print(metrics.accuracy_score(y_noPCA_test,y_noPCA_pred))
```

```
print("Running time: %s seconds " % (time.time() - start_time))
pre_pca_svm_acc = metrics.accuracy_score(y_noPCA_test,y_noPCA_pred)
pre_pca_svm = (time.time() - start_time)
```

```
0.9598603839441536
Running time: 0.053108930587768555 seconds
```

[42]:
```
start_time = time.time()
svclassifier.fit(X_PCA2_train, y_PCA2_train)

y_PCA2_pred = svclassifier.predict(X_PCA2_test)

# Accuracy score for post-PCA data using 2 principal components
print(metrics.accuracy_score(y_PCA2_test,y_PCA2_pred))
print("Running time: %s seconds " % (time.time() - start_time))
two_pca_svm_acc = metrics.accuracy_score(y_PCA2_test,y_PCA2_pred)
two_pca_svm = (time.time() - start_time)
```

```
0.951716114019779
Running time: 0.02633380889892578 seconds
```

[43]:
```
start_time = time.time()
svclassifier.fit(X_PCA9_train, y_PCA9_train)

y_PCA9_pred = svclassifier.predict(X_PCA9_test)

# Accuracy score for post-PCA data using 9 principal components
print(metrics.accuracy_score(y_PCA9_test,y_PCA9_pred))
print("Running time: %s seconds " % (time.time() - start_time))
nine_pca_svm_acc = metrics.accuracy_score(y_PCA9_test,y_PCA9_pred)
nine_pca_svm = (time.time() - start_time)
```

```
0.9621873182082606
Running time: 0.0496220588684082 seconds
```

5.4 Analysis

After running 3 different supervised machie learning algorithms with the pre-PCA and post-PCA data, from the accuracy scores, we observe that: for this particular dataset, Logistic Regression and Linear SVM does a slightly better job with pre-PCA data. For Decision Trees, the pre and post PCA data performance are very close, with post PCA slightly better. In the Linear SVM section, we timed the programming running time for our model. Notice that the running time for post-PCA data is much less than the pre-PCA data since we reduce the dimensionality of the dataset from 20 to 2. From above observations, we see that PCA can help us save computational cost significantly while not sacrificing much classification error.

[44]:
```
```

```python
# initialize list of lists
log_reg_data = [['Accuracy',␣
↪pre_pca_log_reg_acc,two_pca_log_reg_acc,nine_pca_log_reg_acc],
        ['Run time', pre_pca_log_reg,two_pca_log_reg,nine_pca_log_reg]]

# Create the pandas DataFrame
log_reg_df = pd.DataFrame(log_reg_data, columns = ['Logistic Regression',␣
↪'pre-PCA','post-PCA(2)','post-PCA(9)'])

# initialize list of lists
dt_data = [['Accuracy', pre_pca_dt_acc,two_pca_dt_acc,nine_pca_dt_acc],
        ['Run time', pre_pca_dt,two_pca_dt,nine_pca_dt]]

# Create the pandas DataFrame
dt_df = pd.DataFrame(dt_data, columns = ['Decision Trees',␣
↪'pre-PCA','post-PCA(2)','post-PCA(9)'])

# initialize list of lists
svm_data = [['Accuracy', pre_pca_svm_acc,two_pca_svm_acc,nine_pca_svm_acc],
        ['Run time', pre_pca_svm,two_pca_svm,nine_pca_svm]]

# Create the pandas DataFrame
svm_df = pd.DataFrame(svm_data, columns = ['Linear SVM',␣
↪'pre-PCA','post-PCA(2)','post-PCA(9)'])

#print dataframe
display(log_reg_df,dt_df,svm_df)
```

```
  Logistic Regression   pre-PCA  post-PCA(2)  post-PCA(9)
0            Accuracy  0.959860     0.952298     0.961606
1            Run time  6.613969     0.018047     4.568514
```

```
  Decision Trees   pre-PCA  post-PCA(2)  post-PCA(9)
0       Accuracy  0.907504     0.916230     0.922048
1       Run time  0.022270     0.011621     0.048572
```

```
  Linear SVM   pre-PCA  post-PCA(2)  post-PCA(9)
0   Accuracy  0.959860     0.951716     0.962187
1   Run time  0.054014     0.027117     0.050555
```

[ ]: