PCA

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
plt.style.use('ggplot')
```

In [10]:
```
#Будет использоваться набор данных под названием diabetes из библиотеки scikit
diabetes = load_diabetes()
df = pd.DataFrame(data=diabetes.data,
 columns=diabetes.feature_names)

df.head(6)
```

Out[10]:

	age	sex	bmi	bp	s1	s2	s3	s4	s
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.01990
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.06833
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.00286
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.02269
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.03199
5	-0.092695	-0.044642	-0.040696	-0.019442	-0.068991	-0.079288	0.041277	-0.076395	-0.04118

Стандартизация данных

Нам нужно масштабировать наши переменные перед проведением анализа, чтобы избежать вводящих в заблуждение результатов PCA из-за различий в единицах измерения. Для масштабирования наших данных до единиц измерения со средним значением в 0 и отклонениями в 1.

In [11]:
```
#Сначала мы создадим объект класса StandardScaler,
#затем используем его для подгонки к нашей матрице данных и преобразуем данные
#В результате мы получим двумерный массив NumPy
scaler = StandardScaler()
scaler.fit(df)
Diabetes_scaled = scaler.transform(df)
```

In [22]:
```
#Мы можем преобразовать его в фрейм данных pandas.
dataframe_scaled = pd.DataFrame(data=Diabetes_scaled,
                                columns=diabetes.feature_names)

dataframe_scaled.head(6)
```

Out[22]:

	age	sex	bmi	bp	s1	s2	s3	s4	s
0	0.800500	1.065488	1.297088	0.459840	-0.929746	-0.732065	-0.912451	-0.054499	0.41855
1	-0.039567	-0.938537	-1.082180	-0.553511	-0.177624	-0.402886	1.564414	-0.830301	-1.43655
2	1.793307	1.065488	0.934533	-0.119218	-0.958674	-0.718897	-0.680245	-0.054499	0.0602C
3	-1.872441	-0.938537	-0.243771	-0.770658	0.256292	0.525397	-0.757647	0.721302	0.47707
4	0.113172	-0.938537	-0.764944	0.459840	0.082726	0.327890	0.171178	-0.054499	-0.67258
5	-1.948811	-0.938537	-0.855583	-0.408747	-1.450445	-1.666931	0.867796	-1.606102	-0.86576

In [ ]:
```
__Идеальное количество компонент__
Одной из альтернатив подбора идеального количества компонент является проведен
После применения PCA визуализируем процент объясненной дисперсии, используя гр
На основе графика можно выбрать оптимальное количество.
```

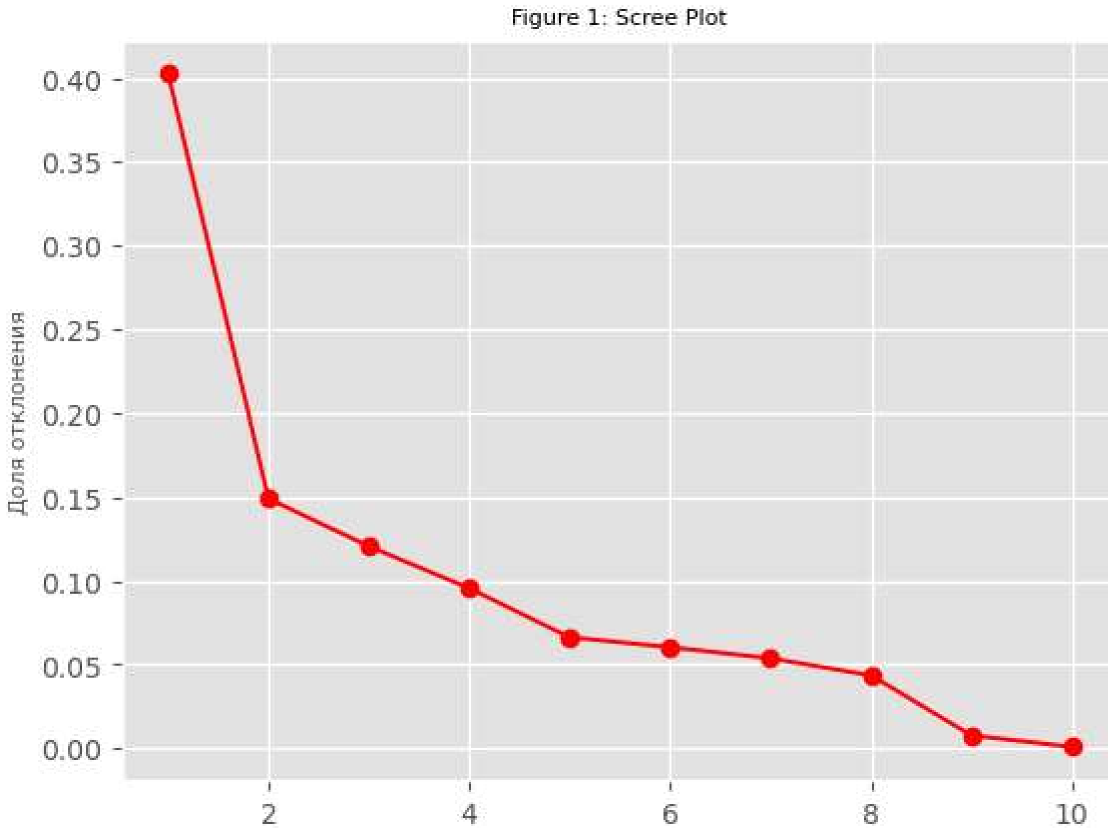
In [23]:
```
#Запустим наш PCA для десяти компонентов!
pca = PCA(n_components=10)
pca.fit_transform(Diabetes_scaled)
```

Out[23]:
```
array([[ 0.58720767, -1.94682793,  0.58923299, ...,  0.75744463,
        -0.18107712, -0.0489563 ],
       [-2.83161209,  1.37208454,  0.02791506, ..., -0.18840749,
         0.50511975,  0.04358938],
       [ 0.27214757, -1.63489803,  0.73927034, ...,  0.84323498,
        -0.0253607 , -0.0541868 ],
       ...,
       [-0.20524634, -1.20544647,  0.4960784 , ..., -0.49169045,
        -0.11323793,  0.05886062],
       [ 0.69286627,  0.21011694, -0.86872976, ...,  0.07867558,
        -0.12721134, -0.04554004],
       [-1.90393365,  3.97577106, -0.04838108, ...,  1.18537538,
         0.73046601, -0.15456893]])
```

In [32]:
```
#Как только мы выполним наш PCA, мы можем извлечь объясненную долю дисперсии и
prop_var = pca.explained_variance_ratio_
eigenvalues = pca.explained_variance_
print(prop_var, eigenvalues, sep ='\n')
```

```
[0.40242142 0.14923182 0.12059623 0.09554764 0.06621856 0.06027192
 0.05365605 0.04336832 0.00783199 0.00085605]
[4.03333938 1.49570218 1.20869692 0.957643   0.66368713 0.60408592
 0.53777715 0.43466661 0.07849751 0.00857994]
```

In [33]:
```
PC_numbers = np.arange(pca.n_components_) + 1

plt.plot(PC_numbers,
         prop_var,
         'ro-')
plt.title('Figure 1: Scree Plot', fontsize=8)
plt.ylabel('Доля отклонения', fontsize=8)
plt.show()
```



Метод локтя Метод интерпретации схемы осыпи заключается в использовании правила изгиба. Этот метод заключается в поиске формы “изгиба” на кривой и сохранении всех компонентов до точки, где кривая выравнивается.

Предполагая, что первые 2 компонента должны сохраняться с учетом правила локтя, мы можем повторно запустить PCA и интерпретировать результаты для первых двух компонентов.

Вычисление основных компонентов и интерпретация результата

In [16]:
```

pca = PCA(n_components=2)
PC = pca.fit_transform(Diabetes_scaled)
```

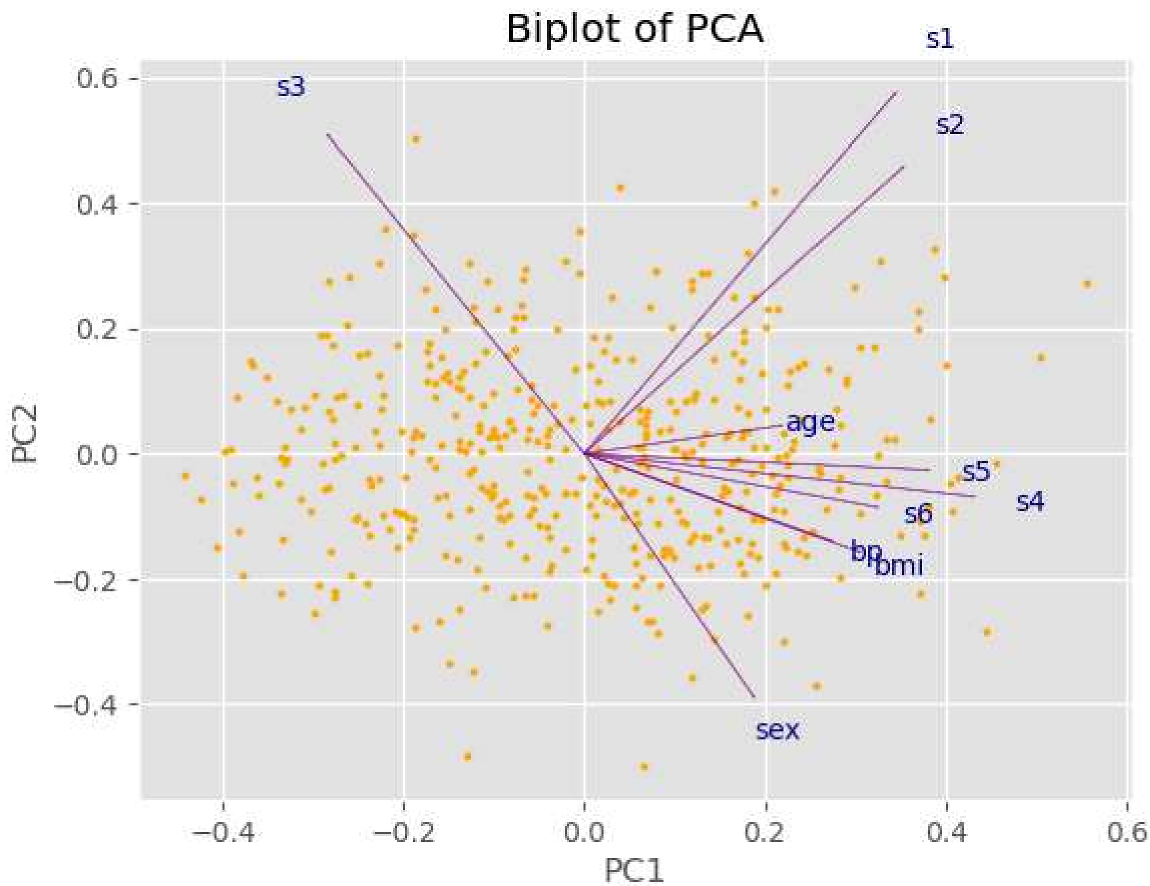
In [17]:
```
pca_diabetes = pd.DataFrame(data = PC,
                            columns = ['PC1', 'PC2'])

pca_diabetes.head(6)
```

Out[17]:

	PC1	PC2
0	0.587208	-1.946828
1	-2.831612	1.372085
2	0.272148	-1.634898
3	0.049310	0.382253
4	-0.756451	0.811968
5	-3.966355	-0.381059

In [34]:
```
#Способ визуализации данных на 2D модель
def biplot(score,coef,labels=None):

    xs = score[:,0]
    ys = score[:,1]
    n = coef.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley,
                s=5,
                color='orange')

    for i in range(n):
        plt.arrow(0, 0, coef[i,0],
                  coef[i,1],color = 'purple',
                  alpha = 0.5)
        plt.text(coef[i,0]* 1.15,
                 coef[i,1] * 1.15,
                 labels[i],
                 color = 'darkblue',
                 ha = 'center',
                 va = 'center')

    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))


    plt.figure()
```

In [19]:
```
plt.title('Biplot of PCA')

biplot(PC,
       np.transpose(pca.components_),
       list(diabetes.feature_names))
```




```
<Figure size 640x480 with 0 Axes>
```

In [ ]: