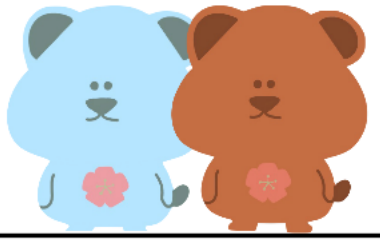
차원 축소(Kaggle 필사)

4팀 엄유진 최유미

EWHA EURON

목차



EWHA EURON

신용카드 서비스 이탈 여부 예측하기

EWHA EURON

#00 Credit Card customers - Predict Churning customers

- 고객 이탈률: ‘customer churning rate’
- 이탈 고객이 늘고 있다는 사실을 알고, 고객 유지를 위해 조치를 취할 수 있도록 누가 이탈할지 예측하려고 한다고 가정.
- 원래 데이터: 10,000 customers, 18 features
  -> 전처리 통해 11개 numerical features로 단순화
- target: 'Attraction_Flag’
  84% - existing(유지) 16% - attrited(이탈)

#01 고객 나이, 소득과 교육 분포 시각화

- 라이브러리 import, DataFrame Loading

- 마지막 2개 칼럼은 제거해준다 (분석 목적에 벗어난 데이터; 나이브베이즈 알고리즘 예측결과)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as ex
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

df=pd.read_csv('/content/BankChurners.csv')
df=df[df.columns[:-2]] # Drop the last two columns
df.head() # Inspect the first 5 rows
```

[Output]

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	Card_Category	Months_on_book	...	Months_Inactive_12_mon	Contacts_Count_12_mon	Cred
0	768805383	Existing Customer	45	M	3	High School	Married	$60K - $80K	Blue	39	...	1	3	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less than $40K	Blue	44	...	1	2	
2	713982108	Existing Customer	51	M	3	Graduate	Married	$80K - $120K	Blue	36	...	1	0	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less than $40K	Blue	34	...	4	1	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	$60K - $80K	Blue	21	...	1	0	

5 rows × 21 columns

#01 고객 나이, 소득과 교육 분포 시각화

시각화를 통해 피처 탐색하기

1. **고객 나이 분포 시각화** (Box Plot & Histogram)

```
fig=make_subplots(rows=2,cols=1)
tr1=go.Box(x=df['Customer_Age'],name='Age Box Plot',boxmean='sd')
tr2=go.Histogram(x=df['Customer_Age'], name='Age Histogram')
fig.add_trace(tr1,row=1,col=1)
fig.add_trace(tr2,row=2,col=1)
fig.update_layout(height=500, width=600, title_text="Distribution of Customer Ages")
fig.show()
```

[Output]

분포도에서 대부분이 40대 중반~ 60대 초반에 집중되어 있고, 연령 분포가 중간 연령대로 편향되어 있음을 확인할 수 있다.

70대 초반에 소수의 고객이 존재하며, 이는 통계적으로 이상치로 간주될 수 있습니다.

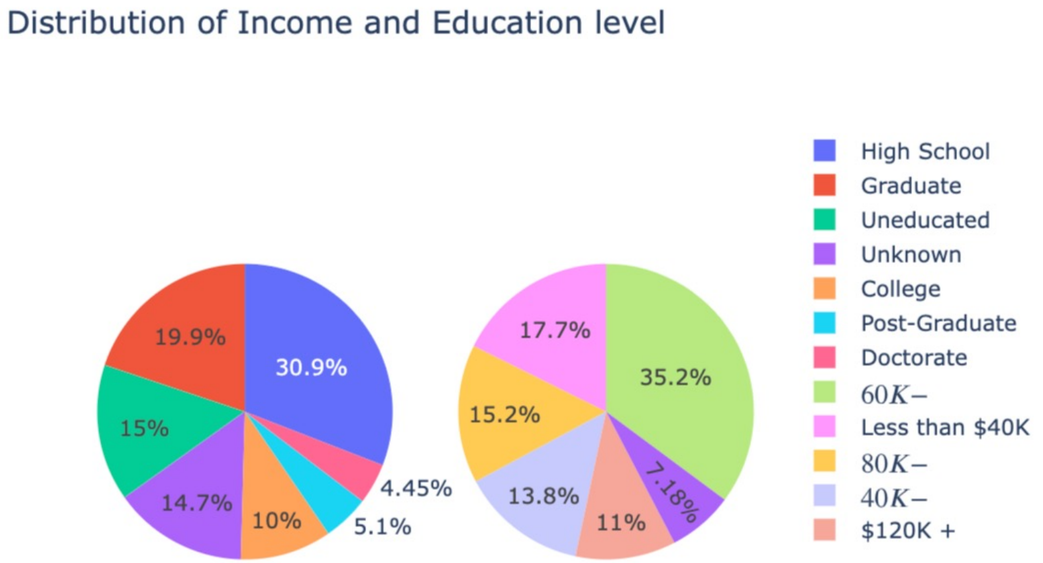
#01 고객 나이, 소득과 교육 분포 시각화

시각화를 통해 피처 탐색하기

2. 고객 소득 & 교육 분포 시각화 (원 그래프)

```
education=pd.DataFrame(df['Education_Level'].value_counts())
labelsedu=df['Education_Level'].unique()
income=pd.DataFrame(df['Income_Category'].value_counts())
labelincome=df['Income_Category'].unique()
#explore education level and income level
fig=make_subplots(rows=1,cols=2,specs=[[{'type':'domain'}, {'type':'domain'}]])
tr3=go.Pie(labels=labelsedu,values=education.iloc[:,0],name='proportion of Education Level')
tr4=go.Pie(labels=labelincome,values=income.iloc[:,0], name='Propotion Of Different Income Levels')
fig.add_trace(tr3,row=1,col=1)
fig.add_trace(tr4,row=1,col=2)
fig.update_layout(height=500, width=600, title_text="Distribution of Income and Education level")
fig.show()
```

[Output]



#02 PCA: leading principal components = 2

PCA(차원축소) 시작 전

- x: 9~21 column, y=target
- 정규분포화
- existing customer ->“1” attrited customer ->“0”

PCA(n_components=2) 적용

```
x=df.iloc[:,9:21] # assign column 9 to 21 as x variable - the features
x=StandardScaler().fit_transform(x) # standarize the variables
df['Attrition_Flag'].replace('Existing Customer','1',inplace=True)
df['Attrition_Flag'].replace('Attrited Customer','0',inplace=True)
y=df['Attrition_Flag'] # assign y variable - the target

pca=PCA(n_components=2)
PC=pca.fit_transform(x)
principalDF=pd.DataFrame(data=PC,columns=['pc1','pc2'])
finalDf = pd.concat([principalDF, df[['Attrition_Flag']]], axis = 1)
finalDf.head()
```

[Output]

	pc1	pc2	Attrition_Flag
0	0.276048	-0.617639	1
1	-0.612402	1.430502	1
2	-0.613733	1.098632	1
3	-2.499317	1.781346	1
4	-0.560120	0.924119	1

#02 PCA: leading principal components = 2

1. Ploting Loadings

- 나중에 예측할 때 각 피처가 어느 정도의 가중치를 가질지 평가하기 위해 'Loadings Table' 을 작성할 수 있다.
- 원래의 각 피처가 각 '새로운 피처' (주요 구성 요소)에 얼마나 기여했는지를 보여줌.

```
PCloadings = pca.components_.T * np.sqrt(pca.explained_variance_)
components=df.columns.tolist()
components=components[9:21]
loadingdf=pd.DataFrame(PCloadings,columns=('PC1','PC2'))
loadingdf["variable"]=components
loadingdf
```

[Output]

	PC1	PC2	variable
0	-0.012248	-0.084536	Months_on_book
1	-0.276207	-0.384630	Total_Relationship_Count
2	-0.030992	-0.105797	Months_Inactive_12_mon
3	-0.017396	-0.314187	Contacts_Count_12_mon
4	0.867614	-0.180299	Credit_Limit
5	-0.261374	0.402668	Total_Revolving_Bal
6	0.890865	-0.216361	Avg_Open_To_Buy
7	-0.012135	0.181603	Total_Amt_Chng_Q4_Q1
8	0.467479	0.763757	Total_Trans_Amt
9	0.359458	0.788716	Total_Trans_Ct
10	-0.012862	0.309368	Total_Ct_Chng_Q4_Q1
11	-0.718652	0.411826	Avg_Utilization_Ratio

#02 PCA: leading principal components = 2

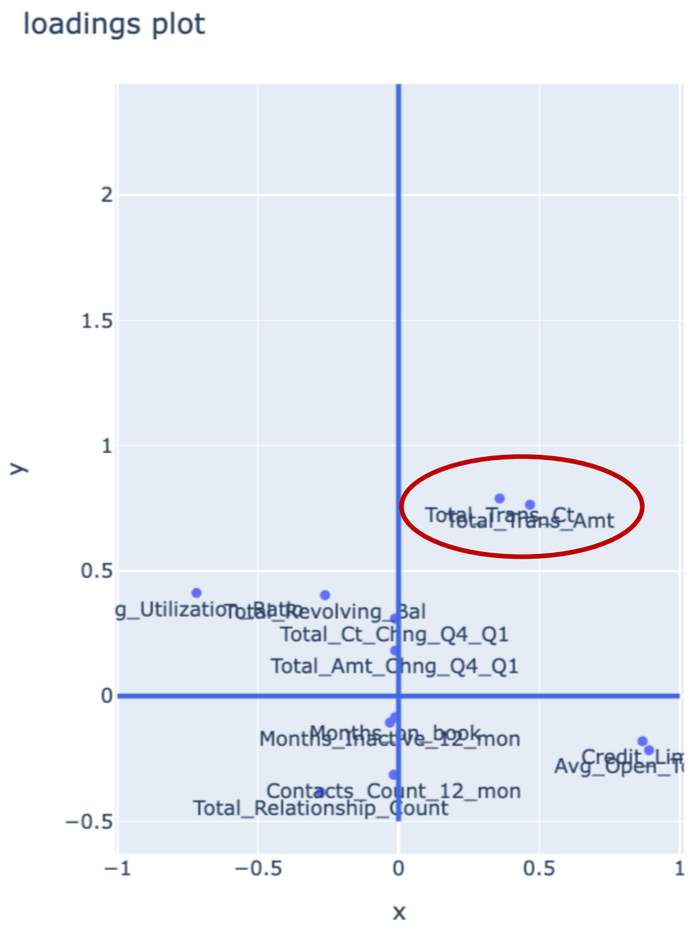
1. Ploting Loadings

- PC1, PC2 모두에서 높은 가중치 가지는 변수 확인

```
fig=ex.scatter(x=loadingdf['PC1'],y=loadingdf['PC2'],text=loadingdf['variable'],)
fig.update_layout(
height=600,width=500,
title_text='loadings plot')
fig.update_traces(textposition='bottom center')
fig.add_shape(type="line",
x0=-0, y0=-0.5,x1=-0,y1=2.5,
line=dict(color="RoyalBlue",width=3)
)
fig.add_shape(type="line",
x0=-1, y0=0,x1=1,y1=0,
line=dict(color="RoyalBlue",width=3)
)
fig.show()
```

- Total transaction count & Total transaction amount -> 예측 정확도가 높다면, 이 두 개가 주요 요인

[Output]



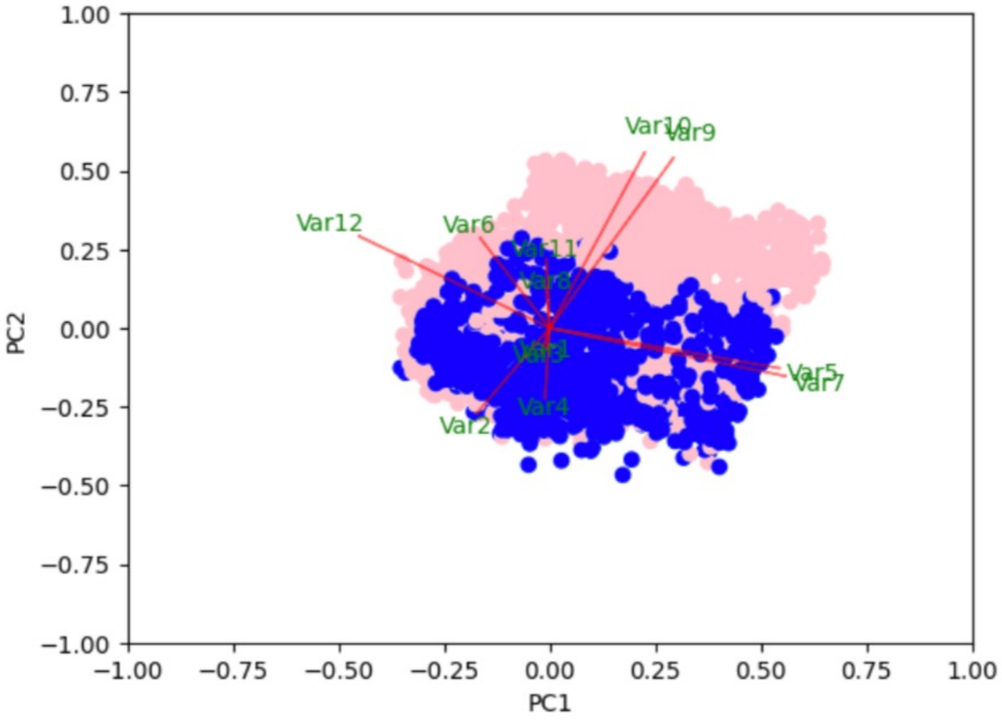
#02 PCA: leading principal components = 2

2. Leading principal components에 따른 산점도 그래프

- 차원 축소(PC1, PC2)로 데이터 분포 어떻게 변했는지 확인

```
def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = 1.0/(xs.max() - xs.min())
    scaley = 1.0/(ys.max() - ys.min())
    colors = {'1':'pink', '0':'blue'}
    plt.scatter(xs * scalex,ys * scaley, c= y.apply(lambda x: colors[x]))
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color = 'g', ha = 'center', va = 'center')
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha = 'center', va = 'center')
    plt.xlim(-1,1)
    plt.ylim(-1,1)
    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))

myplot(PC[:,0:2],np.transpose(pca.components_[0:2, :]))
plt.show()
```

[Output]



#03 Logistic Regression

3. Logistic Regression으로 예측하기

- train_test_split
- 로지스틱 회귀 적용
- 학습, 예측, logistic.score

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
Xfinal=finalDf[['pc1','pc2']]
yfinal=finalDf['Attrition_Flag']
X_train, X_test, y_train, y_test = train_test_split(Xfinal,yfinal,test_size=0.3)
logistic=LogisticRegression()
logistic.fit(X=X_train,y=y_train)
logistic.predict(X_test)
score_2=logistic.score(X_test,y_test)
```

#04 PCA: leading principal components = 2~4

4. 3, 4개 주성분으로 PCA

- principalDF: DataFrame 형태로 변환하고, 각 주성분에 'pc1', 'pc2', 'pc3'라는 열 이름을 부여한다.
- finalDf: 원본 데이터셋의 'Attrition_Flag' 열과 주성분 DataFrame을 결합한다.
- 데이터 분할 및 로지스틱 회귀 모델 적용, 예측 및 logistic.score 평가

```
pca=PCA(n_components=3)
PC=pca.fit_transform(x)
principalDF=pd.DataFrame(data=PC,columns=['pc1','pc2','pc3'])
finalDf = pd.concat([principalDF, df[['Attrition_Flag']]], axis = 1)
Xfinal=finalDf[['pc1','pc2','pc3']]
yfinal=finalDf['Attrition_Flag']
X_train, X_test, y_train, y_test = train_test_split(Xfinal,yfinal,test_size=0.3)
logistic=LogisticRegression()
logistic.fit(X=X_train,y=y_train)
logistic.predict(X_test)
score_3=logistic.score(X_test,y_test)
pca=PCA(n_components=4)
PC=pca.fit_transform(x)
principalDF=pd.DataFrame(data=PC,columns=['pc1','pc2','pc3','pc4'])
finalDf = pd.concat([principalDF, df[['Attrition_Flag']]], axis = 1)
Xfinal=finalDf[['pc1','pc2','pc3','pc4']]
yfinal=finalDf['Attrition_Flag']
X_train, X_test, y_train, y_test = train_test_split(Xfinal,yfinal,test_size=0.3)
logistic=LogisticRegression()
logistic.fit(X=X_train,y=y_train)
logistic.predict(X_test)
score_4=logistic.score(X_test,y_test)
```

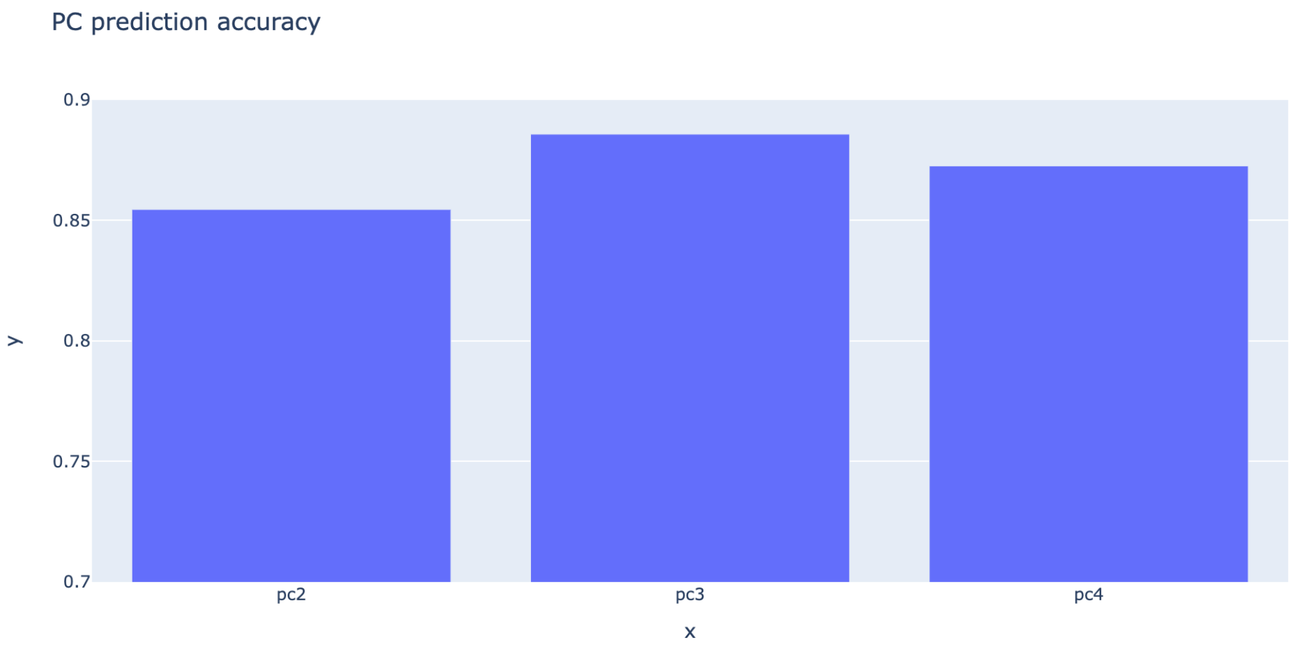
EWHA EURON

#04 PCA: leading principal components = 2~4

5. 정확도 점수 막대그래프 시각화

```
scores=[score_2,score_3,score_4]

ex.bar(y=scores,x=('pc2','pc3','pc4'),range_y=(0.7,0.9),title='PC prediction accuracy')
```

[Output]



- 3개일 때 점수 가장 높음. (2개일 때도 정확도가 84% 달성)
- 따라서 total transaction count(총 거래 건수)와 total transaction amount(총 거래 금액)이 고객 이탈을 예측하는 두 가지 좋은 예측 변수임을 추론할 수 있다.

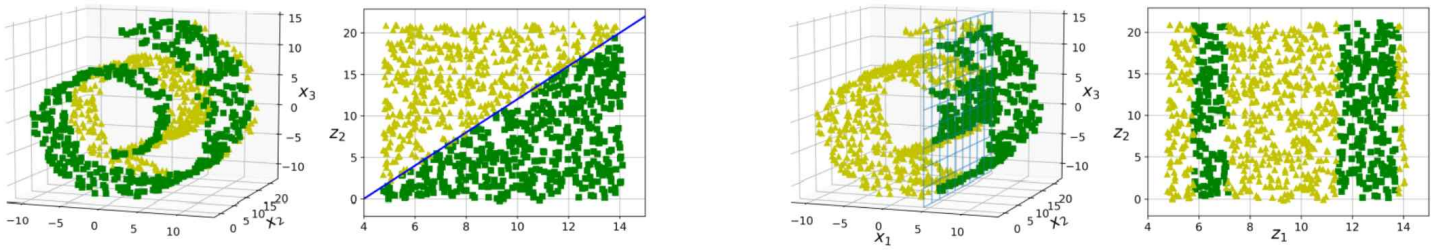
EWHA EURON

유방암 데이터와 차원 축소

EWHA EURON

#00 차원축소 접근 방법

- 차원 축소 접근 방법에는 2가지가 있습니다.
  - 투영(projection) : 모든 훈련 샘플이 고차원 공간 안의 저차원 부분 공간 (subspace)에 놓여있다.
    - PCA, SVD 등
  - manifold : 저차원의 매니폴드 공간에 표현되면 더 간단해질 것이란 가정
    - MSD, T-SNE 등



- 왼쪽은 manifold 가정 시 효율적, 오른쪽은 투영 시 효율적인 데이터

#01 유방암 데이터 불러오기

```
#Dependencies
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm
import matplotlib.patches as mpatches
import seaborn as sns
from sklearn.datasets import load_breast_cancer #Loading the full DataSet From Sklearn Library

# Breast cancer dataset
cancer = load_breast_cancer()
(X_canc, y_canc) = load_breast_cancer(return_X_y = True)
```

- return_X_y=True 를 파라미터로 입력 시 feature와 target값을 각각 반환한다.

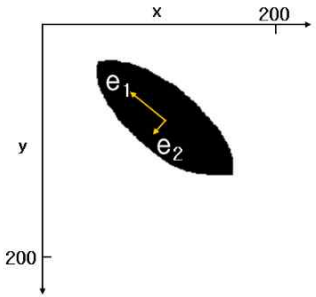
#02 시각화 함수 생성

```
# A Function to Plot labelled Scatter Plot
def plot_labelled_scatter(X, y, class_labels,s):
    num_labels = len(class_labels)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    marker_array = ['o', '^', '*']
    color_array = ['#FFFF00', '#00AAFF', '#000000', '#FF00AA']
    cmap_bold = ListedColormap(color_array)
    bnorm = BoundaryNorm(np.arange(0, num_labels + 1, 1), ncolors=num_labels)
    plt.figure(figsize=s)

    plt.scatter(X[:, 0], X[:, 1], s=80, c=y, cmap=cmap_bold, norm = bnorm, alpha = 0.4, edgecolor='black', lw = 1)
    sp = plt.gca().spines
    sp['top'].set_visible(False)
    sp['right'].set_visible(False)

    plt.grid(which='both',color='lightslategrey',alpha=0.3)

    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)

    h = []
    for c in range(0, num_labels):
        h.append(mpatches.Patch(color=color_array[c], label=class_labels[c]))
    plt.legend(handles=h,fontsize=15,frameon=False)
```

- 인자로 X(Features), y(target), class_labels, s(크기) 입력
- PCA, MDS, T-SNE 각 차원 축소 기법을 적용한 결과를 나타낼 때 사용

#03 PCA(Principal Component Analysis)

- 여러 변수 간의 상관관계를 이용해 이를 대표하는 주성분을 추출해 차원을 축소하는 기법
- 기존 데이터의 정보 유실을 최소화하기 위해 가장 높은 분산을 가지는 데이터의 축을 찾고, 이 축을 기준으로 차원을 축소한다.



- 위와 같이 2차원 좌표평면에서 타원형으로 데이터가 분포할 때, 데이터 특성을 가장 잘 나타내는 e1, e2 두개의 데이터로 분포를 설명한다.

#03 PCA(Principal Component Analysis)

- PCA 스텝
  1. 입력 데이터 세트의 공분산 행렬을 생성한다.
  2. 공분산 행렬의 고유벡터와 고유값을 계산한다.
  3. 고유값이 가장 큰 순으로 K개(PCA 변환 차수)만큼 고유벡터를 추출한다.
  4. 고유값이 가장 큰 순으로 추출된 고유벡터를 이용해 새롭게 입력 데이터를 변환한다.

$$C = [e_1 \cdots e_n] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \cdots & \cdots & \cdots \\ 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} e_1^t \\ \cdots \\ e_n^t \end{bmatrix}$$

- 공분산 행렬의 고유벡터와 고유값 계산은 위 식 이용

- PCA는 컴퓨터 비전 분야, 특히 얼굴 인식에서 많이 사용된다.

#03 PCA(Principal Component Analysis)

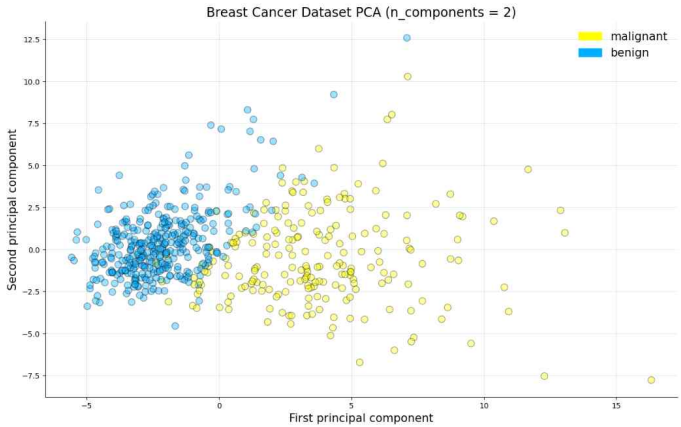
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Before applying PCA, each feature should be centered (zero mean) and with unit variance
#This can be done by using StandardScaler of sklearn Library
canc_norm = StandardScaler().fit(X_canc).transform(X_canc)

pca = PCA(n_components = 2).fit(canc_norm) #Fitting PCA with 2 Components

canc_pca = pca.transform(canc_norm)

print('Number of Features in Breat Cancer DataSet Before PCA : {}₩n₩nNumber of Features in Breast Cancer DataSet After PCA : {}'
      .format(X_canc.shape[1],canc_pca.shape[1]))
```

```
Number of Features in Breat Cancer DataSet Before PCA : 30

Number of Features in Breast Cancer DataSet After PCA : 2
```

- PCA는 여러 속성 값을 연산해야 하므로 스케일링 필요
- n_components = 2 : 2개의 PCA 차원으로 압축

#03 PCA(Principal Component Analysis)

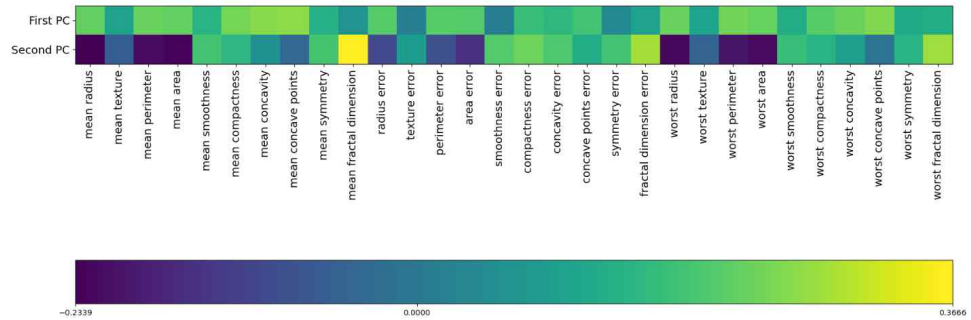
```
plot_labelled_scatter(canc_pca, y_canc, ['malignant', 'benign'],(15,9)) #Using the Helper Function

#Labelling
plt.xlabel('First principal component',fontsize=15)
plt.ylabel('Second principal component',fontsize=15)
plt.title('Breast Cancer Dataset PCA (n_components = 2)',fontsize=17);
```



- 2개의 속성만으로도 일부 겹치는 부분이 존재하나 어느정도는 구분이 된다.

#03 PCA(Principal Component Analysis)

```
fig = plt.figure(figsize=(20,9))
plt.imshow(pca.components_, interpolation = 'none', cmap = 'viridis')
feature_names = list(cancer.feature_names)

plt.gca().set_xticks(np.arange(len(feature_names)));
plt.gca().set_yticks(np.arange(2));
plt.gca().set_xticklabels(feature_names, rotation=90, fontsize=14);
plt.gca().set_yticklabels(['First PC', 'Second PC'], fontsize=14);

plt.colorbar(orientation='horizontal', ticks=[pca.components_.min(), 0,
                                              pca.components_.max()], pad=0.5);
```

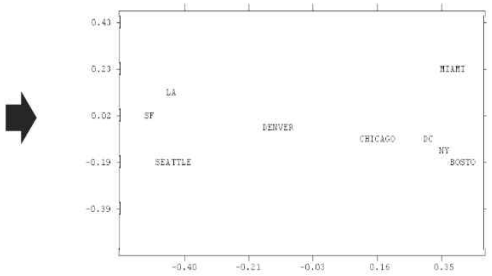


- components_의 각 행에는 주성분이 하나씩 담기며 중요도에 따라 정렬되어 있다.
- 두 개의 주성분을 행으로, 각 열을 features로 나타낸 결과
- 첫 번째 주성분은 모든 특성 사이에 양의 관계가 있고, 두번째 주성분은 섞여있는 것을 알 수 있다.

#04 MDS(Multi dimensional scaling)

- numeric(수치) 변수로만 이루어져 있을 때 사용 가능한 기법.
- 자료들의 '비유사성' 거리를 바탕으로, 다중의 변수들을 2차원 혹은 3차원으로 낮추어 나타내는 기법이다
- 낮은 차원에서의 자료들의 거리가 멀리 떨어져있는 경우 비유사성이 높다고 한다.
- 비유사성 행렬을 이용해 개체들 간 유클리디안 거리를 계산해 점으로 표현한다.
- 즉, 고차원상의 원래 데이터들이 가지는 거리(차이점)을 보전하여 저차원에서 표현하기 좋은 점들의 구성을 찾는다.

		1 BOST	2 NY	3 DC	4 MIAM	5 CHIC	6 SEAT	7 SF	8 LA	9 DENV
1	BOSTON	0	206	429	1504	963	2976	3095	2979	1949
2	NY	206	0	233	1308	802	2815	2934	2786	1771
3	DC	429	233	0	1075	671	2684	2799	2631	1616
4	MIAMI	1504	1308	1075	0	1329	3273	3053	2687	2037
5	CHICAGO	963	802	671	1329	0	2013	2142	2054	996
6	SEATTLE	2976	2815	2684	3273	2013	0	808	1131	1307
7	SF	3095	2934	2799	3053	2142	808	0	379	1235
8	LA	2979	2786	2631	2687	2054	1131	379	0	1059
9	DENVER	1949	1771	1616	2037	996	1307	1235	1059	0

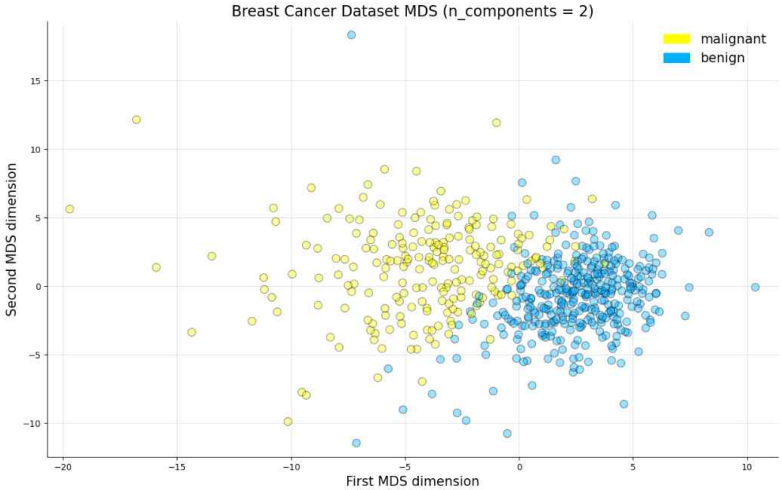


#04 MDS(Multi dimensional scaling)

```
from sklearn.manifold import MDS

mds = MDS(n_components = 2,random_state = 2)

canc_mds = mds.fit_transform(canc_norm)

print('Number of Features in Breat Cancer DataSet Before MDS : {}₩n₩nNumber of Features in Breast Cancer DataSet After MDS : {}'
      .format(X_canc.shape[1],canc_mds.shape[1]))

plot_labelled_scatter(canc_mds, y_canc, ['malignant', 'benign'],(15,9))

plt.xlabel('First MDS dimension',fontsize=15)
plt.ylabel('Second MDS dimension',fontsize=15)
plt.title('Breast Cancer Dataset MDS (n_components = 2)',fontsize=17);
```

- 데이터 포인터 간의 거리를 보존하면서 2차원으로 차원을 축소하였다.
- mds는 분류보다는 2,3차원 위의 점으로 시각화하는 것이 주목적이며, 원데이터를 최대한 보존한다.
- 시각화하는 경우, 데이터 간의 유사도를 확인할 수 있다.(비슷한 변수값을 가지면 가까이에 위치)

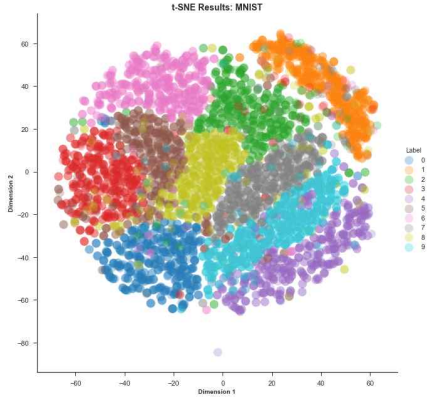
#04 MDS(Multi dimensional scaling)



- MDS는 모든 점들간의 거리 중요도가 같아, 무의미한 정보에 집중한다.
- 그 결과 고차원에서의 원데이터 구조를 보존하지 못한다.

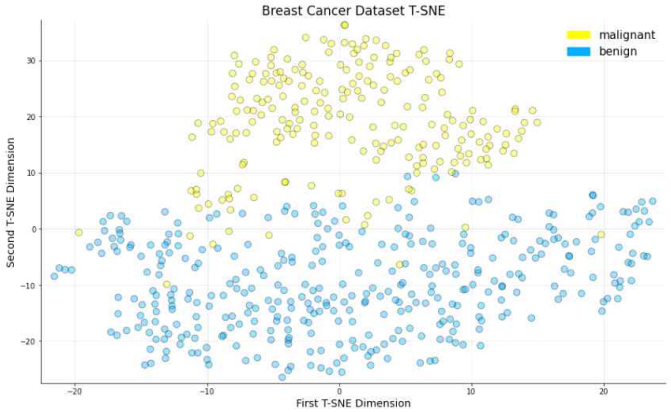
#05 T-SNE(T-distributed Stochastic Neighbor Embedding)

- PCA와 달리 비선형 데이터도 가능한 차원축소 기법이다.
- 유사한 데이터일수록 가깝게 차원 축소를 진행하고, 시각화를 주목적으로 한다.
- 확률 분포를 이용하여 유사도를 계산하고, 최적화한다.



- 위와 같이 동일한 점들의 거리가 최대한 가깝게 일치하도록 한다.

#05 T-SNE(T-distributed Stochastic Neighbor Embedding)




```
from sklearn.manifold import TSNE

tsne = TSNE(random_state = 42)

canc_tsne = tsne.fit_transform(canc_norm)

print('Number of Features in Breat Cancer DataSet Before T-SNE : {}\n\nNumber of Features in Breast Cancer DataSet After T-SNE : {}'
      .format(X_canc.shape[1],canc_tsne.shape[1]))

plot_labelled_scatter(canc_tsne, y_canc, ['malignant', 'benign'],(15,9))

plt.xlabel('First T-SNE Dimension',fontsize=14)
plt.ylabel('Second T-SNE Dimension',fontsize=14)
plt.title('Breast Cancer Dataset T-SNE',fontsize=17);
```

```
Number of Features in Breat Cancer DataSet Before T-SNE : 30

Number of Features in Breast Cancer DataSet After T-SNE : 2
```

- T-SNE는 특히 유방암 데이터 세트와 같이 더 잘 정의된 이웃 패턴을 가진 데이터 세트에서 잘 작동한다.
- 저차원에서의 유사도 최적화 과정이 포함되어서, PCA보다 computation cost가 높으며, 데이터셋 구성에 따라 결과가 달라진다.

THANK YOU