In [ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:
```
import numpy as np
from sklearn.metrics import mean_squared_error
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
plt.rcParams['font.family'] = "sans-serif"
plt.rcParams['font.sans-serif'] = ['Times New Roman']
sns.set_style("whitegrid")
sns.set_style({'font.family':'serif', 'font.serif':'Times New Roman'})
sns.set(font_scale=1.2)
import warnings
warnings.filterwarnings("ignore")
```

Problem 1

In [ ]:
```
data = np.array([[0,1],[1,1],[2,1],[2,3],[3,2],[3,3],[4,5]])
# standardize the data
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
scaler = StandardScaler()
data = scaler.fit_transform(data)
print('standardized data: \n',data)
pca = PCA(n_components=2)
pca_data = pca.fit_transform(data)
print('The two principal components in sorted order:\n', pca.components_)
recon_data = scaler.inverse_transform(pca_data)
print('the new transformed dataset using the first principal component\n',\
      recon_data)
```

```
standardized data:
 [[-1.720618   -0.92827912]
 [-0.91766294 -0.92827912]
 [-0.11470787 -0.92827912]
 [-0.11470787  0.51571062]
 [ 0.6882472  -0.20628425]
 [ 0.6882472   0.51571062]
 [ 1.49120227  1.95970037]]
The two principal components in sorted order:
 [[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]
the new transformed dataset using the first principal component
 [[-0.18984265  1.50971404]
 [ 0.51726413  2.29611156]
 [ 1.22437091  3.08250907]
 [ 2.49599243  1.66829551]
 [ 2.56728845  3.16179981]
 [ 3.20309921  2.45469303]
 [ 5.18182751  1.82687698]]
```

In [ ]:
```
data = np.array([[0,1],[1,1],[2,1],[2,3],[3,2],[3,3],[4,5]])
print(' do not standardize the original data')
pca = PCA(n_components=2)
pca_data = pca.fit_transform(data)
print('The two principal components in sorted order:\n', pca.components_)
recon_data = scaler.inverse_transform(pca_data)
print('the new transformed dataset using the first principal component\n',\
      recon_data)
```

```
 do not standardize the original data
The two principal components in sorted order:
 [[ 0.65908697  0.75206673]
 [ 0.75206673 -0.65908697]]
the new transformed dataset using the first principal component
 [[-8.20287693e-01  1.22729401e+00]
 [ 5.39026785e-04  2.26894507e+00]
 [ 8.21365746e-01  3.31059613e+00]
 [ 2.69461309e+00  1.48485747e+00]
 [ 2.57881614e+00  3.43937786e+00]
 [ 3.51543981e+00  2.52650853e+00]
 [ 6.20951388e+00  1.74242094e+00]]
```

Principal Component Analysis is sensitive to standardizing, and usually we need to do mean centering before PCA. Because by doing Singular Value Decomposisiton, we try to find vectors in linear subspace that can minimize the distortion (which is calculated using L2 norm) and maximize the variance.

Problem 2

In [ ]:
```
file_path = '/content/gdrive/MyDrive/Colab Notebooks/542hw/poly_data.csv'
import pandas as pd
```

In [ ]:
```
df = pd.read_csv(file_path, header=None, sep=' ')
df.head()
```

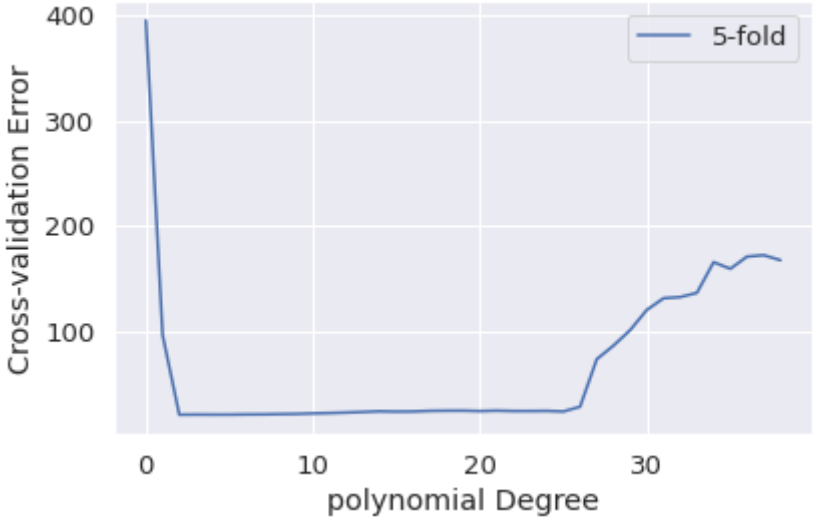
Out[17]:

	0	1
0	0.870951	8.125873
1	0.838599	4.152377
2	-0.044278	24.007025
3	2.149839	-0.523928
4	1.265048	-6.149399

In [ ]:
```
from sklearn.model_selection import KFold
import copy
from sklearn.linear_model import LinearRegression
def polynomial_regression(k,p,x,y):
  new_x = copy.deepcopy(x)
  for i in range(2, p+1):
    new_x = np.hstack((new_x, x**i))
  cv_fold_error = 0
  folds = fold.split(x)
  for t, v in folds:
    X_train, Y_train, X_val, Y_val = new_x[t], y[t], new_x[v], y[v]
    reg = LinearRegression()
    reg.fit(X_train, Y_train)
    y_pred = reg.predict(X_val)
    cv_fold_error += mean_squared_error(Y_val, y_pred)
  return cv_fold_error/5
k=5
x = np.array([df[0]]).T
y = np.array(df[1])
cv_error = []
for p in range(1,40):
  cv_error.append(polynomial_regression(k,p,x,y))
```

In [ ]:
```
print('The value of k I use is 5')
print('polynomial degree = %d fit the data the best' % (cv_error.index(min(
x_axis = [i for i in range(len(cv_error))]
sns.lineplot(x_axis, cv_error, label = '5-fold')
plt.ylabel('Cross-validation Error')
plt.xlabel("polynomial Degree")
plt.legend()
plt.tight_layout()
plt.show()
```

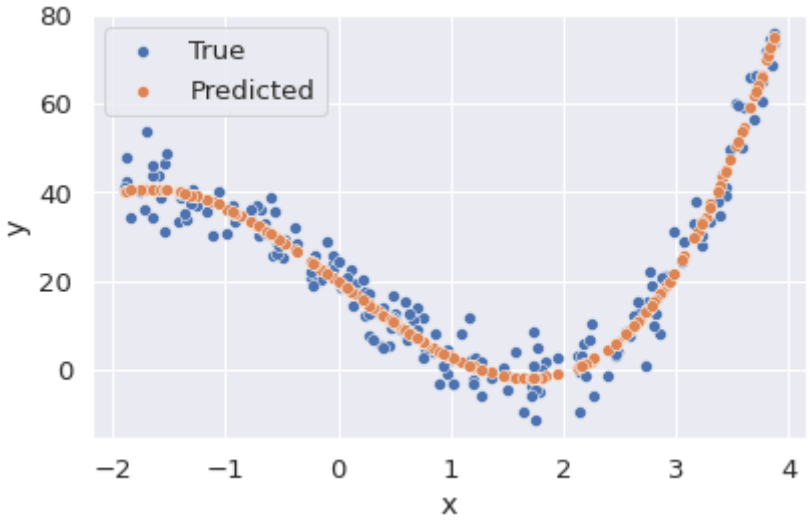
```
The value of k I use is 5
polynomial degree = 3 fit the data the best
```



In [ ]:
```
# best model, regress entire dataset
x = np.array([df[0]]).T
y = np.array(df[1])
new_x = np.hstack((np.ones((x.shape[0], 1)), x))
p = 3
for i in range(2, p+1):
  new_x = np.hstack((new_x, x**i))
beta_ = calculate_beta(new_x, y)
y_pred = new_x @ beta_
print('Tthe polynomial coefficients are', beta_)
print('Predicted y = %.2f + %.2f x + %.2f x^2 + %.2f x^3'%(beta_[0], beta_[

sns.scatterplot(x.T[0], y, label = 'True')
sns.scatterplot(x.T[0], y_pred, label = 'Predicted')
plt.ylabel('y')
plt.xlabel("x")
plt.legend()
plt.tight_layout()
plt.show()
```

```
Tthe polynomial coefficients are [ 19.79510458 -19.03450815  -0.10790684
2.24897906]
Predicted y = 19.80 + -19.03 x + -0.11 x^2 + 2.25 x^3
```



Problem 3

(a)

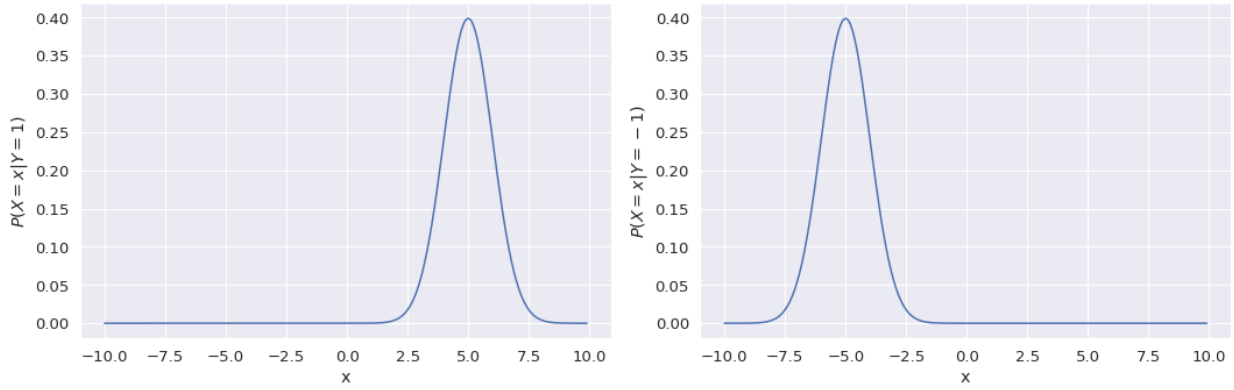
$$P(X = x|Y = +1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5)^2}{2}} = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}}$$

$$P(X = x|Y = -1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x+5)^2}{2}} = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}}$$

Thus

$$
\begin{aligned}
P(X = x, Y = y) &= P(X = x, Y = y|Y = +1)P(Y = +1) \\
&\quad + P(X = x, Y = y|Y = -1)P(Y = -1) \\
&= \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}} \frac{1}{2} + \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}} \frac{1}{2} \\
&= \frac{1}{2\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}}
\end{aligned}
$$

In [ ]:
```
def prob3b(x,y):
  coef = 1/np.sqrt(2*np.pi)
  expo = - (x-5*y)**2/2
  res = coef * np.exp(expo)
  return res
x_axis = np.arange(-10,10,0.1)
y_pos = [prob3b(x,1) for x in x_axis]
y_neg = [prob3b(x,-1) for x in x_axis]
fig, ((ax1, ax2))= plt.subplots(1,2,figsize = (15,5))
sns.lineplot(x_axis, y_pos, ax = ax1)
ax1.set_ylabel('${P(X=x | Y = 1)}$'); ax1.set_xlabel("x")
sns.lineplot(x_axis, y_neg, ax = ax2)
ax2.set_ylabel("${P(X=x | Y = -1)}$"); ax2.set_xlabel("x")
plt.tight_layout()
plt.show()
```



(c)

$$
h^*(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}
$$

(d)

$$
P(X = x|Y = +1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5)^2}{2}} \rightarrow \mathcal{N}(5, 1)
$$

$$
P(X = x|Y = -1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x+5)^2}{2}} \rightarrow \mathcal{N}(-5, 1)
$$

$$\begin{aligned}
Pr(h^*(x) \neq y) &= \mathbb{E}[\mathbb{1}_{h^*(x)\neq y}] \\
&= \frac{1}{2}(Pr(x \geq 0, y = -1) + Pr(x < 0, y = 1)) \\
&= \frac{1}{2}((1 - Pr(x + 5 < 5 | y = -1)) + Pr(x - 5 < -5 | y = 1)) \\
&= \frac{1}{2}(1 - \Phi(5) + \Phi(-5)) \\
&= 1 - \Phi(5)
\end{aligned}$$

(e)

$$P(X = x|Y = +1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5)^2}{2}}$$

$$P(X = x|Y = -1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x+5)^2}{2}}$$

$$P(X = x, Y = y) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(x-5y)^2}{2}}$$

Thus

$$\begin{aligned}
P(X = x) &= P(X = x|Y = +1)P(Y = +1) + P(X = x|Y = -1)P(Y = -1) \\
&= \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-5)^2}{2}} \frac{1}{2} + \frac{1}{\sqrt{2\pi}} e^{-\frac{(x+5)^2}{2}} \frac{1}{2} \\
&= \frac{1}{2\sqrt{2\pi}} (e^{-\frac{(x-5)^2}{2}} + e^{-\frac{(x+5)^2}{2}})
\end{aligned}$$

Thus

$$\begin{aligned}
P(Y = +1|X = x) &= \frac{P(Y = +1, X = x)}{P(X = x)} \\
&= \frac{\frac{1}{2\sqrt{2\pi}} e^{-\frac{(x-5)^2}{2}}}{\frac{1}{2\sqrt{2\pi}} (e^{-\frac{(x-5)^2}{2}} + e^{-\frac{(x+5)^2}{2}})} \\
&= \frac{e^{-\frac{(x-5)^2}{2}}}{e^{-\frac{(x-5)^2}{2}} + e^{-\frac{(x+5)^2}{2}}} = \frac{1}{1 + \frac{e^{-\frac{(x+5)^2}{2}}}{e^{-\frac{(x-5)^2}{2}}}} = \frac{1}{1 + e^{(-\frac{(x+5)^2}{2} + \frac{(x-5)^2}{2})}} \\
&= \frac{1}{1 + e^{-10x}}
\end{aligned}$$

Therefore the distribution satisfies this assumption:

$$P(Y = +1|X = x) = \frac{1}{1 + e^{-\beta_0 - \beta_1 x}}$$

where

$$\beta_0 = 0, \beta_1 = 10$$

Problem 4

Example of 4 points in a 2D plane: if there are two clusters in total, and

cluster 0 consists of ($\sqrt{t}$, a) and ($\sqrt{t}$, -a),

cluster 1 consists of ($-\sqrt{t}$, a) and ($-\sqrt{t}$, -a),

then the two centroids are ($\sqrt{t}$, 0) and ($-\sqrt{t}$, 0).

Thus OPT = $min_{c_1,\ldots,c_k} \sum i = 1^n ||x_i - c(x_i)||_2^2$ = t*OPT for any t>1, any $a \in \mathcal{R}$

For n data points, p dimensions and k clusters,

consider data points ($\sqrt{t}$, a, 0, ... 0_p), ($-\sqrt{t}$, a, 0, ... 0_p), ($\sqrt{t}$, 2a, 0, ... 0_p), ($-\sqrt{t}$, 2a, 0, ... 0_p)...

If k is 2n, then the centroids can be (0, a, 0, ... 0_p), (0, 2a, 0, ... 0_p)... if k is smaller or larger then the clusters for above centroids merges or splits. Thus OPT = $min_{c_1,\ldots,c_k} \sum i = 1^n ||x_i - c(x_i)||_2^2$ = t*OPT for any t>1 still stands

Problem 5

In [ ]:
```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
```

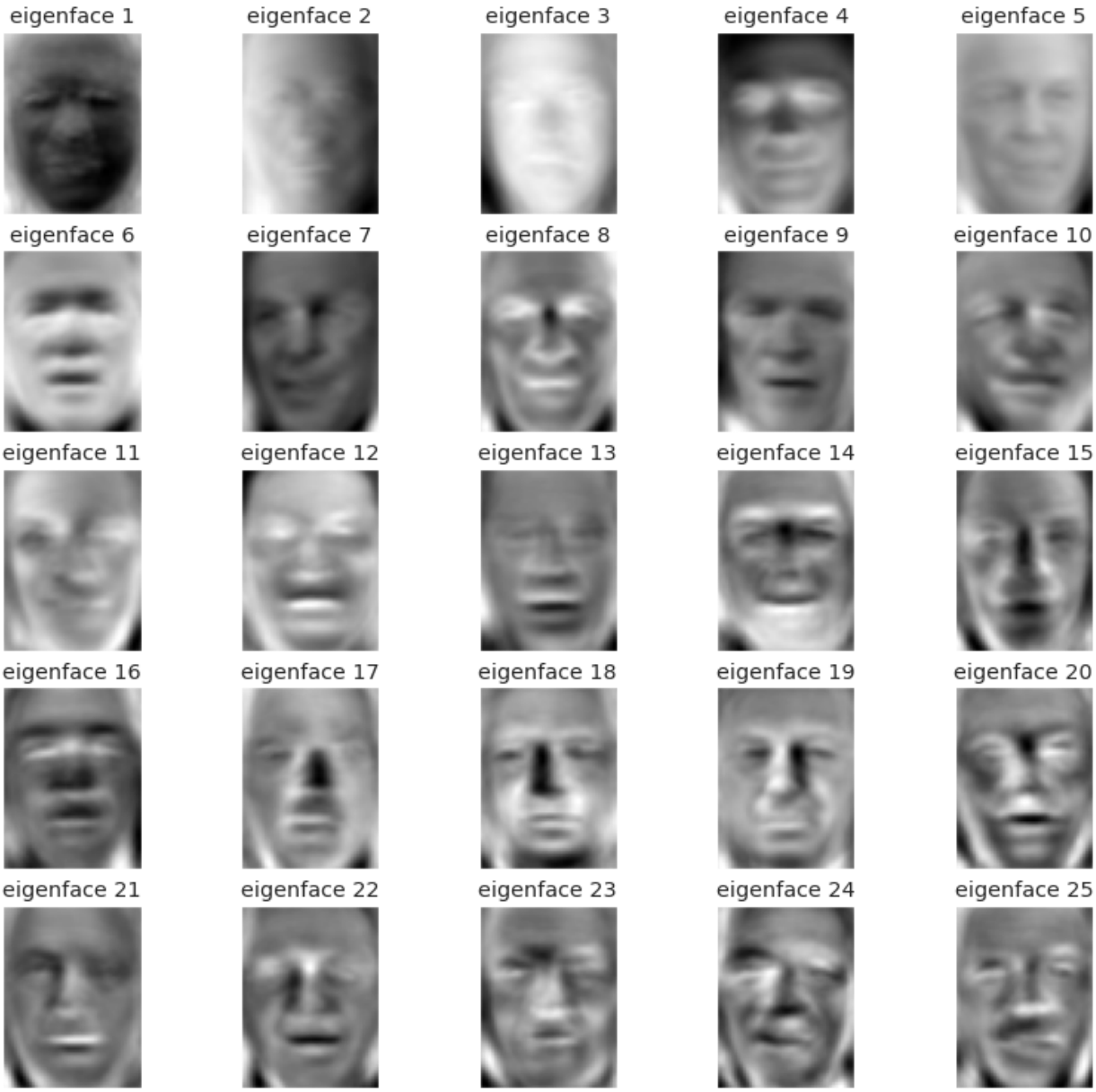
In [ ]:
```
# sklearn version issue, bypass import error of RandomizedPCA
# according to https://stackoverflow.com/questions/54494785/sklearn-0-20-2-
from sklearn.decomposition import PCA as RandomizedPCA
def random_pca(data, n_pc):
  pca = RandomizedPCA(n_components=n_pc, svd_solver='randomized')
  pca.fit(data)
  return pca.components_

def plot_portraits(images, titles, h, w, n_row, n_col):
    plt.figure(figsize=(2.2 * n_col, 2.2 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.20)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
        plt.title(titles[i])
        plt.xticks(()); plt.yticks(())

n_samples, h, w = faces.images.shape
X_images = faces.data
y_names = faces.target
n_features = X_images.shape[1]
print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)

n_pc = 150
C_150 = random_pca(X_images, n_pc)
C_25 = C_150[:25,:]
eigenfaces = C_25.reshape((25, h, w))
eigenface_titles = ["eigenface %d" % (i+1) for i in range(eigenfaces.shape[
plot_portraits(eigenfaces, eigenface_titles, h, w, 5, 5)
```

```
Total dataset size:
n_samples: 1348
n_features: 2914
```


eigenface 1
eigenface 2
eigenface 3
eigenface 4
eigenface 5
eigenface 6
eigenface 7
eigenface 8
eigenface 9
eigenface 10
eigenface 11
eigenface 12
eigenface 13
eigenface 14
eigenface 15
eigenface 16
eigenface 17
eigenface 18
eigenface 19
eigenface 20
eigenface 21
eigenface 22
eigenface 23
eigenface 24
eigenface 25

In [ ]:
```
def reconstruction(data, components):
  subspace = data @ components.T # reconstruct with the first 150 principal
  recon_data = (subspace @ components)
  return recon_data
selected_img = X_images[:8]
# print('original')
plot_portraits(selected_img, ['original'+str(i) for i in range(8)], h, w, n
# print('reconstructed')
recon_data = reconstruction(selected_img,C_150)
plot_portraits(recon_data, ['recon'+str(i) for i in range(8)], h, w, n_row=
```


recon0
recon1
recon2
recon3
recon4
recon5
recon6
recon7