This homework is done by Tianwei Mo.

1.

a) [1.5 2.5 3.5]

b)
```
[[ 1.25   0.25 -1.25]
 [ 0.25   0.75   0.75]
 [-1.25   0.75   2.75]]
```

c) Eigenvalue = [3.56166464 1.1733803   0.01495506]

Eigenvectors =
```
[[-0.45056922 -0.6667717184 -0.59363515]
 [ 0.19247228 -0.72187235    0.66472154]
 [ 0.87174641 -0.18524476 -0.45358856]]
```

d) PCA coefficients =
```
[[-2.95145599 -0.17610969 -0.0888421 ]
 [ 1.37104342 -1.69406159    0.0198819 ]
 [-0.30682473    0.78694448    0.19125108]
 [ 1.8872373      1.0832268    -0.12229089]]
```

e) Xhat =
```
[[3.0000000e+00 2.0000000e+00 1.0000000e+00]
 [2.0000000e+00 4.0000000e+00 5.0000000e+00]
 [1.0000000e+00 2.0000000e+00 3.0000000e+00]
 [4.4408921e-16 2.0000000e+00 5.0000000e+00]]
```

f) Xhat =
```
[[2.05905526 0.95970224]
 [3.98678407 5.0090182 ]
 [1.87287129 3.0867493 ]
 [2.08128939 4.94453025]]
```

g) e = 0.014955061432806003

python code:

```python
x = np.array([
    [3, 2, 1],
    [2, 4, 5],
    [1, 2, 3],
    [0, 2, 5],
])
mean = np.mean(x, axis=0)
cov = np.zeros((3, 3))
for i in range(3):
    for j in range(3):
        for n in range(4):
            cov[i, j] += (x[n, i] - mean[i]) * (x[n, j] - mean[j])
cov = cov / 4
eigvalue, eigvector = np.linalg.eig(cov)

Z = (x - mean).dot(eigvector)

xhat = Z.dot(eigvector.T) + mean

Z_2 = Z[:, :2].dot(eigvector[:, :2].T) + mean

e1 = np.mean(np.sum((Z_2 - x) ** 2, axis=1))
# print(mean)
print(cov)
print(eigvalue, eigvector)
print(Z)
print(xhat)
print(Z_2)
print(e1, eigvalue[-1])

```

2.

a) x-u = [1, 3, 2]
   a1 = v1^T * (x-u) = 4/sqrt2
   a2 = v2^T * (x-u) = -2/sqrt2

b) Approximation = u + a1v1 + a2v2 = [1, 0, 2] + [2, 2, 0] + [-1, 1, 0] = [2, 3, 2]

c) ||x - xhat||^2 = 4

3. The python code:

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
X = np.zeros(3, 5)
y = np.zeros(3)

nfold = 4

# Create a K-fold object
kf = KFold(n_splits=nfold)
kf.get_n_splits(X)

# Number of PCs to try
ncomp_test = np.arange(2,12)
num_nc = len(ncomp_test)

# Accuracy:  acc[icomp,ifold]  is test accuracy when using `ncomp = ncomp_test[icomp]` in fold `ifold`.
acc = np.zeros((num_nc,nfold))

# Loop over number of components to test
for icomp, ncomp in enumerate(ncomp_test):

    # Look over the folds
    for ifold, I in enumerate(kf.split(X)):
        Itr, Its = I

        # Split data into training
        Xtr, Xts, ytr, yts = X[Itr], X[Its], y[Itr], y[Its]
```

```python

        # Create a scaling object and fit the scaling on the training data
        scaling = StandardScaler()
        scaling.fit(Xtr, ytr)

        # Fit the PCA on the scaled training data
        Xtrs = scaling.transform(Xtr)
        pca = PCA(n_components=ncomp, svd_solver='randomized', whiten=True)
        pca.fit(Xtrs, ytr)
        Ztr = pca.transform(Xtrs)

        # Train a classifier on the transformed training data
        # Use a logistic regression classifier
        logreg = LogisticRegression(multi_class='auto', solver='lbfgs')

        # Transform the test data through data scaler and PCA
        Xtss = scaling.transform(Xts)
        pca.fit(Xtss, yts)
        Zts = pca.transform(Xtss)

        # Predict the labels the test data
        logreg.fit(Ztr, ytr)
        yhat = logreg.predict(Zts)

        # Measure the accuracy
        acc[icomp, ifold] = np.mean(yhat == yts)

acc_mean = np.mean(acc, axis=1)
optimal_index = np.argmax(acc_mean)
optimal_order = ncomp_test[optimal_index]
```

```python
        # Measure the accuracy
        acc[icomp, ifold] = np.mean(yhat == yts)

acc_mean = np.mean(acc, axis=1)
optimal_index = np.argmax(acc_mean)
optimal_order = ncomp_test[optimal_index]
print('optimal normal order is {}. Accurcy is {}'.format(optimal_order, acc_mean[optimal_order]))
```

4.

```python
import numpy as np
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

def plt_face(x):
    h = 50
    w = 37
    plt.imshow(x.reshape((h, w)), cmap=plt.cm.gray)
    plt.xticks([])
    plt.yticks([])

X = np.arange(1000 * 28 * 28).reshape(1000, 28, 28)
Y = np.reshape(X, (1000, 28*28))

Y = Y[:500, :]
nc = 100

pca = PCA(n_components=nc)
pca.fit(Y)
Z = pca.transform(Y)
Yhat = pca.inverse_transform(Z)
plt_face(Yhat)
```

5. The python function:

```python
scaling = StandardScaler()
scaling.fit(X)
Z = scaling.transform(X)

U, S, Vtr = svd(Z, full_matrices = False)

# i)
PCs = Vtr
mean = np.mean(X, axis=0)

# ii)
PoV_len = S.shape[0]
lam = S**2
PoV = np.cumsum(lam)/np.sum(lam)
min_PoV = np.min(np.where(PoV >= 0.9)) + 1

# iii)
Z = U[:,:min_PoV]*S[None,:min_PoV]
W = PCs[:min_PoV, :]
print(Z.shape)
Xhat = mean + Z.dot(W)
```