Problem 1:

For the python implementation code, I get the time list which is the list of time series. Then set the words number is 6 and alpha is 3. Then set start characteristic as 'a' and points. Then do normalization to the time series list and transfer into SAX form.
The SAX implementation code is as following:

```
mport numpy as np
mport math

ime_list = [0.11, 0.22, 0.33, 0.44, 0.55, 0.66, 0.77, 0.88, 0.99, 1.11, 1.22, 1.33, 1.44, 1.55, 1.66,
      1.77, 1.88, 1.99, 2.11, 2.22, 2.33, 2.44, 2.55, 2.66, 2.77, 2.88, 2.99, 3.00, 4.00, 5.00]  # list of
ords_num = 6  # words's number
lpha = 3  # alpha number
har = ord('a')
oints = {'0': [-0.40, 0.40],
              '1': [-0.60, 0, 0.60],
              '2': [-0.80, -0.20, 0.20, 0.80],
              '3': [-0.90, -0.40, 0, 0.40, 0.90],
              '4': [-1.09, -0.60, -0.20, 0.20, 0.60, 1.09],
              '5': [-1.20, -0.70, -0.30, 0, 0.30, 0.70, 1.20],
              }  # points
eta = points[str(alpha)]  # beta
 do normalization
 = np.asanyarray(time_list)
ormalize = (s - np.nanmean(s)) / np.nanstd(s)
 do transfer
aa_list = []
 = len(normalize)
eil_num = math.ceil(n / words_num)
or i in range(0, n, ceil_num):
   temp_ts = normalize[i:i + ceil_num]
   paa_list.append(np.mean(temp_ts))
   i = i + ceil_num
 do SAX transfer
```

```
transfer = paa_list
len_transfer = len(transfer)
len_beta = len(beta)
str = ''
for i in range(len_transfer):
    letter = False
    for j in range(len_beta):
        if np.isnan(transfer[i]):
            str += '-'
            letter = True
            break
        if transfer[i] < beta[j]:
            str += chr(char + j)
            letter = True
            break
    if not letter:
        str += chr(char + len_beta)
print(str)
```

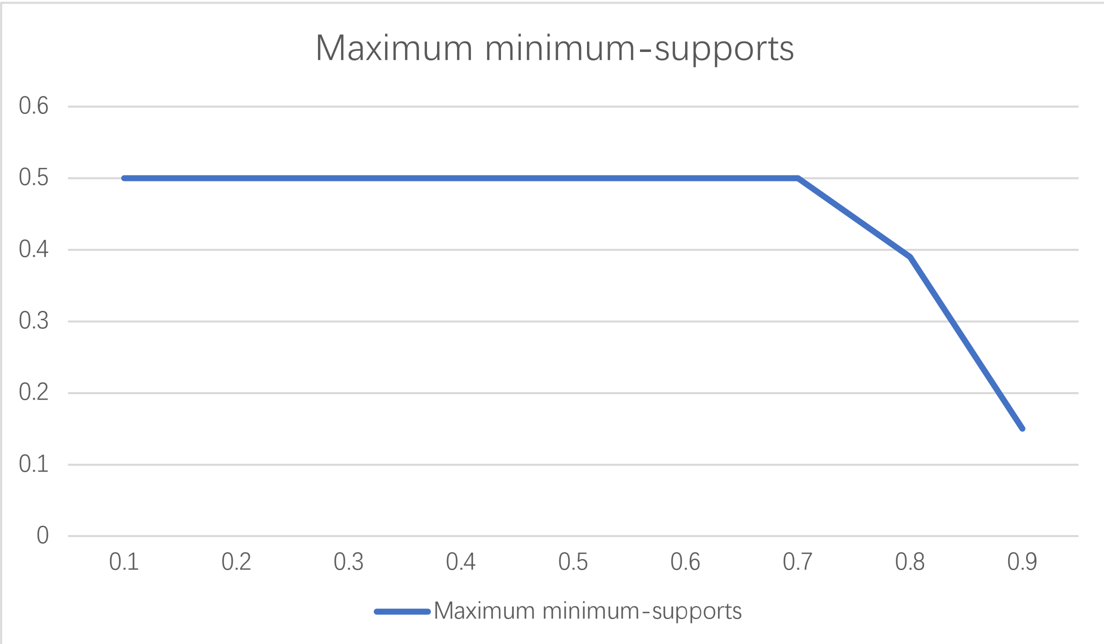
The input is time_list which is the time series, the output is a string.
Output is as following:

```
abcdef

Process finished with exit code 0
```

Problem 2:

For the top 10 rules with confidence above 0.8:

```
Minimum support: 0.3 (1388 instances)
Minimum metric <confidence>: 0.8
Number of cycles performed: 14

Generated sets of large itemsets:

Size of set of large itemsets L(1): 25

Size of set of large itemsets L(2): 69

Size of set of large itemsets L(3): 20

Best rules found:

 1. biscuits=t vegetables=t 1764 ==> bread and cake=t 1487    <conf:(0.84)> lift:(1.17) lev:(0.05) [217] conv:(1.78)
 2. total=high 1679 ==> bread and cake=t 1413    <conf:(0.84)> lift:(1.17) lev:(0.04) [204] conv:(1.76)
 3. biscuits=t milk-cream=t 1767 ==> bread and cake=t 1485    <conf:(0.84)> lift:(1.17) lev:(0.05) [213] conv:(1.75)
 4. biscuits=t fruit=t 1837 ==> bread and cake=t 1541    <conf:(0.84)> lift:(1.17) lev:(0.05) [218] conv:(1.73)
 5. biscuits=t frozen foods=t 1810 ==> bread and cake=t 1510    <conf:(0.83)> lift:(1.16) lev:(0.04) [207] conv:(1.69)
 6. frozen foods=t fruit=t 1861 ==> bread and cake=t 1548    <conf:(0.83)> lift:(1.16) lev:(0.05) [208] conv:(1.66)
 7. frozen foods=t milk-cream=t 1826 ==> bread and cake=t 1516    <conf:(0.83)> lift:(1.15) lev:(0.04) [201] conv:(1.65)
 8. baking needs=t milk-cream=t 1907 ==> bread and cake=t 1580    <conf:(0.83)> lift:(1.15) lev:(0.04) [207] conv:(1.63)
 9. milk-cream=t fruit=t 2038 ==> bread and cake=t 1684    <conf:(0.83)> lift:(1.15) lev:(0.05) [217] conv:(1.61)
10. baking needs=t biscuits=t 1764 ==> bread and cake=t 1456    <conf:(0.83)> lift:(1.15) lev:(0.04) [186] conv:(1.6)
```



Confidence values: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

Maximum minimum-supports: 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.50, 0.39, 0.15

We can get to know when the confidence is 0.7, the maximum minimum support begins to decrease. And the lower one is 0.15.

Problem 3:

From the graph, we can get equations:

V1 = V2/3

V2 = V1/2

V3 = V2/3 + V4

V4 = V5

V5 = V1/2 + V2/3 + V3

Then I use python code to calculate the page rank. First, get the data matrix. Then set the probability is 0.86 which is random teleports. Then I set the threshold as 0.000000001 and set the data size as 5. Then get the initial page rank and data matrix.

Then there is a while loop to calculate the page rank until the distance is less than the threshold which is 0.000000001.

```
import numpy as np

data_matrix = np.array([[0, 1 / 3, 0, 0, 0],
                        [0.5, 0, 0, 0, 0],
                        [0, 1 / 3, 0, 1, 0],
                        [0, 0, 0, 0, 1],
                        [0.5, 1 / 3, 1, 0, 0]])  # data matrix represents edge weights
probability = 0.86
threshold = 0.000000001
data_size = 5
Page_jump = np.array([1 / data_size] * (data_size ** 2)).reshape(data_size, data_size)
Page_rank = np.array([1 / data_size] * data_size).reshape(data_size, 1)  # page rank
data_matrix = probability * data_matrix + (1 - probability) * Page_jump  # data matrix
while True:
    rank_new = np.dot(data_matrix, Page_rank)
    dist = np.linalg.norm(Page_rank - rank_new)  # calculate the distance
    Page_rank = rank_new
    if dist < threshold:  # break when smaller than threshold
        break
print(Page_rank)
```

Then the page rank is as following:

```
[[0.04109193]
 [0.04566953]
 [0.29848523]
 [0.29929454]
 [0.31545876]]
```

Problem 4:

For the python implementation code, I set the neighbor's number is 30, user id is 1, movie id is 2. And then I read the file from the u.data to get the user item matrix. And then I get the mean raing for each user and calculate the user similarity and movie similarity. And then get the most similarity instance between neighbors whose number is 30. And then I use what I have calculated to predict the user-based CF and item-based CF.

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# calculate the mean without the 0
def mean(row):
    row = row[row > 0]
    return sum(row) / len(row)


neighboors_num = 30
user_id = 1
movie_id = 2

# read data from csv
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=r_cols, encoding='latin-1')
# get user-movie matrix
user_movie_matrix = ratings.pivot(index='user_id', columns='movie_id', values='rating').fillna(0)
# each user's mean rating
_user_mean_rate = user_movie_matrix.apply(mean, axis=1)
_movie_mean_rate = user_movie_matrix.T.apply(mean, axis=1)
# calculate the user similarity and movies similarity
user_similarity = pd.DataFrame(cosine_similarity(user_movie_matrix), index=user_movie_matrix.index, columns=user_movie_matrix.index)
movie_similarity = pd.DataFrame(cosine_similarity(user_movie_matrix.T), index=user_movie_matrix.columns, columns=user_movie_matrix.columns)
```

```python
# get the most similarest instance between neignbors, the neighbors' number here is 30
_user_similarest = pd.DataFrame(index=user_similarity .index, columns=range(1, neighboors_num))
for index in _user_similarest.index:
    _user_similarest.loc[index, :neighboors_num - 1] = user_similarity .loc[:, index].sort_values(ascending=False)[
                                                        1:neighboors_num].index

_movie_similarest = pd.DataFrame(index=_movie_similarity .index, columns=range(1, neighboors_num))
for index in _movie_similarest.index:
    _movie_similarest.loc[index, :neighboors_num - 1] = movie_similarity .loc[:, index].sort_values(ascending=False)[
                                                         1:neighboors_num].index
```

```python
# predict the user-based CF
user_prediction = 0
sums = user_similarity .loc[user_id, _user_similarest.loc[user_id, :]].sum()
for user in _user_similarest.loc[user_id, :]:
    if user_movie_matrix.loc[user, movie_id] != 0:
        user_prediction += user_similarity .loc[user_id, user] * (
                user_movie_matrix.loc[user, movie_id] - _user_mean_rate[user])
user_prediction /= sums
user_prediction += _user_mean_rate[user_id]
print('user_CF rating is {}'.format(user_prediction))

# predict the item-based CF
movie_prediction = 0
sums = movie_similarity.loc[movie_id, _movie_similarest.loc[movie_id, :]].sum()
for movie in _movie_similarest.loc[movie_id, :]:
    if user_movie_matrix.loc[user_id, movie] != 0:  # For here, 0 is that user doesn't give the rate
        movie_prediction += movie_similarity.loc[movie_id, movie] * (user_movie_matrix.loc[user_id, movie] - _movie_mean_rate[movie])
movie_prediction /= sums
movie_prediction += _movie_mean_rate[movie_id]
print('item_based rating is {}'.format(movie_prediction))
```

Then the user-based rating and item-based rating is as following:

```
user-based rating is 3.5493460338452056
item-based rating is 3.322041167796573
```

Problem 5:

For the python implementation code, I get the train data by using numpy's random method. Then get the outlier and get the model for outlier detection. Then use this model the predicted label of the training samples. Then print out the LoF scores for any instances in the dataset.

```python
import numpy as np
from sklearn.neighbors import LocalOutlierFactor

np.random.seed(10)

# get train data
inliers = 0.3 * np.random.randn(80, 2)
inliers = np.r_[inliers + 2, inliers - 2]
# get outliers
outliers = np.random.uniform(low=-4, high=4, size=(20, 2))
store = np.r_[inliers, outliers]
outliers_num = len(outliers)
ground_truth = np.ones(len(store), dtype=int)
ground_truth[-outliers_num:] = -1
# get model for outlier detection
clf = LocalOutlierFactor(n_neighbors=20, contamination=0.1)
# compute the predicted labels of the training samples
pred = clf.fit_predict(store)
scores = clf.negative_outlier_factor_
print(scores)
```

Result is as following:

```
[-1.11598986 -1.26279915 -1.08379997 -0.94601932 -0.96350835 -1.11516036
 -1.21636303 -0.98521068 -1.10434859 -1.25563328 -1.87631854 -1.57791083
 -1.21133292 -1.220846   -1.00582923 -0.97979684 -0.97651469 -1.0817715
 -0.98126224 -1.11753836 -1.02058085 -1.07062542 -1.01713436 -0.96139707
 -0.98554968 -0.96174037 -1.02843256 -1.02502362 -1.2878305  -0.95111593
 -0.96908965 -1.12768787 -0.96958491 -1.32565593 -0.95691497 -0.99421711
 -1.05961813 -1.57612032 -1.30970443 -1.17126629 -1.33031231 -1.46488408
 -1.5679798  -0.96730829 -1.44883018 -1.03797347 -1.01046275 -1.06912348
 -0.96884983 -1.43182978 -1.48149842 -1.06167876 -0.95869308 -1.04330012
 -1.12089656 -0.98462664 -1.38927652 -0.95533564 -0.96946391 -1.29343283
 -1.21764168 -1.49234867 -1.11506759 -1.02466979 -1.15733403 -1.09072634
 -1.00602592 -1.16104067 -1.07111865 -1.3037803  -1.19598857 -0.96839646
 -1.27864733 -1.13833584 -1.09141814 -1.01112978 -1.12834728 -0.95290174
 -1.02890193 -1.95604036 -1.15736191 -1.26615243 -1.09066863 -0.94500215
 -0.96346062 -1.11544888 -1.21957687 -0.98381069 -1.10747244 -1.27721425
 -1.87605629 -1.5738839  -1.24056974 -1.2216467  -1.00536465 -0.97979684
 -0.97579291 -1.09015137 -0.97966011 -1.15952455 -1.02110475 -1.07047142
 -1.0168605  -0.96126041 -0.98448739 -0.9614388  -1.0280595  -1.03977844
 -1.39030304 -0.95106859 -0.96908965 -1.12938931 -0.96912931 -1.33094044
 -0.95691497 -0.99246052 -1.05858636 -1.61954077 -1.2958299  -1.17105572
 -1.37757666 -1.46856036 -1.56411716 -0.96700507 -1.44586867 -1.05894493
 -1.00969403 -1.07523838 -0.96868519 -1.43362768 -1.4778186  -1.06476304
 -0.95869308 -1.04259945 -1.12756826 -0.98389916 -1.47247145 -0.95533564
 -0.9692993  -1.29239639 -1.21771389 -1.55046382 -1.15586577 -1.03876349
 -1.21469685 -1.09009263 -1.00574242 -1.16429328 -1.09000333 -1.36710591
 -1.19529721 -0.96794139 -1.28426581 -1.13799111 -1.08886393 -1.0148485
 -1.13896137 -0.95289956 -1.03197789 -1.99483796 -5.8107518  -3.855825
 -1.36020039 -3.84089813 -3.39644569 -7.69431141 -3.70046329 -4.9605476
 -2.22669367 -3.19334831 -1.15316985 -6.99379996 -1.96895121 -1.19779872
 -5.10678886 -9.95281713 -8.07506005 -7.20611154 -4.9876579  -3.23786018]
```