1) part 1:

a)

```
frequent_itemsets = ap.extract_items(4, 7)      #a
print(frequent_itemsets[1])                     #a
```

```
[{'a'}, {'b'}, {'c'}, {'d'}, {'e'}]
```

b)

```
frequent_itemsets = ap.extract_items(4, 7)      #b
print(frequent_itemsets[2])                     #b
```

```
[{'b', 'a'}, {'c', 'a'}, {'e', 'a'}, {'b', 'e'}, {'c', 'e'}, {'e', 'd'}]
```

c)

```
frequent_itemsets = ap.extract_items(7, 7)      #c
print(frequent_itemsets[2])                     #c
```

```
[{'e', 'b'}]
```

d)

```
frequent_itemsets = ap.extract_items(4, 7)                      #d
confident_rules = ap.extract_rules(1, frequent_itemsets, 6)     #d
print([str(x) for x in confident_rules[0]])                     #d
print([str(x) for x in confident_rules[1]])                     #d
```

```
["{'c'} -> {'a'}", "{'c'} -> {'e'}", "{'c', 'a'} -> {'e'}", "{'c', 'e'} -> {'a'}"]
["{'c'} -> {'e', 'a'}"]
```

e)

```
conf = ap.confidence(Rule({'b'}, {'e'}))        #e
print(conf)                                     #e
```

```
0.875
```

Part 2:

a)

```
frequent_itemsets = ap.extract_items(2, 5)      #a
print(frequent_itemsets[1])                     #a
print(frequent_itemsets[2])                     #a
print(frequent_itemsets[3])                     #a
```

```
[{'a'}, {'b'}, {'c'}, {'e'}]
[{'a', 'c'}, {'c', 'b'}, {'e', 'b'}, {'e', 'c'}]
[{'e', 'c', 'b'}]
```

b)

```
frequent_itemsets = ap.extract_items(2, 5)                                  #b
confident_rules = ap.extract_rules(0.65, frequent_itemsets, 4)              #b
print(len(confident_rules[0]) + len(confident_rules[1]) + len(confident_rules[2]))  #b
```

```
14
```

c)

```
frequent_itemsets = ap.extract_items(2, 5)                  #c
confident_rules = ap.extract_rules(0.8, frequent_itemsets, 4)   #c
print([str(x) for x in confident_rules[0]])                 #c
```

```
["{'a'} -> {'c'}", "{'e'} -> {'b'}", "{'b'} -> {'e'}", "{'e', 'c'} -> {'b'}", "{'b', 'c'} -> {'e'}"]
```

d)

```
conf = ap.confidence(Rule({'e'}, {'c'}))    #d
print(conf)                                 #d
```

```
0.6666666666666666
```

e)

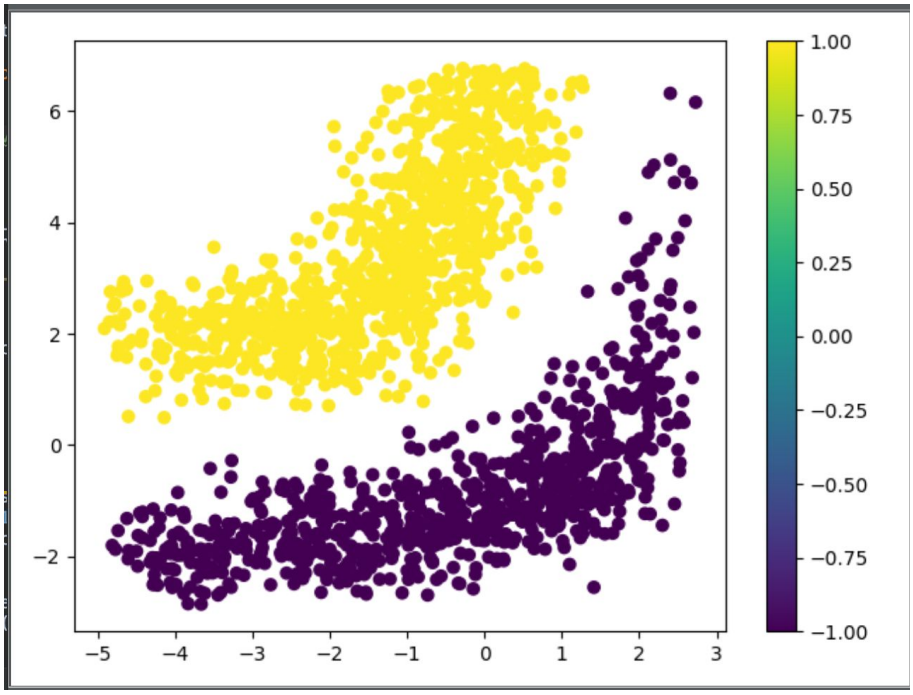
```
supp = ap.support({'b', 'c'})       #e
print(supp)                         #e
```

```
2
```

2) part 1:

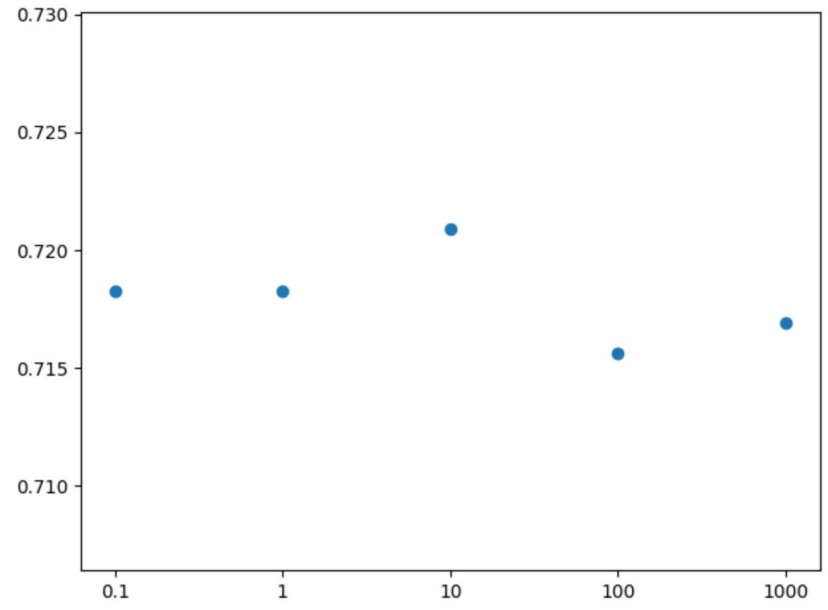
a)

```
# A
train, test = train_test_split(dataset, test_size=0.3)
train, validation = train_test_split(train, test_size=0.3)
plotter.figure()
a = train.iloc[:, 0]
b = train.iloc[:, 1]
c = train.iloc[:, 2]
plotter.scatter(a, b, c=c)
plotter.colorbar()
plotter.show()
```



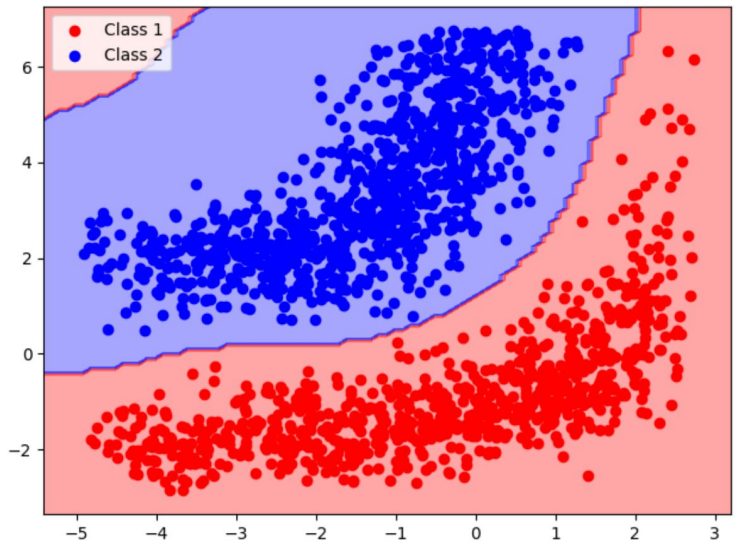
b)

```
# B
params = [0.1, 1, 10, 100, 1000]
accuracy = []
best_param = -1
for param in params:
    clf = svm.SVC(C=param, gamma='auto')
    a = train.iloc[:, 0:1]
    b = train.iloc[:, 2]
    clf = clf.fit(a, b)
    c = validation.iloc[:, 0:1]
    x = clf.predict(c)
    d = validation.iloc[:, 2]
    tmp = accuracy_score(d, x)
    if tmp > best_param:
        best_param = tmp
    accuracy.append(tmp)

params_labels = ['0.1', '1', '10', '100', '1000']

plotter.scatter(params_labels, accuracy, label='Hyperparameters Accuracy')
plotter.show()
```



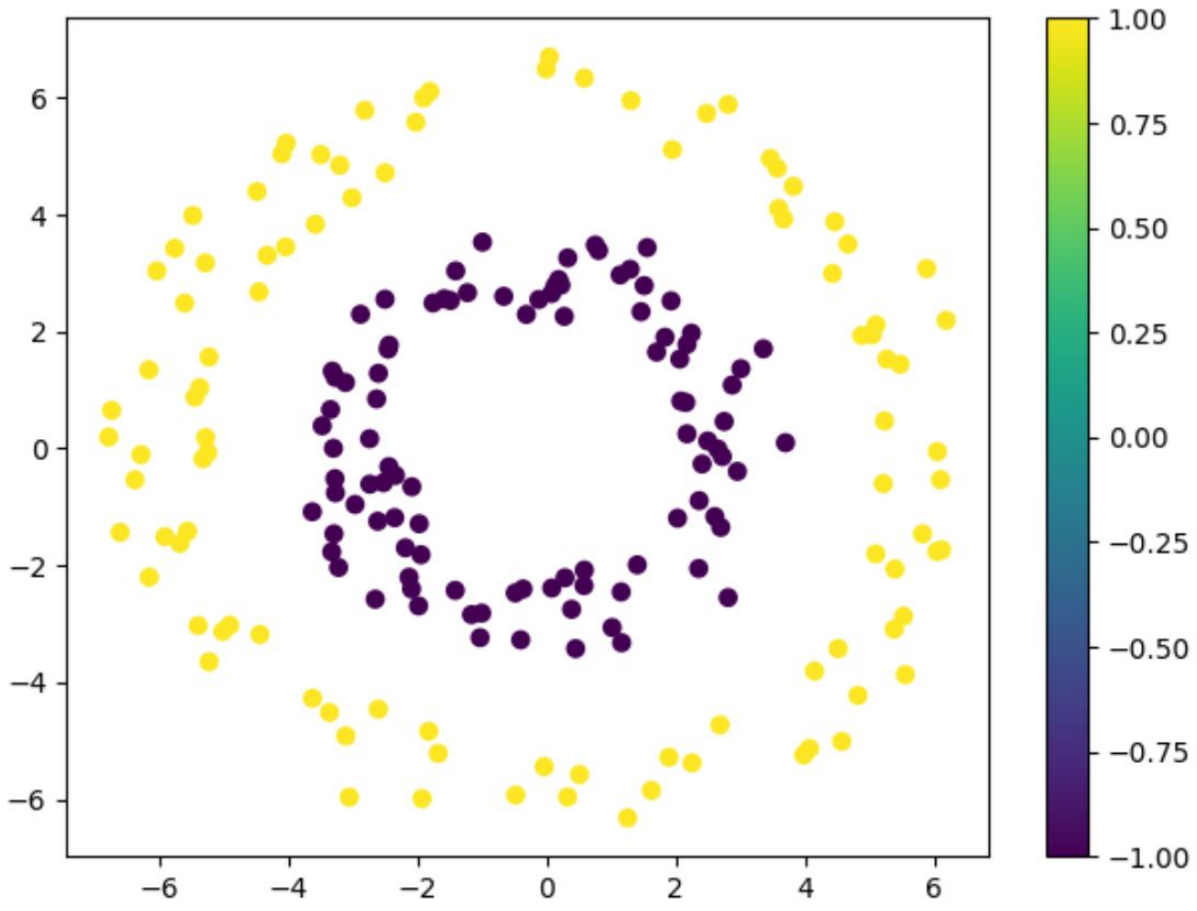
c)

```
# C
clf = svm.SVC(C=best_param)
train1 = train.values[:, :-1]
train2 = train.values[:, -1]
clf.fit(train1, train2)

class1 = train.values[train2 == -1]
class2 = train.values[train2 == 1]

minC10 = min(class1[:, 0])
maxC10 = max(class1[:, 0])
minC11 = min(class1[:, 1])
maxC11 = max(class1[:, 1])
minC20 = min(class2[:, 0])
maxC20 = max(class2[:, 0])
minC21 = min(class2[:, 1])
maxC21 = max(class2[:, 1])

x_min = min(minC10, minC20) - .5
x_max = max(maxC10, maxC20) + .5
y_min = min(minC11, minC21) - .5
y_max = max(maxC11, maxC21) + .5

clrs = colors.ListedColormap(['red', 'blue'])
boundaries = [-2, 0, 2]
norms = colors.BoundaryNorm(boundaries, clrs.N)

np1 = np.arange(x_min, x_max, .1)
np2 = np.arange(y_min, y_max, .1)
x_new, y_new = np.meshgrid(np1, np2)
z = clf.predict(np.c_[x_new.ravel(), y_new.ravel()])
z = z.reshape(x_new.shape)
plotter.contourf(x_new, y_new, z, cmap=clrs, norm=norms, alpha=0.35)

plotter.scatter(class1[:, 0], class1[:, 1], color='red', label='Class 1')
plotter.scatter(class2[:, 0], class2[:, 1], color='blue', label='Class 2')
plotter.legend()
plotter.show()
```



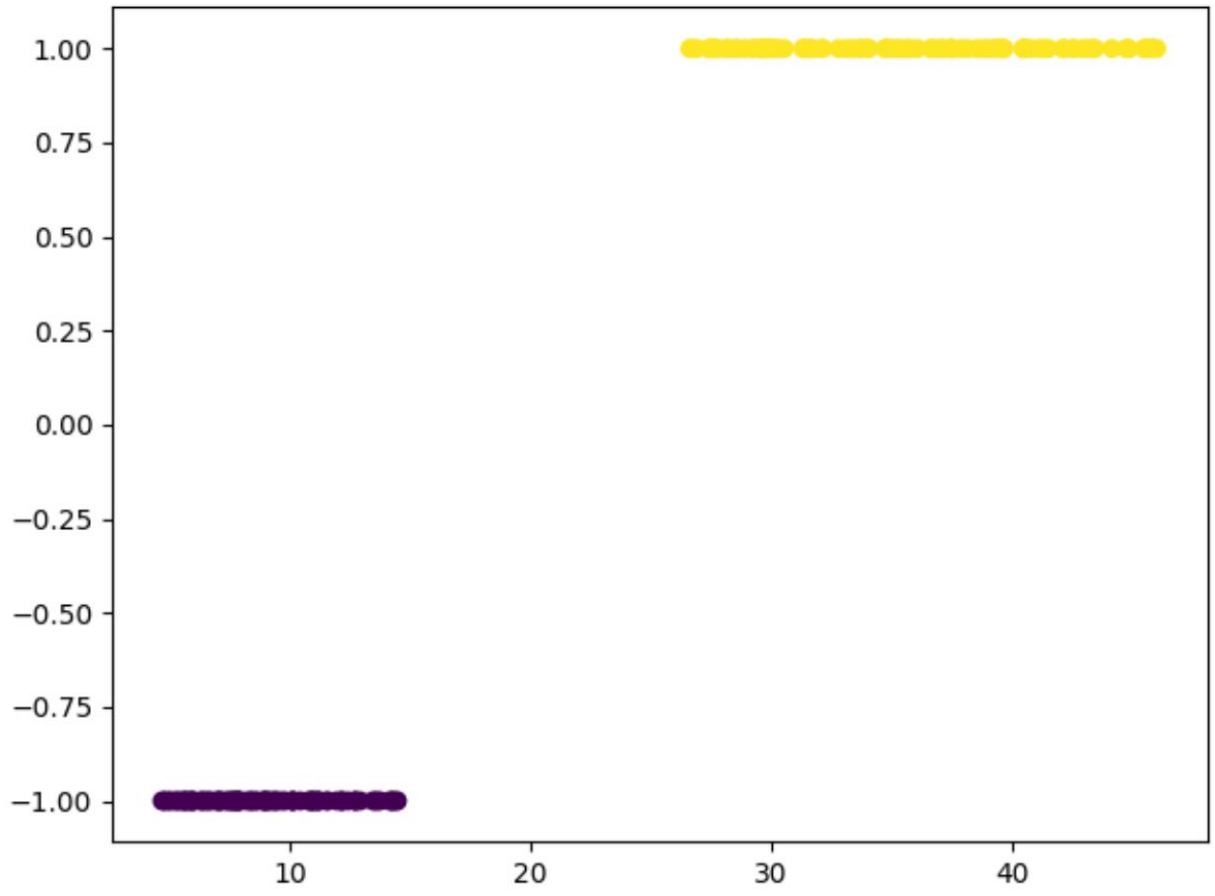
Part 2:
a)

```
# A
train, test = train_test_split(dataset, test_size=0.3)
train, validation = train_test_split(train, test_size=0.3)
plotter.figure()
a = train.iloc[:, 0]
b = train.iloc[:, 1]
c = train.iloc[:, 2]
plotter.scatter(a, b, c=c)
plotter.colorbar()
plotter.show()
```



b)

```
# B
def map_circle(dataset):
    new = np.square(dataset[:, :-1])
    mapp = np.array([1, 1])
    return np.column_stack((np.matmul(new, mapp), dataset[:, -1]))


train_mapped = map_circle(train.values)
test_mapped = map_circle(test.values)

a = train_mapped[:, 0]
b = train_mapped[:, 1]
c = train_mapped[:, 1]
plotter.scatter(a, b, c=c)
plotter.show()
```



c)

```
# C
classifier = svm.SVC().fit(train.iloc[:, 0:1], train.iloc[:, 2])
prd = classifier.predict(test.iloc[:, 0:1])
print("Classification accuracy before mapping: ", accuracy_score(prd, test.iloc[:, 2]))
```

```
Classification accuracy before mapping:  0.8416666666666667
```

d)

```
# D
clf = svm.SVC()
clf.fit(train_mapped[:, :-1], train_mapped[:, -1])
prd = clf.predict(test_mapped[:, :-1])
print("Classification accuracy after mapping: ", accuracy_score(prd, test_mapped[:, -1]))
```

```
Classification accuracy after mapping:  1.0
```