Question 1a

- Answer

insurance	0	1	2
Frequency Count	143691	426067	95491
Class Probability	0.21599582	0.64046244	0.14354174

- Code

```
subData = subData.astype('category')
xTrain = pd.get_dummies(subData[['group_size', 'homeowner', 'married_couple']])

yTrain = np.where(subData['insurance'] == '0', 1, 0)

# Correctly Use sklearn.naive_bayes.CategoricalNB
feature = ['group_size', 'homeowner', 'married_couple']

labelEnc = preprocessing.LabelEncoder()
yTrain = labelEnc.fit_transform(subData['insurance'])
yLabel = labelEnc.inverse_transform([0, 1])

uGroup_Size = np.unique(subData['group_size'])
uHomeowner = np.unique(subData['homeowner'])
uMarried_Couple = np.unique(subData['married_couple'])

featureCategory = [uGroup_Size, uHomeowner, uMarried_Couple]
print(featureCategory)

featureEnc = preprocessing.OrdinalEncoder(categories = featureCategory)
xTrain = featureEnc.fit_transform(subData[['group_size', 'homeowner', 'married_couple']])

_objNB = naive_bayes.CategoricalNB(alpha = 1.0e-10)
thisModel = _objNB.fit(xTrain, yTrain)

print('Number of samples encountered for each class during fitting')
print(yLabel)
print(_objNB.class_count_)
print('\n')

print('Probability of each class:')
print(yLabel)
print(np.exp(_objNB.class_log_prior_))
print('\n')
```

- Explanation

Using the columns of each feature, I used the Naive Bayes function in the sklearn library to train and fit_transform the three features. Then I used the CategoricalNB function to train the data in the insurance column and then produced the count of it to get the frequency count of when insurance = ’0’, insurance = ’1’, and insurance = ’2’.

Question 1b

- Answer

group_size	insurance		
	0	1	2
1	115460	329552	74293
2	25728	91065	19600
3	2282	5069	1505
4	221	381	93

- Code

```
df = pd.read_csv('/Users/tiffwong/Desktop/cs484/assignments/assignment 4/Purchase_Likelihood.csv')

# Define a function to visualize the percent of a particular target category by a nominal predictor
def RowWithColumn (
   rowVar,          # Row variable
   columnVar,       # Column predictor
   show = 'ROW'):   # Show ROW fraction, COLUMN fraction, or BOTH table

   countTable = pd.crosstab(index = rowVar, columns = columnVar, margins = False, dropna = True)
   print("Frequency Table: \n", countTable)
   print( )

   if (show == 'ROW' or show == 'BOTH'):
       rowFraction = countTable.div(countTable.sum(1), axis='index')
       print("Row Fraction Table: \n", rowFraction)
       print( )

   if (show == 'COLUMN' or show == 'BOTH'):
       columnFraction = countTable.div(countTable.sum(0), axis='columns')
       print("Column Fraction Table: \n", columnFraction)
       print( )

   return
```

```
RowWithColumn(rowVar = subData['insurance'], columnVar = subData['group_size'], show = 'ROW')
```

- Explanation

Using Professor Lam's code, I created a function called RowWithColumn, which produces the frequency table, the row fraction table, and column fraction table, of each feature alongside the insurance column. Then I put in the insurance row with the group_size column to get the frequency count of the group_size feature within each possible value of the insurance feature.

Question 1c

- Answer

homeowner	insurance		
	0	1	2
0	78659	183130	46734
1	65032	242937	48757

- Code

```
RowWithColumn(rowVar = subData['insurance'], columnVar = subData['homeowner'], show = 'ROW')
```

- Explanation

Using Professor Lam's code, I created a function called RowWithColumn, which produces the frequency table, the row fraction table, and column fraction table, of each feature alongside the insurance column. Then I put in the insurance row with the homeowner column to get the frequency count of the homeowner feature within each possible value of the insurance feature.

Question 1d

- Answer

married_couple	insurance		
	0	1	2
0	117110	333272	75310
1	26581	92795	20181

- Code

```
RowWithColumn(rowVar = subData['insurance'], columnVar = subData['married_couple'], show = 'ROW')
```

- Explanation

Using Professor Lam's code, I created a function called RowWithColumn, which produces the frequency table, the row fraction table, and column fraction table, of each feature alongside the insurance column. Then I put in the insurance row with the married_couple column to get the frequency count of the married_couple feature within each possible value of the insurance feature.

Question 1e

- Answer

Feature	Cramer's V
group_size	0.027102014055820786
homeowner	0.09708641964781961
married_couple	0.03242164583520746

- Code

```
# Define a function that performs the Chi-square test
def PearsonChiSquareTest (
    xCat,           # input categorical feature
    yCat,           # input categorical target variable
    debug = 'N'     # debugging flag (Y/N)
    ):

    obsCount = pd.crosstab(index = xCat, columns = yCat, margins = False, dropna = True)
    cTotal = obsCount.sum(axis = 1)
    rTotal = obsCount.sum(axis = 0)
    nTotal = np.sum(rTotal)
    expCount = np.outer(cTotal, (rTotal / nTotal))
    print("CROSSTAB")
    print(obsCount)
    if (debug == 'Y'):
        print('Observed Count:\n', obsCount)
        print('Column Total:\n', cTotal)
        print('Row Total:\n', rTotal)
        print('Overall Total:\n', nTotal)
        print('Expected Count:\n', expCount)
        print('\n')

    chiSqStat = ((obsCount - expCount)**2 / expCount).to_numpy().sum()
    chiSqDf = (obsCount.shape[0] - 1.0) * (obsCount.shape[1] - 1.0)
    chiSqSig = scipy.stats.chi2.sf(chiSqStat, chiSqDf)

    cramerV = chiSqStat / nTotal
    if (cTotal.size > rTotal.size):
        cramerV = cramerV / (rTotal.size - 1.0)
    else:
        cramerV = cramerV / (cTotal.size - 1.0)
    cramerV = np.sqrt(cramerV)

    return(chiSqStat, chiSqDf, chiSqSig, cramerV)
```

```
catPred = ['group_size', 'homeowner', 'married_couple']

for pred in catPred:
    chisqstat, chisqdf, chisqsig, cramerV = PearsonChiSquareTest(df[pred], df['insurance'])
    print('Cramer for', pred, cramerV)
```

- Explanation

First I created a function to calculate the Pearson Chi-square statistic of each feature against the insurance column. Next, I used the Chi-square statistic to plug into the Cramer's V statistic formula for all three of the features: group_size, homeowner, and married_couple, in a for-loop.

Question 1f

- Answer

group_ size	homeowner	married_ couple	Prob(insurance = 0)	Prob(insurance = 1)	Prob(insurance = 2)
1	0	1	0.2143906649937	0.6374669358561	0.1481423991503
1	0	0	0.2270372559700	0.6275928478960	0.1453698961339
1	1	1	0.1938423323674	0.6634140238345	0.1427436437981
1	1	0	0.2055878590594	0.6541277133010	0.1402844276396
2	0	1	0.2253417750143	0.6246345878773	0.1500236371084
2	0	0	0.2384413257711	0.6144618299641	0.1470968442648
2	1	1	0.2040786192881	0.6511275239635	0.1447938567484
2	1	0	0.2162806900233	0.6415276173738	0.1421916926029
3	0	1	0.2366531041579	0.6115458857331	0.1518010101089
3	0	0	0.2502010270388	0.6010837615951	0.1487152113662
3	1	1	0.2146842364088	0.6385592917494	0.1467564718418
3	1	0	0.2273421576714	0.6286517281095	0.1440061142192
4	0	1	0.2483178955109	0.5982151227830	0.1534669817061
4	0	0	0.2623075011556	0.5874746662055	0.1502178326389
4	1	1	0.2256558917433	0.6257203429795	0.1486237652772
4	1	0	0.2387670440694	0.6155127373556	0.1457202185750

- Code

```
# Create the all possible combinations of the features' values
xTest = []
num = len(catGroup_Size)*len(catHomeowner)*len(catMarried_Couple)
for j in catGroup_Size:
    for k in catHomeowner:
        for p in catMarried_Couple:
            xTest.append([j,k,p])

xTest = pd.DataFrame(xTest, columns = ['group_size', 'homeowner', 'married_couple'])
xTest = xTest[['group_size', 'homeowner', 'married_couple']].astype('category')

x = df[['group_size', 'homeowner', 'married_couple']].astype('category')
y = df['insurance'].astype('category')
classifier = naive_bayes.MultinomialNB(alpha = 1.0e-10).fit(x, y)

# Score the xTest and append the predicted probabilities to the xTest
yTest = classifier.predict_proba(xTest)
yTest = pd.DataFrame(yTest, columns = ['P(insurance=0)',
                                       'P(insurance=1)',
                                       'P(insurance=2)'])

yTest_score = pd.concat([xTest, yTest], axis = 1)
print(yTest_score)
```

- Explanation

I created an array of all the possible combinations of values from the 3 features: group_size, homeowner, and married_couple. Then, I used the MultinomialNB function in the naive_bayes library to create a classifier for predicting the probabilities of when insurance = 0,1,2. Then I joined the two dataframes and printed it out.

Question 1g

- Answer

The maximum value for this odds Prob(insurance = 1) / Prob(insurance = 2) is the combo (group_size=1, homeowner=1, married_couple=0), where Prob(insurance = 1) / Prob(insurance = 2) = 4.662867606243949.

- Code

```
yTest_score['1g'] =  yTest_score['P(insurance=1)']/yTest_score['P(insurance=2)']
```

- Output

```
             1g
4.1635744867727
4.1772604506592
4.4969278295765
4.5117095494835
4.3030688007784
4.3172132923434
4.6475906469976
4.6628676062439
4.0286022161135
4.0418445165978
4.3511491093742
4.3654516443149
3.8980053959007
3.9108184153980
4.2100961566445
4.2239350405510
```

- Explanation

I divided the columns from the dataframe previously and got the appropriate numbers. Then I used the max() function to find the maximum of that column in the dataframe.

Question 2a

- Answer

In general form, I get this:

$0 = 0.0033449990212 + 0.05333511993\,x + 0.3286838316397y$

After solving in sloe-intercept form, I get this: $y = -0.1622688x - 0.0101769$.

- Code

```
# Build Support Vector Machine classifier
xTrain = trainData[['x','y']]
yTrain = trainData['SpectralCluster']

svm_Model = svm.SVC(kernel = 'linear', decision_function_shape = 'ovr',
                    random_state = 20210325, max_iter = -1)
thisFit = svm_Model.fit(xTrain, yTrain)
y_predictClass = thisFit.predict(xTrain)

print('Mean Accuracy = ', metrics.accuracy_score(yTrain, y_predictClass))
trainData['_PredictedClass_'] = y_predictClass

svm_Mean = trainData.groupby('_PredictedClass_').mean()
print(svm_Mean)

print('Intercept = ', thisFit.intercept_)
print('Coefficients = ', thisFit.coef_)
```

- Command Line Output

```
Mean Accuracy =  0.5
                                x              y  SpectralCluster
_PredictedClass_
0                -0.11036244012 -1.67248147882              0.5
1                 0.07660423956  1.65477923548              0.5
Intercept =  [0.0033449990212]
Coefficients =  [[0.05333511993   0.3286838316397]]
```

- Explanation

I loaded the dataset in and then split up the x- and y- values form the SpectralCluster values. Next, I used the SVC function in the Suppoer Vector Machine library to fit the data into a linear kernel along with all the specifications given in the problem.

Question 2b

- Answer

The misclassification rate is 0.5.

- Code

```
misclass  = 1 - metrics.accuracy_score(yTrain, y_predictClass)
print('Misclassification:', misclass)
```

- Command Line Output

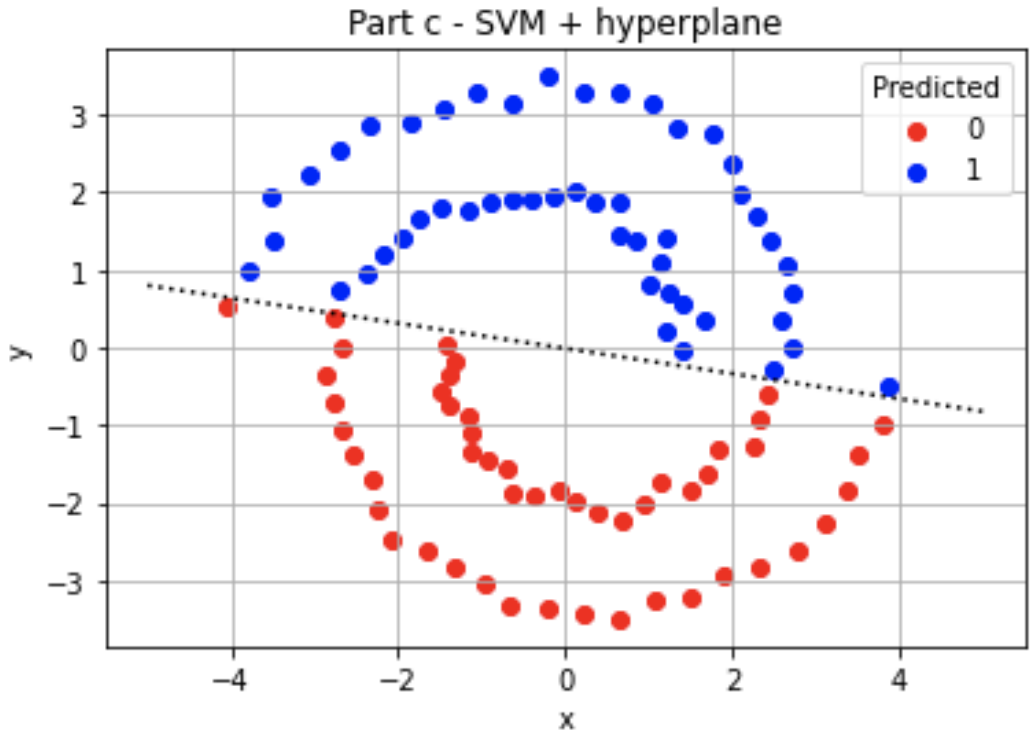
```
---QUESTION 2B---
Misclassification: 0.5
```

- Explanation

I used the metrics function in the sklearn library to calculate the misclassification rate given the two datasets that I have split up from the original dataframe.

Question 2c

- Answer/Command Line Output



- Code

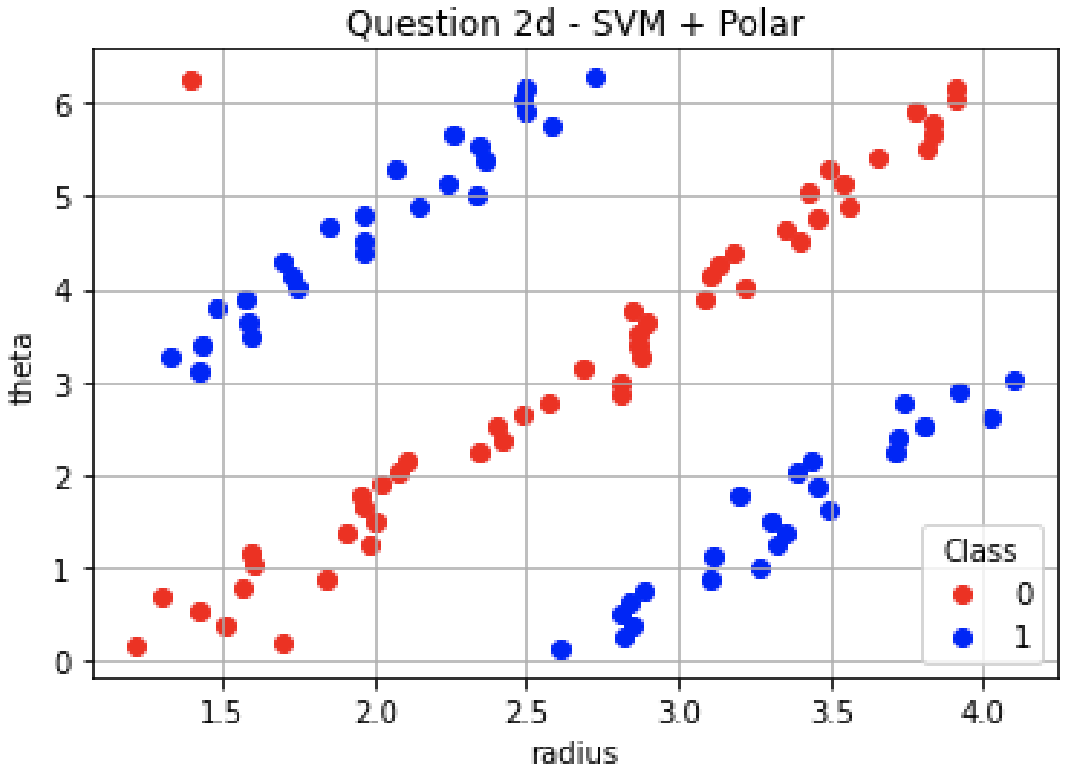
```
# plot the line, the points, and the nearest vectors to the plane
inter = thisFit.intercept_
co1 = thisFit.coef_[0][0]
co2 = thisFit.coef_[0][1]
x = np.linspace(-5, 5)
y = (-inter - co1*np.linspace(-5, 5))/co2
reds = trainData[trainData['_PredictedClass_'] == 0]
plt.scatter(reds['x'], reds['y'], c="red", label = 0)
blues = trainData[trainData['_PredictedClass_'] == 1]
plt.scatter(blues['x'], blues['y'], c = "blue", label = 1)
plt.plot(x, y,
         color = 'black', linestyle = 'dotted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Question 2c - SVM + hyperplane')
plt.grid(True)
plt.legend(title = "_PredictedClass_")
plt.show()
```

- Explanation

I separated the data and followed the instructions in the question to accordingly deal with the 0's and 1's (with what colors they should be too). I added grid lines to the graph, properly labeled the axes, the legend, and the chart title.

Question 2d

- Answer/Command Line Output



- Code

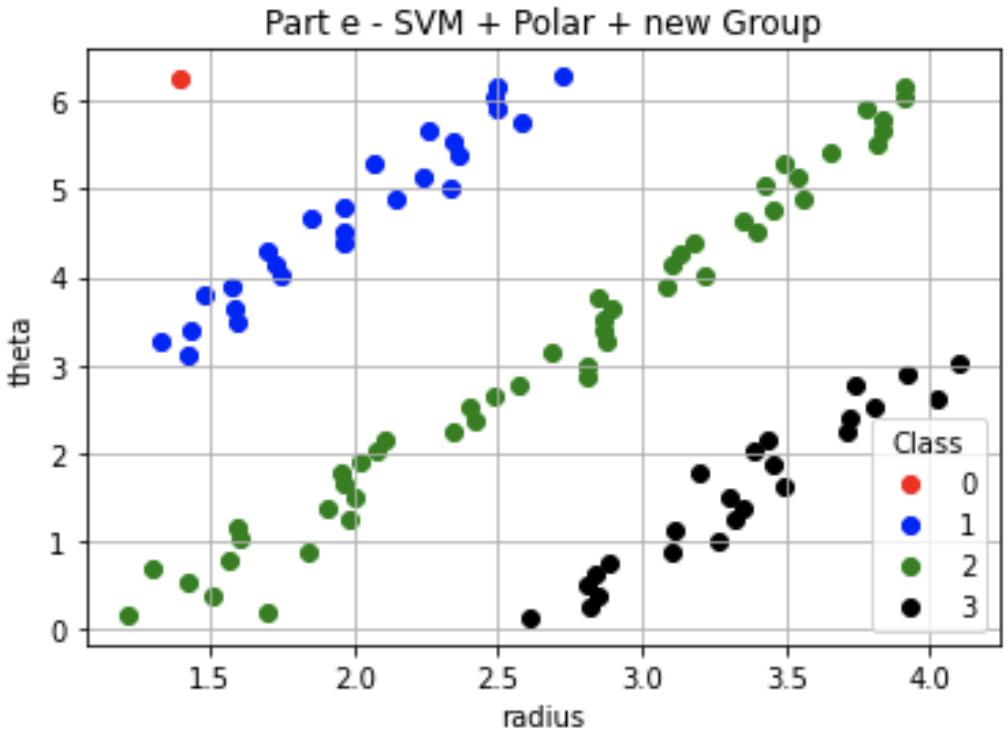
```
# Convert to the polar coordinates -- from profs code
trainData['radius'] = np.sqrt(trainData['x']**2 + trainData['y']**2)
trainData['theta'] = np.arctan2(trainData['y'], trainData['x'])
def customArcTan (z):
    theta = np.where(z < 0.0, 2.0*np.pi+z, z)
    return (theta)
trainData['theta'] = trainData['theta'].apply(customArcTan)
reds = trainData[trainData['SpectralCluster'] == 0]
plt.scatter(reds['radius'], reds['theta'], c="red", label = 0)
blues = trainData[trainData['SpectralCluster'] == 1]
plt.scatter(blues['radius'], blues['theta'], c = "blue", label = 1)
plt.xlabel('radius')
plt.ylabel('theta')
plt.title('Question 2d - SVM + Polar')
plt.grid(True)
plt.legend(title = "Class")
plt.show()
```

- Explanation

I expressed the data in polar coordinates, and plotted the theta-coordinate against the radius-coordinate in a scatterplot. Next, I color-coded the points using the SpectralCluster variable (0 = Red and 1 = Blue). Then, I properly labeled the axes, the legend, the chart title, and added grid lines to the axes.

Question 2e

- Answer/Command Line Output



- Code

```
trainData['group'] = 3
trainData.loc[trainData['radius'] <= 2.5, 'group'] = 2
trainData.loc[(trainData['radius'] < 3) & (trainData['theta'] >= 2), 'group'] = 2
trainData.loc[(trainData['radius'] < 3.5) & (trainData['theta'] >= 3), 'group'] = 2
trainData.loc[(trainData['radius'] < 4) & (trainData['theta'] >= 4), 'group'] = 2
trainData.loc[(trainData['radius'] < 2) & (trainData['theta'] >= 3), 'group'] = 1
trainData.loc[(trainData['radius'] < 2.5) & (trainData['theta'] >= 4), 'group'] = 1
trainData.loc[(trainData['radius'] < 3) & (trainData['theta'] >= 5), 'group'] = 1
trainData.loc[(trainData['radius'] < 1.5) & (trainData['theta'] >= 6), 'group'] = 0
reds = trainData[trainData['group'] == 0]
plt.scatter(reds['radius'], reds['theta'], c="red", label = 0)
blues = trainData[trainData['group'] == 1]
plt.scatter(blues['radius'], blues['theta'], c = "blue", label = 1)
green = trainData[trainData['group'] == 2]
plt.scatter(green['radius'], green['theta'], c = "green", label = 2)
black = trainData[trainData['group'] == 3]
plt.scatter(black['radius'], black['theta'], c = "black", label = 3)
plt.xlabel('radius')
plt.ylabel('theta')
plt.title('Part e - SVM + Polar + new Group')
plt.grid(True)
plt.legend(title = "Class")
plt.show()
```

- Explanation

I created another variable named Group and used it as the new target variable. I plotted the theta-coordinate against the radius-coordinate in a scatterplot. Next, I color-coded the points using the new Group target variable (0 = Red, 1 = Blue, 2 = Green, 3 = Black). Then, I properly labeled the axes, the legend, the chart title, and added grid lines to the axes.

Question 2f

- Answer

- SVM 0: Group 0 versus Group 1

$$1.4691250777389 + 0.9337841470852x - 0.4538024872059y = 0$$

- SVM 1: Group 1 versus Group 2

$$-0.8768942577876 + 1.8920953263167x - 0.8961324867551y = 0$$

- SVM 2: Group 2 versus Group 3

$$-4.1328448780756 + 2.0125835470868x - 0.8375616435865y = 0$$

- Code

```
eqs = []
for gr in [(0,1), (1,2), (2,3)]:
    temp_df = trainData[trainData['group'].isin(gr)]
    tx_train = temp_df[['radius', 'theta']]
    ty_train = temp_df["group"]
    svmmodel = svm.SVC(kernel = "linear",
                       decision_function_shape = "ovr",
                       random_state = 20200408)
    thisfit = svmmodel.fit(tx_train, ty_train)
    y_pred = thisfit.predict(tx_train)

    print(f"SVM {gr}")
    print("Intercept:", thisfit.intercept_)
    print("Coefficients:", thisfit.coef_)
    inter = thisfit.intercept_
    co1 = thisfit.coef_[0][0]
    co2 = thisfit.coef_[0][1]
    x = np.linspace(0, 4.5)
    y = (-inter - co1*x)/co2
    eqs.append((x, y))
    print('\n')
```

- Command Line Output

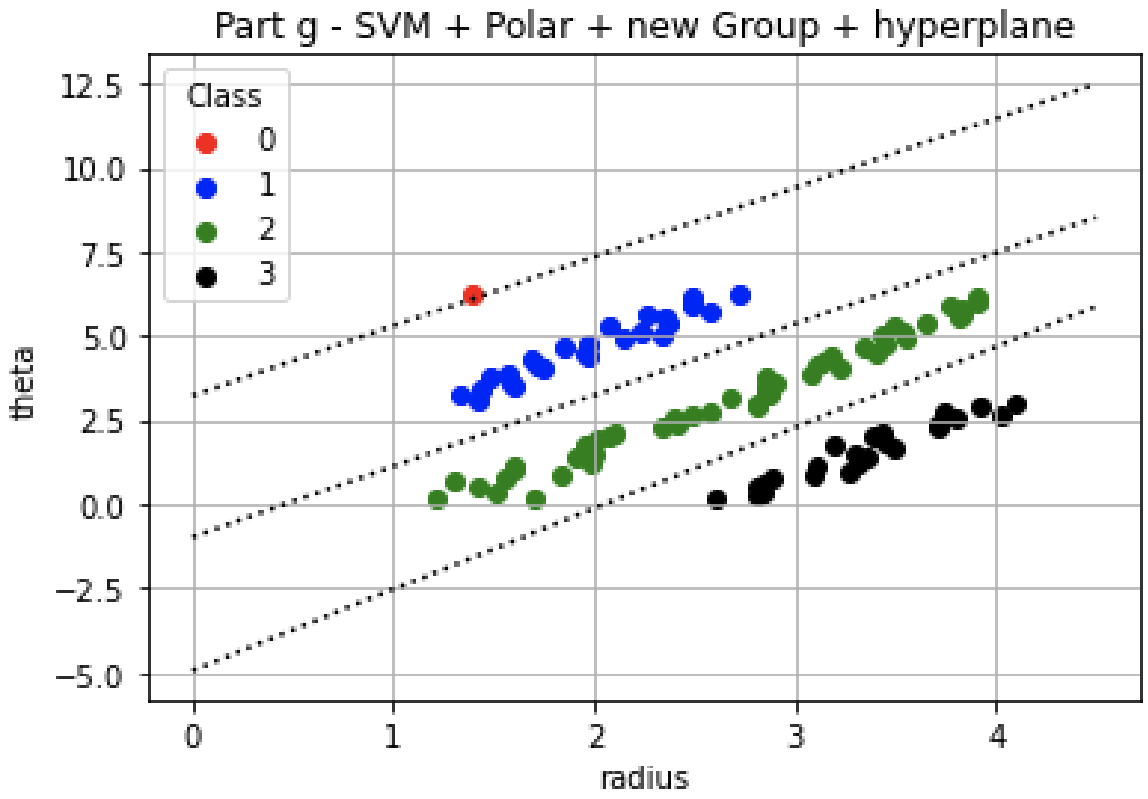
```
SVM (0, 1)
Intercept: [1.4691250777389]
Coefficients: [[ 0.9337841470852 -0.4538024872059]]


SVM (1, 2)
Intercept: [-0.8768942577876]
Coefficients: [[ 1.8920953263167 -0.8961324867551]]


SVM (2, 3)
Intercept: [-4.1328448780756]
Coefficients: [[ 2.0125835470868 -0.8375616435865]]
```

- Explanation

I ran the SVC function in the Support Vector Machine library for the three different combinations of the group features. I repeated the code from Question 1a for the different polar coordinates.

Question 2g

- Answer/Command Line



- Code

```
reds = trainData[trainData['group'] == 0]
plt.scatter(reds['radius'], reds['theta'], c="red", label = 0)
blues = trainData[trainData['group'] == 1]
plt.scatter(blues['radius'], blues['theta'], c = "blue", label = 1)
green = trainData[trainData['group'] == 2]
plt.scatter(green['radius'], green['theta'], c = "green", label = 2)
black = trainData[trainData['group'] == 3]
plt.scatter(black['radius'], black['theta'], c = "black", label = 3)
for eq in eqs:
    plt.plot(eq[0], eq[1],
         color = 'black', linestyle = 'dotted')
plt.xlabel('radius')
plt.ylabel('theta')
plt.title('Part g - SVM + Polar + new Group + hyperplane')
plt.grid(True)
plt.legend(title = "Class")
plt.show()
```

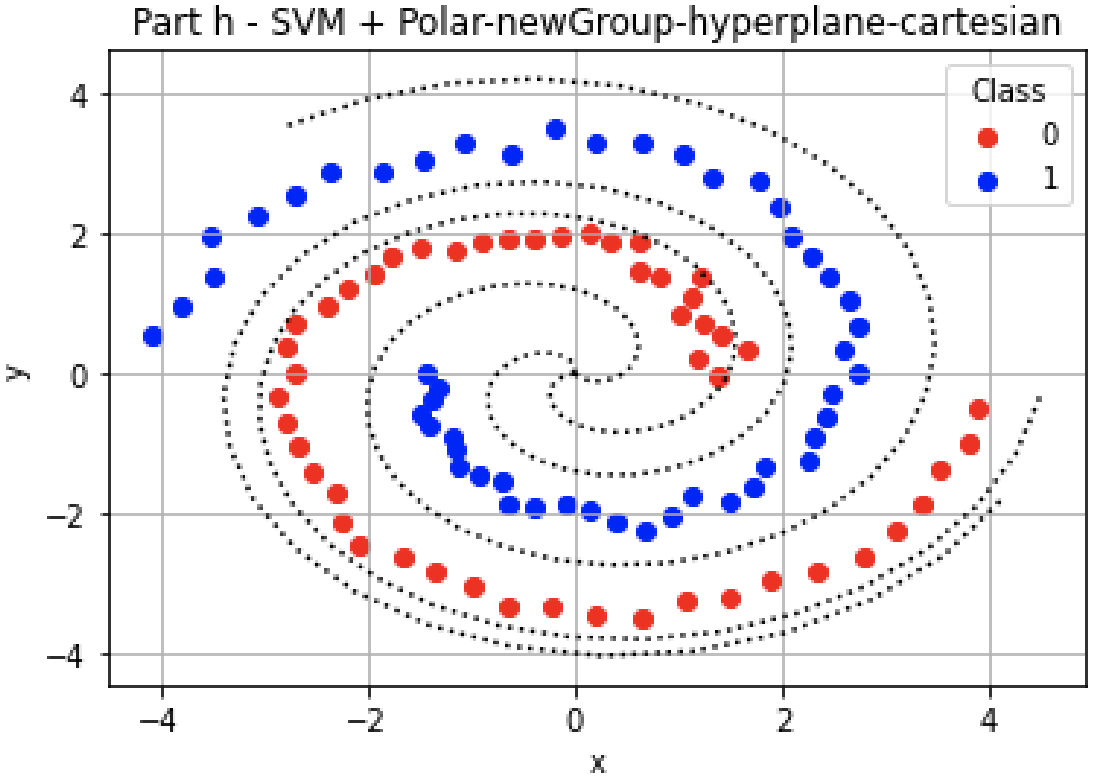
- Explanation

I plotted the theta-coordinate against the radius-coordinate in a scatterplot. Next, I color-coded the points using the new Group target variable (0 = Red, 1 = Blue, 2 = Green, 3 = Black). Then, I added the hyperplanes to the graph. Finally, I properly labeled the axes, the legend, the chart title, and added grid lines to the axes.

Question 2h

- Answer/Command Line Output

Based on my graph, the hypercurve that is not needed would be the



- Code

```
ceqs = []
for eq in eqs:
    ceqs.append((eq[0]*np.cos(eq[1]), eq[0]*np.sin(eq[1])))
reds = trainData[trainData['SpectralCluster'] == 0]
plt.scatter(reds['x'], reds['y'], c="red", label = 0)
blues = trainData[trainData['SpectralCluster'] == 1]
plt.scatter(blues['x'], blues['y'], c = "blue", label = 1)
for eq in ceqs:
    plt.plot(eq[0], eq[1],
         color = 'black', linestyle = 'dotted')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Part h - SVM + Polar-newGroup-hyperplane-cartesian')
plt.grid(True)
plt.legend(title = "Class")
plt.show()
```

- Explanation

I converted the observations along with the hyperplanes from the polar coordinates back to the Cartesian coordinates. Next, I plotted the y-coordinate against the x-coordinate in a scatterplot.

Then, I color-coded the points using the SpectralCluster (0 = Red and 1 = Blue), and plotted the hyper-curves as dotted lines to the graph. Finally, I properly labeled the axes, the legend, and the chart title. Also, grid lines should be added to the axes.