IS 670 Machine Learning for Business Analytics

Group Project 2

Team Members: Priyanka Malla, Jwalant Maheta, Maharshi Patel, Alaba Adewunmi

Data Representation and preprocessing:
Given are the two datasets

- Reading the dataset file: StudentClass.csv

```
stclass = pd.read_csv('/content/drive/MyDrive/DL_data/StudentClass.csv')
stclass.head(10)
```

	StudentID	GRADE	ClassWeekDay	ClassMeetTime	ClassDate	ClassTitle
0	1.0	A	MW	15:30:00	9/1/2004	Math
1	1.0	A	M	19:00:00	9/1/2004	English
2	1.0	C	W	19:00:00	9/1/2004	Physics
3	1.0	B	MW	11:00:00	9/1/2004	Art
4	1.0	B	MW	12:30:00	9/1/2005	SocialStudy
5	1.0	B	TTH	12:30:00	9/1/2005	PE
6	1.0	B	M	19:00:00	9/1/2005	ForeignLanguage
7	1.0	B	MW	11:00:00	9/1/2005	Statistics
8	1.0	B	TU	16:00:00	9/1/2006	Music
9	1.0	C	M	16:00:00	3/1/2007	Drawing

- Reading the dataset file: StudentGraduate.csv

```
[52] stgrad = pd.read_csv('/content/drive/MyDrive/DL_data/StudentGraduate.csv')
     stgrad
```

	ID	HS_ENGLISH	HS_MATH	HS_HISTORY	HS_LAB_SCIENCES	HS_FOREIGN_LANGUAGE	HS_ART	HS_ELECTIVES	HS_GPA	sex	WRITINGScore	GraduateONTime
0	1	8	8	7	8	6	4	1	3.691959	1	428	0
1	2	8	10	7	6	8	10	0	3.712993	1	558	0
2	3	8	7	6	6	4	4	0	4.373719	1	634	1
3	4	8	8	6	6	4	2	0	4.046570	0	386	0
4	5	8	6	6	4	4	3	1	2.560076	0	381	0
...	...	...	...	...	...	...	...	...	...	...	...	...
995	996	8	8	7	6	6	3	2	3.002898	1	441	0
996	997	8	10	6	7	6	2	0	3.910183	0	443	1
997	998	8	10	6	8	6	6	0	3.267001	0	515	0
998	999	8	8	7	8	8	2	1	3.474561	0	459	0
999	1000	8	6	6	6	4	16	9	3.690760	0	552	1

1000 rows × 13 columns

- Printing columns of both the files:

```
print(stclass.columns)
print(stgrad.columns)
```

```
Index(['StudentID', 'GRADE', 'ClassWeekDay', 'ClassMeetTime', 'ClassDate',
       'ClassTitle'],
      dtype='object')
Index(['ID', 'HS_ENGLISH', 'HS_MATH', 'HS_HISTORY', 'HS_LAB_SCIENCES',
       'HS_FOREIGN_LANGUAGE', 'HS_ART', 'HS_ELECTIVES', 'HS_GPA', 'sex',
       'WRITINGScore', 'GraduateONTime', 'Major'],
      dtype='object')
```

Research question:

1. Do students perform better in Morning class (before 12:00pm) or afternoon class?

- First, we split the studentclass.csv dataset into two data frames where one has the records of students attending the morning class (class before 12:00:00PM) and other has the records of student will classes in the afternoon (class after 12:00:00PM).

```
Studentclass_new=Studentclass.dropna(subset=['ClassMeetTime'])
#here we are splitting the dataset based on the classtime into Morning class and afternoon class
Studentclass_afternoon=Studentclass[Studentclass['ClassMeetTime']>'12:00:00']
Studentclass_morning=Studentclass[Studentclass['ClassMeetTime']<'12:00:00']
Studentclass_morning['GRADE'].count()
```

```
1612
```

- Calculating the average of grades of students with classes in the morning which is **2.8128**

```
[ ] count=0
    for i, row in Studentclass_morning.iterrows():
        if row['GRADE']=='A':
            count=count+4
        elif row['GRADE']=='B':
            count=count+3
        elif row['GRADE']=='C':
            count=count+2
        elif row['GRADE']=='D':
            count=count+1
    print(count/Studentclass_morning['GRADE'].count())
```

```
2.8138957816377173
```

- Calculating the average of grades of students with classes in the afternoon which is **2.7753**

```
[ ]  count=0
     for i, row in Studentclass_afternoon.iterrows():
         if row['GRADE']=='A':
             count=count+4
         elif row['GRADE']=='B':
             count=count+3
         elif row['GRADE']=='C':
             count=count+2
         elif row['GRADE']=='D':
             count=count+1
     print(count/Studentclass_afternoon['GRADE'].count())
```

```
2.775314559616537
```

- Calculating the percentage of the students grade average during morning and afternoon classes

Hence the students attending the morning classes has better performance than students in the afternoon class (70.34>69.38)

```
percent_morning=(2.8138957816377173/4)*100
print('percent of average gpa of student attending morning class ',end="")
print(percent_morning)
percent_afternoon=(2.775314559616537/4)*100
print('percent of average gpa of student attending afternoon class ',end="")
print(percent_afternoon)
```

```
percent of average gpa of student attending morning class 70.34739454094293
percent of average gpa of student attending afternoon class 69.38286399041343
```

2. Math, Physics and Statistics are considered quantitative classes. If a student take 2 or more quantitative classes in one semester, will he/she have lower average score?

- First, we have assigned the values 4 to 0 for the grades A to F.
- Then we created a data frame with students having 2 or more quantitative classes (math, statistics, physics)
- Then we calculate the average GPA of quantitative classes in that academic year and total graded average of each student in that year to compare the quantitative grades to overall average.

```
import csv
d={}
grade = {}
points = {'A':4, 'B':3, 'C':2, 'D':1, 'F':0}
year_grades = {}
with open('/content/drive/MyDrive/DL_data/StudentClass.csv', mode='r') as f:
    csvreader = csv.reader(f)
    next(csvreader)
    for line in csvreader:
        if line[0]=='':
            break
        if int(line[4].split('/')[2]) in year_grades:
            year_grades[int(line[4].split('/')[2])].append(points[line[1]])
        else:
            year_grades[int(line[4].split('/')[2])] = [points[line[1]]]
        if int(line[0]) in d:
            if int(line[4].split('/')[2]) in d[int(line[0])]:
                d[int(line[0])][int(line[4].split('/')[2])].append(line[5])
                grade[int(line[0])][int(line[4].split('/')[2])].append(line[1])
            else:
                d[int(line[0])][int(line[4].split('/')[2])] = [line[5]]
                grade[int(line[0])][int(line[4].split('/')[2])] = [line[1]]
        else:
            d[int(line[0])]={int(line[4].split('/')[2]):[line[5]]}
            grade[int(line[0])]={int(line[4].split('/')[2]):[line[1]]}
    for year in year_grades:
        year_grades[year] = sum(year_grades[year])/len(year_grades[year])
    stud_qclass_grade = {}
    for student in d:
        for year in d[student]:
            if ('Math' in d[student][year] and 'Physics' in d[student][year]) or ('Math' in d[student][year] and 'Statistics' in d[student][yea
                s = 0
                for g in grade[student][year]:
                    s = s + points[g]
                stud_qclass_grade[student] = {str(year)+' qclass': s/len(grade[student][year]), str(year): year_grades[year]}
    for std in stud_qclass_grade:
      print(str(std)+"--->"+str(stud_qclass_grade[std]))
```

- Displaying the average of quantitative classes and total average of that student in that year.

- For example for student with student ID 1 in 2004, quantitative GPA is 3.25 and total GPA is 2.7

```
1--->{'2004 qclass': 3.25, '2004': 2.7080557238037555}
2--->{'2006 qclass': 3.4, '2006': 2.78990990990991}
3--->{'2005 qclass': 3.5, '2005': 2.8646926536731634}
4--->{'2006 qclass': 3.4, '2006': 2.78990990990991}
6--->{'2006 qclass': 2.6666666666666665, '2006': 2.78990990990991}
7--->{'2005 qclass': 3.142857142857143, '2005': 2.8646926536731634}
9--->{'2004 qclass': 2.6666666666666665, '2004': 2.7080557238037555}
10--->{'2010 qclass': 3.4285714285714284, '2010': 2.5821596244131455}
11--->{'2006 qclass': 2.3333333333333335, '2006': 2.78990990990991}
12--->{'2004 qclass': 1.3333333333333333, '2004': 2.7080557238037555}
13--->{'2004 qclass': 3.5, '2004': 2.7080557238037555}
14--->{'2005 qclass': 2.2222222222222223, '2005': 2.8646926536731634}
15--->{'2004 qclass': 3.0, '2004': 2.7080557238037555}
16--->{'2010 qclass': 3.142857142857143, '2010': 2.5821596244131455}
17--->{'2011 qclass': 2.25, '2011': 2.4146341463414633}
19--->{'2004 qclass': 3.0, '2004': 2.7080557238037555}
```

- Displaying the number of students having higher and lower quantitative average.

As we can see there are more students have higher quantitative GPA (427>424).

```
[63] gre = []
     less = []
     for std in stud_qclass_grade:
       if stud_qclass_grade[std][list(stud_qclass_grade[std].keys())[0]] >= stud_qclass_grade[std][list(stud_qclass_grade[std].keys())[1]]:
         gre.append(stud_qclass_grade[std][list(stud_qclass_grade[std].keys())[0]])
       else:
         less.append(stud_qclass_grade[std][list(stud_qclass_grade[std].keys())[0]])
     print('Number of student having higher quantitative GPA ',end="")
     print(len(gre))
     print('Number of student having lower quantitative GPA ',end="")

     print(len(less))

     Number of student having higher quantitative GPA 427
     Number of student having lower quantitative GPA 424
```

Number of students having higher quantitative GPA is 427

Number of students having lower quantitative GPA is 424

3. Which Major has higher quantitative class GPA?

- The below code sample shows the average of grades of all the student scores in the respective quantitative classes in every major.

Code sample:

```
  for g in math:
    math_sum = math_sum + points[g]
  math_avg = math_sum/len(math)
  for g in phy:
    phy_sum = phy_sum + points[g]
  phy_avg = phy_sum/len(phy)
  for g in stat:
    stat_sum = stat_sum + points[g]
  stat_avg = stat_sum/len(stat)
  maj_qclass[major] = {'Math':math_avg, 'Physics':phy_avg, 'Statistics':stat_avg}
for maj in maj_qclass:
  print(str(maj)+"---->"+str(maj_qclass[maj]))
```

```
[32] maj = {}
     stud_sub = {}
     maj_qclass = {}
     with open('/content/drive/MyDrive/DL_data/StudentClass.csv', mode='r') as f2:
       csvreader = csv.reader(f2)
       next(csvreader)
       for line in csvreader:
         if line[0]=='':
           break
         if int(line[0]) in stud_sub:
           stud_sub[int(line[0])][line[5]] = line[1]
         else:
           stud_sub[int(line[0])] = {line[5]:line[1]}
     with open('/content/drive/MyDrive/DL_data/StudentGraduate.csv', mode='r') as f1:
       csvreader = csv.reader(f1)
       next(csvreader)
       for line in csvreader:
         if line[0]=='':
           break
         if line[12] in maj:
           maj[line[12]].append(int(line[0]))
         else:
           maj[line[12]] = [int(line[0])]
       for major in maj:
         math = []
         phy = []
         stat = []
         for id in maj[major]:
           if 'Math' in stud_sub[id]:
             math.append(stud_sub[id]['Math'])
           if 'Physics' in stud_sub[id]:
             phy.append(stud_sub[id]['Physics'])
           if 'Statistics' in stud_sub[id]:
             stat.append(stud_sub[id]['Statistics'])
         math_sum = 0
         phy_sum = 0
         stat_sum = 0
         for g in math:
```

✓ 0s completed at 1:32 PM

- Displaying the output of the math, statistics, physics average grades in each major.

```
    math_sum = 0
    phy_sum = 0
    stat_sum = 0
    for g in math:
      math_sum = math_sum + points[g]
    math_avg = math_sum/len(math)
    for g in phy:
      phy_sum = phy_sum + points[g]
    phy_avg = phy_sum/len(phy)
    for g in stat:
      stat_sum = stat_sum + points[g]
    stat_avg = stat_sum/len(stat)
    maj_qclass[major] = {'Math':math_avg, 'Physics':phy_avg, 'Statistics':stat_avg}
  for maj in maj_qclass:
    print(str(maj)+"---->"+str(maj_qclass[maj]))
```

```
FIN---->{'Math': 2.8487394957983194, 'Physics': 2.808333333333333, 'Statistics': 2.715481171548117}
MGMT---->{'Math': 2.7655677655677655, 'Physics': 2.854014598540146, 'Statistics': 2.767272727272727}
MKTG---->{'Math': 2.8227848101265822, 'Physics': 2.8649789029535864, 'Statistics': 2.7983193277310923}
ACCT---->{'Math': 2.772972972972973, 'Physics': 2.7621621621621624, 'Statistics': 2.777173913043478}
I S---->{'Math': 2.6595744680851063, 'Physics': 2.8627450980392157, 'Statistics': 2.84}
```

- Calculating the average of all three quantitative classes to compare.

Hence displaying the average of the quantitative score, says that Marketing has the highest average quantitative grade average than the other majors (MKTG = 8.48).

```
[45] sum_maj = {}
     for maj in maj_qclass:
       #print(str(maj)+"---->"+str(sum(list(maj_qclass[maj].values()))))
       sum_maj[maj]=sum(list(maj_qclass[maj].values()))
     #print(sum_maj)
     keys = list(sum_maj.keys())
     values = list(sum_maj.values())
     sorted_value_index = np.argsort(values)
     sorted_dict = {keys[i]: values[i] for i in sorted_value_index}
     for maj in sorted_dict:
       print(str(maj)+"---->"+str(sorted_dict[maj]))
     #print(sorted_dict)
```

```
ACCT---->8.312309048178614
I S---->8.362319566124322
FIN---->8.37255400067977
MGMT---->8.386855091380639
MKTG---->8.48608304081126
```

4. Can you combine two dataset and build a classification model to predict if a student can Graduate ON Time (Note: We cannot use the last 2 semester's performance to predict the graduation, since it is too late to predict the graduation)

- Reading the dataset files
- Dropping the Null values
- Converting the StudentID into integer Datatype.

```
[42] fp = pd.read_csv('/content/drive/MyDrive/DL_data/StudentClass.csv')
     fp_df = pd.read_csv('/content/drive/MyDrive/DL_data/StudentGraduate.csv')
     ##dropping the Null values
     fp=fp.dropna()
```

```
## Converting the studentid into integer datatype.
fp['StudentID'] = fp['StudentID'].astype(int)
```

```
[9] fp
```

	StudentID	GRADE	ClassWeekDay	ClassMeetTime	ClassDate	ClassTitle
0	1.0	A	MW	15:30:00	9/1/2004	Math
1	1.0	A	M	19:00:00	9/1/2004	English
2	1.0	C	W	19:00:00	9/1/2004	Physics
3	1.0	B	MW	11:00:00	9/1/2004	Art
4	1.0	B	MW	12:30:00	9/1/2005	SocialStudy
...	...	...	...	...	...	...

0s completed at 2:18 PM

- Merging the two datasets into a single data frame to build a classification model.

```
[ ] # Merging two dataset 'StudentGraduate' and 'StudentClass' to built a classification model.
```

```
[12] fp = pd.merge(fp, fp_df, left_on='StudentID', right_on='ID', how='left')
     fp
```

	StudentID	GRADE	ClassWeekDay	ClassMeetTime	ClassDate	ClassTitle	ID	HS_ENGLISH	HS_MATH	HS_HISTORY	HS_LAB_SCIENCES	HS_FOREIGN
0	1	A	MW	15:30:00	9/1/2004	Math	1	8	8	7	8	
1	1	A	M	19:00:00	9/1/2004	English	1	8	8	7	8	
2	1	C	W	19:00:00	9/1/2004	Physics	1	8	8	7	8	
3	1	B	MW	11:00:00	9/1/2004	Art	1	8	8	7	8	
4	1	B	MW	12:30:00	9/1/2005	SocialStudy	1	8	8	7	8	
...	...	...	...	...	...	...	...	...	...	...	...	...
9992	1000	C	MW	15:30:00	9/1/2008	Physics	1000	8	6	6	6	
9993	1000	C	MW	14:00:00	9/1/2008	Math	1000	8	6	6	6	
9994	1000	C	TTH	12:30:00	9/1/2008	ForeignLanguage	1000	8	6	6	6	
9995	1000	B	TU	19:00:00	9/1/2009	Music	1000	8	6	6	6	
9996	1000	D	MW	11:00:00	3/1/2010	Drawing	1000	8	6	6	6	

9997 rows × 19 columns

- Making a list of all the dates and studentID's that have taken subjects in last 2 semester of their major. Because we cannot predict the graduation based on last 2 semester's performance.

```
[ ]  # Making a list of all the dates and studentid's that have taken subjects in last 2 semester of their major.
     # Because we cannot predict the graduation based on last 2 semester's performance.
```

```
[13] map={}
     from datetime import datetime
     for i in range(0,1000):
         list1=[]
         for j in range(fp.shape[0]):
             if fp.iloc[j,0]==i:
                 list1.append(fp.iloc[j,4])
         list1=list(set(list1))
         list1.sort(key=lambda date: datetime.strptime(date, "%m/%d/%Y"))
         list1=list1[-2:]
         map[i]=list1
```

```
[14] map
```

```
942: ['3/1/2007', '9/1/2007'],
943: ['9/1/2006', '3/1/2007'],
944: ['3/1/2007', '9/1/2007'],
945: ['9/1/2006', '9/1/2007'],
946: ['9/1/2006', '3/1/2007'],
947: ['9/1/2007', '3/1/2008'],
948: ['9/1/2006', '3/1/2007'],
949: ['3/1/2007', '9/1/2007'],
950: ['3/1/2008', '9/1/2009'],
951: ['9/1/2008', '9/1/2009'],
952: ['9/1/2010', '3/1/2011'],
953: ['9/1/2006', '3/1/2007'],
954: ['3/1/2008', '9/1/2008'],
955: ['3/1/2008', '9/1/2008'],
956: ['3/1/2007', '9/1/2007'],
957: ['3/1/2008', '9/1/2008'],
958: ['9/1/2006', '3/1/2007'],
```

✓ 0s completed at 2:18 PM

- Making another list of all the row numbers of dataset which needs to be deleted from data frame. Because it contains last 2 semester's performance.

```
[ ]  # # Making another list of all the row numbers of dataset which needs to be deleted from dataframe
     # because it contains last 2 semester's performance.
```

```
[15] indexList=[]
     for index,row in fp.iterrows():
         if row["StudentID"]==1000:
             break
         dates = map[row["StudentID"]]
     #     print(dates)
     #     print(str(row["ClassDate"]) in dates)
     #     print(index,row["StudentID"], row["ClassDate"])
         if str(row["ClassDate"]) in dates:
             indexList.append(index)
     indexList
```

```
[8,
 9,
 17,
 18,
 19,
 25,
 26,
 27,
 28,
 29,
 37,
 38,
 39,
 47,
 48,
 49,
 57,
```

- Dropping all the rows from the data frame. Got the final dataset on which we have data except the of last 2 semester data of individual students.

```
[16] # Dropping all the rows from the dataframe.
     # Got the final dataset on which we have data except the of last 2 semester data of individual students.
```

```
[17] fp.shape
     fp = fp.drop(indexList)
     fp
```

	StudentID	GRADE	ClassWeekDay	ClassMeetTime	ClassDate	ClassTitle	ID	HS_ENGLISH	HS_MATH	HS_HISTORY	HS_LAB_SCIENCES	HS_FOREIGN_LANGUAGE
0	1	A	MW	15:30:00	9/1/2004	Math	1	8	8	7	8	6
1	1	A	M	19:00:00	9/1/2004	English	1	8	8	7	8	6
2	1	C	W	19:00:00	9/1/2004	Physics	1	8	8	7	8	6
3	1	B	MW	11:00:00	9/1/2004	Art	1	8	8	7	8	6
4	1	B	MW	12:30:00	9/1/2005	SocialStudy	1	8	8	7	8	6
...	...	...	...	...	...	...	...	...	...	...	...	...
9992	1000	C	MW	15:30:00	9/1/2008	Physics	1000	8	6	6	6	4
9993	1000	C	MW	14:00:00	9/1/2008	Math	1000	8	6	6	6	4
9994	1000	C	TTH	12:30:00	9/1/2008	ForeignLanguage	1000	8	6	6	6	4
9995	1000	B	TU	19:00:00	9/1/2009	Music	1000	8	6	6	6	4
9996	1000	D	MW	11:00:00	3/1/2010	Drawing	1000	8	6	6	6	4

6636 rows × 19 columns

- Splitting the data into training and testing data sets, the target variable is ‘GraduateONTime’.

```
[27] target = fp['GraduateONTime']
     predictors = fp.drop(['GraduateONTime'], axis=1)
     X_train, X_test, y_train, y_test = train_test_split(predictors, target, test_size = 0.3, random_state = 0)
```

To determine which classification model predicts the student graduation on time,
We are using three classification models

1. Using Logistic Regression:

- Performing logistic regression model on the dataset.
- Also given the confusion matrix for the logistic regression model.

```
[28] from sklearn.linear_model import LogisticRegression
     classifier = LogisticRegression(random_state = 0)
     classifier.fit(X_train, y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: Convergen
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression(random_state=0)
```

```
[29] y_pred = classifier.predict(X_test)
```

```
[30] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
```

```
[[1451    7]
 [ 527    6]]
```

- ***Performance of this Model is given below, as we see the Accuracy of Logistic regression model is 0.73, the F1-scores is 0.84.***

```
[31] print(metrics.classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.73	1.00	0.84	1458
1	0.46	0.01	0.02	533
accuracy			0.73	1991
macro avg	0.60	0.50	0.43	1991
weighted avg	0.66	0.73	0.62	1991

2. Random Forest classification model:

- Training the Random Forest Classification model on the Training set.

```
[32] # Training the Random Forest Classification model on the Training set
     from sklearn.ensemble import RandomForestClassifier
     classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
     classifier.fit(X_train, y_train)
     y_pred = classifier.predict(X_test)
```

- ***Below is the performance of the Random Forest classification model, the accuracy of this model is very high which is 0.99 and the F1-score value is 0.99.***

```
[33] print(metrics.classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.99	1.00	0.99	1458
1	1.00	0.96	0.98	533
accuracy			0.99	1991
macro avg	0.99	0.98	0.99	1991
weighted avg	0.99	0.99	0.99	1991

3. Using K-NN Model:

- Training the K-NN model on the Training set.

```
[34] # Training the K-NN model on the Training set
     from sklearn.neighbors import KNeighborsClassifier
     classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
     classifier.fit(X_train, y_train)
     y_pred = classifier.predict(X_test)
```

Below is the performance of the K-NN model, the accuracy of this model is 0.95 and the F1-score value is 0.97

```
[35] print(metrics.classification_report(y_test,y_pred))
```

```
                   precision    recall  f1-score   support

                0       0.96      0.97      0.97      1458
                1       0.93      0.89      0.91       533

         accuracy                           0.95      1991
        macro avg       0.94      0.93      0.94      1991
     weighted avg       0.95      0.95      0.95      1991
```

- From all the three models and prediction we have the below accuracy and F1 scores:

Model	Accuracy	F1-score
Logistic Regression model	0.73	0.84
Random Forest classification model	0.99	0.99
K-NN classification model	0.95	0.97

So according to these prediction Random Forest classification model has the best performance on the give dataset.