How to work on course projects

Every project is divided into 2 parts. Part 1 of every project is a quiz. The quizzes need to be answered based on data exploration of the dataset given and some generic questions about algorithms discussed in the course. Hence before attempting the quiz you need to conduct exploratory analysis of the data along with data cleaning. For the quizzes, you need to consider only the training dataset. Part 2 is where you create the machine learning model using the exploratory analysis and data cleaning you have done for part 1. More detailed explanation of what needs to be done for each project is available in each of the project descriptions. The below guide is for you to understand how to approach and work on a project and is just an example.

In this guide we will discuss how to work on a sample classification problem. We are not discussing problem details here in terms of what all variables explicitly mean, since the purpose of this guide is to provide you with a process map to follow to work on the projects. This is an example here, you don't need to work on this, just follow a similar process to make a basic submission for any of the projects. Please note however that basic the submission that you make using the process mentioned here will not guarantee a score which is higher than the required threshold in individual projects. At the end of the guide, we will also discuss how to improve on your submissions.

In [1]:
```
# List of commonly shortcuts that can be used in Jupyter notebook
# Shift + Enter: Run the cell
# Esc + A : Insert a cell above the current cell
# Esc + B : Insert a cell below the current cell
# Ctrl + / : Comment the selected code
# Shift + Ctrl + - : Break the cell at the cursor position
```

Classification Problem - whether a person lies in a revenue.grid or not (1 or 0)

We have been provided with train and test data separately, where test data doesn't have response values. We need to use train data to build our model and then use that model to make prediction on test data and upload the csv file to LMS. In this we have been given data bd_train and bd_test and we are required to predict revenue.grid, whether it takes value 1 or not.

In [2]:
```
import numpy as np
import pandas as pd
```

In [3]:
```
# Read the data
bd_train = pd.read_csv(r"C:\Users\anjal\Dropbox\0.0 Data\bd_train.csv")
```

In [4]:
```
bd_test = pd.read_csv(r"C:\Users\anjal\Dropbox\0.0 Data\bd_test.csv")
```

In [5]:
```
bd_train.describe()
```

Out[5]:

	REF_NO	year_last_moved	Average.Credit.Card.Transaction	Balance.Transfer	Term.De
count	8124.000000	8124.000000	8124.000000	8124.000000	8124.0C
mean	5750.928730	1967.511571	23.240174	46.369877	27.5E
std	3316.648614	185.166494	50.714620	79.313730	53.9E
min	1.000000	0.000000	0.000000	0.000000	0.0C
25%	2900.750000	1978.000000	0.000000	0.000000	0.0C
50%	5728.500000	1988.000000	0.000000	17.990000	0.0C
75%	8621.500000	1994.000000	23.480000	65.452500	34.9S
max	11518.000000	1999.000000	662.260000	2951.760000	784.82

In [6]:
```
bd_train.head()
```

Out[6]:

	REF_NO	children	age_band	status	occupation	occupation_partner	home_status	fa
0	4888	Zero	55-60	Widowed	Retired	Unknown	Own Home	
1	8525	Zero	61-65	Partner	Retired	Retired	Own Home	
2	3411	3	31-35	Partner	Professional	Housewife	Own Home	
3	692	Zero	51-55	Partner	Secretarial/Admin	Other	Own Home	
4	10726	1	51-55	Partner	Retired	Retired	Own Home	

5 rows × 32 columns

Combine the train and test datasets

We will need same set of variables/features on both train and test datasets. It's easier to manage when we combine the train and test datasets in the beginning itself and then separate them once we are done with data preparation. Before combining however, we'll need some placeholder column which can be used to differentiate between the observations coming from train and test datasets. Also, we will need to add a column for the response variable to the test dataset in order to have the same columns in both train and test datasets. We'll fill test datasets response column with nan's.

In [7]:
```
bd_train.shape
```

Out[7]: (8124, 32)

In [8]:
```
bd_test.shape
```
Out[8]:
```
(2031, 31)
```

In [9]:
```
bd_train.columns
```
Out[9]:
```
Index(['REF_NO', 'children', 'age_band', 'status', 'occupation',
       'occupation_partner', 'home_status', 'family_income', 'self_employed',
       'self_employed_partner', 'year_last_moved', 'TVarea', 'post_code',
       'post_area', 'Average.Credit.Card.Transaction', 'Balance.Transfer',
       'Term.Deposit', 'Life.Insurance', 'Medical.Insurance',
       'Average.A.C.Balance', 'Personal.Loan', 'Investment.in.Mutual.Fund',
       'Investment.Tax.Saving.Bond', 'Home.Loan', 'Online.Purchase.Amount',
       'Revenue.Grid', 'gender', 'region', 'Investment.in.Commudity',
       'Investment.in.Equity', 'Investment.in.Derivative',
       'Portfolio.Balance'],
      dtype='object')
```

In [10]:
```
bd_test.columns # Revenue.Grid outcome variable not present in bd_test
```
Out[10]:
```
Index(['REF_NO', 'children', 'age_band', 'status', 'occupation',
       'occupation_partner', 'home_status', 'family_income', 'self_employed',
       'self_employed_partner', 'year_last_moved', 'TVarea', 'post_code',
       'post_area', 'Average.Credit.Card.Transaction', 'Balance.Transfer',
       'Term.Deposit', 'Life.Insurance', 'Medical.Insurance',
       'Average.A.C.Balance', 'Personal.Loan', 'Investment.in.Mutual.Fund',
       'Investment.Tax.Saving.Bond', 'Home.Loan', 'Online.Purchase.Amount',
       'gender', 'region', 'Investment.in.Commudity', 'Investment.in.Equity',
       'Investment.in.Derivative', 'Portfolio.Balance'],
      dtype='object')
```

bd_test does not have the outcome variable 'Revenue.Grid' - in order to combine the two we will add the response variable to ld_test

In [11]:
```
bd_test['Revenue.Grid']=np.nan
```

In [12]:
```
# combine the two datasets to pre-process the data together since the data on
 which the model is trained should
# undergo the same preprocessing as the data on which the predictions are made
```

In [13]:
```
bd_train['data'] = 'train'
bd_test['data'] = 'test'
bd_test=bd_test[bd_train.columns] # the columns in the two data frames should
 be in the same order to enable concatenation
bd_all=pd.concat([bd_train,bd_test],axis=0)
```

In [14]:
```
bd_all.head()
```

Out[14]:

	REF_NO	children	age_band	status	occupation	occupation_partner	home_status	fa
0	4888	Zero	55-60	Widowed	Retired	Unknown	Own Home	
1	8525	Zero	61-65	Partner	Retired	Retired	Own Home	
2	3411	3	31-35	Partner	Professional	Housewife	Own Home	
3	692	Zero	51-55	Partner	Secretarial/Admin	Other	Own Home	
4	10726	1	51-55	Partner	Retired	Retired	Own Home	

5 rows × 33 columns

In [15]:
```
bd_all.shape
```

Out[15]:
```
(10155, 33)
```

In [16]:
```
bd_all.dtypes
```

Out[16]:
```
REF_NO                              int64
children                           object
age_band                           object
status                             object
occupation                         object
occupation_partner                 object
home_status                        object
family_income                      object
self_employed                      object
self_employed_partner              object
year_last_moved                     int64
TVarea                             object
post_code                          object
post_area                          object
Average.Credit.Card.Transaction   float64
Balance.Transfer                  float64
Term.Deposit                      float64
Life.Insurance                    float64
Medical.Insurance                 float64
Average.A.C.Balance               float64
Personal.Loan                     float64
Investment.in.Mutual.Fund         float64
Investment.Tax.Saving.Bond        float64
Home.Loan                         float64
Online.Purchase.Amount            float64
Revenue.Grid                      float64
gender                             object
region                             object
Investment.in.Commudity           float64
Investment.in.Equity              float64
Investment.in.Derivative          float64
Portfolio.Balance                 float64
data                               object
dtype: object
```

In [17]:
```
bd_all.describe()
```

Out[17]:

	REF_NO	year_last_moved	Average.Credit.Card.Transaction	Balance.Transfer	Term.De
count	10155.000000	10155.000000	10155.000000	10155.000000	10155.0(
mean	5770.830822	1968.376366	23.441757	46.417760	27.5;
std	3324.837813	180.202242	50.872127	78.477609	53.9!
min	1.000000	0.000000	0.000000	0.000000	0.0(
25%	2903.500000	1978.000000	0.000000	0.000000	0.0(
50%	5770.000000	1988.000000	0.000000	17.960000	0.0(
75%	8665.500000	1994.000000	23.980000	65.385000	34.9(
max	11518.000000	1999.000000	662.260000	2951.760000	784.8:

Handling each variable

In [18]:
```
list(zip(bd_all.columns,bd_all.dtypes,bd_all.nunique())) # gives the type and
 the number of unique values present for each column
```

Out[18]:
```
[('REF_NO', dtype('int64'), 10155),
 ('children', dtype('O'), 5),
 ('age_band', dtype('O'), 13),
 ('status', dtype('O'), 5),
 ('occupation', dtype('O'), 9),
 ('occupation_partner', dtype('O'), 9),
 ('home_status', dtype('O'), 5),
 ('family_income', dtype('O'), 13),
 ('self_employed', dtype('O'), 2),
 ('self_employed_partner', dtype('O'), 2),
 ('year_last_moved', dtype('int64'), 95),
 ('TVarea', dtype('O'), 14),
 ('post_code', dtype('O'), 10040),
 ('post_area', dtype('O'), 2039),
 ('Average.Credit.Card.Transaction', dtype('float64'), 1411),
 ('Balance.Transfer', dtype('float64'), 2183),
 ('Term.Deposit', dtype('float64'), 1419),
 ('Life.Insurance', dtype('float64'), 3111),
 ('Medical.Insurance', dtype('float64'), 1589),
 ('Average.A.C.Balance', dtype('float64'), 2223),
 ('Personal.Loan', dtype('float64'), 1760),
 ('Investment.in.Mutual.Fund', dtype('float64'), 2470),
 ('Investment.Tax.Saving.Bond', dtype('float64'), 832),
 ('Home.Loan', dtype('float64'), 884),
 ('Online.Purchase.Amount', dtype('float64'), 1319),
 ('Revenue.Grid', dtype('float64'), 2),
 ('gender', dtype('O'), 3),
 ('region', dtype('O'), 13),
 ('Investment.in.Commudity', dtype('float64'), 3558),
 ('Investment.in.Equity', dtype('float64'), 3250),
 ('Investment.in.Derivative', dtype('float64'), 3796),
 ('Portfolio.Balance', dtype('float64'), 8317),
 ('data', dtype('O'), 2)]
```

In [19]:
```
# REF_NO,post_code , post_area  : drop
# children : Zero : 0 , 4+ : 4 and then convert to numeric
# age_band : dummies
# status , occupation , occupation_partner , home_status,family_income : dummi
es
# self_employed, ` : dummies
# TVArea , Region , gender : dummies
# Revenue Grid : 1,2 : 1,0
```

In [20]:
```
bd_all['children'].value_counts()
```

Out[20]:
```
Zero    6208
1       1848
2       1607
3        473
4+        19
Name: children, dtype: int64
```

In [21]:
```
# children : Zero : 0 , 4+ : 4 and then convert to numeric
bd_all['children']=np.where(bd_all['children']=='Zero',0,bd_all['children']) # replace 'Zero' with 0
bd_all['children']=np.where(bd_all['children'][:1]=='4',4,bd_all['children'])
# replace '4' and '4+' with 4
bd_all['children']=pd.to_numeric(bd_all['children'],errors='coerce')
```

If we look at variables post_code and post_area, you will notice that they have more than 2000 unique values, having none of the individual values having frequency higher than a good number , we’ll drop those variables.

In [22]:
```
print(bd_all['post_area'].unique()) # gives the unique values in the column
print(bd_all['post_area'].nunique()) # gives the number of unique values
bd_all['post_area'].value_counts() # gives counts of each unique value in the
 column
```

```
['L15' 'CW11' 'PR8' ... 'PO39' 'WA44' 'WV4']
2039
```

Out[22]:
```
PR5      35
PR4      33
WA7      28
TQ12     28
M12      26
WA4      26
LE7      25
BH23     25
LE9      24
M33      24
SK8      22
PR8      22
M13      21
WA5      21
M14      21
TS5      21
HD7      21
WA3      20
NG9      20
PR2      20
PR7      20
PR6      20
L6       20
DY8      20
L12      20
L13      20
CF62     20
L8       19
L15      19
L40      19
         ..
PL29      1
HP8       1
NW8       1
IP2       1
EX9       1
DL4       1
TW5       1
BS3       1
OL5       1
BD7       1
BH2       1
IP21      1
CA9       1
BD14      1
WV4       1
AB13      1
NE17      1
W10       1
PO16      1
OX7       1
SO24      1
N12       1
TN21      1
CA16      1
LL57      1
PL27      1
```

```
PL9      1
SW16     1
MK19     1
TN17     1
Name: post_area, Length: 2039, dtype: int64
```

In [23]:
```
print(bd_all['post_code'].unique())
print(bd_all['post_code'].nunique())
bd_all['post_code'].value_counts()
```

```
['L15 6LL' 'CW11 1RA' 'PR8 6SQ' ... 'CF43 4SU' 'IV30 6AX' 'SM4 5RF']
10040
```

```
Out[23]: SA2 0FN      2
         MK7 7HH      2
         YO8 9JZ      2
         BD20 8BY     2
         L31 6NN      2
         CH3 9HA      2
         S41 0SU      2
         NP13 1QD     2
         CF48 1EW     2
         EN4 8QQ      2
         SN14 6PS     2
         M14 7NQ      2
         KY11 9GF     2
         CW1 5TY      2
         TQ14 8RZ     2
         WS9 9RD      2
         DT9 6LA      2
         LE65 1RN     2
         L9 7BN       2
         IG9 5DZ      2
         CF3 4DF      2
         CM8 2QR      2
         CM6 2JA      2
         GL52 6SX     2
         CV32 6HE     2
         AB23 8QW     2
         HU9 4BT      2
         BH1 1QG      2
         TQ12 6YA     2
         LU7 7UQ      2
                     ..
         S72 8HJ      1
         IV15 9TU     1
         CH2 1QH      1
         WA4 5HP      1
         M4 8LZ       1
         CF62 6LZ     1
         B97 5HF      1
         M46 0AY      1
         EX31 2NY     1
         L53 3NE      1
         WA4 2LF      1
         TQ9 7ES      1
         LU3 3HF      1
         L1 4DN       1
         SA11 3TQ     1
         DT11 0HW     1
         M14 5ES      1
         RG26 3YN     1
         DE56 0HX     1
         DT9 6PT      1
         CF10 1DL     1
         SP10 4LP     1
         TS12 1PE     1
         YO32 9QA     1
         NE33 5SZ     1
         CF45 4LD     1
```

```
G62 6QL     1
RG5 4TJ     1
TN36 4BJ    1
BL0 9LH     1
Name: post_code, Length: 10040, dtype: int64
```

In [24]:
```
# Dropping the variables Post.Code and Post.Area
bd_all.drop(['post_code','post_area'], axis=1, inplace=True)
```

In [25]:
```
# Dropping Ref_no column since it has no use in building the model
bd_all.drop('REF_NO', axis=1, inplace=True)
```

In [26]:
```
# We will create dummies for the rest of the categorical columns using the bui
lt-in function from pandas
```

In [27]:
```
cat_vars=bd_all.select_dtypes(['object']).columns # cat_vars variable contains
 all the categorical variables
cat_vars # we will create dummies for all of them using the built-in pandas li
brary
```

Out[27]:
```
Index(['age_band', 'status', 'occupation', 'occupation_partner', 'home_statu
s',
       'family_income', 'self_employed', 'self_employed_partner', 'TVarea',
       'gender', 'region', 'data'],
      dtype='object')
```

In [28]:
```
for col in cat_vars[:-1]: # we do not consider the 'data' column since that is
 only a placeholder for train and test sets
    dummy=pd.get_dummies(bd_all[col],drop_first=True,prefix=col)
    bd_all=pd.concat([bd_all,dummy],axis=1)
    del bd_all[col]
    print(col) # gives the columns for which the dummies are created and which
 have been deleted
del dummy
# we use a for loop to create dummies for each categorical column/ feature/ va
riable
```

```
age_band
status
occupation
occupation_partner
home_status
family_income
self_employed
self_employed_partner
TVarea
gender
region
```

You can also use the method to create dummies as described in the Project Process Guide for Regression - Python. That functions enables you to ignore categories with low frequencies

In [29]:
```
# we could also have created dummies separately for each variable as follows:
# dummy1=pd.get_dummies(bd_all['age_band'],drop_first=True,prefix='age_band')
# del bd_all['age_band']
# dummy2=pd.get_dummies(bd_all['status'],drop_first=True,prefix='status')
# del bd_all['status']
# dummy3=pd.get_dummies(bd_all['occupation'],drop_first=True,prefix='occupatio
n')
# del bd_all['occupation']
# dummy4=pd.get_dummies(bd_all['occupation_partner'],drop_first=True,prefix='o
ccupation_partner')
# del bd_all['occupation_partner']
# dummy5=pd.get_dummies(bd_all['home_status'],drop_first=True,prefix='home_sta
tus')
# del bd_all['home_status']
# dummy6=pd.get_dummies(bd_all['family_income'],drop_first=True,prefix='family
_income')
# del bd_all['family_income']
# dummy7=pd.get_dummies(bd_all['self_employed'],drop_first=True,prefix='self_e
mployed')
# del bd_all['self_employed']
# dummy8=pd.get_dummies(bd_all['self_employed_partner'],drop_first=True,prefix
='self_employed_partner')
# del bd_all['self_employed_partner']
# dummy9=pd.get_dummies(bd_all['TVarea'],drop_first=True,prefix='TVarea')
# del bd_all['TVarea']
# dummy10=pd.get_dummies(bd_all['gender'],drop_first=True,prefix='gender')
# del bd_all['gender']
# dummy11=pd.get_dummies(bd_all['region'],drop_first=True,prefix='region')
# del bd_all['region']
# test1 = pd.concat([bd_all,dummy1,dummy2,dummy3,dummy4,dummy5,dummy6,dummy7,d
ummy8,dummy9,dummy10,dummy11],axis=1)
# test1.columns
# len(test1.columns)
```

In [30]:
```
bd_all.shape
```

Out[30]:
```
(10155, 96)
```

In [31]:
```
bd_all.dtypes # check if all the columns are numeric
```

Out[31]:
```
children                          float64
year_last_moved                     int64
Average.Credit.Card.Transaction   float64
Balance.Transfer                  float64
Term.Deposit                      float64
Life.Insurance                    float64
Medical.Insurance                 float64
Average.A.C.Balance               float64
Personal.Loan                     float64
Investment.in.Mutual.Fund         float64
Investment.Tax.Saving.Bond        float64
Home.Loan                         float64
Online.Purchase.Amount            float64
Revenue.Grid                      float64
Investment.in.Commudity           float64
Investment.in.Equity              float64
Investment.in.Derivative          float64
Portfolio.Balance                 float64
data                               object
age_band_22-25                      uint8
age_band_26-30                      uint8
age_band_31-35                      uint8
age_band_36-40                      uint8
age_band_41-45                      uint8
age_band_45-50                      uint8
age_band_51-55                      uint8
age_band_55-60                      uint8
age_band_61-65                      uint8
age_band_65-70                      uint8
age_band_71+                        uint8
                                   ...
family_income_Unknown               uint8
self_employed_Yes                   uint8
self_employed_partner_Yes           uint8
TVarea_Border                       uint8
TVarea_Carlton                      uint8
TVarea_Central                      uint8
TVarea_Grampian                     uint8
TVarea_Granada                      uint8
TVarea_HTV                          uint8
TVarea_Meridian                     uint8
TVarea_Scottish TV                  uint8
TVarea_TV South West                uint8
TVarea_Tyne Tees                    uint8
TVarea_Ulster                       uint8
TVarea_Unknown                      uint8
TVarea_Yorkshire                    uint8
gender_Male                         uint8
gender_Unknown                      uint8
region_East Anglia                  uint8
region_East Midlands                uint8
region_Isle of Man                  uint8
region_North                        uint8
region_North West                   uint8
region_Northern Ireland             uint8
region_Scotland                     uint8
region_South East                   uint8
```

```
region_South West                    uint8
region_Unknown                       uint8
region_Wales                         uint8
region_West Midlands                 uint8
Length: 96, dtype: object
```

In [32]:
```
# Analyzing the outcome variable
bd_all['Revenue.Grid'].value_counts()
```

Out[32]:
```
2.0    7256
1.0     868
Name: Revenue.Grid, dtype: int64
```

In [33]:
```
# Change the response variable value to '1' and '0' from '2' and '1'
bd_all['Revenue.Grid']=(bd_all['Revenue.Grid']==1).astype(int)
```

Check for missing values

In [34]:
```
bd_all.isnull().sum()
```

Out[34]:
```
children                           19
year_last_moved                     0
Average.Credit.Card.Transaction     0
Balance.Transfer                    0
Term.Deposit                        0
Life.Insurance                      0
Medical.Insurance                   0
Average.A.C.Balance                 0
Personal.Loan                       0
Investment.in.Mutual.Fund           0
Investment.Tax.Saving.Bond          0
Home.Loan                           0
Online.Purchase.Amount              0
Revenue.Grid                        0
Investment.in.Commudity             0
Investment.in.Equity                0
Investment.in.Derivative            0
Portfolio.Balance                   0
data                                0
age_band_22-25                      0
age_band_26-30                      0
age_band_31-35                      0
age_band_36-40                      0
age_band_41-45                      0
age_band_45-50                      0
age_band_51-55                      0
age_band_55-60                      0
age_band_61-65                      0
age_band_65-70                      0
age_band_71+                        0
                                   ..
family_income_Unknown               0
self_employed_Yes                   0
self_employed_partner_Yes           0
TVarea_Border                       0
TVarea_Carlton                      0
TVarea_Central                      0
TVarea_Grampian                     0
TVarea_Granada                      0
TVarea_HTV                          0
TVarea_Meridian                     0
TVarea_Scottish TV                  0
TVarea_TV South West                0
TVarea_Tyne Tees                    0
TVarea_Ulster                       0
TVarea_Unknown                      0
TVarea_Yorkshire                    0
gender_Male                         0
gender_Unknown                      0
region_East Anglia                  0
region_East Midlands                0
region_Isle of Man                  0
region_North                        0
region_North West                   0
region_Northern Ireland             0
region_Scotland                     0
region_South East                   0
```

```
region_South West                  0
region_Unknown                     0
region_Wales                       0
region_West Midlands               0
Length: 96, dtype: int64
```

In [35]:
```
# Only the feature 'children' has missing values. We ignore the missing values
 in the response variable 'Revenue.Grid'
# Handling missing values in 'children' taking the mean only from the train da
taset
bd_all.loc[bd_all['children'].isnull(),'children']=bd_all.loc[bd_all['data']==
'train','children'].mean()
```

In [36]:
```
# Check again if there are any missing values
bd_all.isnull().sum()
```

Out[36]:
```
children                            0
year_last_moved                     0
Average.Credit.Card.Transaction     0
Balance.Transfer                    0
Term.Deposit                        0
Life.Insurance                      0
Medical.Insurance                   0
Average.A.C.Balance                 0
Personal.Loan                       0
Investment.in.Mutual.Fund           0
Investment.Tax.Saving.Bond          0
Home.Loan                           0
Online.Purchase.Amount              0
Revenue.Grid                        0
Investment.in.Commudity             0
Investment.in.Equity                0
Investment.in.Derivative            0
Portfolio.Balance                   0
data                                0
age_band_22-25                      0
age_band_26-30                      0
age_band_31-35                      0
age_band_36-40                      0
age_band_41-45                      0
age_band_45-50                      0
age_band_51-55                      0
age_band_55-60                      0
age_band_61-65                      0
age_band_65-70                      0
age_band_71+                        0
                                   ..
family_income_Unknown               0
self_employed_Yes                   0
self_employed_partner_Yes           0
TVarea_Border                       0
TVarea_Carlton                      0
TVarea_Central                      0
TVarea_Grampian                     0
TVarea_Granada                      0
TVarea_HTV                          0
TVarea_Meridian                     0
TVarea_Scottish TV                  0
TVarea_TV South West                0
TVarea_Tyne Tees                    0
TVarea_Ulster                       0
TVarea_Unknown                      0
TVarea_Yorkshire                    0
gender_Male                         0
gender_Unknown                      0
region_East Anglia                  0
region_East Midlands                0
region_Isle of Man                  0
region_North                        0
region_North West                   0
region_Northern Ireland             0
region_Scotland                     0
region_South East                   0
```

```
region_South West                0
region_Unknown                   0
region_Wales                     0
region_West Midlands             0
Length: 96, dtype: int64
```

In [37]:
```
# Data preprocessing is complete - the data is in the expected format
```

Let's separate our two data sets and remove the unnecessary columns that we added while combining them.

In [38]:
```
bd_train=bd_all[bd_all['data']=='train']  # Select only those rows where the d
ata column has the value 'train'
del bd_train['data']  # Remove the data column from bd_train data frame
```

In [39]:
```
bd_test=bd_all[bd_all['data']=='test']  # Select only those rows where the dat
a column has the value 'test'
bd_test.drop(['Revenue.Grid','data'],axis=1,inplace=True)  # Remove the data a
nd Interest.Rate column from bd_test data frame
```

```
C:\Users\anjal\Anaconda3\lib\site-packages\pandas\core\frame.py:3697: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  errors=errors)
```

In [40]:
```
del bd_all # After data pre-processing, we do not need the bd_all data frame
```

Model Implementation

Let's build a model on training data. We are not going to build any complex model, it's up to you to use other algorithms that you learn during the course.

In [41]:
```
# In the project process guide for regression, we broke the train data further
 into train and validation datasets to enable us to
# assess the models performance
# Instead, here we will use cross validation to get the best parameters for th
e model and assess model performance
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
params={'class_weight':['balanced',None],
        'penalty':['l1','l2'],
        'C':np.linspace(0.01,1000,10)}
```

In [44]:
```
x_train=bd_train.drop('Revenue.Grid',axis=1)
y_train=bd_train['Revenue.Grid']
# y_train.dtype
```

In [45]:
```
model=LogisticRegression(fit_intercept=True)
```

In [46]:
```
from sklearn.model_selection import GridSearchCV
```

In [47]:
```
grid_search=GridSearchCV(model,param_grid=params,cv=5,scoring="roc_auc")
```

In [48]:
```
grid_search.fit(x_train,y_train) # grid search fits a Logistic Regression mode
l for every combination of parameters
                                  # we can then select the parameter combinatio
n that gives the best performance
                                  # Randomnized search on the other hand would
 take a subset of parameter combinations and fit
                                  # the model. Algorithms in which many paramet
ers need to be tuned, randomized search makes
                                  # more sense
```

Out[48]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit
_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'class_weight': ['balanced', None], 'penalty': ['l1', 'l
2'], 'C': array([1.0000e-02, 1.1112e+02, 2.2223e+02, 3.3334e+02, 4.4445e+02,
       5.5556e+02, 6.6667e+02, 7.7778e+02, 8.8889e+02, 1.0000e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [49]:
```
grid_search.best_estimator_ # returns the parameters giving the best performan
ce
                            # grid search with cv chose C as 0.01, class_weigh
t as 'balanced' and penalty as 'l1'
```

Out[49]:
```
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [50]:
```
logr=grid_search.best_estimator_
```

Using the report function given below you can see the cv performance of top few models that will be the tentative performance

In [51]:
```
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

In [52]:
```
report(grid_search.cv_results_,5)
```

```
Model with rank: 1
Mean validation score: 0.955 (std: 0.005)
Parameters: {'C': 0.01, 'class_weight': 'balanced', 'penalty': 'l1'}

Model with rank: 2
Mean validation score: 0.955 (std: 0.005)
Parameters: {'C': 0.01, 'class_weight': 'balanced', 'penalty': 'l2'}

Model with rank: 3
Mean validation score: 0.951 (std: 0.005)
Parameters: {'C': 888.89, 'class_weight': 'balanced', 'penalty': 'l2'}

Model with rank: 4
Mean validation score: 0.951 (std: 0.005)
Parameters: {'C': 555.56, 'class_weight': 'balanced', 'penalty': 'l2'}

Model with rank: 5
Mean validation score: 0.951 (std: 0.005)
Parameters: {'C': 444.45, 'class_weight': 'balanced', 'penalty': 'l2'}
```

In [53]:
```
logr.fit(x_train,y_train) # we fit the model with the best parameters
```

Out[53]:
```
LogisticRegression(C=0.01, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [54]:
```
train_score=logr.predict_proba(x_train)[:,1] # we can get the probabilities us
ing predict_proba
```

In [55]:
```
train_score
```

Out[55]:
```
array([0.99420408, 0.01137188, 0.06167701, ..., 0.80850512, 0.17254272,
       0.03292699])
```

In case you have to submit probabilities, please use the following code:

In [56]:
```
test_score=logr.predict_proba(bd_test)[:,1]
test_score
```

Out[56]:
```
array([0.25083295, 0.38726091, 0.00597118, ..., 0.00525166, 0.04848874,
       0.11929218])
```

In [57]:
```
pd.DataFrame(test_score).to_csv("mysubmission.csv",index=False)
```

Upload this csv file to LMS under that project submission tab

However, we may want hard classes instead of probabilities

In [58]:
```
cutoffs=np.linspace(0.01,0.99,99)
cutoffs
```

Out[58]:
```
array([0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 , 0.11,
       0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 , 0.21, 0.22,
       0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 , 0.31, 0.32, 0.33,
       0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 , 0.41, 0.42, 0.43, 0.44,
       0.45, 0.46, 0.47, 0.48, 0.49, 0.5 , 0.51, 0.52, 0.53, 0.54, 0.55,
       0.56, 0.57, 0.58, 0.59, 0.6 , 0.61, 0.62, 0.63, 0.64, 0.65, 0.66,
       0.67, 0.68, 0.69, 0.7 , 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77,
       0.78, 0.79, 0.8 , 0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88,
       0.89, 0.9 , 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99])
```

In [59]:
```
train_score=logr.predict_proba(x_train)[:,1] # the predicted response variable
 values
real=y_train # the actual response variable values
print(logr.classes_) # In  order to find the probability of which column is fo
r outcome 1 and which for outcome 0
```

```
[0 1]
```

In [60]:
```
from sklearn.metrics import fbeta_score
```

In [61]:
```
KS_all=[]

for cutoff in cutoffs:

    predicted=(train_score>cutoff).astype(int)

    TP=((predicted==1) & (real==1)).sum()
    TN=((predicted==0) & (real==0)).sum()
    FP=((predicted==1) & (real==0)).sum()
    FN=((predicted==0) & (real==1)).sum()

    P=TP+FN
    N=TN+FP


    KS=(TP/P)-(FP/N)

    KS_all.append(KS)

    # try out what cutoffs you get when you use F_beta scores with different v
alues of betas [0.5 , 5]
    # beta < 1 : you will get cutoff , which is high ( favours precision)
    # beta > 1 : you will get cutoff , which is low (favours precision )
```

In [62]:
```
list(zip(cutoffs,KS_all))
```

Out[62]:
```
[(0.01, 0.10276192847235277),
 (0.02, 0.1574644724340638),
 (0.03, 0.2101988375106062),
 (0.04, 0.25980215324739997),
 (0.05, 0.30922891082669857),
 (0.060000000000000005, 0.3677515890234174),
 (0.06999999999999999, 0.43371638408893454),
 (0.08, 0.5118090733110117),
 (0.09, 0.5500231176868087),
 (0.09999999999999999, 0.583088395937384),
 (0.11, 0.6104148989680873),
 (0.12, 0.6311261870043035),
 (0.13, 0.6496819412759948),
 (0.14, 0.6632267464015161),
 (0.15000000000000002, 0.6780614422388083),
 (0.16, 0.6912423343274785),
 (0.17, 0.7019425207932162),
 (0.18000000000000002, 0.7109393656100275),
 (0.19, 0.7192471255315798),
 (0.2, 0.7271414345159766),
 (0.21000000000000002, 0.7350852814006777),
 (0.22, 0.7435803962015862),
 (0.23, 0.7490435374633546),
 (0.24000000000000002, 0.7535807010502035),
 (0.25, 0.7599202820865871),
 (0.26, 0.7634044477413258),
 (0.27, 0.7672137852544725),
 (0.28, 0.7708853057885672),
 (0.29000000000000004, 0.7717014109410169),
 (0.3, 0.7749982217163993),
 (0.31, 0.7802352669203685),
 (0.32, 0.7843202383916187),
 (0.33, 0.7860127833186837),
 (0.34, 0.7896843038527784),
 (0.35000000000000003, 0.7917902997169989),
 (0.36000000000000004, 0.792330770911345),
 (0.37, 0.7919948023310758),
 (0.38, 0.796404945660734),
 (0.39, 0.7997125531579776),
 (0.4, 0.798963133640553),
 (0.41000000000000003, 0.8015321183422331),
 (0.42000000000000004, 0.8051153597975805),
 (0.43, 0.8059810028503346),
 (0.44, 0.8031643286471326),
 (0.45, 0.8060089473069165),
 (0.46, 0.8083022980504931),
 (0.47000000000000003, 0.8079275882917807),
 (0.48000000000000004, 0.8093940371610464),
 (0.49, 0.8066161041362876),
 (0.5, 0.8064674904353746),
 (0.51, 0.8096372809535665),
 (0.52, 0.8053433611592378),
 (0.53, 0.805244285358629),
 (0.54, 0.8052830265370721),
 (0.55, 0.8063360244691823),
 (0.56, 0.7978967985814377),
 (0.5700000000000001, 0.7983489906970364),
```

```
 (0.5800000000000001, 0.7946558767192191),
 (0.59, 0.7906871287832984),
 (0.6, 0.7911393208988969),
 (0.61, 0.7856050482931018),
 (0.62, 0.7833396420061071),
 (0.63, 0.7807986017610088),
 (0.64, 0.7814381487559636),
 (0.65, 0.7790844633902214),
 (0.66, 0.7739636417215817),
 (0.67, 0.7665386725875043),
 (0.68, 0.7641354493214578),
 (0.6900000000000001, 0.759014627652818 1),
 (0.7000000000000001, 0.7524660982933559),
 (0.7100000000000001, 0.7469318256875607),
 (0.72, 0.7422244549560764),
 (0.73, 0.7312054476447903),
 (0.74, 0.7268232487717141),
 (0.75, 0.7211016212865629),
 (0.76, 0.7129875672572261),
 (0.77, 0.7093439911797134),
 (0.78, 0.7037601806736138),
 (0.79, 0.6917269166086607),
 (0.8, 0.6838389586371235),
 (0.81, 0.6789937709265874),
 (0.8200000000000001, 0.6741981211163556),
 (0.8300000000000001, 0.6637799196215813),
 (0.8400000000000001, 0.6513332046194727),
 (0.85, 0.6420670768574172),
 (0.86, 0.6304968016299239),
 (0.87, 0.61763663569066),
 (0.88, 0.6132544368175836),
 (0.89, 0.6082714321279957),
 (0.9, 0.5906651542788044),
 (0.91, 0.5752252069159989),
 (0.92, 0.5502434978330344),
 (0.93, 0.532637219983843 1),
 (0.9400000000000001, 0.519451882186171),
 (0.9500000000000001, 0.49672478266833997),
 (0.9600000000000001, 0.4740472210508132),
 (0.97, 0.44182789771312725),
 (0.98, 0.4103471971710048),
 (0.99, 0.3536586914881185)]
```

In [63]:
```
mycutoff=cutoffs[KS_all==max(KS_all)][0]
mycutoff # gives the cutoff value where KS is maximum
```

Out[63]:
```
0.51
```

In [64]:
```
test_score=logr.predict_proba(bd_test)[:,1]
test_score
```

Out[64]:
```
array([0.25083295, 0.38726091, 0.00597118, ..., 0.00525166, 0.04848874,
       0.11929218])
```

If you had to submit hardclasses, you can apply the cutoff obtained above and then submit

In [65]:
```
test_classes=(test_score>mycutoff).astype(int)
```

In [66]:
```
pd.DataFrame(test_classes).to_csv("mysubmission.csv",index=False)
# Please give the file a proper name before submission
```

This csv file is what you need to upload on LMS.

Passing the project

You'll clear a project if you clear the threshold. Every project has a different threshold, clearly mentioned in the project problem statement page. Do go through the problem statement page and find that out before assuming that you cleared a project just by making a submission. You are free to make as many submissions as you want; maximum score among all your submissions will be considered as your latest score.

Improving your score

Try a non linear algorithm such as dtrees, random forests, extratrees or gbm; they almost always give better performance than a simple linear model. Tune your parameters for above mentioned algorithms. Try stacking. Please remember to do one thing at a time.

How do I assess my model performance before submission

You can break your given train data into two parts, build model on one check its performance on the other. That will give you an idea about how your model might perform after submission. You can not assess performance on given test data because response has been removed from it. Make sure that the submission that you make is based on the model built on entire training data.

Do i need to do this all by myself?

Projects in the course are an integrated learning process. If you get stuck somewhere, feel free to ask for help on QA forum. However, don't expect outright solutions, such requests will be ignored and deleted from QA forum. Hope this helps; feel free to suggest if we need to include anything else in here.