Bank Direct Target Marketing

M1: Simple Logistic Regression

1) data prep

2) Model

- Logistic Regression
- print model (coef, intercept)

3) Evaluation

- confusion matrix
- classification report

M2: Categorical

- dummy code
- recode - ordinal

M3: Model Selection

- backward selection

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institut
client will subscribe a term deposit (variable y).

Data Set Information: The data is related with direct marketing campaigns of a Portuguese bankin
based on phone calls. Often, more than one contact to the same client was required, in order to acc
be ('yes') or not ('no') subscribed.

Attribute Information:

Bank client data:

- Age (numeric)
- Job : type of job (categorical: 'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management', 'technician', 'unemployed', 'unknown')
- Marital : marital status (categorical: 'divorced', 'married', 'single', 'unknown' ; note: 'divorced' m
- Education (categorical: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.co
- Default: has credit in default? (categorical: 'no', 'yes', 'unknown')
- Housing: has housing loan? (categorical: 'no', 'yes', 'unknown')
- Loan: has personal loan? (categorical: 'no', 'yes', 'unknown')

Related with the last contact of the current campaign:

- Contact: contact communication type (categorical: 'cellular','telephone')
- Month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
- Day_of_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
- Duration: last contact duration, in seconds (numeric). Important note: this attribute highly aff y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the cal should only be included for benchmark purposes and should be discarded if the intention is t

Other attributes:

- Campaign: number of contacts performed during this campaign and for this client (numeric,
- Pdays: number of days that passed by after the client was last contacted from a previous car previously contacted)
- Previous: number of contacts performed before this campaign and for this client (numeric)
- Poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','su

Social and economic context attributes

- Emp.var.rate: employment variation rate - quarterly indicator (numeric)
- Cons.price.idx: consumer price index - monthly indicator (numeric)
- Cons.conf.idx: consumer confidence index - monthly indicator (numeric)
- Euribor3m: euribor 3 month rate - daily indicator (numeric)
- Nr.employed: number of employees - quarterly indicator (numeric)

Output variable (desired target):

- y - has the client subscribed a term deposit? (binary: 'yes', 'no')

Source:

- Dataset from : http://archive.ics.uci.edu/ml/datasets/Bank+Marketing#

```
# Importing Data Analysis Librarys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [13, 10]
```

```
bank = pd.read_csv('https://github.com/kaopanboonyuen/Python-Data-Science/raw/ma
# Convert dependent variable categorical to dummy
y = pd.get_dummies(bank['y'], columns = ['y'], prefix = ['y'], drop_first = True
bank.head()
```

	age	job	marital	education	default	housing	loan	contact	month	da
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

```
# take a look at the type, number of columns, entries, null values etc..
bank.info()
# bank.isnull().any() # one way to search for null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

1. Bank client data Analysis and Categorical Treatment

- Work with the atributes related to bank clients
- To make things more clear, i'm going to creat a new datasets that contains just this part of da

```
bank_client = bank.iloc[: , 0:7]
bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

1.1. Knowing the categorical variables

```
# knowing the categorical variables
print('Jobs:\n', bank_client['job'].unique())

print('Marital:\n', bank_client['marital'].unique())

print('Education:\n', bank_client['education'].unique())

print('Default:\n', bank_client['default'].unique())
print('Housing:\n', bank_client['housing'].unique())
print('Loan:\n', bank_client['loan'].unique())
```

```
Jobs:
 ['housemaid' 'services' 'admin.' 'blue-collar' 'technician' 'retired'
 'management' 'unemployed' 'self-employed' 'unknown' 'entrepreneur'
 'student']
Marital:
 ['married' 'single' 'divorced' 'unknown']
Education:
 ['basic.4y' 'high.school' 'basic.6y' 'basic.9y' 'professional.course'
 'unknown' 'university.degree' 'illiterate']
Default:
 ['no' 'unknown' 'yes']
Housing:
 ['no' 'yes' 'unknown']
Loan:
 ['no' 'yes' 'unknown']
```

1.2. Age

- Trying to find some insights crossing those variables

```
#Trying to find some strange values or null values
```

```
print('Min age: ', bank_client['age'].max())
print('Max age: ', bank_client['age'].min())
print('Null Values: ', bank_client['age'].isnull().any())
```

```
Min age:  98
Max age:  17
Null Values:  False
```

1.3. Bank Client Categorical Treatment

- Jobs, Marital, Education, Default, Housing, Loan. Converting to continuous due the feature sc

```
# Label encoder order is alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_client['job']      = labelencoder_X.fit_transform(bank_client['job'])
bank_client['marital']  = labelencoder_X.fit_transform(bank_client['marital'])
bank_client['education']= labelencoder_X.fit_transform(bank_client['education'])
bank_client['default']  = labelencoder_X.fit_transform(bank_client['default'])
bank_client['housing']  = labelencoder_X.fit_transform(bank_client['housing'])
bank_client['loan']     = labelencoder_X.fit_transform(bank_client['loan'])
```

```
# function for create age grouping, dues to 78 differente age values, we have to

def age(dataframe):
    dataframe.loc[dataframe['age'] <= 32, 'age'] = 1
    dataframe.loc[(dataframe['age'] > 32) & (dataframe['age'] <= 47), 'age'] = 2
    dataframe.loc[(dataframe['age'] > 47) & (dataframe['age'] <= 70), 'age'] = 3
    dataframe.loc[(dataframe['age'] > 70) & (dataframe['age'] <= 98), 'age'] = 4

    return dataframe

age(bank_client);
```

```
bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	3	3	1	0	0	0	0
1	3	7	1	3	1	0	0
2	2	7	1	3	0	2	0
3	2	0	1	1	0	0	0
4	3	7	1	3	0	0	2

2. Related with the last contact of the current campaign

- Treat categorical, see those values

- group continuous variables if necessary

```
# Slicing DataFrame to treat separately, make things more easy
bank_related = bank.iloc[: , 7:11]
bank_related.head()
```

	contact	month	day_of_week	duration
0	telephone	may	mon	261
1	telephone	may	mon	149
2	telephone	may	mon	226
3	telephone	may	mon	151
4	telephone	may	mon	307

```
bank_related.isnull().any()
```

```
contact        False
month          False
day_of_week    False
duration       False
dtype: bool
```

```
print("Kind of Contact: \n", bank_related['contact'].unique())
print("\nWhich monthis this campaing work: \n", bank_related['month'].unique())
print("\nWhich days of week this campaing work: \n", bank_related['day_of_week']
```

```
Kind of Contact:
 ['telephone' 'cellular']

Which monthis this campaing work:
 ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'mar' 'apr' 'sep']

Which days of week this campaing work:
 ['mon' 'tue' 'wed' 'thu' 'fri']
```

2.1 Contact, Month, Day of Week treatment

```
# Label encoder order is alphabetical
from sklearn.preprocessing import LabelEncoder
labelencoder_X = LabelEncoder()
bank_related['contact']     = labelencoder_X.fit_transform(bank_related['contact
bank_related['month']       = labelencoder_X.fit_transform(bank_related['month']
bank_related['day_of_week'] = labelencoder_X.fit_transform(bank_related['day_of_
```

** A way to Converting Categorical variables using dummies if you judge necessary **

> bank_related = pd.get_dummies(data = bank_related, prefix = ['contact'] , columns = ['contact']

> bank_related = pd.get_dummies(data = bank_related, prefix = ['month'] , columns = ['month'] , d

> bank_related = pd.get_dummies(data = bank_related, prefix = ['day_of_week'], columns = ['day_

```
bank_related.head()
```

	contact	month	day_of_week	duration
0	1	6	1	261
1	1	6	1	149
2	1	6	1	226
3	1	6	1	151
4	1	6	1	307

```
# Duration grouping function
def duration(data):

    data.loc[data['duration'] <= 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] <= 180)  , 'duration']
    data.loc[(data['duration'] > 180) & (data['duration'] <= 319)  , 'duration']
    data.loc[(data['duration'] > 319) & (data['duration'] <= 644.5), 'duration']
    data.loc[data['duration']  > 644.5, 'duration'] = 5

    return data
duration(bank_related);
```

```
bank_related.head()
```

	contact	month	day_of_week	duration
0	1	6	1	3
1	1	6	1	2
2	1	6	1	3
3	1	6	1	2
4	1	6	1	3

Social and economic context attributes

```
# The different between "iloc" and "loc" is iloc gets data by index but loc gets
bank_se = bank.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euri
bank_se.head()
```

	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed
0	1.1	93.994	-36.4	4.857	5191.0
1	1.1	93.994	-36.4	4.857	5191.0
2	1.1	93.994	-36.4	4.857	5191.0
3	1.1	93.994	-36.4	4.857	5191.0
4	1.1	93.994	-36.4	4.857	5191.0

Other attributes

```
bank_o = bank.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
bank_o.head()
```

	campaign	pdays	previous	poutcome
0	1	999	0	nonexistent
1	1	999	0	nonexistent
2	1	999	0	nonexistent
3	1	999	0	nonexistent
4	1	999	0	nonexistent

```
# get unique elements
bank_o['poutcome'].unique()
```

```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inpla
```

Model

```
bank_final= pd.concat([bank_client, bank_related, bank_se, bank_o], axis = 1)
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housi
                     'contact', 'month', 'day_of_week', 'duration', 'emp.var.rat
                     'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'p
bank_final.shape
```

```
(41188, 20)
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

# Split data in to training/ testing set
```

```
X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size = 0

```

```
X_train.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_
7271	2	9	1	2	1	0	2	1	6	
13284	1	3	2	3	0	0	0	0	3	
11580	3	1	1	2	1	0	0	1	4	
31835	2	6	1	5	0	2	0	0	6	
19551	3	4	1	6	0	2	0	0	1	

```
# from sklearn.preprocessing import StandardScaler

# # Standardize data to have mean=0 and sd =1
# sc_X = StandardScaler()
# X_train = sc_X.fit_transform(X_train)
# X_test = sc_X.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()  # print model _coefficient, _intercennp
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)

print(confusion_matrix(y_test, logpred))
```

```
[[10686   292]
 [  862   517]]
```

```
#X_train.columns
```

coef_ndarray of shape (1, n_features) or (n_classes, n_features) Coefficient of the features in the d

coef_ is of shape (1, n_features) when the given problem is binary. In particular, when multi_class='
1 (True) and -coef_ corresponds to outcome 0 (False).

```
logmodel.coef_
```

```
array([[ 0.01373506,  0.00906686,  0.03599591,  0.04435978, -0.05716133,
        -0.00387637, -0.00545929, -0.0844546 , -0.15811366,  0.07217424,
         1.30311734, -0.21477841,  0.31648587,  0.05623431, -0.23468372,
        -0.00599573, -0.0522925 , -0.00186006, -0.03405911, -0.02851235]])
```

intercept_ndarray of shape (1,) or (n_classes,) Intercept (a.k.a. bias) added to the decision function

If fit_intercept is set to False, the intercept is set to zero. intercept_ is of shape (1,) when the given
multi_class='multinomial', intercept_ corresponds to outcome 1 (True) and -intercept_ corresponds

```
logmodel.intercept_
```

```
array([0.00312328])
```

Evaluation

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_test, logpred, digits=4))
```

```
              precision    recall  f1-score   support

           0     0.9254    0.9734    0.9488     10978
           1     0.6391    0.3749    0.4726      1379

    accuracy                         0.9066     12357
   macro avg     0.7822    0.6742    0.7107     12357
weighted avg     0.8934    0.9066    0.8956     12357
```

M2: Categorical

dummy code recode - ordin

```
nk = pd.read_csv('https://github.com/kaopanboonyuen/Python-Data-Science/raw/mast
Convert dependent variable categorical to dummy
= pd.get_dummies(bank['y'], columns = ['y'], prefix = ['y'], drop_first = True)
nk.head()
```

	age	job	marital	education	default	housing	loan	contact	month	da
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	
1	57	services	married	high.school	unknown	no	no	telephone	may	
2	37	services	married	high.school	no	yes	no	telephone	may	
3	40	admin.	married	basic.6y	no	no	no	telephone	may	
4	56	services	married	high.school	no	no	yes	telephone	may	

```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

```
# Bank client data Analysis and Categorical Treatment

bank_client = bank.iloc[: , 0:7]
bank_client.head()
```

	age	job	marital	education	default	housing	loan
0	56	housemaid	married	basic.4y	no	no	no
1	57	services	married	high.school	unknown	no	no
2	37	services	married	high.school	no	yes	no
3	40	admin.	married	basic.6y	no	no	no
4	56	services	married	high.school	no	no	yes

```
bank_client = pd.get_dummies(data = bank_client, columns = ['job'] , prefix = ['

bank_client = pd.get_dummies(data = bank_client, columns = ['marital'] , prefix

bank_client = pd.get_dummies(data = bank_client, columns = ['education'], prefix

bank_client = pd.get_dummies(data = bank_client, columns = ['default'] , prefix

bank_client = pd.get_dummies(data = bank_client, columns = ['housing'] , prefix

bank_client = pd.get_dummies(data = bank_client, columns = ['loan'] , prefix = [
```

```
# Related with the last contact of the current campaign
# Treat categorical, see those values group continuous variables if necessary

bank_related = bank.iloc[: , 7:11]
bank_related.head()
```

	contact	month	day_of_week	duration
0	telephone	may	mon	261
1	telephone	may	mon	149
2	telephone	may	mon	226
3	telephone	may	mon	151
4	telephone	may	mon	307

```
bank_related.isnull().any()
```

```
contact        False
month          False
day_of_week    False
duration       False
dtype: bool
```

```
bank_related = pd.get_dummies(data = bank_related, prefix = ['contact'] , column

bank_related = pd.get_dummies(data = bank_related, prefix = ['month'] , columns

bank_related = pd.get_dummies(data = bank_related, prefix = ['day_of_week'], col
```

```
def duration(data):

    data.loc[data['duration'] <= 102, 'duration'] = 1
    data.loc[(data['duration'] > 102) & (data['duration'] <= 180)  , 'duration']
    data.loc[(data['duration'] > 180) & (data['duration'] <= 319)  , 'duration']
    data.loc[(data['duration'] > 319) & (data['duration'] <= 644.5), 'duration']
    data.loc[data['duration']  > 644.5, 'duration'] = 5

    return data
duration(bank_related);
```

```
bank_se = bank.loc[: , ['emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euri
bank_se.head()
```

```
emp.var.rate, cons.price.idx, cons.conf.idx, euribor3m, nr.employed
```

```
# Other attributes

bank_o = bank.loc[: , ['campaign', 'pdays','previous', 'poutcome']]
bank_o.head()
```

	campaign	pdays	previous	poutcome
0	1	999	0	nonexistent
1	1	999	0	nonexistent
2	1	999	0	nonexistent
3	1	999	0	nonexistent
4	1	999	0	nonexistent

```
bank_o['poutcome'].unique()
```

```
array(['nonexistent', 'failure', 'success'], dtype=object)
```

```
bank_o['poutcome'].replace(['nonexistent', 'failure', 'success'], [1,2,3], inpla
```

```
bank_final = bank_final[['age', 'job', 'marital', 'education', 'default', 'housi
                         'contact', 'month', 'day_of_week', 'duration', 'emp.var.rat
                         'cons.conf.idx', 'euribor3m', 'nr.employed', 'campaign', 'p
bank_final.shape
```

```
(41188, 20)
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

X_train, X_test, y_train, y_test = train_test_split(bank_final, y, test_size = 0
```

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
logpred = logmodel.predict(X_test)
```

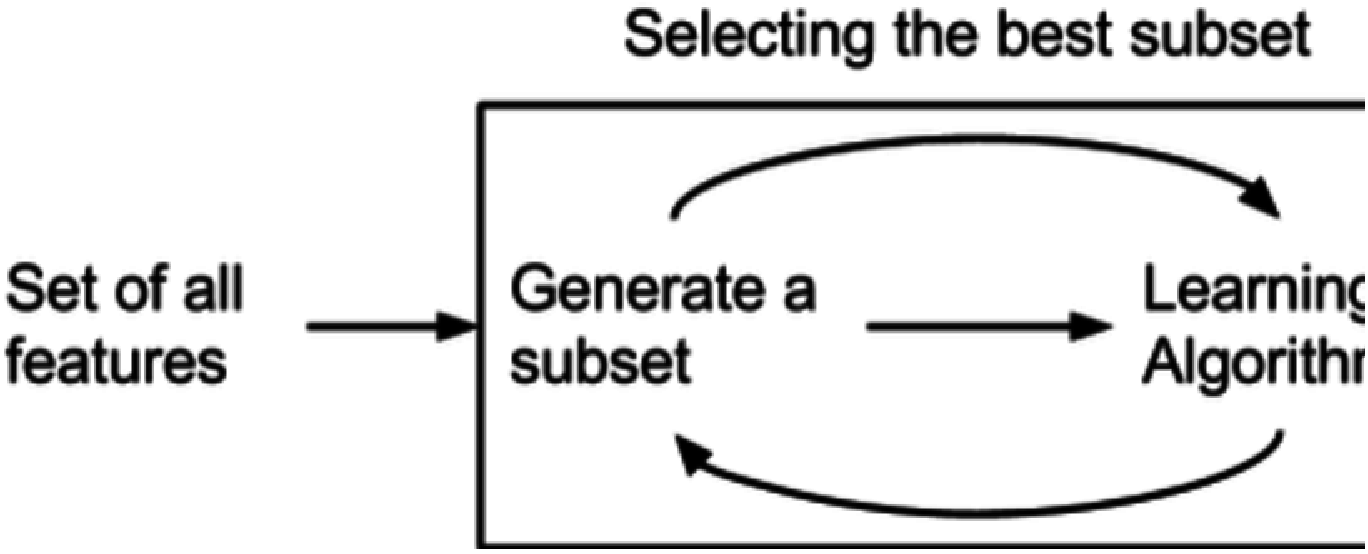
```
print(confusion_matrix(y_test, logpred))
print("accuracy_score = ", round(accuracy_score(y_test, logpred),2)*100)
```

```
[[10686   292]
 [  862   517]]
accuracy_score =  91.0
```

M3: Model Selection

Step Forward Feature Selection: A Practical Example in Python

Reference: https://www.kdnuggets.com/2018/06/step-forward-feature-selection-python.html




```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
```

```
# Build RF classifier to use in feature selection
clf = LogisticRegression()

# Build step forward feature selection
sfs1 = sfs(clf,
           k_features=5,
           forward=True,
           floating=False,
           verbose=2,
           scoring='accuracy',
           cv=5)

# Perform SFFS
sfs1 = sfs1.fit(bank_final, y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
```

```
# Which features?
feat_cols = list(sfs1.k_feature_idx_)
print(feat_cols)
```

```
[6, 10, 15, 16, 17]
```

```
[Parallel(n_jobs=1)]: Done  18 out of  18 | elapsed:   14.8s finished

[2020-02-26 03:23:26] Features: 3/5 -- score: 0.9026172267361169[Parallel(n_jol
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.1s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  17 out of  17 | elapsed:   24.1s finished

[2020-02-26 03:23:50] Features: 4/5 -- score: 0.9026172267361169[Parallel(n_jol
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.2s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:   23.1s finished

[2020-02-26 03:24:13] Features: 5/5 -- score: 0.9001891582645859
```