1 Executive Summary

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. In other words: regularization can be used to train models that generalize better on unseen data, by preventing the algorithm from overfitting the training dataset.

We have some techniques of regularization:

1) Ridge regularization (L2),
2) Lasso regularization (L1),
3) Elastic Net.

L2 generally leads to smaller coefficients, L1 results in sparse coefficient vectors with just a few higher value coefficients.

The need for regularization
Regularization can sometimes leads to better model performance. Regularization can be used to avoid overfitting, a more generic model may be preferred over a very specific one

The foundations of a regularizer
Regularizers are attached to the loss values of a machine learning model, and they are thus included in the optimization step. Combining the original loss value with the regularization component,the model will become simpler with likely losing not much of their predictive abilities.

2 Introduction

I plan to build and train l1-penalized,l2-penalized and Elastic Net logistic regression models for a binary classification problem derived from a dataset.
I aim to investigate whether regulirization techniques can be helpful in improving the precitive power of a logistic regression model. Later using Recursive Feature Elimination we test the importance of key identified features.

3 Loading and Exploring Data

3.1 Loading libraries

In [248]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas import Series, DataFrame
from math import sqrt
import warnings
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
```

3.2 Loading Data

In [249]:
```
data = pd.read_csv('Downloads/bank.csv', delimiter=";")
```

3.3 Data size and structure

In [250]:
```
data.shape
```

Out[250]: (4521, 17)

The dataframe consists of 17 predictors and our response variable is "y".

In [251]:
```
data.head()
```

Out[251]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	pout
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unl
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unl
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unl

In [252]:
```
data.describe(include=[np.object]).transpose() \
    .drop("count", axis=1)
```

Out[252]:

	unique	top	freq
job	12	management	969
marital	3	married	2797
education	4	secondary	2306
default	2	no	4445
housing	2	yes	2559
loan	2	no	3830
contact	3	cellular	2896
month	12	may	1398
poutcome	4	unknown	3705
y	2	no	4000

In [253]:
```
np.unique(data.month)
```

Out[253]:
```
array(['apr', 'aug', 'dec', 'feb', 'jan', 'jul', 'jun', 'mar', 'may',
       'nov', 'oct', 'sep'], dtype=object)
```

In [254]:
```
mp = {'apr':4, 'aug':8, 'dec':12, 'feb':2, 'jan':1, 'jul':7, 'jun':6, 'mar':3, 'may':5,
       'nov':11, 'oct':10, 'sep':9}
data['month'] = data['month'].map(mp)
```

In [255]:
```
data.info()
```

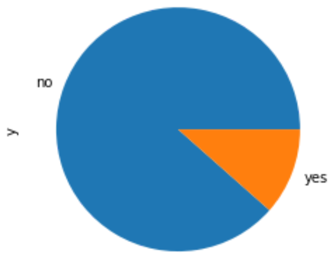
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4521 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4521 non-null   int64 
 1   job        4521 non-null   object
 2   marital    4521 non-null   object
 3   education  4521 non-null   object
 4   default    4521 non-null   object
 5   balance    4521 non-null   int64 
 6   housing    4521 non-null   object
 7   loan       4521 non-null   object
 8   contact    4521 non-null   object
 9   day        4521 non-null   int64 
 10  month      4521 non-null   int64 
 11  duration   4521 non-null   int64 
 12  campaign   4521 non-null   int64 
 13  pdays      4521 non-null   int64 
 14  previous   4521 non-null   int64 
 15  poutcome   4521 non-null   object
 16  y          4521 non-null   object
dtypes: int64(8), object(9)
memory usage: 600.6+ KB
```

4 EDA

4.1 The response variable

In [256]:
```
data['y'].value_counts().plot(kind='pie', figsize=(4,4))
data['y'].value_counts()
```

Out[256]:
```
no     4000
yes     521
Name: y, dtype: int64
```



I am dealing with an imbalanced dataset which should be in cosideration while I am building a model on top of that.

5 Statistics, Missing data, label encoding

In [257]:
```
data.describe()
```

Out[257]:

	age	balance	day	month	duration	campaign	pdays	previous
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	41.170095	1422.657819	15.915284	6.166777	263.961292	2.793630	39.766645	0.542579
std	10.576211	3009.638142	8.247667	2.378380	259.856633	3.109807	100.121124	1.693562
min	19.000000	-3313.000000	1.000000	1.000000	4.000000	1.000000	-1.000000	0.000000
25%	33.000000	69.000000	9.000000	5.000000	104.000000	1.000000	-1.000000	0.000000
50%	39.000000	444.000000	16.000000	6.000000	185.000000	2.000000	-1.000000	0.000000
75%	49.000000	1480.000000	21.000000	8.000000	329.000000	3.000000	-1.000000	0.000000
max	87.000000	71188.000000	31.000000	12.000000	3025.000000	50.000000	871.000000	25.000000

In [258]:
```
data.isna().sum()
```

Out[258]:
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

5.1 Scaling and Transformation

In [259]:
```
numerical_feature = list(data.select_dtypes(include=['int64']).columns)
for feature in numerical_feature:
    #numpy.newaxis represents a new axis in numpy array
    fetched_val = data[feature].values[:, np.newaxis]
    scaler = MinMaxScaler().fit(fetched_val)
    data[feature] = scaler.transform(fetched_val)
```

In [260]:
```
mylist = list(data.select_dtypes(include=['object']).columns)
```

In [261]:
```
#mylist = list(dataset.select_dtypes(include=['object']).columns)
dummies = pd.get_dummies(data[mylist], prefix= mylist, drop_first=True)
data_trans = pd.concat([data,dummies], axis =1 )
data_transans_copy= data_trans.copy(deep=True )
```

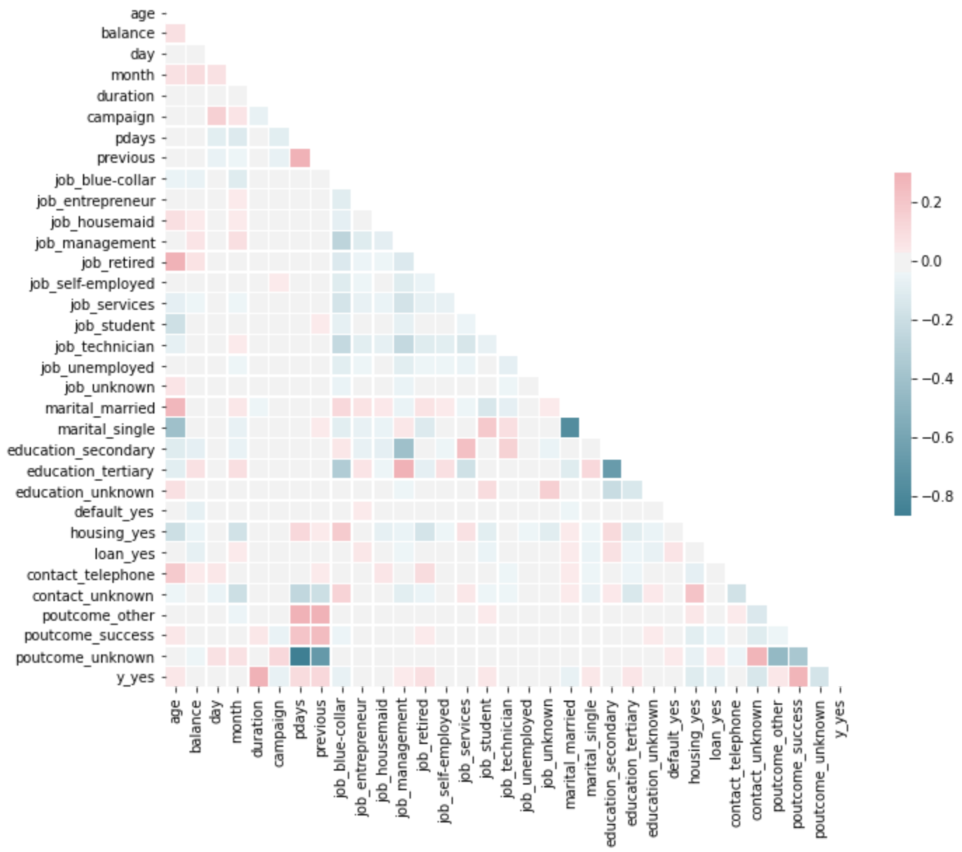
In [262]:
```
y_trans=data_trans.y_yes
data_trans.drop(mylist, axis=1, inplace = True)
```

5.2 The Correlations with Response Variable

y_yes

In [263]:
```
corr = data_trans.corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))
fif, ax = plt.subplots(figsize=(11, 9))
cmap = sns.diverging_palette(220, 10, as_cmap=True)
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Out[263]: <AxesSubplot:>



In [264]:
```
data_trans.drop(['y_yes'], axis=1, inplace = True)
```

5.3 Handling Imbalanced Data

5.3.1 SMOTE

Up-sample Minority Class

In [265]:
```
from sklearn.model_selection import (train_test_split, cross_val_score, RepeatedKFold)
from sklearn.ensemble import GradientBoostingRegressor
from sklearn import metrics
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE
```

In [266]:
```
upsampled = SMOTE(random_state=101)
X_train, X_test, y_train, y_test = train_test_split(data_trans, y_trans, test_size=0.3, random_state=101)
columns = X_train.columns
```

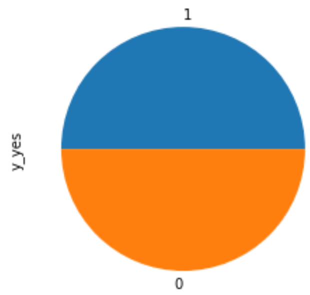
In [267]:
```
upsampled_data_X,upsampled_data_y=upsampled.fit_sample(X_train, y_train)
```

In [268]:
```
upsampled_data_X = pd.DataFrame(data=upsampled_data_X,columns=columns )
upsampled_data_y= pd.DataFrame(data=upsampled_data_y,columns=['y_yes'])
```

In [269]:
```
# a copy of dataset for further use
upsampled_whole= upsampled_data_X.copy(deep=True)
upsampled_whole['y_yes']=upsampled_data_y['y_yes']
```

In [270]:
```
upsampled_data_y['y_yes'].value_counts().plot(kind='pie', figsize=(4,4))
upsampled_data_y['y_yes'].value_counts()
```

Out[270]:
```
1    2798
0    2798
Name: y_yes, dtype: int64
```



6 Building the Model

In [271]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
Model = LogisticRegression()
```

I use cross validation with 5 folds (cv = 5).

I begin by splitting the dataset into five groups or folds. Then I hold out the first fold as a test set, fit out model on the remaining four folds, predict on the test set and compute the metric of interest. Next, I hold out the second fold as out test set, fit on the remaining data, predict on the test set and compute the metric of interest.

In [272]:
```
scores = cross_validate(Model,upsampled_data_X,upsampled_data_y, cv=5,
                        scoring=('r2', 'neg_mean_squared_error'),
                          return_train_score=True)
```

In [273]:
```
np.mean(scores['test_neg_mean_squared_error'])
```

Out[273]:
```
-0.15259957870547683
```

In [274]:
```
np.mean(scores['test_r2'])
```

Out[274]:
```
0.3896013289036545
```

6.1 Predicting the Model and Fitting

In [275]:
```
Model.fit(upsampled_data_X,upsampled_data_y)
```

Out[275]:
```
LogisticRegression()
```

In [276]:
```
prediction = Model.predict(X_test)
```

RMSE answers the question: “How similar(haw far), on average, are the predicted values and real target values in our model?” When the RMSE number is zero, we are having the perfect predictions every time. If the number goes up, we are getting worse.

In [277]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [278]:
```
rmse_val = rmse(np.array(prediction), np.array(y_test))
results = pd.DataFrame()
results["Method"] = ["Logistic Regression-All features"]
results["RMSE"] = rmse_val
results
```

Out[278]:

	Method	RMSE
0	Logistic Regression-All features	0.417911

6.2 Model Score on Training and Test Set

In [279]:
```
Model.score(upsampled_data_X,upsampled_data_y)
```

Out[279]:
```
0.8509649749821301
```

In [280]:
```
Model.score(X_test,y_test)
```

Out[280]:
```
0.8253500368459837
```

In [281]:
```
from sklearn import metrics
print (metrics.r2_score(prediction, y_test))
```

```
-0.014219489120151474
```

6.3 Intercept and Coeficcients

In [282]:
```
print (Model.coef_)
```

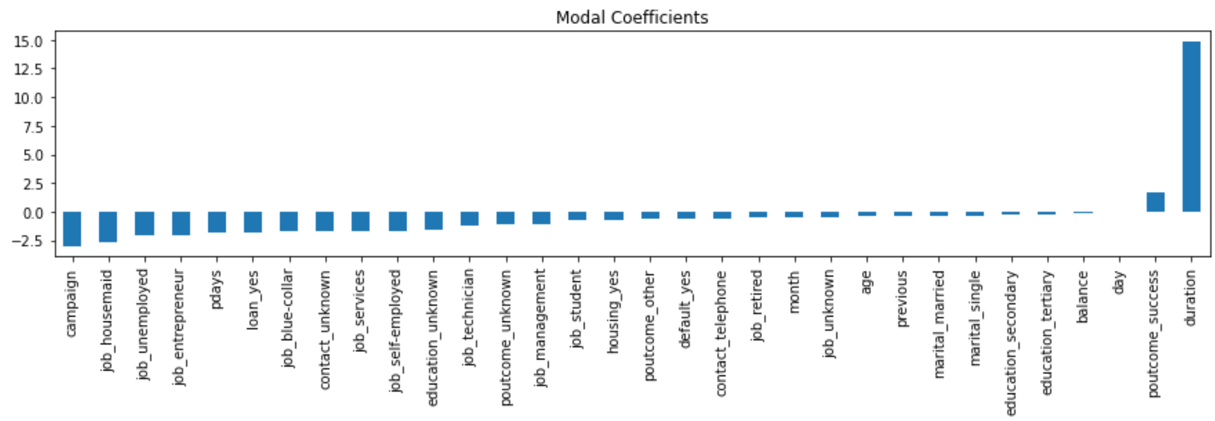
```
[[-0.40549632 -0.13742257 -0.07418233 -0.48926084 14.88395048 -3.04065239
  -1.84513947 -0.36099869 -1.76965137 -2.07579939 -2.73675888 -1.04696213
  -0.55829219 -1.64951084 -1.66689362 -0.72690254 -1.27562288 -2.07937564
  -0.45410511 -0.35603645 -0.34881275 -0.31720809 -0.19850852 -1.63439028
  -0.63572027 -0.68504409 -1.83211975 -0.61006792 -1.67921958 -0.66709247
   1.72829143 -1.05722564]]
```

In [283]:
```
print (Model.intercept_)
```

```
[2.01510816]
```

In [284]:
```
plt.figure(figsize=(15,3))
predictors = upsampled_data_X.columns
coef = Series(Model.coef_[0], predictors).sort_values()
coef.plot(kind='bar', title='Modal Coefficients')
```

Out[284]:
```
<AxesSubplot:title={'center':'Modal Coefficients'}>
```



6.4 Ridge Regularization (L2)

In [285]:
```
from sklearn.model_selection import GridSearchCV
l2 = LogisticRegression(penalty='l2')
parameters = {'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750, 1000]}
RidgeReg = GridSearchCV(l2, parameters, scoring='neg_mean_squared_error', cv = 5)
RidgeReg.fit(upsampled_data_X,upsampled_data_y)
```

Out[285]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750,
                               1000]},
             scoring='neg_mean_squared_error')
```

In [286]:
```
RidgeReg.best_params_
```

Out[286]:
```
{'C': 5}
```

In [287]:
```
l2 = LogisticRegression(penalty='l2', C=5)
```

In [288]:
```
l2.fit(upsampled_data_X,upsampled_data_y)
```

Out[288]:
```
LogisticRegression(C=5)
```

In [289]:
```
prediction = l2.predict(X_test)
```

In [290]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [291]:
```
rmse_val = rmse(np.array(prediction), np.array(y_test))
results.loc[1] = ["L2", rmse_val]
results
```

Out[291]:

	Method	RMSE
0	Logistic Regression-All features	0.417911
1	L2	0.417911

6.5 Model Score on Training and Test Set

In [292]:
```
l2.score(upsampled_data_X,upsampled_data_y)
```

Out[292]:
```
0.8522158684774839
```

In [293]:
```
l2.score(X_test,y_test)
```

Out[293]:
```
0.8253500368459837
```

In [294]:
```
from sklearn import metrics
print (metrics.r2_score(prediction, y_test))
```

```
-0.009412761683562865
```

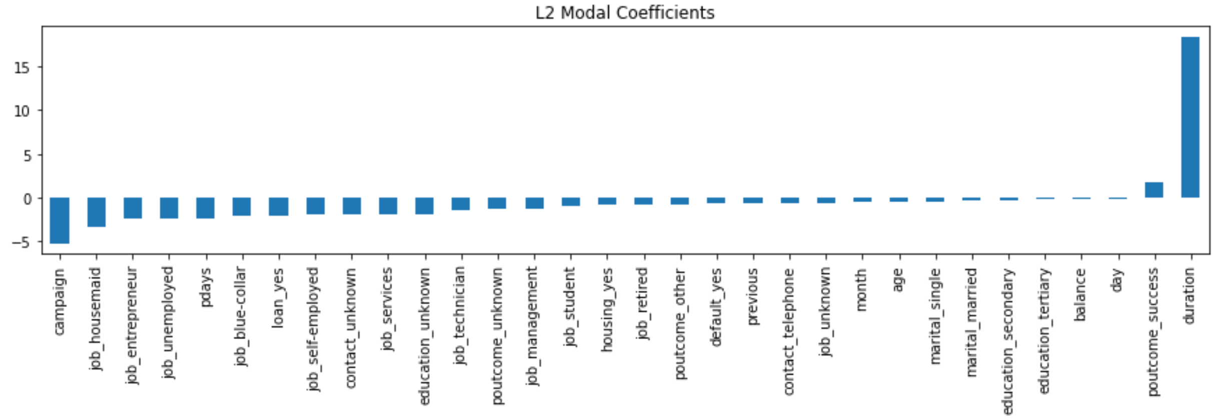
6.6 Intercept and Coeficcients

In [295]:
```
print (l2.coef_)
```

```
[[-0.45205133 -0.07584463 -0.07437816 -0.53364114 18.40858177 -5.25143358
  -2.38857215 -0.60215754 -2.03405547 -2.45409381 -3.39704461 -1.19046268
  -0.73686195 -1.8984188  -1.85487006 -0.87666316 -1.45495758 -2.43654386
  -0.57780965 -0.37731889 -0.40224559 -0.36088976 -0.2184981  -1.81928438
  -0.69686854 -0.7662286  -2.02508912 -0.59235898 -1.89762838 -0.72204638
   1.78307625 -1.22095013]]
```

In [296]:
```python
plt.figure(figsize=(15,3))
predictors = upsampled_data_X.columns
coef = Series(l2.coef_[0], predictors).sort_values()
coef.plot(kind='bar', title='L2 Modal Coefficients')
```

Out[296]:
```
<AxesSubplot:title={'center':'L2 Modal Coefficients'}>
```



6.7 Lasso Regularization (L1)

In [297]:
```python
from sklearn.model_selection import GridSearchCV
l1 = LogisticRegression(penalty='l1', solver='liblinear')
parameters = {'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750, 1000]}
lassoReg = GridSearchCV(l1, parameters, scoring='neg_mean_squared_error', cv = 5)
lassoReg.fit(upsampled_data_X,upsampled_data_y)
```

Out[297]:
```
GridSearchCV(cv=5,
             estimator=LogisticRegression(penalty='l1', solver='liblinear'),
             param_grid={'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750,
                               1000]},
             scoring='neg_mean_squared_error')
```

The usefulness of L1 is that it can push feature coefficients to 0, creating a method for feature selection. In the code below we run a logistic regression with a L1 penalty four times, each time decreasing the value of C. We should expect that as C decreases, more coefficients become 0.

In [298]:
```python
lassoReg.best_params_
```

Out[298]:
```
{'C': 1}
```

In [299]:
```python
l1 = LogisticRegression(penalty='l1', solver='liblinear', C=1)
```

In [300]:
```python
l1.fit(upsampled_data_X,upsampled_data_y)
```

Out[300]:
```
LogisticRegression(C=1, penalty='l1', solver='liblinear')
```

In [301]:
```python
prediction = l1.predict(X_test)
```

RMSE answers the question: “How similar(haw far), on average, are the predicted values and real target values in our model?” When the RMSE number is zero, we are having the perfect predictions every time. If the number goes up, we are getting worse.

In [302]:
```python
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [303]:
```
rmse_val = rmse(np.array(prediction), np.array(y_test))
results.loc[2] = ["L1", rmse_val]
results
```

Out[303]:

	Method	RMSE
0	Logistic Regression-All features	0.417911
1	L2	0.417911
2	L1	0.415258

6.8 Model Score on Training and Test Set

In [304]:
```
l1.score(upsampled_data_X,upsampled_data_y)
```

Out[304]: 0.8543602573266619

In [305]:
```
l1.score(X_test,y_test)
```

Out[305]: 0.8275607958732498

In [306]:
```
from sklearn import metrics
print (metrics.r2_score(prediction, y_test))
```

```
-0.003780718336483968
```

6.9 Intercept and Coeficcients

In [307]:
```
print (l1.coef_)
```

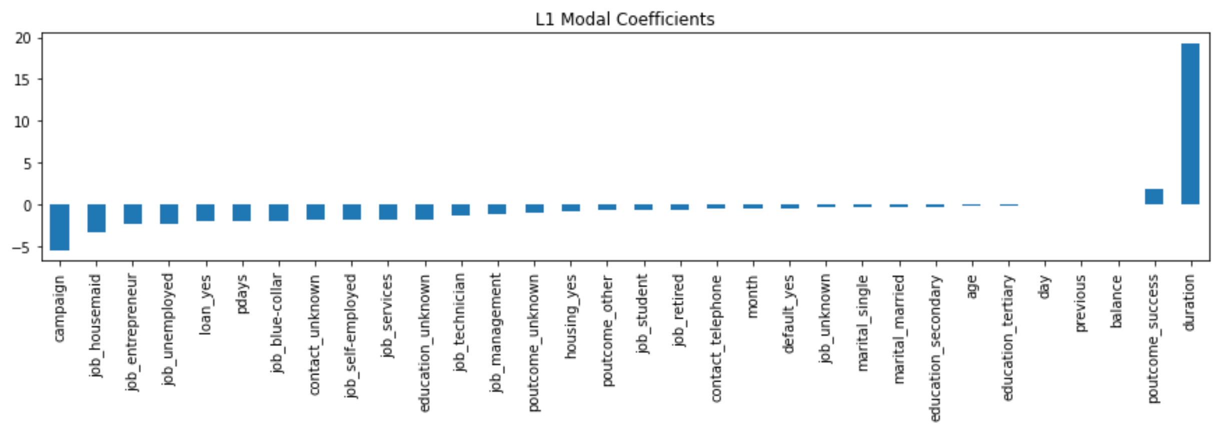
```
[[-0.23067393  0.         -0.03458685 -0.49404362 19.29764681 -5.53002013
  -2.01644313  0.         -1.96807222 -2.38004179 -3.29796022 -1.11884191
  -0.68147325 -1.79562313 -1.76331565 -0.68814968 -1.38720738 -2.32454814
  -0.38390728 -0.32790775 -0.33557768 -0.27906188 -0.13212104 -1.76187387
  -0.4703242  -0.75780762 -2.03921589 -0.56562777 -1.92069985 -0.70249448
   1.80278267 -1.05606862]]
```

In [308]:
```
print (l1.intercept_)
```

```
[1.60983556]
```

In [309]:
```
plt.figure(figsize=(15,3))
predictors = upsampled_data_X.columns
coef = Series(l1.coef_[0], predictors).sort_values()
coef.plot(kind='bar', title='L1 Modal Coefficients')
```

Out[309]: <AxesSubplot:title={'center':'L1 Modal Coefficients'}>



6.10 Elastic Net

In [310]:
```
from sklearn.linear_model import ElasticNet
```

In [311]:
```
from sklearn.model_selection import GridSearchCV
EL= LogisticRegression(penalty='elasticnet', solver='saga')
parameters = {'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750, 1000],'l1_ratio':np.arange(0.0, 1.0, 0.1
ELReg = GridSearchCV(EL, parameters, scoring='neg_mean_squared_error', cv = 5)
ELReg.fit(upsampled_data_X,upsampled_data_y)
```

Out[311]:
```
GridSearchCV(cv=5,
             estimator=LogisticRegression(penalty='elasticnet', solver='saga'),
             param_grid={'C': [1e-15, 1, 5, 10, 20, 25, 50, 60, 100, 500, 750,
                               1000],
                         'l1_ratio': array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])},
             scoring='neg_mean_squared_error')
```

The Elastic-Net mixing parameter, with 0 <= l1_ratio <= 1. Only used if penalty='elasticnet'. Setting l1_ratio=0 is equivalent to using penalty='l2', while setting l1_ratio=1 is equivalent to using penalty='l1'. For 0 < l1_ratio <1, the penalty is a combination of L1 and L2.

In [313]:
```
ELReg.best_params_
```

Out[313]:
```
{'C': 1, 'l1_ratio': 0.7000000000000001}
```

In [314]:
```
EL= LogisticRegression(penalty='elasticnet', solver='saga', C=1, l1_ratio=0.7)
```

In [315]:
```
EL.fit(upsampled_data_X,upsampled_data_y)
```

Out[315]:
```
LogisticRegression(C=1, l1_ratio=0.7, penalty='elasticnet', solver='saga')
```

In [316]:
```
prediction = EL.predict(X_test)
```

RMSE answers the question: “How similar(haw far), on average, are the predicted values and real target values in our model?” When the RMSE number is zero, we are having the perfect predictions every time. If the number goes up, we are getting worse.

In [317]:
```
def rmse(predictions, targets):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

In [318]:
```
rmse_val = rmse(np.array(prediction), np.array(y_test))
results.loc[3] = ["EL", rmse_val]
results
```

Out[318]:

	Method	RMSE
0	Logistic Regression-All features	0.417911
1	L2	0.417911
2	L1	0.415258
3	EL	0.417029

6.11 Model Score on Training and Test Set

In [319]:
```
EL.score(upsampled_data_X,upsampled_data_y)
```

Out[319]:
```
0.8543602573266619
```

In [320]:
```
EL.score(X_test,y_test)
```

Out[320]:
```
0.8260869565217391
```

In [321]:
```
from sklearn import metrics
print (metrics.r2_score(prediction, y_test))
```

```
-0.0027868061948510547
```

6.12 Intercept and Coeficcients

In [322]:
```
print (EL.coef_)
```

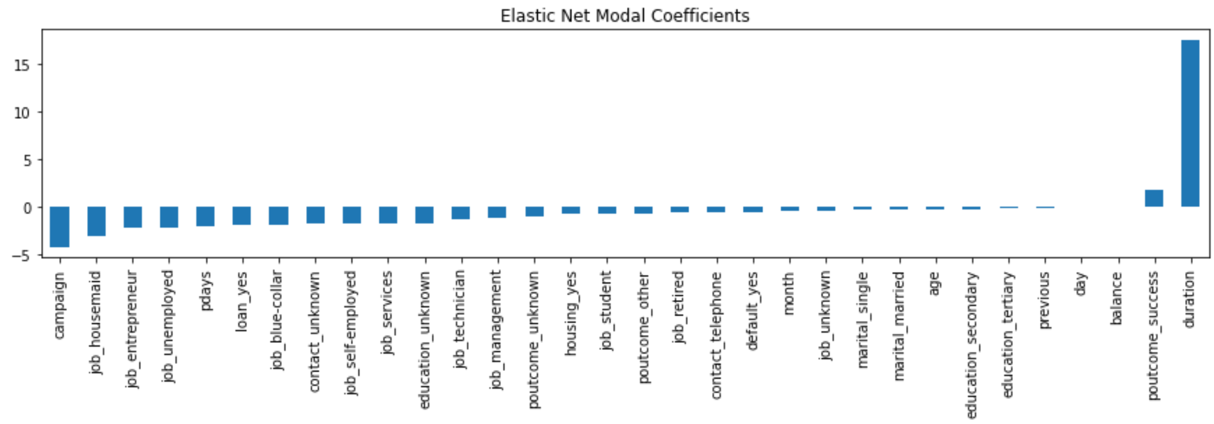
```
[[-0.33611074  0.         -0.06146992 -0.50287073 17.48869561 -4.2459761
  -2.00446856 -0.09033512 -1.89441711 -2.26270933 -3.0776616  -1.09714181
  -0.63074306 -1.74455802 -1.73188536 -0.71699117 -1.34847358 -2.23122415
  -0.40842718 -0.34543074 -0.3513111  -0.30256876 -0.1670773  -1.71351517
  -0.53348868 -0.72995943 -1.95204874 -0.58249531 -1.81760539 -0.69849034
   1.76561924 -1.07164322]]
```

In [323]:
```
print (EL.intercept_)
```

```
[1.8197958]
```

In [324]:
```
plt.figure(figsize=(15,3))
predictors = upsampled_data_X.columns
coef = Series(EL.coef_[0], predictors).sort_values()
coef.plot(kind='bar', title='Elastic Net Modal Coefficients')
```

Out[324]: <AxesSubplot:title={'center':'Elastic Net Modal Coefficients'}>



7 Building the Model with Selected Features

7.1 Recursive Feature Elimination

In [325]:
```
from sklearn.feature_selection import RFE
import statsmodels.api as sm
```

In [326]:
```
final_vars=upsampled_data_X.columns.values.tolist()
```

In [327]:
```
Model_FE = LogisticRegression()
rfe = RFE(Model_FE, n_features_to_select=10)
rfe = rfe.fit(upsampled_data_X,upsampled_data_y)
print(rfe.support_)
print(rfe.ranking_)
# Feature ranking = 1 means that this feature has been recognized as an important feature
# (similar to "true" in supoort)
```

```
[False False False False  True  True False False  True  True  True False
 False False False False False  True False False False False False  True
 False False  True False  True False  True False]
[19 22 23 14  1  1  6 16  1  1  1  5 13  3  2 12  4  1 15 17 18 20 21  1
  8 11  1 10  1  9  1  7]
```

In [328]:
```
# Let us make a list of indices
selected_list = [4,5,8,9,10,17,23,26,28,30]
data_selected_list = upsampled_data_X.iloc[:, selected_list]
```

7.1.1 Building a Model using sm.Logit

In [329]:
```
logit_model=sm.Logit(upsampled_data_y,data_selected_list)
result=logit_model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: inf
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                  y_yes   No. Observations:                 5596
Model:                          Logit   Df Residuals:                     5586
Method:                           MLE   Df Model:                            9
Date:                Sun, 25 Oct 2020   Pseudo R-squ.:                     inf
Time:                        22:28:48   Log-Likelihood:                   -inf
converged:                       True   LL-Null:                        0.0000
Covariance Type:            nonrobust   LLR p-value:                     1.000
=====================================================================================
                        coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------
duration             12.6476      0.390     32.432      0.000      11.883      13.412
campaign            -13.0745      0.949    -13.782      0.000     -14.934     -11.215
job_blue-collar      -1.4162      0.100    -14.220      0.000      -1.611      -1.221
job_entrepreneur     -1.7541      0.243     -7.228      0.000      -2.230      -1.279
job_housemaid        -2.1721      0.292     -7.446      0.000      -2.744      -1.600
job_unemployed       -1.8032      0.257     -7.006      0.000      -2.308      -1.299
education_unknown    -1.7146      0.226     -7.573      0.000      -2.158      -1.271
loan_yes             -2.1782      0.149    -14.619      0.000      -2.470      -1.886
contact_unknown      -2.2300      0.105    -21.294      0.000      -2.435      -2.025
poutcome_success      2.0131      0.209      9.651      0.000       1.604       2.422
=====================================================================================
```

Let us check if the Z values are calculated correctly in this table.

In [330]:
```
cov = result.cov_params()
```

Let us see ghow the Z values are calculated

In [331]:
```
std_err = np.sqrt(np.diag(cov))
```

In [332]:
```
z_values = result.params / std_err
z_values
```

Out[332]:
```
duration             32.432486
campaign            -13.781985
job_blue-collar     -14.219687
job_entrepreneur     -7.228422
job_housemaid        -7.446085
job_unemployed       -7.005831
education_unknown    -7.572706
loan_yes            -14.618947
contact_unknown     -21.294138
poutcome_success      9.651063
dtype: float64
```

8 Conclusion

In this case study I observed that some of the predictors might not be associated with our responce varaiable, therefore, their coefficients better be ommited from the model. I also showed the importance of parameter tuning for having an optimized version of the model. In this dataset regularization did not imporove the predictive power of our model on the test data. However, I should mention that the # of Features was not big here.