DIRECT MARKETING CAMPAIGN RESPONSE PREDICTION

A Python modeling project using the data of KDD Cup 1998

Joe Shan

Sep 2017

Objective

- The objective of KDD Cup 98 was to optimize the profit of direct marketing.
- Profit prediction is a combination of response prediction and individual profit prediction.
- This project only focus on response prediction. Meanwhile, this project is also to show the capability of Python on statistical modeling.

The Campaign of PVA

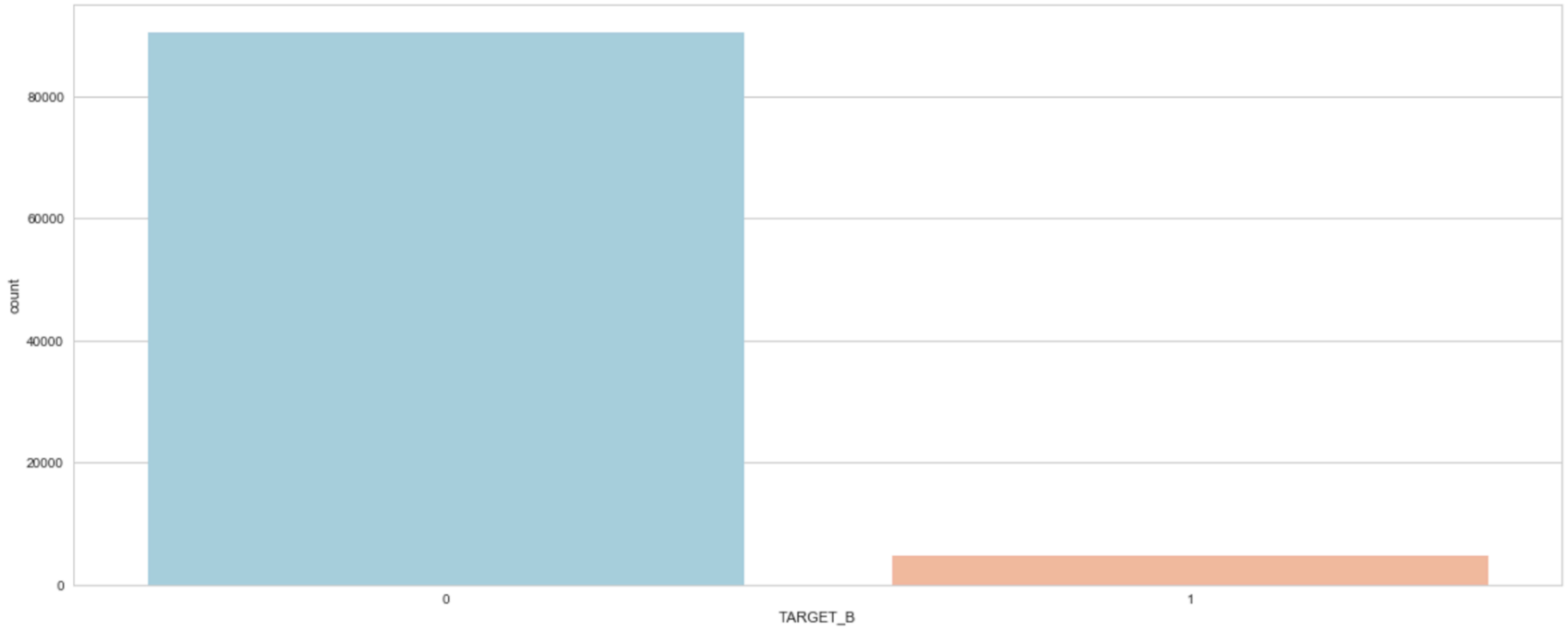
- Paralyzed Veterans of America (PVA) is a not-for-profit organization that provides programs and services for US veterans with spinal cord injuries or disease.

- One of PVA's fund raising appeals was dropped in June 1997 to a total of 3.5 million PVA donors. It included a gift "premium" of personalized name & address labels plus an assortment of 10 note cards and envelopes. All of the donors who received this mailing were acquired by PVA through premium-oriented appeals like this.

The Data Set

- The analysis data set includes:
  - A subset of the 3.5 million donors sent this appeal
  - A flag to indicate respondents to the appeal and the dollar amount of their donation
  - PVA promotion and giving history
  - Overlay demographics, including a mix of household and area level data.
- Train set (from cup98LRN.txt)
  - 76329 rows, 481 columns (80%)
- Test set (from cup98LRN.txt)
  - 19083 rows, 481 columns (20%)
- Column TARGET_B
  - Target Variable: Binary Indicator for Response to the 97NK Mailing

Dependent Variable on Whole Set - 1




```
0    90569
1     4843
Name: TARGET_B, dtype: int64
```

Dependent Variable on Whole Set - 2

- TARGET_B is binary.
  - Logistic Model is valid.
  - Given hundreds of variables, Logistic Model is easier to interpret and more intuitive than decision tree.

- The response rate is about 5%.
  - Unbalanced
  - Traditional: Resampling
  - In this project: balanced class weight

The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`.

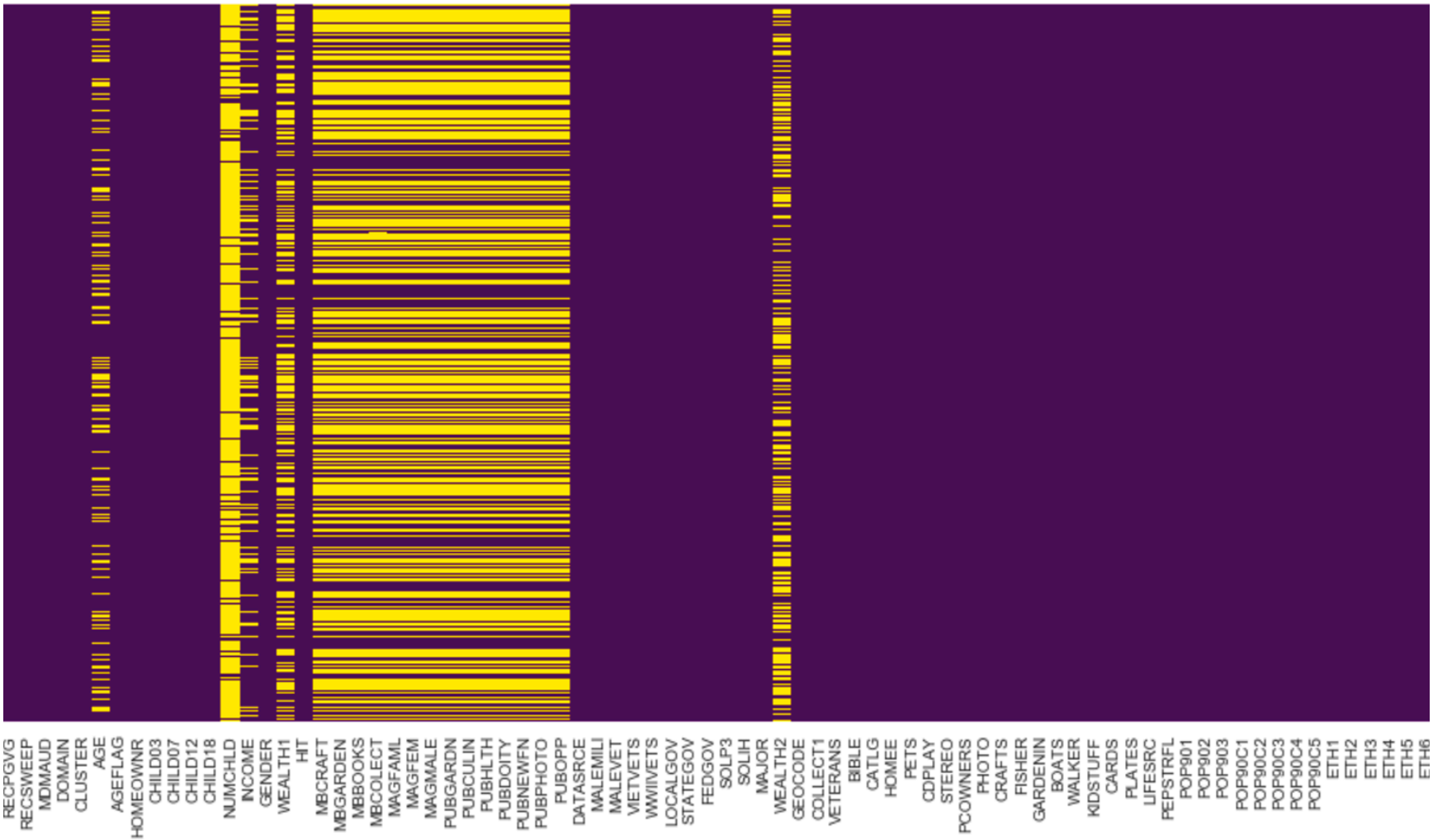
Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

Logistic Model and Python

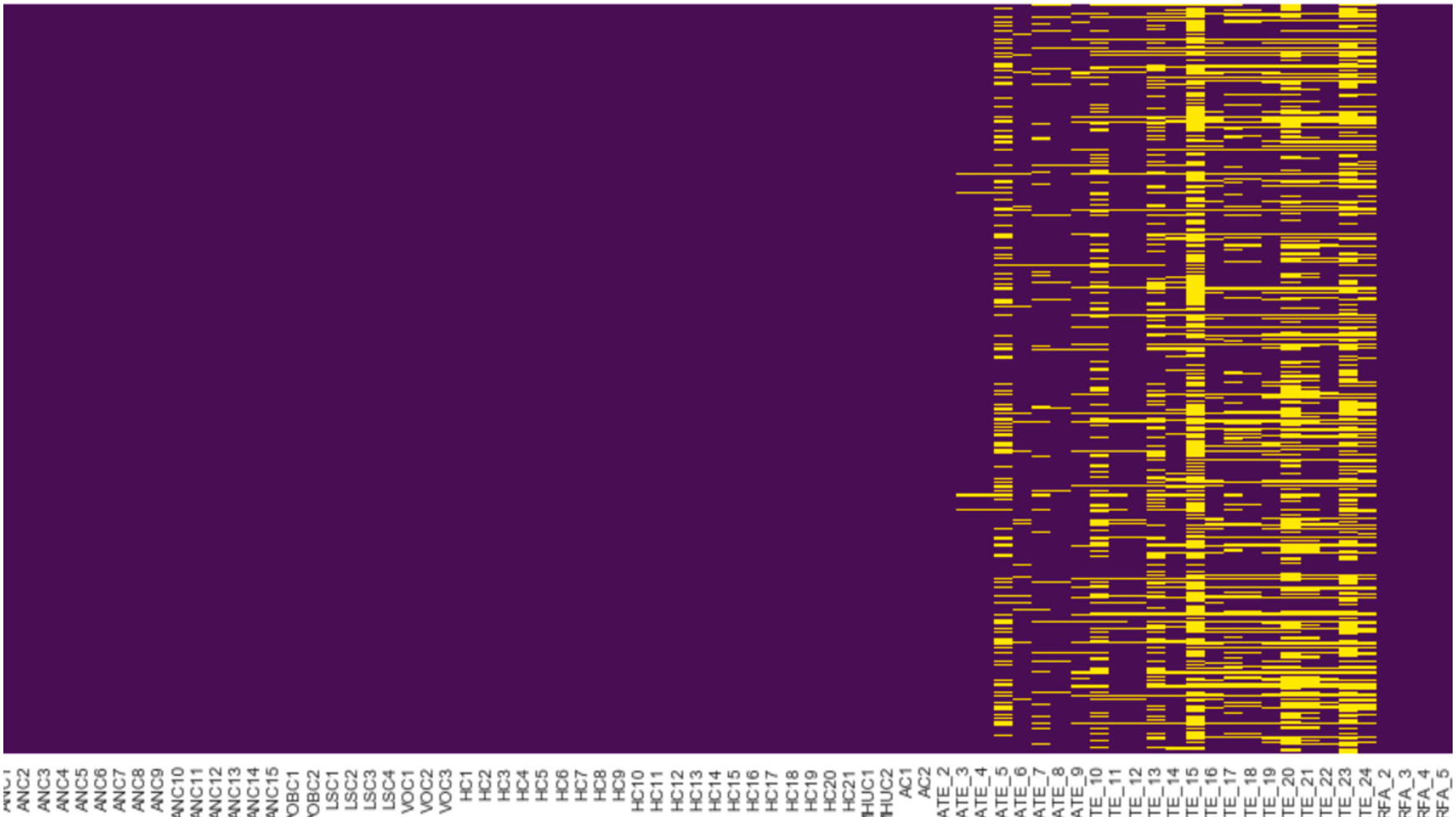
- Python – Why
  - Easy to develop and more transferable
  - Will be more and more powerful and popular
  - Able to realize some amazing functions easily
    (e.g. Class_Weight = 'balanced', REFCV, RFE)

- Python – How
  - Tool: Jupyter Notebook (Anaconda)
  - Library: scikit-learn
  - Some self-created functions (e.g. partial correlation matrix)
  - Code: My github repositories

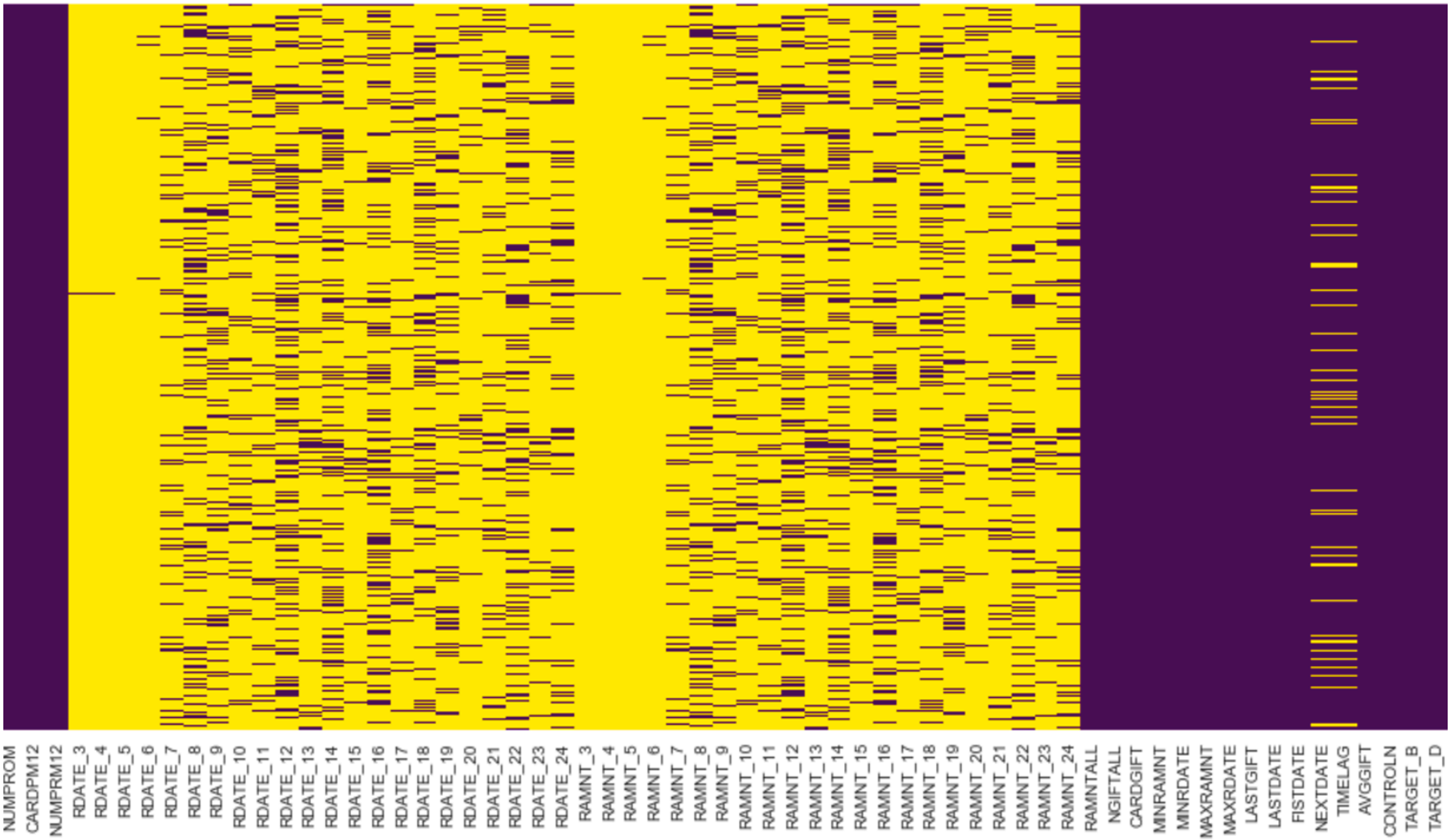
Missing Values - 1



Missing Values - 2



Missing Values - 3



Recode - 1

- Philosophy
  - Minimize the manipulation
  - Let the algorithm handle the most painful part

- Steps
  - Convert flags to 0,1
  - Recode some blank values
  - Recode the date (YYMM to difference in month)
  - Define ordered variables
  - Imputation of missing values
    - The most frequent for nominal, median for numeric
  - Create dummy variables (see the details in the next slide)
  - Create standardized data set for variable reduction (StandardScaler)

    Standardize features by removing the mean and scaling to unit variance

Recode - 2

- Create dummy variables (0-1 binary) for each level of categorical variables

```
# Encoding categorical features
'''
Right now, OneHotEncoder only support numeric variables.
Example:
OneHotEncoder(sparse = False).fit_transform( testdata[['age']] )

For string, there are two commmon ways.

1) Apply LabelEncoder() to convert strings into numbers, then apply OneHotEncoder().
Example:
testdata = pd.DataFrame({'pet': ['cat', 'dog', 'dog', 'fish', None, None],
'age': [4 , 6, 3, 3, None, None],
'salary':[4, 5, 1, 1, None, None]})
a = LabelEncoder().fit_transform(testdata['pet'])
OneHotEncoder( sparse=False ).fit_transform(a.reshape(-1,1))

2) Apply LabelBinarizer() directly. Note: LabelBinarizer() only accepts 1-D array.
Example:
LabelBinarizer().fit_transform(testdata['pet'])

3) Apply pd.get_dummies() in pandas
Example:
pd.get_dummies(testdata['pet'])

Before when OneHotEncoder() supports string, I use pd.get_dummies().
'''
# create dummy variables

data_output = pd.concat([data_output.drop(labels = var_cate_list, axis=1)
                         ,pd.get_dummies(data_output[var_cate_list])], axis=1)
```

Variable Reduction - 1

- Recursive Feature Elimination with Cross-Validation (RFECV)

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), recursive feature elimination (**RFE**) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a `coef_` attribute or through a `feature_importances_` attribute. Then, the least important features are pruned from current set of features.That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

RFECV performs RFE in a cross-validation loop to find the optimal number of features.

- Logic
  - Split data (standardized) into N folds
  - Fit the model using one of the fold
  - Calculate the average model score on all folds
  - Eliminate the variables with the least importance
  - Repeat until only one variable left
  - Identify the optimal number of variables using model score

Variable Reduction - 2

- REFCV can be done using Python easily. (roc_auc as model score)

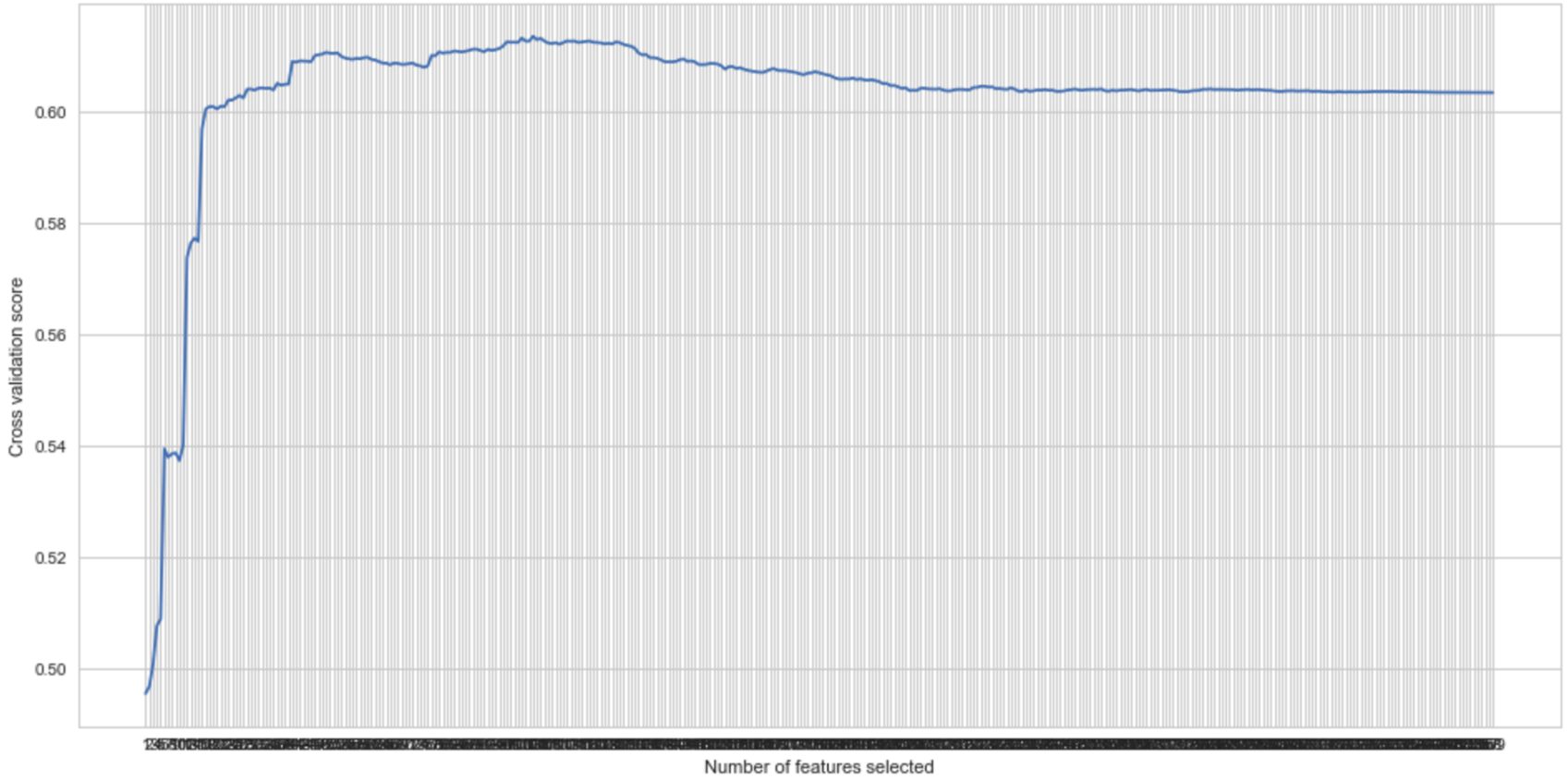
```
# specify the model
estimator = linear_model.LogisticRegression(class_weight='balanced')

# select variables using RFECV
selector = RFECV(estimator, step=1, cv=3, n_jobs=-1, scoring='roc_auc')
selector = selector.fit(X_standardized_train, y_train)
```

- The output is clear and straightforward

```
# plot RFECV result
plt.clf()

# Set figure width and height
fig_width = 16 # width
fig_height = 8 # height
plt.rcParams["figure.figsize"] = [fig_width, fig_height]

plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score")
plt.xticks(range(1, len(selector.grid_scores_) + 1))
plt.plot(range(1, len(selector.grid_scores_) + 1), selector.grid_scores_)
plt.show()

print("\nOptimal number of features : %d\n" % selector.n_features_)
```

```
### show the selected variables
selection = list(zip(selector.ranking_, selector.support_, X_standardized_train.columns))
selection.sort()
for i in selection[:21]:
    print(i)
```

Variable Reduction - 3




```
Optimal number of features : 104
```

Variable Reduction - 4

- Further reduce the variables using RFE

```
# further reduce by RFE
X_standardized_train1 = X_standardized_train[X_standardized_train.columns[selector.support_]]

selector1 = RFE(estimator, n_features_to_select=17, step=1)
selector1 = selector1.fit(X_standardized_train1, y_train)
```

```
Index(['ETH1', 'DW1', 'DW3', 'DW4', 'DW5', 'HU3', 'HU4', 'ETHC1', 'ETHC2',
       'ETHC3', 'RHP1', 'RHP2', 'HUPA1', 'HUPA2', 'HUPA3', 'MC2', 'RFA_2F'],
      dtype='object')
```

- Tune the model by manual model selection based on RFE output
  - Eliminate highly correlated variables (consider VIF and relative importance)
  - Test some other variables may have the potential

Final Model

- scikit-learn doesn't have a built-in option for parameter t-test
- But scikit-learn provides REFCV, class_weight

Variable	Coefficient	Description	Recode logic
Intercept	-0.990087		
DW3	-0.005636	Percent Duplex Structure	
ETHC2	0.001538	Percent White Age 15 - 59	
ETHC3	0.00657	Percent White Age 60+	
RHP1	0.00801	Median Number of Rooms per Housing Unit	
MC2	-0.005853	Percent Persons in Same House in 1985	
RFA_2F	0.240414	Frequency code for Donor's RFA status as of 97NK promotion date 1=One gift in the period of recency 2=Two gift in the period of recency 3=Three gifts in the period of recency 4=Four or more gifts in the period of recency	frequency_dict = {'1':1,'2':2,'3':3,'4':4,'5':5,'X':0}
CARDGIFT	0.033236	Number of lifetime gifts to card promotions to date	

VIF, Correlation and Importance

	Variable	VIF
3	RHP1	1.46
1	ETHC2	1.27
4	MC2	1.24
6	CARDGIFT	1.12
5	RFA_2F	1.11
2	ETHC3	1.08
0	DW3	1.03

Variable correlation matrix

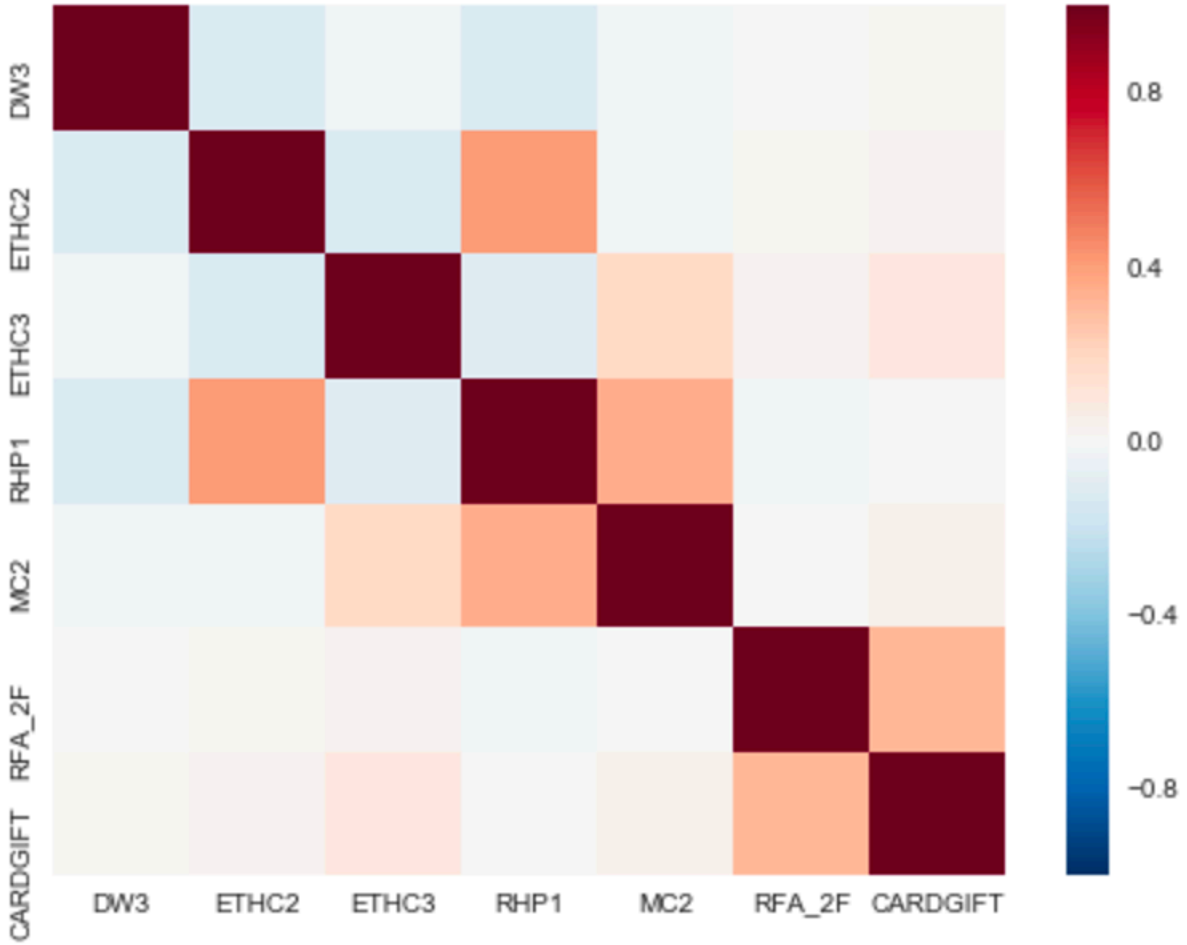
	DW3	ETHC2	ETHC3	RHP1	MC2	RFA_2F	CARDGIFT	TARGET_B
DW3	1.000000	-0.123510	-0.022304	-0.131562	-0.022876	0.003959	0.013234	-0.007879
ETHC2	-0.123510	1.000000	-0.139546	0.409636	-0.027714	0.014595	0.020140	0.011898
ETHC3	-0.022304	-0.139546	1.000000	-0.096236	0.186468	0.031100	0.091574	0.014562
RHP1	-0.131562	0.409636	-0.096236	1.000000	0.345382	-0.021369	0.000602	0.009644
MC2	-0.022876	-0.027714	0.186468	0.345382	1.000000	0.006945	0.044387	-0.009221
RFA_2F	0.003959	0.014595	0.031100	-0.021369	0.006945	1.000000	0.319032	0.071681
CARDGIFT	0.013234	0.020140	0.091574	0.000602	0.044387	0.319032	1.000000	0.054497
TARGET_B	-0.007879	0.011898	0.014562	0.009644	-0.009221	0.071681	0.054497	1.000000

```
# the order of variables by importance
print('Variable importance')
abs(np.std(X_train_new, 0)*estimator.coef_[0]).sort_values(ascending=False)
```

```
Variable importance

RFA_2F       0.257895
CARDGIFT     0.150445
MC2          0.094017
RHP1         0.086395
ETHC3        0.080984
DW3          0.030426
ETHC2        0.022403
```

Variable Correlation Plot

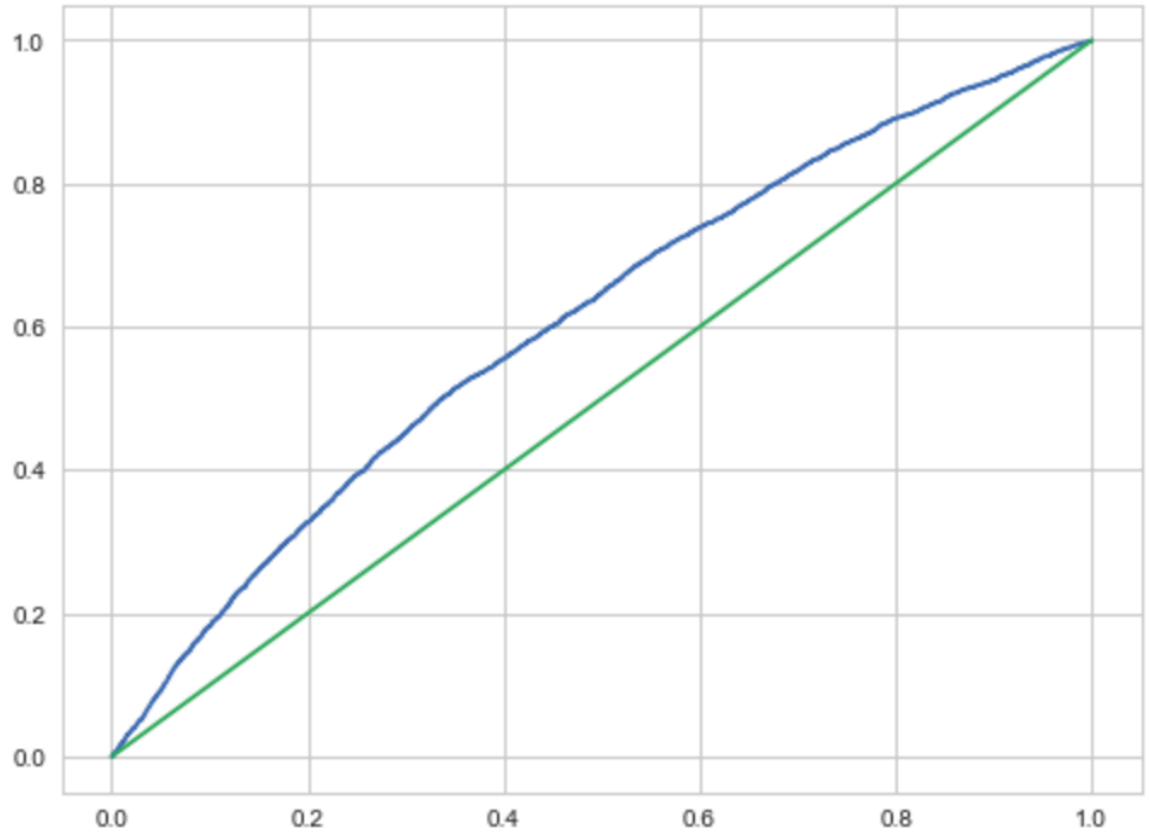
- Correlation matrix



Model Performance on Train Set - 1

- Accuracy and ROC curve

```
Training Set Accuracy 0.618297108569
Training Set Area Under the ROC 0.607437245743
```



Model Performance on Train Set - 2

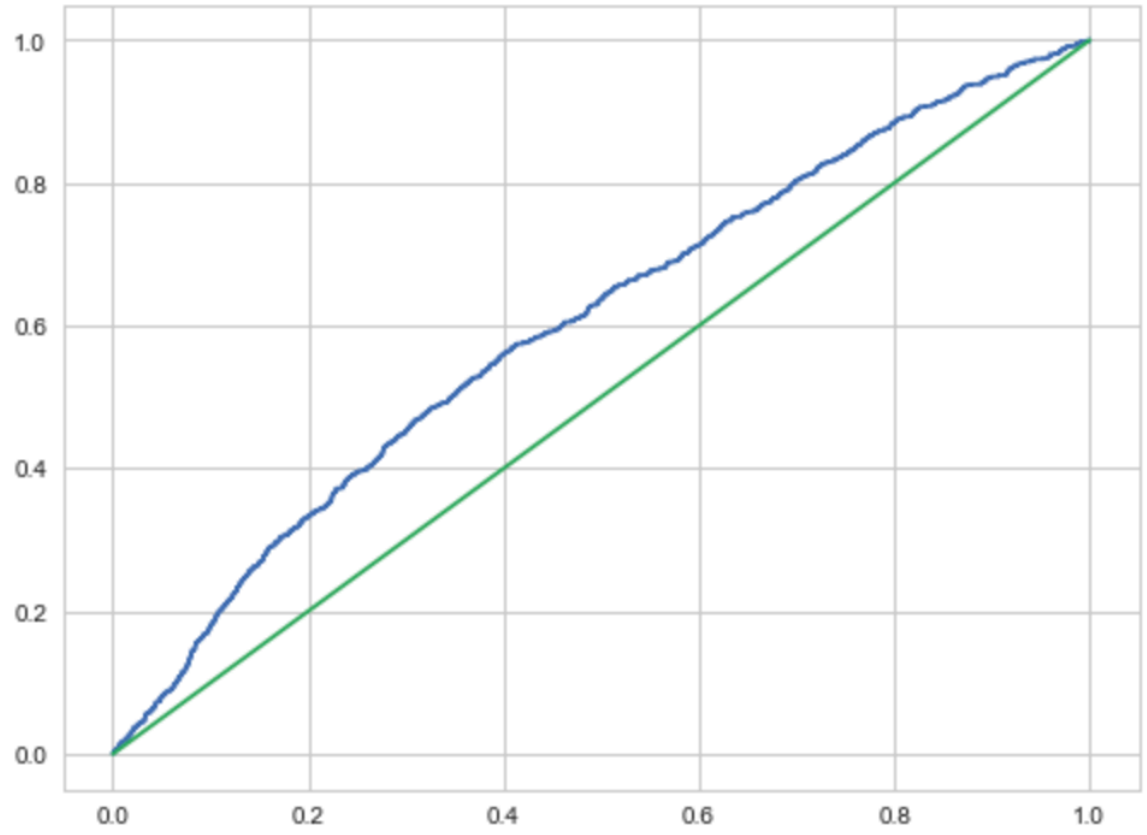
- Performance by decile

decile	size	resp	resp %	cum. resp	cum. resp %
1	7633	679	17.75	679	17.75
2	7633	543	14.20	1222	31.95
3	7633	473	12.37	1695	44.31
4	7633	405	10.59	2100	54.90
5	7633	345	9.02	2445	63.92
6	7632	361	9.44	2806	73.36
7	7633	307	8.03	3113	81.39
8	7633	286	7.48	3399	88.86
9	7633	209	5.46	3608	94.33
10	7633	217	5.67	3825	100.00

Model Performance on Test Set - 1

- Accuracy and ROC curve

```
Testing Set Accuracy 0.619818686789
Testing Set Area Under the ROC 0.600936179492
```

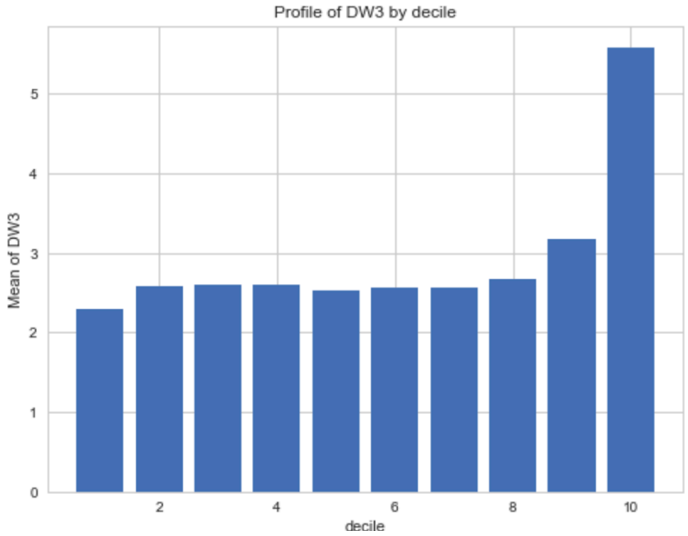


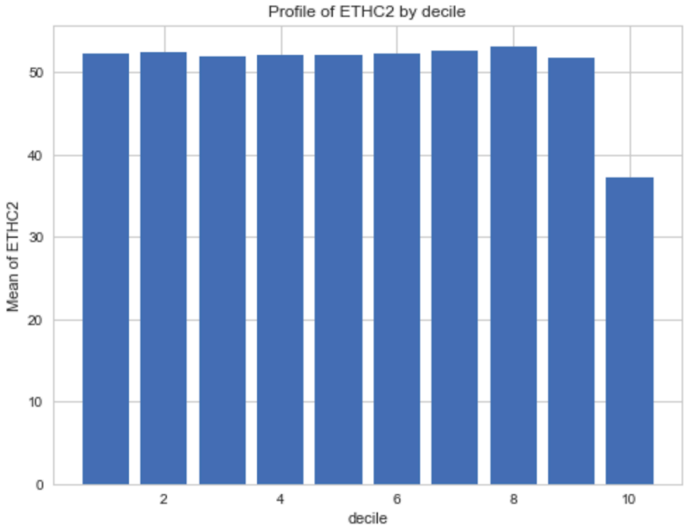
Model Performance on Test Set - 2

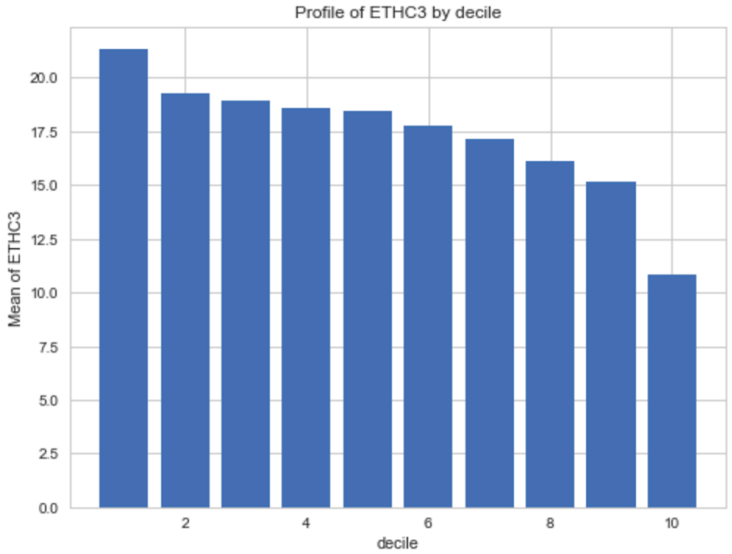
- Performance by decile

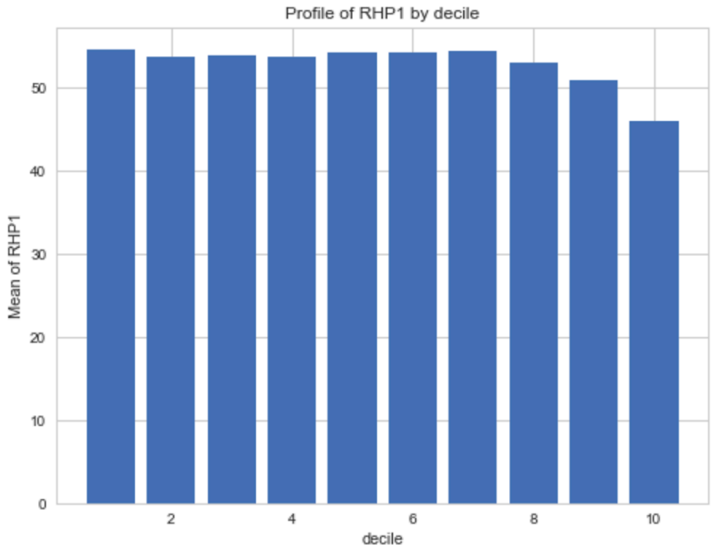
decile	size	resp	resp %	cum. resp	cum. resp %
1	1909	172	16.90	172	16.90
2	1908	158	15.52	330	32.42
3	1908	123	12.08	453	44.50
4	1908	105	10.31	558	54.81
5	1909	82	8.06	640	62.87
6	1908	82	8.06	722	70.92
7	1908	90	8.84	812	79.76
8	1908	83	8.15	895	87.92
9	1908	69	6.78	964	94.70
10	1909	54	5.30	1018	100.00

Variable Profiling by Decile - 1

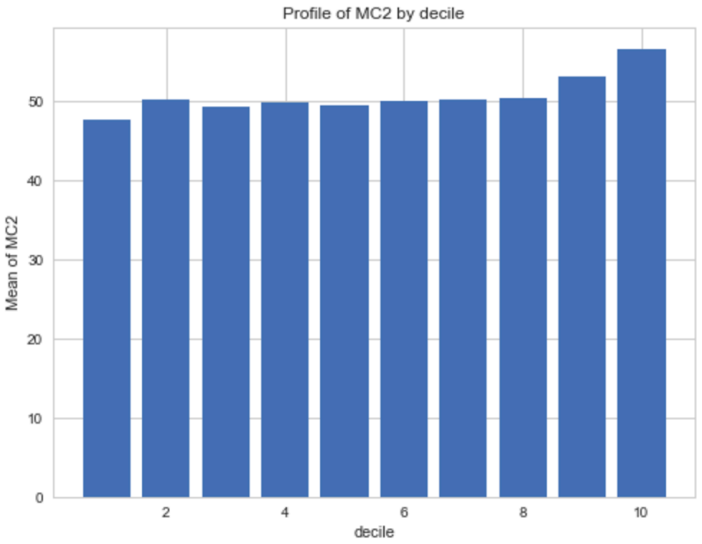


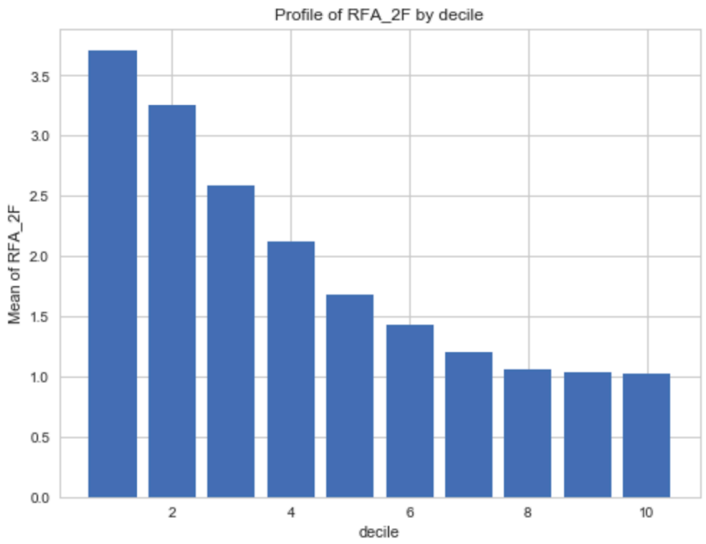


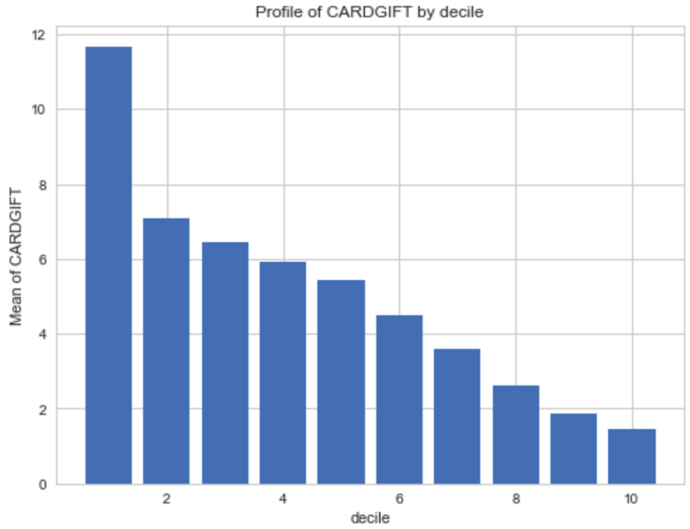




Variable Profiling by Decile - 2







decile	DW3	ETHC2	ETHC3	RHP1	MC2	RFA_2F	CARDGIFT
1	2.287043	52.294904	21.316913	54.599764	47.744137	3.706537	11.655312
2	2.574348	52.481855	19.289925	53.676536	50.237914	3.260841	7.092100
3	2.598978	51.916678	18.947727	53.945500	49.408620	2.586270	6.450282
4	2.606446	52.153806	18.560854	53.775580	49.913402	2.122363	5.933316
5	2.526530	52.032622	18.467313	54.206079	49.541596	1.683873	5.442945
6	2.563548	52.164046	17.730346	54.309617	50.016116	1.429507	4.505110
7	2.566226	52.583519	17.129962	54.511463	50.279051	1.198087	3.584305
8	2.668282	53.068125	16.145159	53.057513	50.380322	1.063540	2.634089
9	3.167824	51.687279	15.131403	50.976287	53.214726	1.034980	1.861260
10	5.576444	37.132058	10.830080	46.086467	56.573038	1.021093	1.439801

Comments

- About the model
  - Logistic model was used to predict the response possibility.
  - There are seven variables in the final model.
  - Frequency is the most important factor to predict the response possibility.
  - The model is stable according to out-of-sample validation. (for the train set auc_roc = 0.607, for the test set auc_roc = 0.601)
  - On the test set, the top one decile by predicted probability ranking captured 16.9% of total responders. (predictive power)
  - The numbers of responders captured by the top four deciles followed a descending order. (stability)

- About building model using Python
  - Flexible and easy to pick up
  - Simple to archive amazing functions (RFECV, RFE, balanced weight)
  - Still requires some self-defined functions (partial correlation, VIF, confusion matrix)
  - sklearn focus more on the machine learning than statistics (no t-test option)

Outlooks

- Plan to develop some feature engineering functions for this set of scripts (e.g. states grouping, capping and flooring, transformation)
- Plan to develop variable reduction methods other than RFECV and RFE (e.g. variable score according to the significance of multiple models)
- Plan to develop customized model score for REFCV (e.g. percentage of responders captured by the top three deciles)

Thank you!

- Link to files
  - https://github.com/joeshan/kdd_cup_98