Improving the efficiency of a bank’s telemarking campaign aimed towards fixed term deposit subscriptions

BY: *NUSAIR IMAM*

MENTOR: *ILYAS USTUN*

PROGRAM MANAGER: *JOSEPHINE PIKE*

Presentation Outline

1. Understanding the data
2. Data analysis, cleaning and preparation
3. Initial application of selected models
4. Feature engineering and parameter tuning
5. Conclusion and Future Considerations

The data - variables

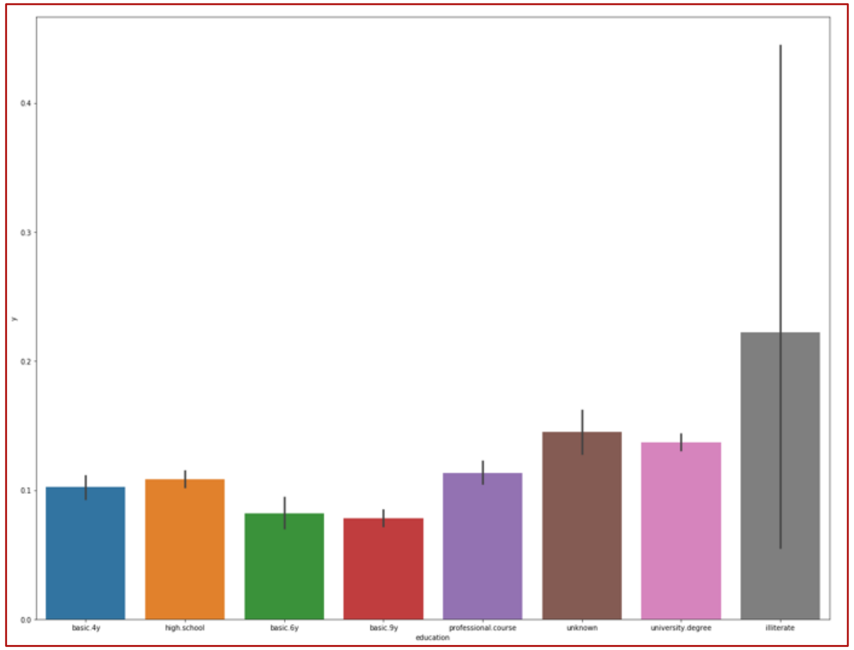
Categorical (dtype = object)	Continuous (dtype = numerical)
Job	Age
Marital	Duration
Education	Campaign
Default	Pdays
Housing	Previous
Loan	Emp.var.rate
Contact	cons.price.idx
Month	Cons.conf.idx
Day_of_week	Euribor3m
Poutcome	Nr.employed

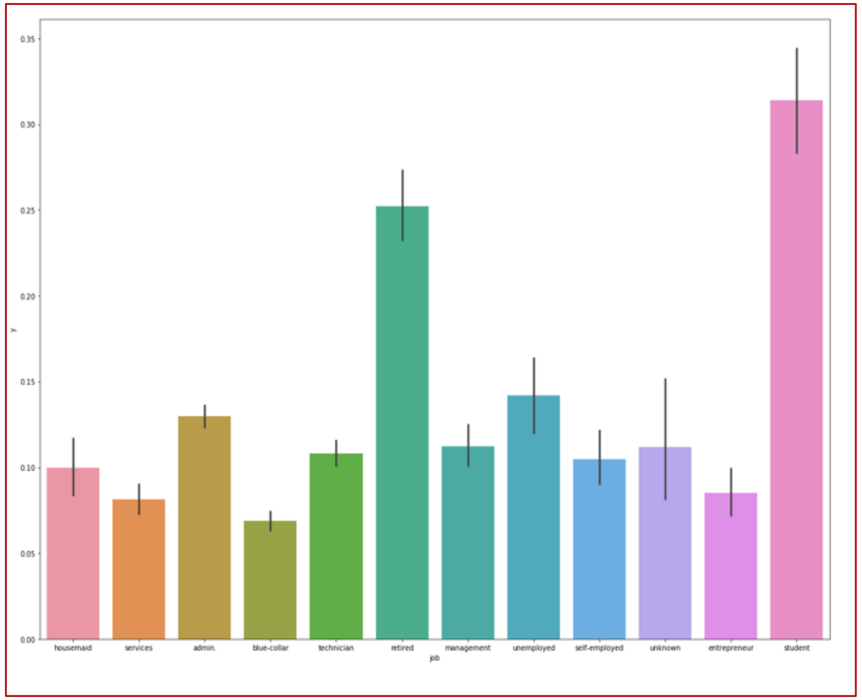
Object Variables

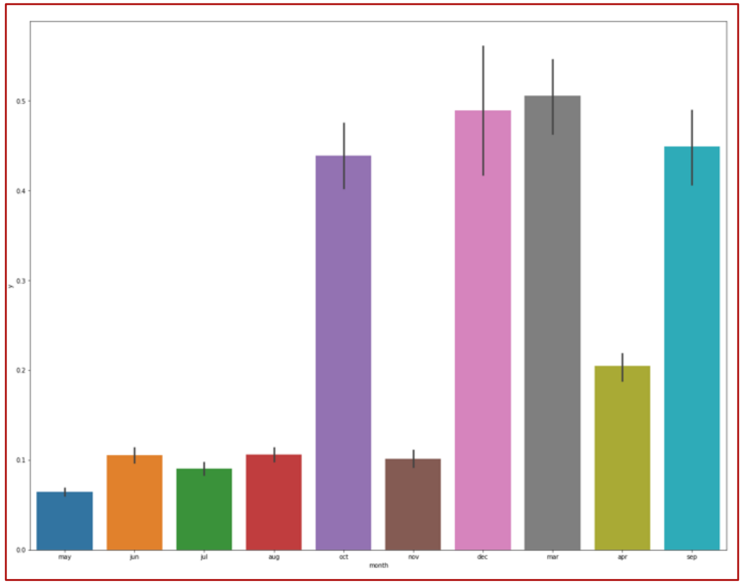
- Job : Occupational information
- Marital : Marital status
- Education : Education level
- Housing : Housing loan (Yes/No)
- Default : Credit default (Yes/No)
- Loan : Personal loan (Yes/No)
- Contact : Method of contact (telephone/cellular)
- Month : Month of the year (last contact)
- Day_of_week: Day (last contact)
- Poutcome: Outcome of previous campaign

Object Variables
Key Takeaways

- Students and retired people have a greater chance of making a deposit
- People who identified as illiterate have a higher chance of making a deposit ; large error bar indicates presence of outliers
- Certain months have a higher success rate (March, April, Sept, Oct, Dec)
- Class imbalance ; only 11% of participants made a deposit

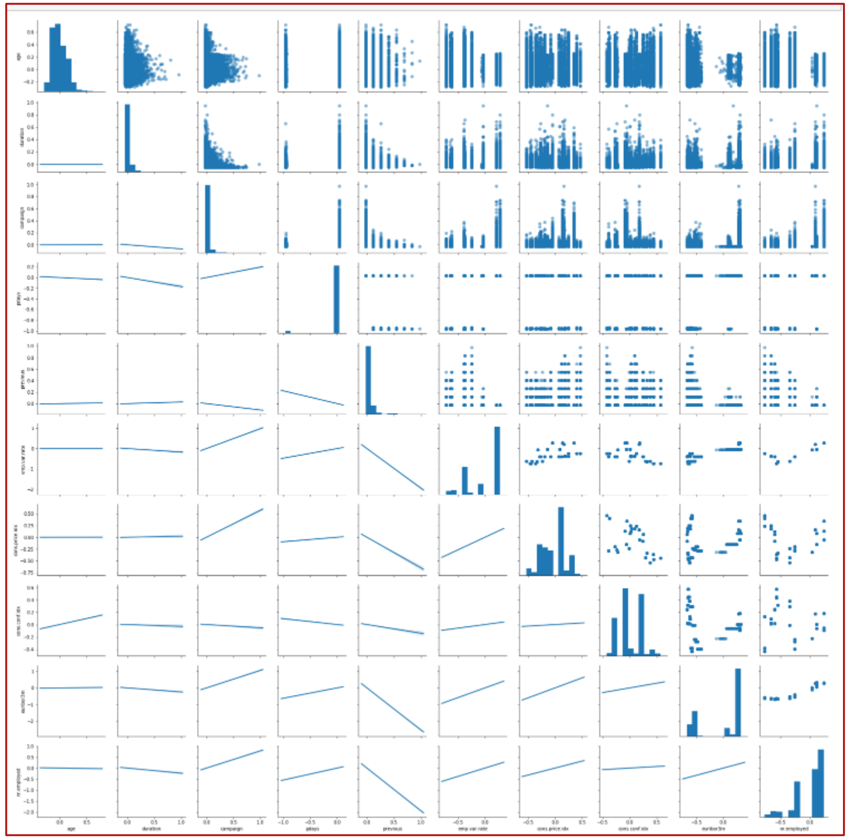






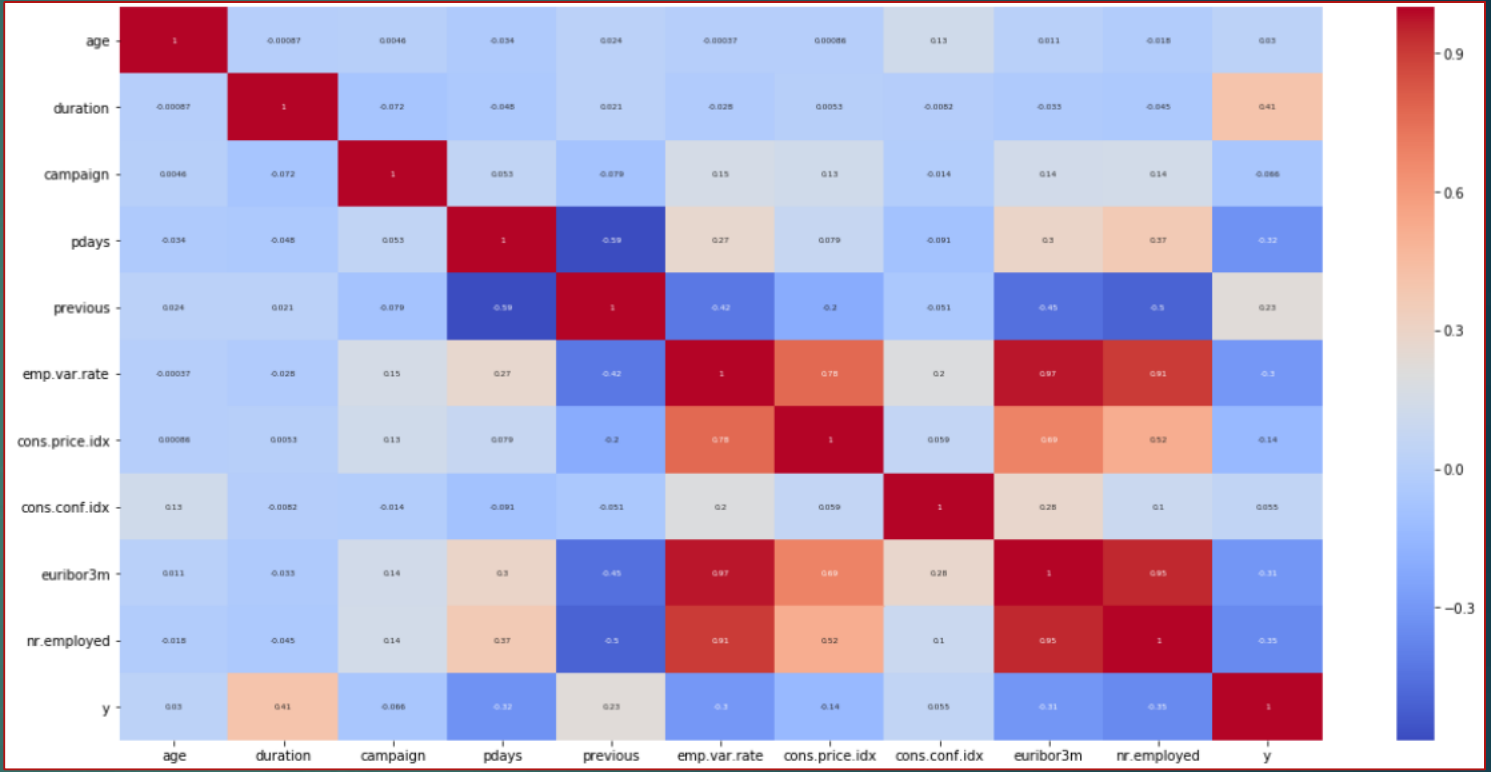
Numerical variables

- Age : Age of client
- Duration : Contact duration (seconds)
- Campaign : Number of calls made to client
- Pdays : Number of days since last contact (previous campaign)
- Previous : Number of calls in previous campaign
- Emp.var.rate : Employment variation rate (quarterly)
- cons.price.idx : Consumer price index (monthly)
- Cons.conf.idx : Consumer condfidence index (monthly)
- Euribor3m : Euro interbank interest rate (daily)
- Nr.employed : Number of employees (quarterly)



Numerical variables - heatmap

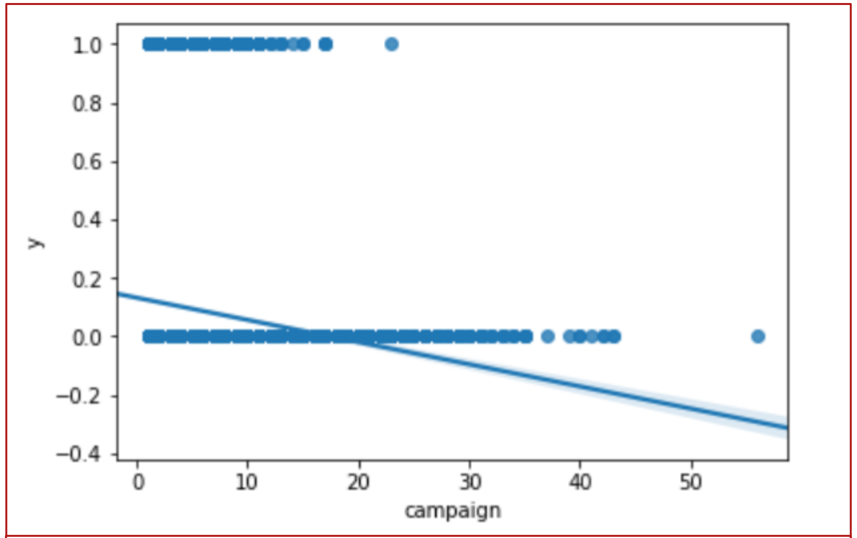
- High correlations between the output and the following variables:
  - Nr.employed
  - Euribor3m
  - Duration
  - Emp.var.rate
  - Pdays
  - Previous (outcome)

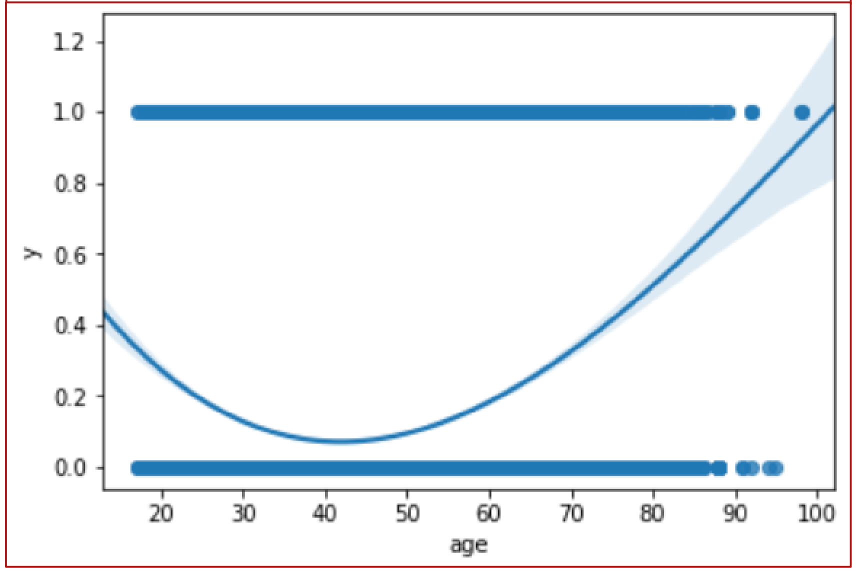


Numerical variables

Key Takeaways

- Rememeber – the variable ‘campaign’ is the number of calls rendered per client during the campaign
  - No person who was contacted more than 20 times subscribed to a fixed term deposit
- People at the ends of the age spectrum are more likely to subscribe





Data Cleaning

- No NaN values, missing data
- Converted output from Yes/No to numerical 1/0
- Creating dummy variables for object variables in order to have a numerical dataset with all the variables.
- Dealt with class imbalance (data skewed towards the negative outcome)
  - Oversampling via Sklearn's SMOTE

Oversampled data

- SMOTE (Synthetic Minority Oversampling Technique)
  - Potentially important information may have been simulated (risk of overfitting)

```python
#dealing with class imbalance
from imblearn.over_sampling import SMOTE
smote =  SMOTE(random_state = 1, ratio = 1.0)
xx = X
yy = Y

x_balanced,y_balanced = smote.fit_sample(xx,yy)

print(x_balanced.shape)
print(y_balanced.shape)

x_var = list(xx.columns)
bal_df = pd.DataFrame(data = x_balanced , columns = x_var)
bal_df.describe()
bal_df.describe()
```

Test, train methodology

- Data split on an 80:20 basis using Sklearn's train_test_split function
- Models trained on train dataset and tested on both train and dataset
- *Initial modelling on Oversampled train dataset with 63 features*
- *Final modelling on reduced set of features based on feature importance from RFC, LogR and feature engineering*

Terms and definitions – *in context of what it means in this application*

- Type 1 error: FP
  - *Incorrectly identifying non-subscibers as subscribers*
- Type 2 error: FN
  - *Not identifying those that are predicted to subscribe*
- **Precision: TP/(TP+FP)**
  - **Out of those that we think will subscribe, what percentage actually did?**
- **Recall: TP/(TP+FN)**
  - **Out of those that did subscribe, what percentage did we predict would?**

Terms and definitions – *in context of what it means in this application*

- Precision: TP/(TP+FP)
  - Out of those that we think will subscribe, what percentage actually did?
    - Example Scenario : sending coupons to those that think are likely to subscribe – you do not want to waste coupons on those that will not subscribe!
- Recall: TP/(TP+FN)
  - Out of those that did subscribe, what percentage did we predict would?
    - Bank telemarketing – this is the primary KPI for this application – we do not want to misclassify people who end up subscribing

Models

- Random Forest Classifier
- K-nearest Classifier
- Logistic Regression
- SVM
  - SVC
  - Linear SVC

	RFC	LogR	KNN	SVC	lSVC
Accuracy (%)	83.5	88.5	90.7	91.0	88.4
Type 1 (%)	7.92	6.76	8.20	6.04	6.96
Type 2 (%)	8.56	4.75	1.11	2.93	4.61
Precision (%)	84.0	87.0	85.6	88.6	86.7
Recall (%)	82.9	90.5	97.8	94.1	90.8

**Initial Results : Trained and tested on entire dataset*

Models – Initial training

- Overall accuracy is consistent and high
  - Slow to compute due to large number of features (n = 63)
- Tendency to predict False positive (Type 1)
  - This manifests itself in the Precision

	RFC	LogR	KNN	SVC	ISVC
Accuracy (%)	85.5	85.2	84.7	88.9	86.8
Type 1 (%)	12.6	11.2	14.6	11.9	11.7
Type 2 (%)	1.98	1.49	0.70	1.12	1.46
Precision (%)	41.9	46.0	41.17	45.3	45.0
Recall (%)	82.1	86.5	93.6	89.8	86.8

**Initial Results : Test of test subset (after test/train split)*

Parameter Tuning

- Sklearn's GridsearchCV
  - SVC left out due to computational constraints

```
knn = KNeighborsClassifier()

params = {'n_neighbors':[1],
          'leaf_size':[1,2,3,5],
          'weights':['uniform', 'distance'],
          'algorithm':['auto', 'ball_tree','kd_tree','brute'],
          'n_jobs':[-1]}
```

```
rfc = ensemble.RandomForestClassifier()

params = {"n_estimators": [10, 5, 15],
          "max_depth": [3, 4, 5],
          "min_samples_split": [2, 3],
          "min_samples_leaf": [5, 10, 20],
          "max_leaf_nodes": [20, 40],
          "min_weight_fraction_leaf": [0.0]}
```

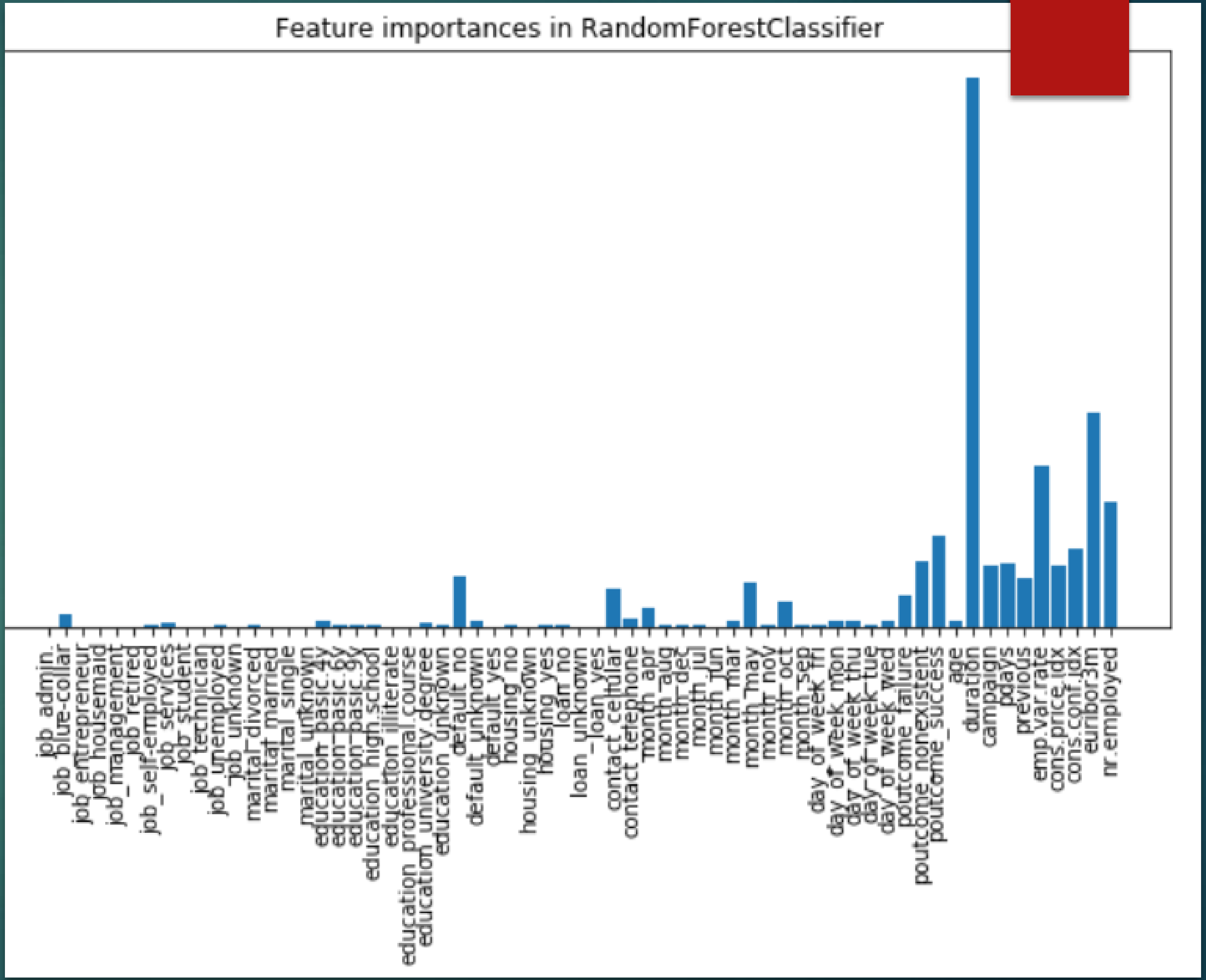
```
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn import svm
Cs = [0.001, 0.01, 0.1, 1, 10]
penalty = ['l1','l2']
param_grid = {'C': Cs, 'penalty' : penalty, 'dual' : [False]}
grid_search = GridSearchCV(svm.LinearSVC(), param_grid,  return_train_score=True)
grid_search.fit(X, Y)
```

```
{'C': [0.0001, 0.001, 0.01, 0.03, 0.1, 0.3, 0.6, 1, 1.3, 1.6, 2, 5, 10, 15, 20, 50, 100]}
```

```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
lr = LogisticRegression()

grid = GridSearchCV(estimator=lr, param_grid=param_grid, scoring='accuracy', verbose=3, n_jobs=-1, return_train_score=True)
```

Feature engineering

- Features shortlisted based on RFC importance diagram
- Duration feature removed since this is not known until after the call (when the outcome is known)
- Conversion rate feature added (# of calls divided by outcome)



Feature engineering

- The reduced dataset exhibited overfitting
- Only after removal of the ‘duration’ column did results become more realistic

```
from sklearn import ensemble
from sklearn.model_selection import cross_val_score
rfc = ensemble.RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features='auto', max_leaf_nodes=40,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=15, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
fit_and_train(rfc)
```

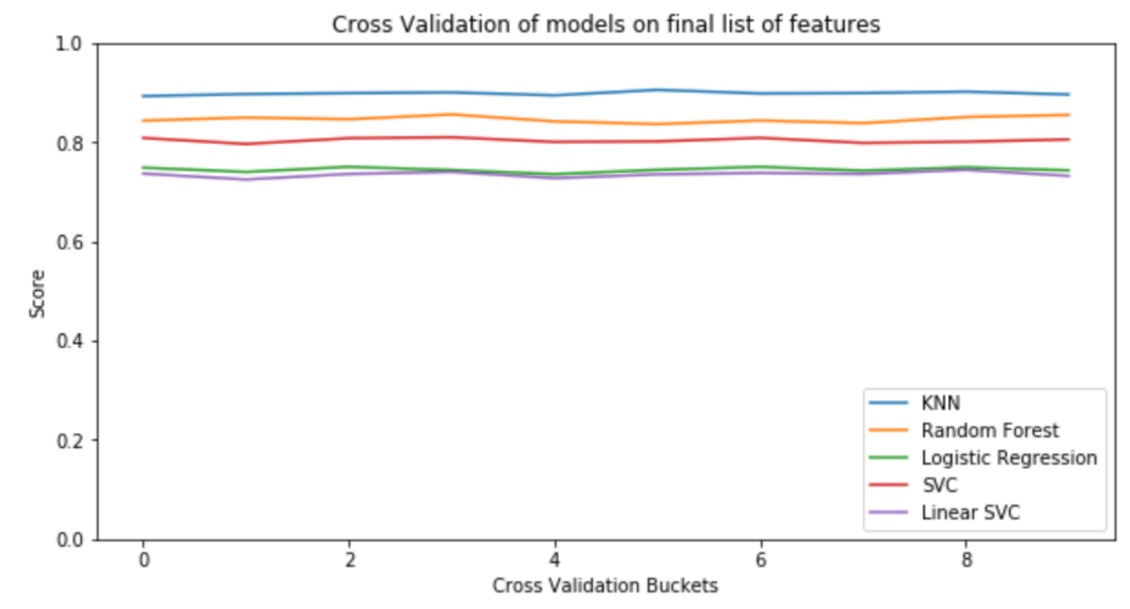
```
----Results based only on training dataset----
Accuracy: 0.9935187085299952
predicted      0      1    All
actual
0          28956    244  29200
1            135  29141  29276
All        29091  29385  58476

Type I errors: 0.42%
Type II errors: 0.23%

Precision: 99.17%
Recall: 99.54%

----Results based on test dataset----
Accuracy: 0.9930916552667579
predicted     0     1    All
actual
0          7286    62   7348
1            39  7233   7272
All        7325  7295  14620

Type I errors: 0.42%
Type II errors: 0.27%

Precision: 99.15%
Recall: 99.46%
```

Results – reduced dataset with 'duration' column

	RFC	LogR	KNN	SVC	ISVC
Accuracy (%)	86.4	88.5	90.7	88.9	88.3
Type 1 (%)	7.09	6.81	7.96	7.74	7.07
Type 2 (%)	6.48	4.61	1.27	3.37	4.58
Precision (%)	85.9	86.9	85.90	85.7	86.5
Recall (%)	87.0	90.73	97.5	83.22	90.8

Results – without 'duration' column

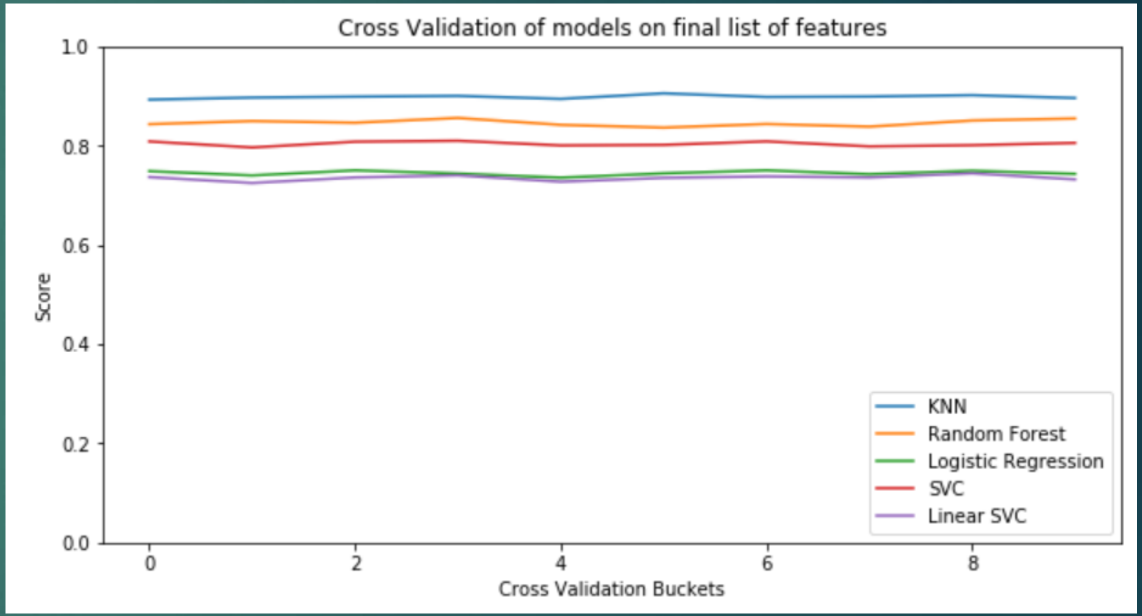
- Consistent cross validation scores
- RFC is the best model despite KNN having better overall results
  - SMOTE method utilizes the same methodology (Euclidean distance) as KNN and therefore KNN is more prone to overfitting on SMOTE-oversampled dataset



	RFC	LogR	KNN	SVC	lSVC
Accuracy (%)	84.7	75.0	90.8	80.9	74.1
Type 1 (%)	5.16	7.05	4.17	6.25	9.17
Type 2 (%)	10.1	17.94	5.08	12.9	16.73
Precision (%)	88.5	81.8	91.4	85.5	78.2
Recall (%)	79.7	63.8	89.8	74.0	66.3

Final Recommendation - Observations

- Random Forest Classifier performed the best overall
  - No client should be called more than 20 times
  - Depending on industry standard hitrate, the number of calls should be optimized by putting a maximum limit.



Future considerations

- Data treatment can be altered
  - PCA on highly correlated columns to reduce multicollinearity
  - Undersampling via numpy's random choice function
- Unsupervised techniques should be explored (Nueral networks)
- More features should be added to model
  - Gross income
  - Registered residence location
  - Size of family