Machine Learning for mailing campaigns

Outline

- Introduction
  - Project aim – business value
- Data cleaning and exploration
  - Elimination of duplicates and unreasonable values
  - Imputation of missing values
  - Is our dataset balanced ? Does it contain outliers ?
- Modelling
  - Data preparation : features engineering, categorical encoding, scaling
  - Evaluation of several machine learning (ML) models
  - Can the best performing model answer our business questions ?
- Conclusions
  - Machine learning model performance
  - The features that identify an ideal mailing target group of residents

Business case

- Extensively mailing a large population is costly and inefficient
- Manually identifying responders, who may turn to target customers, would take too long

Solution : get a software to identify them automatically

The data collected so far will « train » it for the job

Project goal

From the personal data collected from the population of a large residential area we want to:

- Build a machine learning (ML) model that automatically predicts the whether a resident will respond or not to mailings

- Identify the features of a resident that would most probably respond

THE DATA

Dataset exploration and cleaning :

good data → valuable insight

The dataset

- 9 predictive variables, 1 target variable « label »

10000 samples (residents)

2.1. What are our variables?

```
# Number of rows and columns in the dataframe
print(df_population_raw.shape)
df_population_raw.head(3) #visualize dataframe
```

```
(10000, 10)
```

	name	age	lifestyle	zip code	family status	car	sports	earnings	living area	label
0	VnSEFOuL	62.0	cozily	50168.0	married	practical	athletics	102526.0	urban	no response
1	8Tv0hcce	34.0	active	66479.0	married	expensive	soccer	33006.0	urban	no response
2	Zny9ysbk	69.0	healthy	16592.0	single	expensive	badminton	118760.0	urban	response

Data cleaning

1) Check for duplicate values

2) Fill missing values (es. Sport = ‘unknown’)

- missing values imputation

```
print(df_population_raw['sports'].isna().value_counts())
```

```
False    8500
True     1500
Name: sports, dtype: int64
```

```
# there are 1500 cells in the sport column with no value. We replace them with "Unknown"
df_population_raw['sports'] = df_population_raw['sports'].fillna('unknown')
```

- deduplication

```
duplicateID = df_population_raw[df_population_raw.duplicated(['name'])]
print("we have", len(duplicateID.index), "duplicate name values")
```

```
we have 0 duplicate name values
```

3) Check for unreasonable values (es. negative age)

```
# General statistics of the data
df_population_raw.describe()
```

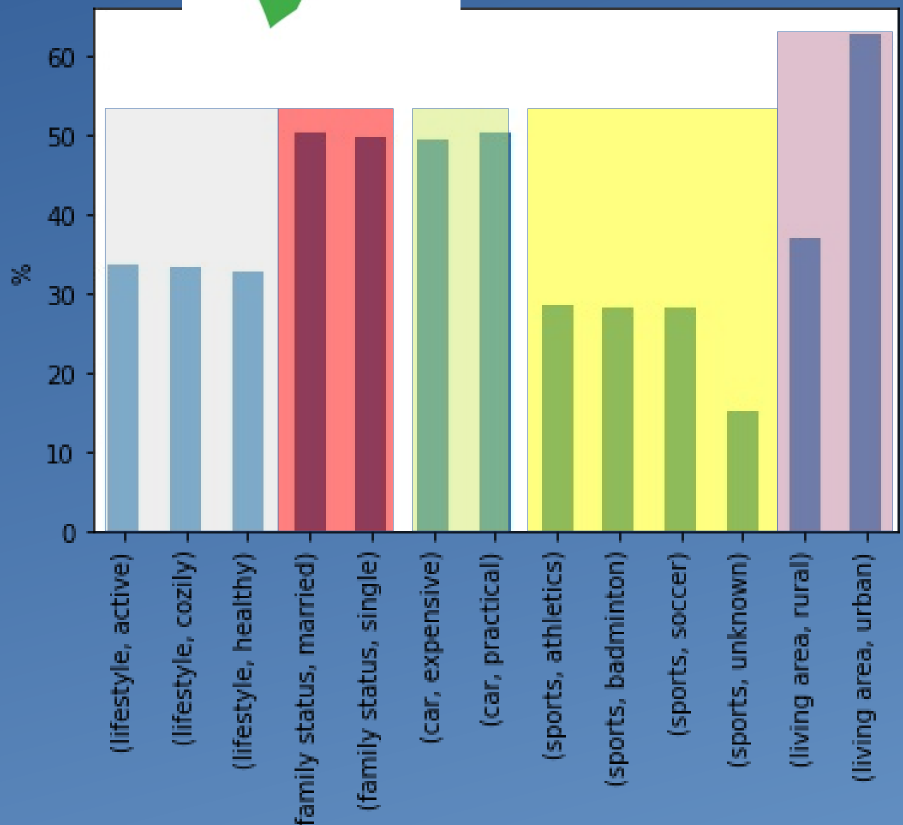
	age	zip code	earnings
count	10000.000000	10000.000000	10000.000000
mean	42.090700	55227.270600	85337.799600
std	15.874416	26139.756227	37554.523323
min	15.000000	10003.000000	20006.000000
25%	28.000000	32708.250000	53237.250000
50%	42.000000	55290.000000	85617.500000
75%	56.000000	77967.750000	118111.000000
max	69.000000	99982.000000	149975.000000

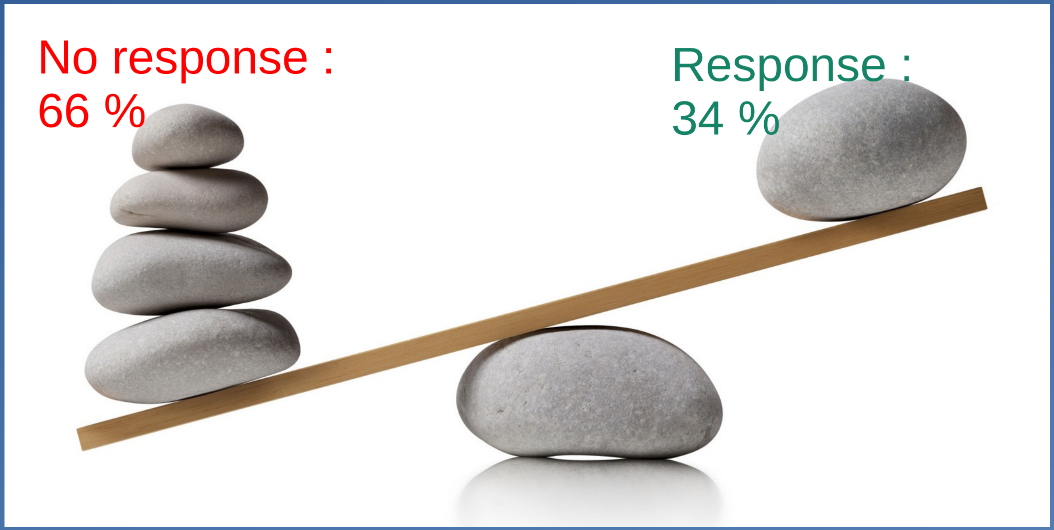
No negative values

Distribution of categorical data

- Are categorical features balanced ?

evenly distributed

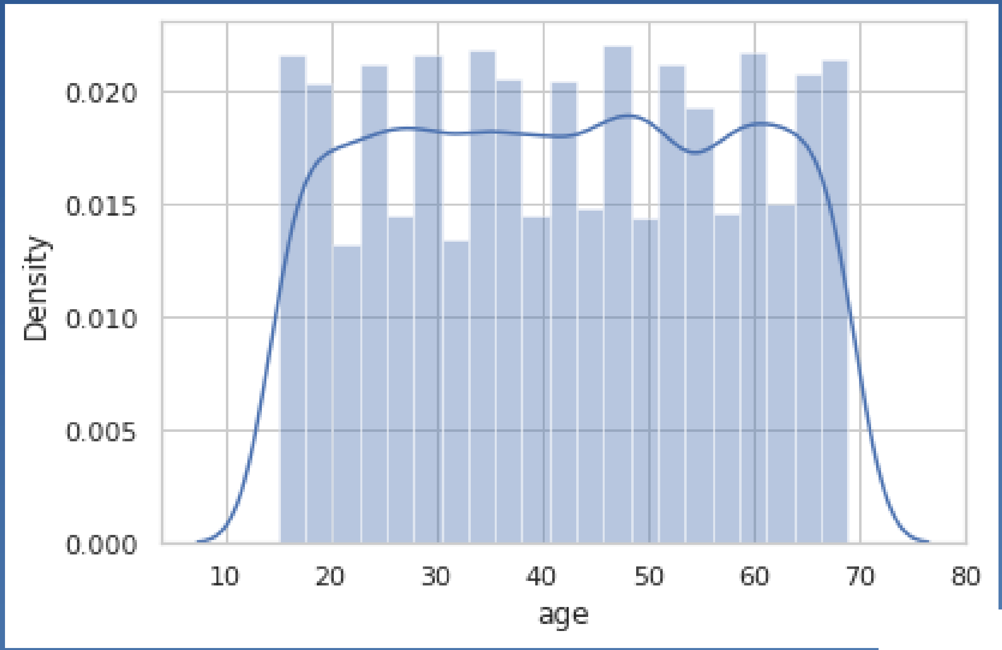


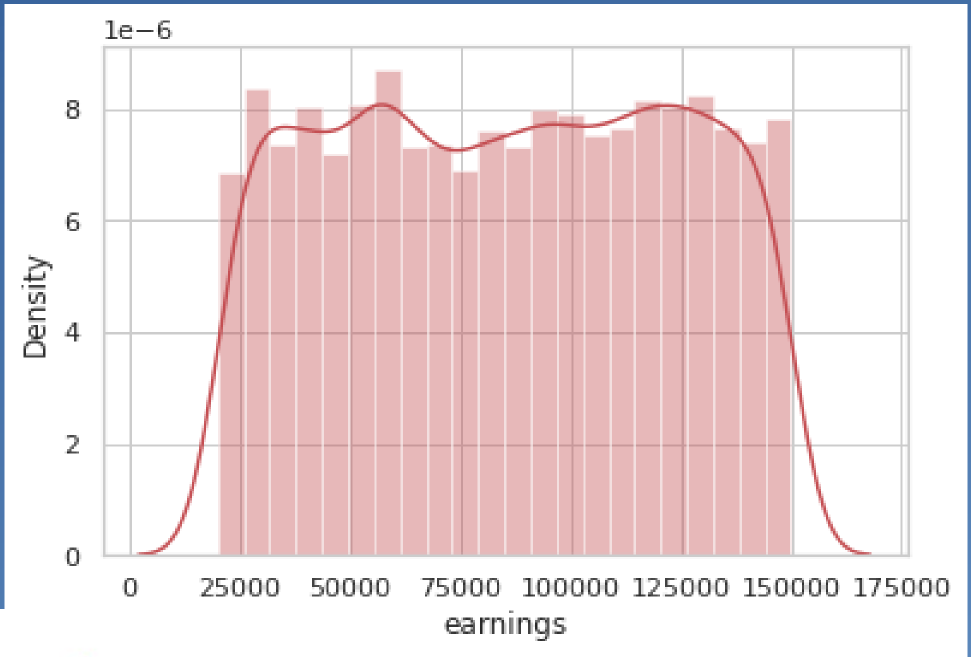


- Categorical features are balanced

Distribution of numerical data

- Are numerical features balanced ?

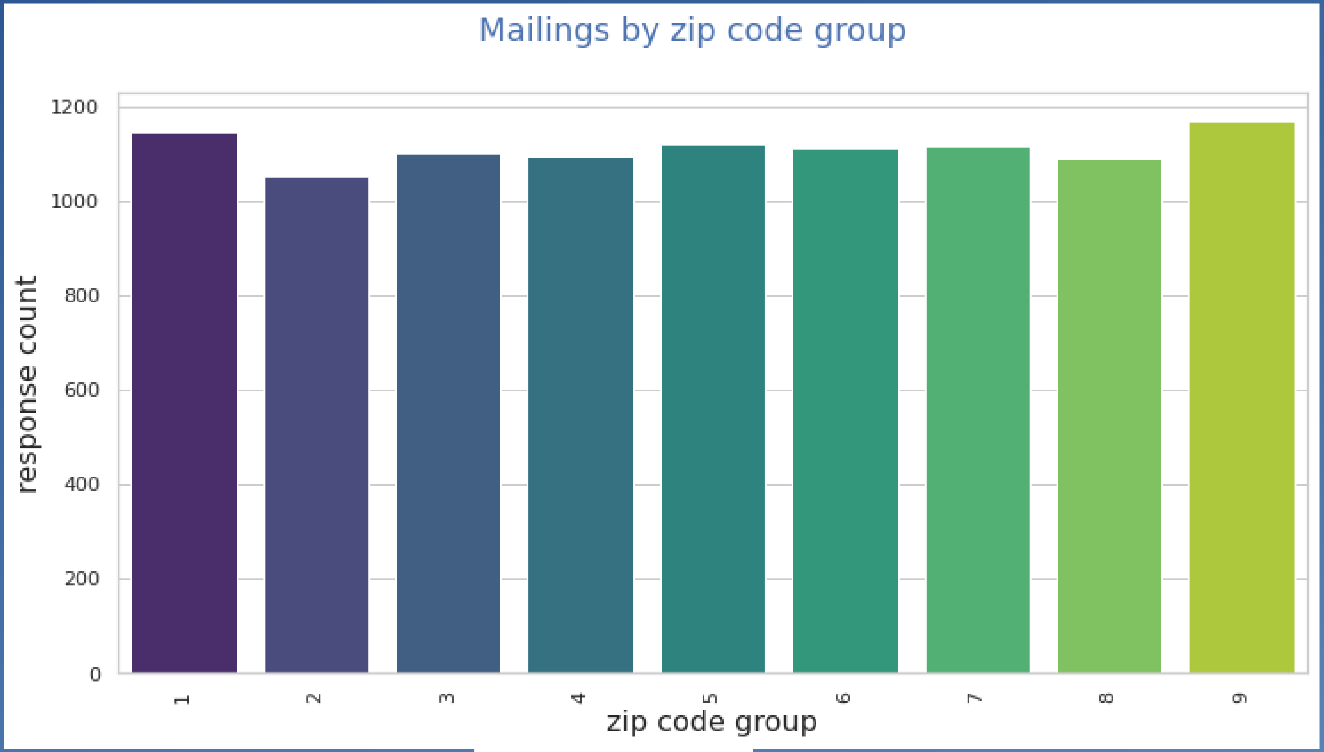




- Numerical features are well balanced, evenly distributed, no outliers

Distribution of geographical data

- Are zip codes evenly distributed ?



« Postcode group » feature

Postcode group 1 : 10000-19999
Postcode group 2 : 20000-29999

Etc.

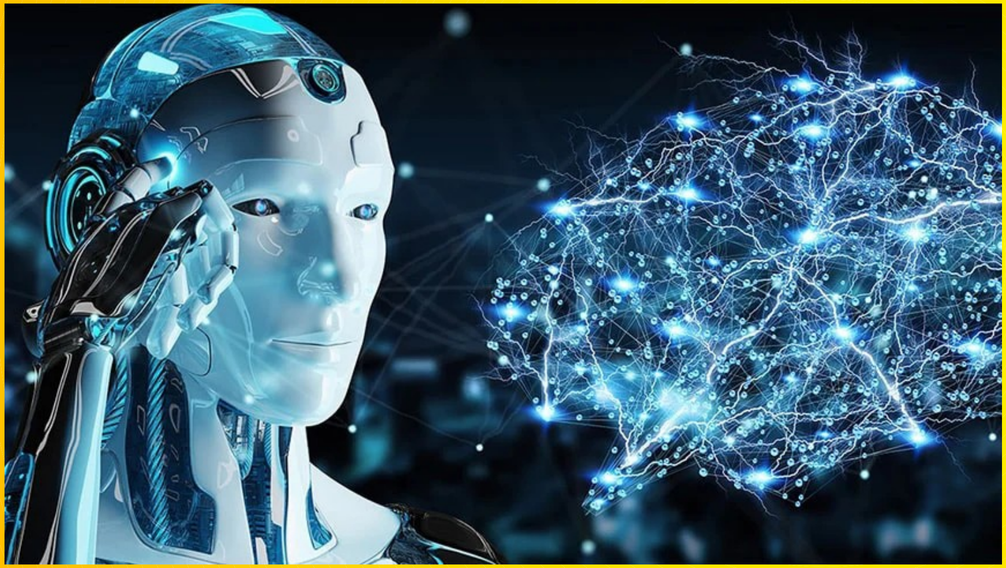
Coded in Python as
Zip code//10000

Preliminary conclusion : the data

- Our data are clean

- The only variable containing empty values is « sport ». Empty cells filled as ‘unknown’

- Data are evenly distributed : this is a balanced dataset, no outliers

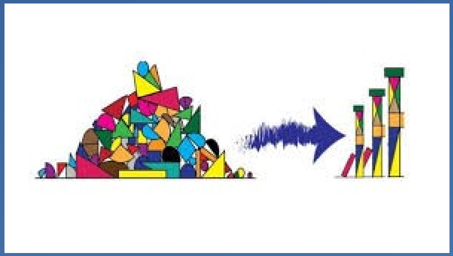


THE MODEL

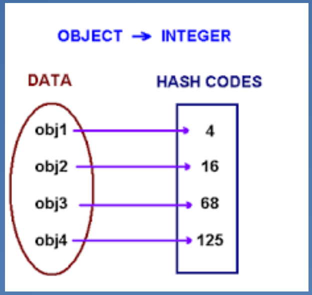
- data preparation (encoding)
- selection of the best model

Data preparation pipeline

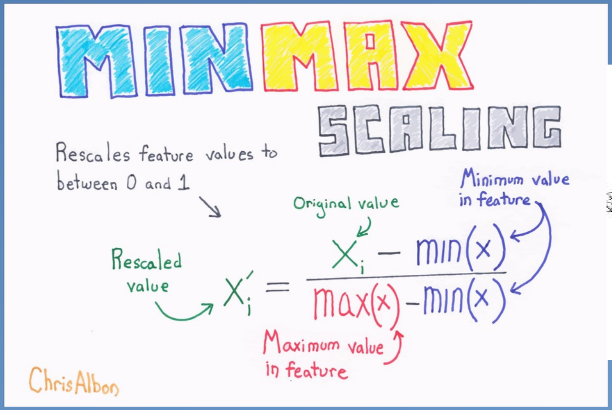
Features engineering
- create ‘zipcode group’ variable

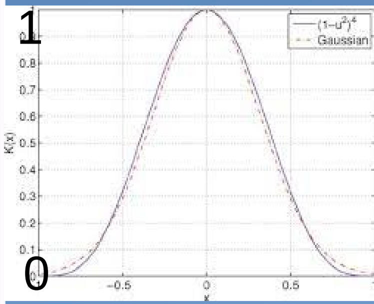


Features encoding
- binary
- one-hot



Features scaling
- apply MinMaxScaler





Feature encoding

Human vs. Artificial intelligence (AI) brain

What humans see

family status	car	sports
married	practical	athletics
married	expensive	soccer
single	expensive	badminton

What an AI needs to see

predictors

married	expensive car owner	urban citizen	sports_athletics	sports_badminton
0	0	1	1	0
0	0	1	0	0
0	0	1	0	1
0	0	1	0	0
0	0	1	0	1
0	0	1	0	1
0	0	0	0	1
0	0	1	0	0

target

response
0
0
1
1
1
0
0

Features transformed using binary and one-hot encoding

1 = yes
0 = no

Model selection : test several

The candidates are those tailored to binary classification : responded or not

Competition results

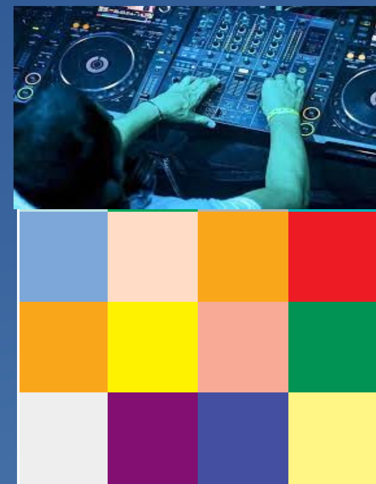
	MLA Name	Fit_time	Accuracy	Precision	Recall	F1
5	RandomForestClassifier	0.008	0.89	0.84	0.84	0.84
7	XGBClassifier	0.005	0.88	0.83	0.84	0.83
2	SVC	0.025	0.84	0.76	0.82	0.70
1	LogisticRegression	0.001	0.81	0.70	0.78	0.64
6	KNeighborsClassifier	0.003	0.79	0.69	0.74	0.65
3	BernoulliNB	0.000	0.67	0.29	0.61	0.19
4	Perceptron	0.000	0.63	0.61	0.49	0.80
0	DummyClassifier	0.000	0.64	0.00	0.00	0.00

- Best 2 are « random forest » and « xg-boost » classifiers

- Recall is the most important metric : as low false negatives as possible since we don’t want to miss responders

Model hyperparameters tuning

Choice of parameters to tune



Random Search

Select best combination

The Random Forest classifier was slower than XG-boost. We tuned it to make it even faster

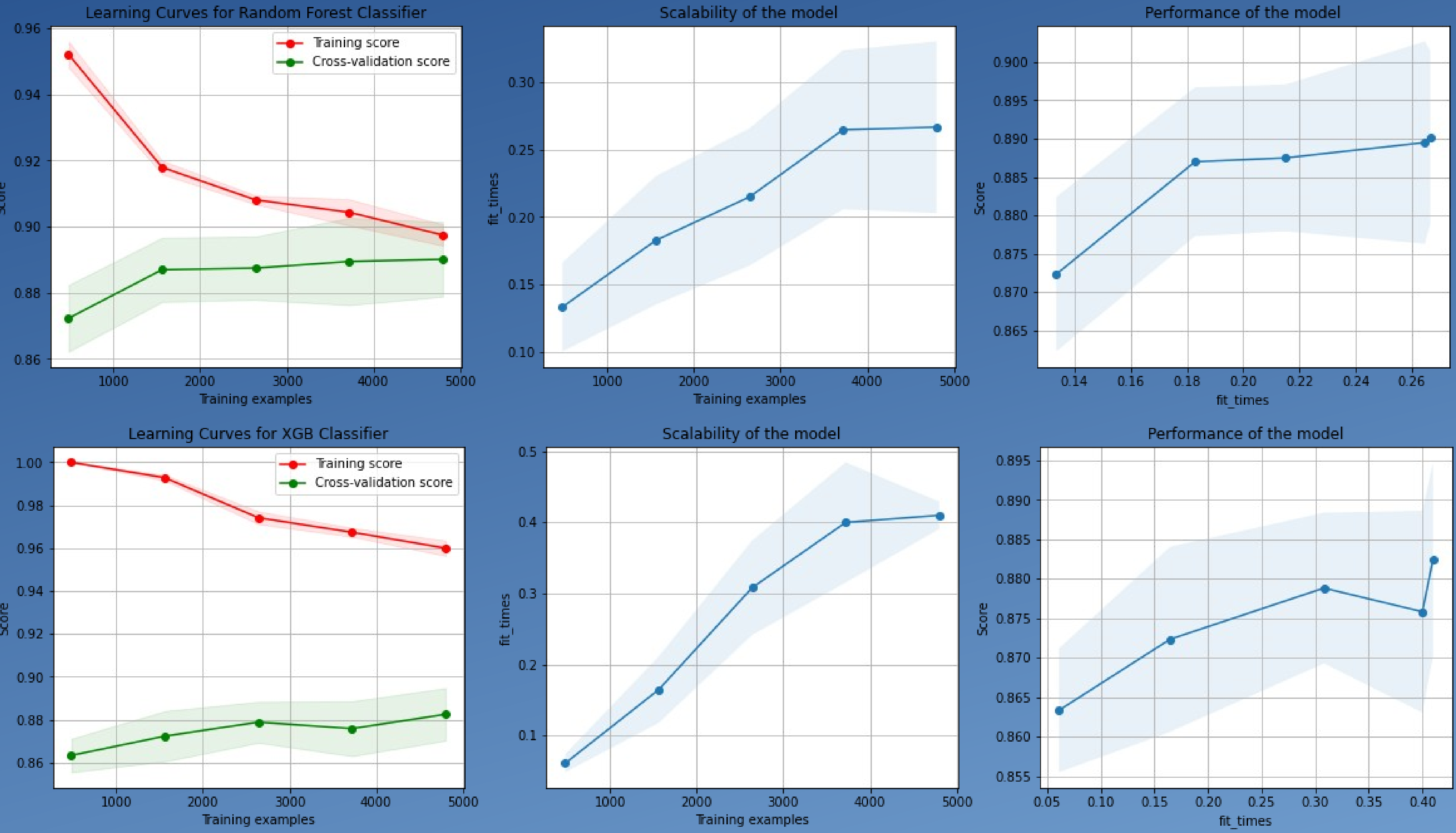
```
#  best model!!
rf1 = RandomForestClassifier( n_estimators = 100, max_depth = 8) #randF.best_estimator_

#initiate time count
global _start_time
start_time = time.time()
rf1.fit(X_train,y_train)
time_taken = round(((time.time() - start_time) / 60),3)
print("fitting with optimized model took",time_taken*1000,'milliseconds vs. 9.0 prior to optimization')
```

```
fitting with optimized model took 4.0 milliseconds vs. 9.0 prior to optimization
```

Best two candidates performance

Random forest converges faster than XG-Boost and is slightly more accurate



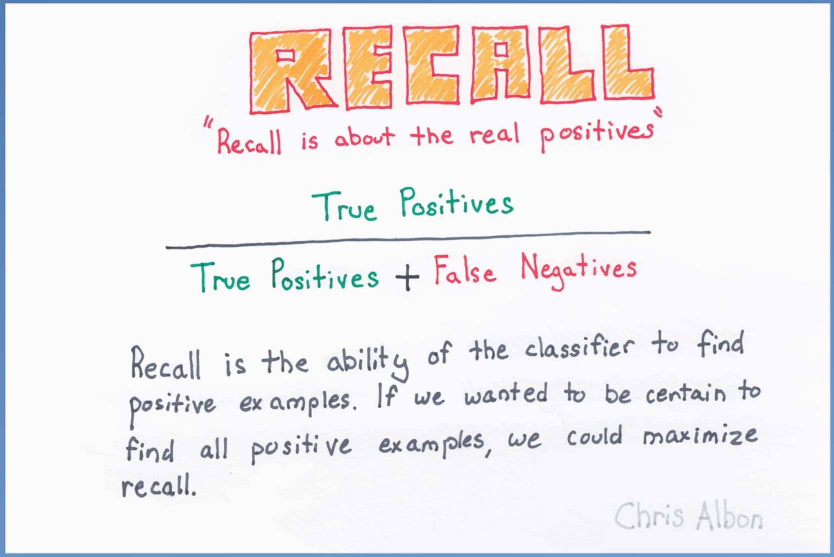
Preliminary conclusion : the model

- Features have been prepared to be treated by a ML algorithm

- Random Forest Classifier is the best performing model among those tested with 90 %accuracy and 84%recall

- Key performance indicator :

  Recall



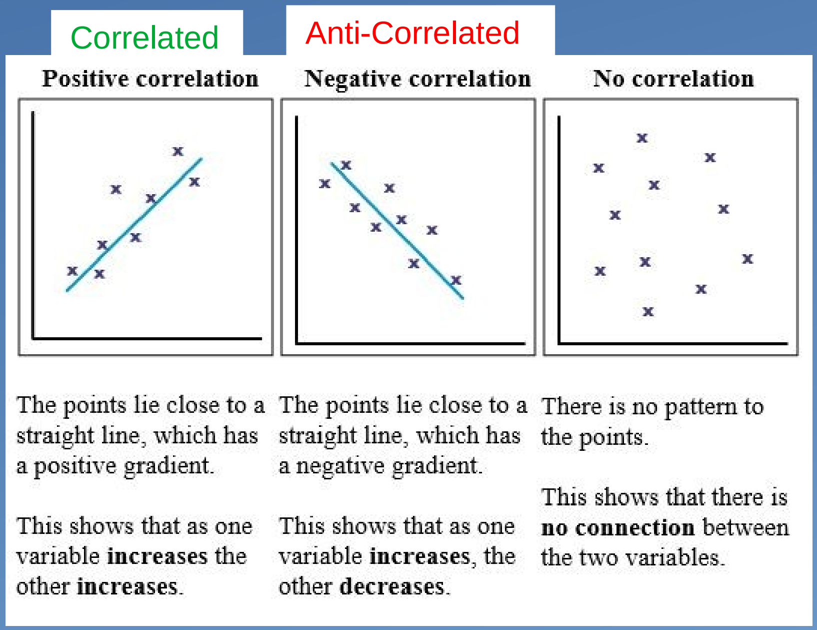
Business Q&A

How does the model respond to business questions?

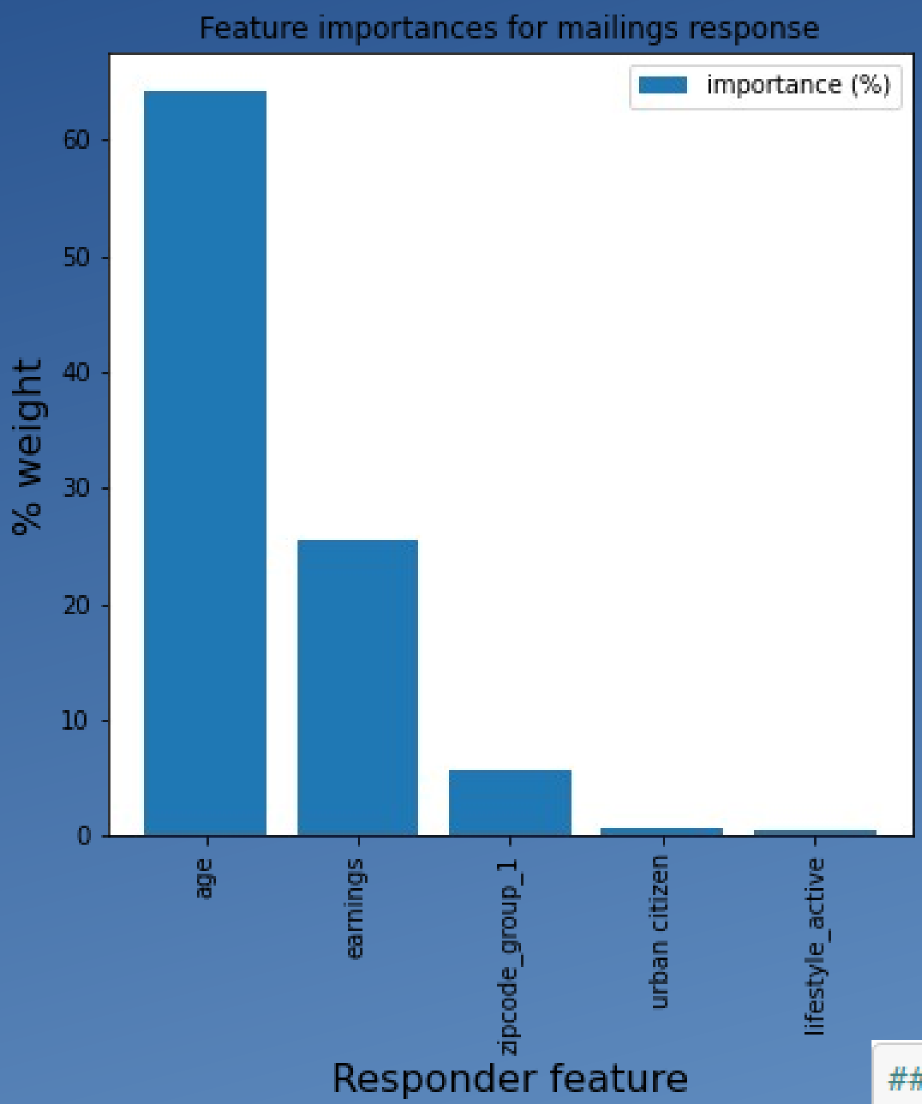
Business questions

- What are the 5 most important variables (resident personal data) for the mailings response prediction ?

- How do those variables correlate to response ?



Important predictors



Major

- resident age

- resident income

- lives in the residential area designed by zipcodes ranging from 10000 to 19999

Minor

- lives in a rural area

- has a healthy lifestyle

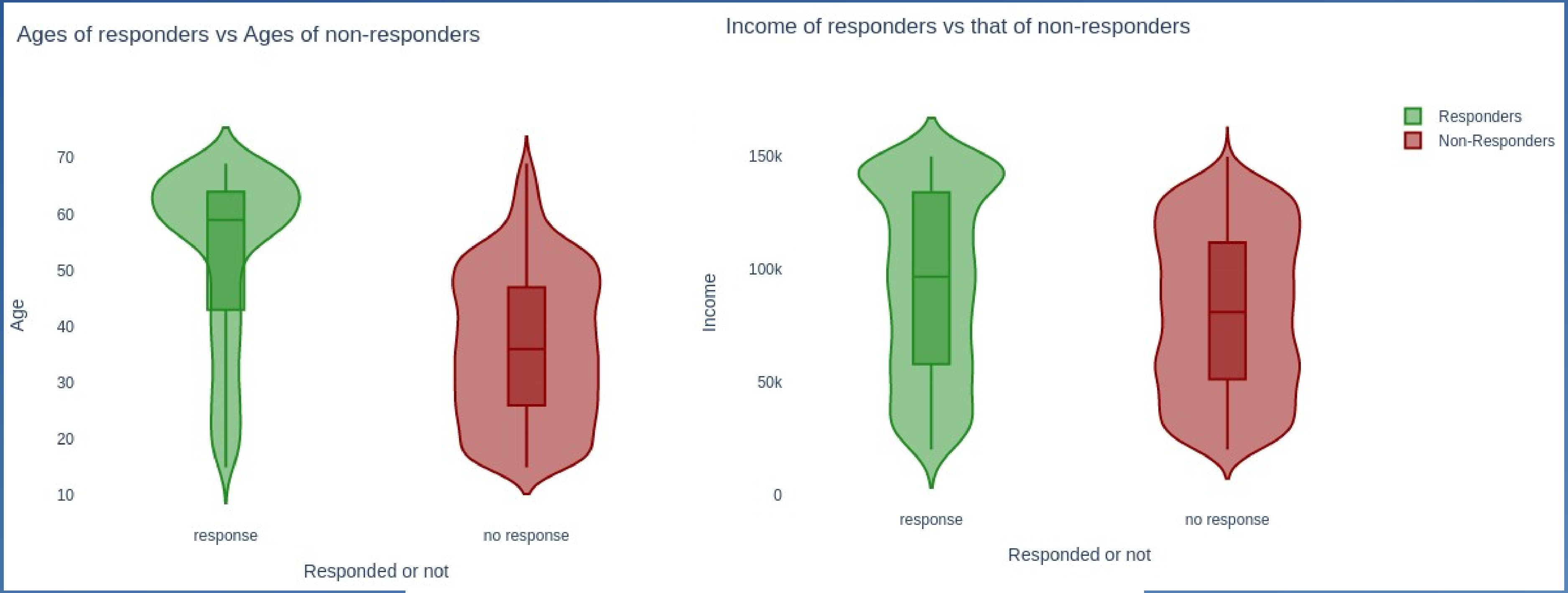
```
## find correlation to response
df_binary[df_binary.columns[0:]].corr()['response'][:]
```

```
zipcode_group_1       0.182073
lifestyle_healthy     0.005076
urban citizen        -0.002543
```

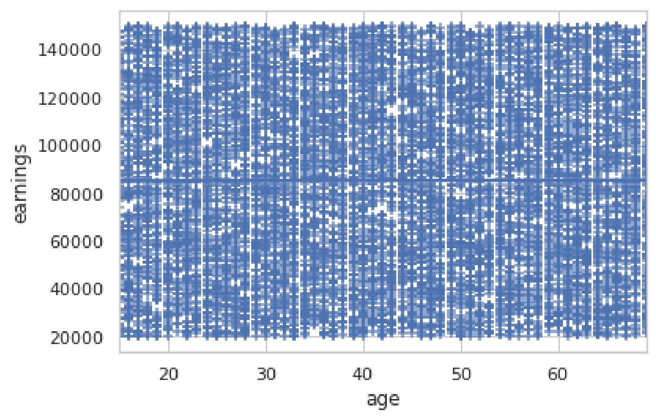
Positive coefficient : correlated

Negative coefficient :anti-correlated

Most reactive groups - age and income



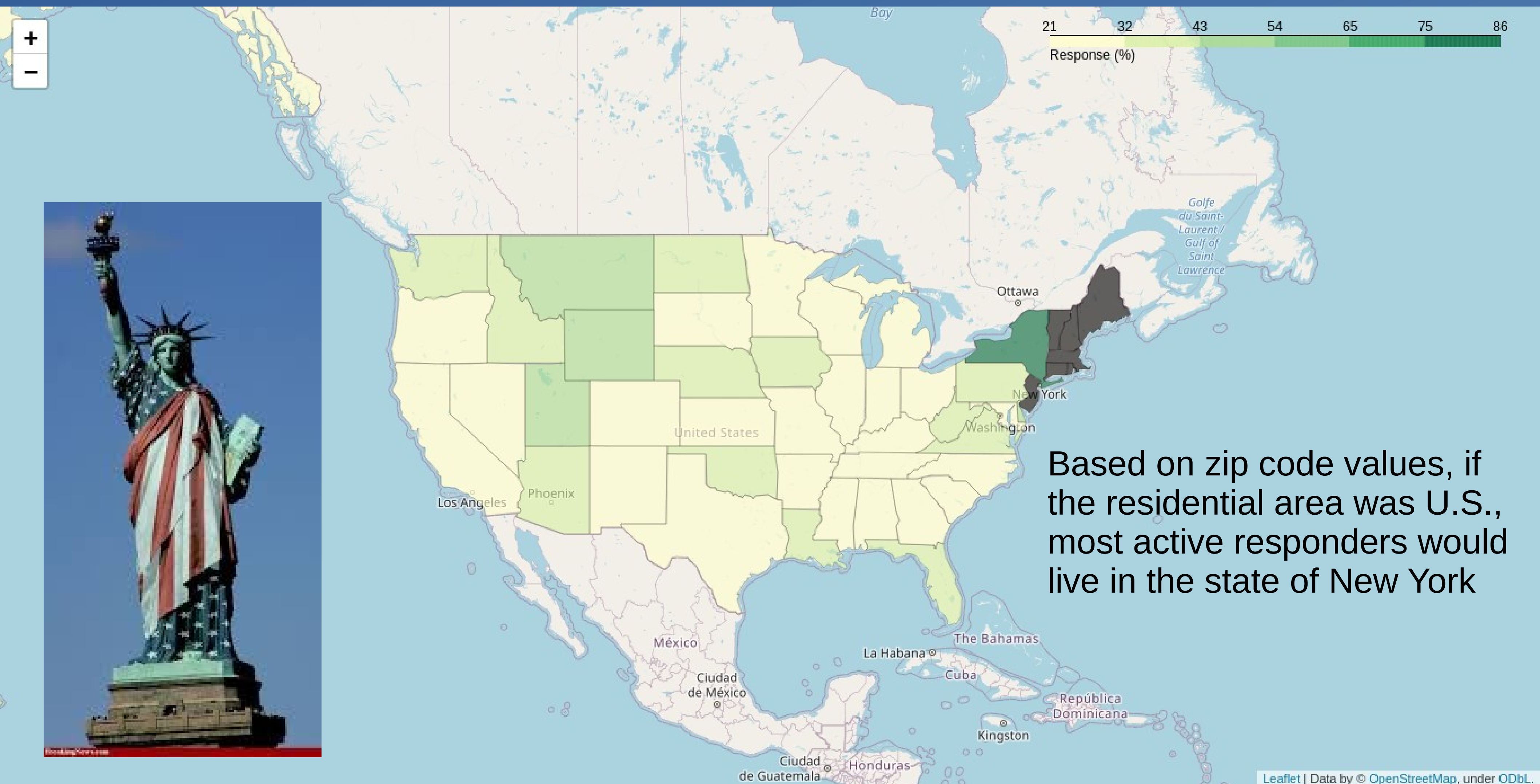
Most responses from older residents (60-70) with high income (150k)



Those two predictors have no correlation

Most reactive groups - geodata

Zip codes predict that most responses come from a specific geographical area



Based on zip code values, if the residential area was U.S., most active responders would live in the state of New York

Conclusions

- The Random Forest Classifier predicts the response with 90 % accuracy and 84 % recall

- The ideal resident with highest response probability should be :

  - 60-70 years old

  - earning 150k / year

  - living in the rural area of a specific region/state

  - having an active lifestyle