Comparison of Building a Regression Model in SAP Analytics Cloud Predictive and Python machine learning

Introduction

This article looks at the create model and train model tasks in SAC Predictive and compares them to a typical Python workflow. It breaks the tasks out into some of the key components and describes how each approach goes about it.

The obvious difference between the two approaches is the trade-off between speed and flexibility; SAC Predictive requires less knowledge and less time to create so is faster, Python is more flexible because you control much more of the generation. Hence particularly the Python approach is just one way of many possibilities.

For the Python IDE I'm using Jupyter Notebooks and running both it and SAC via Google Chrome.

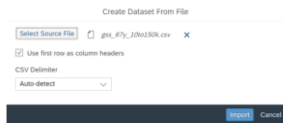
Data Exploration

Reading data

I have a CSV file downloaded from the US General Social Survey (https://gss.norc.org/) containing as the target variable the respondent's income ("realrinc"), and several socio-demographic features as the predictor variables

SAC

We create a dataset to ingest the dataset:



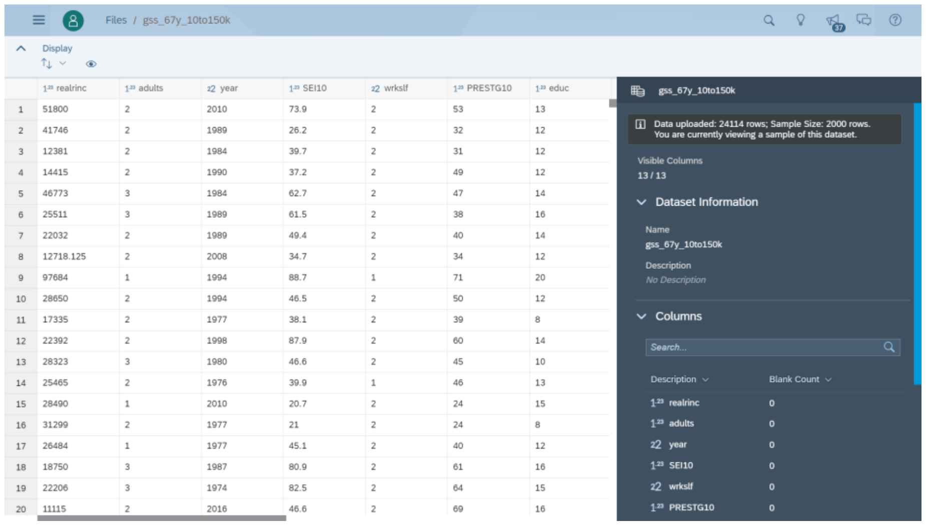
Create Dataset From File

Select Source File gss_67y_10to150k.csv

[✓] Use first row as column headers

CSV Delimiter: Auto-detect

Import Cancel



Files / gss_67y_10to150k

Display

	realrinc	adults	year	SEI10	wrkslf	PRESTG10	educ
1	51800	2	2010	73.9	2	53	13
2	41746	2	1989	26.2	2	32	12
3	12381	2	1984	39.7	2	31	12
4	14415	2	1990	37.2	2	49	12
5	46773	3	1984	62.7	2	47	14
6	25511	3	1989	61.5	2	38	16
7	22032	2	1989	49.4	2	40	14
8	12718.125	2	2008	34.7	2	34	12
9	97684	1	1994	88.7	1	71	20
10	28650	2	1994	46.5	2	50	12
11	17335	2	1977	38.1	2	39	8
12	22392	2	1998	87.9	2	60	14
13	28323	3	1980	46.6	2	45	10
14	25465	2	1976	39.9	1	46	13
15	28490	1	2010	20.7	2	24	15
16	31299	2	1977	21	2	24	8
17	26484	1	1977	45.1	2	40	12
18	18750	3	1987	80.9	2	61	16
19	22206	3	1974	82.5	2	64	15
20	11115	2	2016	46.6	2	69	16

gss_67y_10to150k

Data uploaded: 24114 rows; Sample Size: 2000 rows. You are currently viewing a sample of this dataset.

Visible Columns
13 / 13

Dataset Information

Name: gss_67y_10to150k

Description: *No Description*

Columns

Description	Blank Count
realrinc	0
adults	0
year	0
SEI10	0
wrkslf	0
PRESTG10	0

Python

We can use the pandas library to read in the data to a dataframe:

```
In [6]: import pandas as pd
        csv = pd.read_csv('gss_67y_10to150k.csv')
        ingest = csv.dropna()
        print (type(ingest))
        ingest.head()

        <class 'pandas.core.frame.DataFrame'>
Out[6]:
```

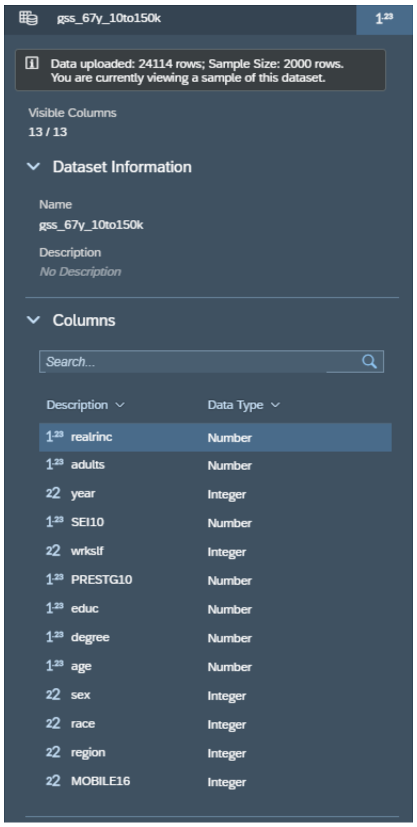
	realrinc	adults	year	SEI10	wrkslf	PRESTG10	educ	degree	age	sex	race	region	MOBILE16
0	11233.75	4.0	2014	61.4	2	55.0	13.0	1.0	53.0	2	1	8	1
1	33075.00	2.0	2012	32.0	2	35.0	11.0	1.0	52.0	1	1	2	2
2	47300.00	2.0	2014	81.0	2	63.0	18.0	4.0	47.0	2	1	9	2
3	18476.25	2.0	2006	74.6	2	46.0	17.0	3.0	32.0	2	1	5	2
4	18459.00	2.0	1983	82.5	2	64.0	16.0	3.0	31.0	2	1	5	3

Summary Statistics

SAC

The dataset display gives us summary information about the whole dataset. If we select a column, information about the distribution of values for that feature, including a histogram, where we can adjust the number of bins via the 'Number of Bars' slider

All Features



gss_67y_10to150k

Data uploaded: 24114 rows; Sample Size: 2000 rows. You are currently viewing a sample of this dataset.

Visible Columns
13 / 13

Dataset Information

Name
gss_67y_10to150k

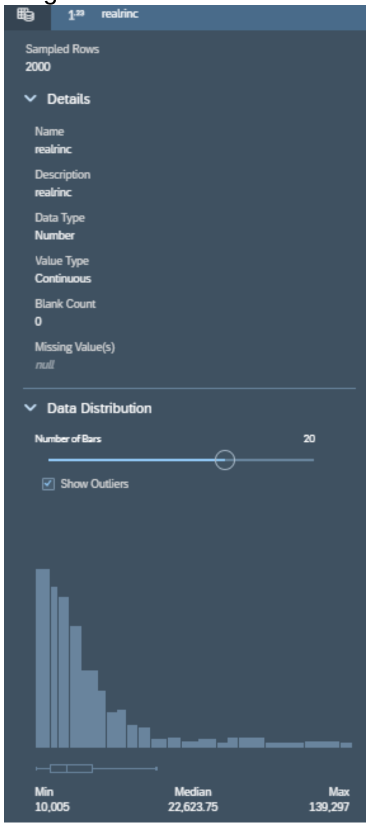
Description
No Description

Columns

Search...

Description	Data Type
realrinc	Number
adults	Number
year	Integer
SEI10	Number
wrkslf	Integer
PRESTG10	Number
educ	Number
degree	Number
age	Number
sex	Integer
race	Integer
region	Integer
MOBILE16	Integer

Single Feature



realrinc

Sampled Rows
2000

Details

Name
realrinc

Description
realrinc

Data Type
Number

Value Type
Continuous

Blank Count
0

Missing Value(s)
null

Data Distribution

Number of Bars 20

Show Outliers

Min	Median	Max
10,005	22,623.75	139,297

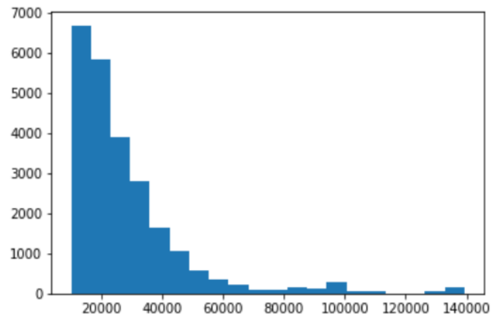
Python

We can use the pandas *describe()* method on a single column or the whole dataframe, as shown here:

```
In [4]: ingest.describe()
Out[4]:
          realrinc        adults          year         SEI10        wrkslf      PRESTG10          educ        degree           age           sex
count  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.000000  24114.00
mean   27818.828925      2.202372   1995.425769     50.308526      1.886622     46.097412     13.833541      1.705068     41.548478      1.407274      1.23
std    19426.521906      0.870984     12.590889     21.562429      0.317061     12.836867      2.879633      1.214014     11.690765      0.491337      0.54
min    10005.000000      1.000000   1974.000000      9.000000      1.000000     16.000000      0.000000      0.000000     18.000000      1.000000      1.00
25%    15668.000000      2.000000   1985.000000     32.000000      2.000000     36.000000     12.000000      1.000000     32.000000      1.000000      1.00
50%    22206.000000      2.000000   1994.000000     46.200000      2.000000     46.000000     14.000000      1.000000     41.000000      1.000000      1.00
75%    32625.000000      2.000000   2006.000000     68.000000      2.000000     54.000000     16.000000      3.000000     51.000000      2.000000      1.00
max   139297.000000      7.000000   2018.000000     93.700000      2.000000     80.000000     20.000000      4.000000     67.000000      2.000000      3.00
```

To show the distribution of a single feature we can write a simple function to display whichever feature we call the function, using the matplotlib library to generate a histogram:

```
In [9]: import matplotlib.pyplot as plt

        def show_hist(feature):
            plt.hist(ingest[feature], bins =20);

        show_hist('realrinc')
```



Training

Split data into training and test set

SAC

SAC uses cross validation rather than splitting data into train and test.

Python

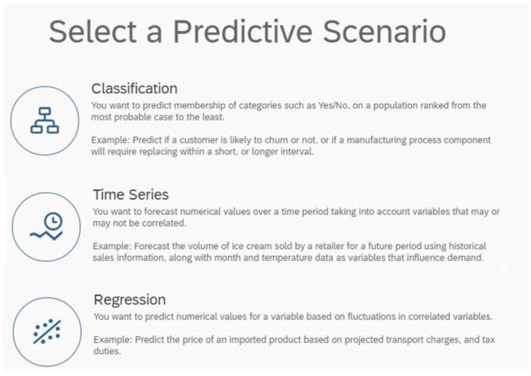
The test data (with known results) is necessary to test the accuracy of our model in order to understand the quality of the model creation choices we make.

```
In [10]: from sklearn.model_selection import train_test_split

         # separate independent and dependent variables
         data = ingest.drop(columns=['realrinc']).astype(int)
         target = ingest['realrinc']

         # create train test split
         X_train, X_test, y_train, y_test = train_test_split(data, target,
                                                             test_size=0.3,
                                                             random_state=42)
```

Algorithm Choice

SAC

With SAC we only need to choose the predicative scenario – Regression (predict a numeric variable), Classification (predict a categorical variable) or Time Series (extrapolate forward in time)



Select a Predictive Scenario

Classification
You want to predict membership of categories such as Yes/No, on a population ranked from the most probable case to the least.

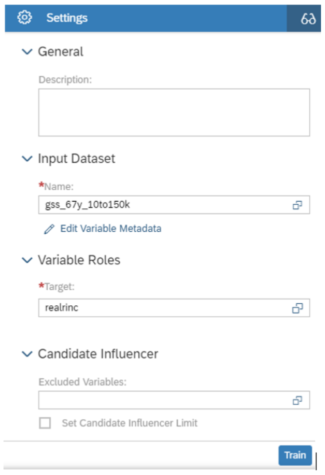
Example: Predict if a customer is likely to churn or not, or if a manufacturing process component will require replacing within a short, or longer interval.

Time Series
You want to forecast numerical values over a time period taking into account variables that may or may not be correlated.

Example: Forecast the volume of ice cream sold by a retailer for a future period using historical sales information, along with month and temperature data as variables that influence demand.

Regression
You want to predict numerical values for a variable based on fluctuations in correlated variables.

Example: Predict the price of an imported product based on projected transport charges, and tax duties.



Settings

General
Description:

Input Dataset
*Name: gss_67y_10to150k
Edit Variable Metadata

Variable Roles
*Target: realrinc

Candidate Influencer
Excluded Variables:
☐ Set Candidate Influencer Limit

Python

We need to understand the capabilities of different algorithms relative to the dataset we have. In this case, as the independent variables do not have strongly linear relationships with the target variable, a linear regression-based algorithm is likely to be ineffective. Decision trees handle non-linear relationships better, but tend to have low predictive accuracy as they learn the specific decision points in the training data too well and have limited generalisation to the test/live data. Ensemble decision tree methods can overcome this, so I have chosen a gradient boosting regressor, which builds many decision trees, each time improving the generalisation capability.

```
In [174]: from sklearn.ensemble import GradientBoostingRegressor

          # create model initially with default parameters
          gb = GradientBoostingRegressor()
```

Model Selection

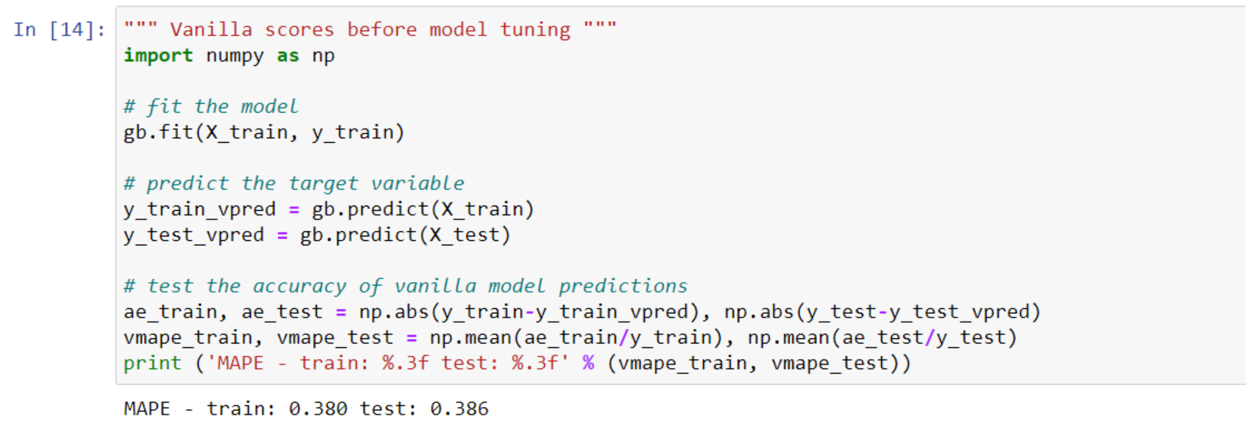
Tuning the parameters for the model

SAC

SAC does automatically all of what is described below.

Python

Check Accuracy Before Tuning



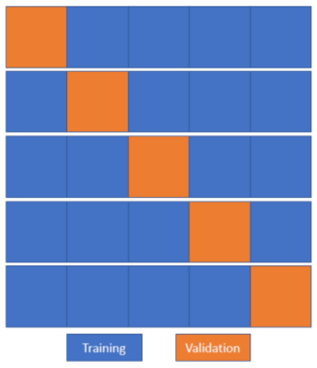
```
In [14]: """ Vanilla scores before model tuning """
         import numpy as np

         # fit the model
         gb.fit(X_train, y_train)

         # predict the target variable
         y_train_vpred = gb.predict(X_train)
         y_test_vpred = gb.predict(X_test)

         # test the accuracy of vanilla model predictions
         ae_train, ae_test = np.abs(y_train-y_train_vpred), np.abs(y_test-y_test_vpred)
         vmape_train, vmape_test = np.mean(ae_train/y_train), np.mean(ae_test/y_test)
         print ('MAPE - train: %.3f test: %.3f' % (vmape_train, vmape_test))

         MAPE - train: 0.380 test: 0.386
```

MAPE = mean absolute percentage error. 38.7% on the test data, relatively high.

Cross Validation to Calculate Optimum Hyper-Parameter Values

Hyper-parameters are the parameters that we give the model (as opposed to the weights and biases the algorithm itself creates. However, to work out the best values for the hyper-parameters we are not just limited to trial and error, there are techniques which can help us, such as the GridSearchCrossValidation algorithm.

Cross validation is a way of splitting the data into several different combinations of say 80:20 (as shown opposite) to run hyper-parameter values on the 80 then validate them on the 20



```
In [18]: from sklearn.model_selection import GridSearchCV

         def cross_val(depth_range, param_range, learn_range, iter_range):

             """ Find the most predictive values for:
             depth_range: max depth, learn_range: learning rate, iter_range: number of iterations,
             param_range: general bucket for multiple parameters.
             Requires values for the four parameters above."""

             # build dictionary of parameter ranges
             param_grid = ({'max_depth':depth_range, 'learning_rate':learn_range, 'n_estimators': iter_range})

             # instantiate GridSearchCrossValidation model
             gs_gb = GridSearchCV(estimator=gb, param_grid=param_grid, cv=5, n_jobs=-1, scoring='neg_mean_absolute_error')

             # fit GridSearch model
             gs_gb = gs_gb.fit(X_train, y_train)

             # instantiate GradientBoostingRegressor model with parameters from GridSearch
             best_model = GradientBoostingRegressor(**gs_gb.best_params_)

             # return the model, the best values from the ranges passed, and the complete set of hyper-parameters
             return best_model, gs_gb.best_params_, gs_gb.best_estimator_

         # call the function passing ranges of values to be tested
         gbr_tuned, params_sel, mod_sel = cross_val(np.arange(5,10), [5,10,15,20,30], [0.05,0.1], [100,400])

         # display the best values from the ranges passed, and the complete set of hyper-parameters
         params_sel, mod_sel
```

```
Out[18]: ({'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 400},
          GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                                    init=None, learning_rate=0.1, loss='ls', max_depth=9,
                                    max_features=None, max_leaf_nodes=None,
                                    min_impurity_decrease=0.0, min_impurity_split=None,
                                    min_samples_leaf=1, min_samples_split=2,
                                    min_weight_fraction_leaf=0.0, n_estimators=400,
                                    n_iter_no_change=None, presort='deprecated',
                                    random_state=None, subsample=1.0, tol=0.0001,
                                    validation_fraction=0.1, verbose=0, warm_start=False))
```

Fit Model

```
In [20]: gbr_tuned.fit(X_train,y_train)
         pass
```

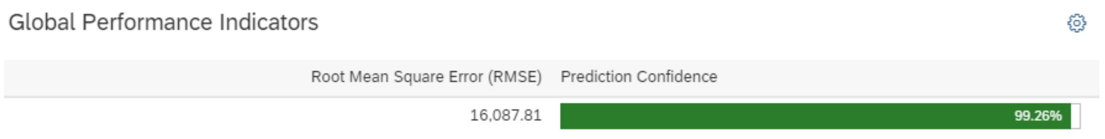
Predict Model

```
In [21]: y_train_pred = gbr_tuned.predict(X_train)
         y_test_pred = gbr_tuned.predict(X_test)
```

Evaluate Predictive Performance

SAC

SAC provides these figures automatically



Global Performance Indicators

Root Mean Square Error (RMSE)	Prediction Confidence
16,087.81	99.26%

Python

Mean absolute percentage error is how I would define confidence interval (subtracting that figure from 100). I've included root mean squared error to mirror SAC and R-squared because it's such a common regression metric.

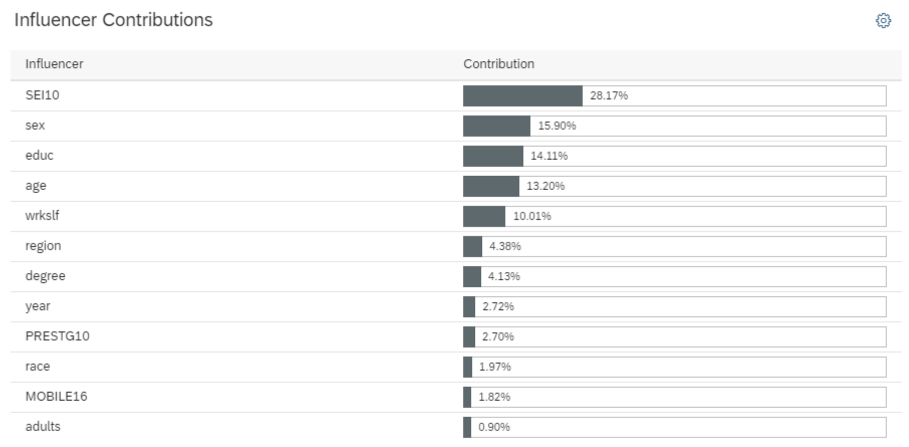
```
In [61]: from sklearn.metrics import mean_squared_error

         def metrics(train, test, train_pred, test_pred):

             """Runs several test for a set of data.
             Requires actual and predicted training (Dataframes) and test (Series)"""

             r2_train, r2_test = r2_score(train, train_pred)*100, r2_score(test, test_pred)*100
             mae_train, mae_test =  np.mean(np.abs(train-train_pred)), np.mean(np.abs(test-test_pred))
             mu_train, mu_test = np.mean(train), np.mean(test)
             ae_train, ae_test = np.abs(train-train_pred), np.abs(test-test_pred)
             mape_train, mape_test = np.mean(ae_train/train)*100, np.mean(ae_test/test)*100
             rmse_train, rmse_test = np.sqrt(np.mean((train-train_pred)**2)),np.sqrt(np.mean((test-test_pred)**2))
             print (\
         'Mean income - train: $%.0f test: $%.0f \n\
         MAE - mean absolute error - train: $%.0f test: $%.0f \n\
         RMSE - root mean squared error - train: $%.0f test: $%.0f \n\
         MAPE - mean absolute percentage error - train: %.2f%% test: %.2f%% \n\
         R^2 - r-squared; the percentage of target explainable by features - train: %.2f%% test: %.2f%%'

         %(mu_train,mu_test, mae_train, mae_test,rmse_train, rmse_test, mape_train, mape_test,r2_train, r2_test))

         metrics(y_train,y_test,y_train_pred,y_test_pred)
```

```
Mean income - train: $27863 test: $27716
MAE - mean absolute error - train: $958 test: $5544
RMSE - root mean squared error - train: $1617 test: $11352
MAPE - mean absolute percentage error - train: 4.27% test: 22.81%
R^2 - r-squared; the percentage of target explainable by features - train: 99.31% test: 65.19%
```

Display Most Predictive Features

SAC

SAC generates this automatically:



Influencer Contributions

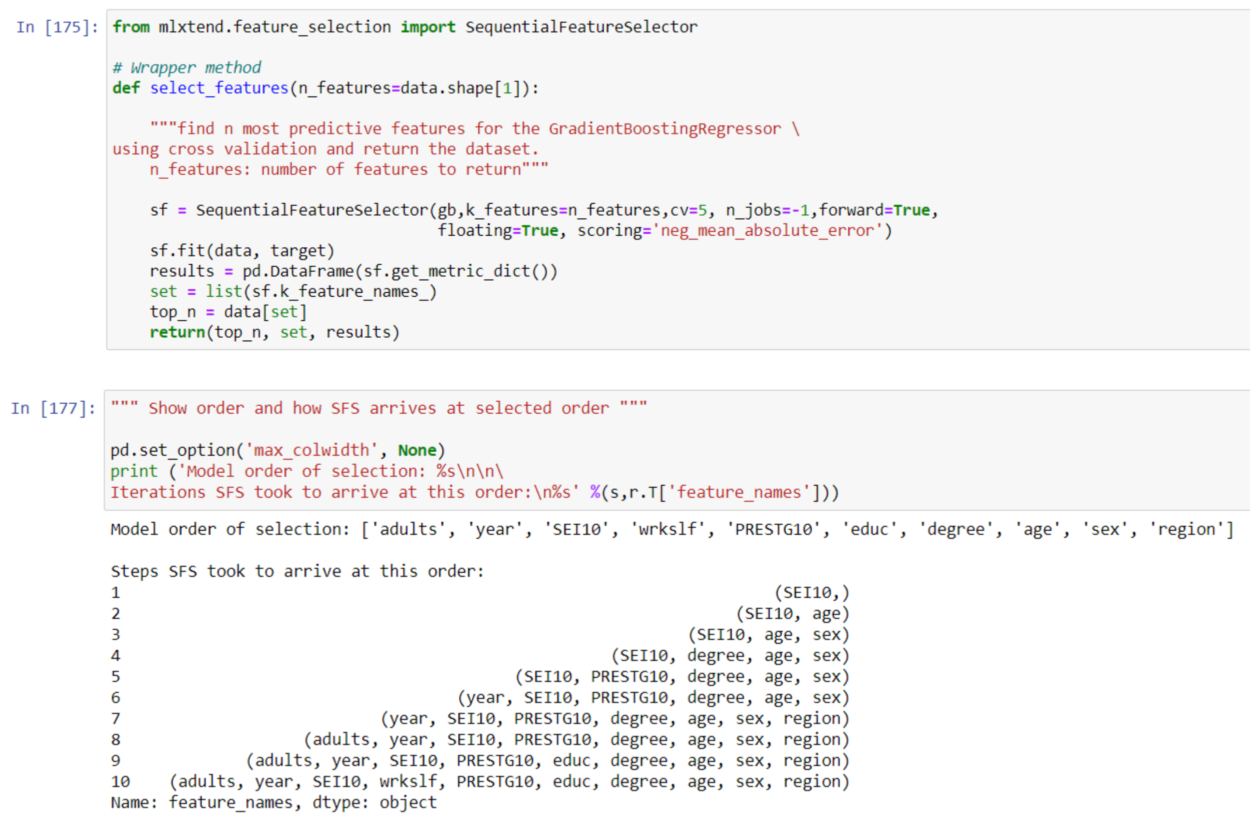
Influencer	Contribution
SEI10	28.17%
sex	15.90%
educ	14.11%
age	13.20%
wrkslf	10.01%
region	4.38%
degree	4.13%
year	2.72%
PRESTG10	2.70%
race	1.97%
MOBILE16	1.82%
adults	0.90%

Python

Feature selection algorithms are really intended to reduce the features used in a model, in order to:

- Discard uninformative features
- Discard deceptive features
- Speed training/testing

However we can also use them to rank predictive influence:



```
In [175]: from mlxtend.feature_selection import SequentialFeatureSelector

          # Wrapper method
          def select_features(n_features=data.shape[1]):

              """find n most predictive features for the GradientBoostingRegressor \
          using cross validation and return the dataset.
              n_features: number of features to return"""

              sf = SequentialFeatureSelector(gb,k_features=n_features,cv=5, n_jobs=-1,forward=True,
                                              floating=True, scoring='neg_mean_absolute_error')
              sf.fit(data, target)
              results = pd.DataFrame(sf.get_metric_dict())
              set = list(sf.k_feature_names_)
              top_n = data[set]
              return(top_n, set, results)
```

```
In [177]: """ Show order and how SFS arrives at selected order """

          pd.set_option('max_colwidth', None)
          print ('Model order of selection: %s\n\n\
          Iterations SFS took to arrive at this order:\n%s' %(s,r.T['feature_names']))

          Model order of selection: ['adults', 'year', 'SEI10', 'wrkslf', 'PRESTG10', 'educ', 'degree', 'age', 'sex', 'region']

          Steps SFS took to arrive at this order:
          1                                                                   (SEI10,)
          2                                                               (SEI10, age)
          3                                                          (SEI10, age, sex)
          4                                                  (SEI10, degree, age, sex)
          5                                        (SEI10, PRESTG10, degree, age, sex)
          6                                  (year, SEI10, PRESTG10, degree, age, sex)
          7                          (year, SEI10, PRESTG10, degree, age, sex, region)
          8                  (adults, year, SEI10, PRESTG10, degree, age, sex, region)
          9            (adults, year, SEI10, PRESTG10, educ, degree, age, sex, region)
          10   (adults, year, SEI10, wrkslf, PRESTG10, educ, degree, age, sex, region)
          Name: feature_names, dtype: object
```

Next Steps

The SAC and Python results diverge from each other across all steps. In a future blog I'll apply the models to unseen data and (albeit with known results) in order to look at and compare the predictions each approach makes on a sample by sample basis.

Angus Menter, BI Practice Manager, DSCallards Ltd