Institute and Faculty of Actuaries

Workshop B2: A dabbler's introduction to Python

Matt Evans, EMC Actuarial and Analytics

Institute
and Faculty
of Actuaries

A dabbler's introduction to Python

Tools, Language and Applications

Agenda

- Welcome dabblers!
- The problem with R
- Python dabbling
- Actuarial applications
- Conclusions

Institute and Faculty of Actuaries

The problem with R

- This presentation not intended as part of that phoney war, but…
  - R has a steep learning curve, don't believe the hype
  - R data structures are not intuitive if you use them infrequently
  - Preferred packages changed in six months (now better, but still need re-learning)
  - Each new algorithm has different package and syntax

- Could Python be the answer? The fanboys say yes.

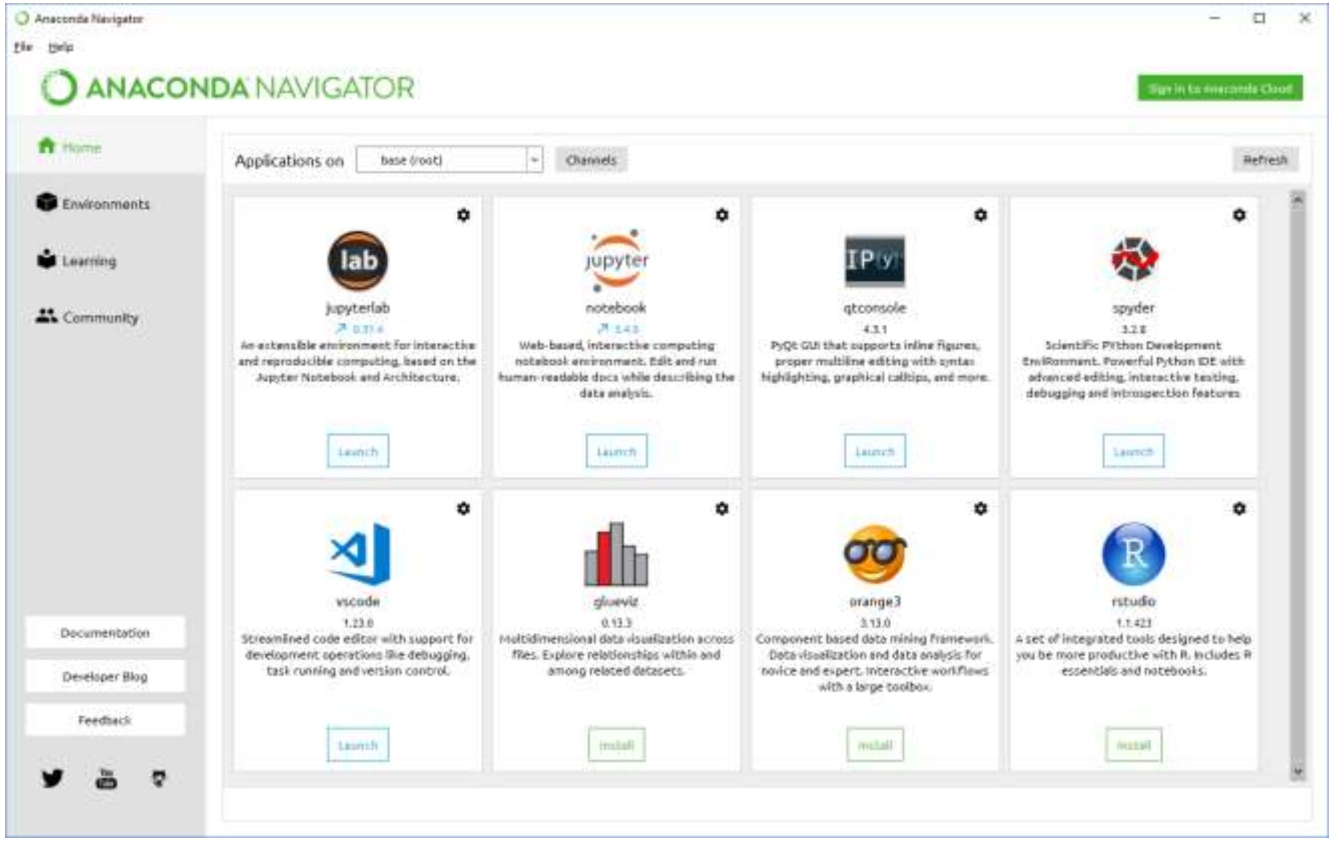
Python dabbling

- Tools:
  - Anaconda for installing Spyder and Python packages
  - Spyder for editing code

- Trying out the language and packages:
  - Some data generation
  - Popular packages: a GLM fit and XGBoost

Python dabbling: Tools

- Anaconda
  - Installs Spyder
  - Jupyter notebook
  - Rstudio too
- Installs Python packages
  - Most need only a reference but…
  - Some packages (e.g. XGBoost) need separate installation, made easier in Anaconda

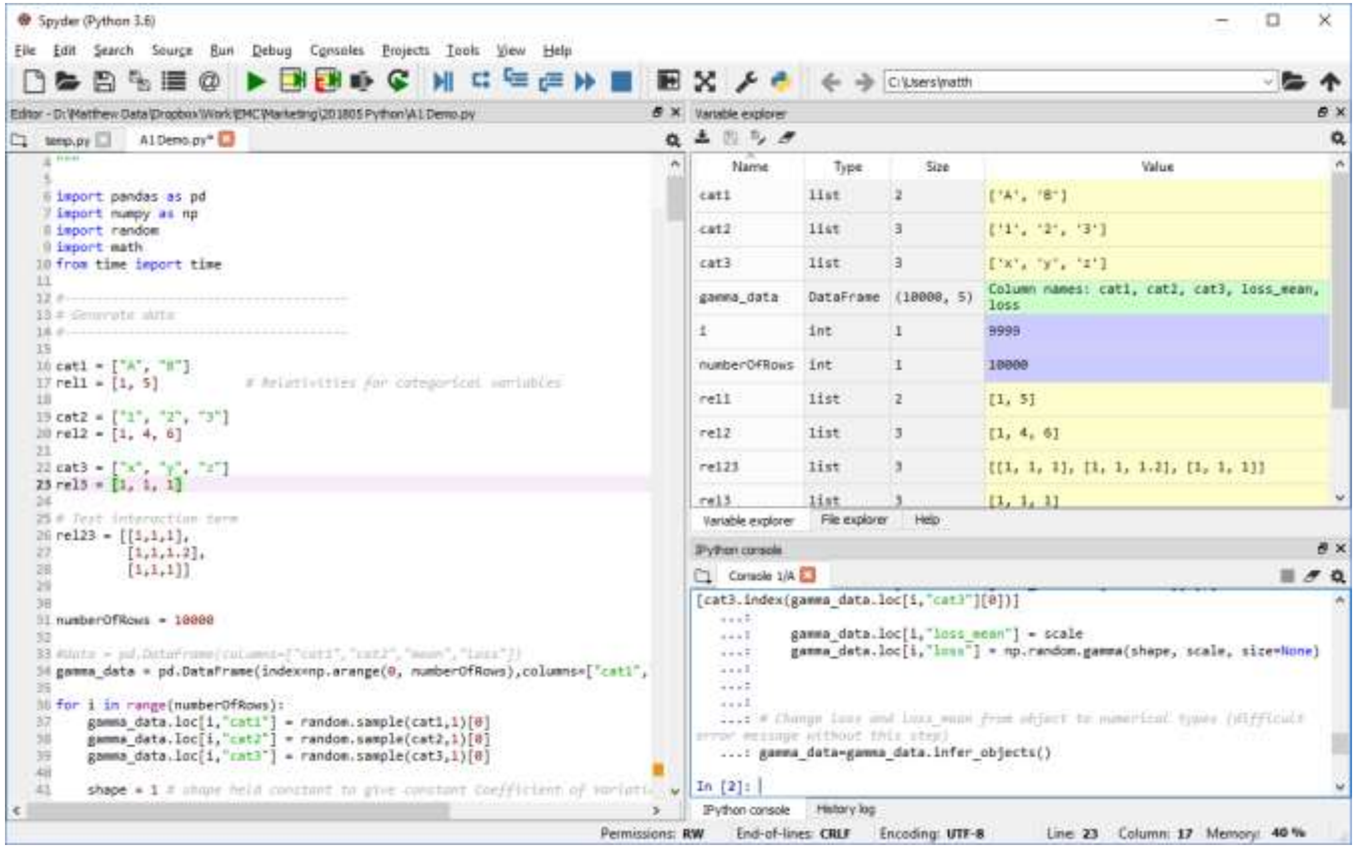


- Anaconda something like a "standard" configuration

Institute and Faculty of Actuaries

Python dabbling: Tools

- Spyder
  - Looks a lot like other modern development environments
  - Run and test code
  - Get code hints
  - Examine variables
  - Show output and graphics



Institute and Faculty of Actuaries

Python dabbling: data generation

- This dummy data is designed to be like a claims severity dataset
  - Three factors and a loss severity
  - Factor levels each have a relativity attached, giving expected mean loss
  - Actual loss is simulated with a gamma distribution around the mean
- Distribution *perfect* for fitting a gamma GLM
- How hard is it to do?

gamma_data - DataFrame

Index	cat1	cat2	cat3	loss_mean	loss
0	A	2	y	4	0.707742
1	B	1	x	5	0.0863207
2	A	3	x	6	0.589577
3	A	2	z	4.8	2.71553
4	B	3	z	30	14.9231
5	A	1	x	1	1.14697
6	B	3	x	30	8.08131
7	A	1	x	1	4.0253
8	B	1	y	5	8.23561
9	A	3	x	6	0.720686
10	A	3	x	6	9.19783
11	B	3	z	30	68.6866
12	A	3	z	6	8.02906
13	B	2	y	20	0.317313
14	B	1	z	5	5.4874
15	A	1	y	1	0.215856
16	A	3	z	6	1.20233

Format Resize ☑ Background color ☑ Column min/max OK Cancel

Institute and Faculty of Actuaries

Python dabbling: data generation

- The code doesn't look too bad…
  - Setting up arrays
  - Introducing matrix for interaction
  - DataFrame
  - Sampling from arrays of factor levels
  - Calculating gamma parameters
  - Simulating gamma loss
  - Inferring object types*

```python

#-------------------------------------
# Generate data
#-------------------------------------

cat1 = ["A", "B"]        # Dataframes, lists and "tuples" not so intuitive
rel1 = [1, 5]            # Relativities for categorical variables

cat2 = ["1", "2", "3"]
rel2 = [1, 4, 6]

cat3 = ["x", "y", "z"]
rel3 = [1, 1, 1]

# Test interaction term
rel23 = [[1,1,1],
         [1,1,1.2],
         [1,1,1]]


numberOfRows = 10000

#data = pd.DataFrame(columns=["cat1","cat2","mean","loss"])
gamma_data = pd.DataFrame(index=np.arange(0, numberOfRows),columns=["cat1","cat2","cat3", "loss_mean","loss"])

for i in range(numberOfRows):
    gamma_data.loc[i,"cat1"] = random.sample(cat1,1)[0]
    gamma_data.loc[i,"cat2"] = random.sample(cat2,1)[0]
    gamma_data.loc[i,"cat3"] = random.sample(cat3,1)[0]

    shape = 1 # shape held constant to give constant Coefficient of Variation

    scale = rel1[cat1.index(gamma_data.loc[i,"cat1"][0])] * \
            rel2[cat2.index(gamma_data.loc[i,"cat2"][0])] * \
            rel3[cat3.index(gamma_data.loc[i,"cat3"][0])] * \
            rel23[cat2.index(gamma_data.loc[i,"cat2"][0])][cat3.index(gamma_data.loc[i,"cat3"][0])]

    gamma_data.loc[i,"loss_mean"] = scale
    gamma_data.loc[i,"loss"] = np.random.gamma(shape, scale, size=None)


# Change loss and loss_mean from object to numerical types (difficult error message without this step)
gamma_data=gamma_data.infer_objects()

```

See impact of interaction term on A:2:z value on previous slide

No end of For loop in Python.

Python dabbling: data generation VERDICT

- Some problems setting up but actually these were minor
- DataFrames in Python still use occasionally confusing notation (for the dabbler) but R is probably harder
- Code noticeably more object-like
- An easy transition if you are used to VBA

Python dabbling: Popular packages

- Sci-kit learn, the data scientist's favourite Python package
  - Lots of algorithms all in one consistent package so you can switch easily
  - Regression and Classification algorithms abound
- Statsmodels package, for models closer to R
  - Proper stats with p-values
- XGBoost, Kaggle champion
  - Regression and Classification applications
  - GridSearch - do we need to know what we're doing any more?

Python dabbling: a GLM fit

- The Sci-kit learn package has a model called *Generalized Linear Model…*
  - But it is only a *linear* model… no link function, not a proper GLM..!
- The Statsmodels package does a proper GLM
  - Code does *two* fits
  - One with interaction; one without

```
#-----------------------------------
# Fit Gamma GLM (Statsmodels)
#-----------------------------------

# Log link function for multiplicative relativities

import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.genmod.families.links as llink

formula='loss ~ cat1 + cat2 + cat3'

mod = smf.glm(formula=formula, data=gamma_data_train, family=sm.families.Gamma(link=llink.log))
res = mod.fit()
print(res.summary())

print('Gamma GLM simple form Relativities - exponentiated parameters')
print(np.exp(res.params))

print("GAMMA GLM simple RMSE: %.2f"
      % math.sqrt(np.mean((res.predict(gamma_data_test[['cat1','cat2','cat3']]) - gamma_data_test['loss']) ** 2)))

formula='loss ~ cat1 + cat2 + cat3 + cat2*cat3'

mod = smf.glm(formula=formula, data=gamma_data_train, family=sm.families.Gamma(link=llink.log))
res = mod.fit()
print(res.summary())

print('Gamma GLM interaction form Relativities - exponentiated parameters')
print(np.exp(res.params))

print("GAMMA GLM interaction RMSE: %.2f"
      % math.sqrt(np.mean((res.predict(gamma_data_test[['cat1','cat2','cat3']]) - gamma_data_test['loss']) ** 2)))
```

Interaction not allowed for. So more like modelling in the real world where we don't have full knowledge of risk.

PERFECT INFORMATION

Institute and Faculty of Actuaries

Python dabbling: a GLM fit – did it work?

- Our simple model does OK, even though it doesn't "know" about the interaction
- Relativities quite close, with the interaction load falling into cat2[T.2]

Original Relativities

Cat1	Relativity
A	1.0
B	5.0

Cat3	Relativity
x	1.0
y	1.0
z	1.0

Cat2	Relativity
1	1.0
2	4.0
3	6.0

Cat2:Cat3	Cat3		
Cat2	x	y	z
1	1.0	1.0	1.0
2	1.0	1.0	1.2
3	1.0	1.0	1.0

- Note the RMSE on 25% hold out sample
  - RMSE 15.66

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                   loss   No. Observations:                 7500
Model:                            GLM   Df Residuals:                     7494
Model Family:                   Gamma   Df Model:                            5
Link Function:                    log   Scale:              1.0151358144328344
Method:                          IRLS   Log-Likelihood:                -21556.
Date:                Thu, 31 May 2018   Deviance:                       8713.5
Time:                        22:15:36   Pearson chi2:                 7.61e+03
No. Iterations:                     7
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     -0.0287      0.028     -1.010      0.313      -0.084       0.027
cat1[T.B]      1.6099      0.023     69.154      0.000       1.564       1.655
cat2[T.2]      1.4591      0.029     51.123      0.000       1.403       1.515
cat2[T.3]      1.7977      0.029     62.908      0.000       1.742       1.854
cat3[T.y]     -0.0026      0.028     -0.091      0.928      -0.058       0.053
cat3[T.z]      0.0380      0.029      1.323      0.186      -0.018       0.094
==============================================================================
Gamma GLM simple form Relativities - exponentiated parameters
Intercept     0.971749
cat1[T.B]     5.002123
cat2[T.2]     4.301979
cat2[T.3]     6.035686
cat3[T.y]     0.997431
cat3[T.z]     1.038735
dtype: float64
GAMMA GLM simple RMSE: 15.66
```

This would be ~4 without the interaction term.

Institute and Faculty of Actuaries

Python dabbling: a GLM fit – did it work?

- Our “perfect information” model does better – it fits interaction
- Relativities quite close, even the interaction loading

Original Relativities

Cat1	Relativity
A	1.0
B	5.0

Cat3	Relativity
x	1.0
y	1.0
z	1.0

Cat2	Relativity
1	1.0
2	4.0
3	6.0

Cat2:Cat3	Cat3		
Cat2	x	y	z
1	1.0	1.0	1.0
2	1.0	1.0	1.2
3	1.0	1.0	1.0

- Note the RMSE is about 0.1% reduced (not a lot!)
  - RMSE 15.64

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                   loss   No. Observations:                 7500
Model:                            GLM   Df Residuals:                     7490
Model Family:                   Gamma   Df Model:                            9
Link Function:                    log   Scale:              1.0135052395413193
Method:                          IRLS   Log-Likelihood:                -21550.
Date:                Thu, 31 May 2018   Deviance:                       8701.5
Time:                        22:15:38   Pearson chi2:                 7.59e+03
No. Iterations:                     7
=========================================================================================
                            coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------
Intercept                -0.0136      0.037     -0.369      0.712      -0.086       0.058
cat1[T.B]                 1.6093      0.023     69.159      0.000       1.564       1.655
cat2[T.2]                 1.3854      0.049     28.493      0.000       1.290       1.481
cat2[T.3]                 1.8303      0.050     36.555      0.000       1.732       1.928
cat3[T.y]                 0.0003      0.049      0.007      0.994      -0.096       0.097
cat3[T.z]                -0.0126      0.050     -0.252      0.801      -0.111       0.086
cat2[T.2]:cat3[T.y]       0.0448      0.069      0.649      0.516      -0.091       0.180
cat2[T.3]:cat3[T.y]      -0.0572      0.070     -0.822      0.411      -0.193       0.079
cat2[T.2]:cat3[T.z]       0.1815      0.070      2.588      0.010       0.044       0.319
cat2[T.3]:cat3[T.z]      -0.0365      0.071     -0.516      0.606      -0.175       0.102
=========================================================================================
Gamma GLM interaction form Relativities - exponentiated parameters
Intercept              0.986538
cat1[T.B]              4.999344
cat2[T.2]              3.996425
cat2[T.3]              6.235984
cat3[T.y]              1.000347
cat3[T.z]              0.987455
cat2[T.2]:cat3[T.y]    1.045852
cat2[T.3]:cat3[T.y]    0.944436
cat2[T.2]:cat3[T.z]    1.199027
cat2[T.3]:cat3[T.z]    0.964129
dtype: float64
GAMMA GLM interaction RMSE: 15.64
```

Interaction load was set at 1.2. We are close here.

Python dabbling: XGBoost

- This regression and classification algorithm uses gradient boosted trees
- It wins a lot of competitions on Kaggle, the data science website, and doesn't do too badly here either
- RMSE on 25% hold out sample:
  - XGBOOST RMSE: 15.65
  - This is *better* than the typical real world GLM model with imperfect information, without knowing *anything*
- A simple test with dummy data only, but encouraging, alongside anecdotal evidence*

```python
#-----------------------------------
# Fit XGBoost
#-----------------------------------
import xgboost
from sklearn.grid_search import GridSearchCV

xgb = xgboost.XGBRegressor(n_estimators=900,
                           learning_rate=0.1,
                           gamma=0.15,
                           subsample=0.75,
                           colsample_bytree=0.7,
                           max_depth=4)

xgb.fit(one_hot_train, gamma_data_train[['loss']]) # X and y

print("XGBOOST RMSE: %.2f"
      % math.sqrt(np.mean((xgb.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))

# Grid search
parameters = {'nthread':[1],
              'n_estimators': [910,920], #number of trees, change it to 1000 for better results
              'learning_rate': [0.10,0.15], #so called `eta` value
              'max_depth': [4,5,6,7],
              'gamma': [0.1,0.15],
              'subsample': [0.75],
              'colsample_bytree':  [0.5,0.6,0.7],
              'seed': [1337]}

reg = GridSearchCV(xgb,
                   parameters,
                   n_jobs=10,
                   verbose=0,
                   cv=10,
                   scoring='neg_mean_squared_error',
                   refit=True)

reg.fit(one_hot_train, gamma_data_train[['loss']]) # X and y

print("XGBOOST RMSE: %.2f"
      % math.sqrt(np.mean((reg.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))
```

GridSearch uses cross-validation to optimise parameters, offering route to further improvement

Python dabbling: Popular packages VERDICT

- Sci-kit learn a bit disappointing as tests with a wide range of regressors failed to get near GLM result, however…
  - Cross validation and train/test facilities are useful
  - Many regressors can be tried out and trained with GridSearch, which should improve results

- Statsmodels works very well
  - Output as good as R
  - Fast to run
  - Model formulae standard and intuitive

```
#-------------------------------------
# Fit Model (Scikit-Learn)
#-------------------------------------

reg = linear_model.BayesianRidge()
reg.fit(one_hot_train, gamma_data_train['loss'])

print("Scikit Model 2 RMSE: %.2f"
      % math.sqrt(np.mean((reg.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))

#-------------------------------------
# Fit Model (Scikit-Learn)
#-------------------------------------

reg = linear_model.ElasticNet()
reg.fit(one_hot_train, gamma_data_train['loss'])

print("Scikit Model 3 RMSE: %.2f"
      % math.sqrt(np.mean((reg.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))

#-------------------------------------
# Fit Model (Scikit-Learn)
#-------------------------------------

reg = linear_model.Lasso()
reg.fit(one_hot_train, gamma_data_train['loss'])

print("Scikit Model 4 RMSE: %.2f"
      % math.sqrt(np.mean((reg.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))

#-------------------------------------
# Fit Model (Scikit-Learn)
#-------------------------------------

reg = linear_model.Lars(n_nonzero_coefs=1)
reg.fit(one_hot_train, gamma_data_train['loss'])

print("Scikit Model 5 RMSE: %.2f"
      % math.sqrt(np.mean((reg.predict(one_hot_test) - gamma_data_test['loss']) ** 2)))
```

Python dabbling: Popular packages VERDICT

- XGBoost
  - On our admittedly simple test, this algorithm gets very close to a GLM with perfect information
    - Full GLM knows structure *and* error distribution
    - XGBoost uses only cross-validation to improve fit
  - In the real world our data is full of hidden interactions and effects… could we get closer with this?
  - Could this approach work well where…:
    - We have little data?
    - We are not resourced to do full GLM modelling?
    - We suspect non-linear characteristics?
    - We are less worried about a “black-box”?

Model	RMSE
GLM without interaction term	15.66
GLM with interaction term	15.64
XGBoost	15.65
Best Sci-kit (1st pass only)	16+

Institute and Faculty of Actuaries

Actuarial applications

- Data wrangling
- Pricing with GLM, GAM, XGBoost and various others
- Pricing using curve fitting and simulation
- Reserving with ChainLadder* package now ported from R to Python
- Capital Modelling
- Reporting and Graphics

Strengths and weaknesses

- Python is from world of IT:
  - Is proper programming, and can be deployed easily in modern IT structures
  - Has a whole load of other stuff in it, not just stats
  - Is supported by wider pool of talent
  - But is weaker as not so statistical (..?)

- R is from world of academic stats
  - Closer to SAS and SPSS
  - Proper statistics with p-values!
  - Rich library of packages
  - Good at data handling
  - But harder to learn if you’re used to VBA
  - And more difficult to deploy (..?)

Conclusions

- On balance, I will likely look to Python first for new projects
  - Data handling and syntax are more intuitive for this VBA-soaked actuary
  - A friendlier introduction to the new data science tools available
  - Python is used in QGIS, open source mapping software

- Others may feel differently…
  - Do you have a lot of SAS experience in your team?
  - Would your team emphasise “proper” statistics?
  - GLM/GAM are more established in R

Institute and Faculty of Actuaries

EMC Actuarial & Analytics
About Us

- Peter England
  - Capital
  - Reserving
  - IFRS 17
  - Stochastic/statistical modelling
  - Research
- peter@emc-actuarial.com

- Matthew Evans
  - Pricing
  - Reserving
  - Data Science
  - InsurTech
- matthew@emc-actuarial.com

Institute and Faculty of Actuaries

References

- Anaconda installer https://www.anaconda.com/download/
  - Spyder code editor and console
  - Jupiter notebooks
- Statsmodels https://www.statsmodels.org/stable/index.html
- Sci-kit learn http://scikit-learn.org/stable/
- Kaggle competitions https://www.kaggle.com/
- XGBoost Champion regression algorithm
- MatPlotLib Python graphing

Institute and Faculty of Actuaries

Questions

Comments

The views expressed in this [publication/presentation] are those of invited contributors and not necessarily those of the IFoA. The IFoA do not endorse any of the views stated, nor any claims or representations made in this [publication/presentation] and accept no responsibility or liability to any person for loss or damage suffered as a consequence of their placing reliance upon any view, claim or representation made in this [publication/presentation].

The information and expressions of opinion contained in this publication are not intended to be a comprehensive study, nor to provide actuarial advice or advice of any nature and should not be treated as a substitute for specific advice concerning individual situations. On no account may any part of this [publication/presentation] be reproduced without the written permission of the IFoA [*or authors, in the case of non-IFoA research*].

Institute and Faculty of Actuaries