Executive summary

This file aims to explain what ProdPresRegression.ipynb codes execute. In real oil production fields, in order to maintain reservoir pressure water injection wells are used. Depending on the number of producers and chosen strategy, the number of injectors can also vary. Additionally, once put on production & injection, the injection rates can be controlled and require efficient target setting. Lower injection rates mean lower pressures and production, while higher injection can cause fractures in the reservoir leading to early water breakthrough or even safety risks.

These codes take in production and injection rates along with reservoir pressure (2 dataframes), convert them cumulative rates and join them to sparse reservoir pressure dataframe. To achieve this, the original codes were built with multiple classes. Classes inherit smaller sub-classes for adding scenarios, converting dataframes and building models (Cases, Cumulatives and RegressionModels classes, respectively), thus they can be developed separately.

We are not always sure whether some injectors support the given producers, so codes will ask different assumptions – well pairs and using their production data, reservoir pressure points will matched through linear regression. You can switch off and on the wells by adding string of “0” and “1”s. The string length should be equal to the number of wells and contain zeros and ones. Example dataset bases on real data and several assumptions (e.g. constant compressibility of fluids) were made.

Running the codes

1. Initialize the ProdPresRegression, add dataframes and create RegressionModels() class.

```
%run ProdPresRegression.ipynb #import codes.
```

```
prod = pd.read_csv('Production Data - 2.csv') #read production and pressure dataframes.
pres = pd.read_csv('Pressure Data - 2.csv')
```

```
T = RegressionModels(prod, pres, ['P1', 'P2'], ['I1', 'I2', 'I3']) #add well names as given in Production dataframe
```

2. Follow instructions to add cases. Note DuplicateError and InputError’s. Example scenarios are given for convenience.

```
T.create_scenarios()
#example scenarios
#11111, 11011, 11101, 11110,
#10111, 10011, 10101, 10110
#01111, 01011, 01101, 01110
```

```
Provide a string of 0s and 1s, to include or exclude wells in this sequence.
For example: if the wells list is [P1 P2 I1 I2 I3], you can add 10101 to consider P1, I1 and I3 wells in the Regression.

Add new case for the following well list: ['P1', 'P2', 'I1', 'I2', 'I3'], or input "end" to finalize.
String length should be 5
Case 0: 11111
Case 0 added.

Case 1: 11011
Case 1 added.

Case 2: 11011
DuplicteError! This case has already been added.

Case 2: 1ads
InputError! Only 0 and 1s can be added. Length of the string should be 5


Case 2:
```

3. After adding scenarios, insert “end”. By invoking “.fit_models()”, models will be matched and report on match statistics, intercepts & coefficient will be provided.

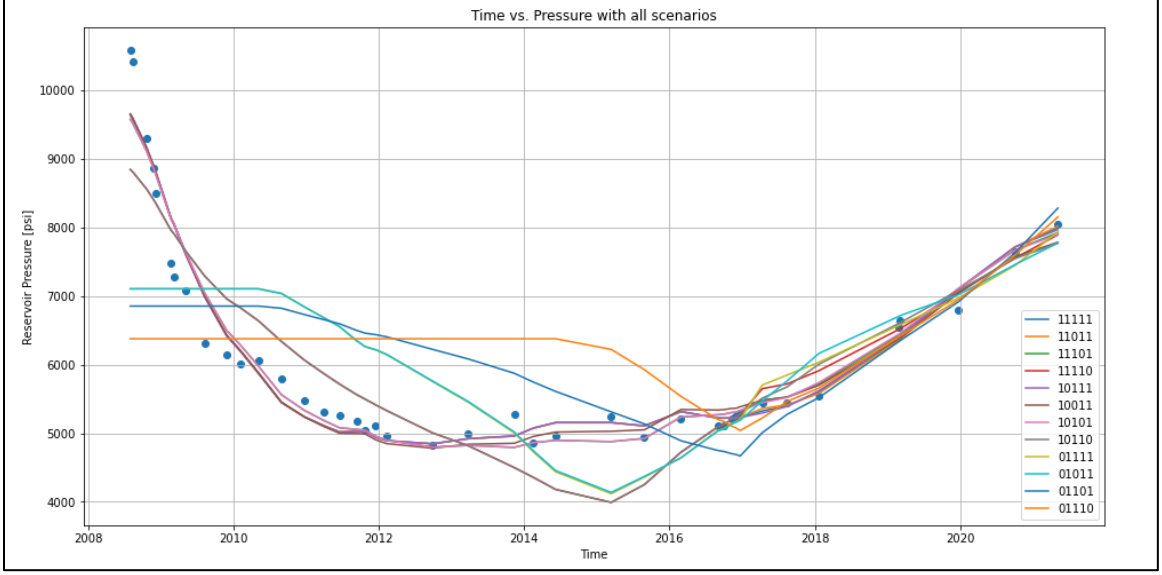
```
1 T.fit_models()
```

```
Models have succesfully been matched!
```

	Scenario Codes	R-square	Models	Intercepts	P1	P2	I1	I2	I3
0	11111	0.953097	LinearRegression()	9776.25	1.501700	-0.501700	-0.199917	0.494101	-2.578191
1	11011	0.95306	LinearRegression()	9775.43	3.154291	-2.154291	-0.000000	1.024677	-5.407735
2	11101	0.948405	LinearRegression()	9704.79	1.526513	-0.526513	0.140249	-0.000000	-2.308485
3	11110	0.830845	LinearRegression()	8921.5	-0.643306	1.643306	-0.788172	1.136009	0.000000
4	10111	0.953064	LinearRegression()	9777.37	1.000000	-0.000000	-0.255059	0.328825	-1.717561
5	10011	0.949906	LinearRegression()	9792.78	1.000000	-0.000000	-0.000000	0.135701	-1.642783
6	10101	0.948373	LinearRegression()	9705.91	1.000000	-0.000000	-0.034148	-0.000000	-1.513126
7	10110	0.830186	LinearRegression()	8922.96	1.000000	-0.000000	0.294629	-1.774112	-0.000000
8	01111	0.432506	LinearRegression()	7103.85	0.000000	1.000000	-0.275896	0.428006	-0.427561
9	01011	0.430978	LinearRegression()	7100.05	0.000000	1.000000	0.000000	1.514084	-1.525449
10	01101	0.335303	LinearRegression()	6849.92	0.000000	1.000000	0.017032	0.000000	-0.163734
11	01110	0.163312	LinearRegression()	6374.39	0.000000	1.000000	0.139534	-0.209623	0.000000

4. Last step is to “.visualize_models()” and give it to engineer’s judgement to asses various cases and interpret which scenarios make sense.

```
1 T.visualize_models()
```



Conclusion

This report showed fast implementation of the ProdPresRegression codes with example dataset, which have multiple objects and computations running in the background. Use of classes, dataframes, example data, computations for reservoir volumes and data visualization involved in the development. With the base classes next version will consider pressure dependent compressibility terms (adding non-linearity) as well as regularization type of constraint on the coefficients to obtain sensible results (injection rate increasing reservoir pressure and production reducing it.)