StatEase®
statistics made easy®
Make the most from every experiment!℠

STAT-EASE 360 + python™

Hank Anderson

hank@statease.com

Agenda

1. What is Python?
2. Connecting Python to Stat-Ease 360
3. Example 1: Multiple response plot
4. Example 2: Bringing data from the cloud into SE360
5. Example 3: Cross-validation
6. Q&A

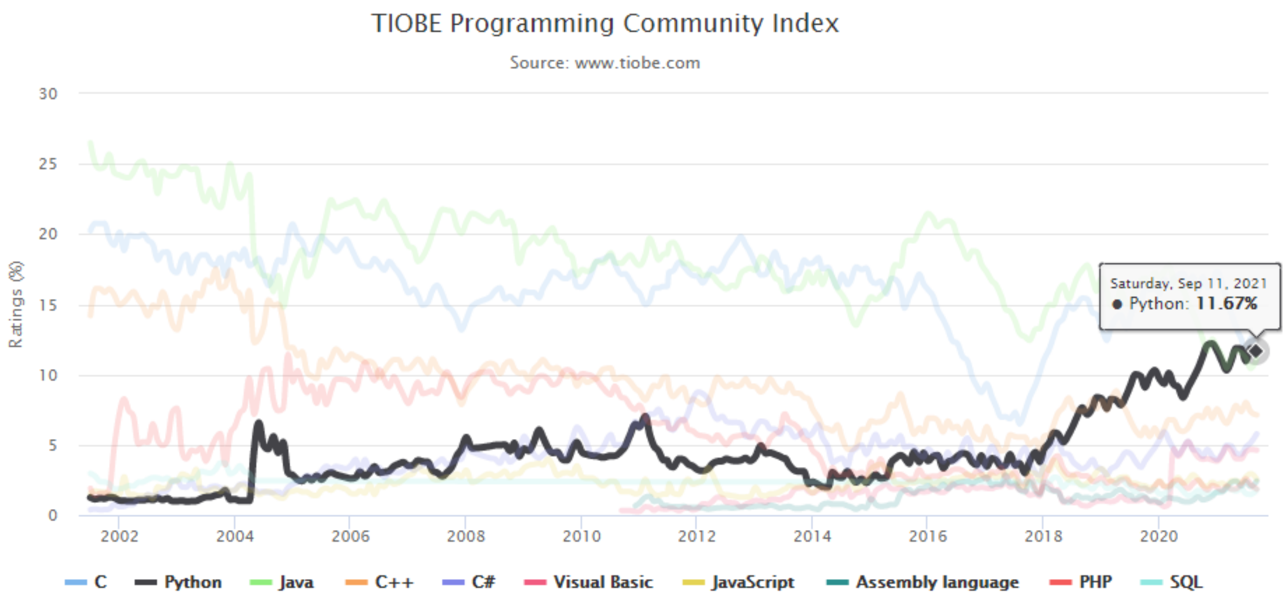
What is Python?

The Python Programming Language

- Created in 1991
- Easy to learn – emphasis on natural syntax
- Highly Productive – requires much less code than other languages
- Extensible – over 320,000 packages in the Python Package Index (PyPI)
- Scalable – used in some of the largest applications in the world (e.g. YouTube, Spotify)
- One of the most popular programming languages in the world, esp. among data scientists
- Used by Stat-Ease for internal testing and prototyping since 2015

What is Python?

General Popularity



What is Python?

Python Statistical and Scientific Packages

statsmodels

SciPy

matplotlib

scikit learn

pandas

NumPy

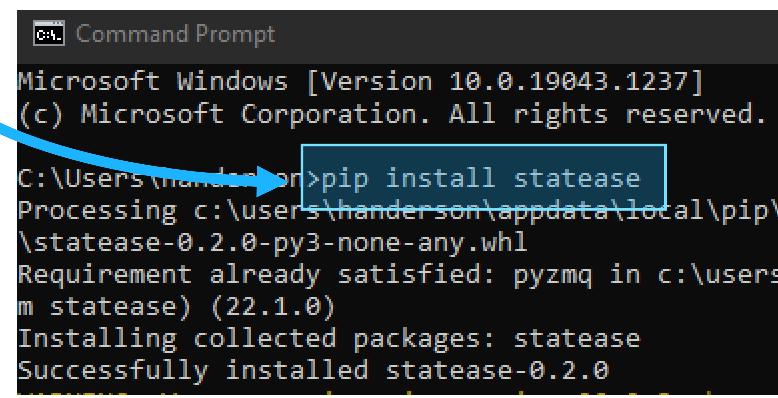
Connecting Python to Stat-Ease 360

Connecting Python to Stat-Ease 360

Setup Tutorial:
https://www.statease.com/docs/se360/tutorials/python-intro/

Quick Start:

1. Install Python >= 3.8
2. Open a command prompt (press Start, type "command prompt")
3. Run `pip install statease`
4. Press the Python icon in SE360



Example 1: Multiple Response Graph

Multiple Response Graph

This example will show how to:

- Initiate the connection from Python to Stat-Ease 360
- Extract 2 already-analyzed models from a Stat-Ease 360 design
- Make predictions for both analyses and store the results in a Python list
- Plot the predictions on one graph with two y-axes using **matplotlib**

Example 1: Multiple Response Graph

Creating the Stat-Ease 360 connection

- Use the **`connect`** function to connect to Stat-Ease 360
- This creates a Client object that represents the connected instance of SE360
- It uses port 4900 by default (you may get a firewall warning)
- The Client will use whatever design is loaded, or you can use **`open_design`**

```python
import statease as se
se_conn = se.connect()
```

Example 1: Multiple Response Graph

Retrieving an analysis

- Use the **`list_analyses`** function to get the names of the completed analyses
- Use the **`get_analysis`** function to get an Analysis object
- This can be used to set a model, or auto-select one. In this case, there is already a model selected for both analyses

```
analysis_names = se_conn.list_analyses()
analyses = [ se_conn.get_analysis(analysis_names[0]), se_conn.get_analysis(analysis_names[1]) ]
```

Example 1: Multiple Response Graph

Generating predicted values

- Generate a set of points to evaluate
- Use the **`predict`** function to send the list to SE360
- SE360 will evaluate the model at each of these points and return the predicted value

```
import numpy as np
x = np.linspace(-1, 1, 50)
```

```
[-1 -0.95918367 -0.91836735 -0.87755102 …
0.87755102  0.91836735  0.95918367  1]
```

```
# generate a list of points to predict at, leaving the
other factors at the center point
centroid = [ 0 ] * (factor_count - 1)
prediction_points = [ [ tick ] + centroid for tick in x ]
```

```
[[-1.0, 0, 0],
 [-0.95918367, 0, 0],
 [-0.91836735, 0, 0],
 [-0.87755102, 0, 0],
 …
]
```

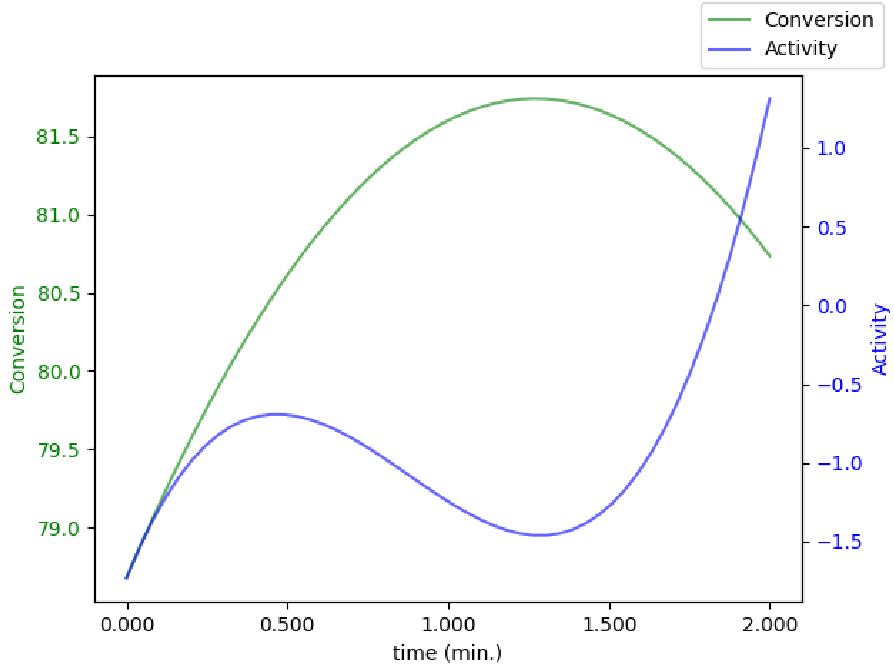
```
y1 = analyses[0].predict(prediction_points, coded=True)
y2 = analyses[1].predict(prediction_points, coded=True)
```

```
[78.67779044888725, 78.87138219655475, 79.05865668437715,
79.23961391235454, 79.41425388048684, 79.5825765887741 … ]
```

Example 1: Multiple Response Graph

Graphing with Matplotlib

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

a_fac = se_conn.get_factor(se_conn.list_factors()[0])
x_labels = [ '{:.3f}'.format(x) for x in np.linspace(a_fac.low,
a_fac.high, 5) ]
x_ticks = np.linspace(-1, 1, 5)

ax.set_xticks(x_ticks)
ax.set_xticklabels(x_labels)
ax.set_xlabel('{} ({})'.format(a_fac.name, a_fac.units))

ax.plot(x, y1, 'green', alpha=0.6, label=analyses[0].name)

ax.set_ylabel(analyses[0].name, color='green')
ax.tick_params(axis='y', labelcolor='green')

ax2 = ax.twinx()
ax2.plot(x, y2, 'blue', alpha=0.6, label=analyses[1].name)
ax2.set_ylabel(analyses[1].name, color='blue')
ax2.tick_params(axis='y', labelcolor='blue')

fig.legend()

plt.show()
```



Example 1: Multiple Response Graph

Graphing with Matplotlib

```
# add lines for predictions where B is set to its high and low
prediction_points = [ [ tick ] + centroid for tick in x ]

b_low = [ 0 ] * (factor_count - 1)
b_low[0] = -1
prediction_points = [ [ tick ] + b_low for tick in x ]
y3 = analyses[0].predict(prediction_points, coded=True)

b_high = [ 0 ] * (factor_count - 1)
b_high[0] = 1
prediction_points = [ [ tick ] + b_high for tick in x ]
y4 = analyses[0].predict(prediction_points, coded=True)
```

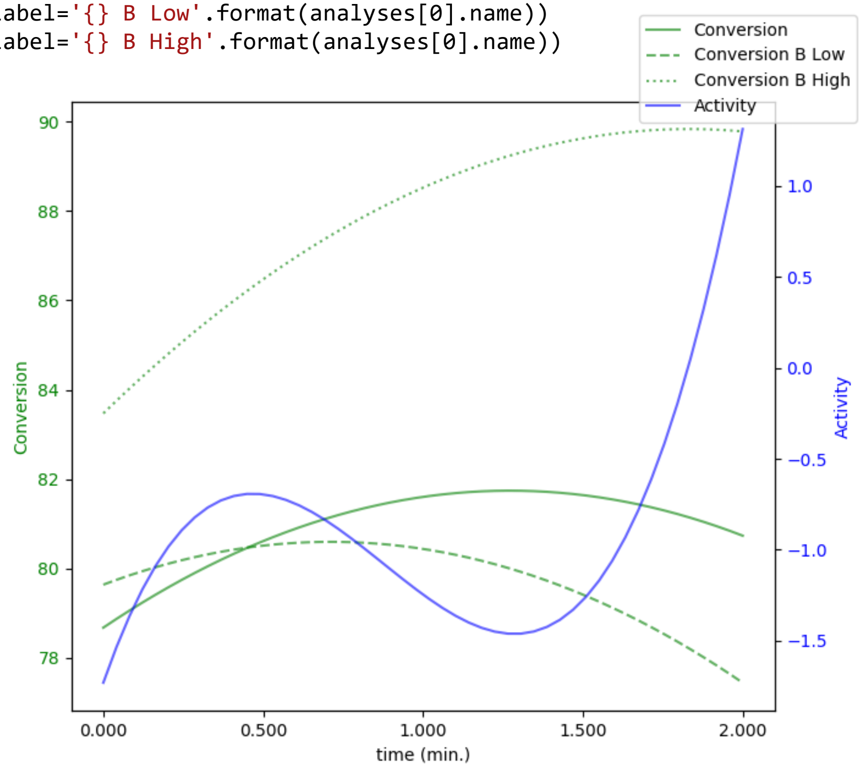
```
[[-1.0, -1, 0],
 [-0.95918367, -1, 0],
 [-0.91836735, -1, 0],
 [-0.87755102, -1, 0],
 …
]
```

```
[[-1.0, 1, 0],
 [-0.95918367, 1, 0],
 [-0.91836735, 1, 0],
 [-0.87755102, 1, 0],
 …
]
```

Example 1: Multiple Response Graph

Graphing with Matplotlib

```
ax.plot(x, y3, 'green', linestyle='dashed', alpha=0.6, label='{} B Low'.format(analyses[0].name))
ax.plot(x, y4, 'green', linestyle='dotted', alpha=0.6, label='{} B High'.format(analyses[0].name))

fig.legend()

plt.show()
```



Example 2: Streamflow

Use a REST API to bring external data into SE360 automatically

Connecting SE360 to USGS/NOAA data

1. Connect Python to two online data sources:
   - Streamflow data from the USGS
   - Precipitation and temperature data from NOAA
2. Retrieve and format data
3. Send the formatted data to SE360
4. Fit a model using Ordinary Least Squares
5. Graph the resulting model with Plotly

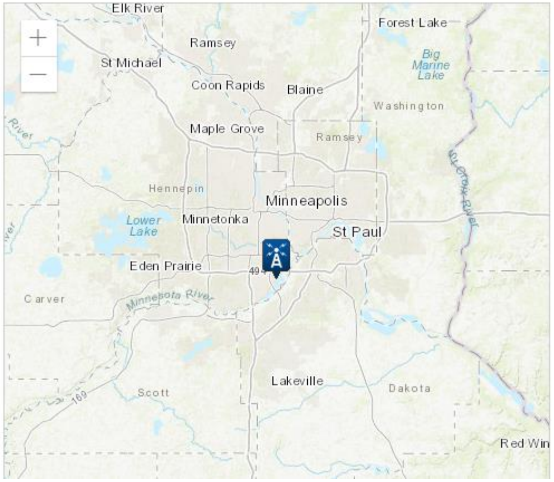
Example 2: Streamflow

Connecting to the NOAA weather data API

```python
import requests
noaa_station_id = 'GHCND:USW00014922' # MSP
datatype_a = 'PRCP'
datatype_b = 'TMAX'
startdate = '2021-09-13'
enddate = '2021-09-13'
url = ("https://www.ncdc.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND&"
       "datatypeid={}&datatypeid={}&stationid={}&startdate={}&enddate={}").format(
  datatype_a,
  datatype_b,
  noaa_station_id,
  startdate,
  enddate,
)
r = requests.get(url, headers={'token': token})
data = r.json()
```

STATION DETAILS	
Name	MINNEAPOLIS ST. PAUL INTERNATIONAL AIRPORT, MN US
Network:ID	GHCND:USW00014922
Latitude/Longitude	44.8831°, -93.2289°
Elevation	265.8 m

PERIOD OF RECORD	
Start Date[1]	1938-04-09
End Date[1]	2021-09-25
Data Coverage[2]	100%



Example 2: Streamflow

Connecting to the NOAA weather data API

```
pprint.pprint(data)
> {'metadata': {'resultset': {'count': 2, 'limit': 25, 'offset': 1}},
   'results': [{'attributes': ',,D,2400',
               'datatype': 'PRCP',
               'date': '2021-09-13T00:00:00',
               'station': 'GHCND:USW00014922',
               'value': 36},
                {'attributes': ',,D,2400',
               'datatype': 'TMAX',
               'date': '2021-09-13T00:00:00',
               'station': 'GHCND:USW00014922',
               'value': 250}]}
```

Tenths of mm

Tenths of Celsius

We'll accumulate the weather data into "rain events" and store them in 3 lists, named **precip**, **temp**, and **flow**.

Example 2: Streamflow

Send factor and response data to SE360

```
import statease as se
se_conn = se.connect()

…

precip_fac = se_conn.get_factor('precip')
precip_fac.values = precip

temp_fac = se_conn.get_factor('temp')
temp_fac.values = temp

resp = se_conn.get_response('flow')
resp.values = flow
```

- Setting variables in Factor/Response objects will write them back to SE360
- Some values are read-only
- Classes are documented in the Help under **Python Integration**

class `statease.factor.Factor`(*client*, *name*) [source]

The Factor class holds information about an individual Factor in Stat-Ease 360. Instances of this class are typically created by `statease.client.SEClient.get_factor()`

Attributes:

name (str): the name of the factor

units (str): the units of the factor

values (tuple): the values of the factor, in run order

low (str, **read only**): the actual low that corresponds to the *coded* low (this is usually, but not necessarily, the minimum observed value)

high (str, **read only**): the actual high that corresponds to the *coded* high (this is usually, but not necessarily, the maximum observed value)

coded_low (str, **read only**): the coded low value, typically -1 or 0

coded_high (str, **read only**): the coded high value, typically 1

`adjust_coding`(*new_high*, *keep_actuals=True*) [source]

Changes the mapping of the actual low and high to the coded low and high.

If *keep_actuals* is True, the actual observed values are preserved and the internal coded values are adjusted. Otherwise, the internal coded values are preserved, and the actual observed values are adjusted accordingly.

property `values`

Get or set the factor values. When setting the factor values, you may use either a list or a dictionary. If fewer values are assigned than there are rows in the design, they will be filled in starting with first row. If a dictionary is used, it must use integers as keys, and it will fill factor values in rows indexed by the dictionary keys. The indices are 0-based, so the first row is index 0, the second index 1, and so on.

Example:

```
>>> # sets the first 4 rows to a list of values
>>> factor.values = [.1, .2, .3, .4]
>>> # sets the 7th through 10th rows to specific values
>>> factor.values = { 6: .1, 7: .2, 8: .3, 9: .4 }
>>> # sets the 6th run to a specific value
>>> factor.values = { 5: .8 }
```

Example 2: Streamflow

Analyze the response using SE360

```
analysis = se_conn.create_analysis('flow', 'flow (2FI)')
analysis.set_model('A+B+AB')
anova = analysis.get_anova()
print(anova)
```

response name

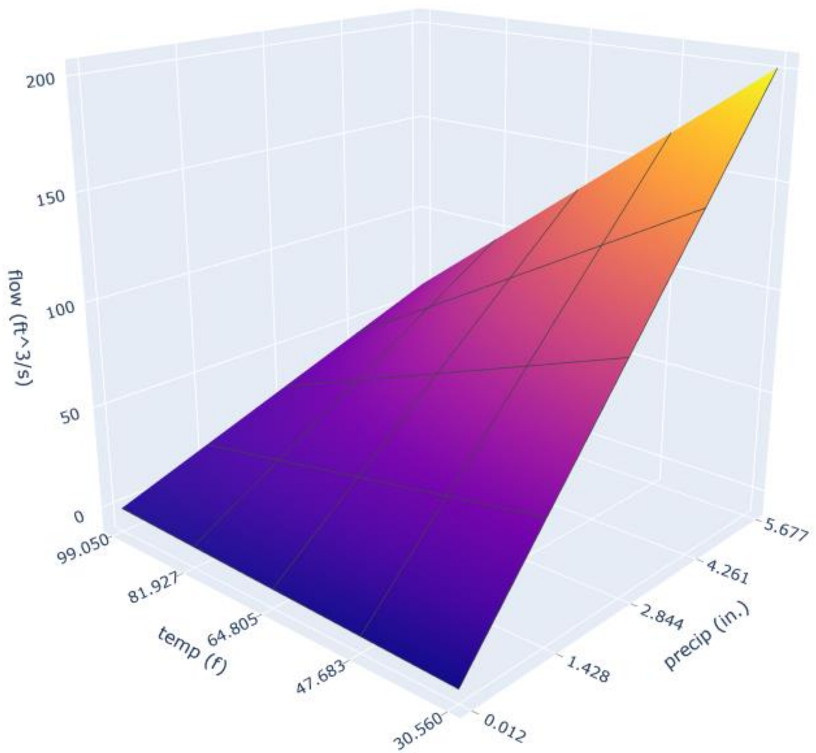
analysis name

```
R2: 0.4787206394680187
Adj R2: 0.47292864657321887
BIC: 2336.7878228612253
AICc: 2322.484009320431

Terms: [
    Term(coefficient=62.98976496436365, df=1, name='Intercept’),
    Term(coefficient=66.42928510916185, df=1, name='A', p=1.3983762180276103e-22, ss=31824.939842166204),
    Term(coefficient=-34.20843244959732, df=1, name='B', p=0.0030448108154509093, ss=2475.0445948486304),
    Term(coefficient=-36.916984344968085, df=1, name='AB', p=0.005804495970041631, ss=2140.453529576931)
]
```

Example 2: Streamflow

Graph with Plotly

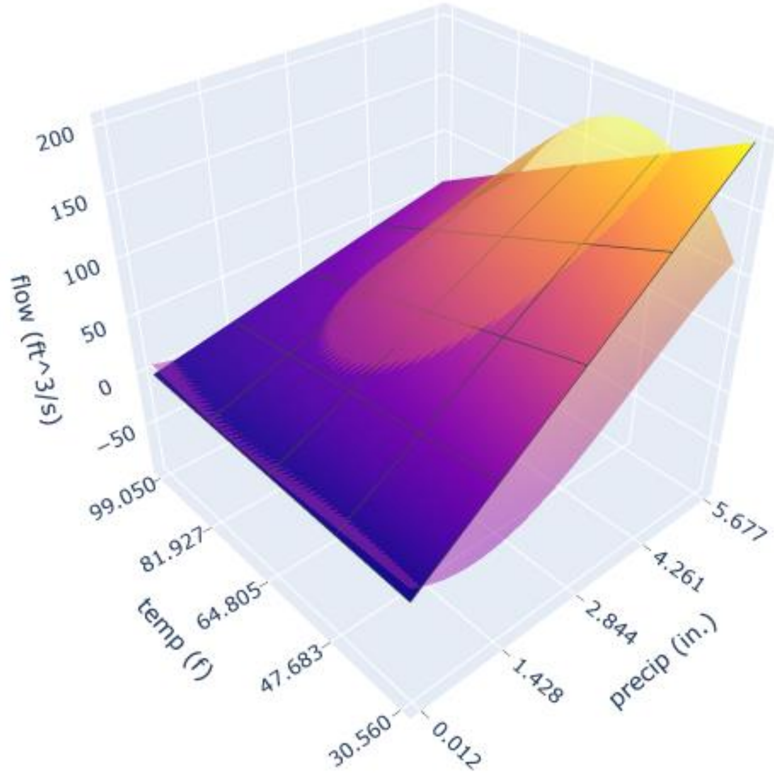
```
x = np.linspace(-1, 1, 20)
y = np.linspace(-1, 1, 20)
z = []

# predict one row of the grid at a time
for i in range(0, x.size):
    prediction_points = [ [ x[i] , yi ] for yi in y ]
    z.append(analysis.predict(prediction_points, coded=True))

fig = go.Figure(data=[
    go.Surface(
        x=x,
        y=y,
        z=z,
        contours=dict(
            x = dict( show=True ),
            y = dict( show=True ),
        ),
    )
])
fig.show()
```



Example 2: Streamflow

Graph with Plotly, pt. 2

```
# generate a second set of predictions for a cubic model
analysis = se_conn.create_analysis('flow', 'flow (cubic)')
analysis.set_model('A+B+AB+A^2+B^2+A^2B+AB^2+A^3+B^3')

z2 = []

for i in range(0, x.size):
    prediction_points = [ [ x[i] , yi ] for yi in y ]
    z2.append(analysis.predict(prediction_points, coded=True))

fig = go.Figure(data=[
    go.Surface(z=z, x=x, y=y,
      contours=dict(
        x = dict( show=True ),
        y = dict( show=True ),
      ),
    ),
    go.Surface(z=z2, x=x, y=y, showscale=False, opacity=0.5),
])
fig.show()
```



Example 3: Cross-Validation

Use Python to run a k-fold cross-validation on a design

Cross-Validation

- Split data and use some to fit a model, some to validate
- Used when it's not feasible to gather new data for validation
- Montgomery, Peck, and Vining[1] call these the "estimation data" and "prediction data"
- Called "training" and "testing" sets in ML nomenclature

estimation

1	24.0204	0	11.9796	15.5	0.5	1	4	55
2	24.0204	0	11.9796	15.5	0.5	1	3	36
3	12.76	13.24	10	16	0.5	0.5	3	55
4	0	26	10	15.5	0.5	1	4	54
5	0	26	10	14.749	1.48866	0.762386	3	47
6	0	23.9721	12.0279	15.6875	0.5	0.8125	3	51
7	12.76	13.24	10	14	2.5	0.5	4	72
8	0	26	10	15.7859	0.5	0.714135	3.65	68
9	0	23.9721	12.0279	13.5	2.5	1	4	72
10	26	0	10	13.5	2.5	1	3	32
11	26	0	10	13.8228	2.5	0.677192	4	46
12	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	64
13	22	[illegible]	14	[illegible]	[illegible]	[illegible]	3	44
14	12.76	13.24	10	13.5	2.5	1	4	73
15	24.0204	0	11.9796	13.7192	2.5	0.780757	3	42
16	26	0	10	14.7314	1.76864	0.5	4	52
17	0	26	10	16	0.5	0.5	3	44
18	0	26	10	15.4791	0.520878	1	3.22	59
19	0	22	14	16	0.5	0.5	3	55
20	22	0	14	13.5	2.5	1	3	41
21	24.0204	0	11.9796	14	2.5	0.5	3.47653	72
22	12.76	13.24	10	15.5	0.5	1	3	62
23	12.76	13.24	10	14	2.5	0.5	3	49
24	0	23.9721	12.0279	14	2.5	0.5	3.51	76
25	0	23.9721	12.0279	13.5	2.5	1	3	51
26	22	0	14	14.8177	1.44216	0.740124	3.32	61
27	22	0	14	14	2.5	0.5	3	45

prediction

28	24.0204	0	11.9796	16	0.5	0.5	3.54483	65
29	0	22	14	14.7423	1.25766	1	3	55
30	0	22	14	13.5	2.5	1	3.455	60
31	26	0	10	14	2.5	0.5	3	39
32	22	0	14	16	0.5	0.5	4	37
33	0	22	14	14	2.5	0.5	3	57
34	22	0	14	15.5	[illegible]	1	4	46
35	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	4	65
36	26	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3.61	52
37	0	23.9721	12.0279	14	2.5	0.5	3	56
38	0	23.9721	12.0279	13.7584	2.5	0.741613	4	68
39	24.0204	0	11.9796	15.5	0.5	1	3.49	54
40	0	23.9721	12.0279	15.1006	1.39938	0.5	3	59
41	26	0	10	15.5	0.5	1	3	36
42	24.0204	0	11.9796	13.5	2.5	1	3.52	63

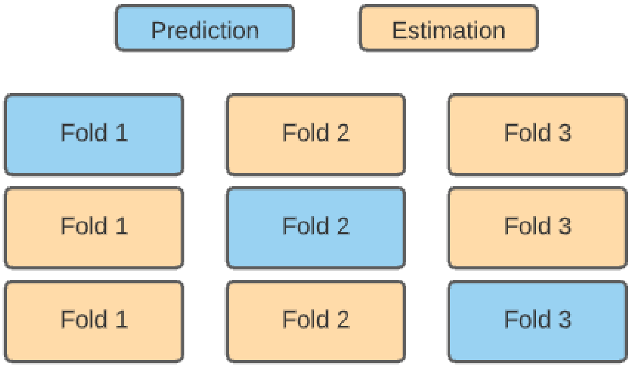
[1] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis*. New York: Wiley, p. 377

Example 3: Cross-Validation

Use Python to run a k-fold cross-validation on a design

K-fold Cross-Validation

- Design is segmented into k subsets or “folds”
- Subsets of data are alternated from being estimation data, to prediction data. (k=3 shown)



Example 3: Cross-Validation

Use Python to run a k-fold cross-validation on a design

Cross-Validation Scoring

- The prediction set in every fold is evaluated and given a score
- A design with a consistently good score is considered to pass validation
- Many options for scoring, depending on the type of problem, objective of analysis, user preference, etc.

Scoring	Function	Comment
Classification		
'accuracy'	`metrics.accuracy_score`	
'balanced_accuracy'	`metrics.balanced_accuracy_score`	
'top_k_accuracy'	`metrics.top_k_accuracy_score`	
'average_precision'	`metrics.average_precision_score`	
'neg_brier_score'	`metrics.brier_score_loss`	
'f1'	`metrics.f1_score`	for binary targets
'f1_micro'	`metrics.f1_score`	micro-averaged
'f1_macro'	`metrics.f1_score`	macro-averaged
'f1_weighted'	`metrics.f1_score`	weighted average
'f1_samples'	`metrics.f1_score`	by multilabel sample
'neg_log_loss'	`metrics.log_loss`	requires `predict_proba` support
'precision' etc.	`metrics.precision_score`	suffixes apply as with 'f1'
'recall' etc.	`metrics.recall_score`	suffixes apply as with 'f1'
'jaccard' etc.	`metrics.jaccard_score`	suffixes apply as with 'f1'
'roc_auc'	`metrics.roc_auc_score`	
'roc_auc_ovr'	`metrics.roc_auc_score`	
'roc_auc_ovo'	`metrics.roc_auc_score`	
'roc_auc_ovr_weighted'	`metrics.roc_auc_score`	
'roc_auc_ovo_weighted'	`metrics.roc_auc_score`	
Clustering		
'adjusted_mutual_info_score'	`metrics.adjusted_mutual_info_score`	
'adjusted_rand_score'	`metrics.adjusted_rand_score`	
'completeness_score'	`metrics.completeness_score`	
'fowlkes_mallows_score'	`metrics.fowlkes_mallows_score`	
'homogeneity_score'	`metrics.homogeneity_score`	
'mutual_info_score'	`metrics.mutual_info_score`	
'normalized_mutual_info_score'	`metrics.normalized_mutual_info_score`	
'rand_score'	`metrics.rand_score`	
'v_measure_score'	`metrics.v_measure_score`	
Regression		
'explained_variance'	`metrics.explained_variance_score`	
'max_error'	`metrics.max_error`	
'neg_mean_absolute_error'	`metrics.mean_absolute_error`	
'neg_mean_squared_error'	`metrics.mean_squared_error`	
'neg_root_mean_squared_error'	`metrics.mean_squared_error`	
'neg_mean_squared_log_error'	`metrics.mean_squared_log_error`	
'neg_median_absolute_error'	`metrics.median_absolute_error`	
'r2'	`metrics.r2_score`	
'neg_mean_poisson_deviance'	`metrics.mean_poisson_deviance`	
'neg_mean_gamma_deviance'	`metrics.mean_gamma_deviance`	
'neg_mean_absolute_percentage_error'	`metrics.mean_absolute_percentage_error`	

Example 3: Cross-Validation

Use Python to run a k-fold cross-validation on a design

Delivery Time Example

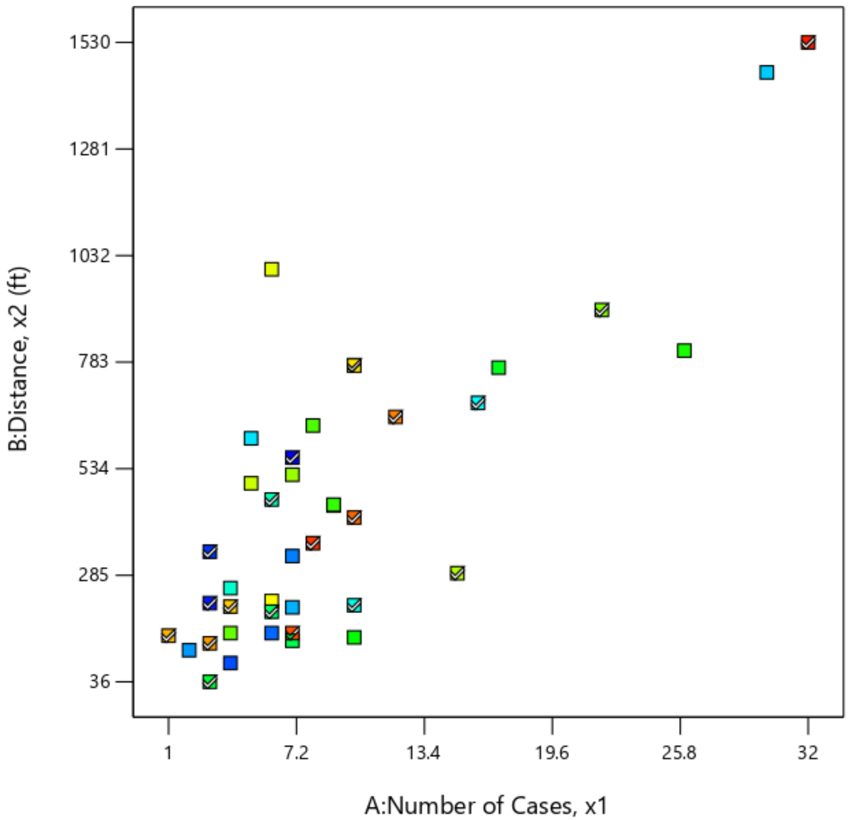
- Example data set from *Introduction to Linear Analysis*[1]
- Vending machine supplier measuring delivery time
- Data are divided into 2 sets, “estimation” and “prediction”
- Estimation set selected using DUPLEX algorithm from Snee[2]
- The reverse was not evaluated

[1] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis*. New York: Wiley.
[2] Snee, R. (1977). *Validation of Regression Models: Methods and Examples*. Technometrics. 19. 415-428.

Example 3: Cross-Validation

Delivery Time Example

Run	Comments	Factor 1 A:Number of Cases, x1	Factor 2 B:Distance, x2 ft	Response 1 Delivery Time, y min
1	P	7	560	16.68
2	P	3	220	11.5
3	P	3	340	12.03
4	E	4	80	14.88
5	E	6	150	13.75
6	E	7	330	18.11
7	E	2	110	8
8	E	7	210	17.83
9	E	30	1460	79.24
10	E	5	605	21.5
11	P	16	688	40.33
12	P	10	215	21
13	E	4	255	13.5
14	P	6	462	19.75
15	E	9	448	24
16	P	10	776	29
17	P	6	200	15.35
18	E	7	132	19
19	P	3	36	9.5
20	E	17	770	35.1
21	E	10	140	17.9
22	E	26	810	52.32
23	E	9	450	18.75
24	E	8	635	19.83
25	E	4	150	10.75
26	P	22	905	51
27	E	7	520	16.8
28	P	15	290	26.16
29	E	5	500	19.9
30	E	6	1000	24
31	E	6	225	18.55
32	P	10	775	31.93
33	P	4	212	6.95
34	P	1	144	7
35	P	3	126	14
36	P	12	655	37.03
37	P	10	420	18.62
38	P	7	150	15.1
39	P	8	360	24.38
40	P	32	1530	64.75



Example 3: Cross-Validation

Delivery Time Example

```python
import statease as se
from se360demo import get_data

delivery_time = get_data('delivery-time.dxpx')
se_conn = se.connect()
se_conn.open_design(delivery_time)

comments = se_conn.get_comments()

estimation_set = []
prediction_set = []
for r in range(0, len(comments)):
    if comments[r] == 'P':
        prediction_set.append(r)
    else:
        estimation_set.append(r)
print(estimation_set)
```

```
[3, 4, 5, 6, 7, 8, 9, 12, 14, 17, 19, 20, 21, 22, 23, 24, 26, 28, 29, 30]
```

Example 3: Cross-Validation

Delivery Time Score

Montgomery et al. used predicted R^2 to score the split data analysis:

$$R^2_{\text{Prediction}} = 1 - \frac{\sum e_i^2}{SS_T}$$

The **scikit-learn** package has the same score available in `sklearn.metrics.r2_score`:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Example 3: Cross-Validation

Delivery Time Example

```
from sklearn.metrics import r2_score
prediction_points = []
prediction_observed = []
for r in prediction_set:
    prediction_points.append([ a_fac.values[r], b_fac.values[r] ])
    prediction_observed.append(response.values[r])

se_conn.set_row_status(prediction_set, RowStatus.IGNORED)

estimate_analysis = se_conn.create_analysis('Delivery Time, y', 'Split')
estimate_analysis.set_model('A+B')
predictions = estimate_analysis.predict(prediction_points, coded=False)

print("R2 Pred: {}".format(r2_score(prediction_observed, predictions)))
```

```
R2 Pred: 0.9216137855560311
```

$$R^2_{\text{Prediction}} = 1 - \frac{\sum e_i^2}{SS_T} = 1 - \frac{322.4452}{4113.5442} = 0.922$$

Example 3: Cross-Validation

Delivery Time Example k-fold (k=4)

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=4, shuffle=False)
X = [ [a_fac.values[r], b_fac.values[r]] for r in range(0, len(a_fac.values)) ]
for train, test in kf.split(X):
    print(test)
    print(train)
```

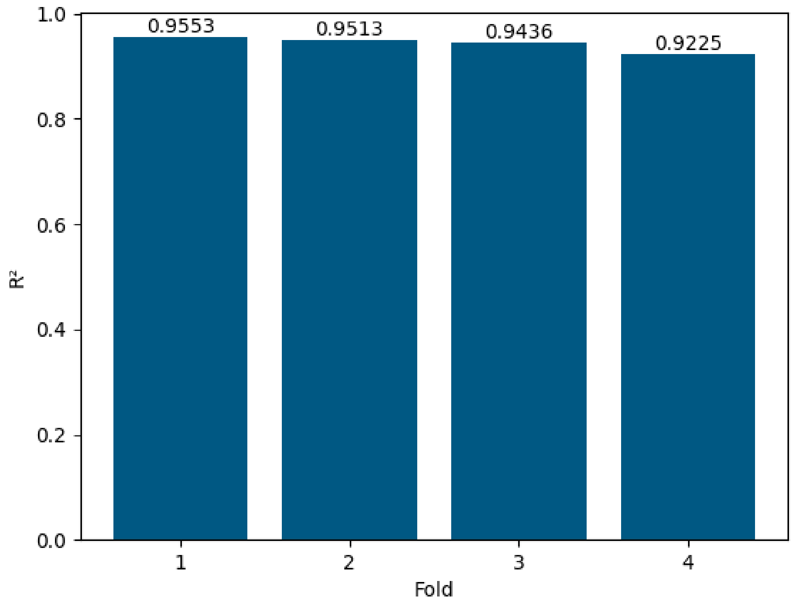
[0 1 2 3 4 5 6 7 8 9]

[10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]

	Run	Comments	Factor 1 A:Number of Cases, x1	Factor 2 B:Distance, x2 ft	Response 1 Delivery Time, y min
	1	P	7	560	16.68
	2	P	3	220	11.5
	3	P	3	340	12.03
	4	E	4	80	14.88
1	5	E	6	150	13.75
	6	E	7	330	18.11
	7	E	2	110	8
	8	E	7	210	17.83
	9	E	30	1460	79.24
	10	E	5	605	21.5
	11	P	16	688	40.33
	12	P	10	215	21
	13	E	4	255	13.5
	14	P	6	462	19.75
2	15	E	9	448	24
	16	P	10	776	29
	17	P	6	200	15.35
	18	E	7	132	19
	19	P	3	36	9.5
	20	E	17	770	35.1
	21	E	10	140	17.9
	22	E	26	810	52.32
	23	E	9	450	18.75
	24	E	8	635	19.83
3	25	E	4	150	10.75
	26	P	22	905	51
	27	E	7	520	16.8
	28	P	15	290	26.16
	29	E	5	500	19.9
	30	E	6	1000	24
	31	E	6	225	18.55
	32	P	10	775	31.93
	33	P	4	212	16.95
	34	P	1	144	7
4	35	P	3	126	14
	36	P	12	655	37.03
	37	P	10	420	18.62
	38	P	7	150	16.1
	39	P	8	360	24.38
	40	P	32	1530	64.75

Example 3: Cross-Validation

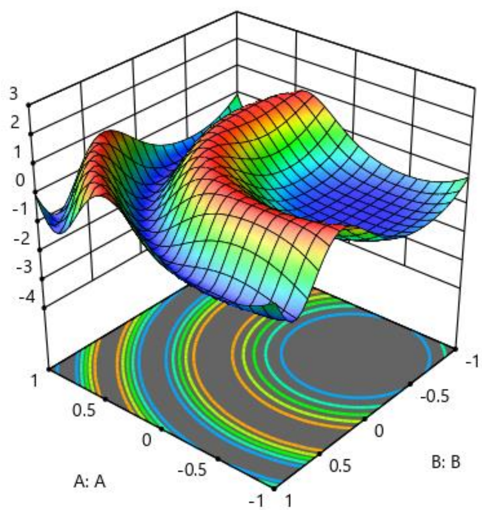
Delivery Time Example k-fold (k=4)



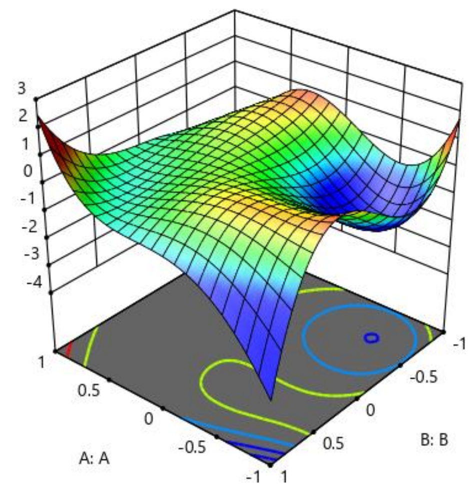
Example 3: Cross-Validation

Model Comparison Example k-fold (k=8)

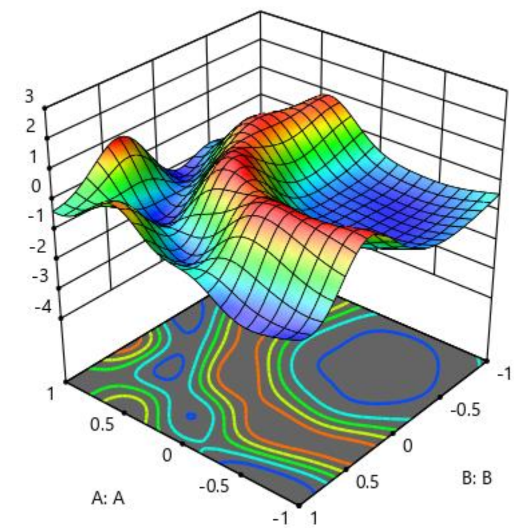
True Surface



OLS (Quartic)



Gaussian

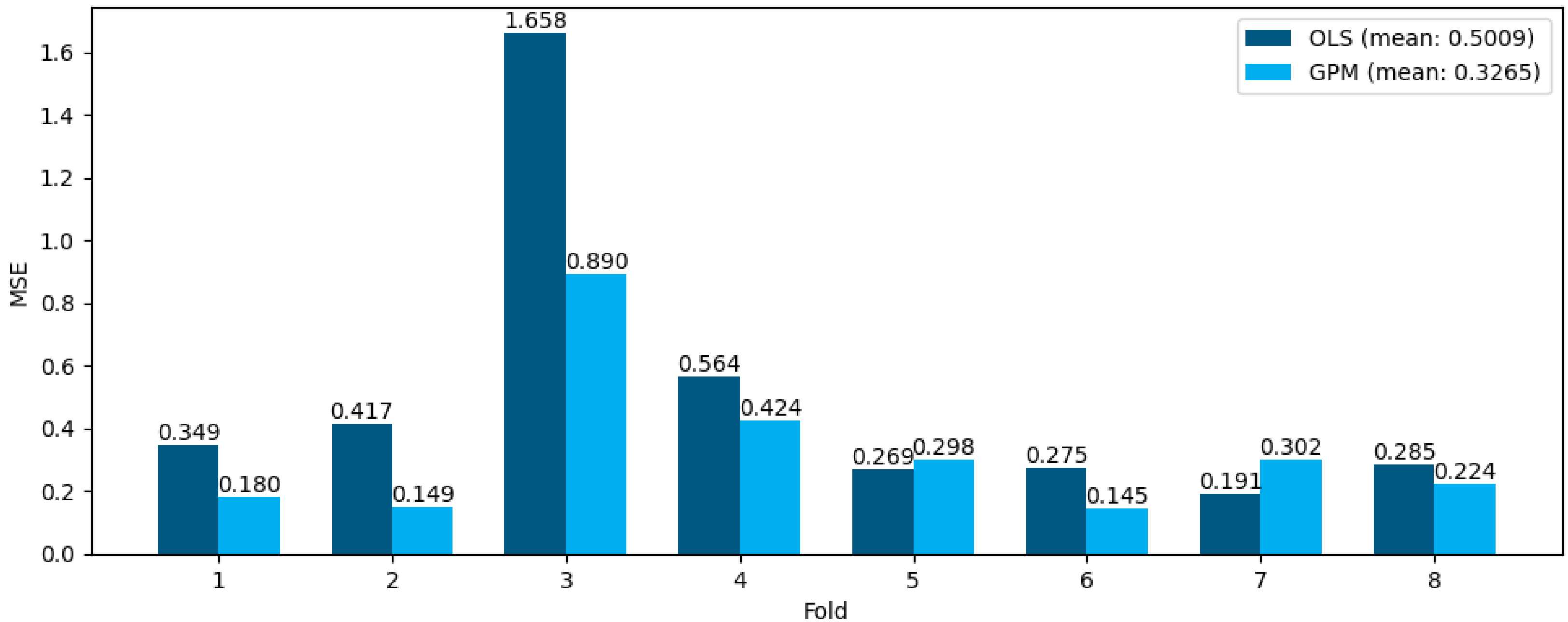


$$sin(\pi(a + b + a^2 + b^2))$$

Example 3: Cross-Validation

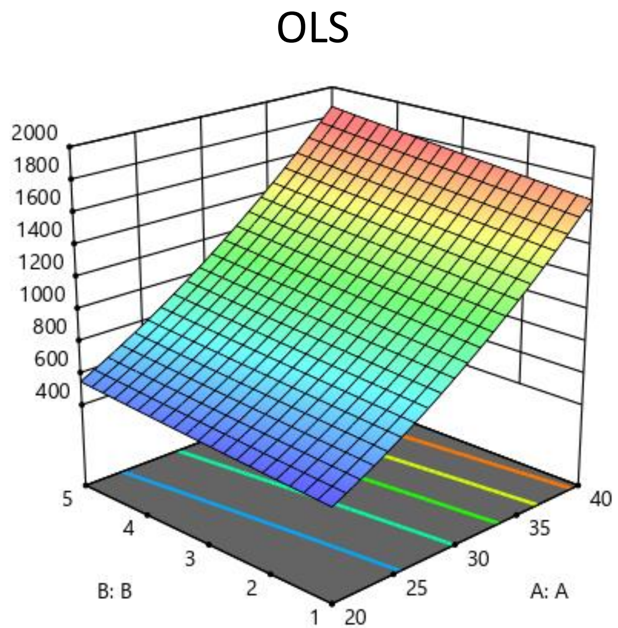
Model Comparison Example k-fold (k=8)

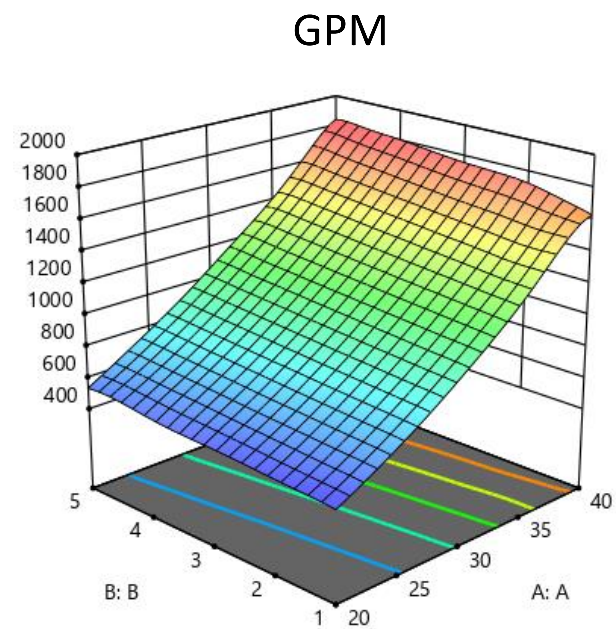
$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$



Example 3: Cross-Validation

Model Comparison Example k-fold (k=8) pt. 2



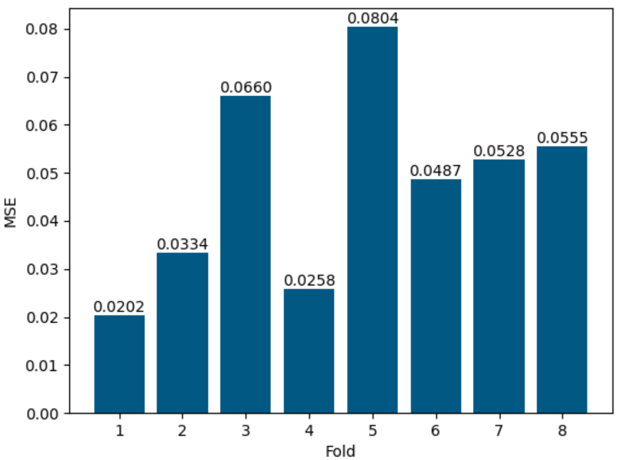


$$a + b + ab + a^2 + b^2$$

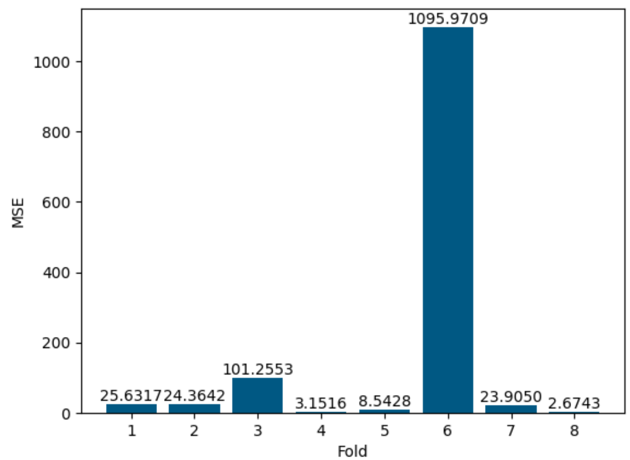
Example 3: Cross-Validation

Model Comparison Example k-fold (k=8) pt. 2

OLS



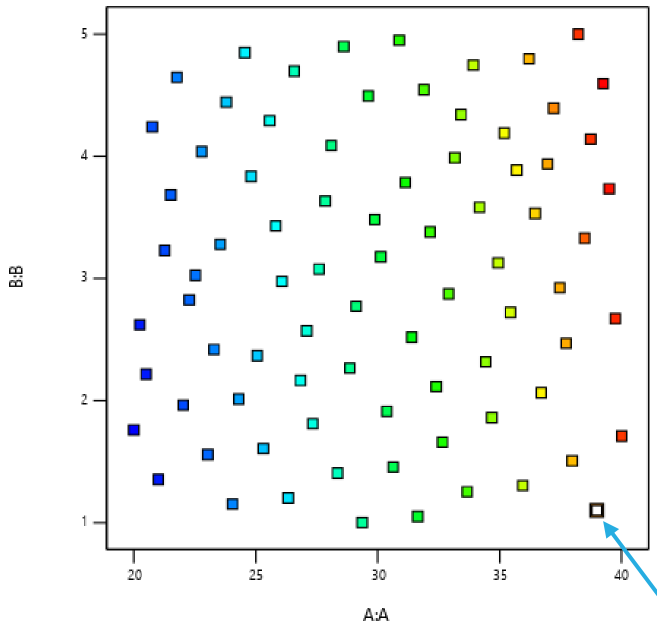
GPM



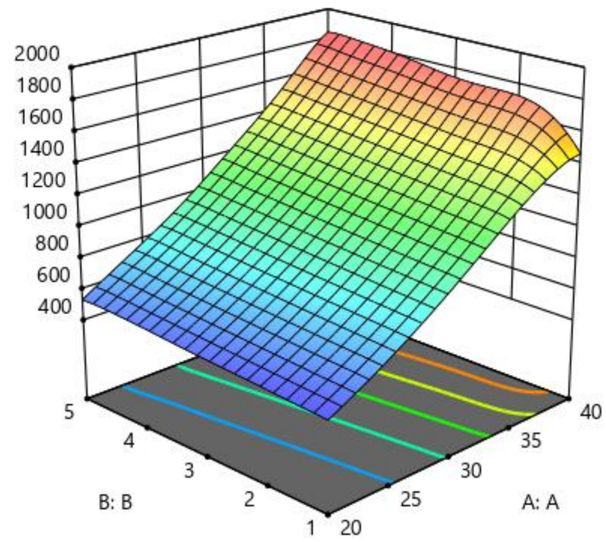
run	obs	pred
51	1510	1510.31
52	1042	1042.38
53	1105	1104.7
54	1736	1724.3
55	493.6	494.369
56	967.9	968.102
57	1378	1371.21
58	654.3	653.027
59	**1604**	**1500.24**
60	1016	1013.2

Example 3: Cross-Validation

Model Comparison Example k-fold (k=8) pt. 2



Run 59



GPM w/ Fold 6 Removed

Conclusion

- Python is an extremely powerful addition to Stat-Ease 360
- Matplotlib and Plotly enable many new graphing options
- Data can easily be retrieved from an API, from the cloud or other location
- Data can be preprocessed, transformed, etc. prior to experimentation
- Access to advanced analysis and validation techniques not (yet) available in SE360
- More endpoints (e.g. access to graph data) are being added going forward

StatEase®
statistics made easy®

*Make the most from every experiment!*SM

Happy coding!

Code: https://github.com/statease/se360-python-demo

Python Package: https://pypi.org/project/se360demo/

```
pip install se360demo
```