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

TIOBE Programming Community Index
Source: www.tiobe.com

Python: 11.67%

Python Integration with Stat-Ease 360 – A Tutorial
What is Python?

Python Statistical and Scientific Packages

- statsmodels
- SciPy
- matplotlib
- scikit-learn
- pandas
- NumPy
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
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

Multiple Response Graph
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

```python
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

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

# 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]

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

[-1, -0.95918367, -0.91836735, -0.87755102, ...
0.87755102, 0.91836735, 0.95918367, 1]

[[1, -0.95918367, 0, 0],
[-0.95918367, 0, 0],
[-0.91836735, 0, 0],
[-0.87755102, 0, 0],
...
]

[78.67779044888725, 78.87138219655475, 79.05865668437715,
79.23961391235454, 79.41425388048684, 79.5825765887741, ...
]```
Example 1: Multiple Response Graph

Graphing with Matplotlib

```python
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)
anx.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

```python
# 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)
```

```python
[[-1.0, -1, 0],
 [-0.95918367, -1, 0],
 [-0.91836735, -1, 0],
 [-0.87755102, -1, 0],
 ... ]
```

```python
[[-1.0, 1, 0],
 [-0.95918367, 1, 0],
 [-0.91836735, 1, 0],
 [-0.87755102, 1, 0],
 ... ]
```
Example 1: Multiple Response Graph

Graphing with Matplotlib

```python
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'
      '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()
```

<table>
<thead>
<tr>
<th>STATION DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>NetworkID</td>
</tr>
<tr>
<td>Latitude/Longitude</td>
</tr>
<tr>
<td>Elevation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PERIOD OF RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Date¹</td>
</tr>
<tr>
<td>End Date¹</td>
</tr>
<tr>
<td>Data Coverage²</td>
</tr>
</tbody>
</table>
Example 2: Streamflow

Connecting to the NOAA weather data API

```python
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}]
```

We’ll accumulate the weather data into “rain events” and store them in 3 lists, named `precip`, `temp`, and `flow`. 

- Tenths of mm
- Tenths of Celsius
Example 2: Streamflow

Send factor and response data to SE360

```python
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**
Example 2: Streamflow

Analyze the response using SE360

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

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=0.00580449570041631, ss=2140.453529576931)
]
Example 2: Streamflow

Graph with Plotly

```python
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

```python
# 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

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.
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 Snee²
- The reverse was not evaluated

Example 3: Cross-Validation

Delivery Time Example

<table>
<thead>
<tr>
<th>Run</th>
<th>Comments</th>
<th>Factor 1 (Number of Cases, x1)</th>
<th>Factor 2 (Distance, x2 ft)</th>
<th>Response 1 (Delivery Time, y min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P</td>
<td>7</td>
<td>500</td>
<td>16.68</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>3</td>
<td>230</td>
<td>11.5</td>
</tr>
<tr>
<td>3</td>
<td>P</td>
<td>3</td>
<td>340</td>
<td>12.03</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>4</td>
<td>50</td>
<td>14.88</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>6</td>
<td>150</td>
<td>13.75</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>7</td>
<td>330</td>
<td>18.11</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>2</td>
<td>110</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>7</td>
<td>210</td>
<td>17.83</td>
</tr>
<tr>
<td>9</td>
<td>E</td>
<td>30</td>
<td>1400</td>
<td>79.24</td>
</tr>
<tr>
<td>10</td>
<td>E</td>
<td>3</td>
<td>625</td>
<td>21.5</td>
</tr>
<tr>
<td>11</td>
<td>P</td>
<td>16</td>
<td>688</td>
<td>40.33</td>
</tr>
<tr>
<td>12</td>
<td>P</td>
<td>10</td>
<td>215</td>
<td>21.2</td>
</tr>
<tr>
<td>13</td>
<td>E</td>
<td>4</td>
<td>235</td>
<td>13.5</td>
</tr>
<tr>
<td>14</td>
<td>P</td>
<td>5</td>
<td>462</td>
<td>19.75</td>
</tr>
<tr>
<td>15</td>
<td>E</td>
<td>9</td>
<td>448</td>
<td>24.1</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>10</td>
<td>776</td>
<td>29.9</td>
</tr>
<tr>
<td>17</td>
<td>E</td>
<td>6</td>
<td>200</td>
<td>15.35</td>
</tr>
<tr>
<td>18</td>
<td>E</td>
<td>7</td>
<td>182</td>
<td>19.1</td>
</tr>
<tr>
<td>19</td>
<td>P</td>
<td>3</td>
<td>36</td>
<td>9.5</td>
</tr>
<tr>
<td>20</td>
<td>E</td>
<td>2</td>
<td>770</td>
<td>35.1</td>
</tr>
<tr>
<td>21</td>
<td>E</td>
<td>10</td>
<td>140</td>
<td>17.9</td>
</tr>
<tr>
<td>22</td>
<td>E</td>
<td>26</td>
<td>810</td>
<td>52.32</td>
</tr>
<tr>
<td>23</td>
<td>E</td>
<td>9</td>
<td>450</td>
<td>18.75</td>
</tr>
<tr>
<td>24</td>
<td>E</td>
<td>8</td>
<td>635</td>
<td>19.85</td>
</tr>
<tr>
<td>25</td>
<td>E</td>
<td>4</td>
<td>150</td>
<td>10.75</td>
</tr>
<tr>
<td>26</td>
<td>P</td>
<td>22</td>
<td>905</td>
<td>51</td>
</tr>
<tr>
<td>27</td>
<td>E</td>
<td>7</td>
<td>520</td>
<td>16.9</td>
</tr>
<tr>
<td>28</td>
<td>P</td>
<td>15</td>
<td>290</td>
<td>26.16</td>
</tr>
<tr>
<td>29</td>
<td>E</td>
<td>5</td>
<td>500</td>
<td>19.9</td>
</tr>
<tr>
<td>30</td>
<td>E</td>
<td>6</td>
<td>1000</td>
<td>24.1</td>
</tr>
<tr>
<td>31</td>
<td>E</td>
<td>6</td>
<td>225</td>
<td>18.55</td>
</tr>
<tr>
<td>32</td>
<td>P</td>
<td>10</td>
<td>775</td>
<td>31.93</td>
</tr>
<tr>
<td>33</td>
<td>P</td>
<td>4</td>
<td>212</td>
<td>6.95</td>
</tr>
<tr>
<td>34</td>
<td>P</td>
<td>1</td>
<td>144</td>
<td>7</td>
</tr>
<tr>
<td>35</td>
<td>P</td>
<td>3</td>
<td>126</td>
<td>14.0</td>
</tr>
<tr>
<td>36</td>
<td>P</td>
<td>12</td>
<td>615</td>
<td>37.03</td>
</tr>
<tr>
<td>37</td>
<td>P</td>
<td>10</td>
<td>420</td>
<td>18.62</td>
</tr>
<tr>
<td>38</td>
<td>P</td>
<td>7</td>
<td>150</td>
<td>15.1</td>
</tr>
<tr>
<td>39</td>
<td>P</td>
<td>8</td>
<td>360</td>
<td>24.34</td>
</tr>
<tr>
<td>40</td>
<td>P</td>
<td>32</td>
<td>1530</td>
<td>64.75</td>
</tr>
</tbody>
</table>

Python Integration with Stat-Ease 360 – A Tutorial
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

```python
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

$$R^2_{\text{Prediction}} = 1 - \frac{\sum e^2}{SS_F} = 1 - \frac{322.4452}{4113.5442} = 0.922$$
Example 3: Cross-Validation

Delivery Time Example k-fold (k=4)

```python
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]
```
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)

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

Python Integration with Stat-Ease 360 – A Tutorial
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

<table>
<thead>
<tr>
<th>Fold</th>
<th>Run</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51</td>
<td>0.0202</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>0.0334</td>
</tr>
<tr>
<td>3</td>
<td>53</td>
<td>0.0660</td>
</tr>
<tr>
<td>4</td>
<td>54</td>
<td>0.0587</td>
</tr>
<tr>
<td>5</td>
<td>55</td>
<td>0.0528</td>
</tr>
<tr>
<td>6</td>
<td>56</td>
<td>0.0555</td>
</tr>
<tr>
<td>7</td>
<td>57</td>
<td>0.0623</td>
</tr>
<tr>
<td>8</td>
<td>58</td>
<td>0.0638</td>
</tr>
</tbody>
</table>

### GPM

<table>
<thead>
<tr>
<th>Fold</th>
<th>Run</th>
<th>MSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>59</td>
<td>1095.9709</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>239050.027</td>
</tr>
<tr>
<td>3</td>
<td>61</td>
<td>85428.2516</td>
</tr>
<tr>
<td>4</td>
<td>62</td>
<td>8035.1516</td>
</tr>
<tr>
<td>5</td>
<td>63</td>
<td>1013.2</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>1500.24</td>
</tr>
<tr>
<td>7</td>
<td>65</td>
<td>653.027</td>
</tr>
<tr>
<td>8</td>
<td>66</td>
<td>1371.21</td>
</tr>
</tbody>
</table>

Python Integration with Stat-Ease 360 – A Tutorial
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