A JMP Script that Determines a Simultaneous 95% Bound using a k-Nearest Neighbor Approach

Rich Newman, Intel®

Don Kent, Intel®

JMP® Discovery Summit 2021

AGENDA

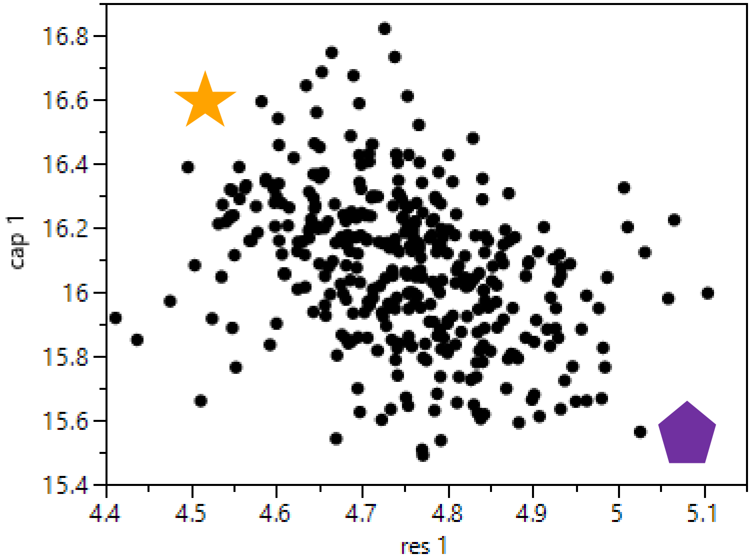
- Motivate the problem
- Share some possible solutions
- Share the solution we went with
- Illustrate the solution with an example
  - Show screenshots from the JMP® Add-in
- Provide highlights from the script
- Summary

MOTIVATING PROBLEM

- We are designing a device
- We need to know what is the "worst case" set of 4 resistance and 4 capacitance values that we may see.
- Worst case is defined as the low resistance / high capacitance and high resistance / low capacitance combinations.
  - Low res_1, low res_2, low res_3, low res_4, hi cap_1, hi cap_2, hi cap_3, hi cap_4
  - Hi res_1, hi res_2, hi res_3, hi res_4, low cap_1, low cap_2, low cap_3, low cap_4
- Worst case may be defined as 95% confidence or 99% confidence or …

ILLUSTRATION OF THE PROBLEM

- For demonstration purposes, let's use 1 resistance and 1 capacitance value.



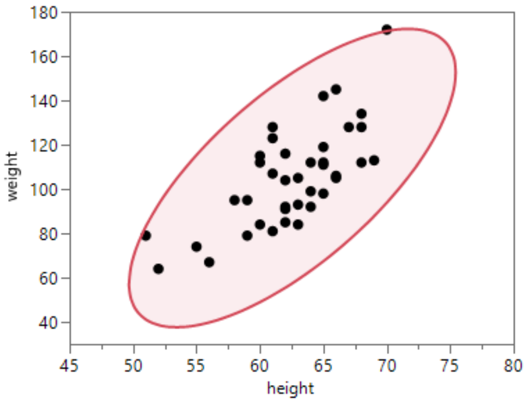
- We want to know 2 things
- What is the "worst" low resistance / high capacitance combination? ★
- What is the "worst" high resistance / low capacitance combination? ⬟

MOTIVATING PROBLEM – A COUPLE OF THINGS TO CONSIDER

- We know that the 8 responses / variables are not independent.
- Each of the 8 capacitance and resistance responses may or may not follow a normal or multivariate normal distribution.
- We ask these types of questions frequently
  - We need a robust solution that is easy to use
- We tend to have relatively large data sets (at least 400 points).
- A practical solution works for us (with some statistics behind it)
  - We do not have a definition of "worst-case" (e.g., 99%)
  - Better to be a little conservative

MOTIVATING PROBLEM – A (MADE-UP) DIFFERENT INDUSTRY EXAMPLE

- We make adjustable desks for the classroom.
- We want our desks to work for 99% of the population
- Two major dimensions go into building a desk
  - A person's height
  - A person's width
    - If we use JMP® *BigClass.JMP* data set (and use weight for width), we could determine height and weight bounds that capture 99% of the population.



CURRENT APPROACH – WHICH WE BELIEVE CAN BE IMPROVED

- We independently find the 95% (or 99% or …) prediction bounds for resistance and capacitance.
  - Two separate, independent bounds
- The worst-case is the combination of these two numbers.

res 1

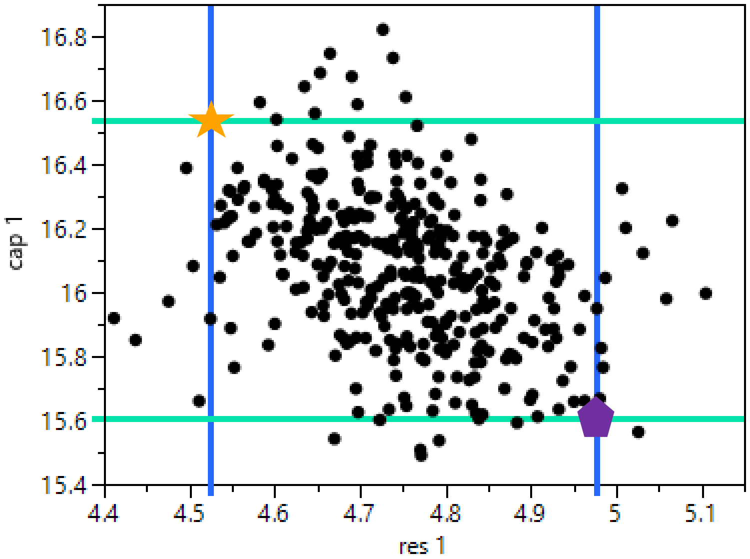
Prediction Interval

Parameter	Future N	Lower PI	Upper PI	1-Alpha
Individual	1	4.523663	4.977208	0.950

cap 1

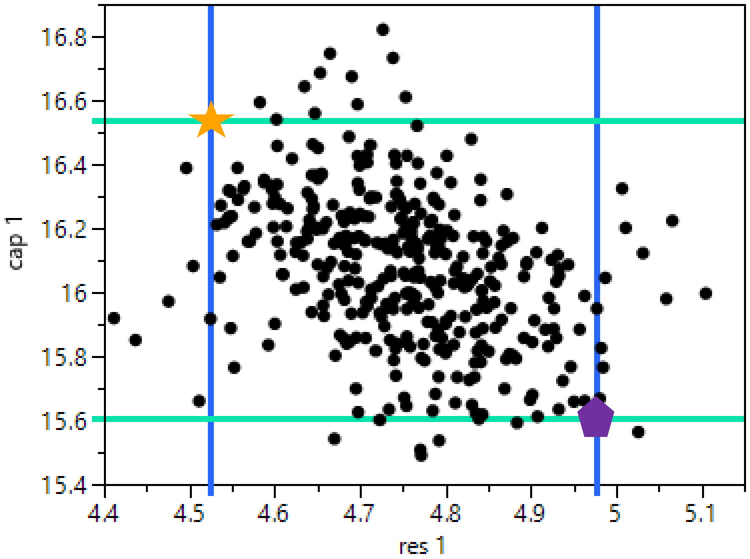
Prediction Interval

Parameter	Future N	Lower PI	Upper PI	1-Alpha
Individual	1	15.60834	16.54047	0.950



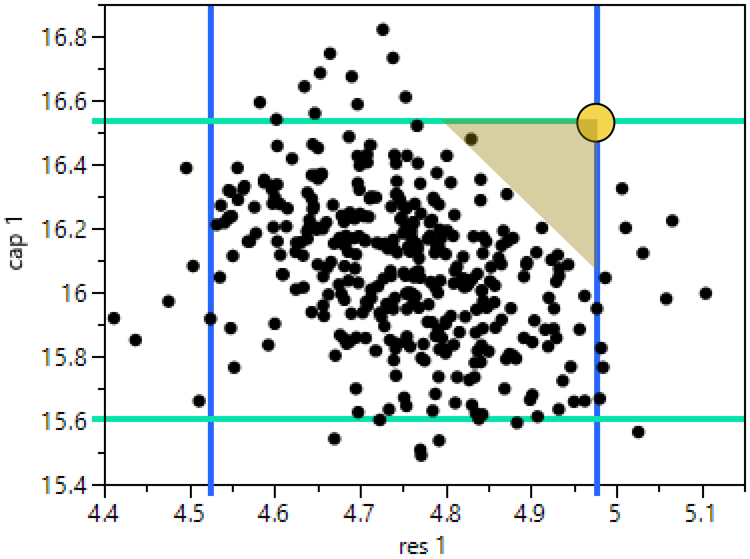
CONCERNS WITH CURRENT APPROACH

- Type I error (α)
  - 95% bound for resistance 1
  - 95% bound for capacitance 1
  - 95% bound for …
  - Overall, the confidence level is not 95%
    - Depends on the correlations of the variables
- Could make an alpha adjustment



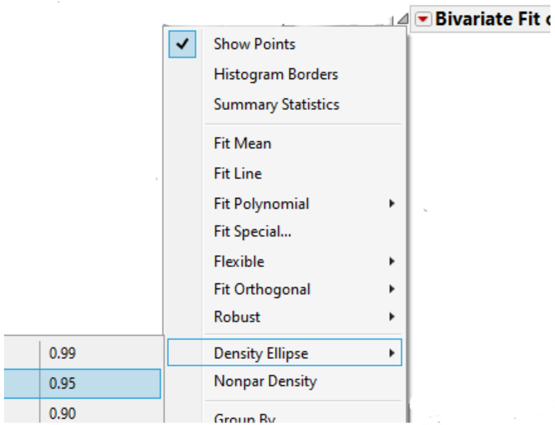
CONCERNS WITH CURRENT APPROACH

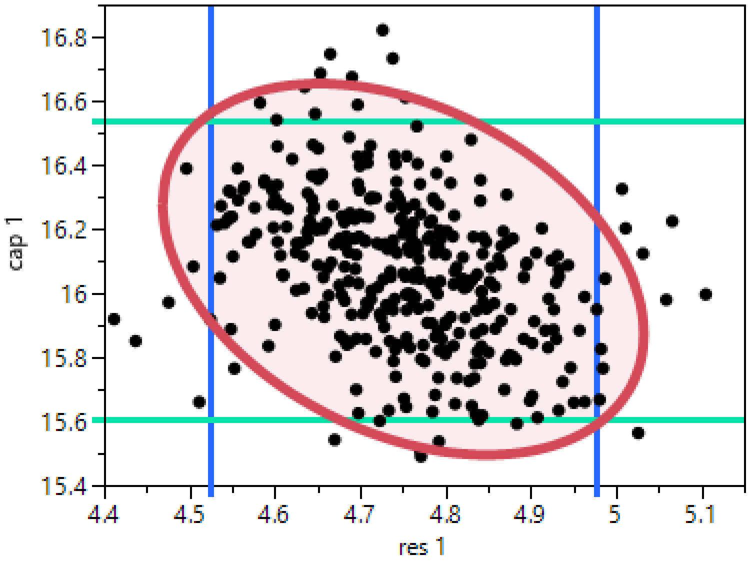
- What if we were interested in the high/high combination?
- For this example, we do not really have data near the estimated worst-case bound.
- Thus, this estimate would be considered highly conservative.
  - From a design perspective, this has a cost associated with it.



ALTERNATIVE APPROACHES THAT ARE "EASILY" DONE IN JMP®

- Density Ellipses
- Found in the Fit Y by X Platform





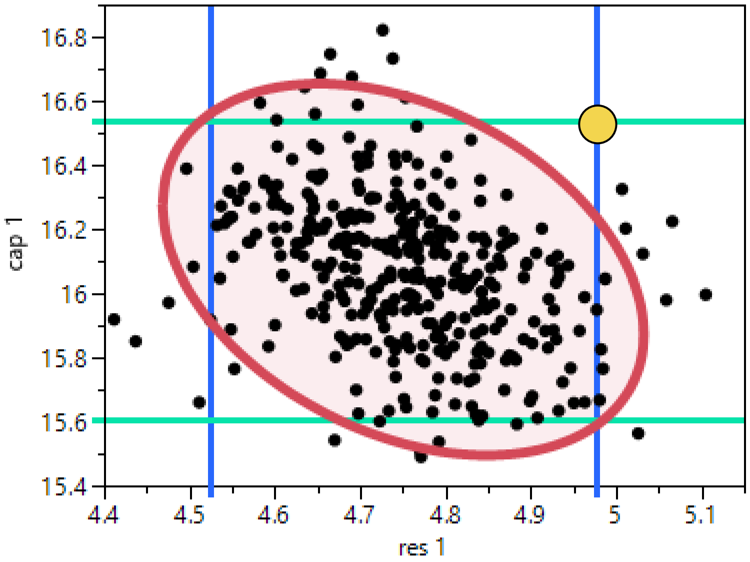
ALTERNATIVE APPROACHES THAT ARE "EASILY" DONE IN JMP®

- Density Ellipse Concerns

- What is the equation of the ellipse?

- What if you have more than 2 variables?
- Pairwise ellipses – same α problem

- What if interested in the high/high corner?
- Which point on the ellipse do you choose?

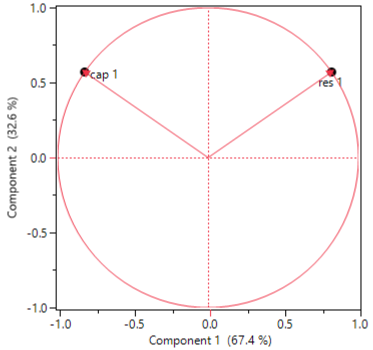


Bivariate Normal Ellipse P=0.950

Variable	Mean	Std Dev	Correlation	Signif. Prob	Number
res 1	4.750436	0.115168	-0.34843	<.0001*	370
cap 1	16.07441	0.236691			

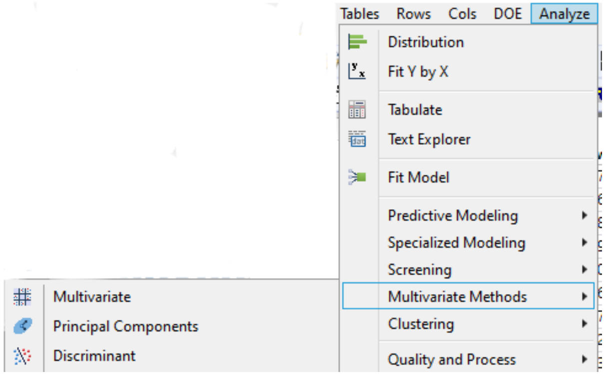
ALTERNATIVE APPROACHES THAT ARE “EASILY” DONE IN JMP®

- Principal Components
- Creates new variables (e.g., Prin 1, Prin 2)
- Such that the new variables are orthogonal
- Found in Multivariate Methods



$$6.1398081545 \cdot res\ 1 + -2.987466486 \cdot cap\ 1 + 18.854984327$$

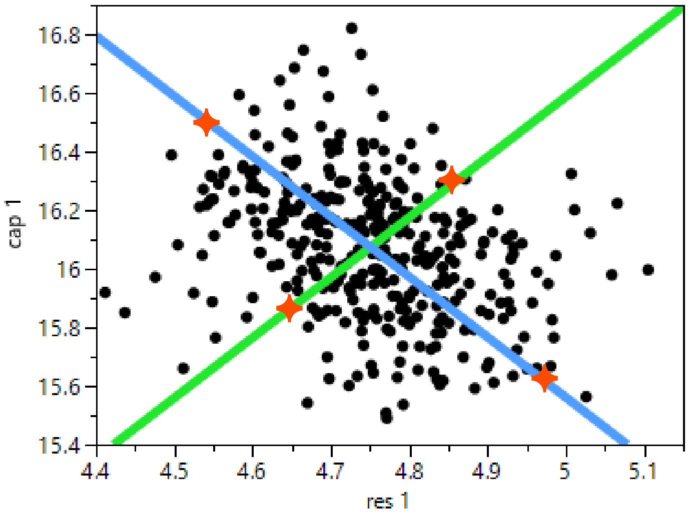
$$6.1398081545 \cdot res\ 1 + 2.9874664861 \cdot cap\ 1 + -77.1885096$$



ALTERNATIVE APPROACHES THAT ARE "EASILY" DONE IN JMP®

- Principal Components Concerns

- The math is difficult when there are more than 2 variables.
- In theory, Principal Components tries to "reduce dimensionality"
  - E.g. – we have 8 variables
  - But 3 main principal components

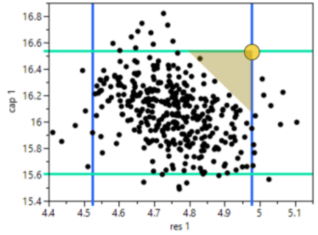


$6.1398081545 \cdot \textit{res 1} + -2.987466486 \cdot \textit{cap 1} + 18.854984327$

$6.1398081545 \cdot \textit{res 1} + 2.9874664861 \cdot \textit{cap 1} + -77.1885096$

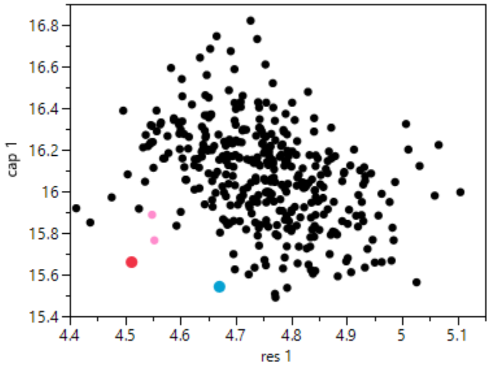
WHERE DOES THAT LEAVE US?

- Goal: Find the simultaneous worst-case bounds
    - (“high/high” or “low/high” or “low/low/low/high/high/low/high” or …)
- We would like to use JMP® to help us solve the problem
- Needs to be able to handle 3 or more variables
- Each variable may or may not be normal
- There are expected to be some correlations between variables
- We tend to have relatively large data sets
- We may ask for a “corner” where there is no data
- An easy, practical solution may be sufficient



CONCEPT – K-NEAREST NEIGHBORS

- You have a point
- Find the distance from that point to every other point
- Determine the point's k nearest neighbors (k points with shortest distances)
- Example
  - The point in red is (4.51, 15.66)
  - The point in blue is (4.67, 15.54)
  - The distance = $\sqrt{(4.51 - 4.67)^2 + (15.66 - 15.54)^2}$
  - Repeat this for every point. Then, sort by distances.
  - The k=2 nearest neighbors are in pink (the 2 shortest distances)

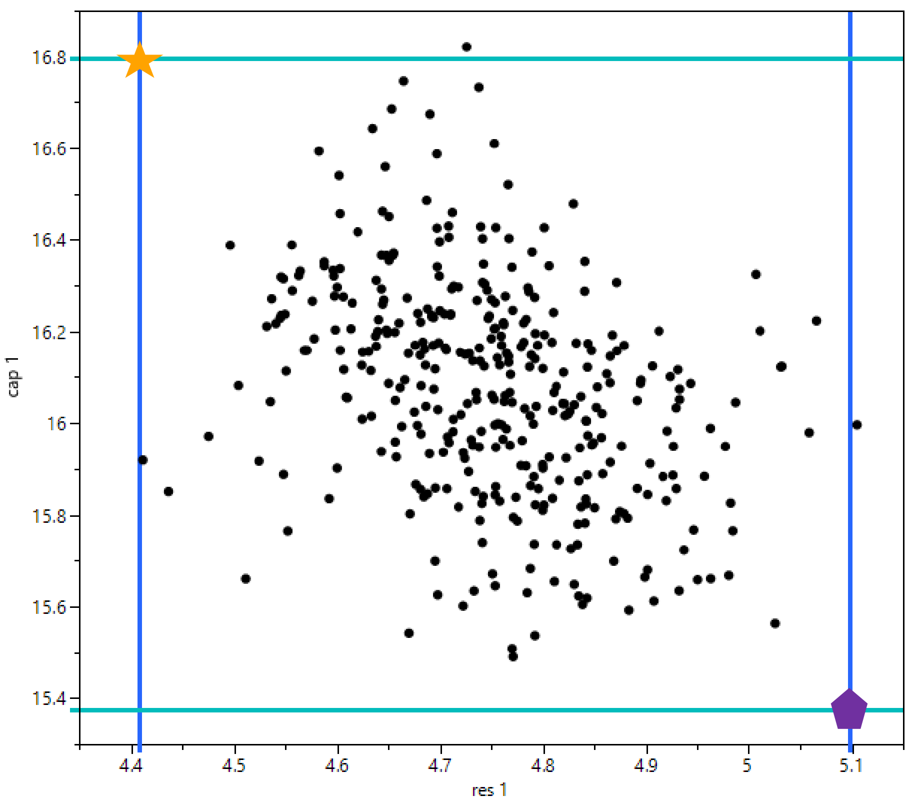


SOLUTION: USE K-NEAREST NEIGHBORS TO FIND WORST-CASE BOUNDS

- Idea:
  - If you have a “large” data set
  - Find the median +/- 3 std devs for each variable (could also use mean)
  - Define targeted corner(s) – the low or high point for each variable
  - Find the distance from each point to the “targeted corner”
  - Sort the distances from smallest to largest.
  - For our needs*, we take the k nearest neighbors to the “targeted corner”
    - In general, collect k neighbors that represent the desired confidence
  - Take the average of the k neighbors

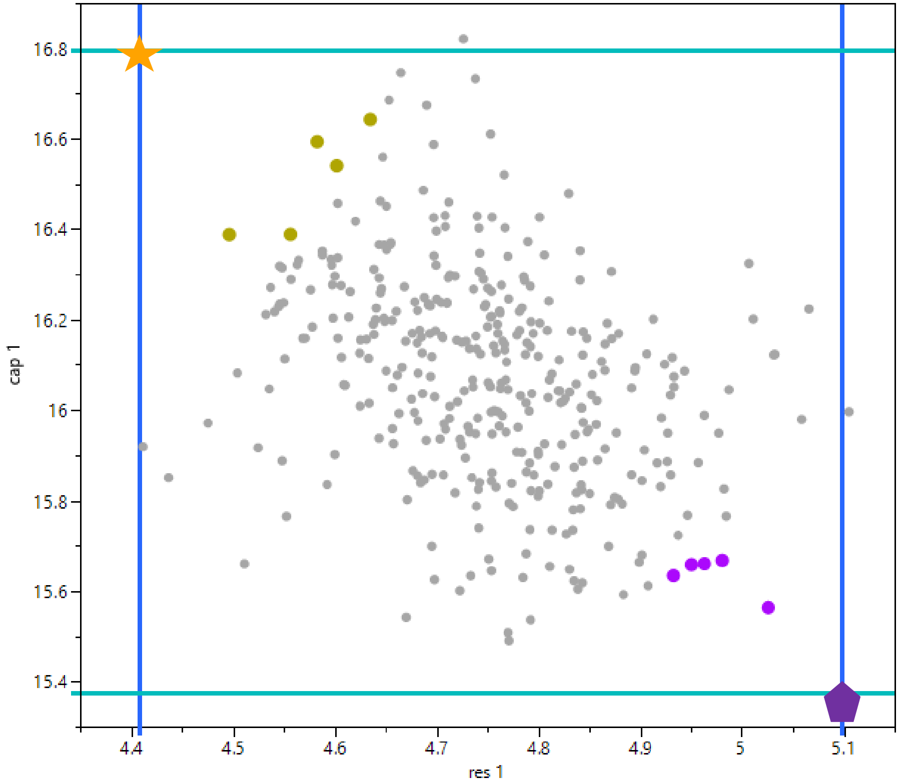
SOLUTION: K-NEAREST NEIGHBORS

- Idea:
  - Find the median +/- 3 std devs for each variable (could also use mean)
  - Define targeted corners – in this example, the low/high and the high/low corner.



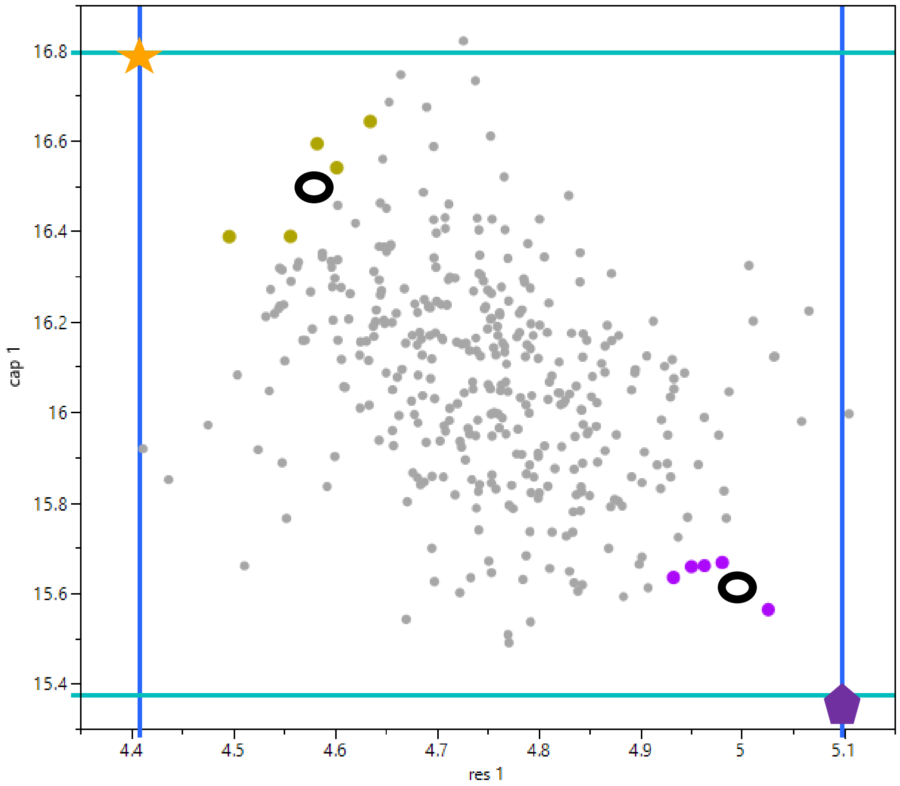
SOLUTION: K-NEAREST NEIGHBORS

- Idea:
  - Find the median +/- 3 std devs for each variable (could also use mean)
  - Define targeted corners – in this example, the low/high and the high/low corner.
  - Next, find the k-nearest neighbors to the targeted corners.
  - As an illustration, the 5 nearest neighbors are found



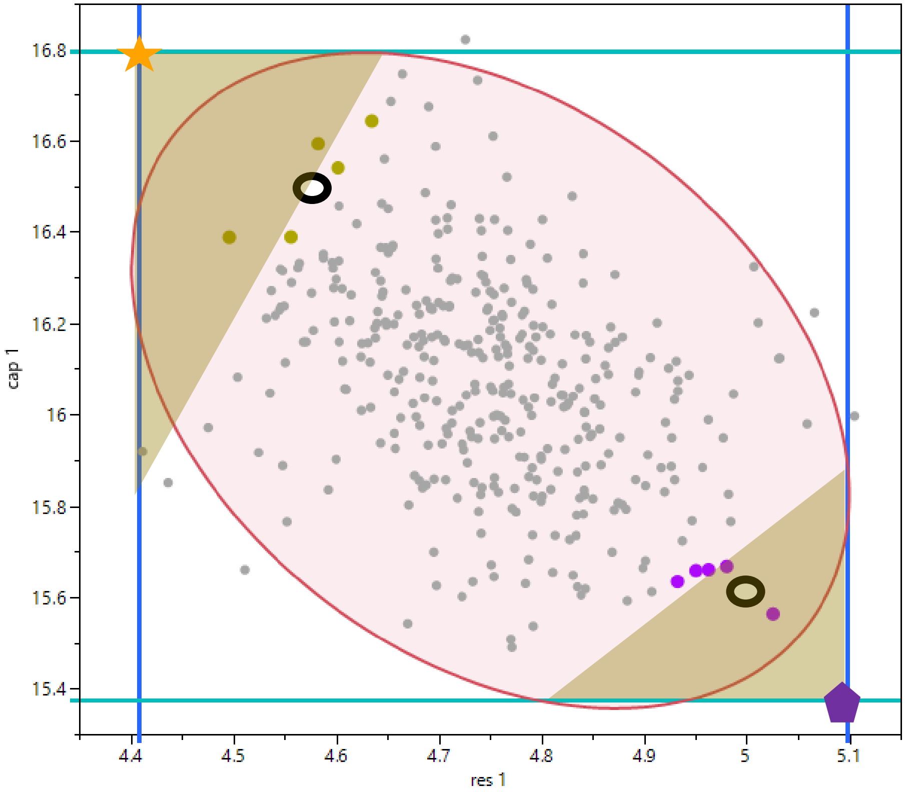
SOLUTION: K-NEAREST NEIGHBORS

- Idea:
  - Find the median +/- 3 std devs for each variable (could also use mean)
  - Define targeted corners – in this example, the low/high and the high/low corner.
  - Next, find the k-nearest neighbors to the targeted corners.
  - As an illustration, the 5 nearest neighbors are found
  - Next, find the average of the k nearest neighbors



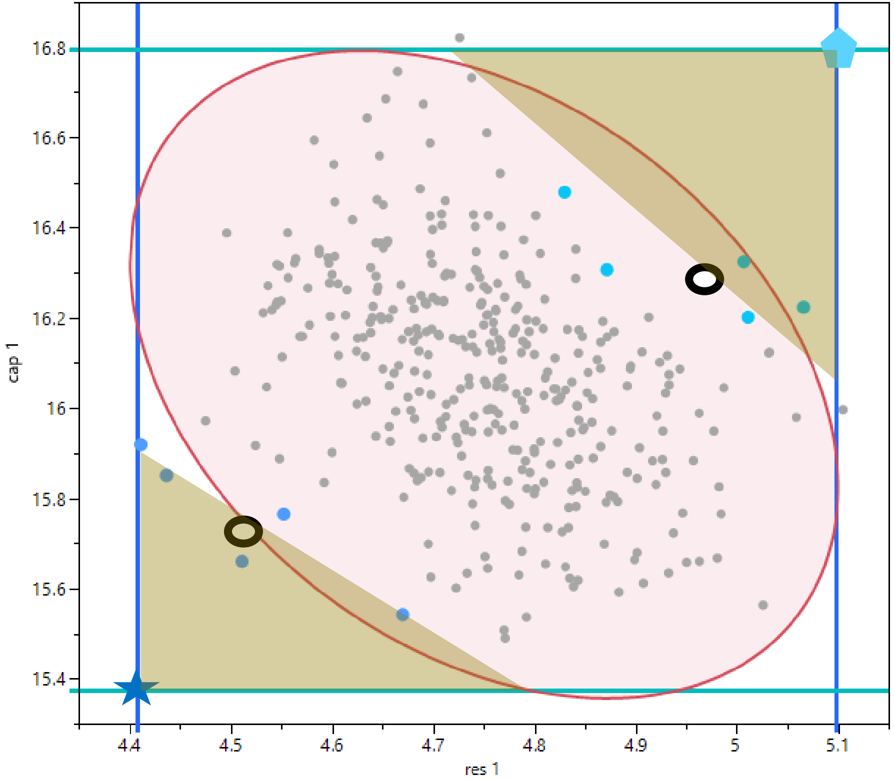
SOLUTION: K-NEAREST NEIGHBORS

- The average of the 5 nearest neighbors appears to be a reasonable worst-case bound
  - There are data points near the worst-case bound (as designed)
  - Not too conservative
  - Didn't have to worry about the distributions or correlations.



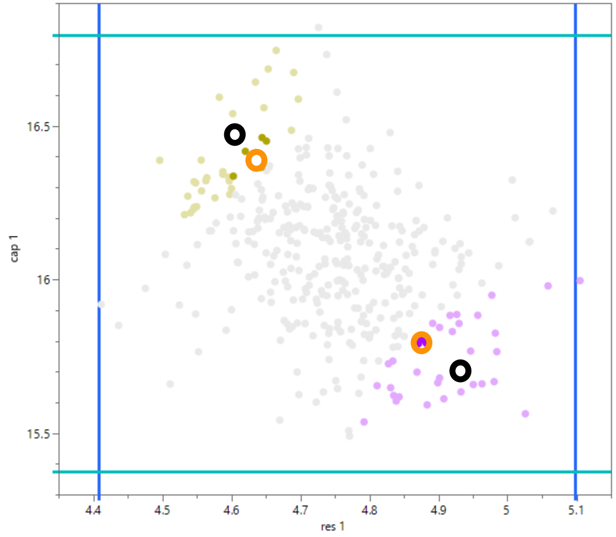
SOLUTION: K-NEAREST NEIGHBORS

- Let's say you were interested in the high/high and low/low worst-case bounds.
- The average of the 5 nearest neighbors appears to be a reasonable worst-case bound
  - In this example, the k-nearest neighbor approach is wonderful relative to our current approach (which is too conservative)



CHOICE OF K AND SHOULD WE AVERAGE?

- k may be based on confidence level, sample size, and philosophy
- Example – n = 1000 points. 95% confidence
  - 25th closest distance for the two corners
  - Average 23rd, 24th, 25th, 26th, 27th
- We have large sample sizes. We want to be a little bit conservative. We are not driven by 95% or 99% confidence levels.
  - *We may average the 1st through 25th points

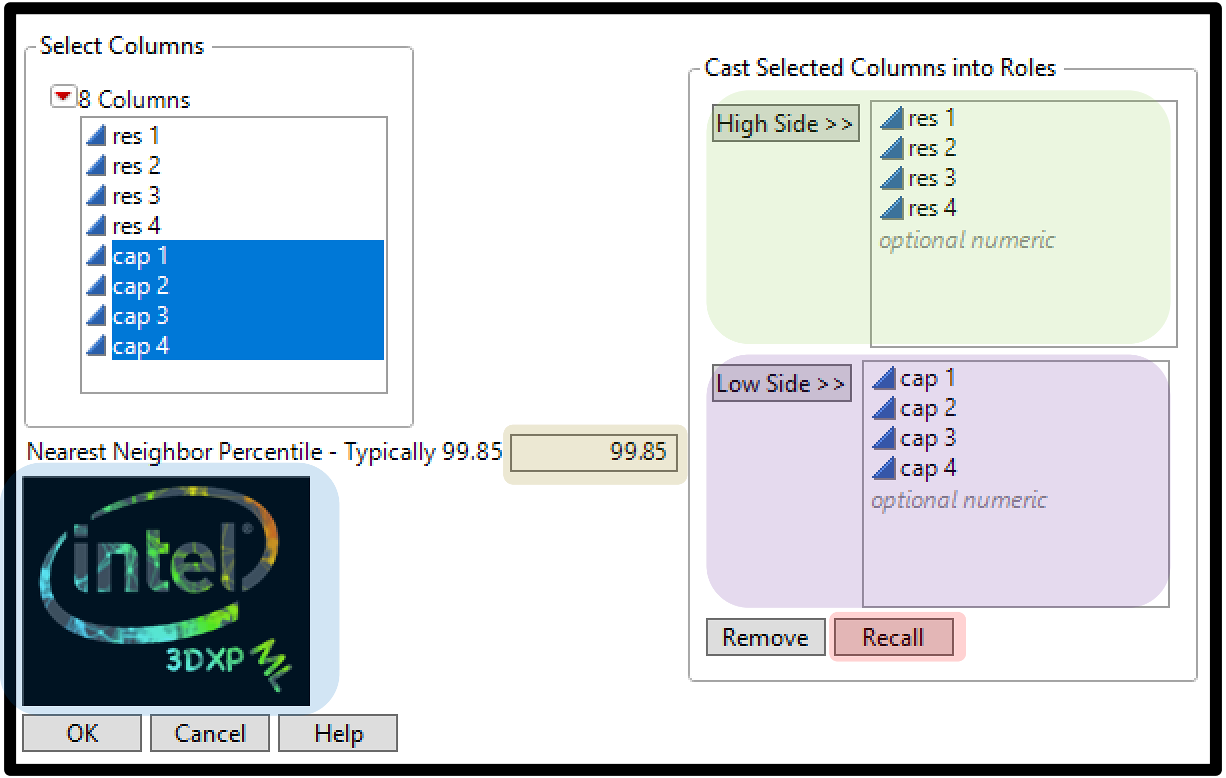


SOLUTION: K-NEAREST NEIGHBORS

- Pros:
  - Do not need to know the distribution of the variables
  - Can easily handle correlated / dependent variables
  - Can easily handle multiple variables
  - There is data “close” to the solution
  - We can build a script / add-in in JMP® to perform the calculations
- Con:
  - Requires a decent sized data set (sample size depends on desired confidence level)

ADD-IN

- User Interface
- High / Low Side
- Confidence level (which drives the k)
- Recall button for convenience
- Our Team Logo



OUTPUT

	Parameter	Count	Median	Std Dev	Side	Num Neighbors Used	Bound	Neighbor Avg	Neighbor Z-score	Neighbor Values	Neighbor Indices
1	cap 1	370	16.0842307	0.23669112	low	5	15.3741573	15.7533605	1.398	[16.0800201163215, ...	[10, 80, 109, 126, 321]
2	cap 2	370	17.0965883	0.24345246	low	5	16.3662309	16.7775085	1.311	[16.6240376055925, ...	[10, 80, 109, 126, 321]
3	cap 3	370	16.6350773	0.94760356	low	5	13.7922666	15.1909094	1.524	[15.5018280414842, ...	[10, 80, 109, 126, 321]
4	cap 4	370	16.2480584	1.19777908	low	5	12.6547212	14.6032812	1.373	[15.374429718801, ...	[10, 80, 109, 126, 321]
5	res 1	370	4.75297882	0.11516757	high	5	5.09848153	4.82126751	0.593	[4.81324536118704, ...	[10, 80, 109, 126, 321]
6	res 2	370	4.79113567	0.19302804	high	5	5.37021979	5.03521616	1.264	[5.09140581005553, ...	[10, 80, 109, 126, 321]
7	res 3	370	4.65214455	0.52302605	high	5	6.2212227	5.09872891	0.854	[5.6330626662091, ...	[10, 80, 109, 126, 321]
8	res 4	370	4.33644006	0.14874524	high	5	4.7826758	4.42175352	0.574	[4.4187660116568, ...	[10, 80, 109, 126, 321]

- A table is provided
- Bound = Median + or - 3 * Std Dev (targeted corner)
- Neighbor Indices -> the rows of the data that are the 5 nearest neighbors
- Neighbor Z-score -> Z-score for the Neighbor Avg
    - This helps us know if our current method is conservative or not

OUTPUT

	Parameter	Count	Median	Std Dev	Side	Num Neighbors Used	Bound	Neighbor Avg	Neighbor Z-score	Neighbor Values	Neighbor Indices
1	cap 1	370	16.0842307	0.23669112	low	5	15.3741573	15.7533605	1.398	[16.0800201163215, ...	[10, 80, 109, 126, 321]
2	cap 2	370	17.0965883	0.24345246	low	5	16.3662309	16.7775085	1.311	[16.6240376055925, ...	[10, 80, 109, 126, 321]
3	cap 3	370	16.6350773	0.94760356	low	5	13.7922666	15.1909094	1.524	[15.5018280414842, ...	[10, 80, 109, 126, 321]
4	cap 4	370	16.2480584	1.19777908	low	5	12.6547212	14.6032812	1.373	[15.374429718801, ...	[10, 80, 109, 126, 321]
5	res 1	370	4.75297882	0.11516757	high	5	5.09848153	4.82126751	0.593	[4.81324536118704, ...	[10, 80, 109, 126, 321]
6	res 2	370	4.79113567	0.19302804	high	5	5.37021979	5.03521616	1.264	[5.09140581005553, ...	[10, 80, 109, 126, 321]
7	res 3	370	4.65214455	0.52302605	high	5	6.2212227	5.09872891	0.854	[5.6330626662091, ...	[10, 80, 109, 126, 321]
8	res 4	370	4.33644006	0.14874524	high	5	4.7826758	4.42175352	0.574	[4.4187660116568, ...	[10, 80, 109, 126, 321]

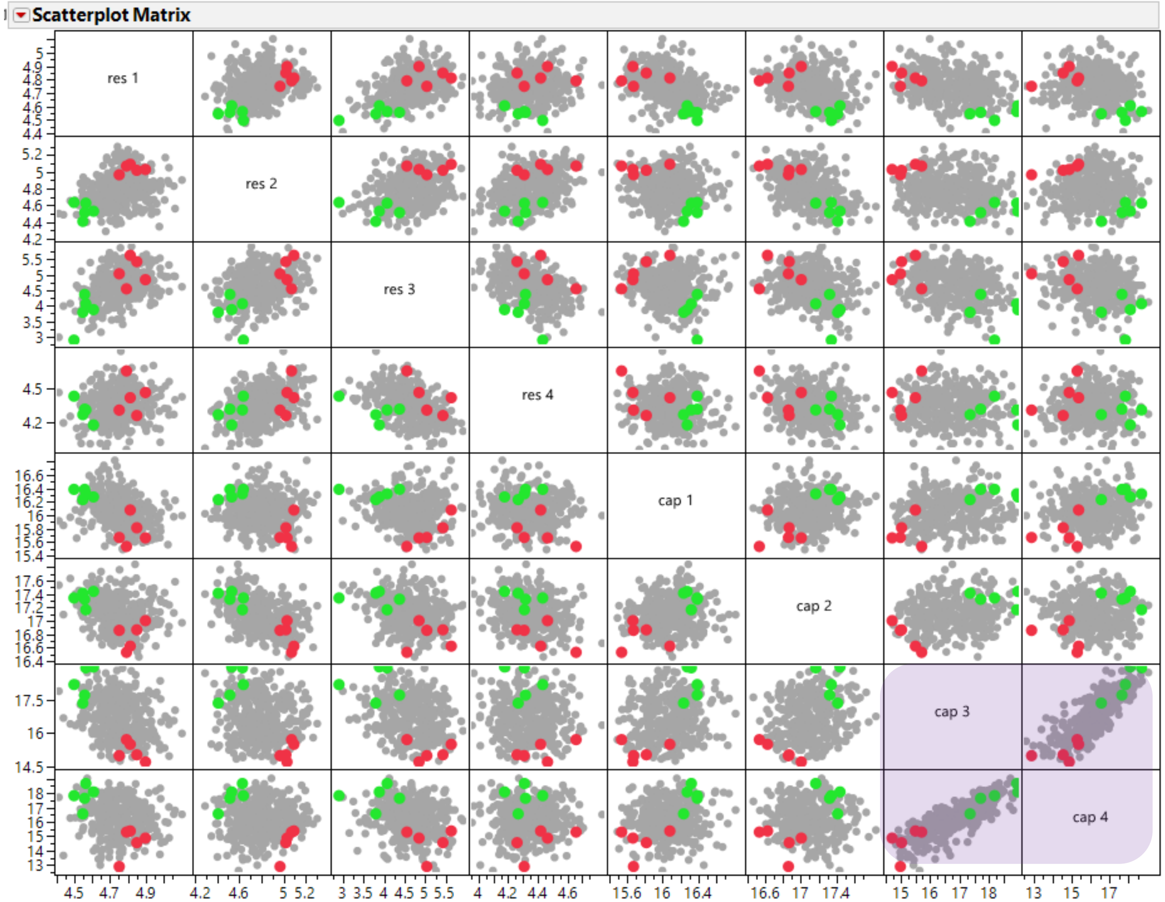
- The 5 rows of data (Neighbor Indices) are selected as part of the output so it is easy to color-code the points.

- In this example, I run the add-in twice.
  - Resistance high, capacitance low
  - Resistance low, capacitance high

	res 1	res 2	res 3	res 4	cap 1	cap 2	cap 3	cap 4
105	5.02586...	4.8942381681	4.9371634472	4.3182663507	15.563...	16.814560715	16.644837783	15.948...
106	4.87873...	4.7509117398	5.0804679806	4.3319742289	15.803...	17.122181328	15.926002033	15.825...
107	4.73546...	4.373291752	4.1062805249	4.314755496	16.066...	17.29324638	18.213252478	17.540...
108	4.84391...	4.8247083099	4.2350969454	4.3366141629	15.972...	16.988901546	18.612577591	18.651...
109	4.89900...	5.0290854243	4.8457331288	4.4651151647	15.663...	17.004434815	14.713915973	14.879...
110	4.98062...	4.8571890774	5.5407293108	4.1381344855	15.667...	17.13371853	15.261269226	13.739...
111	4.96309...	4.6216338752	4.3969595969	4.4226422587	15.660...	16.947239266	17.827760995	16.387...
112	4.79169...	4.5151569215	5.1117486139	3.9544984814	15.735...	17.607730324	16.345028756	15.956...
113	4.89484...	4.95417428	5.4210958221	4.3991590768	16.086...	16.934412206	17.36369678	17.381...
114	4.84213...	5.2145550202	4.9588187686	4.4358926514	16.005...	17.15718536	16.45133484	15.922...
115	4.83525...	4.8086797868	4.4816242063	4.4010138318	15.873...	17.125432668	18.071551954	17.588...
116	4.64358...	4.5486822131	4.2392928911	4.143455398	16.366...	17.407907924	16.879875634	14.633...
117	4.87650...	4.8017485597	4.4675954313	4.2731483606	15.949...	17.284079121	16.234871017	16.598...
118	4.98453...	4.8912727238	5.0077706239	4.4358310453	15.765...	16.99886692	17.53109205	17.622...
119	4.93301...	4.4943340083	4.1406931412	4.4831475847	16.051...	17.157176872	16.229581198	15.885...
120	4.73818...	4.5697461016	4.556376243	4.3326254382	16.732...	16.722257828	15.456124521	15.892...
121	4.92099...	4.6274017071	5.2724783825	4.1826395523	15.982...	17.24636493	16.032728747	15.484...
122	4.84337...	4.7939199528	5.1602299294	4.2018567551	15.887...	17.065464163	16.425543734	16.282...
123	4.73079...	4.5618454587	4.6505673275	4.1817985561	15.963...	17.574534251	18.144761694	17.252...
124	4.80956...	4.4759052198	5.0016508039	4.1033954486	16.027...	17.394319011	17.692529323	17.829...
125	4.84085...	4.586352997	4.672548348	4.32059825	16.287...	17.845443825	16.750775936	15.439...
126	4.85045...	5.0192383418	5.4288749704	4.2591628111	15.815...	16.867689445	15.033398794	14.562...
127	4.82179...	4.9035564699	4.2262461044	4.686431501	16.016...	16.99144021	16.265080302	15.795...
128	4.76475...	4.6299405972	3.5651862659	4.4544255212	15.987...	17.389826751	15.935040461	16.090...

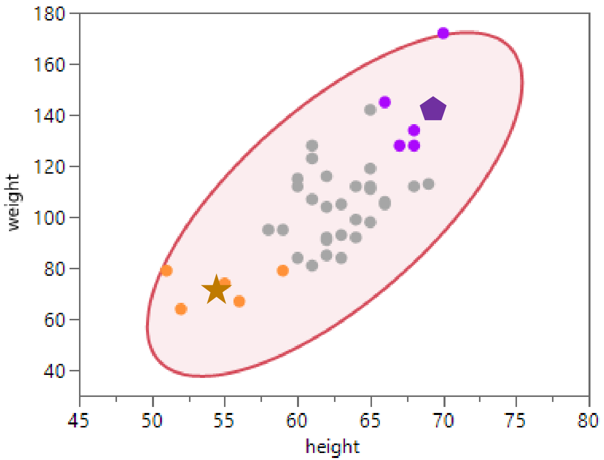
OUTPUT

- Green points represent low res / high cap
- Red points represent high res / low cap



MOTIVATING PROBLEM – A (MADE-UP) DIFFERENT INDUSTRY EXAMPLE

- If we use JMP® *BigClass.JMP* data set (and use weight for waist size), we could determine height and weight bounds that capture 99% of the population.
- Using the add-in (nearest neighbor approach), the high/high is (67.8,141.4) and the low/low is (54.6,72.6)



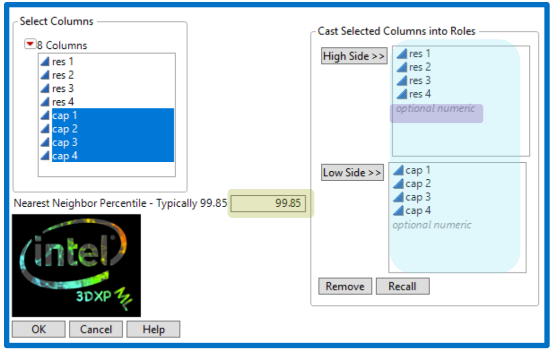
	Parameter	Count	Median	Std Dev	Side	Num Neighbors Used	Bound	Neighbor Avg	Neighbor Z-score	Neighbor Values	Neighbor Indices
1	height	40	63	4.24233849	high	5	75.7270155	67.8	1.131	[66, 67, 68, 68, 70]	[4, 30, 37, 39, 40]
2	weight	40	105	22.201871	high	5	171.605613	141.4	1.64	[145, 128, 128, 134, 172]	[4, 30, 37, 39, 40]

Parameter	Count	Median	Std Dev	Side	Num Neighbors Used	Bound	Neighbor Avg	Neighbor Z-score	Neighbor Values	Neighbor Indices
height	40	63	4.24233849	low	5	50.2729845	54.6	1.98	[55, 52, 51, 56, 59]	[3, 5, 8, 11, 15]
weight	40	105	22.201871	low	5	38.3943869	72.6	1.459	[74, 64, 79, 67, 79]	[3, 5, 8, 11, 15]

SCRIPT HIGHLIGHTS - INTERFACE

- Lines 210,214: Collect inputs
  - hLst (high side) & lLst (low side)
  - Note that only numeric variables are allowed
- Also collecting “nn_quant”
- Lines 222-223: Recall feature

```
207                 PanelBox("Cast Selected Columns into Roles",
208                     LineupBox(NCol(2), Spacing(3),
209                         Button Box("High Side >>", hLst << Append(usrselLst << Get Selected)),
210                         hLst = Col List Box(width(boxwidth), nLines(min(nc,10)), Numeric),
211                     ),
212                     LineupBox(NCol(2), Spacing(3),
213                         Button Box("Low Side >>", lLst << Append(usrselLst << Get Selected)),
214                         lLst = Col List Box(width(boxwidth), nLines(min(nc,10)), Numeric),
215                     ),
216                     HListBox(
217                         Button Box("Remove",
218                             hLst << Remove Selected;
219                             lLst << Remove Selected;
220                             ),
221                         Button Box("Recall",
222                             hLst<<Append(hdatlst);
223                             lLst<<Append(ldatlst);
224                         ),
225                     ),
226                 ),
227             ),
228             HListBox(
229                 ButtonBox("OK",
230                     nn_quant = quantile_box << Get;
231                     hdatlst = Eval(hLst << GetItems);
232                     ldatlst = Eval(lLst << GetItems);
233                     usrDlg << CloseWindow;
234                     analysisScript;
235                 ),
236                 ButtonBox("Cancel", usrDlg << CloseWindow; Throw()),
237                 ButtonBox("Help", helpScript)
238             )
```



SCRIPT HIGHLIGHTS – DATA QUALITY CHECKS: VALIDATE INPUT DATA FROM USER

- Data Quality Checks
  - Lines 261-275: Nearest neighbor percent between 0 and 100 (percent)
  - Lines 277-291: Must have selected at least one column to be high/low

```
260        // start data quality checks
261        if(nn_quant >= 100 | nn_quant <= 0,
262            // send message and throw
263            Caption(
264                {200, 200},
265                "Nearest Neighbor Percentile must be > 0 and < 100 - typical is 99.85",
266                Font("Arial Black"),
267                Font Size(16),
268                TextColor("red"),
269                BackColor("black"),
270                Spoken(0)
271            );
272            Wait(7);
273            Caption(Remove);
274            throw("Nearest Neighbor Percentile must be > 0 and < 100 - typical is 99.85");
275        );
```

```
277        if((nitems(hdatlst) == 0) & (nitems(ldatlst) == 0),
278            // send message and throw
279            Caption(
280                {200, 200},
281                "You must choose at least one column on the high or low side",
282                Font("Arial Black"),
283                Font Size(16),
284                TextColor("red"),
285                BackColor("black"),
286                Spoken(0)
287            );
288            Wait(7);
289            Caption(Remove);
290            throw("You must choose at least one column on the high side");
291        );
```

SCRIPT HIGHLIGHTS – KEY FUNCTION

- *sumcols* loops through the passed columns; it takes either ‘low’ or ‘high’ and returns a dictionary or associative array of information with the key being the column name passed and a list of information including:
  - Data type, Modeling type
  - Count (# of points), Median, StdDev
  - Bound (Med +- 3 sigma)
  - Side (low, high) – which is passed
  - Data (a matrix) – the column of data
  - stdize (matrix) ((data – median) / StdDev +-3)^2
    - Recenters data to +- 3 sigma (high/low) from median

```
    // 3. calculate the median and sigma for each variable
    // 4. calculate either the low corner or high corner
    // 3./4. combined into one function (sumcols)
    loaa = sumcols(ldatlst, "low");
    hiaa = sumcols(hdatlst, "high");

```

```
//SCRIPT FUNCTIONS
sumcols = function({cols, side}, {default local},
    //cols = the column list to convert into a dictionary
    //side = the side to calculate = "low" or "high" which is median - 3 sigma or median + 3 sig
    //return = {data type, modeling type, count, median, stddev, bound (med +- 3 stddev -depend
    aa = associative array();
    for(idx=1, idx<=nitems(cols), idx++,
        dtype = column(cols[idx]) << get data type;
        mtype = column(cols[idx]) << get modeling type;
        data = column(cols[idx]) << get values;
        Summarize(
            ccnt = Count(Column(cols[idx])),
            cmed = Quantile(Column(cols[idx]), 0.5),
            cstd = Std Dev(Column(cols[idx]))
        );
        if(side == "low",
            bound = cmed - 3 * cstd;
            stdize = power((((data - cmed)/cstd) + 3),2);
        , //else high side
            bound = cmed + 3 * cstd;
            stdize = power((((data - cmed)/cstd) - 3),2);
        );
        aa[cols[idx]] = evallist({dtype, mtype, ccnt, cmed, cstd, bound, side, data, stdize});
    );
    return(aa);
);
```

SCRIPT HIGHLIGHTS – SUM DATA AND CALCULATE DISTANCE

- Line 332-341 sum the square of the median +- 3 sigma depending on high/low
- Line 342 calculates the distance from the 3 sigma point of the data
- Line 357 returns the rank of the distance matrix and
- Line 358 finds the points (neighbors) closest to the 3 sigma points

```
330     // 5. for each data point find the unit distance to the combined results from #4
331     // just need to sum the returned standardized matrices for each variable
332     sumdata = 0;
333     sumcounts = 0;
334     for(idx=1, idx<=nitems(ldatlst), idx++,
335         sumdata = sumdata + loaa[ldatlst[idx]][9];
336         sumcounts = sumcounts + loaa[ldatlst[idx]][3];
337     );
338     for(idx=1, idx<=nitems(hdatlst), idx++,
339         sumdata = sumdata + hiaa[hdatlst[idx]][9];
340         sumcounts = sumcounts + hiaa[hdatlst[idx]][3];
341     );
342     sumdata = sqrt(sumdata);
343     avgcount = floor(sumcounts / (nitems(ldatlst) + nitems(hdatlst)));
```

```
351     // 6. order the rows (neighbors) from smallest to largest - don't n
352     // 7. find the number of nearest neighbors to take (minimum of 5)
353     // 8. take 5 neighbors at the minimum
354     calc_neighbor = floor((1 - (nn_quant/100)) * avgcount);
355     num_neighbor = max(5, calc_neighbor);
356     // find the ranking of the matrix to use to find the indices where
357     mtx_rnk = ranking(sumdata);
358     min_row_vector = loc(mtx_rnk <= num_neighbor);
```

SCRIPT HIGHLIGHTS – CALCULATE NEIGHBORS, Z-SCORE AND RETURN

- Finally, lines 346-380 loop through the low/high side and return in a dictionary:
    - Point (values) which are closest to 3 sigma points
    - Average of the minimum neighbors
    - Z-score of the median from the avg. neighbors

```
363     lresultaa = associative array();
364     for(idx=1, idx<=nitems(ldatlst), idx++,
365         median = loaa[ldatlst[idx]][4];
366         sigma = loaa[ldatlst[idx]][5];
367         min_neighbors = loaa[ldatlst[idx]][8][min_row_vector];
368         avg_neighbors = mean(min_neighbors);
369         zscore = abs(round((median - avg_neighbors) / sigma, 3));
370         lresultaa[ldatlst[idx]] = evallist({min_neighbors, avg_neighbors, zscore, min_row_vector});
371     );
372     hresultaa = associative array();
373     for(idx=1, idx<=nitems(hdatlst), idx++,
374         hmedian = hiaa[hdatlst[idx]][4];
375         hsigma = hiaa[hdatlst[idx]][5];
376         hmin_neighbors = hiaa[hdatlst[idx]][8][min_row_vector];
377         havg_neighbors = mean(hmin_neighbors);
378         hzscore = abs(round((hmedian - havg_neighbors) / hsigma, 3));
379         hresultaa[hdatlst[idx]] = evallist({hmin_neighbors, havg_neighbors, hzscore, min_row_vector});
380     );
```

SUMMARY

- Motivating problem – finding the simultaneous "worst-case" bounds
  - Current solution is too conservative – costing us money and time
  - Data may or may not follow the multivariate normal distribution
  - Data is not independent
  - Frequency of use requires "simple" solution – preferably with JMP®
- Solution – build an JMP® Add-in that is easy to use
  - Uses k nearest neighbors concepts
  - Output is easy to understand.

intel®