Techniques for Effective Street Centerline Mapping

28TH ANNUAL
SCAUG
CONFERENCE
SAN ANTONIO
APRIL 9-13, 2018

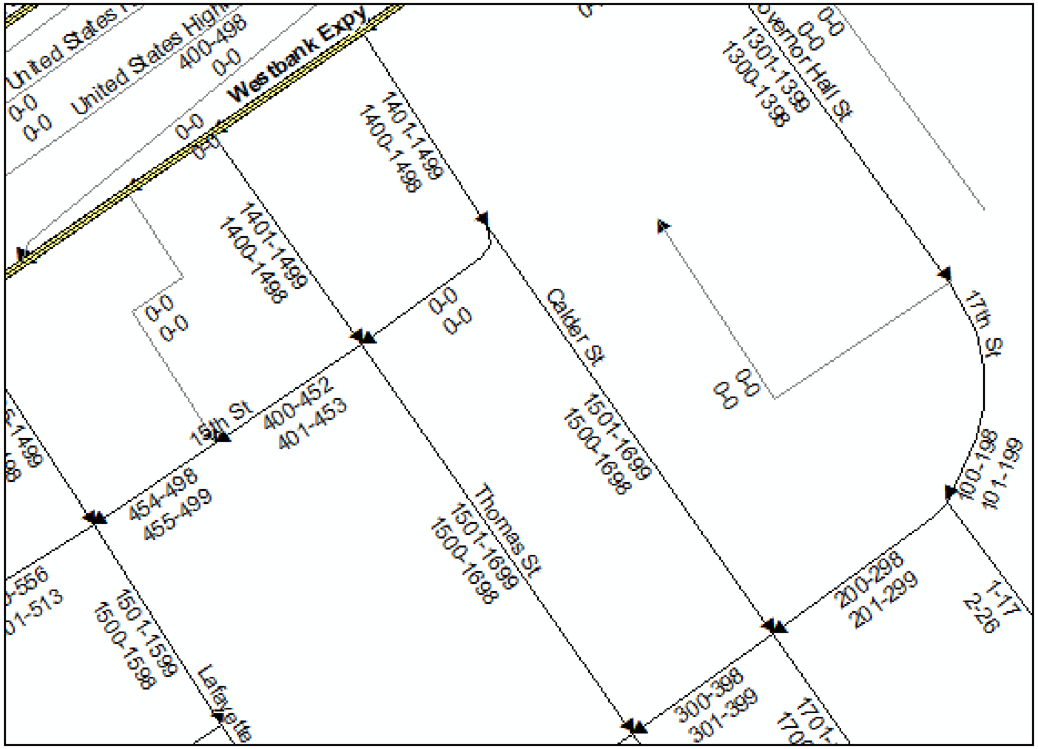
Techniques for Effective Street Centerline Mapping

- Symbology
- Queries
- Scripts
- Geoprocessing Tools
- Topology
- Editing Tips

Symbology

Street Centerline Symbology

- Symbolize by street type, direction, city, etc. for general checking
- Symbolize lines with Unique Value, Many Fields to show left/right city names, left/right zip codes, etc.



Street Symbology for QC

- To show street direction:
  - Copy the layer and move the copy underneath the main layer
  - Symbolize the copy with a simple black line with an arrow at the end (in the Symbol Selector, toward the bottom)
  - You will be able to see the arrow underneath the main street symbology

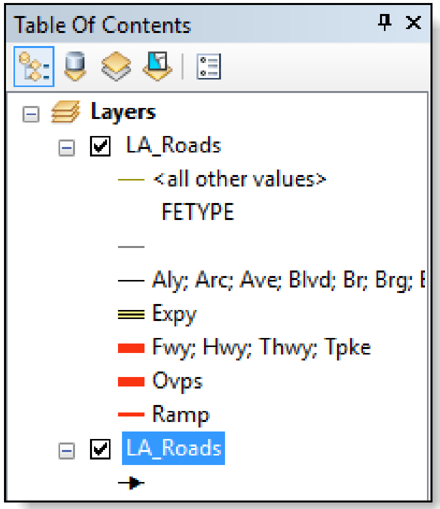
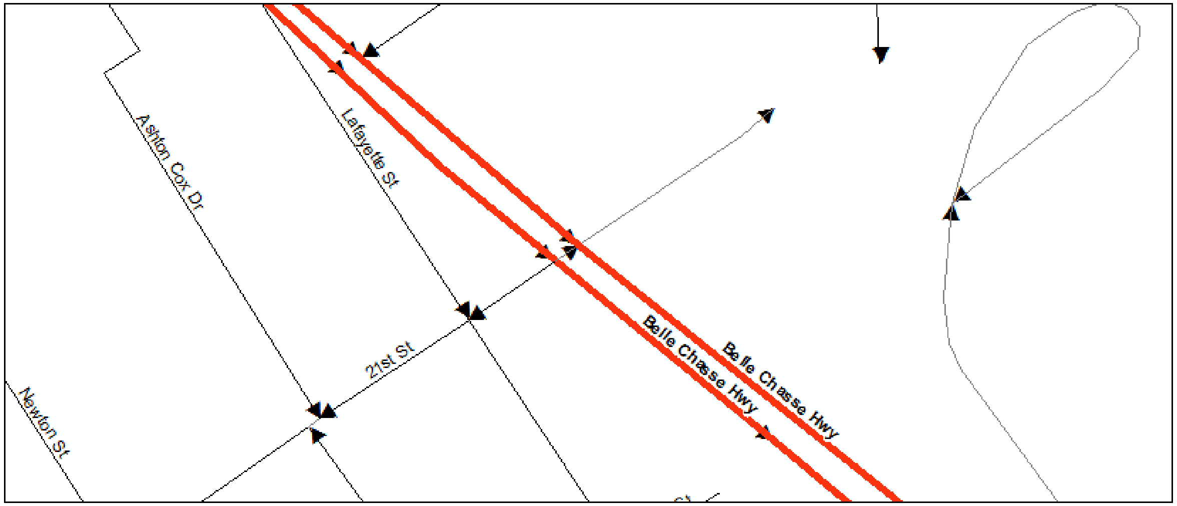


Table Of Contents

⊟ Layers
- ⊟ ☑ LA_Roads
  - <all other values>
  - FETYPE
  - Aly; Arc; Ave; Blvd; Br; Brg; E
  - Expy
  - 🟥 Fwy; Hwy; Thwy; Tpke
  - 🟥 Ovps
  - Ramp
- ⊟ ☑ LA_Roads

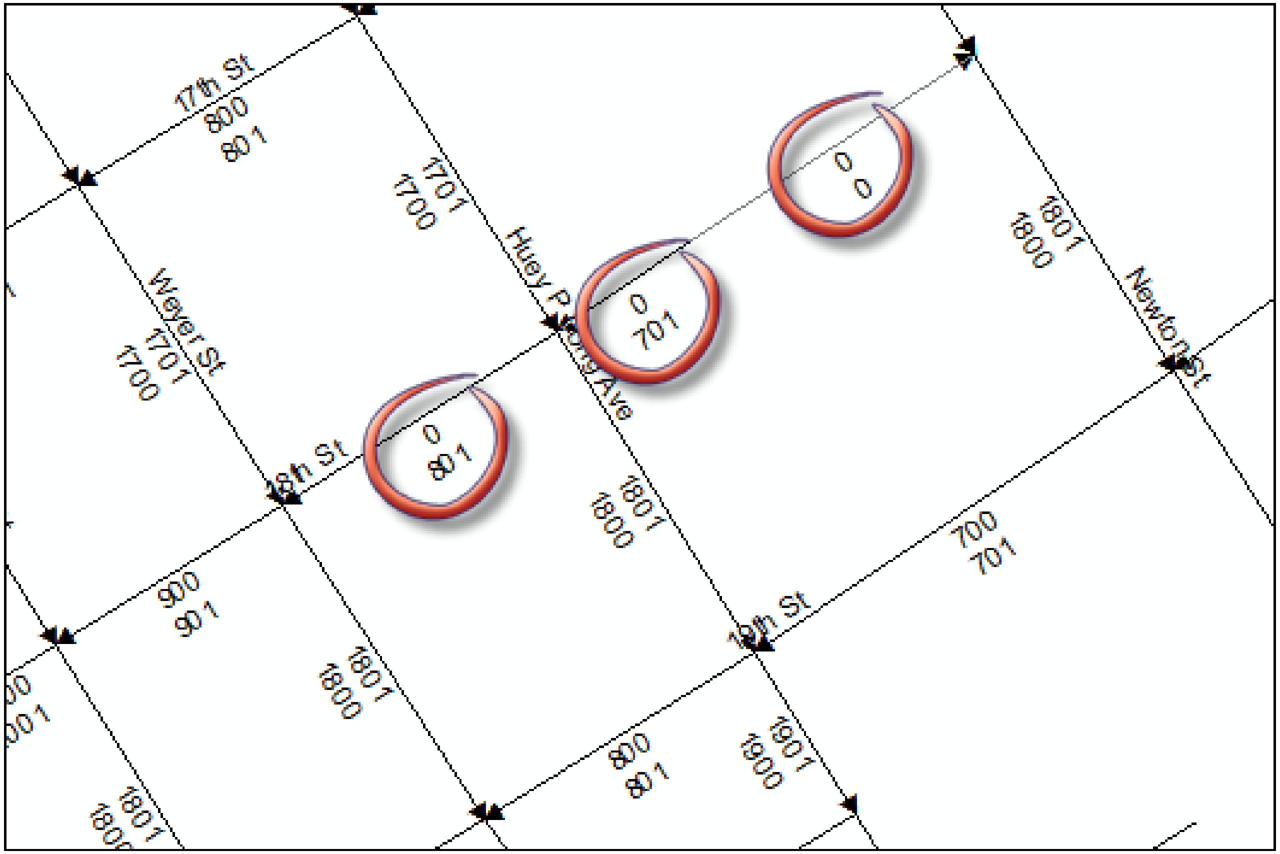


Street Centerline Symbology

- Symbolize streets by:
  - One-way
  - Start address (colors)
  - Jurisdictions, zones, …
- Keep them all as different layers inside your edit MXD.
- Comparing two street layers with thick & light lines on bottom and thin and intense lines on top to see both at one time

Street Labels for QC

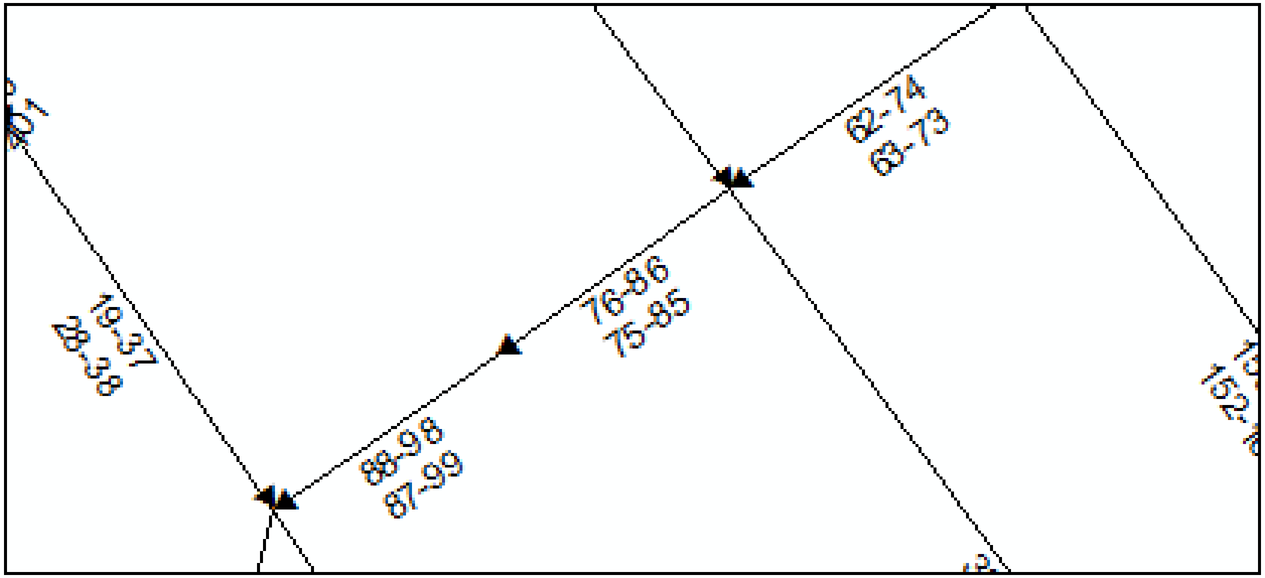
- Address ranges: label with address minimum values (Left-From field) to show anomalies and nulls



Label Expressions

- Use label expressions to get all information from multiple columns
  - Example for address ranges:

```
[LeftFrom] + “-” + [LeftTo] + “\n” +
[RightFrom] + “-” + [RightTo]
```



Queries

Multiple Layers with Filters

- Use definition queries on copies of a layer to give different features different symbols
- Turn the layers on and off separately

  Hint: Use group layers to organize the layers

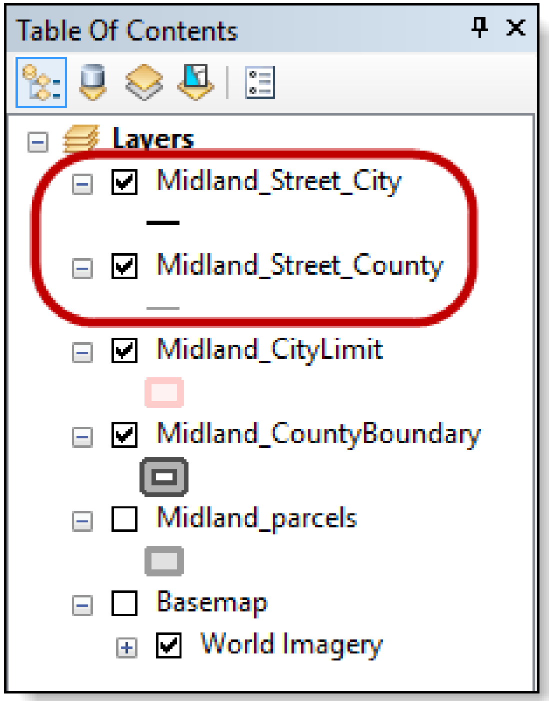
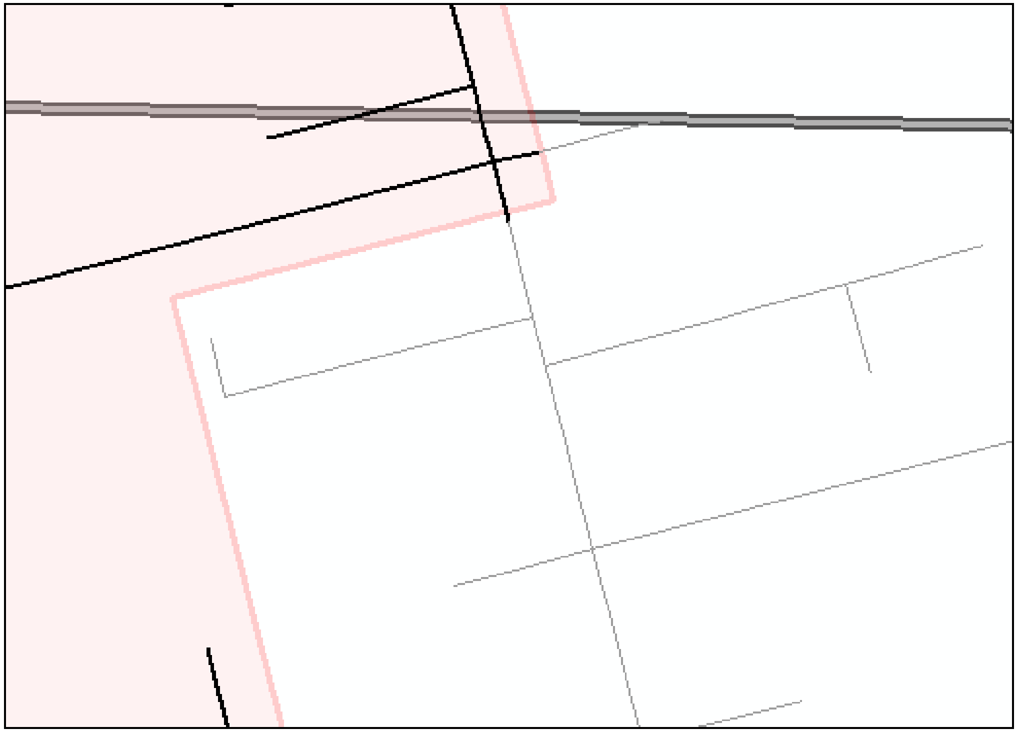


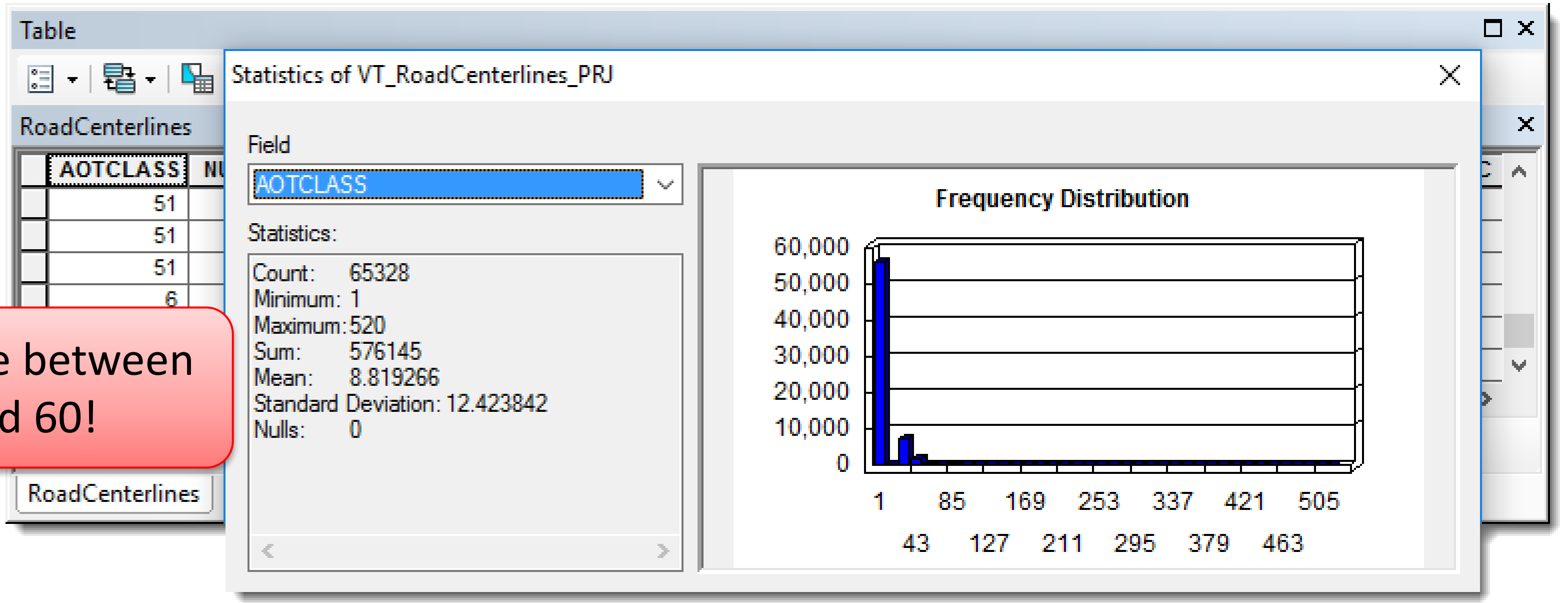
Table Of Contents

- Layers
  - ☑ Midland_Street_City
  - ☑ Midland_Street_County
  - ☑ Midland_CityLimit
  - ☑ Midland_CountyBoundary
  - □ Midland_parcels
  - □ Basemap
    - ☑ World Imagery



Finding Errors

- Sort your table to find values too big or too small, and to find Nulls.
- Use **Summarize** to see mistakes in codes or types… where you should see just a few values.
- Use **Statistics** to find numerical outliers



Statistics of VT_RoadCenterlines_PRJ

Field: AOTCLASS

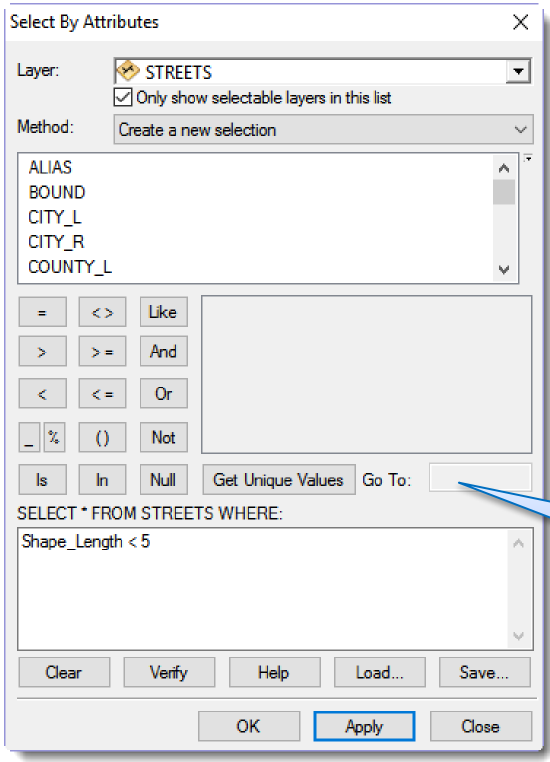
Statistics:

```
Count:     65328
Minimum:  1
Maximum: 520
Sum:       576145
Mean:      8.819266
Standard Deviation: 12.423842
Nulls:     0
```

Should be between 1 and 60!

Selecting Line Fragments

- Often, really small streets (like shorter than 5 feet) are digitizing errors.
- Write a SQL query to select the streets shorter than 5 feet



Select By Attributes

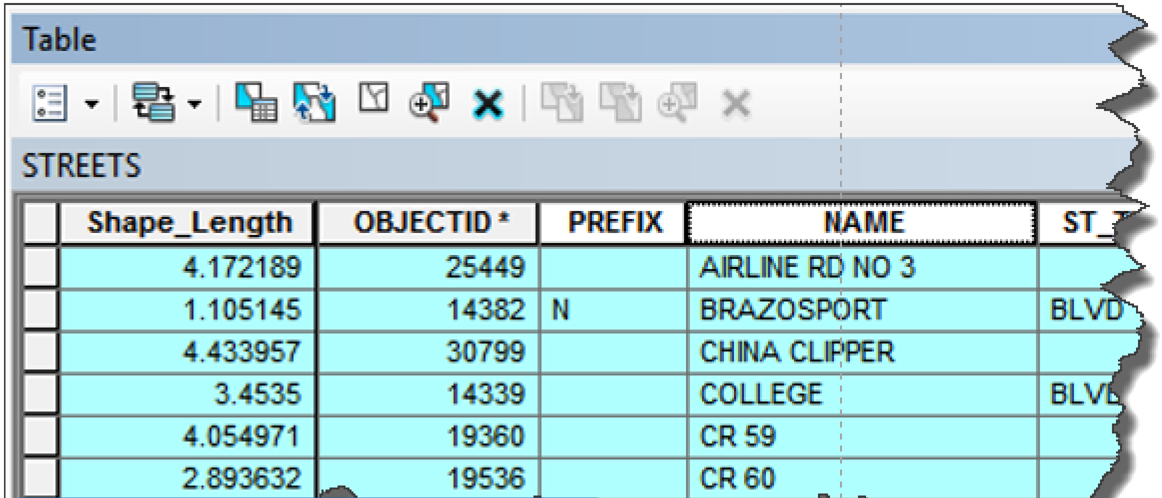
Layer: STREETS

☑ Only show selectable layers in this list

Method: Create a new selection

SELECT * FROM STREETS WHERE:

Shape_Length < 5



Table

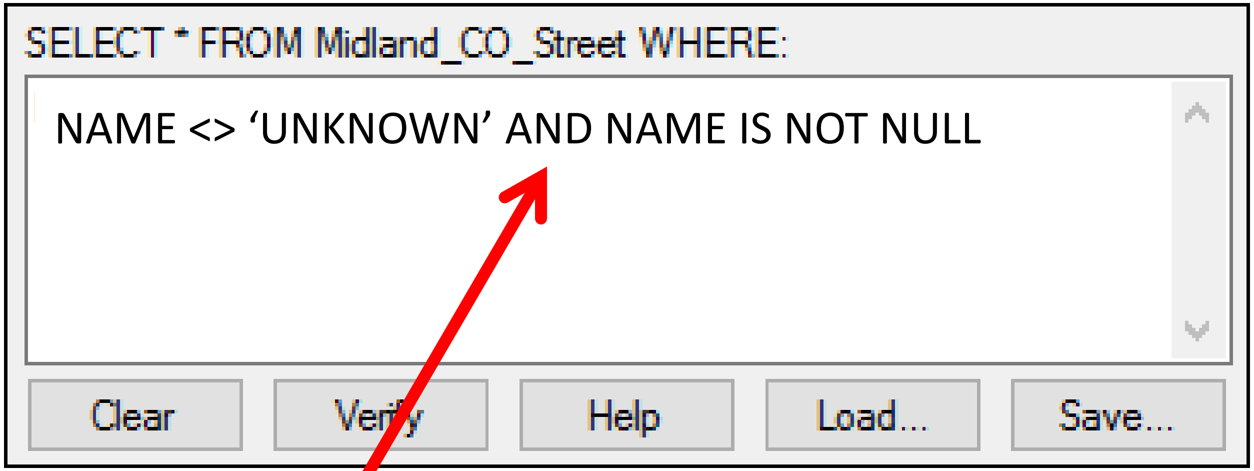
STREETS

Shape_Length	OBJECTID *	PREFIX	NAME	ST_
4.172189	25449		AIRLINE RD NO 3	
1.105145	14382	N	BRAZOSPORT	BLVD
4.433957	30799		CHINA CLIPPER	
3.4535	14339		COLLEGE	BLV
4.054971	19360		CR 59	
2.893632	19536		CR 60	

Shape_Length < 5

Finding Known Street Names

- Use the “not equal to” operator to search for all street names that are not listed as unknown or NULL (left blank)



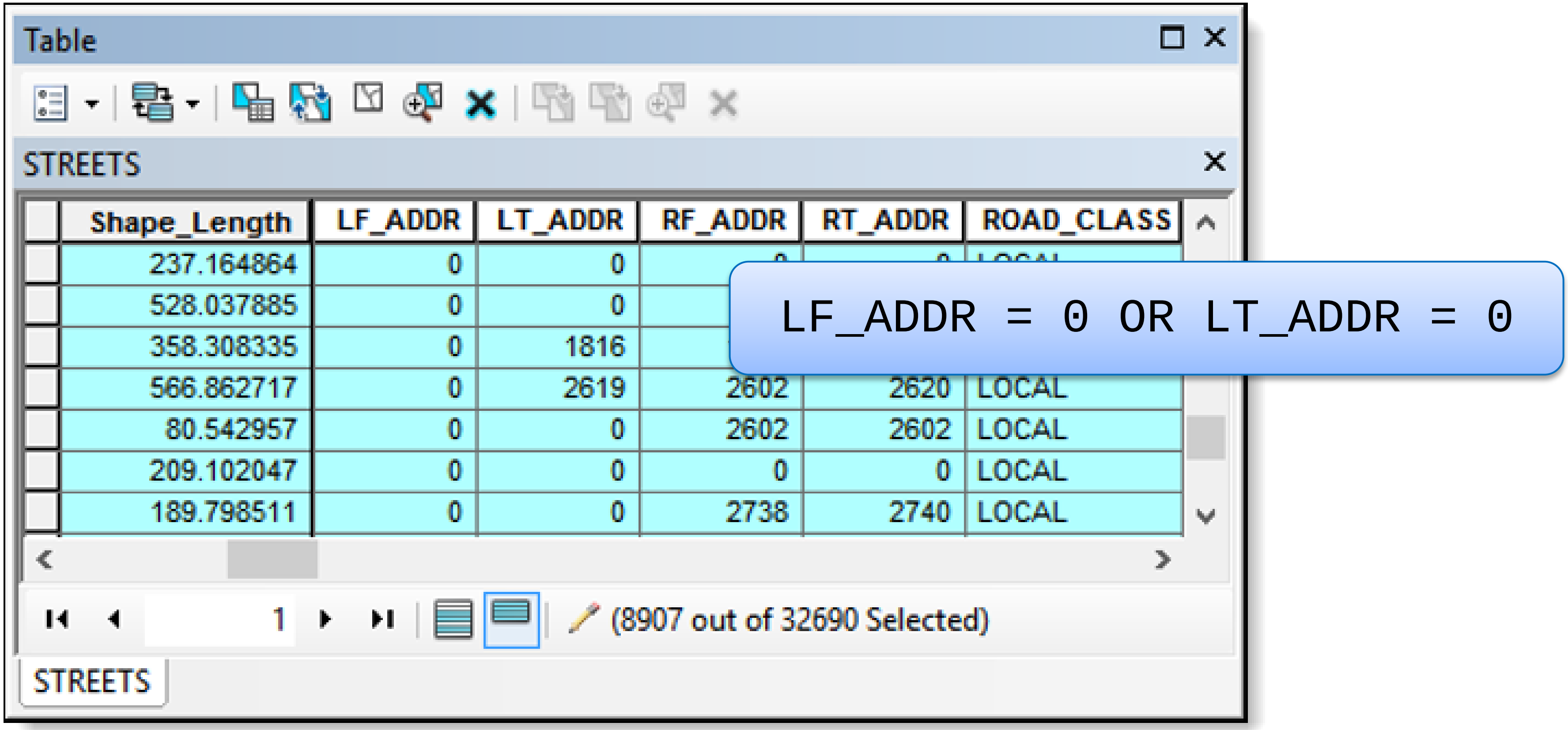
SELECT * FROM Midland_CO_Street WHERE:

```
NAME <> ‘UNKNOWN’ AND NAME IS NOT NULL
```

- Use a Boolean AND to match both values (all roads that are NOT named “UNKNOWN” and at the same time are not NULL)

Find Incorrect Numerical Values

- Use equal to/not equal to in order to find FROM and TO values entered as 0, or any that were not entered as 0
  - Use OR to combine queries



Shape_Length	LF_ADDR	LT_ADDR	RF_ADDR	RT_ADDR	ROAD_CLASS
237.164864	0	0	0	0	LOCAL
528.037885	0	0			
358.308335	0	1816			
566.862717	0	2619	2602	2620	LOCAL
80.542957	0	0	2602	2602	LOCAL
209.102047	0	0	0	0	LOCAL
189.798511	0	0	2738	2740	LOCAL

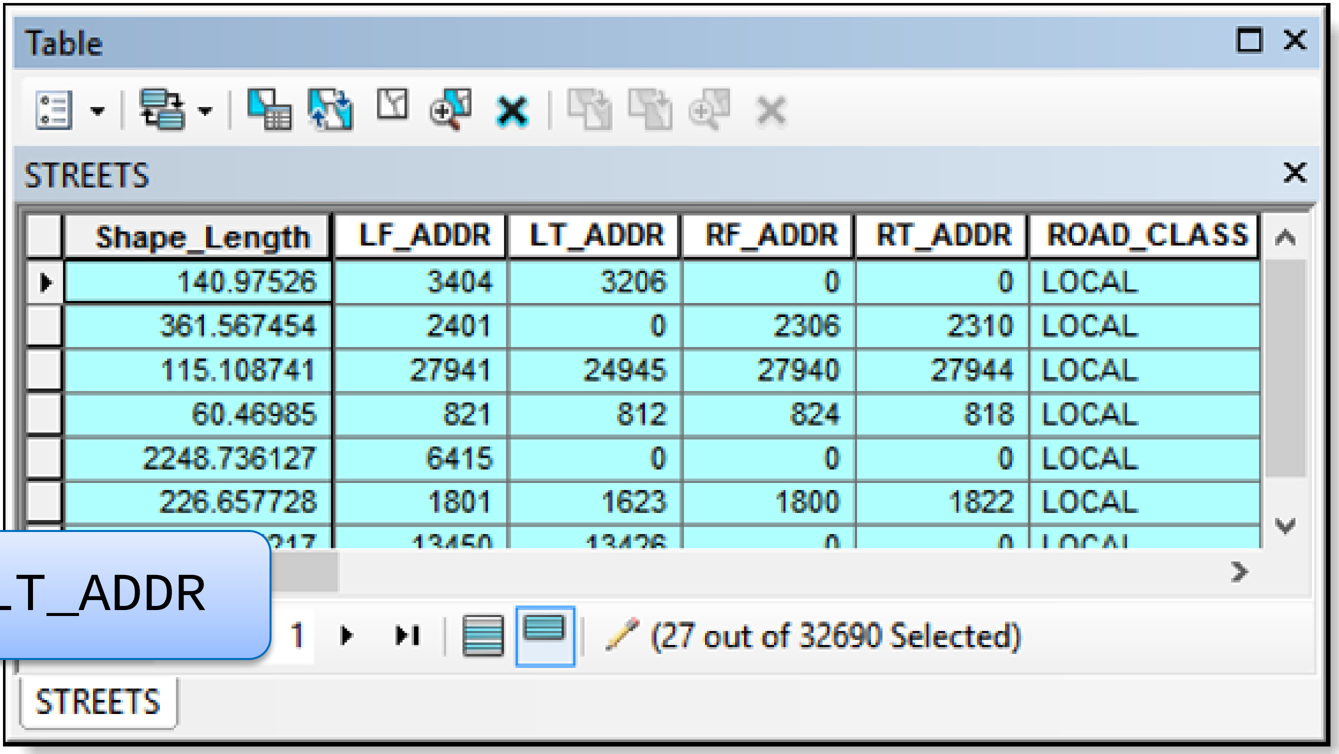
1 (8907 out of 32690 Selected)

```
LF_ADDR = 0 OR LT_ADDR = 0
```

Comparing From/To Values

- You can use mathematical operators to compare from and to values to find anomalies
- For example, you can search for cases where the FROM values are larger than the TO values

Only if address ranges are numeric columns!



Shape_Length	LF_ADDR	LT_ADDR	RF_ADDR	RT_ADDR	ROAD_CLASS
140.97526	3404	3206	0	0	LOCAL
361.567454	2401	0	2306	2310	LOCAL
115.108741	27941	24945	27940	27944	LOCAL
60.46985	821	812	824	818	LOCAL
2248.736127	6415	0	0	0	LOCAL
226.657728	1801	1623	1800	1822	LOCAL
217	13450	13426	0	0	LOCAL

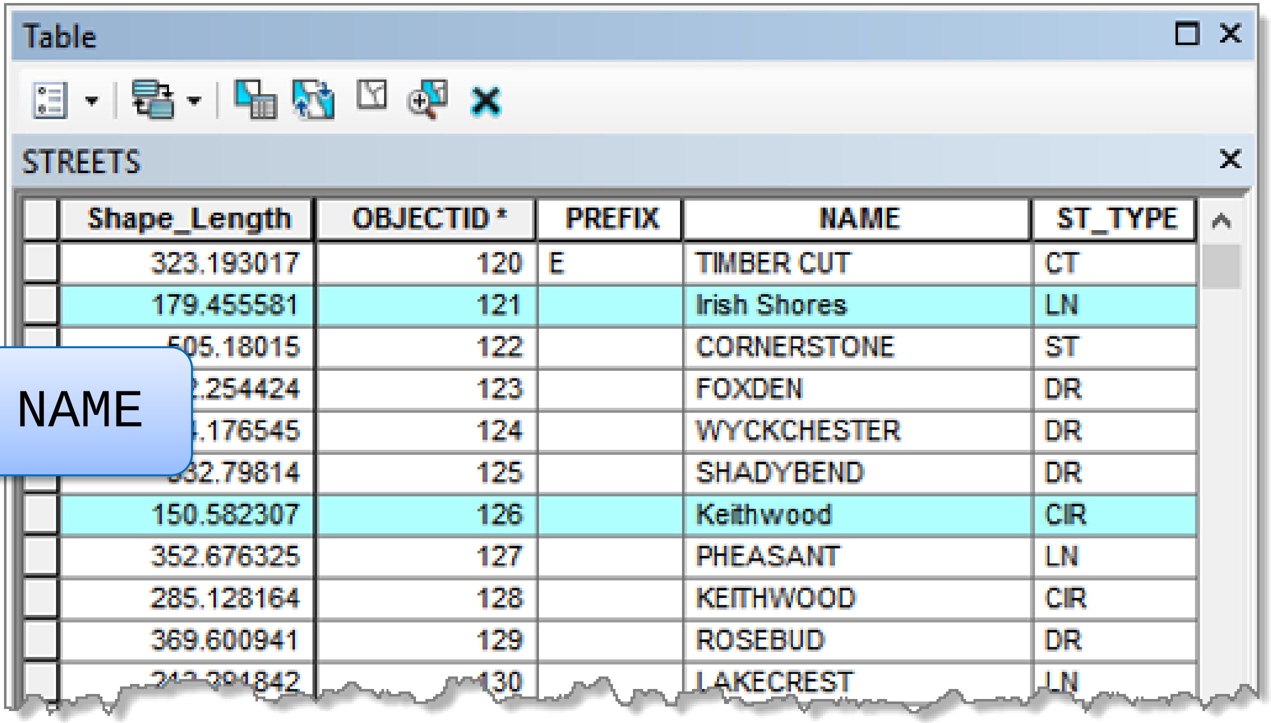
1 (27 out of 32690 Selected)

STREETS

LF_ADDR > LT_ADDR

Finding Case Problems

- When multiple users enter attributes, there may be differences in case
- If your street names are supposed to be all uppercase, and you want to search for any that are not, use **upper**

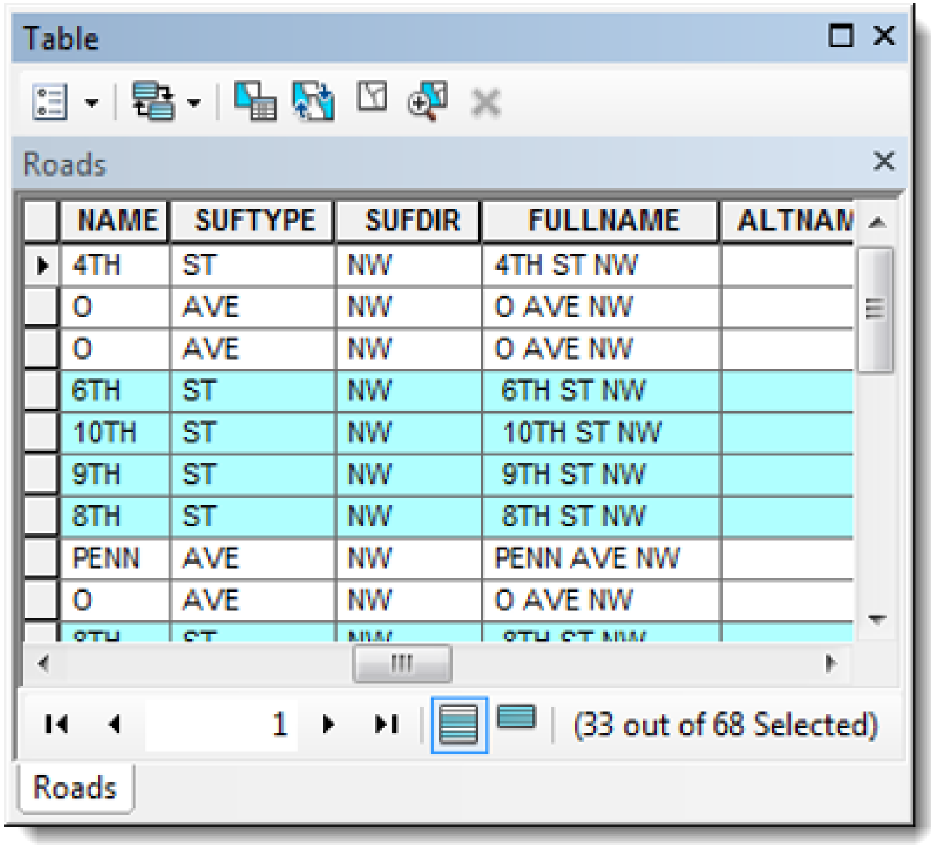


Shape_Length	OBJECTID *	PREFIX	NAME	ST_TYPE
323.193017	120	E	TIMBER CUT	CT
179.455581	121		Irish Shores	LN
505.18015	122		CORNERSTONE	ST
2.254424	123		FOXDEN	DR
4.176545	124		WYCKCHESTER	DR
32.79814	125		SHADYBEND	DR
150.582307	126		Keithwood	CIR
352.676325	127		PHEASANT	LN
285.128164	128		KEITHWOOD	CIR
369.600941	129		ROSEBUD	DR
242.291842	130		LAKECREST	LN

```
upper(NAME)<> NAME
```

String Concatenation

- You have a Roads layer with columns for the street name, suffix, and direction. There is also a FULLNAME column that should be a concatenation of the previous three columns. Use a SQL query to find out if there are any that do not match.



Table

Roads

NAME	SUFTYPE	SUFDIR	FULLNAME	ALTNAM
4TH	ST	NW	4TH ST NW	
O	AVE	NW	O AVE NW	
O	AVE	NW	O AVE NW	
6TH	ST	NW	6TH ST NW	
10TH	ST	NW	10TH ST NW	
9TH	ST	NW	9TH ST NW	
8TH	ST	NW	8TH ST NW	
PENN	AVE	NW	PENN AVE NW	
O	AVE	NW	O AVE NW	

1 (33 out of 68 Selected)

Roads

```
FULLNAME <> NAME ||
' ' || SUFTYPE || '
' || SUFDIR
```

Dealing with White Space

- When a column has been populated using pieces of other columns, sometimes a space is left over at the beginning or the end. This will cause values to be left out of your queries.
- Use the Trim function to ignore white space before processing your query.

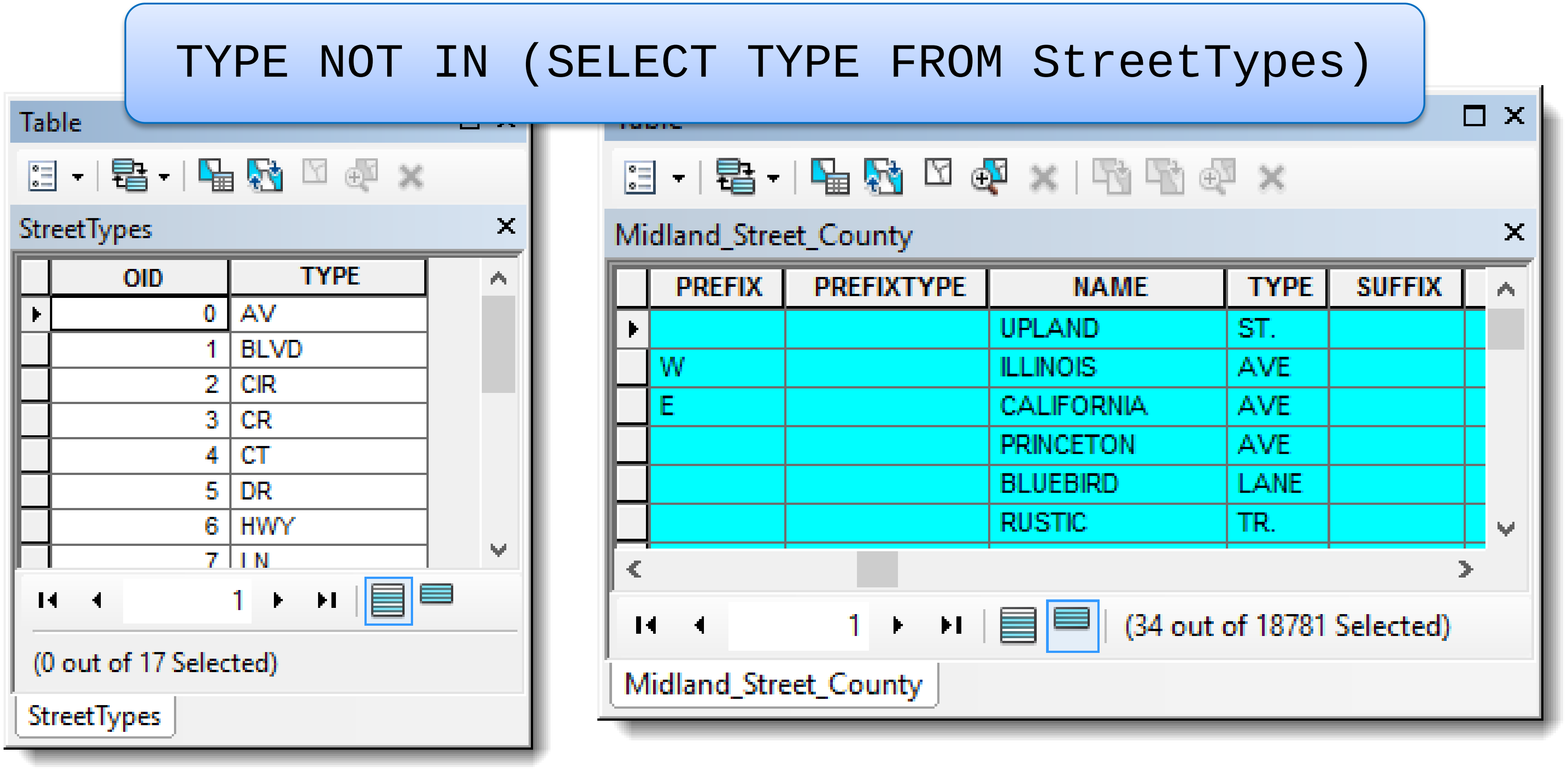
```
TRIM(RIGHT ‘ ‘ FROM PrefixDir) = ‘SE’
```

- Use Trim to find values with leading or trailing spaces

```
TRIM(RIGHT ‘ ‘ FROM PrefixDir) <> PrefixDir
```

Finding Invalid Street Types

- If you have a standalone table of valid street types, use a subquery to select invalid street types from your attribute table



```
TYPE NOT IN (SELECT TYPE FROM StreetTypes)
```

Table

StreetTypes

OID	TYPE
0	AV
1	BLVD
2	CIR
3	CR
4	CT
5	DR
6	HWY
7	LN

(0 out of 17 Selected)

StreetTypes

Table

Midland_Street_County

PREFIX	PREFIXTYPE	NAME	TYPE	SUFFIX
		UPLAND	ST.	
W		ILLINOIS	AVE	
E		CALIFORNIA	AVE	
		PRINCETON	AVE	
		BLUEBIRD	LANE	
		RUSTIC	TR.	

(34 out of 18781 Selected)

Midland_Street_County

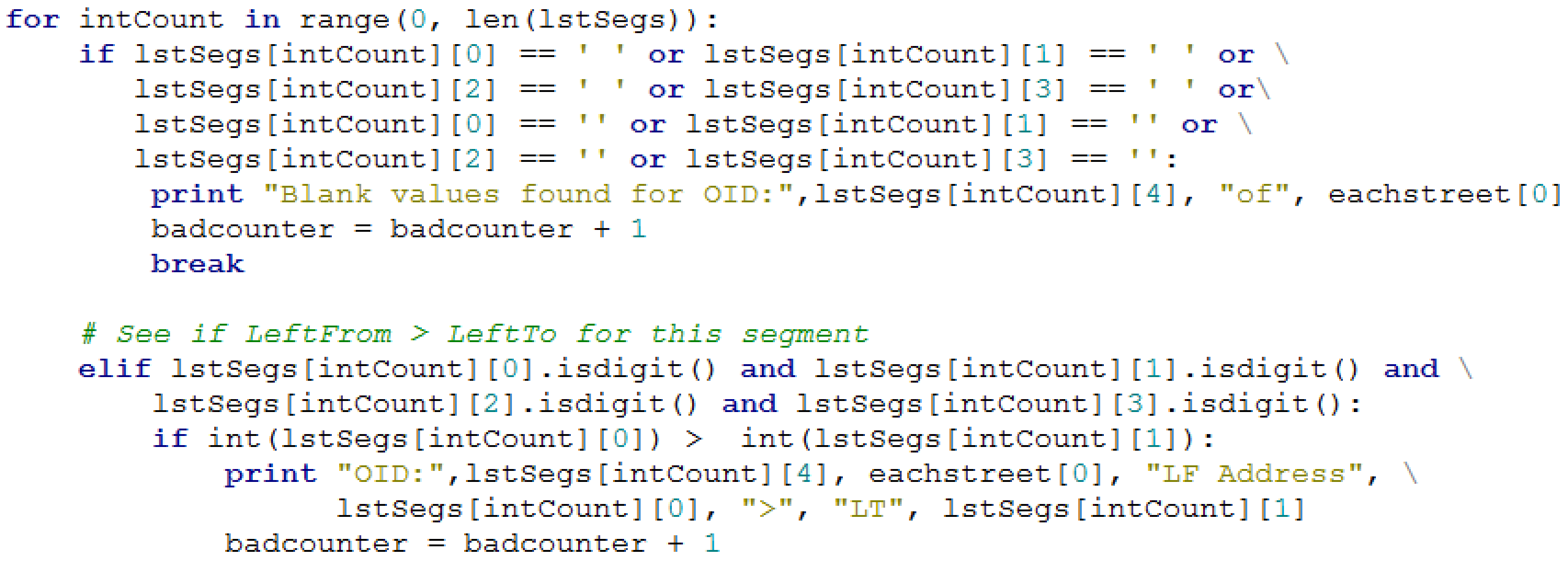
Scripts

Python to check addresses

Need to check street segments to see where there are overlapping address ranges for the same street name

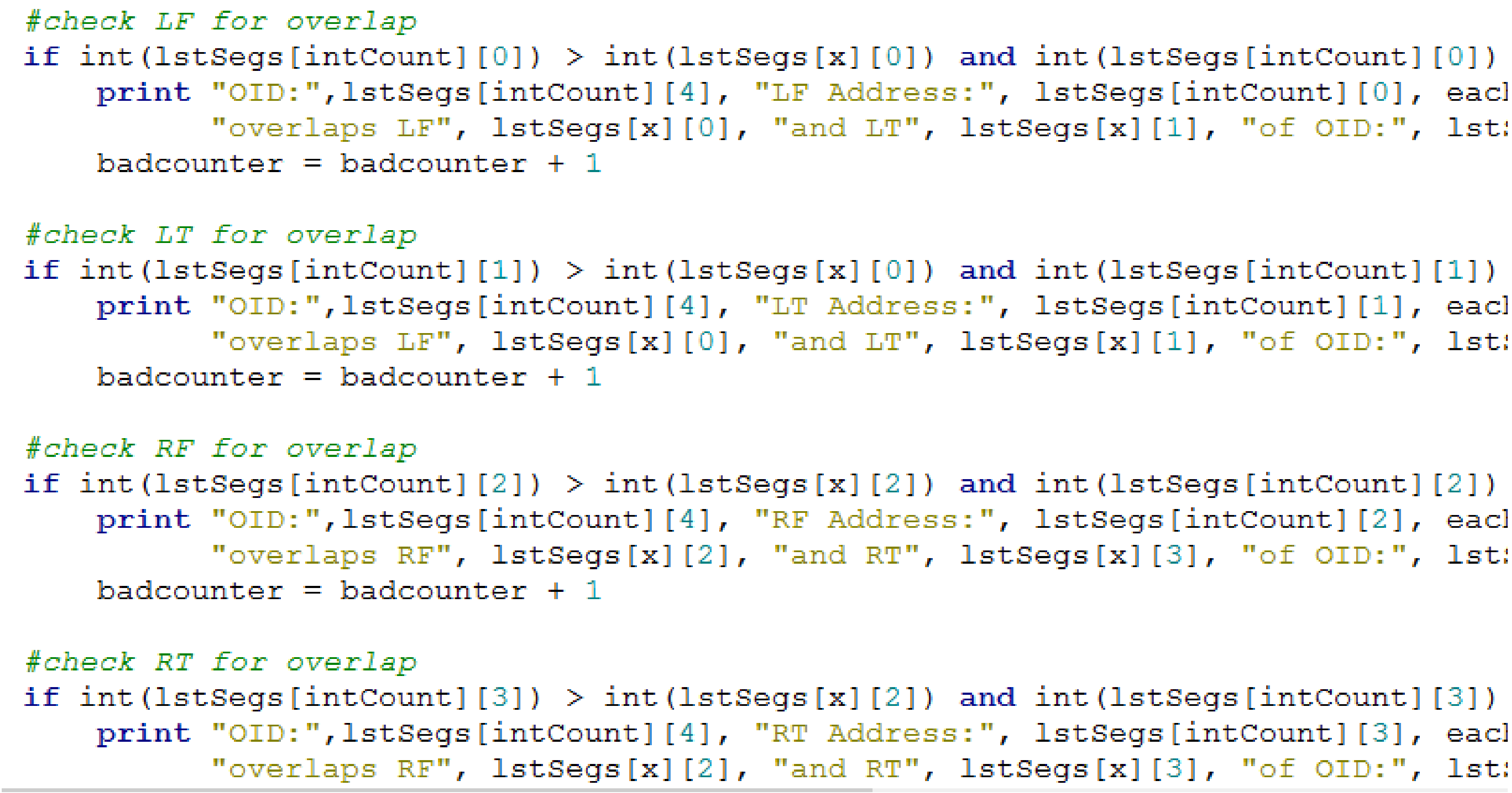
- Make a distinct list of all street names
- Go through the list and get all street segments for that street
- Check the segments to see if the current segments addresses overlap or duplicate the addresses from other segments
- Write out a report of all the bad segments

Python to check addresses



```
for intCount in range(0, len(lstSegs)):
    if lstSegs[intCount][0] == ' ' or lstSegs[intCount][1] == ' ' or \
       lstSegs[intCount][2] == ' ' or lstSegs[intCount][3] == ' ' or\
       lstSegs[intCount][0] == '' or lstSegs[intCount][1] == '' or \
       lstSegs[intCount][2] == '' or lstSegs[intCount][3] == '':
        print "Blank values found for OID:",lstSegs[intCount][4], "of", eachstreet[0]
        badcounter = badcounter + 1
        break

    # See if LeftFrom > LeftTo for this segment
    elif lstSegs[intCount][0].isdigit() and lstSegs[intCount][1].isdigit() and \
        lstSegs[intCount][2].isdigit() and lstSegs[intCount][3].isdigit():
        if int(lstSegs[intCount][0]) >  int(lstSegs[intCount][1]):
            print "OID:",lstSegs[intCount][4], eachstreet[0], "LF Address", \
                  lstSegs[intCount][0], ">", "LT", lstSegs[intCount][1]
            badcounter = badcounter + 1
```

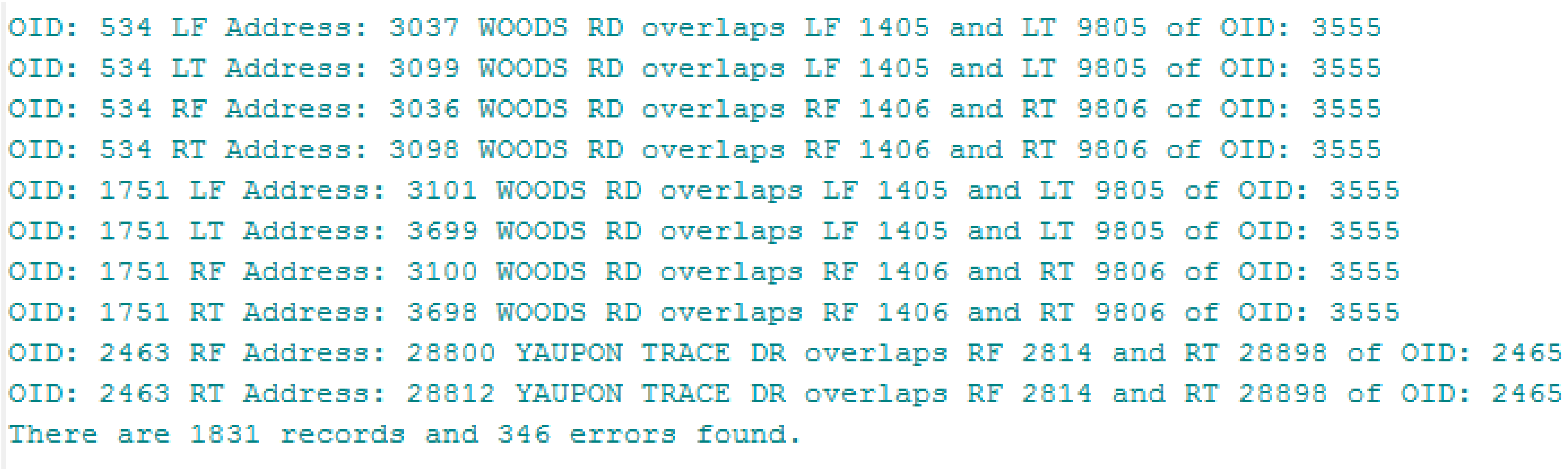
Python to check addresses



```
#check LF for overlap
if int(lstSegs[intCount][0]) > int(lstSegs[x][0]) and int(lstSegs[intCount][0])
    print "OID:",lstSegs[intCount][4], "LF Address:", lstSegs[intCount][0], eac
          "overlaps LF", lstSegs[x][0], "and LT", lstSegs[x][1], "of OID:", lst
    badcounter = badcounter + 1

#check LT for overlap
if int(lstSegs[intCount][1]) > int(lstSegs[x][0]) and int(lstSegs[intCount][1])
    print "OID:",lstSegs[intCount][4], "LT Address:", lstSegs[intCount][1], eac
          "overlaps LF", lstSegs[x][0], "and LT", lstSegs[x][1], "of OID:", lst
    badcounter = badcounter + 1

#check RF for overlap
if int(lstSegs[intCount][2]) > int(lstSegs[x][2]) and int(lstSegs[intCount][2])
    print "OID:",lstSegs[intCount][4], "RF Address:", lstSegs[intCount][2], eac
          "overlaps RF", lstSegs[x][2], "and RT", lstSegs[x][3], "of OID:", lst
    badcounter = badcounter + 1

#check RT for overlap
if int(lstSegs[intCount][3]) > int(lstSegs[x][2]) and int(lstSegs[intCount][3])
    print "OID:",lstSegs[intCount][4], "RT Address:", lstSegs[intCount][3], eac
          "overlaps RF", lstSegs[x][2], "and RT", lstSegs[x][3], "of OID:", lst
```

Python to check addresses

- A report showing address range problems by Object ID



```
OID: 534 LF Address: 3037 WOODS RD overlaps LF 1405 and LT 9805 of OID: 3555
OID: 534 LT Address: 3099 WOODS RD overlaps LF 1405 and LT 9805 of OID: 3555
OID: 534 RF Address: 3036 WOODS RD overlaps RF 1406 and RT 9806 of OID: 3555
OID: 534 RT Address: 3098 WOODS RD overlaps RF 1406 and RT 9806 of OID: 3555
OID: 1751 LF Address: 3101 WOODS RD overlaps LF 1405 and LT 9805 of OID: 3555
OID: 1751 LT Address: 3699 WOODS RD overlaps LF 1405 and LT 9805 of OID: 3555
OID: 1751 RF Address: 3100 WOODS RD overlaps RF 1406 and RT 9806 of OID: 3555
OID: 1751 RT Address: 3698 WOODS RD overlaps RF 1406 and RT 9806 of OID: 3555
OID: 2463 RF Address: 28800 YAUPON TRACE DR overlaps RF 2814 and RT 28898 of OID: 2465
OID: 2463 RT Address: 28812 YAUPON TRACE DR overlaps RF 2814 and RT 28898 of OID: 2465
There are 1831 records and 346 errors found.
```

Other Python Uses

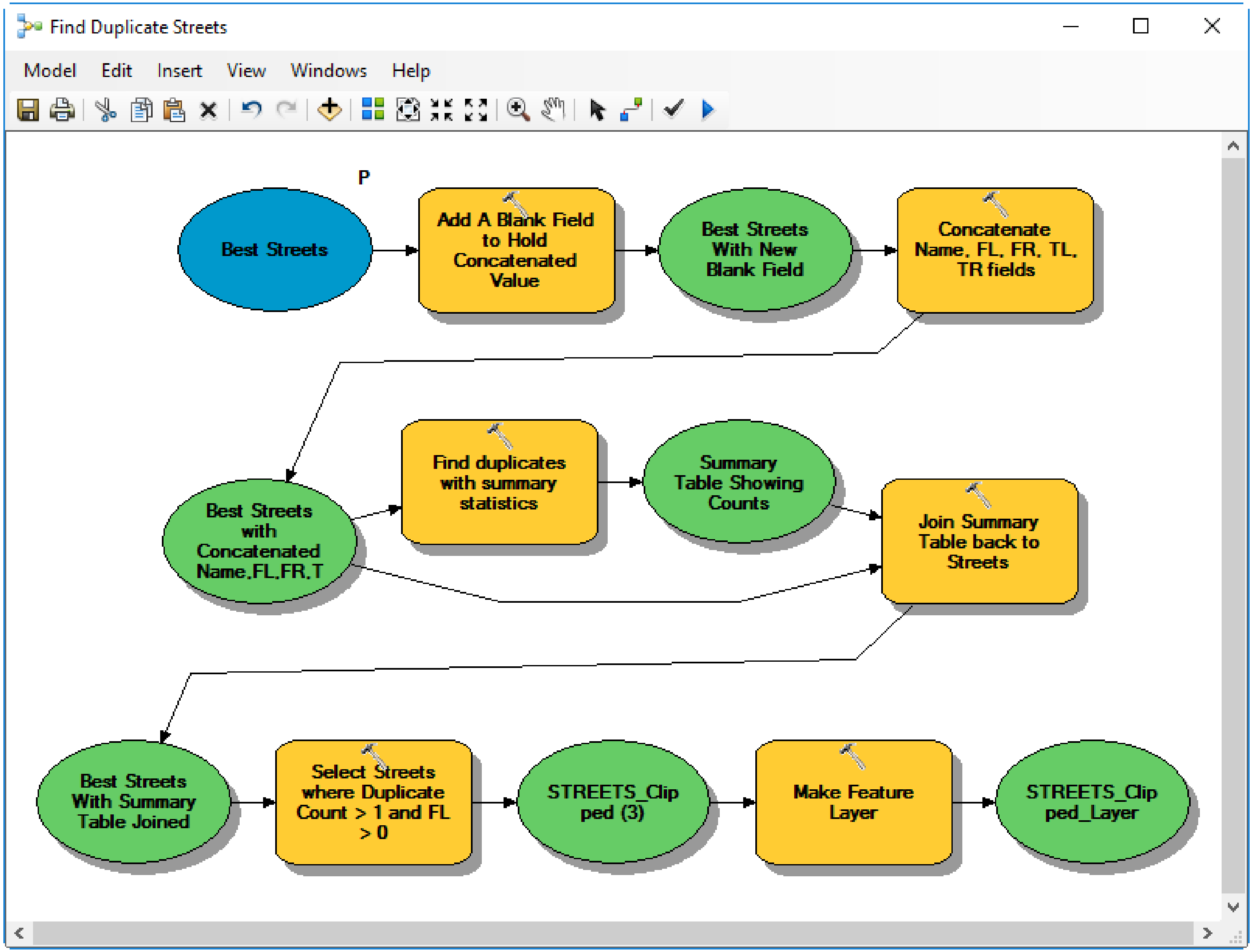
- Script to create a report of street name anomalies.
  - one segment only
- Script to find zonal errors, report them, and then fix them.
  - Zip codes, EMS response areas, …
- Script to compare street segments with address point layer and find discrepancies
- Script to calculate travel time values based on road types, speed limits and segment lengths

In all cases you must know the data structure and be ready to modify your script(s) when changes occur.

Geoprocessing Tools

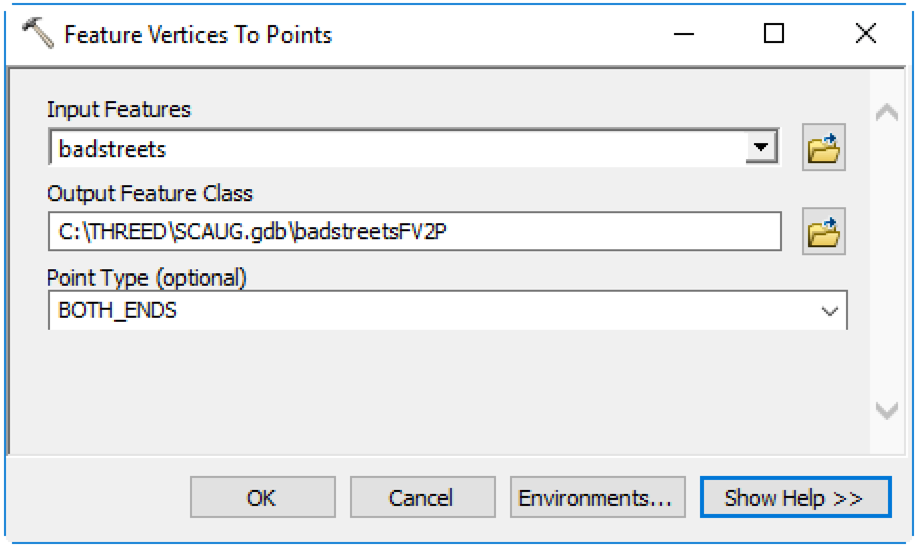
Put it in a model!

- Model to find duplicate streets


Find Duplicate Streets
Model
Edit
Insert
View
Windows
Help
P
Best Streets
Add A Blank Field to Hold Concatenated Value
Best Streets With New Blank Field
Concatenate Name, FL, FR, TL, TR fields
Best Streets with Concatenated Name,FL,FR,T
Find duplicates with summary statistics
Summary Table Showing Counts
Join Summary Table back to Streets
Best Streets With Summary Table Joined
Select Streets where Duplicate Count > 1 and FL > 0
STREETS_Clipped (3)
Make Feature Layer
STREETS_Clipped_Layer

Find bad intersections workflow

- Feature Vertices To Points



Feature Vertices To Points

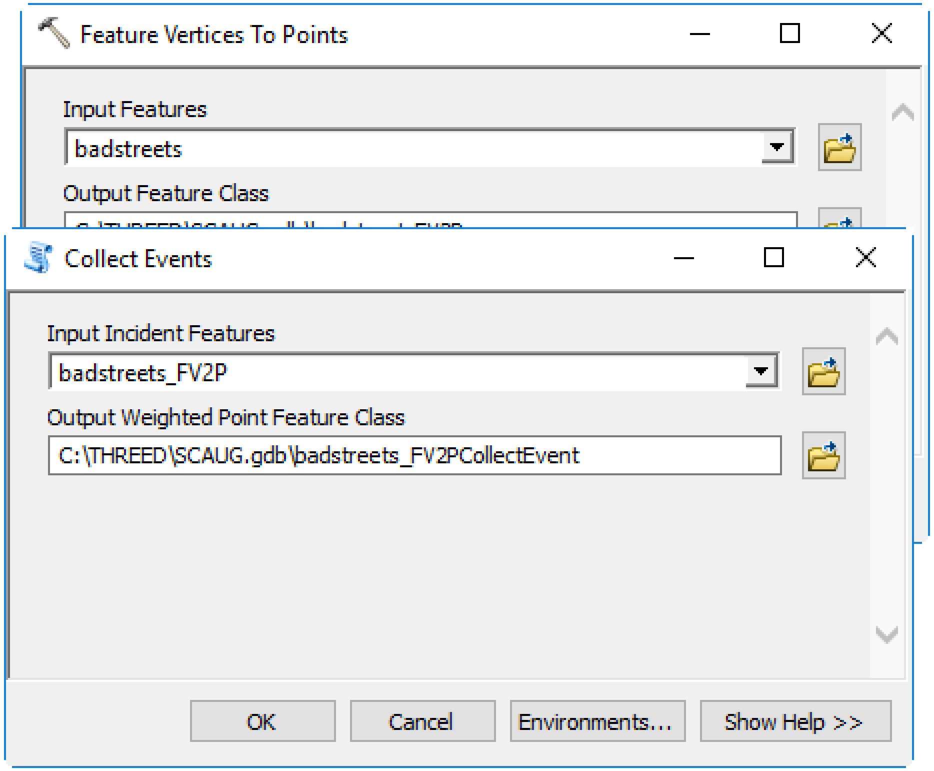
Input Features: badstreets

Output Feature Class: C:\THREED\SCAUG.gdb\badstreetsFV2P

Point Type (optional): BOTH_ENDS

Find bad intersections workflow

- Feature Vertices To Points
- Collect Events



Feature Vertices To Points

Input Features: badstreets

Output Feature Class

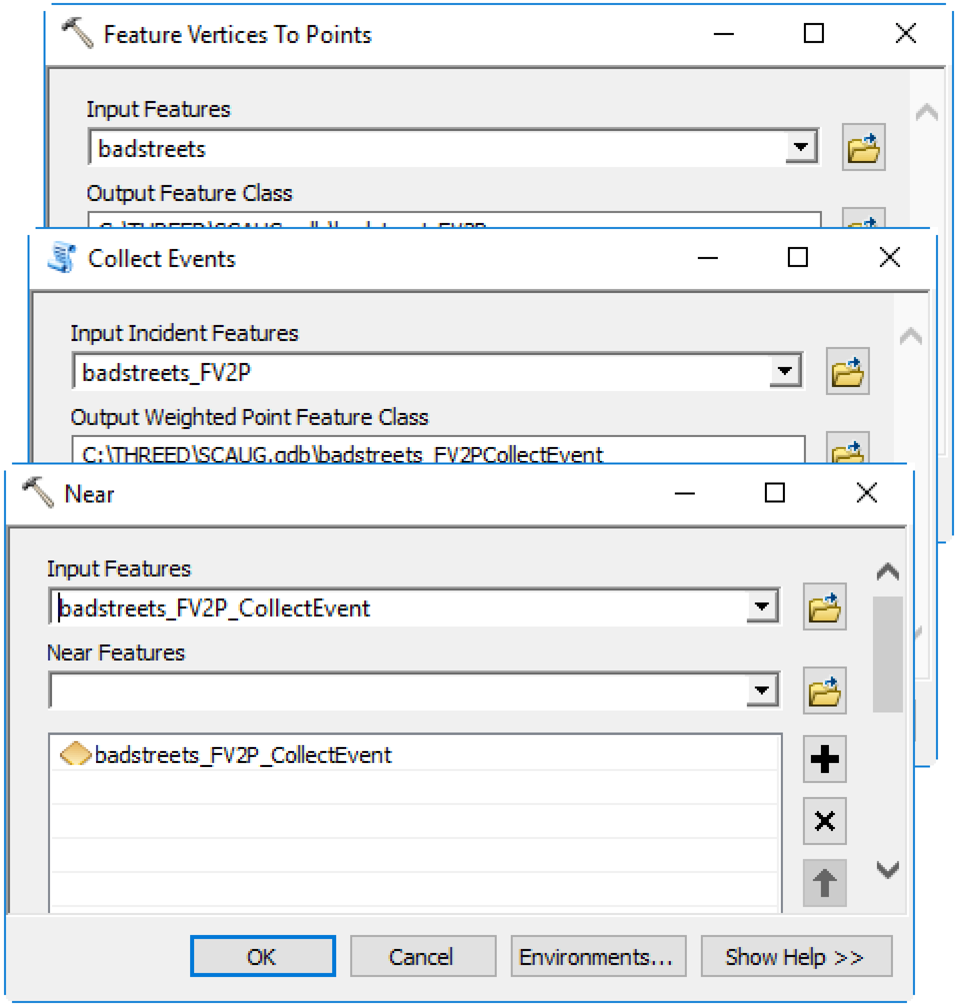
Collect Events

Input Incident Features: badstreets_FV2P

Output Weighted Point Feature Class: C:\THREED\SCAUG.gdb\badstreets_FV2PCollectEvent

Find bad intersections workflow

- Feature Vertices To Points
- Collect Events
- Near
  - Examine att table
  - Symbolize (-1)
  - Inspect map



Feature Vertices To Points

Input Features: badstreets

Output Feature Class

Collect Events

Input Incident Features: badstreets_FV2P

Output Weighted Point Feature Class: C:\THREED\SCAUG.gdb\badstreets_FV2PCollectEvent

Near

Input Features: badstreets_FV2P_CollectEvent

Near Features:

badstreets_FV2P_CollectEvent

Topology

Topologies

- “Features working together”
- Map Topology
  - Edit simultaneous features
  - City limits, council districts, streets
  - Street segments meeting at one intersection
- Geodatabase Topology
  - Employ rules within a geodatabase to find errors
  - Tools to fix errors

Editing Tips

Plan Before You Edit!

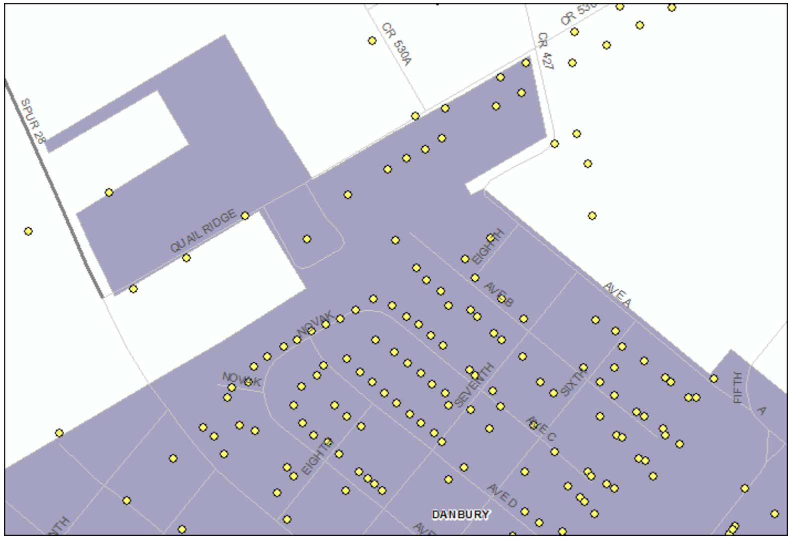
- Think about:
  - Which layers do you need on your map?
  - How should you symbolize the layers so that you can see everything?
  - You may want to create a map document just for editing, different from your regular or published maps

UR 28
530A
CR 171
QUAIL RIDGE
NOVAK
EIGHTH
AVE A
AVE B
AVE C
AVE D
AVE E
FIFTH
A
AVE A
TENTH
EIGHTH
ELEVENTH
AVE G
NINTH
AVE F
SEVENTH
SIXTH
AVE H
AVE I
MAIN
FIFTH
FOURTH
AVE D
AVE E
SECOND
JUNE
E J FILIPP
HOSKINS MOUND
DANBURY
AVE G
THIRD
AVE F
AVE K
AVE J
AVE L
AVE H
AVE I
PANTHER
AVE K
AVE L
SECOND
MEADOW
DANBURY
CR 208
R 171

Common Editing Layers

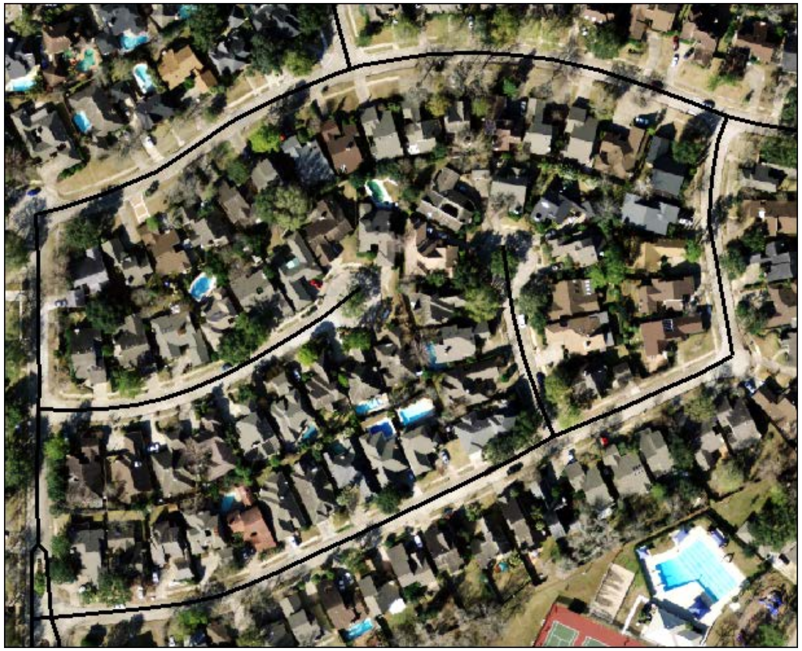
- Some layers you might need for your street edits:
  - Imagery (Esri basemap or aerial photos)
  - Address points, where applicable
  - Parcel polygons
  - Street centerlines themselves
  - Jurisdictional and emergency response boundaries

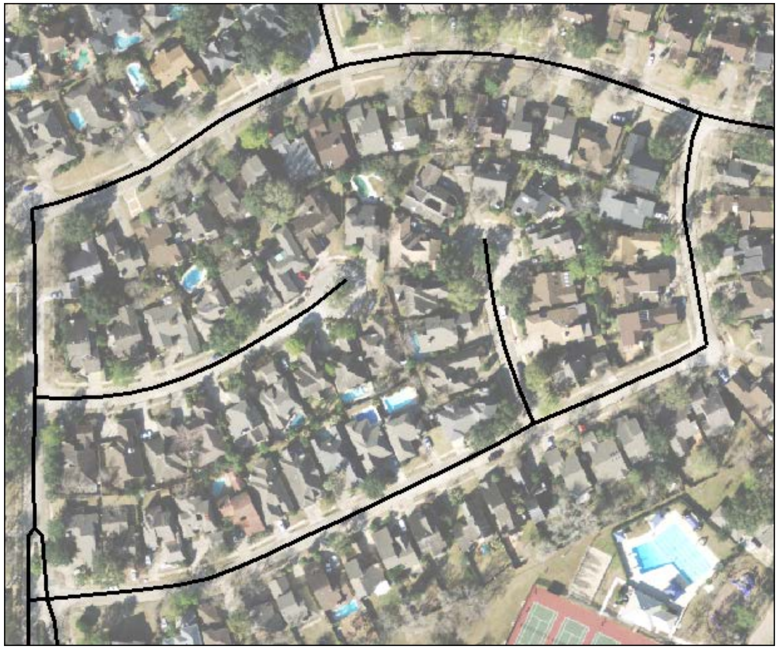
Westbank Expy
United States Highway
United States Highway
Westbank Expy
Governor Hall St
90 (Bus)
90 (Bus)
Lafayette St
Calder St
17th St
Thomas St
15th St
Lafayette St



Imagery

- Used for digitizing new streets and new address points
- Helpful for ground-truthing
  - Divided vs non-divided streets
  - Intersections





- Use transparency to make vector data easier to see

Boundary Polygons

- What jurisdictional or emergency response boundaries are important to your organization?
  - Cities
  - Zip Codes
  - ESN Boundaries
  - Fire Districts
  - Police Beats
  - Anything else?

Planarize

- Enter the street segments as one feature
- Add attributes that all new streets will have in common
  - City, County, Zip
  - Select **all new streets**, enter attributes in Attribute window
- Add attributes that all segments will have in common for one street
  - Name, Type, MPH
  - Select **individual street**, enter attributes
- Then planarize
- Enter values specific to each segment
  - Address Ranges
  - Select **individual street-segment**, enter attributes

Your Turn…

What's your tip?