Altair Exercises

This notebook will explore multiple different visualizations in Altair.

Part 7

The following exercise is based on the show [The Simpsons (https://en.wikipedia.org/wiki/The_Simpsons)](https://en.wikipedia.org/wiki/The_Simpsons), and will be analyzing key quotes from the history of the Simpsons as a network problem.

We will be leveraging an extracted dataset, [quote data from the wikiquotes project (https://en.wikiquote.org/wiki/The_Simpsons)](https://en.wikiquote.org/wiki/The_Simpsons). For each season, we look at each episode. For each episode, we identify the quotes recorded in wikiquotes.

While some quotes are "one-liners" by a single character:

- **Homer**: Aww, it makes no sense; I haven't changed since high school and all of a sudden I'm uncool.

We are interested in interactions between characters. For example:

- **Homer**: Doughnut?
- **Lisa**: No, thanks. Do you have any fruit?
- **Homer**: This has purple stuff inside. Purple is a fruit.

These multi-character quotes will become the "edges" in our network, where appropriate. We're going to start with a simpler analysis of the hierarchical data in this database.

In [1]:
```
# !pip install nx_altair
# !pip install squarify
```

In [2]:
```
import pandas as pd
import json
import networkx as nx
import nx_altair as nxa
import json
import squarify
import altair as alt
import numpy as np
import matplotlib
from sklearn.cluster import AgglomerativeClustering
from networkx.algorithms.community import *
from scipy.cluster.hierarchy import dendrogram,leaves_list
from scipy.cluster.hierarchy import ClusterWarning
from warnings import simplefilter
simplefilter("ignore", ClusterWarning)
```

In [3]:
```
# enable correct rendering (unnecessary in later versions of Altair)
```

In [3]:
```
# enable correct rendering (unnecessary in later versions of Altair)
alt.renderers.enable('default')

# uses intermediate json files to speed things up
alt.data_transformers.enable('json')
```

Out[3]:
```
DataTransformerRegistry.enable('json')
```

Hierarchical representation of Quote Database

We have data for multiple seasons,a nd for each season, we have multiple episodes. For each episode, we have multiple characters who have particiapated in funny/memorable scenes. Our goal is to have a visualization that allows us to compare which seasons/episodes/characters had the most quoted conversations. The questions we wish to be able to answer with the visualization are:

- Does a certain character have many conversations in one episode, and fewer in others?
- Were there any outlier episodes with lots of conversations?
- Was there a season with many conversations?

There are 5076 conversations in our dataset.

In [4]:
```
with open('../assets/simpsonshier.jsonl') as json_file:
    allseasons = json.load(json_file)
```

In [5]:
```
# this variable is a massive JSON object with a hierarchy of seasons -
# hierarchy is a dictionary which has an id, a type (one of 'season','
# label (episodes had names as well as numerical ids), the value (the
# list of nodes that sit underneath)

allseasons

# for example, we can find season 2's quote quote:
season2 = allseasons[1]
print("season ",season2['id'], "had", season2['value'], "quotes")

# for the first episode of season 1 we see the following
season2e1 = season2['children'][0]
print(season2e1)

# you'll see that this episode was called "Bart Gets and F" and it had
# Bart was responsible for 2 of these
```

```
season  s02 had 124 quotes
```

```
{'id': 's02e14', 'type': 'episode', 'label': 'Bart Gets an F', 'value
': 7, 'children': [{'id': 'Bart', 'type': 'character', 'value': 2}, {
'id': 'Mrs. Krabappel', 'type': 'character', 'value': 1}, {'id': 'Mar
tin', 'type': 'character', 'value': 1}, {'id': 'Otto', 'type': 'chara
cter', 'value': 1}, {'id': 'Sherri', 'type': 'character', 'value': 1}
, {'id': 'Terri', 'type': 'character', 'value': 1}]}
```


There are two main ways to display hierarchies. Node link diagrams, and space-filling versions such as TreeMaps. For this exercise we will use TreeMaps.

Unfortunately, Altair doesn't have a treemap layout built in. We'll be using the [squarify (https://github.com/laserson/squarify)](https://github.com/laserson/squarify) library to generate the coordinates. Squarify works by generating one level of the hierarchy at a time. So we need a function that lays out the seasons, and then for each episode re-runs squarify but restricts it to the space allocated to the season. After that we re-run squarify to plot the position of each character in that episode.

In [6]:
```
def rectangleIter(data,width,height,xof=0,yof=0,frame=None,level=-1,pa
    # data: hierarchical structured data
    # width: width we can work in
    # height: height we can work in
    # xof: x offset
    # yof: y offset
    # frame: the dataframe to add the data to, if None, we create one
    # the level of the treemap (will default to 0 on first run)
    # parentid: a string representing the parent of this node
    # returns dataframe of all the rectangles

    if (frame is None):
        frame = pd.DataFrame()
    level = level + 1
    values = []
    children = []
    for parent in data:
        values.append(parent['value'])
        if ('children' in parent):
            children.append(parent['children'])
        else:
            children.append([])

    # normalize
    values = squarify.normalize_sizes(values, width, height)

    # generate the
    padded_rects = squarify.padded_squarify(values, xof, yof, width, h

    i = 0
    for rect in padded_rects:
        # adjust the padding and copy the useful pieces of data over
        parent = data[i]
        rect['width'] = rect['dx']
        rect['height'] = rect['dy']
        del rect['dx']
        del rect['dy']
        rect['x2'] = rect['x'] + rect['width'] - 2
        rect['y2'] = rect['y'] + rect['height'] - 2
        rect['x'] = rect['x'] + 2
        rect['y'] = rect['y'] + 2
        rect['width'] = rect['x2'] - rect['x']
        rect['height'] = rect['y2'] - rect['y']
        rect['id'] = parent['id']
        rect['type'] = parent['type']
        rect['value'] = parent['value']
        rect['level'] = level
        if 'label' in parent:
            rect['label'] = parent['label']
        else:
            rect['label'] = parent['id']
        rect['parentid'] = parentid
        frame = frame.append(rect,ignore_index=True)

        # iterate
        frame = rectangleIter(children[i],rect['width'],rect['height']
                              frame=frame,level=level,parentid=parenti
        i = i + 1
    return(frame)
```

To keep the chart manageable and readable, we will work with a sample of the first 6 seasons.

In [7]:
```
shortseason = allseasons[0:6]                  # let's grab the first
rect_table = rectangleIter(shortseason,800,800) # and run them through
```

In [8]:
```python
# let's look at what's inside
rect_table.sample(5)
```

Out[8]:

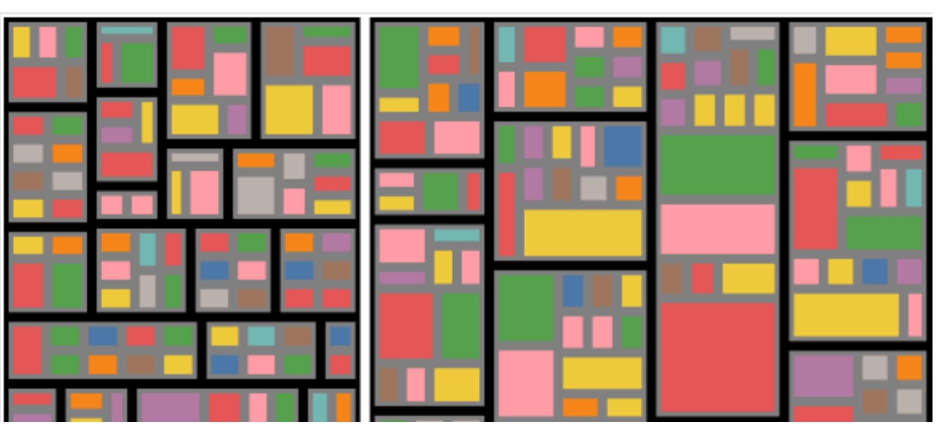
	height	id	label	level	parentid	type	value	width	x
847	37.127468	Marge	Marge	2.0	→ s06 → Fear of Flying (The Simpsons)	character	7.0	70.985785	413.388770
608	26.722277	Men	Men	2.0	→ s05 → The Last Temptation of Homer	character	1.0	7.505350	605.950508
522	16.510884	Principal Skinner	Principal Skinner	2.0	→ s05 → Homer's Barbershop Quartet	character	1.0	13.820494	485.120581
902	25.657713	Krusty the Clown	Krusty the Clown	2.0	→ s06 → Homie the Clown	character	3.0	36.902013	496.374556
371	47.549621	s04e63	Lisa the Beauty Queen	1.0	→ s04	episode	7.0	61.302721	234.571429

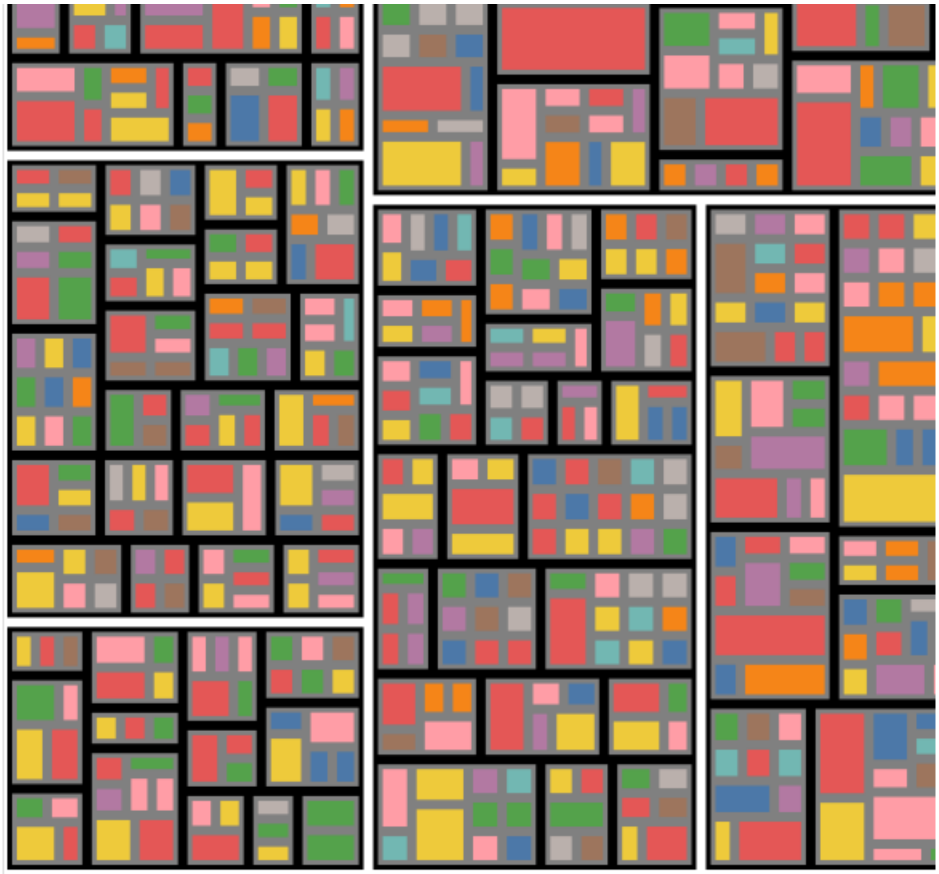
Now let's make a TreeMap.

In [9]:
```python
def staticTreemap(inputFrame):
    # input inputFrame the rectangles frame as described above
    # return a static Altair treemap visualization

    lvl_0 = inputFrame[inputFrame['level'] == 0.0]
    level_0 = alt.Chart(lvl_0).mark_rect(color="#000000").encode(x=alt

    lvl_1 = inputFrame[inputFrame['level'] == 1.0]
    level_1 = alt.Chart(lvl_1).mark_rect(color="808080").encode(x=alt.

    lvl_2 = inputFrame[inputFrame['level'] == 2.0]
    level_2 = alt.Chart(lvl_2).mark_rect().encode(x=alt.X('x:Q',axis=N

    final = (level_0 + level_1 + level_2).properties(
        width=800,
        height=800
    )

    return(final)
```

In [10]:
```python
staticTreemap(rect_table)
```

Out[10]:





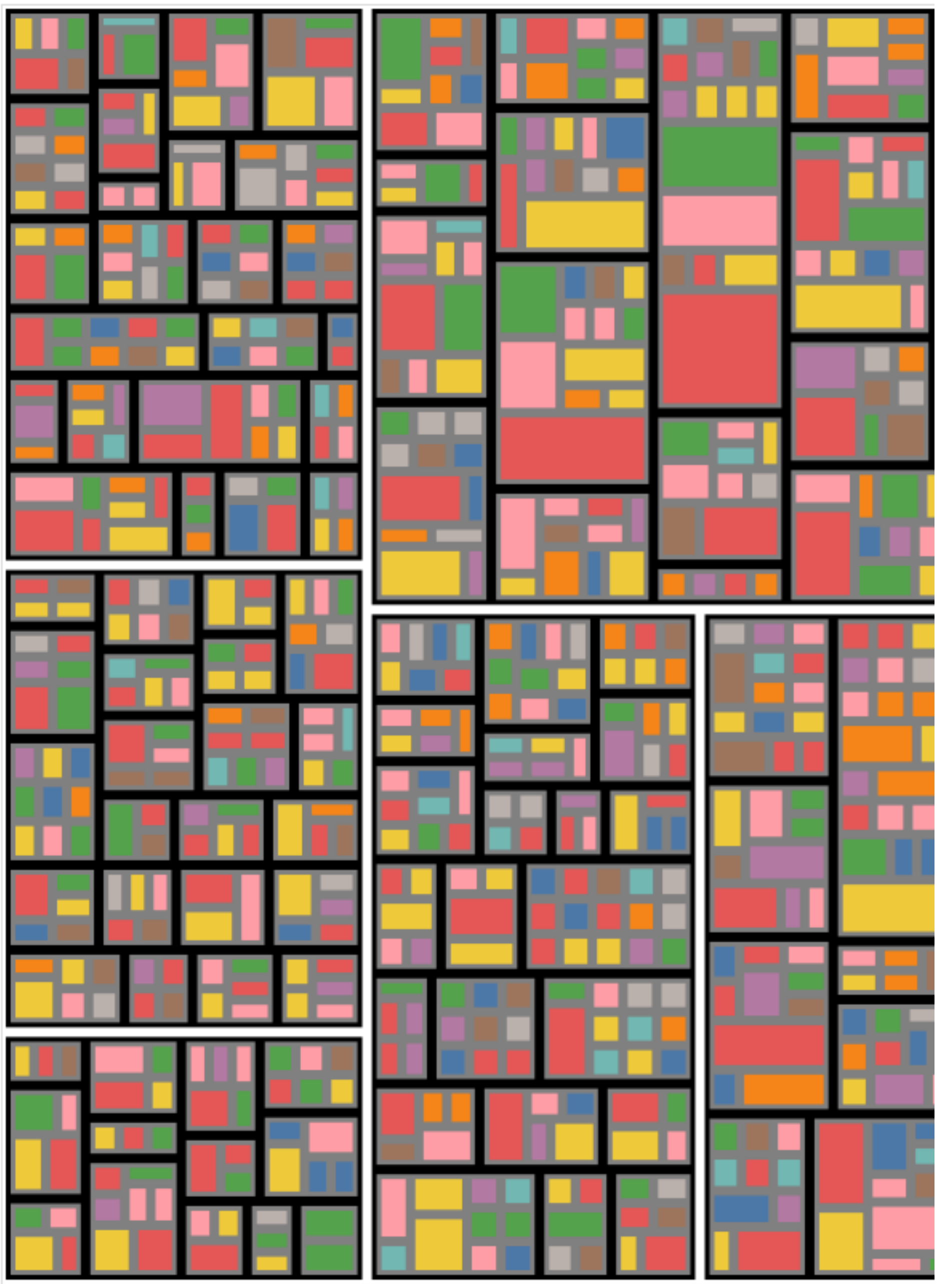
That doesn't do much for us aside from show some relative values. Let's make this interactive so that we can better understand what we are looking at.

In [11]:
```
def interactiveTreemap(inputFrame):
    # input inputFrame the rectangles frame as described above
    # return a static Altair treemap visualization

    label_select = alt.selection_single(empty='all', fields=['id'])
    colorCondition2 = alt.condition(label_select,"id:N",alt.value('lig

    lvl_0 = inputFrame[inputFrame['level'] == 0.0]
    level_0 = alt.Chart(lvl_0).mark_rect(color="#000000").encode(x=alt
                                                              x2='x
                                                              y=alt
                                                              y2='y

    lvl_1 = inputFrame[inputFrame['level'] == 1.0]
    level_1 = alt.Chart(lvl_1).mark_rect(color="808080").encode(x=alt.
                                                             x2='x2
                                                             y=alt.
                                                             y2='y2

    lvl_2 = inputFrame[inputFrame['level'] == 2.0]
    level_2 = alt.Chart(lvl_2).mark_rect().encode(x=alt.X('x:Q',axis=N
                                              x2='x2:Q',
                                              y=alt.Y('y:Q',axis=N
                                              y2='y2:Q',
                                              color=alt.Color("id:
                                              tooltip=['parentid:N

    base = (level_0 + level_1).properties(
        width=800,
        height=800
    )
```

```
    interactive_lvl2 = level_2.add_selection(
        label_select
    ).encode(
        color=colorCondition2
    )

    i_treemap = (base + interactive_lvl2)

    return i_treemap
```

In [12]:
```
interactiveTreemap(rect_table)
```

Out[12]:



Now, you shoudl be able to scroll over to see id's, and if you click to select, you can compare character-by-character, episode-over-episode.

Network Analysis for Conversational Quotes

For the next graphic, we will look more closely at conversation networks. Each quotable conversation can be modeled as a small network. Nodes correspond to characters, and edges are the number of conversations two characters co-occurred in. For example, if Bart, Homer, and Lisa are in the same quote, we would construct 3 undirected edges: Bart to Homer, Bart to Lisa, and Lisa to Homer. By aggregating all these conversations together (over episodes or seasons), we can compute the "weight" of an edge: the total number of quoted conversations those characters interacted over. From this kind of network, we can identify who the central characters are. Who is interacting with the most others in a quotable way? Are there small communities?

For this problem, we are going to use two libraries to help us out: [networkx (https://networkx.org/)](https://networkx.org/)--a library for manipulation and analysis of graph data structures (it will also generate layouts), and [nx-altair (https://github.com/Zsailer/nx_altair)](https://github.com/Zsailer/nx_altair) a library that can generate Altair plots from networkx data.

In [13]:
```
# helper function to load the data
def loadData(filepath):
    articles = []
    with open(filepath) as fp:
        for docid, line in enumerate(fp):
            #print(line)
            #print(docid)
            doc = json.loads(line)
            char1 = doc['c1']
            char2 = doc['c2']
            articles.append(doc)
    return pd.DataFrame(articles)
```

In [14]:
```
# we're going to load a data frame representation of the network to st
simpsons = loadData('../assets/simpsons.jsonl')
```

In [15]:
```
# let's look inside
simpsons.sample(5)
```

Out[15]:

	season	episode	lineid	c1	c2
819	5	Bart Gets Famous	263	Barney	Lisa
2807	10	Bart the Mother	1053	Homer	Milhouse
2844	10	D'oh-in in the Wind	1075	Dr. Hibbert	Marge
3270	12	Treehouse of Horror XI	1304	Carl	Moe
4844	23	Holidays of Future Passed	2331	Bart	Lisa

We see a row for every edge. The `season` and `episode` column has the season the conversation happened in. The `lineid` is a unique id for the conversation (note that if the conversation involved more than two people, we'd see the same lineid multiple times; see the Bart/Homer/Lisa example above). The columns `c1` and `c2` hold the two characters' names (the name in c1 will always be alphabetically before c2). For now, this data is as possible, but you may see some inconsistencies with names. For example, you might find different entries for "Skinner" and "Principal Skinner" even though they are the same character. Since that is not a major focus for this exercise, we will leave it as-is for now.

Next, we'll build our network using networkx.

In [16]:
```
# utility classes

def buildNetwork(quoteFrame):
    # takes as input the quote frame (e.g., simpsons) or some subset c
    # and returns an undirected networkx graph

    weight = quoteFrame.groupby(['c1','c2']).count()
    weight = weight.reset_index()
    toret = nx.Graph()
    for row in weight.iterrows():
        row = row[1]
        if (row['c1'] not in toret.nodes):
            toret.add_node(row['c1'])
            toret.nodes[row['c1']]['appearance'] = 0
            toret.nodes[row['c1']]['label'] = row['c1']
        if (row['c2'] not in toret.nodes):
            toret.add_node(row['c2'])
            toret.nodes[row['c2']]['appearance'] = 0
            toret.nodes[row['c2']]['label'] = row['c2']
        toret.nodes[row['c1']]['appearance'] = toret.nodes[row['c1']][
        toret.nodes[row['c2']]['appearance'] = toret.nodes[row['c2']][
        toret.add_edge(row['c1'],row['c2'])
        toret.edges[row['c1'],row['c2']]['weight'] = int(row['season']
    return toret

def getLayout(positions):
    # helper function to build a dataframe of positions for nodes
    elems = []
    nodes = list(positions.keys())
    for n in nodes:
        elems.append({'node':n,'x':positions[n][0],'y':positions[n][1]
    return(pd.DataFrame(elems))

def setCommunityLabels(G,communities):
    # adds community labels to the networkx graph nodes
```

```python
    id = 0
    for c in communities:
        id = id + 1
        for n in c:
            G.nodes[n]['community'] = id
    return(G)
```

In [17]:
```python
# let's start by grabbing only the network for a single season (6)
season6 = buildNetwork(simpsons[simpsons.season == 6])
```

In [18]:
```python
# season6 is a networkx object. You can ask for the edges or nodes
season6.nodes
```

Out[18]:
```
NodeView(('Abe', 'Crazy Old Man', 'Family', 'Homer', 'Jasper', 'Quimb
y', 'Accountant', 'Krusty the Clown', 'Aide', 'Al Gore', 'Bart', 'Koo
l', 'President', 'the Gang', 'Airport Worker', 'Amish Farmer', 'Annou
ncer', 'Godfrey Jones', 'Kent Brockman', 'Apu', 'Chief Wiggum', 'Moe'
, 'Audience Member 1', 'Audience Member 2', 'McBain', 'Rainier Wolfca
stle', 'Sherman', 'Wolfcastle', 'Australian man', 'Barney', 'Lisa', '
Man', 'Mayor Quimby', 'Woman', "Bart's Brain", 'Database', 'Grampa',
'Groundskeeper Willie', 'Helen', 'Helen Lovejoy', 'Hibbert', 'Jessica
', 'Jessica Lovejoy', 'Lunchlady Doris', 'Marge', 'Marine', 'Martin',
'Maude', 'Milhouse', 'Mrs. Krabappel', 'Ned', 'Ned Flanders', 'Nelson
', 'Principal Skinner', 'Reverend Lovejoy', 'Server', 'Shelby', 'Sher
ri', 'Sideshow Bob', 'Skinner', 'TV Announcer', 'Teacher', 'Bartender
', 'Bob', 'Boy', 'Brother', 'Burns', 'Chespirito', 'Hans Moleman', 'H
opkins', 'Shatner', 'Smithers', 'Spielbergo', 'Carl', 'Lenny', 'Marti
an', 'Mr. Burns', 'Stonecutters', 'Carla', 'Clavin', 'Norm', 'Sam', '
Woody', 'Dr. Hibbert', 'Homer/Marge', 'Clerk', 'Store Owner', 'Comic
Book Guy', 'Congressman', 'Speaker', 'Darth Vader', 'James Earl Jones
', 'Mufasa', 'Mufasa/Vader/Jones', 'Murphy', 'Dr. Zweig', 'Euro-Itchy
', 'Scratchy Land Ticket Attendant', 'Fat Tony', 'Legs', 'Louie', 'Fl
anders', 'Frink', 'Girl', 'Pilot 1', 'Pilot 2', 'Hitler', 'Officer',
"Homer's Brain", "Homer's Liver", 'Jay', 'Lesbian', 'Maude Flanders',
'Mr. Peabody', 'Number One', 'Patty', 'Vendor', 'Hugh', 'Hutz', 'Jay
Sherman', 'Jimbo', 'Judge', 'Largo', 'Leopold', 'Miss Hoover', 'Ralph
', 'Maggie', 'Willie', 'Park Announcer', 'Nurse', 'Mr Burns', 'Old Wo
man', 'Scott', 'Selma'))
```

In [19]:
```python
# we also calculate a special attribute of nodes called 'appearance' w
# degree of the node. This will be useful to us when we want to change
# node
season6.nodes['Bart']['appearance']
```

Out[19]: 35

In [20]:
```python
# which is the same as this
print(season6.degree('Bart'))
```

```
35
```

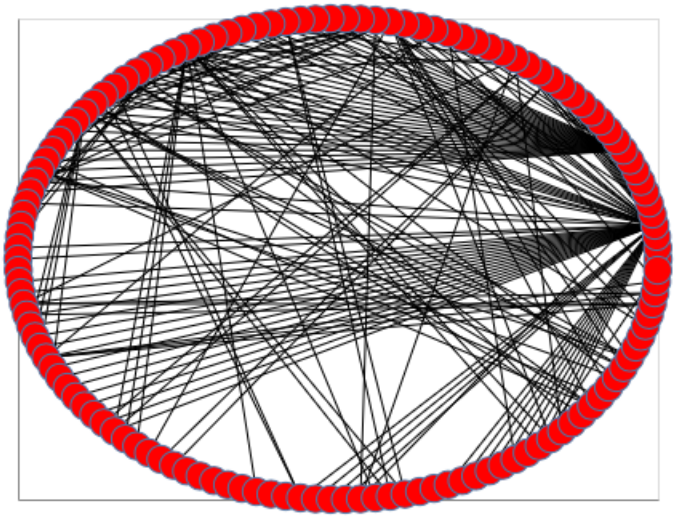
In [21]:
```python
# you can ask for the weights of specific edges:
season6.edges['Homer','Bart']['weight']
```

Out[21]: 19

In [22]:
```python
# You can even ask networkx to find the x-y positions for you:
circular_pos = nx.circular_layout(season6)
# circular_pos
```

In [23]:
```
# once you have the layout, you can ask nx-altair to draw the graph fo
nxa.draw_networkx(season6, pos = circular_pos)
```

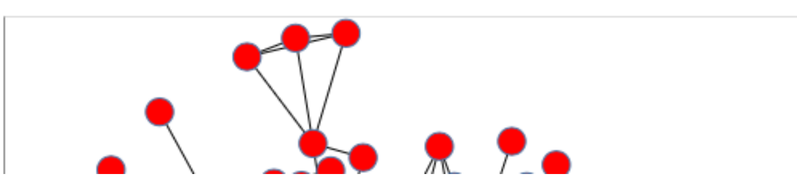
Out[23]:

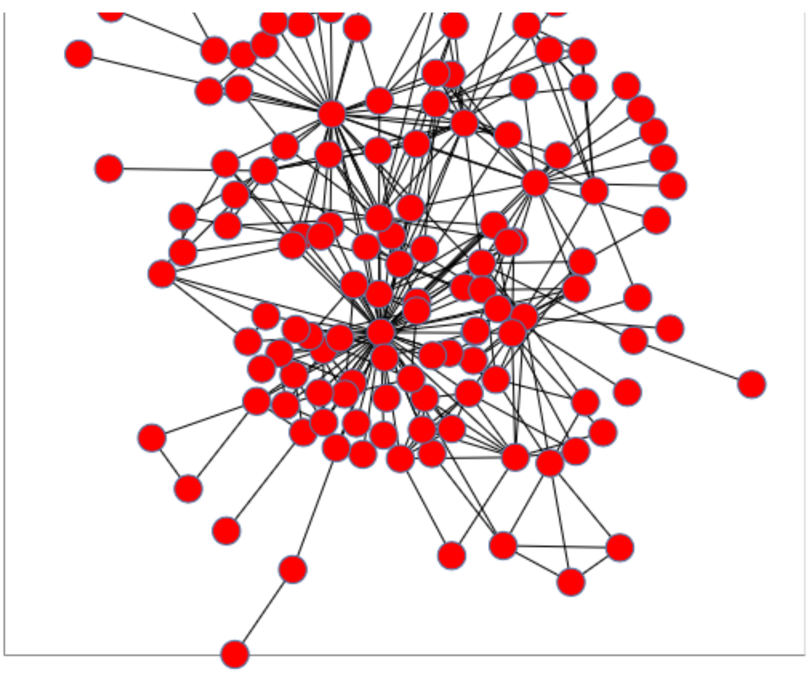


Clearly, a circular layout isn't going to be great here. Thankfully, networkx has many other layouts: [https://networkx.org/documentation/stable//reference/drawing.html#layout (https://networkx.org/documentation/stable//reference/drawing.html#layout)](https://networkx.org/documentation/stable//reference/drawing.html#layout).

In [24]:
```
# let's try a kamada kawai
pos = nx.kamada_kawai_layout(season6)

# to generate a good visualization (you shouldn't need to modify the l
nxa.draw_networkx(season6, pos = pos).properties(
    # nx-altair returns an Altair visualization, so we can modify
    # the properties as usual
    width=500,
    height=500
)
```

Out[24]:





Now let's modify it so that we can add more information.

We will use the `draw_networkx` function to control some properties of the visualization.

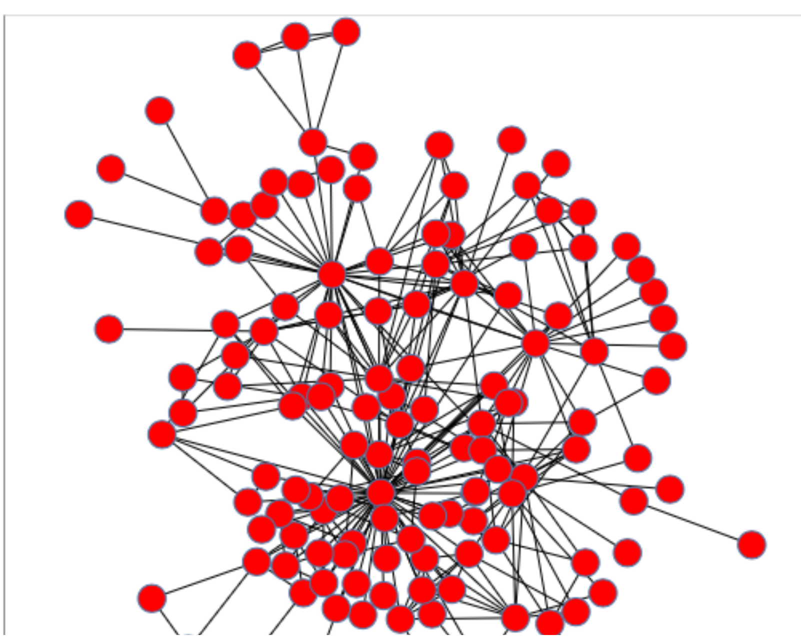
In [25]:
```
# recall that we have calculated 'weight' (an edge feature) and 'appe
# let's modify our network to visualize these:

# nxa.draw_networkx(season6, pos=pos,width='weight',node_color='appea
```

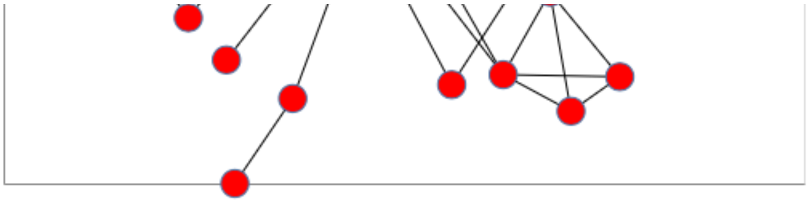
In [26]:
```
e = nxa.draw_networkx_edges(season6, pos=pos)  # get the edge layer
# e  # draw it
```

In [27]:
```
n = nxa.draw_networkx_nodes(season6, pos=pos)  # get the node layer
# n # draw it
```

In [28]:
```
# combine them back
(e+n).properties(
    width=500,height=500
)
```

Out[28]:





We're going to calculate a new feature of nodes based on a community detection algorithm. If you want to know more [read here](https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithm) (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithi
We'll augment our networkx object so we have a `community` "column" that can be used as a nominal feature of the nodes in visualization.

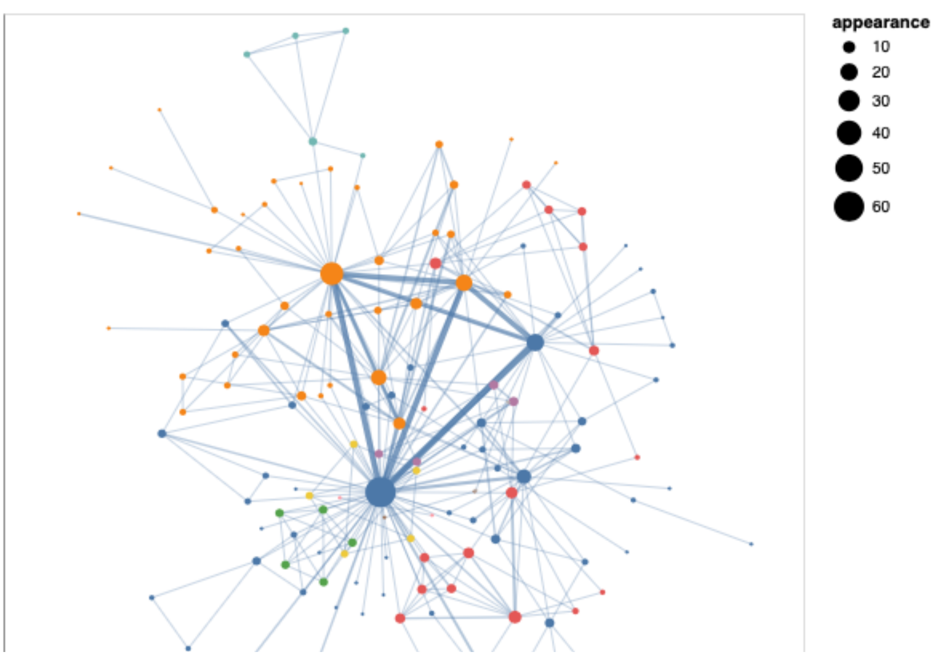
In [29]:
```
season6 = setCommunityLabels(season6, greedy_modularity_communities(se
```

In [30]:
```
# let's see what community Bart is now in:
season6.nodes['Bart']['community']
```

Out[30]: 2

In [31]:
```
e = nxa.draw_networkx_edges(season6, pos=pos)  # get the edge layer
n = nxa.draw_networkx_nodes(season6, pos=pos)  # get the node layer

# modify the code to change the encodings
n = n.mark_circle(opacity=1).encode(
    color=alt.Color('community:N',
                    legend=None
                   ),
    size=alt.Size('appearance:Q'),
    tooltip=['label:N']
)

e = e.mark_line().encode(
    strokeWidth=alt.StrokeWidth(
        'weight:N',
        legend=None),
    strokeOpacity='weight:N'
)


(e+n).properties(
    width=500,height=500
)
```

Out[31]:



Now that we have hte basics, we are going to build a visualization to help us compare pairs of seasons. We'd like to understand which characters have been more central to which seasons and how that has changed.

Key things we want to address:

- color based on community labels
- mouse-over interaction that changes the color of ALL of the visualizations if the character appears evverywhere
- tooltop over the nodes to get the # of appearances
- bars sorted based on changes between seasons

In [32]:
```
# we're going to want the networkx objects for different charts, so le

def getNetwork(season):
    # build a networkx object given the season, annotate with communit
    toret = buildNetwork(simpsons[simpsons.season == season])
    toret = setCommunityLabels(toret,greedy_modularity_communities(tor
    return(toret)
```

In [33]:
```
# get the networkx objects for seasons 5 and 9
s5net = getNetwork(5)
s9net = getNetwork(9)
```

In [34]:
```
# we also want the data for the two bar charts, we're going to do that

def getTotal(G):
    # total appearnce across all characters in a given graph
    app = 0
    for nd in G.nodes:
        app = app + G.nodes[nd]['appearance']
    return(app)

def getComparisonData(G1,G2,threshold=5):
    # generate two dataframes given two graphs
    # the first is the difference in apparances (normalized) when a ch
    # the second is for characters that are either in G1 or G2, but no
    # the threshold defines the cutoff for how many interactions a cha
    # to be included in the second ('difference') data frame
    t1total = getTotal(G1)
    t2total = getTotal(G2)
```

```
    t1total = getTotal(G1)
    union = []
    difference = []
    allentities = set(G1.nodes).union(set(G2.nodes))

    for i in allentities:
        if ((i in G1.nodes) & (i in G2.nodes)):
            diff = G1.nodes[i]['appearance']/t1total-G2.nodes[i]['appe
            union.append({'label':i,'difference': diff})
        elif (i in G1.nodes):
            if (G1.nodes[i]['appearance'] > threshold):
                difference.append({'label':i,'difference':-G1.nodes[i]
        elif (i in G2.nodes):
            if (G2.nodes[i]['appearance'] > threshold):
                difference.append({'label':i,'difference':G2.nodes[i][

    return(pd.DataFrame(union),pd.DataFrame(difference))
```

In [35]:
```
# let's compare the season 5 and 9 networks
union,difference = getComparisonData(s5net,s9net)

# look inside the union dataframe (the difference one will be similar)
union.sample(5)
```

Out[35]:

	label	difference
22	Marge	-0.000047
27	Superintendent Chalmers	-0.001853
6	Burns	0.000797
3	Nelson	0.006309
1	Mayor Quimby	0.003119

In [36]:
```
def getComparisonBar(frame,sel,title):
    # return an Altair chart corresponding to the bar chart example
    # above, given one of the two frames (difference or union)
    # frame is a pandas dataframe
    # sel is the Altair selection object (for interactivity)
    # title is the title for the chart

    sort_vals = frame.sort_values(by='difference',ascending=False)
    sort_vals = list(sort_vals['label'].unique())

    c1 = alt.Chart(frame).mark_bar().encode(
        x=alt.X('difference:Q',
                axis=alt.Axis(ticks=False,
                              grid=True,
                              labels=True),
                title=None),
        y=alt.Y('label:N',
                sort=sort_vals,
                axis=alt.Axis(ticks=True,
                              grid=False),
                title=None),
        color=alt.Color('difference:N',
                        scale=alt.Scale(scheme='blues'),
                        legend=None)
    ).properties(
        title=title,
        width=100,
        height=350
    ).add_selection(
        sel
    ).encode(
        color=alt.condition(sel,
                            alt.Color("difference:N",
                                      scale=alt.Scale(scheme='blues'),
                                      legend=None),
```

```
                            legend=None),
                            alt.value('lightgray'),
                            legend=None)
    )

    return(c1)
```

In [37]:
```
def getNetworkDiagram(G,season,sel):
    # return an Altair chart corresponding to network diagram above fo
    # G is the networkx object
    # season is the season
    # sel is the Altair selection object (for interactivity)

    seasonN = buildNetwork(simpsons[simpsons.season == season])
    pos = nx.kamada_kawai_layout(seasonN)
    edges = nxa.draw_networkx_edges(G, pos=pos)
    nodes = nxa.draw_networkx_nodes(G, pos=pos)

    nodes = nodes.mark_circle(opacity=1,size=250).encode(color=alt.Col

                                                     tooltip=['lab

    edges = edges.mark_line(color='black',strokeWidth=0.5)

    nodes = nodes.properties(
        title="Season "+str(season),
        width=400,
        height=350
    ).add_selection(
        sel
    ).encode(
        color=alt.condition(sel,
                            alt.Color("community:N",
#                                     scale=alt.Scale(scheme='c
                                     legend=None
                                    ),
                            alt.value('lightgray'),
                            legend=None)
    )

    c2 = (edges + nodes).resolve_scale(color='independent')

    return(c2)
```

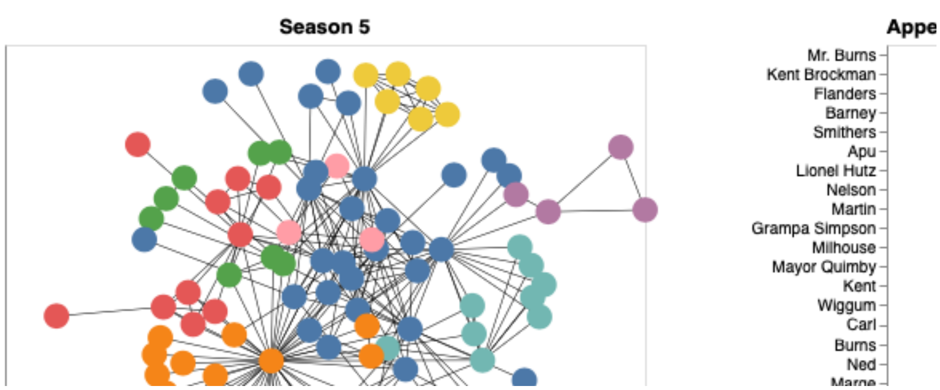
In [38]:

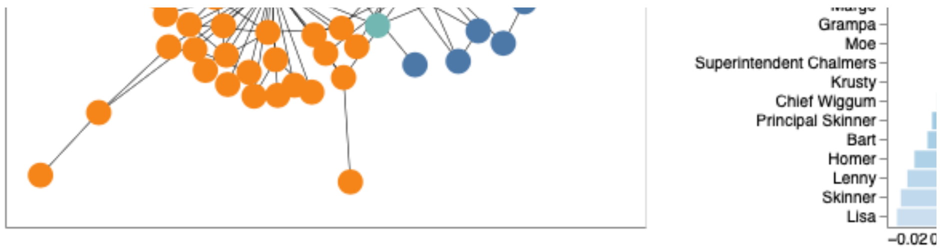
```
# this function will build the dashboard
def getNetworkDashboard(season1,season2):

    # create the selection object, based on mouseover. It should look
    # over as a way of deciding other objects with the same label
    single = alt.selection_single(on='mouseover',fields=['label'])

    # get the two networkx objects
    s1net = getNetwork(season1)
    s2net = getNetwork(season2)

    # get the union and difference dataframes
    union,difference = getComparisonData(s1net,s2net)

    # build the top bar chart
    u = getComparisonBar(union,single,"Appears in Both")

    # build the top network
    s1 = getNetworkDiagram(s1net,season1,single)

    # build the bottom network
    s2 = getNetworkDiagram(s2net,season2,single)

    # in some cases, we don't have new characters given the thresold
    if (len(difference) == 0):
        # we won't return the bottom chart
        return((s1&s2)|u)
    else:
        # we have both bar charts
        # build the bottom bar chart
        d = getComparisonBar(difference,single,"New Characters")
        # return all charts
        return((s1&s2)|(u&d))
```

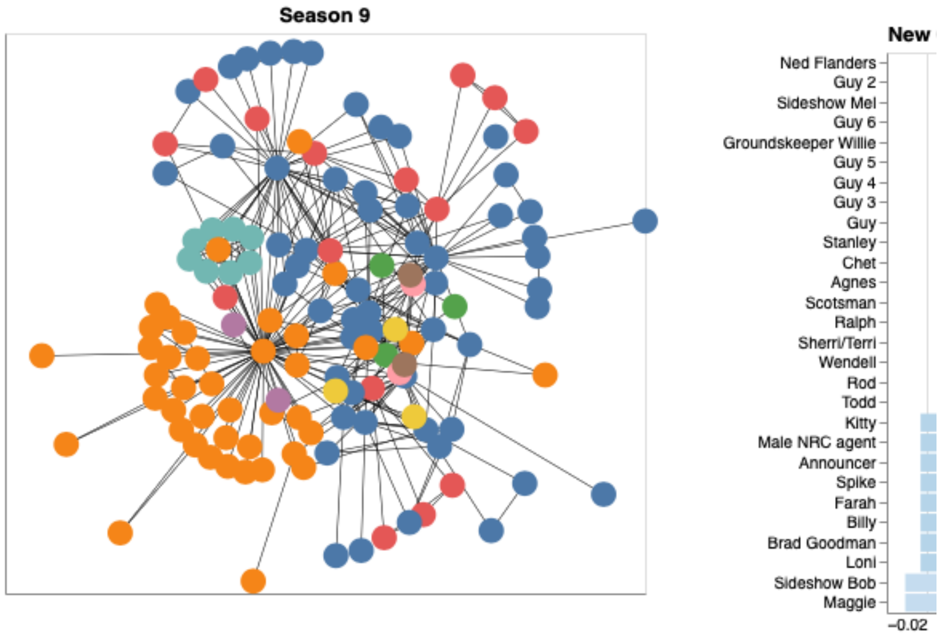
In [39]:

```
# implement dashboard
getNetworkDashboard(5,9)
```

Out[39]:

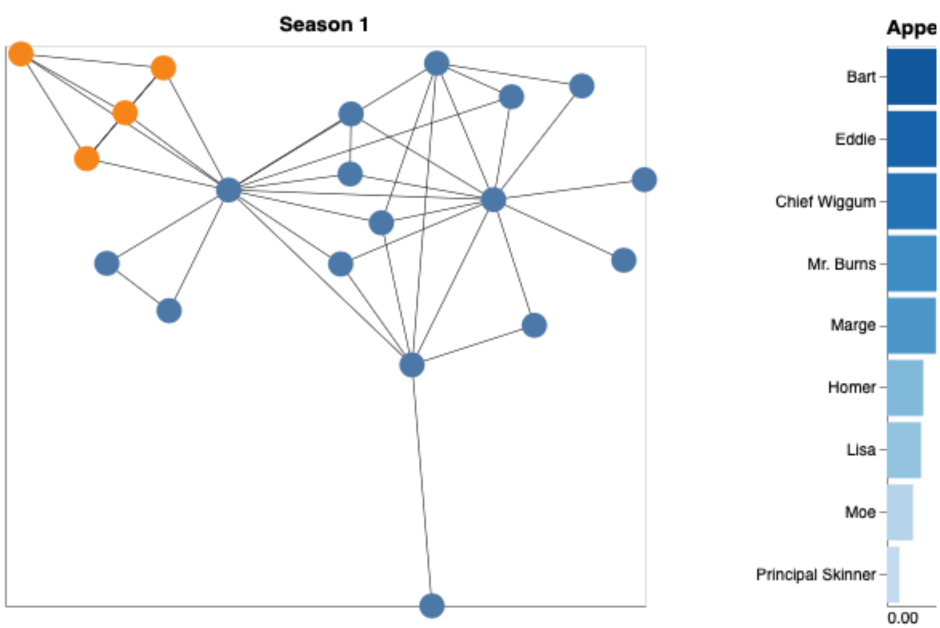


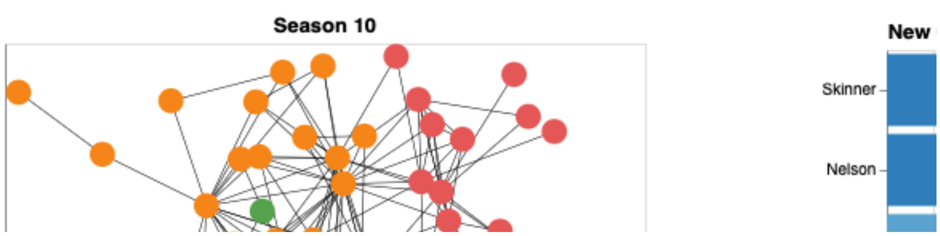


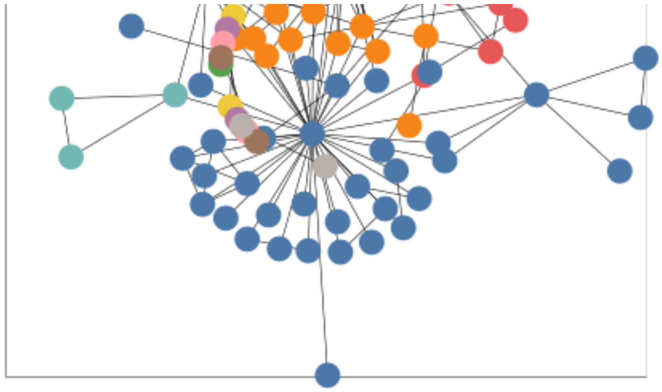


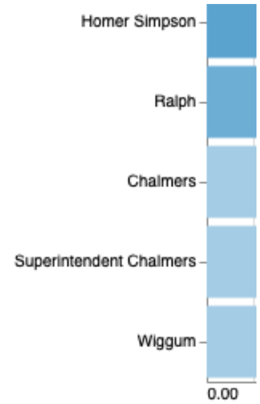
In [40]:
```
# Let's see the differences between season 1 and season 10
getNetworkDashboard(1,10)
```

Out[40]:









Matrix Representation of Network

For the final exercise, we will generate a matrix representation of the network. To reduce noise in the dataset, we will include only characters who have interacted 6 or more times with each other. We will use a tooltip to provide a bit more detail.

In [41]:
```
def getMatrixDetails(df,threshold=6,removeIsolates=True):
    # given a dataframe with characters (c1,c2, etc.)
    # the returned matrix will find the number of interactions in the
    # find statistics to generate a matrix representation
    # threshold will be the minimum number of interactions between cha
    # removeIsolates determiens if isolated nodes (nodes not connected

    # this function returns 3 things
    # the long form dataframe with pairs of nodes and the count
    # the node order of nodes in the matrix given the input
    # a list of list -- an edge list for all nodes
    t = buildNetwork(df)
    for e in t.edges:
        if (t.edges[e]['weight'] < threshold):
            t.remove_edge(e[0],e[1])
    if(removeIsolates):
        t.remove_nodes_from(list(nx.isolates(t)))

    m,names,a,b,w = [],[],[],[],[]

    for n1 in t.nodes:
        e = []
        names.append(n1)
        for n2 in t.nodes:
            if(t.has_edge(n1,n2)):
                a.append(n1)
                b.append(n2)
                w.append(t.edges[n1,n2]['weight'])
                e.append(t.edges[n1,n2]['weight'])
            else:
                e.append(0)
        m.append(e)

    toret = pd.DataFrame()
    toret['p1'] = a
    toret['p2'] = b
    toret['weight'] = w
    return(toret,names,m)
```

In [42]:
```
# let's call this for the entire dataset
df,names,m = getMatrixDetails(simpsons)
```

```
In [43]: # we'll get back to names and m in a moment, but let's look at what's
df.sample(5)
```

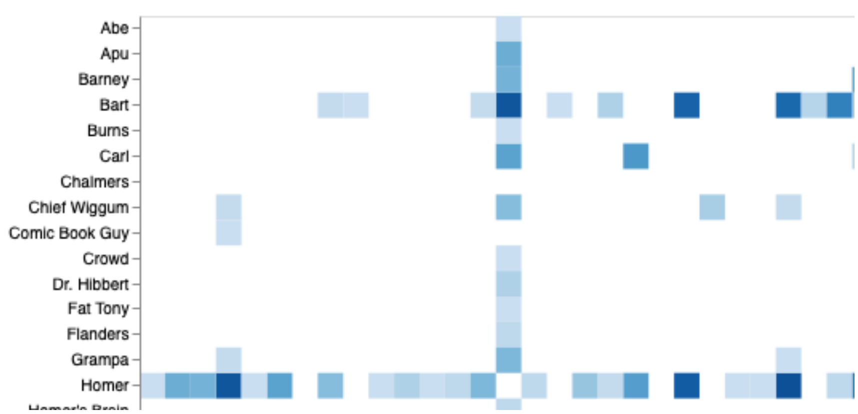
Out[43]:

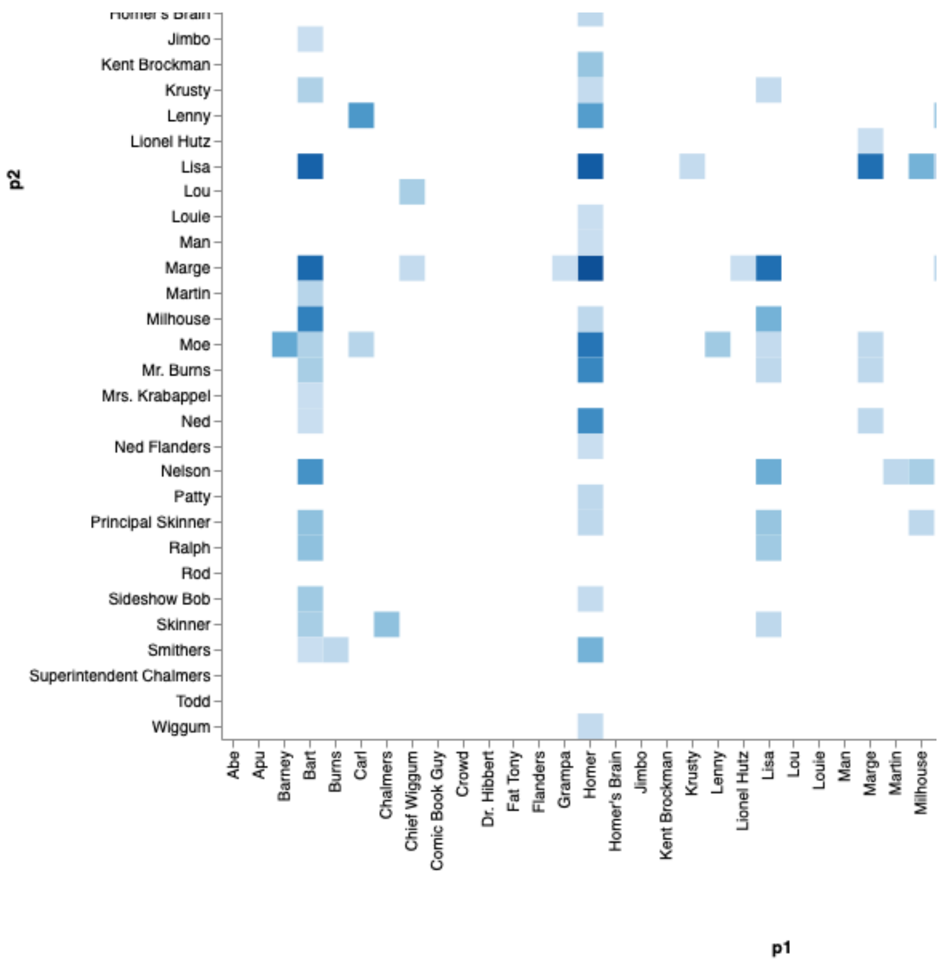
	p1	p2	weight
51	Carl	Homer	24
40	Bart	Principal Skinner	14
45	Bart	Grampa	7
20	Homer	Kent Brockman	13
36	Bart	Moe	10

```
In [44]: def genMatrix1(inframe,threshold=6):
    # takes an input frame as input
    # returns an altair plot for the matrix as described above
    df,names,m = getMatrixDetails(inframe,threshold=threshold)

    # modify the following
    toret = alt.Chart(df).mark_rect().encode(
        x=alt.X('p1:N'),
        y=alt.Y('p2:N'),
        color=alt.Color('weight:N', scale=alt.Scale(scheme='blues')),
        tooltip=['p1','p2','weight']
    ).properties(width=700,height=700)

    return(toret)
```

```
In [45]: genMatrix1(simpsons)
```

Out[45]:





The problem with this layout is that it is rather arbitrary (alphabetical on character names). This makes pattern recognition in the data difficult. One solution is to reorder the rows and columns so that close characters that are similar will end up close to each other.

This can be done with either [Seaborn's clustermap (https://seaborn.pydata.org/generated/seaborn.clustermap.html)](https://seaborn.pydata.org/generated/seaborn.clustermap.html), or what we will use here: Scipy's [agglomerative clustering (https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html) and modify the [linkage analysis (https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html)](https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html) used to generate the dendrogram to find the order of the leaves.

In [46]:
```
# a function to re-order using the agglomerative clustering and dendro

def getNewOrder(mtrx,originalorder):
    # determine the new order given an "edge list representation"
    # accepts the "original order" returns a new order
    model = AgglomerativeClustering(distance_threshold=0, n_clusters=N
    model = model.fit(mtrx)
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances
                                      counts]).astype(float)
```

```
    leaves = leaves_list(linkage_matrix)
    neworder = []
    for l in leaves:
        neworder.append(originalorder[l])
    return(neworder)
```

The scipy clustering code requires a "vector" representation of each node (which we calculated when we ran `getMatixDetail`). This looks much like the edge list representation.

Each vector will be compared to all others giving us the "distance" between characters and that will be used to cluster.

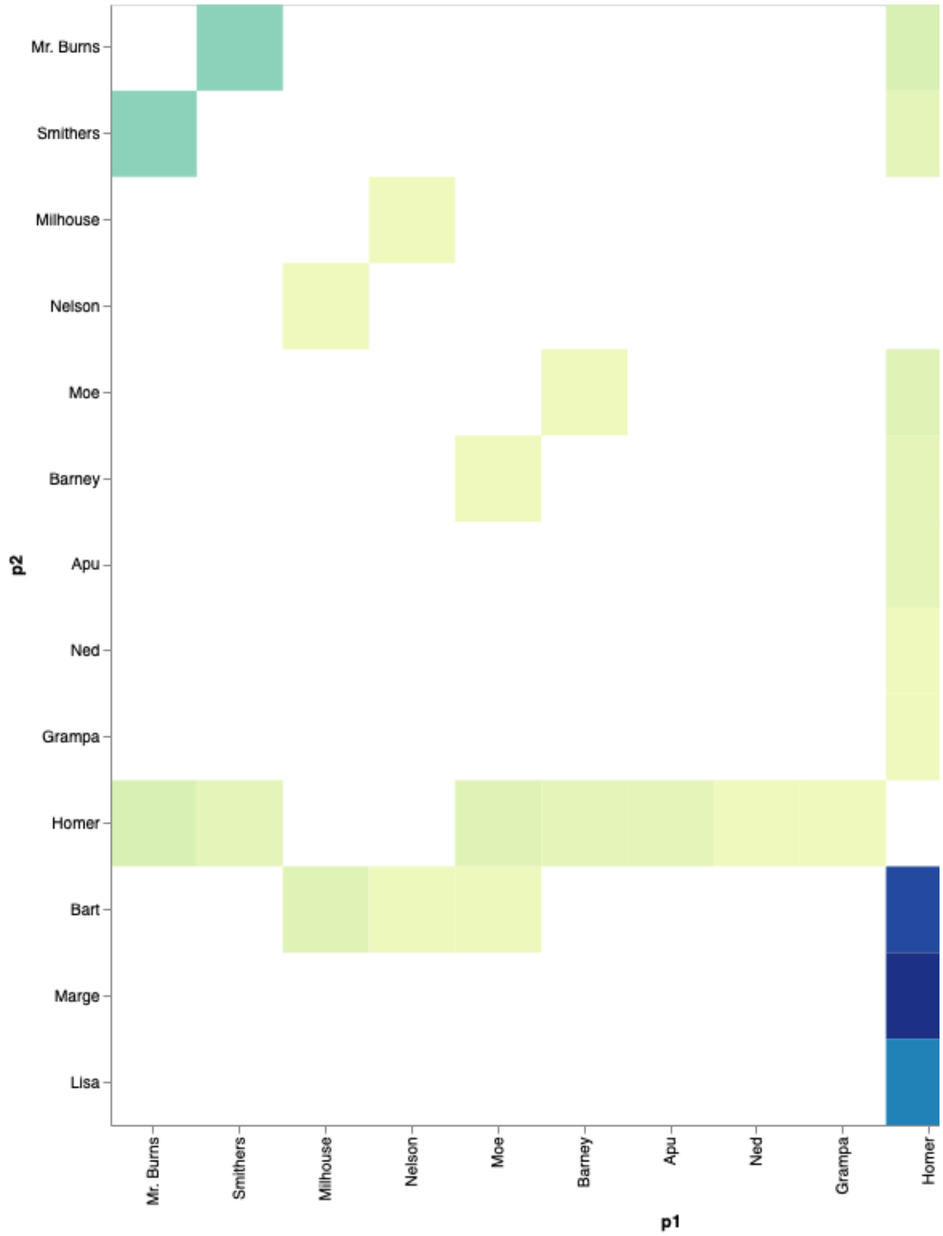
In [47]:
```
df,names,m = getMatrixDetails(simpsons[simpsons.season <= 6],threshold

print("The first character in m is:",names[0])

print("It is represented by the vector:",m[0])
```

```
The first character in m is: Homer
It is represented by the vector: [0, 62, 11, 9, 9, 69, 6, 50, 13, 9,
0, 0, 6]
```

In [48]:
```
def genMatrix2(inframe,threshold=6):
    # takes an input frame as input
    # returns an altair plot for the matrix as described above
    df,names,m = getMatrixDetails(inframe,threshold=threshold)

    neworder = getNewOrder(m,names)

    # modify the following

    toret = alt.Chart(df).mark_rect().encode(
        x=alt.X('p1:N',sort=neworder),
        y=alt.Y('p2:N',sort=neworder),
        color=alt.Color('weight:Q'),
        order=alt.Order("neworder:N"),
        tooltip=['p1','p2','weight'],
    ).properties(width=700,height=700)

    return (toret)
```

In [49]:
```
# If you im
genMatrix2(simpsons[simpsons.season <= 6])
```

Out[49]:



- Pattern 1 - The blue cluster at the bottom right indicates significantly more interaction between those characters. This makes sense, as the Simpson family are the main characters.
- Pattern 2 - The lines extending along Homer, Bart, and less so Lisa's axes indicate interaction with a multitude of characters. This pattern also allows us to see that sometimes these 3 main characters are the only ones the tertiary characters interact with.

Exercise adapted and modified from UMSI homework assignment for SIADS 622.