Unit 6: Visualization

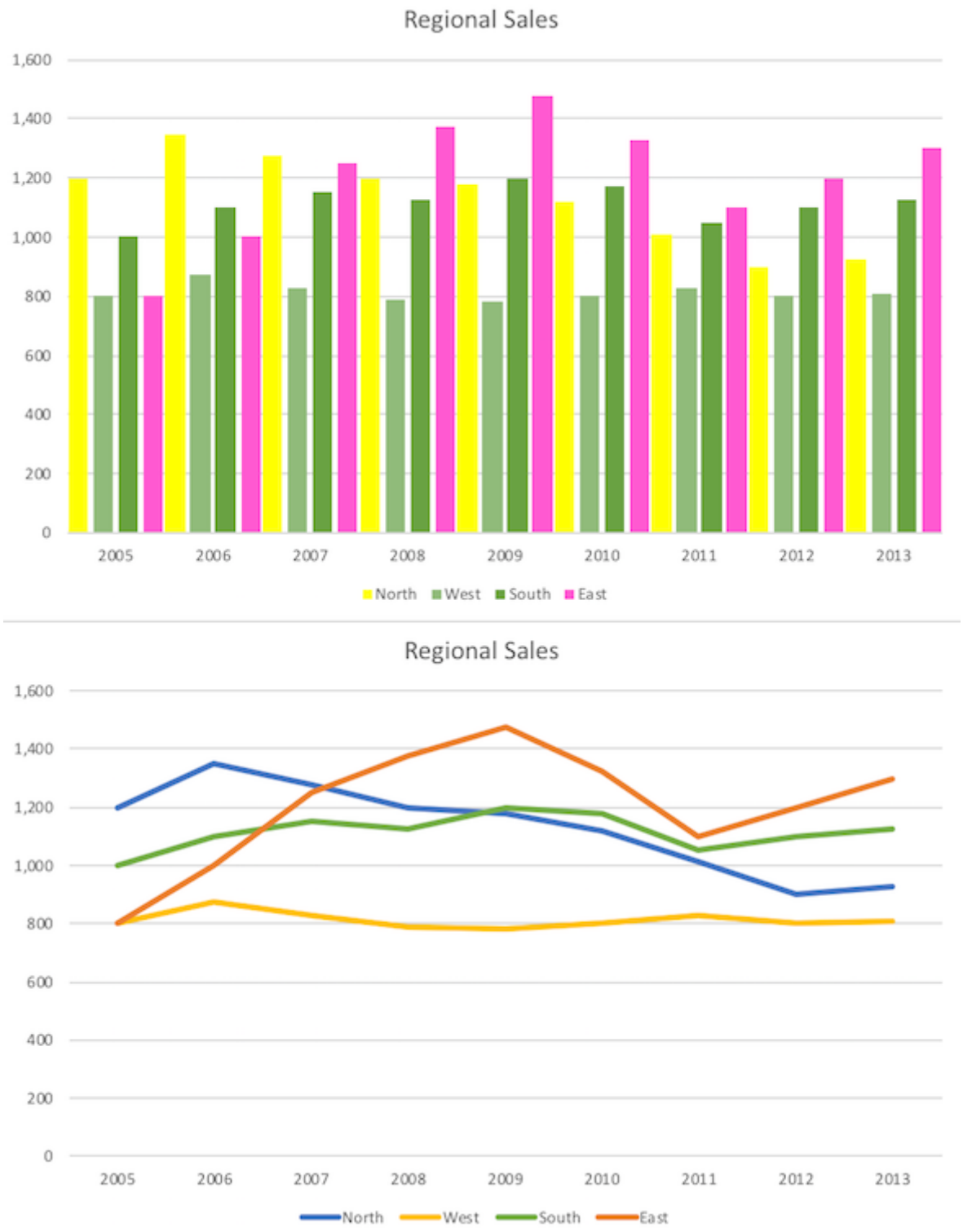
Contents

Lab Questions

For this unit, the notebook itself will not contain lab questions. Instead, we'll create visualizations in Tableau (https://www.tableau.com/) in the second part of the unit.

Getting Started

As we've examined and explored data, we often used visualizations to help make sense of things. While the visualizations we've created have been sufficient for exploration, creating visualizations for reporting or to convey a story requires more attention to design and other details. In this unit, we'll look at some design principles and the appropriate chart types for the given data. As an example, consider the charts in the figure below. The bottom chart is more visually appealing than the top chart. In this unit, we'll explore some of the reasons why this is the case.



A "bad" chart (top) and an "good" chart (bottom)

When looking at examples of types of visualizations, we'll create them with both Seaborn (https://seaborn.pydata.org/)/Matplotlib (https://matplotlib.org/) and Tableau (https://www.tableau.com/). Tableau is a popular data visualization product that provides an interface to many file and database sources and a simple-to-use but feature-rich interface. A version of Tableau can be downloaded from here (https://www.tableau.com/products/desktop/download?signin=academic) but will require a license to use.

We'll also make use of squarify (https://github.com/laserson/squarify) to generate tree maps and Folium (https://python-visualization.github.io/folium/docs-v0.5.0/) for geographic mapping.

In [31]:
```
import sys
!{sys.executable} -m pip install squarify folium
```

```
Requirement already satisfied: squarify in /usr/local/lib/python3.6/site-
packages
Requirement already satisfied: folium in /usr/local/lib/python3.6/site-pa
ckages
Requirement already satisfied: branca in /usr/local/lib/python3.6/site-pa
ckages (from folium)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.6/site-pa
ckages (from folium)
Requirement already satisfied: six in /usr/local/lib/python3.6/site-packa
ges (from folium)
Requirement already satisfied: requests in /usr/local/lib/python3.6/site-
packages (from folium)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python
3.6/site-packages (from jinja2->folium)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/py
thon3.6/site-packages (from requests->folium)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/pytho
n3.6/site-packages (from requests->folium)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in /usr/local/lib/py
thon3.6/site-packages (from requests->folium)
Requirement already satisfied: idna<2.7,>=2.5 in /usr/local/lib/python3.
6/site-packages (from requests->folium)
```

Visualization Design

Effective visualizations communicate their intent clearly and quickly. If viewers have to spend a great deal of time deciphering a visualizations, they might as well be presented with the underlying data. There are a variety of aspects and principles that can be considered when trying to create an effective visualization. One of these is to consider the intended audience. We might choose to create a more technically-focused visualization or one with a deeper level detail if we expect that the viewer is familiar with the data or the context in which it exists as opposed to someone without any prior experience.

Another principle principle might be to keep the visualization as simple as possible. Though there might be a lot of data available for visualization, simpler visualizations are easier to understand and are less likely to be misunderstood that visualizations with many elements. If there is a need to visualize a lot of data, consider creating separate visualizations for groups of data.

Additional principles we might consider when creating a visualization involve the perception of graphical elements in relation to one another and how color can be used effectively.

Perception

When considering perception of graphical elements, we can rely on the Gestalt psychology principles of grouping (https://en.wikipedia.org/wiki/Principles_of_grouping). Among the principles, those that are of particular interest are

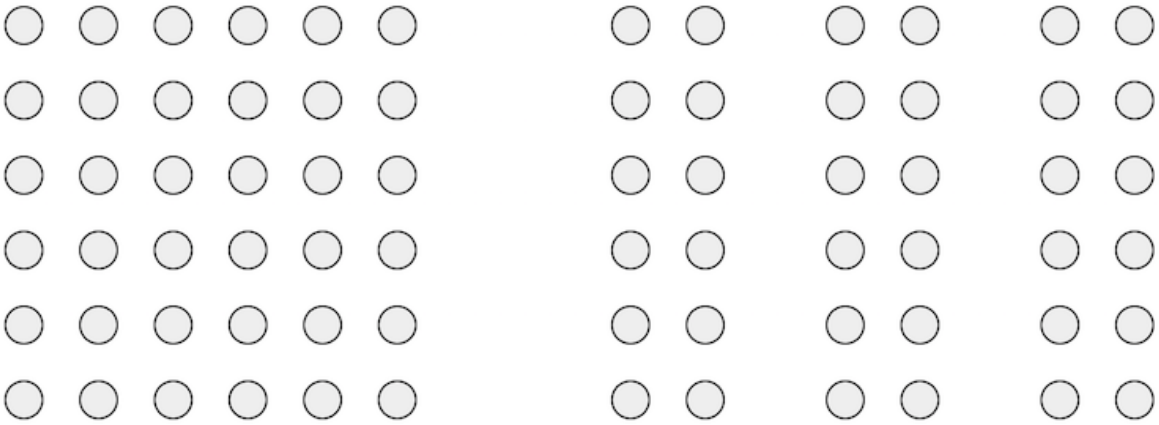
- Proximity

- Similarity
- Enclosure
- Closure
- Continuity
- Connection

Proximity

Proximity refers to the the nearness or distance between objects. In visualizations, nearness can be used to imply a relationship or connectedness between graphical elements. Similarly, distance between elements implies that they are unrelated.

As an example, consider the following image. On the left, the spacing between each column is used the same and we likely perceive the dots as being part of a single collection. On the right, the spacing is such that we see three sets of dots. This notion of proximity is what allows us to understand that bars of differing length but equal width and spacing are used to present related data in bar charts (as we'll see later).

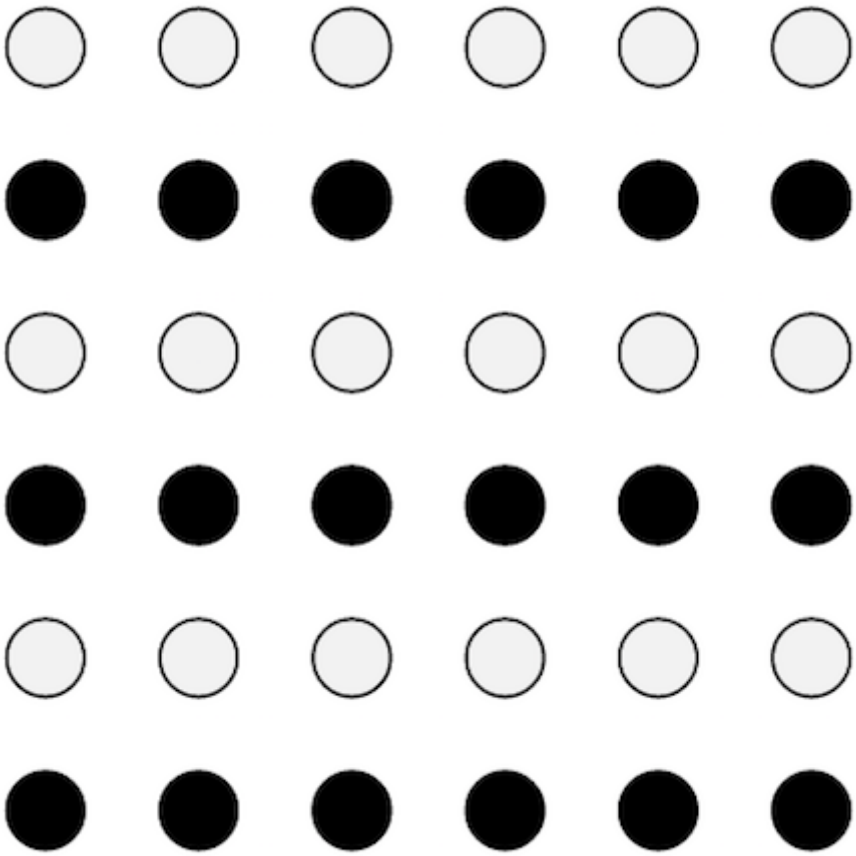


Principle of Proximity, source (https://commons.wikimedia.org/wiki/File:Gestalt_proximity.svg#/media/File:Gestalt_proximity.s

Similarity

The principle of similarity states that objects that share attributes are perceived as being part of a group and object with different attributes are not part of that group.

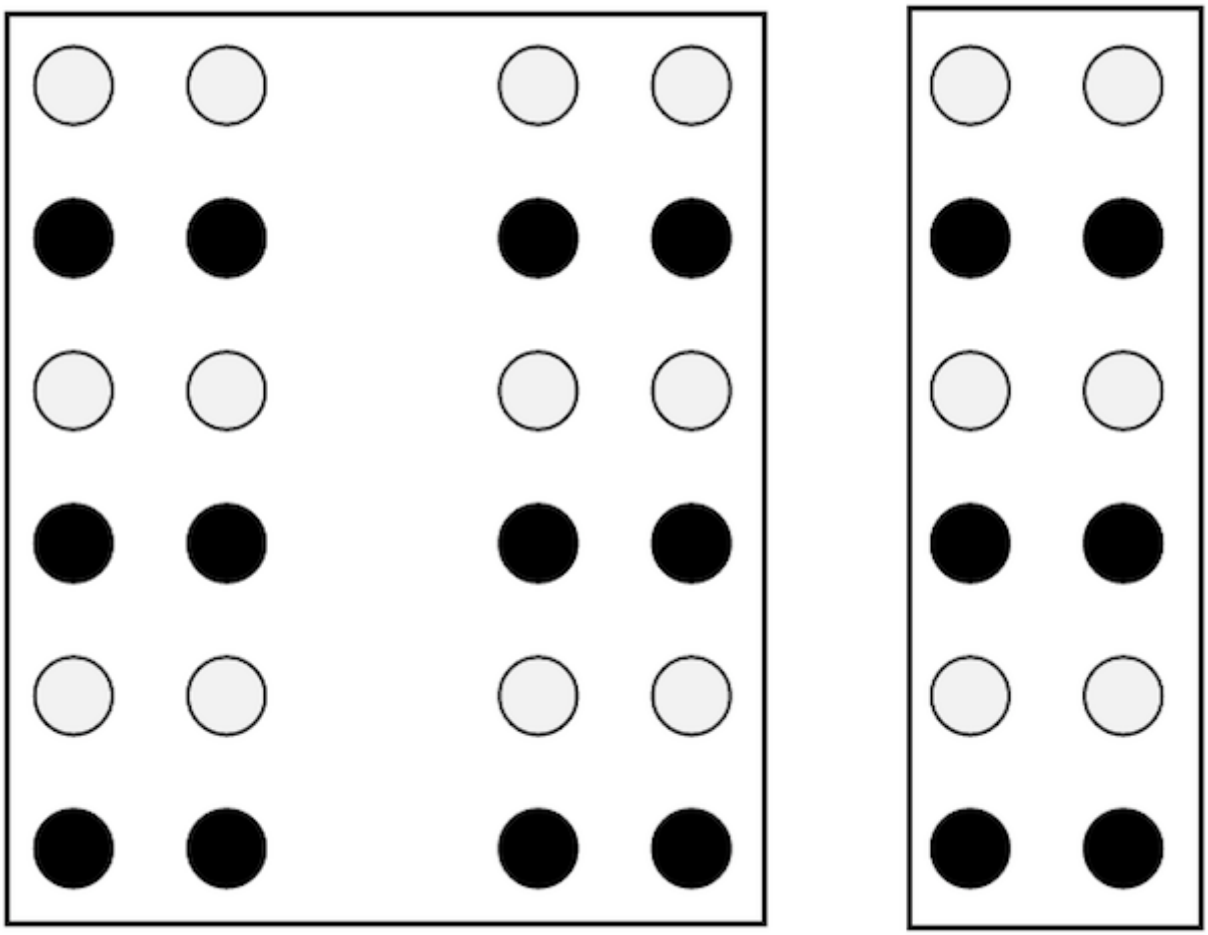
In the following image, the shading of each row of dots conveys the existence of multiple groups. The first and second rows are perceived as belonging to different groups. We can make use of similarity to present similar data on the same plot. For example, we can plot lines representing different (but related) things using different colors.



Principle of Similarity, source (https://commons.wikimedia.org/wiki/File:Gestalt_similarity.svg#/media/File:Gestalt_similarity.s

Enclosure

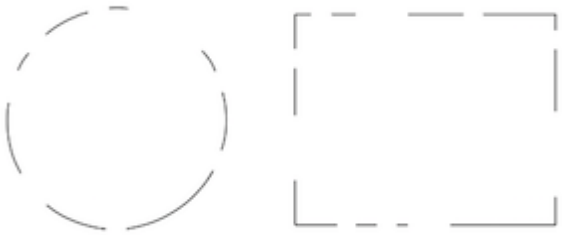
Enclosure refers to the idea that objects with a shared boundary, whether a line or an area with a common shading, are perceived as being part of the same group. Though this principle is not often used in visualizations it can be very good at achieving its purpose. For example, in the image below, we perceive that the dots are placed in two main groups with each group being defined by a border line. Notice that the principle of enclosure is more powerful than similarity (imposed by the color of each dot) and proximity (imposed by the difference in horizontal spacing between dots).



Principle of Enclosure

Closure

Closure refers to our ability or tendency to complete patterns when part of the information is missing. This tendency is strongest for simpler shapes as demonstrated by the following image where we naturally see a circle and rectangle rather than a collection of unrelated line segments. This is useful in visualizations when we wish to establish a trend or have the viewer draw conclusions despite missing data.

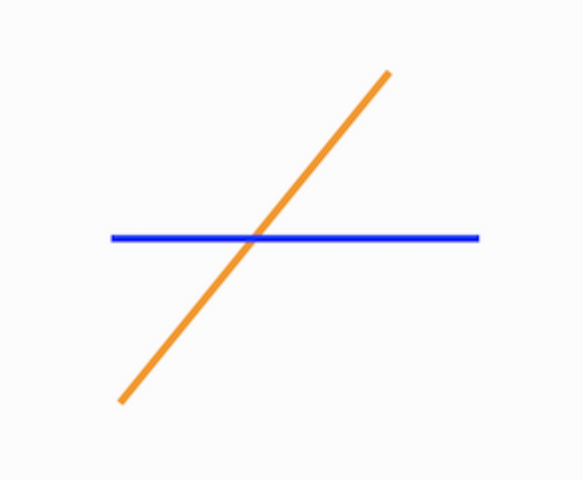


Principle of Closure, source (https://commons.wikimedia.org/wiki/File:Gestalt_closure.svg#/media/File:Gestalt_closure.svg)

Continuity

The principle of continuity states that objects that are aligned or appear to be continuous will be perceived as such. This is demonstrated in the figure below. Despite the fact that there are actually two separate, orange line segments, we perceive them as being part of a longer segment. Plotting

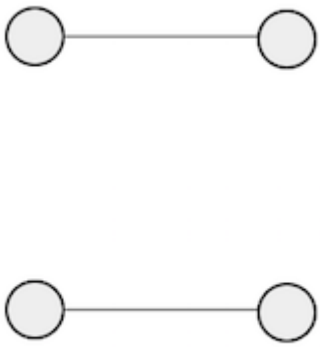
different lines on one chart makes use of this principle allowing us to understand that even if lines intersect, each line continues after the intersection, i.e. the lines represent the same group of data before and after the intersection.



Principle of Continuity

Connection

Connection is more straightforward than some of the other principles. It implies that if elements are connected by other elements, such as lines, we will perceive a grouping based on the connection. This is demonstrated in the figure below.



Principle of Connection

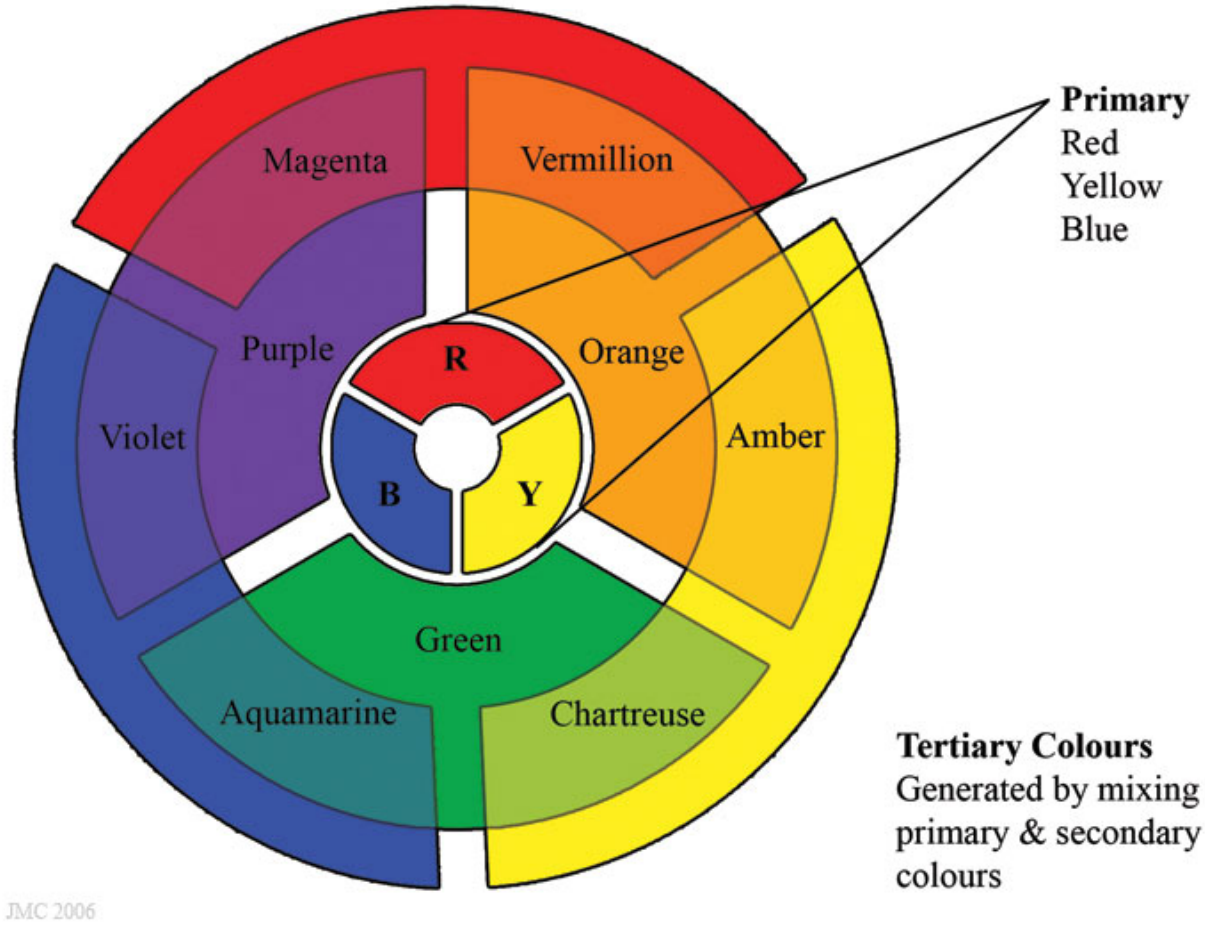
Choosing a Palette

Color plays in an important part in the visualizations we create. Color can be use to distinguish between elements, to set one element apart from other, to indicate value, or even to convey a certain mood within a visualization. At best, ignoring color choices can result in visualizations that are difficult to understand; at worst, it can result in something that viewers find jarring. To choose the appropriate palette, we'll rely on color theory (https://en.wikipedia.org/wiki/Color_theory).

While it's important to be aware of color theory, many visualization software packages will attempt to make the appropriate color choices for the data we're plotting.

The Color Wheel

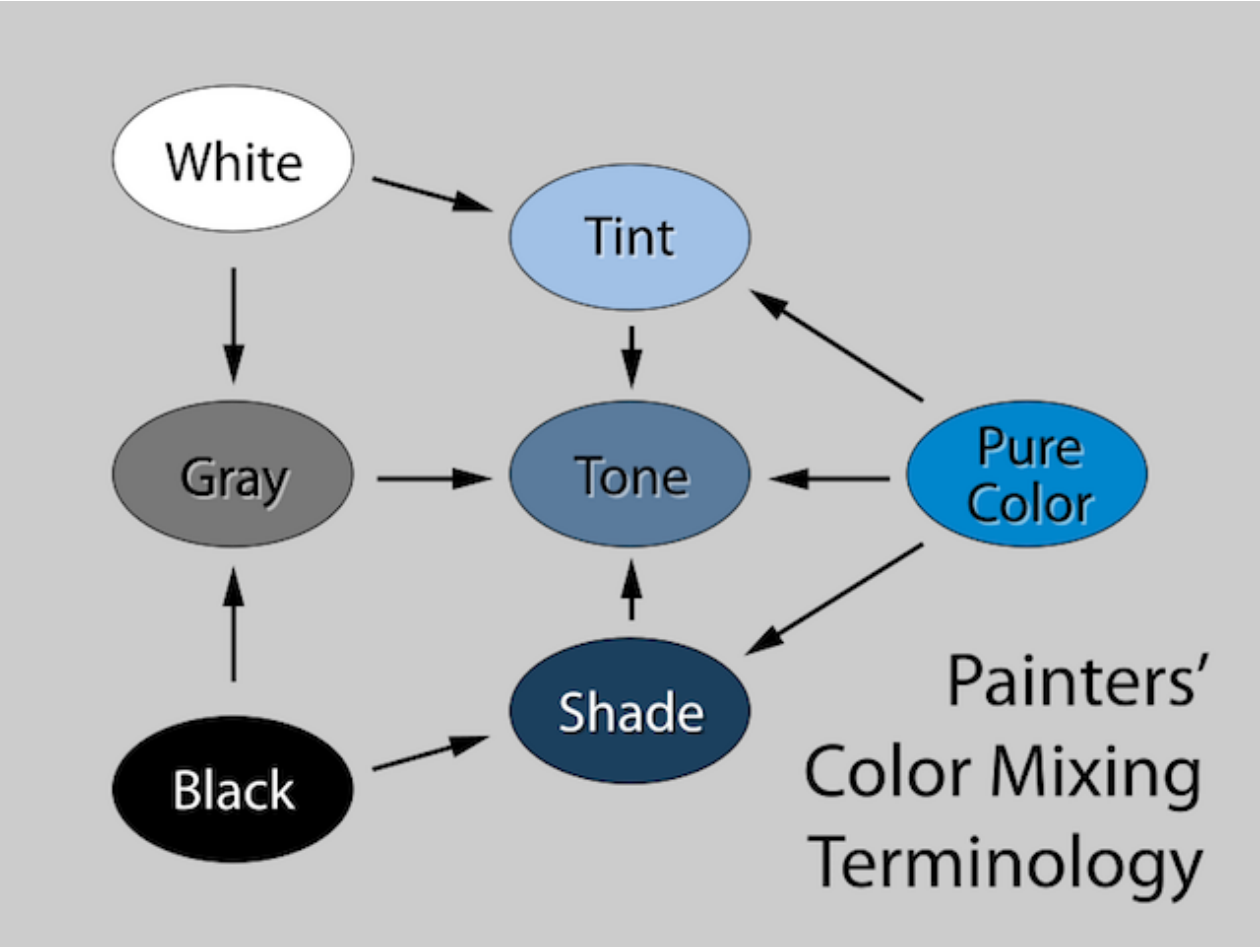
When discussing color and how individual colors relate to one another, we often begin by looking at the color wheel. While there are different color wheels depending on whether we are looking at mixing light (https://en.wikipedia.org/wiki/Additive_color) or mixing pigments (https://en.wikipedia.org/wiki/Subtractive_color) and what we define as the primary colors, we will use the historical Red-Yellow-Blue (RYB) (https://en.wikipedia.org/wiki/RYB_color_model) for our discussion. In this model, we assume that there are three *primary* colors from which most other colors can be derived through mixing; these colors are red, yellow, and blue. Combining any two of these colors gives us the *secondary* colors (orange, green, and purple. Combining a primary with one of its associated secondary colors gives us a *tertiary* color. See the figure below.



The RYB Color Wheel, source (https://en.wikipedia.org/wiki/File:RBYcolourwheel.jpg#/media/File:RBYcolourwheel.jpg)

Hue, Tint, and Shade

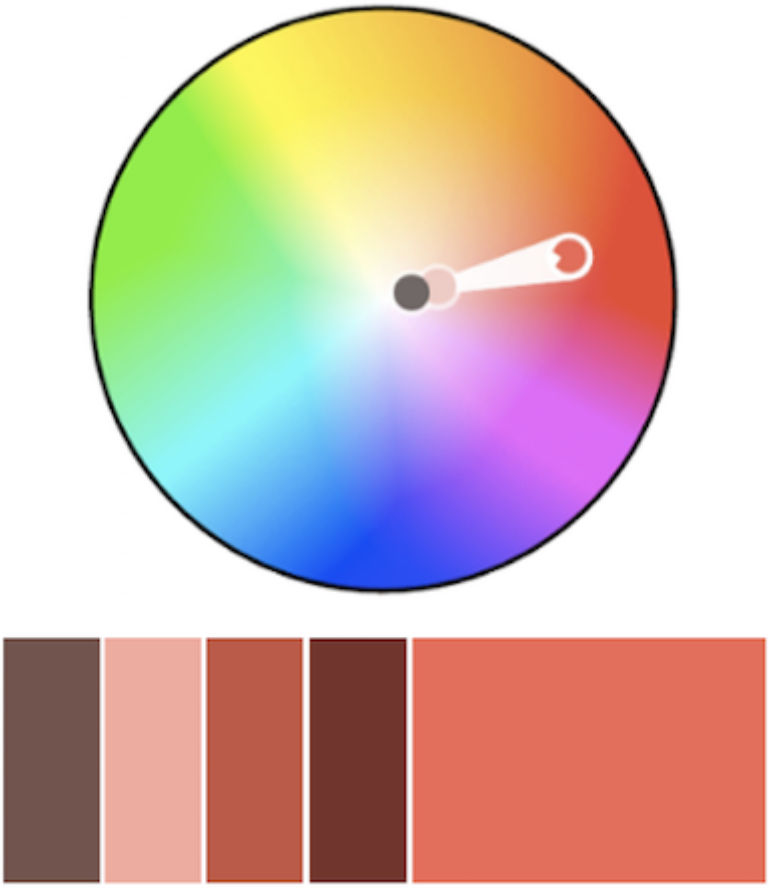
In addition to the the notion of primary, secondary, and tertiary colors, it's also important to be aware of hue, tint, shade, and tone. A *hue* is "pure" color as it exists on the color wheel. Adding white pigment to a hue results in a *tint*. A *shade* of a color is the result of mixing the hue with black pigment. A *tone* is mixing some amount of white and black pigment (gray) with the hue.



Hue, Tint, Tone, and Shade, source (https://commons.wikimedia.org/wiki/File:Tint-tone-shade.svg#/media/File:Tint-tone-shade.svg)

Monochromatic Color Scheme

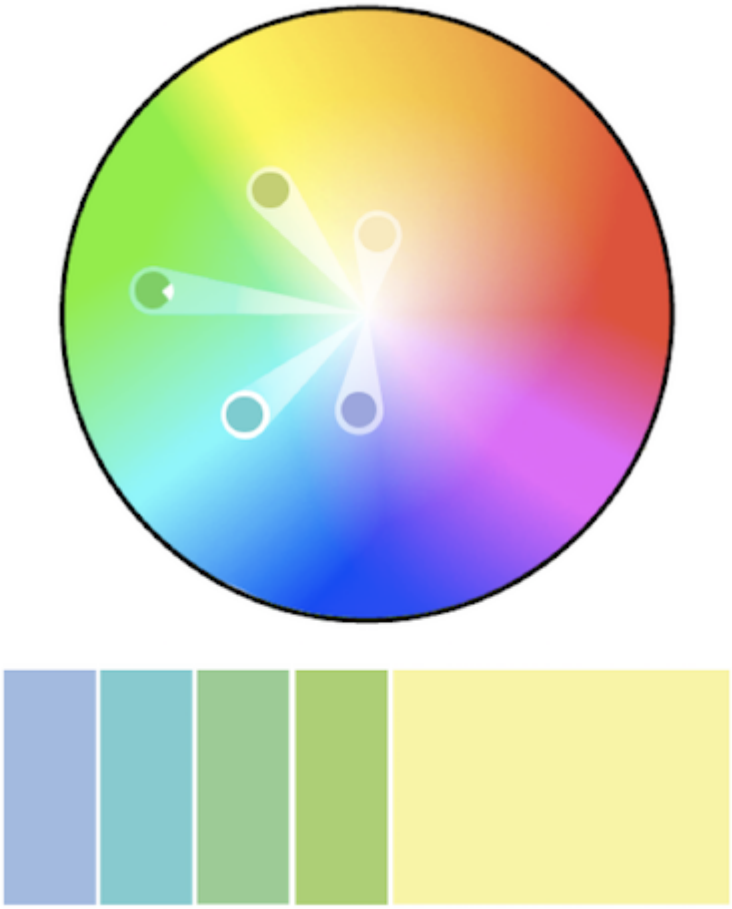
The first type of palette we'll consider is one based on a monochromatic color scheme. In this scheme, we first select a hue and then choose shades, tones, and tints based on this hue. A monochromatic color scheme is often used for sequential, discrete data. We can use a monochromatic scheme with continuous data where we slowly move from a shade to a tint.



A Monochromatic Palette, source (https://blog.hubspot.com/marketing/color-theory-design)

Analogous Color Scheme

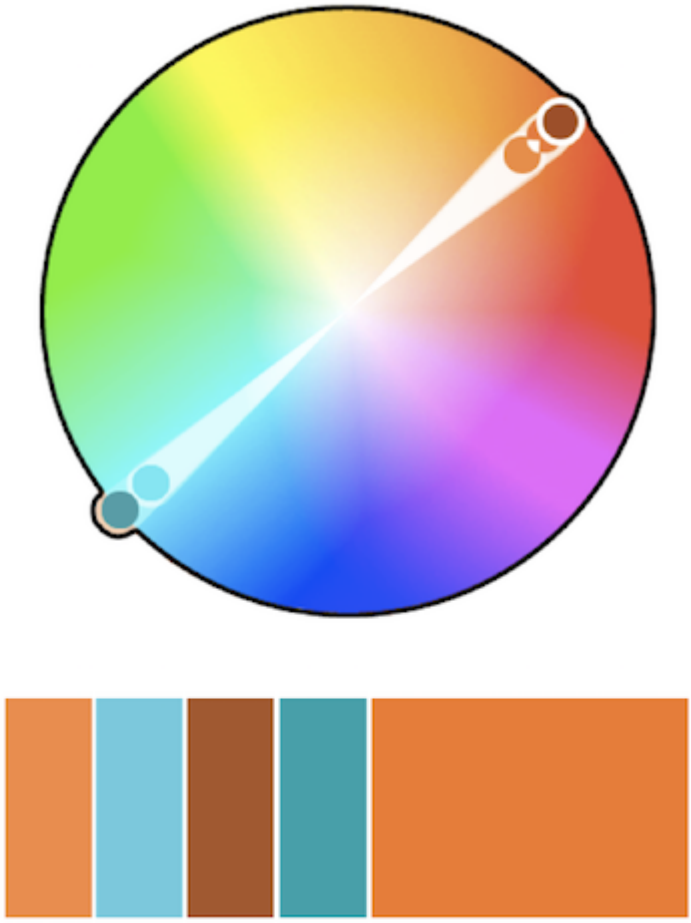
An analogous color scheme is chosen by first selecting a main color. Next, we select the two adjacent colors. If necessary or desired, we an select the next two adjacent colors as well. With these colors, we choose tints and shades. The analogous color scheme is often used with sequential data.



An Analogous Palette, source (https://blog.hubspot.com/marketing/color-theory-design)

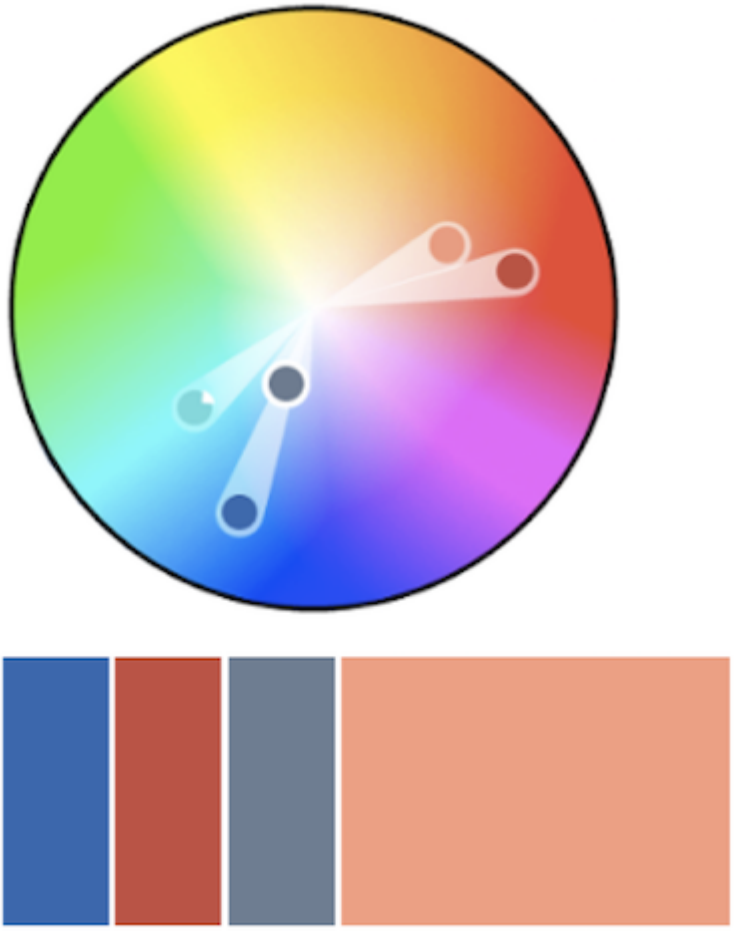
Complementary Color Scheme

A complementary color scheme makes use of colors directly opposite one another on the color wheel. This choice offers the greatest amount of contrast between colors. While we could use the hues of both colors, this can be too striking; instead, we should choose one color as the predominant color and use tints and shades of the other color. A complementary scheme is a good choice when emphasizing extreme values or when we wish to visualize divergent data.



A Complementary Palette, source (https://blog.hubspot.com/marketing/color-theory-design)

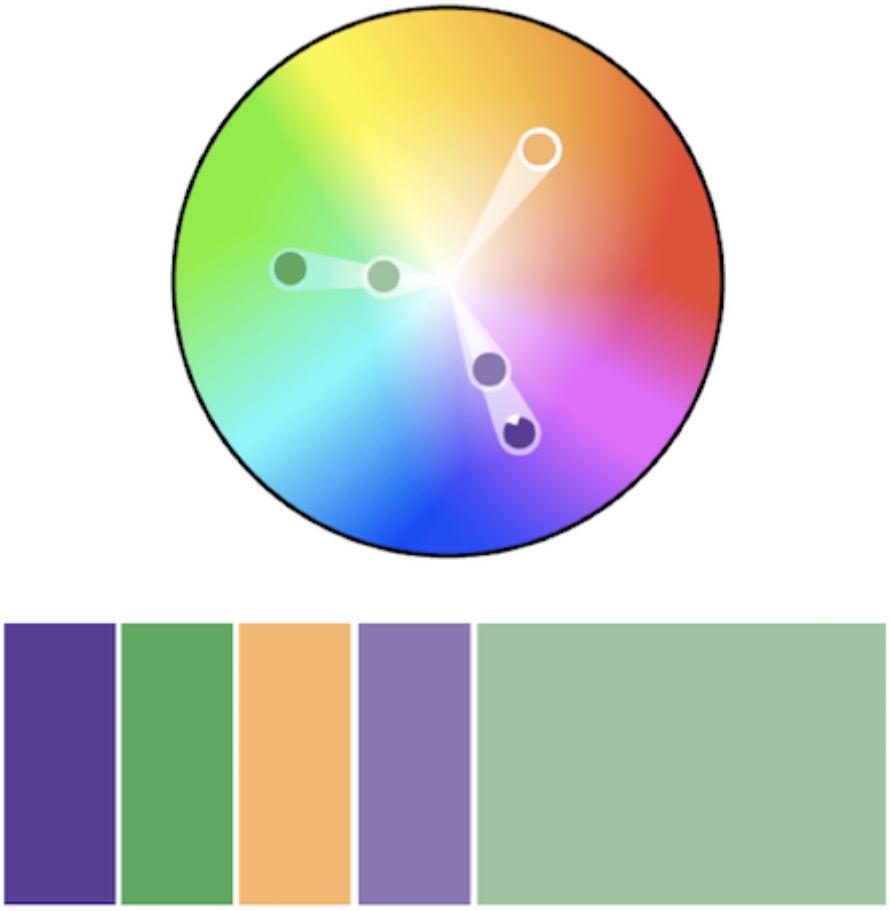
For a more "nuanced" palette, one could use a *split complementary* scheme in which two complementary colors are paired with adjacent colors.



A Split Complementary Palette, source (https://blog.hubspot.com/marketing/color-theory-design)

Triadic Color Scheme

Similar to the complementary color scheme, a triadic color scheme is chosen by selecting three colors on the color wheel with an equal angle between pairs of colors. Often, a main hue is chosen and tints and shades of the other colors are used. A triadic color scheme is often useful when working with qualitative or categorical data. This idea can be extended for four, five, six, etc. colors. It is important to avoid choosing similar tints and shades for different visual elements as this can be confusing.



A Triadic Palette, source (https://blog.hubspot.com/marketing/color-theory-design)

Warm and Cool Colors

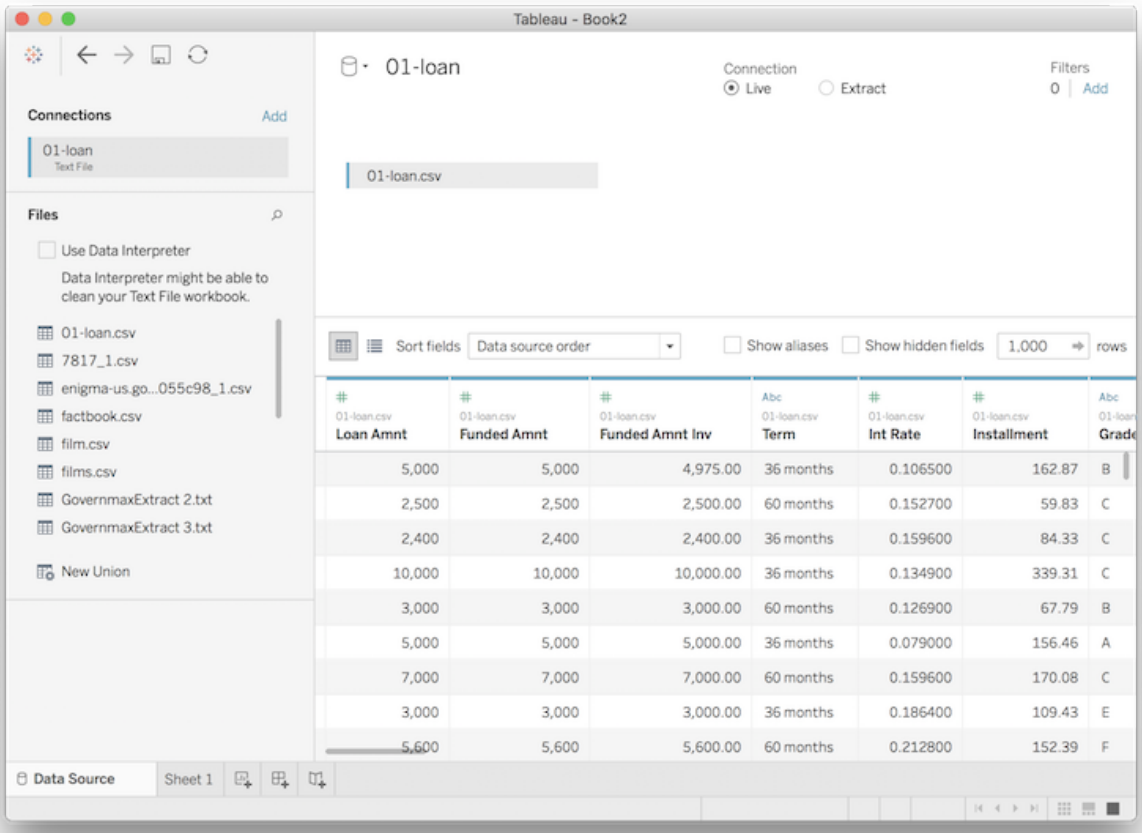
As indicated earlier, colors can be used to convey a mood or emotion. Warm colors - red, yellow, and orange - are often used to convey a sense of passion, happiness, enthusiasm, or energy. Cool colors - green, blue, and purple - are used to convey a sense of calm or relaxation. Neutral colors - black, white, gray, cream, and brown - are often used as backgrounds or for supporting information; these colors are affected by the colors that surround them.

With these design and color principles, we can see some reasons why the first chart in the first figure is less visually appealing than the second chart. The principle of proximity is not correctly applied; spacing should be added to set the years apart. The colors chosen do not effectively distinguish between the four regions; four distinct colors/hues should be chosen.

Types of Visualizations

Next, we'll explore different types of visualizations. For these visualizations, we'll work with the Lending Club (https://www.lendingclub.com/info/download-data.action) load data available at ./data/01-loan.csv (./data/01-loan.csv).

To open the data in Tableau, we can choose to connect to a text file. This will allow us to browse to the CSV and open it.



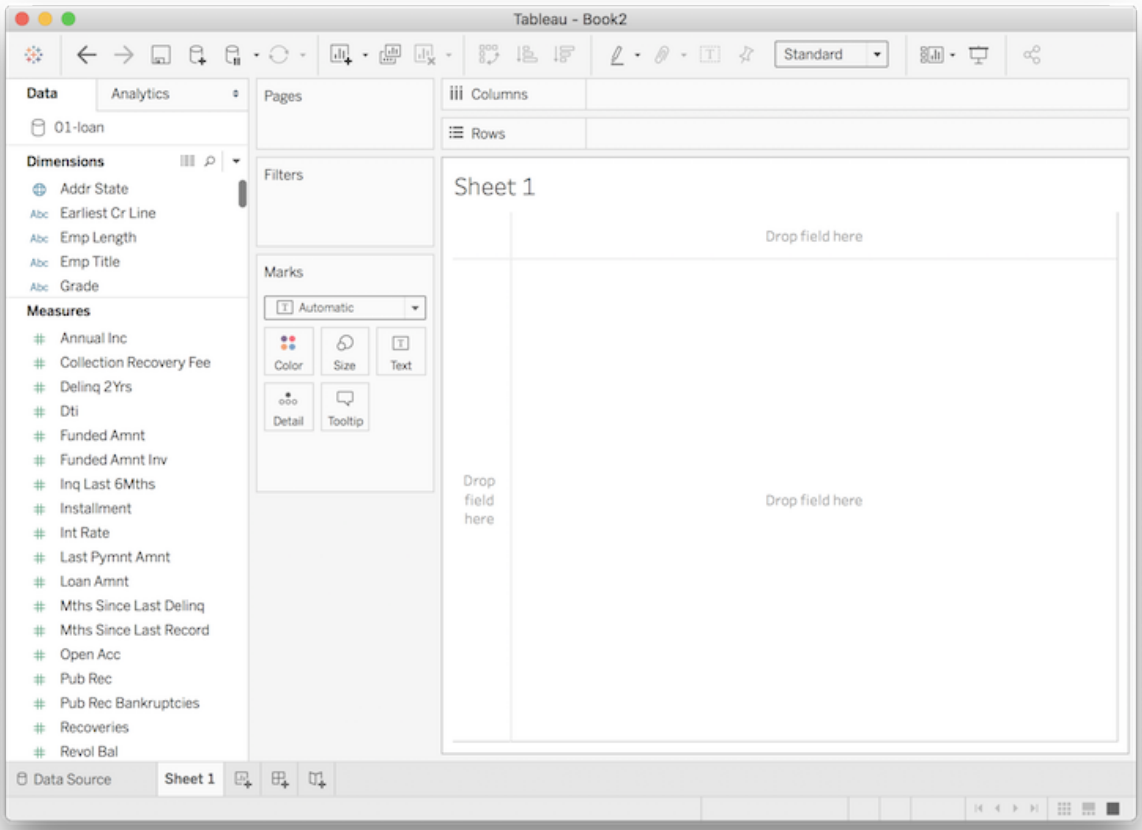
Loading Data in Tableau

Here we can see each of the columns in the source data as well as the one thousand rows. For some data sources, Tableau will determine the type of data based on the source; for other sources, Tableau will try to determine the type on its own. While we have the option of manually specifying the data type when loading the data, we'll rely on Tableau's detected values for now. If we had connected to a database with multiple tables, Tableau would present us with a list of tables and allow us to join them as needed.

Near the bottom of the screen is a collection of tabs. The first and currently selected tab is the "Data Source" tab which displays source information. Next to this tab is an existing sheet named "Sheet 1"; sheets represent individual visualizations in Tableau. The next three tabs allow us to create additional sheets, dashboards, and stories. In this unit, we'll create additional sheets. Later, we'll look at dashboards. To begin, select the "Sheet 1" tab.

Before creating any plots, notice that fields from the data are separated into dimensions and measures. Measures represent quantitative data representing a measured value; we can perform calculations/aggregations on measures. Dimensions give context or provide addition details about facts; dimensions are often qualitative data an can be use to group facts for aggregation.

Near the top of the Tableau interface we have section for columns and rows; dragging fields to these sections allows us to choose what data is used for the horizontal and vertical axes. Below the column and row area is the area in which plots will appear. To the left of this are areas where we can specify filters and marks; the marks sections is used to adjust some of the properties of the visualization.



Loading Empty Sheet in Tableau

As noted earlier, we'll create plots in both Tableau and this notebook. First, we indicate that plots will appear in the notebook itself.

In [32]:
```
%matplotlib inline
```

To load the data in the notebook, we'll use pandas.

In [33]:
```
import pandas as pd
loan_data = pd.read_csv("./data/01-loan.csv")
```

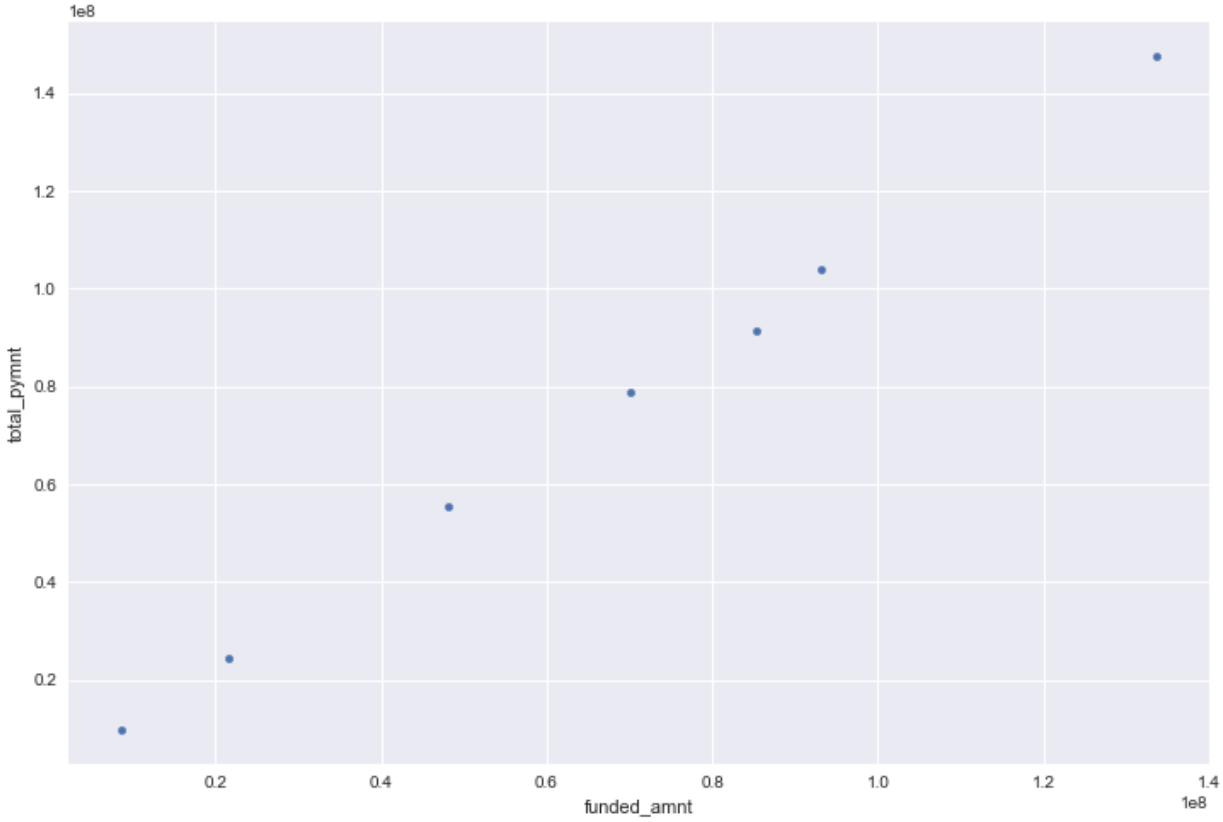
Scatter

The first type of plot we'll work with is one we've seen before - the scatter plot. We've used scatter plots to determine if a relationship exists between data but they can also be used to highlight that relationship.

For our first plot, we'll look at a scatter plot of amount funded versus the total payment, aggregated by grade.

In [34]:
```
import seaborn as sns

sns.set(rc={'figure.figsize':(12,8), "lines.markeredgewidth": 0.5 })

loan_data.groupby("grade").sum().plot.scatter(x="funded_amnt", y="total_pymr
```

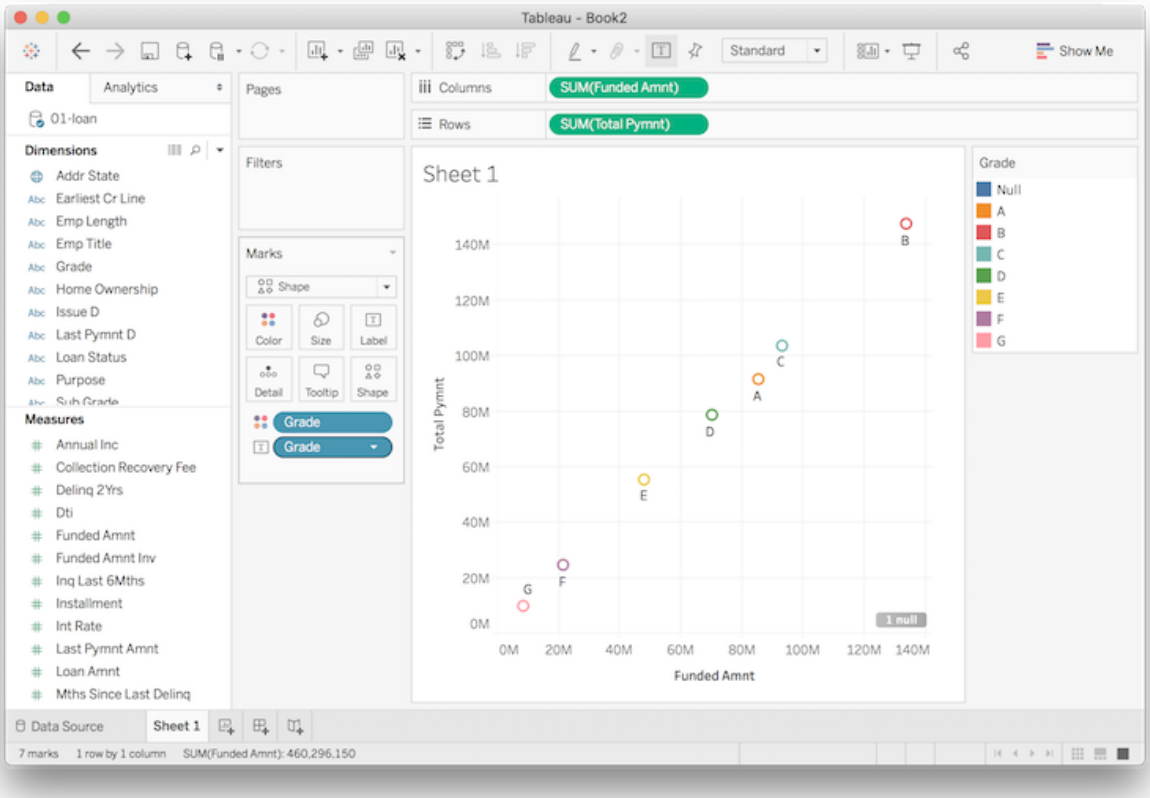
Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10b924780>
```



To create a similar plot in Tableau, we can drag `Funded Amnt` measure to the *Columns* area and `Total Pymnt` measure to the *Rows* area. Tableau will automatically aggregate the data by summing the values and we will see a single marker representing all data. To separate these by letter grade, drag the `Grade` dimension to the *Marks* area; if the icon to the left is not one with four dots, left the click the icon and select *Detail*. Drag `Grade` to the *Marks* area again and change the icon to *Label*; this will add the label for each marker.

Changing the first item in the *Marks* section from detail to *Shape* or *Color* allows us to indicate that each grade should be represented by a different maker shape or color, respectively. We can drag `Grade` to the *Marks* area again if we want to set both shape and color. To configure these options, click the corresponding box at the top of the *Marks* box.

Below is the scatter plot with markers of different colors; notice that Tableau chose different hues for the colors as `Grade` represents qualitative data of differing categories.



Scatter Plot in Tableau

We can name this sheet by right-clicking on the tab, selecting *Rename Sheet*, and entering a new value.

Line

A line plot can used to show changes in one continuous variable with respect to another continuous variable. Often the independent variable in a line plot corresponds to time.

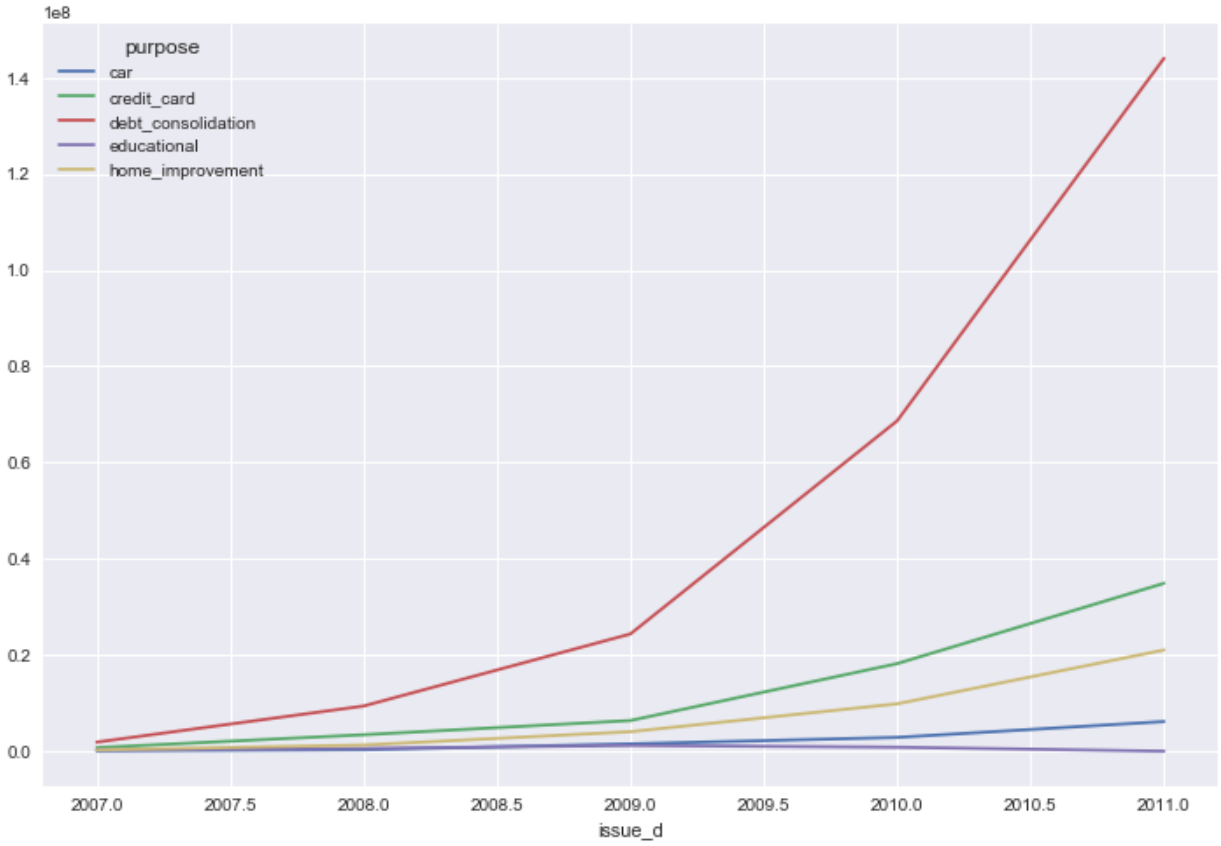
Let's look at loan amounts over time by purpose. To plot this in Python with pandas will require the use of pivot tables. First, we need to convert the data type of the `issue_d` column from a string to a datetime. To do this, we can use the pandas *to_datetime()* (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html) function along with the format specified using strftime (http://strftime.org/) notation. Because dates are of the form "Dec-11", we specify the format with `%b-%y` .

The pivot table will rely on the year value of `issue_d` for the index; to access the year, we use the *dt.year* property for the column. Columns of interest for the pivot table will consist of the unique values of the `purpose` column. We're only interested in the `loan_amnt` column and none of the others so we'll specify a value for the `values` keyword argument. We can specify the aggregation function and the value to replace missing values.

We can calculate the sum of loan amount for each purpose using the pivot table's *sum* method. To find the top five purposes by sum, we sort by values then collect the first five indexes. We can then use the these with the pivot table to plot the corresponding lines.

In [35]:
```
loan_data.issue_d = pd.to_datetime(loan_data.issue_d, format="%b-%y")

pivot = loan_data.pivot_table(index=[loan_data.issue_d.dt.year],
                              columns=["purpose"],
                              values=["loan_amnt"],
                              aggfunc = pd.np.sum,
                              fill_value = 0)
pivot_sum = pivot.sum()
pivot_sum.sort_values(ascending=False)
top_5 = pivot_sum['loan_amnt'].index[:5]

pivot['loan_amnt'][top_5].plot(kind='line')
```

Out[35]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10af7c4a8>
```

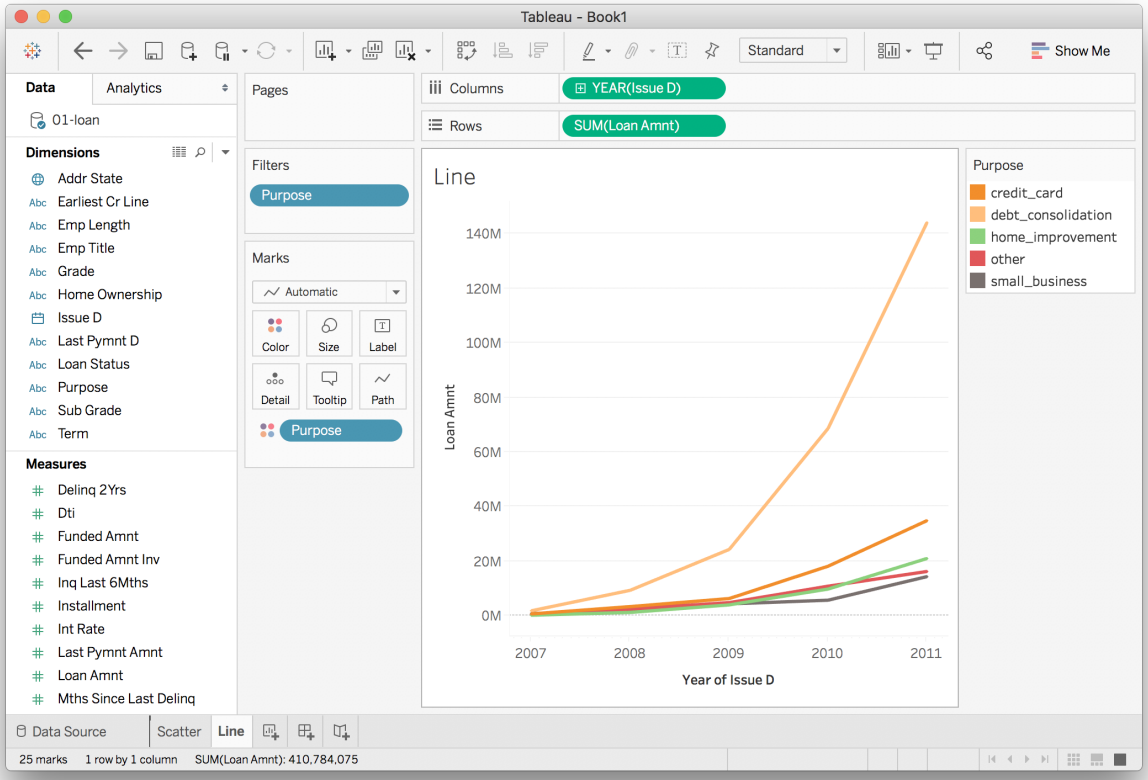


To create another chart, click the *New Worksheet* tab near the bottom of the Tableau window. As before, we first need to modify the data type of the `Issue D` field. To change this to a date type, right-click on the `Issue D` field, select *Change Data Type*, and select *Date*. Next, drag `Issue D` to the *Columns* area. For the vertical axis, we can drag `Loan Amt` to the *Rows* area. If Tableau does not automatically choose sum as the aggregation, change it. To separate the plot by purpose, drag `Purpose` to the *Marks* area and select the color option. If the plot is not a line chart, click the *Show Me* button in the upper right corner of the Tableau window and select one of the line options.

To display only the top 5 purposes, right click `Purpose` in the *Marks* area and select *Filter*. In the *Filter* window, select the *Top* tab, choose *By Field* and select appropriate values for "Top 5 by Loan Amnt Sum".

Notice the `Issue D` field in the *Columns* area displays *Year*. Right-clicking the field or opening the drop-down menu allows us to change the time resolution from year down to the day - this data only has monthly resolution though.

We can rename the sheet to "Line".



Line Plot in Tableau

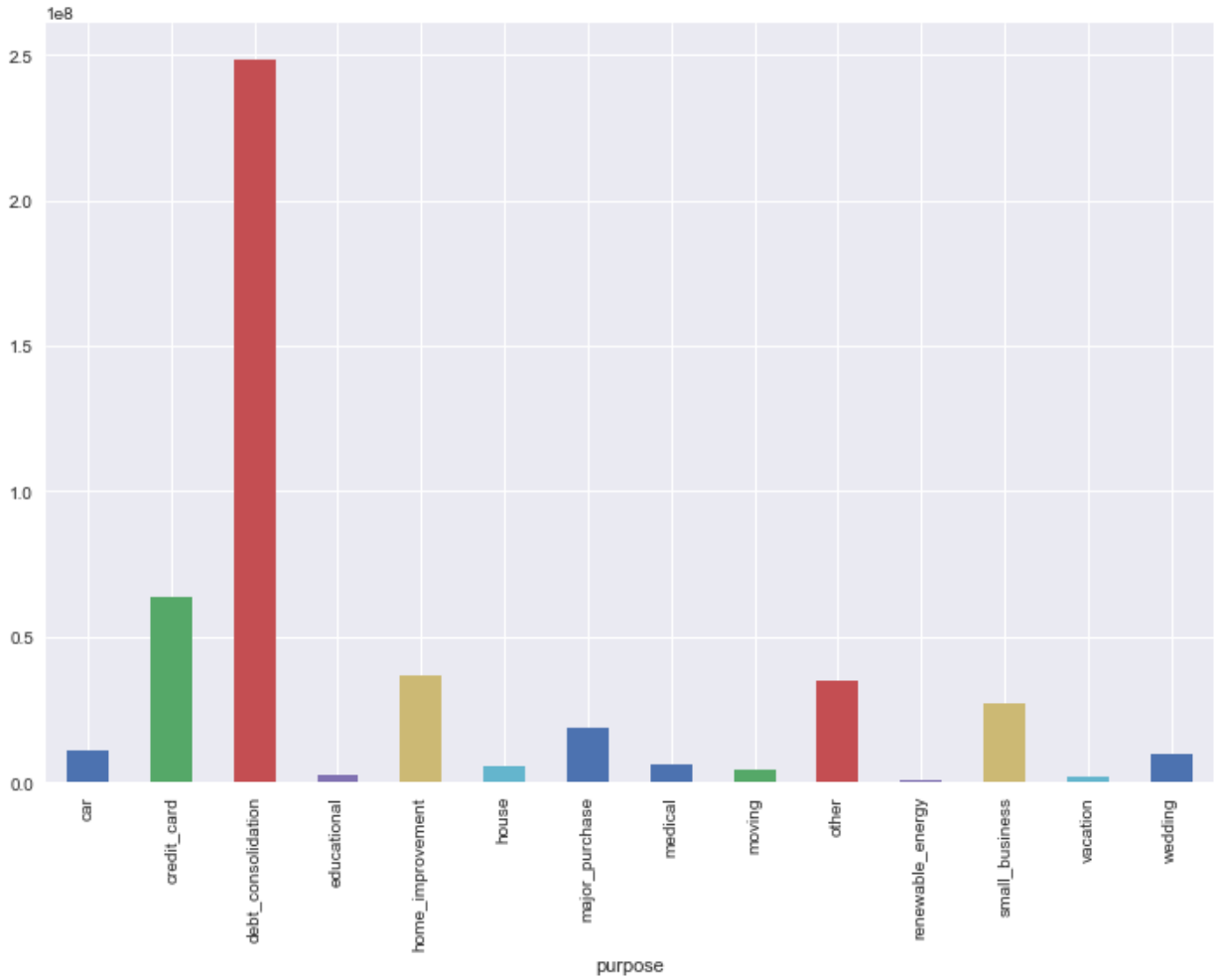
Bar

Bar charts are typically used to convey data across categories, that is categorical data with other data. Let's continue looking at loan amount by purpose but instead consider the total loan amount. Here we have a a categorical variable, `purpose`, and a numeric variable, `loan_amnt`.

In Python, we can use the pivot table we already constructed or use the *groupby* method.

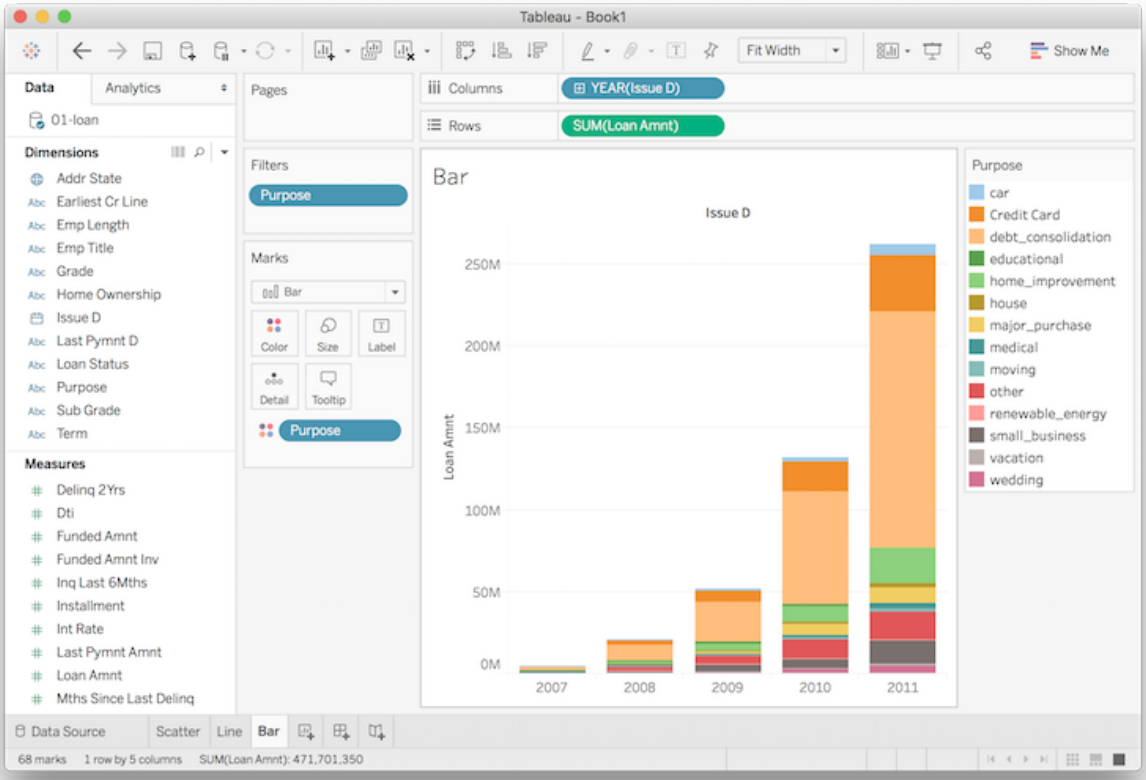
```
In [36]: loan_data.groupby("purpose")['loan_amnt'].sum().plot.bar()
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x10b924a20>
```



To create a similar chart in Tableau, we start by creating a new worksheet. To create a vertical bar chart, drag the `Purpose` field to the *Columns* area and `Loan Amnt` to the *Rows* area; Tableau should automatically choose the sum aggregation. You might notice that one of the columns corresponds to "Null" or missing values. To remove the null values, click the word "Null" and click "Exclude"; this creates the corresponding filter. To clean up the column labels, right-click on one of values and select "Alias...".

We can also easily create stacked bar charts in Tableau where each bar is subdivided. To do this, move `Purpose` from the *Columns* area to *Marks* and drag the *Issue D* field to the *Columns* area. This is very similar to what we did for the line chart; if set of lines is dispaly, click the *Show Me* button and select stacked bars. You might also have to choose color as the marks option for the `Purpose` field.



Bar Chart in Tableau

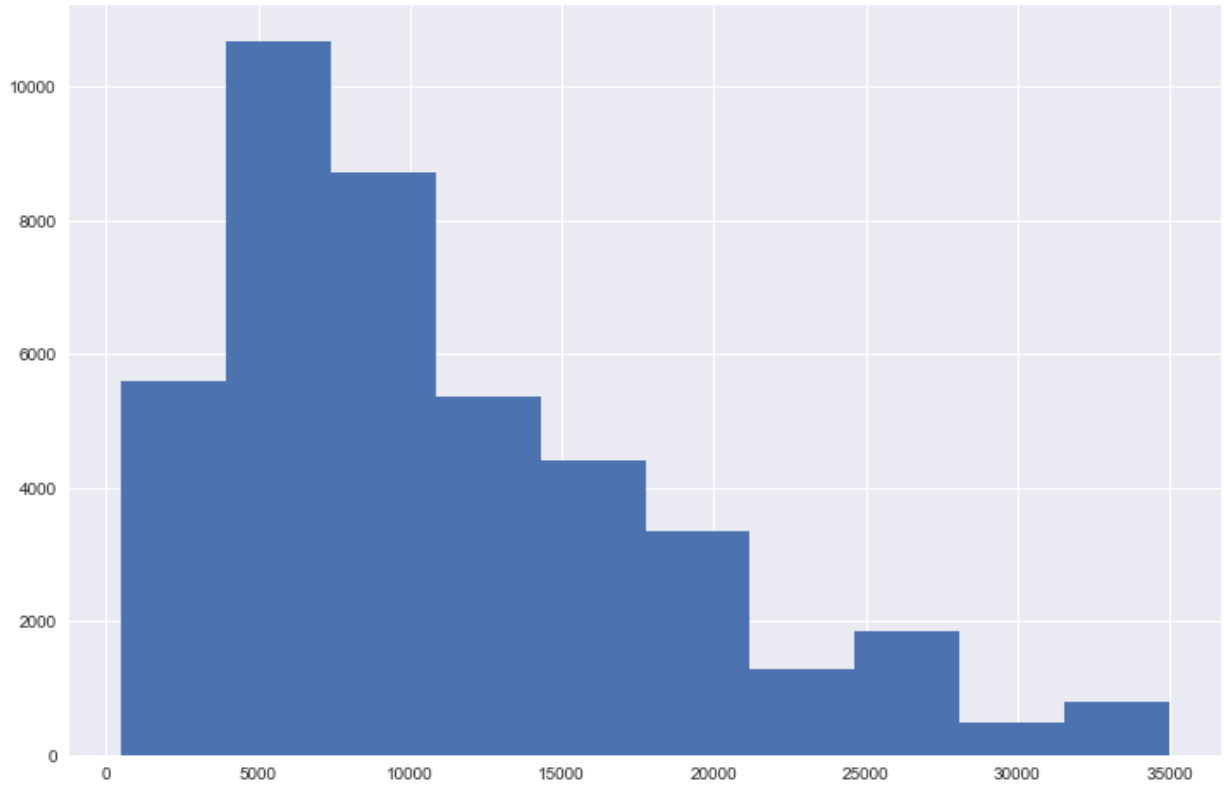
Histogram

We've worked with histograms throughout our exploration of datasets. Histograms are used to visualize the distribution of data for some variable using bars where each bar represents a bin to which data can be assigned and the length or height of each bar represents the number of elements in that bin.

To create a histogram in Python, we can use the *hist()* method associated with a column in a DataFrame.

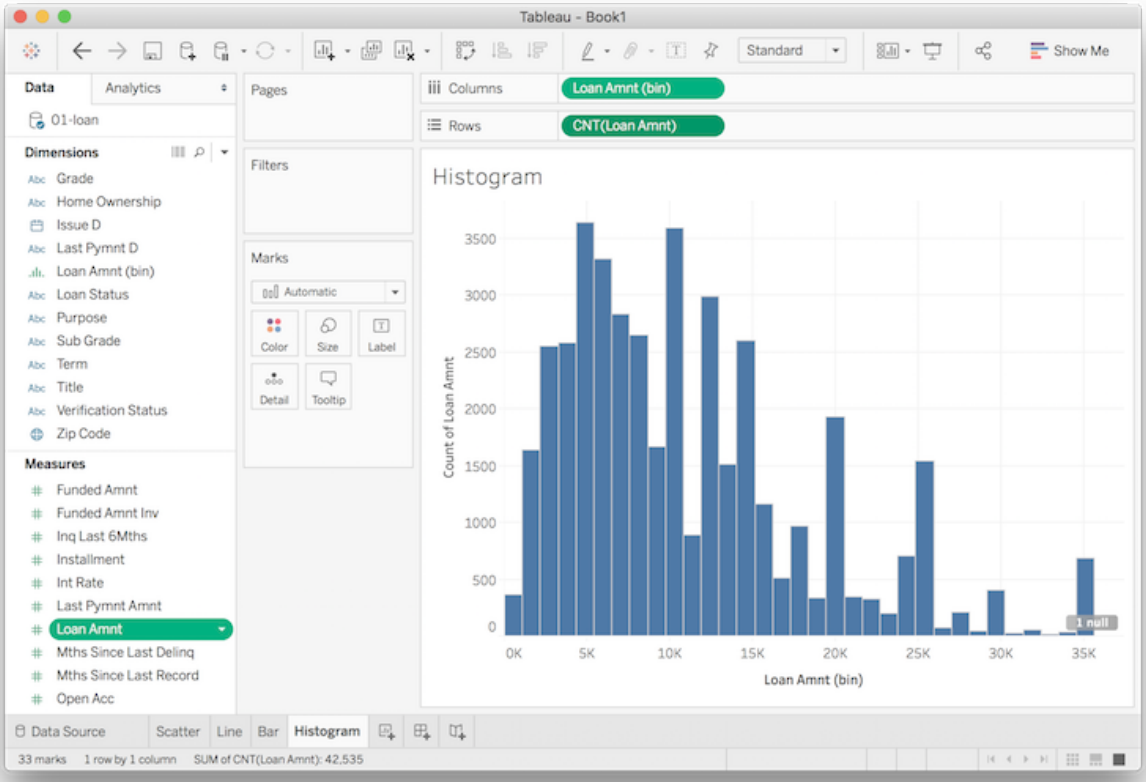
```
In [37]: loan_data.loan_amnt.hist()
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x109304198>
```



To create a histogram in Tableau, we start by dragging the field of interest, `Loan Amnt` in this case, to the *Columns* area. Next, click the *Show Me* button and select histogram. Tableau will automatically replace the field in the *Columns* area with one representing bins and the field in *Rows* with one representing the count for each bin.

Tableau automatically chooses a bin size. To manually specify a size, right click the source field, `Loan Amnt` , and select "Create" and "Bins...". A new field is created that can used in place of the existing field in the *Rows* area.



Histogram in Tableau

Box Plot

Like histograms, we've also used box plots in previous units to help understand data. As we've seen before, box plots can be used to see the range of values and quartile information.

As an example, let's consider the interest rate on various loans. Rather than simply examining the box plot for all interest rates, let's create a box plot for interest rates for each grade assigned to lenders. In Python, we can use the the Seaborn *boxplot()* method.

In the dataset, interest rates are stored as strings representing percentages with the percent sign; we'll have to convert this to numeric data to create a box plot. We'll use the column's *apply()* method to convert the values. The function we'll use is defined and tested with a value below.

In [38]:
```
def to_float(percent):
    if isinstance(percent, str):
        return float(percent.replace("%", "")) / 100
    else:
        return percent

to_float("7.75%")
```

Out[38]:
```
0.0775
```

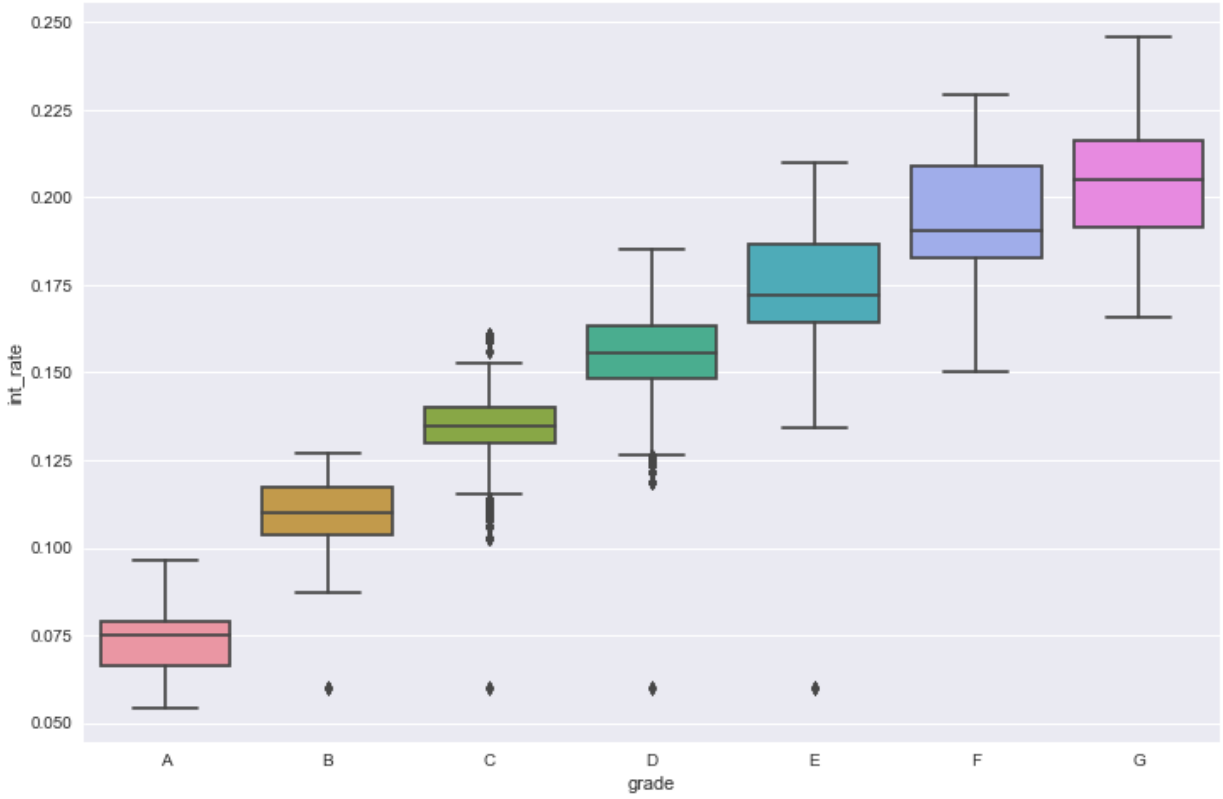
We can now apply the function to the interest rate values.

In [39]:
```
loan_data.int_rate = loan_data.int_rate.apply(to_float)
```

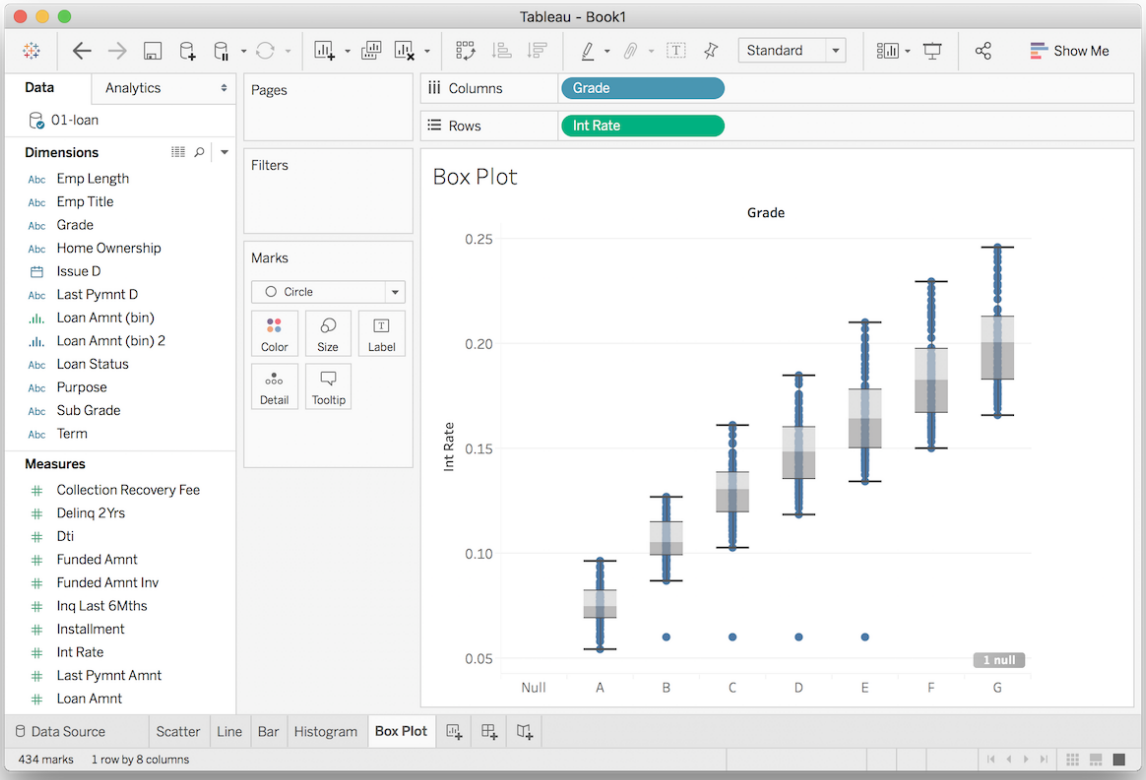
We can now create the box plot.

In [40]:
```
sns.boxplot(x='grade', y='int_rate',
            data=loan_data.sort_values(by='grade'))
```

Out[40]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10a053fd0>
```



To create a similar plot in Tableau, drag `Grade` to the *Columns* area and `Int Rate` to the *Rows* area. By default, Tableau tries to aggregate the data; to stop this, right click the `Int Rate` field in the *Rows* area and select dimension. If a series of box plots are not displayed, click the *Show Me* button and select the box plot option.



Box Plots in Tableau

Heat Map

Heat maps can be used to show the distribution of data through color. Heat maps are one using two-dimension space (like a monitor or a sheet of pater) to present higher-dimensional data; color give us access to an additional dimension.

We looked at a heat map in a previous unit. One way of creating a heat map from a DataFrame is through the use of a pivot table. Let's look at the distribution of loans by grade and purpose. First we create the appropriate pivot table.

In [41]:
```
heat_pivot = loan_data.pivot_table(index=['grade'],
                                   values=['loan_amnt'],
                                   columns=['purpose'],
                                   aggfunc=pd.np.sum)
heat_pivot
```

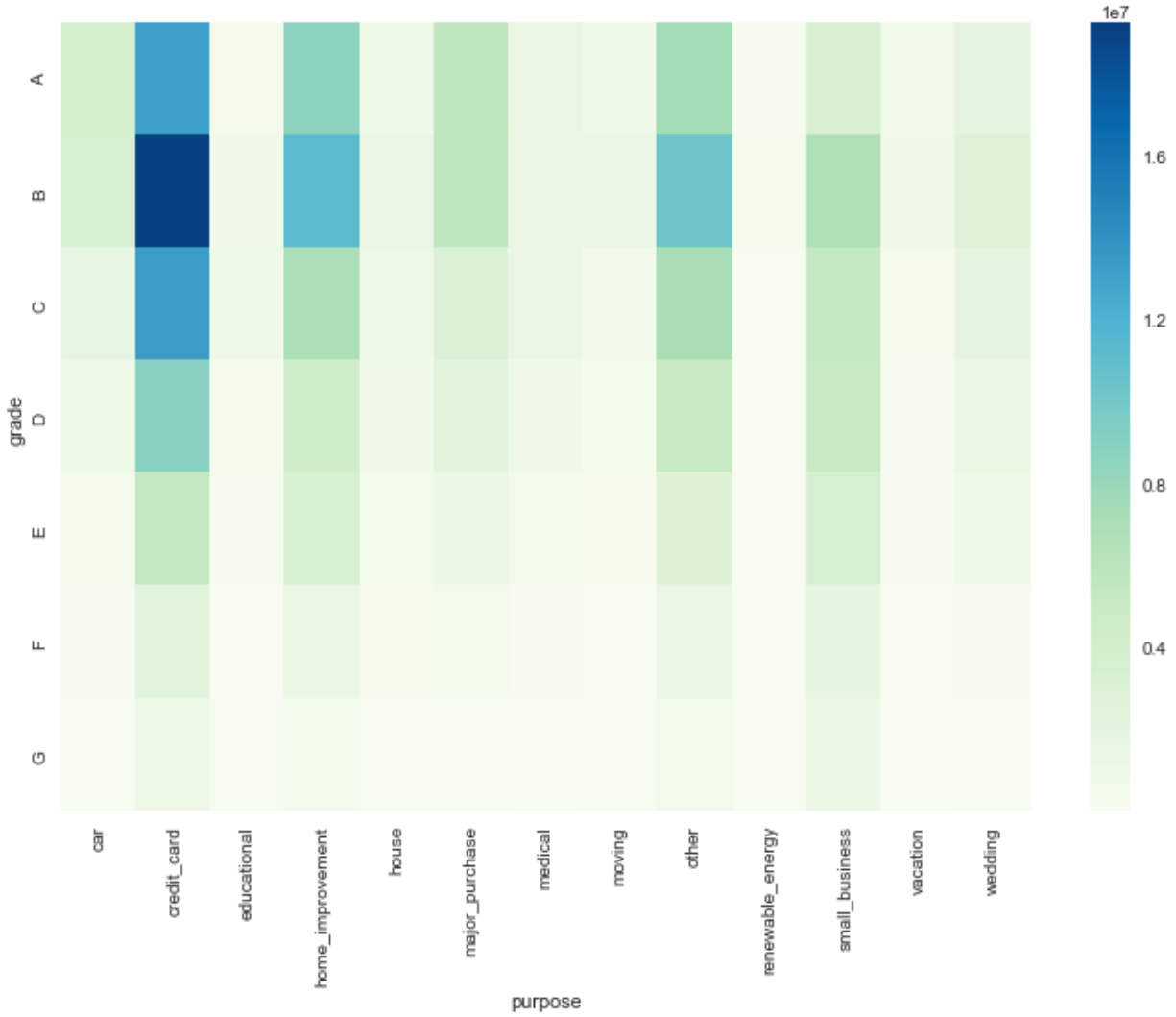
Out[41]:

	loan_amnt						
purpose	car	credit_card	debt_consolidation	educational	home_improvement	house	ma
grade							
A	3856825.0	13141825.0	38232150.0	523400.0	8635475.0	1008375.0	
B	3639925.0	19222500.0	70957275.0	768800.0	11244300.0	1531125.0	
C	1680925.0	13422225.0	50817850.0	849975.0	6953675.0	1037975.0	
D	1047200.0	8867575.0	40425150.0	310675.0	4311100.0	760775.0	
E	503350.0	5456975.0	29799075.0	213100.0	3489400.0	562700.0	
F	166700.0	2462925.0	13381600.0	101050.0	1249700.0	316225.0	
G	23000.0	1040975.0	4763750.0	29400.0	566100.0	143450.0	

With the pivot table, we can now create a heat map. Because `debt_consolidation` contains values with a greater magnitude that the other columns, we drop remove it from the heat map for effect. Recall, that we can specify the color map using the `cmap` keyword argument with a Matplotlib color map name (https://matplotlib.org/users/colormaps.html).

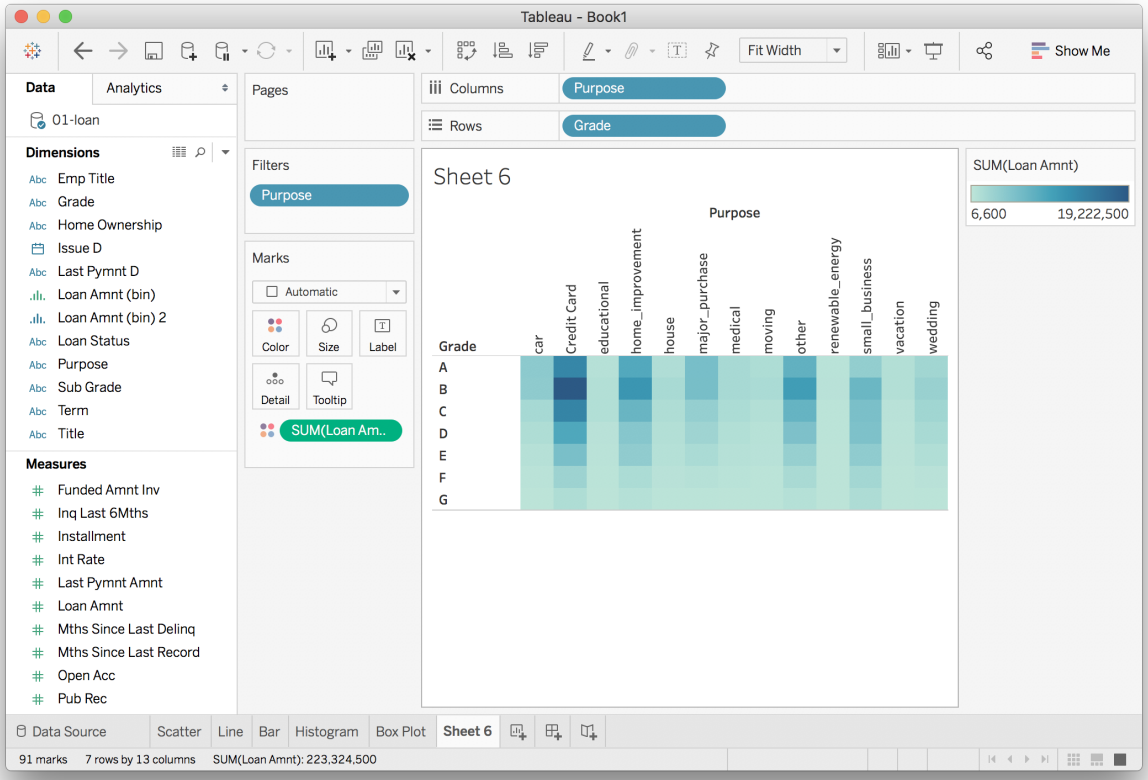
In [42]:
```
heat_pivot_no_debt = heat_pivot['loan_amnt'].drop(['debt_consolidation'], 1)
sns.heatmap(heat_pivot_no_debt, cmap="GnBu")
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10b829198>
```



To create a heat map in Tableau, we can start by specifying the row and column values; drag `Purpose` and `Grade` to the *Columns* and *Rows* areas, respectively. Next, drag `Loan Amnt` to the *Marks* area and choose the color option. If the plot doesn't automatically change to a heat map, select heat maps from the *Show Me* menu.

We can remove null values by selecting the "Null" column and choosing "Exclude". Similarly, we can remove the "debt consolidation" column by clicking on the label for the third column and selecting "Exclude". To rotate the column labels, right-click one of the labels and select "Rotate".



Heat Map in Tableau

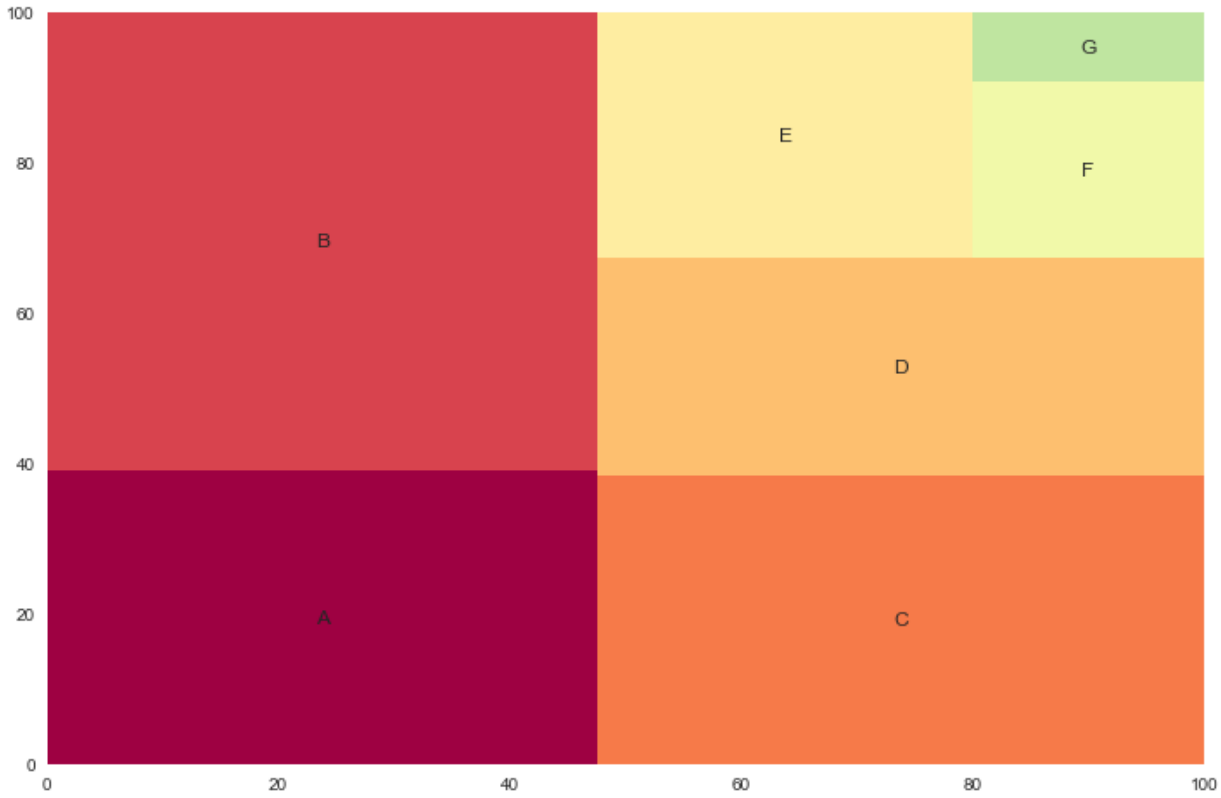
Tree Map

Tree maps make use of a rectangular areas (and often color) to convey information - larger area correspond to larger values. We should exercise caution when using tree maps. Because people tend to not be able to accurately compare similar areas and volumes, tree maps should only be used when differences in area are obvious.

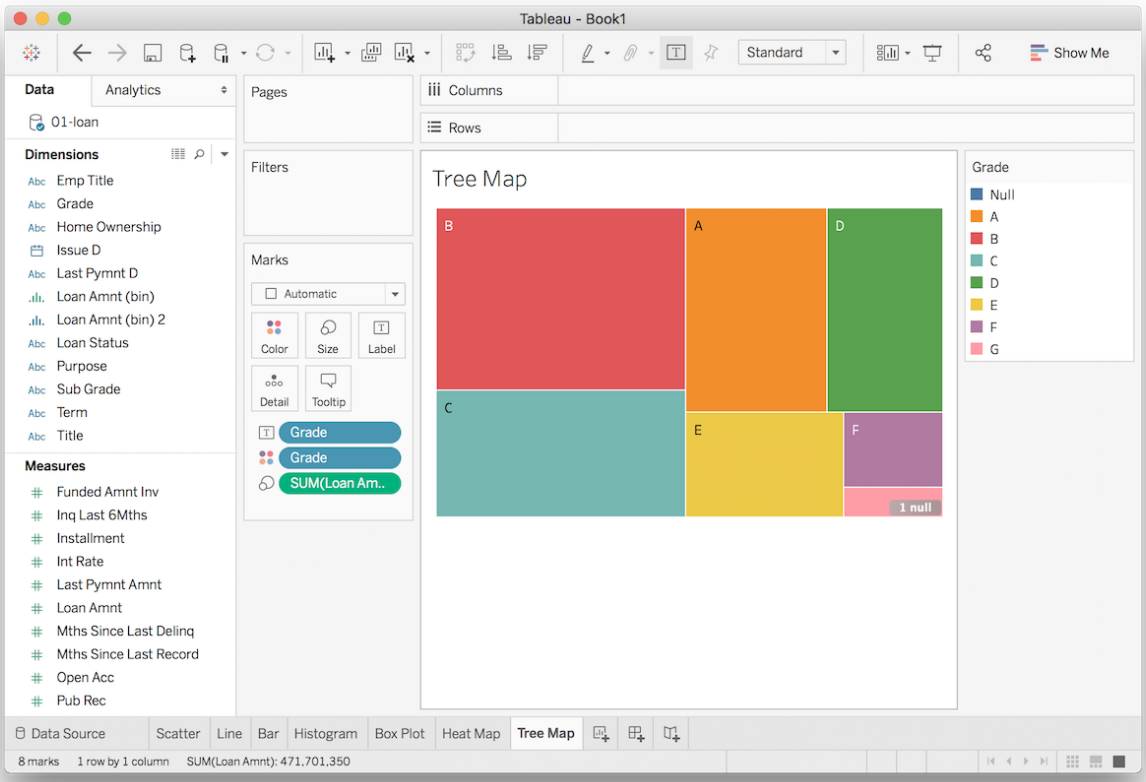
To create a tree map we will use the *squarify* library. We aggregate the loan data by `grade`, select the `loan_amnt` column, and calculate the sum. When creating a tree map we can specify a collection of colors to be used; we can sample an existing color map.

In [43]:
```
import squarify

grades = loan_data.groupby("grade")['loan_amnt'].sum()

# get the green-blue colormap
cmap = sns.mpl.cm.get_cmap('Spectral')
# get 10 colors from the colormap
colors = cmap(pd.np.linspace(0,1,10))

squarify.plot(sizes=grades.values, label=grades.index, color=colors)
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x106d6edd8>
```



A tree map in Tableau relies only on data *Marks* data. Drag `Grade` to the *Marks* area twice and `Loan Amnt` to the *Marks* area once. Designate one of the `Grade` fields as providing label data and the other as providing color data. `Loan Amnt` should determine size.



Tree Map in Tableau

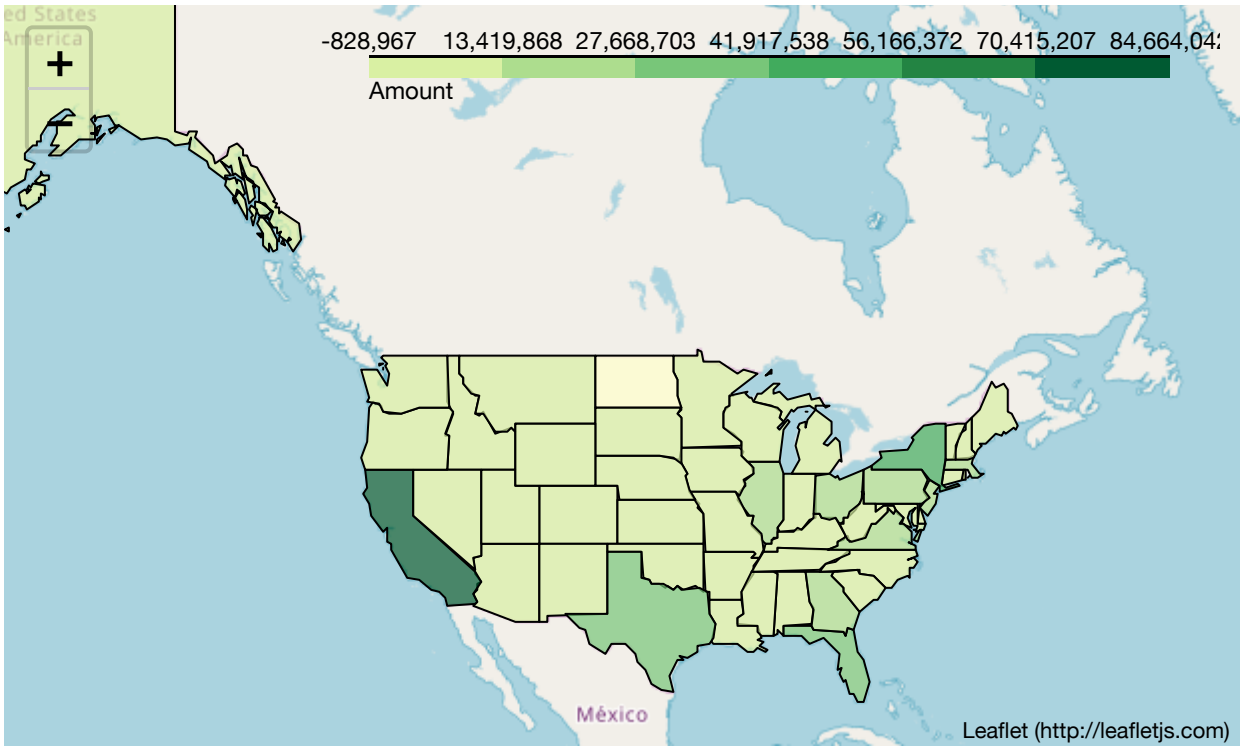
Geographic

The final type of plot we'll consider are those that rely on geographic data. As an example, we'll visualize loan amount by state. To create this in python, we'll make use of *folium*.

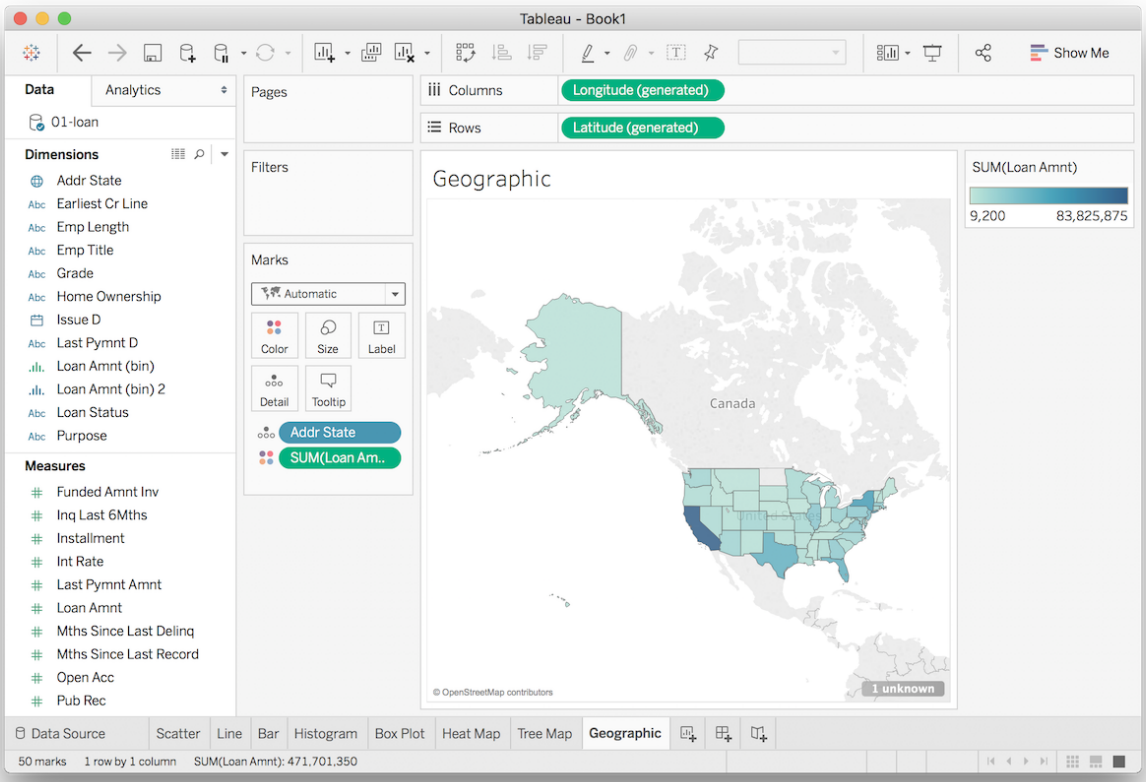
In [44]:

```
import folium

# aggregate by state and convert to DataFrame
states = loan_data.groupby("addr_state")['loan_amnt'].sum().to_frame()
states['state'] = states.index

# load state GeoJSON data
with open("./data/06-us-states.json") as infile:
    state_data = infile.read()

# create map with data
data_map = folium.Map(location=[48, -102], zoom_start=3)
data_map.choropleth(geo_data=state_data,
                    data=states,
                    columns=['state', 'loan_amnt'],
                    key_on='feature.id',
                    fill_color='YlGn',
                    fill_opacity=0.7,
                    legend_name='Amount')
data_map
```

Out[44]:



To create a similar plot in Tableau, drag `Addr State` and `Loan Amnt` to the *Marks* area. Select color for the type of data represented by `Loan Amnt` .



Map in Tableau

Next Steps

The visualizations we've created are static. Often there is a need to create visualizations that are routinely updated with current data or to create interactive visualizations in which users can drill-down into data. In the next unit, we'll look at dashboards to address these needs.

Resources and Further Reading

- Build Common Chart Types in Data Views (https://onlinehelp.tableau.com/current/pro/desktop/en-us/dataview_examples.html?tocpath=Design%20Views%20and%20Analyze%20Data%7CBuild%20Common%20Chart%20T
- The Encyclopedia of Human-Computer Interaction, 2nd Edition; Chapter 35 Data Visualization for Human Perception by Stephen Few (https://www.interaction-design.org/literature/book/the-encyclopedia-of-human-computer-interaction-2nd-ed/data-visualization-for-human-perception)

Exercise

Create plots similar to those discussed in this unit with another dataset that we've worked with previously.

In [ ]: