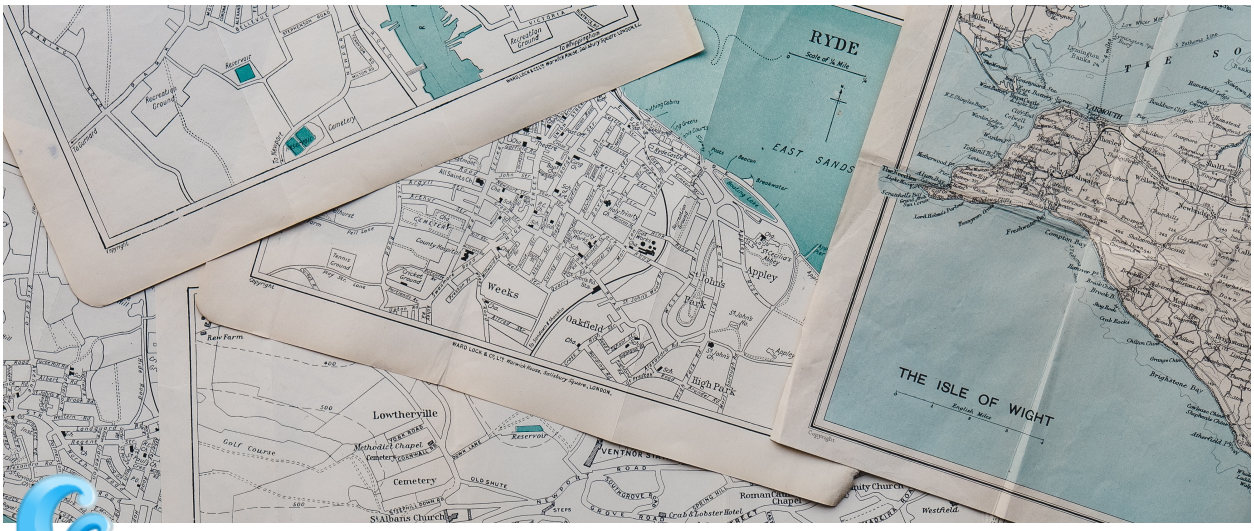


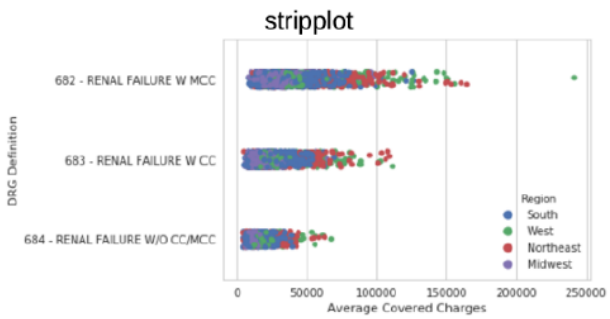

# Additional Plot Types

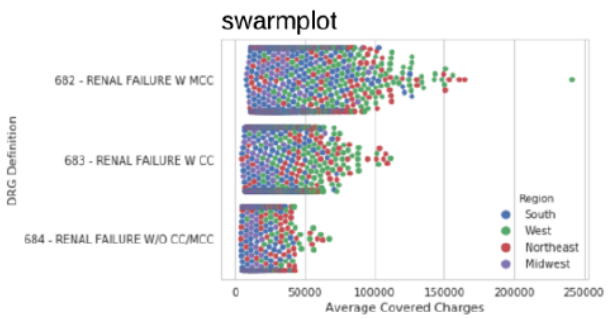
## Categorical Plot Types

### Categorical Data

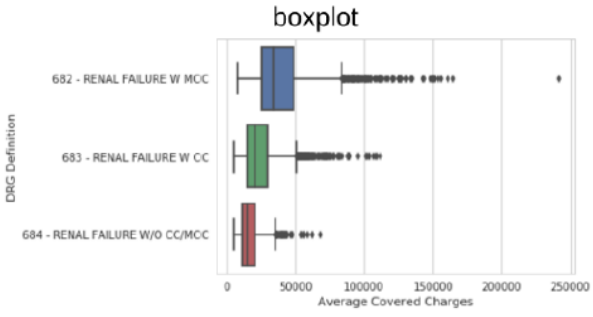
- Data which takes on a limited and fixed number of values
- Normally combined with numeric data Examples include:
  1)Geography (country, state, region)
  2)Gender
  3)Ethnicity
  4)Blood type
  5)Eye color

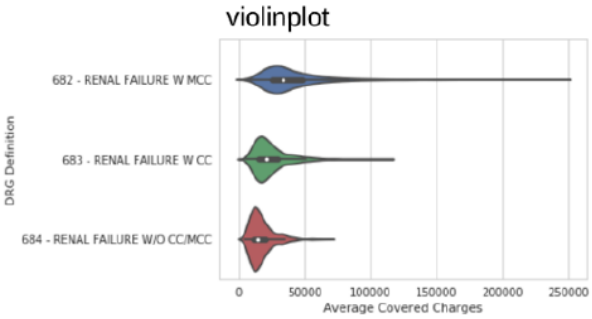
### Plot types - show each observation

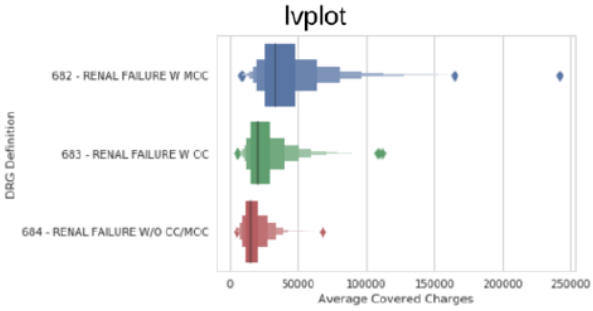


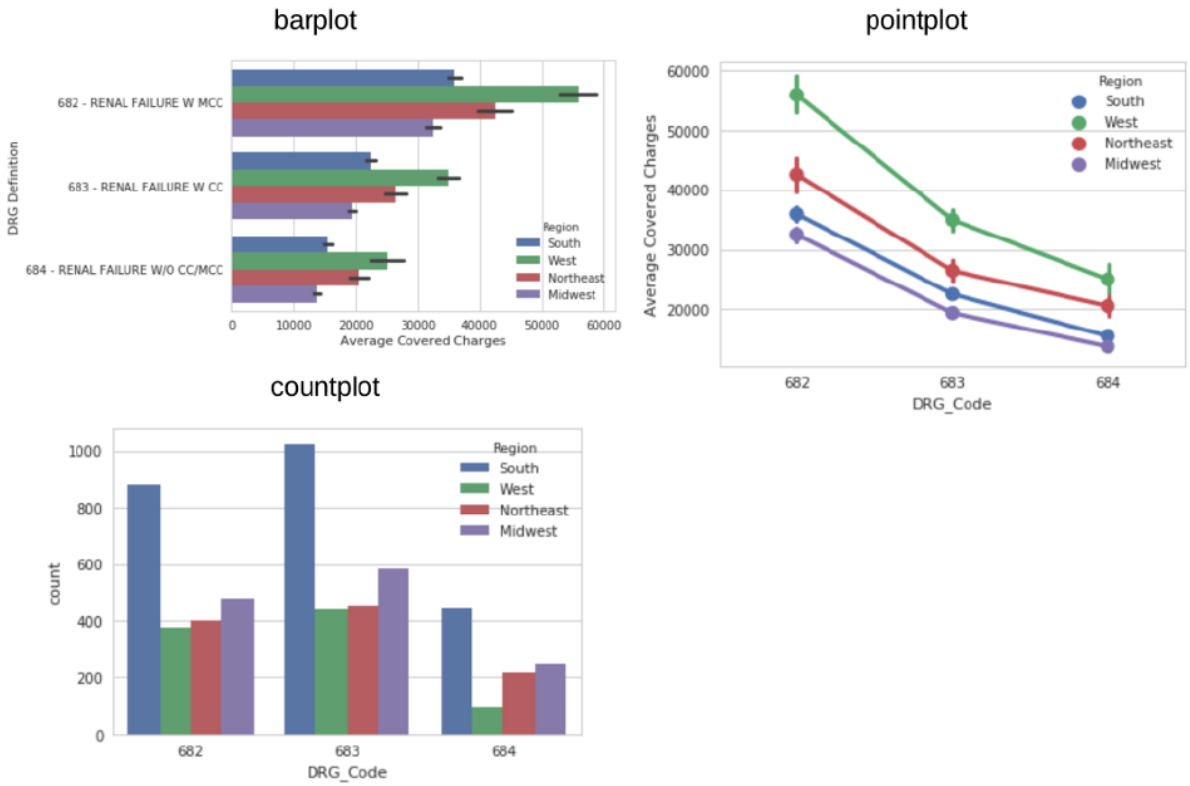

# Plot types - abstract representations



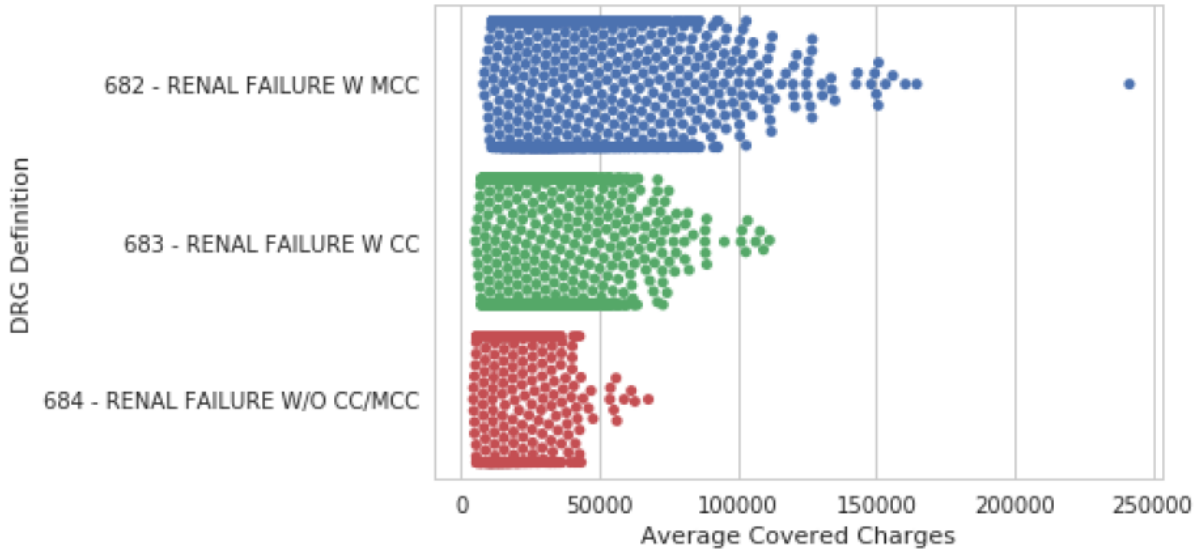




# Plot types - statistical estimates



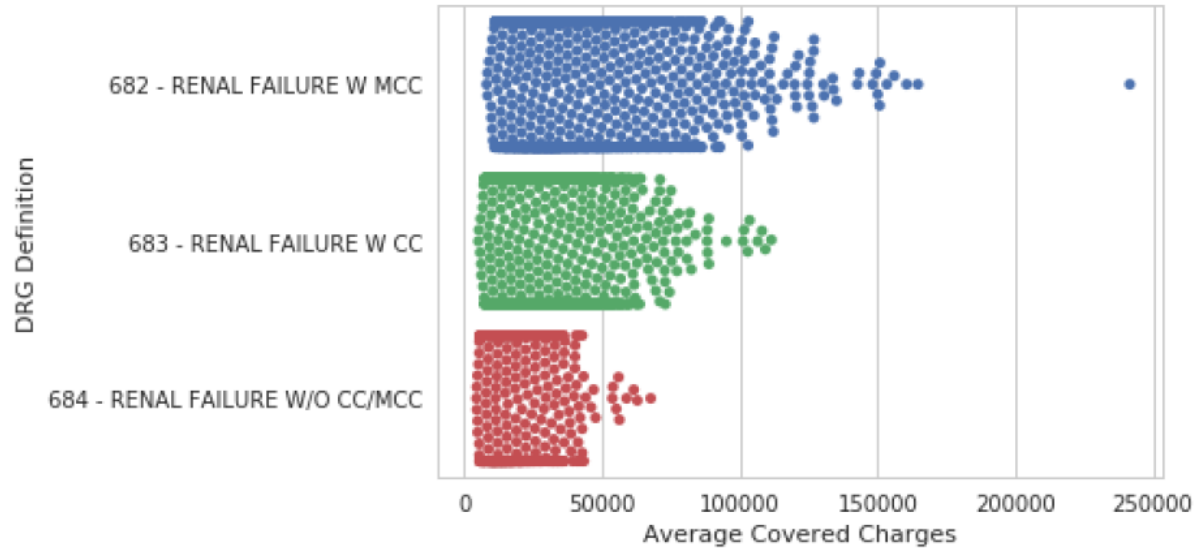

# Plots of each observation - stripplot

```
sns.stripplot(data=df, y="DRG Definition",x="Average Covered Charges",
              jitter=True)
```




## Plots of each observation - swarmplot

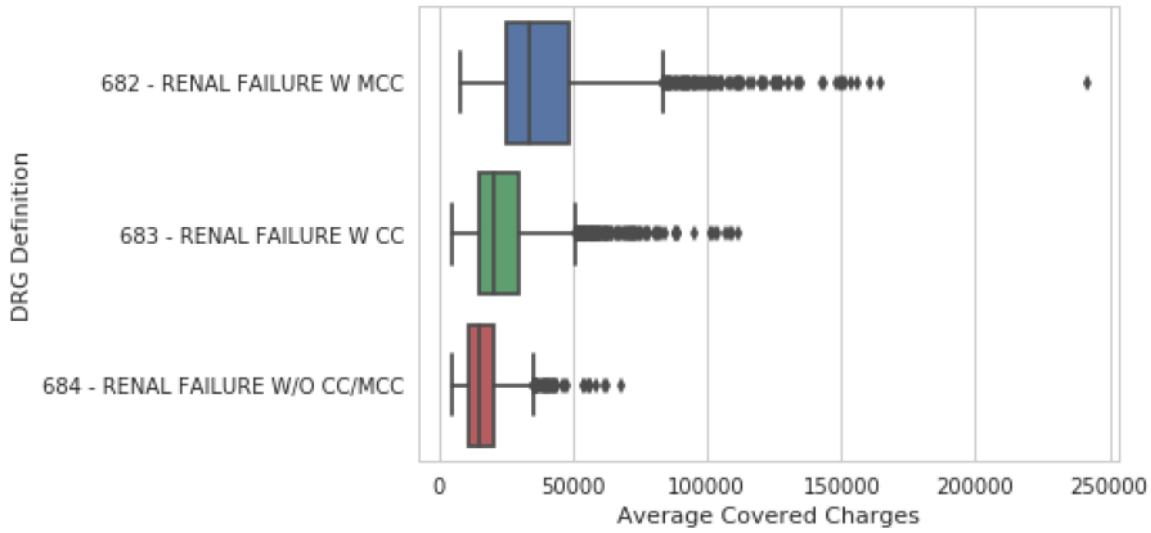
```
sns.swarmplot(data=df, y="DRG Definition",
              x="Average Covered Charges")
```




swarp plots aren’t well for large datsets.

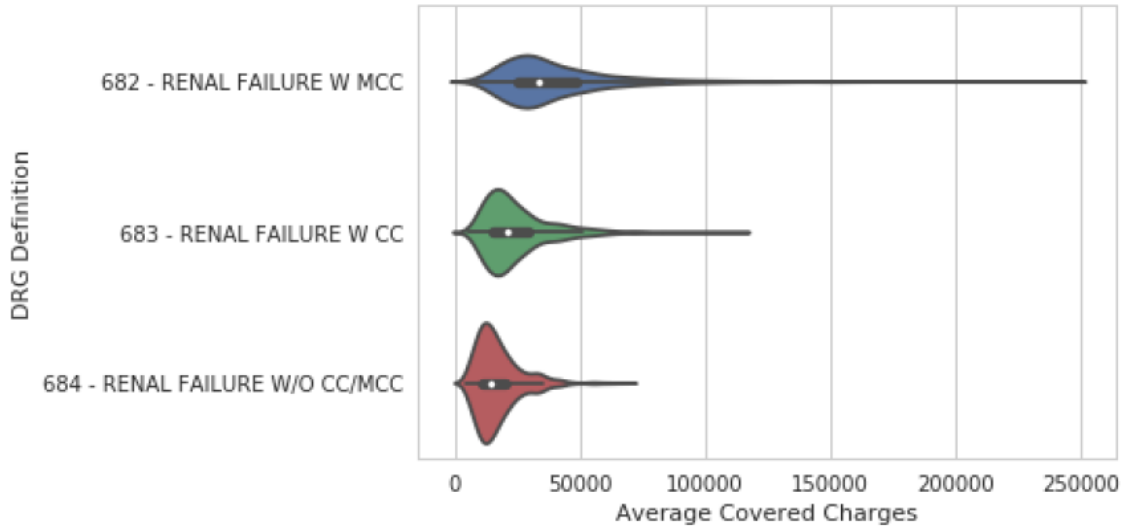
## Abstract representations - boxplot

```
sns.boxplot(data=df, y="DRG Definition",
            x="Average Covered Charges")
```



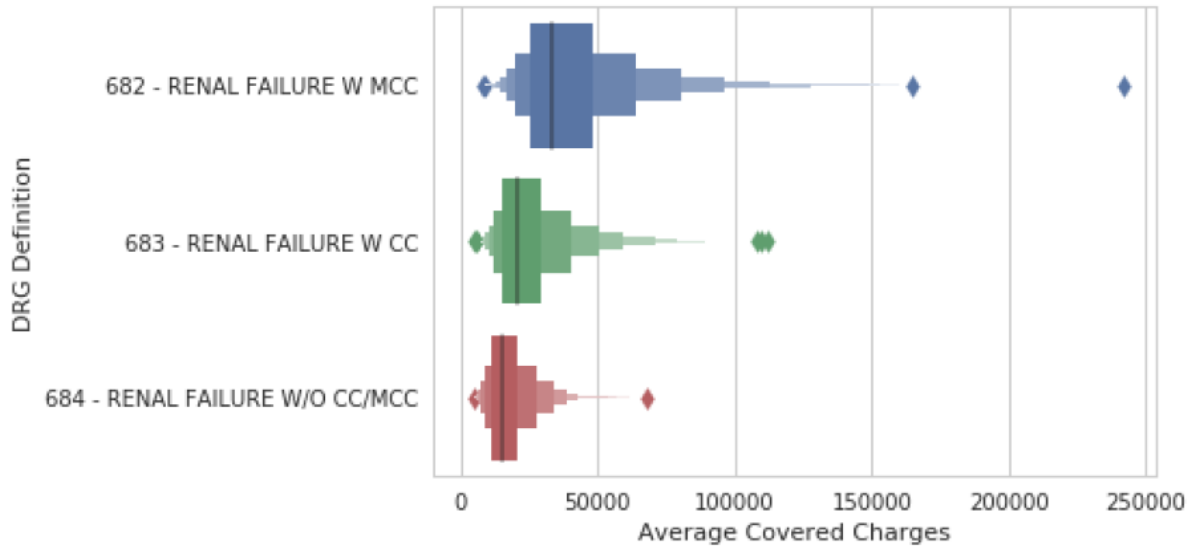

## Abstract representation - violinplot

```
sns.violinplot(data=df, y="DRG Definition",
               x="Average Covered Charges")
```



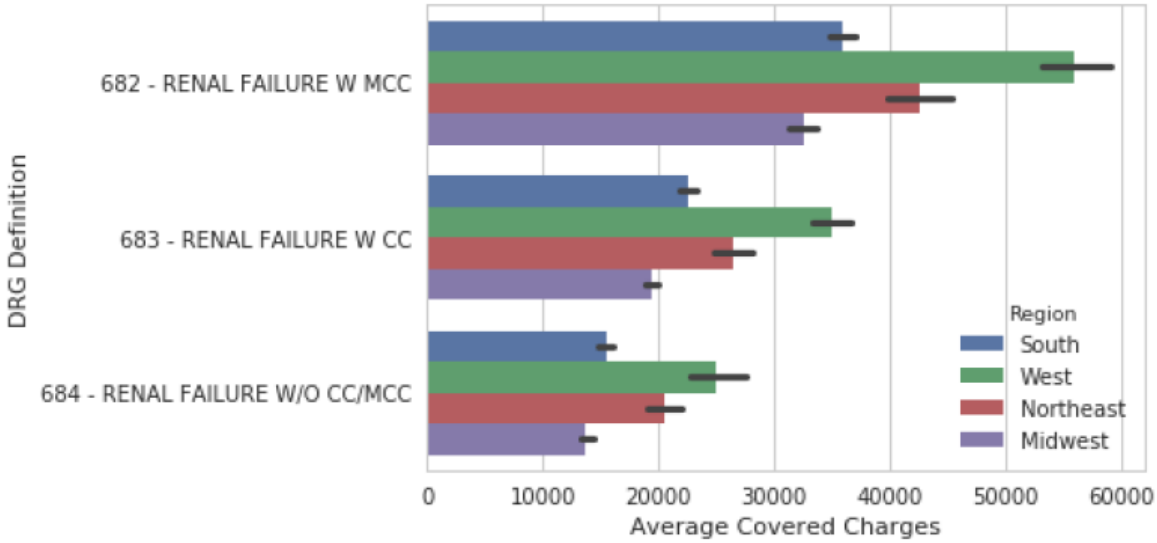

## Abstract representation - lvplot

```
sns.lvplot(data=df, y="DRG Definition",
           x="Average Covered Charges")
```



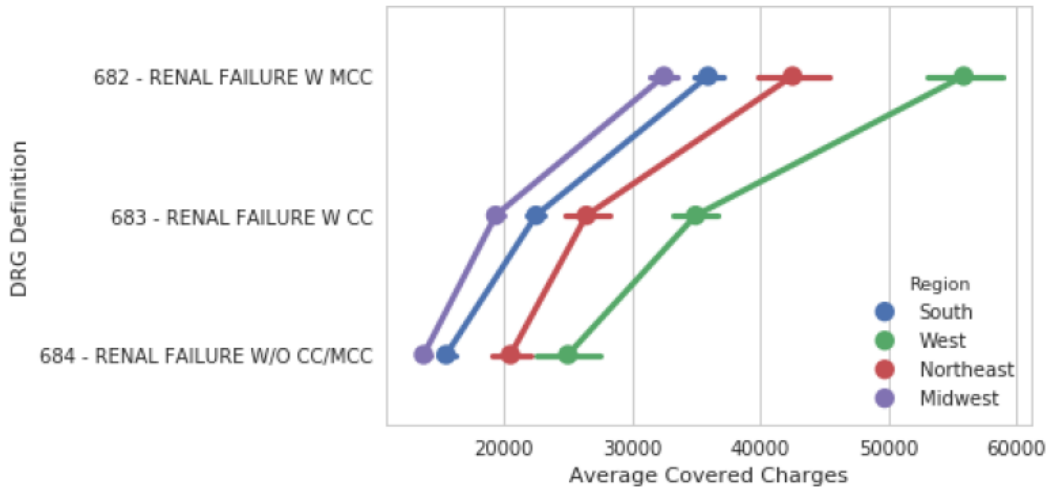

## Statistical estimates - barplot

```
sns.barplot(data=df, y="DRG Definition",
            x="Average Covered Charges",
            hue="Region")
```



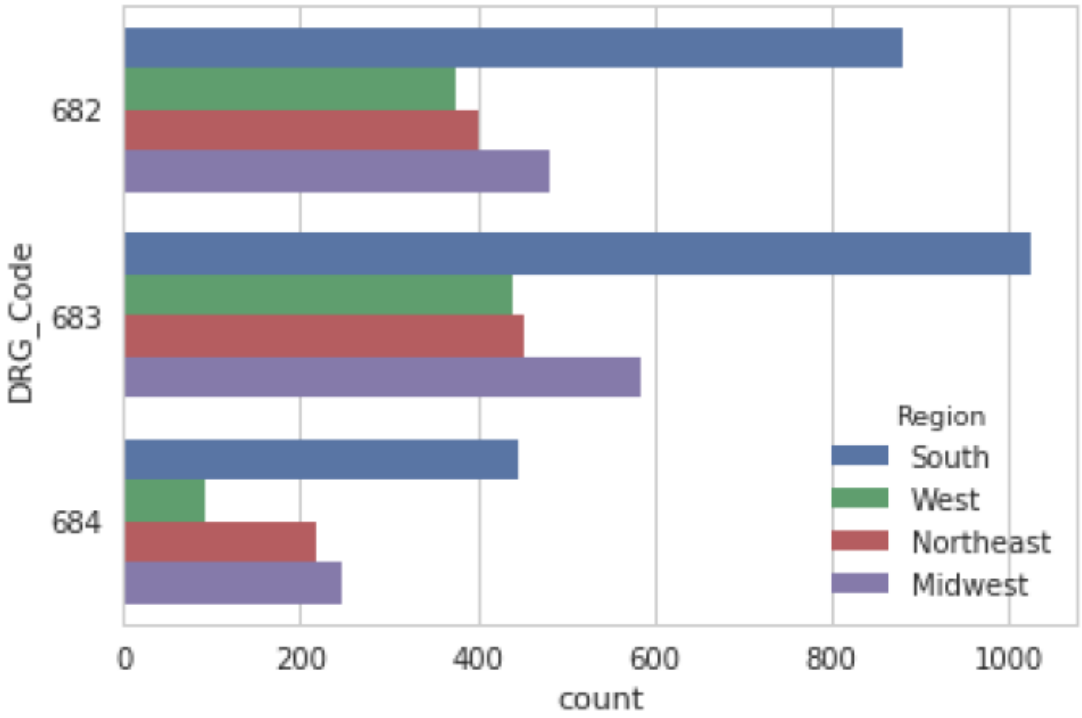

## Statistical estimates - pointplot

```
sns.pointplot(data=df, y="DRG Definition",
              x="Average Covered Charges",
              hue="Region")
```



## Statistical estimates - countplot

```
sns.countplot(data=df, y="DRG_Code", hue="Region")
```



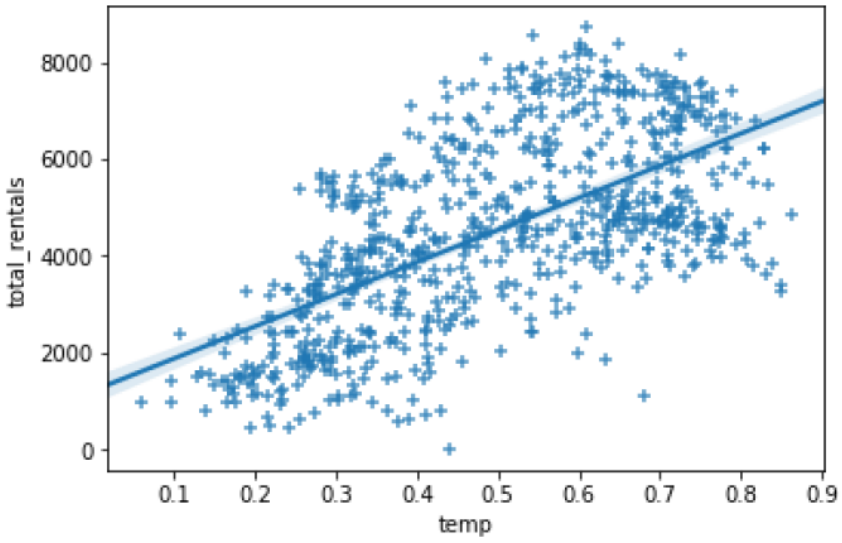

# Regression Plots

## Bicycle Dataset

- Aggregated bicycle sharing data in Washington DC
- Data includes:
  1)Rental amounts
  2)Weather information
  3)Calendar information
  4)Can we predict rental amounts?

## Plotting with regplot()

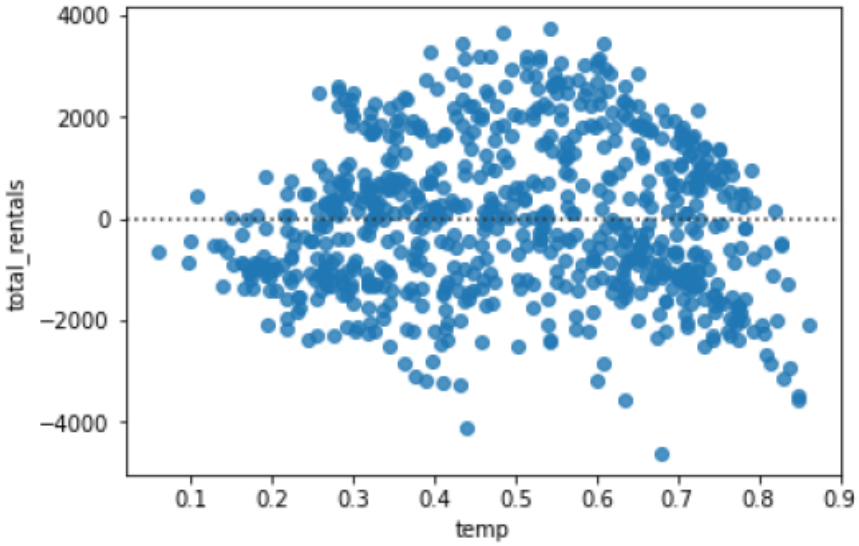
```
sns.regplot(data=df, x='temp',
            y='total_rentals', marker='+')
```




## Evaluating regression with residplot()

- A residual plot is useful for evaluating the fit of a model
- Seaborn supports through `residplot` function

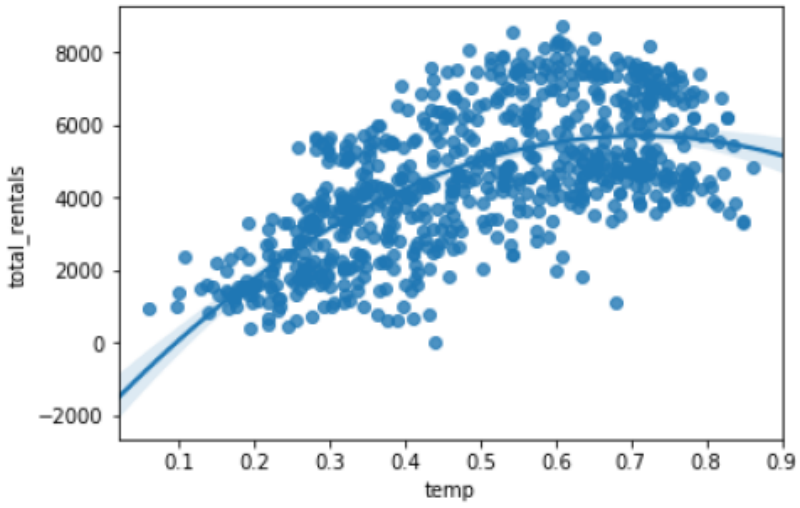
```
sns.residplot(data=df, x='temp', y='total_rentals')
```




## Polynomial regression

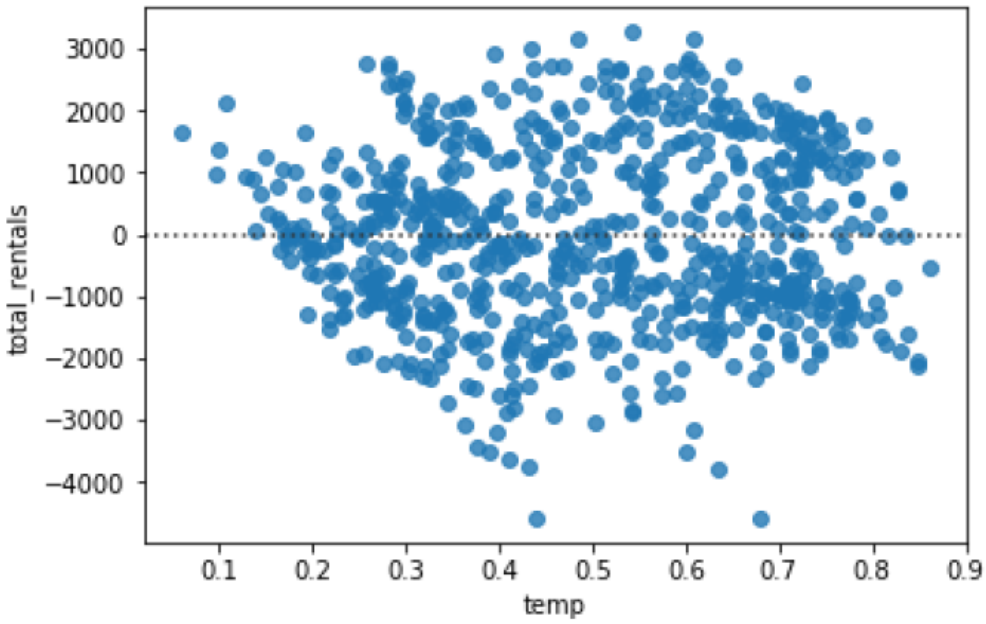
Seaborn supports polynomial regression using the order parameter

```
sns.regplot(data=df, x='temp',
            y='total_rentals', order=2)
```



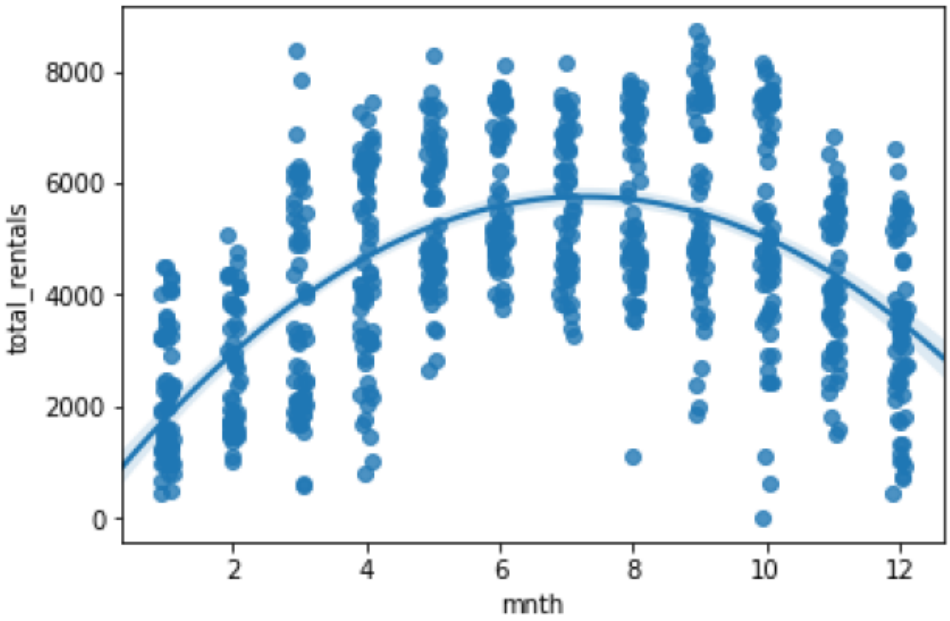

## residplot with polynomial regression

```
sns.residplot(data=df, x='temp',
              y='total_rentals', order=2)
```




## Categorical values

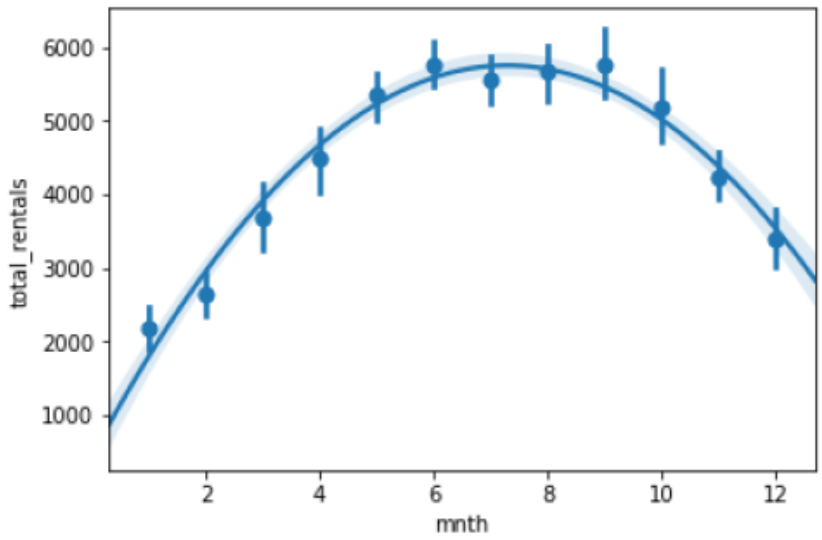
```
sns.regplot(data=df, x='mnth', y='total_rentals',
            x_jitter=.1, order=2)
```




## Estimators

In some cases, an `x_estimator` can be useful for highlighting trends

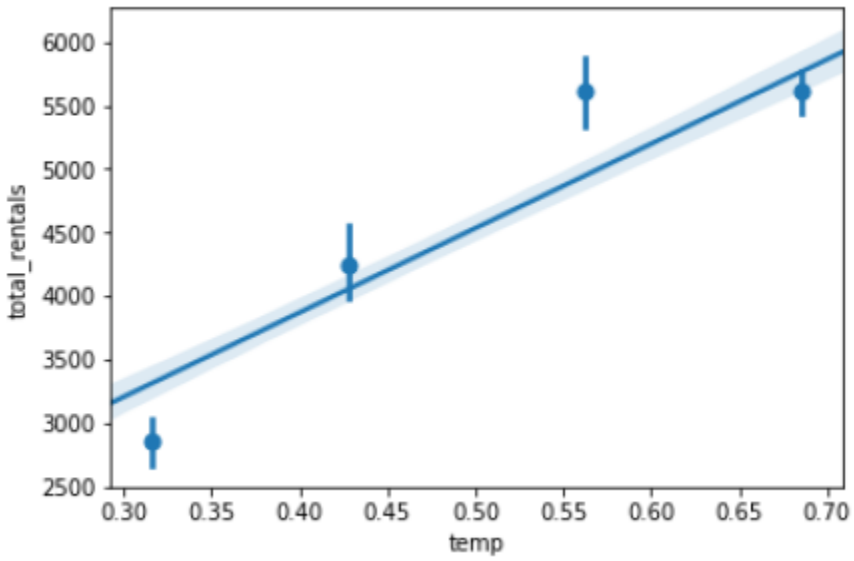
```
sns.regplot(data=df, x='mnth', y='total_rentals',
            x_estimator=np.mean, order=2)
```



## Binning the data

- x_bins can be used to divide the data into discrete bins
- The regression line is still fit against all form of data

```
sns.regplot(data=df,x='temp',y='total_rentals',
            x_bins=4)
```




# Matrix Plots

## Getting data in the right format

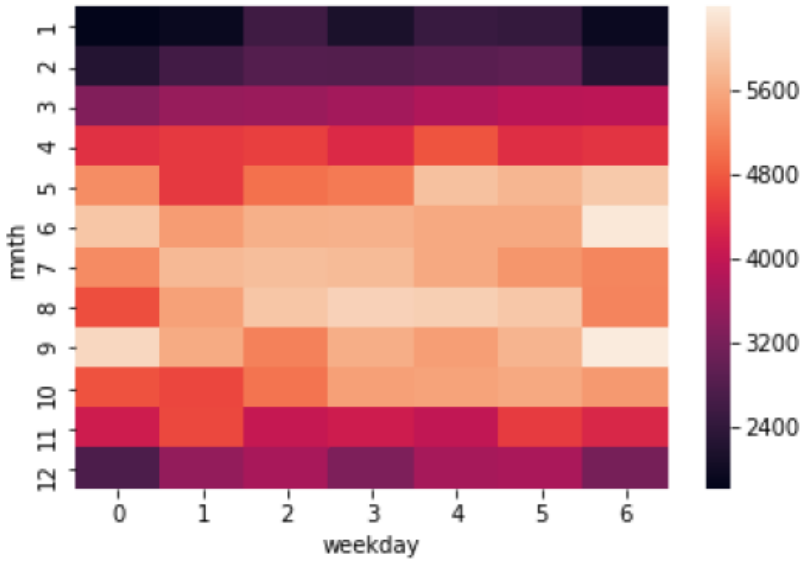
- Seaborn's `heatmap()` function requires data to be in a grid format
- pandas `crosstab()` is frequently used to manipulate the data

```
pd.crosstab(df["mnth"], df["weekday"],
            values=df["total_rentals"],aggfunc='mean').round(0)
```

| weekday | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| mnth | | | | | | | |
| 1 | 1816.0 | 1927.0 | 2568.0 | 2139.0 | 2513.0 | 2446.0 | 1957.0 |
| 2 | 2248.0 | 2604.0 | 2824.0 | 2813.0 | 2878.0 | 2933.0 | 2266.0 |
| 3 | 3301.0 | 3546.0 | 3574.0 | 3670.0 | 3817.0 | 3926.0 | 3939.0 |
| 4 | 4417.0 | 4516.0 | 4556.0 | 4331.0 | 4764.0 | 4387.0 | 4446.0 |
| 5 | 5320.0 | 4512.0 | 5025.0 | 5119.0 | 5893.0 | 5751.0 | 5978.0 |
| 6 | 5940.0 | 5478.0 | 5681.0 | 5701.0 | 5622.0 | 5616.0 | 6344.0 |
| 7 | 5298.0 | 5792.0 | 5844.0 | 5814.0 | 5624.0 | 5406.0 | 5232.0 |
| 8 | 4703.0 | 5518.0 | 5930.0 | 6077.0 | 6038.0 | 5958.0 | 5224.0 |
| 9 | 6160.0 | 5637.0 | 5184.0 | 5668.0 | 5486.0 | 5747.0 | 6394.0 |
| 10 | 4735.0 | 4632.0 | 5065.0 | 5505.0 | 5537.0 | 5623.0 | 5445.0 |
| 11 | 4126.0 | 4658.0 | 4040.0 | 4136.0 | 3994.0 | 4524.0 | 4288.0 |
| 12 | 2740.0 | 3498.0 | 3713.0 | 3270.0 | 3711.0 | 3742.0 | 3195.0 |

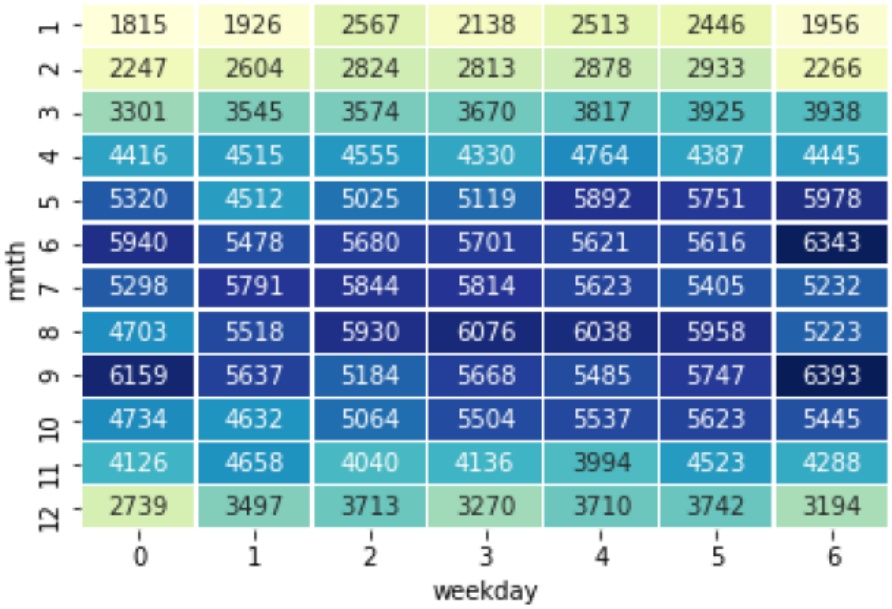
## Build a heatmap

```
sns.heatmap(pd.crosstab(df["mnth"], df["weekday"],
            values=df["total_rentals"], aggfunc='mean')
)
```




## Customize a heatmap

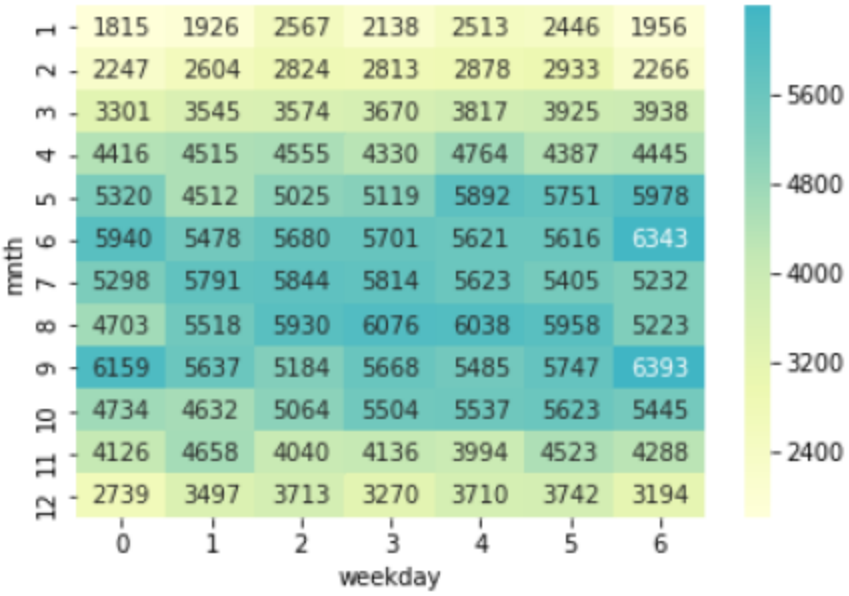
```
sns.heatmap(df_crosstab, annot=True, fmt="d",
cmap="YlGnBu", cbar=False, linewidths=.5)
```




## Centering a heatmap

Seaborn support centering the heatmap colors on a specific value

```
sns.heatmap(df_crosstab, annot=True, fmt="d",
            cmap="YlGnBu", cbar=True,
            center=df_crosstab.loc[9, 6])
```




## Plotting a correlation matrix

- Pandas `corr` function calculates correlations between columns in a dataframe
- The output can be converted to a heatmap with seaborn

```
sns.heatmap(df.corr())
```