

0.1 General Structure

```
rating_density = alt.Chart(df, title="Title").transform_density(
  density='col',
  groupby=['col2'],
  as_=['col', 'density']
).mark_area(
  opacity=0.6,
  line={'color': 'white'}
).encode(
  x=alt.X('col1:Q',
    title='the value',
    scale=alt.Scale(domain=[0, 10000]),
    axis=alt.Axis(format='~s',grid=False,tickCount=10)),
  y=alt.Y('density:Q',
    title='density',
    scale=alt.Scale(domain=[0, 0.000100]),
    axis=alt.Axis(format='.6f',grid=False)).stack(False),
  color=alt.Color('where:N',
    scale=alt.Scale(
      domain=['My state', 'Neighboring states'],
      range=['#A0D2E8', '#A0E8AF']
    ),
    legend=alt.Legend(
      orient='top'
    )
  )
).properties(
  width=600,
  height=400
)
```

```
options(repr.plot.width=7, repr.plot.height=3)

chart <- ggplot(wages) +
  aes(
    x = salary,
    y = when,
    fill = when
  ) +
  geom_violin() +
  geom_point(stat = 'summary', fun = median, color = 'White') +
  geom_point(stat = 'summary', fun = mean, color = 'Black') +
  labs(
    x = "Worker Salary ($)",
    y = "When",
    title = "Salary Distribution Before and After Policy Implementation"
  ) +
  scale_x_continuous(
    limits = c(10, 40), oob = scales::oob_keep,
    labels = scales::dollar_format(),
  )
chart
```

0.2 HEAT MAPS

A 2D histogram is a type of heatmap, where count is mapped to color, you could also have used a mark that maps size to color, which might even be more effective but that is not as commonly seen.

```
alt.Chart(df).mark_rect().encode(alt.X('col1').bin(maxbins=40),
alt.Y('col2').bin(maxbins=40),alt.Color('count()'))
)
```

Instead of squares,hexagonal bins can be used. These have theoretically superior qualities over squares, such as a more natural notation of neighbors. 2 dimensional KDEs in ggplot. This works just like 1D KDEs, except that the kernel on each data point extends in 2 dimensions Use ridges contours, similar to a topographic map. These contour plots are often less intuitive than the density plot above, so the recommendation is to use the density plot instead.

```
ggplot(df) + aes(x = col1, y = col2) + geom_bin2d() or geom_hex()
or geom_density_2d_filled() or geom_density_2d()
```

0.3 AXIS LABEL FORMATTING

Scientific Notation & Grid Tick Modifications Scientific notation, Consider formatting axis labels using plain numbers or appropriate unit prefixes (e.g., thousands, millions, micro, milli) to improve readability.

Note that tickCount cannot be applied to binned data, so the plot has been adjusted here to demonstrate its effect.

```
.alt.X('col').axis(format='e') # For e like notation
.alt.X('col').axis(format='s') # Standard international (SI) units
.alt.X('col').axis(format='s') # Removes trailing zeros
.alt.X('col').axis(format='~$s') # Formatters can also be combined.
.alt.X('col').axis(format='%.1%') # Decimal formats
.alt.X('col').axis(None) #Remove an axis altogether
.alt.Y('col').axis(tickCount=2) #ticks on axis
.alt.Y('col').axis(grid=False) #Remove grid
.alt.Y('col').sort('-y') #Sort y axis
.alt.Y('col').bin(maxbins=400).scale(domain=[0, 2000]), #Set scale
alt.themes.enable('dark') #Dark theme scale
alt.Color('count()').legend(None) # remove a legend
.alt.Y('col').bin(maxbins=400).scale(domain=(0, 2000),
reverse=True), #Reversing an axis
alt.Y('col:Q',bin=alt.Bin(maxbins=20),
title='Y Label', #Set Label
axis=alt.Axis(format='%.1%', titleFontSize=14,
labelFontSize=12))
```

```
ggplot(df) + aes(x = colx,y = coly) + geom_hex() +
guides(fill = "none") # remove a legend
scale_x_continuous(
  labels = scales::label_dollar(scale = .001, suffix = "K"),
  labels = scales::label_number(scale_cut = scales::cut_si('')),
  limits = c(10, 31), oob = scales::oob_keep, #scale limit
  limits = c(2000, 0), trans = 'reverse', #reversing an axis
  labels = scales::label_dollar(), #label formatter
  breaks = scales::pretty_breaks(n = 10) #tick count
)
```

```
ggplot(df) + aes(x = col1, y = col2) + geom_hex() +
labs(
  x = 'col1',
  y = 'col2',
  title = 'title',
  subtitle = 'subtitle'
)
```

0.4 Trendlines - Mean Line

Trendlines (also sometimes called “lines of best fit”, or “fitted lines”) are good to highlight general trends in the data that can be hard to elucidate by looking at the raw data points. This can happen if there are many data points or many groups inside the data.

```
points + points.mark_line().encode(y='mean(Horsepower)')
```

```
ggplot(cars) + aes(x = Year, y = Horsepower, color = Origin) +
geom_point() + geom_line(stat = 'summary', fun = 'mean')
```

0.5 Trendlines - Regression Lines

If you want an easy-to-interpret trend line, use a rolling mean (or a simple mean if the data isn't too noisy). It naturally highlights patterns as we would notice them visually and requires fewer statistical assumptions than methods like linear regression.

```
points + points.mark_line(size=3).transform_regression(
'Year',
'Horsepower',
groupby=['Origin'],
method='poly' #for regression line that is quadratic, polynomial
)
```

0.6 Trendlines - loess/lowess Lines

Where the w stands for “weighted”. You can fit multiple equations (usually linear and quadratic) to smaller subsets of the data, and add them together to get the final line. bandwidth parameter controls how much the loess fit should be influenced by local variation in the data, similar to the effect of the bandwidth parameter for a KDE. bandwidth of 1 corresponds to using all the data and will be similar to a linear regression

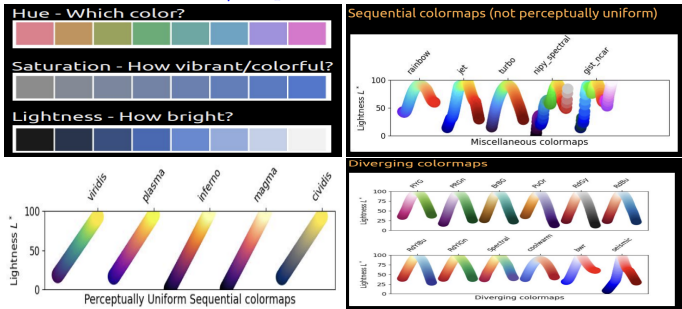
```
points + points.mark_line(size=3).transform_loess(
'Year',
'Horsepower',
groupby=['Origin'],
bandwidth=0.8 #optional param
)
```

```
ggplot(cars) +
aes(x = Year,
y = Horsepower,
color = Origin,
fill = Origin) + #color the confidence interval the same as the lines.
geom_point() +
geom_smooth() #Introduce CI
geom_smooth(se = FALSE, size = 2, span=1) #Introduce bandwidth use span
geom_smooth(se = FALSE, size = 2, method = 'lm') #linear regression instead of loess
```

0.7 When to choose which trendline:

- **Rolling mean / mean:** Best for clear, easy-to-read trends and general audiences. Simple and intuitive, but not suitable for predictions beyond your data.
- **Linear (or similar model):** Use when your data follows a clear equation. Good for showing consistent patterns and making predictions outside your data range.
- **Loess (or smooth curve):** Ideal for showing natural, flexible trends in current data without assuming a strict model, though poor for extrapolation.

0.8 Color schemes/maps:



colormap is useful for (categorical, sequential, diverging, cyclic) **Categorical:** Single color for each category **Sequential:** Shades of a color

```
#Sequential
alt.Chart(iris).mark_circle(size=100).encode(
x='petalWidth',
y='petalLength',
color=alt.Color('petalWidth').scale(scheme='viridis',
reverse=True)
)
#Diverging has a natural midpoint,
# such as a correlation that is defined from -1 to 1
corr_df =
data.gapminder().corr(numeric_only=True).stack().reset_index(name=
alt.Chart(corr_df).mark_rect().encode(
x='level_0',
y='level_1',
tooltip='corr',
color=alt.Color('corr').scale(domain=(-1, 1),
scheme='purpleorange')
).properties(
width=200,
height=200
)
```

```
#Sequential
ggplot(iris) +
aes(x = Petal.Width,
y = Petal.Length,
color = Petal.Width) +
geom_point(size = 5) +
scale_color_viridis_c()
#Diverging has a natural midpoint,
# such as a correlation that is defined from -1 to 1
```

```
ggplot(corr_df) +
  aes(x = level_0,
      y = level_1,
      fill = corr) +
  geom_tile() +
  ggthemes::scale_fill_gradient2_tableau()
```

0.9 Direct labeling instead of using a legend

```
lines = alt.Chart(stocks).mark_line().encode(
  x='date',
  y='price',
  color=alt.Color('symbol').legend(None)
)
text = alt.Chart(stock_order).mark_text(dx=20).encode(
  x='date',
  y='price',
  text='symbol',
  color='symbol'
)
lines + text
```

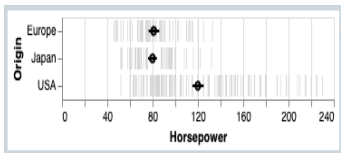
```
ggplot(stocks) +
  aes(x = date,
      y = price,
      color = symbol,
      label = symbol) +
  geom_line() +
  geom_text(data = stock_order, vjust=-1) +
  ggthemes::scale_color_tableau() +
  theme(legend.position = 'none')
```

0.10 Confidence Intervals

```
points.mark_errorband(extent='ci') +
  points.encode(y='mean(Horsepower)').mark_line() #add meanline
  with CI
err_bars = alt.Chart(cars).mark_errorbar(extent='ci',
  rule=alt.LineConfig(size=2)).encode(
```

```
  x='Horsepower',
  y='Origin'
)
(err_bars.mark_tick(color='lightgrey')
+ err_bars
+ err_bars.mark_point(color='black').encode(x='mean(Horsepower)'))
```

```
ggplot(cars) + aes(x = Year,y = Horsepower,color = Origin,fill =
Origin) +
  geom_line(stat = 'summary', fun = mean) +
  geom_ribbon(stat = 'summary', fun.data = mean_cl_boot,
  alpha=0.5, color = NA)
ggplot(cars) +
  aes(x = Horsepower,y = Origin) +
  geom_violin(color = NA, fill = 'steelblue', alpha = 0.4) +
  geom_pointrange(stat = 'summary', fun.data = mean_cl_boot, size
= 0.7)
```



0.11 Figure composition

```
mpg_weight & hp_weight #Concatenate plots vertically
mpg_weight | hp_weight #Concatenate plots Horizontally
plot_grid(hp_weight, mpg_weight) #Concatenate plots Horizontally
plot_grid(origin_count, mpg_weight, ncol=1) #Concatenate plots
vertically
alt.Chart(cars).mark_circle().encode(...).interactive() #Panning
and zooming
brush = alt.selection_interval()
alt.Chart(cars).mark_circle().encode(...).add_params(brush)
#Interval selections
```

0.12 Highlighting points with selections

```
brush = alt.selection_interval()
alt.Chart(cars).mark_circle().encode(
  x='Horsepower',
  y='Miles_per_Gallon',
  color=alt.condition(brush, 'Origin', alt.value('lightgray'))
).add_params(brush)
```

0.13 Linking selections across plots

```
brush = alt.selection_interval(resolve='union') # The default is
'global' #resolve='intersect'
points = alt.Chart(cars).mark_circle().encode(
  x='Horsepower',
  y='Miles_per_Gallon',
  color=alt.condition(brush, 'Origin', alt.value('lightgray'))
).add_params(brush)
points | points.encode(y='Weight_in_lbs')
```

0.14 Click selections

```
brush = alt.selection_interval()
click = alt.selection_point(fields=['Origin'], on='mouseover')
points = alt.Chart(cars).mark_circle().encode(
  x='Horsepower',
  y='Miles_per_Gallon',
  color=alt.condition(brush, 'Origin', alt.value('lightgray'))
).add_params(brush)
bars = alt.Chart(cars).mark_bar().encode(
  x='count()',
  y='Origin',
  color='Origin',
  opacity=alt.condition(click, alt.value(0.9), alt.value(0.2))
).add_params(click)
points & bars
```