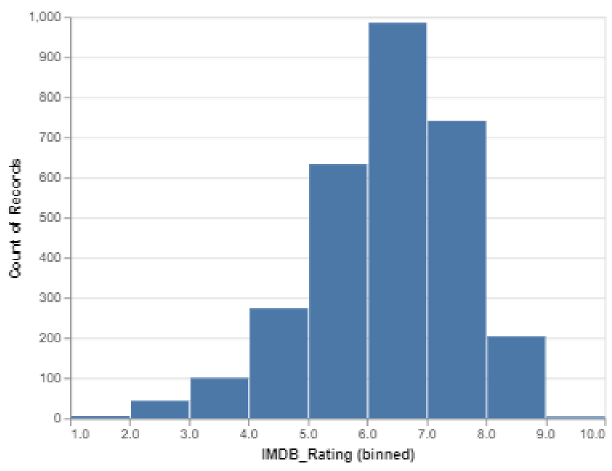
```
import altair as alt
import numpy as np
import pandas as pd
from vega_datasets import data

alt.themes.enable('dark')
```

```
     ThemeRegistry.enable('dark')
```

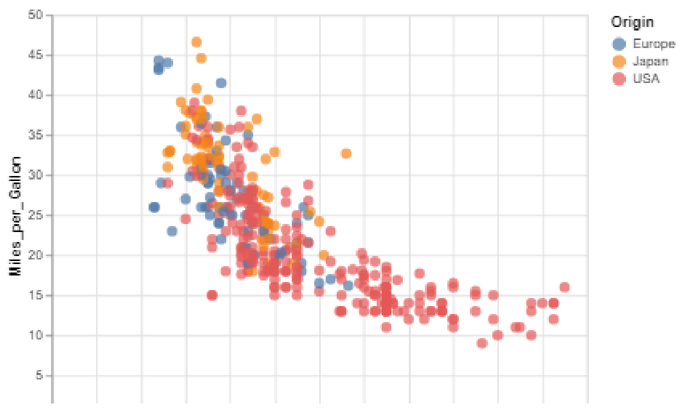
HISTOGRAM

```
import altair as alt
from vega_datasets import data

source = data.movies.url

alt.Chart(source).mark_bar().encode(
    alt.X("IMDB_Rating:Q", bin=True),
    y='count()',
)
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



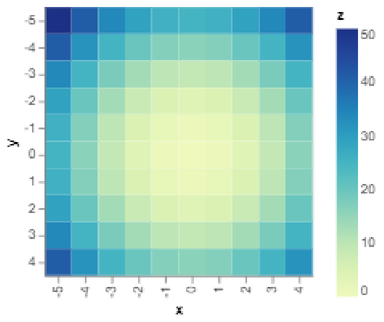
SCATTER PLOT

```
import altair as alt
from vega_datasets import data

source = data.cars()

alt.Chart(source).mark_circle(size=60).encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
    tooltip=['Name', 'Origin', 'Horsepower', 'Miles_per_Gallon']
).interactive()
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



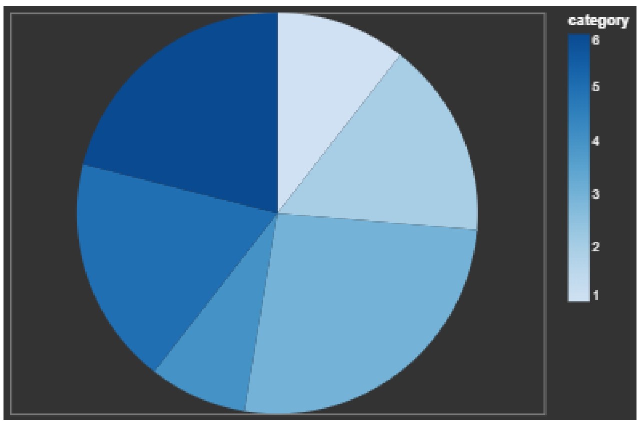
CORELATION HEATMAP

```
import altair as alt
import numpy as np
import pandas as pd

# Compute x^2 + y^2 across a 2D grid
x, y = np.meshgrid(range(-5, 5), range(-5, 5))
z = x ** 2 + y ** 2

# Convert this grid to columnar data expected by Altair
source = pd.DataFrame({'x': x.ravel(),
                     'y': y.ravel(),
                     'z': z.ravel()})

alt.Chart(source).mark_rect().encode(
    x='x:O',
    y='y:O',
    color='z:Q'
)
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



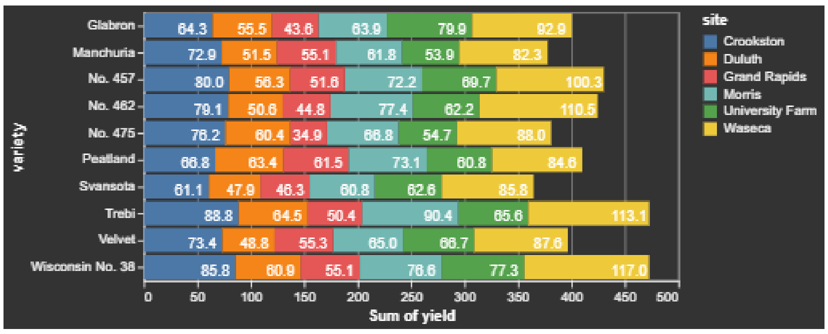
PIE CHART

```
import pandas as pd
import altair as alt

source = pd.DataFrame({"category": [1, 2, 3, 4, 5, 6], "value": [4, 6, 10, 3, 7, 8]})

alt.Chart(source).mark_arc().encode(
    theta="value",
    color="category"
)
```



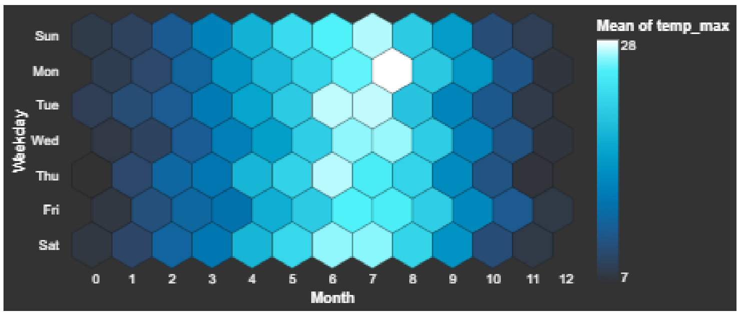
FACETED STACKED BAR CHART

```
import altair as alt
from vega_datasets import data

source=data.barley()

bars = alt.Chart(source).mark_bar().encode(
    x=alt.X('sum(yield):Q', stack='zero'),
    y=alt.Y('variety:N'),
    color=alt.Color('site')
)

text = alt.Chart(source).mark_text(dx=-15, dy=3, color='white').encode(
    x=alt.X('sum(yield):Q', stack='zero'),
    y=alt.Y('variety:N'),
    detail='site:N',
    text=alt.Text('sum(yield):Q', format='.1f')
)

bars + text
```



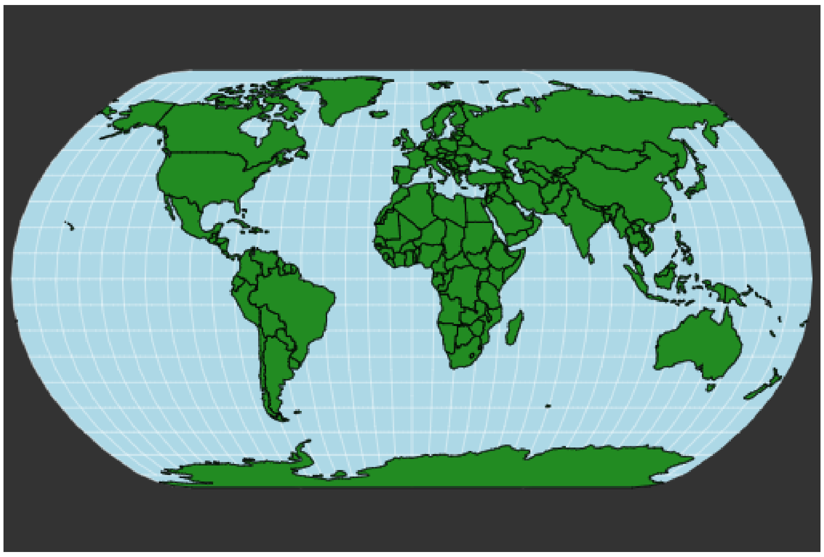
HEXBIN CHART

```
import altair as alt
from vega_datasets import data

source = data.seattle_weather()

# Size of the hexbins
size = 15
# Count of distinct x features
xFeaturesCount = 12
# Count of distinct y features
yFeaturesCount = 7
# Name of the x field
xField = 'date'
# Name of the y field
yField = 'date'

# the shape of a hexagon
hexagon = "M0,-2.3094010768L2,-1.1547005384 2,1.1547005384 0,2.3094010768 -2,1.1547005384 -2,-1.1547005384Z"

alt.Chart(source).mark_point(size=size**2, shape=hexagon).encode(
    x=alt.X('xFeaturePos:Q', axis=alt.Axis(title='Month',
                                           grid=False, tickOpacity=0, domainOpacity=0)),
    y=alt.Y('day(' + yField + '):O', axis=alt.Axis(title='Weekday',
                                                   labelPadding=20, tickOpacity=0, domainOpacity=0)),
    stroke=alt.value('black'),
    strokeWidth=alt.value(0.2),
    fill=alt.Color('mean(temp_max):Q', scale=alt.Scale(scheme='darkblue')),
    tooltip=['month(' + xField + '):O', 'day(' + yField + '):O', 'mean(temp_max):Q']
).transform_calculate(
    # This field is required for the hexagonal X-Offset
    xFeaturePos='(day(datum.' + yField + ') % 2) / 2 + month(datum.' + xField + ')'
).properties(
    # Exact scaling factors to make the hexbins fit
    width=size * xFeaturesCount * 2,
    height=size * yFeaturesCount * 1.7320508076,  # 1.7320508076 is approx. sin(60°)*2
).configure_view(
    strokeWidth=0
)
```



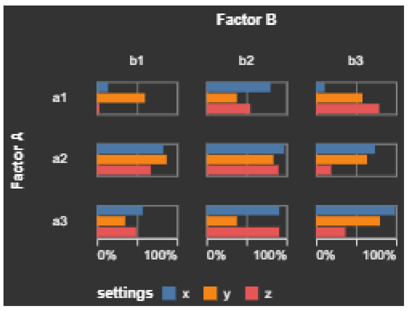
WORLD MAP

```
import altair as alt
from vega_datasets import data

# Data generators for the background
sphere = alt.sphere()
graticule = alt.graticule()

# Source of land data
source = alt.topo_feature(data.world_110m.url, 'countries')

# Layering and configuring the components
alt.layer(
    alt.Chart(sphere).mark_geoshape(fill='lightblue'),
    alt.Chart(graticule).mark_geoshape(stroke='white', strokeWidth=0.5),
    alt.Chart(source).mark_geoshape(fill='ForestGreen', stroke='black')
).project(
    'naturalEarth1'
).properties(width=600, height=400).configure_view(stroke=None)
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



COMPACT FACED GRID OF BAR CHARTS

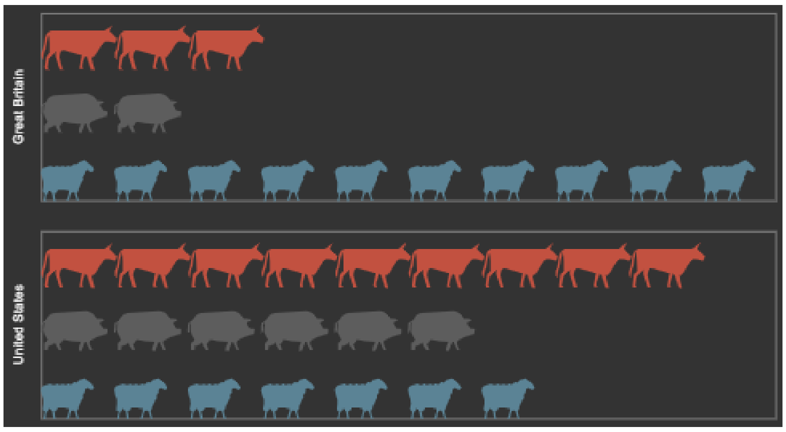
```
import altair as alt
import pandas as pd

source = pd.DataFrame(
    [
        {"a": "a1", "b": "b1", "c": "x", "p": "0.14"},
        {"a": "a1", "b": "b1", "c": "y", "p": "0.60"},
        {"a": "a1", "b": "b1", "c": "z", "p": "0.03"},
        {"a": "a1", "b": "b2", "c": "x", "p": "0.80"},
        {"a": "a1", "b": "b2", "c": "y", "p": "0.38"},
        {"a": "a1", "b": "b2", "c": "z", "p": "0.55"},
        {"a": "a1", "b": "b3", "c": "x", "p": "0.11"},
        {"a": "a1", "b": "b3", "c": "y", "p": "0.58"},
        {"a": "a1", "b": "b3", "c": "z", "p": "0.79"},
        {"a": "a2", "b": "b1", "c": "x", "p": "0.83"},
        {"a": "a2", "b": "b1", "c": "y", "p": "0.87"},
        {"a": "a2", "b": "b1", "c": "z", "p": "0.67"},
        {"a": "a2", "b": "b2", "c": "x", "p": "0.97"},
        {"a": "a2", "b": "b2", "c": "y", "p": "0.84"},
        {"a": "a2", "b": "b2", "c": "z", "p": "0.90"},
        {"a": "a2", "b": "b3", "c": "x", "p": "0.74"},
        {"a": "a2", "b": "b3", "c": "y", "p": "0.64"},
        {"a": "a2", "b": "b3", "c": "z", "p": "0.19"},
        {"a": "a3", "b": "b1", "c": "x", "p": "0.57"},
        {"a": "a3", "b": "b1", "c": "y", "p": "0.35"},
        {"a": "a3", "b": "b1", "c": "z", "p": "0.49"},
        {"a": "a3", "b": "b2", "c": "x", "p": "0.91"},
        {"a": "a3", "b": "b2", "c": "y", "p": "0.38"},
        {"a": "a3", "b": "b2", "c": "z", "p": "0.91"},
        {"a": "a3", "b": "b3", "c": "x", "p": "0.99"},
        {"a": "a3", "b": "b3", "c": "y", "p": "0.80"},
        {"a": "a3", "b": "b3", "c": "z", "p": "0.37"},
    ]
)

alt.Chart(source, width=60, height=alt.Step(8)).mark_bar().encode(
    y=alt.Y("c:N", axis=None),
    x=alt.X("p:Q", title=None, axis=alt.Axis(format="%")),
    color=alt.Color(
        "c:N", title="settings", legend=alt.Legend(orient="bottom", titleOrient="left")
    ),
    row=alt.Row("a:N", title="Factor A", header=alt.Header(labelAngle=0)),
    column=alt.Column("b:N", title="Factor B"),
)
```



ISOTYPE VISUALIZATION

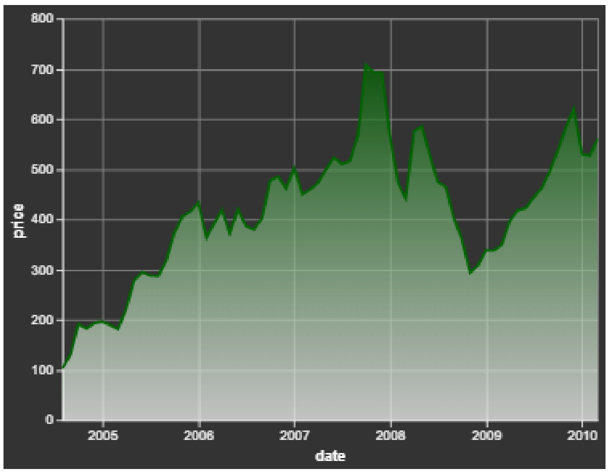
```
import altair as alt
import pandas as pd

source = pd.DataFrame([
      {'country': 'Great Britain', 'animal': 'cattle'},
      {'country': 'Great Britain', 'animal': 'cattle'},
      {'country': 'Great Britain', 'animal': 'cattle'},
      {'country': 'Great Britain', 'animal': 'pigs'},
      {'country': 'Great Britain', 'animal': 'pigs'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'Great Britain', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'cattle'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'pigs'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'},
      {'country': 'United States', 'animal': 'sheep'}
    ])

domains = ['person', 'cattle', 'pigs', 'sheep']

shape_scale = alt.Scale(
    domain=domains,
    range=[
        'M1.7 -1.7h-0.8c0.3 -0.2 0.6 -0.5 0.6 -0.9c0 -0.6 -0.4 -1 -1 -1c-0.6 0 -1 0.4 -1 1c0 0.4 0.2 0.7 0.6 0.9h-0.8c-0.4 0 -0.7 0.3 -0.
        'M4 -2c0 0 0.9 -0.7 1.1 -0.8c0.1 -0.1 -0.1 0.5 -0.3 0.7c-0.2 0.2 1.1 1.1 1.1 1.2c0 0.2 -0.2 0.8 -0.4 0.7c-0.1 0 -0.8 -0.3 -1.3 -0
        'M1.2 -2c0 0 0.7 0 1.2 0.5c0.5 0.5 0.4 0.6 0.5 0.6c0.1 0 0.7 0 0.8 0.1c0.1 0 0.2 0.2 0.2 0.2c0 0 -0.6 0.2 -0.6 0.3c0 0.1 0.4 0.9
        'M-4.1 -0.5c0.2 0 0.2 0.2 0.5 0.2c0.3 0 0.3 -0.2 0.5 -0.2c0.2 0 0.2 0.2 0.4 0.2c0.2 0 0.2 -0.2 0.5 -0.2c0.2 0 0.2 0.2 0.4 0.2c0.2
    ]
)
```

```
color_scale = alt.Scale(
    domain=domains,
    range=['rgb(162,160,152)', 'rgb(194,81,64)', 'rgb(93,93,93)', 'rgb(91,131,149)']
)

alt.Chart(source).mark_point(filled=True, opacity=1, size=100).encode(
    alt.X('x:O', axis=None),
    alt.Y('animal:O', axis=None),
    alt.Row('country:N', header=alt.Header(title='')),
    alt.Shape('animal:N', legend=None, scale=shape_scale),
    alt.Color('animal:N', legend=None, scale=color_scale),
).transform_window(
    x='rank()',
    groupby=['country', 'animal']
).properties(width=550, height=140)
```



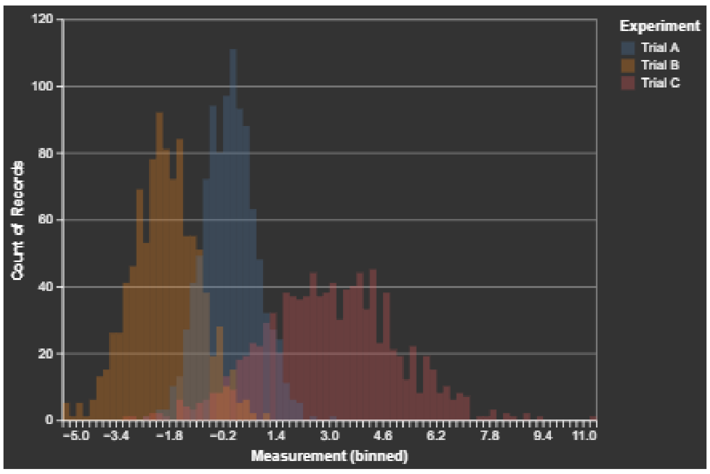
AREA CHART WITH GRADIENT

```
import altair as alt
from vega_datasets import data

source = data.stocks()

alt.Chart(source).transform_filter(
    'datum.symbol==="GOOG"'
).mark_area(
    line={'color':'darkgreen'},
    color=alt.Gradient(
        gradient='linear',
        stops=[alt.GradientStop(color='white', offset=0),
               alt.GradientStop(color='darkgreen', offset=1)],
        x1=1,
        x2=1,
        y1=1,
        y2=0
    )
).encode(
    alt.X('date:T'),
    alt.Y('price:Q')
)# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



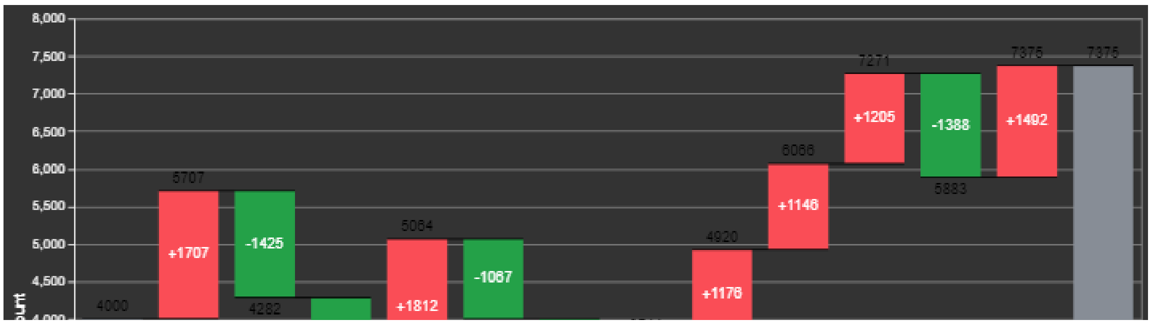
LAYERED AREA CHART

```
import pandas as pd
import altair as alt
import numpy as np
np.random.seed(42)

# Generating Data
source = pd.DataFrame({
    'Trial A': np.random.normal(0, 0.8, 1000),
    'Trial B': np.random.normal(-2, 1, 1000),
    'Trial C': np.random.normal(3, 2, 1000)
})

alt.Chart(source).transform_fold(
    ['Trial A', 'Trial B', 'Trial C'],
    as_=['Experiment', 'Measurement']
).mark_bar(
    opacity=0.3,
    binSpacing=0
).encode(
    alt.X('Measurement:Q', bin=alt.Bin(maxbins=100)),
    alt.Y('count()', stack=None),
    alt.Color('Experiment:N')
)
```



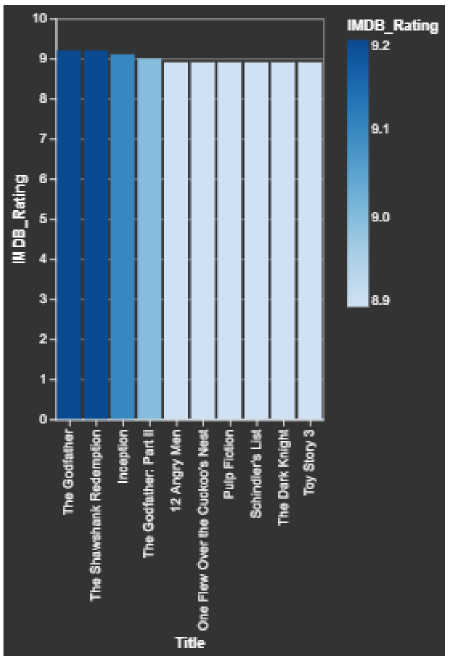
WATERFALL CHART

```
import altair as alt
import pandas as pd

data = [
    {"label": "Begin", "amount": 4000},
    {"label": "Jan", "amount": 1707},
    {"label": "Feb", "amount": -1425},
    {"label": "Mar", "amount": -1030},
    {"label": "Apr", "amount": 1812},
    {"label": "May", "amount": -1067},
    {"label": "Jun", "amount": -1481},
    {"label": "Jul", "amount": 1228},
    {"label": "Aug", "amount": 1176},
    {"label": "Sep", "amount": 1146},
    {"label": "Oct", "amount": 1205},
    {"label": "Nov", "amount": -1388},
    {"label": "Dec", "amount": 1492},
    {"label": "End", "amount": 0},
]
source = pd.DataFrame(data)

# The "base_chart" defines the transform_window, transform_calculate, and X axis
base_chart = alt.Chart(source).transform_window(
    window_sum_amount="sum(amount)",
    window_lead_label="lead(label)",
).transform_calculate(
    calc_lead="datum.window_lead_label === null ? datum.label : datum.window_lead_label",
    calc_prev_sum="datum.label === 'End' ? 0 : datum.window_sum_amount - datum.amount",
    calc_amount="datum.label === 'End' ? datum.window_sum_amount : datum.amount",
    calc_text_amount="(datum.label !== 'Begin' && datum.label !== 'End' && datum.calc_amount > 0 ? '+' : '') + datum.calc_amount",
    calc_center="(datum.window_sum_amount + datum.calc_prev_sum) / 2",
    calc_sum_dec="datum.window_sum_amount < datum.calc_prev_sum ? datum.window_sum_amount : ''",
    calc_sum_inc="datum.window_sum_amount > datum.calc_prev_sum ? datum.window_sum_amount : ''",
).encode(
    x=alt.X(
        "label:O",
        axis=alt.Axis(title="Months", labelAngle=0),
        sort=None,
    )
)

# alt.condition does not support multiple if else conditions which is why
# we use a dictionary instead. See https://stackoverflow.com/a/66109641
# for more information
color_coding = {
    "condition": [
        {"test": "datum.label === 'Begin' || datum.label === 'End'", "value": "#878d96"},
        {"test": "datum.calc_amount < 0", "value": "#24a148"},
    ],
    "value": "#fa4d56",
}

bar = base_chart.mark_bar(size=45).encode(
    y=alt.Y("calc_prev_sum:Q", title="Amount"),
    y2=alt.Y2("window_sum_amount:Q"),
    color=color_coding,
)

# The "rule" chart is for the horizontal lines that connect the bars
rule = base_chart.mark_rule(
    xOffset=-22.5,
    x2Offset=22.5,
).encode(
    y="window_sum_amount:Q",
    x2="calc_lead",
)

# Add values as text
text_pos_values_top_of_bar = base_chart.mark_text(
    baseline="bottom",
    dy=-4
).encode(
    text=alt.Text("calc_sum_inc:N"),
    y="calc_sum_inc:Q"
)
text_neg_values_bot_of_bar = base_chart.mark_text(
    baseline="top",
    dy=4
).encode(
    text=alt.Text("calc_sum_dec:N"),
    y="calc_sum_dec:Q"
)
text_bar_values_mid_of_bar = base_chart.mark_text(baseline="middle").encode(
    text=alt.Text("calc_text_amount:N"),
    y="calc_center:Q",
    color=alt.value("white"),
)

alt.layer(
    bar,
    rule,
    text_pos_values_top_of_bar,
    text_neg_values_bot_of_bar,
    text_bar_values_mid_of_bar
).properties(
    width=800,
    height=450
)# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



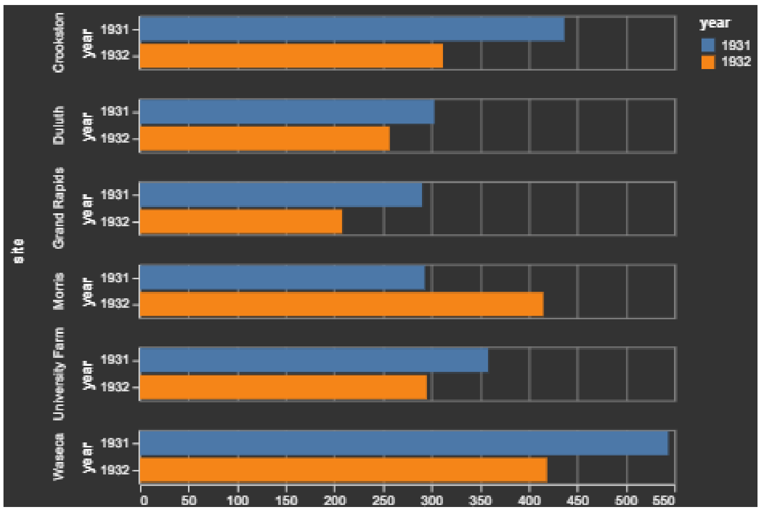
TOP K ITEMS

```
import altair as alt
from vega_datasets import data

source = data.movies.url

# Top 10 movies by IMBD rating
alt.Chart(
    source,
).mark_bar().encode(
    x=alt.X('Title:N', sort='-y'),
    y=alt.Y('IMDB_Rating:Q'),
    color=alt.Color('IMDB_Rating:Q')

).transform_window(
    rank='rank(IMDB_Rating)',
    sort=[alt.SortField('IMDB_Rating', order='descending')]
).transform_filter(
    (alt.datum.rank < 10)
)
```



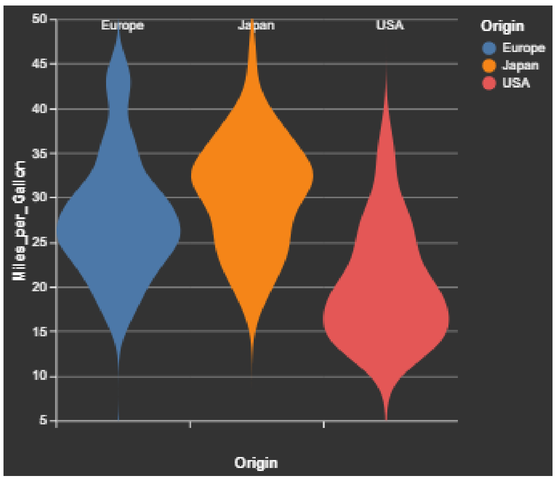
HORIZONTAL GROUPED BAR CHART

```
import altair as alt
from vega_datasets import data

source = data.barley()

alt.Chart(source).mark_bar().encode(
    x='sum(yield):Q',
    y='year:O',
    color='year:N',
    row='site:N'
)
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```



VIOLIN PLOT

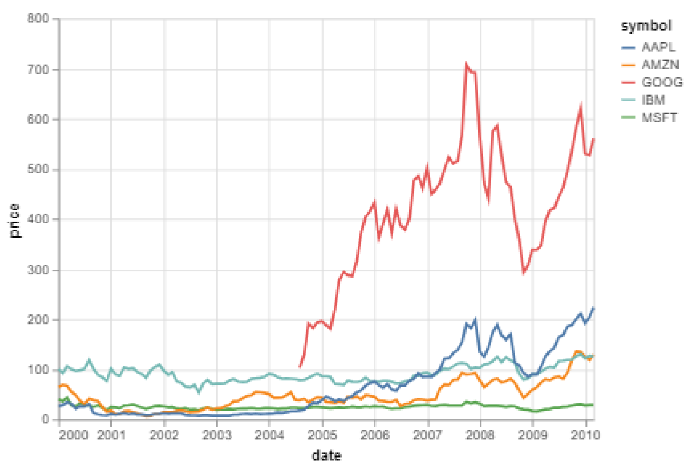
```
import altair as alt
from vega_datasets import data

alt.Chart(data.cars()).transform_density(
    'Miles_per_Gallon',
    as_=['Miles_per_Gallon', 'density'],
    extent=[5, 50],
    groupby=['Origin']
).mark_area(orient='horizontal').encode(
    y='Miles_per_Gallon:Q',
    color='Origin:N',
    x=alt.X(
        'density:Q',
        stack='center',
        impute=None,
        title=None,
        axis=alt.Axis(labels=False, values=[0],grid=False, ticks=True),
    ),
    column=alt.Column(
        'Origin:N',
        header=alt.Header(
            titleOrient='bottom',
            labelOrient='bottom',
            labelPadding=0,
        ),
    )
).properties(
    width=100
).configure_facet(
    spacing=0
).configure_view(
    stroke=None
)
```



MULTIPLE LINE CHART

```
import altair as alt
from vega_datasets import data
```

```
source = data.stocks()

alt.Chart(source).mark_line().encode(
    x='date:T',
    y='price:Q',
    color='symbol:N',
)
# No channel encoding options are specified in this chart
# so the code is the same as for the method-based syntax.
```


symbol
AAPL
AMZN
GOOG
IBM
MSFT
price
date