Библиотеки визуализации данных
Matplotlib, Seaborn, Altair, Plotly

Библиотеки визуализации (Visualization Libraries)

Способность создавать легко понятные, но сложные графики жизненно важна для того, чтобы стать успешным специалистом по анализу данных. Создание визуализаций - отличный способ рассказать историю, лежащую в основе ваших данных. Визуализации выделяют взаимосвязи в данных и раскрывают информацию, видимую человеческому глазу, которую нельзя передать только числами и цифрами.

Визуализации можно создавать несколькими способами:

- Matplotlib
- Pandas (через Matplotlib)
- Seaborn
- Altair
- Plotly, Plotly Express

Basic Scatter Plots with Matplotlib

Scatter plots могут быть созданы из Pandas Series (Scatter plots can be created from Pandas Series)

Code

```
Import matplotlib.pyplot as plt


plt.plot(data.sepal_length,
         data.sepal_width,
         marker='o')
```

Output

Прежде чем мы перейдем к рассмотрению методов библиотек seaborn и plotly, обсудим самый простой и зачастую удобный способ визуализировать данные из pandas DataFrame — это воспользоваться функцией plot. Реализация функции plot в pandas основана на библиотеке matplotlib.

Basic Scatter Plots with Matplotlib

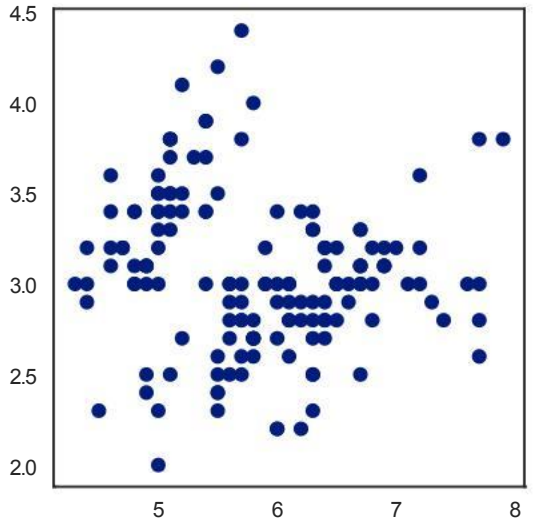
Scatter plots могут быть созданы из Pandas Series (Scatter plots can be created from Pandas Series)

Code

```
Import matplotlib.pyplot as plt


plt.plot(data.sepal_length,
         data.sepal_width,
         marker='o')
```

Output



Basic Scatter Plots with Matplotlib

Также можно добавить несколько слоев данных (Multiple layers of data can also be added)

Code

```
plt.plot(data.sepal_length,
         data.sepal_width,
         marker='o',
         label='sepal')

plt.plot(data.petal_length,
         data.petal_width,
          marker='o',
         label='petal')
```

Output

Basic Scatter Plots with Matplotlib

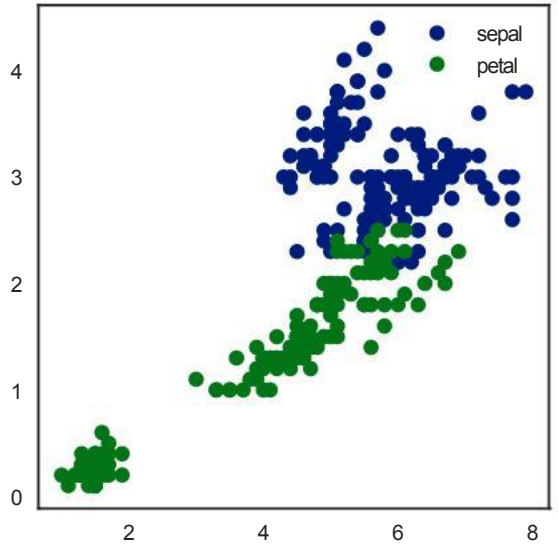
Также можно добавить несколько слоев данных (Multiple layers of data can also be added)

Code

```
plt.plot(data.sepal_length,
         data.sepal_width,
         marker='o',
         label='sepal')

plt.plot(data.petal_length,
         data.petal_width,
          marker='o',
         label='petal')
```

Output



Histograms with Matplotlib

Создание гистограмм (Histograms can be created from Pandas Series)

Code

```
plt.hist(data.sepal_length, bins=25)
```

Output

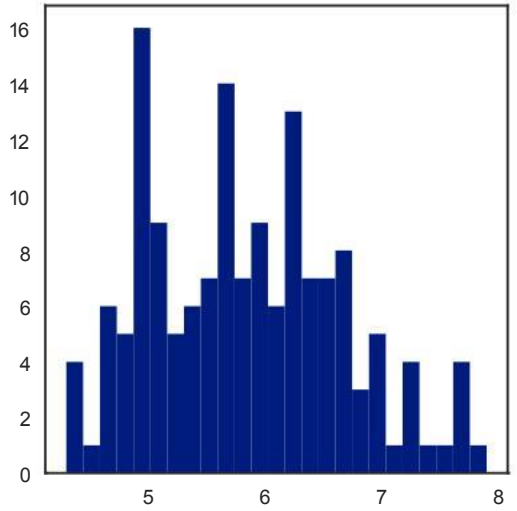
Histograms with Matplotlib

Histograms can be created from Pandas Series

Code

```
plt.hist(data.sepal_length, bins=25)
```

Output



Настройка (Customizing) Matplotlib Plots

Matplotlib позволяет очень гибко настраивать графики. На графике можно изменить почти все, что угодно, но потребуется порыться в документации и найти нужные параметры

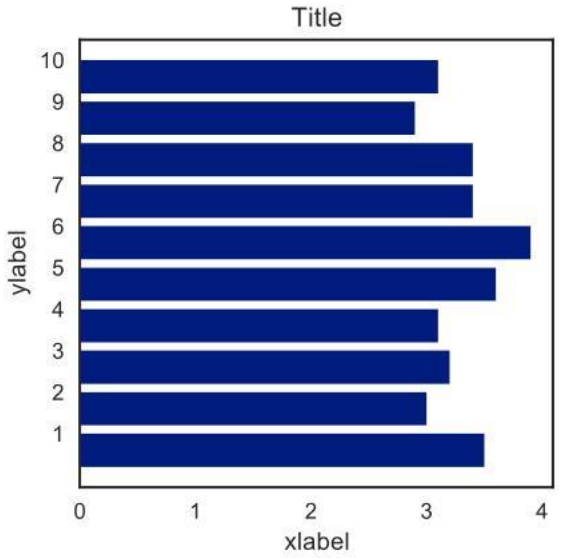
Code

```
fig, ax = plt.subplots()

ax.barh(np.arange(10),
         data.sepal_width.iloc[:10])

# Set position of ticks and tick labels
ax.set_yticks(np.arange(0.4,10.4,1.0))
ax.set_yticklabels(np.arange(1,11))
ax.set(xlabel='xlabel', ylabel='ylabel',
       title='Title')
```

Output



Использование статистических вычислений (Incorporating Statistical Calculations)

Статистические расчеты могут быть включены с помощью методов Pandas (Statistical calculations can be included with Pandas methods)

Code

```
(data
 .groupby('species')
 .mean()
 .plot(color=['red','blue',
              'black','green'],
       fontsize=10.0, figsize=(4,4)))
```

Output

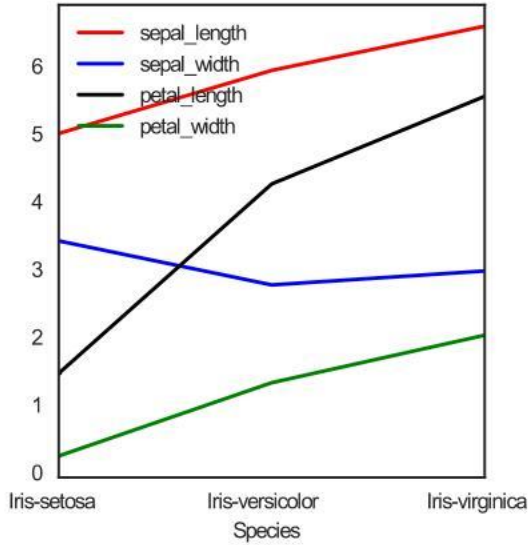
Incorporating Statistical Calculations

Statistical calculations can be included with Pandas methods

Code

```
(data
 .groupby('species')
 .mean()
 .plot(color=['red','blue',
              'black','green'],
       fontsize=10.0, figsize=(4,4)))
```

Output



Statistical Plotting with Seaborn

Создание визуализаций совместного распределения и диаграмм рассеяния в seaborn (Joint distribution and scatter plots can be created)

Code

Output

```
import seaborn as sns

sns.jointplot(x='sepal_length',
              y='sepal_width',
              data=data, size=4)
```

Seaborn — это, по сути, более высокоуровневое API на базе библиотеки matplotlib. Seaborn содержит более адекватные дефолтные настройки оформления графиков. Также в библиотеке есть достаточно сложные типы визуализации, которые в matplotlib потребовали бы большого количество кода.

Statistical Plotting with Seaborn

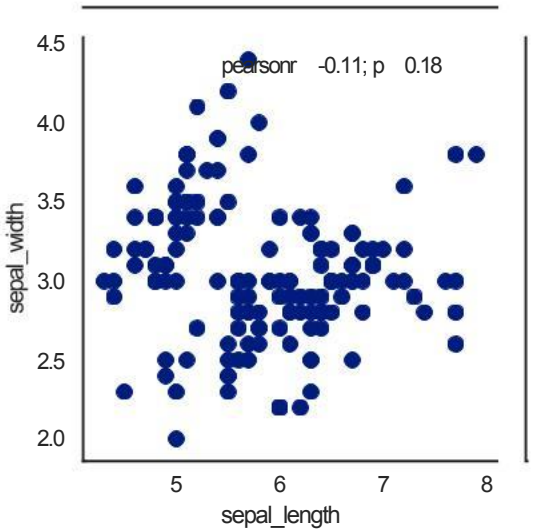
Создание визуализаций совместного распределения и диаграмм рассеяния в seaborn (Joint distribution and scatter plots can be created)

Code

```
import seaborn as sns

sns.jointplot(x='sepal_length',
              y='sepal_width',
              data=data, size=4)
```

Output



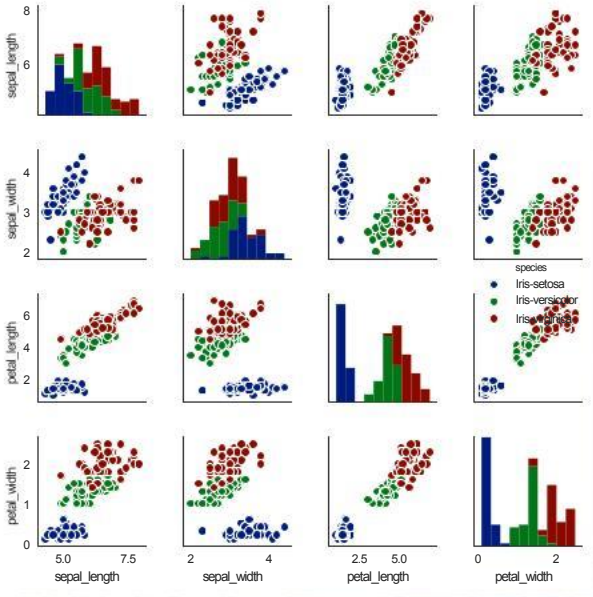
Statistical Plotting with Seaborn

Графики попарной корреляции всех переменных в Seaborn (Correlation plots of all variable pairs can also be mad with Seaborn)

Code

```
sns.pairplot(data, hue='species', size=3)
```

Output



Altair

Altair - это библиотека Python, предназначенная для статистической визуализации. Оно носит декларативный характер

Как специалисту по анализу данным, Altair позволит вам сосредоточить свое время и больше усилий на ваших данных - на их понимании, анализе и визуализации, а не на коде, необходимом для этого. Это означает, что вы можете определить данные и результат, который вы ожидаете увидеть (как должна выглядеть визуализация в конце), и Альтаир автоматически выполнит необходимые манипуляции за вас.

Mark Name	Method	Description
area	`mark_area()`	A filled area plot.
bar	`mark_bar()`	A bar plot.
circle	`mark_circle()`	A scatter plot with filled circles.
geoshape	`mark_geoshape()`	A geographic shape
image	`mark_image()`	A scatter plot with image markers.
line	`mark_line()`	A line plot.
point	`mark_point()`	A scatter plot with configurable point shapes.
rect	`mark_rect()`	A filled rectangle, used for heatmaps
rule	`mark_rule()`	A vertical or horizontal line spanning the axis.
square	`mark_square()`	A scatter plot with filled squares.
text	`mark_text()`	A scatter plot with points represented by text.
tick	`mark_tick()`	A vertical or horizontal tick mark.

Altair in Python: Data Visualizations

Визуализации данных в Python с помощью Altair

Code

```
!pip install altair vega_datasets
import pandas as pd
import altair as alt

# Importing the Vega Dataset
from vega_datasets import data as vega_data

movies_df = pd.read_json(vega_data.movies.url)

# Checking the type of data that we get
print("movies_df is of the type: ", type(movies_df))

print("movies_df: ",

movies_df.shape
```

Output

```
movies_df is of the type:  <class
'pandas.core.frame.DataFrame'>

movies_df:  (3201, 16)
```

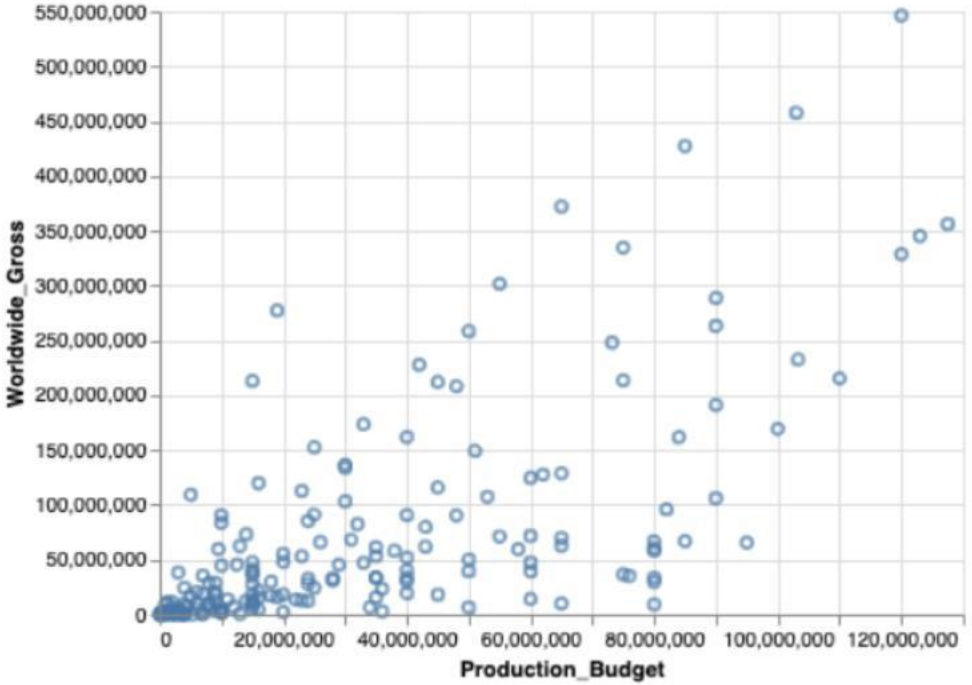
Altair in Python: Data Visualizations

Визуализации данных в Python с помощью Altair

Code

```
alt.Chart(movies_2000).mark_
point().encode(
alt.X('Production_Budget'),
alt.Y('Worldwide_Gross') )
```

Output



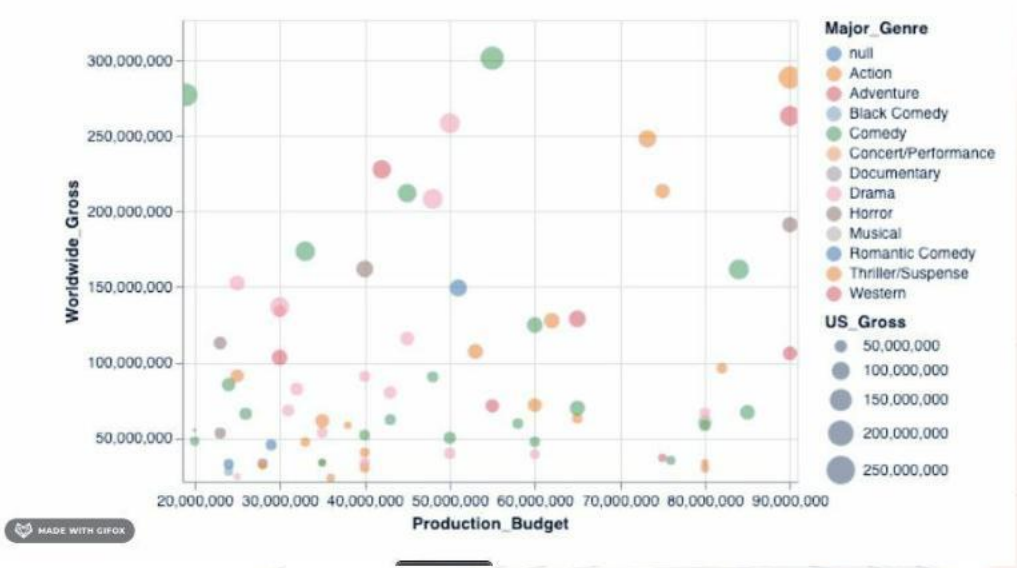
Altair in Python: Data Visualizations

Визуализации данных в Python с помощью Altair

Code

```
alt.Chart(movies_2000).mark_point
(filled=True).encode(
alt.X('Production_Budget'),
alt.Y('Worldwide_Gross'),
alt.Size('US_Gross'),
alt.Color('Major_Genre'),
alt.OpacityValue(0.7), tooltip =
[alt.Tooltip('Title'),
alt.Tooltip('Production_Budget'),
alt.Tooltip('Worldwide_Gross'),
alt.Tooltip('US_Gross') ]
).interactive()
```

Output



Product Gallery_vizual

Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее изменение: 24 марта

Комментировать Поделиться

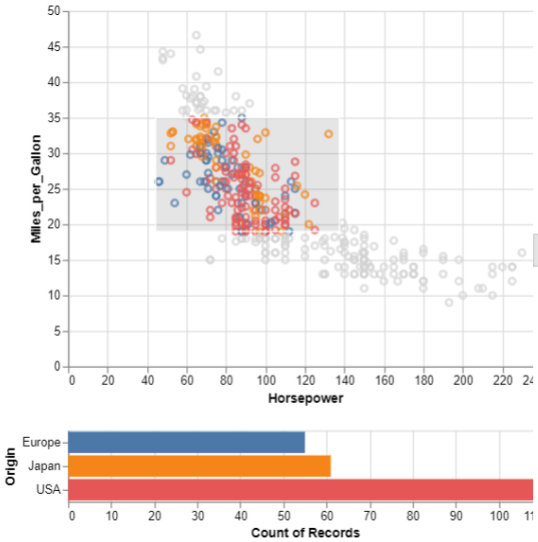
+ Код + Текст

ОЗУ Диск

Редактирование

```
bar = alt.Chart(cars).mark_bar().encode(
  x='count()',
  y='Origin',
  color='Origin'
).transform_filter(interval)

points & bar
```



Фрагменты кода

alt

- Visualization: Linked Scatter-Plot and Histogram in Altair
- Visualization: Linked Brushing in Altair
- Visualization: Interactive Brushing in Altair
- Visualization: Time Series Line Plot in Altair
- Visualization: Scatter Plot with Rolling Mean in Altair
- Visualization: Histogram in Altair
- Visualization: Bar Plot in Altair
- Visualization: Stacked Histogram in Altair
- Visualization: Interactive Scatter Plot in Altair
- Jupyter Widgets
- Serving resources

Visualization: Linked Scatter-Plot and Histogram in Alt... Вставить

Altair selections can be used for a variety of things. This example shows a scatter plot and a histogram with selections over both that allow exploring the relationships between points

```
# load an example dataset
from vega_datasets import data
cars = data.cars()

import altair as alt

interval = alt.selection_interval()

points = alt.Chart(cars).mark_point().encode(
  x='Horsepower',
  y='Miles_per_Gallon',
  color=alt.condition(interval, 'Origin', alt.value('lightg
).properties(
  selection=interval
)

histogram = alt.Chart(cars).mark_bar().encode(
  x='count()',
  y='Origin',
  color='Origin'
).transform_filter(interval)

points & histogram
```

Посмотреть исходный блокнот

Product Gallery_vizual

Файл Изменить Вид Вставка Среда выполнения Инструменты Справка Последнее изменение: 24 марта

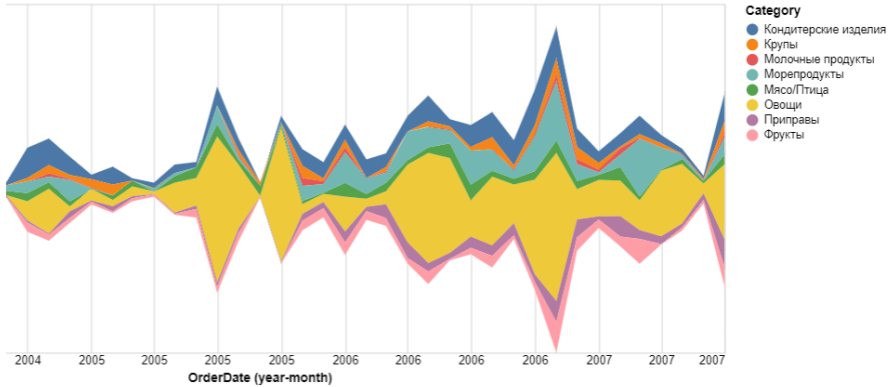
Комментировать Поделиться

+ Код + Текст

ОЗУ Диск

Редактирование

```
[ ]   alt.Chart(df).mark_area().encode(
          alt.X('yearmonth(OrderDate):T',
              axis=alt.Axis(format='%Y', domain=False, tickSize=0)
          ),
          alt.Y('sum(Profit):Q', stack='center', axis=None),
          alt.Color('Category:N',
      #         scale=alt.Scale(scheme='category20b')
          )
      ).properties(width =800).interactive()
```



Фрагменты кода

alt

Visualization: Linked Scatter-Plot and Histogr...
Visualization: Linked Brushing in Altair
Visualization: Interactive Brushing in Altair
Visualization: Time Series Line Plot in Altair
Visualization: Scatter Plot with Rolling Mean i...
Visualization: Histogram in Altair
Visualization: Bar Plot in Altair
Visualization: Stacked Histogram in Altair
Visualization: Interactive Scatter Plot in Altair
Jupyter Widgets
Serving resources

Visualization: Time Series Line Plo... Вставить

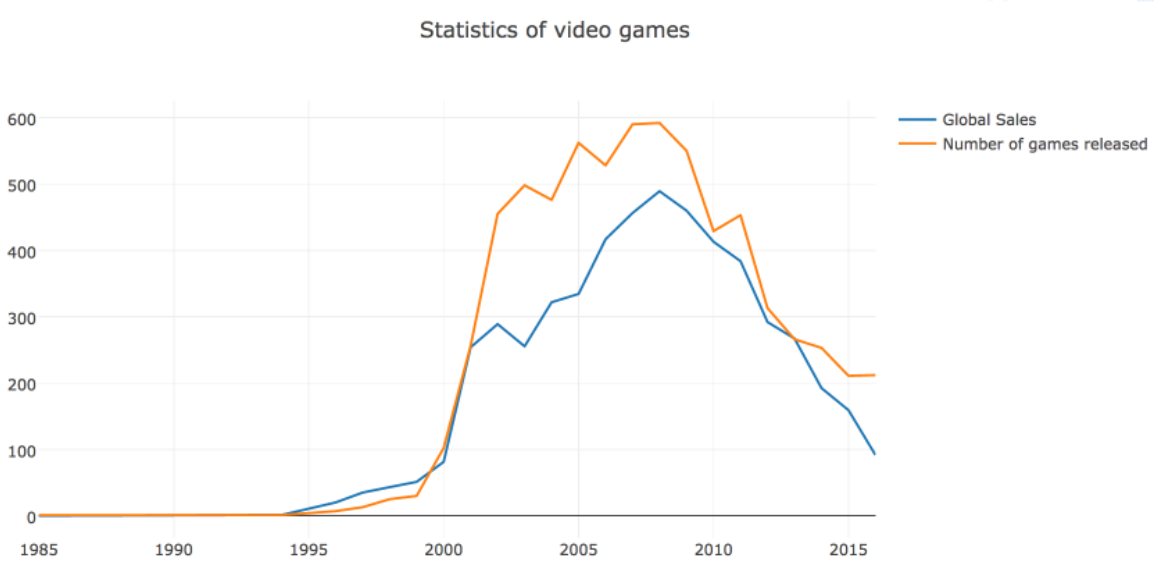
Altair handles temporal types natively by using the `:T` type marker. An example is in this plot of stock prices over time

```
from vega_datasets import data
stocks = data.stocks()

import altair as alt
alt.chart(stocks).mark_line().encode(
  x='date:T',
  y='price',
  color='symbol'
).interactive(bind_y=False)
```

Посмотреть исходный блокнот

Plotly

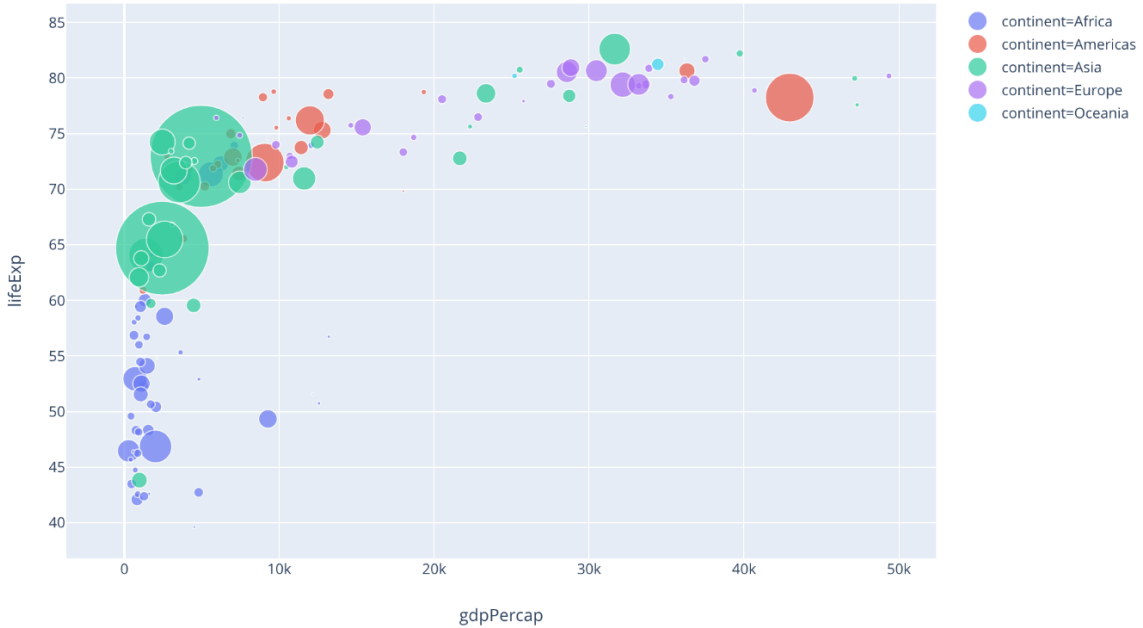


Познакомимся также с библиотекой plotly. **Plotly** — это open-source библиотека, которая позволяет строить интерактивные графики в jupyter.notebook'е без необходимости зарываться в javascript код.
Прелесть интерактивных графиков заключается в том, что можно посмотреть точное численное значение при наведении мыши, скрыть неинтересные ряды в визуализации, приблизить определенный участок графика и т.д.

Plotly Express

Plotly Express - это новая высокоуровневая библиотека визуализации Python: это оболочка для Plotly.py, которая предоставляет простой синтаксис для сложных диаграмм. Вдохновленный Seaborn и ggplot2, он был специально разработан для того, чтобы иметь краткий, непротиворечивый и простой в освоении API: с помощью всего одного импорта вы можете создавать красивые интерактивные графики всего за один вызов функции, включая фасетирование, карты, анимацию и линии тренда.

```
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent", size="pop", size_max=60)
```

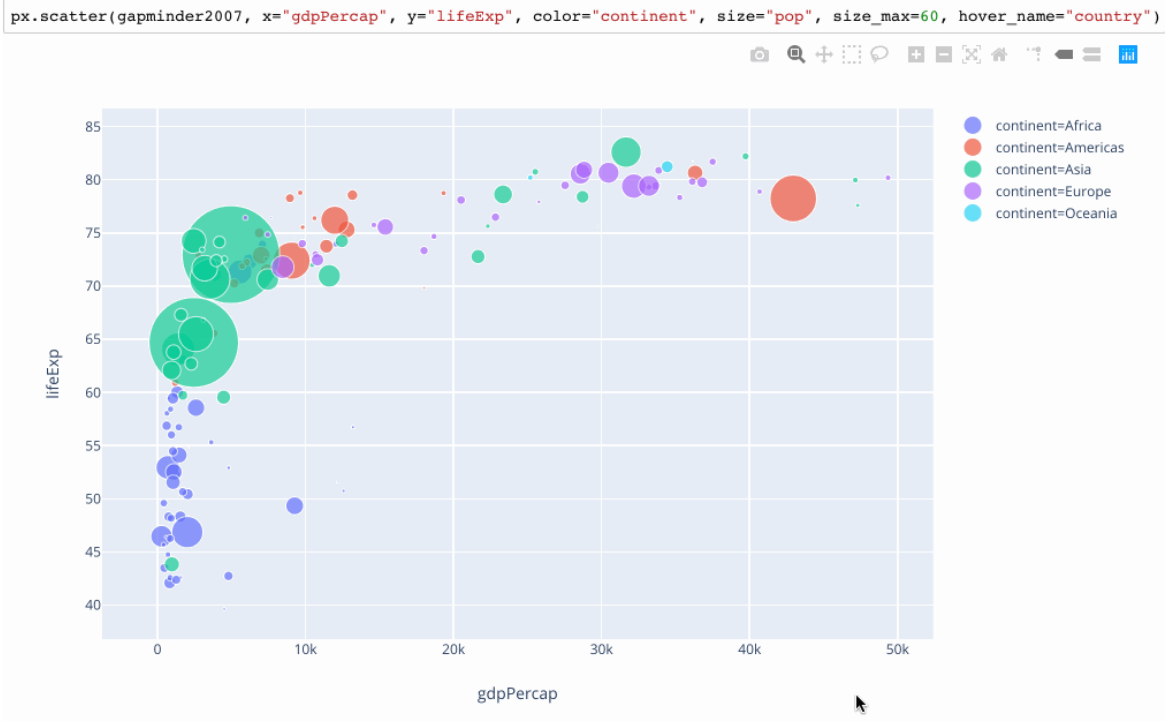


Plotly Express полностью бесплатна: с его бесплатной лицензией MIT с открытым исходным кодом вы можете использовать ее как угодно (да, даже в коммерческих продуктах!).

Plotly Express

Plotly Express полностью бесплатна : с его бесплатной лицензией MIT с открытым исходным кодом вы можете использовать ее как угодно (да, даже в коммерческих продуктах!). Лучше всего то, что Plotly Express полностью совместима с остальной экосистемой Plotly: используйте её в своем Dash приложении, экспортируйте свои фигуры практически в любой формат файла или отредактируйте их в графическом интерфейсе с помощью редактора диаграмм JupyterLab !

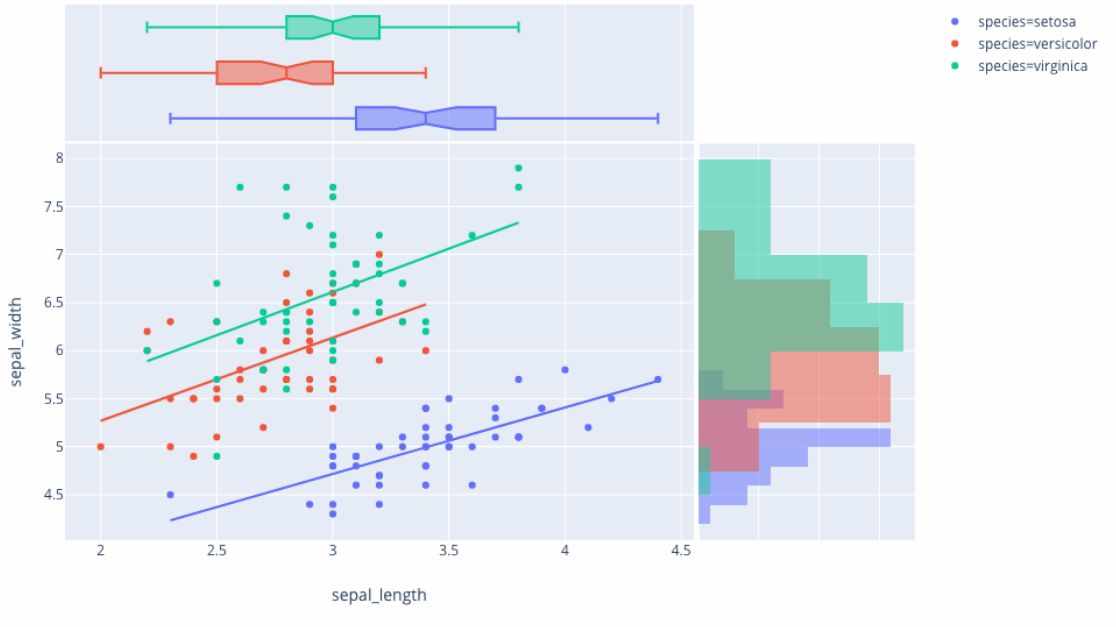
```
px.scatter(gapminder2007, x="gdpPercap", y="lifeExp", color="continent", size="pop", size_max=60, hover_name="country")
```



Plotly Express

Plotly Express поддерживает точечные и линейные графики в 3d, полярных и троичных координатах, а также в 2d координатах и на картах. Гистограммы доступны как в двухмерном декартовом, так и в полярном вариантах. Для визуализации распределений вы можете использовать гистограммы и прямоугольники или скрипки в одномерных настройках или контуры плотности для двумерных распределений. Большинство двумерных декартовых графиков принимают непрерывные или категориальные данные, а также автоматически обрабатывают данные даты и времени.

```
px.scatter(iris, x="sepal_width", y="sepal_length", color="species", marginal_y="histogram",
           marginal_x="box", trendline="ols")
```



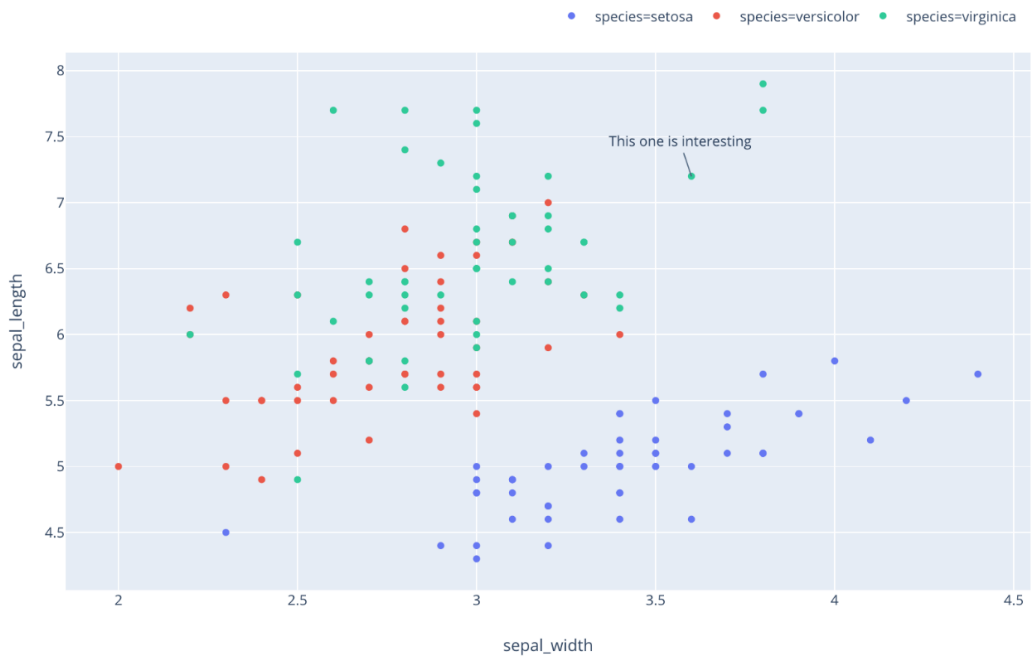
Plotly Express

Plotly Express является Plotly.py , как Seaborn является Matplotlib : оберткой высокого уровня , который позволяет быстро создавать фигуры, а затем использовать силу базового API и вносить изменения впоследствии.
Темы позволяют вам управлять настройками всей фигуры, такими как поля, шрифты, цвета фона, расположение тиков и многое другое.

Главной целью с Plotly Express было облегчить использование Plotly.py для исследования и быстрой итерации.

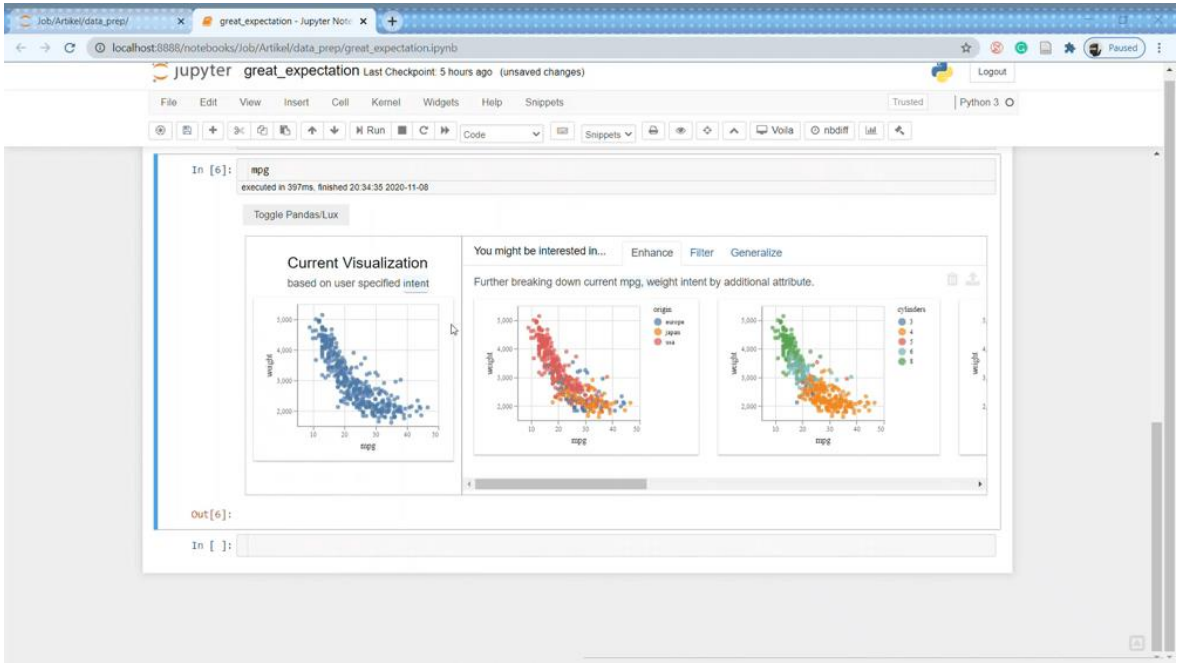
```
import plotly_express as px
fig = px.scatter(px.data.iris(), x="sepal_width", y="sepal_length", color="species")

import plotly.graph_objs as go
fig.update(layout=dict(
           legend=dict(orientation="h", y=1.1, x=0.5),
           annotations=[go.layout.Annotation(text="This one is interesting", x=3.6, y=7.2)]
          ))
```

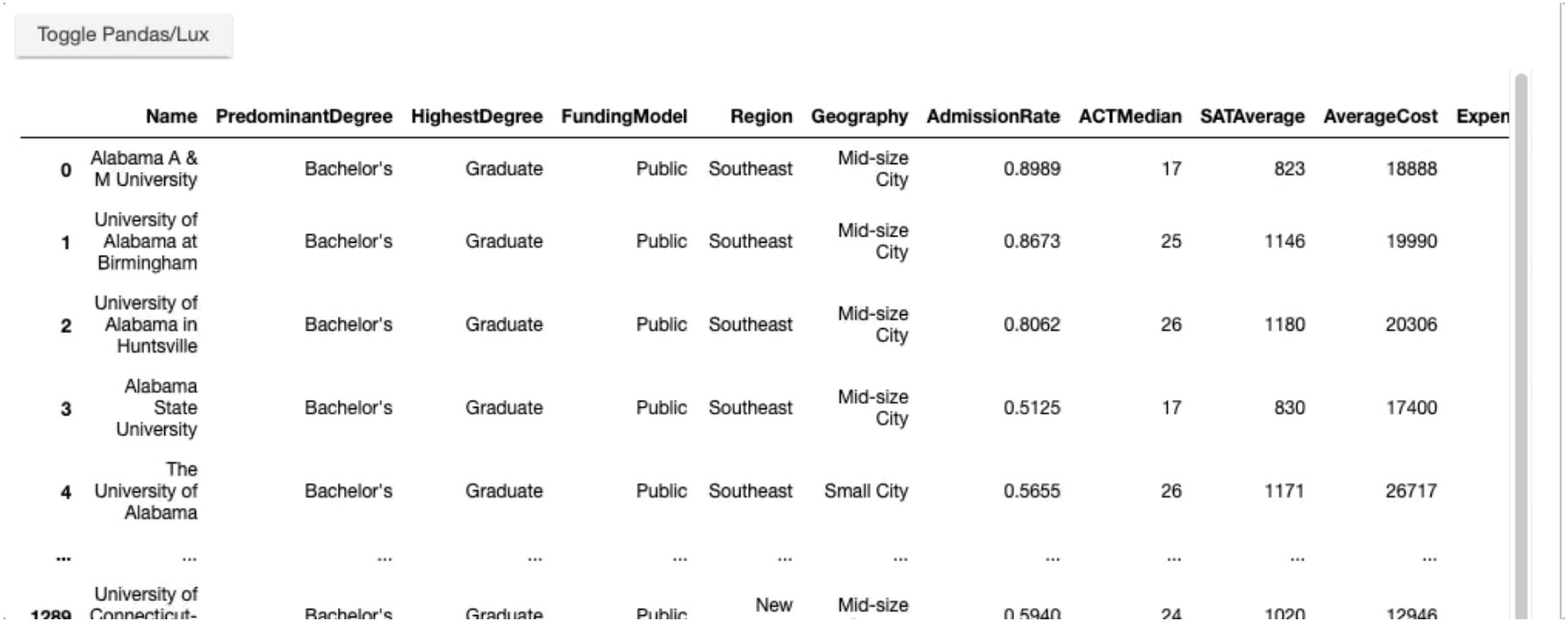


Lux

Быстрое исследование данных на основе рекомендаций с Lux. Lux — это пакет с открытым исходным кодом на Python, разработанный, чтобы помочь нам более разумно исследовать данные с помощью их рекомендаций. Lux предназначен для тесной интеграции с Pandas и может использоваться как есть, без изменения существующего кода Pandas. Lux сохраняет семантику фреймов данных Pandas — это означает, что вы можете применить любую команду из API Pandas к фреймам данных в Lux.



Чтобы визуализировать свой фрейм данных в Lux, просто распечатайте фрейм данных. Вы должны увидеть отображение таблицы Pandas по умолчанию с дополнительной кнопкой переключения. Нажав на кнопку переключить (Toggle), теперь вы можете просматривать данные визуально через Lux. Вы должны увидеть несколько категорий визуализаций, рекомендованных вам при просмотре различных вкладок.



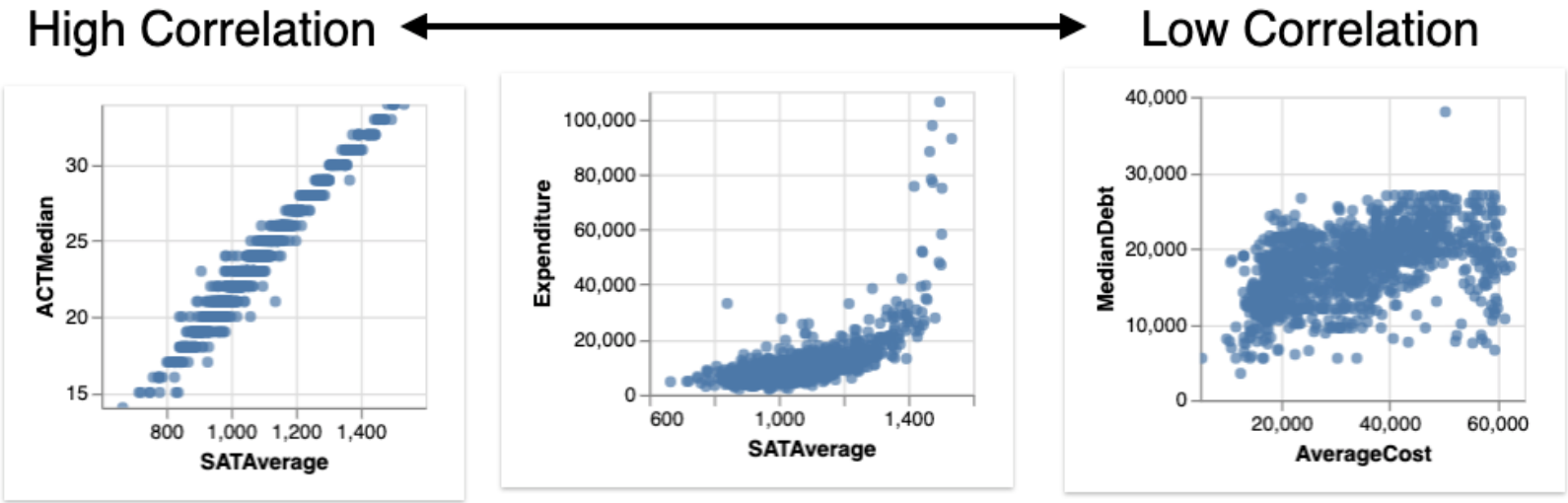
	Name	PredominantDegree	HighestDegree	FundingModel	Region	Geography	AdmissionRate	ACTMedian	SATAverage	AverageCost	Expen
0	Alabama A & M University	Bachelor's	Graduate	Public	Southeast	Mid-size City	0.8989	17	823	18888	
1	University of Alabama at Birmingham	Bachelor's	Graduate	Public	Southeast	Mid-size City	0.8673	25	1146	19990	
2	University of Alabama in Huntsville	Bachelor's	Graduate	Public	Southeast	Mid-size City	0.8062	26	1180	20306	
3	Alabama State University	Bachelor's	Graduate	Public	Southeast	Mid-size City	0.5125	17	830	17400	
4	The University of Alabama	Bachelor's	Graduate	Public	Southeast	Small City	0.5655	26	1171	26717	
...	...	...	...	...	...	...	...	...	...	...	
1289	University of Connecticut-	Bachelor's	Graduate	Public	New	Mid-size	0.5940	24	1020	12946	

Визуализация кадров данных с рекомендациями

Рекомендации выделяют интересные закономерности и тенденции в фрейме данных. Lux предлагает различные типы рекомендаций, известные как аналитические действия. Эти аналитические действия представляют собой различные виды анализа, которые можно выполнить с данными. Lux рекомендует набор действий в зависимости от содержимого вашего фрейма данных и ваших целей и интересов анализа (описанных ниже).

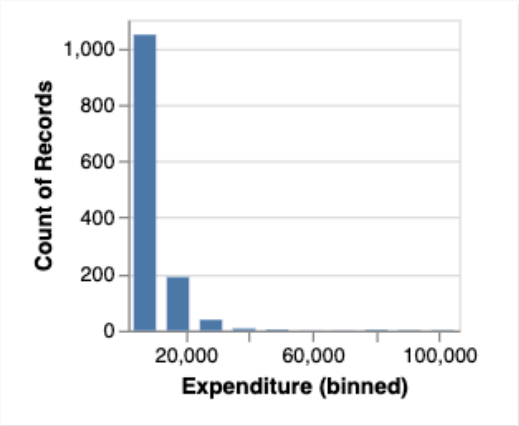
Как показано в приведенном выше примере, по умолчанию мы отображаем три типа действий, отображаемых в виде разных вкладок:

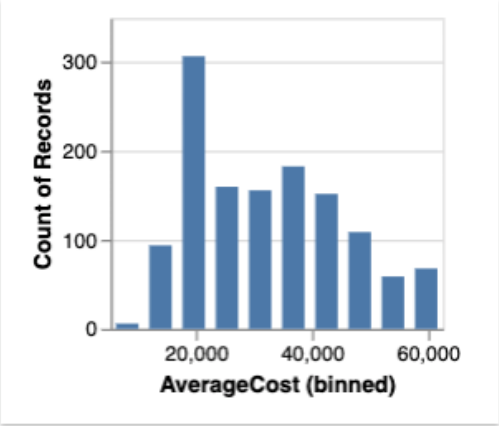
Корреляция отображает взаимосвязь между двумя количественными переменными, ранжированными диаграммами рассеяния от наиболее до наименее коррелированных.

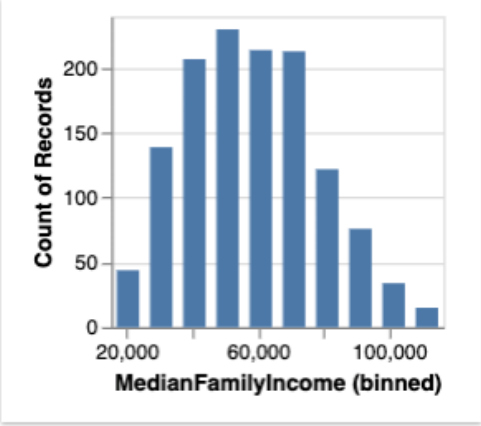


Распределение отображает гистограммы распределения различных количественных атрибутов в фрейме данных, ранжированные по распределению с наибольшей и наименьшей асимметрией.

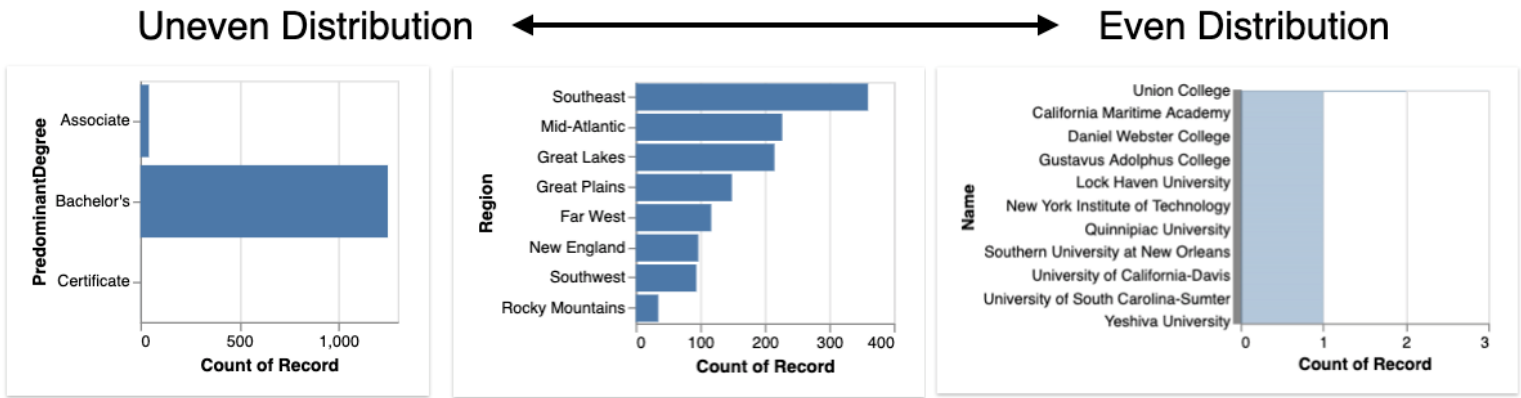
Skewed Distribution ⟷ Normal Distribution







Распространение отображает распределения гистограмм различных категориальных атрибутов в фрейме данных, ранжированные по гистограмме с наибольшей и наименьшей неравномерностью.



Выражение интереса и целей анализа с помощью намерения пользователя

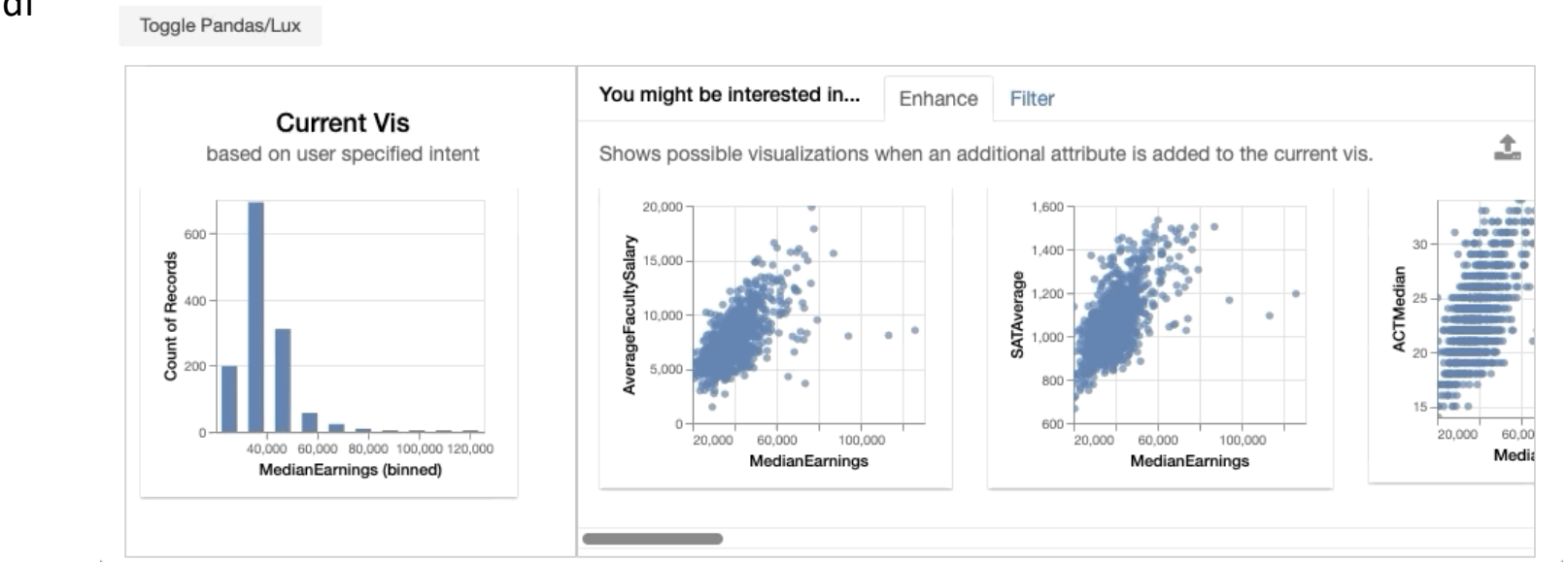
Мы видели пример того, как можно генерировать рекомендации для фрейма данных без предоставления дополнительной информации. Помимо этих основных рекомендаций, вы можете дополнительно указать свое намерение анализа , т. е. атрибуты данных и значения, которые вы хотите визуализировать. Например, предположим, что вы хотите узнать больше о среднем заработке студентов после поступления в колледж. Вы можете установить свое намерение в Lux, чтобы указать, что вас интересует атрибут MedianEarning.

df.intent = ["MedianEarnings"]

Можно указать различные признаки, которые вас заинтересовали, например, предположим, что вас интересует средний заработок студентов в колледжах, финансируемых государством.

df.intent = ["MedianEarnings", "FundingModel=Public"]

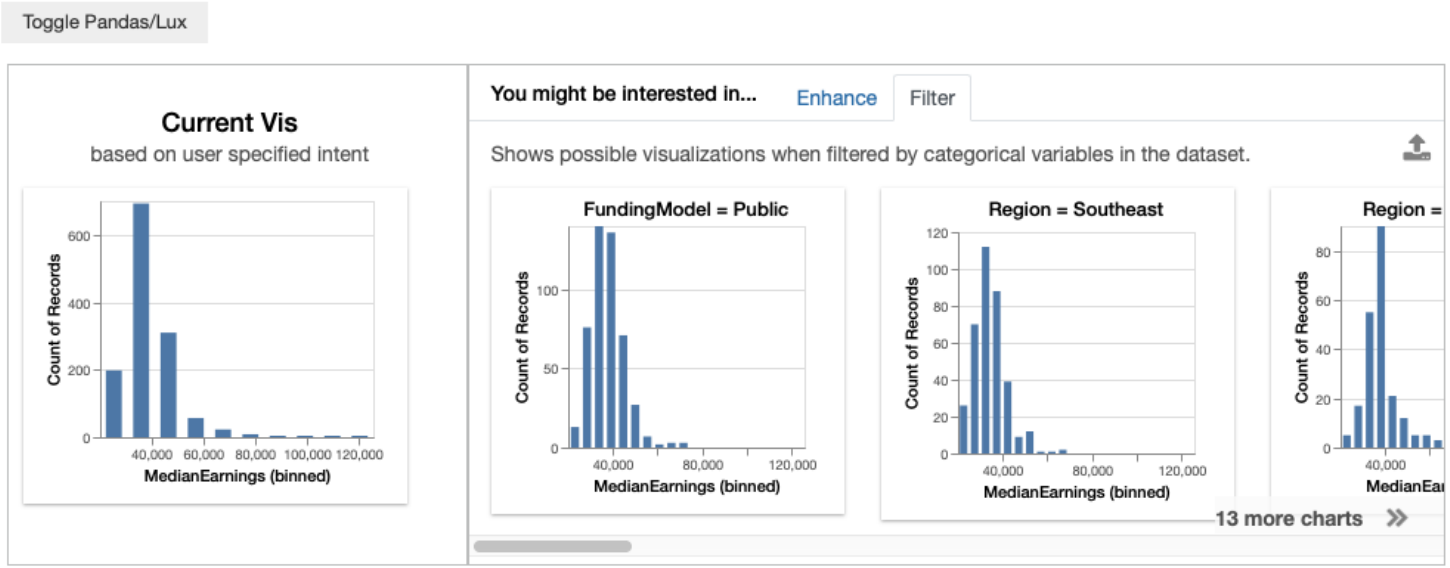
df



Рекомендации по управлению через намерение пользователя
В предыдущем примере, когда MedianEarning выражается как намерение, текущее намерение представляется df.intent = ["MedianEarnings"]
С учетом обновленного намерения генерируются дополнительные действия («Улучшить» и «Фильтровать»).

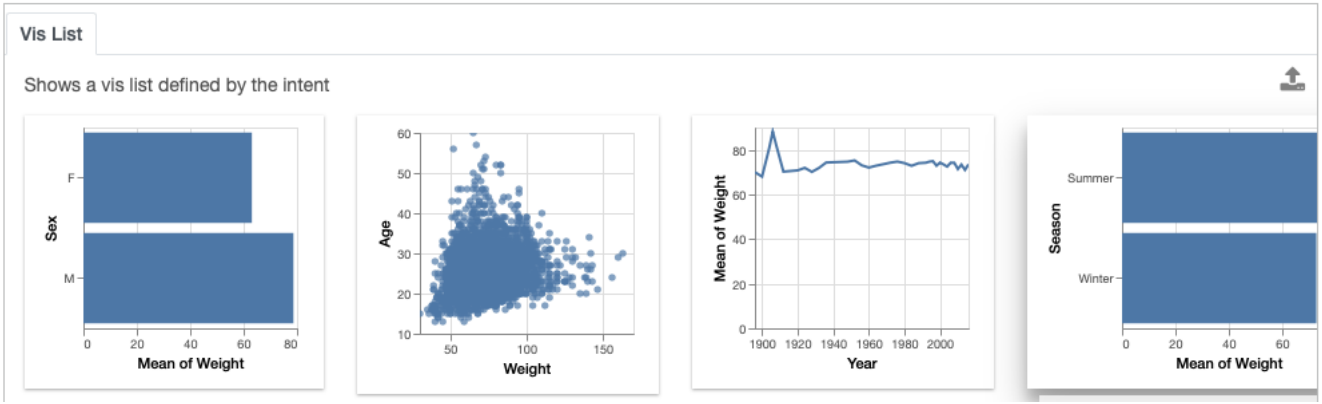
Улучшение добавляет дополнительный атрибут к предполагаемой визуализации. Улучшение позволяет пользователям сравнить влияние добавленной переменной на предполагаемую визуализацию. Например, улучшить отображает визуализацию, включающую C' = {MedianEarnings, added attribute}, в том числе: {MedianEarnings, Expenditure}, {MedianEarnings, AverageCost}, {MedianEarnings, AverageFacultySalary}.

Фильтр добавляет дополнительный фильтр к предполагаемой визуализации. Фильтр позволяет пользователям просмотреть, как выглядит предполагаемая визуализация для различных подмножеств данных. Например, фильтр отображает визуализации, включающие C' = {MedianEarnings, добавленный фильтр }, в том числе:
{MedianEarnings, FundingModel=Public }
{MedianEarnings, регион = юго-восток }
{MedianEarnings, регион = Великие озера }.
Итак, Lux — это система EDA, основанная на рекомендациях, которая помогает нам быстро понять наши данные. Пакет помогает нам, предоставляя нам все возможные комбинации данных и исследуя данные на основе нашего собственного намерения.



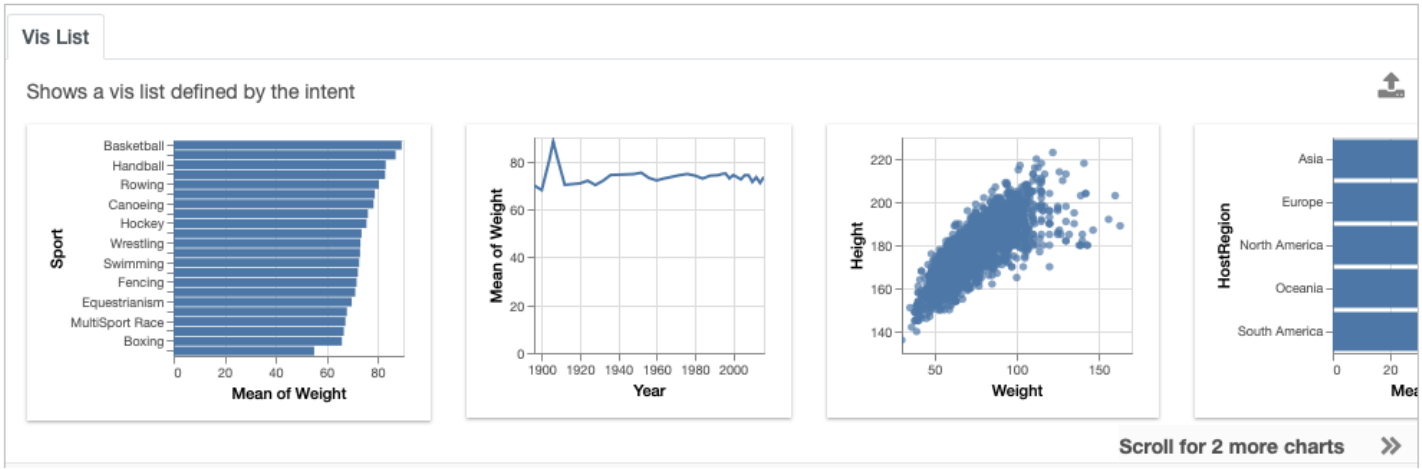
Часто пользователь может иметь в виду определенные визуализации, которые он хочет указать. В этом случае пользователи могут быстро определить свои собственные визуализации с помощью Lux и визуализировать свои данные по требованию.
Объект Vis представляет собой отдельную визуализацию, отображаемую в Lux, которая может быть сгенерирована автоматически или определена пользователем.
Чтобы сгенерировать Vis , пользователи должны указать свое намерение и исходный фрейм данных в качестве входных данных. Объекты Vis — это мощные программные представления визуализаций, которые можно экспортировать в код визуализации.
Здесь мы создаем коллекцию значений веса по отношению ко всем другим атрибутам, используя подстановочный знак «?» символ.

```
from lux.vis.VisList import VisList vc = VisList(["Weight","?"],df) vc
```



В качестве альтернативы мы можем указать желаемые атрибуты через список по отношению к Weight:

```
vc = VisList(["Weight",['Sport','Year','Height','HostRegion','SportType']],df) vc
```



Итак, LUX — это система исследовательского анализа данных (EDA), основанная на рекомендациях, которая помогает нам быстро обойти наши данные. Пакет предоставляет нам все возможные комбинации данных и визуализирует данные на основе наших требований.

AutoViz

В науке о данных чрезвычайно важны эффективные визуализации, которые предлагают яркое свидетельство сильных сезонных закономерностей, резких тенденций, связанных с успешной маркетинговой кампанией, или заметных выбросов, которые необходимо устранить. Визуализация для небольших наборов данных проста и очень полезна, но может быть практически невозможной для больших наборов данных с сотнями, если не тысячами переменных, где мы должны решить, какие идеи лучше всего выделить из набора данных.

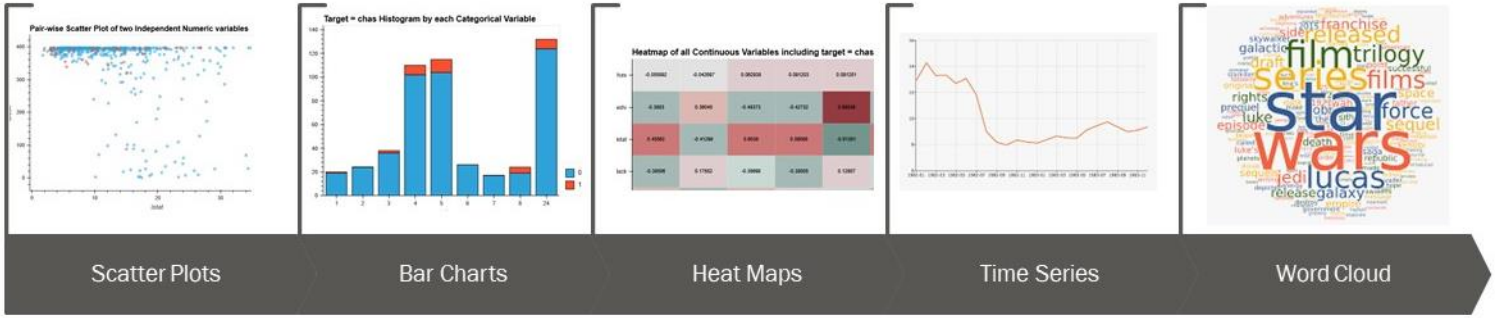
Кроме того, специалистам по обработке и анализу данных часто приходится использовать нестандартные библиотеки визуализации, которые требуют довольно много кода для получения визуальных эффектов с нужным эффектом. К счастью, существует альтернатива созданию визуализаций с использованием этого метода грубой силы.

AutoViz решает многие из вышеупомянутых проблем, которые могут возникнуть при выполнении работы по визуализации данных. Инструмент можно вызвать с помощью одной строки кода, передав ему либо объект фрейма данных pandas, либо необработанный CSV-файл для импорта. Если количество наблюдений велико, AutoViz выберет случайную выборку; аналогичным образом, если количество переменных велико (что вы можете решить), AutoViz может найти наиболее важные функции и построить впечатляющие визуализации только с использованием этих автоматически выбранных функций.

Пользователь может установить количество выборочных строк и максимальное количество объектов для визуализации, просто передав параметр в AutoViz. AutoViz способен адаптироваться к любому количеству различных контекстов данных, таких как регрессия, классификация или даже данные временных рядов.

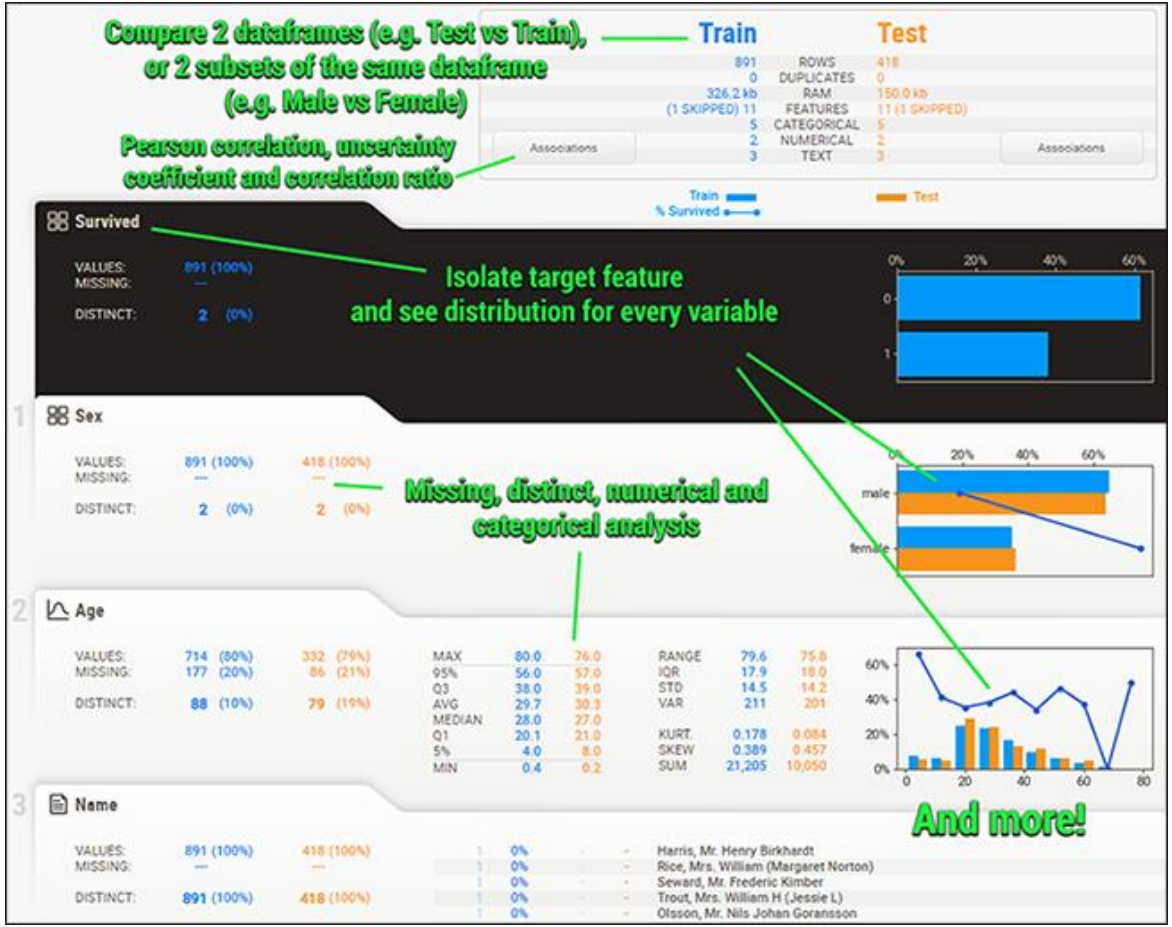
В подходе к визуализации с использованием **AutoViz есть много преимуществ**. Библиотека легко интерпретируется и может быть переведена в очень подробный режим с помощью флага verbose=1 или 2. Модель XGBoost неоднократно используется для определения наиболее согласованного набора функций, которые считаются важными, каждый раз используя случайный набор функций; затем наиболее заметные выбранные функции могут служить ориентиром для будущих графиков и визуализаций. Звучит так, будто это может занять время, но на самом деле все делается очень быстро. Чтобы сделать это эффективно, AutoViz классифицирует выбранные переменные как категориальные, числовые, логические, текстовые (НЛП) и др., чтобы понять, как лучше всего их отображать. Наконец, используя встроенную эвристику, инструмент возвращает визуальные эффекты, которые, как считается, оказывают наибольшее влияние. AutoViz также очень систематичен: он использует все выбранные переменные с различными типами диаграмм, чтобы обеспечить наилучшее понимание, позволяя диаграммам говорить самим за себя. Субъективное знание предметной области часто может привести к тому, что даже очень опытный специалист по данным выберет лишь несколько диаграмм, чтобы выделить информацию из набора данных. Объективный выбор функций и графиков AutoViz может указать группам данных на лучшие подходы с использованием систематической методологии и может значительно повысить производительность команды с самого начала проекта.

Как выглядит AutoViz на практике? Конечно, единственный способ понять библиотеку визуализации — это наблюдать за некоторыми ее графиками.



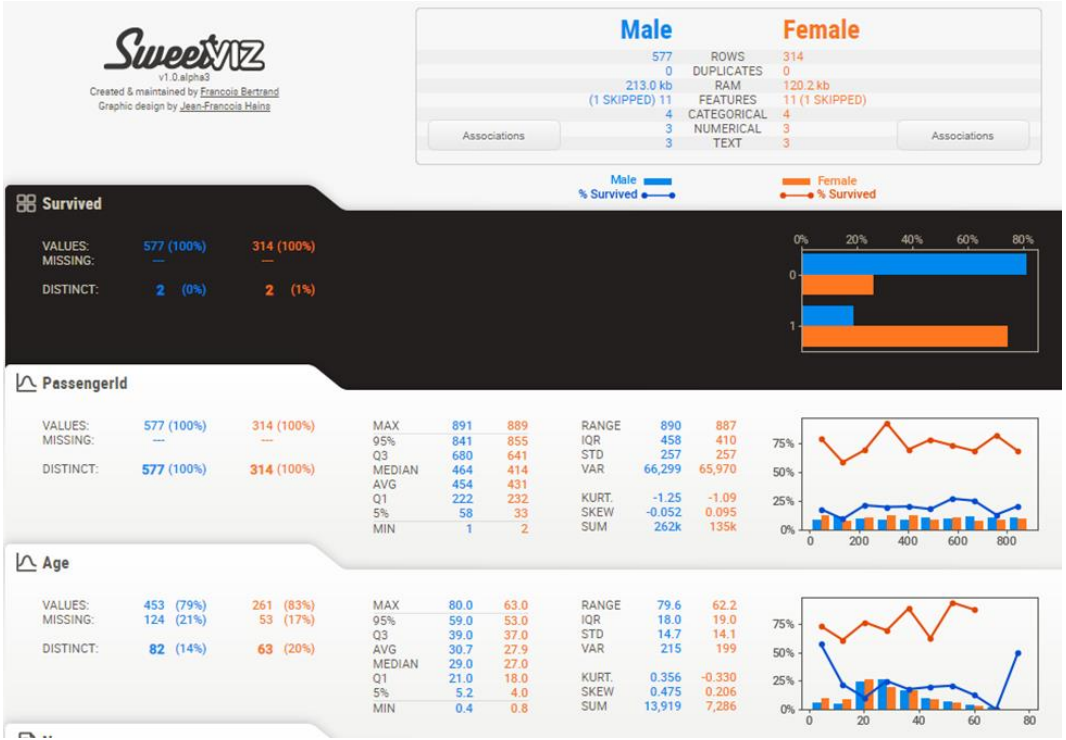
Sweetviz

Sweetviz — это библиотека Python с открытым исходным кодом, которая генерирует отчеты с удобной визуализацией для выполнения EDA с помощью всего двух строк кода. Библиотека позволяет быстро создать подробный отчет по всем характеристикам набора данных без особых усилий. В возможности Sweetviz также входит целевой анализ, сравнение двух датасетов, сравнение двух частей датасета, выделенных по определенному признаку, выявление корреляций и ассоциаций, также sweetviz создает позволяет создавать и сохранять отчет как HTML файл.



Некоторые особенности библиотеки **SweetViz**:

1. Показывает, как целевое значение (зависимые характеристики) соотносится с другими функциями.
2. Легко интегрирует ассоциации для числовых (корреляция Пирсона), категориальных (коэффициент неопределенности) и категориально-числовых (коэффициент корреляции) типов данных, чтобы предоставить максимум информации для всех типов данных.
3. Автоматически определяет числовые, категориальные и текстовые функции с дополнительными ручными настройками.
4. Сводная информация о типе, уникальных значениях, пропущенных значениях, повторяющихся строках, наиболее частых значениях.
5. Числовой анализ, такой как Мин./Макс./Диапазон, квартили, среднее значение, режим, стандартное отклонение, сумма, медианное абсолютное отклонение, коэффициент вариации, эксцесс, асимметрия.



dataprep

Пакет **dataprep.eda** упрощает разведочный анализ данных процесс, позволяя пользователю исследовать важные характеристики с помощью простых API. Каждый API позволяет пользователю анализировать набор данных от высокого уровня до низкого уровня и с разных точек зрения. В частности, dataprep.eda обеспечивает **следующие функции**:
Анализируйте распределения столбцов с помощью plot(). Функция plot() исследует распределения столбцов и статистику набора данных. Он определит тип столбца, а затем выведет различные графики и статистику, подходящие для соответствующего типа. Пользователь может опционально передать один или два интересующих столбца в качестве параметров: если передан один столбец, его распределение будет построено различными способами, и будет вычислена статистика столбца. Если переданы два столбца, будут созданы графики, изображающие взаимосвязь между двумя столбцами.

Проанализируйте корреляции с plot_correlation(). Функция plot_correlation() исследует корреляцию между столбцами различными способами и с использованием нескольких показателей корреляции. По умолчанию он строит корреляционные матрицы с различными метриками. Пользователь может дополнительно передать один или два интересующих столбца в качестве параметров: если передается один столбец, будет вычислена и ранжирована корреляция между этим столбцом и всеми другими столбцами. Если переданы два столбца, будут построены точечный график и линия регрессии.

[2]:
```
from dataprep.eda import plot_correlation
from dataprep.datasets import load_dataset
df = load_dataset("wine-quality-red")
plot_correlation(df)
```

[2]:

Stats | Pearson | Spearman | KendallTau

	Pearson	Spearman	KendallTau
Highest Positive Correlation	0.672	0.79	0.607
Highest Negative Correlation	-0.683	-0.707	-0.528
Lowest Correlation	0.002	0.001	0.0
Mean Correlation	0.019	0.028	0.021

Проанализируйте пропущенные значения с помощью plot_missing(). Эта функция plot_missing() обеспечивает тщательный анализ отсутствующих значений и их влияние на набор данных. По умолчанию он будет генерировать различные графики, которые отображают количество пропущенных значений для каждого столбца и любые основные закономерности пропущенных значений в наборе данных. Чтобы понять влияние отсутствующих значений в одном столбце на другие столбцы, пользователь может передать имя столбца в качестве параметра. Затем plot_missing() будет сгенерировано распределение каждого столбца с отсутствующими значениями из данного столбца и без них, что позволит полностью понять их влияние.

[3]:
```
from dataprep.eda import plot_missing
from dataprep.datasets import load_dataset
df = load_dataset("titanic")
plot_missing(df)
```

Computing isnull-ecf95c20b8939e415526a9cc2fcd1707: 0%

[3]:

Stats | Bar Chart | Spectrum | Heat Map | Dendrogram

Missing Statistics

Missing Cells	866
Missing Cells (%)	8.1%
Missing Columns	3
Missing Rows	708
Avg Missing Cells per Column	72.17
Avg Missing Cells per Row	0.97

Анализируйте различия столбцов с помощью plot_diff(). Функция plot_diff() исследует различия в распределении столбцов и статистике в нескольких наборах данных. Он определит тип столбца, а затем выведет различные графики и статистику, подходящие для соответствующего типа. Пользователь может дополнительно установить базовый уровень, который используется в качестве целевого набора данных для сравнения с другими наборами данных.

```
[4]: from dataprep.eda import plot_diff
     from dataprep.datasets import load_dataset
     df1 = load_dataset("house_prices_test")
     df2 = load_dataset("house_prices_train")
     plot_diff([df1, df2])
```

[4]: Show Stats