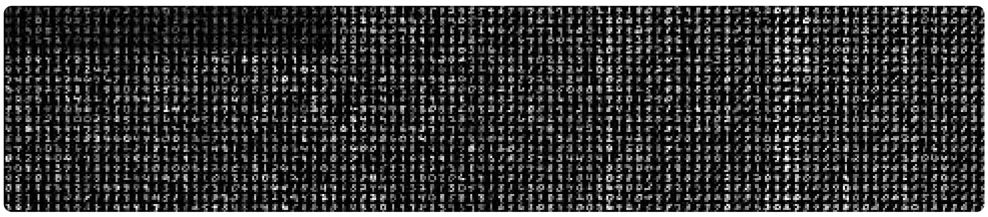
Digit-recognization

Recognizing the Handwritten Digit Images



MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

In this competition, your goal is to correctly identify digits from a dataset of tens of thousands of handwritten images. We’ve curated a set of tutorial-style kernels which cover everything from regression to neural networks. We encourage you to experiment with different algorithms to learn first-hand what works well and how techniques compare.

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

000 001 002 003 ... 026 027 028 029 030 031 ... 054 055 056 057 058 059 ... 082 083 | | | | ... | | 728 729 730 731 ... 754 755 756 757 758 759 ... 782 783 The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

Your submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit you predict. For example, if you predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then your submission file would look like:

ImageId,Label 1,3 2,7 3,8 (27997 more lines) The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that you have correctly classified all but 3% of the images.

```
!pip install jovian --upgrade --quiet
```

```
import jovian
```

```
# Execute this to save new versions of the notebook
jovian.commit(project="digit-recognization")
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Downloading the Dataset

```
!pip install pandas numpy sklearn matplotlib seaborn plotly
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (1.3.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.21.6)
Requirement already satisfied: sklearn in /usr/local/lib/python3.7/dist-packages (0.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages
(3.2.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/dist-packages
(0.11.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (5.5.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
(from pandas) (2022.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
python-dateutil>=2.7.3->pandas) (1.15.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-packages
(from sklearn) (1.0.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-
packages (from matplotlib) (1.4.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
(from matplotlib) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib) (3.0.9)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-
packages (from kiwisolver>=1.0.1->matplotlib) (4.2.0)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages
(from seaborn) (1.4.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-
packages (from plotly) (8.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
packages (from scikit-learn->sklearn) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages
(from scikit-learn->sklearn) (1.1.0)
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns
import plotly.express as px
%matplotlib inline
```

```
!pip install opendatasets
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: opendatasets in /usr/local/lib/python3.7/dist-packages
(0.1.22)
```

```
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (4.64.0)
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (1.5.12)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from
opendatasets) (7.1.2)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (1.15.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (2.23.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (1.24.3)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from
kaggle->opendatasets) (2022.5.18.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages
(from kaggle->opendatasets) (6.1.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-
packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
(from requests->kaggle->opendatasets) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-
packages (from requests->kaggle->opendatasets) (3.0.4)
```

```python
import opendatasets as od
```

```python
od.download('https://www.kaggle.com/competitions/digit-recognizer/data')
```

```
Skipping, found downloaded files in "./digit-recognizer" (use force=True to force
download)
```

```python
import os
```

```python
os.listdir()
```

```
['.config', 'digit-recognizer', 'sample_data']
```

```python
os.listdir('./digit-recognizer')
```

```
['test.csv', 'train.csv', 'sample_submission.csv']
```

```python
train_df=pd.read_csv('./digit-recognizer/train.csv')
```

```
test_df=pd.read_csv('./digit-recognizer/test.csv')
```

```
final_result=pd.read_csv('./digit-recognizer/sample_submission.csv')
```

```
train_df
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pix
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
41996	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
41997	7	0	0	0	0	0	0	0	0	0	...	0	0	0	
41998	6	0	0	0	0	0	0	0	0	0	...	0	0	0	
41999	9	0	0	0	0	0	0	0	0	0	...	0	0	0	

42000 rows × 785 columns

```
test_df
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
27995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
27996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
27997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
27998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
27999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

28000 rows × 784 columns

```
train_df.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	420

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2

8 rows × 785 columns

Preparing the Data for Training

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```
train_df.nunique().head(20)
```

```
label      10
pixel0      1
pixel1      1
pixel2      1
pixel3      1
pixel4      1
pixel5      1
pixel6      1
pixel7      1
pixel8      1
pixel9      1
pixel10     1
pixel11     1
pixel12     3
pixel13     3
pixel14     2
pixel15     2
pixel16     1
pixel17     1
pixel18     1
dtype: int64
```

```
train_df.corr()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pi
label	1.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.033424	0.025050	0.0
pixel0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
pixel779	0.006075	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	-0.000240	-0.000174	-0.0
pixel780	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel781	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel782	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
pixel783	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

785 rows × 785 columns

```
(train_df.isna().sum()!=0).sum()
```

```
0
```

```
train_df.isna().sum()
```

```
label       0
pixel0      0
pixel1      0
pixel2      0
pixel3      0
           ..
pixel779    0
pixel780    0
pixel781    0
pixel782    0
pixel783    0
Length: 785, dtype: int64
```

```
train_df.shape,test_df.shape
```

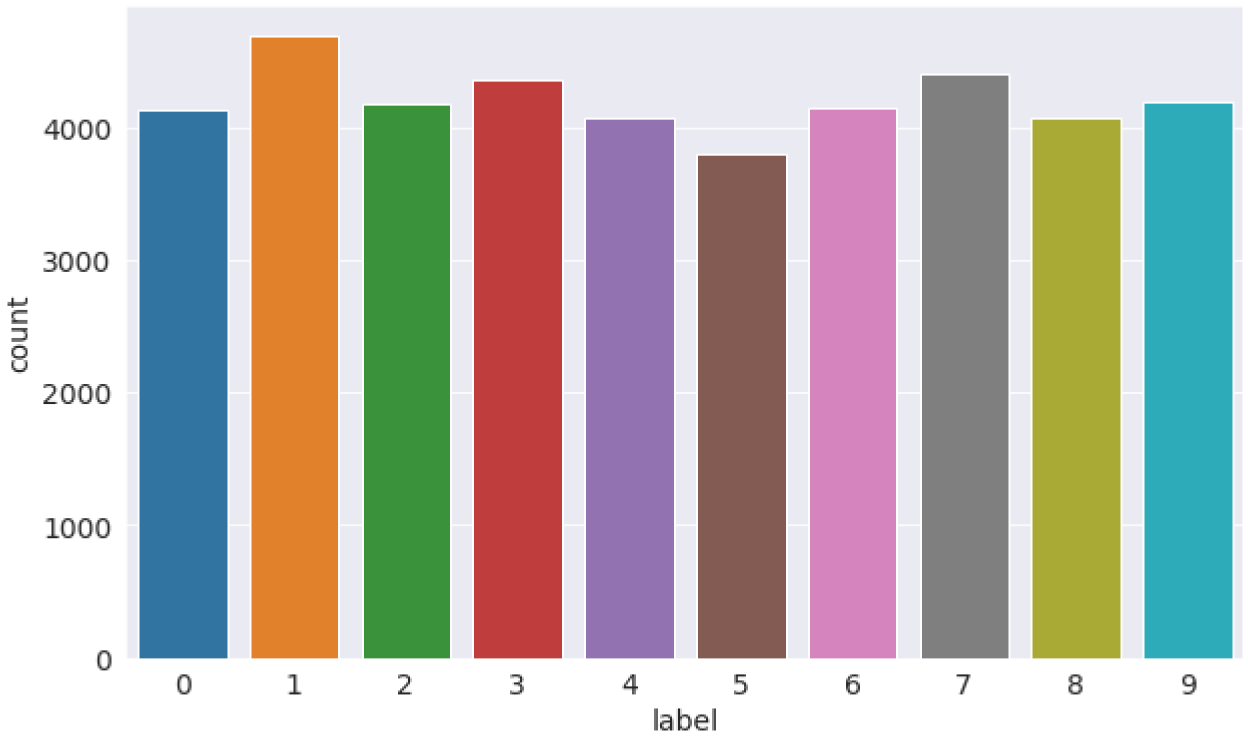
```
((42000, 785), (28000, 784))
```

```
order = list(np.sort(train_df['label'].unique()))
print(order)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
sns.set_style('darkgrid')
matplotlib.rcParams['font.size']=14
matplotlib.rcParams['figure.figsize']=(10,6)
matplotlib.rcParams['figure.facecolor']='#00000000'
sns.countplot(train_df.label)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.


<matplotlib.axes._subplots.AxesSubplot at 0x7fcada4998d0>
```



```
help(px)
```

```
Help on package plotly.express in plotly:

NAME
    plotly.express

DESCRIPTION
    `plotly.express` is a terse, consistent, high-level wrapper around
`plotly.graph_objects`
    for rapid data exploration and figure generation. Learn more at
https://plotly.express/

PACKAGE CONTENTS
    _chart_types
    _core
    _doc
    _imshow
    _special_inputs
    colors (package)
    data (package)
    imshow_utils
    trendline_functions (package)

CLASSES
    builtins.object
        plotly.express._special_inputs.Constant
        plotly.express._special_inputs.IdentityMap
        plotly.express._special_inputs.Range

    class Constant(builtins.object)
     |  Constant(value, label=None)
     |
     |  Objects of this class can be passed to Plotly Express functions that expect
column
     |  identifiers or list-like objects to indicate that this attribute should take on
a
     |  constant value. An optional label can be provided.
     |
     |  Methods defined here:
     |
     |  __init__(self, value, label=None)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class IdentityMap(builtins.object)
     |  `dict`-like object which acts as if the value for any key is the key itself.
Objects
     |  of this class can be passed in to arguments like `color_discrete_map` to
     |  use the provided data values as colors, rather than mapping them to colors
cycled
     |  from `color_discrete_sequence`. This works for any `_map` argument to Plotly
```

```
Express
     |  functions, such as `line_dash_map` and `symbol_map`.
     |
     |  Methods defined here:
     |
     |  __contains__(self, key)
     |
     |  __getitem__(self, key)
     |
     |  copy(self)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class Range(builtins.object)
     |  Range(label=None)
     |
     |  Objects of this class can be passed to Plotly Express functions that expect
column
     |  identifiers or list-like objects to indicate that this attribute should be
mapped
     |  onto integers starting at 0. An optional label can be provided.
     |
     |  Methods defined here:
     |
     |  __init__(self, label=None)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
```

```
FUNCTIONS
    area(data_frame=None, x=None, y=None, line_group=None, color=None, symbol=None,
hover_name=None, hover_data=None, custom_data=None, text=None, facet_row=None,
facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, symbol_sequence=None,
symbol_map=None, markers=False, orientation=None, groupnorm=None, log_x=False,
log_y=False, range_x=None, range_y=None, line_shape=None, title=None, template=None,
width=None, height=None)
            In a stacked area plot, each row of `data_frame` is represented as
            vertex of a polyline mark in 2D space. The area between successive
            polylines is filled.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        line_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            group rows of `data_frame` into lines.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        facet_row: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the vertical direction.
        facet_col: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the horizontal direction.
        facet_col_wrap: int
            Maximum number of facet columns. Wraps the column variable at this
            width, so that the column facets span multiple rows. Ignored if 0, and
            forced to 0 if `facet_row` or a `marginal` is set.
        facet_row_spacing: float between 0 and 1
            Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
            when facet_col_wrap is used.
        facet_col_spacing: float between 0 and 1
            Spacing between facet columns, in paper units Default is 0.02.
        animation_frame: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
```

```
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
markers: boolean (default `False`)
    If `True`, markers are shown on lines.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
groupnorm: str (default `None`)
    One of `'fraction'` or `'percent'`. If `'fraction'`, the value of each
    point is divided by the sum of all values at that location coordinate.
    `'percent'` is the same but multiplied by 100 to show percentages.
    `None` will stack up all values at each location coordinate.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
line_shape: str (default `'linear'`)
    One of `'linear'` or `'spline'`.
title: str
    The figure title.
template: str or dict or plotly.graph_objects.layout.Template instance
    The figure template name (must be a key in plotly.io.templates) or
    definition.
width: int (default `None`)
    The figure width in pixels.
height: int (default `None`)
    The figure height in pixels.

Returns
-------
    plotly.graph_objects.Figure
    bar(data_frame=None, x=None, y=None, color=None, pattern_shape=None,
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, hover_name=None, hover_data=None, custom_data=None, text=None,
base=None, error_x=None, error_x_minus=None, error_y=None, error_y_minus=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, color_continuous_scale=None,
pattern_shape_sequence=None, pattern_shape_map=None, range_color=None,
color_continuous_midpoint=None, opacity=None, orientation=None, barmode='relative',
log_x=False, log_y=False, range_x=None, range_y=None, text_auto=False, title=None,
template=None, width=None, height=None)
            In a bar plot, each row of `data_frame` is represented as a rectangular
            mark.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        pattern_shape: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign pattern shapes to marks.
```

```
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
base: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position the base of the bar.
```

```
error_x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars. If `error_x_minus` is `None`, error bars will
    be symmetrical, otherwise `error_x` is used for the positive direction
    only.
error_x_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars in the negative direction. Ignored if `error_x`
    is `None`.
error_y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars. If `error_y_minus` is `None`, error bars will
    be symmetrical, otherwise `error_y` is used for the positive direction
    only.
error_y_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars in the negative direction. Ignored if `error_y`
    is `None`.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
pattern_shape_sequence: list of str
    Strings should define valid plotly.js patterns-shapes. When
    `pattern_shape` is set, values in that column are assigned patterns-
    shapes by cycling through `pattern_shape_sequence` in the order
    described in `category_orders`, unless the value of `pattern_shape` is
    a key in `pattern_shape_map`.
pattern_shape_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js patterns-shapes. Used to override
    `pattern_shape_sequences` to assign a specific patterns-shapes to lines
    corresponding with specific values. Keys in `pattern_shape_map` should
    be values in the column denoted by `pattern_shape`. Alternatively, if
    the values of `pattern_shape` are valid patterns-shapes names, the
    string `'identity'` may be passed to cause them to be used directly.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
```

```
            `color_continuous_scale`.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
            (default `'v'` if `x` and `y` are provided and both continous or both
            categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
            `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
            provided)
        barmode: str (default `'relative'`)
            One of `'group'`, `'overlay'` or `'relative'` In `'relative'` mode,
            bars are stacked above zero for positive values and below zero for
            negative values. In `'overlay'` mode, bars are drawn on top of one
            another. In `'group'` mode, bars are placed beside each other.
        log_x: boolean (default `False`)
            If `True`, the x-axis is log-scaled in cartesian coordinates.
        log_y: boolean (default `False`)
            If `True`, the y-axis is log-scaled in cartesian coordinates.
        range_x: list of two numbers
            If provided, overrides auto-scaling on the x-axis in cartesian
            coordinates.
        range_y: list of two numbers
            If provided, overrides auto-scaling on the y-axis in cartesian
            coordinates.
        text_auto: bool or string (default `False`)
            If `True` or a string, the x or y or z values will be displayed as
            text, depending on the orientation A string like `'.2f'` will be
            interpreted as a `texttemplate` numeric formatting directive.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure
```

    bar_polar(data_frame=None, r=None, theta=None, color=None, pattern_shape=None,
hover_name=None, hover_data=None, custom_data=None, base=None, animation_frame=None,

```
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, color_continuous_scale=None, pattern_shape_sequence=None,
pattern_shape_map=None, range_color=None, color_continuous_midpoint=None, barnorm=None,
barmode='relative', direction='clockwise', start_angle=90, range_r=None,
range_theta=None, log_r=False, title=None, template=None, width=None, height=None)
            In a polar bar plot, each row of `data_frame` is represented as a wedge
            mark in polar coordinates.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        r: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the radial axis in polar coordinates.
        theta: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the angular axis in polar coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        pattern_shape: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign pattern shapes to marks.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
```

```
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
base: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position the base of the bar.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
pattern_shape_sequence: list of str
    Strings should define valid plotly.js patterns-shapes. When
    `pattern_shape` is set, values in that column are assigned patterns-
    shapes by cycling through `pattern_shape_sequence` in the order
    described in `category_orders`, unless the value of `pattern_shape` is
    a key in `pattern_shape_map`.
pattern_shape_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js patterns-shapes. Used to override
    `pattern_shape_sequences` to assign a specific patterns-shapes to lines
    corresponding with specific values. Keys in `pattern_shape_map` should
    be values in the column denoted by `pattern_shape`. Alternatively, if
    the values of `pattern_shape` are valid patterns-shapes names, the
    string `'identity'` may be passed to cause them to be used directly.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
barnorm: str (default `None`)
    One of `'fraction'` or `'percent'`. If `'fraction'`, the value of each
    bar is divided by the sum of all values at that location coordinate.
    `'percent'` is the same but multiplied by 100 to show percentages.
    `None` will stack up all values at each location coordinate.
barmode: str (default `'relative'`)
    One of `'group'`, `'overlay'` or `'relative'` In `'relative'` mode,
    bars are stacked above zero for positive values and below zero for
    negative values. In `'overlay'` mode, bars are drawn on top of one
    another. In `'group'` mode, bars are placed beside each other.
direction: str
```

```
            One of '`counterclockwise'` or `'clockwise'`. Default is `'clockwise'`
            Sets the direction in which increasing values of the angular axis are
            drawn.
        start_angle: int (default `90`)
            Sets start angle for the angular axis, with 0 being due east and 90
            being due north.
        range_r: list of two numbers
            If provided, overrides auto-scaling on the radial axis in polar
            coordinates.
        range_theta: list of two numbers
            If provided, overrides auto-scaling on the angular axis in polar
            coordinates.
        log_r: boolean (default `False`)
            If `True`, the radial axis is log-scaled in polar coordinates.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    box(data_frame=None, x=None, y=None, color=None, facet_row=None, facet_col=None,
facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None,
hover_data=None, custom_data=None, animation_frame=None, animation_group=None,
category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, orientation=None, boxmode=None, log_x=False, log_y=False,
range_x=None, range_y=None, points=None, notched=False, title=None, template=None,
width=None, height=None)
            In a box plot, rows of `data_frame` are grouped together into a
            box-and-whisker mark to visualize their distribution.

            Each box spans from quartile 1 (Q1) to quartile 3 (Q3). The second
            quartile (Q2) is marked by a line inside the box. By default, the
            whiskers correspond to the box' edges +/- 1.5 times the interquartile
            range (IQR: Q3-Q1), see "points" for other options.
```

```
Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the x axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
boxmode: str (default `'group'`)
    One of `'group'` or `'overlay'` In `'overlay'` mode, boxes are on drawn
    top of one another. In `'group'` mode, boxes are placed beside each
    other.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
points: str or boolean (default `'outliers'`)
    One of `'outliers'`, `'suspectedoutliers'`, `'all'`, or `False`. If
    `'outliers'`, only the sample points lying outside the whiskers are
    shown. If `'suspectedoutliers'`, all outlier points are shown and those
    less than 4*Q1-3*Q3 or greater than 4*Q3-3*Q1 are highlighted with the
    marker's `'outliercolor'`. If `'outliers'`, only the sample points
    lying outside the whiskers are shown. If `'all'`, all sample points are
    shown. If `False`, no sample points are shown and the whiskers extend
    to the full range of the sample.
notched: boolean (default `False`)
    If `True`, boxes are drawn with notches.
title: str
```

```
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure
```

    choropleth(data_frame=None, lat=None, lon=None, locations=None, locationmode=None, geojson=None, featureidkey=None, color=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None, hover_data=None, custom_data=None, animation_frame=None, animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None, color_discrete_map=None, color_continuous_scale=None, range_color=None, color_continuous_midpoint=None, projection=None, scope=None, center=None, fitbounds=None, basemap_visible=None, title=None, template=None, width=None, height=None)

```
            In a choropleth map, each row of `data_frame` is represented by a
            colored region mark on a map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        lat: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to latitude on a map.
        lon: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to longitude on a map.
        locations: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are to be
```

```
    interpreted according to `locationmode` and mapped to
    longitude/latitude.
locationmode: str
    One of 'ISO-3', 'USA-states', or 'country names' Determines the set of
    locations used to match entries in `locations` to regions on the map.
geojson: GeoJSON-formatted dict
    Must contain a Polygon feature collection, with IDs, which are
    references from `locations`.
featureidkey: str (default: `'id'`)
    Path to field in GeoJSON feature object with which to match the values
    passed in to `locations`.The most common alternative to the default is
    of the form `'properties.<key>`.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
projection: str
    One of `'equirectangular'`, `'mercator'`, `'orthographic'`, `'natural
    earth'`, `'kavrayskiy7'`, `'miller'`, `'robinson'`, `'eckert4'`,
    `'azimuthal equal area'`, `'azimuthal equidistant'`, `'conic equal
    area'`, `'conic conformal'`, `'conic equidistant'`, `'gnomonic'`,
    `'stereographic'`, `'mollweide'`, `'hammer'`, `'transverse mercator'`,
    `'albers usa'`, `'winkel tripel'`, `'aitoff'`, or `'sinusoidal'`Default
    depends on `scope`.
scope: str (default `'world'`).
    One of `'world'`, `'usa'`, `'europe'`, `'asia'`, `'africa'`, `'north
    america'`, or `'south america'`Default is `'world'` unless `projection`
    is set to `'albers usa'`, which forces `'usa'`.
center: dict
    Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
fitbounds: str (default `False`).
    One of `False`, `locations` or `geojson`.
basemap_visible: bool
    Force the basemap visibility.
title: str
    The figure title.
template: str or dict or plotly.graph_objects.layout.Template instance
    The figure template name (must be a key in plotly.io.templates) or
    definition.
width: int (default `None`)
    The figure width in pixels.
height: int (default `None`)
```

```
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    choropleth_mapbox(data_frame=None, geojson=None, featureidkey=None, locations=None,
color=None, hover_name=None, hover_data=None, custom_data=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, opacity=None, zoom=8, center=None, mapbox_style=None,
title=None, template=None, width=None, height=None)
            In a Mapbox choropleth map, each row of `data_frame` is represented by a
            colored region on a Mapbox map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        geojson: GeoJSON-formatted dict
            Must contain a Polygon feature collection, with IDs, which are
            references from `locations`.
        featureidkey: str (default: `'id'`)
            Path to field in GeoJSON feature object with which to match the values
            passed in to `locations`.The most common alternative to the default is
            of the form `'properties.<key>`.
        locations: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are to be
            interpreted according to `locationmode` and mapped to
            longitude/latitude.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
        animation_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            provide object-constancy across animation frames: rows with matching
            `animation_group`s will be treated as if they describe the same object
            in each frame.
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
            names, and the values should be lists of strings corresponding to the
            specific display order desired.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        zoom: int (default `8`)
            Between 0 and 20. Sets map zoom level.
        center: dict
            Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
        mapbox_style: str (default `'basic'`, needs Mapbox API token)
            Identifier of base map style, some of which require a Mapbox API token
            to be set using `plotly.express.set_mapbox_access_token()`. Allowed
            values which do not require a Mapbox API token are `'open-street-map'`,
            `'white-bg'`, `'carto-positron'`, `'carto-darkmatter'`, `'stamen-
            terrain'`, `'stamen-toner'`, `'stamen-watercolor'`. Allowed values
            which do require a Mapbox API token are `'basic'`, `'streets'`,
            `'outdoors'`, `'light'`, `'dark'`, `'satellite'`, `'satellite-
            streets'`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
```

```
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    density_contour(data_frame=None, x=None, y=None, z=None, color=None,
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, hover_name=None, hover_data=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, orientation=None,
color_discrete_sequence=None, color_discrete_map=None, marginal_x=None,
marginal_y=None, trendline=None, trendline_options=None, trendline_color_override=None,
trendline_scope='trace', log_x=False, log_y=False, range_x=None, range_y=None,
histfunc=None, histnorm=None, nbinsx=None, nbinsy=None, text_auto=False, title=None,
template=None, width=None, height=None)
            In a density contour plot, rows of `data_frame` are grouped together
            into contour marks to visualize the 2D distribution of an aggregate
            function `histfunc` (e.g. the count or sum) of the value `z`.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        z: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the z axis in cartesian coordinates. For
            `density_heatmap` and `density_contour` these values are used as the
            inputs to `histfunc`.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        facet_row: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the vertical direction.
        facet_col: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the horizontal direction.
        facet_col_wrap: int
            Maximum number of facet columns. Wraps the column variable at this
            width, so that the column facets span multiple rows. Ignored if 0, and
            forced to 0 if `facet_row` or a `marginal` is set.
        facet_row_spacing: float between 0 and 1
            Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
            when facet_col_wrap is used.
        facet_col_spacing: float between 0 and 1
            Spacing between facet columns, in paper units Default is 0.02.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
```

```
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
marginal_x: str
    One of `'rug'`, `'box'`, `'violin'`, or `'histogram'`. If set, a
    horizontal subplot is drawn above the main plot, visualizing the
    x-distribution.
```

```
marginal_y: str
    One of `'rug'`, `'box'`, `'violin'`, or `'histogram'`. If set, a
    vertical subplot is drawn to the right of the main plot, visualizing
    the y-distribution.
trendline: str
    One of `'ols'`, `'lowess'`, `'rolling'`, `'expanding'` or `'ewm'`. If
    `'ols'`, an Ordinary Least Squares regression line will be drawn for
    each discrete-color/symbol group. If `'lowess'`, a Locally Weighted
    Scatterplot Smoothing line will be drawn for each discrete-color/symbol
    group. If `'rolling'`, a Rolling (e.g. rolling average, rolling median)
    line will be drawn for each discrete-color/symbol group. If
    `'expanding'`, an Expanding (e.g. expanding average, expanding sum)
    line will be drawn for each discrete-color/symbol group. If `'ewm'`, an
    Exponentially Weighted Moment (e.g. exponentially-weighted moving
    average) line will be drawn for each discrete-color/symbol group. See
    the docstrings for the functions in
    `plotly.express.trendline_functions` for more details on these
    functions and how to configure them with the `trendline_options`
    argument.
trendline_options: dict
    Options passed as the first argument to the function from
    `plotly.express.trendline_functions`  named in the `trendline`
    argument.
trendline_color_override: str
    Valid CSS color. If provided, and if `trendline` is set, all trendlines
    will be drawn in this color rather than in the same color as the traces
    from which they draw their inputs.
trendline_scope: str (one of `'trace'` or `'overall'`, default `'trace'`)
    If `'trace'`, then one trendline is drawn per trace (i.e. per color,
    symbol, facet, animation frame etc) and if `'overall'` then one
    trendline is computed for the entire dataset, and replicated across all
    facets.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
histfunc: str (default `'count'` if no arguments are provided, else `'sum'`)
```

```
        One of ``'count'`, ``'sum'`, ``'avg'`, ``'min'`, or ``'max'`.Function used
        to aggregate values for summarization (note: can be normalized with
        `histnorm`). The arguments to this function are the values of `z`.
    histnorm: str (default `None`)
        One of ``'percent'`, ``'probability'`, ``'density'`, or ``'probability
        density'` If `None`, the output of `histfunc` is used as is. If
        ``'probability'`, the output of `histfunc` for a given bin is divided by
        the sum of the output of `histfunc` for all bins. If ``'percent'`, the
        output of `histfunc` for a given bin is divided by the sum of the
        output of `histfunc` for all bins and multiplied by 100. If
        ``'density'`, the output of `histfunc` for a given bin is divided by the
        size of the bin. If ``'probability density'`, the output of `histfunc`
        for a given bin is normalized such that it corresponds to the
        probability that a random event whose distribution is described by the
        output of `histfunc` will fall into that bin.
    nbinsx: int
        Positive integer. Sets the number of bins along the x axis.
    nbinsy: int
        Positive integer. Sets the number of bins along the y axis.
    text_auto: bool or string (default `False`)
        If `True` or a string, the x or y or z values will be displayed as
        text, depending on the orientation A string like ``'.2f'` will be
        interpreted as a `texttemplate` numeric formatting directive.
    title: str
        The figure title.
    template: str or dict or plotly.graph_objects.layout.Template instance
        The figure template name (must be a key in plotly.io.templates) or
        definition.
    width: int (default `None`)
        The figure width in pixels.
    height: int (default `None`)
        The figure height in pixels.

    Returns
    -------
        plotly.graph_objects.Figure
```

    density_heatmap(data_frame=None, x=None, y=None, z=None, facet_row=None,
facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,
hover_name=None, hover_data=None, animation_frame=None, animation_group=None,
category_orders=None, labels=None, orientation=None, color_continuous_scale=None,
range_color=None, color_continuous_midpoint=None, marginal_x=None, marginal_y=None,
opacity=None, log_x=False, log_y=False, range_x=None, range_y=None, histfunc=None,

```
histnorm=None, nbinsx=None, nbinsy=None, text_auto=False, title=None, template=None,
width=None, height=None)
            In a density heatmap, rows of `data_frame` are grouped together into
            colored rectangular tiles to visualize the 2D distribution of an
            aggregate function `histfunc` (e.g. the count or sum) of the value `z`.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        z: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the z axis in cartesian coordinates. For
            `density_heatmap` and `density_contour` these values are used as the
            inputs to `histfunc`.
        facet_row: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the vertical direction.
        facet_col: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the horizontal direction.
        facet_col_wrap: int
            Maximum number of facet columns. Wraps the column variable at this
```

```
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
```

```
orientation: str, one of ``'h'`` for horizontal or ``'v'`` for vertical.
    (default ``'v'`` if `x` and `y` are provided and both continous or both
    categorical,  otherwise ``'v'``(``'h'``) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise ``'v'``(``'h'``) if only `x`(`y`) is
    provided)
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
marginal_x: str
    One of ``'rug'``, ``'box'``, ``'violin'``, or ``'histogram'``. If set, a
    horizontal subplot is drawn above the main plot, visualizing the
    x-distribution.
marginal_y: str
    One of ``'rug'``, ``'box'``, ``'violin'``, or ``'histogram'``. If set, a
    vertical subplot is drawn to the right of the main plot, visualizing
    the y-distribution.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
histfunc: str (default ``'count'`` if no arguments are provided, else ``'sum'``)
    One of ``'count'``, ``'sum'``, ``'avg'``, ``'min'``, or ``'max'``.Function used
    to aggregate values for summarization (note: can be normalized with
    `histnorm`). The arguments to this function are the values of `z`.
```

```
        histnorm: str (default `None`)
            One of `'percent'`, `'probability'`, `'density'`, or `'probability
            density'` If `None`, the output of `histfunc` is used as is. If
            `'probability'`, the output of `histfunc` for a given bin is divided by
            the sum of the output of `histfunc` for all bins. If `'percent'`, the
            output of `histfunc` for a given bin is divided by the sum of the
            output of `histfunc` for all bins and multiplied by 100. If
            `'density'`, the output of `histfunc` for a given bin is divided by the
            size of the bin. If `'probability density'`, the output of `histfunc`
            for a given bin is normalized such that it corresponds to the
            probability that a random event whose distribution is described by the
            output of `histfunc` will fall into that bin.
        nbinsx: int
            Positive integer. Sets the number of bins along the x axis.
        nbinsy: int
            Positive integer. Sets the number of bins along the y axis.
        text_auto: bool or string (default `False`)
            If `True` or a string, the x or y or z values will be displayed as
            text, depending on the orientation A string like `'.2f'` will be
            interpreted as a `texttemplate` numeric formatting directive.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    density_mapbox(data_frame=None, lat=None, lon=None, z=None, hover_name=None,
hover_data=None, custom_data=None, animation_frame=None, animation_group=None,
category_orders=None, labels=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, opacity=None, zoom=8, center=None, mapbox_style=None,
radius=None, title=None, template=None, width=None, height=None)
            In a Mapbox density map, each row of `data_frame` contributes to the
intensity of
            the color of the region around the corresponding point on the map
```

```
Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
lat: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks according to latitude on a map.
lon: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks according to longitude on a map.
z: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the z axis in cartesian coordinates.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
```

```
        array_like object. Values from this column or array_like are used to
        provide object-constancy across animation frames: rows with matching
        `animation_group`s will be treated as if they describe the same object
        in each frame.
    category_orders: dict with str keys and list of str values (default `{}`)
        By default, in Python 3.6+, the order of categorical values in axes,
        legends and facets depends on the order in which these values are first
        encountered in `data_frame` (and no order is guaranteed by default in
        Python below 3.6). This parameter is used to force a specific ordering
        of values per column. The keys of this dict should correspond to column
        names, and the values should be lists of strings corresponding to the
        specific display order desired.
    labels: dict with str keys and str values (default `{}`)
        By default, column names are used in the figure for axis titles, legend
        entries and hovers. This parameter allows this to be overridden. The
        keys of this dict should correspond to column names, and the values
        should correspond to the desired label to be displayed.
    color_continuous_scale: list of str
        Strings should define valid CSS-colors This list is used to build a
        continuous color scale when the column denoted by `color` contains
        numeric data. Various useful color scales are available in the
        `plotly.express.colors` submodules, specifically
        `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
        and `plotly.express.colors.cyclical`.
    range_color: list of two numbers
        If provided, overrides auto-scaling on the continuous color scale.
    color_continuous_midpoint: number (default `None`)
        If set, computes the bounds of the continuous color scale to have the
        desired midpoint. Setting this value is recommended when using
        `plotly.express.colors.diverging` color scales as the inputs to
        `color_continuous_scale`.
    opacity: float
        Value between 0 and 1. Sets the opacity for markers.
    zoom: int (default `8`)
        Between 0 and 20. Sets map zoom level.
    center: dict
        Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
    mapbox_style: str (default `'basic'`, needs Mapbox API token)
        Identifier of base map style, some of which require a Mapbox API token
        to be set using `plotly.express.set_mapbox_access_token()`. Allowed
        values which do not require a Mapbox API token are `'open-street-map'`,
        `'white-bg'`, `'carto-positron'`, `'carto-darkmatter'`, `'stamen-
        terrain'`, `'stamen-toner'`, `'stamen-watercolor'`. Allowed values
```

```
            which do require a Mapbox API token are ``'basic'``, ``'streets'``,
            ``'outdoors'``, ``'light'``, ``'dark'``, ``'satellite'``, ``'satellite-
            streets'``.
        radius: int (default is 30)
            Sets the radius of influence of each point.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    ecdf(data_frame=None, x=None, y=None, color=None, text=None, line_dash=None,
symbol=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, hover_name=None, hover_data=None, animation_frame=None,
animation_group=None, markers=False, lines=True, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, line_dash_sequence=None,
line_dash_map=None, symbol_sequence=None, symbol_map=None, marginal=None, opacity=None,
orientation=None, ecdfnorm='probability', ecdfmode='standard', render_mode='auto',
log_x=False, log_y=False, range_x=None, range_y=None, title=None, template=None,
width=None, height=None)
            In a Empirical Cumulative Distribution Function (ECDF) plot, rows of
`data_frame`
            are sorted by the value `x` (or `y` if `orientation` is ``'h'``) and their
cumulative
            count (or the cumulative sum of `y` if supplied and `orientation` is `h`)
is drawn
            as a line.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
```

```
x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the x axis in cartesian coordinates. If
    `orientation` is `'h'`, the cumulative sum of this argument is plotted
    rather than the cumulative count. Either `x` or `y` can optionally be a
    list of column references or array_likes,  in which case the data will
    be treated as if it were 'wide' rather than 'long'.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates. If
    `orientation` is `'v'`, the cumulative sum of this argument is plotted
    rather than the cumulative count. Either `x` or `y` can optionally be a
    list of column references or array_likes,  in which case the data will
    be treated as if it were 'wide' rather than 'long'.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
line_dash: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign dash-patterns to lines.
symbol: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
markers: boolean (default `False`)
    If `True`, markers are shown on lines.
lines: boolean (default `True`)
    If `False`, lines are not drawn (forced to `True` if `markers` is
    `False`).
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
```

```
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
line_dash_sequence: list of str
    Strings should define valid plotly.js dash-patterns. When `line_dash`
    is set, values in that column are assigned dash-patterns by cycling
    through `line_dash_sequence` in the order described in
    `category_orders`, unless the value of `line_dash` is a key in
    `line_dash_map`.
line_dash_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js dash-patterns. Used to override
    `line_dash_sequences` to assign a specific dash-patterns to lines
    corresponding with specific values. Keys in `line_dash_map` should be
    values in the column denoted by `line_dash`. Alternatively, if the
    values of `line_dash` are valid line-dash names, the string
    `'identity'` may be passed to cause them to be used directly.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
marginal: str
    One of ``'rug'``, ``'box'``, ``'violin'``, or ``'histogram'``. If set, a
    subplot is drawn alongside the main plot, visualizing the distribution.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
orientation: str, one of ``'h'`` for horizontal or ``'v'`` for vertical.
    (default ``'v'`` if `x` and `y` are provided and both continous or both
    categorical,  otherwise ``'v'``(``'h'``) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise ``'v'``(``'h'``) if only `x`(`y`) is
    provided)
ecdfnorm: string or `None` (default ``'probability'``)
    One of ``'probability'`` or ``'percent'`` If `None`, values will be raw
    counts or sums. If ``'probability', values will be probabilities
    normalized from 0 to 1. If ``'percent', values will be percentages
    normalized from 0 to 100.
ecdfmode: string (default ``'standard'``)
    One of ``'standard'``, ``'complementary'`` or ``'reversed'`` If ``'standard'``,
    the ECDF is plotted such that values represent data at or below the
    point. If ``'complementary'``, the CCDF is plotted such that values
    represent data above the point. If ``'reversed'``, a variant of the CCDF
    is plotted such that values represent data at or above the point.
render_mode: str
    One of ``'auto'``, ``'svg'`` or ``'webgl'``, default ``'auto'`` Controls the
    browser API used to draw marks. ``'svg'`` is appropriate for figures of
    less than 1000 data points, and will allow for fully-vectorized output.
    ``'webgl'`` is likely necessary for acceptable performance above 1000
    points but rasterizes part of the output.  ``'auto'`` uses heuristics to
    choose the mode.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
title: str
    The figure title.
template: str or dict or plotly.graph_objects.layout.Template instance
    The figure template name (must be a key in plotly.io.templates) or
```

```
        definition.
    width: int (default `None`)
        The figure width in pixels.
    height: int (default `None`)
        The figure height in pixels.

    Returns
    -------
        plotly.graph_objects.Figure

funnel(data_frame=None, x=None, y=None, color=None, facet_row=None, facet_col=None,
facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None,
hover_data=None, custom_data=None, text=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, opacity=None, orientation=None, log_x=False, log_y=False,
range_x=None, range_y=None, title=None, template=None, width=None, height=None)
        In a funnel plot, each row of `data_frame` is represented as a
        rectangular sector of a funnel.

    Parameters
    ----------
    data_frame: DataFrame or array-like or dict
        This argument needs to be passed for column names (and not keyword
        names) to be used. Array-like and dict are tranformed internally to a
        pandas DataFrame. Optional: if missing, a DataFrame gets constructed
        under the hood using the other arguments.
    x: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        position marks along the x axis in cartesian coordinates. Either `x` or
        `y` can optionally be a list of column references or array_likes,  in
        which case the data will be treated as if it were 'wide' rather than
        'long'.
    y: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        position marks along the y axis in cartesian coordinates. Either `x` or
        `y` can optionally be a list of column references or array_likes,  in
        which case the data will be treated as if it were 'wide' rather than
        'long'.
    color: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign color to marks.
    facet_row: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to facetted subplots in the vertical direction.
    facet_col: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to facetted subplots in the horizontal direction.
    facet_col_wrap: int
        Maximum number of facet columns. Wraps the column variable at this
        width, so that the column facets span multiple rows. Ignored if 0, and
        forced to 0 if `facet_row` or a `marginal` is set.
    facet_row_spacing: float between 0 and 1
        Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
        when facet_col_wrap is used.
    facet_col_spacing: float between 0 and 1
        Spacing between facet columns, in paper units Default is 0.02.
    hover_name: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like appear in bold
        in the hover tooltip.
    hover_data: list of str or int, or Series or array-like, or dict
        Either a list of names of columns in `data_frame`, or pandas Series, or
        array_like objects or a dict with column names as keys, with values
        True (for default formatting) False (in order to remove this column
        from hover information), or a formatting string, for example ':.3f' or
        '|%a' or list-like data to appear in the hover tooltip or tuples with a
        bool or formatting string as first element, and list-like data to
        appear in hover as second element Values from these columns appear as
        extra data in the hover tooltip.
    custom_data: list of str or int, or Series or array-like
        Either names of columns in `data_frame`, or pandas Series, or
        array_like objects Values from these columns are extra data, to be used
        in widgets or Dash callbacks for example. This data is not user-visible
        but is included in events emitted by the figure (lasso selection etc.)
    text: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like appear in the
        figure as text labels.
    animation_frame: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to animation frames.
    animation_group: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        provide object-constancy across animation frames: rows with matching
        `animation_group`s will be treated as if they describe the same object
        in each frame.
    category_orders: dict with str keys and list of str values (default `{}`)
        By default, in Python 3.6+, the order of categorical values in axes,
        legends and facets depends on the order in which these values are first
        encountered in `data_frame` (and no order is guaranteed by default in
        Python below 3.6). This parameter is used to force a specific ordering
        of values per column. The keys of this dict should correspond to column
        names, and the values should be lists of strings corresponding to the
        specific display order desired.
    labels: dict with str keys and str values (default `{}`)
        By default, column names are used in the figure for axis titles, legend
        entries and hovers. This parameter allows this to be overridden. The
        keys of this dict should correspond to column names, and the values
        should correspond to the desired label to be displayed.
    color_discrete_sequence: list of str
        Strings should define valid CSS-colors. When `color` is set and the
        values in the corresponding column are not numeric, values in that
        column are assigned colors by cycling through `color_discrete_sequence`
        in the order described in `category_orders`, unless the value of
        `color` is a key in `color_discrete_map`. Various useful color
        sequences are available in the `plotly.express.colors` submodules,
        specifically `plotly.express.colors.qualitative`.
    color_discrete_map: dict with str keys and str values (default `{}`)
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    opacity: float
        Value between 0 and 1. Sets the opacity for markers.
    orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
        (default `'v'` if `x` and `y` are provided and both continous or both
        categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
        `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
        provided)
    log_x: boolean (default `False`)
```

```
            If `True`, the x-axis is log-scaled in cartesian coordinates.
        log_y: boolean (default `False`)
            If `True`, the y-axis is log-scaled in cartesian coordinates.
        range_x: list of two numbers
            If provided, overrides auto-scaling on the x-axis in cartesian
            coordinates.
        range_y: list of two numbers
            If provided, overrides auto-scaling on the y-axis in cartesian
            coordinates.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    funnel_area(data_frame=None, names=None, values=None, color=None,
color_discrete_sequence=None, color_discrete_map=None, hover_name=None,
hover_data=None, custom_data=None, labels=None, title=None, template=None, width=None,
height=None, opacity=None)
            In a funnel area plot, each row of `data_frame` is represented as a
            trapezoidal sector of a funnel.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        names: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            labels for sectors.
        values: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set values associated to sectors.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
```

```
            should correspond to the desired label to be displayed.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.

        Returns
        -------
            plotly.graph_objects.Figure

    get_trendline_results(fig)
        Extracts fit statistics for trendlines (when applied to figures generated with
        the `trendline` argument set to `"ols"`).

        Arguments:
            fig: the output of a `plotly.express` charting call
        Returns:
            A `pandas.DataFrame` with a column "px_fit_results" containing the
`statsmodels`
            results objects, along with columns identifying the subset of the data the
            trendline was fit on.

    histogram(data_frame=None, x=None, y=None, color=None, pattern_shape=None,
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, hover_name=None, hover_data=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, pattern_shape_sequence=None, pattern_shape_map=None,
marginal=None, opacity=None, orientation=None, barmode='relative', barnorm=None,
histnorm=None, log_x=False, log_y=False, range_x=None, range_y=None, histfunc=None,
cumulative=None, nbins=None, text_auto=False, title=None, template=None, width=None,
height=None)
            In a histogram, rows of `data_frame` are grouped together into a
            rectangular mark to visualize the 1D distribution of an aggregate
            function `histfunc` (e.g. the count or sum) of the value `y` (or `x` if
            `orientation` is `'h'`).
```

```
Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the x axis in cartesian coordinates. If
    `orientation` is `'h'`, these values are used as inputs to `histfunc`.
    Either `x` or `y` can optionally be a list of column references or
    array_likes,  in which case the data will be treated as if it were
    'wide' rather than 'long'.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates. If
    `orientation` is `'v'`, these values are used as inputs to `histfunc`.
    Either `x` or `y` can optionally be a list of column references or
    array_likes,  in which case the data will be treated as if it were
    'wide' rather than 'long'.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
pattern_shape: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign pattern shapes to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
```

```
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
pattern_shape_sequence: list of str
    Strings should define valid plotly.js patterns-shapes. When
    `pattern_shape` is set, values in that column are assigned patterns-
    shapes by cycling through `pattern_shape_sequence` in the order
    described in `category_orders`, unless the value of `pattern_shape` is
    a key in `pattern_shape_map`.
pattern_shape_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js patterns-shapes. Used to override
    `pattern_shape_sequences` to assign a specific patterns-shapes to lines
    corresponding with specific values. Keys in `pattern_shape_map` should
    be values in the column denoted by `pattern_shape`. Alternatively, if
    the values of `pattern_shape` are valid patterns-shapes names, the
    string `'identity'` may be passed to cause them to be used directly.
marginal: str
    One of `'rug'`, `'box'`, `'violin'`, or `'histogram'`. If set, a
    subplot is drawn alongside the main plot, visualizing the distribution.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
barmode: str (default `'relative'`)
    One of `'group'`, `'overlay'` or `'relative'` In `'relative'` mode,
    bars are stacked above zero for positive values and below zero for
    negative values. In `'overlay'` mode, bars are drawn on top of one
    another. In `'group'` mode, bars are placed beside each other.
barnorm: str (default `None`)
    One of `'fraction'` or `'percent'`. If `'fraction'`, the value of each
    bar is divided by the sum of all values at that location coordinate.
    ``'percent'`` is the same but multiplied by 100 to show percentages.
    `None` will stack up all values at each location coordinate.
histnorm: str (default `None`)
    One of ``'percent'``, ``'probability'``, ``'density'``, or ``'probability
    density'`` If `None`, the output of `histfunc` is used as is. If
    ``'probability'``, the output of `histfunc` for a given bin is divided by
    the sum of the output of `histfunc` for all bins. If ``'percent'``, the
    output of `histfunc` for a given bin is divided by the sum of the
    output of `histfunc` for all bins and multiplied by 100. If
    ``'density'``, the output of `histfunc` for a given bin is divided by the
    size of the bin. If ``'probability density'``, the output of `histfunc`
    for a given bin is normalized such that it corresponds to the
    probability that a random event whose distribution is described by the
    output of `histfunc` will fall into that bin.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
histfunc: str (default ``'count'`` if no arguments are provided, else ``'sum'``)
    One of ``'count'``, ``'sum'``, ``'avg'``, ``'min'``, or ``'max'``.Function used
    to aggregate values for summarization (note: can be normalized with
    `histnorm`). The arguments to this function are the values of `y`(`x`)
    if `orientation` is ``'v'``(``'h'``).
cumulative: boolean (default `False`)
    If `True`, histogram values are cumulative.
nbins: int
    Positive integer. Sets the number of bins.
text_auto: bool or string (default `False`)
    If `True` or a string, the x or y or z values will be displayed as
    text, depending on the orientation A string like ``'.2f'`` will be
    interpreted as a `texttemplate` numeric formatting directive.
title: str
    The figure title.
template: str or dict or plotly.graph_objects.layout.Template instance
    The figure template name (must be a key in plotly.io.templates) or
    definition.
```

```
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    icicle(data_frame=None, names=None, values=None, parents=None, path=None, ids=None,
color=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, color_discrete_sequence=None, color_discrete_map=None,
hover_name=None, hover_data=None, custom_data=None, labels=None, title=None,
template=None, width=None, height=None, branchvalues=None, maxdepth=None)
            An icicle plot represents hierarchial data with adjoined rectangular
            sectors that all cascade from root down to leaf in one direction.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        names: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            labels for sectors.
        values: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set values associated to sectors.
        parents: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            parents in sunburst and treemap charts.
        path: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects List of columns names or columns of a rectangular
            dataframe defining the hierarchy of sectors, from root to leaves. An
            error is raised if path AND ids or parents is passed
        ids: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set ids of sectors
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        branchvalues: str
            'total' or 'remainder' Determines how the items in `values` are summed.
            Whenset to 'total', items in `values` are taken to be valueof all its
            descendants. When set to 'remainder', itemsin `values` corresponding to
            the root and the branches:sectors are taken to be the extra part not
            part of thesum of the values at their leaves.
        maxdepth: int
            Positive integer Sets the number of rendered sectors from any given
            `level`. Set `maxdepth` to -1 to render all thelevels in the hierarchy.

        Returns
        -------
            plotly.graph_objects.Figure
```

imshow(img, zmin=None, zmax=None, origin=None, labels={}, x=None, y=None, animation_frame=None, facet_col=None, facet_col_wrap=None, facet_col_spacing=None, facet_row_spacing=None, color_continuous_scale=None, color_continuous_midpoint=None, range_color=None, title=None, template=None, width=None, height=None, aspect=None, contrast_rescaling=None, binary_string=None, binary_backend='auto',

```
binary_compression_level=4, binary_format='png', text_auto=False)
        Display an image, i.e. data on a 2D regular raster.

        Parameters
        ----------

        img: array-like image, or xarray
            The image data. Supported array shapes are

            - (M, N): an image with scalar data. The data is visualized
              using a colormap.
            - (M, N, 3): an image with RGB values.
            - (M, N, 4): an image with RGBA values, i.e. including transparency.

        zmin, zmax : scalar or iterable, optional
            zmin and zmax define the scalar range that the colormap covers. By default,
            zmin and zmax correspond to the min and max values of the datatype for
integer
            datatypes (ie [0-255] for uint8 images, [0, 65535] for uint16 images,
etc.). For
            a multichannel image of floats, the max of the image is computed and zmax
is the
            smallest power of 256 (1, 255, 65535) greater than this max value,
            with a 5% tolerance. For a single-channel image, the max of the image is
used.
            Overridden by range_color.

        origin : str, 'upper' or 'lower' (default 'upper')
            position of the [0, 0] pixel of the image array, in the upper left or lower
left
            corner. The convention 'upper' is typically used for matrices and images.

        labels : dict with str keys and str values (default `{}`)
            Sets names used in the figure for axis titles (keys ``x`` and ``y``),
            colorbar title and hoverlabel (key ``color``). The values should correspond
            to the desired label to be displayed. If ``img`` is an xarray, dimension
            names are used for axis titles, and long name for the colorbar title
            (unless overridden in ``labels``). Possible keys are: x, y, and color.

        x, y: list-like, optional
            x and y are used to label the axes of single-channel heatmap visualizations
and
            their lengths must match the lengths of the second and first dimensions of
```

```
the
            img argument. They are auto-populated if the input is an xarray.

        animation_frame: int or str, optional (default None)
            axis number along which the image array is sliced to create an animation
plot.
            If `img` is an xarray, `animation_frame` can be the name of one the
dimensions.

        facet_col: int or str, optional (default None)
            axis number along which the image array is sliced to create a facetted
plot.
            If `img` is an xarray, `facet_col` can be the name of one the dimensions.

        facet_col_wrap: int
            Maximum number of facet columns. Wraps the column variable at this width,
            so that the column facets span multiple rows.
            Ignored if `facet_col` is None.

        facet_col_spacing: float between 0 and 1
            Spacing between facet columns, in paper units. Default is 0.02.

        facet_row_spacing: float between 0 and 1
            Spacing between facet rows created when ``facet_col_wrap`` is used, in
            paper units. Default is 0.0.7.

        color_continuous_scale : str or list of str
            colormap used to map scalar data to colors (for a 2D image). This parameter
is
            not used for RGB or RGBA images. If a string is provided, it should be the
name
            of a known color scale, and if a list is provided, it should be a list of
CSS-
            compatible colors.

        color_continuous_midpoint : number
            If set, computes the bounds of the continuous color scale to have the
desired
            midpoint. Overridden by range_color or zmin and zmax.

        range_color : list of two numbers
            If provided, overrides auto-scaling on the continuous color scale,
including
```

```
            overriding `color_continuous_midpoint`. Also overrides zmin and zmax. Used
only
            for single-channel images.

        title : str
            The figure title.

        template : str or dict or plotly.graph_objects.layout.Template instance
            The figure template name or definition.

        width : number
            The figure width in pixels.

        height: number
            The figure height in pixels.

        aspect: 'equal', 'auto', or None
          - 'equal': Ensures an aspect ratio of 1 or pixels (square pixels)
          - 'auto': The axes is kept fixed and the aspect ratio of pixels is
            adjusted so that the data fit in the axes. In general, this will
            result in non-square pixels.
          - if None, 'equal' is used for numpy arrays and 'auto' for xarrays
            (which have typically heterogeneous coordinates)

        contrast_rescaling: 'minmax', 'infer', or None
            how to determine data values corresponding to the bounds of the color
            range, when zmin or zmax are not passed. If `minmax`, the min and max
            values of the image are used. If `infer`, a heuristic based on the image
            data type is used.

        binary_string: bool, default None
            if True, the image data are first rescaled and encoded as uint8 and
            then passed to plotly.js as a b64 PNG string. If False, data are passed
            unchanged as a numerical array. Setting to True may lead to performance
            gains, at the cost of a loss of precision depending on the original data
            type. If None, use_binary_string is set to True for multichannel (eg) RGB
            arrays, and to False for single-channel (2D) arrays. 2D arrays are
            represented as grayscale and with no colorbar if use_binary_string is
            True.

        binary_backend: str, 'auto' (default), 'pil' or 'pypng'
            Third-party package for the transformation of numpy arrays to
            png b64 strings. If 'auto', Pillow is used if installed,  otherwise
            pypng.

        binary_compression_level: int, between 0 and 9 (default 4)
            png compression level to be passed to the backend when transforming an
            array to a png b64 string. Increasing `binary_compression` decreases the
            size of the png string, but the compression step takes more time. For most
            images it is not worth using levels greater than 5, but it's possible to
            test `len(fig.data[0].source)` and to time the execution of `imshow` to
            tune the level of compression. 0 means no compression (not recommended).

        binary_format: str, 'png' (default) or 'jpg'
            compression format used to generate b64 string. 'png' is recommended
            since it uses lossless compression, but 'jpg' (lossy) compression can
            result if smaller binary strings for natural images.

        text_auto: bool or str (default `False`)
            If `True` or a string, single-channel `img` values will be displayed as
text.
            A string like `'.2f'` will be interpreted as a `texttemplate` numeric
formatting directive.

        Returns
        -------
        fig : graph_objects.Figure containing the displayed image

        See also
        --------

        plotly.graph_objects.Image : image trace
        plotly.graph_objects.Heatmap : heatmap trace

        Notes
        -----

        In order to update and customize the returned figure, use
        `go.Figure.update_traces` or `go.Figure.update_layout`.

        If an xarray is passed, dimensions names and coordinates are used for
        axes labels and ticks.

    line(data_frame=None, x=None, y=None, line_group=None, color=None, line_dash=None,
symbol=None, hover_name=None, hover_data=None, custom_data=None, text=None,
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, error_x=None, error_x_minus=None, error_y=None,
error_y_minus=None, animation_frame=None, animation_group=None, category_orders=None,
labels=None, orientation=None, color_discrete_sequence=None, color_discrete_map=None,
line_dash_sequence=None, line_dash_map=None, symbol_sequence=None, symbol_map=None,
markers=False, log_x=False, log_y=False, range_x=None, range_y=None, line_shape=None,
render_mode='auto', title=None, template=None, width=None, height=None)
            In a 2D line plot, each row of `data_frame` is represented as vertex of
            a polyline mark in 2D space.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates. Either `x` or
            `y` can optionally be a list of column references or array_likes,  in
            which case the data will be treated as if it were 'wide' rather than
            'long'.
        line_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            group rows of `data_frame` into lines.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        line_dash: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign dash-patterns to lines.
        symbol: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
error_x: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars. If `error_x_minus` is `None`, error bars will
    be symmetrical, otherwise `error_x` is used for the positive direction
    only.
error_x_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars in the negative direction. Ignored if `error_x`
    is `None`.
error_y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars. If `error_y_minus` is `None`, error bars will
    be symmetrical, otherwise `error_y` is used for the positive direction
    only.
error_y_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars in the negative direction. Ignored if `error_y`
    is `None`.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
orientation: str, one of ``'h'`` for horizontal or ``'v'`` for vertical.
    (default ``'v'`` if `x` and `y` are provided and both continous or both
    categorical,  otherwise ``'v'``(``'h'``) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise ``'v'``(``'h'``) if only `x`(`y`) is
    provided)
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string ``'identity'`` may be
    passed to cause them to be used directly.
line_dash_sequence: list of str
    Strings should define valid plotly.js dash-patterns. When `line_dash`
    is set, values in that column are assigned dash-patterns by cycling
    through `line_dash_sequence` in the order described in
    `category_orders`, unless the value of `line_dash` is a key in
    `line_dash_map`.
line_dash_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js dash-patterns. Used to override
    `line_dash_sequences` to assign a specific dash-patterns to lines
    corresponding with specific values. Keys in `line_dash_map` should be
    values in the column denoted by `line_dash`. Alternatively, if the
    values of `line_dash` are valid line-dash names, the string
    ``'identity'`` may be passed to cause them to be used directly.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
```

```
            column denoted by `symbol`. Alternatively, if the values of `symbol`
            are valid symbol names, the string `'identity'` may be passed to cause
            them to be used directly.
        markers: boolean (default `False`)
            If `True`, markers are shown on lines.
        log_x: boolean (default `False`)
            If `True`, the x-axis is log-scaled in cartesian coordinates.
        log_y: boolean (default `False`)
            If `True`, the y-axis is log-scaled in cartesian coordinates.
        range_x: list of two numbers
            If provided, overrides auto-scaling on the x-axis in cartesian
            coordinates.
        range_y: list of two numbers
            If provided, overrides auto-scaling on the y-axis in cartesian
            coordinates.
        line_shape: str (default `'linear'`)
            One of `'linear'` or `'spline'`.
        render_mode: str
            One of `'auto'`, `'svg'` or `'webgl'`, default `'auto'` Controls the
            browser API used to draw marks. `'svg`' is appropriate for figures of
            less than 1000 data points, and will allow for fully-vectorized output.
            `'webgl'` is likely necessary for acceptable performance above 1000
            points but rasterizes part of the output.  `'auto'` uses heuristics to
            choose the mode.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure
```

    line_3d(data_frame=None, x=None, y=None, z=None, color=None, line_dash=None,
text=None, line_group=None, symbol=None, hover_name=None, hover_data=None,
custom_data=None, error_x=None, error_x_minus=None, error_y=None, error_y_minus=None,
error_z=None, error_z_minus=None, animation_frame=None, animation_group=None,
category_orders=None, labels=None, color_discrete_sequence=None,

```
color_discrete_map=None, line_dash_sequence=None, line_dash_map=None,
symbol_sequence=None, symbol_map=None, markers=False, log_x=False, log_y=False,
log_z=False, range_x=None, range_y=None, range_z=None, title=None, template=None,
width=None, height=None)
            In a 3D line plot, each row of `data_frame` is represented as vertex of
            a polyline mark in 3D space.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates.
        z: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the z axis in cartesian coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        line_dash: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign dash-patterns to lines.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        line_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            group rows of `data_frame` into lines.
        symbol: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
error_x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars. If `error_x_minus` is `None`, error bars will
    be symmetrical, otherwise `error_x` is used for the positive direction
    only.
error_x_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars in the negative direction. Ignored if `error_x`
    is `None`.
error_y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars. If `error_y_minus` is `None`, error bars will
    be symmetrical, otherwise `error_y` is used for the positive direction
    only.
error_y_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars in the negative direction. Ignored if `error_y`
    is `None`.
```

```
error_z: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size z-axis error bars. If `error_z_minus` is `None`, error bars will
    be symmetrical, otherwise `error_z` is used for the positive direction
    only.
error_z_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size z-axis error bars in the negative direction. Ignored if `error_z`
    is `None`.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
```

```
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    line_dash_sequence: list of str
        Strings should define valid plotly.js dash-patterns. When `line_dash`
        is set, values in that column are assigned dash-patterns by cycling
        through `line_dash_sequence` in the order described in
        `category_orders`, unless the value of `line_dash` is a key in
        `line_dash_map`.
    line_dash_map: dict with str keys and str values (default `{}`)
        Strings values define plotly.js dash-patterns. Used to override
        `line_dash_sequences` to assign a specific dash-patterns to lines
        corresponding with specific values. Keys in `line_dash_map` should be
        values in the column denoted by `line_dash`. Alternatively, if the
        values of `line_dash` are valid line-dash names, the string
        `'identity'` may be passed to cause them to be used directly.
    symbol_sequence: list of str
        Strings should define valid plotly.js symbols. When `symbol` is set,
        values in that column are assigned symbols by cycling through
        `symbol_sequence` in the order described in `category_orders`, unless
        the value of `symbol` is a key in `symbol_map`.
    symbol_map: dict with str keys and str values (default `{}`)
        String values should define plotly.js symbols Used to override
        `symbol_sequence` to assign a specific symbols to marks corresponding
        with specific values. Keys in `symbol_map` should be values in the
        column denoted by `symbol`. Alternatively, if the values of `symbol`
        are valid symbol names, the string `'identity'` may be passed to cause
        them to be used directly.
    markers: boolean (default `False`)
        If `True`, markers are shown on lines.
    log_x: boolean (default `False`)
        If `True`, the x-axis is log-scaled in cartesian coordinates.
    log_y: boolean (default `False`)
        If `True`, the y-axis is log-scaled in cartesian coordinates.
    log_z: boolean (default `False`)
        If `True`, the z-axis is log-scaled in cartesian coordinates.
    range_x: list of two numbers
        If provided, overrides auto-scaling on the x-axis in cartesian
        coordinates.
    range_y: list of two numbers
```

```
            If provided, overrides auto-scaling on the y-axis in cartesian
            coordinates.
        range_z: list of two numbers
            If provided, overrides auto-scaling on the z-axis in cartesian
            coordinates.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    line_geo(data_frame=None, lat=None, lon=None, locations=None, locationmode=None,
geojson=None, featureidkey=None, color=None, line_dash=None, text=None, facet_row=None,
facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,
hover_name=None, hover_data=None, custom_data=None, line_group=None, symbol=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, line_dash_sequence=None,
line_dash_map=None, symbol_sequence=None, symbol_map=None, markers=False,
projection=None, scope=None, center=None, fitbounds=None, basemap_visible=None,
title=None, template=None, width=None, height=None)
            In a geographic line plot, each row of `data_frame` is represented as
            vertex of a polyline mark on a map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        lat: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to latitude on a map.
        lon: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to longitude on a map.
        locations: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are to be
            interpreted according to `locationmode` and mapped to
            longitude/latitude.
        locationmode: str
            One of 'ISO-3', 'USA-states', or 'country names' Determines the set of
            locations used to match entries in `locations` to regions on the map.
        geojson: GeoJSON-formatted dict
            Must contain a Polygon feature collection, with IDs, which are
            references from `locations`.
        featureidkey: str (default: `'id'`)
            Path to field in GeoJSON feature object with which to match the values
            passed in to `locations`.The most common alternative to the default is
            of the form `'properties.<key>`.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        line_dash: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign dash-patterns to lines.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        facet_row: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the vertical direction.
        facet_col: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to facetted subplots in the horizontal direction.
        facet_col_wrap: int
            Maximum number of facet columns. Wraps the column variable at this
            width, so that the column facets span multiple rows. Ignored if 0, and
            forced to 0 if `facet_row` or a `marginal` is set.
        facet_row_spacing: float between 0 and 1
```

```
        Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
        when facet_col_wrap is used.
    facet_col_spacing: float between 0 and 1
        Spacing between facet columns, in paper units Default is 0.02.
    hover_name: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like appear in bold
        in the hover tooltip.
    hover_data: list of str or int, or Series or array-like, or dict
        Either a list of names of columns in `data_frame`, or pandas Series, or
        array_like objects or a dict with column names as keys, with values
        True (for default formatting) False (in order to remove this column
        from hover information), or a formatting string, for example ':.3f' or
        '|%a' or list-like data to appear in the hover tooltip or tuples with a
        bool or formatting string as first element, and list-like data to
        appear in hover as second element Values from these columns appear as
        extra data in the hover tooltip.
    custom_data: list of str or int, or Series or array-like
        Either names of columns in `data_frame`, or pandas Series, or
        array_like objects Values from these columns are extra data, to be used
        in widgets or Dash callbacks for example. This data is not user-visible
        but is included in events emitted by the figure (lasso selection etc.)
    line_group: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        group rows of `data_frame` into lines.
    symbol: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign symbols to marks.
    animation_frame: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to animation frames.
    animation_group: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        provide object-constancy across animation frames: rows with matching
        `animation_group`s will be treated as if they describe the same object
        in each frame.
    category_orders: dict with str keys and list of str values (default `{}`)
        By default, in Python 3.6+, the order of categorical values in axes,
        legends and facets depends on the order in which these values are first
        encountered in `data_frame` (and no order is guaranteed by default in
        Python below 3.6). This parameter is used to force a specific ordering
        of values per column. The keys of this dict should correspond to column
        names, and the values should be lists of strings corresponding to the
        specific display order desired.
    labels: dict with str keys and str values (default `{}`)
        By default, column names are used in the figure for axis titles, legend
        entries and hovers. This parameter allows this to be overridden. The
        keys of this dict should correspond to column names, and the values
        should correspond to the desired label to be displayed.
    color_discrete_sequence: list of str
        Strings should define valid CSS-colors. When `color` is set and the
        values in the corresponding column are not numeric, values in that
        column are assigned colors by cycling through `color_discrete_sequence`
        in the order described in `category_orders`, unless the value of
        `color` is a key in `color_discrete_map`. Various useful color
        sequences are available in the `plotly.express.colors` submodules,
        specifically `plotly.express.colors.qualitative`.
    color_discrete_map: dict with str keys and str values (default `{}`)
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    line_dash_sequence: list of str
        Strings should define valid plotly.js dash-patterns. When `line_dash`
        is set, values in that column are assigned dash-patterns by cycling
        through `line_dash_sequence` in the order described in
        `category_orders`, unless the value of `line_dash` is a key in
        `line_dash_map`.
    line_dash_map: dict with str keys and str values (default `{}`)
        Strings values define plotly.js dash-patterns. Used to override
        `line_dash_sequences` to assign a specific dash-patterns to lines
        corresponding with specific values. Keys in `line_dash_map` should be
        values in the column denoted by `line_dash`. Alternatively, if the
        values of `line_dash` are valid line-dash names, the string
        `'identity'` may be passed to cause them to be used directly.
    symbol_sequence: list of str
        Strings should define valid plotly.js symbols. When `symbol` is set,
        values in that column are assigned symbols by cycling through
        `symbol_sequence` in the order described in `category_orders`, unless
        the value of `symbol` is a key in `symbol_map`.
```

```
        symbol_map: dict with str keys and str values (default `{}`)
            String values should define plotly.js symbols Used to override
            `symbol_sequence` to assign a specific symbols to marks corresponding
            with specific values. Keys in `symbol_map` should be values in the
            column denoted by `symbol`. Alternatively, if the values of `symbol`
            are valid symbol names, the string `'identity'` may be passed to cause
            them to be used directly.
        markers: boolean (default `False`)
            If `True`, markers are shown on lines.
        projection: str
            One of `'equirectangular'`, `'mercator'`, `'orthographic'`, `'natural
            earth'`, `'kavrayskiy7'`, `'miller'`, `'robinson'`, `'eckert4'`,
            `'azimuthal equal area'`, `'azimuthal equidistant'`, `'conic equal
            area'`, `'conic conformal'`, `'conic equidistant'`, `'gnomonic'`,
            `'stereographic'`, `'mollweide'`, `'hammer'`, `'transverse mercator'`,
            `'albers usa'`, `'winkel tripel'`, `'aitoff'`, or `'sinusoidal'`Default
            depends on `scope`.
        scope: str (default `'world'`).
            One of `'world'`, `'usa'`, `'europe'`, `'asia'`, `'africa'`, `'north
            america'`, or `'south america'`Default is `'world'` unless `projection`
            is set to `'albers usa'`, which forces `'usa'`.
        center: dict
            Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
        fitbounds: str (default `False`).
            One of `False`, `locations` or `geojson`.
        basemap_visible: bool
            Force the basemap visibility.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    line_mapbox(data_frame=None, lat=None, lon=None, color=None, text=None,
hover_name=None, hover_data=None, custom_data=None, line_group=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, zoom=8, center=None,
mapbox_style=None, title=None, template=None, width=None, height=None)
            In a Mapbox line plot, each row of `data_frame` is represented as
            vertex of a polyline mark on a Mapbox map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        lat: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to latitude on a map.
        lon: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to longitude on a map.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
```

```
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
line_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    group rows of `data_frame` into lines.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
```

```
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        zoom: int (default `8`)
            Between 0 and 20. Sets map zoom level.
        center: dict
            Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
        mapbox_style: str (default `'basic'`, needs Mapbox API token)
            Identifier of base map style, some of which require a Mapbox API token
            to be set using `plotly.express.set_mapbox_access_token()`. Allowed
            values which do not require a Mapbox API token are `'open-street-map'`,
            `'white-bg'`, `'carto-positron'`, `'carto-darkmatter'`, `'stamen-
            terrain'`, `'stamen-toner'`, `'stamen-watercolor'`. Allowed values
            which do require a Mapbox API token are `'basic'`, `'streets'`,
            `'outdoors'`, `'light'`, `'dark'`, `'satellite'`, `'satellite-
            streets'`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    line_polar(data_frame=None, r=None, theta=None, color=None, line_dash=None,
hover_name=None, hover_data=None, custom_data=None, line_group=None, text=None,
symbol=None, animation_frame=None, animation_group=None, category_orders=None,
labels=None, color_discrete_sequence=None, color_discrete_map=None,
line_dash_sequence=None, line_dash_map=None, symbol_sequence=None, symbol_map=None,
markers=False, direction='clockwise', start_angle=90, line_close=False,
line_shape=None, render_mode='auto', range_r=None, range_theta=None, log_r=False,
title=None, template=None, width=None, height=None)
            In a polar line plot, each row of `data_frame` is represented as vertex
            of a polyline mark in polar coordinates.

        Parameters
        ----------
```

```
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
r: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the radial axis in polar coordinates.
theta: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the angular axis in polar coordinates.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
line_dash: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign dash-patterns to lines.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
line_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    group rows of `data_frame` into lines.
```

```
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
symbol: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
line_dash_sequence: list of str
    Strings should define valid plotly.js dash-patterns. When `line_dash`
    is set, values in that column are assigned dash-patterns by cycling
    through `line_dash_sequence` in the order described in
    `category_orders`, unless the value of `line_dash` is a key in
    `line_dash_map`.
line_dash_map: dict with str keys and str values (default `{}`)
    Strings values define plotly.js dash-patterns. Used to override
    `line_dash_sequences` to assign a specific dash-patterns to lines
    corresponding with specific values. Keys in `line_dash_map` should be
    values in the column denoted by `line_dash`. Alternatively, if the
    values of `line_dash` are valid line-dash names, the string
    `'identity'` may be passed to cause them to be used directly.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
markers: boolean (default `False`)
    If `True`, markers are shown on lines.
direction: str
    One of '`counterclockwise'` or `'clockwise'`. Default is `'clockwise'`
    Sets the direction in which increasing values of the angular axis are
    drawn.
start_angle: int (default `90`)
    Sets start angle for the angular axis, with 0 being due east and 90
    being due north.
line_close: boolean (default `False`)
    If `True`, an extra line segment is drawn between the first and last
    point.
line_shape: str (default `'linear'`)
    One of `'linear'` or `'spline'`.
render_mode: str
```

```
            One of ``'auto'``, ``'svg'`` or ``'webgl'``, default ``'auto'`` Controls the
            browser API used to draw marks. ``'svg`' is appropriate for figures of
            less than 1000 data points, and will allow for fully-vectorized output.
            ``'webgl'`` is likely necessary for acceptable performance above 1000
            points but rasterizes part of the output.  ``'auto'`` uses heuristics to
            choose the mode.
        range_r: list of two numbers
            If provided, overrides auto-scaling on the radial axis in polar
            coordinates.
        range_theta: list of two numbers
            If provided, overrides auto-scaling on the angular axis in polar
            coordinates.
        log_r: boolean (default `False`)
            If `True`, the radial axis is log-scaled in polar coordinates.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    line_ternary(data_frame=None, a=None, b=None, c=None, color=None, line_dash=None,
line_group=None, symbol=None, hover_name=None, hover_data=None, custom_data=None,
text=None, animation_frame=None, animation_group=None, category_orders=None,
labels=None, color_discrete_sequence=None, color_discrete_map=None,
line_dash_sequence=None, line_dash_map=None, symbol_sequence=None, symbol_map=None,
markers=False, line_shape=None, title=None, template=None, width=None, height=None)
            In a ternary line plot, each row of `data_frame` is represented as
            vertex of a polyline mark in ternary coordinates.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        a: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the a axis in ternary coordinates.
        b: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the b axis in ternary coordinates.
        c: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the c axis in ternary coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        line_dash: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign dash-patterns to lines.
        line_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            group rows of `data_frame` into lines.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
```

```
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
```

```
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    line_dash_sequence: list of str
        Strings should define valid plotly.js dash-patterns. When `line_dash`
        is set, values in that column are assigned dash-patterns by cycling
        through `line_dash_sequence` in the order described in
        `category_orders`, unless the value of `line_dash` is a key in
        `line_dash_map`.
    line_dash_map: dict with str keys and str values (default `{}`)
        Strings values define plotly.js dash-patterns. Used to override
        `line_dash_sequences` to assign a specific dash-patterns to lines
        corresponding with specific values. Keys in `line_dash_map` should be
        values in the column denoted by `line_dash`. Alternatively, if the
        values of `line_dash` are valid line-dash names, the string
        `'identity'` may be passed to cause them to be used directly.
    symbol_sequence: list of str
        Strings should define valid plotly.js symbols. When `symbol` is set,
        values in that column are assigned symbols by cycling through
        `symbol_sequence` in the order described in `category_orders`, unless
        the value of `symbol` is a key in `symbol_map`.
    symbol_map: dict with str keys and str values (default `{}`)
        String values should define plotly.js symbols Used to override
        `symbol_sequence` to assign a specific symbols to marks corresponding
        with specific values. Keys in `symbol_map` should be values in the
        column denoted by `symbol`. Alternatively, if the values of `symbol`
        are valid symbol names, the string `'identity'` may be passed to cause
        them to be used directly.
    markers: boolean (default `False`)
        If `True`, markers are shown on lines.
    line_shape: str (default `'linear'`)
        One of `'linear'` or `'spline'`.
    title: str
        The figure title.
    template: str or dict or plotly.graph_objects.layout.Template instance
        The figure template name (must be a key in plotly.io.templates) or
        definition.
    width: int (default `None`)
        The figure width in pixels.
    height: int (default `None`)
        The figure height in pixels.

    Returns
```

```
        -------
            plotly.graph_objects.Figure

    parallel_categories(data_frame=None, dimensions=None, color=None, labels=None,
color_continuous_scale=None, range_color=None, color_continuous_midpoint=None,
title=None, template=None, width=None, height=None, dimensions_max_cardinality=50)
            In a parallel categories (or parallel sets) plot, each row of
            `data_frame` is grouped with other rows that share the same values of
            `dimensions` and then plotted as a polyline mark through a set of
            parallel axes, one for each of the `dimensions`.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        dimensions: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are used for
            multidimensional visualization.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        dimensions_max_cardinality: int (default 50)
            When `dimensions` is `None` and `data_frame` is provided, columns with
            more than this number of unique values are excluded from the output.
            Not used when `dimensions` is passed.

        Returns
        -------
            plotly.graph_objects.Figure

    parallel_coordinates(data_frame=None, dimensions=None, color=None, labels=None,
color_continuous_scale=None, range_color=None, color_continuous_midpoint=None,
title=None, template=None, width=None, height=None)
            In a parallel coordinates plot, each row of `data_frame` is represented
            by a polyline mark which traverses a set of parallel axes, one for each
            of the `dimensions`.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        dimensions: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are used for
            multidimensional visualization.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    pie(data_frame=None, names=None, values=None, color=None,
color_discrete_sequence=None, color_discrete_map=None, hover_name=None,
hover_data=None, custom_data=None, labels=None, title=None, template=None, width=None,
height=None, opacity=None, hole=None)
            In a pie plot, each row of `data_frame` is represented as a sector of a
            pie.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        names: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            labels for sectors.
        values: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set values associated to sectors.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
```

```
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        hole: float
            Sets the fraction of the radius to cut out of the pie.Use this to make
            a donut chart.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter(data_frame=None, x=None, y=None, color=None, symbol=None, size=None,
hover_name=None, hover_data=None, custom_data=None, text=None, facet_row=None,
facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,
error_x=None, error_x_minus=None, error_y=None, error_y_minus=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
orientation=None, color_discrete_sequence=None, color_discrete_map=None,
color_continuous_scale=None, range_color=None, color_continuous_midpoint=None,
symbol_sequence=None, symbol_map=None, opacity=None, size_max=None, marginal_x=None,
marginal_y=None, trendline=None, trendline_options=None, trendline_color_override=None,
trendline_scope='trace', log_x=False, log_y=False, range_x=None, range_y=None,
render_mode='auto', title=None, template=None, width=None, height=None)
            In a scatter plot, each row of `data_frame` is represented by a symbol
            mark in 2D space.
```

```
Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the x axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
symbol: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
size: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign mark sizes.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
error_x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars. If `error_x_minus` is `None`, error bars will
    be symmetrical, otherwise `error_x` is used for the positive direction
    only.
error_x_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size x-axis error bars in the negative direction. Ignored if `error_x`
    is `None`.
error_y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars. If `error_y_minus` is `None`, error bars will
    be symmetrical, otherwise `error_y` is used for the positive direction
    only.
error_y_minus: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    size y-axis error bars in the negative direction. Ignored if `error_y`
    is `None`.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
size_max: int (default `20`)
    Set the maximum mark size when using `size`.
marginal_x: str
    One of `'rug'`, `'box'`, `'violin'`, or `'histogram'`. If set, a
    horizontal subplot is drawn above the main plot, visualizing the
    x-distribution.
marginal_y: str
```

```
        One of ``'rug'``, ``'box'``, ``'violin'``, or ``'histogram'``. If set, a
        vertical subplot is drawn to the right of the main plot, visualizing
        the y-distribution.
    trendline: str
        One of ``'ols'``, ``'lowess'``, ``'rolling'``, ``'expanding'`` or ``'ewm'``. If
        ``'ols'``, an Ordinary Least Squares regression line will be drawn for
        each discrete-color/symbol group. If ``'lowess'``, a Locally Weighted
        Scatterplot Smoothing line will be drawn for each discrete-color/symbol
        group. If ``'rolling'``, a Rolling (e.g. rolling average, rolling median)
        line will be drawn for each discrete-color/symbol group. If
        ``'expanding'``, an Expanding (e.g. expanding average, expanding sum)
        line will be drawn for each discrete-color/symbol group. If ``'ewm'``, an
        Exponentially Weighted Moment (e.g. exponentially-weighted moving
        average) line will be drawn for each discrete-color/symbol group. See
        the docstrings for the functions in
        `plotly.express.trendline_functions` for more details on these
        functions and how to configure them with the `trendline_options`
        argument.
    trendline_options: dict
        Options passed as the first argument to the function from
        `plotly.express.trendline_functions`  named in the `trendline`
        argument.
    trendline_color_override: str
        Valid CSS color. If provided, and if `trendline` is set, all trendlines
        will be drawn in this color rather than in the same color as the traces
        from which they draw their inputs.
    trendline_scope: str (one of ``'trace'`` or ``'overall'``, default ``'trace'``)
        If ``'trace'``, then one trendline is drawn per trace (i.e. per color,
        symbol, facet, animation frame etc) and if ``'overall'`` then one
        trendline is computed for the entire dataset, and replicated across all
        facets.
    log_x: boolean (default `False`)
        If `True`, the x-axis is log-scaled in cartesian coordinates.
    log_y: boolean (default `False`)
        If `True`, the y-axis is log-scaled in cartesian coordinates.
    range_x: list of two numbers
        If provided, overrides auto-scaling on the x-axis in cartesian
        coordinates.
    range_y: list of two numbers
        If provided, overrides auto-scaling on the y-axis in cartesian
        coordinates.
    render_mode: str
        One of ``'auto'``, ``'svg'`` or ``'webgl'``, default ``'auto'`` Controls the
```

```
            browser API used to draw marks. ``'svg'`` is appropriate for figures of
            less than 1000 data points, and will allow for fully-vectorized output.
            ``'webgl'`` is likely necessary for acceptable performance above 1000
            points but rasterizes part of the output.  ``'auto'`` uses heuristics to
            choose the mode.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter_3d(data_frame=None, x=None, y=None, z=None, color=None, symbol=None,
size=None, text=None, hover_name=None, hover_data=None, custom_data=None, error_x=None,
error_x_minus=None, error_y=None, error_y_minus=None, error_z=None, error_z_minus=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
size_max=None, color_discrete_sequence=None, color_discrete_map=None,
color_continuous_scale=None, range_color=None, color_continuous_midpoint=None,
symbol_sequence=None, symbol_map=None, opacity=None, log_x=False, log_y=False,
log_z=False, range_x=None, range_y=None, range_z=None, title=None, template=None,
width=None, height=None)
            In a 3D scatter plot, each row of `data_frame` is represented by a
            symbol mark in 3D space.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the x axis in cartesian coordinates.
        y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the y axis in cartesian coordinates.
        z: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the z axis in cartesian coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        size: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign mark sizes.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        error_x: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size x-axis error bars. If `error_x_minus` is `None`, error bars will
            be symmetrical, otherwise `error_x` is used for the positive direction
            only.
        error_x_minus: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size x-axis error bars in the negative direction. Ignored if `error_x`
            is `None`.
        error_y: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size y-axis error bars. If `error_y_minus` is `None`, error bars will
            be symmetrical, otherwise `error_y` is used for the positive direction
            only.
        error_y_minus: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size y-axis error bars in the negative direction. Ignored if `error_y`
            is `None`.
        error_z: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size z-axis error bars. If `error_z_minus` is `None`, error bars will
            be symmetrical, otherwise `error_z` is used for the positive direction
            only.
        error_z_minus: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            size z-axis error bars in the negative direction. Ignored if `error_z`
            is `None`.
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
        animation_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            provide object-constancy across animation frames: rows with matching
            `animation_group`s will be treated as if they describe the same object
            in each frame.
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
            names, and the values should be lists of strings corresponding to the
            specific display order desired.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        size_max: int (default `20`)
            Set the maximum mark size when using `size`.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        symbol_sequence: list of str
```

```
        Strings should define valid plotly.js symbols. When `symbol` is set,
        values in that column are assigned symbols by cycling through
        `symbol_sequence` in the order described in `category_orders`, unless
        the value of `symbol` is a key in `symbol_map`.
    symbol_map: dict with str keys and str values (default `{}`)
        String values should define plotly.js symbols Used to override
        `symbol_sequence` to assign a specific symbols to marks corresponding
        with specific values. Keys in `symbol_map` should be values in the
        column denoted by `symbol`. Alternatively, if the values of `symbol`
        are valid symbol names, the string `'identity'` may be passed to cause
        them to be used directly.
    opacity: float
        Value between 0 and 1. Sets the opacity for markers.
    log_x: boolean (default `False`)
        If `True`, the x-axis is log-scaled in cartesian coordinates.
    log_y: boolean (default `False`)
        If `True`, the y-axis is log-scaled in cartesian coordinates.
    log_z: boolean (default `False`)
        If `True`, the z-axis is log-scaled in cartesian coordinates.
    range_x: list of two numbers
        If provided, overrides auto-scaling on the x-axis in cartesian
        coordinates.
    range_y: list of two numbers
        If provided, overrides auto-scaling on the y-axis in cartesian
        coordinates.
    range_z: list of two numbers
        If provided, overrides auto-scaling on the z-axis in cartesian
        coordinates.
    title: str
        The figure title.
    template: str or dict or plotly.graph_objects.layout.Template instance
        The figure template name (must be a key in plotly.io.templates) or
        definition.
    width: int (default `None`)
        The figure width in pixels.
    height: int (default `None`)
        The figure height in pixels.

    Returns
    -------
        plotly.graph_objects.Figure

scatter_geo(data_frame=None, lat=None, lon=None, locations=None, locationmode=None,
geojson=None, featureidkey=None, color=None, text=None, symbol=None, facet_row=None,
facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,
hover_name=None, hover_data=None, custom_data=None, size=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, symbol_sequence=None, symbol_map=None, opacity=None,
size_max=None, projection=None, scope=None, center=None, fitbounds=None,
basemap_visible=None, title=None, template=None, width=None, height=None)
            In a geographic scatter plot, each row of `data_frame` is represented
            by a symbol mark on a map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        lat: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to latitude on a map.
        lon: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to longitude on a map.
        locations: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are to be
            interpreted according to `locationmode` and mapped to
            longitude/latitude.
        locationmode: str
            One of 'ISO-3', 'USA-states', or 'country names' Determines the set of
            locations used to match entries in `locations` to regions on the map.
        geojson: GeoJSON-formatted dict
            Must contain a Polygon feature collection, with IDs, which are
            references from `locations`.
        featureidkey: str (default: `'id'`)
            Path to field in GeoJSON feature object with which to match the values
            passed in to `locations`.The most common alternative to the default is
            of the form `'properties.<key>`.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
```

```
    array_like object. Values from this column or array_like are used to
    assign color to marks.
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
symbol: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign symbols to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
size: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign mark sizes.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
color_continuous_scale: list of str
    Strings should define valid CSS-colors This list is used to build a
    continuous color scale when the column denoted by `color` contains
    numeric data. Various useful color scales are available in the
    `plotly.express.colors` submodules, specifically
    `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
    and `plotly.express.colors.cyclical`.
range_color: list of two numbers
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
size_max: int (default `20`)
    Set the maximum mark size when using `size`.
projection: str
    One of `'equirectangular'`, `'mercator'`, `'orthographic'`, `'natural
    earth'`, `'kavrayskiy7'`, `'miller'`, `'robinson'`, `'eckert4'`,
    `'azimuthal equal area'`, `'azimuthal equidistant'`, `'conic equal
    area'`, `'conic conformal'`, `'conic equidistant'`, `'gnomonic'`,
    `'stereographic'`, `'mollweide'`, `'hammer'`, `'transverse mercator'`,
    `'albers usa'`, `'winkel tripel'`, `'aitoff'`, or `'sinusoidal'`Default
    depends on `scope`.
scope: str (default `'world'`).
    One of `'world'`, `'usa'`, `'europe'`, `'asia'`, `'africa'`, `'north
    america'`, or `'south america'`Default is `'world'` unless `projection`
    is set to `'albers usa'`, which forces `'usa'`.
```

```
        center: dict
            Dict keys are ``'lat'`` and ``'lon'`` Sets the center point of the map.
        fitbounds: str (default `False`).
            One of `False`, `locations` or `geojson`.
        basemap_visible: bool
            Force the basemap visibility.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter_mapbox(data_frame=None, lat=None, lon=None, color=None, text=None,
hover_name=None, hover_data=None, custom_data=None, size=None, animation_frame=None,
animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, opacity=None, size_max=None, zoom=8, center=None,
mapbox_style=None, title=None, template=None, width=None, height=None)
            In a Mapbox scatter plot, each row of `data_frame` is represented by a
            symbol mark on a Mapbox map.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        lat: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to latitude on a map.
        lon: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks according to longitude on a map.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        size: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign mark sizes.
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
        animation_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            provide object-constancy across animation frames: rows with matching
            `animation_group`s will be treated as if they describe the same object
            in each frame.
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
            names, and the values should be lists of strings corresponding to the
            specific display order desired.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        size_max: int (default `20`)
```

```
            Set the maximum mark size when using `size`.
        zoom: int (default `8`)
            Between 0 and 20. Sets map zoom level.
        center: dict
            Dict keys are `'lat'` and `'lon'` Sets the center point of the map.
        mapbox_style: str (default `'basic'`, needs Mapbox API token)
            Identifier of base map style, some of which require a Mapbox API token
            to be set using `plotly.express.set_mapbox_access_token()`. Allowed
            values which do not require a Mapbox API token are `'open-street-map'`,
            `'white-bg'`, `'carto-positron'`, `'carto-darkmatter'`, `'stamen-
            terrain'`, `'stamen-toner'`, `'stamen-watercolor'`. Allowed values
            which do require a Mapbox API token are `'basic'`, `'streets'`,
            `'outdoors'`, `'light'`, `'dark'`, `'satellite'`, `'satellite-
            streets'`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter_matrix(data_frame=None, dimensions=None, color=None, symbol=None,
size=None, hover_name=None, hover_data=None, custom_data=None, category_orders=None,
labels=None, color_discrete_sequence=None, color_discrete_map=None,
color_continuous_scale=None, range_color=None, color_continuous_midpoint=None,
symbol_sequence=None, symbol_map=None, opacity=None, size_max=None, title=None,
template=None, width=None, height=None)
            In a scatter plot matrix (or SPLOM), each row of `data_frame` is
            represented by a multiple symbol marks, one in each cell of a grid of
            2D scatter plots, which plot each pair of `dimensions` against each
            other.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        dimensions: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are used for
            multidimensional visualization.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        size: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign mark sizes.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
```

```
        names, and the values should be lists of strings corresponding to the
        specific display order desired.
    labels: dict with str keys and str values (default `{}`)
        By default, column names are used in the figure for axis titles, legend
        entries and hovers. This parameter allows this to be overridden. The
        keys of this dict should correspond to column names, and the values
        should correspond to the desired label to be displayed.
    color_discrete_sequence: list of str
        Strings should define valid CSS-colors. When `color` is set and the
        values in the corresponding column are not numeric, values in that
        column are assigned colors by cycling through `color_discrete_sequence`
        in the order described in `category_orders`, unless the value of
        `color` is a key in `color_discrete_map`. Various useful color
        sequences are available in the `plotly.express.colors` submodules,
        specifically `plotly.express.colors.qualitative`.
    color_discrete_map: dict with str keys and str values (default `{}`)
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    color_continuous_scale: list of str
        Strings should define valid CSS-colors This list is used to build a
        continuous color scale when the column denoted by `color` contains
        numeric data. Various useful color scales are available in the
        `plotly.express.colors` submodules, specifically
        `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
        and `plotly.express.colors.cyclical`.
    range_color: list of two numbers
        If provided, overrides auto-scaling on the continuous color scale.
    color_continuous_midpoint: number (default `None`)
        If set, computes the bounds of the continuous color scale to have the
        desired midpoint. Setting this value is recommended when using
        `plotly.express.colors.diverging` color scales as the inputs to
        `color_continuous_scale`.
    symbol_sequence: list of str
        Strings should define valid plotly.js symbols. When `symbol` is set,
        values in that column are assigned symbols by cycling through
        `symbol_sequence` in the order described in `category_orders`, unless
        the value of `symbol` is a key in `symbol_map`.
    symbol_map: dict with str keys and str values (default `{}`)
        String values should define plotly.js symbols Used to override
```

```
            `symbol_sequence` to assign a specific symbols to marks corresponding
            with specific values. Keys in `symbol_map` should be values in the
            column denoted by `symbol`. Alternatively, if the values of `symbol`
            are valid symbol names, the string `'identity'` may be passed to cause
            them to be used directly.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        size_max: int (default `20`)
            Set the maximum mark size when using `size`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter_polar(data_frame=None, r=None, theta=None, color=None, symbol=None,
size=None, hover_name=None, hover_data=None, custom_data=None, text=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, color_continuous_scale=None,
range_color=None, color_continuous_midpoint=None, symbol_sequence=None,
symbol_map=None, opacity=None, direction='clockwise', start_angle=90, size_max=None,
range_r=None, range_theta=None, log_r=False, render_mode='auto', title=None,
template=None, width=None, height=None)
            In a polar scatter plot, each row of `data_frame` is represented by a
            symbol mark in polar coordinates.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        r: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the radial axis in polar coordinates.
        theta: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the angular axis in polar coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        size: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign mark sizes.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
        animation_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            provide object-constancy across animation frames: rows with matching
            `animation_group`s will be treated as if they describe the same object
            in each frame.
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
            names, and the values should be lists of strings corresponding to the
            specific display order desired.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
```

```
    If provided, overrides auto-scaling on the continuous color scale.
color_continuous_midpoint: number (default `None`)
    If set, computes the bounds of the continuous color scale to have the
    desired midpoint. Setting this value is recommended when using
    `plotly.express.colors.diverging` color scales as the inputs to
    `color_continuous_scale`.
symbol_sequence: list of str
    Strings should define valid plotly.js symbols. When `symbol` is set,
    values in that column are assigned symbols by cycling through
    `symbol_sequence` in the order described in `category_orders`, unless
    the value of `symbol` is a key in `symbol_map`.
symbol_map: dict with str keys and str values (default `{}`)
    String values should define plotly.js symbols Used to override
    `symbol_sequence` to assign a specific symbols to marks corresponding
    with specific values. Keys in `symbol_map` should be values in the
    column denoted by `symbol`. Alternatively, if the values of `symbol`
    are valid symbol names, the string `'identity'` may be passed to cause
    them to be used directly.
opacity: float
    Value between 0 and 1. Sets the opacity for markers.
direction: str
    One of '`counterclockwise'` or `'clockwise'`. Default is `'clockwise'`
    Sets the direction in which increasing values of the angular axis are
    drawn.
start_angle: int (default `90`)
    Sets start angle for the angular axis, with 0 being due east and 90
    being due north.
size_max: int (default `20`)
    Set the maximum mark size when using `size`.
range_r: list of two numbers
    If provided, overrides auto-scaling on the radial axis in polar
    coordinates.
range_theta: list of two numbers
    If provided, overrides auto-scaling on the angular axis in polar
    coordinates.
log_r: boolean (default `False`)
    If `True`, the radial axis is log-scaled in polar coordinates.
render_mode: str
    One of `'auto'`, `'svg'` or `'webgl'`, default `'auto'` Controls the
    browser API used to draw marks. `'svg`' is appropriate for figures of
    less than 1000 data points, and will allow for fully-vectorized output.
    `'webgl'` is likely necessary for acceptable performance above 1000
    points but rasterizes part of the output.  `'auto'` uses heuristics to
```

```
            choose the mode.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    scatter_ternary(data_frame=None, a=None, b=None, c=None, color=None, symbol=None,
size=None, text=None, hover_name=None, hover_data=None, custom_data=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, color_continuous_scale=None,
range_color=None, color_continuous_midpoint=None, symbol_sequence=None,
symbol_map=None, opacity=None, size_max=None, title=None, template=None, width=None,
height=None)
            In a ternary scatter plot, each row of `data_frame` is represented by a
            symbol mark in ternary coordinates.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        a: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the a axis in ternary coordinates.
        b: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the b axis in ternary coordinates.
        c: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            position marks along the c axis in ternary coordinates.
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        symbol: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign symbols to marks.
        size: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign mark sizes.
        text: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in the
            figure as text labels.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        animation_frame: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign marks to animation frames.
        animation_group: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            provide object-constancy across animation frames: rows with matching
            `animation_group`s will be treated as if they describe the same object
            in each frame.
        category_orders: dict with str keys and list of str values (default `{}`)
            By default, in Python 3.6+, the order of categorical values in axes,
            legends and facets depends on the order in which these values are first
            encountered in `data_frame` (and no order is guaranteed by default in
            Python below 3.6). This parameter is used to force a specific ordering
            of values per column. The keys of this dict should correspond to column
            names, and the values should be lists of strings corresponding to the
            specific display order desired.
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
```

```
            `color_continuous_scale`.
        symbol_sequence: list of str
            Strings should define valid plotly.js symbols. When `symbol` is set,
            values in that column are assigned symbols by cycling through
            `symbol_sequence` in the order described in `category_orders`, unless
            the value of `symbol` is a key in `symbol_map`.
        symbol_map: dict with str keys and str values (default `{}`)
            String values should define plotly.js symbols Used to override
            `symbol_sequence` to assign a specific symbols to marks corresponding
            with specific values. Keys in `symbol_map` should be values in the
            column denoted by `symbol`. Alternatively, if the values of `symbol`
            are valid symbol names, the string `'identity'` may be passed to cause
            them to be used directly.
        opacity: float
            Value between 0 and 1. Sets the opacity for markers.
        size_max: int (default `20`)
            Set the maximum mark size when using `size`.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    set_mapbox_access_token(token)
        Arguments:
            token: A Mapbox token to be used in `plotly.express.scatter_mapbox` and
`plotly.express.line_mapbox` figures. See        https://docs.mapbox.com/help/how-
mapbox-works/access-tokens/ for more details

    strip(data_frame=None, x=None, y=None, color=None, facet_row=None, facet_col=None,
facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None,
hover_data=None, custom_data=None, animation_frame=None, animation_group=None,
category_orders=None, labels=None, color_discrete_sequence=None,
color_discrete_map=None, orientation=None, stripmode=None, log_x=False, log_y=False,
range_x=None, range_y=None, title=None, template=None, width=None, height=None)
```

```
    In a strip plot each row of `data_frame` is represented as a jittered
    mark within categories.

Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
x: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the x axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates. Either `x` or
    `y` can optionally be a list of column references or array_likes,  in
    which case the data will be treated as if it were 'wide' rather than
    'long'.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
```

```
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
    should correspond to the desired label to be displayed.
```

```
color_discrete_sequence: list of str
    Strings should define valid CSS-colors. When `color` is set and the
    values in the corresponding column are not numeric, values in that
    column are assigned colors by cycling through `color_discrete_sequence`
    in the order described in `category_orders`, unless the value of
    `color` is a key in `color_discrete_map`. Various useful color
    sequences are available in the `plotly.express.colors` submodules,
    specifically `plotly.express.colors.qualitative`.
color_discrete_map: dict with str keys and str values (default `{}`)
    String values should define valid CSS-colors Used to override
    `color_discrete_sequence` to assign a specific colors to marks
    corresponding with specific values. Keys in `color_discrete_map` should
    be values in the column denoted by `color`. Alternatively, if the
    values of `color` are valid colors, the string `'identity'` may be
    passed to cause them to be used directly.
orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
    (default `'v'` if `x` and `y` are provided and both continous or both
    categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
    `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
    provided)
stripmode: str (default `'group'`)
    One of `'group'` or `'overlay'` In `'overlay'` mode, strips are on
    drawn top of one another. In `'group'` mode, strips are placed beside
    each other.
log_x: boolean (default `False`)
    If `True`, the x-axis is log-scaled in cartesian coordinates.
log_y: boolean (default `False`)
    If `True`, the y-axis is log-scaled in cartesian coordinates.
range_x: list of two numbers
    If provided, overrides auto-scaling on the x-axis in cartesian
    coordinates.
range_y: list of two numbers
    If provided, overrides auto-scaling on the y-axis in cartesian
    coordinates.
title: str
    The figure title.
template: str or dict or plotly.graph_objects.layout.Template instance
    The figure template name (must be a key in plotly.io.templates) or
    definition.
width: int (default `None`)
    The figure width in pixels.
height: int (default `None`)
    The figure height in pixels.
```

```
        Returns
        -------
            plotly.graph_objects.Figure

    sunburst(data_frame=None, names=None, values=None, parents=None, path=None,
ids=None, color=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, color_discrete_sequence=None, color_discrete_map=None,
hover_name=None, hover_data=None, custom_data=None, labels=None, title=None,
template=None, width=None, height=None, branchvalues=None, maxdepth=None)
            A sunburst plot represents hierarchial data as sectors laid out over
            several levels of concentric rings.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        names: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            labels for sectors.
        values: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set values associated to sectors.
        parents: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            parents in sunburst and treemap charts.
        path: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects List of columns names or columns of a rectangular
            dataframe defining the hierarchy of sectors, from root to leaves. An
            error is raised if path AND ids or parents is passed
        ids: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set ids of sectors
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
```

```
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        branchvalues: str
            'total' or 'remainder' Determines how the items in `values` are summed.
            Whenset to 'total', items in `values` are taken to be valueof all its
            descendants. When set to 'remainder', itemsin `values` corresponding to
            the root and the branches:sectors are taken to be the extra part not
            part of thesum of the values at their leaves.
        maxdepth: int
            Positive integer Sets the number of rendered sectors from any given
            `level`. Set `maxdepth` to -1 to render all thelevels in the hierarchy.

        Returns
        -------
            plotly.graph_objects.Figure

    timeline(data_frame=None, x_start=None, x_end=None, y=None, color=None,
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None,
facet_col_spacing=None, hover_name=None, hover_data=None, custom_data=None, text=None,
animation_frame=None, animation_group=None, category_orders=None, labels=None,
color_discrete_sequence=None, color_discrete_map=None, color_continuous_scale=None,
range_color=None, color_continuous_midpoint=None, opacity=None, range_x=None,
range_y=None, title=None, template=None, width=None, height=None)
            In a timeline plot, each row of `data_frame` is represented as a
rectangular
```

```
    mark on an x axis of type `date`, spanning from `x_start` to `x_end`.

Parameters
----------
data_frame: DataFrame or array-like or dict
    This argument needs to be passed for column names (and not keyword
    names) to be used. Array-like and dict are tranformed internally to a
    pandas DataFrame. Optional: if missing, a DataFrame gets constructed
    under the hood using the other arguments.
x_start: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. (required) Values from this column or array_like are
    used to position marks along the x axis in cartesian coordinates.
x_end: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. (required) Values from this column or array_like are
    used to position marks along the x axis in cartesian coordinates.
y: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    position marks along the y axis in cartesian coordinates.
color: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign color to marks.
facet_row: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the vertical direction.
facet_col: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to facetted subplots in the horizontal direction.
facet_col_wrap: int
    Maximum number of facet columns. Wraps the column variable at this
    width, so that the column facets span multiple rows. Ignored if 0, and
    forced to 0 if `facet_row` or a `marginal` is set.
facet_row_spacing: float between 0 and 1
    Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
    when facet_col_wrap is used.
facet_col_spacing: float between 0 and 1
    Spacing between facet columns, in paper units Default is 0.02.
hover_name: str or int or Series or array-like
```

```
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in bold
    in the hover tooltip.
hover_data: list of str or int, or Series or array-like, or dict
    Either a list of names of columns in `data_frame`, or pandas Series, or
    array_like objects or a dict with column names as keys, with values
    True (for default formatting) False (in order to remove this column
    from hover information), or a formatting string, for example ':.3f' or
    '|%a' or list-like data to appear in the hover tooltip or tuples with a
    bool or formatting string as first element, and list-like data to
    appear in hover as second element Values from these columns appear as
    extra data in the hover tooltip.
custom_data: list of str or int, or Series or array-like
    Either names of columns in `data_frame`, or pandas Series, or
    array_like objects Values from these columns are extra data, to be used
    in widgets or Dash callbacks for example. This data is not user-visible
    but is included in events emitted by the figure (lasso selection etc.)
text: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like appear in the
    figure as text labels.
animation_frame: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    assign marks to animation frames.
animation_group: str or int or Series or array-like
    Either a name of a column in `data_frame`, or a pandas Series or
    array_like object. Values from this column or array_like are used to
    provide object-constancy across animation frames: rows with matching
    `animation_group`s will be treated as if they describe the same object
    in each frame.
category_orders: dict with str keys and list of str values (default `{}`)
    By default, in Python 3.6+, the order of categorical values in axes,
    legends and facets depends on the order in which these values are first
    encountered in `data_frame` (and no order is guaranteed by default in
    Python below 3.6). This parameter is used to force a specific ordering
    of values per column. The keys of this dict should correspond to column
    names, and the values should be lists of strings corresponding to the
    specific display order desired.
labels: dict with str keys and str values (default `{}`)
    By default, column names are used in the figure for axis titles, legend
    entries and hovers. This parameter allows this to be overridden. The
    keys of this dict should correspond to column names, and the values
```

```
        should correspond to the desired label to be displayed.
    color_discrete_sequence: list of str
        Strings should define valid CSS-colors. When `color` is set and the
        values in the corresponding column are not numeric, values in that
        column are assigned colors by cycling through `color_discrete_sequence`
        in the order described in `category_orders`, unless the value of
        `color` is a key in `color_discrete_map`. Various useful color
        sequences are available in the `plotly.express.colors` submodules,
        specifically `plotly.express.colors.qualitative`.
    color_discrete_map: dict with str keys and str values (default `{}`)
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    color_continuous_scale: list of str
        Strings should define valid CSS-colors This list is used to build a
        continuous color scale when the column denoted by `color` contains
        numeric data. Various useful color scales are available in the
        `plotly.express.colors` submodules, specifically
        `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
        and `plotly.express.colors.cyclical`.
    range_color: list of two numbers
        If provided, overrides auto-scaling on the continuous color scale.
    color_continuous_midpoint: number (default `None`)
        If set, computes the bounds of the continuous color scale to have the
        desired midpoint. Setting this value is recommended when using
        `plotly.express.colors.diverging` color scales as the inputs to
        `color_continuous_scale`.
    opacity: float
        Value between 0 and 1. Sets the opacity for markers.
    range_x: list of two numbers
        If provided, overrides auto-scaling on the x-axis in cartesian
        coordinates.
    range_y: list of two numbers
        If provided, overrides auto-scaling on the y-axis in cartesian
        coordinates.
    title: str
        The figure title.
    template: str or dict or plotly.graph_objects.layout.Template instance
        The figure template name (must be a key in plotly.io.templates) or
        definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

    treemap(data_frame=None, names=None, values=None, parents=None, ids=None,
path=None, color=None, color_continuous_scale=None, range_color=None,
color_continuous_midpoint=None, color_discrete_sequence=None, color_discrete_map=None,
hover_name=None, hover_data=None, custom_data=None, labels=None, title=None,
template=None, width=None, height=None, branchvalues=None, maxdepth=None)
            A treemap plot represents hierarchial data as nested rectangular
            sectors.

        Parameters
        ----------
        data_frame: DataFrame or array-like or dict
            This argument needs to be passed for column names (and not keyword
            names) to be used. Array-like and dict are tranformed internally to a
            pandas DataFrame. Optional: if missing, a DataFrame gets constructed
            under the hood using the other arguments.
        names: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            labels for sectors.
        values: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set values associated to sectors.
        parents: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used as
            parents in sunburst and treemap charts.
        ids: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            set ids of sectors
        path: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects List of columns names or columns of a rectangular
            dataframe defining the hierarchy of sectors, from root to leaves. An
            error is raised if path AND ids or parents is passed
        color: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like are used to
            assign color to marks.
        color_continuous_scale: list of str
            Strings should define valid CSS-colors This list is used to build a
            continuous color scale when the column denoted by `color` contains
            numeric data. Various useful color scales are available in the
            `plotly.express.colors` submodules, specifically
            `plotly.express.colors.sequential`, `plotly.express.colors.diverging`
            and `plotly.express.colors.cyclical`.
        range_color: list of two numbers
            If provided, overrides auto-scaling on the continuous color scale.
        color_continuous_midpoint: number (default `None`)
            If set, computes the bounds of the continuous color scale to have the
            desired midpoint. Setting this value is recommended when using
            `plotly.express.colors.diverging` color scales as the inputs to
            `color_continuous_scale`.
        color_discrete_sequence: list of str
            Strings should define valid CSS-colors. When `color` is set and the
            values in the corresponding column are not numeric, values in that
            column are assigned colors by cycling through `color_discrete_sequence`
            in the order described in `category_orders`, unless the value of
            `color` is a key in `color_discrete_map`. Various useful color
            sequences are available in the `plotly.express.colors` submodules,
            specifically `plotly.express.colors.qualitative`.
        color_discrete_map: dict with str keys and str values (default `{}`)
            String values should define valid CSS-colors Used to override
            `color_discrete_sequence` to assign a specific colors to marks
            corresponding with specific values. Keys in `color_discrete_map` should
            be values in the column denoted by `color`. Alternatively, if the
            values of `color` are valid colors, the string `'identity'` may be
            passed to cause them to be used directly.
        hover_name: str or int or Series or array-like
            Either a name of a column in `data_frame`, or a pandas Series or
            array_like object. Values from this column or array_like appear in bold
            in the hover tooltip.
        hover_data: list of str or int, or Series or array-like, or dict
            Either a list of names of columns in `data_frame`, or pandas Series, or
            array_like objects or a dict with column names as keys, with values
            True (for default formatting) False (in order to remove this column
            from hover information), or a formatting string, for example ':.3f' or
            '|%a' or list-like data to appear in the hover tooltip or tuples with a
            bool or formatting string as first element, and list-like data to
            appear in hover as second element Values from these columns appear as
            extra data in the hover tooltip.
        custom_data: list of str or int, or Series or array-like
            Either names of columns in `data_frame`, or pandas Series, or
            array_like objects Values from these columns are extra data, to be used
            in widgets or Dash callbacks for example. This data is not user-visible
            but is included in events emitted by the figure (lasso selection etc.)
        labels: dict with str keys and str values (default `{}`)
            By default, column names are used in the figure for axis titles, legend
            entries and hovers. This parameter allows this to be overridden. The
            keys of this dict should correspond to column names, and the values
            should correspond to the desired label to be displayed.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.
        branchvalues: str
            'total' or 'remainder' Determines how the items in `values` are summed.
            Whenset to 'total', items in `values` are taken to be valueof all its
            descendants. When set to 'remainder', itemsin `values` corresponding to
            the root and the branches:sectors are taken to be the extra part not
            part of thesum of the values at their leaves.
        maxdepth: int
            Positive integer Sets the number of rendered sectors from any given
            `level`. Set `maxdepth` to -1 to render all thelevels in the hierarchy.

        Returns
        -------
            plotly.graph_objects.Figure
```

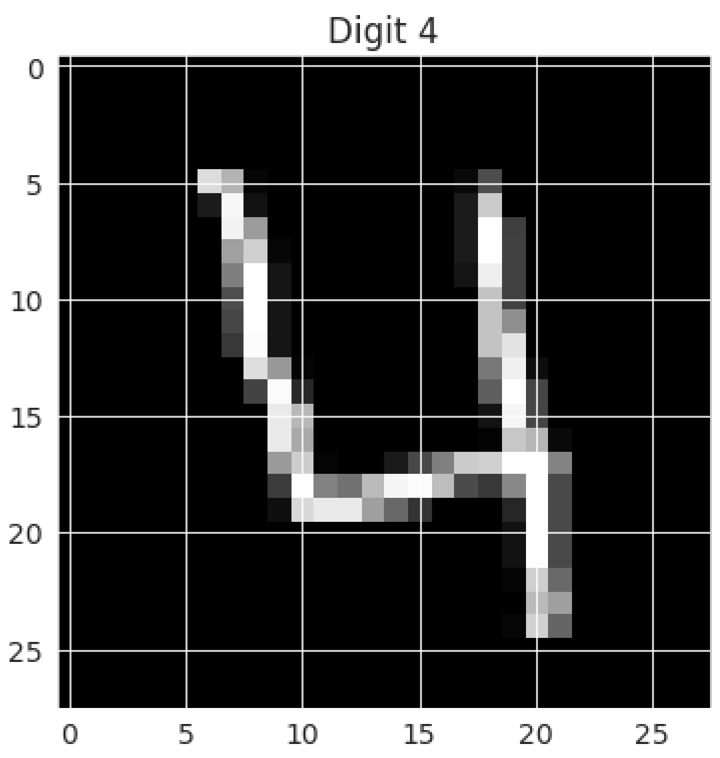
violin(data_frame=None, x=None, y=None, color=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None, hover_data=None, custom_data=None, animation_frame=None, animation_group=None, category_orders=None, labels=None, color_discrete_sequence=None, color_discrete_map=None, orientation=None, violinmode=None, log_x=False, log_y=False,

```
range_x=None, range_y=None, points=None, box=False, title=None, template=None,
width=None, height=None)
        In a violin plot, rows of `data_frame` are grouped together into a
        curved mark to visualize their distribution.

    Parameters
    ----------
    data_frame: DataFrame or array-like or dict
        This argument needs to be passed for column names (and not keyword
        names) to be used. Array-like and dict are tranformed internally to a
        pandas DataFrame. Optional: if missing, a DataFrame gets constructed
        under the hood using the other arguments.
    x: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        position marks along the x axis in cartesian coordinates. Either `x` or
        `y` can optionally be a list of column references or array_likes,  in
        which case the data will be treated as if it were 'wide' rather than
        'long'.
    y: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        position marks along the y axis in cartesian coordinates. Either `x` or
        `y` can optionally be a list of column references or array_likes,  in
        which case the data will be treated as if it were 'wide' rather than
        'long'.
    color: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign color to marks.
    facet_row: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to facetted subplots in the vertical direction.
    facet_col: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to facetted subplots in the horizontal direction.
    facet_col_wrap: int
        Maximum number of facet columns. Wraps the column variable at this
        width, so that the column facets span multiple rows. Ignored if 0, and
        forced to 0 if `facet_row` or a `marginal` is set.
    facet_row_spacing: float between 0 and 1
        Spacing between facet rows, in paper units. Default is 0.03 or 0.0.7
        when facet_col_wrap is used.
    facet_col_spacing: float between 0 and 1
        Spacing between facet columns, in paper units Default is 0.02.
    hover_name: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like appear in bold
        in the hover tooltip.
    hover_data: list of str or int, or Series or array-like, or dict
        Either a list of names of columns in `data_frame`, or pandas Series, or
        array_like objects or a dict with column names as keys, with values
        True (for default formatting) False (in order to remove this column
        from hover information), or a formatting string, for example ':.3f' or
        '|%a' or list-like data to appear in the hover tooltip or tuples with a
        bool or formatting string as first element, and list-like data to
        appear in hover as second element Values from these columns appear as
        extra data in the hover tooltip.
    custom_data: list of str or int, or Series or array-like
        Either names of columns in `data_frame`, or pandas Series, or
        array_like objects Values from these columns are extra data, to be used
        in widgets or Dash callbacks for example. This data is not user-visible
        but is included in events emitted by the figure (lasso selection etc.)
    animation_frame: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        assign marks to animation frames.
    animation_group: str or int or Series or array-like
        Either a name of a column in `data_frame`, or a pandas Series or
        array_like object. Values from this column or array_like are used to
        provide object-constancy across animation frames: rows with matching
        `animation_group`s will be treated as if they describe the same object
        in each frame.
    category_orders: dict with str keys and list of str values (default `{}`)
        By default, in Python 3.6+, the order of categorical values in axes,
        legends and facets depends on the order in which these values are first
        encountered in `data_frame` (and no order is guaranteed by default in
        Python below 3.6). This parameter is used to force a specific ordering
        of values per column. The keys of this dict should correspond to column
        names, and the values should be lists of strings corresponding to the
        specific display order desired.
    labels: dict with str keys and str values (default `{}`)
        By default, column names are used in the figure for axis titles, legend
        entries and hovers. This parameter allows this to be overridden. The
        keys of this dict should correspond to column names, and the values
        should correspond to the desired label to be displayed.
    color_discrete_sequence: list of str
        Strings should define valid CSS-colors. When `color` is set and the
        values in the corresponding column are not numeric, values in that
        column are assigned colors by cycling through `color_discrete_sequence`
        in the order described in `category_orders`, unless the value of
        `color` is a key in `color_discrete_map`. Various useful color
        sequences are available in the `plotly.express.colors` submodules,
        specifically `plotly.express.colors.qualitative`.
    color_discrete_map: dict with str keys and str values (default `{}`)
        String values should define valid CSS-colors Used to override
        `color_discrete_sequence` to assign a specific colors to marks
        corresponding with specific values. Keys in `color_discrete_map` should
        be values in the column denoted by `color`. Alternatively, if the
        values of `color` are valid colors, the string `'identity'` may be
        passed to cause them to be used directly.
    orientation: str, one of `'h'` for horizontal or `'v'` for vertical.
        (default `'v'` if `x` and `y` are provided and both continous or both
        categorical,  otherwise `'v'`(`'h'`) if `x`(`y`) is categorical and
        `y`(`x`) is continuous,  otherwise `'v'`(`'h'`) if only `x`(`y`) is
        provided)
    violinmode: str (default `'group'`)
        One of `'group'` or `'overlay'` In `'overlay'` mode, violins are on
        drawn top of one another. In `'group'` mode, violins are placed beside
        each other.
    log_x: boolean (default `False`)
        If `True`, the x-axis is log-scaled in cartesian coordinates.
    log_y: boolean (default `False`)
        If `True`, the y-axis is log-scaled in cartesian coordinates.
    range_x: list of two numbers
        If provided, overrides auto-scaling on the x-axis in cartesian
        coordinates.
    range_y: list of two numbers
        If provided, overrides auto-scaling on the y-axis in cartesian
        coordinates.
    points: str or boolean (default `'outliers'`)
        One of `'outliers'`, `'suspectedoutliers'`, `'all'`, or `False`. If
        `'outliers'`, only the sample points lying outside the whiskers are
        shown. If `'suspectedoutliers'`, all outlier points are shown and those
        less than 4*Q1-3*Q3 or greater than 4*Q3-3*Q1 are highlighted with the
        marker's `'outliercolor'`. If `'outliers'`, only the sample points
        lying outside the whiskers are shown. If `'all'`, all sample points are
```

```
            shown. If `False`, no sample points are shown and the whiskers extend
            to the full range of the sample.
        box: boolean (default `False`)
            If `True`, boxes are drawn inside the violins.
        title: str
            The figure title.
        template: str or dict or plotly.graph_objects.layout.Template instance
            The figure template name (must be a key in plotly.io.templates) or
            definition.
        width: int (default `None`)
            The figure width in pixels.
        height: int (default `None`)
            The figure height in pixels.

        Returns
        -------
            plotly.graph_objects.Figure

DATA
    NO_COLOR = 'px_no_color_constant'
    __all__ = ['scatter', 'scatter_3d', 'scatter_polar', 'scatter_ternary'...

FILE
    /usr/local/lib/python3.7/dist-packages/plotly/express/__init__.py
```

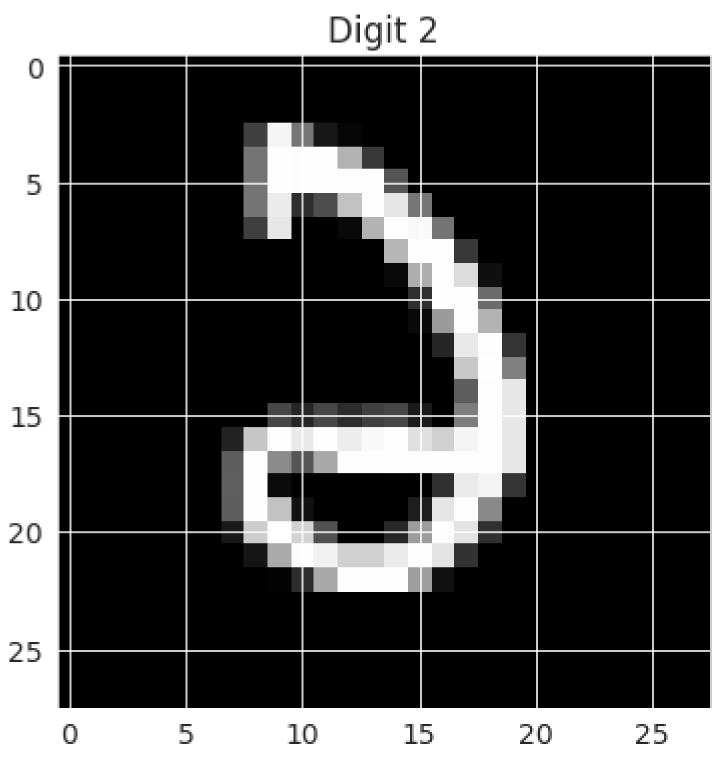
```python
four = train_df.iloc[3, 1:]
four.shape
four = four.values.reshape(28,28)
plt.imshow(four, cmap='gray')
plt.title("Digit 4")
```

```
Text(0.5, 1.0, 'Digit 4')
```



```
four = train_df.iloc[112, 1:]
four.shape
four = four.values.reshape(28,28)
plt.imshow(four, cmap='gray')
plt.title("Digit 2")
```

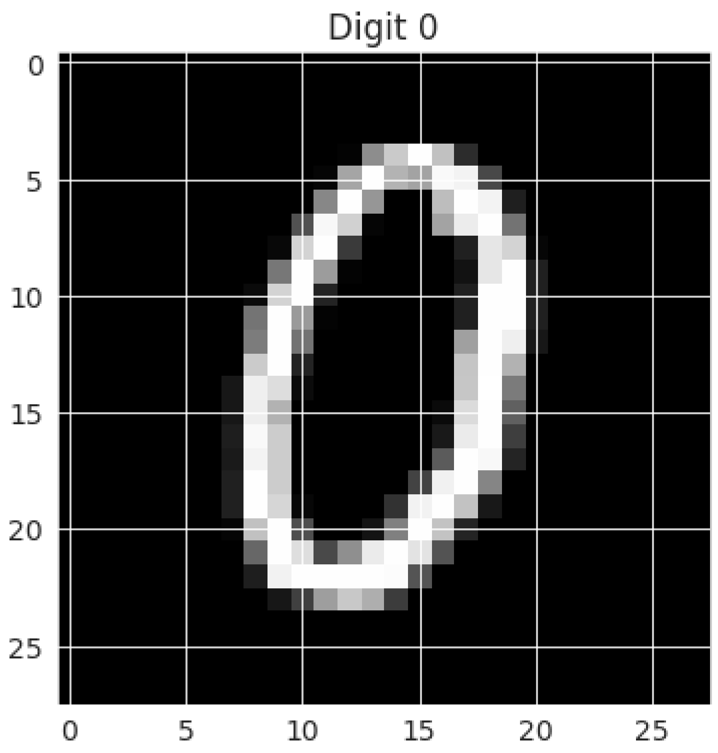
```
Text(0.5, 1.0, 'Digit 2')
```



```
seven = train_df.iloc[5, 1:]
seven.shape
seven = seven.values.reshape(28, 28)
```

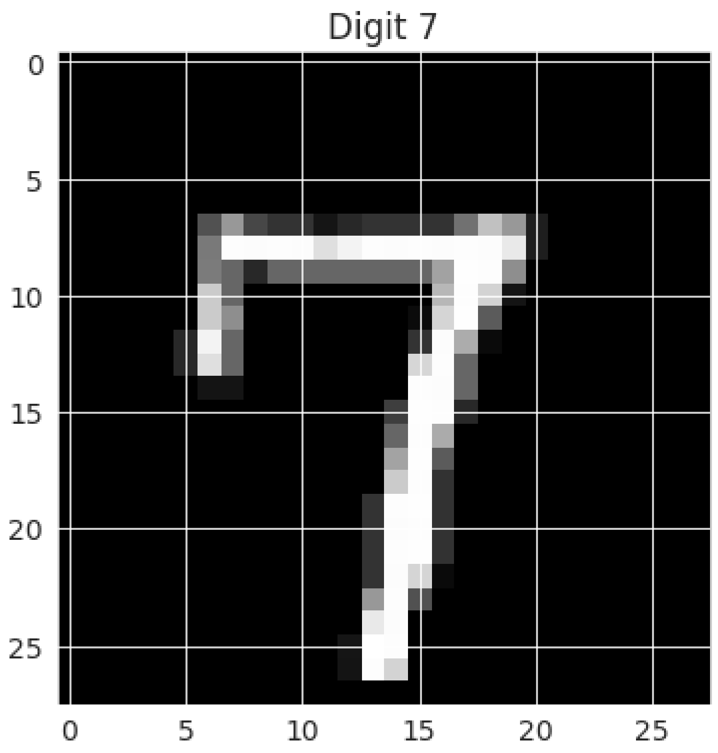
```
plt.imshow(seven, cmap='gray')
plt.title("Digit 0")
```

```
Text(0.5, 1.0, 'Digit 0')
```



```
seven = train_df.iloc[6, 1:]
seven.shape
seven = seven.values.reshape(28, 28)
plt.imshow(seven, cmap='gray')
plt.title("Digit 7")
```

```
Text(0.5, 1.0, 'Digit 7')
```



```
(test_df.isna().sum()!=0).sum()
```

```
0
```

```
input_col=list(train_df.columns)[1:]
target_col='label'
```

```
print(input_col)
```

```
['pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5', 'pixel6', 'pixel7',
'pixel8', 'pixel9', 'pixel10', 'pixel11', 'pixel12', 'pixel13', 'pixel14', 'pixel15',
'pixel16', 'pixel17', 'pixel18', 'pixel19', 'pixel20', 'pixel21', 'pixel22', 'pixel23',
'pixel24', 'pixel25', 'pixel26', 'pixel27', 'pixel28', 'pixel29', 'pixel30', 'pixel31',
'pixel32', 'pixel33', 'pixel34', 'pixel35', 'pixel36', 'pixel37', 'pixel38', 'pixel39',
'pixel40', 'pixel41', 'pixel42', 'pixel43', 'pixel44', 'pixel45', 'pixel46', 'pixel47',
'pixel48', 'pixel49', 'pixel50', 'pixel51', 'pixel52', 'pixel53', 'pixel54', 'pixel55',
'pixel56', 'pixel57', 'pixel58', 'pixel59', 'pixel60', 'pixel61', 'pixel62', 'pixel63',
'pixel64', 'pixel65', 'pixel66', 'pixel67', 'pixel68', 'pixel69', 'pixel70', 'pixel71',
'pixel72', 'pixel73', 'pixel74', 'pixel75', 'pixel76', 'pixel77', 'pixel78', 'pixel79',
'pixel80', 'pixel81', 'pixel82', 'pixel83', 'pixel84', 'pixel85', 'pixel86', 'pixel87',
'pixel88', 'pixel89', 'pixel90', 'pixel91', 'pixel92', 'pixel93', 'pixel94', 'pixel95',
'pixel96', 'pixel97', 'pixel98', 'pixel99', 'pixel100', 'pixel101', 'pixel102',
'pixel103', 'pixel104', 'pixel105', 'pixel106', 'pixel107', 'pixel108', 'pixel109',
'pixel110', 'pixel111', 'pixel112', 'pixel113', 'pixel114', 'pixel115', 'pixel116',
'pixel117', 'pixel118', 'pixel119', 'pixel120', 'pixel121', 'pixel122', 'pixel123',
'pixel124', 'pixel125', 'pixel126', 'pixel127', 'pixel128', 'pixel129', 'pixel130',
'pixel131', 'pixel132', 'pixel133', 'pixel134', 'pixel135', 'pixel136', 'pixel137',
'pixel138', 'pixel139', 'pixel140', 'pixel141', 'pixel142', 'pixel143', 'pixel144',
'pixel145', 'pixel146', 'pixel147', 'pixel148', 'pixel149', 'pixel150', 'pixel151',
'pixel152', 'pixel153', 'pixel154', 'pixel155', 'pixel156', 'pixel157', 'pixel158',
'pixel159', 'pixel160', 'pixel161', 'pixel162', 'pixel163', 'pixel164', 'pixel165',
'pixel166', 'pixel167', 'pixel168', 'pixel169', 'pixel170', 'pixel171', 'pixel172',
'pixel173', 'pixel174', 'pixel175', 'pixel176', 'pixel177', 'pixel178', 'pixel179',
'pixel180', 'pixel181', 'pixel182', 'pixel183', 'pixel184', 'pixel185', 'pixel186',
'pixel187', 'pixel188', 'pixel189', 'pixel190', 'pixel191', 'pixel192', 'pixel193',
'pixel194', 'pixel195', 'pixel196', 'pixel197', 'pixel198', 'pixel199', 'pixel200',
'pixel201', 'pixel202', 'pixel203', 'pixel204', 'pixel205', 'pixel206', 'pixel207',
'pixel208', 'pixel209', 'pixel210', 'pixel211', 'pixel212', 'pixel213', 'pixel214',
'pixel215', 'pixel216', 'pixel217', 'pixel218', 'pixel219', 'pixel220', 'pixel221',
'pixel222', 'pixel223', 'pixel224', 'pixel225', 'pixel226', 'pixel227', 'pixel228',
'pixel229', 'pixel230', 'pixel231', 'pixel232', 'pixel233', 'pixel234', 'pixel235',
'pixel236', 'pixel237', 'pixel238', 'pixel239', 'pixel240', 'pixel241', 'pixel242',
'pixel243', 'pixel244', 'pixel245', 'pixel246', 'pixel247', 'pixel248', 'pixel249',
'pixel250', 'pixel251', 'pixel252', 'pixel253', 'pixel254', 'pixel255', 'pixel256',
'pixel257', 'pixel258', 'pixel259', 'pixel260', 'pixel261', 'pixel262', 'pixel263',
'pixel264', 'pixel265', 'pixel266', 'pixel267', 'pixel268', 'pixel269', 'pixel270',
'pixel271', 'pixel272', 'pixel273', 'pixel274', 'pixel275', 'pixel276', 'pixel277',
'pixel278', 'pixel279', 'pixel280', 'pixel281', 'pixel282', 'pixel283', 'pixel284',
'pixel285', 'pixel286', 'pixel287', 'pixel288', 'pixel289', 'pixel290', 'pixel291',
'pixel292', 'pixel293', 'pixel294', 'pixel295', 'pixel296', 'pixel297', 'pixel298',
'pixel299', 'pixel300', 'pixel301', 'pixel302', 'pixel303', 'pixel304', 'pixel305',
'pixel306', 'pixel307', 'pixel308', 'pixel309', 'pixel310', 'pixel311', 'pixel312',
'pixel313', 'pixel314', 'pixel315', 'pixel316', 'pixel317', 'pixel318', 'pixel319',
'pixel320', 'pixel321', 'pixel322', 'pixel323', 'pixel324', 'pixel325', 'pixel326',
'pixel327', 'pixel328', 'pixel329', 'pixel330', 'pixel331', 'pixel332', 'pixel333',
'pixel334', 'pixel335', 'pixel336', 'pixel337', 'pixel338', 'pixel339', 'pixel340',
'pixel341', 'pixel342', 'pixel343', 'pixel344', 'pixel345', 'pixel346', 'pixel347',
'pixel348', 'pixel349', 'pixel350', 'pixel351', 'pixel352', 'pixel353', 'pixel354',
'pixel355', 'pixel356', 'pixel357', 'pixel358', 'pixel359', 'pixel360', 'pixel361',
'pixel362', 'pixel363', 'pixel364', 'pixel365', 'pixel366', 'pixel367', 'pixel368',
'pixel369', 'pixel370', 'pixel371', 'pixel372', 'pixel373', 'pixel374', 'pixel375',
'pixel376', 'pixel377', 'pixel378', 'pixel379', 'pixel380', 'pixel381', 'pixel382',
'pixel383', 'pixel384', 'pixel385', 'pixel386', 'pixel387', 'pixel388', 'pixel389',
'pixel390', 'pixel391', 'pixel392', 'pixel393', 'pixel394', 'pixel395', 'pixel396',
'pixel397', 'pixel398', 'pixel399', 'pixel400', 'pixel401', 'pixel402', 'pixel403',
'pixel404', 'pixel405', 'pixel406', 'pixel407', 'pixel408', 'pixel409', 'pixel410',
'pixel411', 'pixel412', 'pixel413', 'pixel414', 'pixel415', 'pixel416', 'pixel417',
'pixel418', 'pixel419', 'pixel420', 'pixel421', 'pixel422', 'pixel423', 'pixel424',
'pixel425', 'pixel426', 'pixel427', 'pixel428', 'pixel429', 'pixel430', 'pixel431',
'pixel432', 'pixel433', 'pixel434', 'pixel435', 'pixel436', 'pixel437', 'pixel438',
'pixel439', 'pixel440', 'pixel441', 'pixel442', 'pixel443', 'pixel444', 'pixel445',
'pixel446', 'pixel447', 'pixel448', 'pixel449', 'pixel450', 'pixel451', 'pixel452',
'pixel453', 'pixel454', 'pixel455', 'pixel456', 'pixel457', 'pixel458', 'pixel459',
'pixel460', 'pixel461', 'pixel462', 'pixel463', 'pixel464', 'pixel465', 'pixel466',
'pixel467', 'pixel468', 'pixel469', 'pixel470', 'pixel471', 'pixel472', 'pixel473',
'pixel474', 'pixel475', 'pixel476', 'pixel477', 'pixel478', 'pixel479', 'pixel480',
'pixel481', 'pixel482', 'pixel483', 'pixel484', 'pixel485', 'pixel486', 'pixel487',
'pixel488', 'pixel489', 'pixel490', 'pixel491', 'pixel492', 'pixel493', 'pixel494',
'pixel495', 'pixel496', 'pixel497', 'pixel498', 'pixel499', 'pixel500', 'pixel501',
'pixel502', 'pixel503', 'pixel504', 'pixel505', 'pixel506', 'pixel507', 'pixel508',
'pixel509', 'pixel510', 'pixel511', 'pixel512', 'pixel513', 'pixel514', 'pixel515',
'pixel516', 'pixel517', 'pixel518', 'pixel519', 'pixel520', 'pixel521', 'pixel522',
'pixel523', 'pixel524', 'pixel525', 'pixel526', 'pixel527', 'pixel528', 'pixel529',
'pixel530', 'pixel531', 'pixel532', 'pixel533', 'pixel534', 'pixel535', 'pixel536',
'pixel537', 'pixel538', 'pixel539', 'pixel540', 'pixel541', 'pixel542', 'pixel543',
'pixel544', 'pixel545', 'pixel546', 'pixel547', 'pixel548', 'pixel549', 'pixel550',
'pixel551', 'pixel552', 'pixel553', 'pixel554', 'pixel555', 'pixel556', 'pixel557',
'pixel558', 'pixel559', 'pixel560', 'pixel561', 'pixel562', 'pixel563', 'pixel564',
'pixel565', 'pixel566', 'pixel567', 'pixel568', 'pixel569', 'pixel570', 'pixel571',
'pixel572', 'pixel573', 'pixel574', 'pixel575', 'pixel576', 'pixel577', 'pixel578',
'pixel579', 'pixel580', 'pixel581', 'pixel582', 'pixel583', 'pixel584', 'pixel585',
'pixel586', 'pixel587', 'pixel588', 'pixel589', 'pixel590', 'pixel591', 'pixel592',
'pixel593', 'pixel594', 'pixel595', 'pixel596', 'pixel597', 'pixel598', 'pixel599',
'pixel600', 'pixel601', 'pixel602', 'pixel603', 'pixel604', 'pixel605', 'pixel606',
'pixel607', 'pixel608', 'pixel609', 'pixel610', 'pixel611', 'pixel612', 'pixel613',
'pixel614', 'pixel615', 'pixel616', 'pixel617', 'pixel618', 'pixel619', 'pixel620',
'pixel621', 'pixel622', 'pixel623', 'pixel624', 'pixel625', 'pixel626', 'pixel627',
'pixel628', 'pixel629', 'pixel630', 'pixel631', 'pixel632', 'pixel633', 'pixel634',
'pixel635', 'pixel636', 'pixel637', 'pixel638', 'pixel639', 'pixel640', 'pixel641',
'pixel642', 'pixel643', 'pixel644', 'pixel645', 'pixel646', 'pixel647', 'pixel648',
'pixel649', 'pixel650', 'pixel651', 'pixel652', 'pixel653', 'pixel654', 'pixel655',
'pixel656', 'pixel657', 'pixel658', 'pixel659', 'pixel660', 'pixel661', 'pixel662',
'pixel663', 'pixel664', 'pixel665', 'pixel666', 'pixel667', 'pixel668', 'pixel669',
'pixel670', 'pixel671', 'pixel672', 'pixel673', 'pixel674', 'pixel675', 'pixel676',
'pixel677', 'pixel678', 'pixel679', 'pixel680', 'pixel681', 'pixel682', 'pixel683',
'pixel684', 'pixel685', 'pixel686', 'pixel687', 'pixel688', 'pixel689', 'pixel690',
'pixel691', 'pixel692', 'pixel693', 'pixel694', 'pixel695', 'pixel696', 'pixel697',
'pixel698', 'pixel699', 'pixel700', 'pixel701', 'pixel702', 'pixel703', 'pixel704',
'pixel705', 'pixel706', 'pixel707', 'pixel708', 'pixel709', 'pixel710', 'pixel711',
'pixel712', 'pixel713', 'pixel714', 'pixel715', 'pixel716', 'pixel717', 'pixel718',
'pixel719', 'pixel720', 'pixel721', 'pixel722', 'pixel723', 'pixel724', 'pixel725',
'pixel726', 'pixel727', 'pixel728', 'pixel729', 'pixel730', 'pixel731', 'pixel732',
'pixel733', 'pixel734', 'pixel735', 'pixel736', 'pixel737', 'pixel738', 'pixel739',
'pixel740', 'pixel741', 'pixel742', 'pixel743', 'pixel744', 'pixel745', 'pixel746',
'pixel747', 'pixel748', 'pixel749', 'pixel750', 'pixel751', 'pixel752', 'pixel753',
'pixel754', 'pixel755', 'pixel756', 'pixel757', 'pixel758', 'pixel759', 'pixel760',
'pixel761', 'pixel762', 'pixel763', 'pixel764', 'pixel765', 'pixel766', 'pixel767',
'pixel768', 'pixel769', 'pixel770', 'pixel771', 'pixel772', 'pixel773', 'pixel774',
'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779', 'pixel780', 'pixel781',
'pixel782', 'pixel783']
```

```
len(input_col)
```

```
784
```

```
target_col
```

```
'label'
```

```
x_train=train_df[input_col].copy()
y_train=train_df[target_col].copy()
```

```
x_train
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
41995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
41996	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
41997	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
41998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
41999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	

42000 rows × 784 columns

```
y_train
```

```
0        1
1        0
2        1
3        4
4        0
        ..
41995    0
41996    1
41997    7
41998    6
41999    9
Name: label, Length: 42000, dtype: int64
```

```
len(x_train.select_dtypes(include=np.number).columns.tolist())
```

```
784
```

```
len(x_train.select_dtypes(include='object').columns.tolist())
```

```
0
```

```
final_result.nunique()
```

```
ImageId    28000
Label          1
dtype: int64
```

Scaling Numerical Features

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler=MinMaxScaler()
```

```
scaler.fit(x_train)
```

```
MinMaxScaler()
```

```
x_train.describe().loc[['min','max']]
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixe
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.0	254.0	253.0	2

2 rows × 784 columns

```
x_train[list(x_train.columns)]=scaler.transform(x_train)
```

```
test_df[list(test_df.columns)]=scaler.transform(test_df)
```

```
x_train.describe().loc[['min','max']]
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixe
min	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
max	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	1.0	1.0	

2 rows × 784 columns

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Training ,validation and Test Sets

```
from sklearn.model_selection import train_test_split
```

```
train_df1,val_df,train_target1,val_target=train_test_split(x_train,y_train,test_size=0.
train_df1
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
34941	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
24433	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
24432	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
8832	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
30291	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6265	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
11284	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
38158	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
860	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
15795	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

33600 rows × 784 columns

```
train_target1
```

```
34941    6
24433    5
24432    3
8832     4
30291    7
        ..
6265     9
11284    9
38158    2
860      6
15795    0
Name: label, Length: 33600, dtype: int64
```

```
val_df
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
5457	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
38509	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
25536	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
31803	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
39863	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8388	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
29359	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
40276	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
18421	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4335	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

8400 rows × 784 columns

```
val_target
```

```
5457     8
38509    1
25536    9
31803    9
39863    8
        ..
8388     4
29359    9
40276    3
18421    0
4335     9
Name: label, Length: 8400, dtype: int64
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Feature engineering (PCA)

```
from sklearn.decomposition import PCA
```

```
pca=PCA(n_components=100,whiten=True,random_state=0)
```

```
pca.fit(train_df1)
```

```
PCA(n_components=100, random_state=0, whiten=True)
```

```
x_pca=pd.DataFrame(pca.transform(train_df1))
```

```
x_pca
```

	0	1	2	3	4	5	6	7	8	
0	-0.391219	0.618243	0.905939	1.510900	0.708962	-0.373727	-0.251591	-0.127486	-0.882974	-1.4042
1	0.066184	-0.931912	0.218047	-2.054992	-1.160201	-0.240248	-0.732072	-0.897596	-0.228508	1.5230

	0	1	2	3	4	5	6	7	8	
2	-0.527412	-0.710763	-2.087662	-0.010597	-0.006801	-0.810396	-0.868462	0.697773	0.676190	-0.9192
3	0.143042	2.237146	-0.461969	0.084014	0.253986	0.728195	-1.176522	-0.608893	-0.959525	1.4339
4	-1.184995	1.313060	-0.364298	-0.785105	-0.744517	0.744058	1.812118	0.509428	-0.039731	-0.0834
...	...	...	...	...	...	...	...	...	...	
33595	-0.311307	1.221371	1.028816	-1.261570	0.095184	0.840779	0.035839	-0.096468	1.200096	-2.5052
33596	-0.125762	1.137106	-0.224426	-0.356209	0.187499	-1.250354	-1.885924	-0.322159	0.794436	1.3543
33597	-0.048342	1.204703	1.316535	1.009733	0.671092	0.023980	0.088547	0.832885	1.512864	0.4943
33598	0.403934	0.037689	-0.321993	1.811440	0.455946	-1.546345	0.220041	0.429402	-0.287922	-1.7060
33599	1.893039	-0.514821	-1.216820	-0.263443	-2.259474	1.085811	0.233250	-1.385292	0.195615	1.1612

33600 rows × 100 columns

```
val_pca=pd.DataFrame(pca.transform(val_df))
```

```
val_pca
```

	0	1	2	3	4	5	6	7	8	
0	-0.748579	-0.560861	0.445139	-0.026135	1.039314	0.498973	0.501034	-0.883723	-1.268170	-0.23771
1	-1.615301	-0.841731	-0.058496	0.649196	-1.410457	-0.092844	-0.091268	-0.601753	0.203045	-0.38942
2	-0.926212	1.239800	-0.150015	-1.407186	0.164623	-0.755606	0.466046	0.136449	0.327561	-0.87395
3	-0.150085	1.607163	-0.975785	-0.759084	-0.275050	-0.469331	-0.612351	1.307957	0.587775	0.13220
4	-0.370607	-0.216628	0.792374	-1.313839	0.620894	0.271577	-1.170444	-1.216409	-0.800675	0.24155
...	...	...	...	...	...	...	...	...	...	.
8395	1.665549	0.653251	0.935562	0.845655	0.996062	-0.108207	-1.364993	0.095632	1.546244	-0.17147
8396	-0.411787	1.958058	-0.701423	-0.541352	0.691875	-0.613217	-0.679177	-0.356443	-0.100970	1.33853
8397	0.349724	-1.879109	-1.466483	-0.417150	-0.311808	-0.613813	0.372312	1.587043	0.347649	-1.47872
8398	0.881998	0.246716	0.788855	0.568901	-2.422851	0.244769	0.482822	-0.857607	0.517917	1.79357
8399	-0.445515	0.931556	-0.854881	0.812690	0.595628	0.660506	-0.548539	-0.630281	0.140393	1.63287

8400 rows × 100 columns

```
test_pca=pd.DataFrame(pca.transform(test_df))
```

```
test_pca
```

	0	1	2	3	4	5	6	7	8	
0	1.447045	-0.293526	1.734437	1.410040	0.740647	1.407807	0.585860	0.126452	-1.075259	0.9977
1	1.985568	-0.051512	0.357119	-0.200298	-2.354183	1.108465	2.169992	-0.573414	1.377558	1.7810
2	-0.874822	0.174878	0.344947	-0.195751	-0.445353	0.315245	-1.003626	-1.173962	-0.704332	-1.5470
3	-0.462036	1.156223	1.067774	0.917536	-0.597512	0.465337	-0.265690	0.328555	-0.347917	-0.8442
4	-0.172266	-1.430686	0.007074	0.340952	0.683171	0.150090	0.149259	2.293688	-1.255818	-0.0138

	0	1	2	3	4	5	6	7	8	
...	...	...	...	...	...	...	...	...	...	
27995	0.225878	1.598409	-1.663705	0.753488	1.115809	0.131282	-0.718937	0.530777	0.538540	0.8044
27996	-0.488908	1.228619	-1.138136	0.471071	-1.151911	-0.008038	-0.602870	1.728247	-0.179155	0.7612
27997	-0.224678	-1.234196	-1.965226	-1.163345	0.381071	0.084132	-0.357345	1.317338	-0.692443	0.1469
27998	-0.104362	1.204314	-0.608726	-0.852844	1.102240	-1.511585	-1.150187	1.036411	1.336573	0.8753
27999	1.009085	0.247740	1.835463	0.714453	0.478920	0.924038	-0.831238	-0.176924	0.473052	1.2078

28000 rows × 100 columns

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Training Logistic Regression

```
from sklearn.linear_model import LogisticRegression
model= LogisticRegression(solver='liblinear')
model.fit(x_pca,train_target1)
```

```
LogisticRegression(solver='liblinear')
```

```
model.score(val_pca,val_target)
```

```
0.9077380952380952
```

```
model.score(x_pca,train_target1)
```

```
0.9144642857142857
```

Support Vector Machine(SVM)

```
from sklearn.svm import SVC
```

```
svc=SVC(kernel='poly',random_state=42)
```

```
svc.fit(x_pca,train_target1)
```

```
SVC(kernel='poly', random_state=42)
```

```
svc.score(x_pca,train_target1)
```

```
0.9975892857142857
```

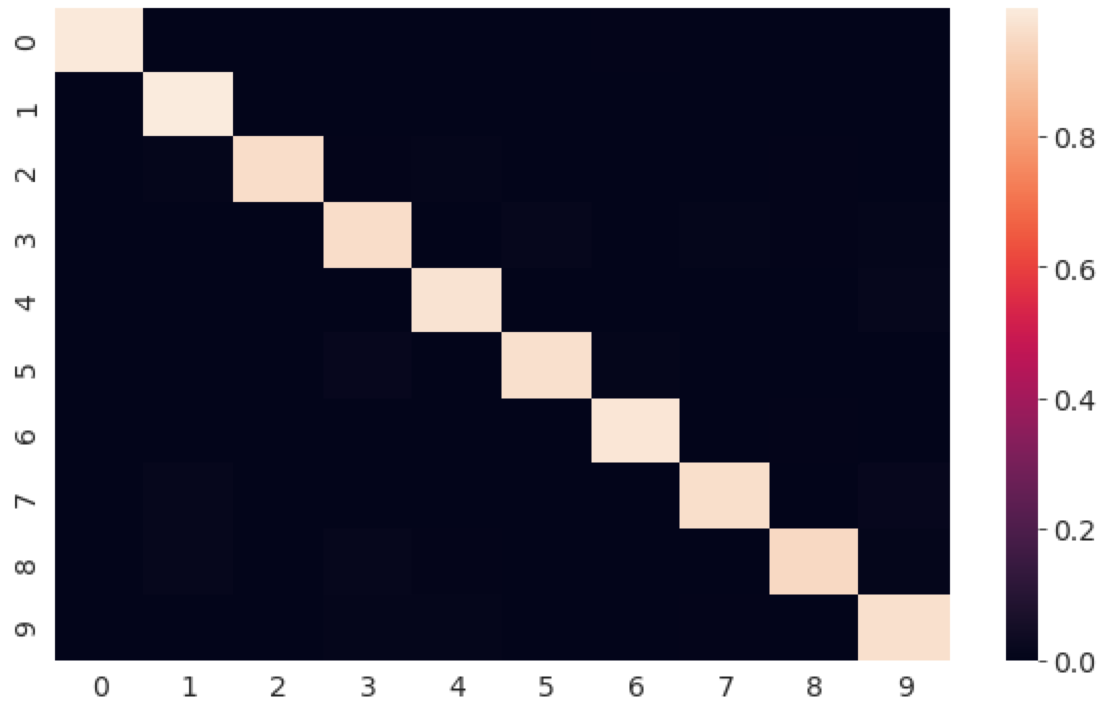
```
svc.score(val_pca,val_target)
```

```
0.9709523809523809
```

```
from sklearn.metrics import confusion_matrix
```

```
sns.heatmap(confusion_matrix(val_target,svc.predict(val_pca),normalize='true'))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcad49c9d90>
```



```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
tree=DecisionTreeClassifier(random_state=42)
```

```
tree.fit(x_pca,train_target1)
```

```
DecisionTreeClassifier(random_state=42)
```

```
tree.score(x_pca,train_target1)
```

```
1.0
```

```
tree.score(val_pca,val_target)
```

```
0.8221428571428572
```

```
pd.value_counts(tree.predict(val_pca))
```

```
1    909
3    893
7    887
9    862
2    853
4    850
8    830
0    811
6    783
5    722
dtype: int64
```

```
tree.tree_.max_depth
```

```
52
```

```
tree.feature_importances_
```

```
array([0.13637325, 0.10242484, 0.0545917 , 0.08728101, 0.08490173,
       0.07494858, 0.06650667, 0.05139944, 0.00777384, 0.02035688,
       0.00906935, 0.01989771, 0.03094434, 0.01402829, 0.01905037,
       0.02158237, 0.00656922, 0.00856178, 0.01244423, 0.00660766,
       0.0046089 , 0.0048251 , 0.00485058, 0.0037834 , 0.00269578,
       0.01048937, 0.00763017, 0.00229529, 0.00211022, 0.00217192,
       0.00227108, 0.00420487, 0.00283926, 0.00334479, 0.00240526,
       0.00440858, 0.00206295, 0.00183103, 0.00153994, 0.00130218,
       0.00266997, 0.0018926 , 0.00194218, 0.00248013, 0.00224939,
       0.00327276, 0.002641  , 0.00123888, 0.00133793, 0.00157153,
       0.0018541 , 0.00124594, 0.00240659, 0.00235347, 0.00158701,
       0.00085346, 0.00139144, 0.00131449, 0.001113  , 0.00219193,
       0.00126967, 0.00123083, 0.00190923, 0.0012924 , 0.00109728,
       0.00235039, 0.0013732 , 0.00202393, 0.00206919, 0.00181678,
       0.00119157, 0.00206496, 0.00065744, 0.00117611, 0.00125242,
       0.00160975, 0.00111603, 0.00124375, 0.00104156, 0.00166022,
       0.00113714, 0.00147476, 0.00108768, 0.00177918, 0.00138165,
       0.00121999, 0.00146933, 0.00211454, 0.00160983, 0.00114833,
       0.00102155, 0.00065628, 0.00104248, 0.00182754, 0.00092644,
       0.00150662, 0.00113949, 0.00109189, 0.00146921, 0.00085766])
```

Hyperparameter Tuning and Overfitting

```
?DecisionTreeClassifier
```

```
model=DecisionTreeClassifier(max_depth=15,random_state=42)
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

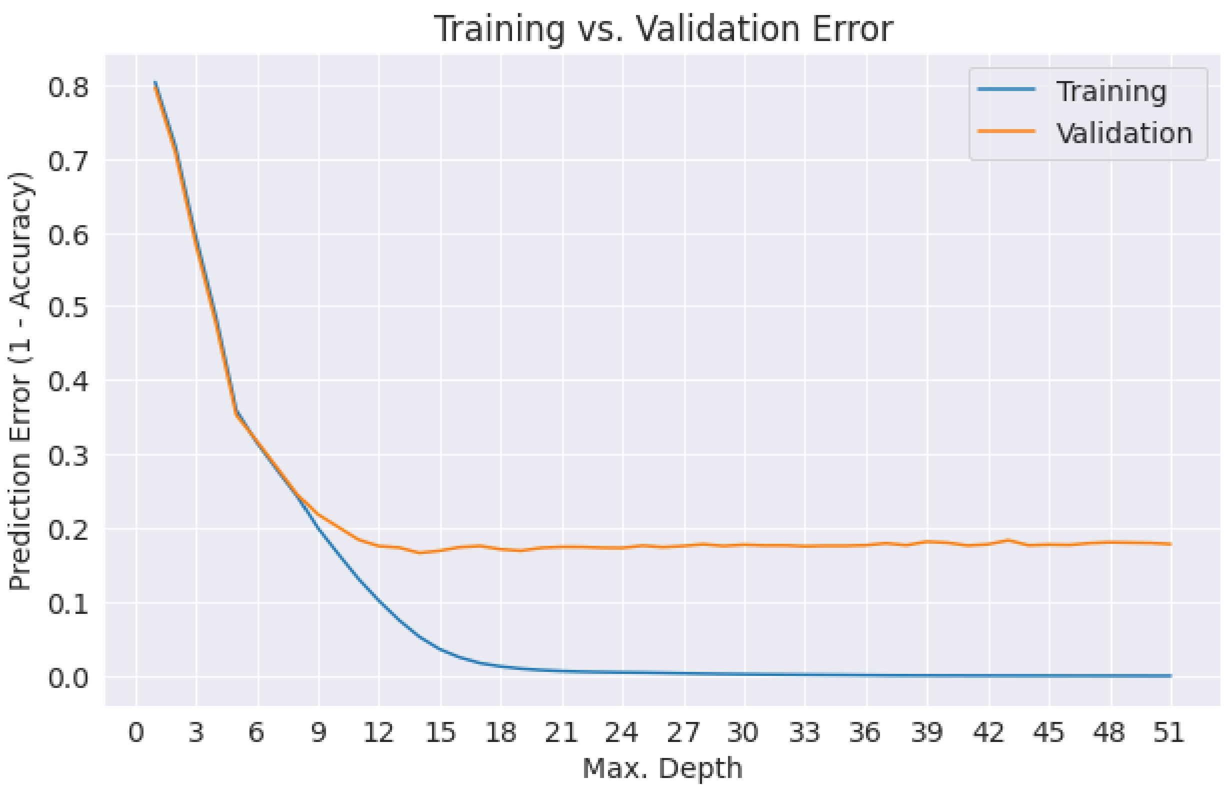
```
(0.9638690476190476, 0.830595238095238)
```

```
def max_depth_error(md):
    model=DecisionTreeClassifier(max_depth=md,random_state=42)
    model.fit(x_pca,train_target1)
    train_error=1- model.score(x_pca,train_target1)
    val_error=1- model.score(val_pca,val_target)
    return({'max_depth': md ,'training error': train_error,'val-error': val_error})
```

```
error_df=pd.DataFrame([max_depth_error(md) for md in range(1,52)])
```

```
plt.figure()
plt.plot(error_df['max_depth'], error_df['training error'])
plt.plot(error_df['max_depth'], error_df['val-error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,52, 3))
plt.xlabel('Max. Depth')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])
```

```
<matplotlib.legend.Legend at 0x7fcad4886510>
```



```
model=DecisionTreeClassifier(max_leaf_nodes=700,max_depth=15,random_state=42)
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

```
(0.9078273809523809, 0.8342857142857143)
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Naive-bayes

```
from sklearn.naive_bayes import GaussianNB # GaussianNB --87%  # MultinomialNB --Not wo
```

```
clf=GaussianNB()
clf.fit(x_pca,train_target1)
clf.score(x_pca,train_target1),clf.score(val_pca,val_target)
```

```
(0.8709821428571428, 0.8721428571428571)
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
```

```
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
model=KNeighborsClassifier(n_neighbors=3)
```

```
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

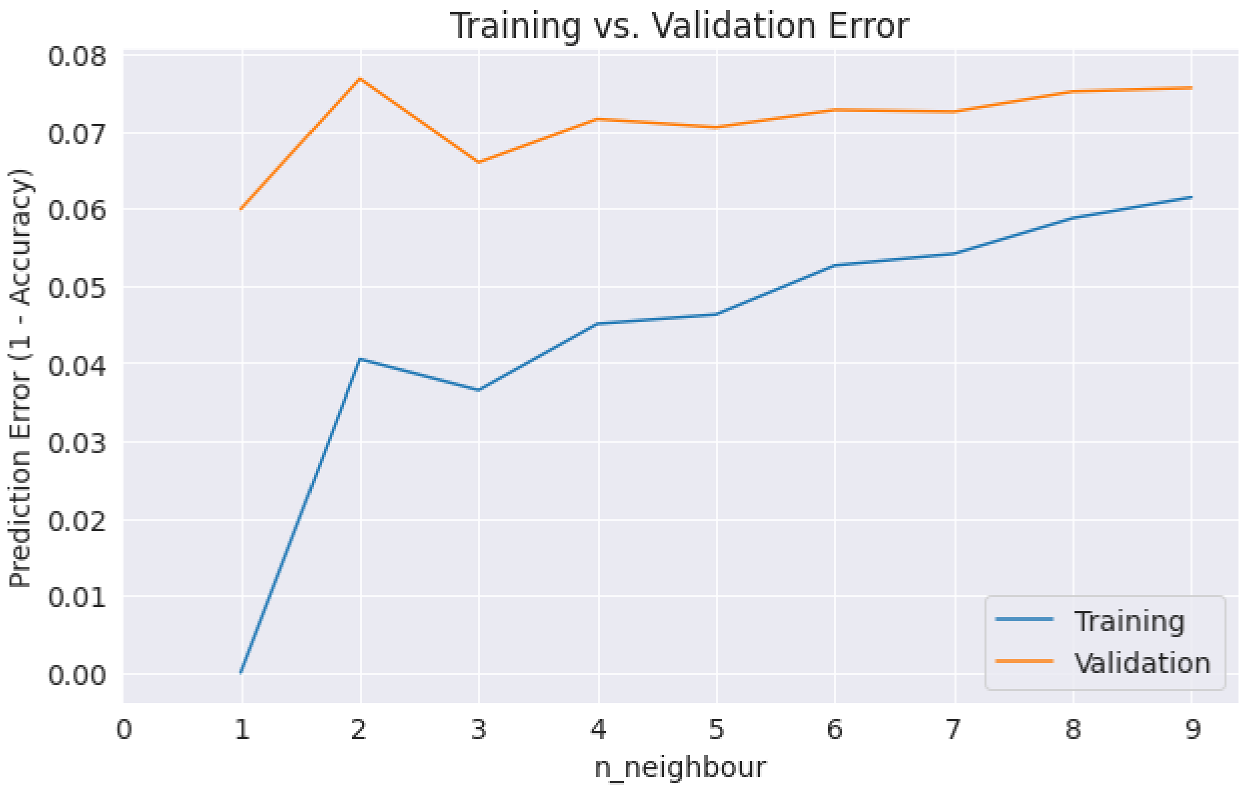
```
(0.9635416666666666, 0.934047619047619)
```

```
def test_func(md):
    clf=KNeighborsClassifier(n_neighbors=md)
    clf.fit(x_pca,train_target1)
    train_error=1- clf.score(x_pca,train_target1)
    val_error=1- clf.score(val_pca,val_target)
    return({'Neighbor': md ,'training error': train_error,'val-error': val_error})
```

```
error_df=pd.DataFrame([test_func(md) for md in range(1,10)])
```

```
plt.figure()
plt.plot(error_df['Neighbor'], error_df['training error'])
plt.plot(error_df['Neighbor'], error_df['val-error'])
plt.title('Training vs. Validation Error')
plt.xticks(range(0,10, 1))
plt.xlabel('n_neighbour')
plt.ylabel('Prediction Error (1 - Accuracy)')
plt.legend(['Training', 'Validation'])
```

```
<matplotlib.legend.Legend at 0x7fcad3427cd0>
```



```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
model=RandomForestClassifier(n_jobs=-1,random_state=42)
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

```
(1.0, 0.9430952380952381)
```

```
model.classes_
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
def test_params(**params):
    model=RandomForestClassifier(n_jobs=-1,random_state=42,**params)
    model.fit(x_pca,train_target1)
    return model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

```
test_params(max_depth=26)
```

```
(1.0, 0.9453571428571429)
```

```python
test_params(max_features='log2')
```

```
(1.0, 0.9439285714285715)
```

```python
test_params(min_samples_split=100, min_samples_leaf=60)
```

```
(0.9160119047619047, 0.8955952380952381)
```

```python
test_params(bootstrap=False)
```

```
(1.0, 0.9491666666666667)
```

```python
test_params(class_weight='balanced')
```

```
(1.0, 0.9453571428571429)
```

```python
test_params(n_estimators=200)
```

```
(1.0, 0.9470238095238095)
```

```python
help(model)
```

```
Help on RandomForestClassifier in module sklearn.ensemble._forest object:

class RandomForestClassifier(ForestClassifier)
 |  RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
 |
 |  A random forest classifier.
 |
 |  A random forest is a meta estimator that fits a number of decision tree
 |  classifiers on various sub-samples of the dataset and uses averaging to
 |  improve the predictive accuracy and control over-fitting.
 |  The sub-sample size is controlled with the `max_samples` parameter if
 |  `bootstrap=True` (default), otherwise the whole dataset is used to build
 |  each tree.
 |
 |  Read more in the :ref:`User Guide <forest>`.
 |
 |  Parameters
 |  ----------
 |  n_estimators : int, default=100
 |      The number of trees in the forest.
 |
 |      .. versionchanged:: 0.22
 |         The default value of ``n_estimators`` changed from 10 to 100
 |         in 0.22.
 |
 |  criterion : {"gini", "entropy"}, default="gini"
 |      The function to measure the quality of a split. Supported criteria are
 |      "gini" for the Gini impurity and "entropy" for the information gain.
 |      Note: this parameter is tree-specific.
 |
 |  max_depth : int, default=None
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
 |
 |  min_samples_split : int or float, default=2
 |      The minimum number of samples required to split an internal node:
 |
 |      - If int, then consider `min_samples_split` as the minimum number.
 |      - If float, then `min_samples_split` is a fraction and
 |        `ceil(min_samples_split * n_samples)` are the minimum
 |        number of samples for each split.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_samples_leaf : int or float, default=1
 |      The minimum number of samples required to be at a leaf node.
 |      A split point at any depth will only be considered if it leaves at
 |      least ``min_samples_leaf`` training samples in each of the left and
 |      right branches.  This may have the effect of smoothing the model,
 |      especially in regression.
 |
 |      - If int, then consider `min_samples_leaf` as the minimum number.
 |      - If float, then `min_samples_leaf` is a fraction and
 |        `ceil(min_samples_leaf * n_samples)` are the minimum
 |        number of samples for each node.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_weight_fraction_leaf : float, default=0.0
 |      The minimum weighted fraction of the sum total of weights (of all
 |      the input samples) required to be at a leaf node. Samples have
 |      equal weight when sample_weight is not provided.
 |
 |  max_features : {"auto", "sqrt", "log2"}, int or float, default="auto"
 |      The number of features to consider when looking for the best split:
 |
 |      - If int, then consider `max_features` features at each split.
 |      - If float, then `max_features` is a fraction and
 |        `round(max_features * n_features)` features are considered at each
 |        split.
 |      - If "auto", then `max_features=sqrt(n_features)`.
 |      - If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").
 |      - If "log2", then `max_features=log2(n_features)`.
 |      - If None, then `max_features=n_features`.
 |
 |      Note: the search for a split does not stop until at least one
 |      valid partition of the node samples is found, even if it requires to
 |      effectively inspect more than ``max_features`` features.
 |
 |  max_leaf_nodes : int, default=None
 |      Grow trees with ``max_leaf_nodes`` in best-first fashion.
 |      Best nodes are defined as relative reduction in impurity.
 |      If None then unlimited number of leaf nodes.
 |
 |  min_impurity_decrease : float, default=0.0
 |      A node will be split if this split induces a decrease of the impurity
 |      greater than or equal to this value.
 |
 |      The weighted impurity decrease equation is the following::
 |
 |          N_t / N * (impurity - N_t_R / N_t * right_impurity
 |                              - N_t_L / N_t * left_impurity)
 |
 |      where ``N`` is the total number of samples, ``N_t`` is the number of
 |      samples at the current node, ``N_t_L`` is the number of samples in the
 |      left child, and ``N_t_R`` is the number of samples in the right child.
 |
 |      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted sum,
 |      if ``sample_weight`` is passed.
 |
 |      .. versionadded:: 0.19
```

```
 |
 |  bootstrap : bool, default=True
 |      Whether bootstrap samples are used when building trees. If False, the
 |      whole dataset is used to build each tree.
 |
 |  oob_score : bool, default=False
 |      Whether to use out-of-bag samples to estimate the generalization score.
 |      Only available if bootstrap=True.
 |
 |  n_jobs : int, default=None
 |      The number of jobs to run in parallel. :meth:`fit`, :meth:`predict`,
 |      :meth:`decision_path` and :meth:`apply` are all parallelized over the
 |      trees. ``None`` means 1 unless in a :obj:`joblib.parallel_backend`
 |      context. ``-1`` means using all processors. See :term:`Glossary
 |      <n_jobs>` for more details.
 |
 |  random_state : int, RandomState instance or None, default=None
 |      Controls both the randomness of the bootstrapping of the samples used
 |      when building trees (if ``bootstrap=True``) and the sampling of the
 |      features to consider when looking for the best split at each node
 |      (if ``max_features < n_features``).
 |      See :term:`Glossary <random_state>` for details.
 |
 |  verbose : int, default=0
 |      Controls the verbosity when fitting and predicting.
 |
 |  warm_start : bool, default=False
 |      When set to ``True``, reuse the solution of the previous call to fit
 |      and add more estimators to the ensemble, otherwise, just fit a whole
 |      new forest. See :term:`the Glossary <warm_start>`.
 |
 |  class_weight : {"balanced", "balanced_subsample"}, dict or list of dicts,
default=None
 |      Weights associated with classes in the form ``{class_label: weight}``.
 |      If not given, all classes are supposed to have weight one. For
 |      multi-output problems, a list of dicts can be provided in the same
 |      order as the columns of y.
 |
 |      Note that for multioutput (including multilabel) weights should be
 |      defined for each class of every column in its own dict. For example,
 |      for four-class multilabel classification weights should be
 |      [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of
 |      [{1:1}, {2:5}, {3:1}, {4:1}].
```

```
|
|       The "balanced" mode uses the values of y to automatically adjust
|       weights inversely proportional to class frequencies in the input data
|       as ``n_samples / (n_classes * np.bincount(y))``
|
|       The "balanced_subsample" mode is the same as "balanced" except that
|       weights are computed based on the bootstrap sample for every tree
|       grown.
|
|       For multi-output, the weights of each column of y will be multiplied.
|
|       Note that these weights will be multiplied with sample_weight (passed
|       through the fit method) if sample_weight is specified.
|
|   ccp_alpha : non-negative float, default=0.0
|       Complexity parameter used for Minimal Cost-Complexity Pruning. The
|       subtree with the largest cost complexity that is smaller than
|       ``ccp_alpha`` will be chosen. By default, no pruning is performed. See
|       :ref:`minimal_cost_complexity_pruning` for details.
|
|       .. versionadded:: 0.22
|
|   max_samples : int or float, default=None
|       If bootstrap is True, the number of samples to draw from X
|       to train each base estimator.
|
|       - If None (default), then draw `X.shape[0]` samples.
|       - If int, then draw `max_samples` samples.
|       - If float, then draw `max_samples * X.shape[0]` samples. Thus,
|         `max_samples` should be in the interval `(0.0, 1.0]`.
|
|       .. versionadded:: 0.22
|
|   Attributes
|   ----------
|   base_estimator_ : DecisionTreeClassifier
|       The child estimator template used to create the collection of fitted
|       sub-estimators.
|
|   estimators_ : list of DecisionTreeClassifier
|       The collection of fitted sub-estimators.
|
|   classes_ : ndarray of shape (n_classes,) or a list of such arrays
```

```
 |          The classes labels (single output problem), or a list of arrays of
 |          class labels (multi-output problem).
 |
 |      n_classes_ : int or list
 |          The number of classes (single output problem), or a list containing the
 |          number of classes for each output (multi-output problem).
 |
 |      n_features_ : int
 |          The number of features when ``fit`` is performed.
 |
 |          .. deprecated:: 1.0
 |              Attribute `n_features_` was deprecated in version 1.0 and will be
 |              removed in 1.2. Use `n_features_in_` instead.
 |
 |      n_features_in_ : int
 |          Number of features seen during :term:`fit`.
 |
 |          .. versionadded:: 0.24
 |
 |      feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |          Names of features seen during :term:`fit`. Defined only when `X`
 |          has feature names that are all strings.
 |
 |          .. versionadded:: 1.0
 |
 |      n_outputs_ : int
 |          The number of outputs when ``fit`` is performed.
 |
 |      feature_importances_ : ndarray of shape (n_features,)
 |          The impurity-based feature importances.
 |          The higher, the more important the feature.
 |          The importance of a feature is computed as the (normalized)
 |          total reduction of the criterion brought by that feature.  It is also
 |          known as the Gini importance.
 |
 |          Warning: impurity-based feature importances can be misleading for
 |          high cardinality features (many unique values). See
 |          :func:`sklearn.inspection.permutation_importance` as an alternative.
 |
 |      oob_score_ : float
 |          Score of the training dataset obtained using an out-of-bag estimate.
 |          This attribute exists only when ``oob_score`` is True.
 |
 |  oob_decision_function_ : ndarray of shape (n_samples, n_classes) or
(n_samples, n_classes, n_outputs)
 |      Decision function computed with out-of-bag estimate on the training
 |      set. If n_estimators is small it might be possible that a data point
 |      was never left out during the bootstrap. In this case,
 |      `oob_decision_function_` might contain NaN. This attribute exists
 |      only when ``oob_score`` is True.
 |
 |  See Also
 |  --------
 |  sklearn.tree.DecisionTreeClassifier : A decision tree classifier.
 |  sklearn.ensemble.ExtraTreesClassifier : Ensemble of extremely randomized
 |      tree classifiers.
 |
 |  Notes
 |  -----
 |  The default values for the parameters controlling the size of the trees
 |  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
 |  unpruned trees which can potentially be very large on some data sets. To
 |  reduce memory consumption, the complexity and size of the trees should be
 |  controlled by setting those parameter values.
 |
 |  The features are always randomly permuted at each split. Therefore,
 |  the best found split may vary, even with the same training data,
 |  ``max_features=n_features`` and ``bootstrap=False``, if the improvement
 |  of the criterion is identical for several splits enumerated during the
 |  search of the best split. To obtain a deterministic behaviour during
 |  fitting, ``random_state`` has to be fixed.
 |
 |  References
 |  ----------
 |  .. [1] L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.
 |
 |  Examples
 |  --------
 |  >>> from sklearn.ensemble import RandomForestClassifier
 |  >>> from sklearn.datasets import make_classification
 |  >>> X, y = make_classification(n_samples=1000, n_features=4,
 |  ...                            n_informative=2, n_redundant=0,
 |  ...                            random_state=0, shuffle=False)
 |  >>> clf = RandomForestClassifier(max_depth=2, random_state=0)
 |  >>> clf.fit(X, y)
 |  RandomForestClassifier(...)
 |  >>> print(clf.predict([[0, 0, 0, 0]]))
 |  [1]
 |
 |  Method resolution order:
 |      RandomForestClassifier
 |      ForestClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseForest
 |      sklearn.base.MultiOutputMixin
 |      sklearn.ensemble._base.BaseEnsemble
 |      sklearn.base.MetaEstimatorMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from ForestClassifier:
 |
 |  predict(self, X)
 |      Predict class for X.
 |
 |      The predicted class of an input sample is a vote by the trees in
 |      the forest, weighted by their probability estimates. That is,
 |      the predicted class is the one with highest mean probability
 |      estimate across the trees.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      y : ndarray of shape (n_samples,) or (n_samples, n_outputs)
 |          The predicted classes.
 |
 |  predict_log_proba(self, X)
 |      Predict class log-probabilities for X.
 |
 |      The predicted class log-probabilities of an input sample is computed as
 |      the log of the mean predicted class probabilities of the trees in the
 |      forest.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      p : ndarray of shape (n_samples, n_classes), or a list of such arrays
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  predict_proba(self, X)
 |      Predict class probabilities for X.
 |
 |      The predicted class probabilities of an input sample are computed as
 |      the mean predicted class probabilities of the trees in the forest.
 |      The class probability of a single tree is the fraction of samples of
 |      the same class in a leaf.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input samples. Internally, its dtype will be converted to
 |          ``dtype=np.float32``. If a sparse matrix is provided, it will be
 |          converted into a sparse ``csr_matrix``.
 |
 |      Returns
 |      -------
 |      p : ndarray of shape (n_samples, n_classes), or a list of such arrays
 |          The class probabilities of the input samples. The order of the
 |          classes corresponds to that in the attribute :term:`classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True labels for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of ``self.predict(X)`` wrt. `y`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.ClassifierMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseForest:
```

```
|
|  apply(self, X)
|      Apply trees in the forest to X, return leaf indices.
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      X_leaves : ndarray of shape (n_samples, n_estimators)
|          For each datapoint x in X and for each tree in the forest,
|          return the index of the leaf x ends up in.
|
|  decision_path(self, X)
|      Return the decision path in the forest.
|
|      .. versionadded:: 0.18
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The input samples. Internally, its dtype will be converted to
|          ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      indicator : sparse matrix of shape (n_samples, n_nodes)
|          Return a node indicator matrix where non zero elements indicates
|          that the samples goes through the nodes. The matrix is of CSR
|          format.
|
|      n_nodes_ptr : ndarray of shape (n_estimators + 1,)
|          The columns from indicator[n_nodes_ptr[i]:n_nodes_ptr[i+1]]
|          gives the indicator value for the i-th estimator.
|
|  fit(self, X, y, sample_weight=None)
|      Build a forest of trees from the training set (X, y).
|
|      Parameters
|      ----------
|      X : {array-like, sparse matrix} of shape (n_samples, n_features)
|          The training input samples. Internally, its dtype will be converted
|          to ``dtype=np.float32``. If a sparse matrix is provided, it will be
|          converted into a sparse ``csc_matrix``.
|
|      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
|          The target values (class labels in classification, real numbers in
|          regression).
|
|      sample_weight : array-like of shape (n_samples,), default=None
|          Sample weights. If None, then samples are equally weighted. Splits
|          that would create child nodes with net zero or negative weight are
|          ignored while searching for a split in each node. In the case of
|          classification, splits are also ignored if they would result in any
|          single class carrying a negative weight in either child node.
|
|      Returns
|      -------
|      self : object
|          Fitted estimator.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from BaseForest:
|
|  feature_importances_
|      The impurity-based feature importances.
|
|      The higher, the more important the feature.
|      The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance.
|
|      Warning: impurity-based feature importances can be misleading for
|      high cardinality features (many unique values). See
|      :func:`sklearn.inspection.permutation_importance` as an alternative.
|
|      Returns
|      -------
|      feature_importances_ : ndarray of shape (n_features,)
|          The values of this array sum to 1, unless all trees are single node
|          trees consisting of only the root node, in which case it will be an
|          array of zeros.
|
|  n_features_
|      DEPRECATED: Attribute `n_features_` was deprecated in version 1.0 and will be
removed in 1.2. Use `n_features_in_` instead.
|
|      Number of features when fitting the estimator.
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.ensemble._base.BaseEnsemble:
|
|  __getitem__(self, index)
|      Return the index'th estimator in the ensemble.
|
|  __iter__(self)
|      Return iterator over estimators in the ensemble.
|
|  __len__(self)
|      Return the number of estimators in the ensemble.
|
|  ----------------------------------------------------------------------
|  Data and other attributes inherited from sklearn.ensemble._base.BaseEnsemble:
|
|  __annotations__ = {'_required_parameters': typing.List[str]}
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
|
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

xgboost

```
!pip install xgboost
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: xgboost in /usr/local/lib/python3.7/dist-packages (0.90)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from
xgboost) (1.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from
xgboost) (1.21.6)
```

```
from xgboost import XGBClassifier
```

```
model=XGBClassifier(random_state=42,n_jobs=-1,n_estimators=100,max_depth=10,learning_ra
```

```
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

```
(1.0, 0.9557142857142857)
```

```
model=XGBClassifier(random_state=42,n_jobs=-1,n_estimators=300,max_depth=10,learning_ra
```

```
model.fit(x_pca,train_target1)
model.score(x_pca,train_target1),model.score(val_pca,val_target)
```

```
(1.0, 0.9591666666666666)
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Neural network

```
from sklearn.neural_network import MLPClassifier
```

```
mlp=MLPClassifier(hidden_layer_sizes=(200,200,200),activation='relu',learning_rate_init
```

```
help(mlp)
```

```
Help on MLPClassifier in module sklearn.neural_network._multilayer_perceptron object:

class MLPClassifier(sklearn.base.ClassifierMixin, BaseMultilayerPerceptron)
 |  MLPClassifier(hidden_layer_sizes=(100,), activation='relu', *, solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001,
power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,
max_fun=15000)
 |
 |  Multi-layer Perceptron classifier.
 |
 |  This model optimizes the log-loss function using LBFGS or stochastic
 |  gradient descent.
 |
 |  .. versionadded:: 0.18
 |
 |  Parameters
 |  ----------
 |  hidden_layer_sizes : tuple, length = n_layers - 2, default=(100,)
 |      The ith element represents the number of neurons in the ith
 |      hidden layer.
 |
 |  activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
 |      Activation function for the hidden layer.
 |
 |      - 'identity', no-op activation, useful to implement linear bottleneck,
 |        returns f(x) = x
 |
 |      - 'logistic', the logistic sigmoid function,
 |        returns f(x) = 1 / (1 + exp(-x)).
 |
 |      - 'tanh', the hyperbolic tan function,
 |        returns f(x) = tanh(x).
 |
 |      - 'relu', the rectified linear unit function,
 |        returns f(x) = max(0, x)
 |
 |  solver : {'lbfgs', 'sgd', 'adam'}, default='adam'
 |      The solver for weight optimization.
 |
 |      - 'lbfgs' is an optimizer in the family of quasi-Newton methods.
 |
 |      - 'sgd' refers to stochastic gradient descent.
 |
 |      - 'adam' refers to a stochastic gradient-based optimizer proposed
 |        by Kingma, Diederik, and Jimmy Ba
 |
 |      Note: The default solver 'adam' works pretty well on relatively
 |      large datasets (with thousands of training samples or more) in terms of
 |      both training time and validation score.
 |      For small datasets, however, 'lbfgs' can converge faster and perform
 |      better.
 |
 |  alpha : float, default=0.0001
 |      L2 penalty (regularization term) parameter.
 |
 |  batch_size : int, default='auto'
 |      Size of minibatches for stochastic optimizers.
 |      If the solver is 'lbfgs', the classifier will not use minibatch.
 |      When set to "auto", `batch_size=min(200, n_samples)`.
 |
 |  learning_rate : {'constant', 'invscaling', 'adaptive'}, default='constant'
 |      Learning rate schedule for weight updates.
 |
 |      - 'constant' is a constant learning rate given by
 |        'learning_rate_init'.
 |
 |      - 'invscaling' gradually decreases the learning rate at each
 |        time step 't' using an inverse scaling exponent of 'power_t'.
 |        effective_learning_rate = learning_rate_init / pow(t, power_t)
 |
 |      - 'adaptive' keeps the learning rate constant to
 |        'learning_rate_init' as long as training loss keeps decreasing.
 |        Each time two consecutive epochs fail to decrease training loss by at
 |        least tol, or fail to increase validation score by at least tol if
 |        'early_stopping' is on, the current learning rate is divided by 5.
 |
 |      Only used when ``solver='sgd'``.
 |
 |  learning_rate_init : float, default=0.001
 |      The initial learning rate used. It controls the step-size
 |      in updating the weights. Only used when solver='sgd' or 'adam'.
 |
 |  power_t : float, default=0.5
 |      The exponent for inverse scaling learning rate.
 |      It is used in updating effective learning rate when the learning_rate
 |      is set to 'invscaling'. Only used when solver='sgd'.
 |
 |  max_iter : int, default=200
 |      Maximum number of iterations. The solver iterates until convergence
 |      (determined by 'tol') or this number of iterations. For stochastic
 |      solvers ('sgd', 'adam'), note that this determines the number of epochs
 |      (how many times each data point will be used), not the number of
 |      gradient steps.
 |
 |  shuffle : bool, default=True
 |      Whether to shuffle samples in each iteration. Only used when
```

```
 |      solver='sgd' or 'adam'.
 |  
 |  random_state : int, RandomState instance, default=None
 |      Determines random number generation for weights and bias
 |      initialization, train-test split if early stopping is used, and batch
 |      sampling when solver='sgd' or 'adam'.
 |      Pass an int for reproducible results across multiple function calls.
 |      See :term:`Glossary <random_state>`.
 |  
 |  tol : float, default=1e-4
 |      Tolerance for the optimization. When the loss or score is not improving
 |      by at least ``tol`` for ``n_iter_no_change`` consecutive iterations,
 |      unless ``learning_rate`` is set to 'adaptive', convergence is
 |      considered to be reached and training stops.
 |  
 |  verbose : bool, default=False
 |      Whether to print progress messages to stdout.
 |  
 |  warm_start : bool, default=False
 |      When set to True, reuse the solution of the previous
 |      call to fit as initialization, otherwise, just erase the
 |      previous solution. See :term:`the Glossary <warm_start>`.
 |  
 |  momentum : float, default=0.9
 |      Momentum for gradient descent update. Should be between 0 and 1. Only
 |      used when solver='sgd'.
 |  
 |  nesterovs_momentum : bool, default=True
 |      Whether to use Nesterov's momentum. Only used when solver='sgd' and
 |      momentum > 0.
 |  
 |  early_stopping : bool, default=False
 |      Whether to use early stopping to terminate training when validation
 |      score is not improving. If set to true, it will automatically set
 |      aside 10% of training data as validation and terminate training when
 |      validation score is not improving by at least tol for
 |      ``n_iter_no_change`` consecutive epochs. The split is stratified,
 |      except in a multilabel setting.
 |      If early stopping is False, then the training stops when the training
 |      loss does not improve by more than tol for n_iter_no_change consecutive
 |      passes over the training set.
 |      Only effective when solver='sgd' or 'adam'.
 |  
 |  validation_fraction : float, default=0.1
 |      The proportion of training data to set aside as validation set for
 |      early stopping. Must be between 0 and 1.
 |      Only used if early_stopping is True.
 |
 |  beta_1 : float, default=0.9
 |      Exponential decay rate for estimates of first moment vector in adam,
 |      should be in [0, 1). Only used when solver='adam'.
 |
 |  beta_2 : float, default=0.999
 |      Exponential decay rate for estimates of second moment vector in adam,
 |      should be in [0, 1). Only used when solver='adam'.
 |
 |  epsilon : float, default=1e-8
 |      Value for numerical stability in adam. Only used when solver='adam'.
 |
 |  n_iter_no_change : int, default=10
 |      Maximum number of epochs to not meet ``tol`` improvement.
 |      Only effective when solver='sgd' or 'adam'.
 |
 |      .. versionadded:: 0.20
 |
 |  max_fun : int, default=15000
 |      Only used when solver='lbfgs'. Maximum number of loss function calls.
 |      The solver iterates until convergence (determined by 'tol'), number
 |      of iterations reaches max_iter, or this number of loss function calls.
 |      Note that number of loss function calls will be greater than or equal
 |      to the number of iterations for the `MLPClassifier`.
 |
 |      .. versionadded:: 0.22
 |
 |  Attributes
 |  ----------
 |  classes_ : ndarray or list of ndarray of shape (n_classes,)
 |      Class labels for each output.
 |
 |  loss_ : float
 |      The current loss computed with the loss function.
 |
 |  best_loss_ : float
 |      The minimum loss reached by the solver throughout fitting.
 |
 |  loss_curve_ : list of shape (`n_iter_`,)
```

```
 |      The ith element in the list represents the loss at the ith iteration.
 |
 |  t_ : int
 |      The number of training samples seen by the solver during fitting.
 |
 |  coefs_ : list of shape (n_layers - 1,)
 |      The ith element in the list represents the weight matrix corresponding
 |      to layer i.
 |
 |  intercepts_ : list of shape (n_layers - 1,)
 |      The ith element in the list represents the bias vector corresponding to
 |      layer i + 1.
 |
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  n_iter_ : int
 |      The number of iterations the solver has run.
 |
 |  n_layers_ : int
 |      Number of layers.
 |
 |  n_outputs_ : int
 |      Number of outputs.
 |
 |  out_activation_ : str
 |      Name of the output activation function.
 |
 |  See Also
 |  --------
 |  MLPRegressor : Multi-layer Perceptron regressor.
 |  BernoulliRBM : Bernoulli Restricted Boltzmann Machine (RBM).
 |
 |  Notes
 |  -----
```

```
|  MLPClassifier trains iteratively since at each time step
|  the partial derivatives of the loss function with respect to the model
|  parameters are computed to update the parameters.
|
|  It can also have a regularization term added to the loss function
|  that shrinks model parameters to prevent overfitting.
|
|  This implementation works with data represented as dense numpy arrays or
|  sparse scipy arrays of floating point values.
|
|  References
|  ----------
|  Hinton, Geoffrey E.
|      "Connectionist learning procedures." Artificial intelligence 40.1
|      (1989): 185-234.
|
|  Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of
|      training deep feedforward neural networks." International Conference
|      on Artificial Intelligence and Statistics. 2010.
|
|  He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level
|      performance on imagenet classification." arXiv preprint
|      arXiv:1502.01852 (2015).
|
|  Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic
|      optimization." arXiv preprint arXiv:1412.6980 (2014).
|
|  Examples
|  --------
|  >>> from sklearn.neural_network import MLPClassifier
|  >>> from sklearn.datasets import make_classification
|  >>> from sklearn.model_selection import train_test_split
|  >>> X, y = make_classification(n_samples=100, random_state=1)
|  >>> X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
|  ...                                                     random_state=1)
|  >>> clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
|  >>> clf.predict_proba(X_test[:1])
|  array([[0.038..., 0.961...]])
|  >>> clf.predict(X_test[:5, :])
|  array([1, 0, 1, 0, 1])
|  >>> clf.score(X_test, y_test)
|  0.8...
|
 |  Method resolution order:
 |      MLPClassifier
 |      sklearn.base.ClassifierMixin
 |      BaseMultilayerPerceptron
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, hidden_layer_sizes=(100,), activation='relu', *, solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001,
power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10,
max_fun=15000)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  partial_fit(self, X, y, classes=None)
 |      Update the model with a single iteration over the given data.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input data.
 |
 |      y : array-like of shape (n_samples,)
 |          The target values.
 |
 |      classes : array of shape (n_classes,), default=None
 |          Classes across all calls to partial_fit.
 |          Can be obtained via `np.unique(y_all)`, where y_all is the
 |          target vector of the entire dataset.
 |          This argument is required for the first call to partial_fit
 |          and can be omitted in the subsequent calls.
 |          Note that y doesn't need to contain all labels in `classes`.
 |
 |      Returns
 |      -------
 |      self : object
 |          Trained MLP model.
 |
 |  predict(self, X)
 |      Predict using the multi-layer perceptron classifier.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input data.
 |
 |      Returns
 |      -------
 |      y : ndarray, shape (n_samples,) or (n_samples, n_classes)
 |          The predicted classes.
 |
 |  predict_log_proba(self, X)
 |      Return the log of probability estimates.
 |
 |      Parameters
 |      ----------
 |      X : ndarray of shape (n_samples, n_features)
 |          The input data.
 |
 |      Returns
 |      -------
 |      log_y_prob : ndarray of shape (n_samples, n_classes)
 |          The predicted log-probability of the sample for each class
 |          in the model, where classes are ordered as they are in
 |          `self.classes_`. Equivalent to `log(predict_proba(X))`.
 |
 |  predict_proba(self, X)
 |      Probability estimates.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features)
 |          The input data.
 |
 |      Returns
 |      -------
 |      y_prob : ndarray of shape (n_samples, n_classes)
 |          The predicted probability of the sample for each class in the
 |          model, where classes are ordered as they are in `self.classes_`.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True labels for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of ``self.predict(X)`` wrt. `y`.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.base.ClassifierMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseMultilayerPerceptron:
 |
 |  fit(self, X, y)
 |      Fit the model to data matrix X and target(s) y.
 |
 |      Parameters
 |      ----------
 |      X : ndarray or sparse matrix of shape (n_samples, n_features)
 |          The input data.
 |
 |      y : ndarray of shape (n_samples,) or (n_samples, n_outputs)
 |          The target values (class labels in classification, real numbers in
 |          regression).
 |
 |      Returns
 |      -------
 |      self : object
 |          Returns a trained MLP model.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
 |
 |  __getstate__(self)
 |
 |  __repr__(self, N_CHAR_MAX=700)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  get_params(self, deep=True)
 |      Get parameters for this estimator.
 |
 |      Parameters
 |      ----------
 |      deep : bool, default=True
 |          If True, will return the parameters for this estimator and
 |          contained subobjects that are estimators.
 |
 |      Returns
 |      -------
 |      params : dict
 |          Parameter names mapped to their values.
 |
 |  set_params(self, **params)
 |      Set the parameters of this estimator.
 |
 |      The method works on simple estimators as well as on nested objects
 |      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
```

```
 |      parameters of the form ``<component>__<parameter>`` so that it's
 |      possible to update each component of a nested object.
 |
 |      Parameters
 |      ----------
 |      **params : dict
 |          Estimator parameters.
 |
 |      Returns
 |      -------
 |      self : estimator instance
 |          Estimator instance.
```

```
mlp.fit(x_pca,train_target1)
mlp.score(x_pca,train_target1),mlp.score(val_pca,val_target)
```

```
(1.0, 0.9766666666666667)
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

Finally, I Achieved 97.6% accuracy on Validation Dataset and Selected this model for Final Test set.

```
test_df
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pi
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
27995	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
27996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
27997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
27998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
27999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

28000 rows × 784 columns

```
final_result
```

	ImageId	Label
0	1	0
1	2	0
2	3	0
3	4	0
4	5	0
...	...	...
27995	27996	0
27996	27997	0
27997	27998	0
27998	27999	0
27999	28000	0

28000 rows × 2 columns

```
final_result['Label']=mlp.predict(test_pca)
```

```
final_result
```

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3
...	...	...
27995	27996	9
27996	27997	7
27997	27998	3
27998	27999	9
27999	28000	2

28000 rows × 2 columns

```
final_result.to_csv('final_result.csv')
```

```
jovian.commit()
```

```
[jovian] Detected Colab notebook...
```

```
[jovian] Uploading colab notebook to Jovian...
Committed successfully! https://jovian.ai/btech60309-19/digit-recognization
```

```
'https://jovian.ai/btech60309-19/digit-recognization'
```

#Summary and References On sbmitting on Kaggle I got accuracy of 97.14%.

You can Download the Dataset from following link:

https://www.kaggle.com/competitions/digit-recognizer

```
jovian.com
```