Data Tidying and Cleaning

What's the most efficient way to perform data transformations in pandas: Step-by-Step guide

Step 1: Import the necessary libraries

The first step is to import the necessary libraries. In this case, you will need to import pandas. You can do this by typing the following command in your Python environment:

```
import pandas as pd
```

Step 2: Load your data

The next step is to load your data into a pandas DataFrame. You can do this using the pandas read_csv() function if your data is in a CSV file. For example:

```
df = pd.read_csv('your_file.csv')
```

Step 3: Inspect your data

Before you start transforming your data, it's a good idea to inspect it first to understand its structure and content. You can do this using the head() function, which returns the first n rows of your DataFrame. For example:

```
df.head()
```

Step 4: Perform data transformations

There are many ways to perform data transformations in pandas, but the most efficient way is usually to use vectorized operations. These are operations that are performed on entire arrays of data at once, rather than on individual elements. This can significantly speed up your data processing.

Here are a few examples of common data transformations and how to perform them in a vectorized way:

Adding a new column based on existing columns:

```
df['new_column'] = df['column1'] + df['column2']
```

Applying a function to a column:

```
df['column'] = df['column'].apply(lambda x: x**2)
```

Replacing values in a column:

```
df['column'] = df['column'].replace({'old_value': 'new_value'})
```

Step 5: Check your transformations

After performing your transformations, it's a good idea to check that they have been applied correctly. You can do this by inspecting your DataFrame again using the head() function.

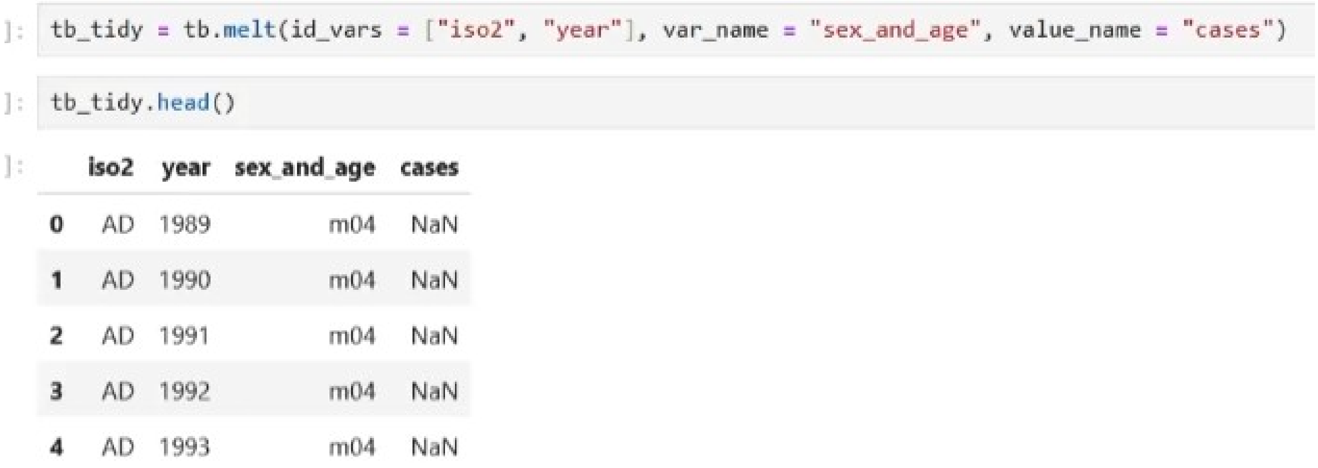
Step 6: Save your transformed data

Finally, once you're happy with your transformations, you can save your transformed data back to a CSV file using the to_csv() function. For example:

```
df.to_csv('your_transformed_file.csv', index=False)
```

Remember, the key to efficient data transformations in pandas is to use vectorized operations wherever possible. This will ensure that your transformations are performed as quickly and efficiently as possible.

Melt



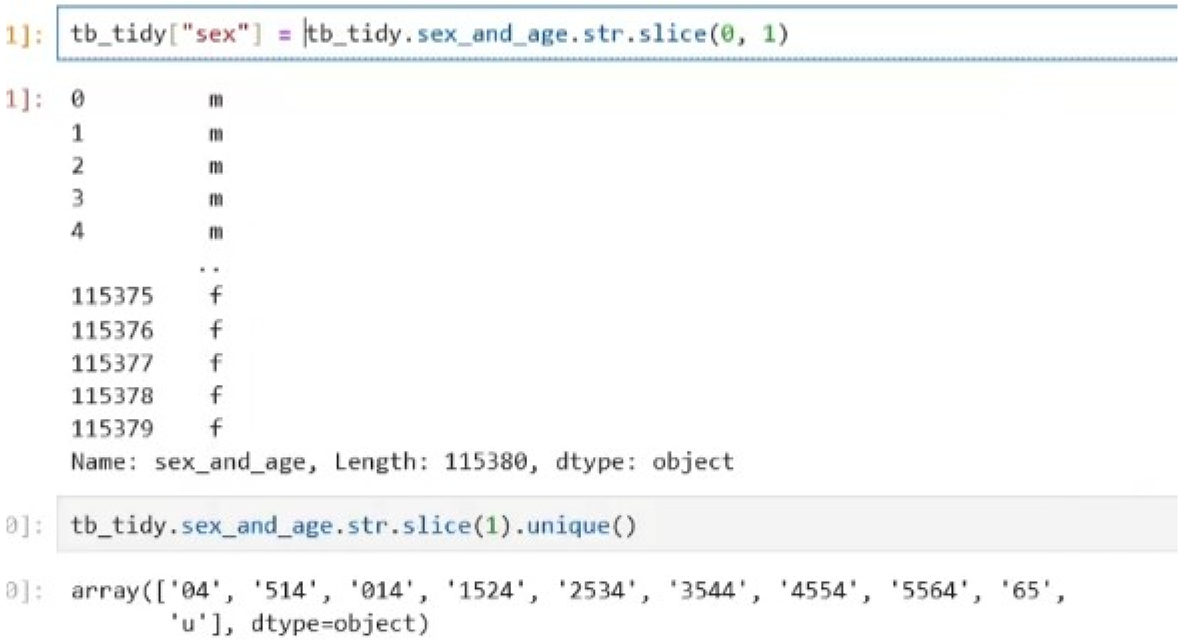
```
]:  tb_tidy = tb.melt(id_vars = ["iso2", "year"], var_name = "sex_and_age", value_name = "cases")
```

```
]:  tb_tidy.head()
```

]:

	iso2	year	sex_and_age	cases
0	AD	1989	m04	NaN
1	AD	1990	m04	NaN
2	AD	1991	m04	NaN
3	AD	1992	m04	NaN
4	AD	1993	m04	NaN

Slice



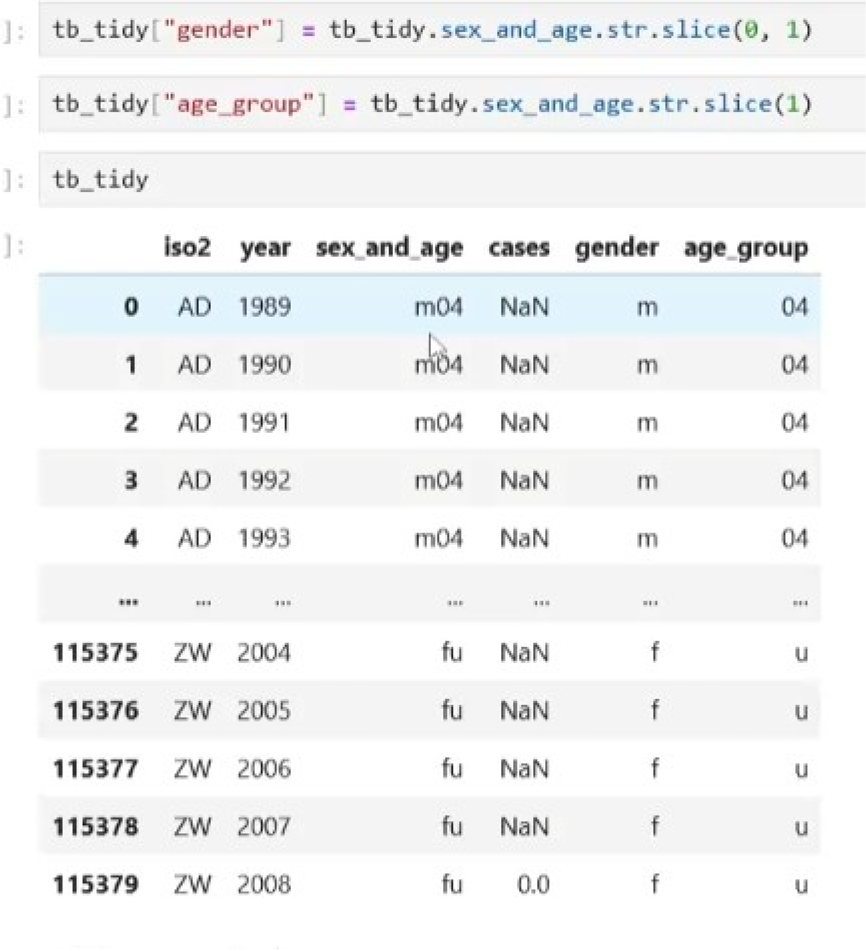
```
1]:  tb_tidy["sex"] = tb_tidy.sex_and_age.str.slice(0, 1)
```

```
1]:  0         m
     1         m
     2         m
     3         m
     4         m
              ..
     115375    f
     115376    f
     115377    f
     115378    f
     115379    f
     Name: sex_and_age, Length: 115380, dtype: object
```

```
0]:  tb_tidy.sex_and_age.str.slice(1).unique()
```

```
0]:  array(['04', '514', '014', '1524', '2534', '3544', '4554', '5564', '65',
            'u'], dtype=object)
```

Slices



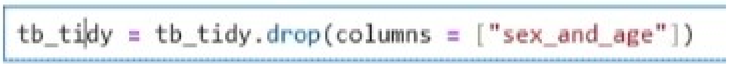
```
tb_tidy["gender"] = tb_tidy.sex_and_age.str.slice(0, 1)
```

```
tb_tidy["age_group"] = tb_tidy.sex_and_age.str.slice(1)
```

```
tb_tidy
```

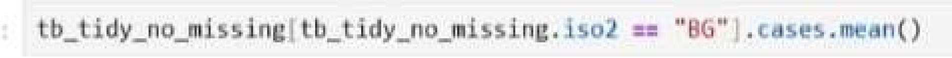
	iso2	year	sex_and_age	cases	gender	age_group
0	AD	1989	m04	NaN	m	04
1	AD	1990	m04	NaN	m	04
2	AD	1991	m04	NaN	m	04
3	AD	1992	m04	NaN	m	04
4	AD	1993	m04	NaN	m	04
...	...	...	...	...	...	...
115375	ZW	2004	fu	NaN	f	u
115376	ZW	2005	fu	NaN	f	u
115377	ZW	2006	fu	NaN	f	u
115378	ZW	2007	fu	NaN	f	u
115379	ZW	2008	fu	0.0	f	u

Drop column



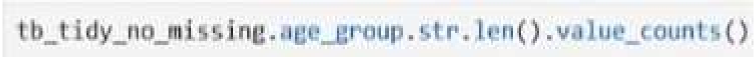
```
tb_tidy = tb_tidy.drop(columns = ["sex_and_age"])
```

Mean or whatever by condition



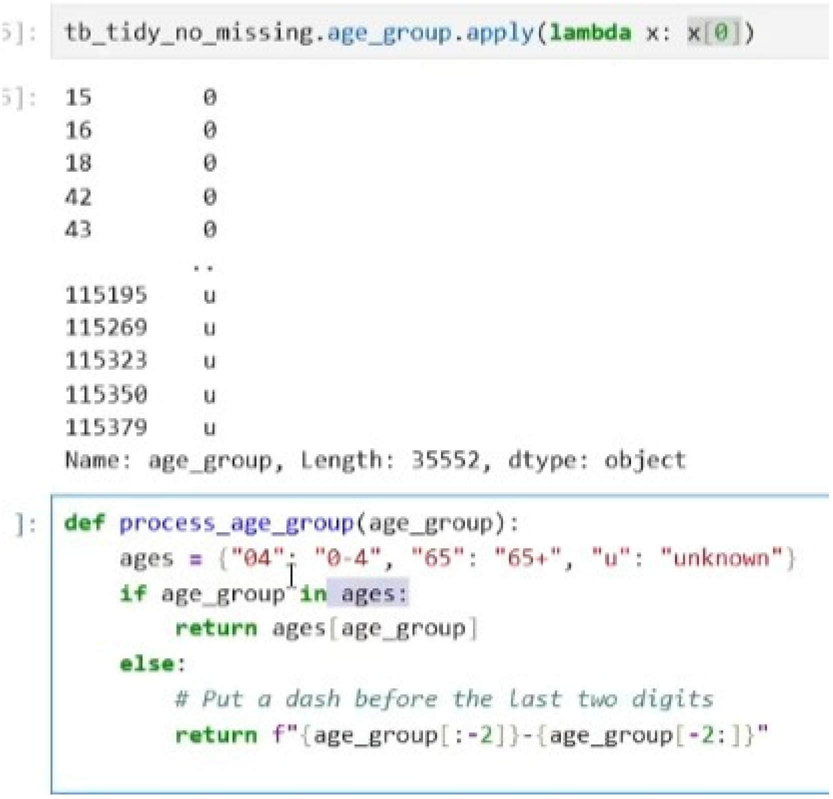
```
tb_tidy_no_missing[tb_tidy_no_missing.iso2 == "BG"].cases.mean()
```

Values count



```
tb_tidy_no_missing.age_group.str.len().value_counts()
```

Apply and function

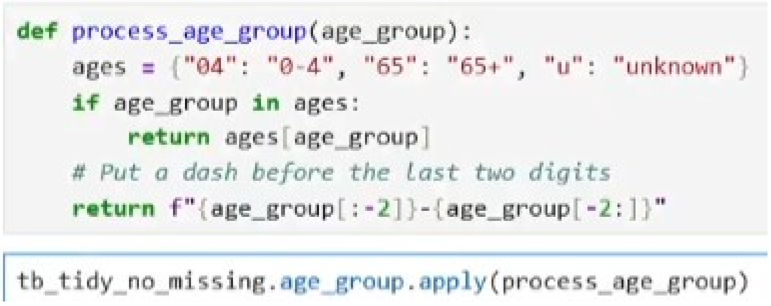


```
tb_tidy_no_missing.age_group.apply(lambda x: x[0])
```

```
15        0
16        0
18        0
42        0
43        0
         ..
115195    u
115269    u
115323    u
115350    u
115379    u
Name: age_group, Length: 35552, dtype: object
```

```
def process_age_group(age_group):
    ages = {"04": "0-4", "65": "65+", "u": "unknown"}
    if age_group in ages:
        return ages[age_group]
    else:
        # Put a dash before the last two digits
        return f"{age_group[:-2]}-{age_group[-2:]}"
```

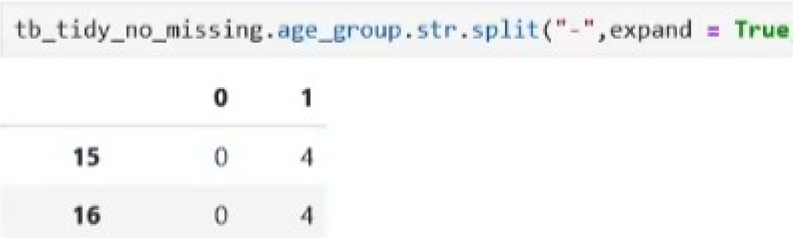
Apply function



```
def process_age_group(age_group):
    ages = {"04": "0-4", "65": "65+", "u": "unknown"}
    if age_group in ages:
        return ages[age_group]
    # Put a dash before the last two digits
    return f"{age_group[:-2]}-{age_group[-2:]}"
```

```
tb_tidy_no_missing.age_group.apply(process_age_group)
```

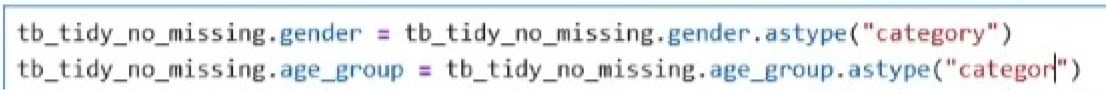
Split and expand



```
tb_tidy_no_missing.age_group.str.split("-",expand = True
```

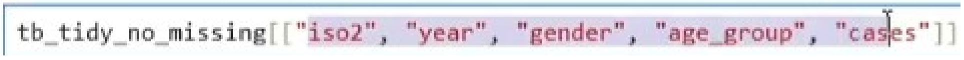
	0	1
15	0	4
16	0	4

To category



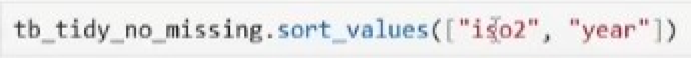
```
tb_tidy_no_missing.gender = tb_tidy_no_missing.gender.astype("category")
tb_tidy_no_missing.age_group = tb_tidy_no_missing.age_group.astype("categor")
```

Rearrange columns

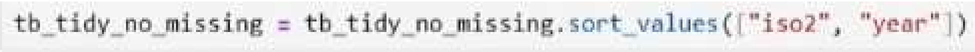


```
tb_tidy_no_missing[["iso2", "year", "gender", "age_group", "cases"]]
```

Sort by iso2 and then by year

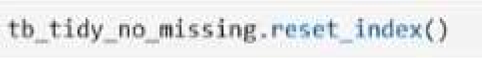


```
tb_tidy_no_missing.sort_values(["iso2", "year"])
```

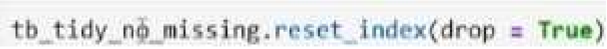


```
tb_tidy_no_missing = tb_tidy_no_missing.sort_values(["iso2", "year"])
```

Reset index - преподреждане

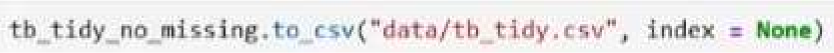


```
tb_tidy_no_missing.reset_index()
```



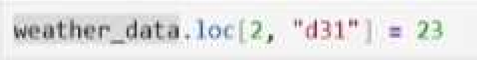
```
tb_tidy_no_missing.reset_index(drop = True)
```

Презаписване в ново csv



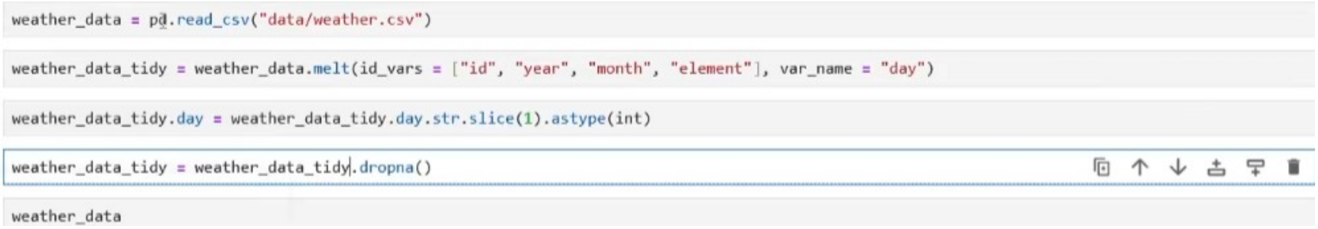
```
tb_tidy_no_missing.to_csv("data/tb_tidy.csv", index = None)
```

Дава нова стойност на 2-ри ред, 23 колона



```
weather_data.loc[2, "d31"] = 23
```

Melt, slice, dropna и оставям само тези, които ми трябват



```
weather_data = pd.read_csv("data/weather.csv")
```

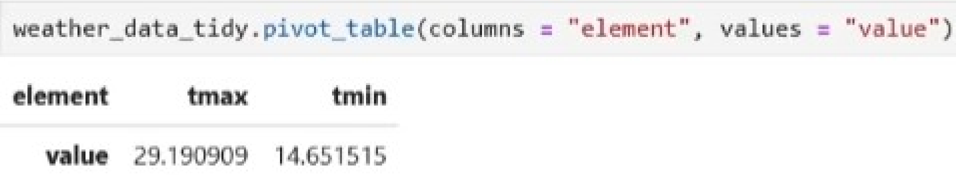
```
weather_data_tidy = weather_data.melt(id_vars = ["id", "year", "month", "element"], var_name = "day")
```

```
weather_data_tidy.day = weather_data_tidy.day.str.slice(1).astype(int)
```

```
weather_data_tidy = weather_data_tidy.dropna()
```

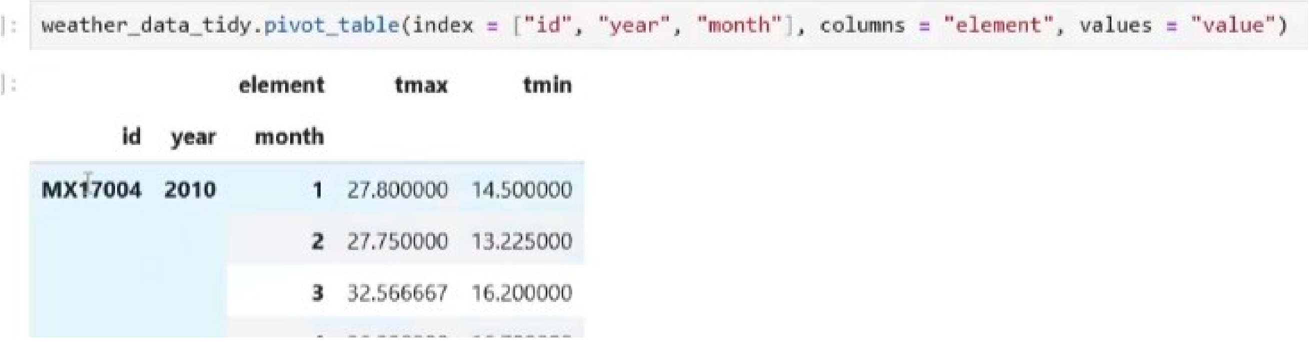
```
weather_data
```

Pivot



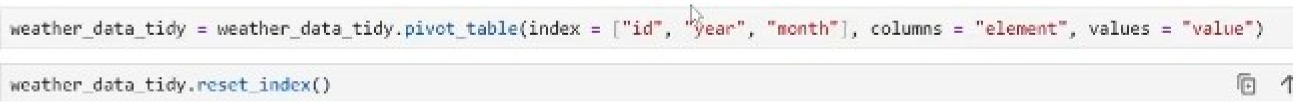
```
weather_data_tidy.pivot_table(columns = "element", values = "value")
```

```
element       tmax       tmin
  value  29.190909  14.651515
```



```
weather_data_tidy.pivot_table(index = ["id", "year", "month"], columns = "element", values = "value")
```

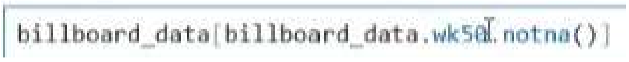
```
                    element       tmax       tmin
     id  year  month
MX17004  2010      1         27.800000  14.500000
                   2         27.750000  13.225000
                   3         32.566667  16.200000
```



```
weather_data_tidy = weather_data_tidy.pivot_table(index = ["id", "year", "month"], columns = "element", values = "value")
```

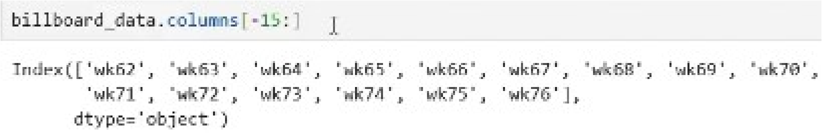
```
weather_data_tidy.reset_index()
```

Which are in week 50? - notna



```
billboard_data[billboard_data.wk50.notna()]
```

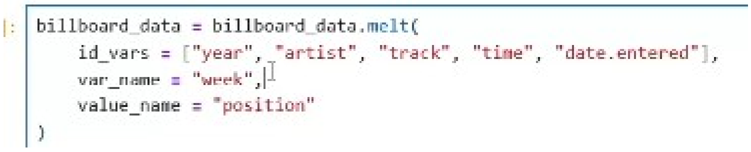
Last 15 columns



```
billboard_data.columns[-15:]
```

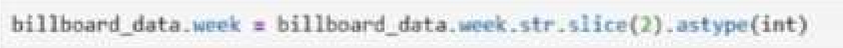
```
Index(['wk62', 'wk63', 'wk64', 'wk65', 'wk66', 'wk67', 'wk68', 'wk69', 'wk70',
       'wk71', 'wk72', 'wk73', 'wk74', 'wk75', 'wk76'],
      dtype='object')
```

Melt more examples



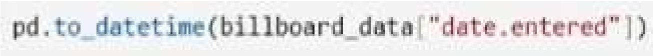
```
billboard_data = billboard_data.melt(
    id_vars = ["year", "artist", "track", "time", "date.entered"],
    var_name = "week",
    value_name = "position"
)
```

Slice and astype

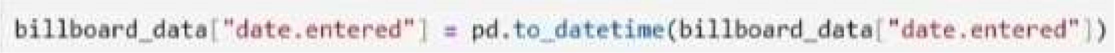


```
billboard_data.week = billboard_data.week.str.slice(2).astype(int)
```

Data datetime

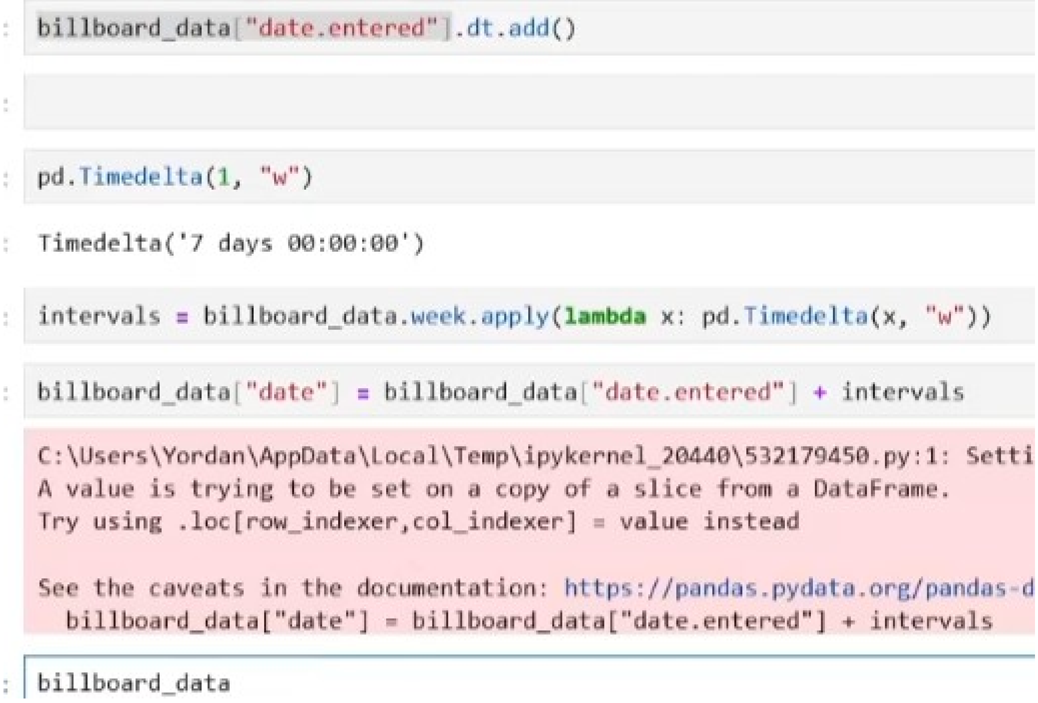


```
pd.to_datetime(billboard_data["date.entered"])
```



```
billboard_data["date.entered"] = pd.to_datetime(billboard_data["date.entered"])
```

Add week of the year



```
billboard_data["date.entered"].dt.add()
```

```
pd.Timedelta(1, "w")
```

```
Timedelta('7 days 00:00:00')
```

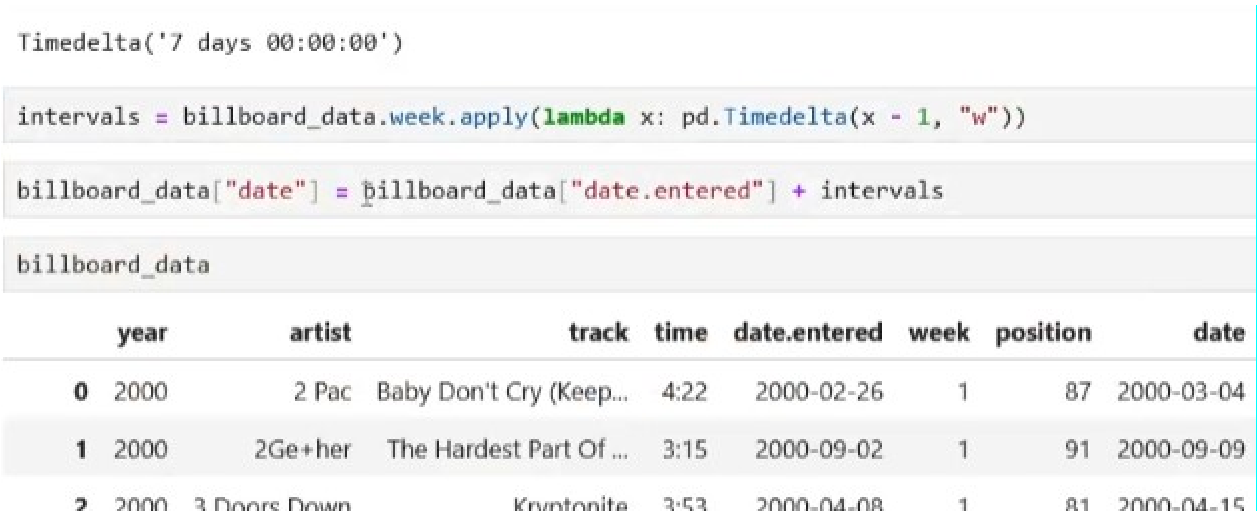
```
intervals = billboard_data.week.apply(lambda x: pd.Timedelta(x, "w"))
```

```
billboard_data["date"] = billboard_data["date.entered"] + intervals
```

```
C:\Users\Yordan\AppData\Local\Temp\ipykernel_20440\532179450.py:1: Setti
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-d
  billboard_data["date"] = billboard_data["date.entered"] + intervals
```

```
billboard_data
```



```
Timedelta('7 days 00:00:00')
```

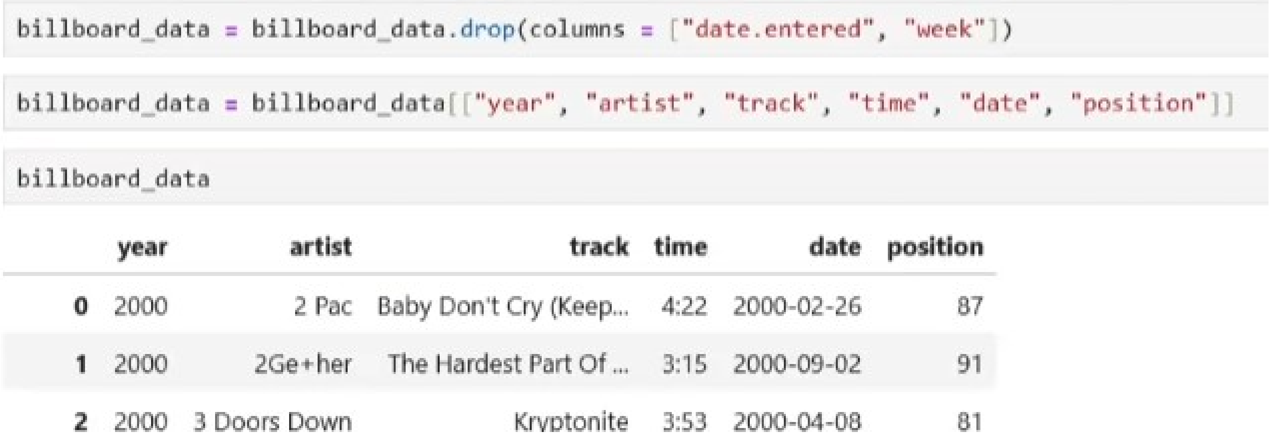
```
intervals = billboard_data.week.apply(lambda x: pd.Timedelta(x - 1, "w"))
```

```
billboard_data["date"] = billboard_data["date.entered"] + intervals
```

```
billboard_data
```

	year	artist	track	time	date.entered	week	position	date
0	2000	2 Pac	Baby Don't Cry (Keep...	4:22	2000-02-26	1	87	2000-03-04
1	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	1	91	2000-09-09
2	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	1	81	2000-04-15

Drop and rearange columns



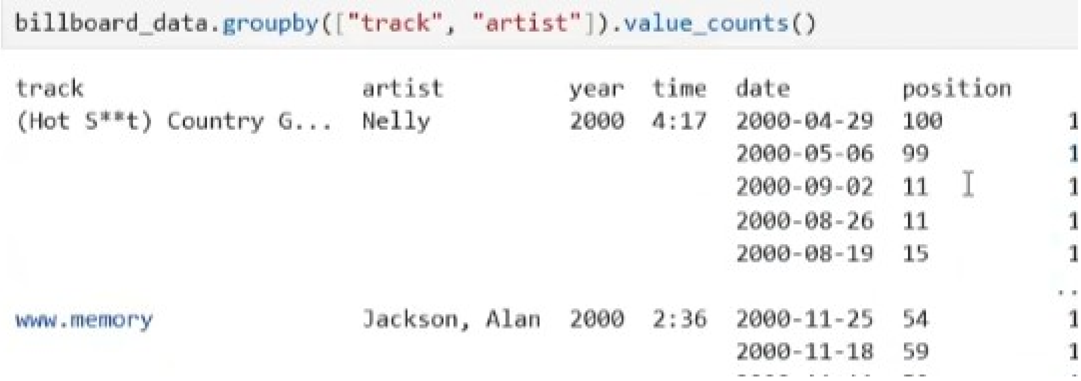
```
billboard_data = billboard_data.drop(columns = ["date.entered", "week"])
```

```
billboard_data = billboard_data[["year", "artist", "track", "time", "date", "position"]]
```

```
billboard_data
```

	year	artist	track	time	date	position
0	2000	2 Pac	Baby Don't Cry (Keep...	4:22	2000-02-26	87
1	2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91
2	2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81

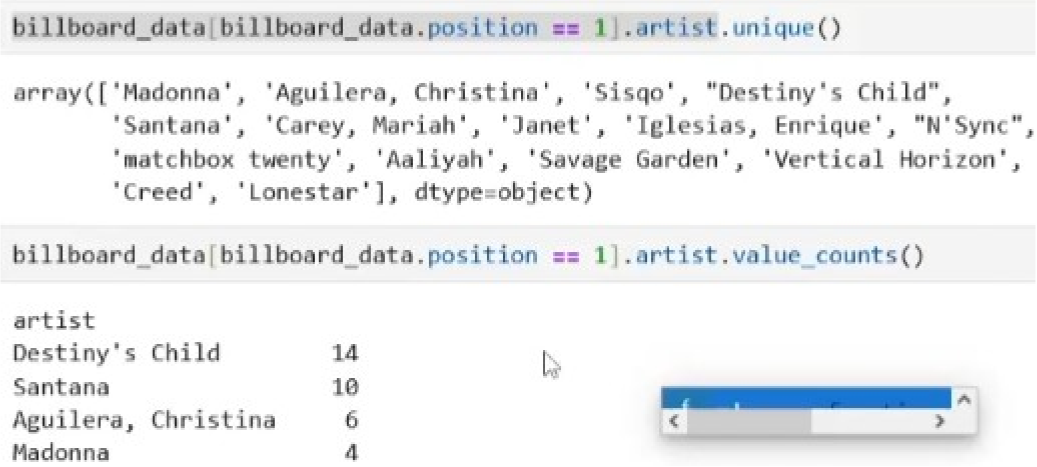
Group by



```
billboard_data.groupby(["track", "artist"]).value_counts()
```

```
track                      artist          year  time  date        position
(Hot S**t) Country G...    Nelly           2000  4:17  2000-04-29  100          1
                                                       2000-05-06  99           1
                                                       2000-09-02  11           1
                                                       2000-08-26  11           1
                                                       2000-08-19  15           1
                                                                                ..
www.memory                 Jackson, Alan   2000  2:36  2000-11-25  54           1
                                                       2000-11-18  59           1
```

Condition if



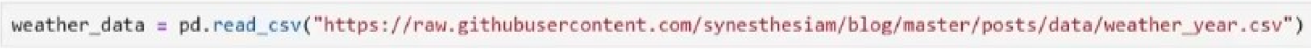
```
billboard_data[billboard_data.position == 1].artist.unique()
```

```
array(['Madonna', 'Aguilera, Christina', 'Sisqo', "Destiny's Child",
       'Santana', 'Carey, Mariah', 'Janet', 'Iglesias, Enrique', "N'Sync",
       'matchbox twenty', 'Aaliyah', 'Savage Garden', 'Vertical Horizon',
       'Creed', 'Lonestar'], dtype=object)
```

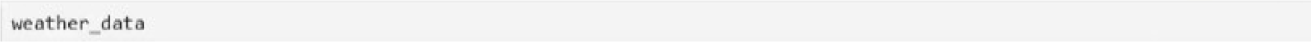
```
billboard_data[billboard_data.position == 1].artist.value_counts()
```

```
artist
Destiny's Child        14
Santana                10
Aguilera, Christina     6
Madonna                 4
```

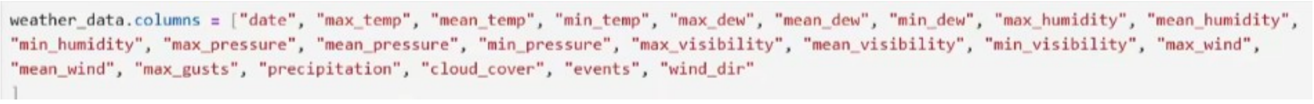
Read csv from link - processing, basic steps



```
weather_data = pd.read_csv("https://raw.githubusercontent.com/synesthesiam/blog/master/posts/data/weather_year.csv")
```

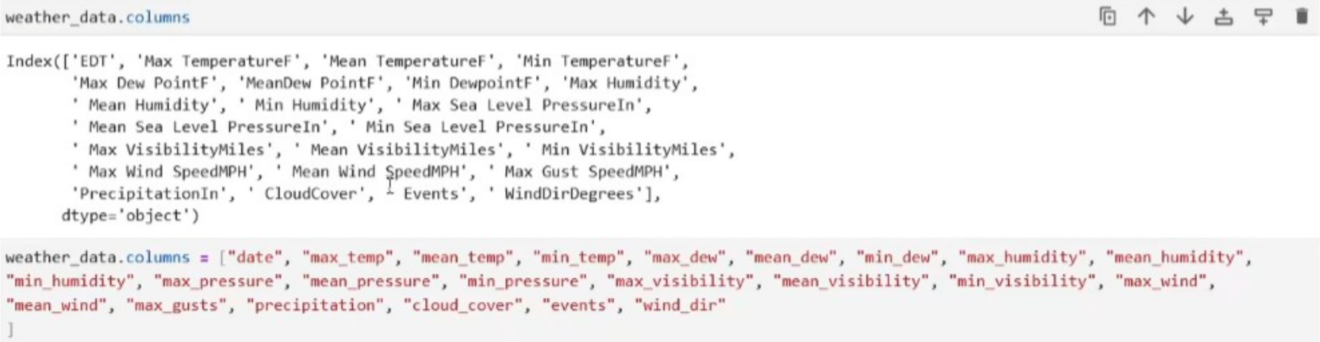


```
weather_data
```



```
weather_data.columns = ["date", "max_temp", "mean_temp", "min_temp", "max_dew", "mean_dew", "min_dew", "max_humidity", "mean_humidity",
"min_humidity", "max_pressure", "mean_pressure", "min_pressure", "max_visibility", "mean_visibility", "min_visibility", "max_wind",
"mean_wind", "max_gusts", "precipitation", "cloud_cover", "events", "wind_dir"
]
```

Convert column names

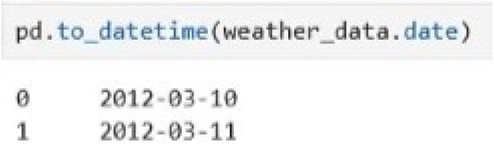


```
weather_data.columns
```

```
Index(['EDT', 'Max TemperatureF', 'Mean TemperatureF', 'Min TemperatureF',
       'Max Dew PointF', 'MeanDew PointF', 'Min DewpointF', 'Max Humidity',
       ' Mean Humidity', ' Min Humidity', ' Max Sea Level PressureIn',
       ' Mean Sea Level PressureIn', ' Min Sea Level PressureIn',
       ' Max VisibilityMiles', ' Mean VisibilityMiles', ' Min VisibilityMiles',
       ' Max Wind SpeedMPH', ' Mean Wind SpeedMPH', ' Max Gust SpeedMPH',
       'PrecipitationIn', ' CloudCover', ' Events', ' WindDirDegrees'],
      dtype='object')
```

```
weather_data.columns = ["date", "max_temp", "mean_temp", "min_temp", "max_dew", "mean_dew", "min_dew", "max_humidity", "mean_humidity",
"min_humidity", "max_pressure", "mean_pressure", "min_pressure", "max_visibility", "mean_visibility", "min_visibility", "max_wind",
"mean_wind", "max_gusts", "precipitation", "cloud_cover", "events", "wind_dir"
]
```

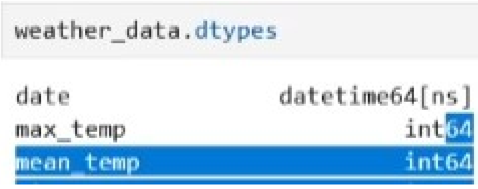
To datetime



```
pd.to_datetime(weather_data.date)
```

```
0      2012-03-10
1      2012-03-11
```

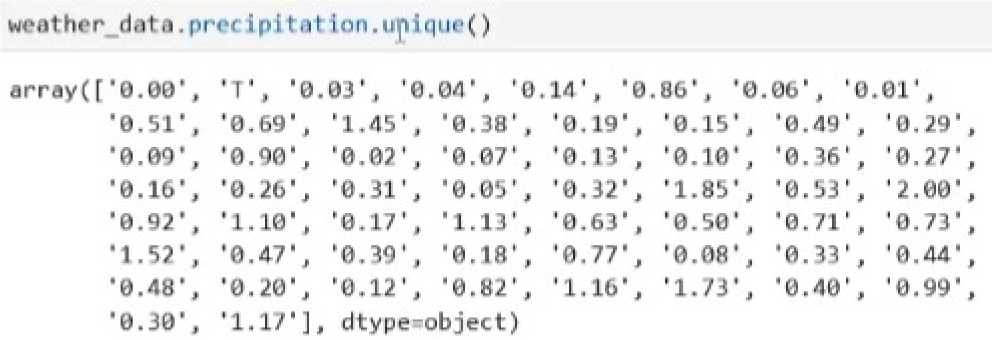
Dtypes



```
weather_data.dtypes
```

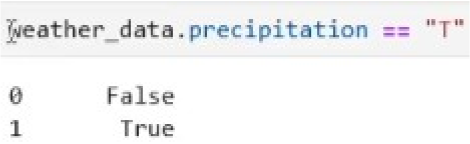
```
date              datetime64[ns]
max_temp                   int64
mean_temp                  int64
```

Object column analysis



```
weather_data.precipitation.unique()
```

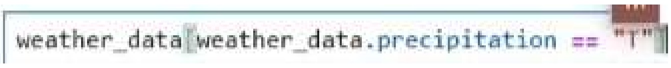
```
array(['0.00', 'T', '0.03', '0.04', '0.14', '0.86', '0.06', '0.01',
       '0.51', '0.69', '1.45', '0.38', '0.19', '0.15', '0.49', '0.29',
       '0.09', '0.90', '0.02', '0.07', '0.13', '0.10', '0.36', '0.27',
       '0.16', '0.26', '0.31', '0.05', '0.32', '1.85', '0.53', '2.00',
       '0.92', '1.10', '0.17', '1.13', '0.63', '0.50', '0.71', '0.73',
       '1.52', '0.47', '0.39', '0.18', '0.77', '0.08', '0.33', '0.44',
       '0.48', '0.20', '0.12', '0.82', '1.16', '1.73', '0.40', '0.99',
       '0.30', '1.17'], dtype=object)
```



```
weather_data.precipitation == "T"
```

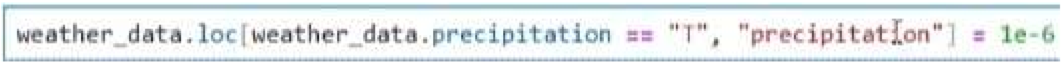
```
0      False
1       True
```

How many are with ‘T’



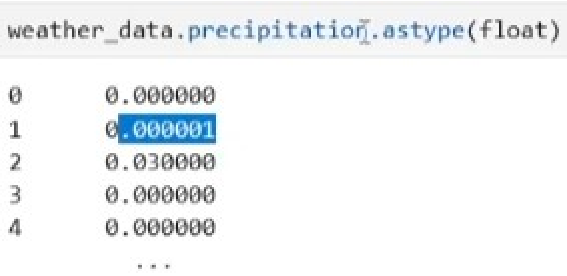
```
weather_data[weather_data.precipitation == "T"]
```

Replace column value ‘T’ with a very small number



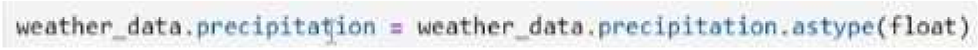
```
weather_data.loc[weather_data.precipitation == "T", "precipitation"] = 1e-6
```

Astype float



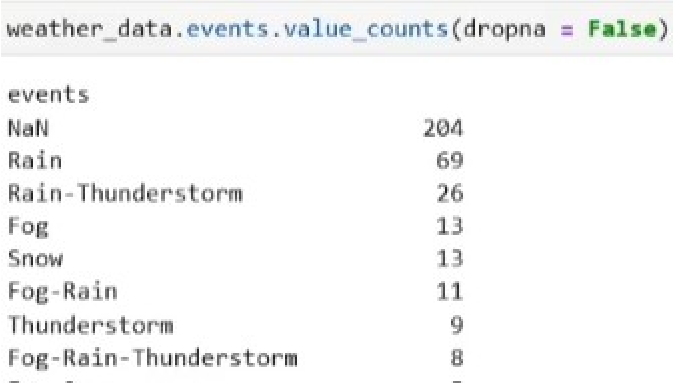
```
weather_data.precipitation.astype(float)
```

```
0       0.000000
1       0.000001
2       0.030000
3       0.000000
4       0.000000
          ...
```



```
weather_data.precipitation = weather_data.precipitation.astype(float)
```

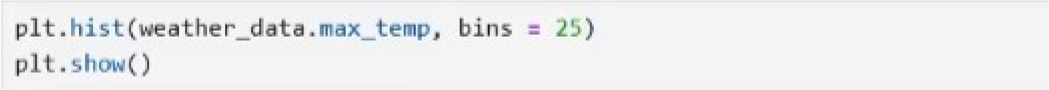
Count events including NAN



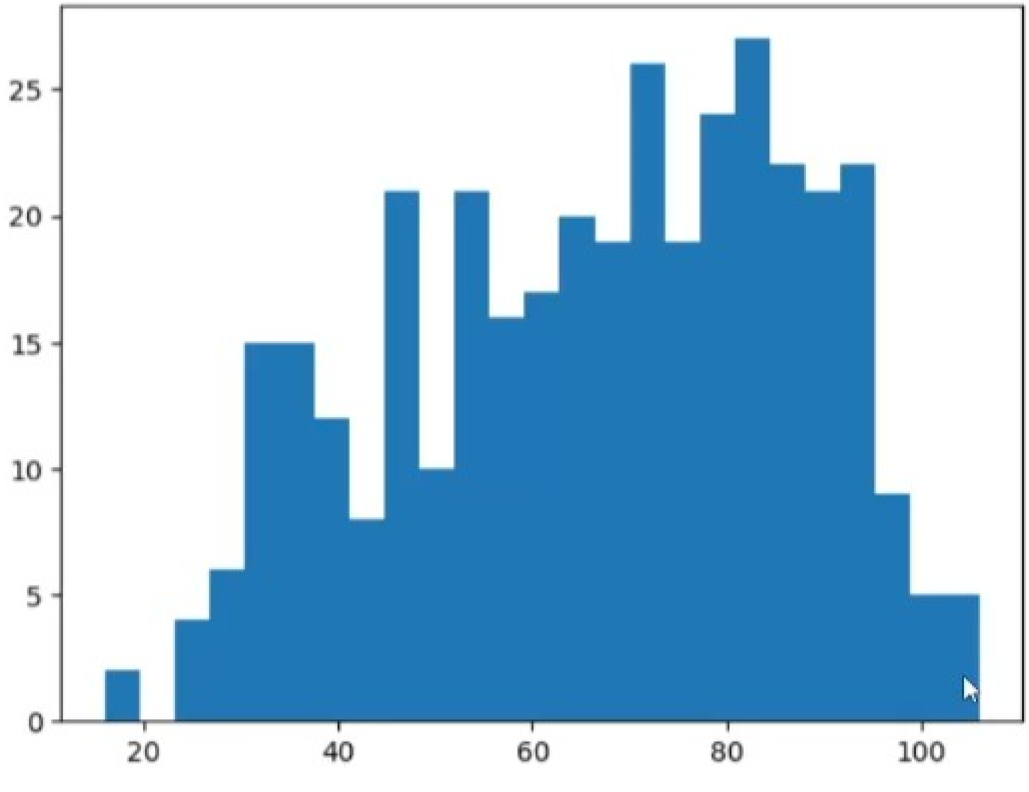
```
weather_data.events.value_counts(dropna = False)
```

```
events
NaN                          204
Rain                          69
Rain-Thunderstorm             26
Fog                           13
Snow                          13
Fog-Rain                      11
Thunderstorm                   9
Fog-Rain-Thunderstorm          8
```

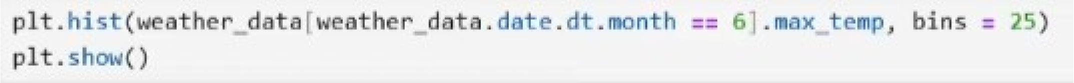
Hist



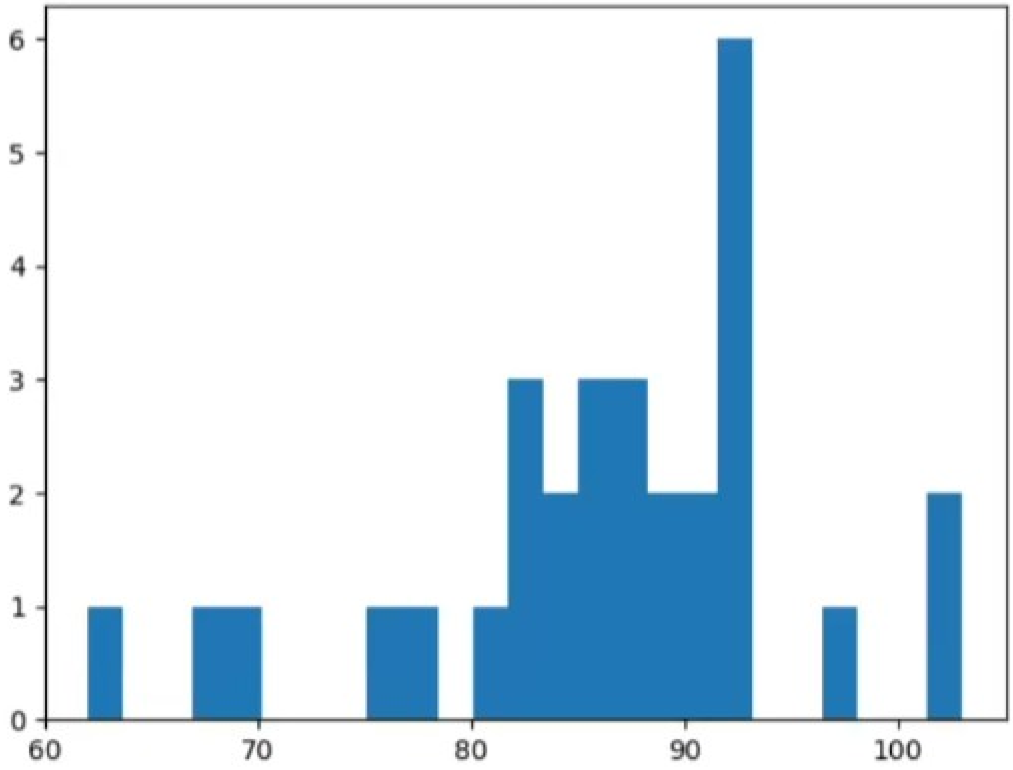
```
plt.hist(weather_data.max_temp, bins = 25)
plt.show()
```



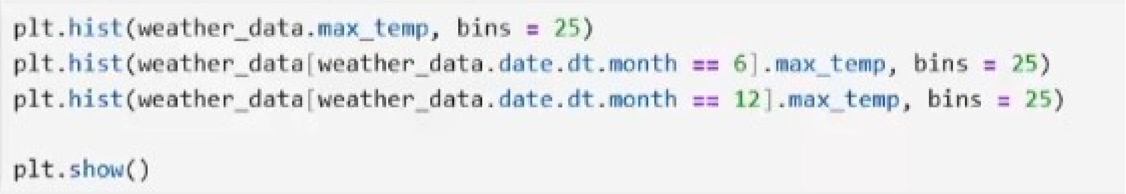
Hist max temperatures for june - 6



```
plt.hist(weather_data[weather_data.date.dt.month == 6].max_temp, bins = 25)
plt.show()
```

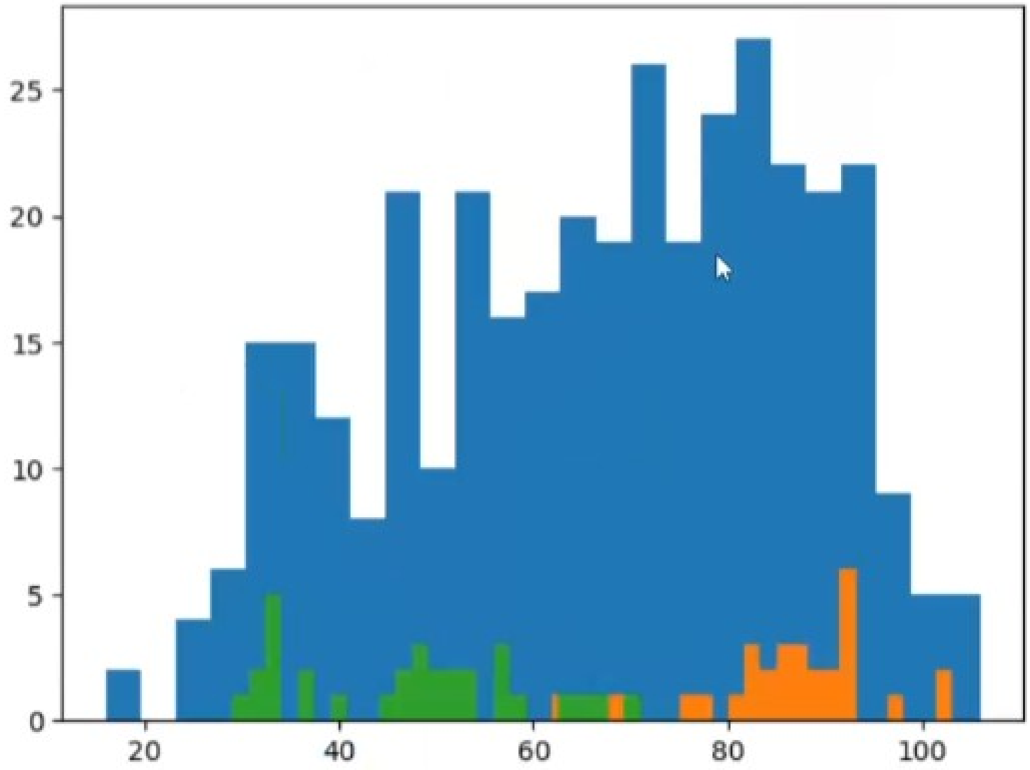


Plot all together

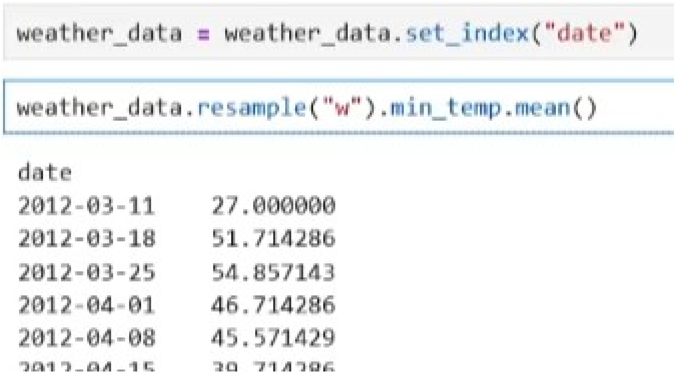


```
plt.hist(weather_data.max_temp, bins = 25)
plt.hist(weather_data[weather_data.date.dt.month == 6].max_temp, bins = 25)
plt.hist(weather_data[weather_data.date.dt.month == 12].max_temp, bins = 25)

plt.show()
```



Average temperature for every week

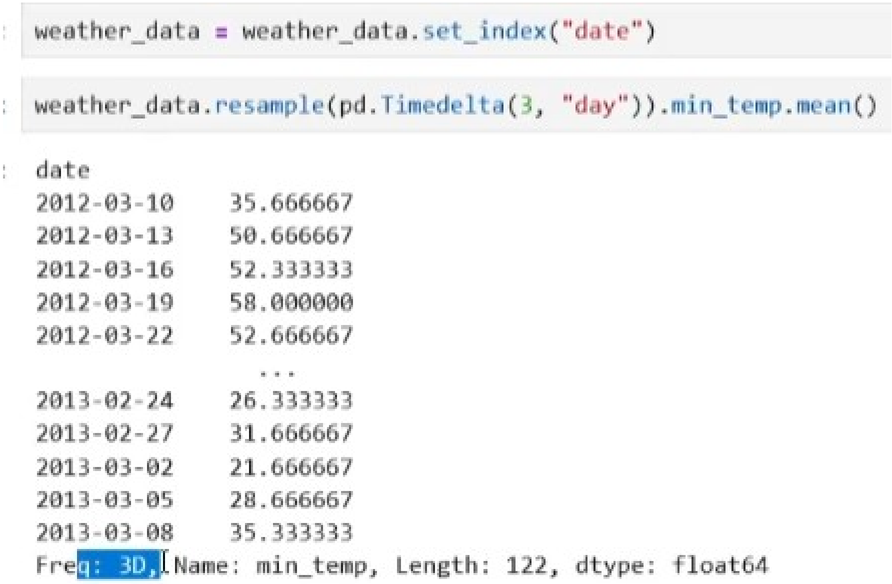


```
weather_data = weather_data.set_index("date")
```

```
weather_data.resample("w").min_temp.mean()
```

```
date
2012-03-11    27.000000
2012-03-18    51.714286
2012-03-25    54.857143
2012-04-01    46.714286
2012-04-08    45.571429
```

M in temp for every 3 days - by 3 days - 10.03/13.03/16.03

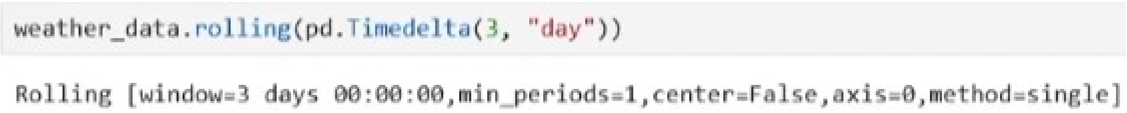


```
weather_data = weather_data.set_index("date")
```

```
weather_data.resample(pd.Timedelta(3, "day")).min_temp.mean()
```

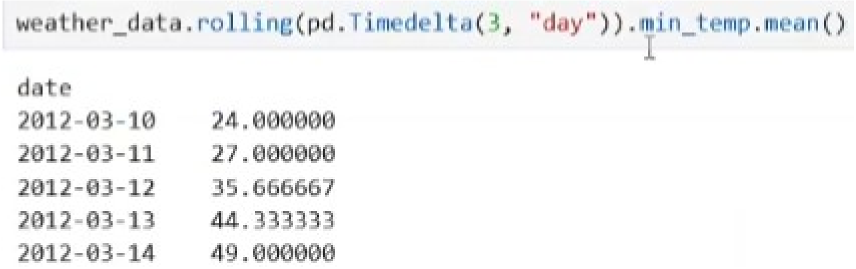
```
date
2012-03-10    35.666667
2012-03-13    50.666667
2012-03-16    52.333333
2012-03-19    58.000000
2012-03-22    52.666667
                 ...
2013-02-24    26.333333
2013-02-27    31.666667
2013-03-02    21.666667
2013-03-05    28.666667
2013-03-08    35.333333
Freq: 3D, Name: min_temp, Length: 122, dtype: float64
```

Or rolling - there is overlap - every day 10.03/11.03/12.03



```
weather_data.rolling(pd.Timedelta(3, "day"))
```

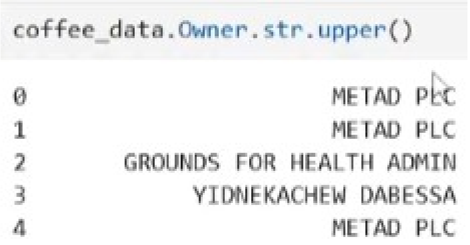
```
Rolling [window=3 days 00:00:00,min_periods=1,center=False,axis=0,method=single]
```



```
weather_data.rolling(pd.Timedelta(3, "day")).min_temp.mean()
```

```
date
2012-03-10    24.000000
2012-03-11    27.000000
2012-03-12    35.666667
2012-03-13    44.333333
2012-03-14    49.000000
```

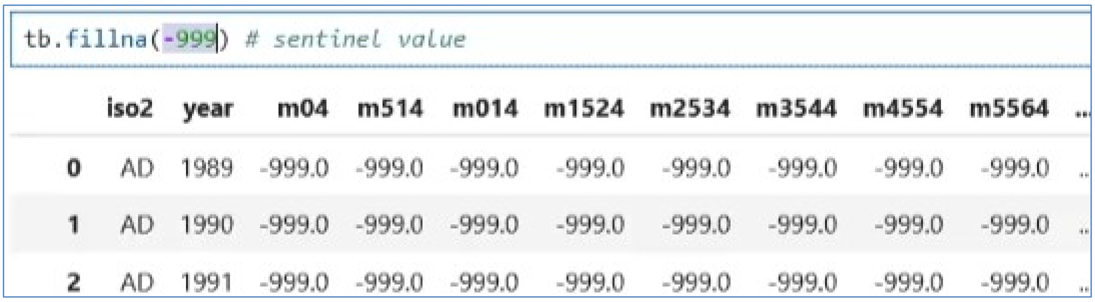
Str transformations



```
coffee_data.Owner.str.upper()
```

```
0                      METAD PLC
1                      METAD PLC
2       GROUNDS FOR HEALTH ADMIN
3              YIDNEKACHEW DABESSA
4                      METAD PLC
```

We could replace the missing data with a dummy value

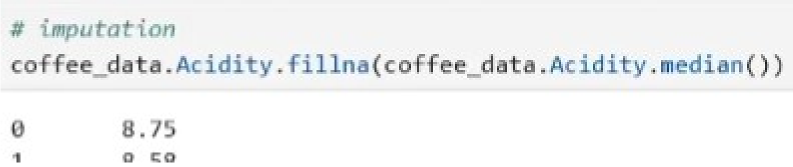


```
tb.fillna(-999) # sentinel value
```

	iso2	year	m04	m514	m014	m1524	m2534	m3544	m4554	m5564	...
0	AD	1989	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	..
1	AD	1990	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	..
2	AD	1991	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	-999.0	..

Median Conclusion:

- Use the mean when your data is normally distributed without outliers.
- Use the median when your data is skewed or contains outliers.



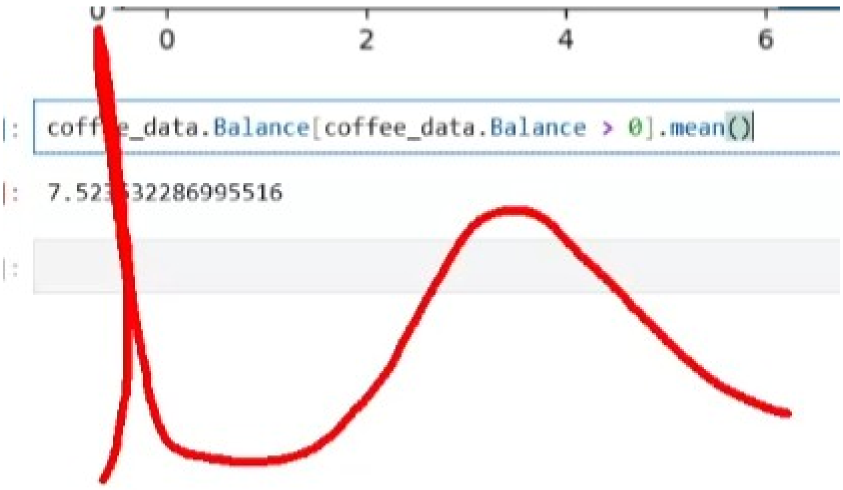
```
# imputation
coffee_data.Acidity.fillna(coffee_data.Acidity.median())
```

```
0        8.75
```

Mean of only positive example - when we have outliers DATASET

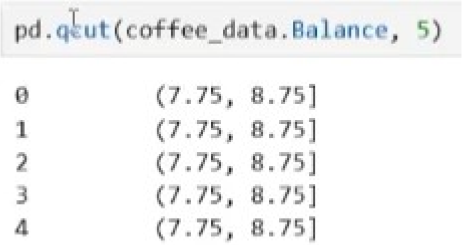
TRANSFORMATIONS

https://scikit-learn.org/stable/modules/preprocessing.html#non-linear-transformation



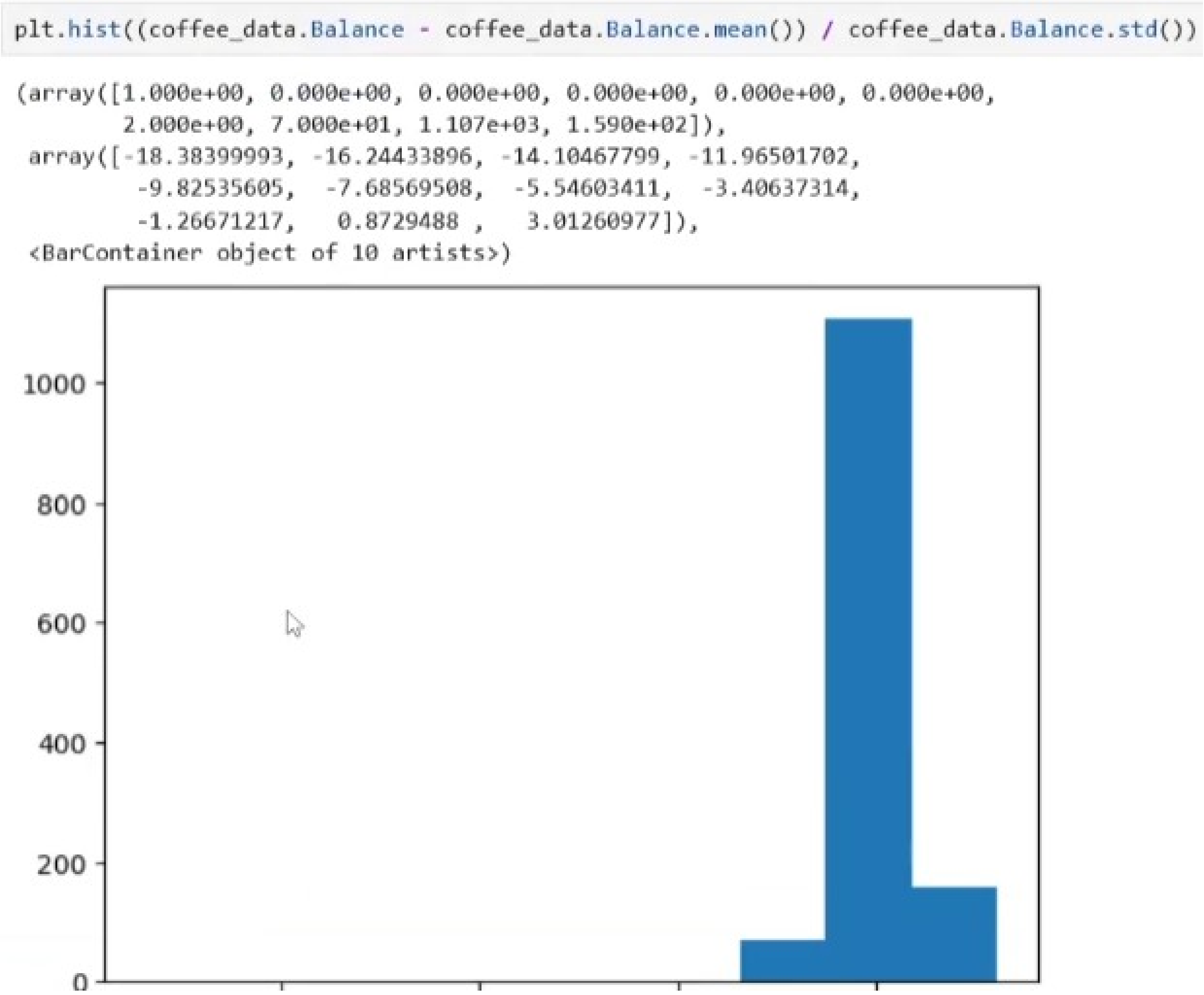
```
]: coffee_data.Balance[coffee_data.Balance > 0].mean()
]: 7.523[illegible]32286995516
```

QCUT



```
pd.qcut(coffee_data.Balance, 5)
```

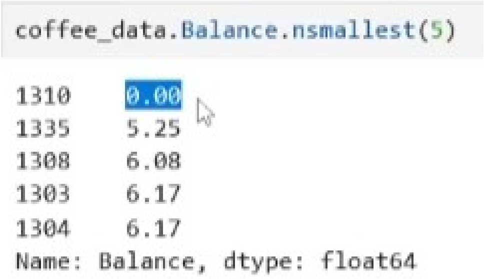
```
0        (7.75, 8.75]
1        (7.75, 8.75]
2        (7.75, 8.75]
3        (7.75, 8.75]
4        (7.75, 8.75]
```



```
plt.hist((coffee_data.Balance - coffee_data.Balance.mean()) / coffee_data.Balance.std())
```

```
(array([1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        2.000e+00, 7.000e+01, 1.107e+03, 1.590e+02]),
 array([-18.38399993, -16.24433896, -14.10467799, -11.96501702,
         -9.82535605,  -7.68569508,  -5.54603411,  -3.40637314,
         -1.26671217,   0.8729488 ,   3.01260977]),
 <BarContainer object of 10 artists>)
```

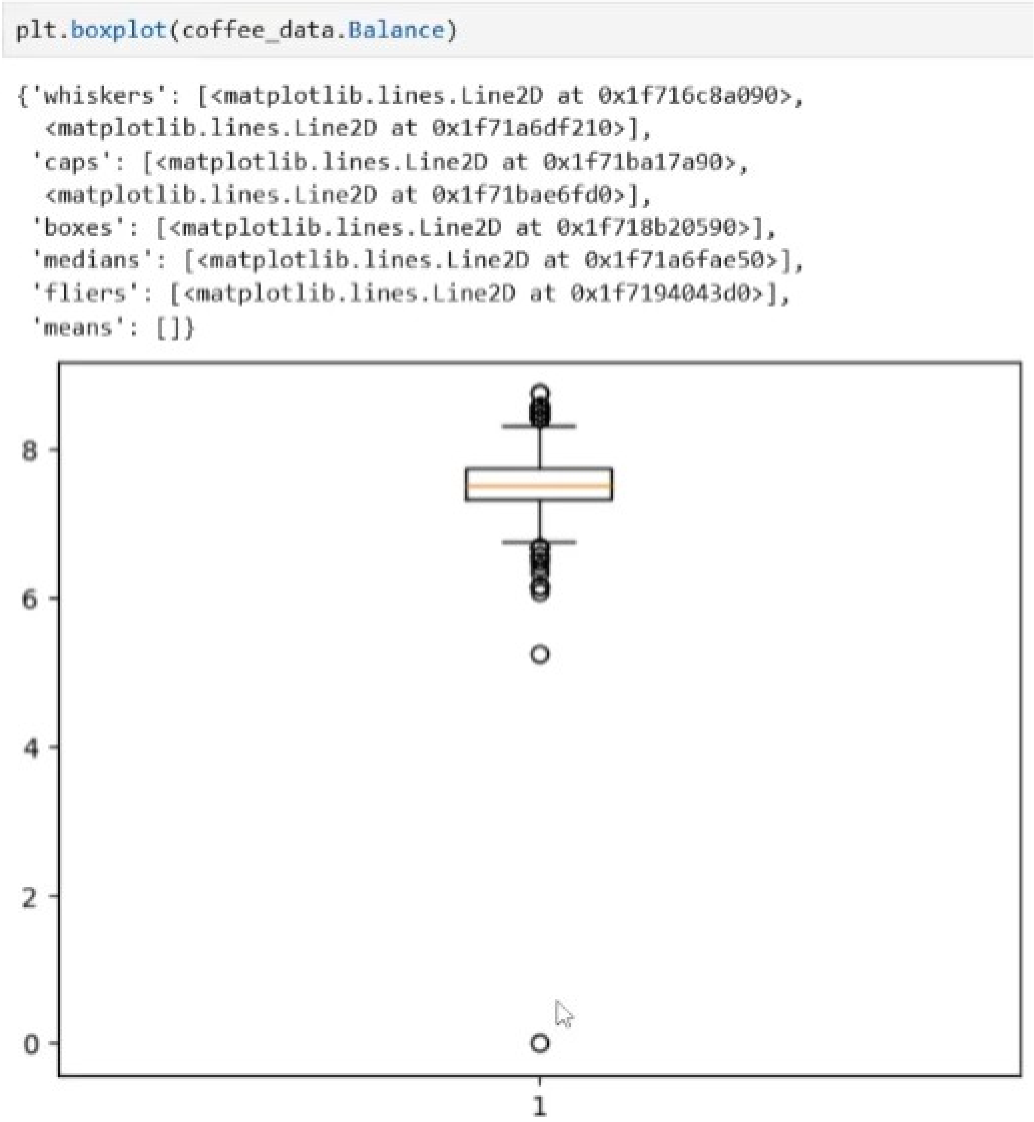
Find smallest 5



```
coffee_data.Balance.nsmallest(5)
```

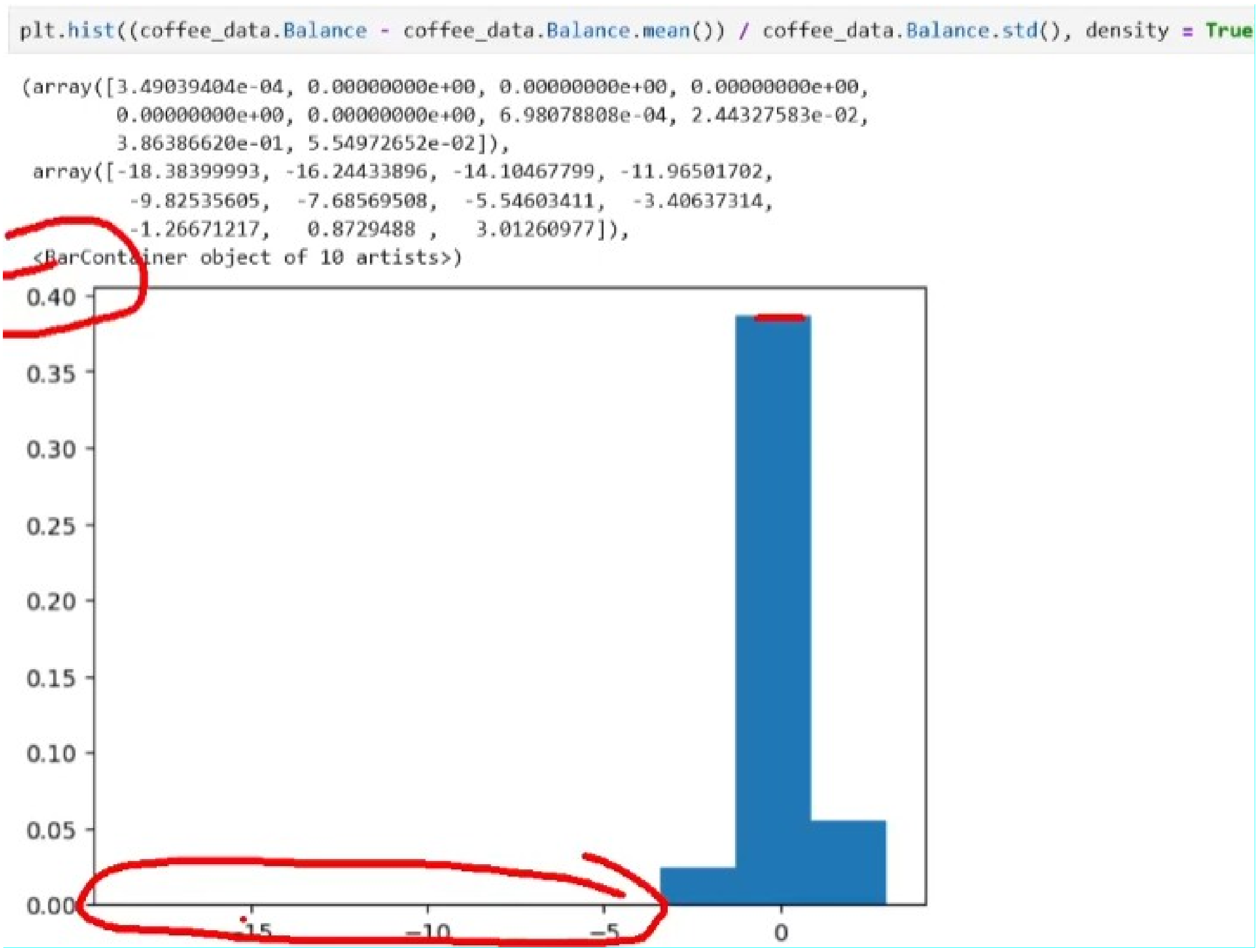
```
1310    0.00
1335    5.25
1308    6.08
1303    6.17
1304    6.17
Name: Balance, dtype: float64
```

Find outliers



```
plt.boxplot(coffee_data.Balance)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1f716c8a090>,
  <matplotlib.lines.Line2D at 0x1f71a6df210>],
 'caps': [<matplotlib.lines.Line2D at 0x1f71ba17a90>,
  <matplotlib.lines.Line2D at 0x1f71bae6fd0>],
 'boxes': [<matplotlib.lines.Line2D at 0x1f718b20590>],
 'medians': [<matplotlib.lines.Line2D at 0x1f71a6fae50>],
 'fliers': [<matplotlib.lines.Line2D at 0x1f7194043d0>],
 'means': []}
```



```
plt.hist((coffee_data.Balance - coffee_data.Balance.mean()) / coffee_data.Balance.std(), density = True
```

```
(array([3.49039404e-04, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 6.98078808e-04, 2.44327583e-02,
        3.86386620e-01, 5.54972652e-02]),
 array([-18.38399993, -16.24433896, -14.10467799, -11.96501702,
         -9.82535605,  -7.68569508,  -5.54603411,  -3.40637314,
         -1.26671217,   0.8729488 ,   3.01260977]),
 <BarContainer object of 10 artists>)
```

Median Conclusion:

- Use the mean when your data is normally distributed without outliers.
- Use the median when your data is skewed or contains outliers.

Mean of only positive example - when we have outliers DATASET

TRANSFORMATIONS

https://scikit-learn.org/stable/modules/preprocessing.html#non-linear-transformation

Pandas api request with authentication ChatGPT
To make an API request with authentication and load the data into a pandas DataFrame, you can use Python's requests library. Below is a step-by-step guide for handling API requests that require authentication, such as using an API key or basic authentication.

1. Install Required Libraries
   If you haven't installed pandas or requests, you can install them using pip: bash
   Копиране на код
   pip install pandas requests
2. Import Libraries
   Start by importing the necessary libraries:

python Копиране на
код
import pandas as pd import
requests

3. Set Up the Authentication

Depending on the type of authentication the API requires, you can set it up as follows: API Key in Headers
If the API uses an API key in the headers: python
Копиране на код

```
url = "https://api.example.com/data" headers = {
"Authorization": "Bearer YOUR_API_KEY", "Content-Type":
"application/json"
}
```

response = requests.get(url, headers=headers) Basic
Authentication
If the API uses basic authentication (username and password): python
Копиране на код

```
url = "https://api.example.com/data"
auth = ("your_username", "your_password")

response = requests.get(url, auth=auth)
```

4. Convert the JSON Response to a DataFrame

Assuming the API returns data in JSON format, convert the response to a pandas DataFrame:
python Копиране на
код
data = response.json() df =
pd.DataFrame(data)

Display the DataFrame

print(df.head()) Example
Workflow
Here’s an example that combines all the steps: python
Копиране на код import
pandas as pd import requests

Example API URL

```
url = "https://api.example.com/data"
```

Headers for API Key Authentication

```
headers = {
"Authorization": "Bearer YOUR_API_KEY", "Content-Type":
"application/json"
}
```

Make the GET request with headers

```
response = requests.get(url, headers=headers)
```

Check if the request was successful

```
if response.status_code == 200:
# Convert the JSON response to a DataFrame data =
response.json()
df = pd.DataFrame(data)
```

```
# Display the DataFrame
print(df.head())
```

```
else:
print(f"Failed to retrieve data: {response.status_code}") Notes:
```

- Handling Errors: It's good practice to check the response status code before attempting to parse the JSON.
- Pagination: If the API paginates the results, you may need to loop through pages and combine the data.
- Rate Limits: Be mindful of API rate limits, and handle them appropriately (e.g., by adding delays or checking headers for rate limit information).

This method enables you to securely access API data using authentication and load it into a pandas DataFrame for further analysis.