Pandas, Matplotlib & Seaborn

Penyusun Modul: Fitria Yunita Dewi

Editor: Citra Chairunnisa

pandas

Pandas is a package commonly used to deal with data analysis. It simplifies the loading of data from external sources such as text files and databases, as well as providing ways of analyzing and manipulating them (its features simplify a lot of the common tasks that would take many lines of code to write in the basic Python language). **Pandas** just like NumPy is written internally in C so it can work fast to process large datasets .

Pandas is best suited for **structured, labelled data**, in other words, **tabular data**, that has headings associated with each column of data. The official **Pandas** website describes **Pandas**' data-handling strengths as:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.
- Any other form of observational / statistical data sets. The data actually need not be labelled at all to be placed into a **pandas** data structure.

Pandas Data Structure

Series:

Series is a **one-dimensional** labelled data structure which can hold data such as strings, integers and even other Python objects.

index	*values*
A	6
B	3.14
C	-4
D	0

DataFrame:

DataFrame is composed of one or more **Series**. The names of the **Series** form the column names, and the row labels form the **Index**.

index ← *columns* →

	foo	bar	baz
A	x	6	True
B	y	10	True
C	z	NaN	False

Creating Series

```
import pandas as pd
s1 = pd.Series([1, 2, 3, 4])
s2 = pd.Series([1, 2, 3, 4], index=['A', 'B', 'C', 'D'])
```

s1	
0	1
1	2
2	3
3	4
dtype: int64	

s2	
A	1
B	2
C	3
D	4
dtype: int64	

Creating DataFrame

```
df = pd.DataFrame({
    'foo': ['x', 'y', 'z'],
    'bar': [6, 10, None],
    'baz': [True, True, False]
    })
```

```
df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	NaN	False

Column Selection

```
df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	NaN	False

```
df['foo']
```

```
0    x
1    y
2    z
Name: foo, dtype: object
```

```
df['bar']
```

```
0     6.0
1    10.0
2     NaN
Name: bar, dtype: float64
```

```
df['baz']
```

```
0     True
1     True
2    False
Name: baz, dtype: bool
```

```
df[['foo','bar']]
```

	foo	bar
0	x	6.0
1	y	10.0
2	z	NaN

Row Selection

```
df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	NaN	False

```
df.loc[0]
```

```
foo        x
bar      6.0
baz     True
Name: 0, dtype: object
```

```
df.loc[1:2]
```

	foo	bar	baz
1	y	10.0	True
2	z	NaN	False

Conditional Filtering

```
df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	NaN	False

```
df[df['baz']]
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True

```
df[(df['foo'] == 'x') | (df['foo'] == 'z')]
```

	foo	bar	baz
0	x	6.0	True
2	z	NaN	False

Data Alignment

```
index_names = ['A','B','C','D','E']
df1 = pd.DataFrame({
     'a': [0, 1, 2, 3],
     'b': [1, 2, 3, 4],
     'c': [2, 3, 4, 5]}, index=index_names[0:4])
df2 = pd.DataFrame({
     'a': [0, 1, 2, 3, 4],
     'b': [1, 2, 3, 4, 5]}, index=index_names)
```

df1

	a	b	c
A	0	1	2
B	1	2	3
C	2	3	4
D	3	4	5

df2

	a	b
A	0	1
B	1	2
C	2	3
D	3	4
E	4	5

df1+df2

	a	b	c
A	0.0	2.0	NaN
B	2.0	4.0	NaN
C	4.0	6.0	NaN
D	6.0	8.0	NaN
E	NaN	NaN	NaN

Handling Missing Values

```
df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	NaN	False

Drop row(s) that contain Null

```
new_df = df.dropna()
new_df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True

Drop column(s) that contain Null

```
new_df = df.dropna(axis=1)
new_df
```

	foo	baz
0	x	True
1	y	True
2	z	False

```
new_df = df.fillna(0)
new_df
```

	foo	bar	baz
0	x	6.0	True
1	y	10.0	True
2	z	0.0	False

Indexing

Use:
- **`iloc[]`** to select rows and columns by their position
- **`loc[]`** to select by name

```
df = pd.DataFrame({
    'foo': ['a', 'b', 'c', 'd'],
    'bar': [6, 10, -2, 1],
    'baz': [True, True, False, True]
    })
```

```
df
```

	foo	bar	baz
0	a	6	True
1	b	10	True
2	c	-2	False
3	d	1	True

```
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
df = df.set_index('foo')
df
```

	bar	baz
foo		
a	6	True
b	10	True
c	-2	False
d	1	True

```
df.loc['a']
```

```
bar        6
baz     True
Name: a, dtype: object
```

```
df.iloc[0]
```

```
bar        6
baz     True
Name: a, dtype: object
```

```
df = df.set_index([['one', 'one', 'two', 'two'], df.index])
df
```

		bar	baz
	foo		
one	a	6	True
	b	10	True
two	c	-2	False
	d	1	True

```
one = df.loc['one']
one
```

	bar	baz
foo		
a	6	True
b	10	True

Let's Try it Out with a DataFrame from CSV File

Download datasets from https://data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh

```
meteorites = pd.read_csv('Meteorite_Landings.csv', nrows=8) # take the first 8 rows
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21	Fell	1880	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720	Fell	1951	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000	Fell	1952	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914	Fell	1976	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780	Fell	1902	-33.16667	-64.95000	(-33.16667, -64.95)
5	Adhi Kot	379	Valid	EH4	4239	Fell	1919	32.10000	71.80000	(32.1, 71.8)
6	Adzhi-Bogdo (stone)	390	Valid	LL3-6	910	Fell	1949	44.83333	95.16667	(44.83333, 95.16667)
7	Agen	392	Valid	H5	30000	Fell	1814	44.21667	0.61667	(44.21667, 0.61667)

The Anatomy

```
# Series
meteorites.name
```

```
0                  Aachen
1                  Aarhus
2                    Abee
3                Acapulco
4                 Achiras
5                Adhi Kot
6     Adzhi-Bogdo (stone)
7                    Agen
Name: name, dtype: object
```

```
# Columns
meteorites.columns
```

```
Index(['name', 'id', 'nametype', 'recclass', 'mass (g)', 'fall', 'year',
       'reclat', 'reclong', 'GeoLocation'],
      dtype='object')
```

```
# Index
meteorites.index
```

```
RangeIndex(start=0, stop=8, step=1)
```

Inspecting the Data

```python
# take all the data
meteorites = pd.read_csv('Meteorite_Landings.csv')
```

Information about the DataFrame

```python
meteorites.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45716 entries, 0 to 45715
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   name         45716 non-null  object
 1   id           45716 non-null  int64
 2   nametype     45716 non-null  object
 3   recclass     45716 non-null  object
 4   mass (g)     45585 non-null  float64
 5   fall         45716 non-null  object
 6   year         45425 non-null  float64
 7   reclat       38401 non-null  float64
 8   reclong      38401 non-null  float64
 9   GeoLocation  38401 non-null  object
dtypes: float64(4), int64(1), object(5)
memory usage: 3.5+ MB
```

What type of data does each column currently hold?

```python
meteorites.dtypes
```

```
name            object
id               int64
nametype        object
recclass        object
mass (g)       float64
fall            object
year           float64
reclat         float64
reclong        float64
GeoLocation     object
dtype: object
```

How many rows and columns are there?

```python
meteorites.shape
```

```
(45716, 10)
```

What are the column names?

```python
meteorites.columns
```

```
Index(['name', 'id', 'nametype', 'recclass', 'mass (g)', 'fall', 'year',
       'reclat', 'reclong', 'GeoLocation'],
      dtype='object')
```

What does the data look like?

```python
meteorites.head()
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880.0	50.77500	6.08333	(50.775, 6.08333)
1	Aarhus	2	Valid	H6	720.0	Fell	1951.0	56.18333	10.23333	(56.18333, 10.23333)
2	Abee	6	Valid	EH4	107000.0	Fell	1952.0	54.21667	-113.00000	(54.21667, -113.0)
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976.0	16.88333	-99.90000	(16.88333, -99.9)
4	Achiras	370	Valid	L6	780.0	Fell	1902.0	-33.16667	-64.95000	(-33.16667, -64.95)

```python
meteorites.tail()
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
45711	Zillah 002	31356	Valid	Eucrite	172.0	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45712	Zinder	30409	Valid	Pallasite, ungrouped	46.0	Found	1999.0	13.78333	8.96667	(13.78333, 8.96667)
45713	Zlin	30410	Valid	H4	3.3	Found	1939.0	49.25000	17.66667	(49.25, 17.66667)
45714	Zubkovsky	31357	Valid	L6	2167.0	Found	2003.0	49.78917	41.50460	(49.78917, 41.5046)
45715	Zulu Queen	30414	Valid	L3.7	200.0	Found	1976.0	33.98333	-115.68333	(33.98333, -115.68333)

Column and Row Selection

Selecting Column(s)

```
meteorites.name
```

```
0              Aachen
1              Aarhus
2                Abee
3            Acapulco
4             Achiras
               ...
45711      Zillah 002
45712          Zinder
45713            Zlin
45714       Zubkovsky
45715      Zulu Queen
Name: name, Length: 45716, dtype: object
```

```
meteorites[['name', 'mass (g)']]
```

	name	mass (g)
0	Aachen	21.0
1	Aarhus	720.0
2	Abee	107000.0
3	Acapulco	1914.0
4	Achiras	780.0
...	...	...
45711	Zillah 002	172.0
45712	Zinder	46.0
45713	Zlin	3.3
45714	Zubkovsky	2167.0
45715	Zulu Queen	200.0

45716 rows × 2 columns

Selecting Row(s)

```
meteorites[100:104]
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
100	Benton	5026	Valid	LL6	2840.0	Fell	1949.0	45.95000	-67.55000	(45.95, -67.55)
101	Berduc	48975	Valid	L6	270.0	Fell	2008.0	-31.91000	-58.32833	(-31.91, -58.32833)
102	Béréba	5028	Valid	Eucrite-mmict	18000.0	Fell	1924.0	11.65000	-3.65000	(11.65, -3.65)
103	Berlanguillas	5029	Valid	L6	1440.0	Fell	1811.0	41.68333	-3.80000	(41.68333, -3.8)

Indexing

```
meteorites.iloc[100:104, [0, 3, 4, 6]]
```

	name	recclass	mass (g)	year
100	Benton	LL6	2840.0	01/01/1949 12:00:00 AM
101	Berduc	L6	270.0	01/01/2008 12:00:00 AM
102	Béréba	Eucrite-mmict	18000.0	01/01/1924 12:00:00 AM
103	Berlanguillas	L6	1440.0	01/01/1811 12:00:00 AM

```
meteorites.loc[100:104, 'mass (g)':'year']
```

	mass (g)	fall	year
100	2840.0	Fell	01/01/1949 12:00:00 AM
101	270.0	Fell	01/01/2008 12:00:00 AM
102	18000.0	Fell	01/01/1924 12:00:00 AM
103	1440.0	Fell	01/01/1811 12:00:00 AM
104	960.0	Fell	01/01/2004 12:00:00 AM

Filtering

Important: Take note of the syntax here. We surround each condition with parentheses, and we use bitwise operators **(&, |, ~)** instead of logical operators (and, or, not).

```
(meteorites['mass (g)'] > 50) & (meteorites.fall == 'Found')
```

```
0        False
1        False
2        False
3        False
4        False
          ...
45711     True
45712    False
45713    False
45714     True
45715     True
Length: 45716, dtype: bool
```

```
meteorites[(meteorites['mass (g)'] > 50) & (meteorites.fall == 'Found')]
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
37	Northwest Africa 5815	50693	Valid	L5	256.80	Found	NaN	0.00000	0.00000	(0.0, 0.0)
757	Dominion Range 03239	32591	Valid	L6	69.50	Found	2002.0	NaN	NaN	NaN
804	Dominion Range 03240	32592	Valid	LL5	290.90	Found	2002.0	NaN	NaN	NaN
1111	Abajo	4	Valid	H5	331.00	Found	1982.0	26.80000	-105.41667	(26.8, -105.41667)
1112	Abar al' Uj 001	51399	Valid	H3.8	194.34	Found	2008.0	22.72192	48.95937	(22.72192, 48.95937)
...	...	...	...	...	...	...	...	...	...	...
45709	Zhongxiang	30406	Valid	Iron	100000.00	Found	1981.0	31.20000	112.50000	(31.2, 112.5)
45710	Zillah 001	31355	Valid	L6	1475.00	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45711	Zillah 002	31356	Valid	Eucrite	172.00	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45714	Zubkovsky	31357	Valid	L6	2167.00	Found	2003.0	49.78917	41.50460	(49.78917, 41.5046)
45715	Zulu Queen	30414	Valid	L3.7	200.00	Found	1976.0	33.98333	-115.68333	(33.98333, -115.68333)

18854 rows × 10 columns

Filtering alternative with query()

```
meteorites.query("`mass (g)` > 50 and fall == 'Found'")
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
37	Northwest Africa 5815	50693	Valid	L5	256.80	Found	NaN	0.00000	0.00000	(0.0, 0.0)
757	Dominion Range 03239	32591	Valid	L6	69.50	Found	2002.0	NaN	NaN	NaN
804	Dominion Range 03240	32592	Valid	LL5	290.90	Found	2002.0	NaN	NaN	NaN
1111	Abajo	4	Valid	H5	331.00	Found	1982.0	26.80000	-105.41667	(26.8, -105.41667)
1112	Abar al' Uj 001	51399	Valid	H3.8	194.34	Found	2008.0	22.72192	48.95937	(22.72192, 48.95937)
...	...	...	...	...	...	...	...	...	...	...
45709	Zhongxiang	30406	Valid	Iron	100000.00	Found	1981.0	31.20000	112.50000	(31.2, 112.5)
45710	Zillah 001	31355	Valid	L6	1475.00	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45711	Zillah 002	31356	Valid	Eucrite	172.00	Found	1990.0	29.03700	17.01850	(29.037, 17.0185)
45714	Zubkovsky	31357	Valid	L6	2167.00	Found	2003.0	49.78917	41.50460	(49.78917, 41.5046)
45715	Zulu Queen	30414	Valid	L3.7	200.00	Found	1976.0	33.98333	-115.68333	(33.98333, -115.68333)

18854 rows × 10 columns

Calculating summary statistics

Get some summary statistics on the data itself

```
meteorites.describe(include='all')
```

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation
count	45716	45716.000000	45716	45716	4.558500e+04	45716	45425	38401.000000	38401.000000	38401
unique	45716	NaN	2	466	NaN	2	266	NaN	NaN	17100
top	Yamato 86397	NaN	Valid	L6	NaN	Found	01/01/2003 12:00:00 AM	NaN	NaN	(0.0, 0.0)
freq	1	NaN	45641	8285	NaN	44609	3323	NaN	NaN	6214
mean	NaN	26889.735104	NaN	NaN	1.327808e+04	NaN	NaN	-39.122580	61.074319	NaN
std	NaN	16860.683030	NaN	NaN	5.749889e+05	NaN	NaN	46.378511	80.647298	NaN
min	NaN	1.000000	NaN	NaN	0.000000e+00	NaN	NaN	-87.366670	-165.433330	NaN
25%	NaN	12688.750000	NaN	NaN	7.200000e+00	NaN	NaN	-76.714240	0.000000	NaN
50%	NaN	24261.500000	NaN	NaN	3.260000e+01	NaN	NaN	-71.500000	35.666670	NaN
75%	NaN	40656.750000	NaN	NaN	2.026000e+02	NaN	NaN	0.000000	157.166670	NaN
max	NaN	57458.000000	NaN	NaN	6.000000e+07	NaN	NaN	81.166670	354.473330	NaN

Important: `NaN` values signify missing data. For instance, the `fall` column contains strings, so there is no value for `mean`; likewise, `mass (g)` is numeric, so we don't have entries for the categorical summary statistics (`unique`, `top`, `freq`).

How many of the meteorites were found versus observed falling?

```
meteorites.fall.value_counts()
```

```
Found    44609
Fell      1107
Name: fall, dtype: int64
```

Tip: Pass in `normalize=True` to see this result as percentages. Check the documentation for additional functionality.

What was the mass of the heaviest meteorite?

```
meteorites['mass (g)'].max()
```

```
60000000.0
```

Your Turn

Let's take a break for some exercises to check your understanding

1. Create a DataFrame by reading in the `2019_Yellow_Taxi_Trip_Data.csv` file.
2. Find the dimensions (number of rows and number of columns) in the data.
3. Calculate summary statistics for the `fare_amount`, `tip_amount`, `tolls_amount`, and `total_amount` columns.
4. Isolate the `fare_amount`, `tip_amount`, `tolls_amount`, and `total_amount` for the longest trip (trip_distance).

Data Wrangling

Let's continue our process of data wrangling to prepare our data for analysis.
Now we'll be working with the data from previous exercise, 2019 Yellow Taxi Trip Data provided by NYC Open Data.

```
taxis = pd.read_csv('../data/2019_Yellow_Taxi_Trip_Data.csv')
taxis.head()
```

	vendorid	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	ratecodeid	store_and_fwd_flag	pulocationid	dolocationid	payment_type	fare_amount
0	2	2019-10-23T16:39:42.000	2019-10-23T17:14:10.000	1	7.93	1	N	138	170	1	29.5
1	1	2019-10-23T16:32:08.000	2019-10-23T16:45:26.000	1	2.00	1	N	11	26	1	10.5
2	2	2019-10-23T16:08:44.000	2019-10-23T16:21:11.000	1	1.36	1	N	163	162	1	9.5
3	2	2019-10-23T16:22:44.000	2019-10-23T16:43:26.000	1	1.00	1	N	170	163	1	13.0
4	2	2019-10-23T16:45:11.000	2019-10-23T16:58:49.000	1	1.96	1	N	163	236	1	10.5

Data Cleaning – Drop Unused Columns

```
taxis.columns
```

```
Index(['Unnamed: 0', 'vendorid', 'tpep_pickup_datetime',
       'tpep_dropoff_datetime', 'passenger_count', 'trip_distance',
       'ratecodeid', 'store_and_fwd_flag', 'pulocationid', 'dolocationid',
       'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
       'tolls_amount', 'improvement_surcharge', 'total_amount',
       'congestion_surcharge'],
      dtype='object')
```

Let's start by dropping the `ID` columns and the `store_and_fwd_flag column`, which we won't be using.

```
mask = taxis.columns.str.contains('id$|store_and_fwd_flag')
mask
```

```
array([False,  True, False, False, False, False,  True,  True,  True,
        True, False, False, False, False, False, False, False, False,
       False])
```

```
columns_to_drop = taxis.columns[mask]
taxis = taxis.drop(columns=columns_to_drop)
taxis.columns
```

```
Index(['Unnamed: 0', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'payment_type', 'fare_amount',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount', 'congestion_surcharge'],
      dtype='object')
```

Data Cleaning – Rename Columns

```
taxis.rename(
    columns={
        'tpep_pickup_datetime': 'pickup',
        'tpep_dropoff_datetime': 'dropoff'
    },
    inplace=True
)
taxis.head()
```

	Unnamed: 0	pickup	dropoff	passenger_count
0	0	2019-11-08T10:14:52.000	2019-11-08T10:37:42.000	5
1	1	2019-11-08T10:50:54.000	2019-11-08T10:59:11.000	5
2	2	2019-11-08T10:08:31.000	2019-11-08T10:12:34.000	1
3	3	2019-11-08T10:13:59.000	2019-11-08T10:27:47.000	1
4	4	2019-11-08T10:34:08.000	2019-11-08T11:10:25.000	1

Before we continue, let's change the datatypes of 'pickup' and 'dropoff' columns

```
taxis.dtypes
```

```
pickup                    object
dropoff                   object
passenger_count            int64
trip_distance            float64
payment_type               int64
fare_amount              float64
extra                    float64
mta_tax                  float64
tip_amount               float64
tolls_amount             float64
improvement_surcharge    float64
total_amount             float64
congestion_surcharge     float64
dtype: object
```

```
taxis['pickup'] = pd.to_datetime(taxis['pickup'])
taxis['dropoff'] = pd.to_datetime(taxis['dropoff'])
taxis.dtypes
```

```
Unnamed: 0                        int64
pickup                   datetime64[ns]
dropoff                  datetime64[ns]
passenger_count                   int64
trip_distance                   float64
payment_type                      int64
fare_amount                     float64
extra                           float64
mta_tax                         float64
tip_amount                      float64
tolls_amount                    float64
improvement_surcharge           float64
total_amount                    float64
congestion_surcharge            float64
dtype: object
```

Data Cleaning – Create New Columns

There are several ways to do this:

- Use indexing
- Use `assign()`
- Use `insert()`

```
taxis['cost_before_tip']=taxis['total_amount'] - taxis['tip_amount'] # using indexing
taxis = taxis.assign(tip_percent=taxis['tip_amount'] / taxis['cost_before_tip']) # using assign()
taxis.head()
```

extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge	cost_before_tip	tip_percent
0.0	0.5	3.46	0.0	0.3	20.76	2.5	17.3	0.2
				0.3	12.36	2.5	10.3	0.2
				0.3	9.96	2.5	8.3	0.2
				0.3	16.56	2.5	13.8	0.2
				0.3	36.96	2.5	30.8	0.2

```
taxis.insert(3, "elapsed_time", taxis['dropoff']-taxis['pickup'])
taxis.head()
```

Unnamed: 0	pickup	dropoff	elapsed_time	passenger_count	trip_distance	payment_type
0	2019-11-08 10:14:52	2019-11-08 10:37:42	0 days 00:22:50	5	1.55	1
1	2019-11-08 10:50:54	2019-11-08 10:59:11	0 days 00:08:17	5	0.84	1
2	2019-11-08 10:08:31	2019-11-08 10:12:34	0 days 00:04:03	1	0.72	1
3	2019-11-08	2019-11-08	0 days 00:13:48	1	1.62	1

Data Cleaning – Sort by Values

```
taxis.sort_values(['passenger_count', 'pickup'], ascending=[False, True])
```

	Unnamed: 0	pickup	dropoff	elapsed_time	passenger_count	trip_distance	payment_type
34540	34540	2019-11-08 06:02:37	2019-11-08 06:37:14	0 days 00:34:37	6	10.21	1
8767	8767	2019-11-08 06:16:36	2019-11-08 06:32:09	0 days 00:15:33	6	1.33	1
47543	47543	2019-11-08 06:19:06	2019-11-08 13:51:58	0 days 07:32:52	6	6.14	1
21429	21429	2019-11-08 06:24:32	2019-11-09 05:59:52	0 days 23:35:20	6	4.14	1
8768	8768	2019-11-08 06:40:20	2019-11-09 05:52:41	0 days 23:12:21	6	3.46	2
...	...	...	...	...	...	...	...
46020	46020	2019-11-08 13:59:17	2019-11-08 14:08:39	0 days 00:09:22	0	0.80	1

Data Visualization

The human brain excels at finding patterns in visual representations of the data; so in this section, we will learn how to visualize data that will help us better understand our data. Python features many libraries that provide useful tools for visualization.

The most well-known, Matplotlib, enables users to generate visualizations like histograms, scatterplots, bar charts, pie charts and much more.

Seaborn is another useful visualization library that is built on top of Matplotlib. It provides data visualizations that are typically more aesthetic and statistically sophisticated.

Having a solid understanding of how to use both of these libraries is essential for any data scientist or data analyst as they both provide easy methods for visualizing data for insight.

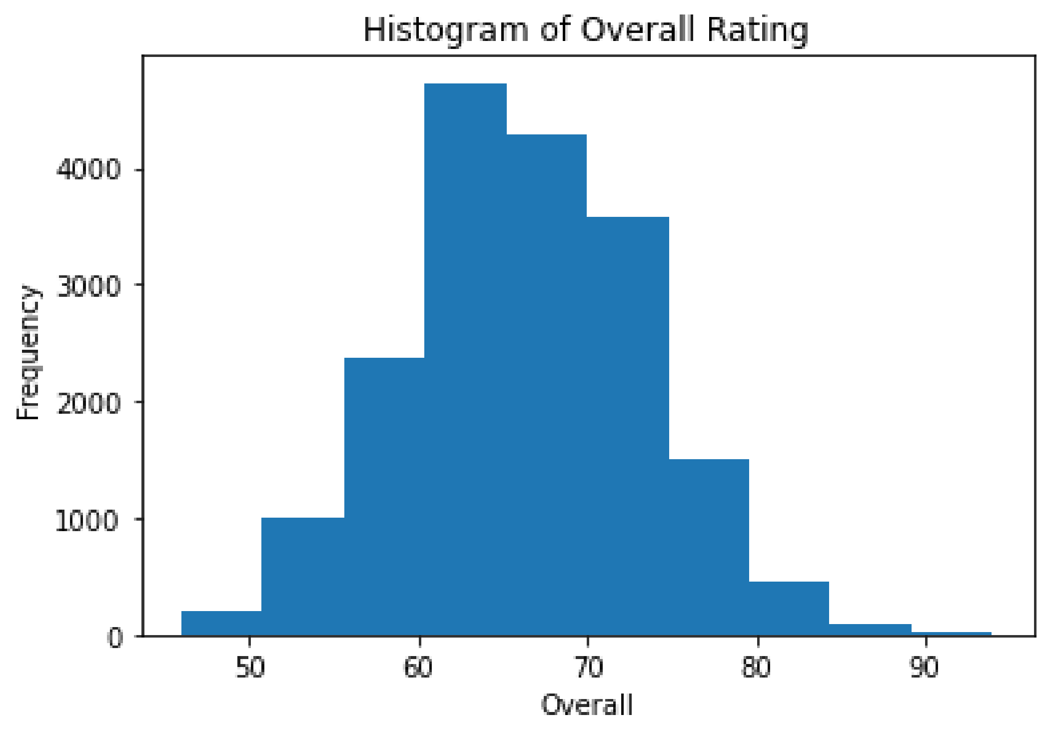
Generating Histograms

When analyzing a new data set, researchers are often interested in the distribution of values for a set of columns. One way to do so is through a histogram.

```python
import matplotlib.pyplot as plt
df = pd.read_csv("fifa_eda.csv")
df.head()
```

	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage
0	158023	L. Messi	31	Argentina	94	94	FC Barcelona	110500.0	565.0
1	20801	Cristiano Ronaldo	33	Portugal	94	94	Juventus	77000.0	405.0
2	190871	Neymar Jr	26	Brazil	92	93	Paris Saint-Germain	118500.0	290.0
3	193080	De Gea	27	Spain	91	93	Manchester United	72000.0	260.0
4	192985	K. De Bruyne	27	Belgium	91	92	Manchester City	102000.0	355.0

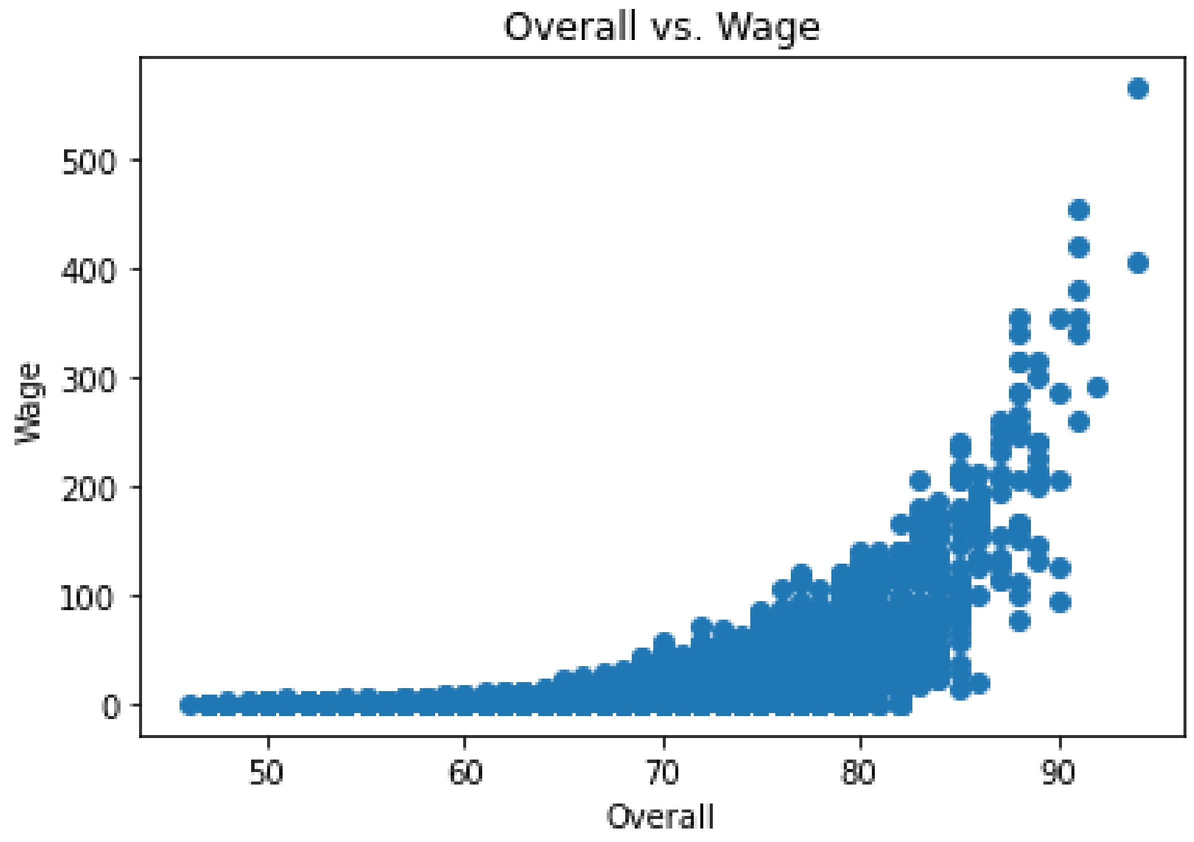
```python
plt.hist(df['Overall'])
plt.xlabel('Overall')
plt.ylabel('Frequency')
plt.title('Histogram of Overall Rating')
plt.show()
```



Generating Scatterplots

Scatterplots are a useful data visualization tool that helps with identifying variable dependence.

```
plt.scatter(df['Overall'], df['Wage'])
plt.title('Overall vs. Wage')
plt.ylabel('Wage')
plt.xlabel('Overall')
plt.show()
```



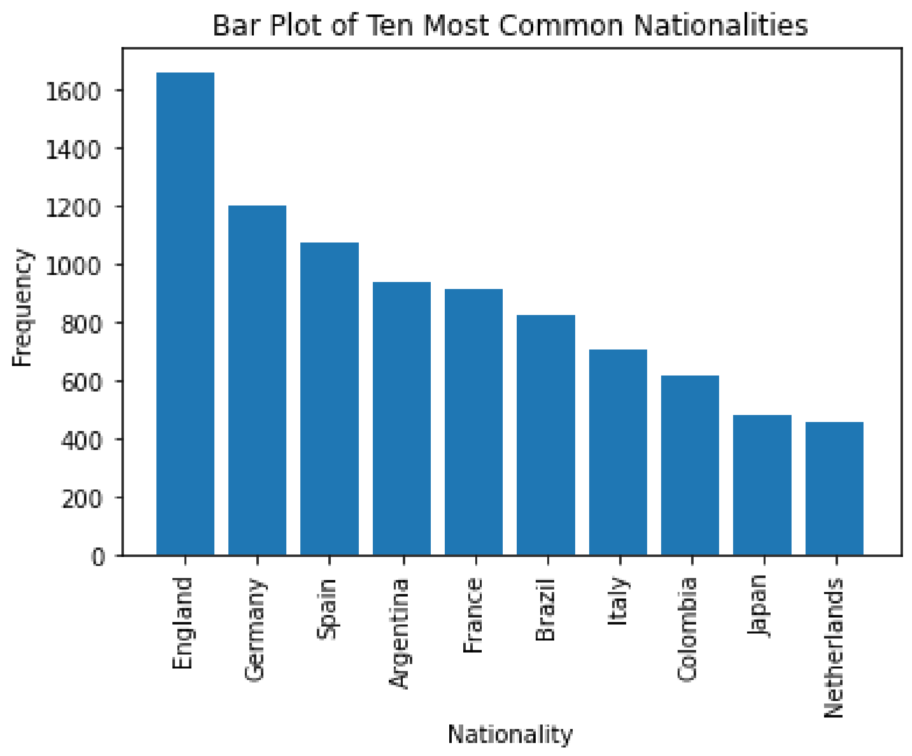
Generating Bar Charts

Bar charts are another useful visualization tool for analyzing categories in data. For example, we want to see the most common nationalities found in our FIFA19 data set

```
# creating new series of no. of players based on their nationality
nationality_count = df.Nationality.value_counts()
nationality_count
```

```
England         1662
Germany         1198
Spain           1072
Argentina        937
France           914
                ...
Puerto Rico        1
Fiji               1
St Lucia           1
Palestine          1
Lebanon            1
Name: Nationality, Length: 164, dtype: int64
```

```
plt.bar(nationality_count.index[0:10], nationality_count.values[0:10]) # we only look at the first 10
plt.xlabel('Nationality')
plt.ylabel('Frequency')
plt.title('Bar Plot of Ten Most Common Nationalities')
plt.xticks(rotation=90)
plt.show()
```



Generating Pie Charts

Pie charts are a useful way to visualize proportions in your data. For example, in this data set, we can use a pie chart to visualize the proportion of players from England, Germany and Spain.

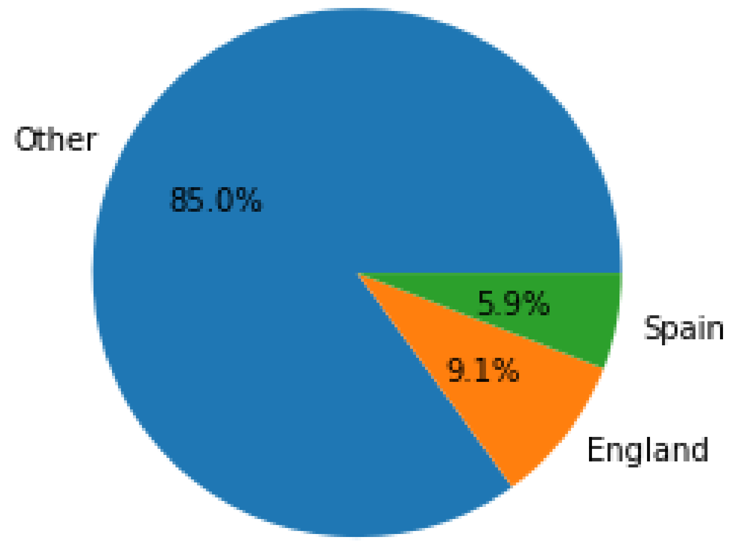
```
# add column named Nationality2
# assign value to each row satisfying the condition
# loc[rows where the condition is satisfied, column]
# here we create 4 categories of Nationality2

df.loc[df.Nationality =='England',  'Nationality2'] = 'England'
df.loc[df.Nationality =='Spain',  'Nationality2'] = 'Spain'
df.loc[df.Nationality =='Germany',  'Nationality2'] = 'Germany'
df.loc[~df.Nationality.isin(['England', 'German', 'Spain']),  'Nationality2'] = 'Other'

# count values in Nationality2 column
nationality2_count = df['Nationality2'].value_counts()
# same as df.value_counts(['Nationality2']) or df.Nationality2.value_counts()
nationality2_count
```

```
Other      15473
England     1662
Spain       1072
Name: Nationality2, dtype: int64
```

```
plt.pie(nationality2_count, labels=nationality2_count.index,
        autopct='%1.1f%%')
plt.show()
```



Now Let’s Move On to Seaborn

Seaborn is a library built on top of Matplotlib that enables more sophisticated visualization and aesthetic plot formatting. Once you’ve mastered Matplotlib, you may want to move up to Seaborn for more complex visualizations.

For example, simply using the Seaborn set() method can dramatically improve the appearance of your Matplotlib plots. Let’s take a look.

First, import Seaborn as sns

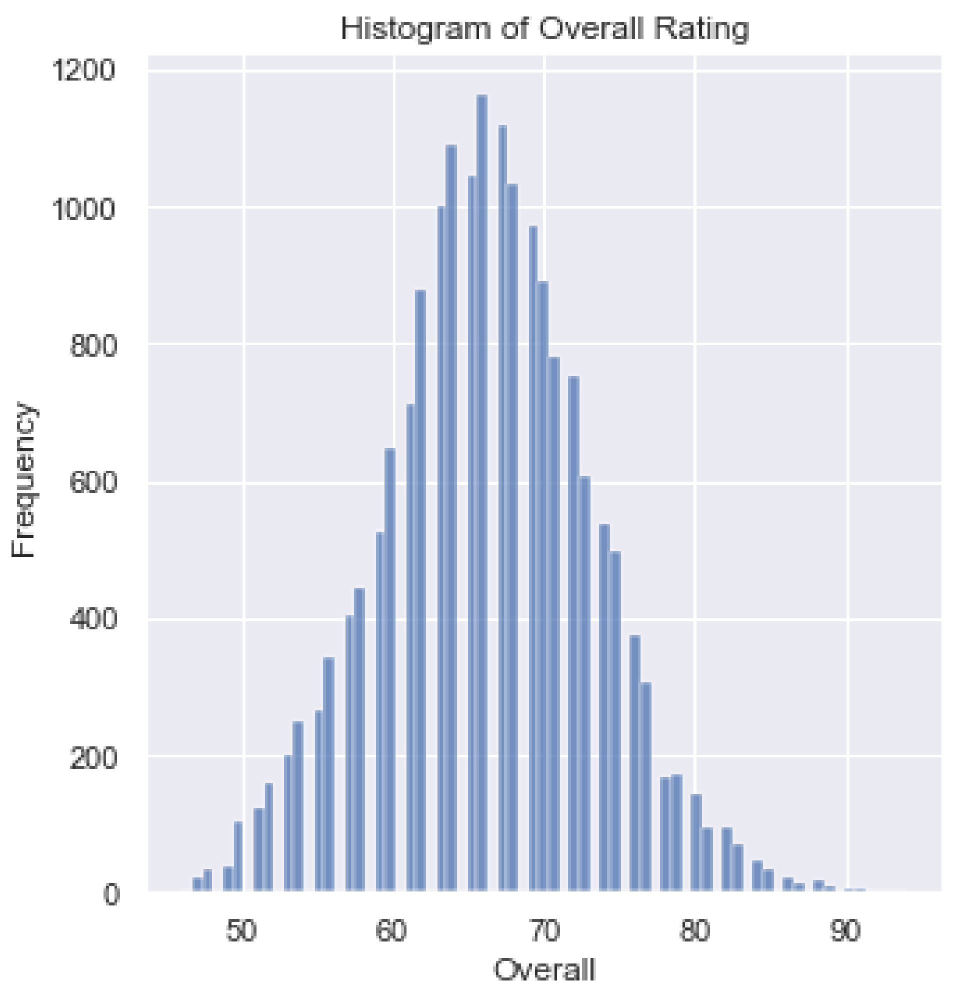
```
import seaborn as sns
```

Generating Histograms

We can also generate all of the same visualizations we did in Matplotlib using Seaborn.

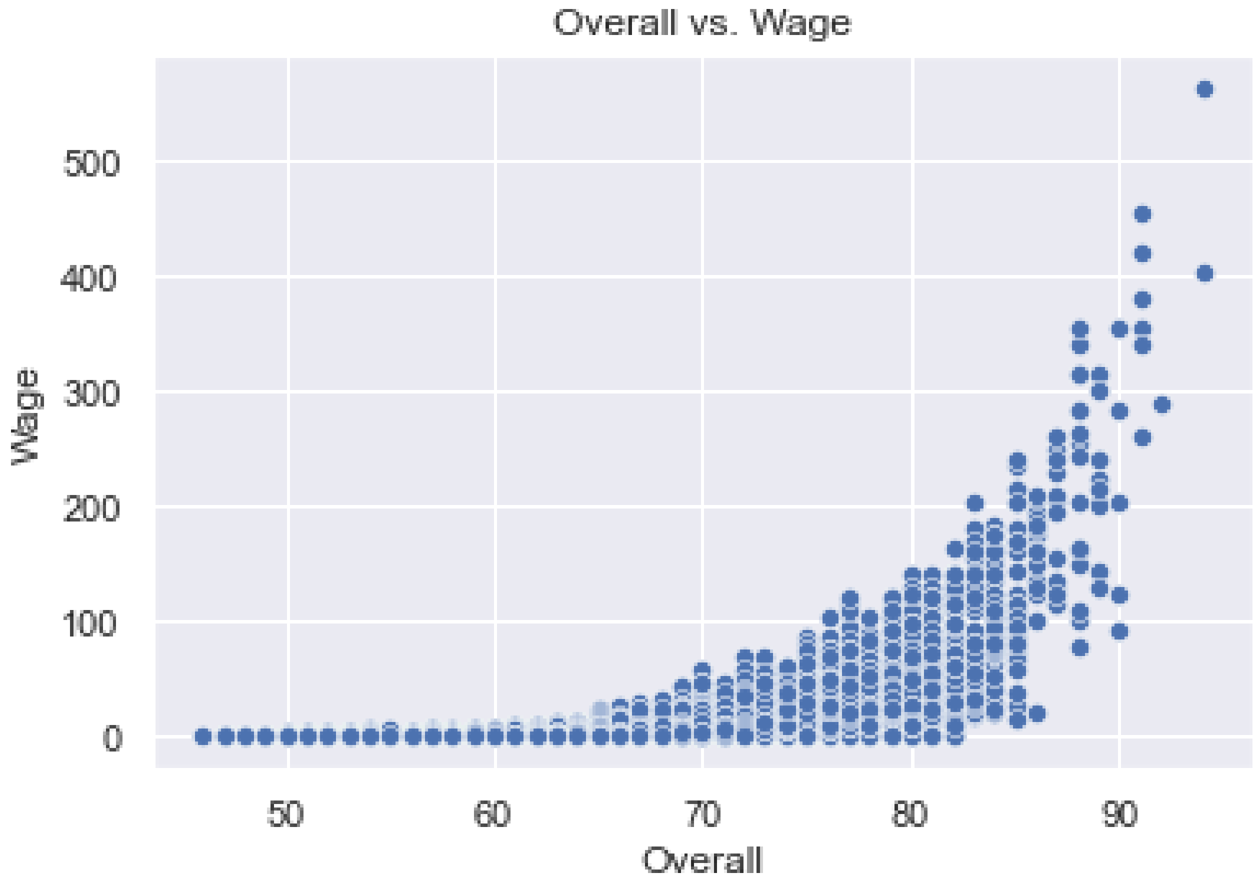
To regenerate our histogram of the overall column, we use the distplot method on the Seaborn object:

```
sns.displot(df['Overall'])
plt.xlabel('Overall')
plt.ylabel('Frequency')
plt.title('Histogram of Overall Rating')
plt.show()
```



Generating Scatterplots

```
sns.scatterplot(x=df['Overall'], y=df['Wage'])
plt.title('Overall vs. Wage')
plt.ylabel('Wage')
plt.xlabel('Overall')
plt.show()
```

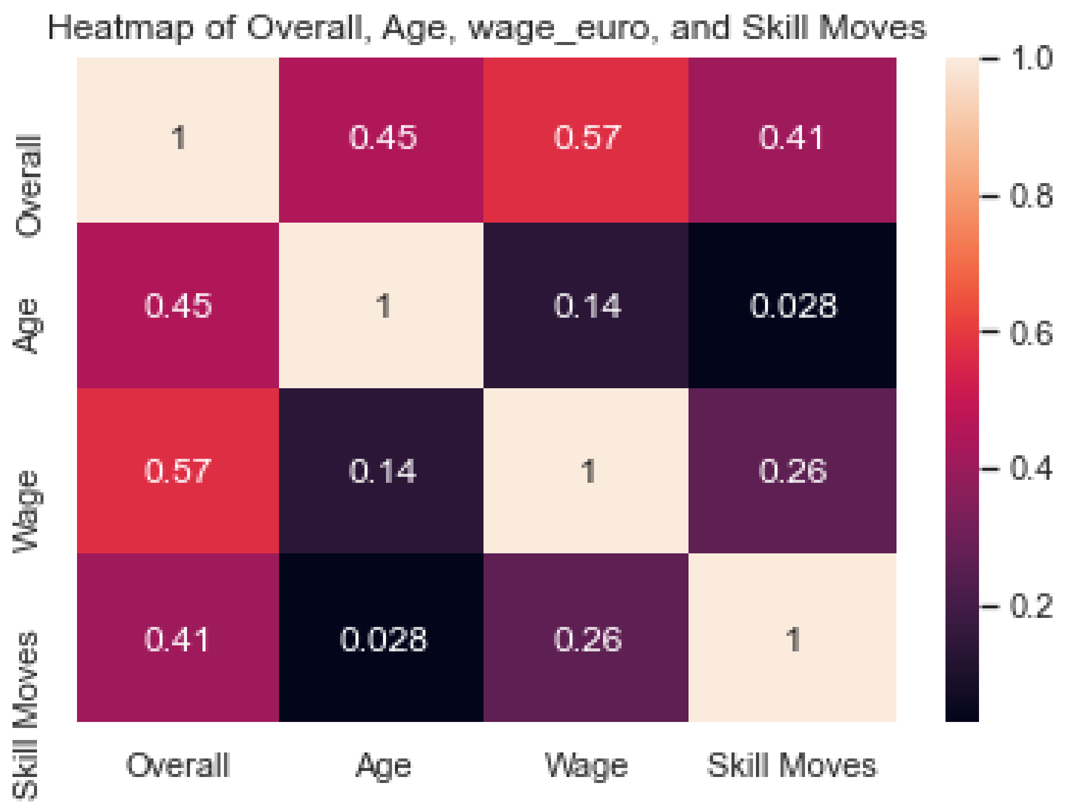


Generating Heatmaps

Seaborn is also known for making correlation heatmaps, which can be used to identify variable dependence. To generate one, first we need to calculate the correlation between a set of numerical columns. Let's do this for age, overall, wage_euro and skill moves

These correlation values can help us selecting features later on when we learn more about machine learning. Features/variables with high correlation are more linearly dependent and hence have almost the same effect. So, when two features have high correlation, we can drop one of the two features.

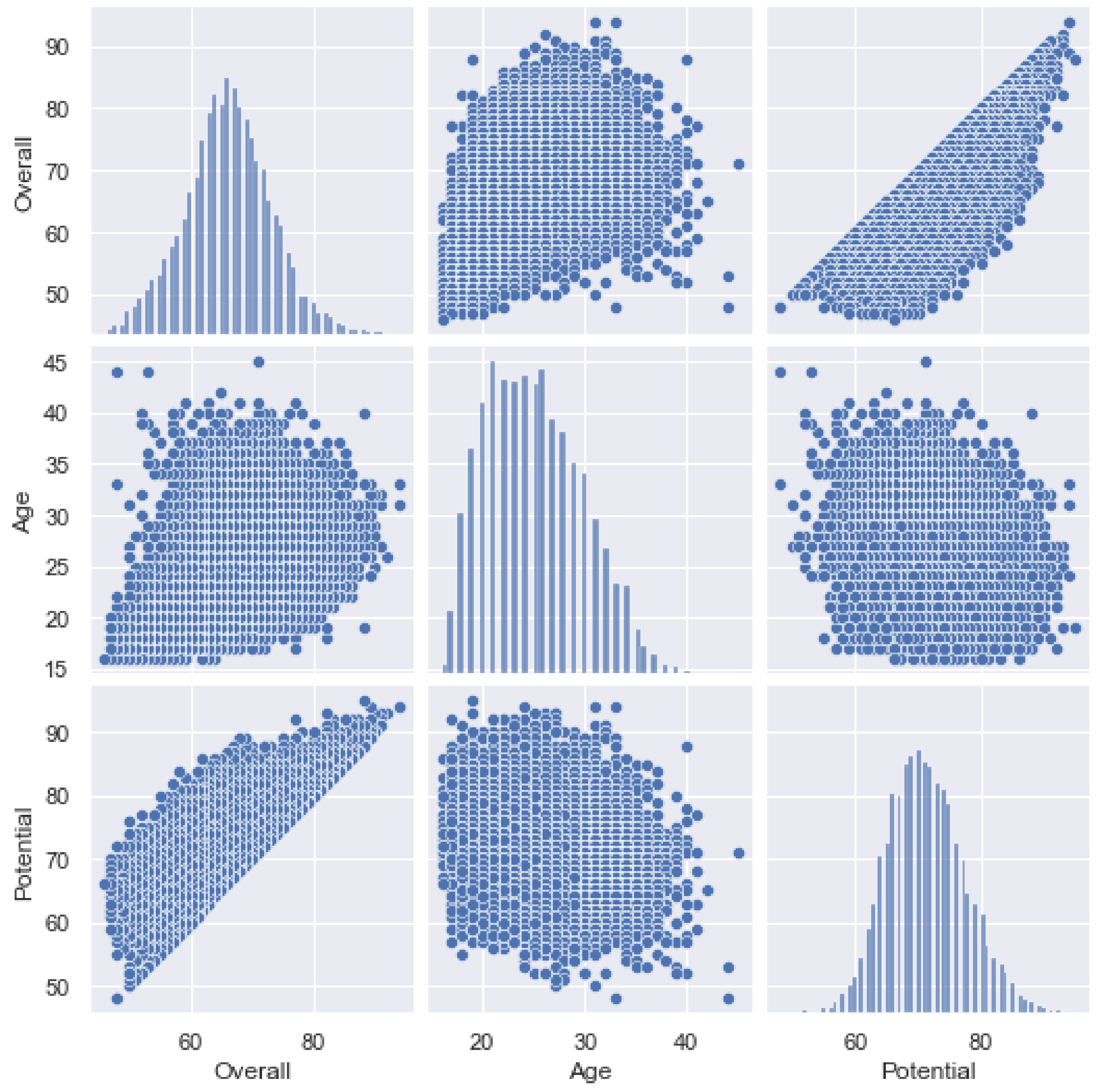
```
corr = df[['Overall', 'Age', 'Wage', 'Skill Moves']].corr()
sns.heatmap(corr, annot=True)
plt.title('Heatmap of Overall, Age, wage_euro, and Skill Moves')
plt.show()
```



Generating Pair Plots

The last Seaborn tool we'll discuss is the pairplot method. This allows you to generate a matrix of distributions and scatter plots for a set of numerical features. Let's do this for age, overall and potential:

```
data = df[['Overall', 'Age', 'Potential']]
sns.pairplot(data)
plt.show()
```



Your Turn

Now explore the data from all the datasets we have covered today (taxis, meteorites and fifa)

Visualize the data and tell us the insights you gain from it!
(you may choose whichever plot you are comfortable to play around with)

Thank You