03

Pandas, Matplotlib & Seaborn

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 Create

```
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

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

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

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

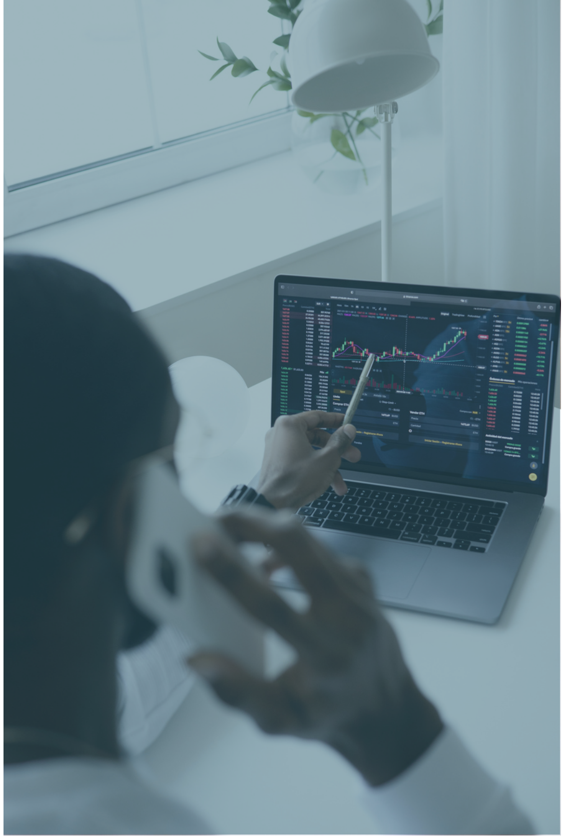
Drop column(s) that contain Null

```
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
```

foo	bar	baz
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
```

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

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

foo	bar	baz
a	6	True
b	10	True

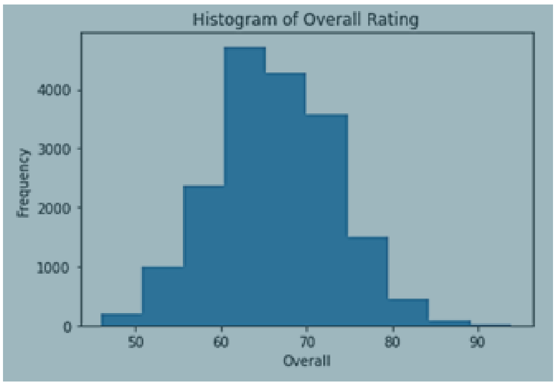
Data Visualization

matplotlib

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()
```

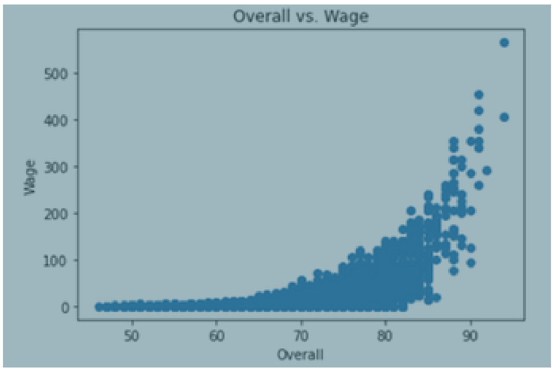
	ID	Name	Age	Nationality	Overall	Potential	Club	Value	Wage
0	158023	L. Messi	31	Argentina	94	94	FC Barcelona	110500.0	565.0
1	20801	Cristiano Ronaldo	33	Portugal	94	94	Juventus	77000.0	405.0
2	190871	Neymar Jr	26	Brazil	92	93	Paris Saint-Germain	118500.0	290.0
3	193080	De Gea	27	Spain	91	93	Manchester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

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



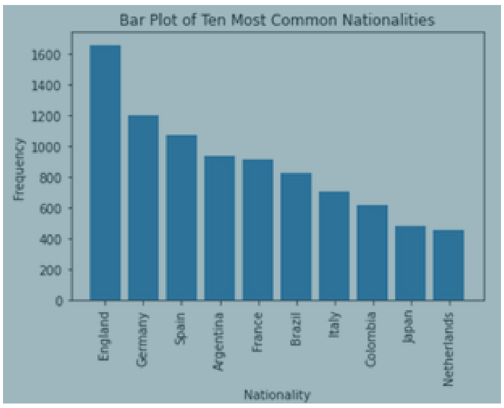
- Generating bar charts

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.

```python
# 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
```

```python
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.

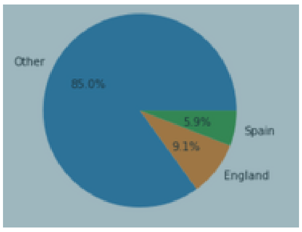
```
# add column named Nationality2
# assign value to each row satisfying the condition
# loc[rows where the condition is satisfied, column]
# here we create 4 categories of Nationality2

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()
```



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.

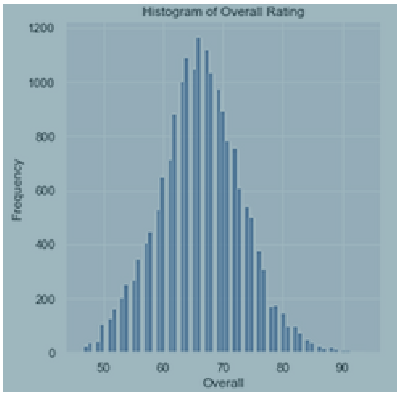
First, import Seaborn as sns

```
import seaborn as sns
```

- Generating histograms

We can also generate all ofthe same visualizations we did in Matplotlib using Seaborn. To regenerate our histogram of the overall column, we use the distplotmethod on the Seaborn object:

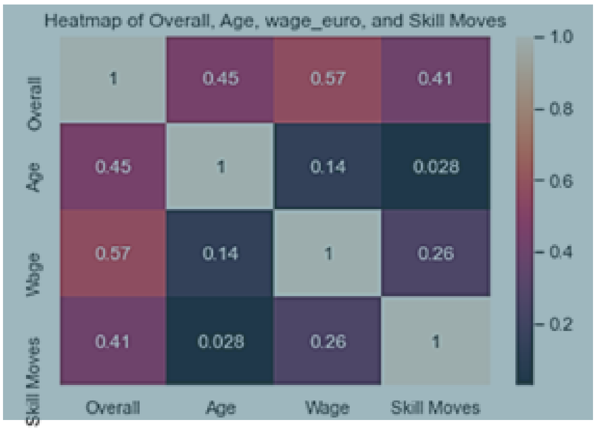
```
sns.displot(df['Overall'])
plt.xlabel('Overall')
plt.ylabel('Frequency')
plt.title('Histogram of Overall Rating')
plt.show()
```



- 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_euroand skill moves.
These correlation values can help us selecting features later onwhen 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()
```



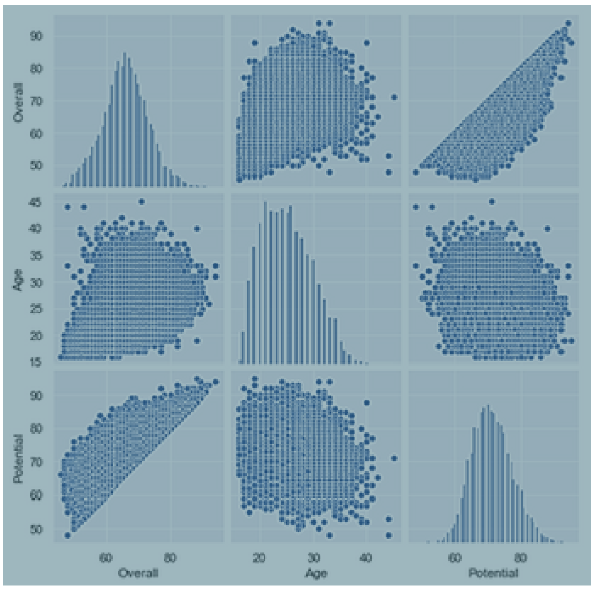
- Generating scatterplots

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

- Generating pair plots

The last Seaborn tool we'll discuss is the pairplotmethod. 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()
```



Picture Source

- https://upload.wikimedia.org/wikipedia/en/thumb/5/56/Matplotlib_logo.svg/2560px-Matplotlib_logo.svg.png
- https://seaborn.pydata.org/_static/logo-wide-lightbg.svg
- pexels.com