[***Download the dataset from here***]()

Import the necessary packages and the dataset:

```
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', 100)
```

1. pd.read_csv, pd.read_excel

These are used to read a CSV or an excel file to a pandas DataFrame format.

Using the read_csv function to read the ***FIFA dataset***:

```
df = pd.read_csv("fifa.csv")
```

If you have an excel file instead of a csv file you will use pd.read_excel.

By default, *.head()* the first 5 rows of the DataFrame.

```
df.head()
```

	Unnamed: 0	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club_name	league_n
0	0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	27	1987-06-24	169	67	Argentina	FC Barcelona	Spain Pri Div
1	1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	29	1985-02-05	185	80	Portugal	Real Madrid	Spain Pri Div
2	2	9014	https://sofifa.com/player/9014/arjen-robben/15...	A. Robben	Arjen Robben	30	1984-01-23	180	80	Netherlands	FC Bayern München	Germ Bunde
3	3	41236	https://sofifa.com/player/41236/zlatan-ibrahim...	Z. Ibrahimović	Zlatan Ibrahimović	32	1981-10-03	195	95	Sweden	Paris Saint-Germain	French L
4	4	167495	https://sofifa.com/player/167495/manuel-neuer/...	M. Neuer	Manuel Neuer	28	1986-03-27	193	92	Germany	FC Bayern München	Germ Bunde

To show 7 rows:

```
df.head(7)
```

2. df.columns

Print out all the columns of the dataset:

```
df.columns
```

Output:

```
Index(['Unnamed: 0', 'sofifa_id', 'player_url', 'short_name', 'long_name', 'age', 'dob', 'height_cm', 'weight_kg',
'nationality', 'club_name', 'league_name', 'league_rank', 'overall', 'potential', 'value_eur', 'wage_eur',
'player_positions', 'preferred_foot', 'international_reputation', 'weak_foot', 'skill_moves', 'work_rate',
'body_type', 'real_face', 'release_clause_eur', 'player_tags', 'team_position', 'team_jersey_number',
'loaned_from', 'joined', 'contract_valid_until', 'nation_position', 'nation_jersey_number', 'pace', 'shooting',
'passing', 'dribbling', 'defending', 'physic', 'gk_diving', 'gk_handling', 'gk_kicking', 'gk_reflexes', 'gk_speed',
'gk_positioning', 'player_traits', 'attacking_crossing', 'attacking_finishing', 'attacking_heading_accuracy',
'attacking_short_passing', 'attacking_volleys', 'skill_dribbling', 'skill_curve', 'skill_fk_accuracy',
'skill_long_passing', 'skill_ball_control', 'movement_acceleration', 'movement_sprint_speed',
'movement_agility', 'movement_reactions', 'movement_balance', 'power_shot_power', 'power_jumping',
'power_stamina', 'power_strength', 'power_long_shots', 'mentality_aggression', 'mentality_interceptions',
'mentality_positioning', 'mentality_vision', 'mentality_penalties',
'mentality_composure', 'defending_marking', 'defending_standing_tackle', 'defending_sliding_tackle',
'goalkeeping_diving', 'goalkeeping_handling', 'goalkeeping_kicking', 'goalkeeping_positioning',
'goalkeeping_reflexes'], dtype='object')
```

3. df.drop()

Drop some unnecessary columns using df.drop().

```
df = df.drop(columns=['Unnamed: 0', 'weak_foot', 'real_face'])
```

dropped these three columns: ‘Unnamed: 0’, ‘weak_foot’, ‘real_face’.

4. .len()

Provides with the length of the DataFrame.

```
len(df)
```

Output:

```
16155
```

This DataFrame has 16155 rows of data.

5. df.query()

You can filter or query using a boolean expression.

For ex. checking for which rows ‘shooting’ is bigger than ‘passing’.

```
df.query("shooting > passing")
```

This will return the rows only where the shooting is bigger than passing.

6. df.iloc()

This function takes as a parameter the rows and column indices and gives you the subset of the DataFrame accordingly.

Here I am taking the first 10 rows of data and index 5th to index 10th columns:

```
df.iloc[:10, 5:10]
```

	league_name	league_rank	overall	potential	value_eur
0	Spain Primera Division	1.0	93	95	100500000
1	Spain Primera Division	1.0	92	92	79000000
2	German 1. Bundesliga	1.0	90	90	54500000
3	French Ligue 1	1.0	90	90	52500000
4	German 1. Bundesliga	1.0	90	90	63500000
5	Spain Primera Division	1.0	89	91	49500000
6	Spain Primera Division	1.0	89	89	36000000
7	English Premier League	1.0	88	90	40500000
8	English Premier League	1.0	88	88	40500000
9	German 1. Bundesliga	1.0	88	88	39000000

7. df.loc()

This function does almost the similar operation as .iloc() function.

But here we can specify exactly which row index we want and also the name of the columns we want in our subset. Here is an example:

```
df.loc[[3, 10, 14, 23], ['nationality', 'weight_kg', "height_cm"]]
```

	nationality	weight_kg	height_cm
3	Sweden	95	195
10	France	72	170
14	Germany	66	170
23	Argentina	70	180

8. df[‘’].dtypes

Know data types of the variables.

```
df.height_cm.dtypes
```

Output: dtype('int64')

Get the data type of each and every column as well using this syntax:

```
df.dtypes
```

Output:

```
height_cm                 int64
weight_kg                 int64
nationality              object
random_col                int32
club_name                object
league_name              object
league_rank             float64
overall                   int64
potential                 int64
value_eur                 int64
wage_eur                  int64
player_positions         object
preferred_foot           object
international_reputation      int64
skill_moves               int64
work_rate                object
body_type                object
team_position            object
team_jersey_number      float64
nation_position          object
nation_jersey_number    float64
pace                    float64
shooting                float64
passing                 float64
dribbling               float64
defending               float64
physic                  float64
cumsum_2                  int64
rank_calc               float64
dtype: object
```

9. df.select_dtypes()

You can select the variables or columns of a certain data type using this function.

For example, I want to select the columns with data types 'int64' only. Here is how to do that:

```
df.select_dtypes(include='int64')
```

	height_cm	weight_kg	overall	potential	value_eur	wage_eur	international_reputation	skill_moves	cumsum_2
0	169	67	93	95	100500000	550000	5	4	100500000
1	185	80	92	92	79000000	375000	5	5	79000000
2	180	80	90	90	54500000	275000	5	4	54500000
3	195	95	90	90	52500000	275000	5	4	52500000
4	193	92	90	90	63500000	300000	5	1	63500000
...	...	...	...	...	...	...	...	...	...
16150	187	81	41	61	20000	2000	1	2	113310000
16151	178	57	41	50	30000	2000	1	3	113340000
16152	190	76	40	50	15000	2000	1	2	236897000
16153	180	70	40	49	15000	2000	1	2	954536000
16154	175	72	40	40	0	2000	1	2	945000

16155 rows × 9 columns

We got all the columns that have the data type 'int64'.

If we use ‘exclude’, we will get the columns that do not have the data type ‘int64’:

```
df.select_dtypes(exclude='int64')
```

	nationality	random_col	club_name	league_name	league_rank	player_positions	preferred_foot	work_rate	body_type	team_position	team_j
0	Argentina	32	FC Barcelona	Spain Primera Division	1.0	CF	Left	Medium/Low	Normal	CF	
1	Portugal	35	Real Madrid	Spain Primera Division	1.0	LW, LM	Right	High/Low	Normal	LW	
2	Netherlands	56	FC Bayern München	German 1. Bundesliga	1.0	RM, LM, RW	Left	High/Low	Normal	SUB	
3	Sweden	16	Paris Saint-Germain	French Ligue 1	1.0	ST	Right	Medium/Low	Normal	ST	
4	Germany	37	FC Bayern München	German 1. Bundesliga	1.0	GK	Right	Medium/Medium	Normal	GK	
...	...	...	...	...	...	...	...	...	...	...	
16150	Wales	65	Newport County	English League Two	4.0	CB	Right	Medium/Medium	Normal	RES	
16151	Wales	13	Newport County	English League Two	4.0	ST	Right	Medium/Medium	Lean	RES	
16152	Poland	6	Wisła Kraków	Polish T-Mobile Ekstraklasa	1.0	LM, LB	Left	Medium/Medium	Normal	RES	
16153	England	57	Fleetwood Town	English League One	3.0	CB	Right	Medium/Medium	Normal	RES	
16154	Malta	14	Exeter City	English League Two	4.0	CM, CAM	Right	Medium/Medium	Lean	RES	

16155 rows × 20 columns

10. df.insert()

Inserts a column in the specified position.

To demonstrate that I will first create an array of random numbers that have the length of our DataFrame:

```
random_col = np.random.randint(100, size=len(df))
```

I will insert this array as a column in the DataFrame df at column 3 position. Remember, the column index starts from zero.

```
df.insert(3, 'random_col', random_col)
```

Here is the part of the DataFrame again: `df.head()`

	sofifa_id	player_url	short_name	random_col	long_name	age	dob	height_cm	weight_kg	nationality	club_name	league_
0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	1	Lionel Andrés Messi Cuccittini	27	1987-06-24	169	67	Argentina	FC Barcelona	Spain P D
1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	48	Cristiano Ronaldo dos Santos Aveiro	29	1985-02-05	185	80	Portugal	Real Madrid	Spain P D
2	9014	https://sofifa.com/player/9014/arjen-robben/15...	A. Robben	46	Arjen Robben	30	1984-01-23	180	80	Netherlands	FC Bayern München	Gen Bun
3	41236	https://sofifa.com/player/41236/zlatan-ibrahim...	Z. Ibrahimović	53	Zlatan Ibrahimović	32	1981-10-03	195	95	Sweden	Paris Saint-Germain	French
4	167495	https://sofifa.com/player/167495/manuel-neuer/...	M. Neuer	83	Manuel Neuer	28	1986-03-27	193	92	Germany	FC Bayern München	Gen Bun

Look, the column ‘random_col’ is inserted at position three.

11. df['’].cumsum()

It provides you with the cumulative sum.

Use the ‘value_eur’ and ‘wage_eur’columns for this example.

```
df[['value_eur', 'wage_eur']].cumsum()
```

Output:

	value_eur	wage_eur
0	100500000	550000
1	179500000	925000
2	234000000	1200000
3	286500000	1475000
4	350000000	1775000
...	...	...
16150	17138496000	210919000
16151	17138526000	210921000
16152	17138541000	210923000
16153	17138556000	210925000
16154	17138556000	210927000

16155 rows × 2 columns

As you can see in every row it provides you with the cumulative sum of all the values of the previous rows.

12. df.sample()

When the size of the dataset is too big, you can take a representative sample from it to perform the analysis and predictive modeling. That may save you some time. Also, too much data may ruin the visualization sometimes. we can use this function to get a certain number of data points or a certain fraction or data point. Here I am taking a sample of 200 data points from the FIFA dataset. It takes a random sample.

```
df.sample(n = 200)
```

I am taking 25% of the FIFA dataset here:

```
df.sample(frac = 0.25)
```

13. df['’].where()

This function helps you query a dataset based on a boolean condition.

For an example, the *random_col* we made before has the values ranging from 0 to 100. Here is how we make a series to see which of them are bigger than 50.

```
df['random_col'].where(df['random_col'] > 50)
```

Output:

```
0         NaN
1         NaN
2        56.0
3         NaN
4         NaN
         ...
16150    65.0
16151     NaN
16152     NaN
16153    57.0
16154     NaN
Name: random_col, Length: 16155, dtype: float64
```

Look, where the values do not meet the condition that means the value is not greater than 50, returns NaN.

We can replace NaN with 0 or any other value using this syntax:

```
df['random_col'].where(df['random_col'] > 50, 0)
```

Output:

```
0         0
1         0
2        56
3         0
4         0
         ..
16150    65
16151     0
16152     0
16153    57
16154     0
Name: random_col, Length: 16155, dtype: int32
```

14. df[‘’].unique()

This is very useful where we have categorical variables.

It is used to find out the unique values of a categorical column.

Let’s see what are the unique values of the ‘skill_moves’ column in our FIFA dataset:

```
df.skill_moves.unique()
```

Output:

```
array([4, 5, 1, 3, 2], dtype=int64)
```

So, we have five unique values in the skill_moves columns.

If we print out the head of the dataset to check out the values of the columns you may not see all the unique values in it. So, to know all the unique values .unique() function comes out really handy.

15. df[‘’].nunique()

Lets you know how many unique values do you have in a column.

As an example, if you want to see how many different nationalities are there in this dataset, you can use this simple line of code

```
df.nationality.nunique()
```

Output:

```
149
```

The great thing is, this function can be used on the total dataset as well to know the number of unique values in each column:

```
df.nunique()
```

Output:

```
height_cm                    48
weight_kg                    54
nationality                 149
random_col                  100
club_name                   577
league_name                  37
league_rank                   4
overall                      53
potential                    49
value_eur                   161
wage_eur                     41
player_positions            907
preferred_foot                2
international_reputation      5
skill_moves                   5
work_rate                     9
body_type                     3
team_position                29
team_jersey_number           99
nation_position              28
nation_jersey_number         26
pace                         74
shooting                     70
passing                      67
dribbling                    67
defending                    69
physic                       63
cumsum_2                  14859
rank_calc                   161
dtype: int64
```

16. df['’].rank()

Provides you with the rank based on a certain column.

In the FIFA dataset, if we want to rank the players based on the ‘value_eur’ column,

```
df['rank_calc'] = df["value_eur"].rank()
```

Using the line of code above, I created a new column named ‘rank_calc’.

This new column will give you the ranks of each player based on the ‘value_eur’. The column will be added at the end by default. Please run the line of code by yourself to check.

17. .isin()

I am going to make a subset of the dataset that will contain only a few nationalities of players using .isin() function.

```
nationality = ["Argentina", "Portugal", "Sweden", "England"]
df[df.nationality.isin(nationality)]
```

Resulting dataset containing only those few countries mentioned in the list above,

	height_cm	weight_kg	nationality	random_col	club_name	league_name	league_rank	overall	potential	value_eur	wage_eur	player_positions	pr
0	169	67	Argentina	32	FC Barcelona	Spain Primera Division	1.0	93	95	100500000	550000	CF	
1	185	80	Portugal	35	Real Madrid	Spain Primera Division	1.0	92	92	79000000	375000	LW, LM	
3	195	95	Sweden	16	Paris Saint-Germain	French Ligue 1	1.0	90	90	52500000	275000	ST	
23	180	70	Argentina	24	Manchester United	English Premier League	1.0	86	88	45500000	230000	CAM, CM, RM	
26	172	74	Argentina	31	Manchester City	English Premier League	1.0	86	87	45500000	230000	ST	
...	...	...	...	...	...	...	...	...	...	...	...	...	
16143	183	70	England	1	Tranmere Rovers	English League Two	4.0	43	56	25000	2000	LM, ST	
16145	185	77	England	28	Wycombe Wanderers	English League Two	4.0	43	43	6000	2000	GK	
16146	179	79	England	88	Burton Albion	English League Two	4.0	42	56	25000	2000	CM, CDM, RM	
16149	196	80	England	45	Tranmere Rovers	English League Two	4.0	42	52	20000	2000	GK	
16153	180	70	England	57	Fleetwood Town	English League One	3.0	40	49	15000	2000	CB	

18. df.replace()

It replaces the values of a column.

When we need to replace only one unique value of a column we simply need to pass the old value and the new value.

Imagine, we just found out that the 'league_rank' 1.0 needs to be replaced by 1.1 now. To do that:

```
df.replace(1.0, 1.1)
```

	height_cm	weight_kg	nationality	random_col	club_name	league_name	league_rank	overall	potential	value_eur	wage_eur	player_positions	p
0	169	67	Argentina	7.0	FC Barcelona	Spain Primera Division	1.1	93	95	100500000	550000	CF	
1	185	80	Portugal	76.0	Real Madrid	Spain Primera Division	1.1	92	92	79000000	375000	LW, LM	
2	180	80	Netherlands	98.0	FC Bayern München	German 1. Bundesliga	1.1	90	90	54500000	275000	RM, LM, RW	
3	195	95	Sweden	41.0	Paris Saint-Germain	French Ligue 1	1.1	90	90	52500000	275000	ST	
4	193	92	Germany	44.0	FC Bayern München	German 1. Bundesliga	1.1	90	90	63500000	300000	GK	
...	...	...	...	...	...	...	...	...	...	...	...	...	
16150	187	81	Wales	55.0	Newport County	English League Two	4.0	41	61	20000	2000	CB	
16151	178	57	Wales	32.0	Newport County	English League Two	4.0	41	50	30000	2000	ST	
16152	190	76	Poland	81.0	Wisła Kraków	Polish T-Mobile Ekstraklasa	1.1	40	50	15000	2000	LM, LB	
16153	180	70	England	23.0	Fleetwood Town	English League One	3.0	40	49	15000	2000	CB	
16154	175	72	Malta	65.0	Exeter City	English League Two	4.0	40	40	0	2000	CM, CAM	

16155 rows × 29 columns

Look at the league_rank column in the dataset now, 1.0 is replaced by 1.1.

If we need to change more than one value, we can pass a dictionary to the replace function where the key should be the original value and the value should be the replacement.

```
df.replace({1.0: 1.1,  4.0: 4.1, 3.0: 3.1})
```

	height_cm	weight_kg	nationality	random_col	club_name	league_name	league_rank	overall	potential	value_eur	wage_eur	player_positions
0	169.0	67.0	Argentina	7.0	FC Barcelona	Spain Primera Division	1.1	93.0	95.0	100500000.0	550000.0	CF
1	185.0	80.0	Portugal	76.0	Real Madrid	Spain Primera Division	1.1	92.0	92.0	79000000.0	375000.0	LW, LM
2	180.0	80.0	Netherlands	98.0	FC Bayern München	German 1. Bundesliga	1.1	90.0	90.0	54500000.0	275000.0	RM, LM, RW
3	195.0	95.0	Sweden	41.0	Paris Saint-Germain	French Ligue 1	1.1	90.0	90.0	52500000.0	275000.0	ST
4	193.0	92.0	Germany	44.0	FC Bayern München	German 1. Bundesliga	1.1	90.0	90.0	63500000.0	300000.0	GK
...	...	...	...	...	...	...	...	...	...	...	...	...
16150	187.0	81.0	Wales	55.0	Newport County	English League Two	4.1	41.0	61.0	20000.0	2000.0	CB
16151	178.0	57.0	Wales	32.0	Newport County	English League Two	4.1	41.0	50.0	30000.0	2000.0	ST
16152	190.0	76.0	Poland	81.0	Wisła Kraków	Polish T-Mobile Ekstraklasa	1.1	40.0	50.0	15000.0	2000.0	LM, LB
16153	180.0	70.0	England	23.0	Fleetwood Town	English League One	3.1	40.0	49.0	15000.0	2000.0	CB
16154	175.0	72.0	Malta	65.0	Exeter City	English League Two	4.1	40.0	40.0	0.0	2000.0	CM, CAM

19. df.rename()

It is used to rename the column/s. Here I am changing the ‘weight_kg’ and ‘height_cm’ columns to “Weight (kg)” and “Height (cm)”:

```
df.rename(columns = {"weight_kg": "Weight (kg)", "height_cm": "Height (cm)"})
```

	Height (cm)	Weight (kg)	nationality	random_col	club_name	league_name	league_rank	overall	potential	value_eur	wage_eur	player_positions	preferre
0	169	67	Argentina	7	FC Barcelona	Spain Primera Division	1.0	93	95	100500000	550000	CF	
1	185	80	Portugal	76	Real Madrid	Spain Primera Division	1.0	92	92	79000000	375000	LW, LM	
2	180	80	Netherlands	98	FC Bayern München	German 1. Bundesliga	1.0	90	90	54500000	275000	RM, LM, RW	
3	195	95	Sweden	41	Paris Saint-Germain	French Ligue 1	1.0	90	90	52500000	275000	ST	
4	193	92	Germany	44	FC Bayern München	German 1. Bundesliga	1.0	90	90	63500000	300000	GK	
...	...	...	...	...	...	...	...	...	...	...	...	...	
16150	187	81	Wales	55	Newport County	English League Two	4.0	41	61	20000	2000	CB	
16151	178	57	Wales	32	Newport County	English League Two	4.0	41	50	30000	2000	ST	
16152	190	76	Poland	81	Wisła Kraków	Polish T-Mobile Ekstraklasa	1.0	40	50	15000	2000	LM, LB	
16153	180	70	England	23	Fleetwood Town	English League One	3.0	40	49	15000	2000	CB	
16154	175	72	Malta	65	Exeter City	English League Two	4.0	40	40	0	2000	CM, CAM	

20. .fillna()

Replaces the null values with some other value of your choice.

Here are some of the columns towards the end of the FIFA dataset:

am_jersey_number	loaned_from	joined	contract_valid_until	nation_position	nation_jersey_number	pace	shooting	passing	dribbling	defending	physic
10.0	NaN	2004-07-01	2018.0	CF	10.0	93.0	89.0	86.0	96.0	27.0	63.0
7.0	NaN	2009-07-01	2018.0	LW	7.0	93.0	93.0	81.0	91.0	32.0	79.0
10.0	NaN	2009-08-28	2017.0	RS	11.0	93.0	86.0	83.0	92.0	32.0	64.0
10.0	NaN	2012-07-01	2016.0	ST	10.0	76.0	91.0	81.0	86.0	34.0	86.0
1.0	NaN	2011-07-01	2019.0	GK	1.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
35.0	NaN	2013-10-22	2015.0	NaN	NaN	66.0	25.0	29.0	30.0	39.0	53.0
39.0	NaN	2014-08-14	2015.0	NaN	NaN	64.0	41.0	27.0	35.0	27.0	41.0
43.0	NaN	2012-07-01	2015.0	NaN	NaN	58.0	27.0	35.0	31.0	52.0	68.0
26.0	NaN	2014-03-11	2021.0	NaN	NaN	72.0	27.0	27.0	30.0	35.0	61.0
17.0	NaN	2006-06-26	2015.0	NaN	NaN	38.0	36.0	42.0	45.0	31.0	33.0

Replace those null values with some values of compatible data types

For example in the 'pace' column, the values should be numeric but here and there you will see NaN values.

The most generic but not so efficient way is to replace those NaN values with zeros.

```
df['pace'].fillna(0, inplace=True)
```

›e	team_position	team_jersey_number	nation_position	nation_jersey_number	pace	shooting	passing	dribbling	defending	physic	cumsum_2	rank_calc
al	CF	10.0	CF	10.0	93.0	89.0	86.0	96.0	27.0	63.0	100500000	16155.0
al	LW	7.0	LW	7.0	93.0	93.0	81.0	91.0	32.0	79.0	79000000	16154.0
al	SUB	10.0	RS	11.0	93.0	86.0	83.0	92.0	32.0	64.0	54500000	16152.0
al	ST	10.0	ST	10.0	76.0	91.0	81.0	86.0	34.0	86.0	52500000	16151.0
al	GK	1.0	GK	1.0	0.0	NaN	NaN	NaN	NaN	NaN	63500000	16153.0
...	...	...	...	...	...	...	...	...	...	...	...	...
al	RES	35.0	NaN	NaN	66.0	25.0	29.0	30.0	39.0	53.0	113310000	418.0
an	RES	39.0	NaN	NaN	64.0	41.0	27.0	35.0	27.0	41.0	113340000	749.5
al	RES	43.0	NaN	NaN	58.0	27.0	35.0	31.0	52.0	68.0	236897000	337.5
al	RES	26.0	NaN	NaN	72.0	27.0	27.0	30.0	35.0	61.0	954536000	337.5
an	RES	17.0	NaN	NaN	38.0	36.0	42.0	45.0	31.0	33.0	945000	156.0

If you notice, the NaN in the pace column is zero now.

You can replace it with some other value of your choice.

It is also common to replace values with the mean or median.

To replace the NaN values of the pace column with the mean of space column:

```
df['pace'].fillna(df['pace'].mean(), inplace = True)
```

21. df.groupby()

This is the most popular function for data summarizing.

You can group the data as per a certain variable and find out useful information about those groups.

For example, here I am grouping the data by nationality and calculating the total 'value_eur' for each nationality:

```
df.groupby("nationality")['value_eur'].sum()
```

Output:

```
nationality
Albania                25860000
Algeria                70560000
Angola                  6070000
Antigua & Barbuda       1450000
Argentina            1281372000
                        ...
Uzbekistan              7495000
Venezuela              41495000
Wales                 113340000
Zambia                  4375000
Zimbabwe                6000000
Name: value_eur, Length: 149, dtype: int64
```
The sum of 'value_eur' for all the players of Albania is 25860000.

It is also possible to group by several variables and use several aggregate functions.

We will see for each nationality and each league rank's mean value_eur, median value_eur, mean wage_eur, and median wage_eur.

```
df.groupby(['nationality', 'league_rank'])['value_eur', 'wage_eur'].agg([np.mean, np.median])
```
Output:

		value_eur		wage_eur	
		mean	median	mean	median
nationality	league_rank				
Albania	1.0	8.584615e+05	475000	13230.769231	7000
	2.0	5.057143e+05	500000	5857.142857	7000
Algeria	1.0	2.130536e+06	1300000	25107.142857	20000
	2.0	8.388462e+05	450000	12307.692308	8000
Angola	1.0	1.149000e+06	575000	15600.000000	9000
...	...	...	...	...	...
Zambia	1.0	9.583333e+05	925000	11000.000000	9000
	2.0	1.500000e+06	1500000	20000.000000	20000
Zimbabwe	1.0	6.183333e+05	500000	8777.777778	9000
	3.0	2.025000e+05	202500	3500.000000	3500
	4.0	3.000000e+04	30000	2000.000000	2000

318 rows × 4 columns

22. .pct_change()

You can get the percent change from the previous value of a variable.

For this demonstration, I will use the value_eur column and get the percent change from the previous for each row of data. The first row will be NaN because there is no value to compare before.

```
df.value_eur.pct_change()
```
Output

```
0          NaN
1    -0.213930
```

```
2       -0.310127
3       -0.036697
4        0.209524
           ...
16150    0.000000
16151    0.500000
16152   -0.500000
16153    0.000000
16154   -1.000000
Name: value_eur, Length: 16155, dtype: float64
```

23. df.count()

It provides you the number of data in the DataFrame in the specified direction. When the direction is 0, it provides the number of data in the columns:

```
df.count(0)
```

Output:

```
Unnamed: 0                  16155
sofifa_id                   16155
player_url                  16155
short_name                  16155
long_name                   16155
                            ...
goalkeeping_diving          16155
goalkeeping_handling        16155
goalkeeping_kicking         16155
goalkeeping_positioning     16155
goalkeeping_reflexes        16155
Length: 81, dtype: int64
```

You can see the number of data in each column.

When the direction is 1, it provides the number of data in the rows:

```
df.count(1)
```

Output:

```
0        72
1        72
2        72
3        72
4        71
         ..
16150    68
16151    68
16152    68
16153    68
16154    69
Length: 16155, dtype: int64
```

As you can see, each row does not have the same number of data. If you observe the dataset carefully, you will see that it has a lot of null values in several columns.

24. df[‘’].value_counts()

We can get the value counts of each category using this function.

Here I am getting how many values are there in each league_rank.

```
df['league_rank'].value_counts()
```

Output:

```
1.0    11738
2.0     2936
3.0      639
4.0      603
Name: league_rank, dtype: int64
```

It returns the result sorted by default. If you want the result in ascending order, simply set ascending=True:

```
df['league_rank'].value_counts(ascending=True)
```

Output:

```
4.0      603
3.0      639
2.0     2936
1.0    11738
Name: league_rank, dtype: int64
```

25. pd.crosstab()

It gives you a frequency table that is a cross-tabulation of two variables. I am making a cross-tabulation of league_rank and international_reputation here:

```
pd.crosstab(df['league_rank'], df['international_reputation'])
```

international_reputation	1	2	3	4	5
league_rank					
1.0	10305	1160	227	37	9
2.0	2772	163	1	0	0
3.0	637	2	0	0	0
4.0	601	2	0	0	0

So, we got the number count of all the combinations of league_rank and international_reputation. We can see that the majority of players have international_reputation and league_rank both 1.

It can be improved further. We can add margins in both directions that will be the total and also we can get the normalized values if necessary:

```
pd.crosstab(df['league_rank'], df['international_reputation'],
        margins = True,
        margins_name="Total",
        normalize = True)
```

international_reputation league_rank	1	2	3	4	5	Total
1.0	0.647462	0.072883	0.014262	0.002325	0.000565	0.737497
2.0	0.174164	0.010241	0.000063	0.000000	0.000000	0.184468
3.0	0.040023	0.000126	0.000000	0.000000	0.000000	0.040148
4.0	0.037761	0.000126	0.000000	0.000000	0.000000	0.037886
Total	0.899409	0.083375	0.014325	0.002325	0.000565	1.000000

26. pd.qcut()

This function bins the data or segments the data based on the distribution of the data.

So, we get the range for each player. Here I am going to segment the value_eur in 5 portions and get which player falls in which portion:

```
pd.qcut(df['value_eur'], q = 5)
```

Output:

```
0       (1100000.0, 100500000.0]
1       (1100000.0, 100500000.0]
2       (1100000.0, 100500000.0]
3       (1100000.0, 100500000.0]
4       (1100000.0, 100500000.0]
                  ...
16150         (-0.001, 100000.0]
16151         (-0.001, 100000.0]
16152         (-0.001, 100000.0]
16153         (-0.001, 100000.0]
16154         (-0.001, 100000.0]
Name: value_eur, Length: 16155, dtype: category
Categories (5, interval[float64]): [(-0.001, 100000.0] < (100000.0, 230000.0] < (230000.0, 500000.0] <
(500000.0, 1100000.0] < (1100000.0, 100500000.0]]
```

You can use the value_counts on the above line of code to see how players fall in which range:

```
pd.qcut(df['value_eur'], q = 5).value_counts()
```

Output:

```
(-0.001, 100000.0]          3462
(230000.0, 500000.0]        3305
(100000.0, 230000.0]        3184
(500000.0, 1100000.0]       3154
(1100000.0, 100500000.0]    3050
Name: value_eur, dtype: int64
```

As you can see the numbers are pretty close. By default, qcut tries to divide them equally. But in real life, it doesn't want to be equal always. Because the distribution is not uniform most of the time.

27. pd.cut()

Another method for binning. If we want to make 5 bins using cut, it will divide the entire value_eur range into equal five portions and the population in each bin will follow accordingly.

```
pd.cut(df['value_eur'], bins = 5).value_counts()
```

Output:

```
(-100500.0, 20100000.0]      16102
(20100000.0, 40200000.0]        40
(40200000.0, 60300000.0]        10
(60300000.0, 80400000.0]         2
(80400000.0, 100500000.0]        1
Name: value_eur, dtype: int64
```

The interval in each range is equal. But the population in each group is very different.

28. df[‘’].describe()

This is a great function that provides some basic statistical measures. Here I am using the describe function on the wage_eur column:

```
df['wage_eur'].describe()
```

Output:

```
count     16155.000000
mean      13056.453110
std       23488.182571
min           0.000000
25%        2000.000000
50%        5000.000000
75%       10000.000000
max      550000.000000
Name: wage_eur, dtype: float64
```

As the output shows, we have eight different measures. Each of them is very significant.

29. nlargest and nsmallest

This gives you the dataset with n number of largest values or smallest values of a specified variable. As an example, I wanted to get the rows with the top 5 wage_eur:

```
df.nlargest(5, "wage_eur")
```

	Unnamed: 0	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club_name	league_n
0	0	158023	https://sofifa.com/player/158023/lionel-messi/...	L. Messi	Lionel Andrés Messi Cuccittini	27	1987-06-24	169	67	Argentina	FC Barcelona	Spain Pri Div
1	1	20801	https://sofifa.com/player/20801/c-ronaldo-dos-...	Cristiano Ronaldo	Cristiano Ronaldo dos Santos Aveiro	29	1985-02-05	185	80	Portugal	Real Madrid	Spain Pri Div
4	4	167495	https://sofifa.com/player/167495/manuel-neuer/...	M. Neuer	Manuel Neuer	28	1986-03-27	193	92	Germany	FC Bayern München	Germ Bunde
5	5	176580	https://sofifa.com/player/176580/luis-suarez/1...	L. Suárez	Luis Alberto Suárez Díaz	27	1987-01-24	181	81	Uruguay	FC Barcelona	Spain Pri Div
2	2	9014	https://sofifa.com/player/9014/arjen-robben/15...	A. Robben	Arjen Robben	30	1984-01-23	180	80	Netherlands	FC Bayern München	Germ Bunde

In the same way, I can make a subset of the dataset with the 5 smallest wage_eur data:

```
df.nsmallest(5, "wage_eur")
```

	Unnamed: 0	sofifa_id	player_url	short_name	long_name	age	dob	height_cm	weight_kg	nationality	club_name	leagu
151	151	209119	https://sofifa.com/player/209119/francisco-amo...	F. Amorebieta	Francisco Amorebieta	28	1985-10-27	194	83	Venezuela	NaN	
283	283	178007	https://sofifa.com/player/178007/miguel-luis-p...	Miguel Veloso	Miguel Luís Pinto Veloso	28	1986-05-11	180	78	Portugal	NaN	
289	289	209097	https://sofifa.com/player/209097/omar-luis-car...	O. Cardosa	Omar Luis Cardosa	30	1983-11-26	188	84	Paraguay	NaN	
424	424	209102	https://sofifa.com/player/209102/marco-aurelio...	M. Etxeberría	Marco Aurelio Etxeberría	27	1987-04-11	183	77	Paraguay	NaN	
456	456	178416	https://sofifa.com/player/178416/jeremain-lens...	J. Lens	Jeremain Lens	26	1987-11-24	178	73	Netherlands	NaN	

30. df.explode()

Explode can be useful when you have a list of data in some rows. It is hard to analyze, visualize or perform some predictive modeling when you have integers in some columns and lists in some columns. Explode helps to break down those lists. For example, look at this DataFrame:

```
df1 = pd.DataFrame({"city": ['A', 'B', 'C'],
                "day1": [22, 25, 21],
                'day2':[31, 12, 67],
                'day3': [27, 20, 15],
                'day4': [34, 37, [41, 45, 67, 90, 21]],
                'day5': [23, 54, 36]})
df1
```

	city	day1	day2	day3	day4	day5
0	A	22	31	27	34	23
1	B	25	12	20	37	54
2	C	21	67	15	[41, 45, 67, 90, 21]	36

Let’s explode column d4:

```
df1.explode(jupyter notebook
'day4').reset_index(drop=True)
```

	city	day1	day2	day3	day4	day5
0	A	22	31	27	34	23
1	B	25	12	20	37	54
2	C	21	67	15	41	36
3	C	21	67	15	45	36
4	C	21	67	15	67	36
5	C	21	67	15	90	36
6	C	21	67	15	21	36