Data Analysis

OUTLINE

- **Data Analysis**
- **Statistical Methods**
  - Statistical Functions
  - Statistical Summary
  - Central Tendency: Arithmetic Mean, Median, Mode
  - Distribution Representation: Range, Quantiles, Standard Deviation, Outliers
- **Visualization Methods**
  - Variable Visualization: Pie Chart, Bar Chart, Line Graph, Scatter Plot, Heatmap
  - Statistical Visualization: Histogram, Correlation and Causation, Statistical Category Variables, Grouping, ANOVA

Data Analysis

- **Data analysis** techniques use **statistical** and **visualization** methods.
- Data analysis is carried out by providing an **assessment** of the dataset that has been determined to obtain the added business value that can be achieved if AI and data science solutions are realized with this data.

Importing Data to Pandas

- Start Jupyter Notebook in your work folder.
- Open or create a new .ipynb script (Python 3)
- Import pandas and numpy library. (Make sure you've installed it before).
- Load the previously downloaded CSV file into a DataFrame with `read_csv(…)` command.

```
In [1]: import pandas as pd
        import numpy as np
```

```
In [2]: path = "epl-goalScorer(20-21).csv"
        df = pd.read_csv(path)
```

	Unnamed: 0	id	player_name	games	time	goals	xG	assists	xA
0	0	647	Harry Kane	35	3097	23	22.174859	14	7.577094
1	1	1250	Mohamed Salah	37	3085	22	20.250847	5	6.528526
2	2	1228	Bruno Fernandes	37	3117	18	16.019454	12	11.474996
3	3	453	Son Heung-Min	37	3139	17	11.023287	10	9.512992
4	4	822	Patrick Bamford	38	3085	17	18.401863	7	3.782247
...	...	...	...	...	...	...	...	...	...
517	517	9415	Jaden Philogene-Bidace	1	1	0	0.000000	0	0.000000
518	518	9423	Gaetano Berardi	2	113	0	0.074761	0	0.000000
519	519	9524	Anthony Elanga	1	67	0	0.000000	0	0.000000
520	520	9540	Femi Seriki	1	1	0	0.000000	0	0.000000
521	521	9552	Tyrese Francois	1	13	0	0.000000	0	0.000000

522 rows × 19 columns

Load Data into Pandas

The `head()` and `tail()` methods on the DataFrame display the first/last few rows of the data we load.

```
df.head(3)
```

	Unnamed: 0	id	player_name	games	time	goals	xG	assists
0	0	647	Harry Kane	35	3097	23	22.174859	14
1	1	1250	Mohamed Salah	37	3085	22	20.250847	5
2	2	1228	Bruno Fernandes	37	3117	18	16.019454	12

```
df.tail(7)
```

	Unnamed: 0	id	player_name	games	time	goals	xG	assists
515	515	9395	Sidnei Tavares	2	84	0	0.020798	0
516	516	9406	Nathan Broadhead	1	1	0	0.000000	0
517	517	9415	Jaden Philogene-Bidace	1	1	0	0.000000	0
518	518	9423	Gaetano Berardi	2	113	0	0.074761	0
519	519	9524	Anthony Elanga	1	67	0	0.000000	0
520	520	9540	Femi Seriki	1	1	0	0.000000	0
521	521	9552	Tyrese Francois	1	13	0	0.000000	0

Inspect the Data Type of Each Column

- The `dtypes` attribute on the DataFrame contains the data type of each column.
- See the Pandas User Guide for details on each type.
- `dtype:object` at the end of dtypes output represents `Series`, a Python object returned by dtypes itself (not part of any column type).

```
print(df.dtypes)
```

```
Unnamed: 0        int64
id                int64
player_name      object
games             int64
time              int64
goals             int64
xG              float64
assists           int64
xA              float64
shots             int64
key_passes        int64
yellow_cards      int64
red_cards         int64
position         object
team_title       object
npg               int64
npxG            float64
xGChain         float64
xGBuildup       float64
dtype: object
```

Select Data from DataFrame

- The first two columns are just numeric IDs which usually have no real meaning, e.g. Unnamed:0, id
- From the DataFrame `df`, it's enough to start from the *player_name* column (for zero-based indexes, we use the 2nd column and so on).

```
df_noid = df.iloc[:,2:]
df_noid
```

	player_name	games	time	goals	xG	assists	xA
0	Harry Kane	35	3097	23	22.174859	14	7.577094
1	Mohamed Salah	37	3085	22	20.250847	5	6.528526
2	Bruno Fernandes	37	3117	18	16.019454	12	11.474996
3	Son Heung-Min	37	3139	17	11.023287	10	9.512992
4	Patrick Bamford	38	3085	17	18.401863	7	3.782247
...	...	...	...	...	...	...	...
517	Jaden Philogene-Bidace	1	1	0	0.000000	0	0.000000
518	Gaetano Berardi	2	113	0	0.074761	0	0.000000
519	Anthony Elanga	1	67	0	0.000000	0	0.000000
520	Femi Seriki	1	1	0	0.000000	0	0.000000

Display Data in a Specific Order

- We can display data in a certain order.
- For example, this data is displayed sorted by *player_name* in ascending order and the first 10 rows of the results are displayed.

```
df1 = df_noid.sort_values(by="player_name",ascending=True)
df1.head(10)
```

	player_name	games	time	goals	xG	assists	xA	shots	key_passes	yellow_cards	red_cards	position	team_title	npg	npxG	x
164	Aaron Connolly	16	755	2	4.412484	1	0.140037	22	5	0	0	F M S	Brighton	2	4.412484	4.
281	Aaron Cresswell	35	3086	0	0.883484	8	7.347331	19	57	3	0	D	West Ham	0	0.883484	10.
390	Aaron Ramsdale	37	3330	0	0.000000	0	0.053671	0	1	1	0	GK	Sheffield United	0	0.000000	2.
139	Aaron Wan-Bissaka	34	3060	2	0.932454	4	2.547083	7	31	3	0	D	Manchester United	2	0.932454	12.
126	Abdoulaye Doucouré	28	2409	2	2.369523	3	2.381616	18	20	6	0	M	Everton	2	2.369523	10.
380	Aboubakar Kamara	11	303	0	0.654920	0	0.328969	4	6	1	1	F M S	Fulham	0	0.654920	1.
163	Adam Lallana	29	1528	1	1.614306	1	2.628758	22	25	0	0	F M S	Brighton	1	1.614306	9.
242	Adam Webster	28	2506	1	1.272957	0	0.253216	25	5	4	0	D	Brighton	1	1.272957	7.
119	Adama Traoré	36	2604	2	1.996262	2	5.229081	41	54	4	0	D F M S	Wolverhampton Wanderers	2	1.996262	9.
75	Ademola Lookman	34	2765	4	6.251116	4	5.258407	69	61	5	0	F M S	Fulham	4	5.489947	15.

Display Data in a Specific Order

The following example sorts the data by the number of *assists* in descending order, then if they are equal, by *team_title* in ascending order, then displays the first 10 rows of the results.

```
df1 = df_noid.sort_values(by=["assists", "team_title"], \
                          ascending=[False,True])
df1.head(10)
```

	player_name	games	time	goals	xG	assists	xA	shots	key_passes	yellow_cards	red_cards	position	team_title	npg	npxG	xG
0	Harry Kane	35	3097	23	22.174859	14	7.577094	138	49	1	0	F	Tottenham	19	19.130183	24.9
2	Bruno Fernandes	37	3117	18	16.019454	12	11.474996	121	95	6	0	M S	Manchester United	9	8.407840	26.9
58	Kevin De Bruyne	24	1918	5	9.908440	11	10.003763	79	72	1	0	M S	Manchester City	3	7.624933	21.0
51	Jack Grealish	26	2187	6	5.192684	10	9.334137	50	81	5	0	F M S	Aston Villa	6	5.192684	17.4
3	Son Heung-Min	37	3139	17	11.023287	10	9.512992	68	75	0	0	F M S	Tottenham	16	10.262118	20.6
57	Raphinha	30	2369	6	6.219143	9	9.524061	67	65	3	0	M S	Leeds	6	6.219143	16.7
6	Jamie Vardy	34	2848	15	19.942946	9	5.087882	82	28	1	0	F S	Leicester	7	13.092427	18.2
15	Marcus Rashford	37	2941	11	9.579710	9	4.185122	79	44	4	0	F M S	Manchester United	11	9.579710	20.4
83	Pascal Groß	33	2379	3	5.010526	8	5.368290	32	69	3	0	D M S	Brighton	0	1.965887	10.4
49	Timo Werner	35	2605	6	13.432796	8	6.667277	80	36	2	0	F M S	Chelsea	6	13.432796	20.5

Statistical Methods

Statistical Data Description

- The data exploration method can be carried out by applying concepts from statistics.
- Pandas provide several statistical functions that can be applied to a DataFrame.

Statistical Functions in Pandas

count	Number of non-NA observations
sum	Sum of values
mean	Mean of values
mad	Mean absolute deviation
median	Arithmetic median of values
min	Minimum
max	Maximum
mode	Mode
abs	Absolute Value
prod	Product of values
quantile	Sample quantile (value at %), 1st quartile = quantile(0.25)

std	Bessel-corrected sample standard deviation (consider a sample set, not the entire population)
var	Unbiased variance
sem	Standard error of the mean
skew	Sample skewness (3rd moment)
kurt	Sample kurtosis (4th moment)
cumsum	Cumulative sum
cumprod	Cumulative product
cummax	Cumulative maximum
cummin	Cumulative minimum

Statistical Summary with the describe() function

The `describe()` method displays basic statistics for each column of numeric data in a DataFrame, including the amount of data (*count*), arithmetic mean (*mean*), standard deviation (*std*), smallest value (*min*), first quartile (25%), quartile second/median (50%), third quartile (75%), and largest value (*max*).

```
df_noid.describe()
```

	games	time	goals	xG	assists	xA	shots	key_passes	yellow_cards	red_cards	npg	
count	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.
mean	19.643678	1420.068966	1.862069	2.000806	1.289272	1.376029	17.379310	12.963602	2.061303	0.091954	1.668582	1.
std	11.619836	1031.604819	3.338851	3.317946	2.083350	1.886510	21.572664	16.164361	2.203661	0.295800	2.909929	2.
min	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	10.000000	470.250000	0.000000	0.074668	0.000000	0.049245	2.000000	1.000000	0.000000	0.000000	0.000000	0.
50%	21.000000	1342.000000	1.000000	0.737295	0.000000	0.691122	10.000000	7.000000	2.000000	0.000000	0.500000	0.
75%	30.000000	2319.000000	2.000000	2.053378	2.000000	2.050509	23.750000	19.000000	3.000000	0.000000	2.000000	1.
max	38.000000	3420.000000	23.000000	22.174859	14.000000	11.474996	138.000000	95.000000	12.000000	2.000000	19.000000	19.

Statistical Summary with the describe() function

If you want to display non-numeric column statistics, use `describe(include='all')` which also includes the number of unique values in the column (unique), the mode value (top), and the mode frequency (freq).

	player_name	games	time	goals	xG	assists	xA	shots	key_passes	yellow_cards	red_cards	position	t
count	522	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522.000000	522	
unique	522	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	14	
top	Willian José	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	M S	
freq	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	106	
mean	NaN	19.643678	1420.068966	1.862069	2.000806	1.289272	1.376029	17.379310	12.963602	2.061303	0.091954	NaN	
std	NaN	11.619836	1031.604819	3.338851	3.317946	2.083350	1.886510	21.572664	16.164361	2.203661	0.295800	NaN	
min	NaN	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	NaN	
25%	NaN	10.000000	470.250000	0.000000	0.074668	0.000000	0.049245	2.000000	1.000000	0.000000	0.000000	NaN	
50%	NaN	21.000000	1342.000000	1.000000	0.737295	0.000000	0.691122	10.000000	7.000000	2.000000	0.000000	NaN	
75%	NaN	30.000000	2319.000000	2.000000	2.053378	2.000000	2.050509	23.750000	19.000000	3.000000	0.000000	NaN	
max	NaN	38.000000	3420.000000	23.000000	22.174859	14.000000	11.474996	138.000000	95.000000	12.000000	2.000000	NaN	

Central Tendency: Arithmetic Mean, Median, Mode

- The data items for each column can be viewed as a sample from a certain statistical distribution.
- The **central tendency** basically gives an overview of the location where most of the data items in the distribution gather.
- There are 3 central tendency measures that are most widely used: the arithmetic ***mean, median,*** and ***mode***.
- Although *mode* can be applied to numeric data, Pandas assumes the use of the mode concept only for non-numeric data while the *arithmetic mean* and *median* are only applied to numeric data.

Arithmetic Mean

- The average value that is commonly understood by most people.
- The **arithmetic mean** of a set of numbers = sum of all the numbers divided by the number of numbers in the set.
- Given a set of N numbers S={x_1,..., x_N}, the arithmetic mean μ_S or $\bar{x}$ of S is defined as:

$$\mu_S = \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{x_1 + \ ... + x_N}{N}$$

- It is a measure of the data center (central tendency) that can be used for interval and ratio type data.
- Properties: the total distance of each number x_i to the arithmetic mean $\bar{x}$ is 0.
- Can be used as a number that represents the entire collection, as long as the data distribution is not skew (asymmetric).

Arithmetic Mean

- Using Pandas, the arithmetic mean can be calculated with the `mean()` function.
- Example: calculate the average goal, games, the number of assists, and the number of shots for each player.

```
df_noid[['goals', 'games', 'assists', 'shots']].mean()
```

```
goals        1.862069
games       19.643678
assists      1.289272
shots       17.379310
dtype: float64
```

Median

- The **median** of a data set is the data item that is in the middle.
- Given a set of N numbers S={x_1,..., x_N}, the median x_j of S is defined as:
  - If N is odd, then the median is the data item in the $\left(\frac{N+1}{2}\right)$ position or ranking, after the data items are sorted.
  - If N is even, then the median is the arithmetic mean between the data items at position-$\left(\frac{N}{2}\right)$ and the data items at position- $\left(\frac{N}{2}+1\right)$ after the data items are sorted.
- The median can be used for ordinal, interval, and ratio data, but not for nominal or categorical data. Note that data of ordinal type does not have to be numbers, but must have order implicitly or explicitly. However, Pandas assumes by default that median calculations are performed on numeric data only.
- The median is a robust measure of data centers so it is not affected by the presence of outliers.
- The median is more suitable to be used as a representative of the distribution of data than the arithmetic mean if the distribution is skew.

Median

- Using Pandas, the median can be calculated with the `median()` function.
- Example: calculate the median of goals, games, the number of assists, and the number of shots for each player.

```
df_noid[['goals', 'games', 'assists', 'shots']].median()
```

```
goals       1.0
games      21.0
assists     0.0
shots      10.0
dtype: float64
```

Mode

- **Mode** is the value that appears most often in a data set.
- Used as a measure of the data center (central tendency) for data of nominal/categorical type.
  - Not guaranteed to be unique in a data distribution (it can be more than one mode in a distribution).
  - Show the value that has the highest probability of being obtained when the data is sampled.
- Example:
  - The data set {1,2,2,3,4,4,7,8} has two modes: 2 and 4.
  - If the data follows a continuous distribution, for example {0.935, …, 1.134,…, 2.643, …, 3.459, …, 3.995, ….} then statistically, it should not be assumed that there will be two data with exactly the same value. You can perform discretization so that data of nominal type is obtained, then look for the mode.
- For a perfectly symmetrical distribution (normal distribution), the mode value will be equal to the arithmetic mean and median values.

Mode

- Using Pandas, the mode can be calculated with the `mode()` function.
- Example: looking for the mode in the team_title column shows the team that has the most number of players because it appears the most in the data.

```
df_noid[['team_title']].mode()
```

	team_title
0	Everton
1	West Bromwich Albion

Distribution Representation: Range, Quantiles, Standard Deviation, Outliers

- If the description of the data center shows the location of the data items gathered, then the description of the data distribution describes how far the data items spread from the data center.
- There are several measurements that can be used to give an idea of the distribution of data including *range, quantiles, standard deviation, variance,* and *outliers*.

Range

- **Range** is defined as the difference between the maximum and minimum values in the data set.
- Large range values indicate that the data tends to be spread out, while a small range can indicate that the data tends to cluster. However, this is not always reliable, especially if the maximum or minimum data values are outliers.
- There is no special function in Pandas to calculate ranges as these can easily be calculated using the `min()` and `max()` functions.
- On the other hand, because the value of the range only depends on two data items, this measure is usually only suitable for small datasets.

Quantiles

- A **quantile** of a data set is defined as an intersection point that determines how many data items are smaller than it and how many are larger.
- **Quartiles** are 4-quantile consisting of three points or three data items.
  - First quartile (Q_1): data value so that 25% of the total data is less than it.
  - Second quartile (Q_2) or median: data value so that half of the existing data is less than it. It can be used as a measure of data centers (central tendency) as an alternative to the mean (especially if the data distribution is skewed).
  - Third quartile (Q_3): data value so that 75% of all data is less than it.
- Quartiles can be used for ordinal, interval, and ratio type data.

Quartiles

- Using Pandas, quartiles can be calculated with the `quantile()` function.
- One of the important parameters is `q`, which is between 0 and 1.
- Example: calculate the third quartile in the distribution of number of goals and all columns.

```
df_noid.quantile(0.75) # 3rd quartile
```

```
games              30.000000
time             2319.000000
goals               2.000000
xG                  2.053378
assists             2.000000
xA                  2.050509
shots              23.750000
key_passes         19.000000
yellow_cards        3.000000
red_cards           0.000000
npg                 2.000000
npxG                1.945799
xGChain             8.308002
xGBuildup           5.254647
Name: 0.75, dtype: float64
```

```
df_noid[['goals']].quantile(q=0.75)
```

```
goals    2.0
Name: 0.75, dtype: float64
```

Standard Deviation and Variance

- **Standard deviation** is one measure of the distribution of data used for interval and ratio type data.
- For a set of numbers S={x_1,..., x_N } with mean μ_S, the standard deviation

$$\sigma_s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_S)^2} = \sqrt{\frac{(x_1 - \mu_S)^2 + \cdots + (x_N - \mu_S)^2}{N-1}}$$

- The square of σ_S, i.e. ${\sigma_S}^2$ is called the **variance.**
- Standard deviation value
  - large = data are generally spread far from the arithmetic mean value
  - small = data are generally collected close to the arithmetic mean value
- The standard deviation can also be viewed as the degree of uncertainty of data measurement
  - For example: in repeated measurements with the same instrument, if the standard deviation of the measurement data is large, it means that the measurement precision is low.

Standard Deviation and Variance

- Pandas provide `var()` and `std()` functions to calculate variance and standard deviation.
- Example: calculate the variance and standard deviation of the number of goals scored by each player.

```
df_noid[['goals']].var(), df_noid[['goals']].std()
```

```
(goals    11.147925
 dtype: float64,
 goals    3.338851
 dtype: float64)
```

Outliers

- An **outlier** is a data item that is very different from most of the other data items in the set.
- Outliers can appear due to errors that occur when collecting data in the field or damage to measurement tools.
- If there are outliers, the data analyst usually remove the outlier from the data before further processing.
- However, outliers can also appear not because of measurement errors, but because they are actually contained in the data. This can indicate an interesting anomaly for further analysis.
- There are no definite criteria for whether a data item can be classified as an outlier. Therefore, in data science, there are several alternative methods or criteria for detecting anomalies.

Define Outliers (Roughly) based on Statistics

- *3-sigma rule*: Suppose S is the set of values to look for outliers, μ_s is the arithmetic mean of S and σ_S is the standard variation. If the data is approximately normally distributed:
  - x_i is an outlier if $x_i < \mu_S - \sigma_S$ or $x_i > \mu_s + \sigma_S$
    → the probability that the data is further from the mean than the standard deviation is 31.73%.
  - x_i is an outlier if $x_i < \mu_S - 2\sigma_S$ or $x_i > \mu_s + 2\sigma_S$
    → the probability that the data is further from the mean than 2 times the standard deviation is 4.55%.
  - x_i is an outlier if $x_i < \mu_S - 3\sigma_S$ or $x_i > \mu_s + 3\sigma_S$
    → the probability that the data is further from the mean than 3 times the standard deviation is 0.27%.
  - Disadvantages: (i) the assumption of normal distribution (not sure), (ii) the mean and standard deviation are affected by the outlier value itself, and (iii) cannot detect outliers if the amount of data is small (small sample size).
- *Tukey's fences*: use **interquartile range** $IQR = Q_3 - Q_1$.
  - x_i is an outlier if $x_i < Q_1 - 1.5(IQR)$ or $x_i > Q_3 + 1.5(IQR)$.
  - x_i is an extreme outlier if $x_i < Q_1 - 3(IQR)$ or $x_i > Q_3 + 3(IQR)$.
- Other (probably better) methods: Visualization, Grubb's test, Dixon's Q test, Expectation Maximization Algorithm, k-Nearest Neighbor Distance, Density-based local outlier factor (variation density-based clustering), etc.

Finding Outliers with the 3-Sigma Rule

```
mean = df_noid[['goals']].mean()
stdev = df_noid[['goals']].std()
iso = (df_noid[['goals']] < mean - 3*stdev) \
                | (df_noid[['goals']] > mean + 3*stdev)

df1 = df_noid[['player_name','goals']].assign(is_outlier=iso)
df1.loc[df1['is_outlier']]
```

- First, calculate the mean and std.
- Then, the `iso` variable are Boolean Series which represent the criteria for outliers according to the 3-sigma rule.
- The Series is assigned as additional columns in the DataFrame.
- The DataFrame is then filtered only for rows with a value of True in one of the columns of `is_outlier`.

	player_name	goals	is_outlier
0	Harry Kane	23	True
1	Mohamed Salah	22	True
2	Bruno Fernandes	18	True
3	Son Heung-Min	17	True
4	Patrick Bamford	17	True
5	Dominic Calvert-Lewin	16	True
6	Jamie Vardy	15	True
7	Ollie Watkins	14	True
8	Ilkay Gündogan	13	True
9	Alexandre Lacazette	13	True
10	Callum Wilson	12	True
11	Kelechi Iheanacho	12	True
12	Danny Ings	12	True
13	Chris Wood	12	True

Finding Outliers with the *Tukey's fences*

```
q1 = df_noid['goals'].quantile(q=0.25)
q3 = df_noid['goals'].quantile(q=0.75)
iqr = q3 - q1
iso = (df_noid['goals'] < q1 - 1.5*iqr) | (df_noid['goals'] > q3 + 1.5*iqr)
iseo = (df_noid['goals'] < q1 - 3*iqr) | (df_noid['goals'] > q3 + 3*iqr)
df1 = df_noid[['player_name', 'goals']].assign(is_outlier = iso, is_extreme_outlier = iseo)
df1.loc[df1['is_outlier'] | df1['is_extreme_outlier']]
```

- First, calculate the first and third quartiles and the interquartile range (IQR).
- Then, the `iso` and `iseo` variables are Boolean Series which represent the criteria for outliers and extreme outliers according to Tukey.
- The two Series are assigned as additional columns in the DataFrame.
- The DataFrame is then filtered only for rows with a value of True in one of the columns of `is_outlier` and `is_extreme_outlier`.
- The results are on the next slide.

Finding Outliers with the *Tukey's fences*

	player_name	goals	is_outlier	is_extreme_outlier
0	Harry Kane	23	True	True
1	Mohamed Salah	22	True	True
2	Bruno Fernandes	18	True	True
3	Son Heung-Min	17	True	True
4	Patrick Bamford	17	True	True
5	Dominic Calvert-Lewin	16	True	True
6	Jamie Vardy	15	True	True
7	Ollie Watkins	14	True	True
8	Ilkay Gündogan	13	True	True
9	Alexandre Lacazette	13	True	True
10	Callum Wilson	12	True	True
11	Kelechi Iheanacho	12	True	True
12	Danny Ings	12	True	True
13	Chris Wood	12	True	True
14	Wilfried Zaha	11	True	True
15	Marcus Rashford	11	True	True
16	Sadio Mané	11	True	True
17	Gareth Bale	11	True	True
18	Matheus Pereira	11	True	True
19	Pierre-Emerick Aubameyang	10	True	True
20	Michail Antonio	10	True	True
21	Christian Benteke	10	True	True
22	Raheem Sterling	10	True	True
23	Edinson Cavani	10	True	True
24	Anwar El Ghazi	10	True	True
25	Tomas Soucek	10	True	True
26	Roberto Firmino	9	True	True
27	Jesse Lingard	9	True	True
28	Riyad Mahrez	9	True	True
29	Harvey Barnes	9	True	True
30	Diogo Jota	9	True	True
31	Che Adams	9	True	True
32	James Ward-Prowse	8	True	False
33	Jarrod Bowen	8	True	False
34	Neal Maupay	8	True	False
35	Gabriel Jesus	8	True	False
36	Nicolas Pepe	8	True	False
37	Phil Foden	8	True	False
38	Joe Willock	8	True	False
39	James Maddison	8	True	False
40	Stuart Dallas	8	True	False
41	Jack Harrison	8	True	False
42	Bertrand Traoré	7	True	False
43	Jorginho	7	True	False
44	Rodrigo	7	True	False
45	Richarlison	7	True	False
46	Ferrán Torres	7	True	False
47	Mason Greenwood	7	True	False
48	David McGoldrick	7	True	False
49	Timo Werner	6	True	False
50	Danny Welbeck	6	True	False
51	Jack Grealish	6	True	False
52	Tammy Abraham	6	True	False
53	Gylfi Sigurdsson	6	True	False
54	James Rodríguez	6	True	False
55	Youri Tielemans	6	True	False
56	Mason Mount	6	True	False
57	Raphinha	6	True	False

Frequency Table

- The frequency table is used to display the frequency or the number of data for the nominal-type columns.
- `value_counts()` returns the frequency of each unique value in the column.
- The highest value is the mode in that column.
- Example: Calculate the number of players for each team. There is data with two/three team title because there are players who play for two/three clubs in the same season (there are player transfers).

```
In [18]: df['team_title'].value_counts()

Out[18]: West Bromwich Albion                28
         Everton                             28
         Fulham                              27
         Wolverhampton Wanderers             27
         Southampton                         27
         Sheffield United                    27
         Manchester United                   27
         Liverpool                           27
         Leicester                           27
         Brighton                            26
         Arsenal                             26
         Newcastle United                    26
         Chelsea                             25
         Burnley                             25
         Tottenham                           24
         Manchester City                     24
         Crystal Palace                      24
         West Ham                            23
         Leeds                               23
         Aston Villa                         23
         West Bromwich Albion,West Ham        1
         Everton,Southampton                  1
         Arsenal,West Bromwich Albion         1
         Chelsea,Fulham                       1
         Aston Villa,Chelsea                  1
         Arsenal,Newcastle United             1
         Liverpool,Southampton                1
         Arsenal,Brighton                     1
         Name: team_title, dtype: int64
```

Grouping Data based on Columns

- Data analysis can also be done by grouping data based on certain columns. Pandas provides a `groupby()` function that can group data by column.
- The output is a DataFrameGroupBy object that similar to the original DataFrame, but has been grouped according to the parameters given to the groupby() function.
- Then, statistical functions also can be applied to DataFrameGroupBy objects as regular DataFrames.
- Example: calculate the average goals per player for each team, then sorts the results by that average.

```
df_noid.groupby('team_title')[['goals']]\
        .mean().sort_values(by='goals',ascending=False)
```

	goals
team_title	
Arsenal,Newcastle United	8.000000
Manchester City	3.208333
Aston Villa,Chelsea	3.000000
Liverpool,Southampton	3.000000
Everton,Southampton	3.000000
Tottenham	2.750000
Leeds	2.608696
Manchester United	2.518519
West Ham	2.478261
Leicester	2.370370
Liverpool	2.370370
Chelsea	2.240000
Aston Villa	2.130435
Arsenal	1.961538
Crystal Palace	1.625000
Everton	1.607143
Southampton	1.555556
Brighton	1.500000
Newcastle United	1.384615
Burnley	1.280000
Wolverhampton Wanderers	1.222222
West Bromwich Albion	1.178571
Chelsea,Fulham	1.000000
Fulham	0.925926
Sheffield United	0.666667
Arsenal,Brighton	0.000000
West Bromwich Albion,West Ham	0.000000
Arsenal,West Bromwich Albion	0.000000

Correlation Analysis

- Correlation analysis was performed on the data to determine the dependency relationship between the two numeric columns in the data.
- Pandas provide a `corr()` method based on Pearson's correlation coefficient which has a range of -1 to 1 and describes whether one variable/column is linearly dependent on another variable/column.
  - 0 = no linear correlation
  - 1 = positive linear correlation
  - -1 = negative linear correlation

In [23]:
```
df.loc[:,'games':].corr()
```

Out[23]:

	games	time	goals	xG	assists	xA	shots	key_passes	yellow_cards	red_cards
games	1.000000	0.944591	0.439730	0.463869	0.504168	0.562806	0.599164	0.617867	0.565963	0.160326
time	0.944591	1.000000	0.398930	0.411203	0.473555	0.516638	0.529534	0.575065	0.592223	0.186333
goals	0.439730	0.398930	1.000000	0.932798	0.617490	0.607330	0.873363	0.567752	0.097151	0.053679
xG	0.463869	0.411203	0.932798	1.000000	0.636205	0.627495	0.910214	0.570488	0.093761	0.048815
assists	0.504168	0.473555	0.617490	0.636205	1.000000	0.885850	0.721220	0.835299	0.209349	-0.021444
xA	0.562806	0.516638	0.607330	0.627495	0.885850	1.000000	0.759568	0.946506	0.243912	0.006284
shots	0.599164	0.529534	0.873363	0.910214	0.721220	0.759568	1.000000	0.743370	0.249957	0.073932
key_passes	0.617867	0.575065	0.567752	0.570488	0.835299	0.946506	0.743370	1.000000	0.343357	0.022780
yellow_cards	0.565963	0.592223	0.097151	0.093761	0.209349	0.243912	0.249957	0.343357	1.000000	0.165064
red_cards	0.160326	0.186333	0.053679	0.048815	-0.021444	0.006284	0.073932	0.022780	0.165064	1.000000
npg	0.437110	0.392631	0.971591	0.894286	0.587316	0.585152	0.852989	0.539726	0.093270	0.055542
npxG	0.465546	0.408231	0.905710	0.979218	0.615503	0.611100	0.901386	0.545537	0.089065	0.047354
xGChain	0.726598	0.703801	0.727953	0.763909	0.752587	0.814487	0.843152	0.807958	0.401884	0.104005
xGBuildup	0.697196	0.731377	0.290990	0.282746	0.473254	0.547983	0.448197	0.618754	0.562467	0.167660

Visualization Methods

Visualization

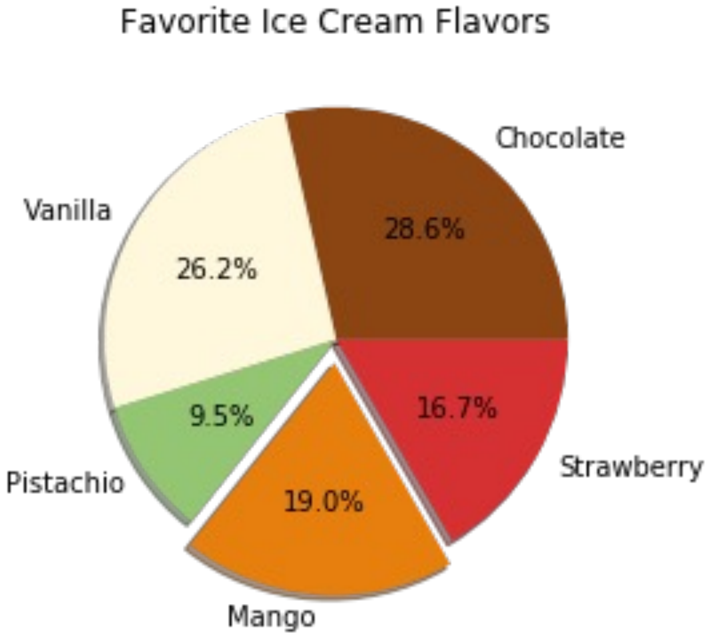
- Visualization plays an important role in the fields of machine learning and data science.
- Visualization is needed to filter the important information found in a number of data into a form that is meaningful and easy to understand.
- A good visualization can tell a story about data in a way that a sentence can't.
- Next, we'll explore some common visualization techniques using Python libraries such as **Matplotlib** and **Seaborn** to create informative graphs that provide information and knowledge about the dataset.

Variable Visualization

- Pie Chart
- Bar Chart
- Line Graph
- Scatter Plot
- Heatmap

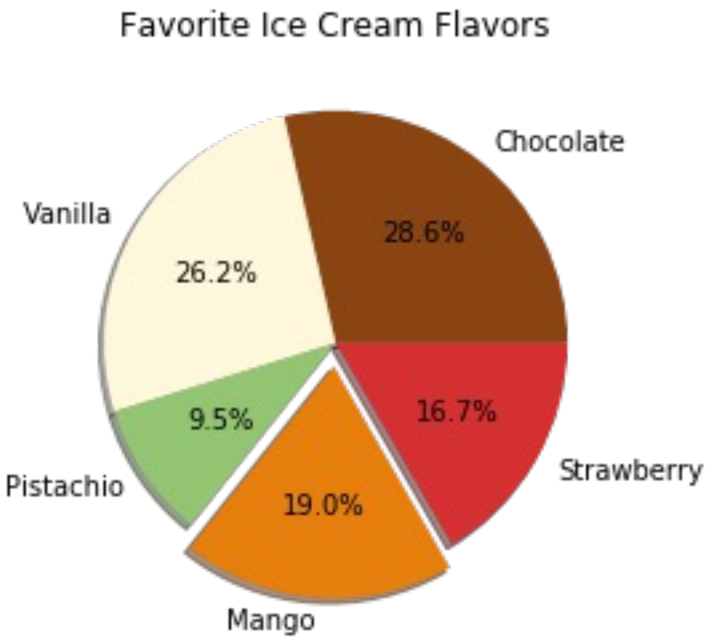
Pie Chart

- A **Pie Chart** is used to show how much of each type of category is in the dataset compared to the whole.
- The Matplotlib library provides a `pie()` method to create a pie chart.
- Example: Pie chart based on voting on ice cream flavours.
  - Label variable = ice cream flavours tuples
  - Voting variable = voting tuples
  - The data represents the number of votes for the favourite ice cream flavour.



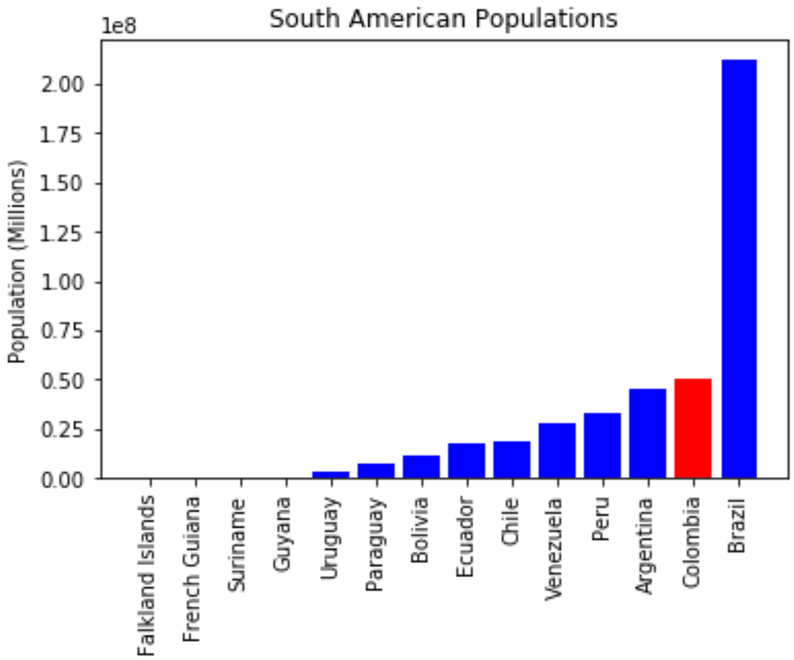
Pie Chart

```
import matplotlib.pyplot as plt

flavors = ('Chocolate', 'Vanilla',
'Pistachio', 'Mango', 'Strawberry')
votes = (12, 11, 4, 8, 7)
colors = ('#8B4513', '#FFF8DC', '#93C572',
'#E67F0D', '#D53032')
explode = (0, 0, 0, 0.1, 0)

plt.title('Favorite Ice Cream Flavors')
plt.pie(
    votes,
    labels=flavors,
    autopct='%1.1f%%',
    colors=colors,
    explode=explode,
    shadow=True
    )
plt.show()
```



Bar Chart

- A **Bar Chart** is a visualization tool that can be used to compare categorical data.
- Similar to the pie chart, it can be used to compare data categories against each other. A bar chart can show more categories of data than a pie chart.
- The Matplotlib library provides a `bar()` method to create a bar chart that has two arguments i.e. the x-axis contains the categorical data and the y-axis contains the numeric data to map.
- Example: The bar chart shows the population of each country in South America.
  - x-axis = country names
  - y-axis = population
  - Sorting the country from the largest to the lowest population
  - The highlight is shown for Colombia

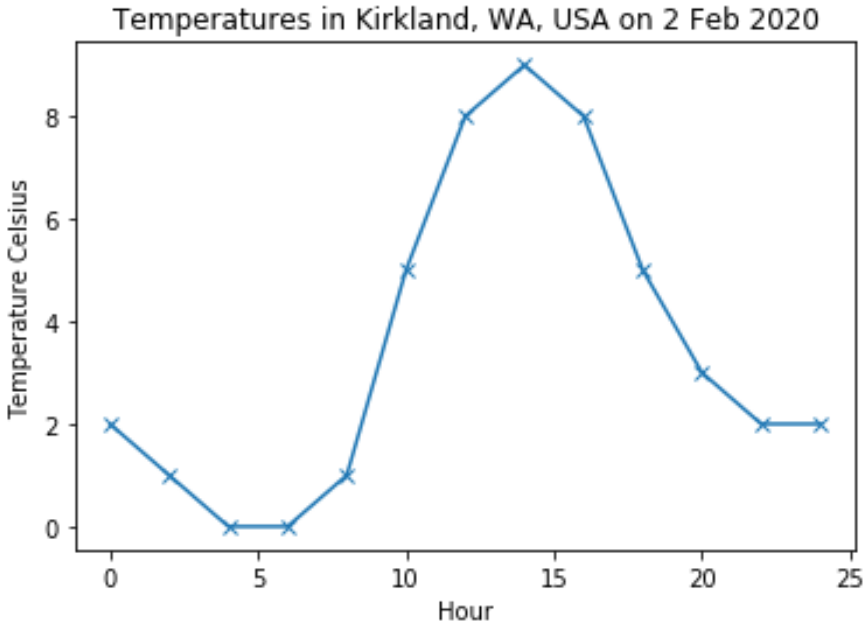


Bar Chart

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

countries = ('Argentina', 'Bolivia', 'Brazil','Chile', 'Colombia', 'Ecuador', 'Falkland
Islands', 'French Guiana', 'Guyana', 'Paraguay', 'Peru','Suriname', 'Uruguay', 'Venezuela')

populations = (45076704, 11626410, 212162757, 19109629, 50819826, 17579085, 3481, 287750,
785409, 7107305, 32880332, 585169, 3470475, 28258770)

df = pd.DataFrame({
    'Country': countries,
    'Population': populations})
df.sort_values(by='Population', inplace=True)

x_coords = np.arange(len(df))
colors = ['#0000FF' for _ in range(len(df))]
colors[-2] = '#FF0000'
plt.figure(figsize=(20,10))
plt.bar(x_coords, df['Population'], tick_label=df['Country'], color=colors)
plt.xticks(rotation=90)
plt.ylabel('Population (Millions)')
plt.title('South American Populations')
plt.show()
```

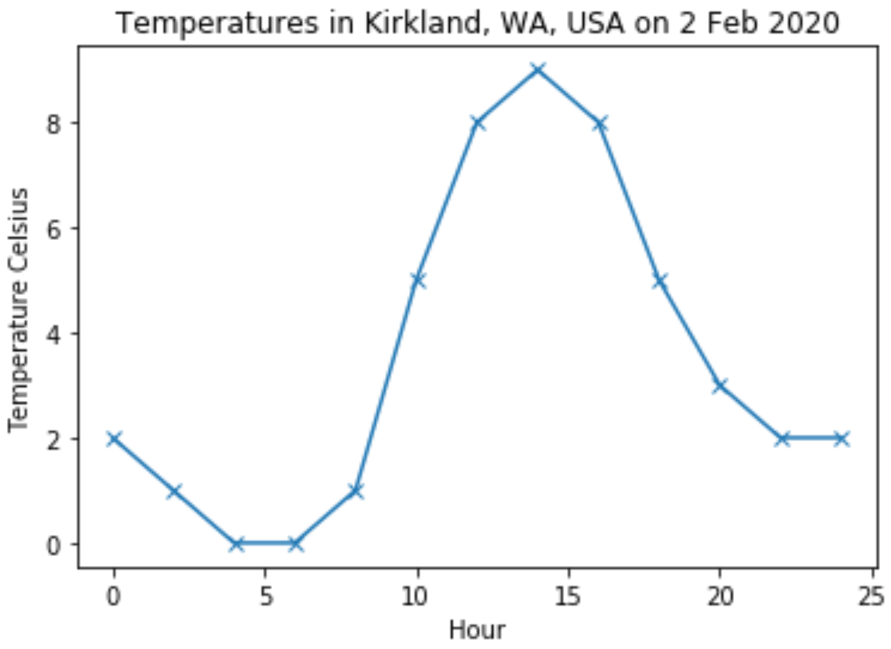
Line Graph

- A **Line Graph** is a visualization tool that is more useful to show the progress of data over several periods.
- For example, a line chart to create graph temperatures over time, stock prices over time, weight by the day, or other continuous metrics.
- The Matplotlib library provides a `plot()` method to create a line graph that has two arguments also
- Example: A line graph of temperature measurement results in Celsius for every hour of the day at a location.



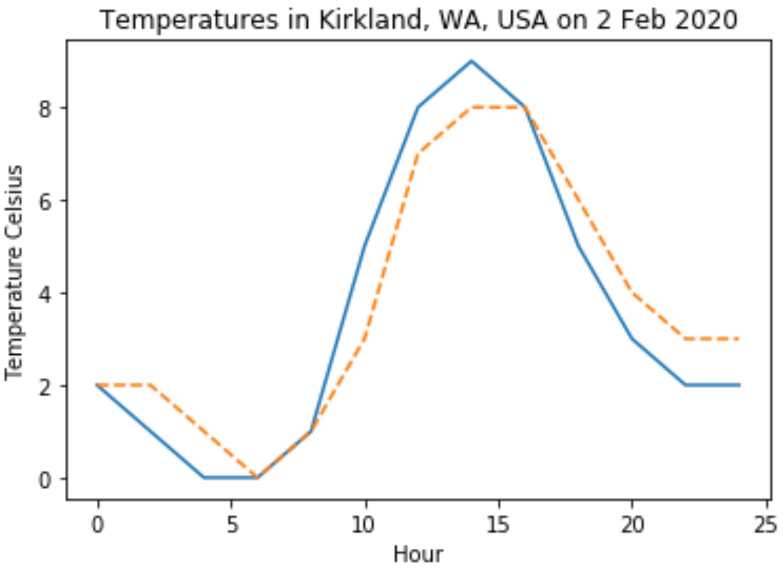
Line Graph

```
import matplotlib.pyplot as plt

temperature_c = [2, 1, 0, 0, 1, 5,
8, 9, 8, 5, 3, 2, 2]
hour = [0, 2, 4, 6, 8, 10, 12, 14,
16, 18, 20, 22, 24]

plt.plot(
  hour,
  temperature_c,
  marker='x',
)
plt.title('Temperatures in Kirkland,
WA, USA on 2 Feb 2020')
plt.ylabel('Temperature Celsius')
plt.xlabel('Hour')
plt.show()
```



Line Graph

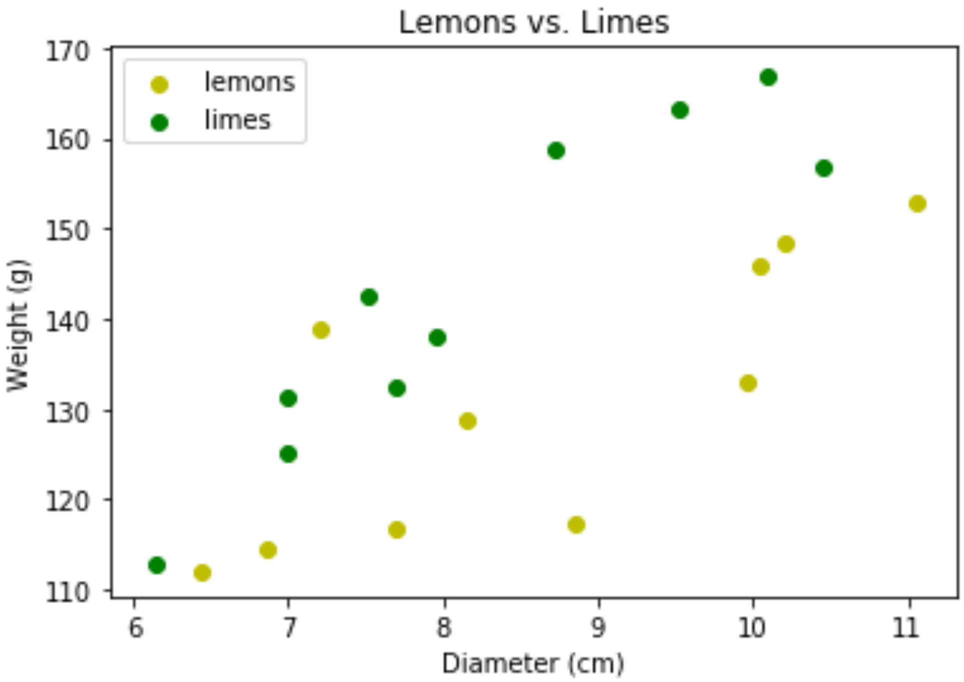
- We can even have multiple lines on the same chart in one figure.
- Usually, we illustrate two-line graphs to describe two kinds of data: actual data and predictive data.



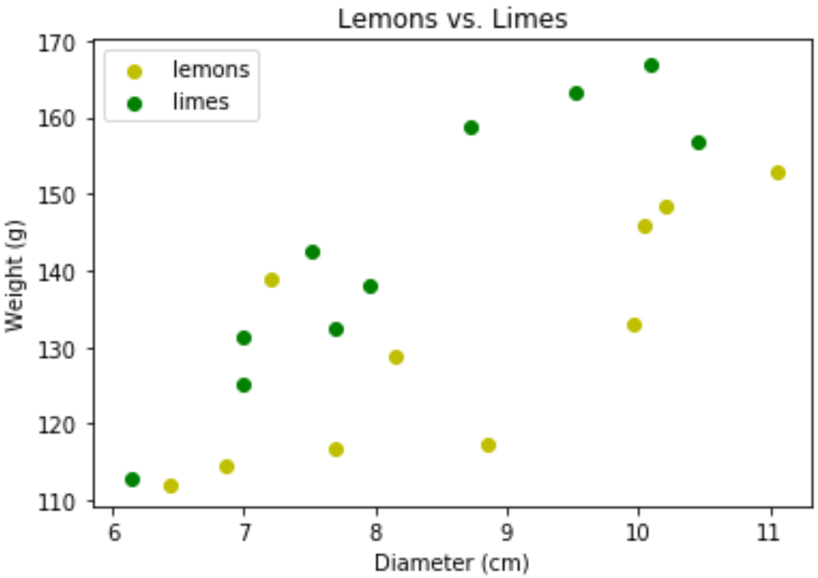

```
import matplotlib.pyplot as plt

temperature_c_actual = [2, 1, 0, 0, 1, 5, 8, 9, 8, 5, 3, 2, 2]
temperature_c_predicted = [2, 2, 1, 0, 1, 3, 7, 8, 8, 6, 4, 3, 3]
hour = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]

plt.plot(hour, temperature_c_actual)
plt.plot(hour, temperature_c_predicted, linestyle='--')
plt.title('Temperatures in Kirkland, WA, USA on 2 Feb 2020')
plt.ylabel('Temperature Celsius')
plt.xlabel('Hour')
plt.show()
```

Scatter Plot

- **Scatter plot** works well for data with two numeric components.
- Scatter plots can provide useful information, especially regarding patterns or outliers.
- The Matplotlib library provides a `scatter()` method to create a scatter plot.
- Example: Plot the difference in physiological characteristics between lemon and lime.
  - Weight (g)
  - Diameter(cm)



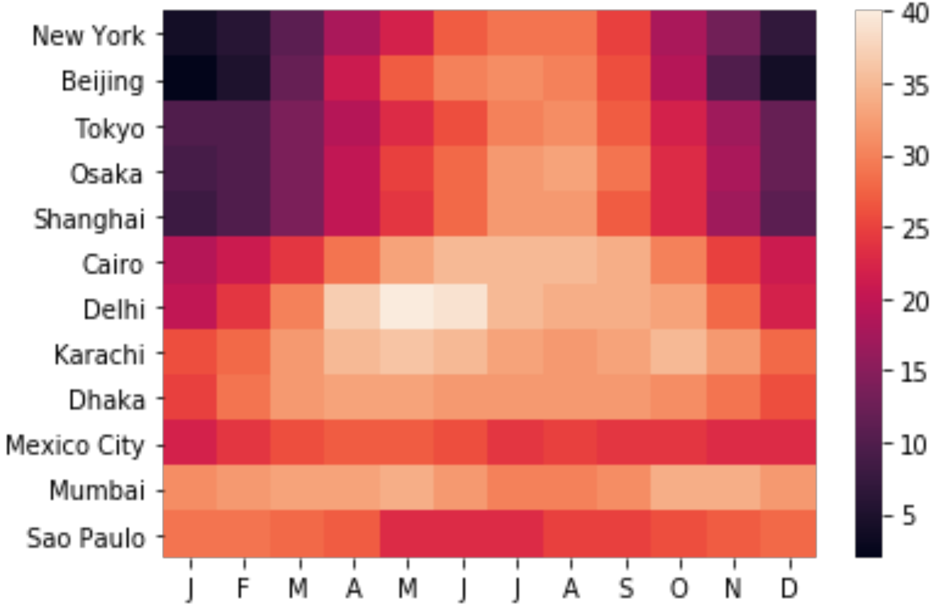
Scatter Plot



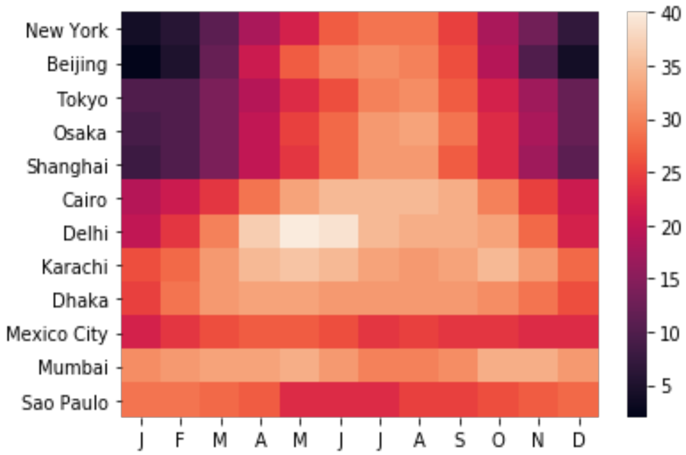

```
import matplotlib.pyplot as plt

lemon_diameter = [6.44, 6.87, 7.7, 8.85,
8.15, 9.96, 7.21, 10.04, 10.2, 11.06]
lemon_weight = [112.05, 114.58, 116.71,
117.4, 128.93, 132.93, 138.92, 145.98,
148.44, 152.81]

lime_diameter = [6.15, 7.0, 7.0, 7.69, 7.95,
7.51, 10.46, 8.72, 9.53, 10.09]
lime_weight = [112.76, 125.16, 131.36, 132.41,
138.08, 142.55, 156.86, 158.67, 163.28, 166.74]

plt.title('Lemons vs. Limes')
plt.xlabel('Diameter (cm)')
plt.ylabel('Weight (g)')
plt.scatter(lemon_diameter, lemon_weight,
color='y')
plt.scatter(lime_diameter, lime_weight,
color='g')
plt.legend(['lemons', 'limes'])
plt.show()
```

Heatmap

- **Heatmap** is a visualization that uses colour coding to represent the relative values/density of data across a surface.
- These colours can be used to visually inspect the data to find groups with similar values and detect trends in the data.
- To create a heatmap, we can use the Seaborn library.
- Seaborn is a visualization library that is built on top of Matplotlib and provides a higher-level interface and can create more attractive graphs.
- Example: Heatmap to map average monthly temperature data for the 12 largest cities in the world.



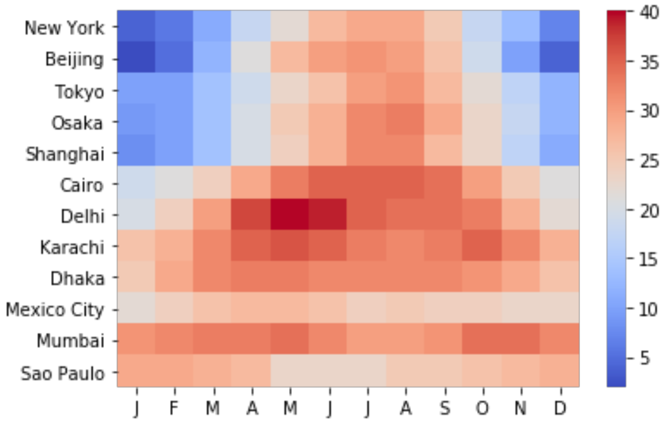
Heatmap




```
import seaborn as sns

cities = ['New York', 'Beijing', 'Tokyo', 'Osaka', 'Shanghai',
'Cairo', 'Delhi', 'Karachi', 'Dhaka', 'Mexico City', 'Mumbai',
'Sao Paulo']

temperatures = [
  [ 4,  6, 11, 18, 22, 27, 29, 29, 25, 18, 13,  7], # New York
  [ 2,  5, 12, 21, 27, 30, 31, 30, 26, 19, 10,  4], # Beijing
  [10, 10, 14, 19, 23, 26, 30, 31, 27, 22, 17, 12], # Tokyo
  [ 9, 10, 14, 20, 25, 28, 32, 33, 29, 23, 18, 12], # Osaka
  [ 8, 10, 14, 20, 24, 28, 32, 32, 27, 23, 17, 11], # Shanghai
  [19, 21, 24, 29, 33, 35, 35, 35, 34, 30, 25, 21], # Cairo
  [20, 24, 30, 37, 40, 39, 35, 34, 34, 33, 28, 22], # Delhi
  [26, 28, 32, 35, 36, 35, 33, 32, 33, 35, 32, 28], # Karachi
  [25, 29, 32, 33, 33, 32, 32, 32, 32, 31, 29, 26], # Dhaka
  [22, 24, 26, 27, 27, 26, 24, 25, 24, 24, 23, 23], # Mexico City
  [31, 32, 33, 33, 34, 32, 30, 30, 31, 34, 34, 32], # Mumbai
  [29, 29, 28, 27, 23, 23, 23, 25, 25, 26, 27, 28], # Sao Paulo
]

sns.heatmap(temperatures, yticklabels=cities, xticklabels=months)
```

Heatmap

- Seaborn provides several colour schemes which can be changed with `cmap` arguments.
- You can find colour schemes in the Matplotlib colormap documentation.



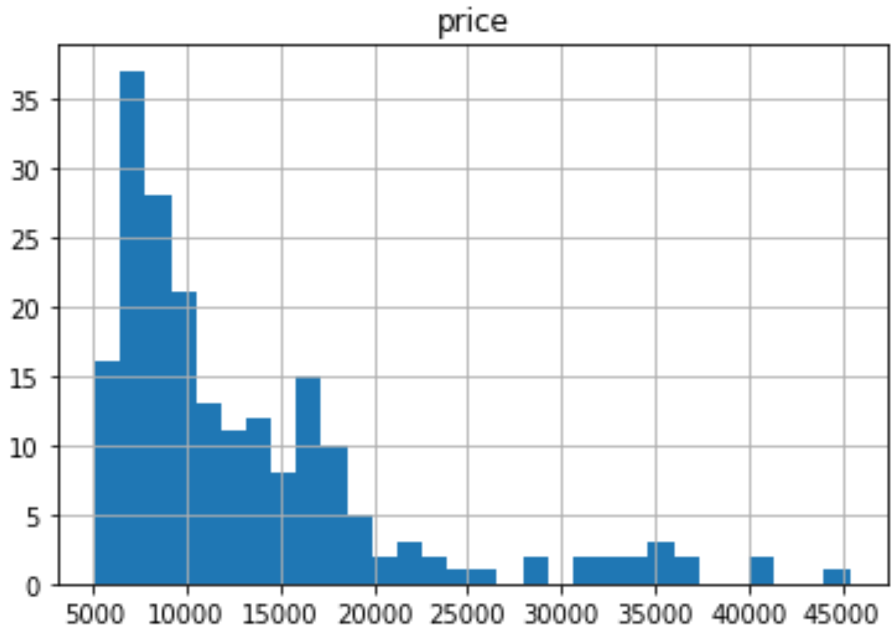

```
import seaborn as sns

cities = ['New York', 'Beijing', 'Tokyo', 'Osaka', 'Shanghai',
'Cairo', 'Delhi', 'Karachi', 'Dhaka', 'Mexico City', 'Mumbai',
'Sao Paulo']

temperatures = [
  [ 4,  6, 11, 18, 22, 27, 29, 29, 25, 18, 13,  7], # New York
  [ 2,  5, 12, 21, 27, 30, 31, 30, 26, 19, 10,  4], # Beijing
  [10, 10, 14, 19, 23, 26, 30, 31, 27, 22, 17, 12], # Tokyo
  [ 9, 10, 14, 20, 25, 28, 32, 33, 29, 23, 18, 12], # Osaka
  [ 8, 10, 14, 20, 24, 28, 32, 32, 27, 23, 17, 11], # Shanghai
  [19, 21, 24, 29, 33, 35, 35, 35, 34, 30, 25, 21], # Cairo
  [20, 24, 30, 37, 40, 39, 35, 34, 34, 33, 28, 22], # Delhi
  [26, 28, 32, 35, 36, 35, 33, 32, 33, 35, 32, 28], # Karachi
  [25, 29, 32, 33, 33, 32, 32, 32, 32, 31, 29, 26], # Dhaka
  [22, 24, 26, 27, 27, 26, 24, 25, 24, 24, 23, 23], # Mexico City
  [31, 32, 33, 33, 34, 32, 30, 30, 31, 34, 34, 32], # Mumbai
  [29, 29, 28, 27, 23, 23, 23, 25, 25, 26, 27, 28], # Sao Paulo
]

sns.heatmap(temperatures, yticklabels=cities, xticklabels=months,
cmap = 'coolwarm')
```

Statistical Visualization

- Histogram
- Correlation & Causation
- Statistical Category Variables
- Grouping (Pivot)
- ANOVA

Histogram

- The **histogram** is a visualization that is quite important in understanding the distribution of our data.
- Pandas provide a `hist()` method to create a histogram.
- Traditionally, histogram plots require only one data dimension intended to show a number of values or sets of values serially.
- Pandas `DataFrame.hist()` retrieve the DataFrame and display a histogram plot showing the distribution of values in a series.
- Example: Histogram based on price distribution



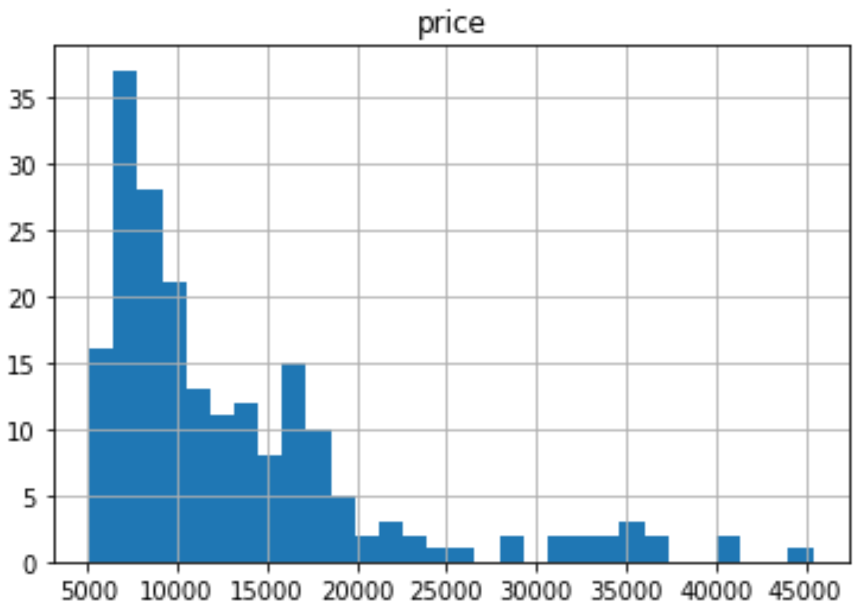
Histogram

As an example of a case study, the data we use is **automobile.csv**, which is data on car specifications from various brands and prices.

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	width	height	curb-weight	engine-type	num-of-cylinders	engine-size	fuel-system	bore	stroke	compre
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130	mpfi	3.47	2.68	
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.811148	0.890278	48.8	2548	dohc	four	130	mpfi	3.47	2.68	
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.822681	0.909722	52.4	2823	ohcv	six	152	mpfi	2.68	3.47	
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.848630	0.919444	54.3	2337	ohc	four	109	mpfi	3.19	3.40	
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.848630	0.922222	54.3	2824	ohc	five	136	mpfi	3.19	3.40	

Histogram

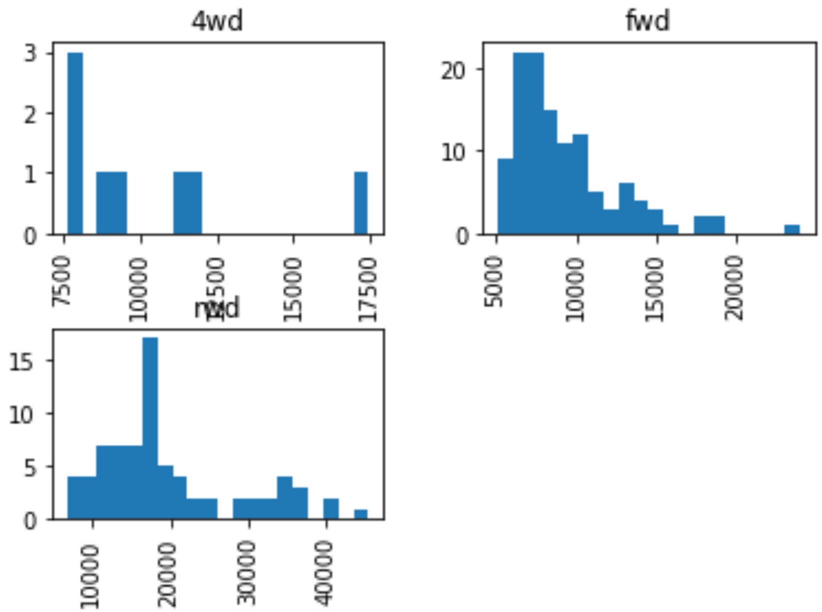
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

path='https://s3-api.us-geo.object
storage.softlayer.net/cf-courses-
data/CognitiveClass/DA0101EN/
automobileEDA.csv'

df = pd.read_csv(path)

df.hist(column='price', bins=30);
```



Histogram

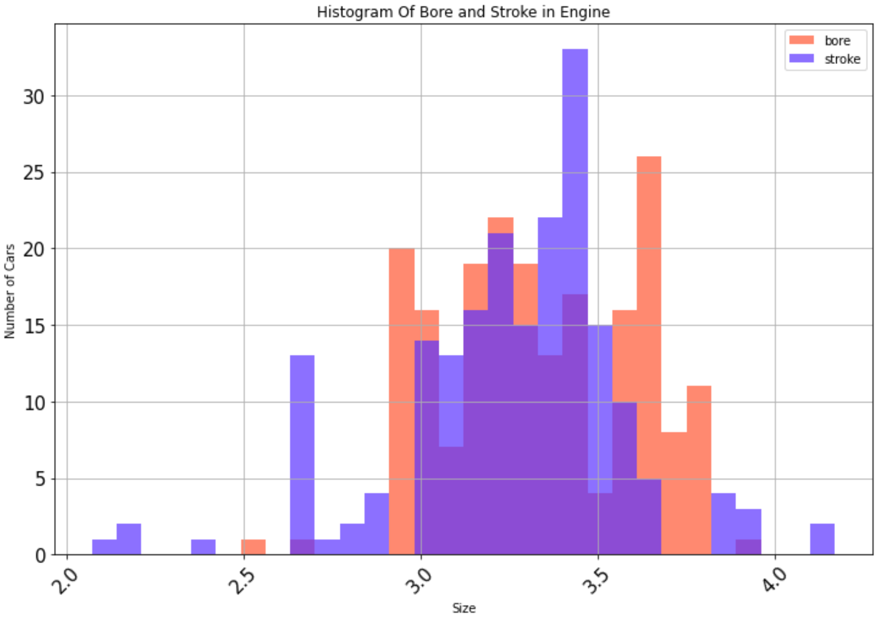
We can also plot multiple groups side by side. Example: creating a price histogram grouped by vehicle drive wheel (fwd, 4wd, rwd)

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

path='https://s3-api.us-geo.object
storage.softlayer.net/cf-courses-
data/CognitiveClass/DA0101EN/
automobileEDA.csv’

df = pd.read_csv(path)

df.hist(column='price', by='drive-
wheels', bins=20);
```



Histogram

We can also combine multiple histograms into one figure.
Example: creating a histogram based on bore and stoke into one figure




```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

path='https://s3-api.us-geo.object
storage.softlayer.net/cf-courses-
data/CognitiveClass/DA0101EN/
automobileEDA.csv'

df = pd.read_csv(path)
df[['bore','stroke']].plot(kind=
'hist', alpha=0.7, bins=30, title=
'Histogram Of Bore and Stroke in
Engine', rot=45, grid=True, figsize=
(12,8), fontsize=15, color=['#FF5733',
'#5C33FF'])

plt.xlabel('Size')
plt.ylabel("Number of Cars");
```

Correlation & Causation

- **Correlation** is a measurement that shows the value of interdependence between variables.
- **Causation** is a relationship between cause and effect between two variables.
- It is important to know the difference between the two and that correlation does not describe causation.
- Determining correlation is much simpler; determining causation requires further analysis.

Pearson Correlation

- **Pearson correlation** measures the linear dependence between two variables X and Y. The resulting coefficient is a value between -1 and 1:
  - 1: Total positive linear correlation.
  - 0: There is no linear correlation, the two variables most likely do not influence each other.
  - -1: Total negative linear correlation.
- Pearson Correlation is the default method of the `corr` function. We can calculate the Pearson Correlation from variable 'int64' or 'float64’.

p-Value

- Sometimes we want to know the significance of the estimated correlation, we can use the p-Value.
- The **p-Value** is the probability value that the correlation between these two variables is statistically significant.
- Usually, a significance level of 0.05, means 95% confidence that the correlation between the variables is significant.
- By convention, when:
  - p-Value is $<$ 0.001: there is strong evidence that the correlation is significant
  - p-Value is $<$ 0.05: there is moderate evidence that the correlation is significant.
  - p-Value is $<$ 0.1: there is weak evidence that the correlation is significant.
  - p-Value is $>$ 0.1: no evidence that the correlation is significant.
- We can use the scipy library to calculate correlation and p-value

Pearson Correlation and p-Value

Calculate the Pearson Correlation Coefficient and the p-Value of 'wheel-base' and 'price'.

```
from scipy import stats

pearson_coef, p_value = stats.pearsonr(df['wheel-base'], df['price'])
print("The Pearson Correlation Coefficient is", pearson_coef,
" with a P-value of P =", p_value)
```

```
The Pearson Correlation Coefficient is 0.584641822265508  with a P-value of P = 8.076488270733218e-20
```

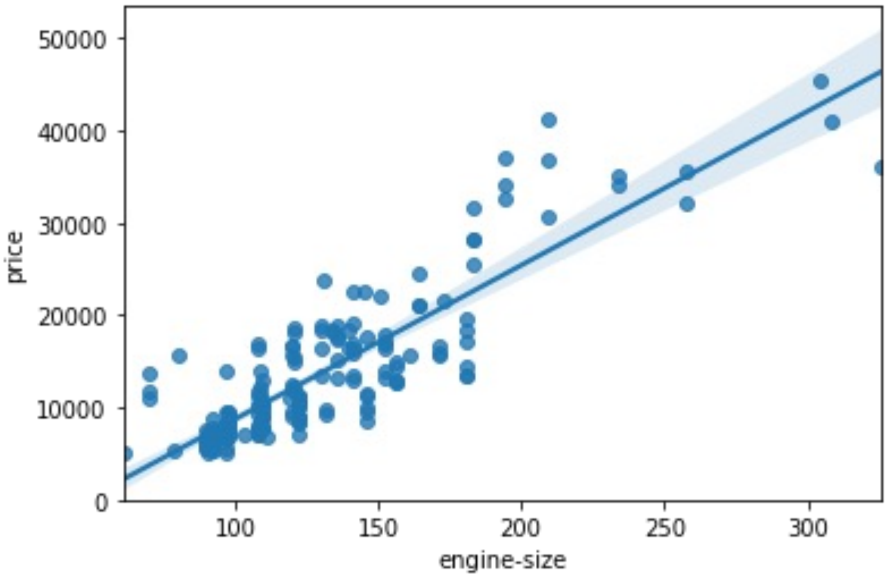
Since the p-Value is $<$ 0.001, the correlation between wheel-base and price is statistically significant, although the linear relationship is not very strong (0.588)

Regplot

- When visualizing individual variables, it's important to first understand what types of variables exist (using `df.dtypes`).
  This help us find the right visualization method for that variable.
- Suppose a continuous numeric variable having the int64 or float64 types can be visualized using a scatterplot with the appropriate lines.
- To understand the (linear) relationship between variables, we can use the `regplot` function. This function plots a scatterplot plus the appropriate regression line for the data.
- We can also check the correlation between variables using the `corr()` method.

Regplot

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.regplot(x="engine-size",
y="price", data=df)
plt.ylim(0,)

df[["engine-size", "price"]].corr()
```

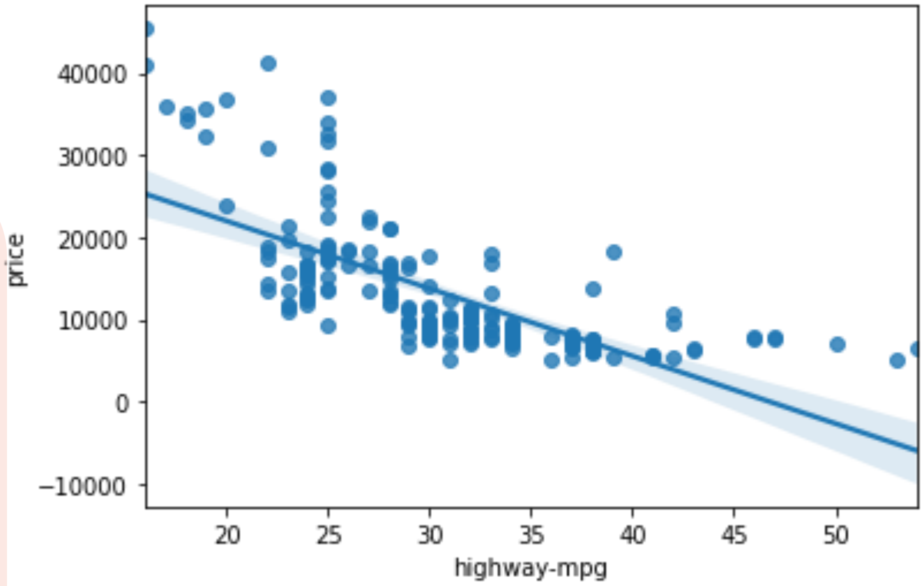


	engine-size	price
engine-size	1.000000	0.872335
price	0.872335	1.000000

The figure shows a **strong positive correlation** between variables. The result of the correlation between engine-size and price is around 0.87. When the engine capacity increases, the price of the car will also increase, this shows a linear relationship between these two variables. Engine size has the potential to be a price predictor.

Regplot

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.regplot(x="highway-mpg",
y="price", data=df)
plt.ylim(0,)

df[["highway-mpg", "price"]].corr()
```

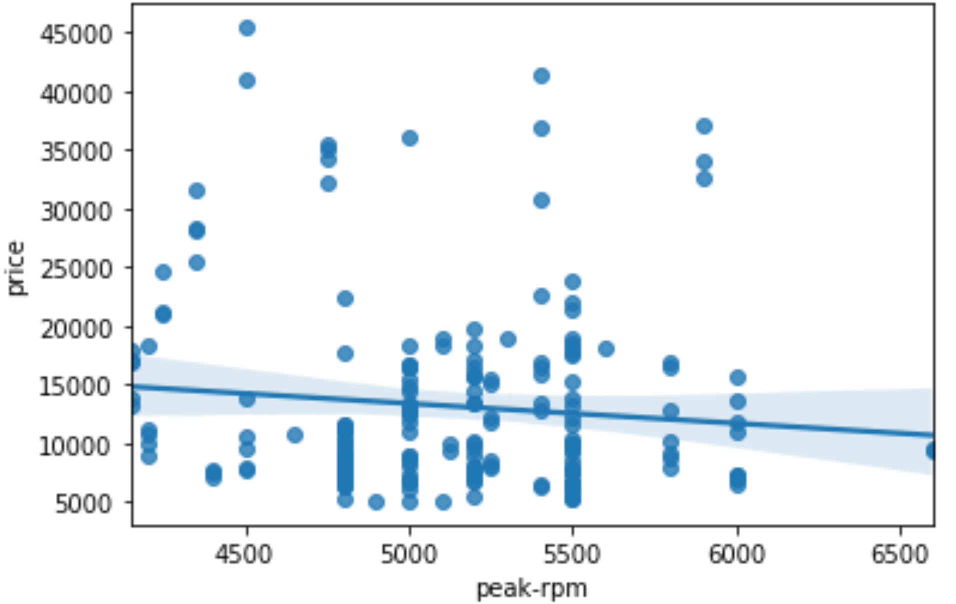


	highway-mpg	price
highway-mpg	1.000000	-0.704692
price	-0.704692	1.000000

The figure shows a **strong negative correlation** between variables. The result of the correlation between highway-mpg and price is around -0.705. When the highway-mpg increases, the price of the car will decrease, this shows an inverse or negative relationship between these two variables. Engine size has the potential to be a price predictor.

Regplot

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.regplot(x="peak-rpm",
y="price", data=df)
plt.ylim(0,)

df[["peak-rpm", "price"]].corr()
```



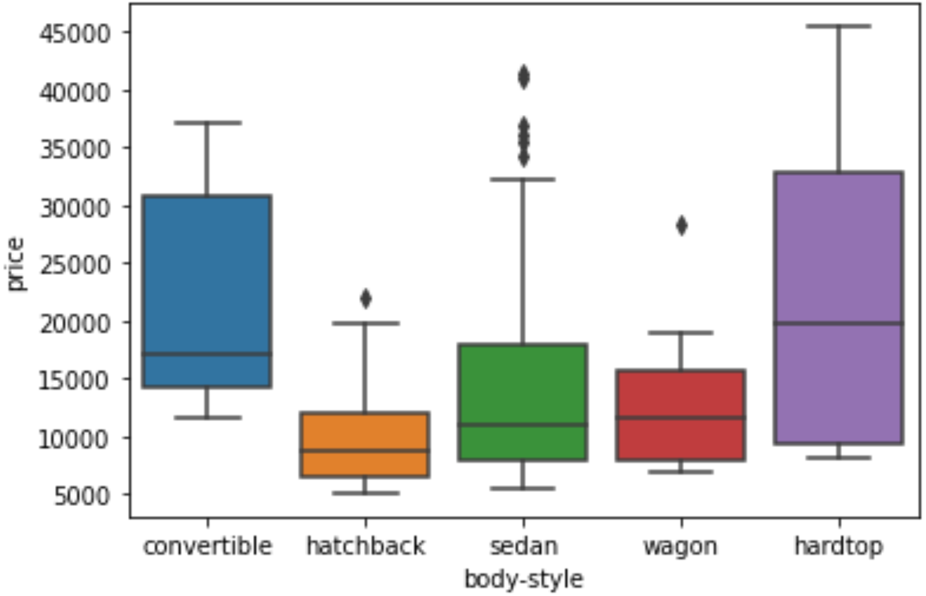
	peak-rpm	price
peak-rpm	1.000000	-0.101616
price	-0.101616	1.000000

Peak rpm is not a good price predictor because the regression line is nearly horizontal. The data points are very spread out and far from the lines, this shows a lot of variability. Therefore peak-rpm is not a reliable variable to predict price. The result of the correlation between peak-rpm and price is around -0.102.

Statistical Category Variables

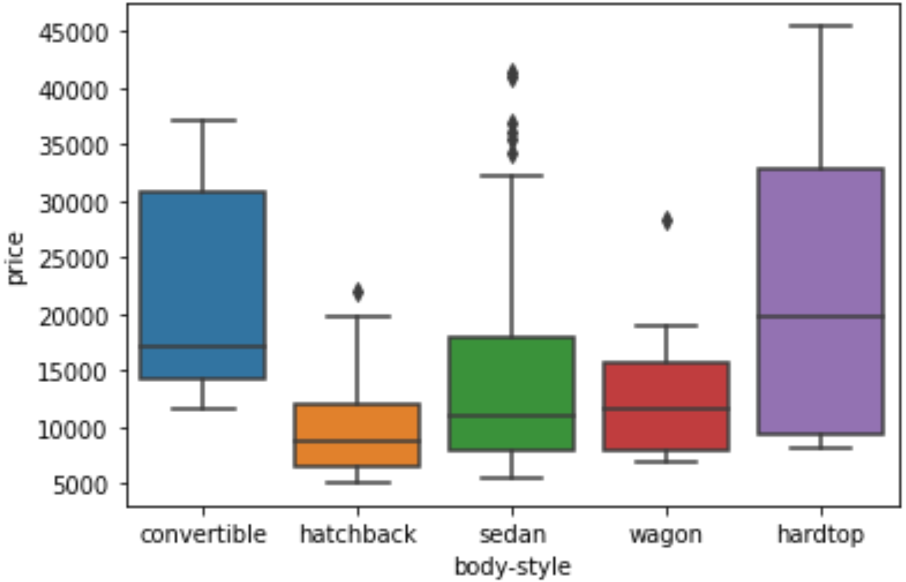
- **Statistical category variables** are variables that describe the characteristics of a data unit and are selected from a group of categories.
- Categorical variables can be of type "object" or "int64".

  A good way to visualize categorical variables is to use a **boxplot**.
- Boxplots describe statistical variables such as the 1st quartile, median/2nd quartile, 3rd quartile, maximum value, minimum value, and outlier.



Statistical Category Variables

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.boxplot(x="body-style",
y="price", data=df)
```



The price distribution between vehicle body-style categories has significant overlap, so body style cannot be a good predictor of price.

Grouping

- The `groupby` method is used to group data according to different categories.
- The data is grouped by one or several variables and the analysis is performed on individual groups.
- As an example, let's group by the "drive-wheels" variable. We see that there are 3 different categories of drive-wheels.

```
df['drive-wheels'].unique()
```

```
array(['rwd', 'fwd', '4wd'], dtype=object)
```

Grouping

- If we want to know, on average, which type of drive-wheels is the most expensive, we can classify the " drive-wheels" and then average them.
- We can select the 'drive-wheels', 'body-style' and 'price' fields and assign them to the "df_group_one" variable.
- Then we calculate the average price for each of the different data categories.
- It was found that rwd is on average the most expensive, while 4wd and fwd cost approximately the same.

```
df_group_one = df[['drive-
wheels','body-style','price’]]

# grouping results
df_group_one = df_group_one.
groupby(['drive-wheels’],
as_index=False).mean()

df_group_one
```

	drive-wheels	price
0	4wd	10241.000000
1	fwd	9244.779661
2	rwd	19757.613333

Grouping

- We can also **group by multiple variables**.
- For example, let's group by combination of ‘drive-wheels' and ' body-style’.
- We can store the result in 'grouped_test1' variable.

```
df_gptest = df[['drive-wheels’,
'body-style','price’]]

grouped_test1 = df_gptest.groupby
(['drive-wheels','body-style’],
as_index=False).mean()

grouped_test1
```

	drive-wheels	body-style	price
0	4wd	hatchback	7603.000000
1	4wd	sedan	12647.333333
2	4wd	wagon	9095.750000
3	fwd	convertible	11595.000000
4	fwd	hardtop	8249.000000
5	fwd	hatchback	8396.387755
6	fwd	sedan	9811.800000
7	fwd	wagon	9997.333333
8	rwd	convertible	23949.600000
9	rwd	hardtop	24202.714286
10	rwd	hatchback	14337.777778
11	rwd	sedan	21711.833333
12	rwd	wagon	16994.222222

Grouping

- The grouped data is much easier to visualize when it is built into a **pivot table**.
- We can convert a DataFrame to a pivot table using the `pivot` method to create a pivot table from groups.
- If there are several pivot cells that are empty, we can fill those values with 0 or other values.

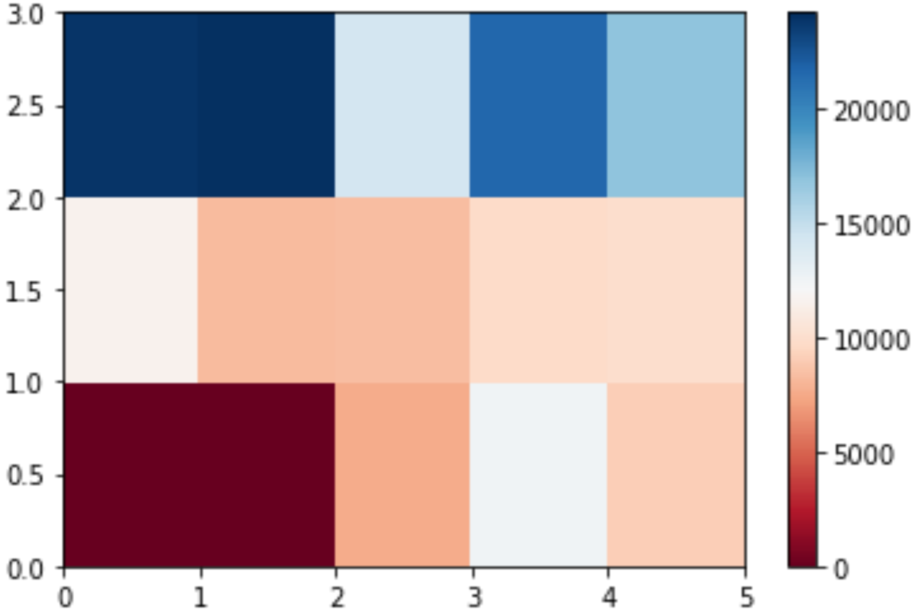
```
grouped_pivot = grouped_test1.pivot(
index='drive-wheels', columns='body-
style')

#fill missing values with 0
grouped_pivot = grouped_pivot.fillna(0)

grouped_pivot
```

	price				
body-style	convertible	hardtop	hatchback	sedan	wagon
drive-wheels					
4wd	0.0	0.000000	7603.000000	12647.333333	9095.750000
fwd	11595.0	8249.000000	8396.387755	9811.800000	9997.333333
rwd	23949.6	24202.714286	14337.777778	21711.833333	16994.222222

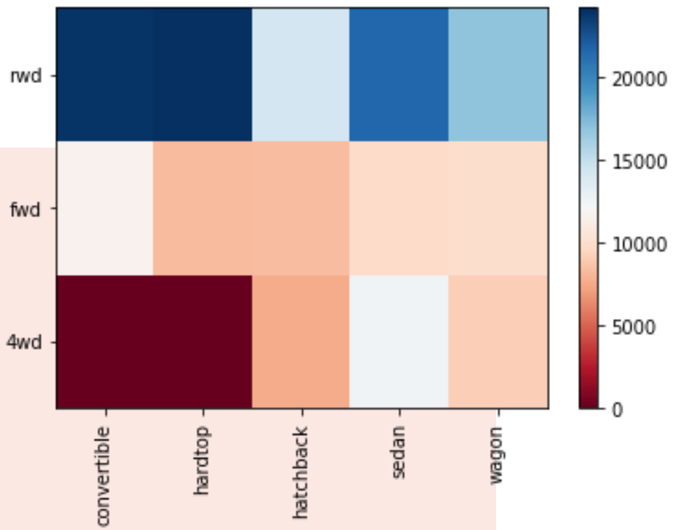
Grouping

- We can visualize the results of the pivot in the form of a **heatmap**.
- The heatmap plots the target variable (price) with the ‘drive-wheels' and 'body-style' variables on the vertical and horizontal axes. This allows us to visualize how price is related to 'wheel-drive' and 'body-style'.
- The default label doesn't tell us enough information yet. We need change the label on the heatmap so that it has legend information.

```
import matplotlib.pyplot as plt

#use the grouped results
plt.pcolor(grouped_pivot, cmap='RdBu')
plt.colorbar()
plt.show()
```



Grouping




```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
im = ax.pcolor(grouped_pivot, cmap='RdBu')

#label names
row_labels = grouped_pivot.columns.levels[1]
col_labels = grouped_pivot.index
#move ticks and labels to the center
ax.set_xticks(np.arange(grouped_pivot.shape[1]) + 0.5, minor=False)
ax.set_yticks(np.arange(grouped_pivot.shape[0]) + 0.5, minor=False)
#insert labels
ax.set_xticklabels(row_labels, minor=False)
ax.set_yticklabels(col_labels, minor=False)
#rotate label if too long
plt.xticks(rotation=90)

fig.colorbar(im)
plt.show()
```

Analysis of Varians (ANOVA)

- **Analysis of Variance (ANOVA**) is a statistical method used to test whether there is a significant difference between the averages of two or more groups. ANOVA returns two parameters:
  - F-Score: ANOVA assumes the average of all groups is the same, ANOVA calculates how far the actual average deviates from the assumption and reports it as an F-Score. A bigger score means there is a bigger difference between the means.
  - P-Value: The P-Value indicates how statistically significant the calculated score is.
- ANOVA analyzes the differences between different groups of the same variable, the `groupby` function is useful in the case of ANOVA.

Analysis of Varians (ANOVA)

- If the price variable in the car dataset is highly correlated with other variables, ANOVA will return a fairly large F-Score and a small p-value.
- Let's see if the type of 'drive-wheels' affects the 'price'.
- We can get value by using the `get_group` method.
- We can use the `f_oneway` function in stats module to get F-Score and P-Value.

```
grouped_test2=df_gptest[['drive-wheels', 'price']].groupby(['drive-wheels’])
grouped_test2.get_group('4wd’)['price’]

# ANOVA
f_val, p_val = stats.f_oneway(grouped_test2.get_group('fwd')['price'], grouped_
test2.get_group('rwd')['price'], grouped_test2.get_group('4wd')['price'])
print( "ANOVA results: F=", f_val, ", P =", p_val)
```

```
ANOVA results: F= 67.95406500780399 , P = 3.3945443577151245e-23
```

The ANOVA results are good, with a large F-Score indicating a strong correlation and a P value of close to 0 implying almost certain statistical significance.

THANK YOU