WEEK 2

LINE PLOT

The most basic plot is the line plot.

SCATTER PLOT

When you have a time scale along the horizontal axis, the line plot is your friend. But in many other cases, when you're trying to assess if there's a correlation between two variables, for example, the scatter plot is the better choice. Below is an example of how to build a scatter plot.

HISTOGRAM PLOT

To explore data set and to get idea about distribution. Too few bins will oversimplify reality and won't show you the details. Too many bins will overcomplicate reality and won't show the bigger picture.

Tips : Data Visualization

Many options – different plot types, many customizations

Choices depend on – data, what you want to tell

LABELS

```
# Basic scatter plot, log scale
plt.scatter(gdp_cap, life_exp)
plt.xscale('log')
# Strings
xlab = 'GDP per Capita [in USD]'
ylab = 'Life Expectancy [in years]'
title = 'World Development in 2007'
# Add axis labels
plt.xlabel(xlab)
```

TICKS

can control the y-ticks by specifying two arguments: plt.yticks([0,1,2], [“one”,”two”,”three”]). The ticks corresponding to the numbers 0, 1 and 2 will be replaced by one, two and three, respectively. Let's do a similar thing for the x-axis of our world development chart, with the xticks() function. The tick values 1000, 10000 and 100000 should be replaced by 1k, 10k and 100k

SIZES

```
# Import numpy as np
import numpy as np
# Store pop as a numpy array: np_pop
np_pop = np.array(pop)
# Double np_pop
np_pop = np_pop * 2
# Update: set s argument to np_pop
plt.scatter(gdp_cap, life_exp, s = np_pop)
# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])
# Display the plot
plt.show()
```

COLORS

```
# Scatter plot
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c = col, alpha = 0.8)
# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])
# Additional customizations
plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')
# Add grid() call
plt.grid(b)
# Show the plot
plt.show()
```

FIRST START WITH PANDAS

Assume that we want to select countries with an area over 8 sqr.km. Three simple steps ar

1. Select the area column.
   a. Different ways – try
      i. brics['area']
      ii. brics.loc[:,'area']
      iii. brics.iloc[:,2]
2. Do comparison on area column.
   a. Use comparison operator and apply to selected column. Try
      i. brics['area'] > 8, this will give you Boolean results.
3. Use result to select countries.
   a. Apply the comparison results to dataframe.
      i. Either assign the comparison result to a new variable first and app dataframe, i.e., result = brics['area'] > 8 then brics[result]
      ii. Apply the comparison directly to the dataframe i.e., brics[brics['area

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
RU	Russia	Moscow	17.100	143.5
CH	China	Beijing	9.597	1357.0

CHOOSING A VALUE IN BETWEEN

```
import numpy as np
area810 = np.logical_and(brics['area']>8,brics['area']<10)
brics[area810]
```

	country	capital	area	population
BR	Brazil	Brasilia	8.516	200.4
CH	China	Beijing	9.597	1357.0

LOOP OVER DATAFRAME

Iterating over a Pandas DF is typically done with iterrows() method. Use for loop for everyobservation is iterated over every iteration the row label and actual row contents,

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)
# Adapt for loop
for lab,row in cars.iterrows():
    print(lab +": "+ str(row['cars_per_cap']))
```

ADDING COLUMN TO DATAFRAME

```
# Import cars data
import pandas as pd
cars = pd.read_csv('cars.csv', index_col = 0)
# Code for loop that adds COUNTRY column
for lab,row in cars.iterrows():
    cars.loc[lab, "COUNTRY"] = row["country"].upper()
# Print cars
print(cars)
```

WEEK3

PICKLE(.p or .pkl)

Pickle is a serialized way of storing a Pandas dataframe. Basically, you are writing down the exact representation of the dataframe to disk. This means the types of the columns are and the indices are the same. Download taxi_owners.p and load the data using pd.read_pickle()

Taxi_owner =

pd.read_pickle(“Taxi_owners.p’)

Taxi_owner.head()

There are several ways to store data for analysis,but rectangular data, sometimes called "tabular data" is the most common form. In this example, with dogs, each observation, or each dog, is a row, and each variable, or each dog property, is a column.

Exploring a DataFrame can be done by many methods such as .head(), .info(). The .info() method displays the names of columns, the data types they contain, and whether they have any missing values.

df.describe()	The describe method computes some summary statistics numerical columns, like mean and median. "count" is the non-missing values in each column. describe is good for overview of numeric variables
df.shape	The shape attribute contains a tuple that holds the numbe followed by the number of columns.
df.values	The values attribute contains the data values in a 2-dimer NumPy array.
df.columns	The columns attribute contains column names.
df.index	The index attribute contains row numbers or row names.

SORTING AND SUBNETTING(FILTERING)

```
dogs.sort_values("weight_kg")
```

	name	breed	color	height_cm	weight_kg	date_of_birth
5	Stella	Chihuahua	Tan	18	2	2015-04-20
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

Setting the ascending argument to False will sort the data the other way around, from heaviest dog to lightest dog.

SORTING IN DESCENDING ORDER

```
dogs.sort_values("weight_kg", ascending=False)
```

	name	breed	color	height_cm	weight_kg	date_of_birth
6	Bernie	St. Bernard	White	77	74	2018-02-27
4	Max	Labrador	Black	59	29	2017-01-20
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
5	Stella	Chihuahua	Tan	18	2	2015-04-20

We can sort with multiple variables. You need to pass a list of variables as an argument.

SORTIN BY MULTIPLE VARIABLES

```
dogs.sort_values(["weight_kg", "height_cm"])
```

	name	breed	color	height_cm	weight_kg	date_of_birth
5	Stella	Chihuahua	Tan	18	2	2015-04-20
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
0	Bella	Labrador	Brown	56	24	2013-07-01
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

To change the direction values are sorted in, pass a list to the ascending argument to specify which direction sorting should be done for each variable.

SORTING BY MULTIPLE VARIABLES

```
dogs.sort_values(["weight_kg", "height_cm"], ascending=[True, False])
```

	name	breed	color	height_cm	weight_kg	date_of_birth
5	Stella	Chihuahua	Tan	18	2	2015-04-20
3	Cooper	Schnauzer	Gray	49	17	2011-12-11
0	Bella	Labrador	Brown	56	24	2013-07-01
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
1	Charlie	Poodle	Black	43	24	2016-09-16
4	Max	Labrador	Black	59	29	2017-01-20
6	Bernie	St. Bernard	White	77	74	2018-02-27

Selecting columns or subnetting columns can be done as follows

SUBSETTING MULTIPLE COLUMNS

```
dogs[["breed", "height_cm"]]
```

	breed	height_cm
0	Labrador	56
1	Poodle	43
2	Chow Chow	46
3	Schnauzer	49
4	Labrador	59
5	Chihuahua	18
6	St. Bernard	77

```
cols_to_subset = ["breed", "height_cm"]
dogs[cols_to_subset]
```

	breed	height_cm
0	Labrador	56
1	Poodle	43
2	Chow Chow	46
3	Schnauzer	49
4	Labrador	59
5	Chihuahua	18
6	St. Bernard	77

SUBSETTING BASED ON TEXT DATA

```
dogs[dogs["breed"] == "Labrador"]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
4	Max	Labrador	Black	59	29	2017-01-20

SUBSETTING BASED ON DATES

```
dogs[dogs["date_of_birth"] < "2015-01-01"]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
3	Cooper	Schnauzer	Gray	49	17	2011-12-11

SUBSETTING BASED ON MULTIPLE CONDITIONS

```
is_lab = dogs["breed"] == "Labrador"
is_brown = dogs["color"] == "Brown"
dogs[is_lab & is_brown]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01

SUBSETTING using .isin()

```
is_black_or_brown = dogs["color"].isin(["Black", "Brown"])
dogs[is_black_or_brown]
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	24	2013-07-01
1	Charlie	Poodle	Black	43	24	2016-09-16
2	Lucy	Chow Chow	Brown	46	24	2014-08-25
4	Max	Labrador	Black	59	29	2017-01-20

ADDING A NEW COLUMN

```
dogs["height_m"] = dogs["height_cm"] / 100
print(dogs)
```

	name	breed	color	height_cm	weight_kg	date_of_birth	height_m
0	Bella	Labrador	Brown	56	24	2013-07-01	0.56
1	Charlie	Poodle	Black	43	24	2016-09-16	0.43
2	Lucy	Chow Chow	Brown	46	24	2014-08-25	0.46
3	Cooper	Schnauzer	Gray	49	17	2011-12-11	0.49
4	Max	Labrador	Black	59	29	2017-01-20	0.59
5	Stella	Chihuahua	Tan	18	2	2015-04-20	0.18
6	Bernie	St. Bernard	White	77	74	2018-02-27	0.77

DOGGY MASS INDEX

BMI = WEIGT IN KG/(HEIGHT IN M)^2

```
dogs["bmi"] = dogs["weight_kg"] / dogs["height_m"] ** 2
print(dogs.head())
```

	name	breed	color	height_cm	weight_kg	date_of_birth	height_m	bmi
0	Bella	Labrador	Brown	56	24	2013-07-01	0.56	76.530612
1	Charlie	Poodle	Black	43	24	2016-09-16	0.43	129.799892
2	Lucy	Chow Chow	Brown	46	24	2014-08-25	0.46	113.421550
3	Cooper	Schnauzer	Gray	49	17	2011-12-11	0.49	70.803832
4	Max	Labrador	Black	59	29	2017-01-20	0.59	83.399394

SUMMARIZING NUMERICAL DATA

```
dogs["height_cm"].mean()
```

```
49.714285714285715
```

- .median() , .mode()
- .min() , .max()
- .var() , .std()
- .sum()

SUMMARIZING DATES

Oldest dog:

```
dogs["date_of_birth"].min()
```

```
'2011-12-11'
```

Youngest dog:

```
dogs["date_of_birth"].max()
```

```
'2018-02-27'
```

THE .AGG() METHOD

```
def pct30(column):
    return column.quantile(0.3)
```

```
dogs["weight_kg"].agg(pct30)
```

```
22.599999999999998
```

SUMMARIES ON MULTIPLE COLUMNS

```
dogs[["weight_kg", "height_cm"]].agg(pct30)
```

```
weight_kg    22.6
height_cm    45.4
dtype: float64
```

MULTIPLE SUMMARIES

```
def pct40(column):
    return column.quantile(0.4)
```

```
dogs["weight_kg"].agg([pct30, pct40])
```

```
pct30    22.6
pct40    24.0
Name: weight_kg, dtype: float64
```

CUMULATIVE SUM

```
dogs["weight_kg"]
```

```
0    24
1    24
2    24
3    17
4    29
5     2
6    74
Name: weight_kg, dtype: int64
```

```
dogs["weight_kg"].cumsum()
```

```
0    24
1    48
2    72
3    89
4    118
5    120
6    194
Name: weight_kg, dtype: int64
```

DROPPING DUPLICATES

It is possible that the data set contains duplicated values and sometimes we do not want to have them in the analysis.

```
vet_visits.drop_duplicates(subset="name")
```

	date	name	breed	weight_kg
0	2018-09-02	Bella	Labrador	24.87
1	2019-06-07	Max	Chow Chow	24.01
2	2019-03-19	Charlie	Poodle	24.95
3	2018-01-17	Stella	Chihuahua	1.51
4	2019-10-19	Lucy	Chow Chow	24.07
7	2019-03-30	Cooper	Schnauzer	16.91
10	2019-01-04	Bernie	St. Bernard	74.98

DROPPING DUPLICATE PAIRS

```
unique_dogs = vet_visits.drop_duplicates(subset=["name", "breed"])
print(unique_dogs)
```

	date	name	breed	weight_kg
0	2018-09-02	Bella	Labrador	24.87
1	2019-03-13	Max	Chow Chow	24.13
2	2019-03-19	Charlie	Poodle	24.95
3	2018-01-17	Stella	Chihuahua	1.51
4	2019-10-19	Lucy	Chow Chow	24.07
6	2019-06-07	Max	Labrador	28.35
7	2019-03-30	Cooper	Schnauzer	16.91
10	2019-01-04	Bernie	St. Bernard	74.98

After dropping duplicates, you may want to count the values,.

Easy as 1, 2, 3

```
unique_dogs["breed"].value_counts()
```

```
Labrador      2
Schnauzer     1
St. Bernard   1
Chow Chow     2
Poodle        1
Chihuahua     1
Name: breed, dtype: int64
```

```
unique_dogs["breed"].value_counts(sort=True)
```

```
Labrador      2
Chow Chow     2
Schnauzer     1
St. Bernard   1
Poodle        1
Chihuahua     1
Name: breed, dtype: int64
```

Normalization helps us to understand data better. The normalize argument can be used to turn the counts into proportions of the total. 25% of the dogs that go to this vet are Labradors.

```
unique_dogs["breed"].value_counts(normalize=True)
```

```
Labrador      0.250
Chow Chow     0.250
Schnauzer     0.125
St. Bernard   0.125
Poodle        0.125
Chihuahua     0.125
Name: breed, dtype: float64
```

SUMMARIES BY GROUP

```
dogs[dogs["color"] == "Black"]["weight_kg"].mean()
dogs[dogs["color"] == "Brown"]["weight_kg"].mean()
dogs[dogs["color"] == "White"]["weight_kg"].mean()
dogs[dogs["color"] == "Gray"]["weight_kg"].mean()
dogs[dogs["color"] == "Tan"]["weight_kg"].mean()
```

```
26.0
24.0
74.0
17.0
2.0
```

GROUPED SUMMARIES

```
dogs.groupby("color")["weight_kg"].mean()
```

```
color
Black    26.5
Brown    24.0
Gray     17.0
Tan       2.0
White    74.0
Name: weight_kg, dtype: float64
```

MULTIPLE GROUPED SUMMARIES

```
dogs.groupby("color")["weight_kg"].agg([min, max, sum])
```

color	min	max	sum
Black	24	29	53
Brown	24	24	48
Gray	17	17	17
Tan	2	2	2
White	74	74	74

GROUPING BY MULTIPLE VARIABLES

```
dogs.groupby(["color", "breed"])["weight_kg"].mean()
```

```
color  breed
Black  Chow Chow      25
       Labrador       29
       Poodle         24
Brown  Chow Chow      24
       Labrador       24
Gray   Schnauzer      17
Tan    Chihuahua       2
White  St. Bernard    74
Name: weight_kg, dtype: int64
```

MANY GROUPS, MANY SUMMARIES

```
dogs.groupby(["color", "breed"])[["weight_kg", "height_cm"]].mean()
```

color	breed	weight_kg	height_cm
Black	Labrador	29	59
	Poodle	24	43
Brown	Chow Chow	24	46
	Labrador	24	56
Gray	Schnauzer	17	49
Tan	Chihuahua	2	18
White	St. Bernard	74	77

GROUP BY A PIVOT TABLE

```
dogs.groupby("color")["weight_kg"].mean()
```

```
color
Black    26
Brown    24
Gray     17
Tan       2
White    74
Name: weight_kg, dtype: int64
```

```
dogs.pivot_table(values="weight_kg",
                 index="color")
```

color	weight_kg
Black	26.5
Brown	24.0
Gray	17.0
Tan	2.0
White	74.0

DIFFERENT STATISTICS

```
import numpy as np
dogs.pivot_table(values="weight_kg", index="color", aggfunc=np.median)
```

color	weight_kg
Black	26.5
Brown	24.0
Gray	17.0
Tan	2.0
White	74.0

MULTIPLE STATISTICS

```
dogs.pivot_table(values="weight_kg", index="color", aggfunc=[np.mean, np.median])
```

	mean	median
	weight_kg	weight_kg
color		
Black	26.5	26.5
Brown	24.0	24.0
Gray	17.0	17.0
Tan	2.0	2.0
White	74.0	74.0

PIVOT ON TWO VARIABLES

```
dogs.groupby(["color", "breed"])["weight_kg"].mean()
```

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed")
```

breed	Chihuahua	Chow Chow	Labrador	Poodle	Schnauzer	St. Bernard
color						
Black	NaN	NaN	29.0	24.0	NaN	NaN
Brown	NaN	24.0	24.0	NaN	NaN	NaN
Gray	NaN	NaN	NaN	NaN	17.0	NaN
Tan	2.0	NaN	NaN	NaN	NaN	NaN
White	NaN	NaN	NaN	NaN	NaN	74.0

You also previously computed the mean weight grouped by two variables: color and breed. We can also do this using the pivot_table method. To group by two variables, we can pass a second variable name into the columns argument.

FILLING MISSING VALUES IN PIVOT TABLES

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed", fill_value=0)
```

breed	Chihuahua	Chow Chow	Labrador	Poodle	Schnauzer	St. Bernard
color						
Black	0	0	29	24	0	0
Brown	0	24	24	0	0	0
Gray	0	0	0	0	17	0
Tan	2	0	0	0	0	0
White	0	0	0	0	0	74

If we set the margins argument to True, the last row and last column of the pivot table contain the mean of all the values in the column or row, not including the missing values that were filled in with 0s. For example, in the last row of the Labrador column, we can see that the mean weight of the Labradors is 26 kilograms. In the last column of the Brown row, the mean weight of the Brown dogs is 24 kilograms. The value in the bottom right, in the last row and last column, is the mean weight of all the dogs in the dataset. Using margins equals True allows us to see a summary statistic for multiple levels of the dataset: the entire dataset, grouped by one variable, by another variable, and by two variables.

SUMMING WITH PIVOT TABLES

```
dogs.pivot_table(values="weight_kg", index="color", columns="breed",
                 fill_value=0, margins=True)
```

breed	Chihuahua	Chow Chow	Labrador	Poodle	Schnauzer	St. Bernard	All
color							
Black	0	0	29	24	0	0	26.500000
Brown	0	24	24	0	0	0	24.000000
Gray	0	0	0	0	17	0	17.000000
Tan	2	0	0	0	0	0	2.000000
White	0	0	0	0	0	74	74.000000
All	2	24	26	24	17	74	27.714286

MULTI-LEVEL INDEXES aka HIERACHICAL INDEXES

```
dogs_ind3 = dogs.set_index(["breed", "color"])
print(dogs_ind3)
```

		name	height_cm	weight_kg
breed	color			
Labrador	Brown	Bella	56	25
Poodle	Black	Charlie	43	23
Chow Chow	Brown	Lucy	46	22
Schnauzer	Grey	Cooper	49	17
Labrador	Black	Max	59	29
Chihuahua	Tan	Stella	18	2
St. Bernard	White	Bernie	77	74

SUBSET THE OUTER LEVEL WITH A LIST

```
dogs_ind3.loc[["Labrador", "Chihuahua"]]
```

		name	height_cm	weight_kg
breed	color			
Labrador	Brown	Bella	56	25
	Black	Max	59	29
Chihuahua	Tan	Stella	18	2

SUBSET INNER LEVELS WITH A LIST OF TUPLES

```
dogs_ind3.loc[[("Labrador", "Brown"), ("Chihuahua", "Tan")]]
```

		name	height_cm	weight_kg
breed	color			
Labrador	Brown	Bella	56	25
Chihuahua	Tan	Stella	18	2

WHAT'S A MISSING VALUE?

In a pandas DataFrame, missing values are indicated with N-a-N, which stands for "not a number."

DETECTING MISSING VALUES

```
dogs.isna()
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	False	False	False	False	True	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	True	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False

If we chain dot-isna with dot-any, we get one value for each variable that tells us if there are any missing values in that column.

DECTECTING ANY MISSING VALUES

```
dogs.isna().any()
```

```
name             False
breed            False
color            False
height_cm        False
weight_kg         True
date_of_birth    False
dtype: bool
```

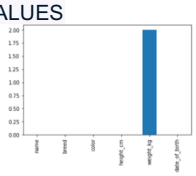
COUNTING MISSING VALUES

```
dogs.isna().sum()
```

```
name             0
breed            0
color            0
height_cm        0
weight_kg        2
date_of_birth    0
dtype: int64
```

PLOTTING MISSING VALUES

```
import matplotlib.pyplot as plt
dogs.isna().sum().plot(kind="bar")
plt.show()
```



One option is to remove the rows in the DataFrame that contain missing values. This can be done using the dropna method. However, this may not be ideal if you have a lot of missing data, since that means losing a lot of observations.

REMOVING MISSING VALUES

```
dogs.dropna()
```

	name	breed	color	height_cm	weight_kg	date_of_birth
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6	Bernie	St. Bernard	White	77	74.0	2018-02-27

REPLACING MISSING VALUES

```
dogs.fillna(0)
```

	name	breed	color	height_cm	weight_kg	date_of_birth
0	Bella	Labrador	Brown	56	0.0	2013-07-01
1	Charlie	Poodle	Black	43	24.0	2016-09-16
2	Lucy	Chow Chow	Brown	46	24.0	2014-08-25
3	Cooper	Schnauzer	Gray	49	0.0	2011-12-11
4	Max	Labrador	Black	59	29.0	2017-01-20
5	Stella	Chihuahua	Tan	18	2.0	2015-04-20
6	Bernie	St. Bernard	White	77	74.0	2018-02-27

WEEK 4

Exploratory Data Analysis, or EDA for short, is the process of cleaning and reviewing data to derive insights such as descriptive statistics and correlation and generate hypotheses for experiments.EDA results often inform the next steps for the dataset, whether that be generating hypotheses, preparing the data for use in a machine learning model, or even throwing the data out and gathering new data!

PROCESS of reviewing and cleaning data to 1.Derive insights and 2.Generate Hypothesis

Can explore

using .head(), .info(), .describe(), .shape()

.VALUE COUNTS() let you select a specific column to get a close look at the information. Is not only explore initial statistics summary but also effect to visualize the data(SEABORN)

DATA VALIDATION

Is an import early step in EDA. We want to understand whether data types and ranges are as expected before we progress too far in our analysis.

Use .info() or .dtypes

UPDATING DATATYPES

Use .astype() function allow us to change data types.(books["year].astype(int))

VALADING CATEGORICAL DATA

can do by comparing values in a column to a list of expected values using .isin() Let say a "genre" column has value only between finction and non-fction. This can be apply to Series. When apply we get True or False for each row.

WORKING WITH NUMERICAL DATA

.select_dtypes allow you to only view numerical colunms in DF by callig it and passing "number" as the argument.

VALIDATING NUMBERICAL DATA

can view more detailed picture of the distribution of year data using Seaborn's boxplot function. Example it shows the boundaries of each quartile of year data. In picture, the box's start is 25th and end is 75th percentiles.the box middle is median.

Also possible to view it passing in y value. As the year data grouped by a categorical variable.

EXPLORING GROUPS OF DATA

Explore the characteristic of subsets of data further with the help of the .groupby() function, which groups data by a given category. Aggregating function indicates how to summarize grouped data. .

AGGREGATING UNGROUPED DATA

agg() function aggregates data cross all rows in a given column and is typically used when we want to apply more than one function.. It applies this function across a DataFrame.

SPECIFYING AGGREGATIONS FOR COLUMNS

Use dictionary to apply aggr() function in specific column.

NAMED SUMMARY COLUMNS

Combining .agg() and .groupby() ,can apply these new exploration skills to grouped data. Like showing the mean and standard deviation for rating of each book genre along with the median year. Can also create column name

VISUALIZING CATEGORICAL SUMMARIES

Can display simi8lar information visually using a barplot. Barplot will automatically caculate the mean of a quantitative variable like rating across grouped categorical data. In seaborn, barplot show a 95% confidence interval for the mean as a vertical line on the top of each bar.

NAMED AGGREGATION

Helpful to name new columns when aggregating so that it's clear in the code output what aggregations are being applied.

VISUALIZING CAATEGORICAL SUMMARIES

Seaborn has many great visualizing including bar plot for displaying an aggregated average value of category of data.

Bar plot include a 95% conficdence interval for the categorical mean which is vertical bar. Since CI are caculated using botht the number of values and the variability of those values, they give a helpful indication of how much data can be relied upon.

WHY IS MISSING DATA A PROBLEM?

Affects distributions which is missing heights of taller students

Less representative of the population which is certain groups of disproportionately and represented eg. lacking data on oldest students

Can result in drawing incorrect conclusions

CHECKING FOR MISSING VALUES

Print(salaries.isna().sum())

STRATEGIES FOR ADDRESSING MISSING DATA

Drop missing values which is 5% or less of total values

Impute mean, median mode which is depends on distribution and context

Impute by sub-group which is different experience levels have different median salalry

DROPPING MISSING VALUES

To calculate our missing values threshold we multiply the length of our DataFrame by five percent, giving us an upper limit of 30 as shown in the example below.

Threshold = len(salaries) * 0.05

Use Boolean indexing to filter for columns with missing valuie less tha or equal to this threshold storing them as a variable called cols to drop. We drop missing valiues by calling .dropna(), passing cols_to_drop to the subset argument. We set inplace to True so the DataFrame is updated.

IMPUTING A SUMMARY STATISTIC

We then filter for the remaining columns with missing values, giving us four columns. To impute the mode for the first three columns, we loop through them and call the .fillna() method, passing the respective column's mode and indexing the first item, which contains the mode, in square brackets.

CHECKING THE REMAINING MISSING VALUES

Checking for missing values again, we see salary_USD is now the only column with missing values and the volume has changed from 60 missing values to 41. This is because some rows may have contained missing values for our subset columns as well as salary, so they were dropped.

IMPUTING BY SUB-GROUP

We'll impute median salary by experience level by grouping salaries by experience and calculating the median. We use the .to_dict() method, storing the grouped data as a dictionary. Printing the dictionary returns the median salary for each experience level, with executives earning the big bucks!

We then impute using the .fillna() method, providing the Experience column and calling the .map() method, inside which we pass the salaries dictionary.

WORKING WITH CATEGORICAL DATA

Now let's explore how to create and analyze categorical data. Recall that we can use the select_dtypes method to filter any non-numeric data. Chaining .head() allows us to preview these columns in our salaries DataFrame, showing columns such as Designation,Experience, Employment_Status, and Company_Size. You can follow along by loading data from ds_salaries_clean.csv to salaries DataFrame. We can count how many unique job titles there are using pandas .nunique() method. There are 50 in total! If we plot the graph using a bar chart, the fifth most popular job title, Research Scientist, appears less than 20 times. The current format of the data limits our ability to generate insights. We can use the pandas.Series.string.contains() method, which allows us to search a column for a specific string or multiple strings. Say we want to know which job titles have Scientist in them. We use the str.contains() method on the Designation column, passing the word Scientist. This returns True or False values depending on whether the row contains this word.

WHAT IF WE WANT TO FILTER FOR ROWS CONTAINING ONE OR MORE PHRASES?

Now we have a sense of how this method works, let's define a list of job titles we want to find. We start by creating a list with the different categories of data roles, which will become the values of a new column in our DataFrame.

We then need to create variables containing our filters. We will look for Data Scientist or NLP for data science roles. We'll use Analyst or Analytics for data analyst roles. We repeat this for data engineer, machine learning engineer, managerial, and consultant roles.

WORKING WITH NUMERICAL DATA

Time to switch our focus on to working with numeric data. To obtain Salary in USD we'll need to perform a few tasks. First, we need

to remove the commas from the values in the Salary_In_Rupees column. Next, we change the data type to float. Lastly, we'll make a new column by converting the currency.

CONVERTING STRINGS TO NUMBERS

```
pd.Series.str.replace("characters to remove", "characters to replace them with")
```

```
salaries["Salary_In_Rupees"] = salaries["Salary_In_Rupees"].str.replace(",", "")
print(salary["Salary_In_Rupees"].head())
```

```
1    20688070.00
2     8674985.00
3     1591390.00
4    11935425.00
5     5729004.00
Name: Salary_In_Rupees, dtype: object
```

```
salaries["Salary_In_Rupees"] = salaries["Salary_In_Rupees"].astype(float)
```

• 1 Indian Rupee = 0.012 US Dollars

```
salaries["Salary_USD"] = salaries["Salary_In_Rupees"] * 0.012
```

PREVIEWING THE NEW COLUMN

```
print(salaries[["Salary_In_Rupees", "Salary_USD"]].head())
```

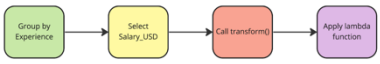
```
   Salary_In_Rupees  Salary_USD
0        20688070.0  248256.840
1         8674985.0  104099.820
2         1591390.0   19096.680
3        11935425.0  143225.100
4         5729004.0   68748.048
```

Adding summary statistics into a DataFrame

```
salaries.groupby("Company_Size")["Salary_USD"].mean()
```

```
Company_Size
L    111934.432174
M    110706.628527
S     69880.980179
Name: Salary_USD, dtype: float64
```

Adding summary statistics into a DataFrame




```
salaries["std_dev"] = salaries.groupby("Experience")["Salary_USD"].transform(lambda x: x.std())
```

Adding summary statistics into a DataFrame

```
print(salaries[["Experience", "std_dev"]].value_counts())
```

```
Experience  std_dev
SE          52995.385395    257
MI          63217.397343    197
EN          43367.256303     83
EX          86426.611619     24
```

Adding summary statistics into a DataFrame

```
salaries["median_by_comp_size"] = salaries.groupby("Company_Size") \
                                  ["Salary_USD"].transform(lambda x: x.median())
```

```
print(salaries[["Company_Size", "median_by_comp_size"]].head())
```

```
  Company_Size  median_by_comp_size
0            S             60833.424
1            M            105914.964
2            S             60833.424
3            L             95483.400
4            L             95483.400
```

HANDLING OUTLIERS

An observation far away from other data points.which can be median house price: $400,000 and Outlier house price: $5,000,000 Should consider why the value is different: Location, number of bedrooms, overall size etc.

USING DESCRIPTIVE STATICS

```
print(salaries["Salary_USD"].describe())
```

```
count       518.000
mean     104905.826
std       62660.107
min        3819.000
25%       61191.000
50%       95483.000
75%      137496.000
max      429675.000
Name: Salary_USD, dtype: float64
```

Can define an outlier mathematically, First we need to know the interquartile range or IQR which is the difference between 75th and 25th percentiles.

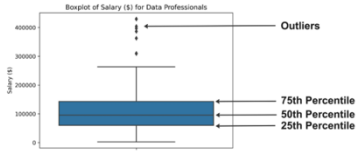
USING THE INTERQUARTILE RANGE

IQR = 75th – 25th percentile

Recall that these percentiles are included in box plots, like this one showing salaries of data professionals.

IQR in box plots

```
sns.boxplot(data=salaries,
            y="Salary_USD")
plt.show()
```



Once we have the IQR, we can find an upper outlier by looking for values above the sum of the 75th percentile plus 1.5 times the IQR. Lower outliers have values below the sum of the 25th percentile minus 1.5 times the IQR.

USING THE INTERQUARTILE RANGE

Interquartile range(IQR)

Upper Outliers > 75th percentile + (1.5 * IQR)

Lower Outliers < 25th percentile – (1.5 * IQR)

IDENTIFYING THRESHOLDS

```
# 75th percentile
seventy_fifth = salaries["Salary_USD"].quantile(0.75)

# 25th percentile
twenty_fifth = salaries["Salary_USD"].quantile(0.25)

# Interquartile range
salaries_iqr = seventy_fifth - twenty_fifth

print(salaries_iqr)
```

IDENTIFYING OUTLIERS

```
# Upper threshold
upper = seventy_fifth + (1.5 * salaries_iqr)


# Lower threshold
lower = twenty_fifth - (1.5 * salaries_iqr)


print(upper, lower)
```

SUBSETTING OUR DATA

```
salaries[(salaries["Salary_USD"] < lower) | (salaries["Salary_USD"] > upper)] \
        [["Experience", "Employee_Location", "Salary_USD"]]
```

WHY LOOK FOR OUTLIERS?

Exterme values meaning may not accurately represent our data

Can change the mean and standard deviation

Statistical tests and machine learning models need normally distributed data

WHAT TO DO ABOUT OUTLIERS?

Questions to ask:

Why do these outliers exist?

ANS:More senior roles/different countries pay more and consider leavning them in the dataset

Is the Data accurate?

- Could there have been an error in data collection?

ANS: If so remove them

DROPPING OUTLIERS

```
no_outliers = salaries[(salaries["Salary_USD"] > lower) & (salaries["Salary_USD"] < upper)]
```

```
print(no_outliers["Salary_USD"].describe())
```

WEEK 5

parse_dates keyword argument to the CSV import: make time data to be not strings(DateTime data)

dtype: check datatype of imported CSV

pd.to_datetime: converts the argument passed to it to DateTime data. if a DataFrame has month, day, and year data stored in three different columns, as this one does, we can combine these columns into a single DateTime value by passing them to pd.to_datetime.

dtmonth , dt.day , dt.year: can extract only month day or year from datetime column

Line plots are a great way to examine relationships between variables.

In sea born: line plot aggregate y values at each value of x and show the estimated mean and a confidence interval for that estimate. The blue line represents the mean marriage duration for our dataset, while the confidence intervals in the lighter blue shading indicate the area that, with 95% probability, the population mean duration could fall between. The wide confidence intervals suggest that further analysis is needed!

CORRELATION

Describes the direction of the relationship between two variables as well as its strength. Can use variables to predict future outcomes.

corr(): a quickway to see the pairwise correlation of numeric columns in DF..caculate the Pearson correlation coefficient, measuring the linear relationship

Negative correlation indicate as one variable increases , the other decreases

A value closer to zero is indicative of a weak relationship

Value closer to one or negative one indicates stronger relationships

Number varies from -1 to 1,

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

HEATMAP: has the benefit of color coding so that strong positive and negative correlations are easier to spot. “annot = True” labels the correlations coefficient inside each cell.

SCATTERPLOT(SNS): pairplot plots all “pairwise relationships” between numerical varaibles in one visualization. It display the distribution of each variable’s observation. Useful for quick overview of relationshjips within dataset. But can be difficult to interpret , with big datasets which lead to very small plot lablels like the one we see here

Vars argument: can limit the number of plotted relationships by setting it equal to the variables of interest. This visual reassures us that what our correlation coefficients told us was true: variables representing the income of each partner as well as the marriage duration variable all have fairly weak relationships with each other. We also notice in the lower right plot that the distribution of marriage durations includes many shorter marriages and fewer longer marriages.

Categorical relationships

Value_counts() : check categorical varaibles Are hard to summarize numerically so rely on visualizations to explore relationship(Use HISTOGRAM).While setting x to something with hist, layer the information we want by setting "hue". The relationship between each level isn't super clear buz they are stacking on top of each other.

Seaborn KDE Plots and cumulative plot

Fixes the above issue. More interpretable, especially when multiple distributions are shown. More indentifialbe. BUT due to smoothing algorithm used in KDE plots , the curve can indlucde values that don't make sense, so important to set good smoothing parameters.Use the word "cut" in argument to tell seaborn how far past minimum and maximum data values the curve should go when smoothing is applied.
For seeing cumulative distribution function, set cumulative keyword argument to True.
The graph describe the probability that x is less than or equal the the value of on the x axis for each level of "hue" argument.

Consideration for Categorical Data

-converting data analysis into action. EDA is performed for a variety of reasons. 1. Like detecting patterns and relationships in data, 2 . generating questions or hypotheses or 3 . preparing data for machine learning models.
One of the most IMPORTANT CONSIDERATION is about the representation of classes . Example , collect data on people's attittudes to marriage. As part of our data collection, we find out the martial status with the classes including single married and divorced.CLASSES = labels. Survey people's attidues towarss marriage is Martial status which is single married and divorced.
CLASS IMBALANCE is where one class occurs more frequently than others. This can bias results, particularly if this class does not occur more frequently in the population.
The VALUE_COUNTS method can show you relative frequencies for each class by setting normalize keyword argument to True.
CROSS TABULATION can also show you class frequency. Enabling us to examine the frequency of combinations of classes.
Pd.crosstab(planes["Source"], planes["Destination"]). Can also caculate media and compare with this.
Pd.crosstab(planes["Source"] , planes["Destination"] , values=planes["Price"] , aggfunc="median")
EXTRACTING MONTHS AND WEEKDAYS AND HOURS(generating new feature)
If for example price vary from week to week or day to day, you xan extract to new variable and look at it. You can also extract hour using .dthour
Becasuse they are numeric we can check the correlation between these new datetime features and other variables with heatmap.
ANOTHER TECNIQUE TO GENERATE NEW FEATURE.(grouping numeric data and label them as classes)
Use descriptive statics to label flights as econmomy, premium, business class, or first class based on prices within specific ranges or bins
The way is to split the price range using quartile. First sore the 25th percentile using the quantile methopd. Get 50th percentile by calling the median. Get the 75th using the quantile method and store the maximum..,(These created are all bins or range). Then create label which will be our descriptive statics variables. Then set bins to all the ranges starting from 0.Then in new variable let say "Price Category" , use pd.cut and pass price column and label and bins
Then use can preview the price categories.
Print(planes["Price", "Price_Category"]].head()
Can plot different categories per arline by passing our new column to the hue argument when calling sns.countplot
How can correlation help your business?
Correlation is widely used in real-life decision making. You will find correlation in Marketing, Finance, Sales, basically we could mention domains endlessly.
A few benefits:
Pattern recognition. In the big data world looking at millions of rows of raw data will not tell you anything about the business. Using existing information for better decision making will be crucial in the future. It can reveal new business opportunities, give insights about existing processes, and help to communicate clearly. Recognizing patterns is one of the main goals of data science and correlation analysis can help with that.
Financial decision making – investment decisions. Diversifying is essential. Investing in negatively correlated sectors can help you mitigate risk.
For example: if the airline industry is negatively correlated with the social media industry, the investor may choose to invest in a social media stock. If a negative event affects one of those industries, the other sector will be a safer place for the money [11]
Projections. If a company finds a positive correlation between two variables and has some predictions on the one variable involved in the correlation then they can try to make predictions on the second variable as well.
For example: Company X finds a positive correlation between the number of tourists in city Y and its sales. A 10% rise in visitors for the coming year is predicted in city Y. Company X can anticipate an increase in sales as well. Of course, when it gets to predictions, one should always consider the above-mentioned correlation-causation issue.

WEEK 1

DATASCIENCE WORKFLOW

Data Collection and Storage - First, we collect data from many sources, such as surveys, web traffic results, geo-tagged social media posts, and financial transactions. Once collected, we store that data in a safe and accessible way.

Data Preparation: This includes "cleaning data" for instance finding missing or duplicate values and converting data into a more organized format

Exploration & Visualization: Then we explore and visualize the cleaned data. This could involve building dashboards to track how the data changes over time or performing comparisons between two sets of data.

Experimentation & Prediction: Finally we run experiments and predictions on the data. For example, this could involve buidlng a system that forecasts temperature change or performing a test to find which web page acquires more customers. Another example is to create a model predict fraudulent activities

EXAMPLES OF APPLICATIONS OF DATA SCIENCE

TRADITIONAL MACHINE LEARNING

IOT

DEEP LEARNING

CASE STUDY: FRAUD DETECTION

Suppose you work in fraud detection at a large bank. You'd like to use data to determine the probability that the transaction is fake. To answer this question, you might start by gathering information about each purchase, such as the amount, date, location, purchase type, and card-holder's address. You'll need many examples of transactions, including this information, as well as a label that tells you whether each transaction is valid or fraudulent. Luckily, you probably have this information in a database. These record are called "training data" and are used to build an algorithm(a model).Each time a new transaction occurs, you'll give your algorithms information, like amount and data, and it will answer the original question: What is the probability that this transaction is fraudulent?

What do we need for machine learning?

A well-defined question – What is the probability that this transaction is fraudulent?
A set of example data – Old transactions labeled as fraudulent or valid.
A new set of data to use our algorithm on – New credit card transactions.

CASE STUDY: IMAGE RECOGNITION

We could express the picture as a matric of numbers where each number represents a pixel. However this approach would probably fail if we fed the matrix into a traditional machine learning model. There's simply too much input data! We need more advanced algorithms from a subfield of machine learning called deep learning. In deep learning, multiple layers of mini-algorithms called "neurons" , work together to draw complex conclusions. Deep learning takes much more training data than a traditional machine learning model but is also able to learn relationships that traditional model cannot. Deep learning is used to solve data-intensive problems, such as image classification or language understanding.

Deep Learning

Many neurons work together. Require much more training data. Used in complex problems(image Classfication, Language Learning/Understanding)

CASE STUDY: SMART WATCH

Now suppose you're trying to build a smart watch to monitor physical activity. You want to be able to auto-detect different activities such as walking or running. Your smart watch is equipped with a special sensor called an "accelerometer" that monitors motion in three dimensions. The data generated by sensor is the basis of your machine learning problem. You could ask several volunteers to wear your watch and record when they are running or walking. You could then develop an algorithm that recognizes accelerometer data as representing one of those two states: walking or running
Your smart watch is part of a fast growing field called "the Internet of Things" aka IOT which is often combined with Data Science
IOT refers to gadgets that are not standatd computers but still can transmit data. This includes smart watches, internet connected home securioty systems, electronic toll collection system, building energy management systems and etc…
IOT data is great resource for data science project.

ROLES. IN DATA SCIENCE

Data Engineer, Data Analyst, Data Scientist, Machine Learning Scientist/ Engineer

DATA ENGINEER

Control the flow of data. They build custom data pipelines and storage systems. They design infrastructure so that data is not only collected but easy to obtain and process. Within the data science workflow, they focus on the first stage: data collection and storage.

DATA ANALYST

Describe the present via data.Do this by exploring the data and creating visualizations and dashboards. To do these tasks , they often have to clean data first. Analysts have less programming and stats experience than the other roles. Within the workflow, they focus on the middle two stages: data preparation and exploration and visualization.

DATA SCIENTIST

Have a strong background in statistics, enabling them to find new insights from data, rather than solely describing data. They also use traditional machine learning for predictions and forecasting. Within the workflow , they focus on the last three stages: data preparation and exploration and visualization and experimentation and prediction.

MACHINE LEARNING SCIENTIST

Similar to data scientist but with a ML specialization. Perhaps the buzziest part of DataScience. Used to extrapolate what's likely to be true from what we alr know. These scientist use training data to classify larger, unrulier data, whether its to classify images that contain a car, or create a chatbot. They go beyond traditional machine learning with deep learning. Within the workflow, they do the last three stages with a strong focus on prediction.

DATA COLLECTION

DATA SOURCES

Sources of data could be Company Data(collected by companies, helps them make data-driven decisions) and Open data(Free, open data sources, can be used shared and built on by anyone).

COMPANY DATA

Some of the most common sources of data are web events, survey data, customer data, logistics data and financial transactions.
Web data: when you visit a web page or click on a link , usually this information is tracked by companies in order to caculate conversion rates or monitor the popularity of different pieces of content. The following information is captured: the name of the event, which could meant the URL of the page visited or an identifier for the element that was clicked, the timestamp of the event and an identifier for the use that performed the action.
SurveyData: data can also be collected by asking people for their opinions in surveys this can be in the form of a face-to-face interview , online questionnaire or a focus group.

OPEN DATA

Multiple way to access open data. Two of them are API's and public records.
Public data API: easy way to requesting data from a third party over the internet.
Many companies have public APIs to let anyone access their data. Some notable one include Twitter, Wikipedia, Yahoo Finance and Google Map , etc..
Sentimental analysis
See if positive tweets are correlated with more download?
Public record are another great way of gathering data. They can be collected and shared by international organizations like
1.The world bank, UN, WTO, national statistical offices who use census and survey data
2.Government agencies who make information about for eg. The weather, environment or population publicy available.

DATA TYPES

Important when storing data and visualizing/analyzing the data

QUANTITATIVE VS QUALITATIVE DATA

Quantitative can be counted, measure and expressed using number. It is descriptive and conceptual. Eg. The fridge is 60 inches tall, has two apples in it and cost 1000 dollars
Qualitative can be observed but not measured like the fridge is red, was build in Italy and might need to be cleaned out because it smells like fish

OTHER DATA TYPES

Image data, Text data, Geospatial data(location information) , Network data,etc…

DATA STORAGE AND RETRIEVAL

Multiple things to take into consideration
Data storage location: single PC, a cluster of servers , cloud storage.
Data Types: Unstructred- Email, text, video and audio files, web pages, social media, etc,, typically stored in Document database.
Structured -Relational database such as MySQL, PostgreSQL
Retrieval: At a basic level , we'll want to be able to request a specific piece of data such as "All of the images that were created on March 3rd or "All of the customer addresses in Montana" / NoSQL, SQL

HOW DO WE SCALE DATA COLLECTION – DATA PIPELINES

More than one data source (public records, APIs,, DATABASES) and different data types(Unstructured data, tabular data, real time streaming data eg,tweets).

WHAT IS DATA PIPELINE?

Moves data into defined stages – for example, from data ingestion through an API to loading data into a database. A key feature is that pipelines automate this movement.
Automated collection and storage - it would be tedious to ask a data engineer to manually run programs to collect and store data.
- Scheduled hourly, daily, weekly, etc.
- Triggered by an event

Monitored with generated alerted – for example, when 95% of storage capacity has been reach or if an API is responding with an error.
Necessary for big data projects – when working with a lot of data from different sources.
Data engineers work to customize solutions.

Extract Transform Load (ETL)

DATA PIPELINES –TRANSFORM & LOAD

With all the data coming in, how de we keep it organized and easy to use? Example transformations:

Joining data sources into one data set.

Converting data structures to fit database schemas.

Removing irrelevant data - For example, the Twitter API, not only gives you a tweet but other details, like the number of followers the author has, which is not useful in this scenario.

Note: Data preparation and exploration does not occur at this stage.

Finally, we load the data into storage so that it can be used for visualization and analysis.

Once we've set up all those steps, we automate. For example, we can say every time we get a tweet, we transform it in a certain way and store it in a specific table in our database.

DATA PIPELINE CHARACTERISTICS

Which of the following statements is true?

A data pipeline is essential for every data science project.

In the transform phase of ETL, data analysts perform exploratory data analysts.

Data engineers design and build custom data pipelines for projects.

Data pipeline do not require automation

DATA PREPARATION

Why Prepare Data?

Real-life data is messy. Data rarely comes in ready for analysis. Data preparation is done to prevent: errors, incorrect results, biasing algorithms.

KEYS

Tideness, Remove duplicates, Unique ID , Homogeneity(different units used in different countries, Abbreviations, Data Types(str, int) , Missing values)

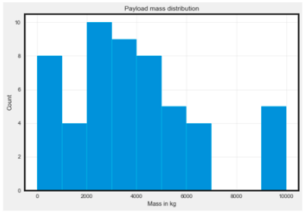
EXPLORATORY DATA ANALYSIS(EDA)

Consists of exploring data and formulating hypothesis about it and assessing its main characteristics with a strong emphasis on visualization.

OUTLINERS – UNUSUAL VALUE

Another thing you do during EDA is look for outliers, that is unusual values. Whether they are errors or valid , it's nice to know about them as they can throw your results off. Here we can see we have only 5 launches with a weight greater than 7000 kg, when the average mass is closer 3800 kg.

Outliers



VISUALIZATION TIPS

Use color purposefully(colorblindness) , readable fonts, label label label, title , x-Axis, y-Axis, easy to understand chart

WHAT ARE EXPERIMENTS IN DATA SCIENCE

Experiments help drive decisions and draw conclusions

Forma question , Form a hypothesis, collect data, test the hypothesis with statistical test, interpret results

CASE STUDY: which is the better blog post title?

Test the hypothesis with a statistical test: is the difference in titles' click through rates significant?

Interprert results:

Choose a title Or ask more questions and desing another experiment!

A/B TESTING STEPS

Picking a metric to track, caculating sample size, running the experiment, checking for significance

TIME SERIES FORECASTING

Modeling in DS • Represent a real-world process with statistics • Mathematical relationships between variables, including random variables. • Based on statistical assumptions and historical data.

• Time series data •A series of data points sequenced by time. Stock prices, Gas prices , Unemployment rates, Heart rates, Inflation rate, CO2 levels, World temperature, Height of ocean tides, Fuel consumption during winter

•Forcasting time series. How much rainfall will we get next month? Will traffic ease up in the next half hour? How will the stock market move in the next six hour? What will be earth's population in 20 years? What will be % of EV cars in 5 years?

•Derive a model from historical data to generate predictions. •Modeling methods use a combination of statistical and machine learning methods.

WHAT IS SUPERVISED MACHINE LEARNING?

•Machine learning: Precdictions from data

•Supervised machine learning: Predictions from data with labels and features.(recommendation systems, diagnosing biomedical images, recognizing hand-written digits, predicting customer churn

HOW DO WE KNOW THE MODEL IS GOOD?

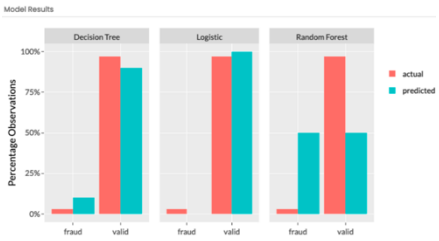
•Supervised machine learning recap(Make a prediction based on data, Data has features and labels(Label is what we want to predict, features are data that might predict the label)) •Trained model can make predictions

FEATURES AND LABELS

Key elements in supervised machine learning. It's important to understand the difference between them when building a model.

MODEL EVALuATING

You are evaluating three different supervised machine learning models for decting fraudulent bank transactions. You knows that one of the weakness of the training data was that there was many more example of valid transaction compared to fraudulent transactions. On the right , you can see how each model predicted that a transaction was fraudulent or valid compared to how often the transaction actually was fraudulent or valid:



According to this data, which model is the best at detecting both fraudulent and valid transactions, and why?

Decision tree - it has high rates of accuracy for both valid transactions and fraudulent transactions. Logistic classifier - overall, it is accurate in >90% of cases.

Random forest - it has equal accuracy for both fraudulent and valid transactions.

UNSUPERVISED MACHINE LEARNING - CLUSTERING

Clustering is a set of machine learning algorithms that divide data into categories, called clusters. Clustering can help us see patterns in messy datasets. Machine Learning Scientists use clustering to divide customers into segments, images into categories, or behaviors into typical and anomalous.

SUPERVISED OR UNSUPERVISED?

- To know whether a new clothing style will be successful based on previous season's trends.
- To create groupings of clothing items that have similar features.
- To see if a customer will purchase an item based on what previous customers purchased.
- To divide customers into different categories based on their shopping habits.
- To know a group of problems/issues based on VMS satisfaction survey.