# Statistics

* Descriptive
  + Summarize data
  + Mean, median, mode, and more
* Inferential
* Collect a subset of the data points: **sample**

## Problem Framing

### Exploratory Analysis

summarize and visualization

### Data mining

Automatic discovery of structured relationships and patterns

### Data understanding

Summary statistics: Mean, median, mode…

## Data cleaning

Identifying and repairing issues

Outlier detection

Imputation

#### Missing data

Some algorithms support missing data

* K-nearest
* Naïve Bayes

## Data selection

Reducing the amount of that to what is relevant

Feature selection

Data sampling

## Data preparation

Change structure or shape

* Standardization
* Normalization
* Hot-encoding

## Types of data

A table with check marks and red x

Description automatically generated

### Nominal

Categorical data

### Ordinal

Ranked data. Have an order. Like high school and university.

### Interval data

Can be measured against a scale each point has the same distance from one another. Like intervals of 5 min.

### Ratio data

Similar to interval data, but have a true meaningful zero, like weight.

## Summary statistics

### Mean

### Median

Middle value. Half points above and half points bellow.

Less sensitive to outliers

### Mode

Most frequent

Useful for qualitative data

### Percentile

Value where x% of data is bellow that position.

Percentile x is the value (or the interpolation) for that given position.

### Range

Maximum value – Minimum value

Very sensitive to outliers

### Variance

Measure of the spread

### Standard deviation

### Standard error

Applied only for mean of means of samples.

## Covariance

Measures the direction of the relationship

### Correlation

Standardize the covariance giving also the strength it goes from -1 to 1, this comes from cosine

**Note:** High correlation do not mean causation, maybe a third variable is causing both to raise.

## Z and T scores

**p < 0.05** tells us that if the two groups were actually the same (null hypothesis is true), then the probability of seeing this result (or a more extreme result) by random chance alone is less than 5%.

### z-score

t-score

# Python

## Built-in functions

### Abs(number)

### filer(function, list/tuple) -> interable

### map(function, list/tuple) -> interable

### max(list/tuple)

### min(list/tuple)

### round(number)

### sorted(list/tuple)

### sum(list/tuple)

### zip(list1/tuple, list2/tuple, …)

## numpy (to work with arrays)

### Arithmetic

Is done element wise

### Creating arrays

#### np.array(list)

#### np.arange(start, stop, step)

#### np.linspace(start, stop, quantity)

#### np.random.normal(mean, std, size=)

#### np.random.randint(start, stop, quantity)

#### np.ones(shape\_tuple)

#### np.zeros(shape\_tuple)

### Selecting items

#### slicing

array[0][0] is equivalent to array[0, 0]

#### filtering

array[(array > 2) & (array < 5)]

### .argmax()

### .argmin()

### .max()

### .min()

### .reshape(shape\_tuple)

### .shape

### np.abs(array) -> elementwise abs

### np.exp(array) -> exi

### np.log(array) -> elementwise ln

### np.mean(array)

### np.median(array)

### np.percentile(array, percentile)

### np.sin(array) -> elementwise sin

### np.sqrt(array) -> elementwise squares root

### np.square(array) -> elementwise squares

### np.std(array) -> deviation

### np.sum(array, axis)

sum of all elements for a given axis, 0 will give the sum of the columns, 1 will give the sums of the rows. If axis is not given will sum all values.

### np.var(array) -> variance

## scipy (for linear algebra and stats)

### from scipy import linalg

#### linalg.det(numpyArray) -> determinant

### linalg.inv(numpyArray) -> inverse from scipy import stats

#### B = stats.binom(trials, successProbabillity)

#### B.cdf(x) -> probability of getting at most x success

Cumulative density function

#### B.pmf(x) -> probability of getting exactly x success

Probability mass function

#### C = stats.norm (mean, std)

#### C.pdf(x)

Chances to be near x.

#### C.cdf(x)

Probability of be less than x

#### D = stats.geom (successProbabillity)

Finds the probability of getting a success in a given number of trials

#### E = stats.poisson(averageFrequency/period)

Instead of number of trials as the binom, poison fix the period and count how many times an event happens on that period.

#### F = stats.exp(rateParameter)

This distribution is commonly used to model the time between independent events that happen at a constant average rate.

#### F.pdf(x)

will give the likelihood of the time between events being exactly x unit.

#### F.cdf(x)

will give the probability that the event occurs within x unit of time.

#### G = stats.beta(alpha, beta)

It’s especially useful when we want to represent beliefs about the likelihood of success in a situation where outcomes can vary continuously between “never happens” (0) and “always happens” (1).

## pandas (Series and Dataframes)

### Create series

#### pd.Series(list, index=[…]\*))

#### pd.Series(npArray, index=[…]\*))

#### pd.Series(dict)

keys will be the indexes

#### pd.Series(scalar, index=[…]\*)

will repeat the scalar

### Create dataframes

#### pd.read\_csv(‘path’)

#### pd.DataFrame(dict, index=[…]\*)

keys will be the column names.

**Example**

dict = {

‘player1’: pd.Series([1,2,3,4], index=[‘game1’, ‘game2’, ‘game3’, ‘game4’])

‘player2’: pd.Series([4,3,2,1], index=[‘game1’, ‘game2’, ‘game3’, ‘game4’])

}

Or

dict = {

‘player1’: pd.Series([1,2,3,4])

‘player2’: pd.Series([4,3,2,1])

}

|  |  |  |
| --- | --- | --- |
|  | player1 | player2 |
| Game1 | 1 | 4 |
| Game2 | 2 | 3 |
| Game3 | 3 | 2 |
| Game4 | 4 | 1 |

#### pd.DataFrame(npArray, index=[…]\*, columns=[…]\*)

### Selecting data

#### df[(df[‘column1’] >= 70) & (df[‘column1’] <= 100)]

using Booleans, returns all rows satisfying this condition

#### .loc[rows\_labels, columns\_labels]

.loc[ : , [‘player1’, ‘player2’]]

#### .iloc[rows\_num, columns\_num]

.iloc[ : , [1, 2]]

### .drop(rowLabels: list)

Drop rows by index

### .drop(columns=columnsLabels: list)

### .groupby(‘columnLabel’).AGGREGATOR

#### .groupby(‘columnLabel’).mean()

### .fillna(value)

### .head(numOfLines)

### .rename(rowLabels: dict)

Key is old name, value is new name

### .rename(columns=columnLabels: dict)

Key is old name, value is new name

### pd.concat(dataframe1, dataframe2, axis=0)

add rows from dataframe2 to daframe1

### pd.merge(dataframe1, dataframe2, on=’columnLabel’)

Do a join. Equivalent to pd.concat(axis=1)

## Matplotlib

### from matplotlib import pyplot as plt

### Line plot

plt.plot(df3['days'], df3['temp'])

plt.xlabel('Days')

plt.ylabel('temperature [C]')

plt.title('Example of line plot')

plt.show()

plt.plot(df3['days'], df3['temp'], label='temp1')

plt.plot(df4['days'], df4['temp'], label='temp2')

plt.xlabel('Days')

plt.ylabel('temperature [C]')

plt.title('Example of 2 lines in the same plot')

# plt.legend()

plt.show()

### Scatter plot

plt.scatter(df3['days'], df3['temp'], label='temp1')

plt.scatter(df4['days'], df4['temp'], label='temp2')

plt.xlabel('Days')

plt.ylabel('temperature [C]')

plt.title('Example of 2 scatters in the same plot')

plt.legend()

plt.show()

### Bar plot

plt.bar(df3['days'], df3['temp'], label='temp1')

plt.bar(df4['days'], df4['temp'], label='temp2', alpha=0.5, width=0.5)

plt.xlabel('Days')

plt.ylabel('temperature [C]')

plt.title('Example of 2 bar plots in the same plot')

plt.legend()

plt.show()

### Pie Plot

plt.pie(df2['e'], labels=df2.index, autopct='%1.1f%%')

plt.title('Example of pichart')

plt.show()

### Histogram

plt.hist(salaries, rwidth=0.9)

plt.xlabel('Salaries')

plt.ylabel('Frequency')

plt.title('Example of histogram')

plt.legend()

plt.show()

### Subplots

figure, subplots = plt.subplots(1, 2, figsize=(8,4))

subplots[0].plot(df3['days'], df3['temp'])

subplots[0].set\_title('Line1')

subplots[1].plot(df4['days'], df4['temp'], color='green')

subplots[1].set\_title('Line2')

for subplot in subplots:

subplot.set(xlabel='day', ylabel='temperatur [C]')

plt.show()

## seaborn

### import seaborn as sns

### sns.boxplot(x=’columnLabelToGroup’, y=’columnLabelToPlot’, data=dataFrame)

### sns.violinplot(x=’columnLabelToGroup’, y=’columnLabelToPlot’, data=dataFrame)

similar to the boxplot, but put aggregate the density to the graphic

### sns.heatmap(dataFrame)

### sns.kdeplot(array/dataframe)

### sns.kdeplot(x=array1, y=array2)

will give the bivariate distribution (contour plot)

# Spark

## DataFrames

### Create DataFrames

#### originalRDD.toDF()

#### originalDF.toDF(“columName1”\*, “columName2”\*, …)

#### spark.createDataFrame( [[ ]], schema: string)

employeesDF = (

    spark.createDataFrame(

        data,

        "Id: long, Name: string, Salary: long"

    )

)

#### spark.read.option(options).fileType(path)

yellowTaxiDF = (

    spark.read.option("header", "true")

   .option("mode", 'PERMISSIVE' | 'DROPMALFORMED' | 'FAILFAST')

    #       .option("inferSchema", "true") # should be avoid, define the schema

    # instead

    .schema(yellowTaxiSchema).csv(

        "apache-spark-3-fundamentals/DataFiles/Raw/YellowTaxis\_202210.csv")

    )

### Create schemas

Example

taxiBasesSchema = (

    StructType([

        StructField("License Number", StringType(), True),

        StructField("Entity Name", StringType(), True),

        StructField("Telephone Number", LongType(), True),

        StructField("SHL Endorsed", StringType(), True),

        StructField("Type of Base", StringType(), True),

        StructField("Address", StructType([

            StructField("Building", StringType(), True),

            StructField("Street", StringType(), True),

            StructField("City", StringType(), True),

            StructField("State", StringType(), True),

            StructField("Postcode", StringType(), True),

        ]), True),

        StructField("GeoLocation", StructType([

            StructField("Latitude", StringType(), True),

            StructField("Longitude", StringType(), True),

            StructField("Location", StringType(), True),

        ]), True),

        StructField("Date", StringType(), True),

        StructField("Time", StringType(), True),

    ])

)

### Flattening multilevel object

Example

# Flattening nested fields

taxiBasesFlatDF = (

    taxiBasesDF

        .select(

            col('license Number').alias("BaseLicenseNumber"),

            col("Entity Name").alias("EntityName"),

            col(**"Address.Building"**).alias("AddressBuilding"),

            col(**"Address.Street"**).alias("AddressStreet"),

            col(**"Address.City"**).alias("AddressCity"),

            col(**"Address.State"**).alias("AddressState"),

            col(**"Address.Postcode"**).alias("AddressPostCode"),

            col(**"Geolocation.Latitude"**).alias("GeoLatitude"),

            col(**"Geolocation.Longitude"**).alias("GeoLongitude"),

        )

)

### Functions

#### .coalesce(numOfPartitions)

reduce the number of partitions

#### .describe(“columnName1”, “columnName2”, …)

Return statistics such as mean…

#### .drop(“columnName”)

#### .dropDuplicates(“columnNames”)

        .dropDuplicates() #if you dont specify any column names,

                        # complete row will be used to check duplicates

#### .filter(“condition”)

        .filter("passenger\_count > 0")

Or

        .filter(col("trip\_distance") > 0) #filter and where are the same

#### .groupBy(colum1, column2, …).agg(function1, function2, …)

        .groupBy("PickupLocationId", "DropLocationId")

        .agg(

            round(avg("TripTimeMinutes").alias("AvgTripTime")),

            round(sum("TotalAmount").alias("SumAmount"))

        )

#### .na.drop(‘all’ | ‘any’)

#### .na.fill(values: dict)

values = {'payment\_type': 5, 'RateCodeID': 1}

#### .orderBy(col().desc()\*)

#### .printSchema()

#### .rdd.getNumPartitions()

#### .select(“columnName1” | col(“columName1”) .cast(type).alias(“newname”), …)

#### .show()

#### .where(condition)

        .where("passenger\_count > 0")

Or

        .where(col("trip\_distance") > 0) #filter and where are the same

#### .withColumn(“columnName”, calulatedValue)

        .withColumn("TripType", tripTypeColumn)

# "If-Else" (when otherwise) example

tripTypeColumn = (

    when(

        col("RatecodeID") == 6,

            "SharedTrip"

    ).otherwise("SoloTrip")

)

#### .withColumnRenamed(“oldName”, “newName”)

        .withColumnRenamed("passenger\_count", "PassengerCount")

#### .write.option().mode.fileType()

        .write

        .option("header", "true")

        .option("dateFormat", "yyy-MM-dd HH:mm:ss.S")

        .mode("overwrite") # options - append, errorIfExists, ignore

        .csv("apache-spark-3-fundamentals/DataFiles/Raw/YellowTaxiMatheus.csv")

#### Array

##### array\_contains

##### array\_join

##### concat

##### explode

##### filter

#### Date & Time

##### current\_timestamp

##### date\_add

##### date\_format

##### next\_day

##### to\_date

##### to\_timestamp

#### Mathematical

##### ceil

##### floor

##### log

##### round

##### sqrt

#### Other

##### greatest

##### isnull

##### least

##### when

#### String

##### lenght

##### lower

##### split

##### substring

##### trim

##### upper

## RDDs

### Create RDDs

#### sc.parallelize([])

From a list OR list of lists

#### sc.textFile(path)

From reading a text file

### Functions

#### .collect() -> List of elements in the RDD

#### .distinct() -> return distinct elements (keys for a pair RDD)

#### .filter(function)

#### .first() -> first element in the RDD

#### .getNumPartitions()

#### .keys()

Return a pair RDD keys

#### .map(function)

#### .sortByKey()

Sort a pair RDD by its key

#### .take(num) -> list of num elements in the RDD

#### .values()

Return a pair RDD values

## SQL

### Create Views

#### dataFrame.createOrReplaceTempView(“viewName”)

#### dataFrame.createOrReplaceGlobalTempView(“viewName”)

### Catalogs

#### In-memory vs persistent

In memory is default, to create a persistent one initiate sparkSession with the following function

.enableHiveSupport()

#### Managed vs External Database

Managed are saved in the default spark path (database and metadata).

External Database, only the metadata is created.

#### Create Database

spark.sql("CREATE DATABASE IF NOT EXISTS databaseName")

#### dataFrame.write.mode(‘overwrite’).saveAsTable(“database.tableName”)

### Concatenate

#### EXCEPT

#### INTERSECT

#### UNION

#### UNION ALL

### User-defined function (UDF)

Spark standard optimizations are not applicable to UDFs.

Nulls can cause issues if not handled correctly.

#### udf()

To use with dataFrames

#### spark.udf.register()

To use with SQL

### spark.sql(“query”)

### Window Operation

#### SUM(columName) OVER(PARTITION BY columnName)

If PARTITION BY clause is not given, it will apply the operation to the whole dataset

Can use any aggregation operation