# kb-anonymity

**Data Protection and Privacy Exam 2022-2023**

La Corte (S4784539) - Scarrà (S4798949)

# 1.

# Basics of the implemented algorithm

The theory behind the algorithm

# Basics of the implemented algorithm

It's practically infeasible to generate test cases that can exercise all possible program states in a program.

Industrial systems:

- has to be tested by a third-party with realistic values,
- has to be tested on datasets which carry sensitive information.

Simply **masking data sensitive values is not enough** because correlations among fields in the data can reveal masked information; in addition to, masked data has a lower utility level in terms of behaviour preservation.

kb-anonymity combines k-anonymity with the concept of program behaviour preservation.

# Basics of the implemented algorithm

## PRIVACY PRESERVATION

The goal is to ensure that the **identity of every person cannot be revealed** in the released dataset.

Like k-anonymity, it replaces some informations in the original dataset to ensure privacy preservation so that the replaced data can be released to third party developers.

## BEHAVIOUR PRESERVATION

The goal is to ensure that the **program behaviour can be reproduced** by the released dataset.

Unlike k-anonymity, it ensures that the replaced data exhibits the same kind of program behaviour exhibited by the original data.

# Basics of the implemented algorithm

We perform **selective data value replacement** in the original dataset.

Each data point in the new dataset can still be used as a test case but cannot identify any individual in the original dataset.

- **k-anonymity** provides guidance on choosing data fields to mask,
- **concolic execution**, based on path condition, can guide the generation of new test cases which must satisfy certain behaviour-related properties,
- **fake values** may be introduced and certain statics can thus be distorted, rendering the new dataset unsuitable for purposes other than testing and debugging.

## Requirements

1. **All values in the released dataset are concrete**
   In order to execute programs

2. **All released tuples are distinguishable from each other**
   No redundant test cases

3. **Minimal k-anonymity**
   For each raw tuple there exist at least k-1 other raw tuples mapped to a single tuple in the released dataset

4. **Same behavior**
   Each raw tuple must be mapped in a tuple that exhibit the same behavior when run on the program.

# Basics of the implemented algorithm

## Privacy Preservation Levels

- **None**
- **Standard k-anonymity:** every released data point must be indistinguishable from at least k–1 other released data points.
- **No Tuple Repeat:** every released tuple is distinguishable from every raw tuple, but allows them to share some field values.
- **No Field Repeat:** every value in the released dataset has not appeared in any raw data point.

## Behavior Preservation Levels

- **None**
- **Same Path:** each released data point must follow the same execution path as the path followed by the raw data point mapped to it.
- **Same Path with Input Restrictions:** aims to consider more program behaviors beyond execution paths, such as particular input values.
- **Same Program States:** the sequence of program states exhibited by each release tuple and the raw tuple mapped to it should be the same.

# Combining Privacy and Preservation Levels

Privacy Preservation (Low → High)

| Privacy Preservation | None | Same Path | Same Path with Input Restrictions | Same Program States |
|---|---|---|---|---|
| No Field Repeat | N/I | ✓ | ✗ | ✗ |
| No Tuple Repeat | N/I | ✓ | ✓ | ✗ |
| Standard k-Anonymity | N/I | N/I | N/I | ✗ |
| None | N/I | N/I | N/I | N/I |

Behavior Preservation (Low → High)

# 2.

# Implementation

The process of analyzing, implementing and testing the algorithm

# Timeline

**ANALYSIS**

The first step was understanding the paper

**PSEUDOCODE**

Then we wrote the pseudocode based on the one in the paper with some adjustments aimed to the implementation

**IMPLEMENTATION**

Then we implemented the algorithm, developing the three main modules

**TESTING**

Finally, we developed a test suite which evaluates many different combinations of parameters

# Implementation of the three modules

## Program Execution

Here we initialize **PC_Buckets**: dictionaries which groups tuples depending on their path condition.

- **Key**: path condition.
- **Value**: list of tuples.

## K-Anonymity

k-anonymization of a single Bucket which outputs non duplicate anonymized tuples, together with the path condition and the original Bucket.

To achieve k-anonymity we leverage **anonypy** library.

## Constraint Generation

Construct constraints for configurations:

- Same **P**ath, no **F**ield Repeat
- Same **P**ath, No **T**uple Repeat
- Same Path with **I**nput Restrictions, No **T**uple Repeat

Then it generates new tuples based on achieved constraints.

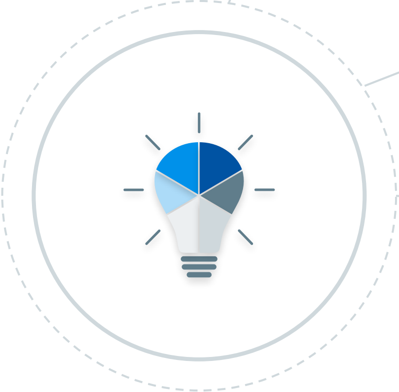
# Anonypy

AnonyPy provides **privacy preserving techniques for the anonymization**.

- K-Anonymity, L-Diversity and T-Closeness,
- Anonymization method aims at making the individual record be indistinguishable among a group of records by using techniques of **generalization and suppression**.
- AnonyPy uses ***Mondrian*** algorithm to partition the original data into smaller and smaller groups
- The algorithm assumes that we have converted all attributes into numerical or categorical values and that we are able to measure the *span* of a given attribute Xi.

In order to use the *Preserver()* method of this library we created a Pandas Dataframe for each PC_Bucket and marked the categorical features of it, which suits perfectly our implementation.

# Our Solver

### Generate Set of Possible Values

**Numeric** → the set of possible values is a range from *min* to *max*

**Categorical** → the set contains all the values from the dataset

### Edit Set of Possible Values

Each constraint removes entries in the set of possible values, depending on its operator and value.

### Generate the Tuple

For each field of the tuple to create, one possible value is randomly extracted and removed out of the set containing the remaining ones.

# Categorical!

Our solver supports categorical features

3.

# Experimental Results

The process of collecting and showing metrics of the algorithm

Our **Test Suite**

# 42 Combinations of Parameters

Obtained by combining different values for k, n and bpl

# 210 Simulations

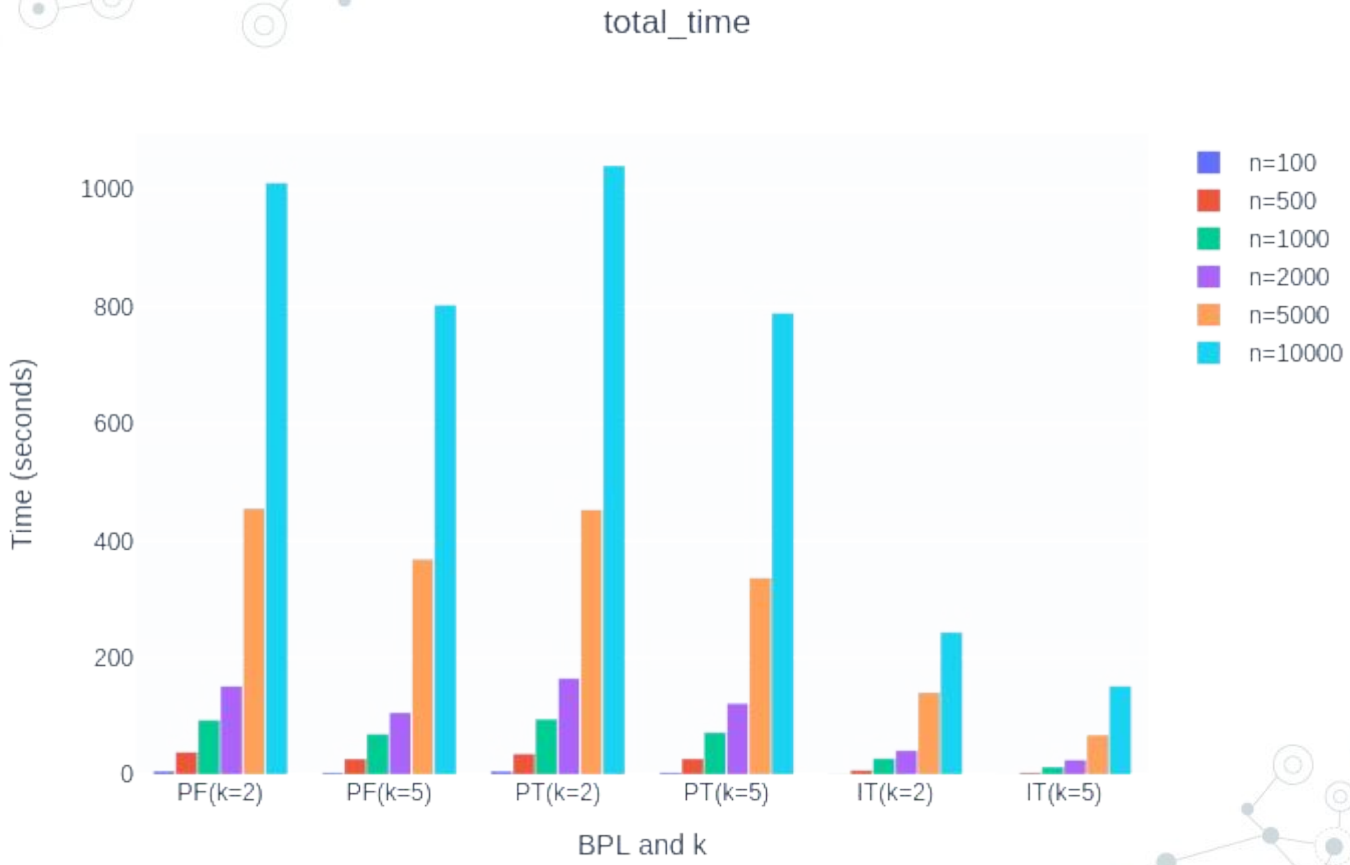
Each combination is run 5 times and metrics are gathered by their average

# Data Visualization

All the results referred to a metric are grouped in the same chart

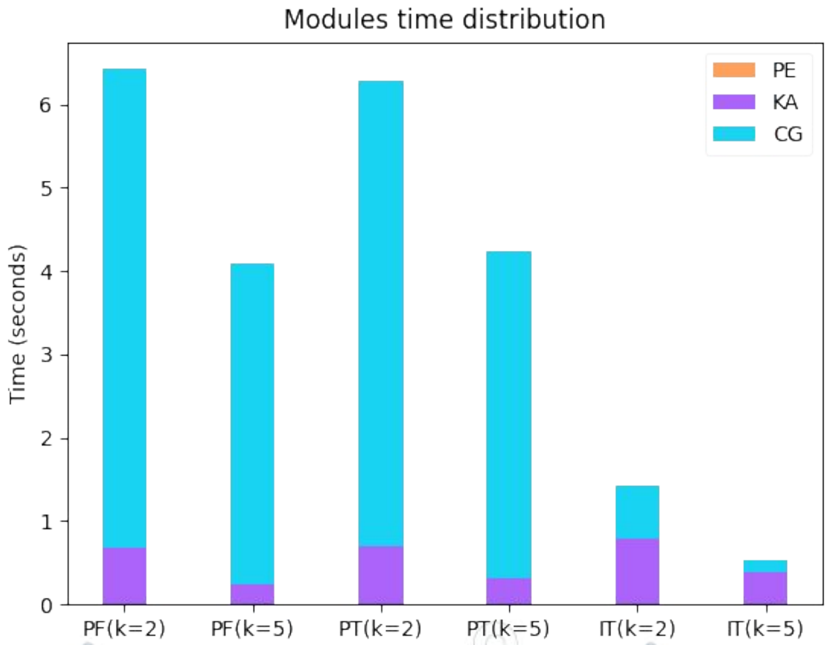
| | |
|---|---|
| k | [2, 5] |
| n | [100, 500, 1000, 10000, 50000] |
| bpl | [PF, PT, IT] |

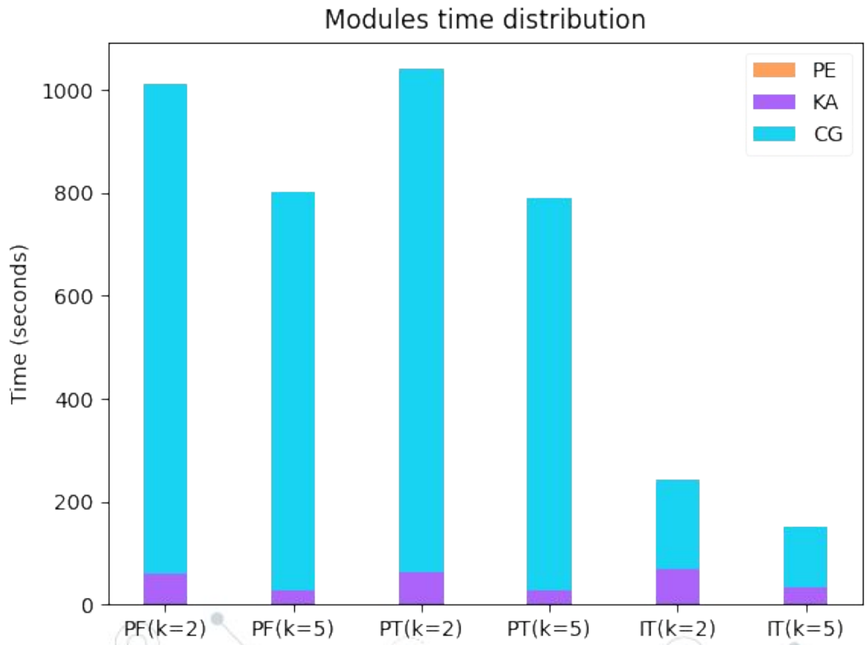
# **Total Time Spent** by the Algorithm



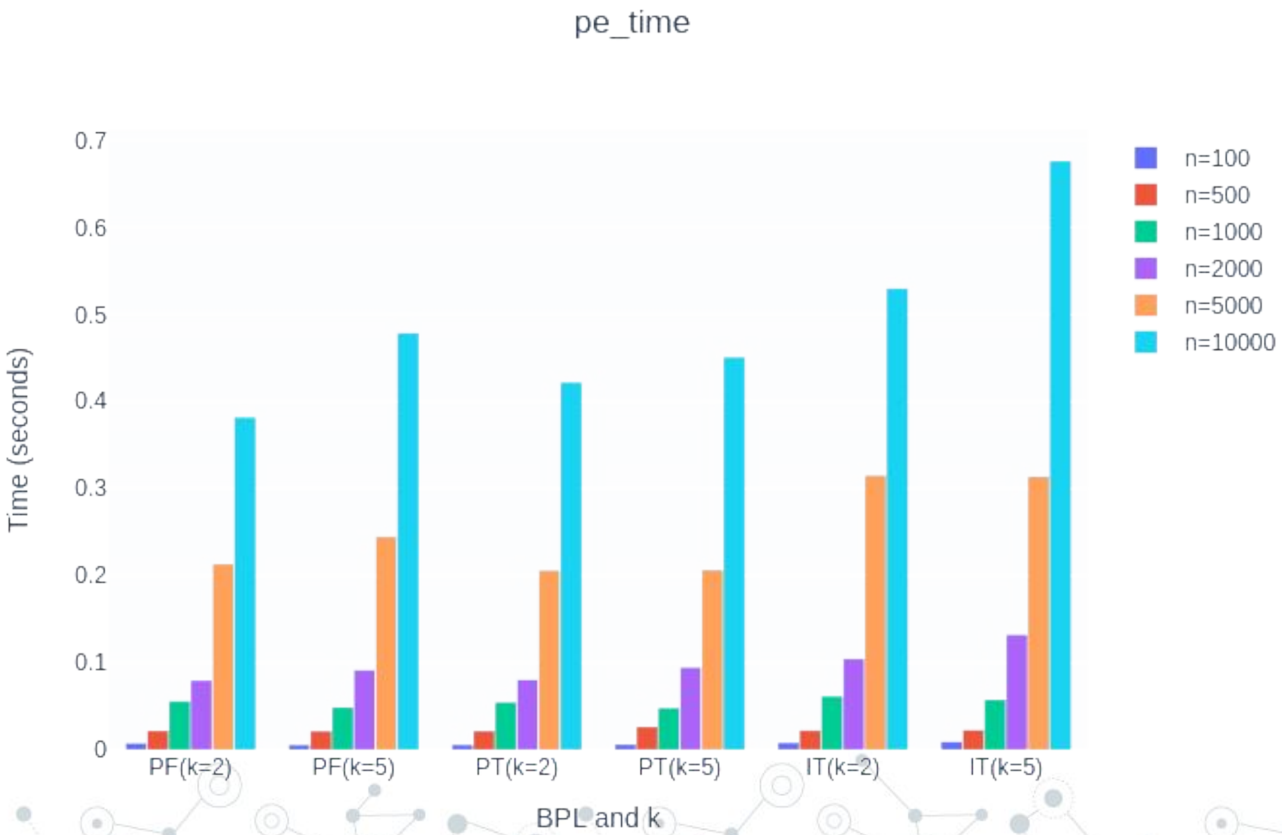
# Time required for each module

Using different datasets: **n=100** (left), **n=10000** (right)

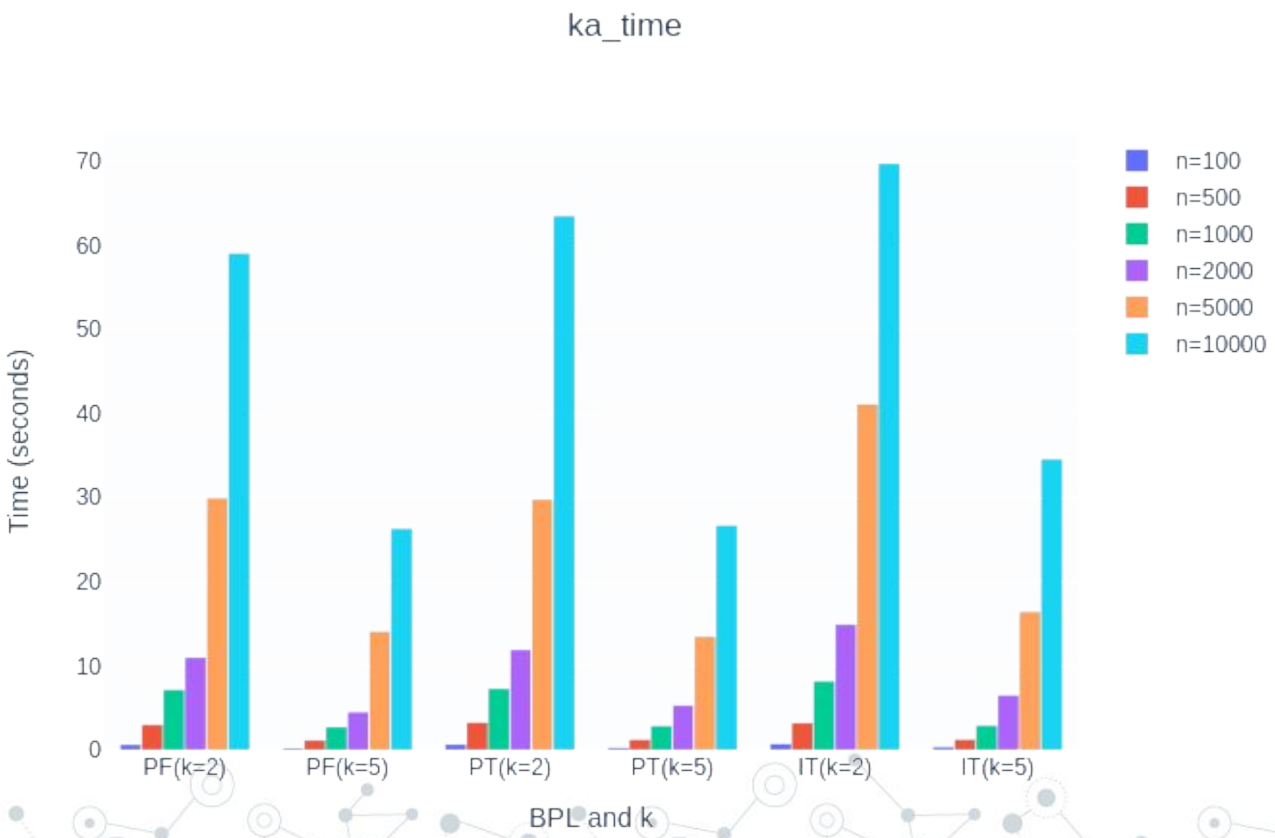


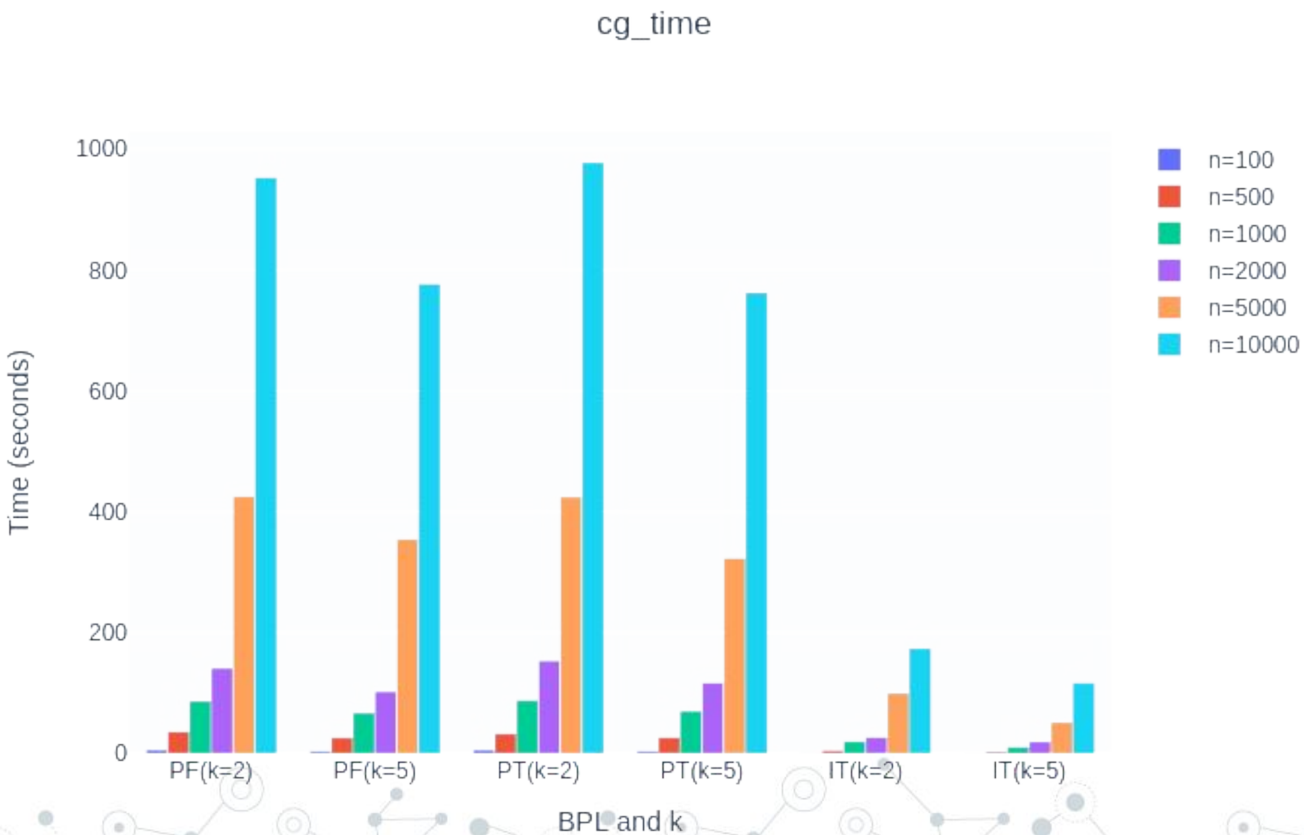
# Time Required: Program Execution Module



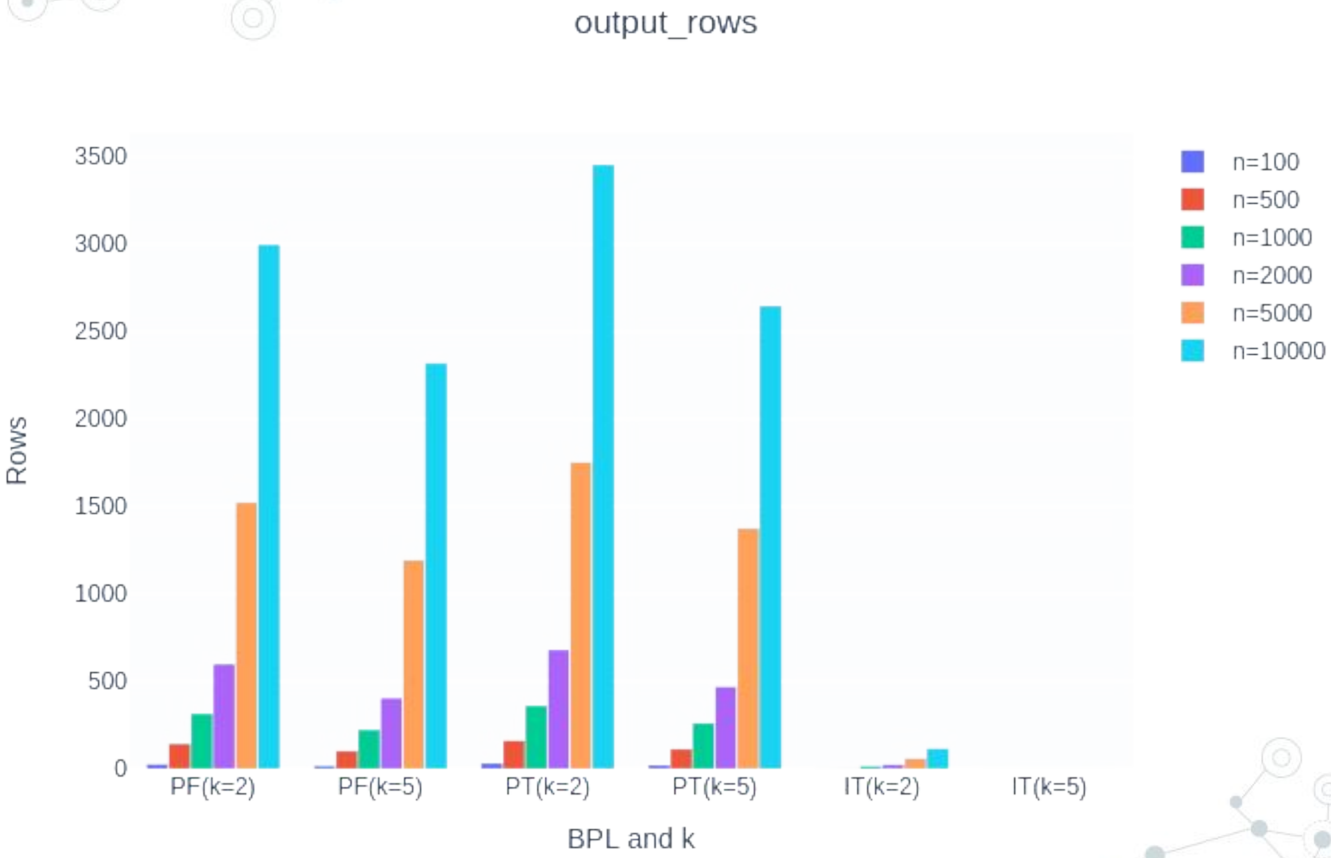
# Time Required: k-anonymity Module



# Time Required: Constraint Generation Module



# Rows in the output dataset



# Thanks!

**Lorenzo La Corte**
STUDENT

**Davide Scarrà**
STUDENT