Universidad de Murcia
INGENIERÍA INFORMÁTICA

TECNOLOGÍAS ESPECÍFICAS DE LA INGENIERÍA INFORMÁTICA

Boletín de prácticas de los bloques 3 y 4

Autor: Victorio J. Molina Bermejo
DNI: 48632380-F Grupo: 3.2

FACULTAD DE INFORMÁTICA Junio 2020

Índice

1. Especificación de la práctica

Imaginemos que dada, una lista de elementos, queremos obtener una nueva versión de la misma sin repeticiones. Python nos ofrece los conjuntos para ello, pero también podríamos conseguir lo mismo usando listas, diccionarios, etc. Queremos evaluar las distintas posibilidades que tenemos en Python para saber cuál es la más eficiente y compararlas con una implementación propia en C de una función de eliminación de duplicados. Para ello, debemos implementar un programa de Python que reciba tres parámetros: un fichero de entrada con una lista de elementos, uno por línea, un fichero de salida donde se guardará esa misma lista de elementos, pero sin repeticiones, y un fichero de salida donde se guardará en PDF la gráfica que mostrará el tiempo usado por cada técnica para distintos tamaños de datos; esta gráfica se generará usando Matplotlib. Un detalle a tener en cuenta es que el orden de los elementos en el fichero de salida puede ser distinto de aquel que tienen en el fichero de entrada.

Podemos suponer que los elementos del fichero de entrada son números naturales entre 0 y un cierto límite. Para hacer las pruebas, se recomienda generar un fichero que contenga 200000 números naturales aleatorios entre 0 y 99999 (ambos inclusive) y hacer un estudio de cada técnica con los primeros 2000 elementos del fichero, los primeros 4000 elementos del fichero, los primeros 6000 elementos del fichero, y así sucesivamente hasta el total de 200000 elementos. Para generar el fichero de entrada, se sugiere hacer un pequeño programa en Python que reciba como parámetro el nombre del fichero y, usando Numpy, se generen los 200000 números naturales entre 0 y 99999 (el número de elementos a generar también podría ser un parámetro de entrada).

Al menos, hay que analizar dos formas de obtener una lista sin repeticiones usando dos estructuras de datos distintas de Python y una tercera forma que debemos implementar en código en C y que usaremos desde Python a través de ctypes. En esta tercera forma, la función de C a implementar debe recibir la lista de elementos original y devolver la lista de elementos sin duplicados. La memoria dinámica que sea necesaria reservar para el intercambio de información entre el código de Python y el de C se reservará desde Python. Los alumnos tienen que decidir qué código de C implementar para obtener el resultado deseado de la forma más eficiente posible.

Además de la gráfica generada en PDF, que habrá que incluir en el documento LATEX final donde se explique el trabajo realizado, también habrá que incluir el resultado del perfilado (profiling) del tiempo de ejecución de cada técnica de eliminación de duplicados usada y una gráfica que muestre el consumo de memoria por parte del programa durante su ejecución. Para realizar estos perfilados, pueden usarse los módulos *line_profiler* y *memory_profiler* descritos en teoría. En el perfilado del tiempo de ejecución no es necesario hacer un estudio para cada posible tamañoo de la lista de elementos de la que hay que eliminar duplicados. Basta con que el perfilado se haga de tal manera que refleje lo mismo que la gráfica generada en PDF en cuanto al orden de las distintas técnicas en lo que a eficiencia se refiere.

2. Resolución de la práctica

2.1. Código

2.1.1. Código Python

A continuación se presenta el código Python utilizado para resolver el enunciado, desde el módulo main, y el fichero contenedor de las funciones de eliminación de duplicados, en la que se incluye el Wrapper que permite usar la biblioteca C existente, hasta el script para la generación del fichero de entrada.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'''
Created on Thu May  7 01:40:07 2020

@author: VictorioMolina
'''

import sys
import utils
import matplotlib.pyplot as plt
import numpy as np
import time
from inspect import getmembers, isfunction
from matplotlib.backends.backend_pdf import PdfPages
from matplotlib import gridspec
```

```
18
19 plt.rcParams.update({'font.size': 14})
20 plt.rcParams.update({'figure.max_open_warning': 0})
21
22
23 # Method that turns the file lines into a list
24 def file_to_list(path):
25     try:
26         infile = open(path, 'r')
27     except IOError:
28         # The file doesn't exist
29         print("The file {} doesn't exists".format(path))
30         sys.exit(-1)
31     else:
32         # The file exists
33         elements = []
34
35         # Add the file lines to the list of elements
36         elements.extend(int(line) for line in infile)
37
38         infile.close()
39         return elements
40
41 # Method that writes the result of the program in an specific
    file
42 def result_to_file(result, path):
43     try:
44         outfile = open(path, 'x')
45     except FileExistsError:
46         print("The file {} already exists".format(sys.argv[2]))
47         sys.exit(-1)
48     else:
49         for element in result:
50             outfile.write(str(element) + '\n')
51         outfile.close()
52
53 # Method that generates a plot and save it in an specific PDF
    file
54 def generate_plots(title, x, y, pdf):
55     fig = plt.figure(figsize=(14, 19.8)) # A4 Size
56
57     fig.suptitle(title, fontsize=30)
58
59     spec = gridspec.GridSpec(ncols=3, nrows=2, width_ratios
    =[0.5,4,0.5])
```

```
60
61      # Bar plot
62      ax = fig.add_subplot(spec[1])
63      plt.xlabel("Execution Time")
64      width = 0.75 # Bars' width
65      ind = np.arange(len(y))
66      ax.barh(ind, y, width, color="blue")
67      ax.set_yticks(ind + width / 4)
68      ax.set_yticklabels(x, minor=False)
69
70      # Pie chart
71      ax1 = fig.add_subplot(spec[4])
72      explode = (0, 0, 0, 0, 0, 0.05)  # only "explode" the 6th
    slice which represents the C function
73      ax1.pie(
74          [_ * 1000 for _ in y],
75          explode=explode,
76          labels=x,
77          autopct='%1.1f%%',
78          shadow=False,
79          startangle=90
80      )
81      ax1.axis('equal')  # Equal aspect ratio ensures that pie is
    drawn as a circle.
82
83      # Save current figures in a pdf page
84      pdf.savefig()
85
86
87
88  # Method that studies the efficiency of each technique for
    different data sizes
89  # generating graphics that will be saved in a pdf
90  def study_techniques_efficiency(seq, path):
91      try:
92          pdf_outfile = open(path, 'x')
93      except FileExistsError:
94          print("The file {} already exists".format(path))
95          sys.exit(-1)
96      else:
97          # Create a multipage PDF file
98          pdf = PdfPages(path)
99
100         # Get all the functions of the module
101         functions_list = [o for o in getmembers(utils,
```

```
    isfunction)]
102
103         # Make an study of each technique in size intervals of
    2000 elements
104         size_interval = 2000
105         for i in range(int(len(seq) / size_interval)):
106             # For each technique, calling its function with the
    corresponding time taking
107             cutten_list = seq[0:(i+1)*size_interval]
108             exec_times = [] # Will store the exec time of each
    function
109
110             for function_name, function_obj in functions_list:
111                 # Calculate the exec time
112                 t0 = time.time_ns()
113                 result = function_obj(cutten_list)
114                 t_exec = (time.time_ns() - t0) / 1.0e9
115
116                 # Store the ecex time
117                 exec_times.append(t_exec)
118                 print("{} has taken {} seconds to execute".
    format(function_name, t_exec))
119
120             # Generate a plot with the execution time stats
121             generate_plots(
122                     "TIMES FOR THE FIRST {} NUMBERS".format((i
    +1)*size_interval),
123                     [f[0] for f in functions_list],
124                     exec_times,
125                     pdf
126             )
127
128         # Close the pdf
129         pdf.close()
130
131 def main():
132     """
133
    ----------------------------------------------------------------------------

134         PROGRAM ARGUMENTS:
135             -> infile: input file with a list of elements, one
    per line
136             -> outfile: output file where that same list of
    elements will be saved,
```

```
                    but without repetitions
            -> pdf_outfile: output file where the graph that will show the time used
                    by each technique for different data sizes will be saved in pdf

    ------------------------------------------------------------------------


    """

    # Command line argument control
    if len(sys.argv) != 4:
        print("Usage: {} infile outfile pdf_outfile".format(sys.argv[0]))
        sys.exit(0)

    # Reading the sequence of the input file
    seq = file_to_list(sys.argv[1])

    # Calling the external function through its wrapper
    result = utils.remove_duplicates_6(seq)

    # Writing the result in the outfile
    result_to_file(result, sys.argv[2])

    # Finally, study the different techniques efficiency
    study_techniques_efficiency(seq, sys.argv[3])

if __name__ == '__main__':
    main()
```

File 1: main.py

```
# -*- coding: utf-8 -*-

'''
This file contains a number of useful functions for removing
    duplicates.

@author: VictorioMolina
'''

import ctypes, os
from functools import reduce
import numpy as np
import time


# Loading the shared library into ctypes
LIBREMOVE_DUPLICATES = ctypes.CDLL(
    os.path.abspath(
        os.path.join(os.path.dirname(__file__), "../C/
    libremove_duplicates.so.1")
    )
)

# @profile
def remove_duplicates_1(seq):
    # Not preserving the order
    return list(set(seq))

# @profile
def remove_duplicates_2(seq):
    # Preserving the order
    found = set()
    return [x for x in seq if not (x in found or found.add(x))]

# @profile
def remove_duplicates_3(seq):
    # Not preserving the order
    keys = {}
    for x in seq:
        keys[x] = 1
    return list(keys.keys())

# @profile
def remove_duplicates_4(seq):
    # Not preserving the order and using NumPy
```

```python
    return list(np.unique(seq))

# @profile
def remove_duplicates_5(seq):
    # Preserving the order and using NumPy
    indexes = sorted(np.unique(seq, return_index=True)[1])
    return [seq[i] for i in indexes]

# Python Wrapper for calling the C function
# @profile
def remove_duplicates_6(seq):
    # Object corresponding to the function within the library
    func_remove_duplicates = LIBREMOVE_DUPLICATES.remove_duplicates

    # Function protoype
    func_remove_duplicates.argtypes = [
        ctypes.POINTER(ctypes.c_uint),
        ctypes.POINTER(ctypes.c_uint),
        ctypes.c_uint
    ]

    # This function returns void
    func_remove_duplicates.restype = None

    # Variable in which we will collect the result of the function.
    result = (ctypes.c_uint * len(seq))()

    # Call to the shared library function.
    func_remove_duplicates((ctypes.c_uint * len(seq))(*seq), result, len(seq))

    # Copying result of the function to another vector
    vout=[*result]  # This type of copy is the 'fastest' in Python

    # Removing all zeros least the first one
    while len(vout) > 1 and vout[-1] == 0:
        vout.pop()

    # Finally, we return the output vector
    return vout
```

File 2: /src/Python/utils.py

```python
# Generate a file made up of random integers

import numpy as np

quantity = int(input("How many? "))
limit = int(input("Max integer: "))
filename = "numbers.txt"

numbers = np.random.randint(low=0, high=limit+1, size=quantity)

f = open(filename, 'w')
for n in numbers:
    f.write("{}\n".format(n))
```

File 3: /test/generate_test_file.py

2.1.2. Código C

En lo que al código C respecta, he implementado un fichero de cabecera con los prototipos de función, un fichero .c con la implementación del método de eliminación de duplicados dando uso del algoritmo *Heap Sort*, además de generar un archivo Makefile para la compilación del código con la biblioteca gcc, que conlleva a la creación de la biblioteca compartida nativa de Linux *libremove_duplicates.so.1.0.1* utilizada en el wrapper de Python.

```c
#ifndef __REMOVE_DUPLICATES_H__
#define __REMOVE_DUPLICATES_H__

/* Working with memory already reserved by Python! */

// Function Prototypes

void remove_duplicates(const unsigned int *vin,
                       unsigned int *vout,
                       const unsigned int size);

void heap_sort(unsigned int arr[], const unsigned int size);

void create_max_heap(unsigned int arr[],
                     const unsigned int size,
                     unsigned int root_index);

void swap_nodes(unsigned int *ptr1, unsigned int *ptr2);
```

```
19
20 #endif
```

File 4: /src/C/remove_duplicates.h

```
1  #include "remove_duplicates.h"
2
3  // Function to remove duplicated elements in an array
4  void remove_duplicates(const unsigned int *vin, unsigned int *
       vout, const unsigned int size)
5  {
6      /*
7
8          Complexity Analysis
9          -------------------------------------------
10         TIME                    O(n * log(n)) + O(n) = O(n *
       log(n))
11         SPACE                   O(n) + O(1) = O(n)
12         -------------------------------------------
13
14     */
15
16     // Creating copy of the entry vector... (good choice)
17     unsigned int arr[size];
18     for(int i = 0; i < size; i++)
19     {
20         arr[i] = vin[i];
21     }
22
23     // Sorting the array with the heap-sort algorithm
24     heap_sort(arr, size);
25
26     int i = 0;
27     int j = 0;
28     while (i < size)
29     {
30         // Insert element to the result vector
31         vout[j] = arr[i];
32         j = j + 1;
33
34         while (arr[i] == arr[i + 1])
35         {
36             i = i + 1;
37         }
38         i = i + 1;
```

```
    }
}

// Heap-sort main function
void heap_sort(unsigned int arr[], const unsigned int size)
{
    /*

        Complexity Analysis
        -------------------------------------------
        best, worst, average TIME      O(n * log(n))
        SPACE                          O(1)
        -------------------------------------------

    */

    for (signed int i = size / 2 - 1; i >= 0; i--)
    {
        create_max_heap(arr, size, i);
    }

    for (signed int i = size - 1; i > 0; i--)
    {
        swap_nodes(&arr[0], &arr[i]);
        create_max_heap(arr, i, 0);
    }
}

// Heap-sort helper function
void create_max_heap(unsigned int arr[], const unsigned int size
    , unsigned int root_index)
{
    unsigned int root = root_index;
    unsigned int left = 2 * root_index + 1;
    unsigned int right = 2 * root_index + 2;

    // If the left son is greater than the root
    if (left < size && arr[left] > arr[root])
    {
        root = left;
    }

    // If the right son is greater than the root
    if (right < size && arr[right] > arr[root])
    {
```

```c
        root = right;
    }

    // If the root has changed
    if (root != root_index)
    {
        swap_nodes(&arr[root_index], &arr[root]);

        // Continue resolving the new tree
        create_max_heap(arr,  size, root);
    }
}

// Heap-sort helper function
void swap_nodes(unsigned int *ptr1, unsigned int *ptr2)
{
    unsigned int tmp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = tmp;
}
```

File 5: /src/C/remove_duplicates.c

```
all: libremove_duplicates.so.1.0.1 libremove_duplicates.so.1

libremove_duplicates.so.1.0.1: remove_duplicates.c
    remove_duplicates.h
  gcc -Wall -c -fPIC -g remove_duplicates.c
  gcc -shared -Wl,-soname,libremove_duplicates.so.1 -o
    libremove_duplicates.so.1.0.1 remove_duplicates.o

libremove_duplicates.so.1: libremove_duplicates.so.1.0.1
  ln -s libremove_duplicates.so.1.0.1 libremove_duplicates.so.1

clean:
  rm -fr libremove_duplicates.so.1.0.1 libremove_duplicates.so.1
     remove_duplicates.o *~
```

File 6: /src/C/Makefile

3. Guía para la ejecución del programa

Para la ejecución del programa basta con teclear siguiente comando en el terminal:

```
python3 main.py infile outfile pdf_outfile
```

Siendo *infile* el fichero de entrada que contiene el listado de números; *outfile* la ruta del fichero (inexistente) en el que se guardará el resultado del programa, es decir, la lista de numeros del fichero de entrada pero sin duplicados; y *pdf_outfile* la ruta del fichero (también inexistente) en el que se guardarán las gráficas estadísticas creadas durante el estudio de los tiempo de ejecución de las distintas técnicas de eliminación de duplicados implementadas en el programa.

Cabe destacar que he usado en el módulo main he usado *#!/usr/bin/env python3* para la portabilidad en diferentes sistemas en caso de que tengan el intérprete de idiomas instalado en diferentes ubicaciones, lo cual además nos permite ejecutar el programa de la siguiente manera:

```
./main.py infile outfile pdf_outfile
```

4. Herramientas utilizadas durante el desarrollo

4.0.1. Numpy

Numpy es el paquete fundamental para la computación científica con Python. Entre otras cosas contiene:

- Un poderoso objeto de matriz N-dimensional
- Funciones sofisticadas de broadcasting, que permite trabajar con arrays de diferentes tamaños
- Herramientas para integrar código C/C++
- Álgebra lineal útil, transformación de Fourier, y capacidades para la generación de números aleatorios

Además, una gran cantidad de paquetes de cómputo científico se apoyan de Numpy como biblioteca básica, siendo Matplotlib, biblioteca de la que hablaré en la siguiente sección, una de ellas.

En mi caso, me ha sido de utilidad en la eliminación de duplicados, apoyandome en las técnicas que proporciona Numpy, y la generación de numeros aleatorios.

4.0.2. Matplotlib

Matplotlib es el estándar de facto para realizar todo tipo de gráficos en Python, y por tanto resulta obvio que, dada la capcidad de configuración y la calidad de resultados que esta nos ofrece, durante el estudio de los tiempos de ejecución, para la generación de las gráficas para los distintos tamaños de datos, la haya utilizado.

He optado por "plotear"la información en gráficos de barras, ya que es la mejor opción para entender visualmente, de manera clara y sencilla, la comparativa de tiempos, y de tarta, para representar el porcentaje del tiempo total que utiliza cada técnica.

También he importado *PdfPages* de *matplotlib.backends.backend_pdf* para crear el pdf en el que guardar todos los gráficos obtenidos.

4.0.3. Ctypes

Ctypes permite usar bibliotecas existentes (programadas con otros lenguajes) escribiendo simples wrappers de código en Python. Básicamente sirve como puente o interfaz entre el código Python y el código C, pues nos proporciona el mecanismo para realizar la llamada a la función C, mediante sus propios tipos de datos.

Antes de realizar la llamada debemos indicar el prototipo de la función externa (a la que se va invocar) mediante los atributos *argtypes* y *restype*.

4.0.4. Profiling

En ingeniería de software el análisis de rendimiento, comúnmenete llamado profiling, es la investigación del comportamiento de un programa usando información reunida desde el análisis dinámico del mismo. Esta técnica nos permite conocer el consumo de memoria de un programa, frecuencia y duración de las llamadas a funciones... ayudándonos a optimizarlo en algunos aspectos.

El profiling del tiempo de ejecución lo he realizado con el módulo *line_profiler*, mientras que el de la memoria lo he hecho con *memory_profiler*.

En la jerarquía de directorios del programa, concretamente en el directorio */doc/profiling* encontramos los distintos ficheros (visualizables) `main.py.lprof` y `mprofile_20200519085312.dat` que contienen el resultado del análisis del tiempo y memoria utilizada, respectivamente.

4.0.5. Git

Como muchas de las grandes cosas en esta vida, Git comenzó con un poco de destrucción creativa y una gran polémica.

En el 2005, la relación entre la comunidad que desarrollaba el kernel de Linux y la compañía que desarrollaba BitKeeper se vino abajo y la herramienta dejó de ser ofrecida de manera gratuita. Esto impulsó a la comunidad de desarrollo de Linux (y en particular a Linus Torvalds, el creador de Linux) a desarrollar su propia herramienta basada en algunas de las lecciones que aprendieron mientras usaban BitKeeper.

Para el control de versiones del proyecto he utilizado esta herramienta, pues me ha permitido manipular y gestionar el código en la línea de tiempo, reflejando los distintos cambios realizados. Además cuenta con una versión gráfica llamada gitg, en la que he podido ir apreciando con los ojos los estados de las ramas que he ido creando.

En la siguiente figura muestro el esato final del proyecto, justo antes de importar este propio fichero Latex en la carpeta de documentación.

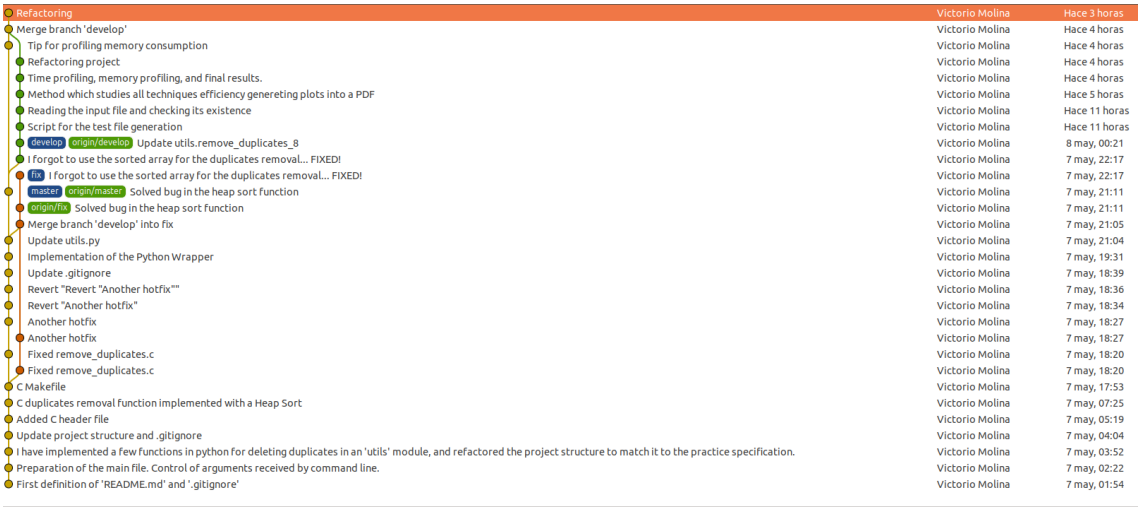


Figura 1: Estado final del proyecto git

4.0.6. GitHub

He subido el proyecto a mi repositorio online de GitHub. Este es el enlace al mismo: https://github.com/VictorioMolina/Python-remove-duplicates-profiling

5. Análisis del orden de complejidad de la función C

El algoritmo de eliminación de duplicados que he implementado es sencillo, primero ordenamos el vector de entrada (para así poder eliminar los duplicados de una pasada) mediante uno de los mejores algoritmos de ordenación en lo que a tiempo y memoria se refiere, para luego, como ya he dicho, realizar la tarea de unificar la estructura.

El algoritmo de ordenación que he utilizado es el *Heap Sort*, el cual tiene un orden de complejidad, **para todos los casos**, de *n*log(n)* refiriendonos al tiempo de ejecución, y constante en cuanto a memoria. Dentro de lo que cabe es un buen orden, pues obtenemos más eficiencia para tamaños de datos gigantescos en comparación al resto de algoritmos.

Sumando el peso de la ordenación y de la unificación de la estructura, finalmente obtenemos un orden de complejidad n*log(n) para el tiempo, y n para la

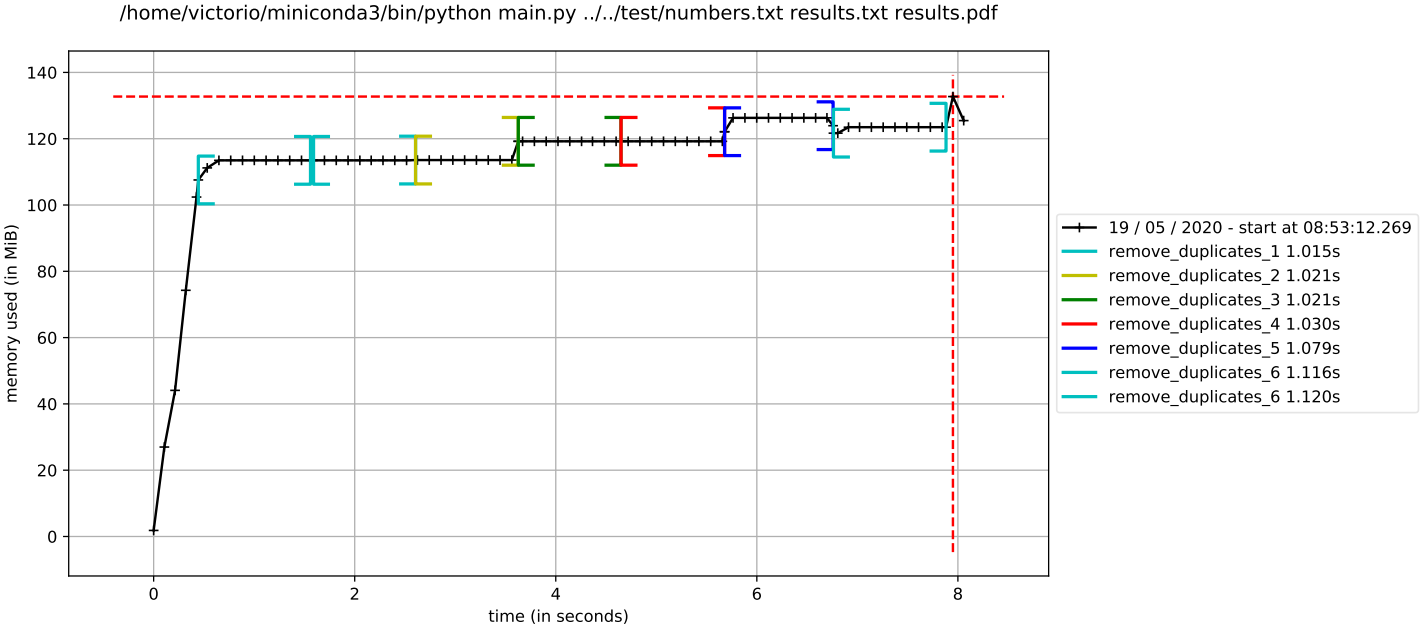
memoria. Este último podría haber sido constante si no hubiese realizado la copia del vector que recibimos como argumento, pero he preferido no realizarla para evitar la destrucción del vector original y realizar los ajustes necesarios para que Python no se vuelva loco.

El código C está correctamente comentado, por lo que no veo necesario divagar mucho en el funcionamiento del mismo, pues es un tema que está mas relacionado con la algoritmia, y no con la esencia de esta práctica.

6. Resultados del profiling

6.1. Memory profiling

A continuación, en la siguiente página, muestro la gráfica resultante del análisis del uso de memoria realizado por las funciones python de juguete y el wrapper. En ella se pueden contrastar los resultados junto a la explicación que he realizado en el apartado anterior. Este gráfico ha sido generado a través del archivo **/doc/profiling/mprofile_20200519085312.dat**


/home/victorio/miniconda3/bin/python main.py ../../test/numbers.txt results.txt results.pdf
memory used (in MiB)
time (in seconds)
0
20
40
60
80
100
120
140
0
2
4
6
8
19 / 05 / 2020 - start at 08:53:12.269
remove_duplicates_1 1.015s
remove_duplicates_2 1.021s
remove_duplicates_3 1.021s
remove_duplicates_4 1.030s
remove_duplicates_5 1.079s
remove_duplicates_6 1.116s
remove_duplicates_6 1.120s

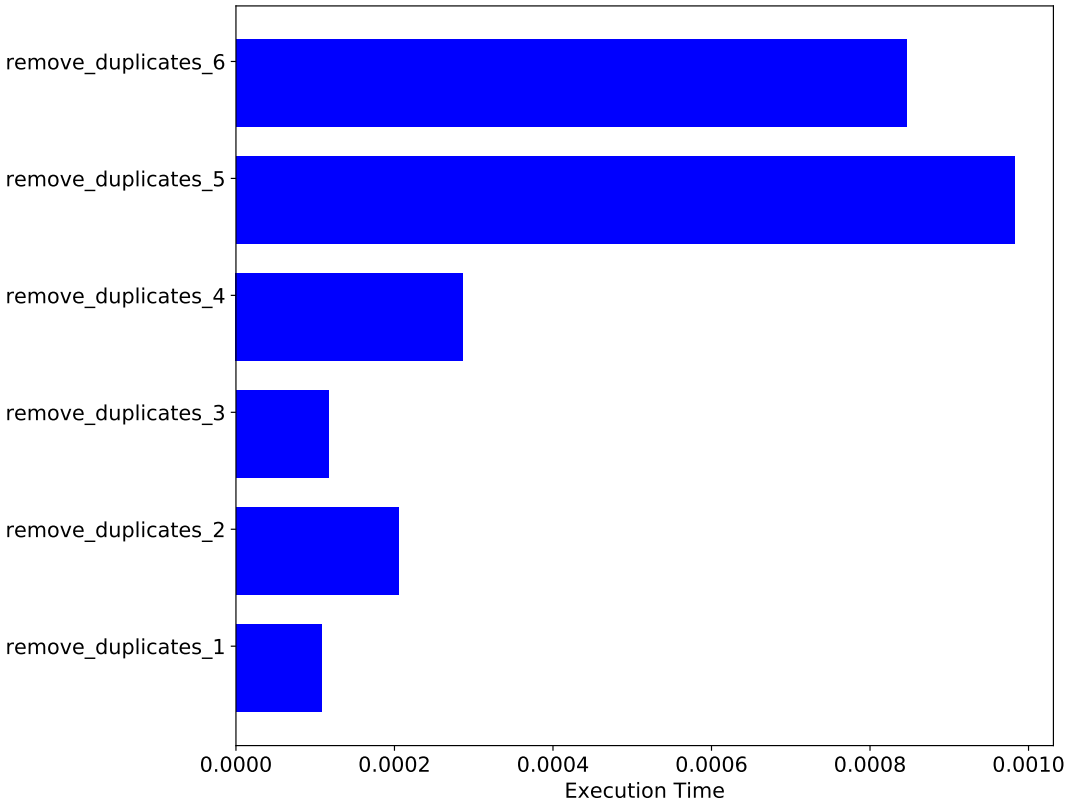
6.1.1. Linear profiling y gráficas generadas con Matplotlib

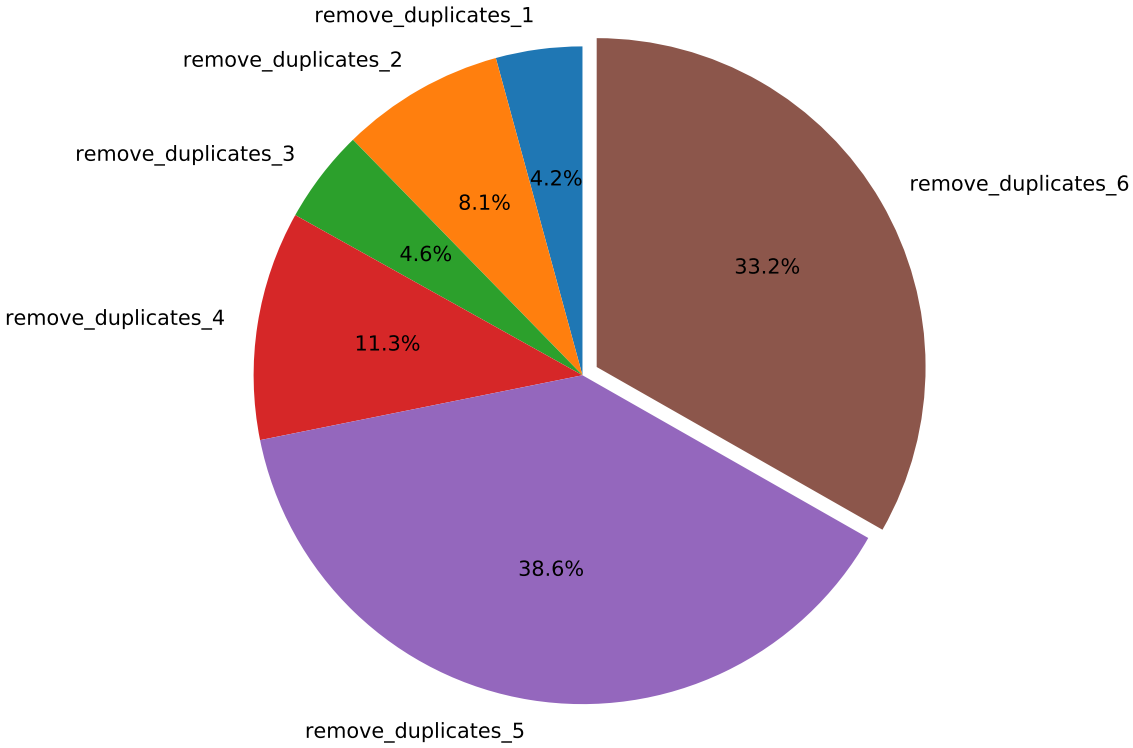
El profiling del tiempo de ejecución lo he realizado con el módulo *linear_profiling*, tal y como he comentado anteriormente. En el directorio **/doc/profiling** se puede encontrar su resultado, concretamente en el fichero **main.py.lprof** Para terminar con el contraste de resultados, voy a incluir una serie de figuras (creadas con Matplotlib) compuestas por dos gráficos, uno de barras y otro de tarta, como ya comenté anteriormente.

Tal y como se comenta en el enunciado, el análisis lo he realizado, ademas de usando el módulo linear_profiling, generando gráficas que muestran el tiempo usado por cada técnica para los distintos tamaños de datos, de 2000 en 2000, hasta llegar al tamaño total de la lista de elementos del fichero de entrada.

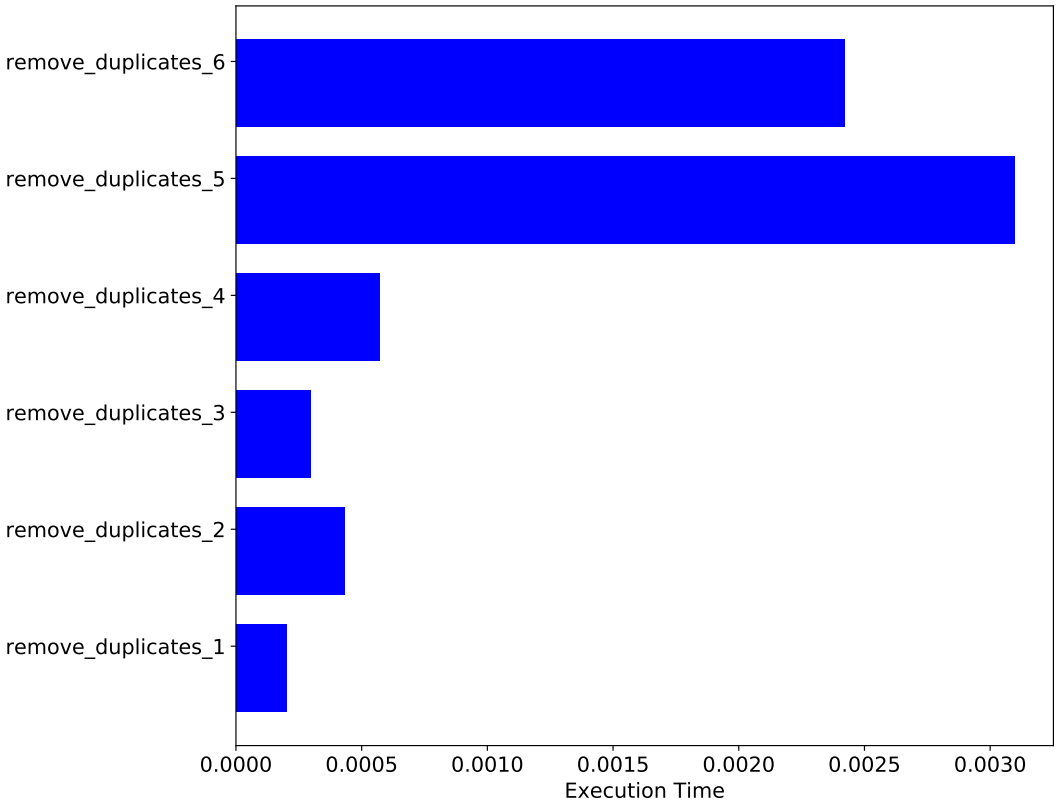
Pd: el siguiente contenido lo almacena el programa en el fichero correspondiente al tercer argumento que le pasa el usuario.

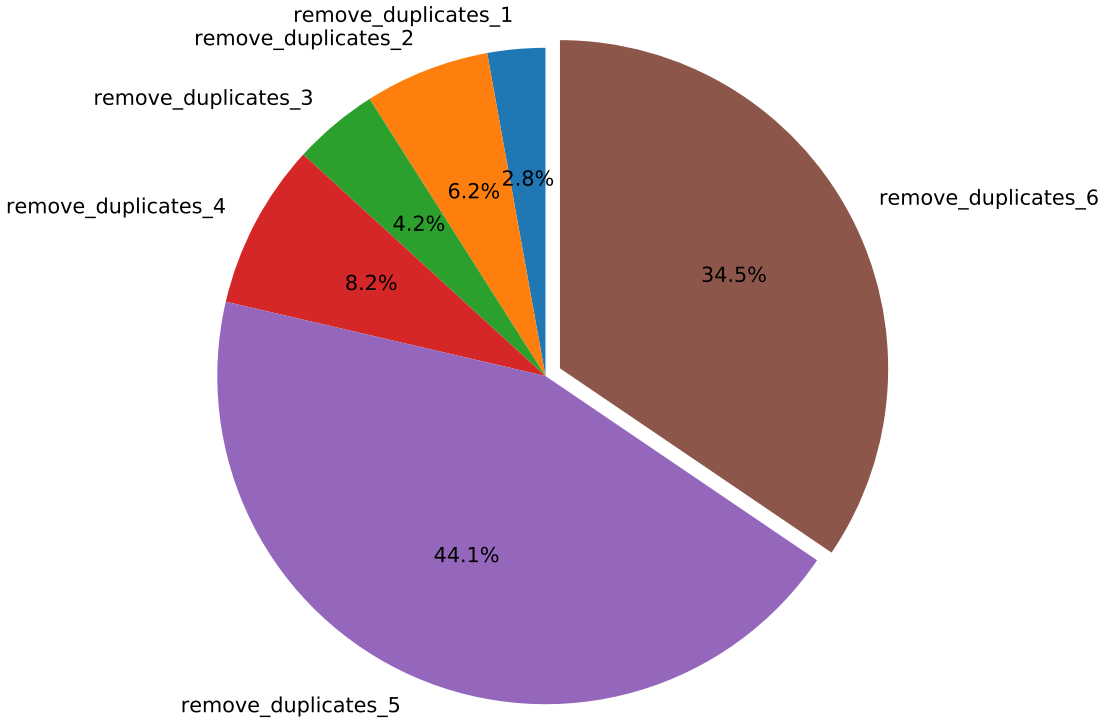
TIMES FOR THE FIRST 2000 NUMBERS



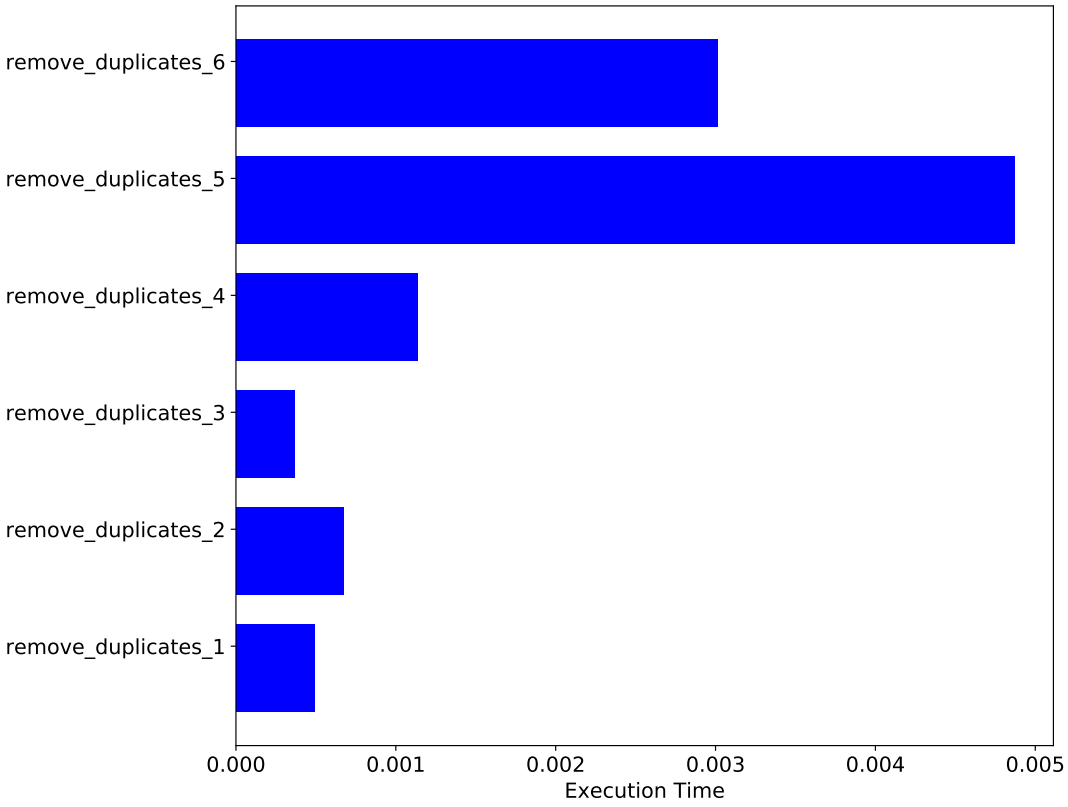


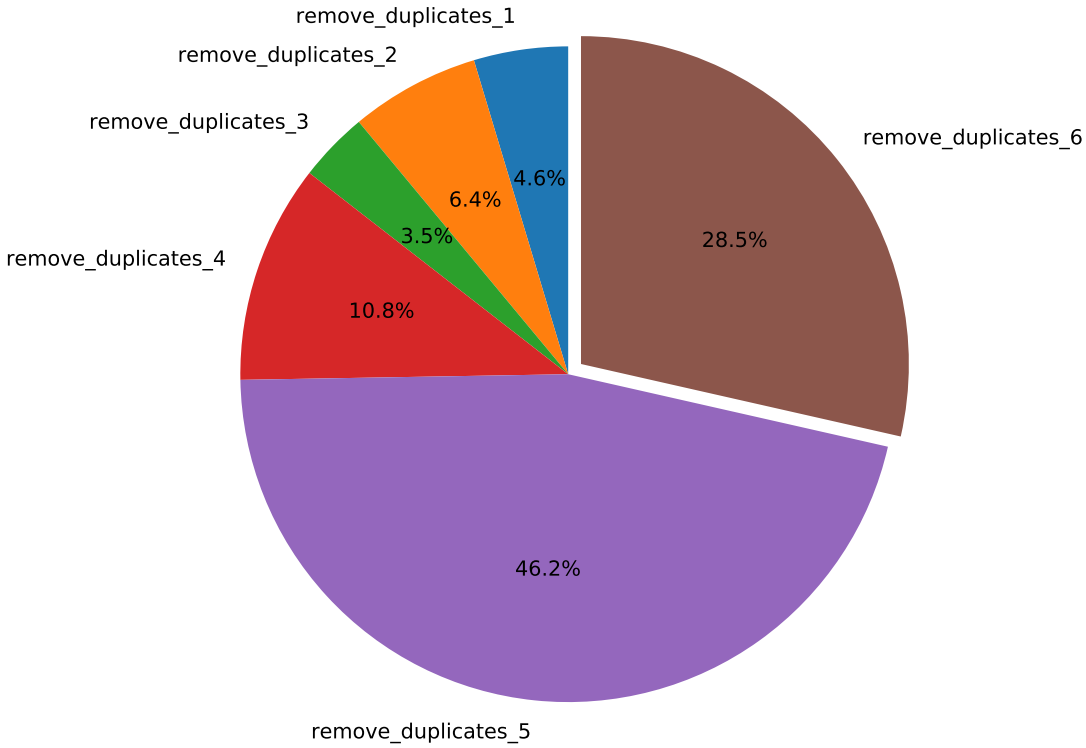
TIMES FOR THE FIRST 4000 NUMBERS



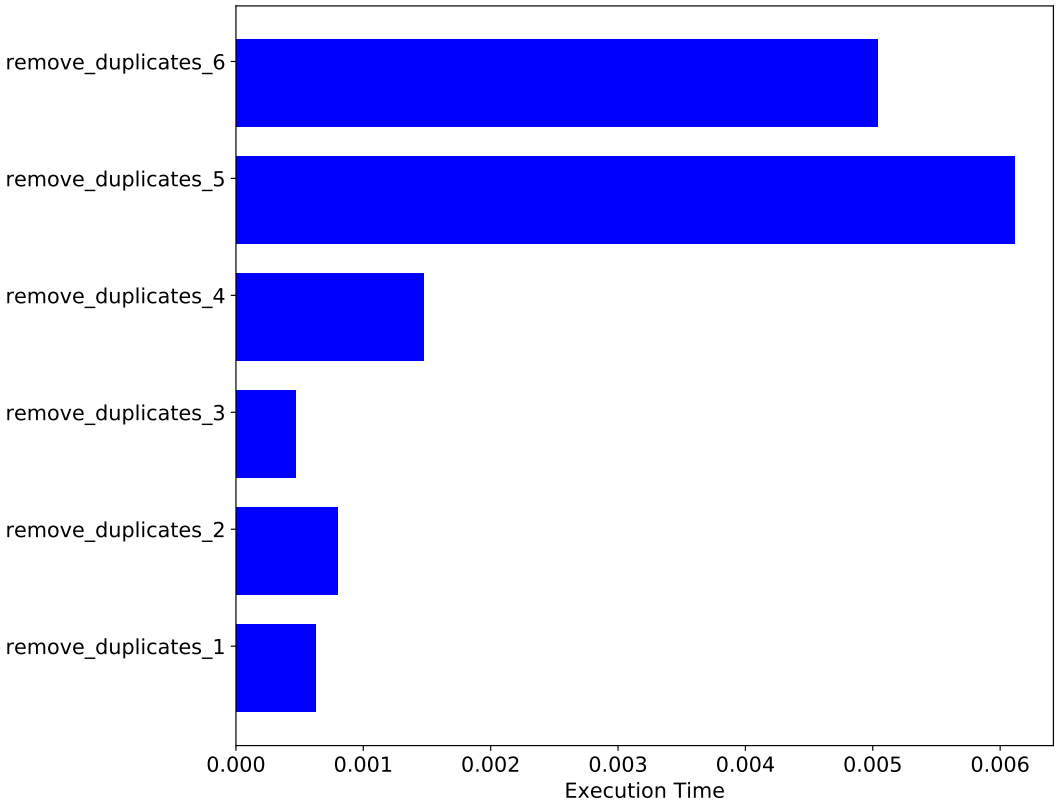


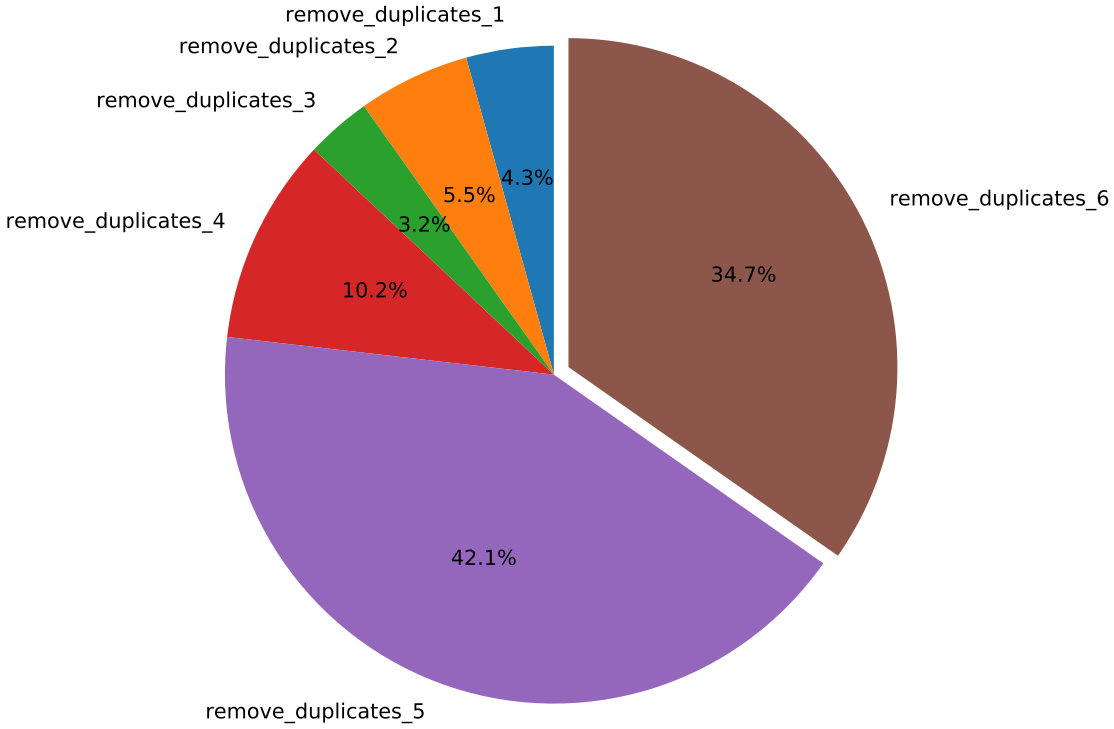
TIMES FOR THE FIRST 6000 NUMBERS



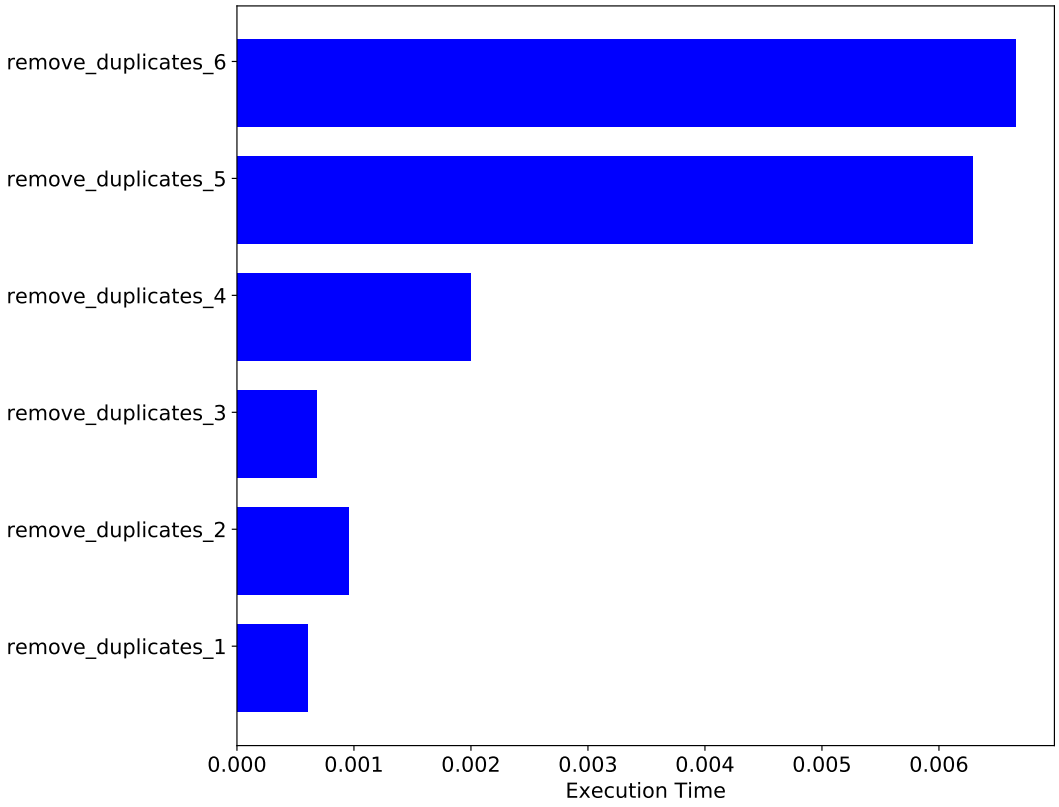


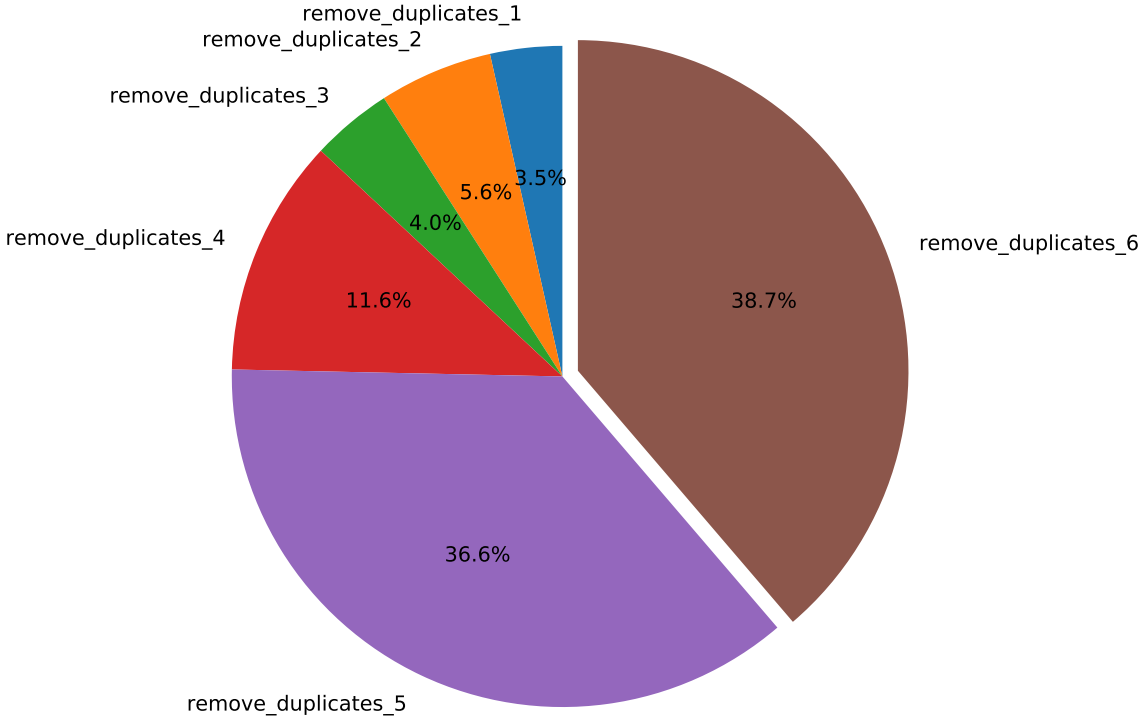
TIMES FOR THE FIRST 8000 NUMBERS



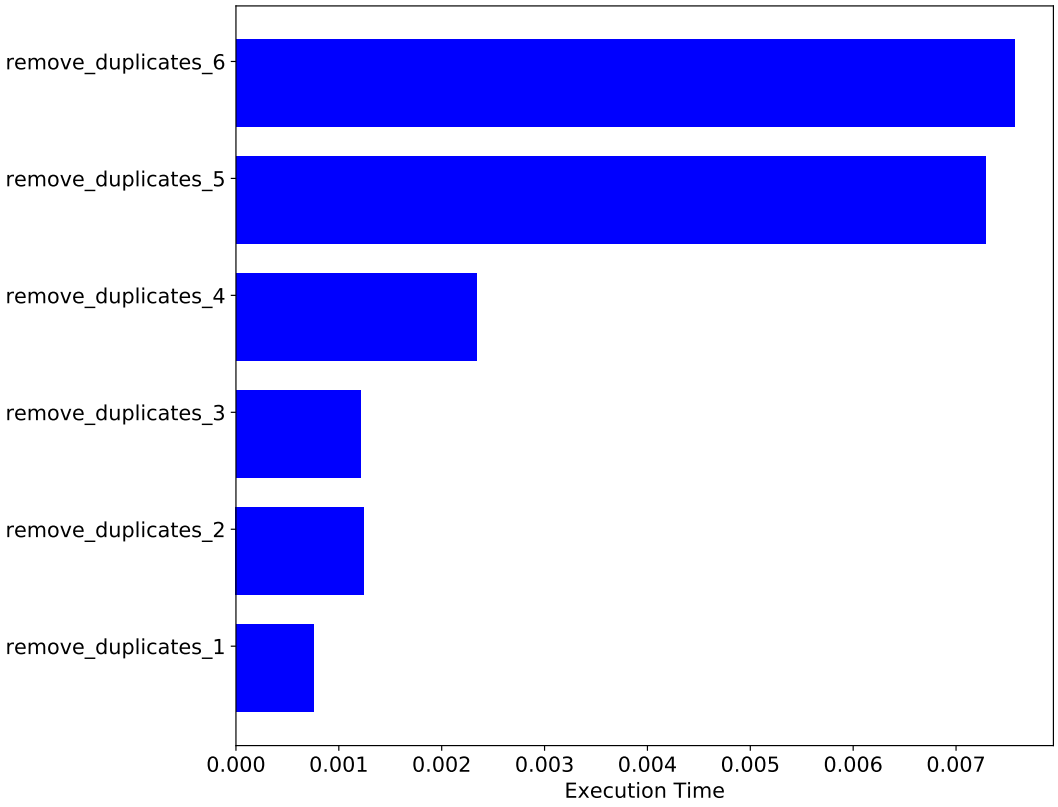


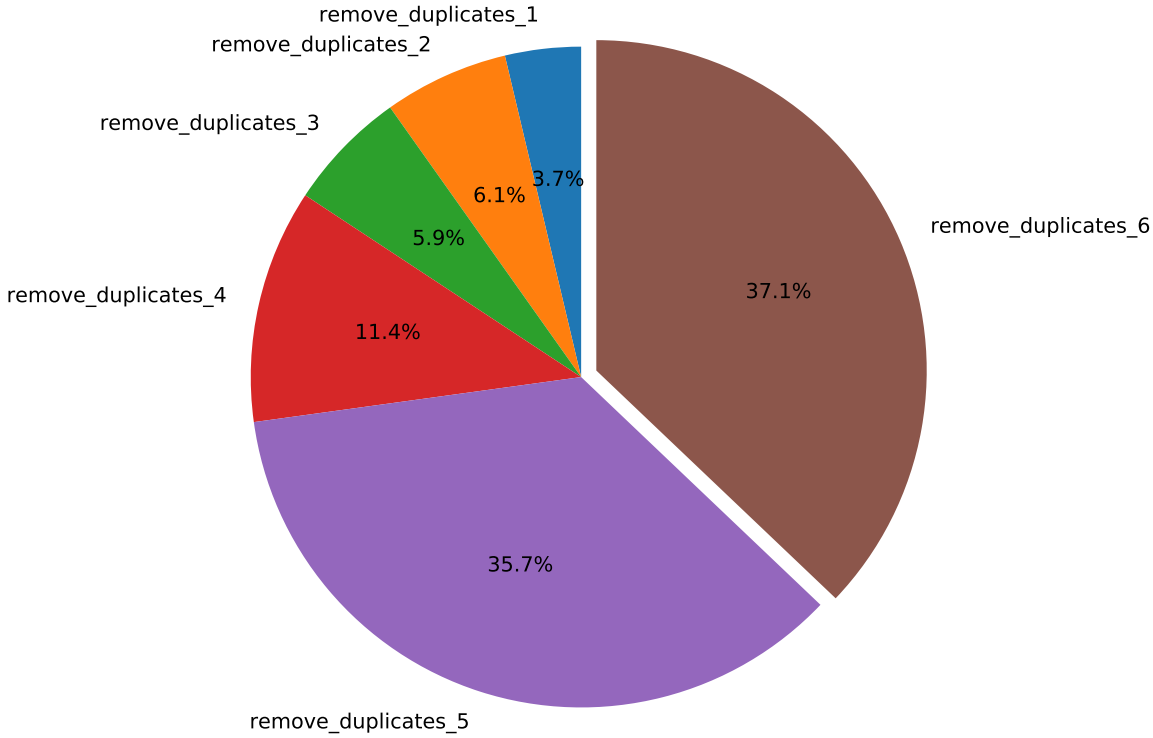
TIMES FOR THE FIRST 10000 NUMBERS



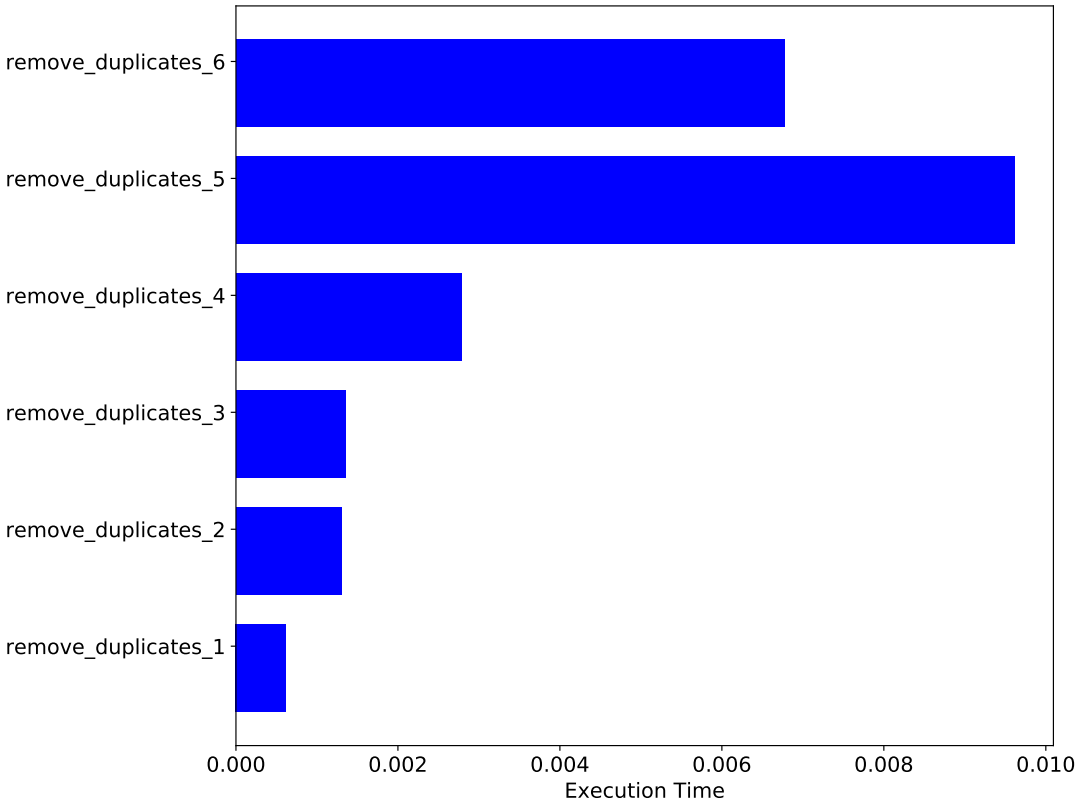


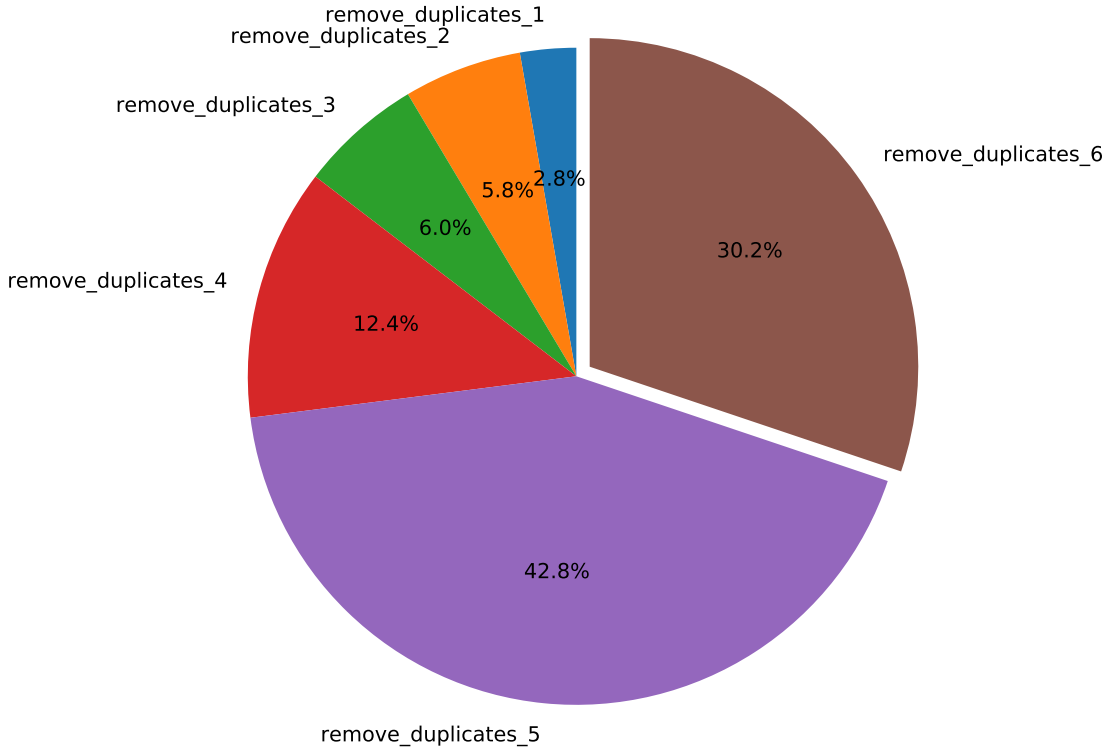
TIMES FOR THE FIRST 12000 NUMBERS



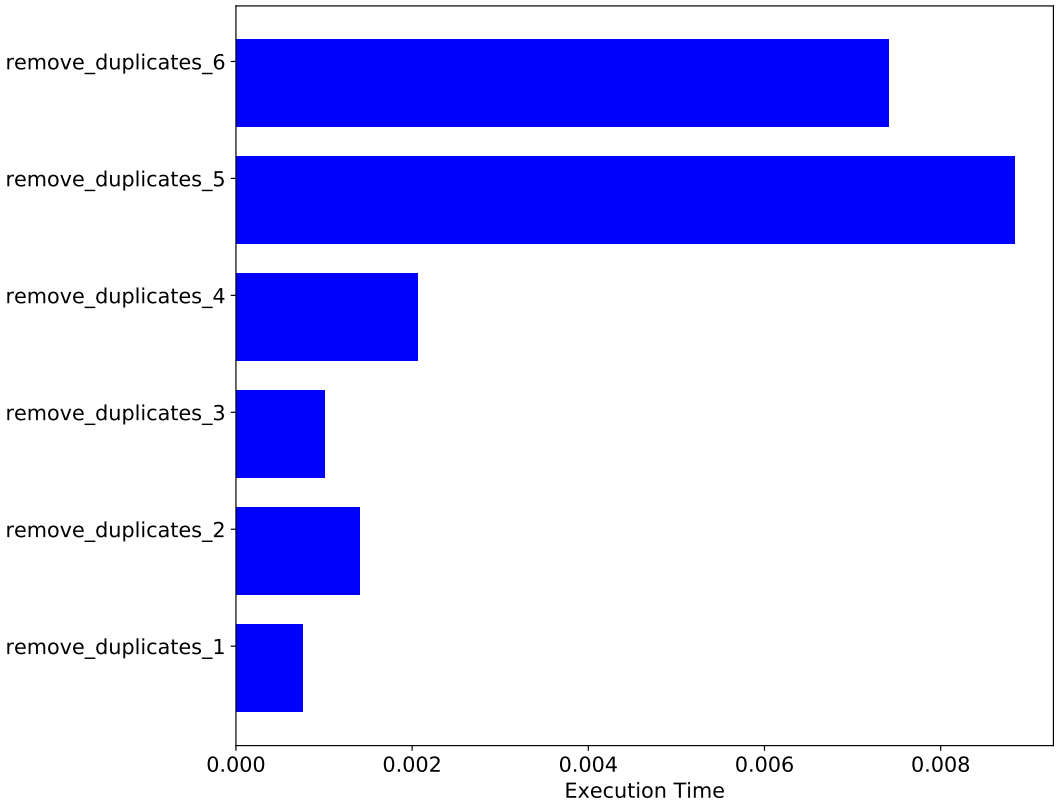


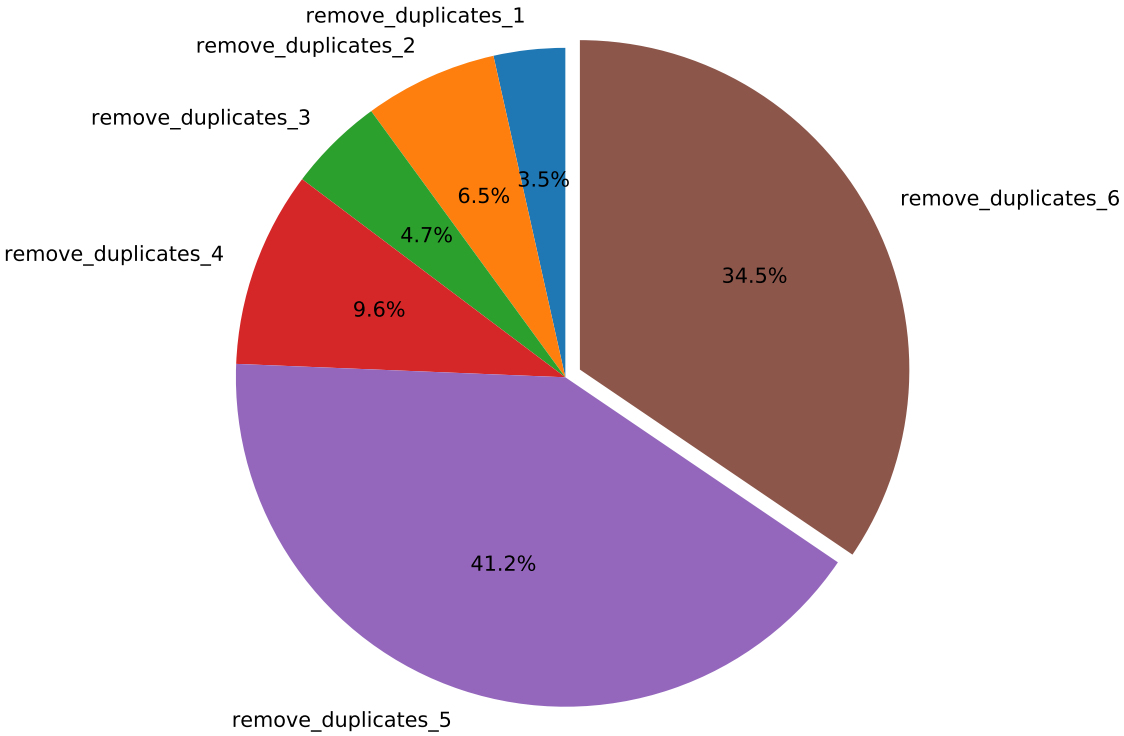
TIMES FOR THE FIRST 14000 NUMBERS



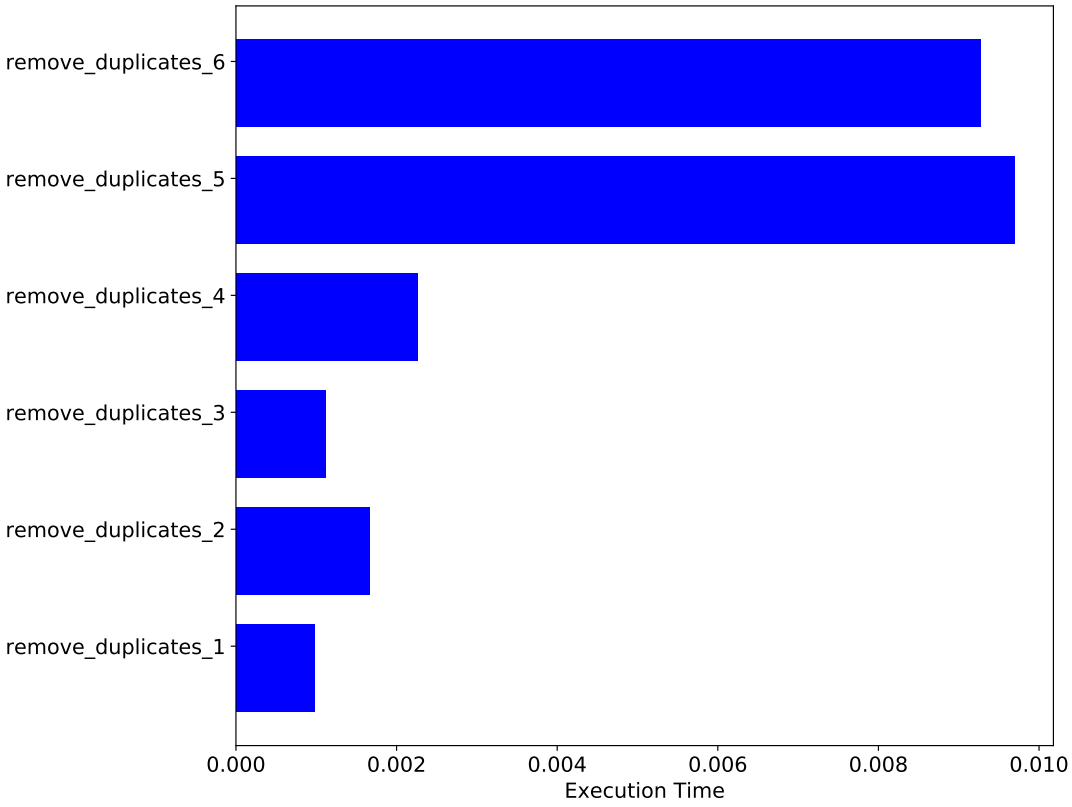


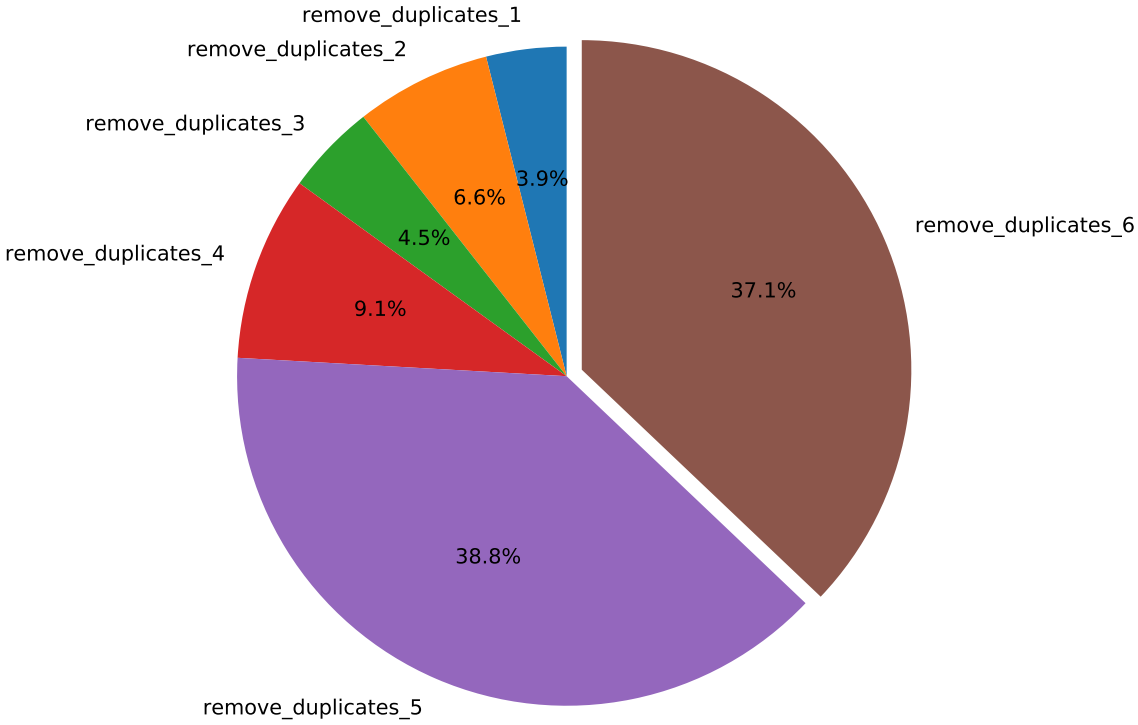
TIMES FOR THE FIRST 16000 NUMBERS



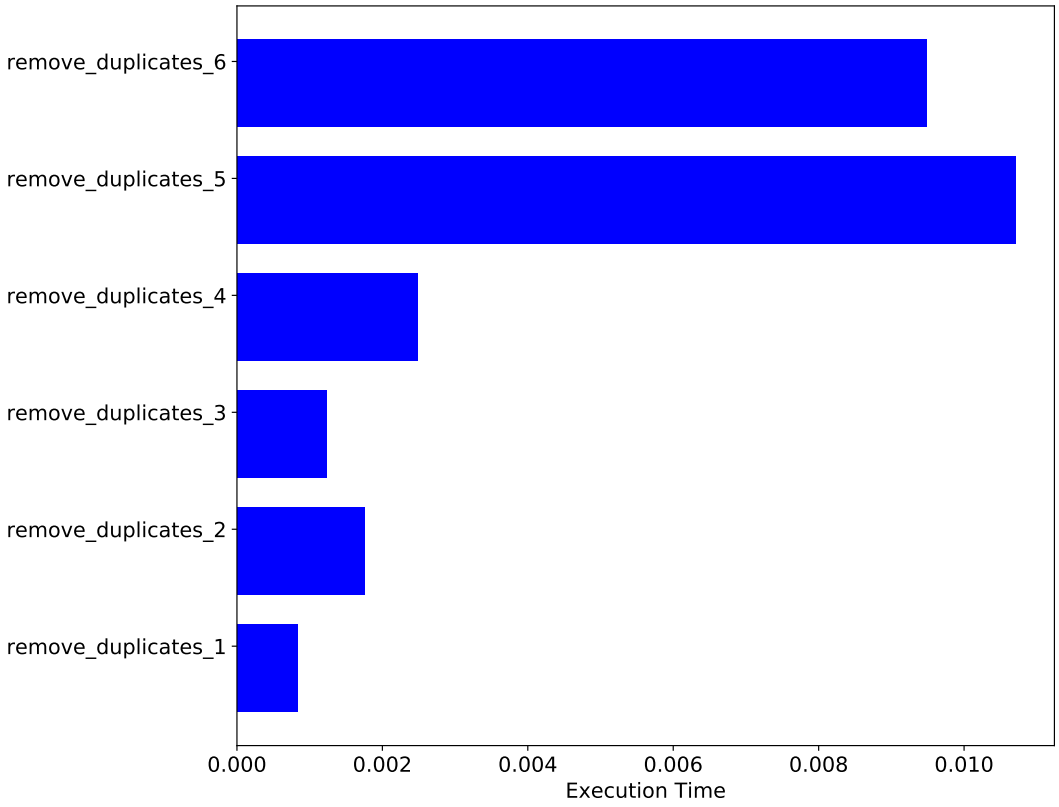


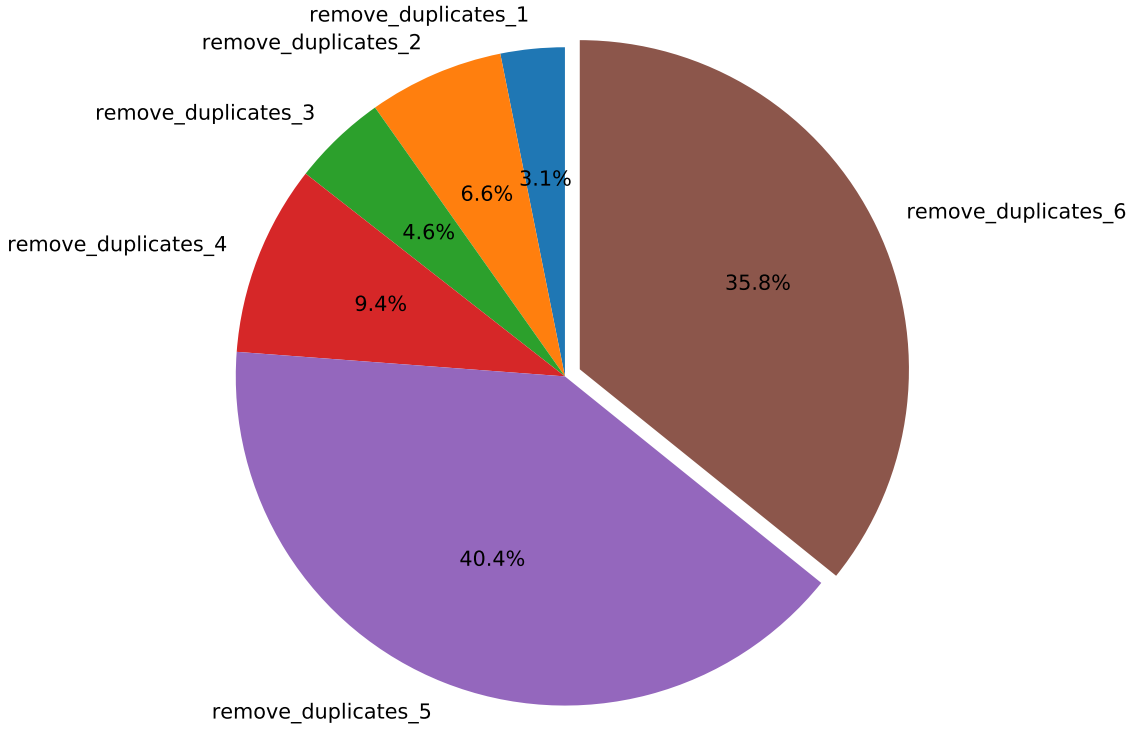
TIMES FOR THE FIRST 18000 NUMBERS



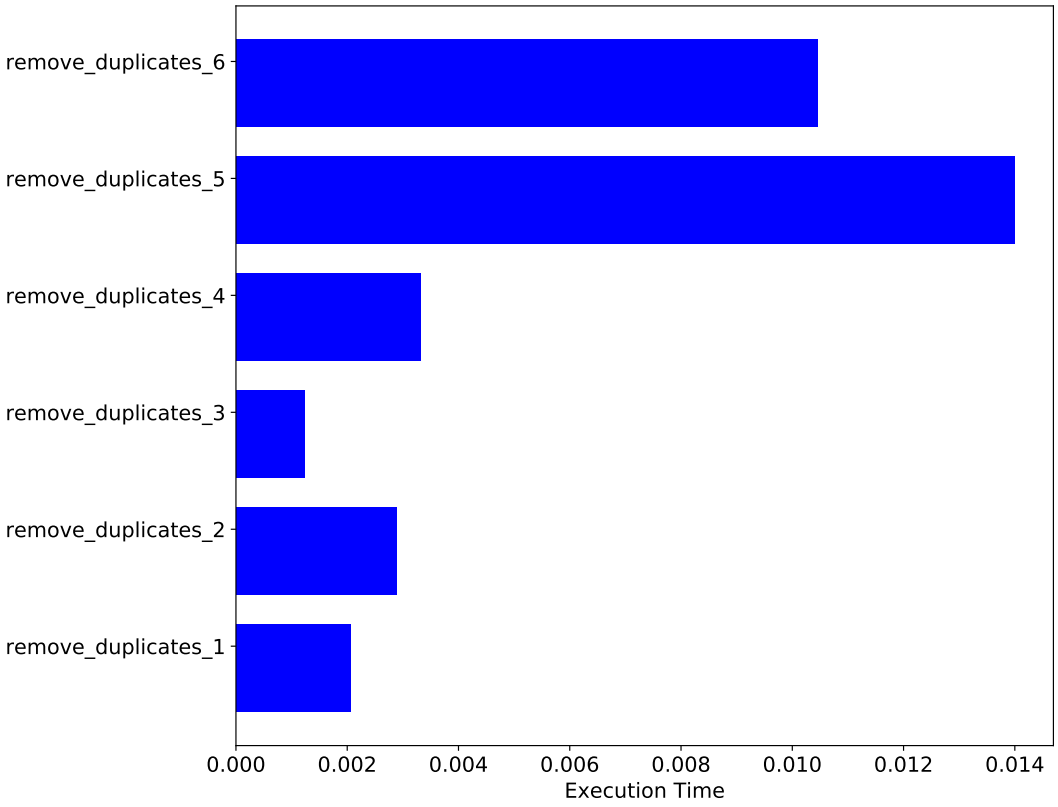


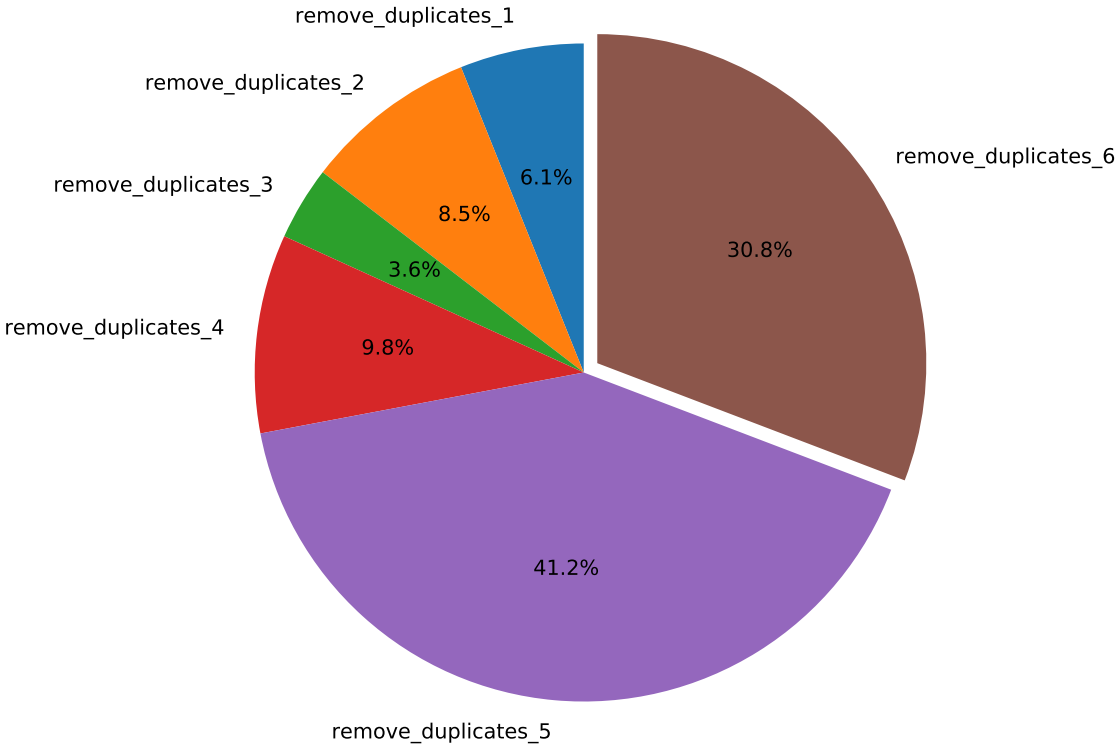
TIMES FOR THE FIRST 20000 NUMBERS



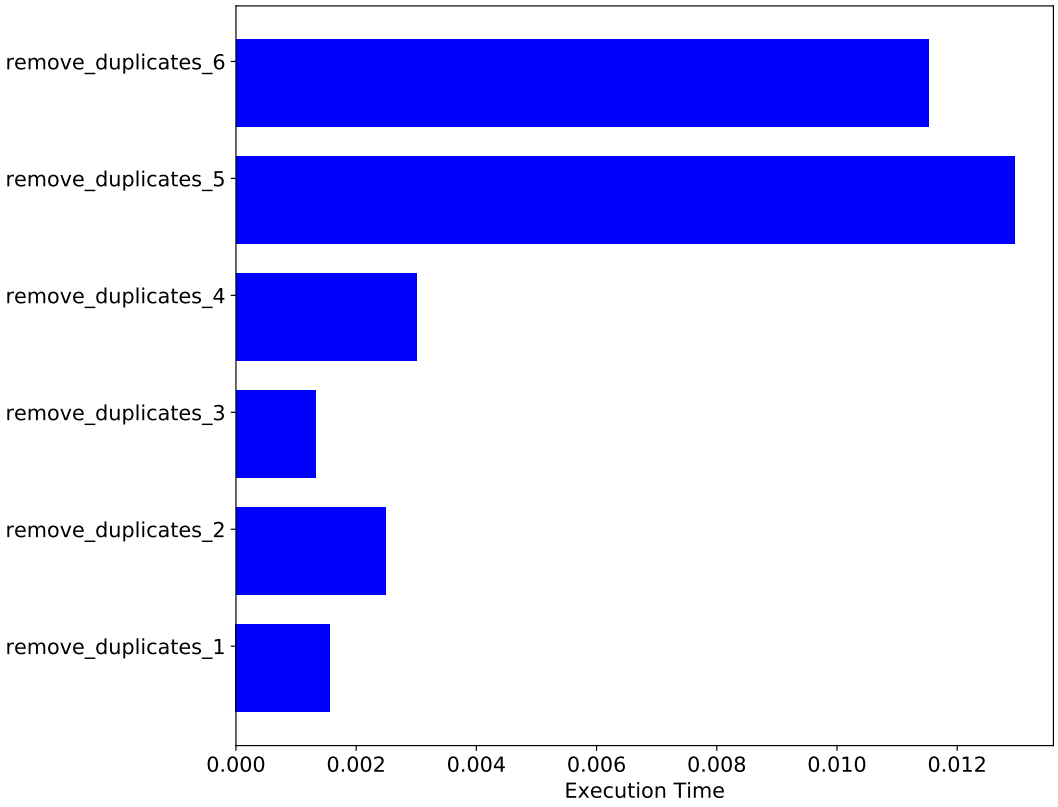


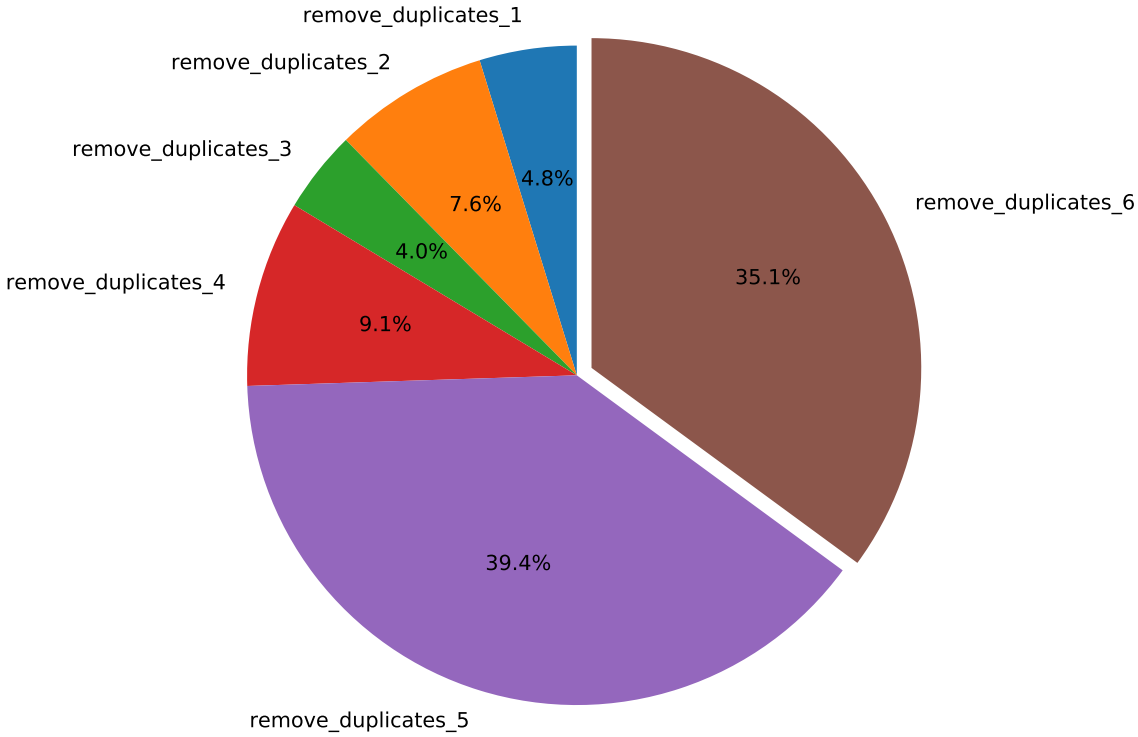
TIMES FOR THE FIRST 22000 NUMBERS



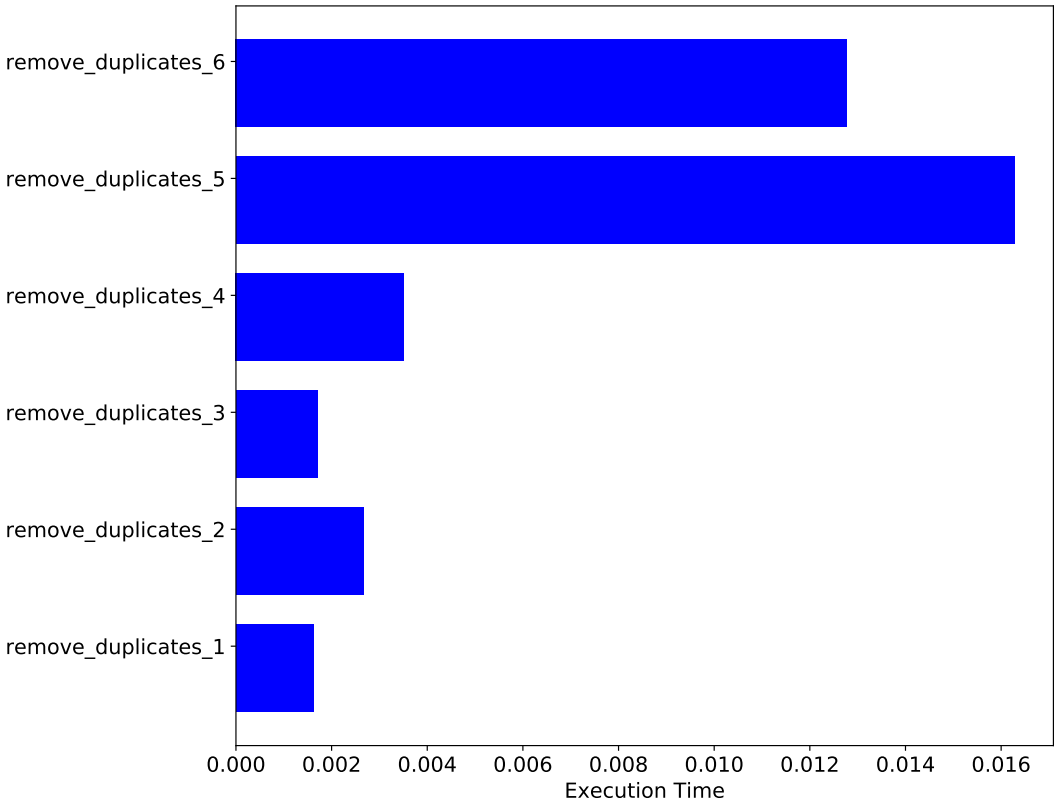


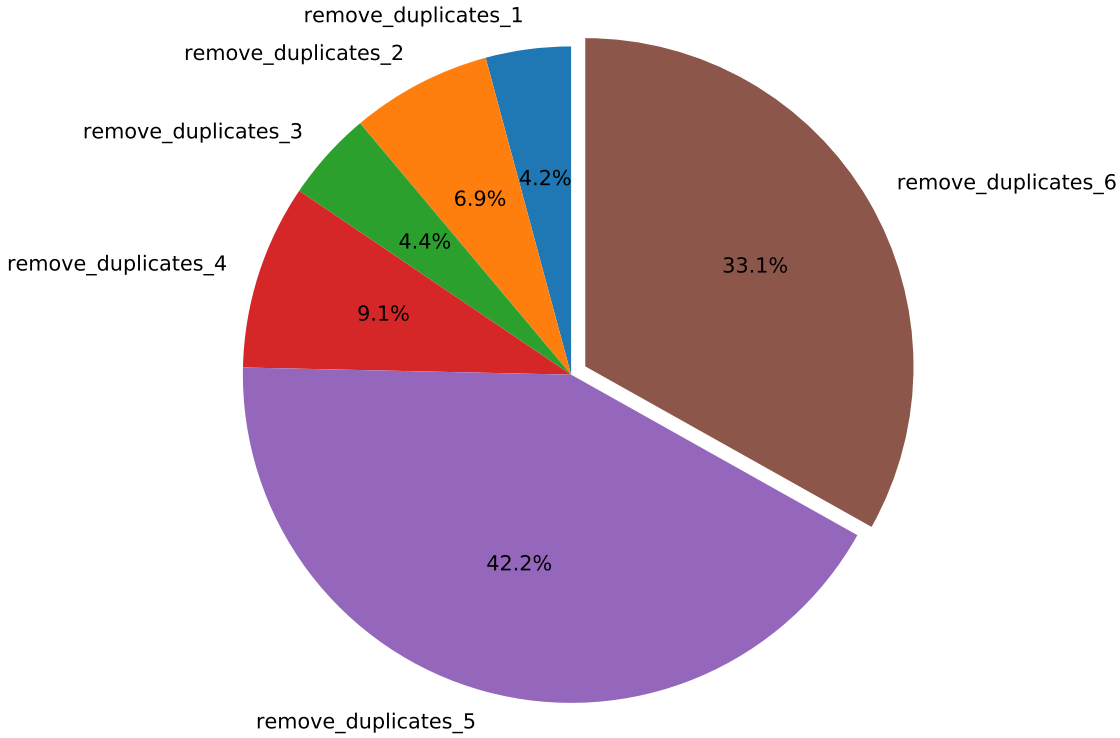
TIMES FOR THE FIRST 24000 NUMBERS



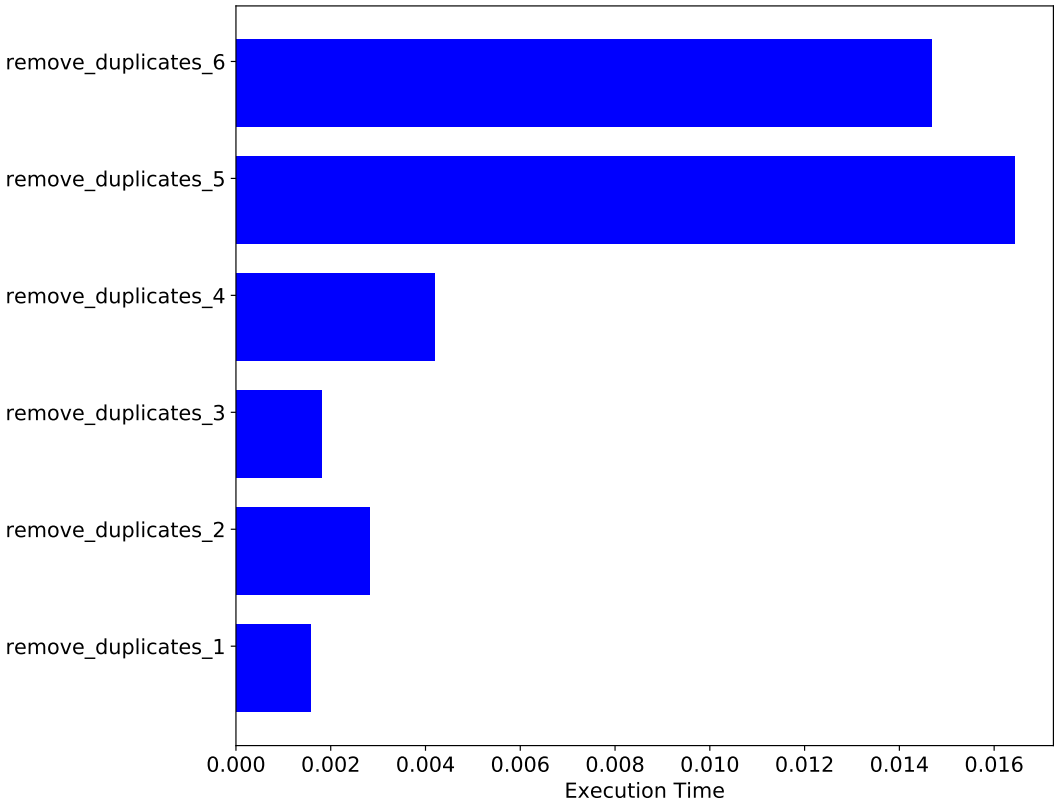


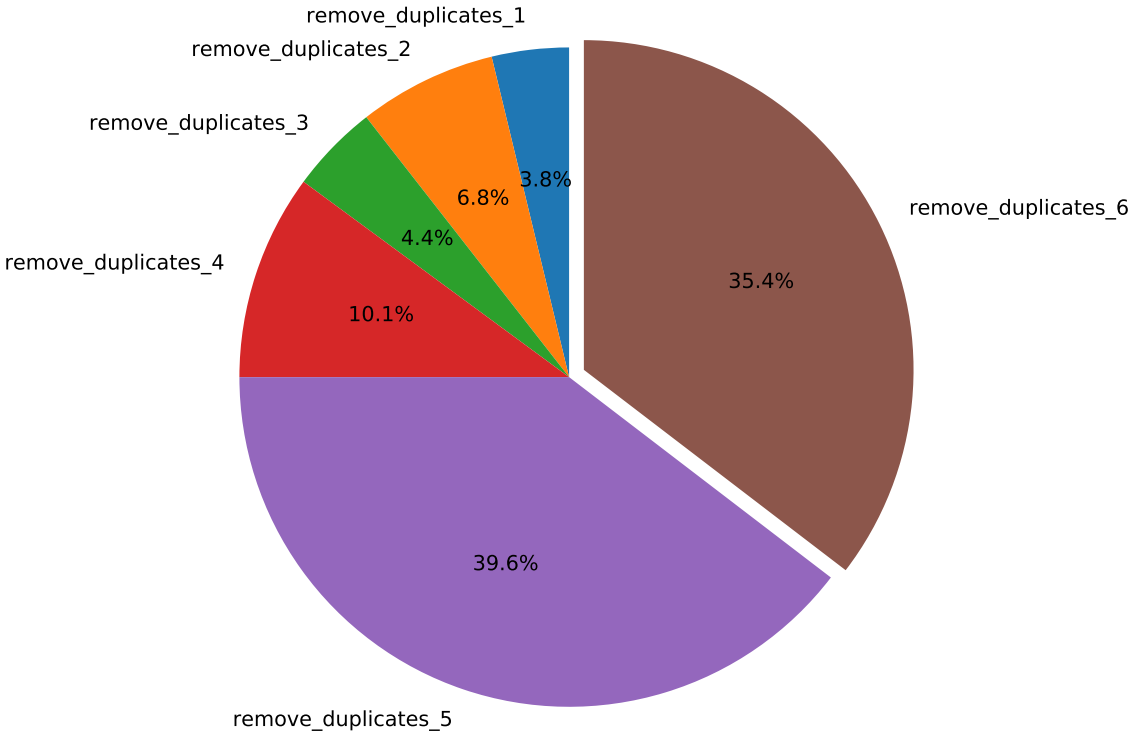
TIMES FOR THE FIRST 26000 NUMBERS



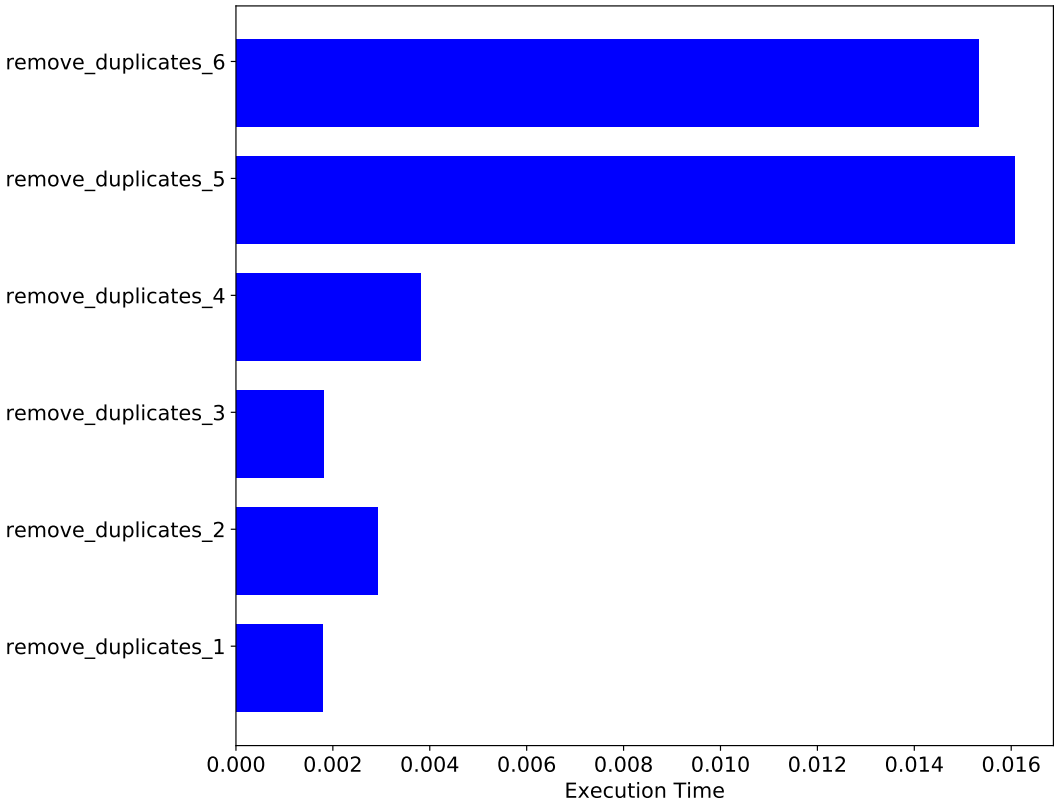


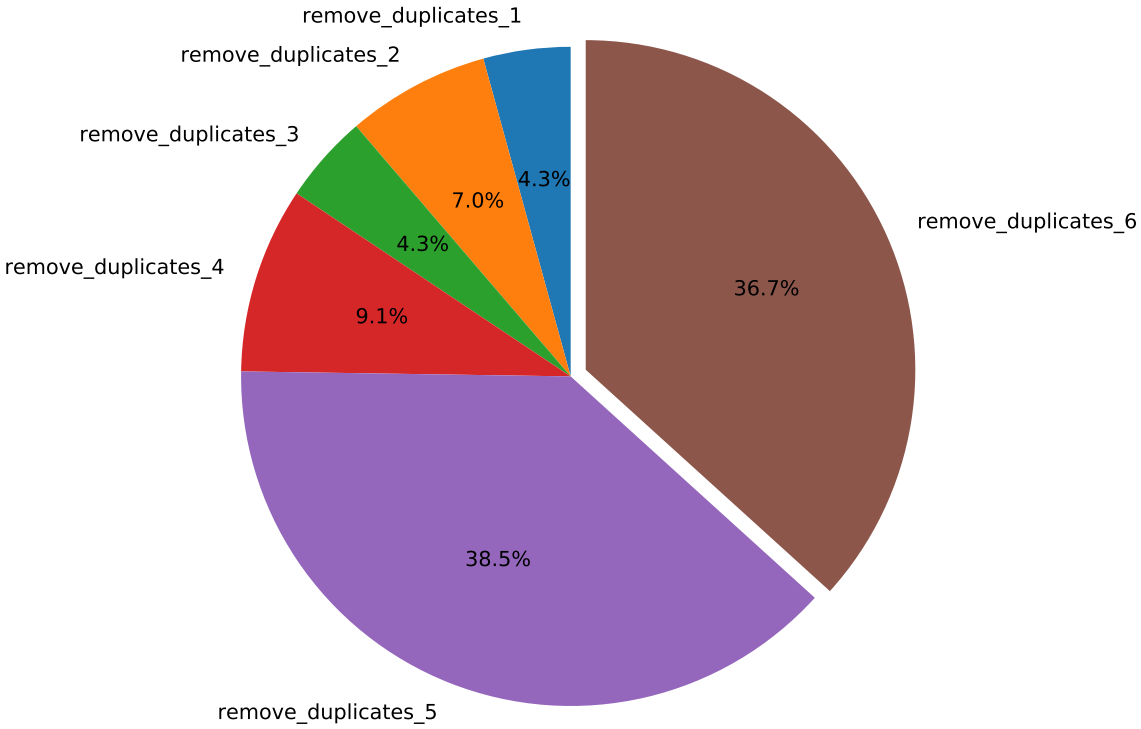
TIMES FOR THE FIRST 28000 NUMBERS



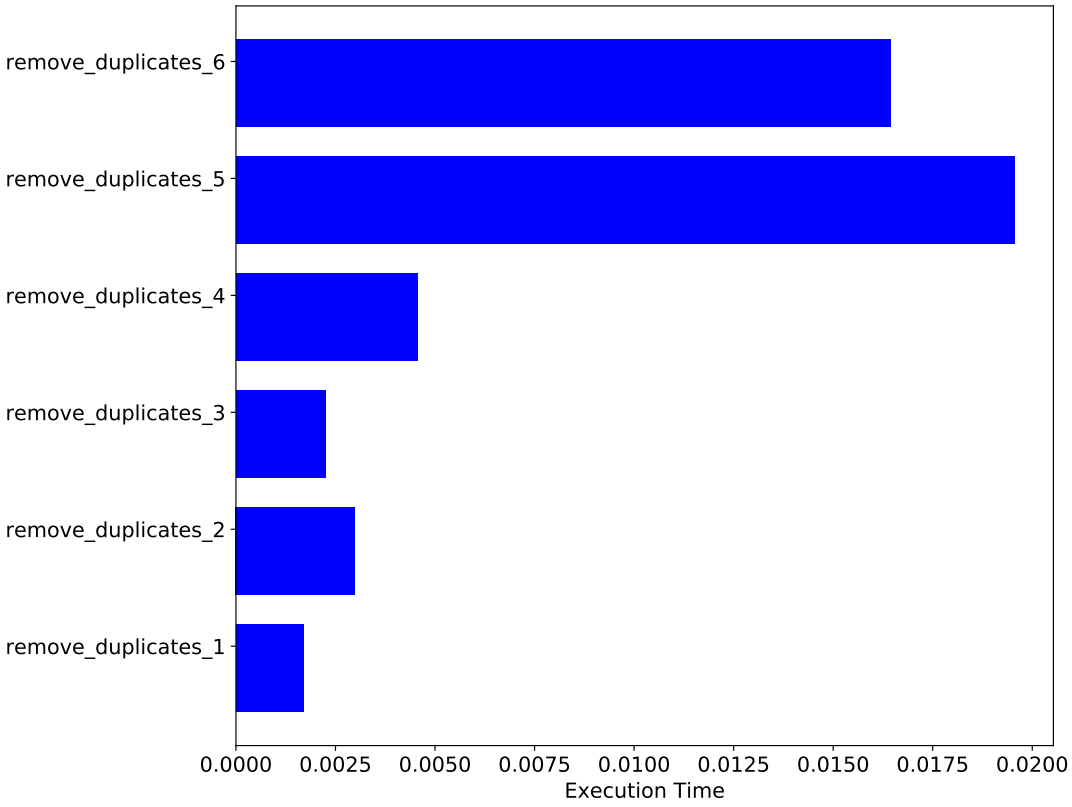


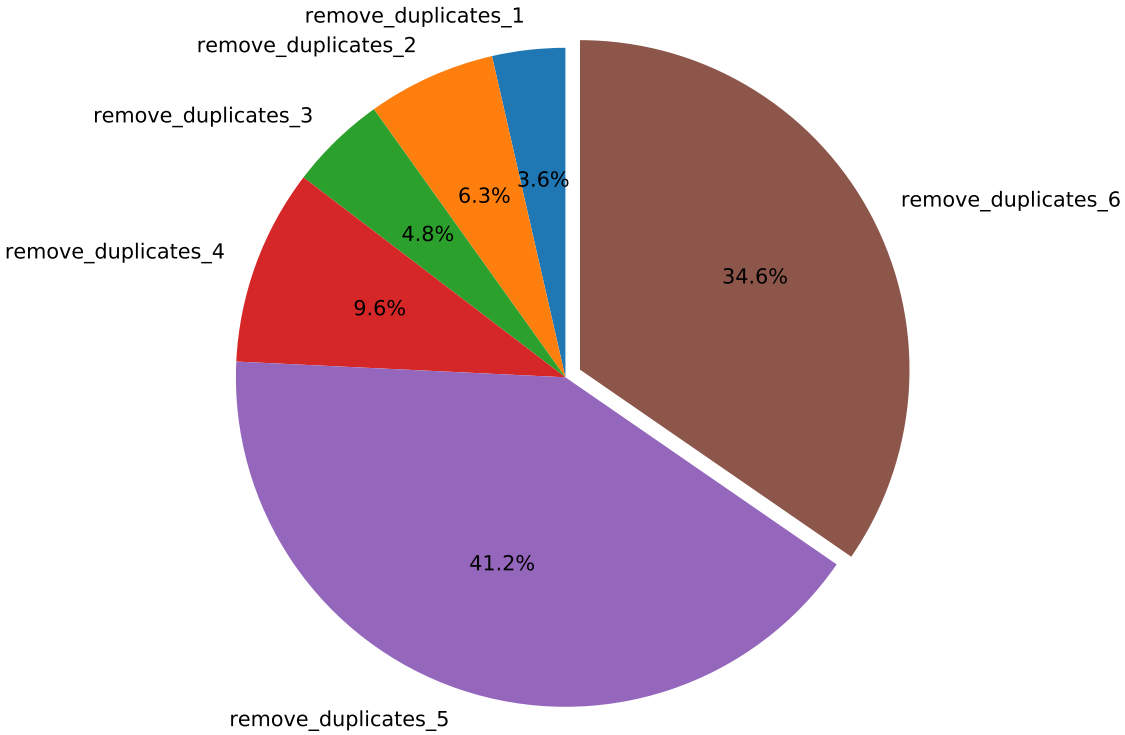
TIMES FOR THE FIRST 30000 NUMBERS



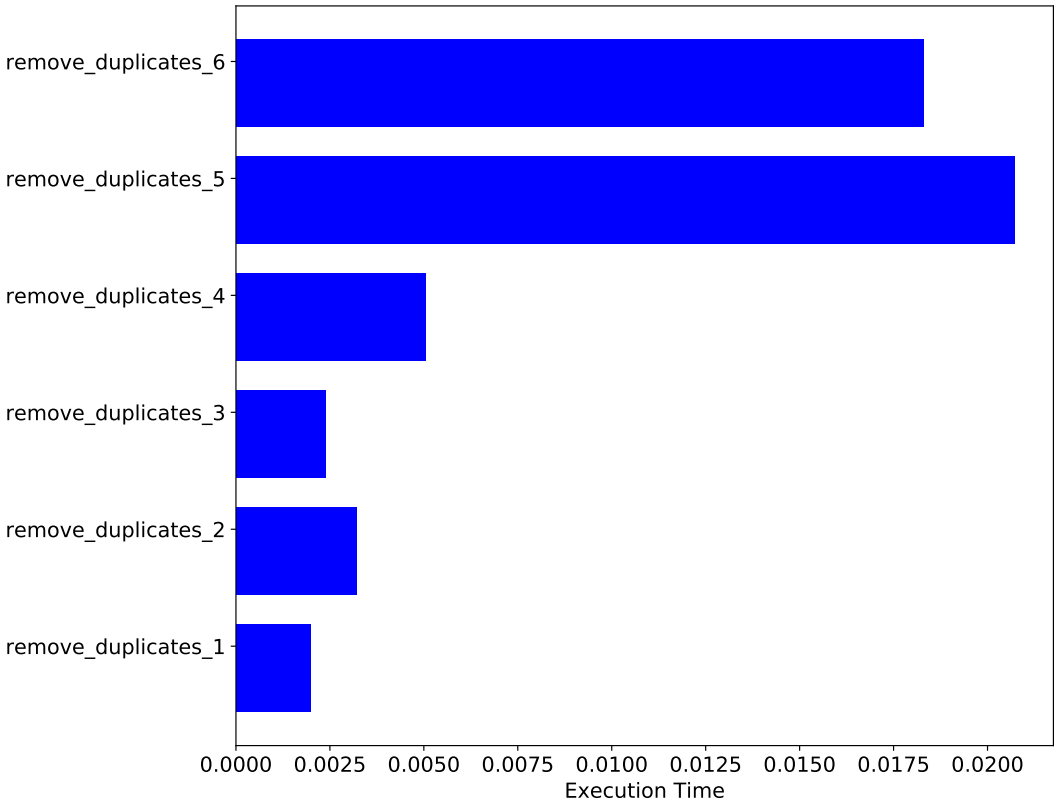


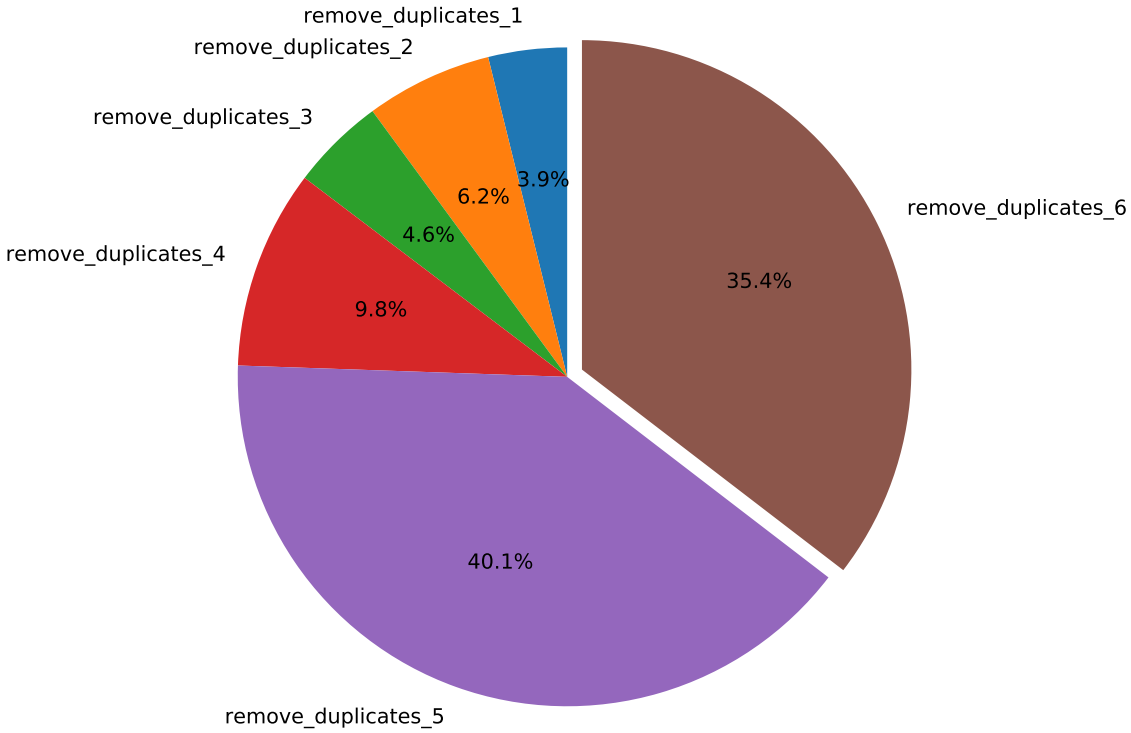
TIMES FOR THE FIRST 32000 NUMBERS



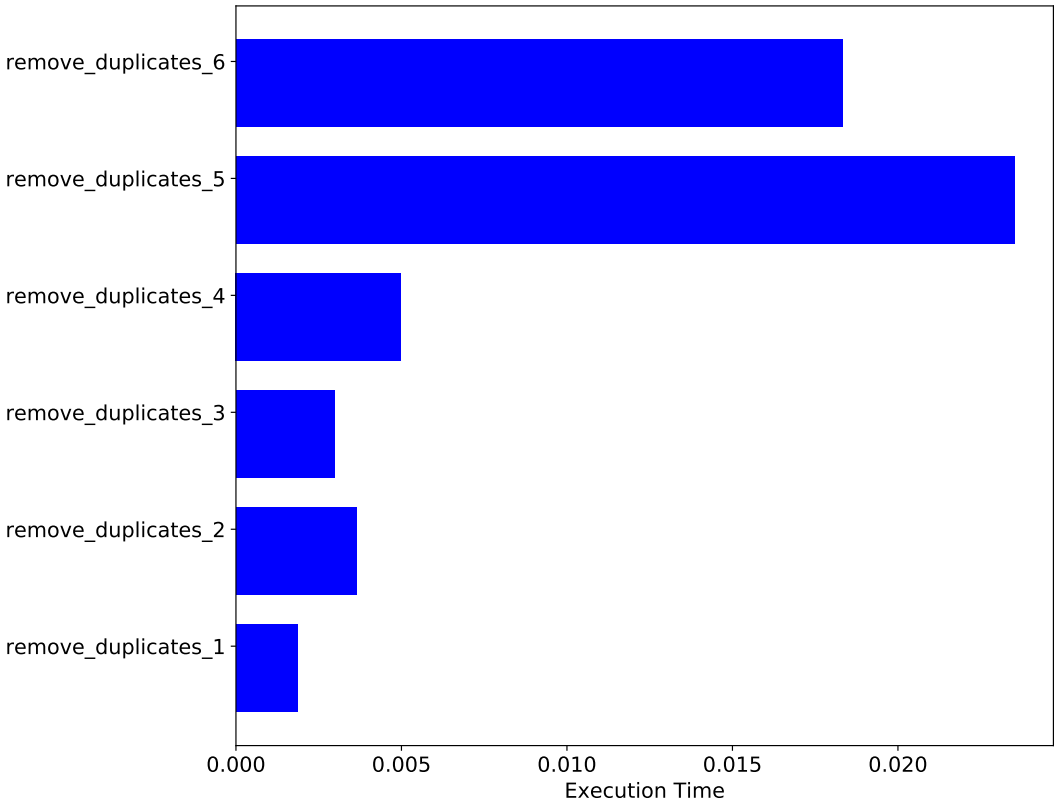


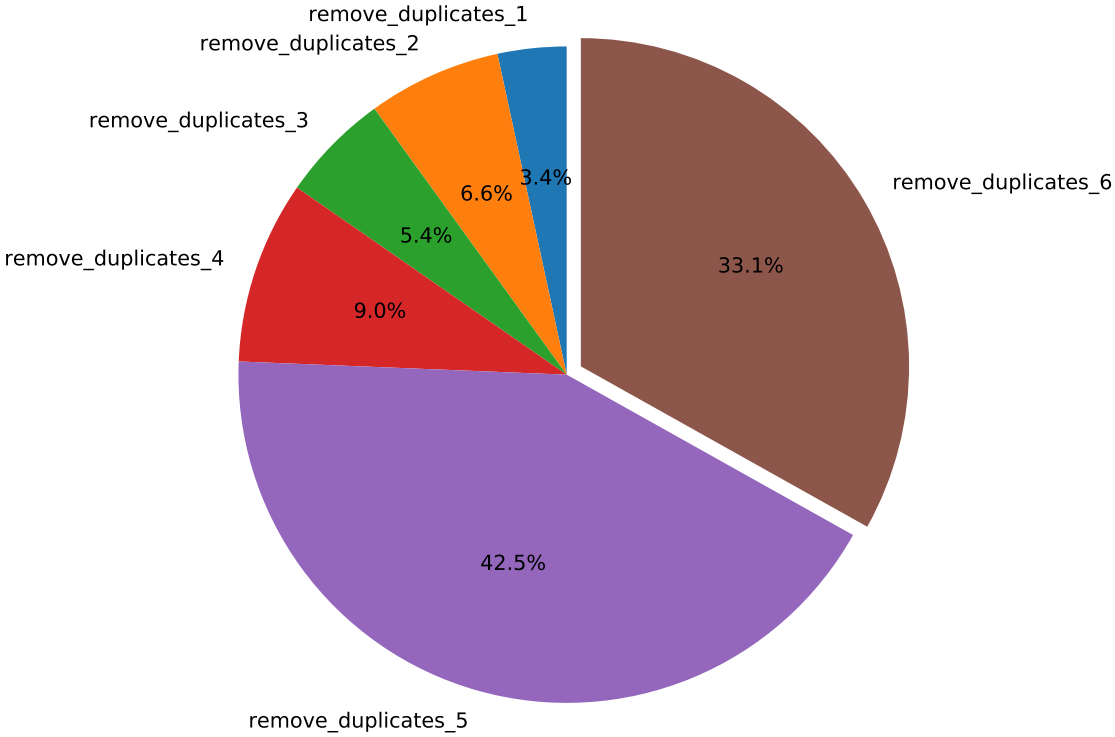
TIMES FOR THE FIRST 34000 NUMBERS



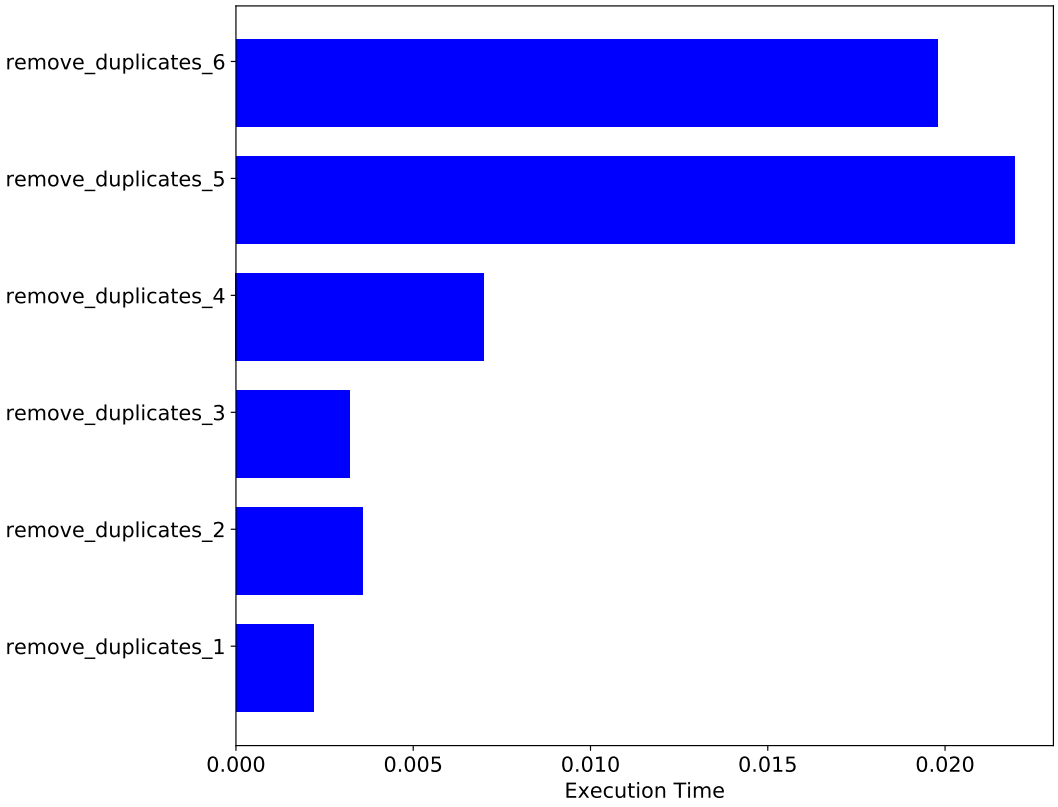


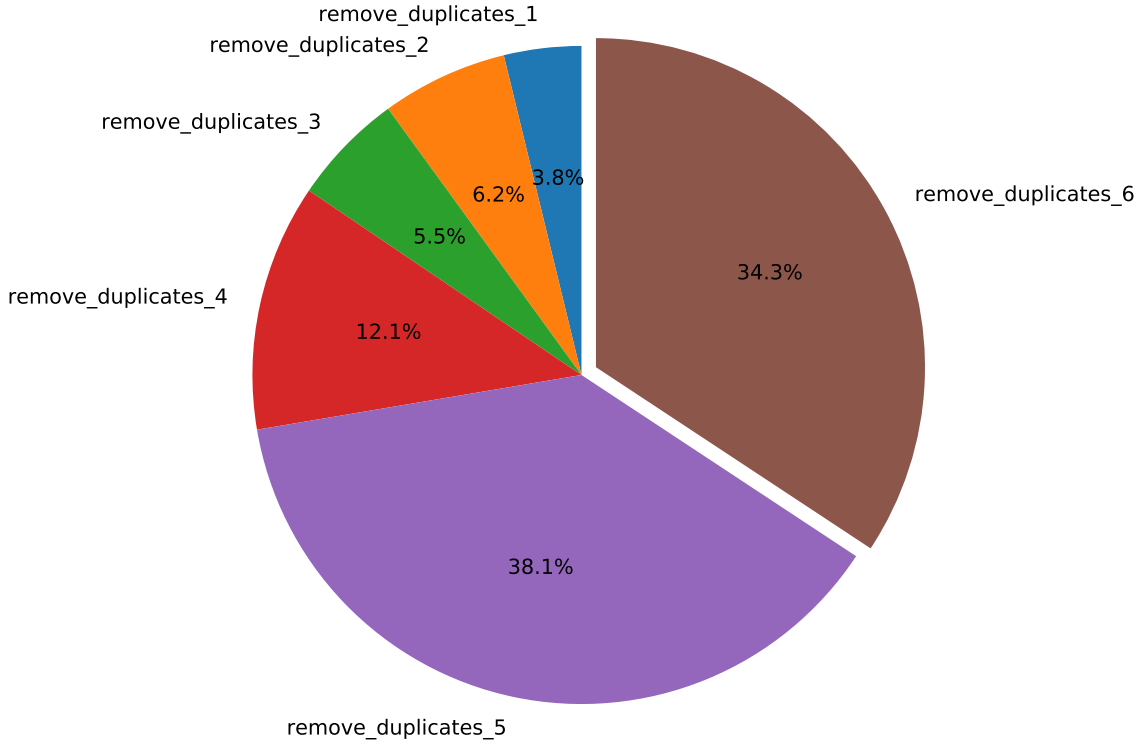
TIMES FOR THE FIRST 36000 NUMBERS



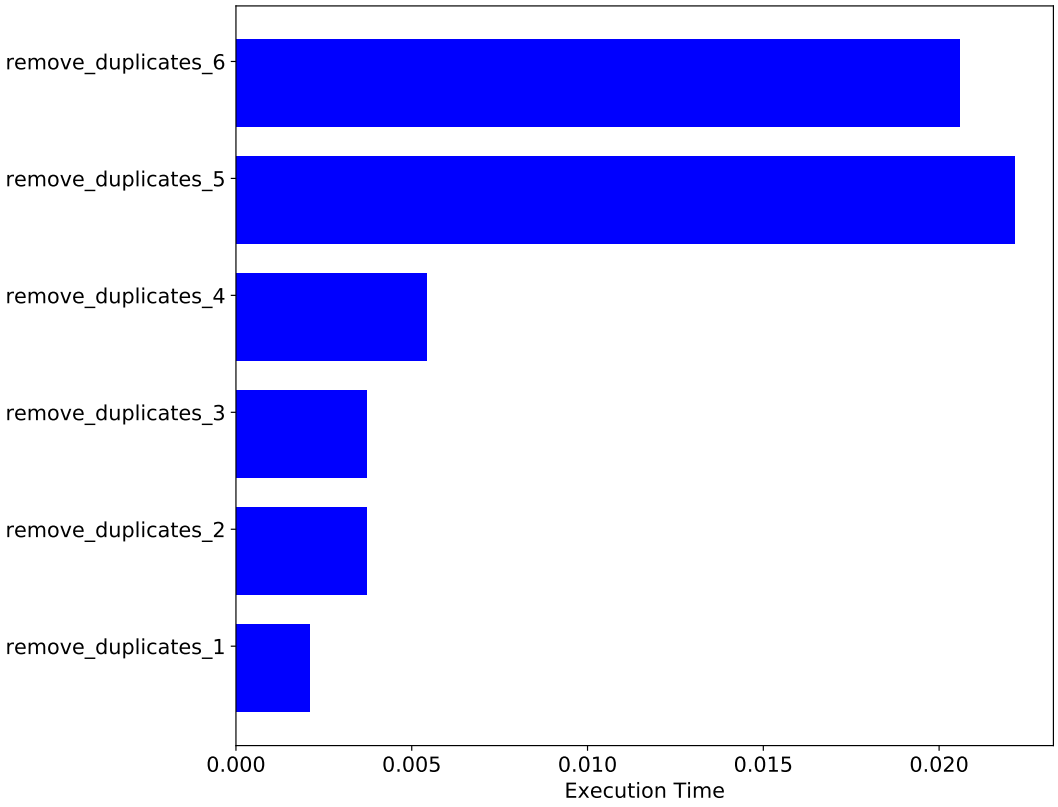


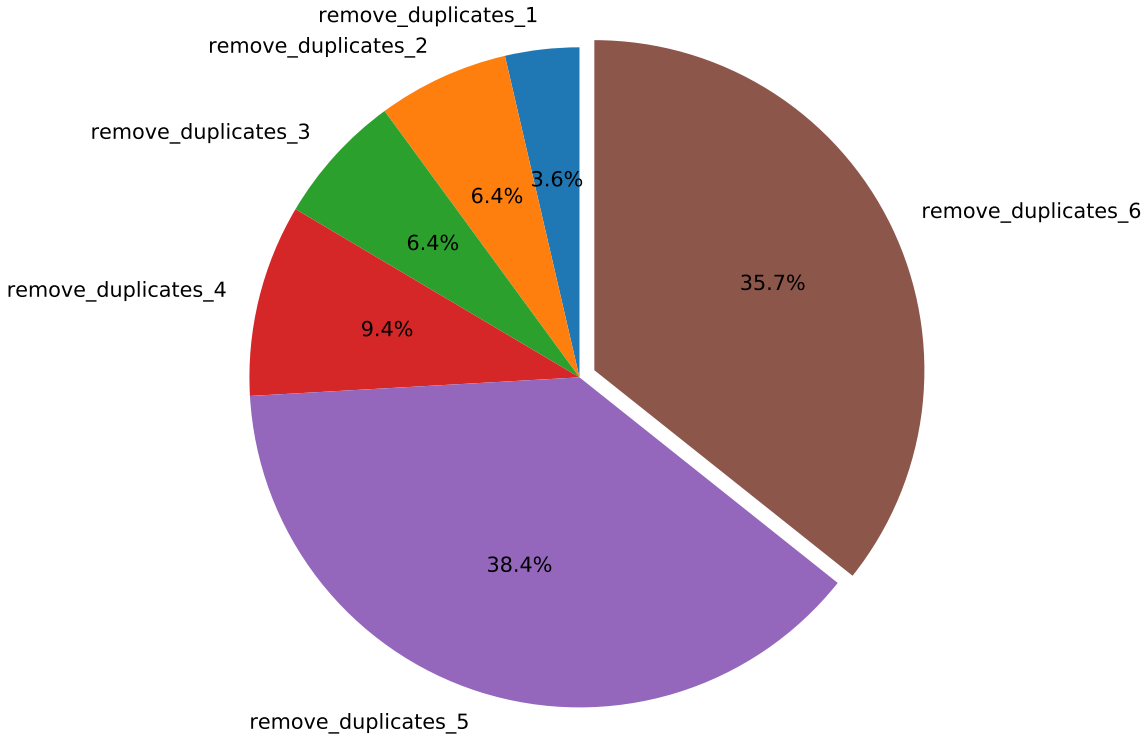
TIMES FOR THE FIRST 38000 NUMBERS



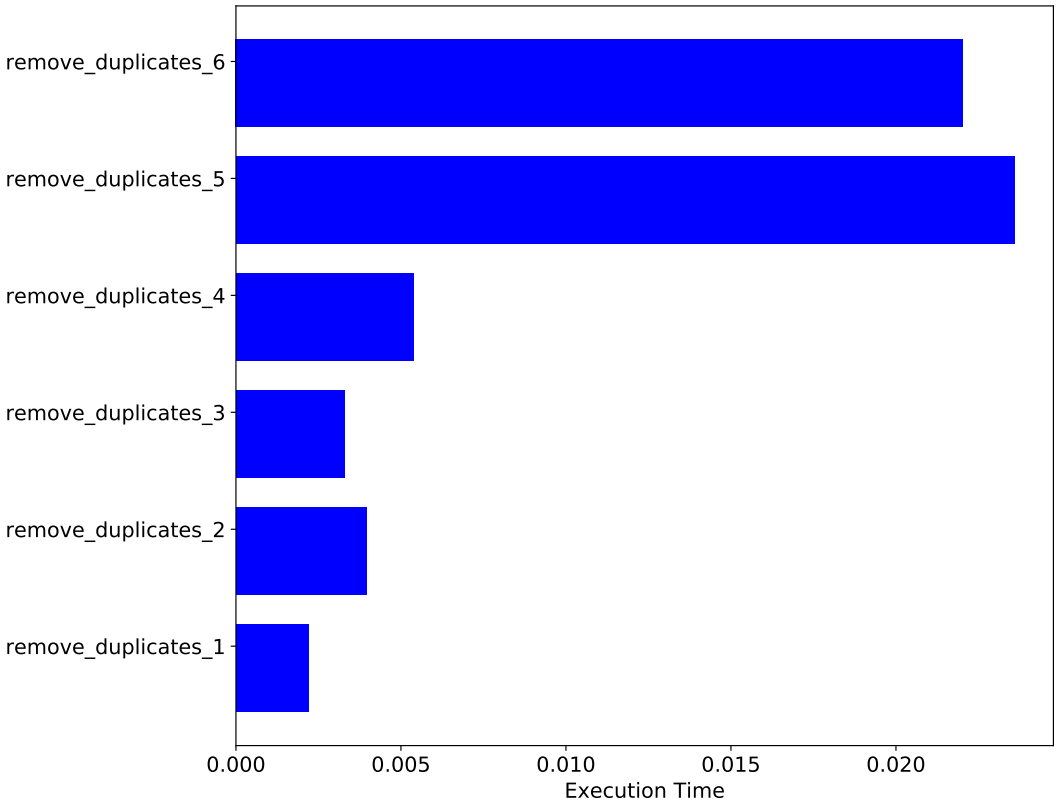


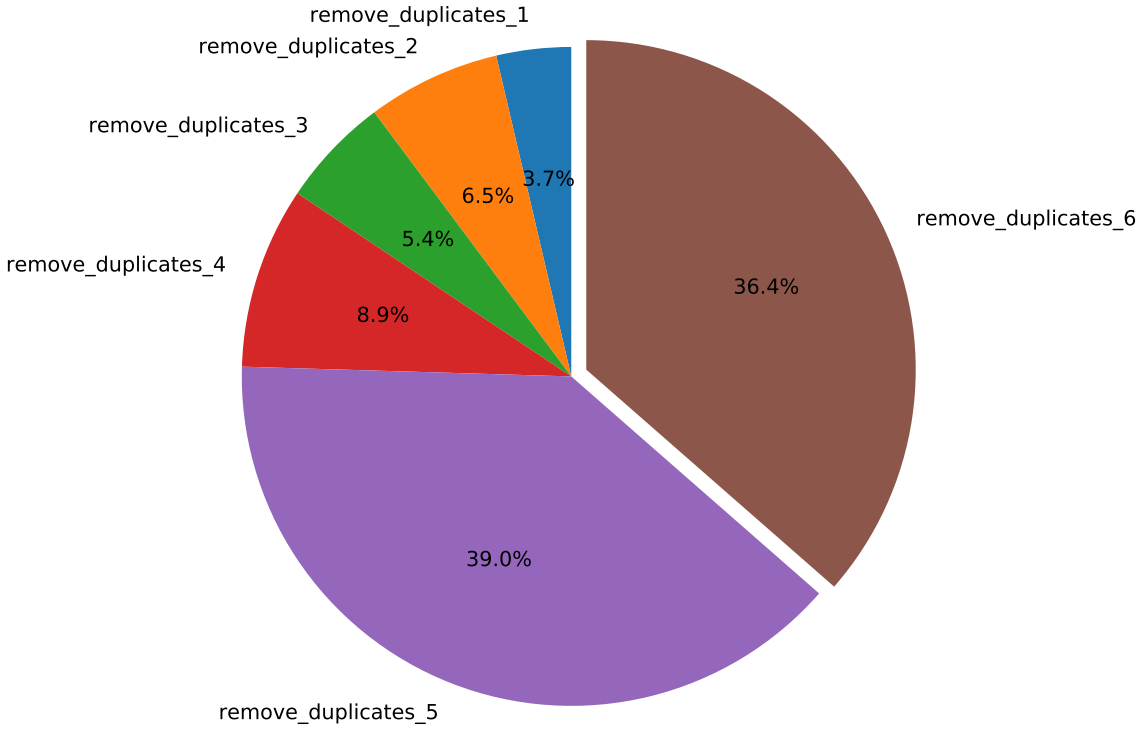
TIMES FOR THE FIRST 40000 NUMBERS



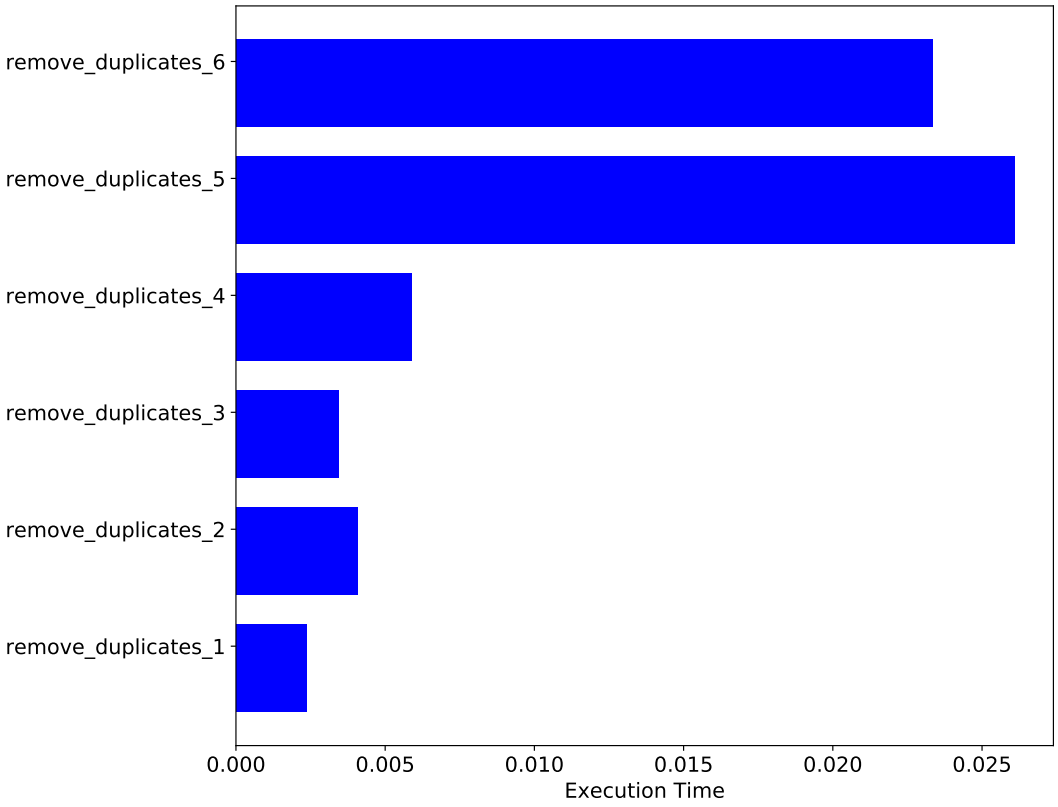


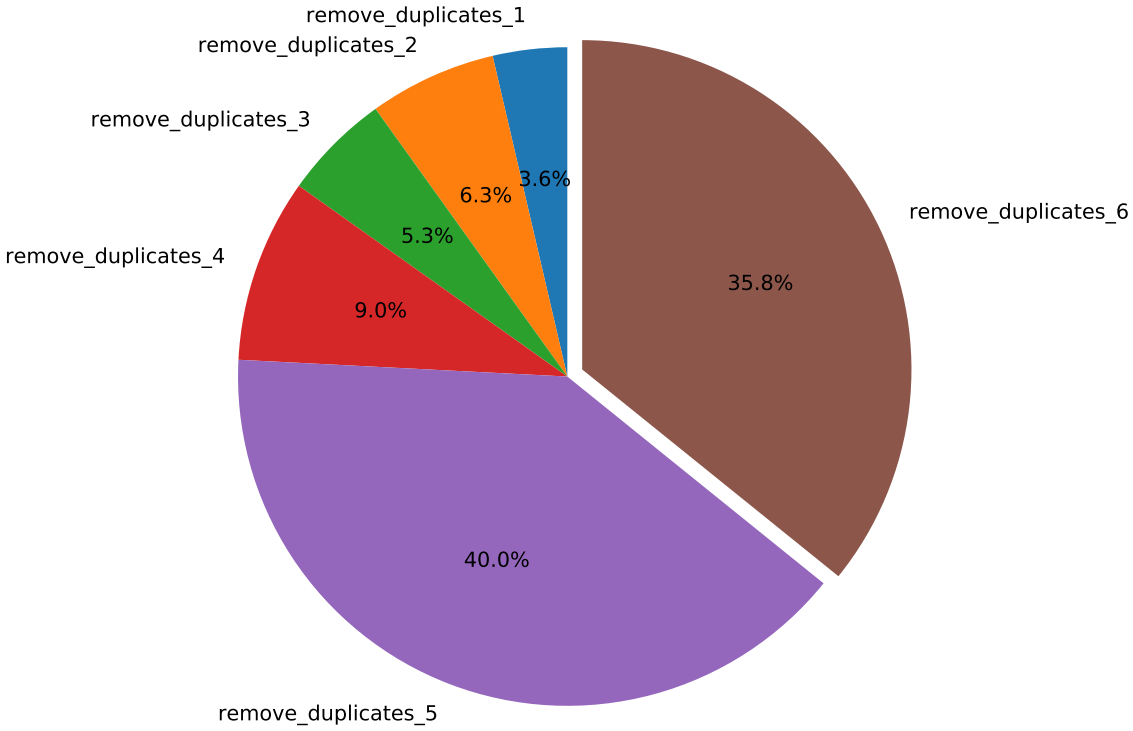
TIMES FOR THE FIRST 42000 NUMBERS



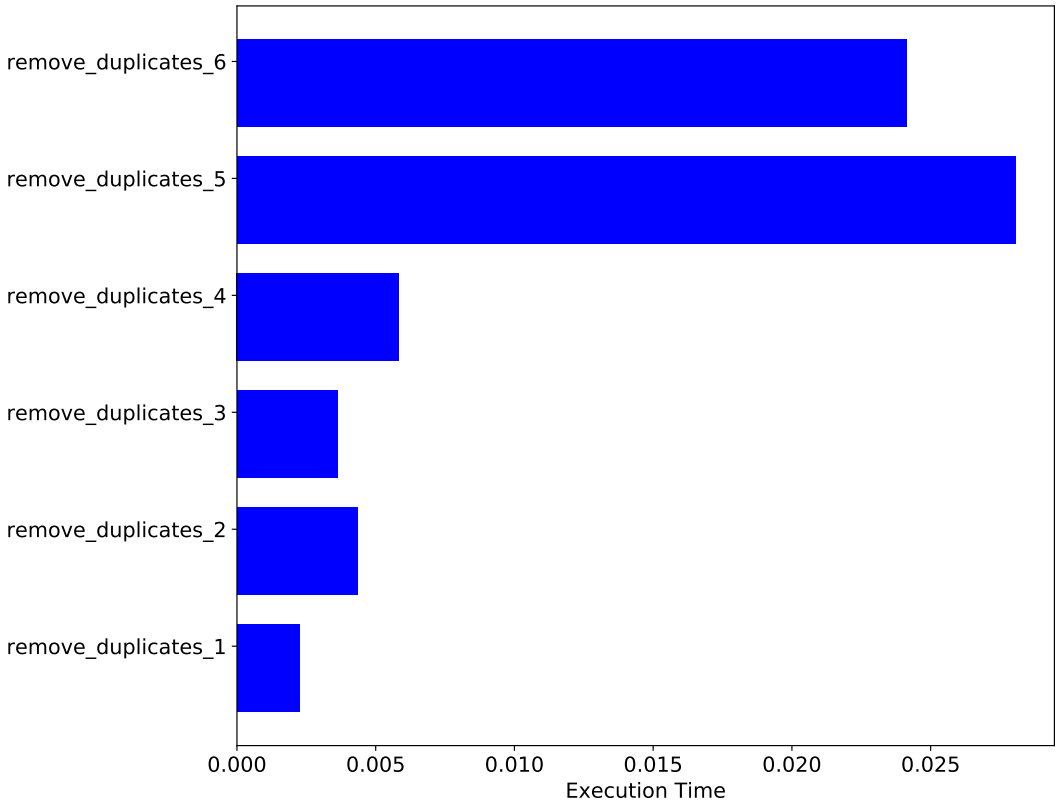


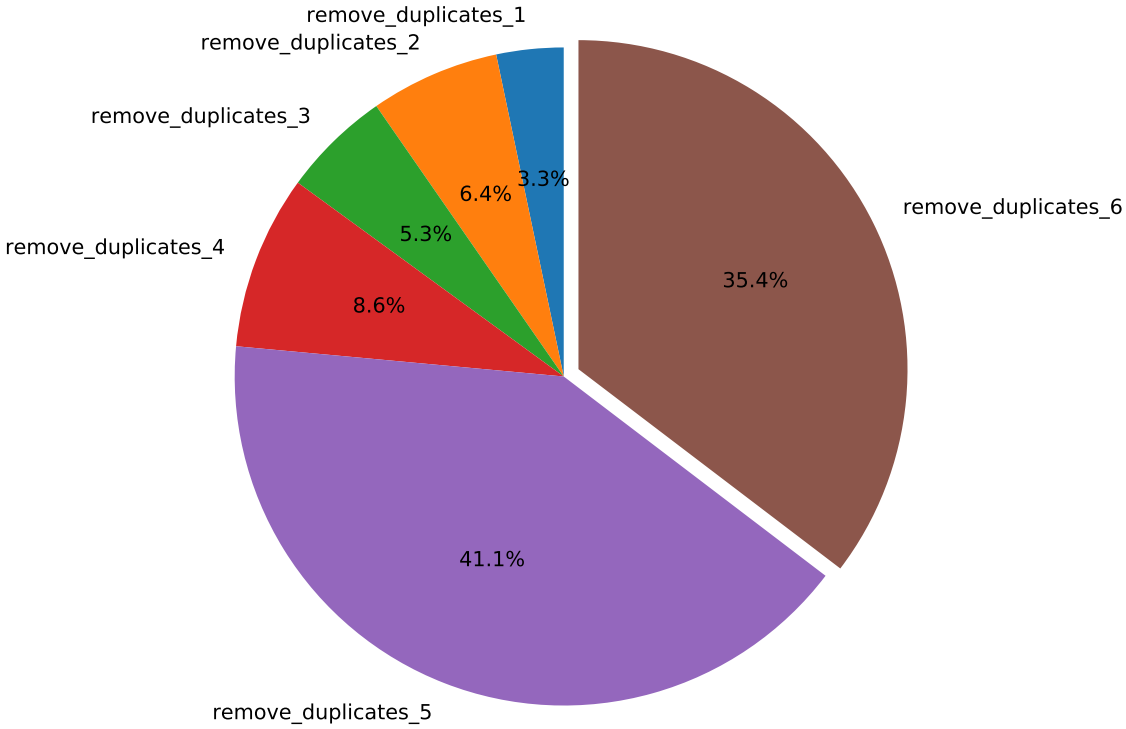
TIMES FOR THE FIRST 44000 NUMBERS



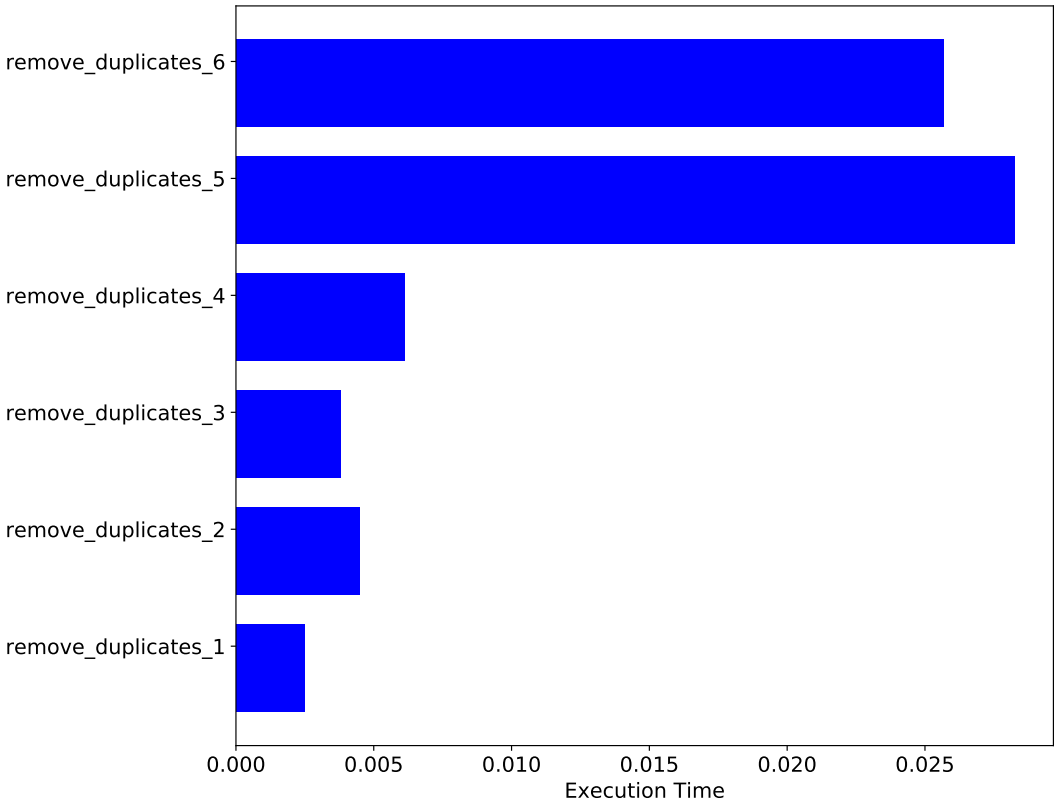


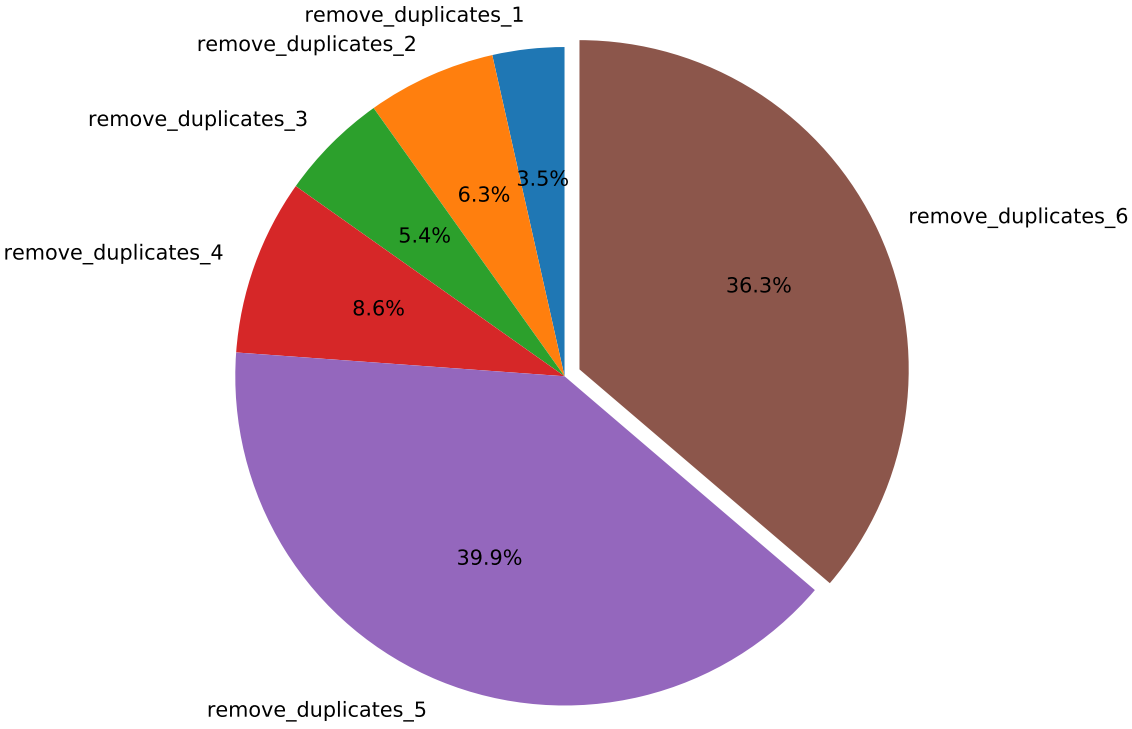
TIMES FOR THE FIRST 46000 NUMBERS



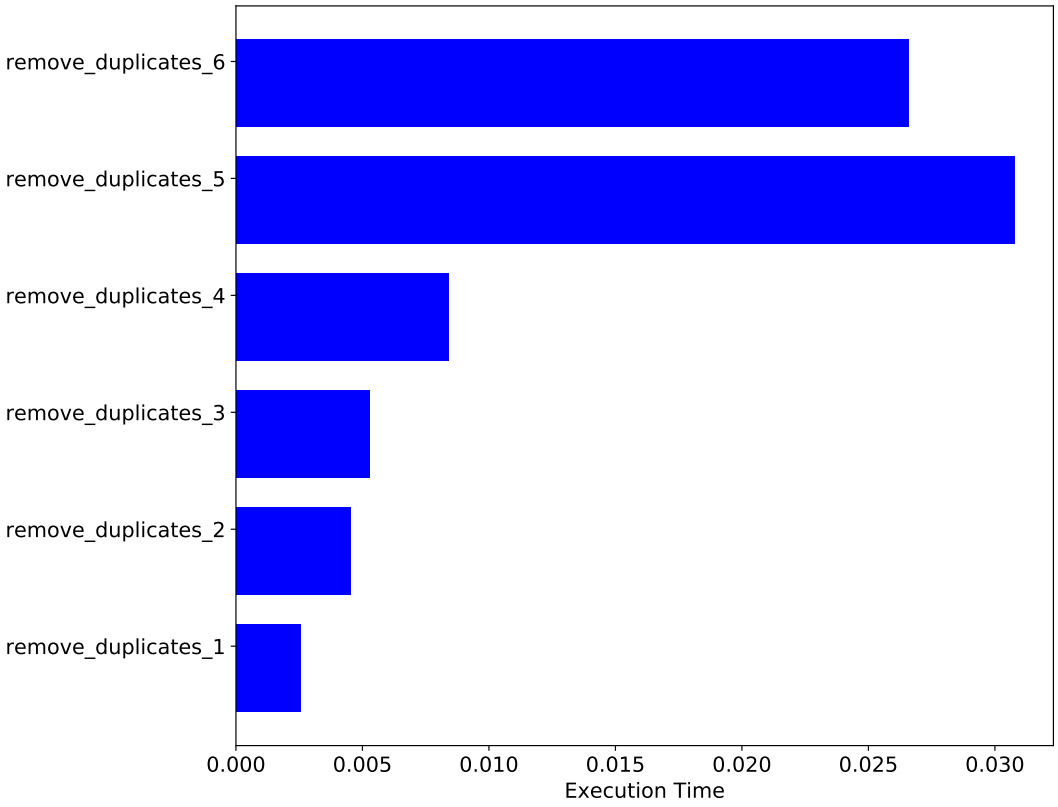


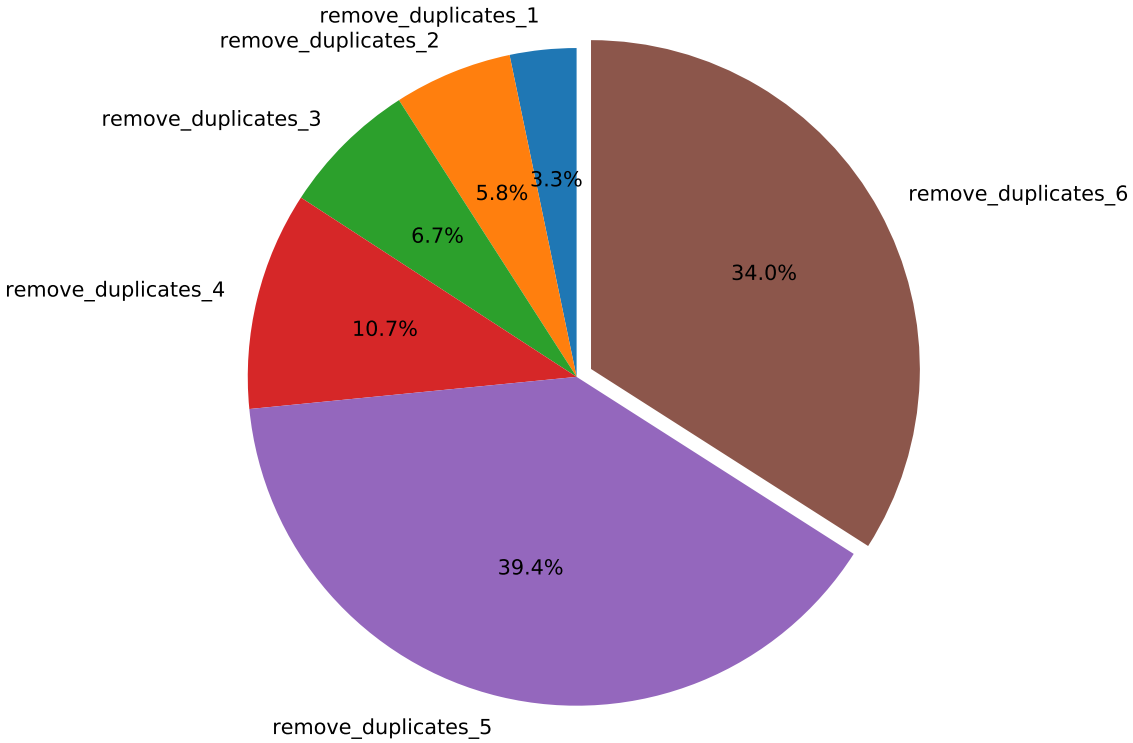
TIMES FOR THE FIRST 48000 NUMBERS



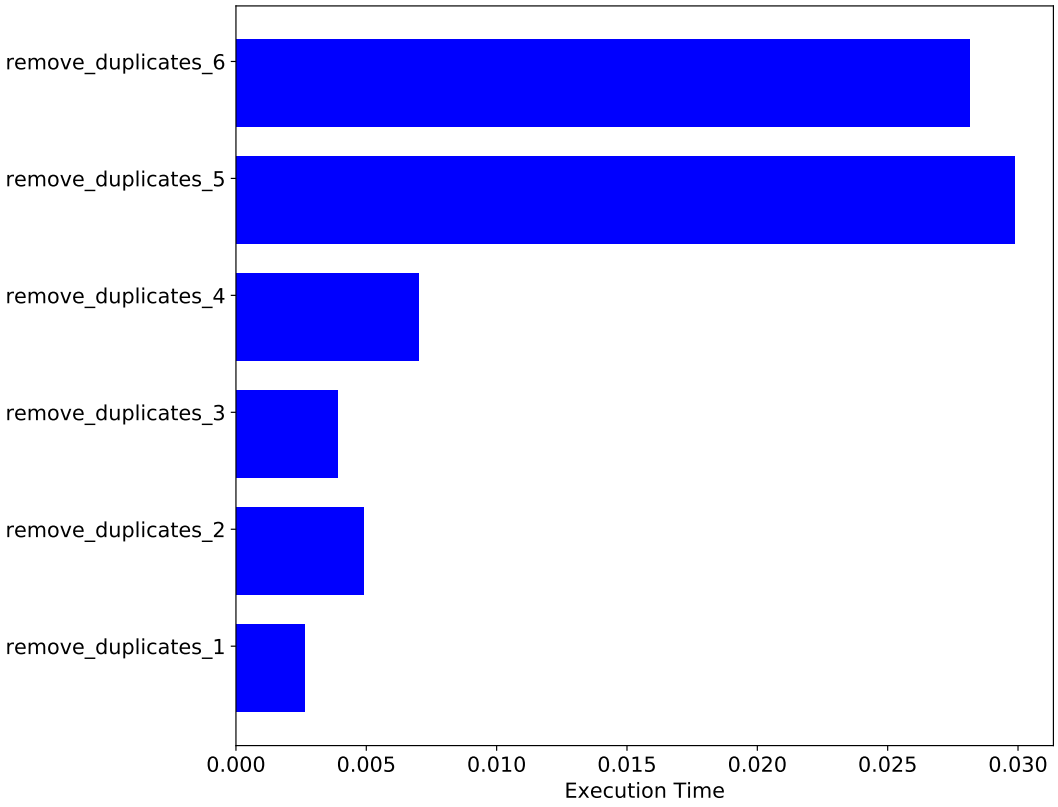


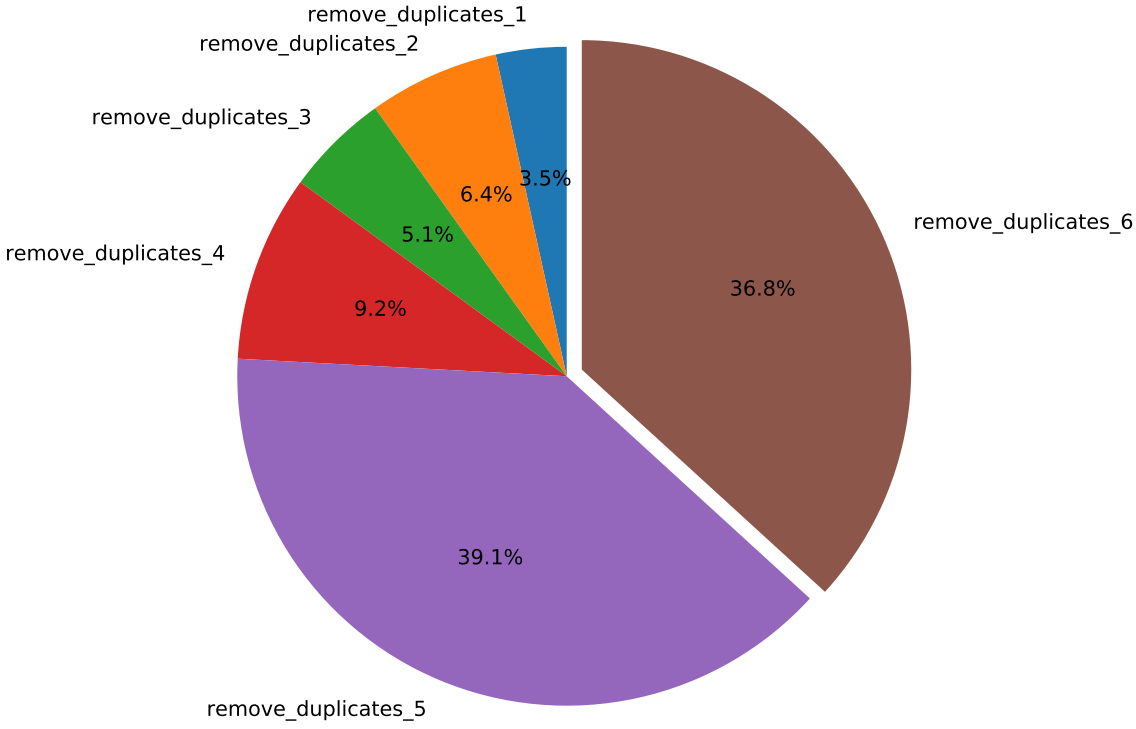
TIMES FOR THE FIRST 50000 NUMBERS



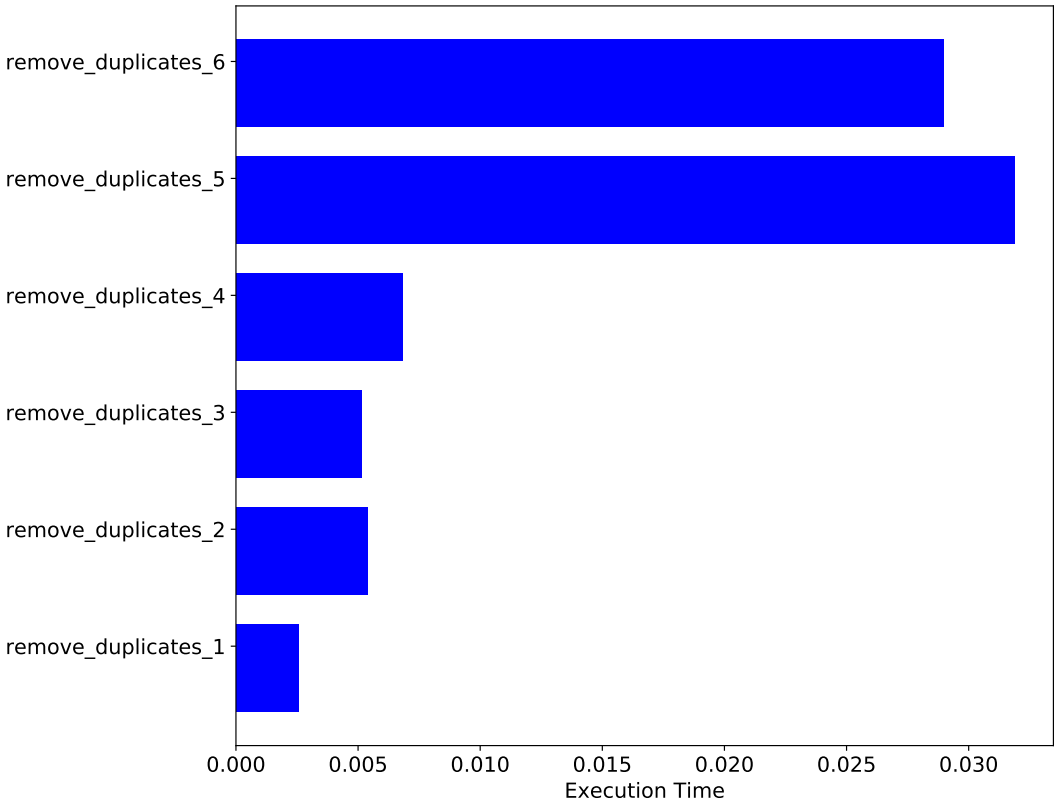


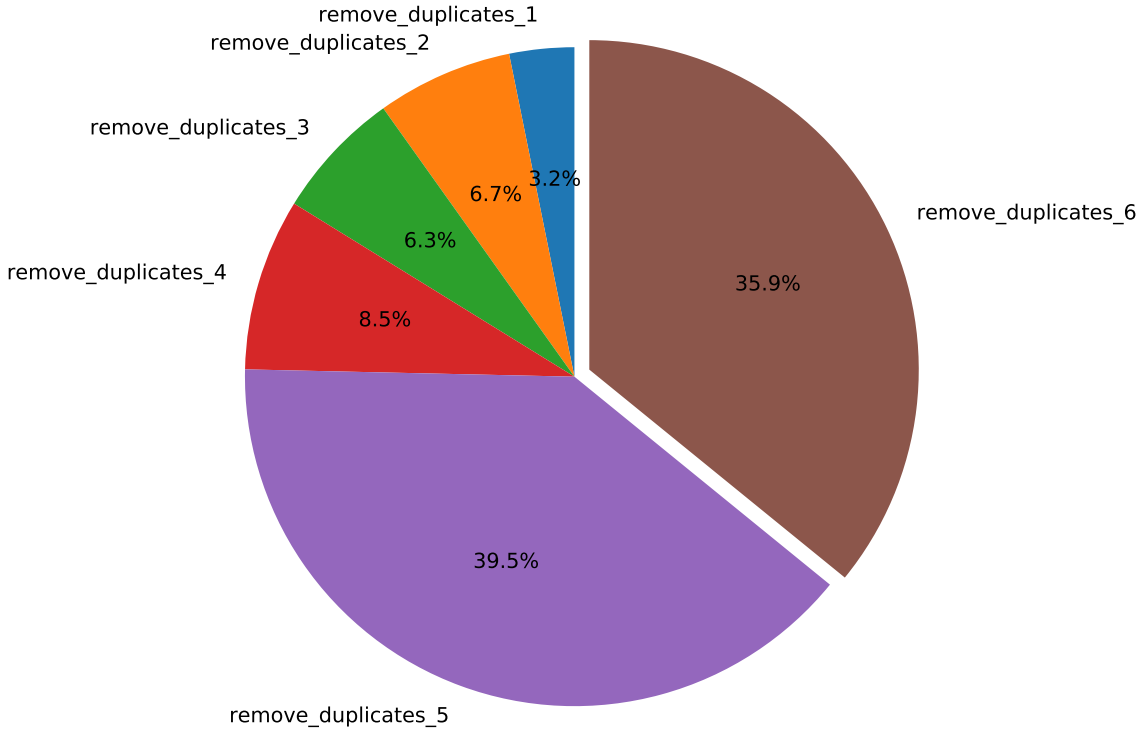
TIMES FOR THE FIRST 52000 NUMBERS



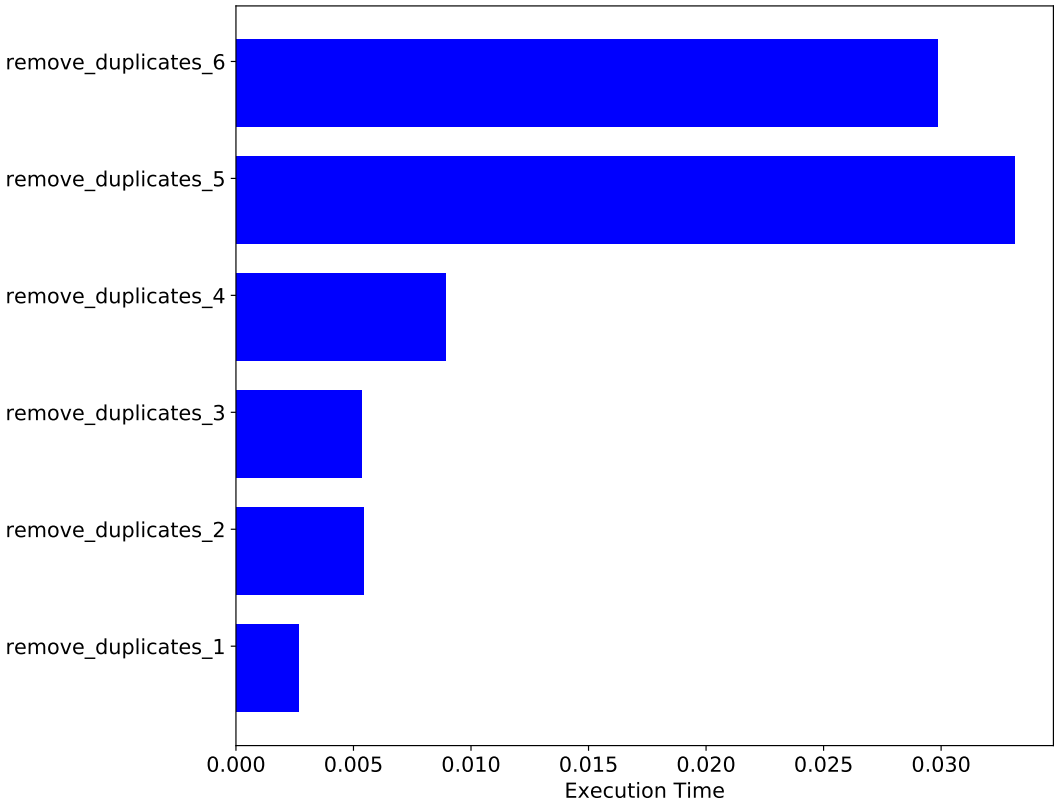


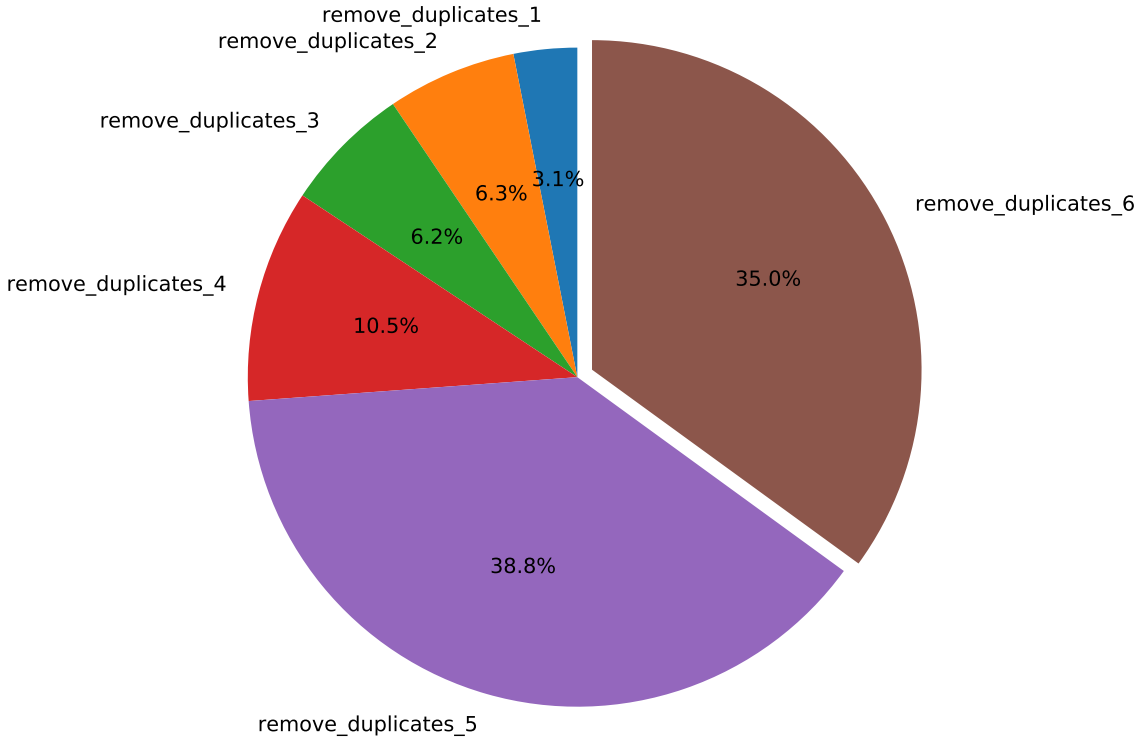
TIMES FOR THE FIRST 54000 NUMBERS



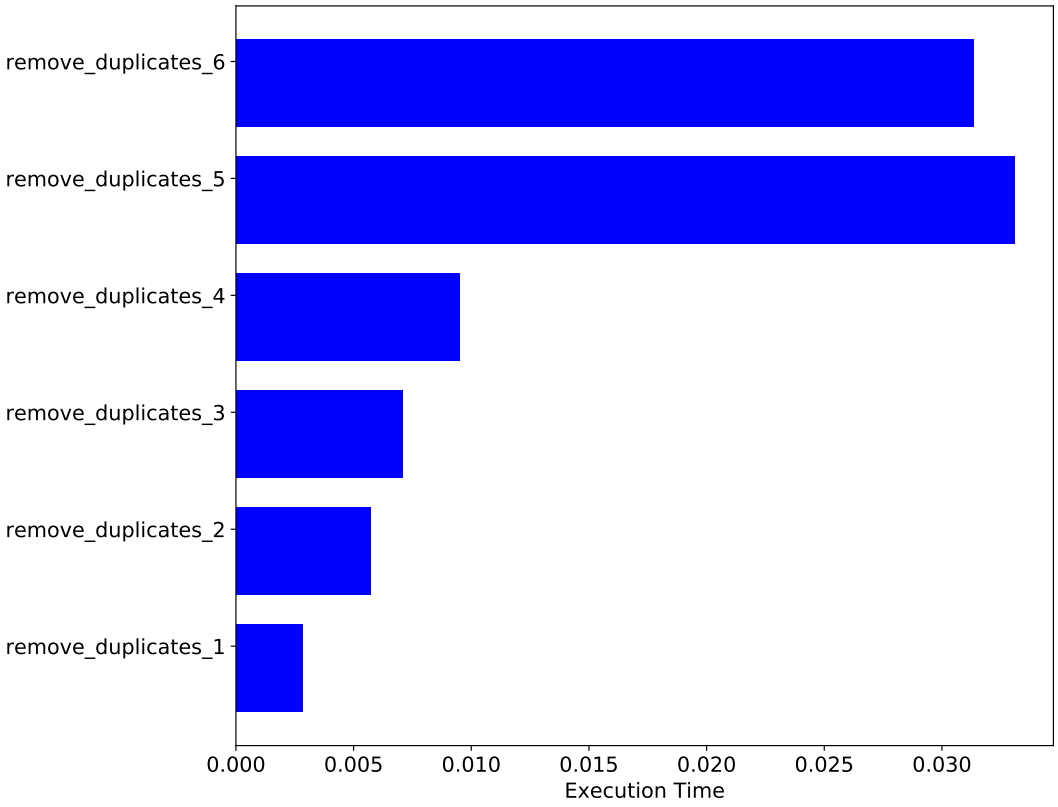


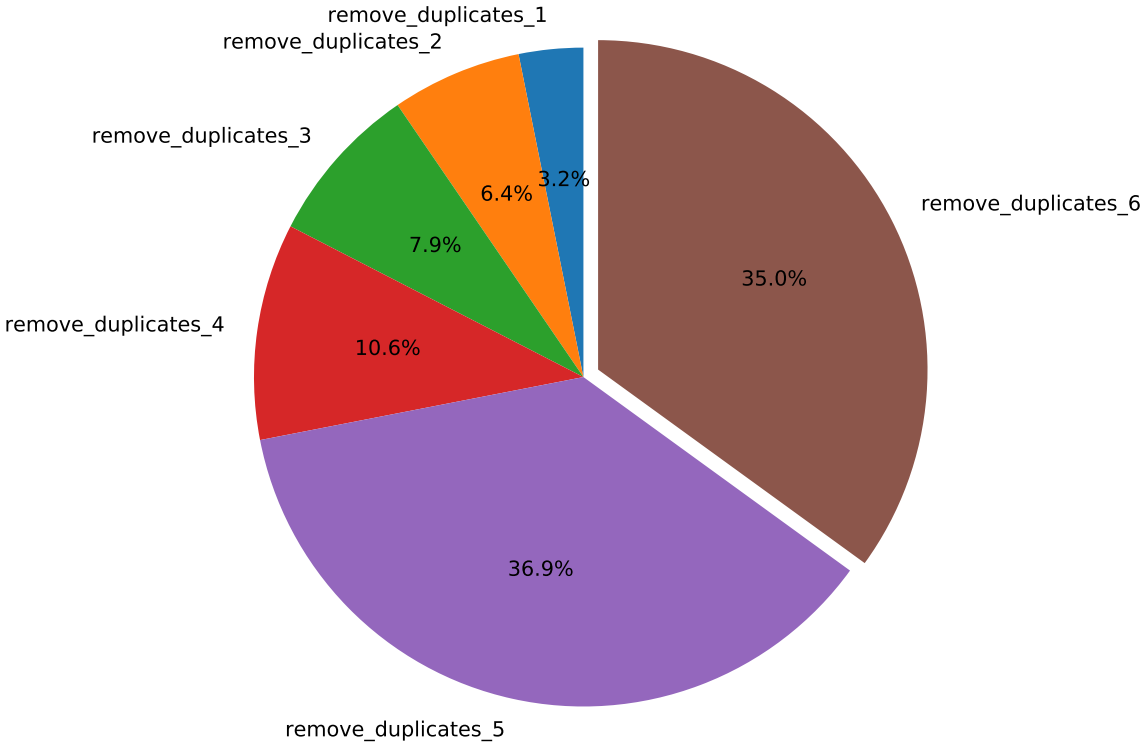
TIMES FOR THE FIRST 56000 NUMBERS



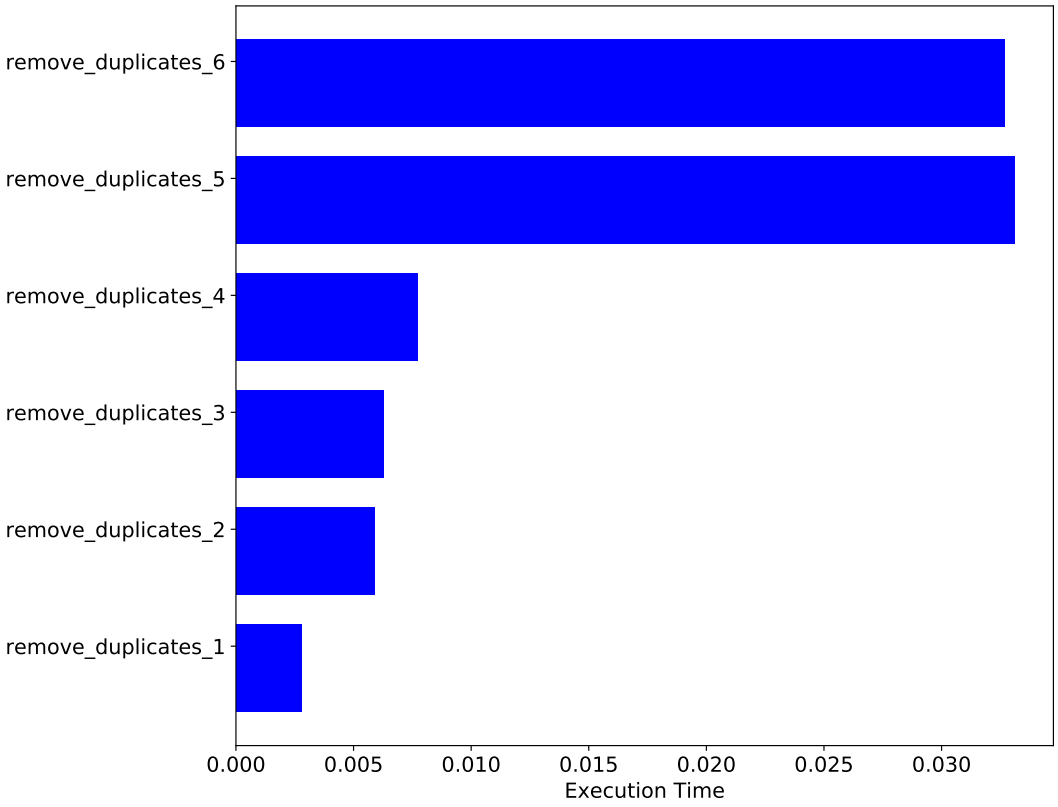


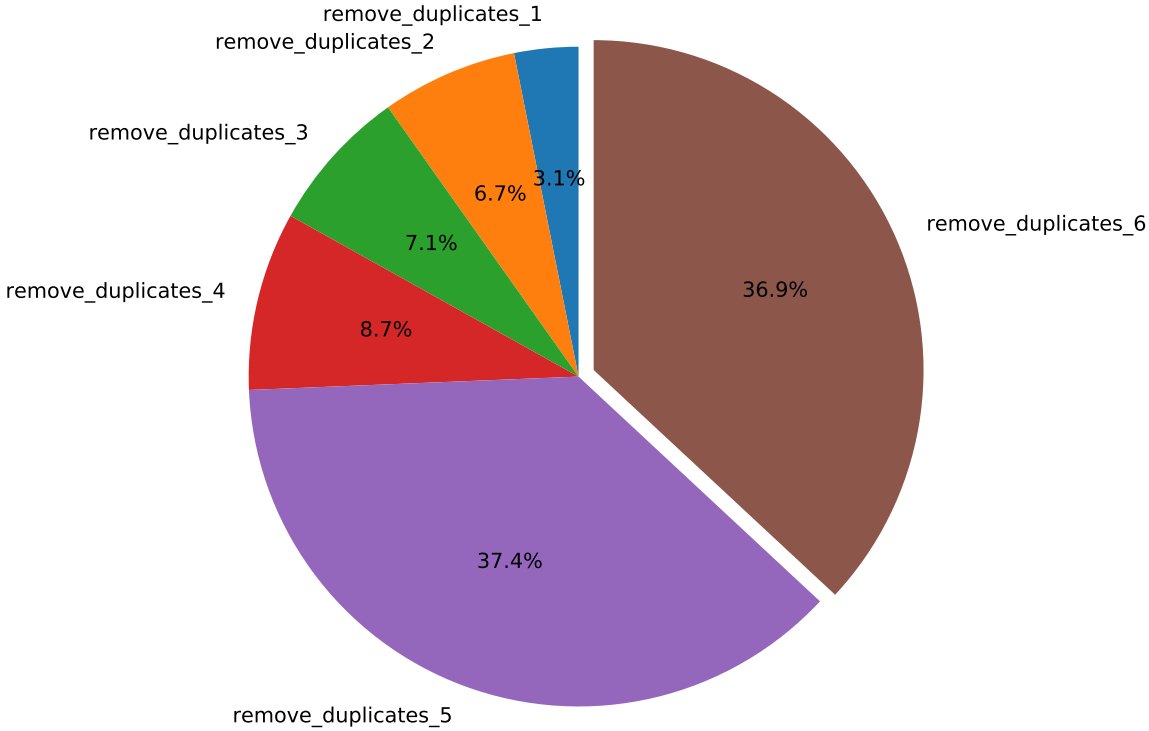
TIMES FOR THE FIRST 58000 NUMBERS



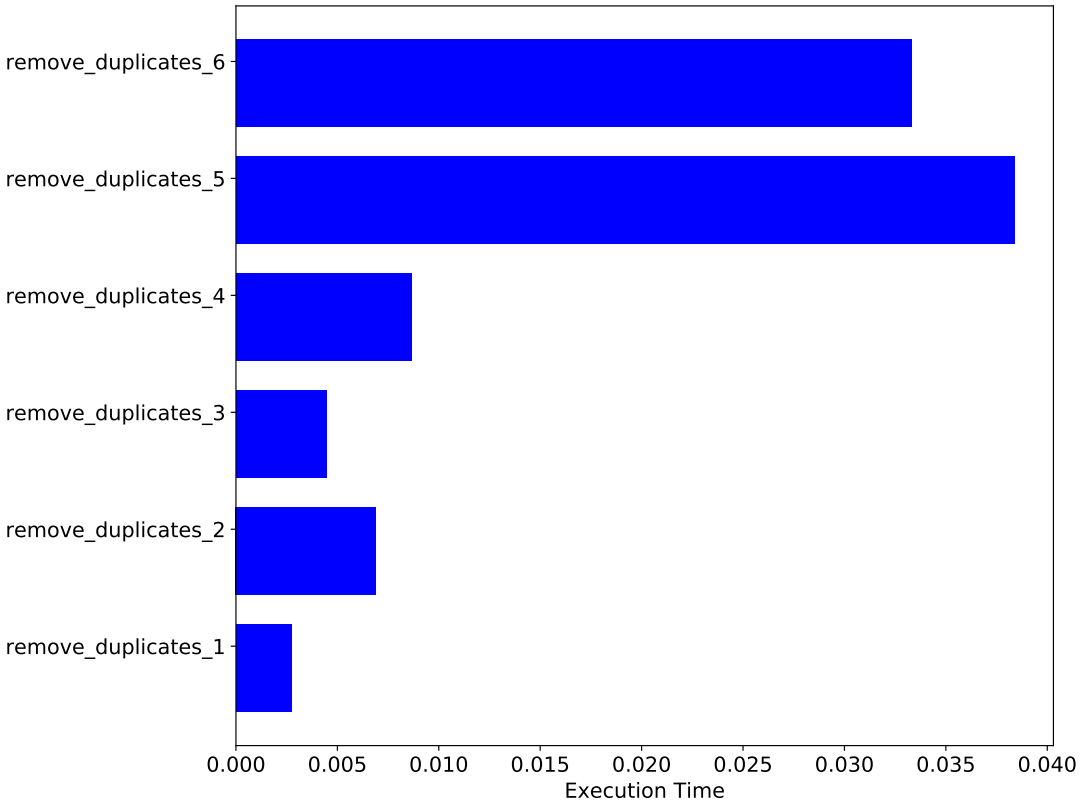


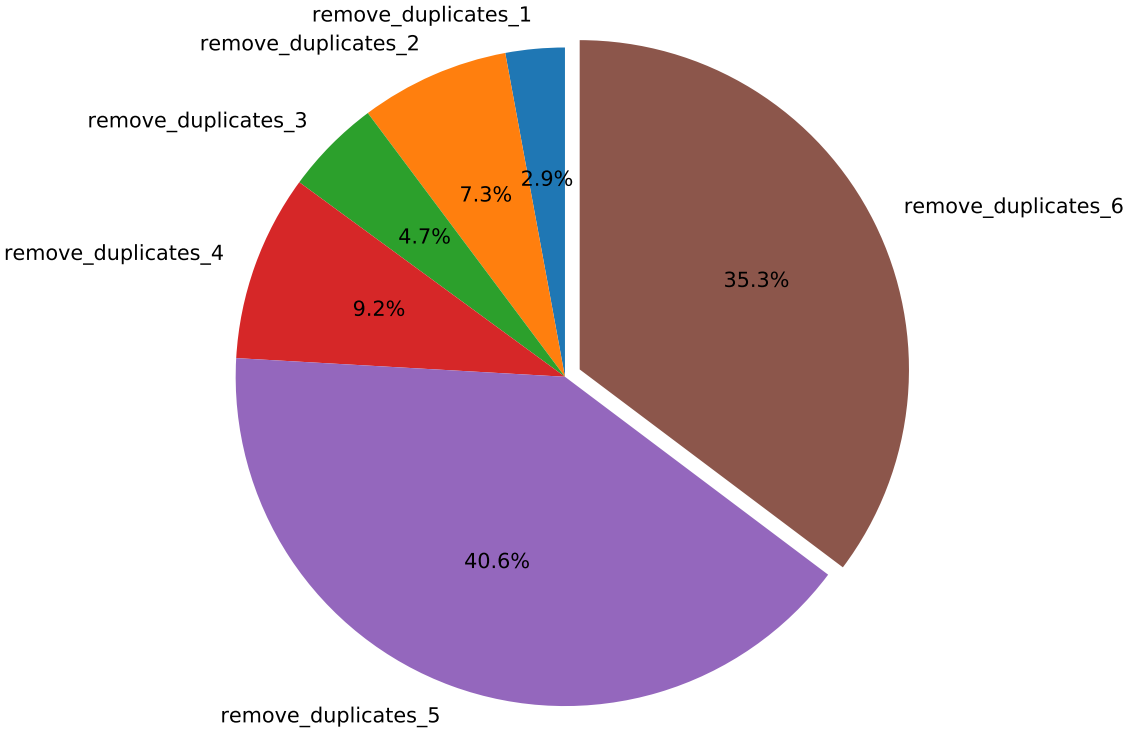
TIMES FOR THE FIRST 60000 NUMBERS



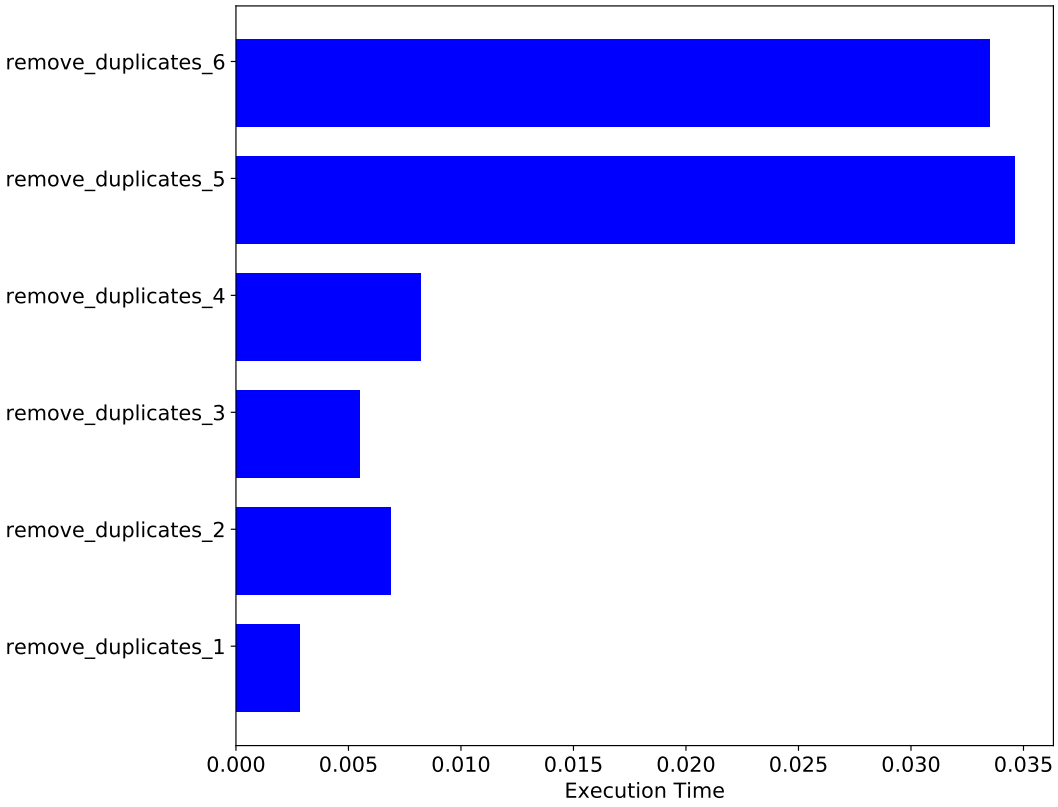


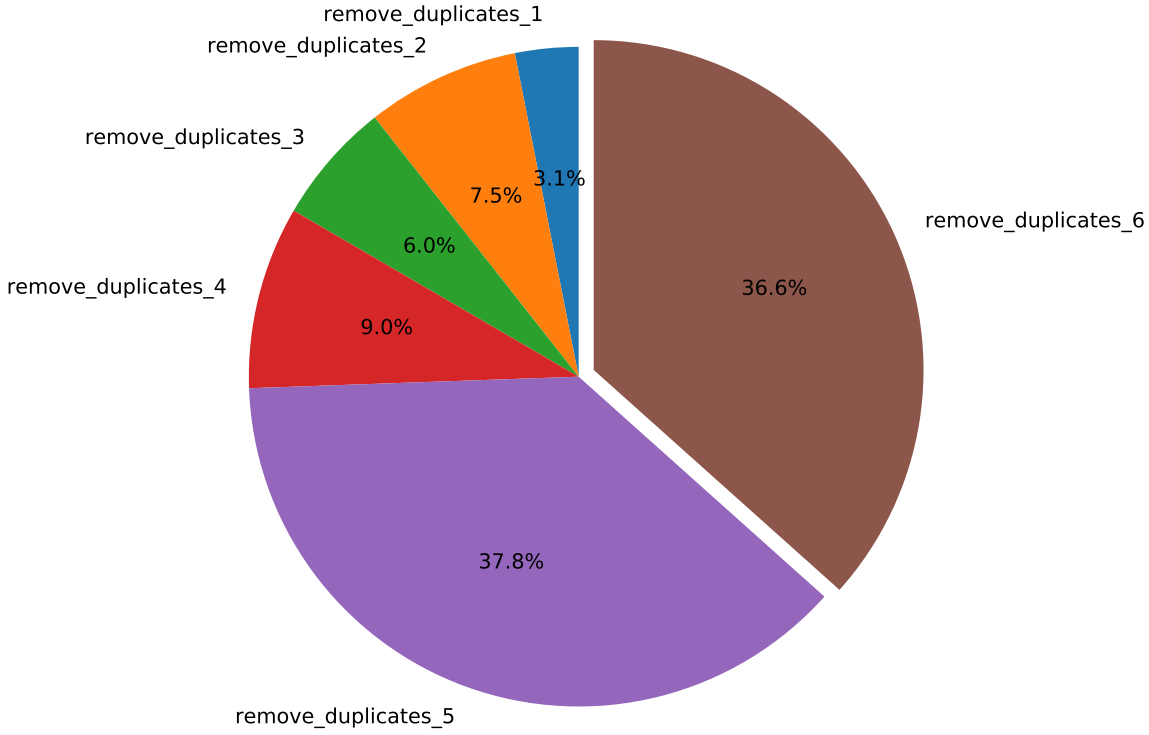
TIMES FOR THE FIRST 62000 NUMBERS



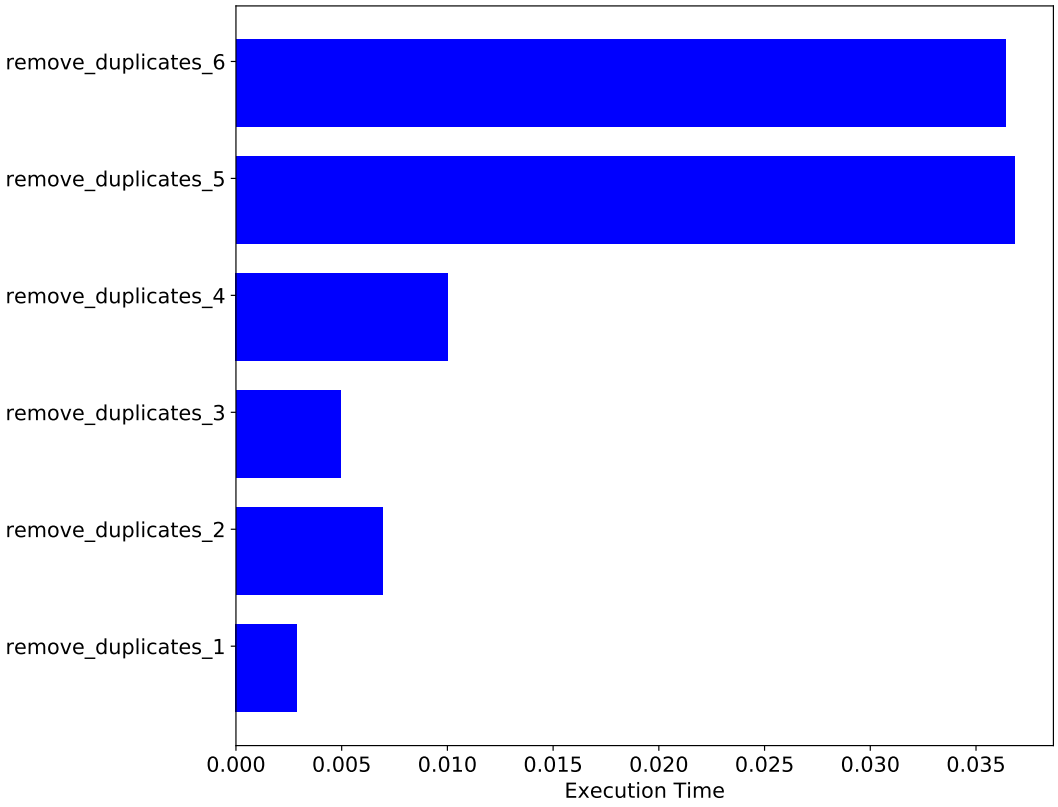


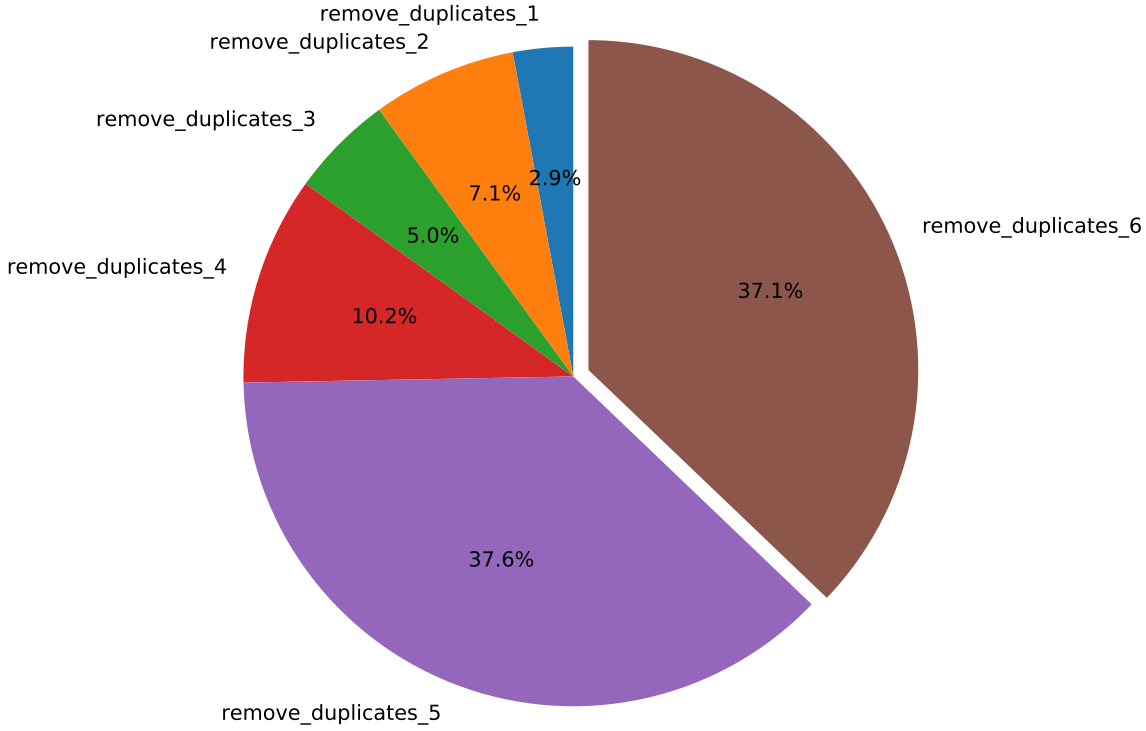
TIMES FOR THE FIRST 64000 NUMBERS



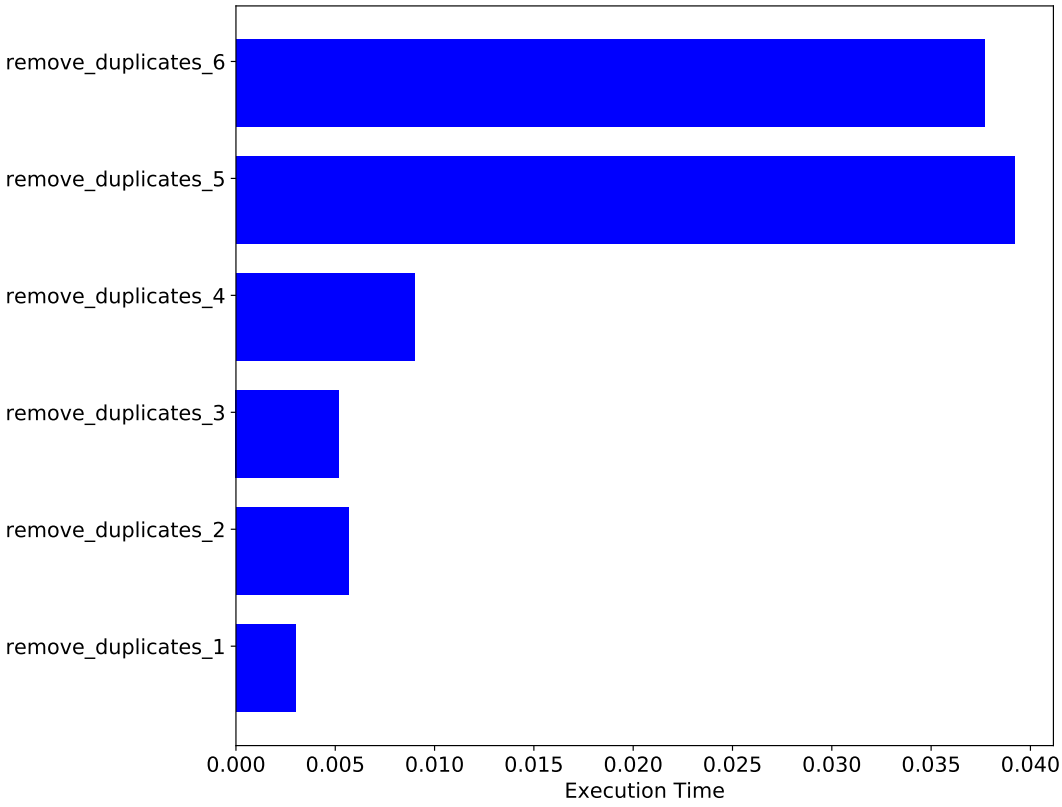


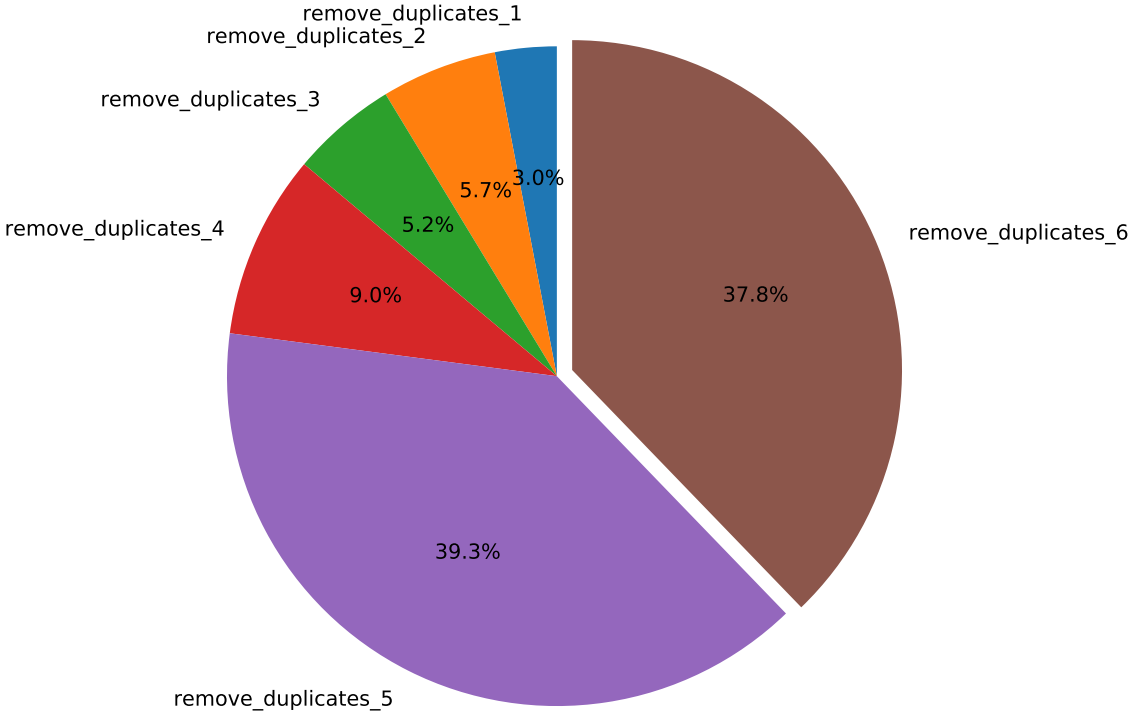
TIMES FOR THE FIRST 66000 NUMBERS



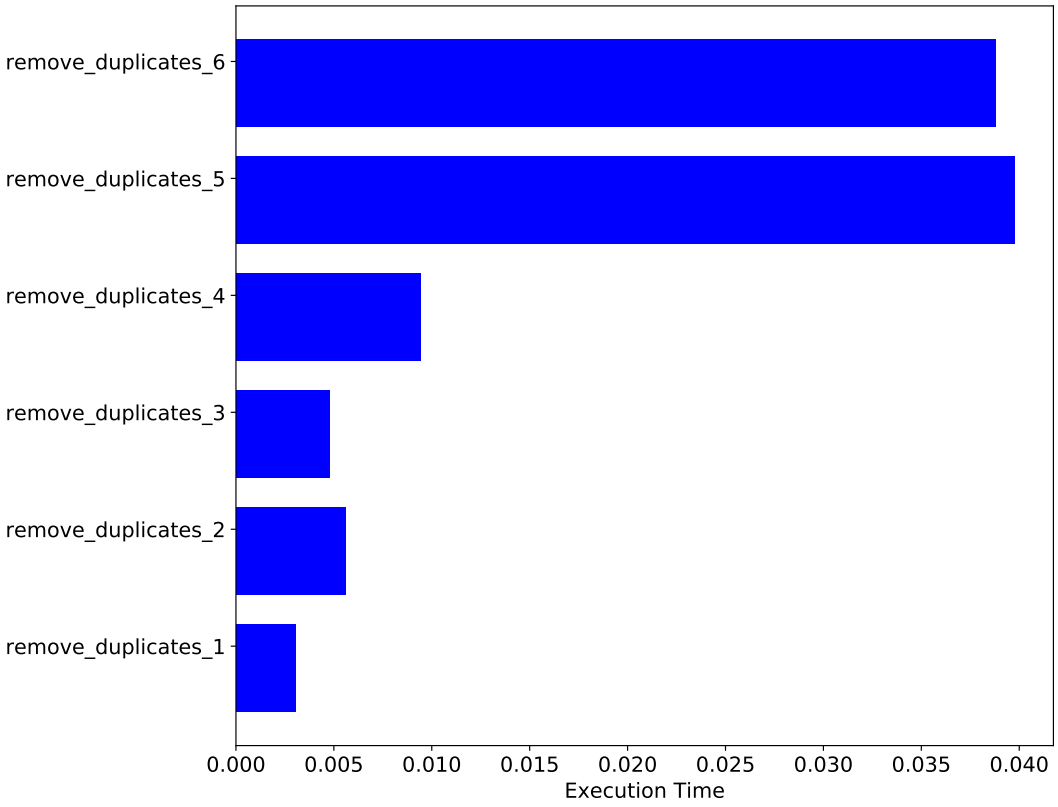


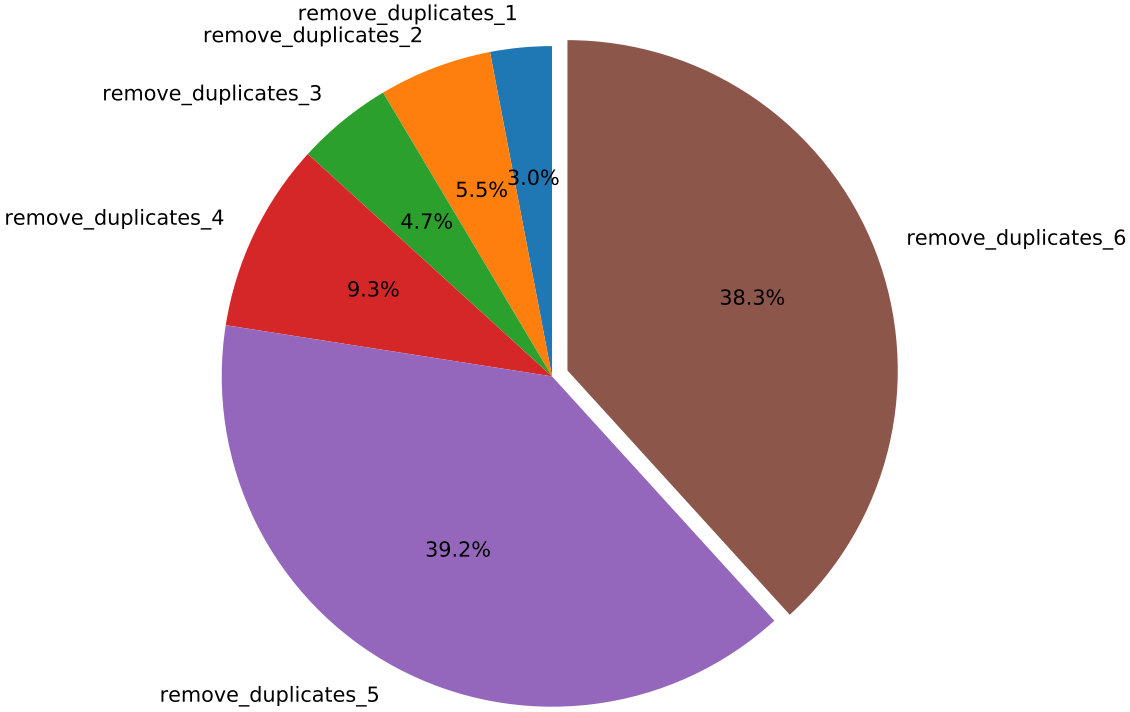
TIMES FOR THE FIRST 68000 NUMBERS



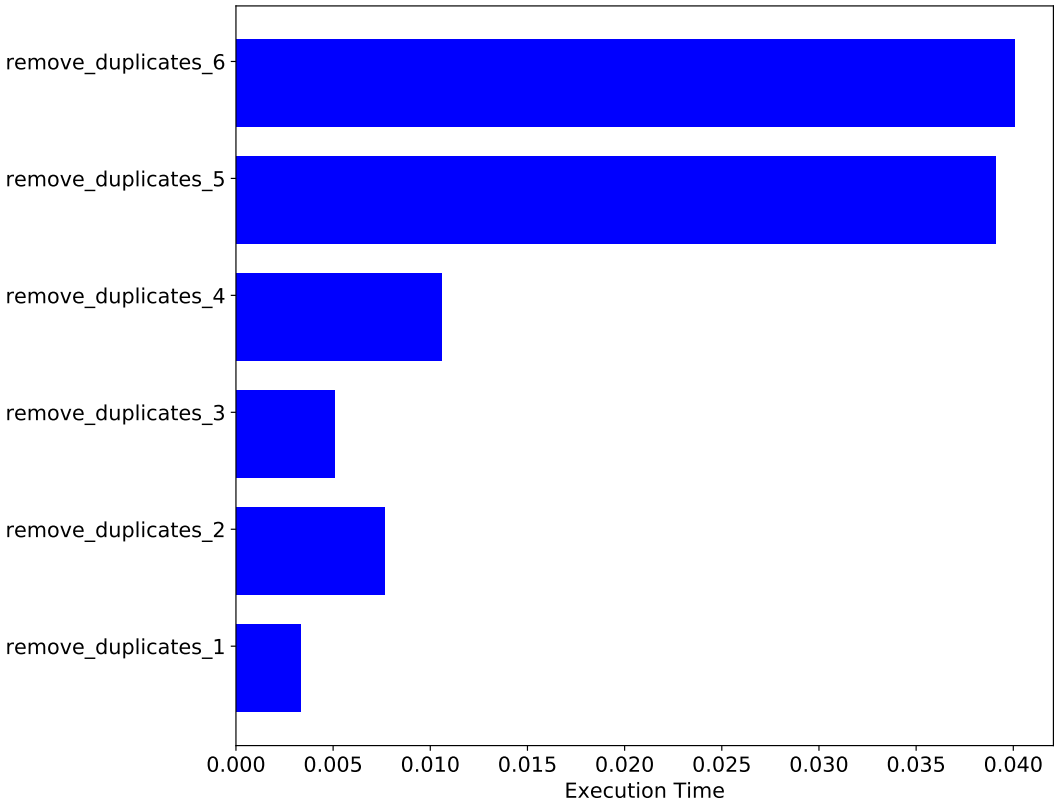


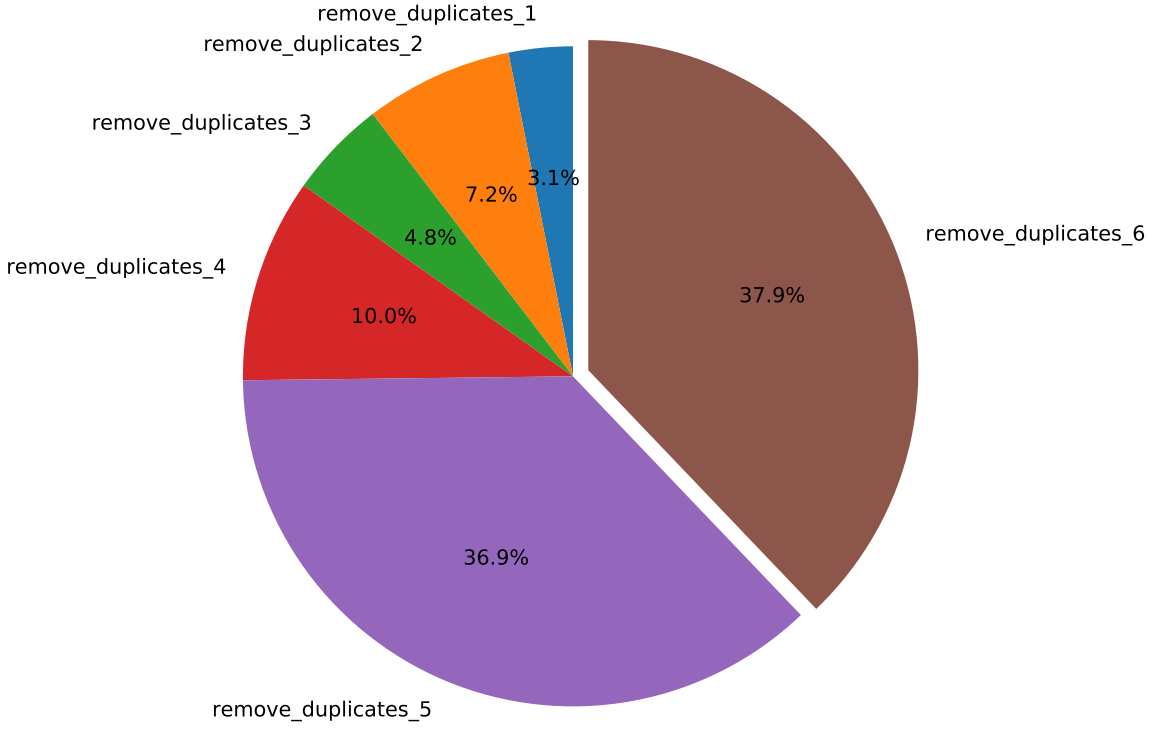
TIMES FOR THE FIRST 70000 NUMBERS



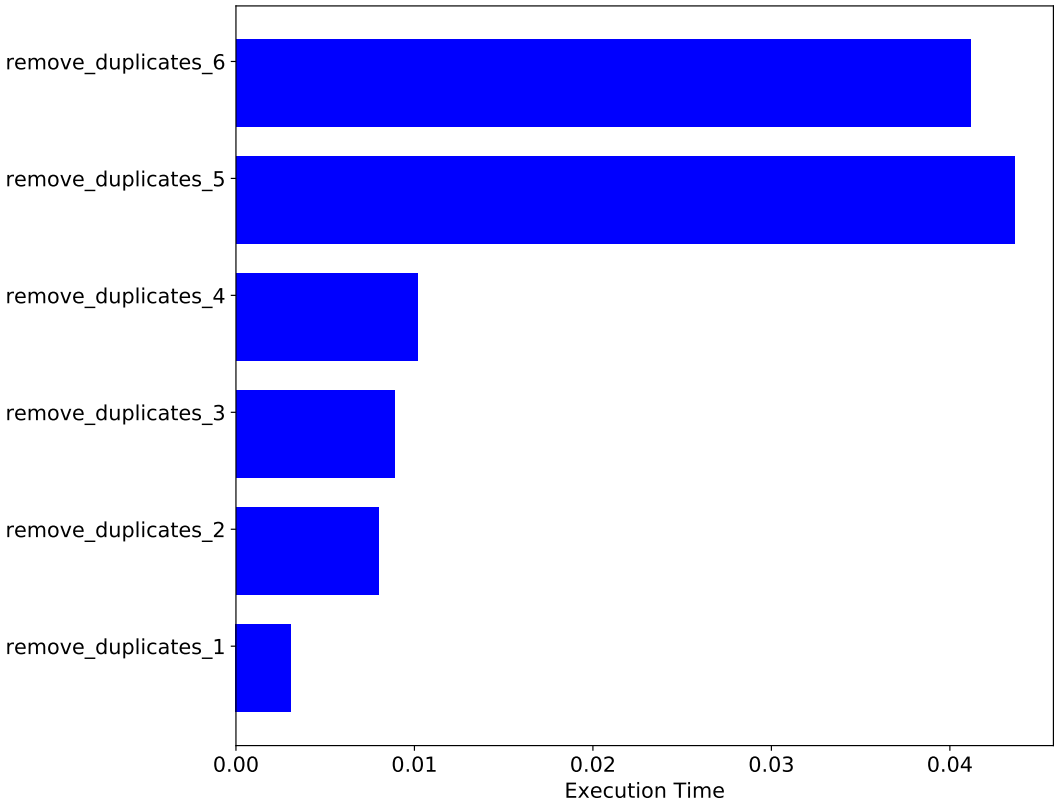


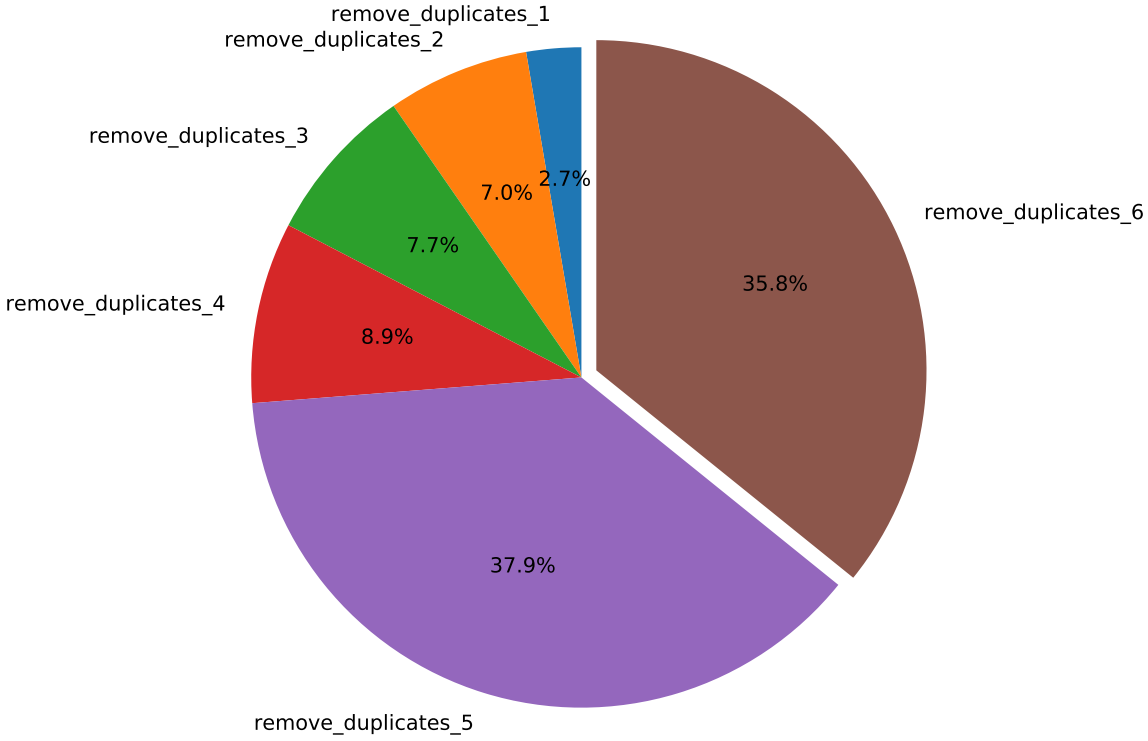
TIMES FOR THE FIRST 72000 NUMBERS



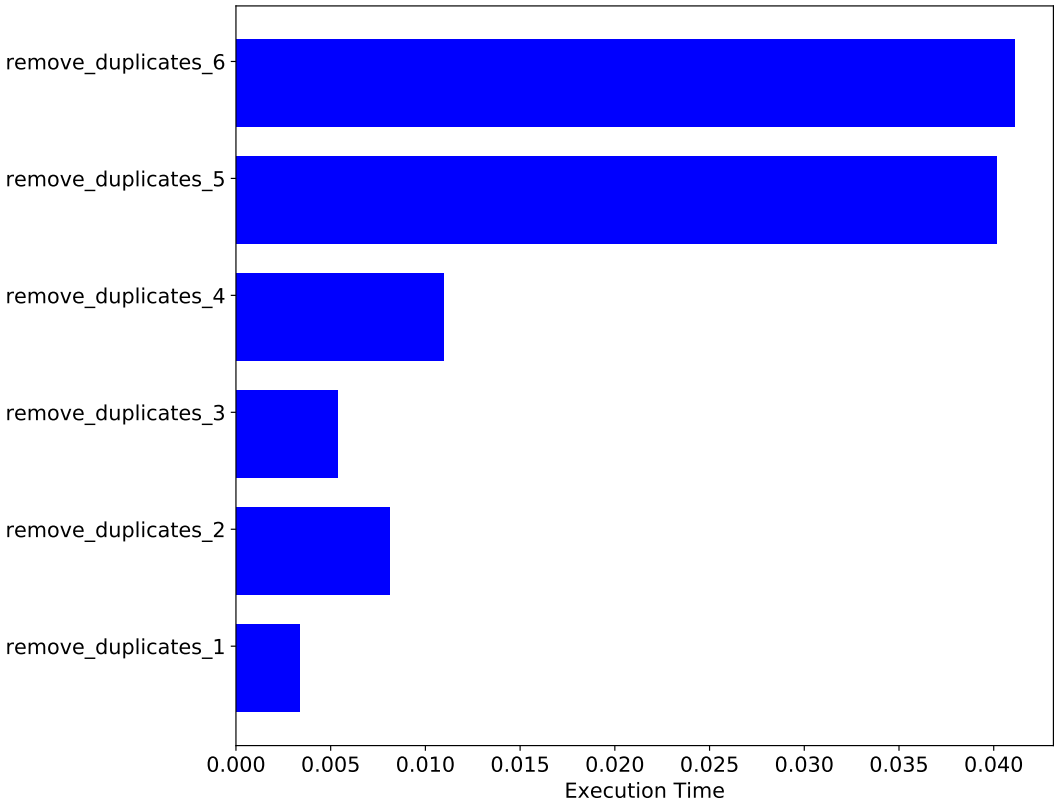


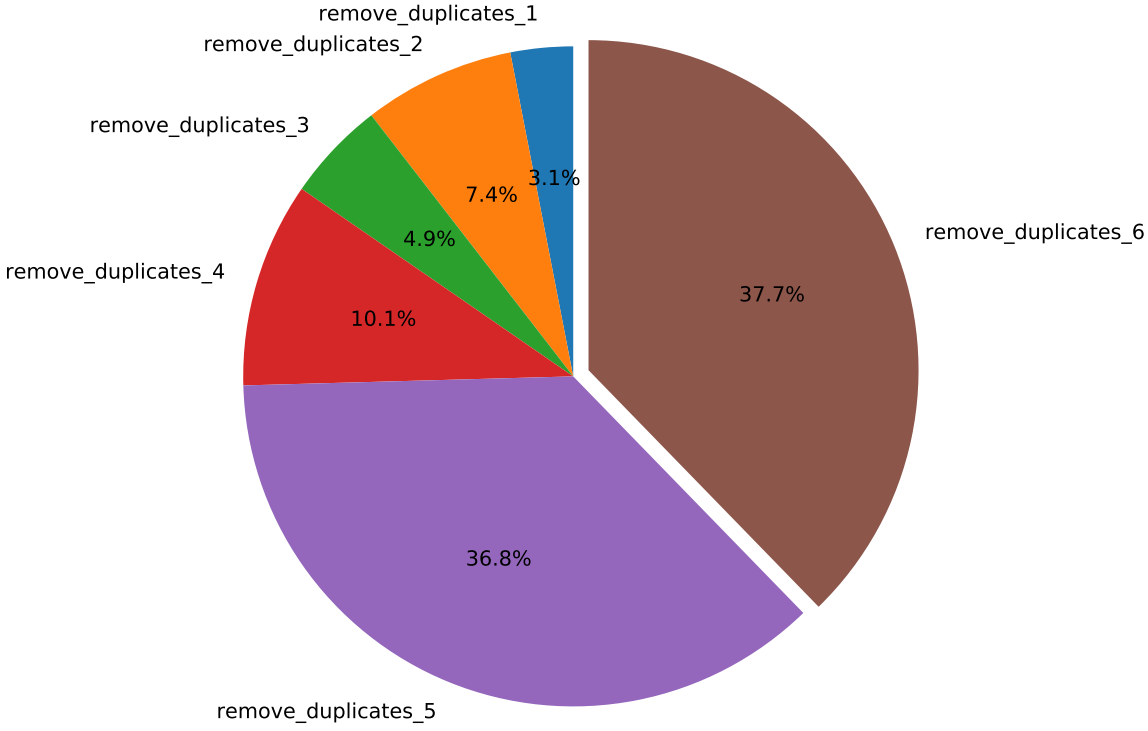
TIMES FOR THE FIRST 74000 NUMBERS



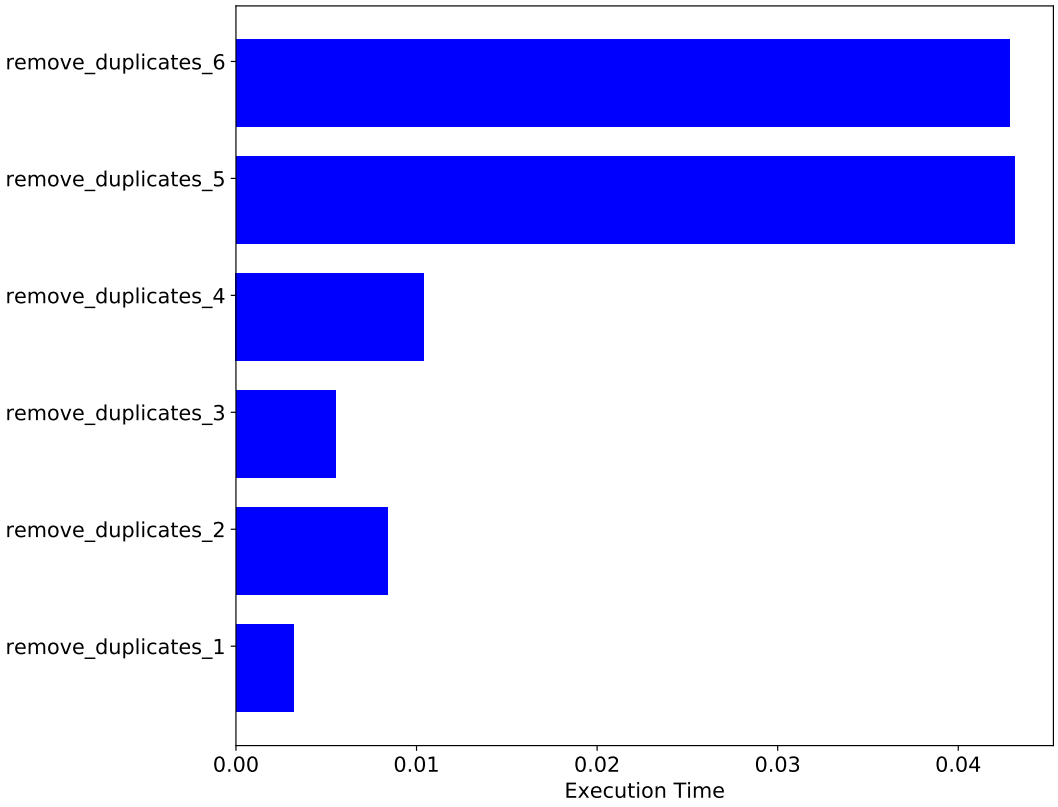


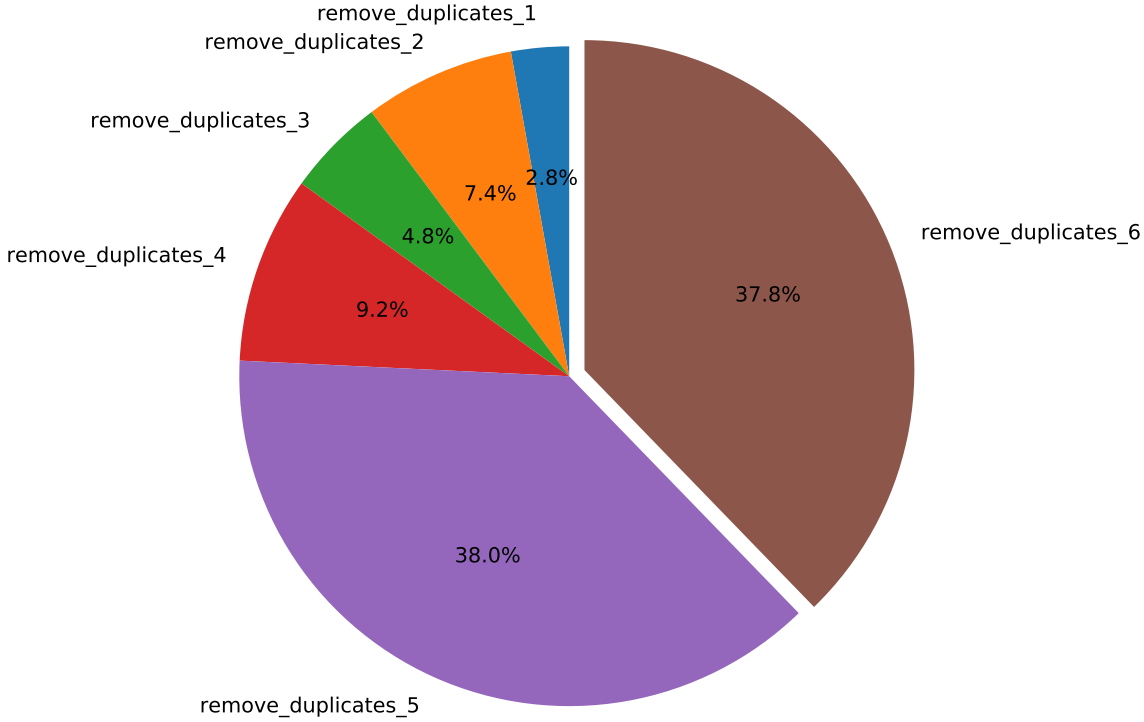
TIMES FOR THE FIRST 76000 NUMBERS



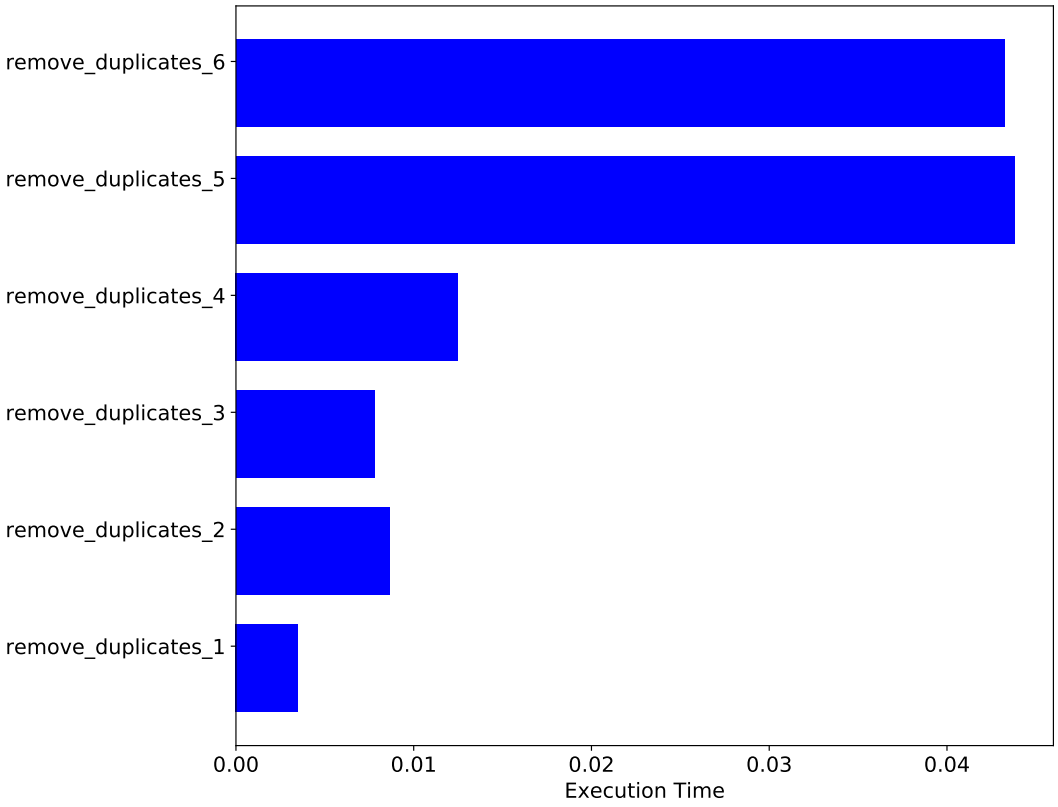


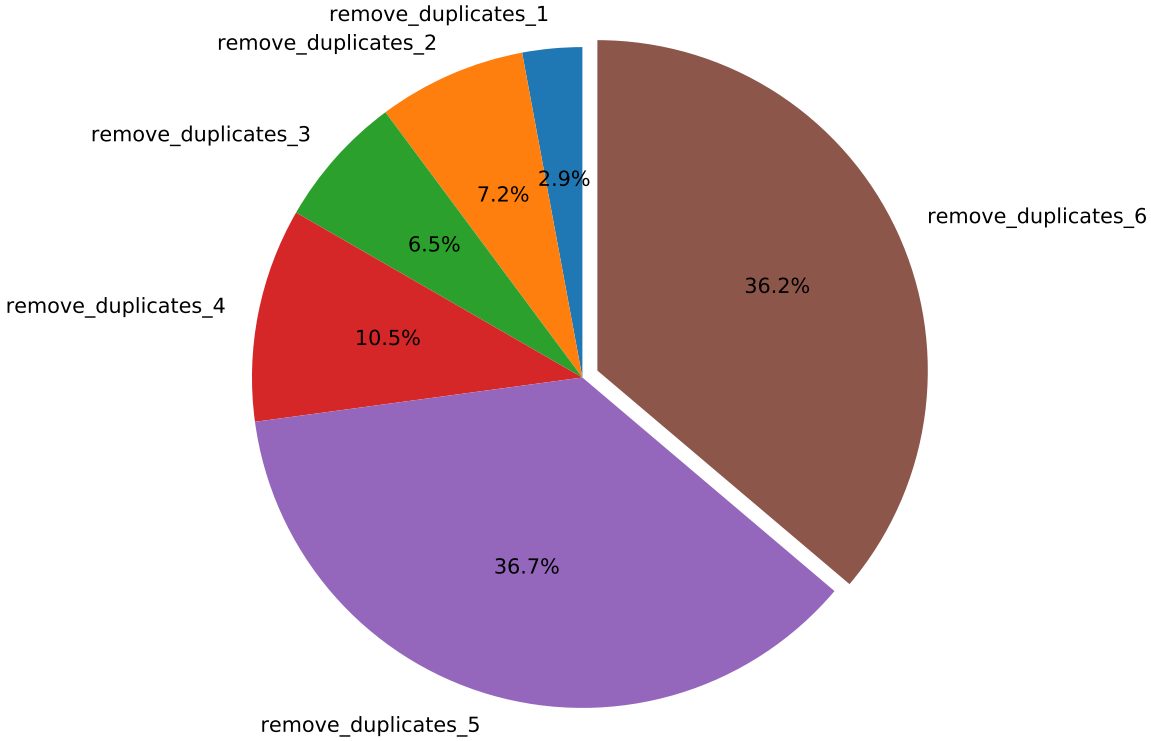
TIMES FOR THE FIRST 78000 NUMBERS



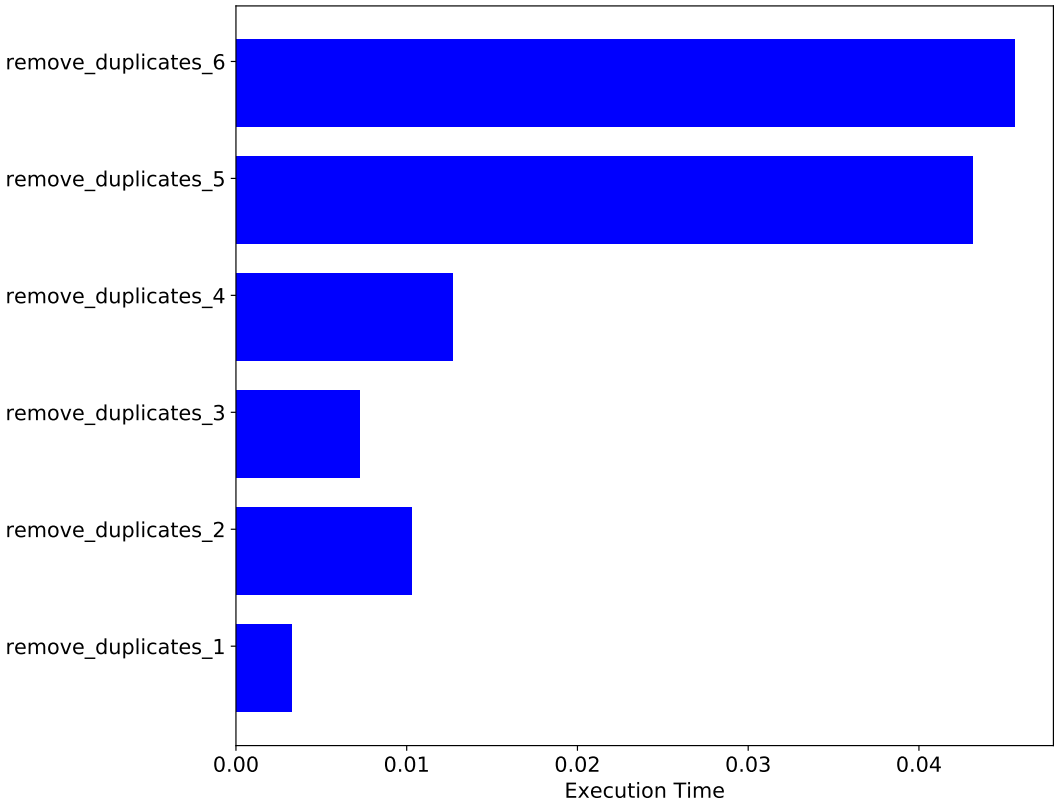


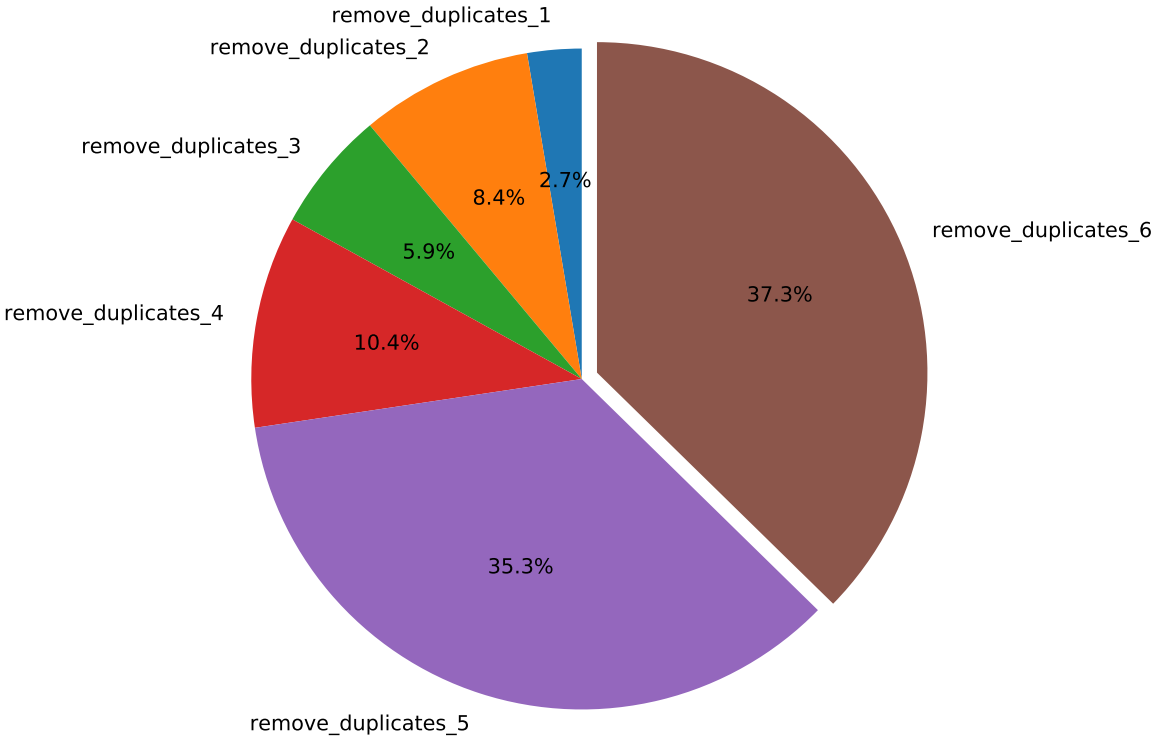
TIMES FOR THE FIRST 80000 NUMBERS



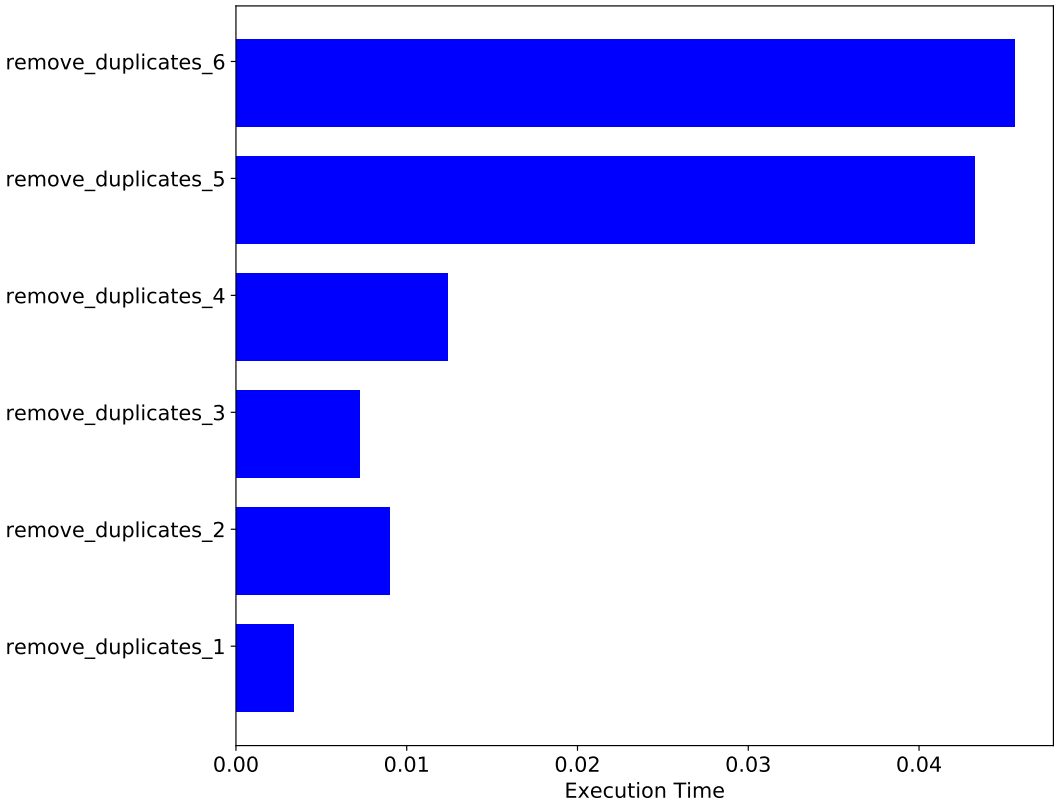


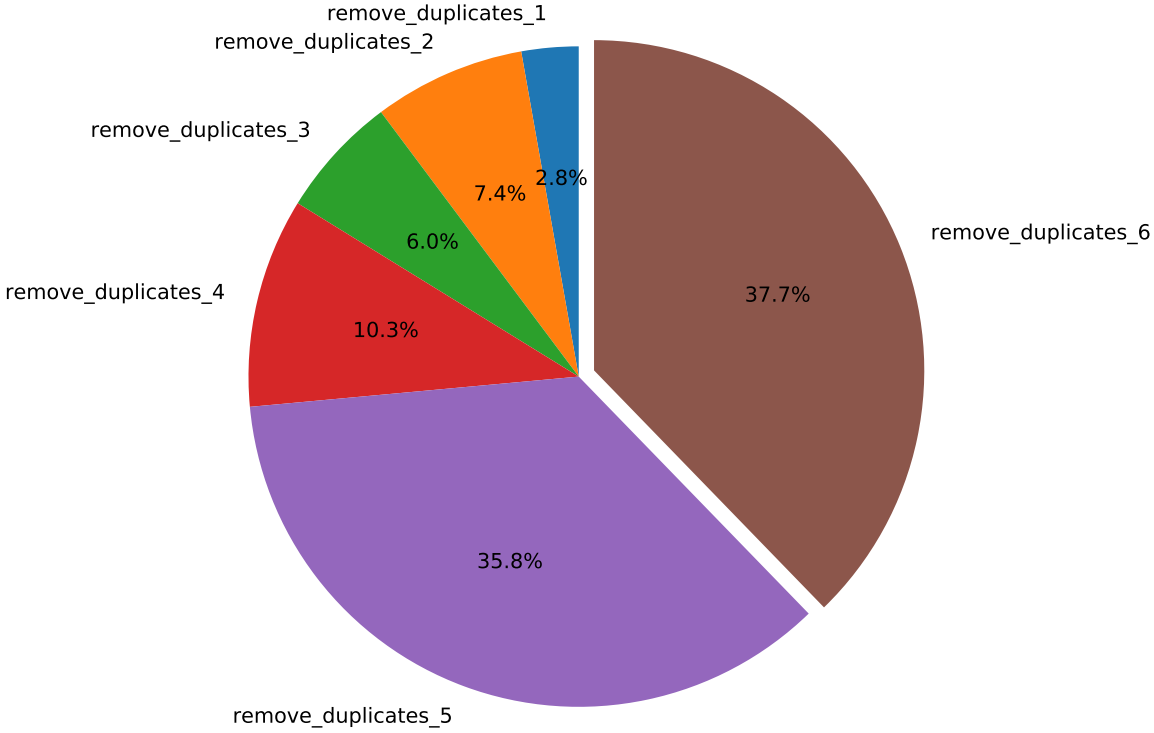
TIMES FOR THE FIRST 82000 NUMBERS



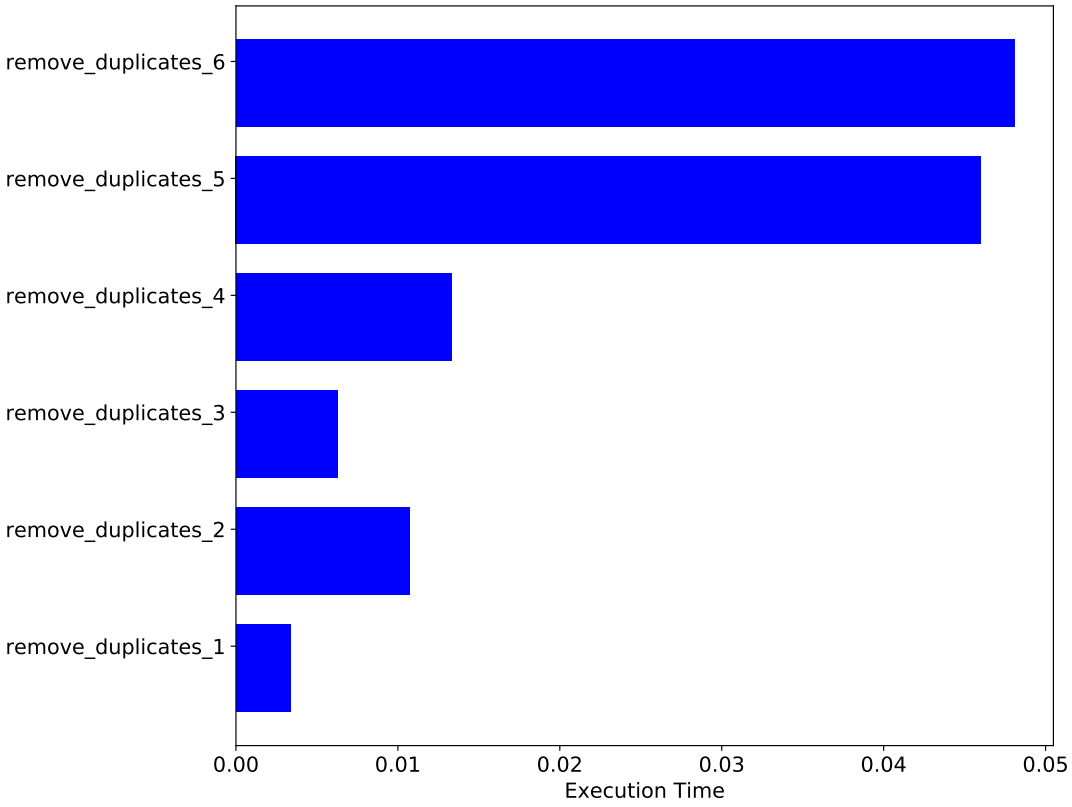


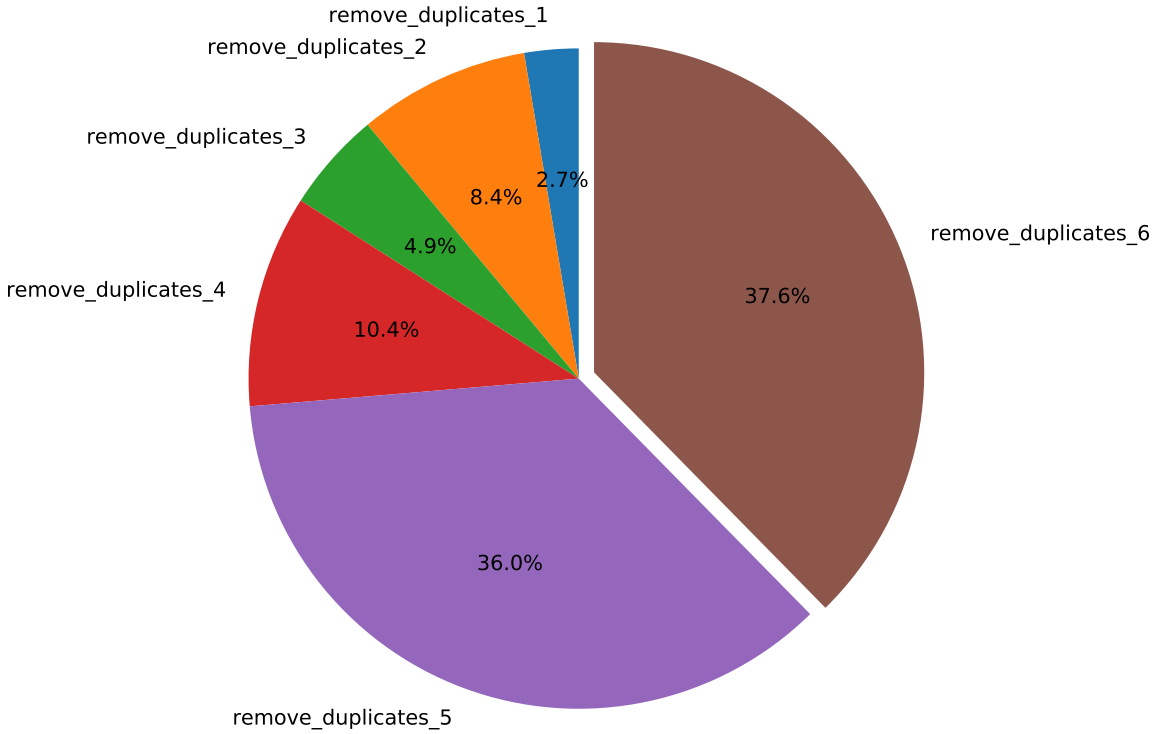
TIMES FOR THE FIRST 84000 NUMBERS



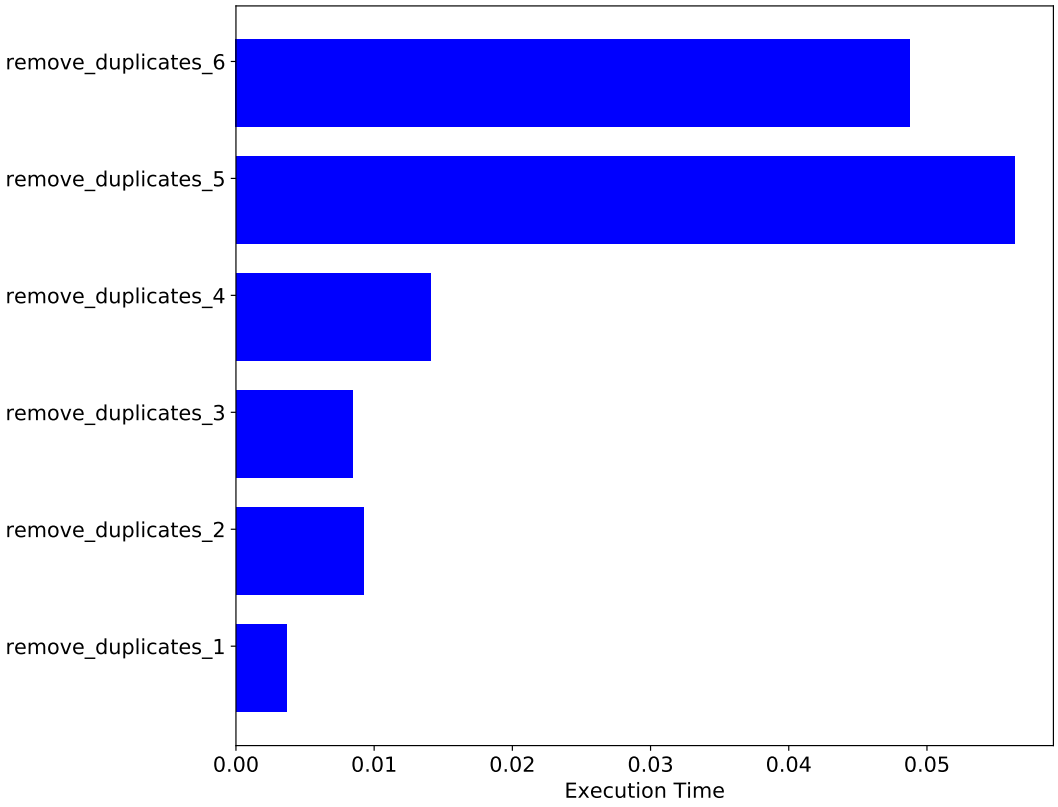


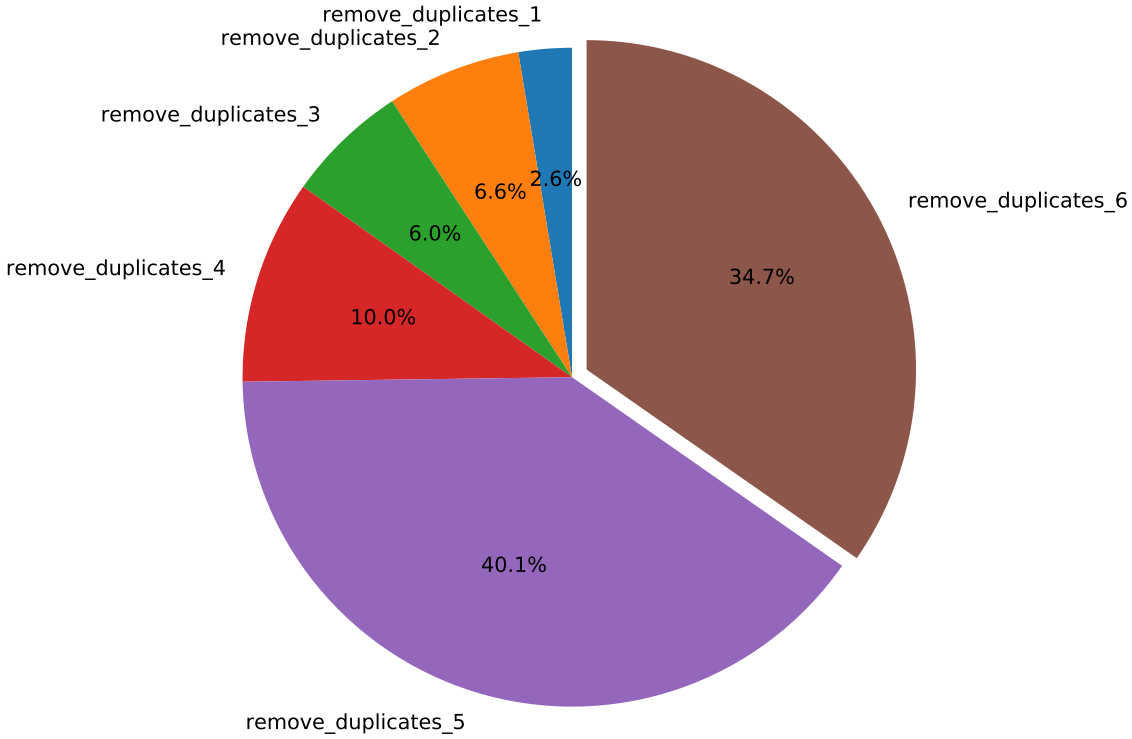
TIMES FOR THE FIRST 86000 NUMBERS



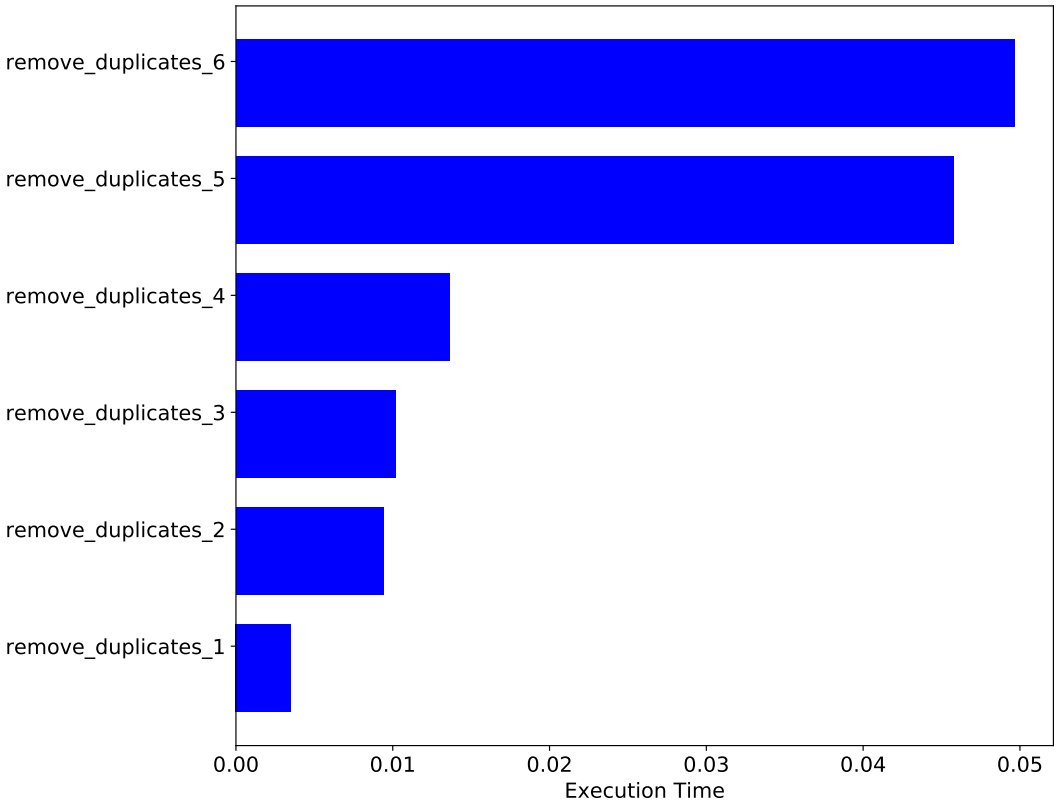


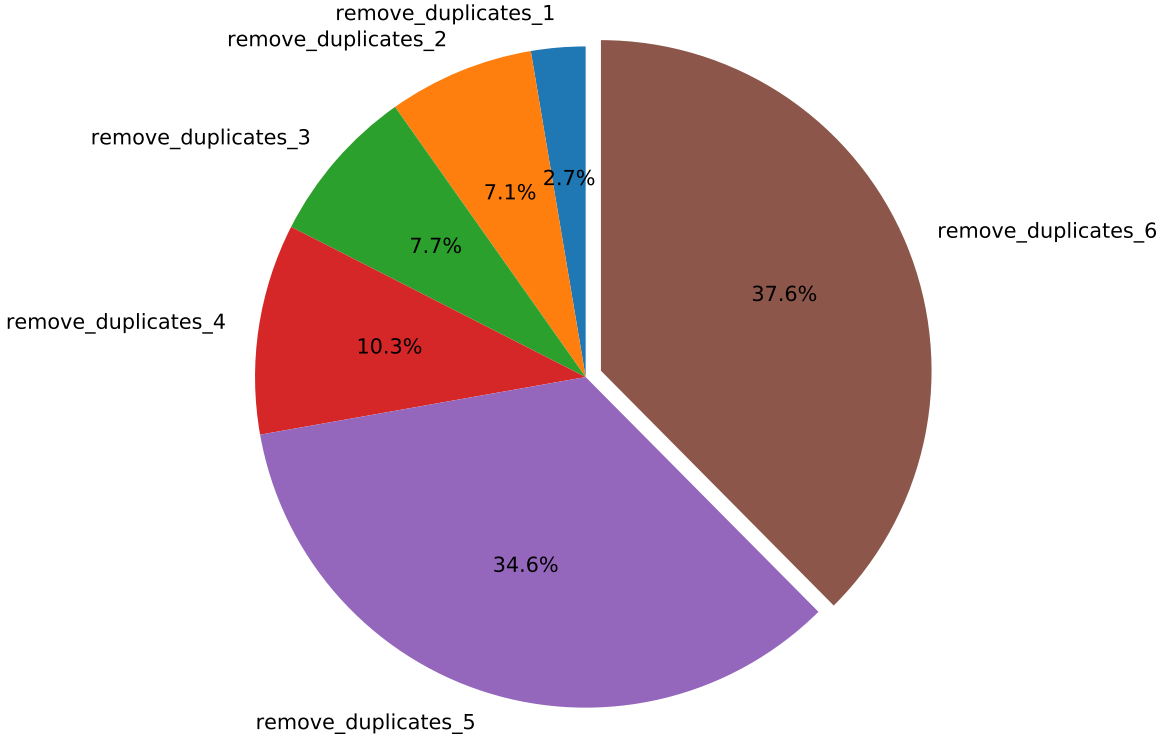
TIMES FOR THE FIRST 88000 NUMBERS



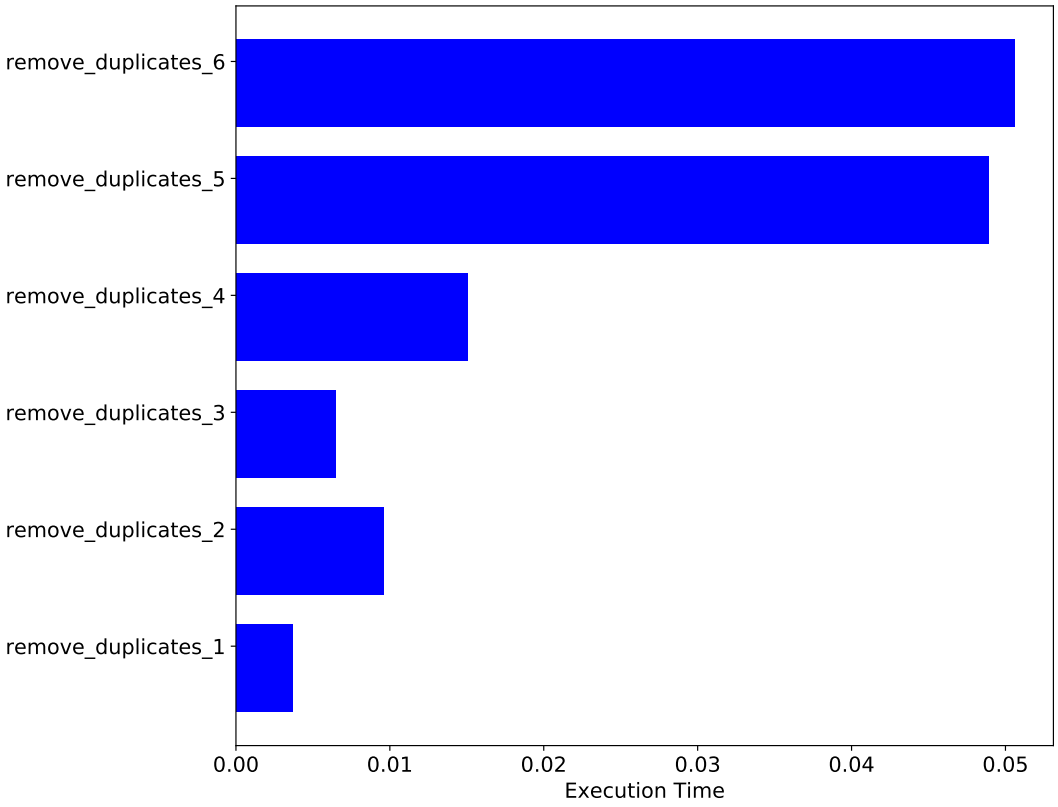


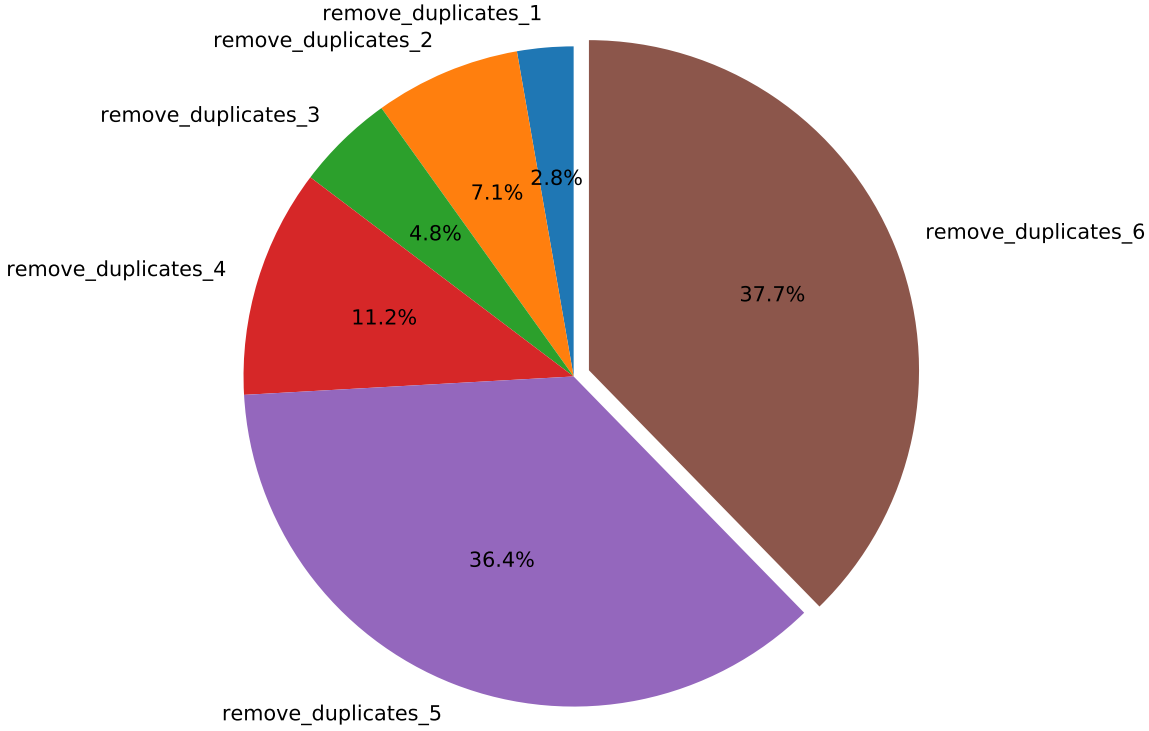
TIMES FOR THE FIRST 90000 NUMBERS



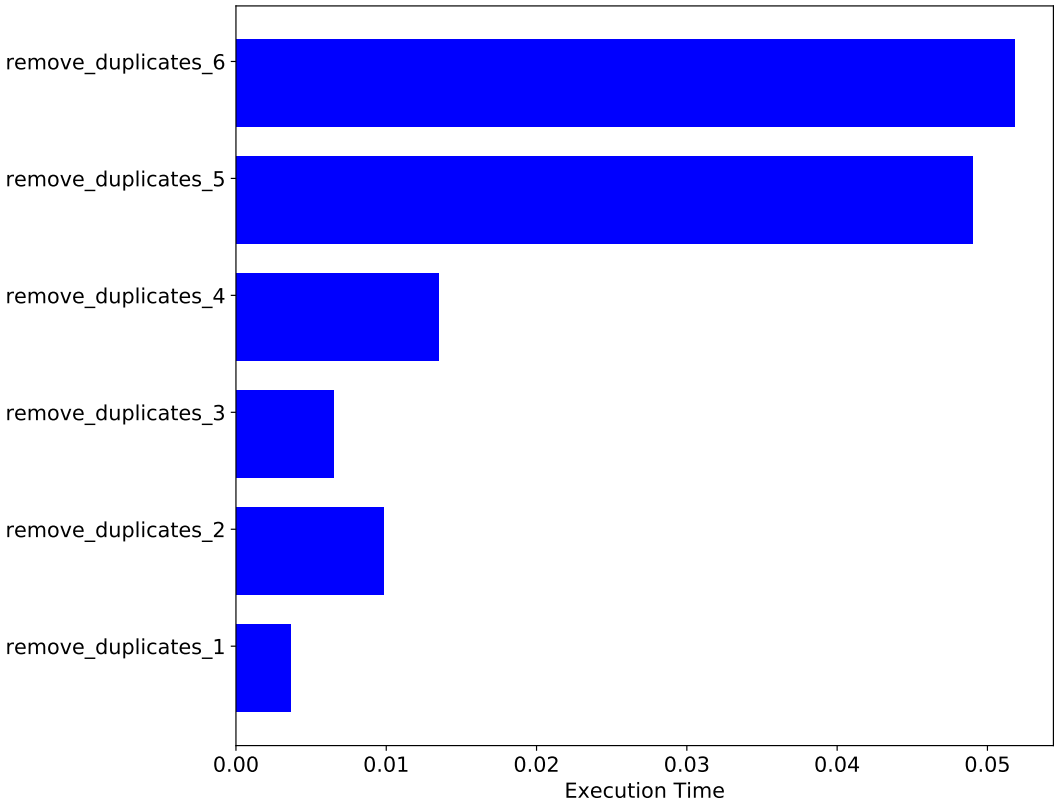


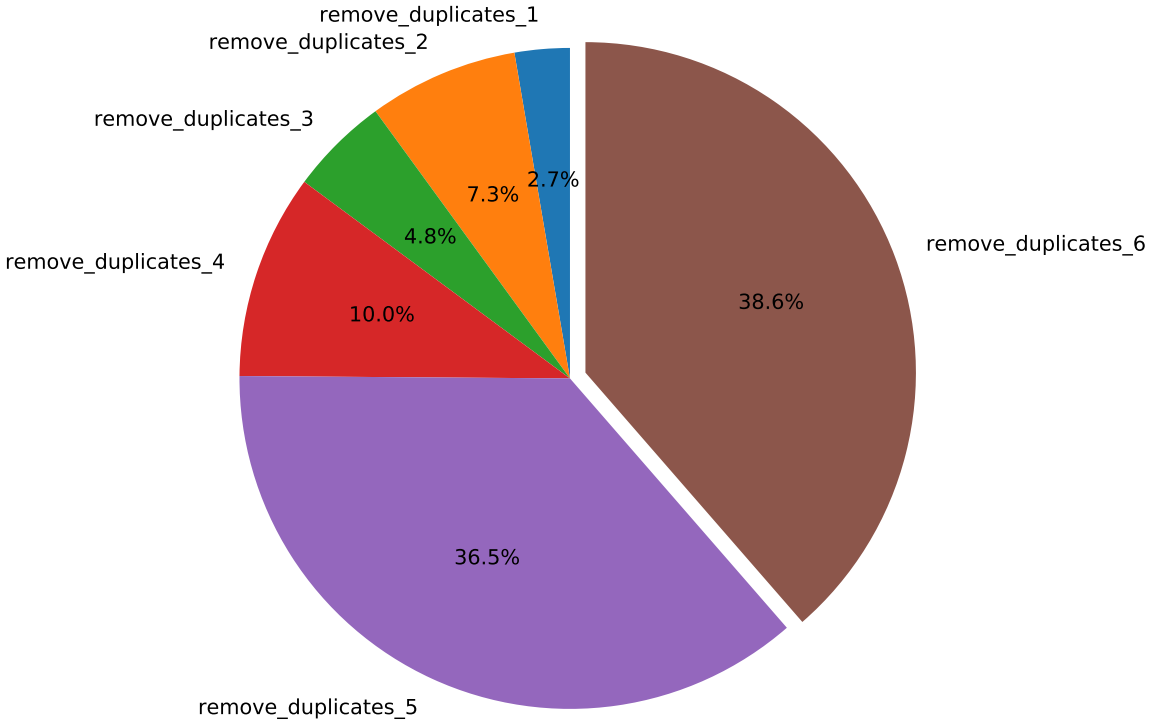
TIMES FOR THE FIRST 92000 NUMBERS



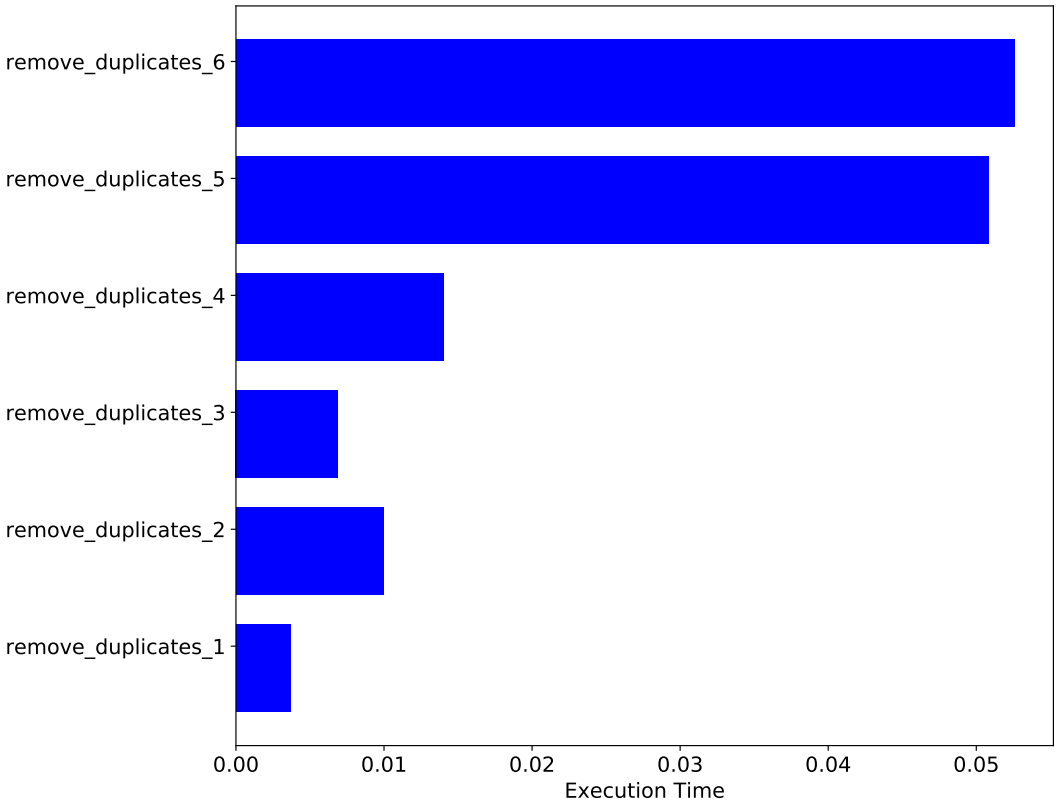


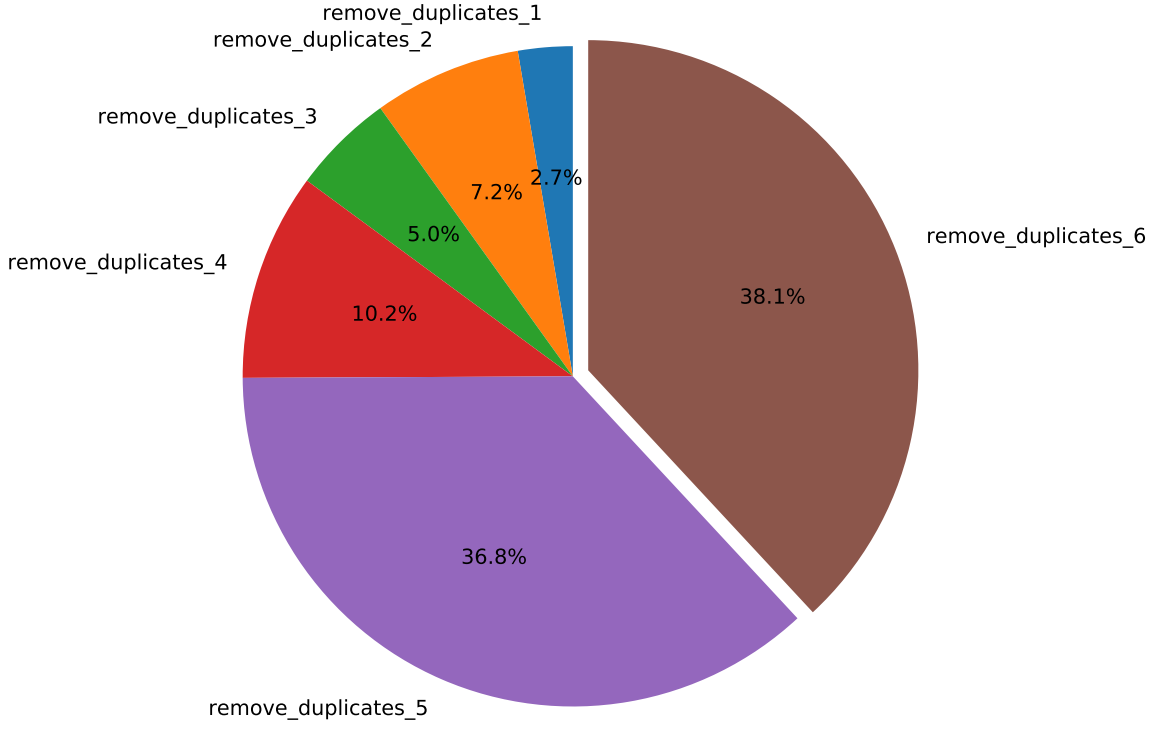
TIMES FOR THE FIRST 94000 NUMBERS



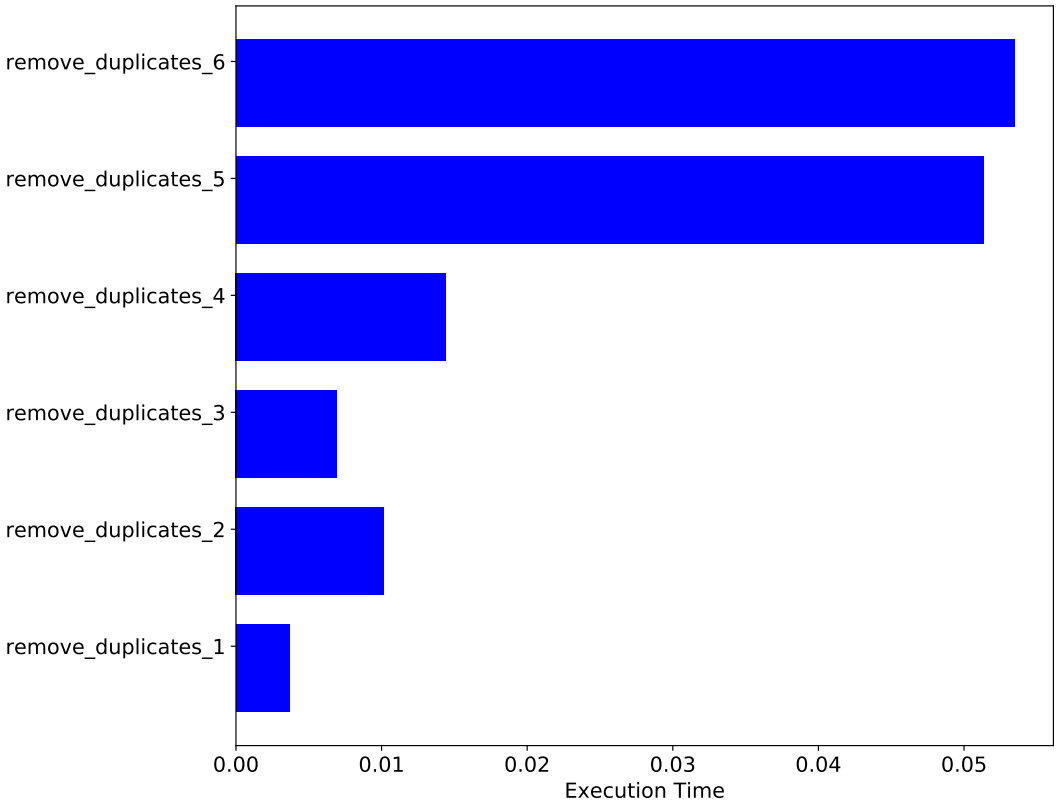


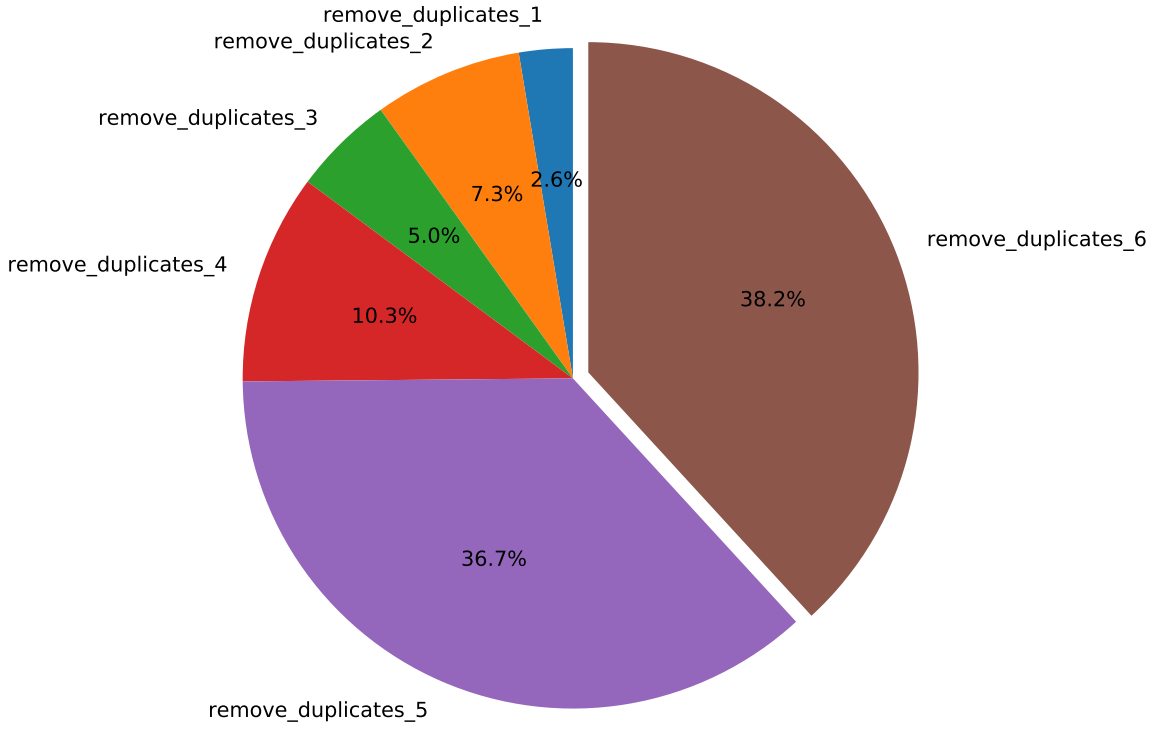
TIMES FOR THE FIRST 96000 NUMBERS



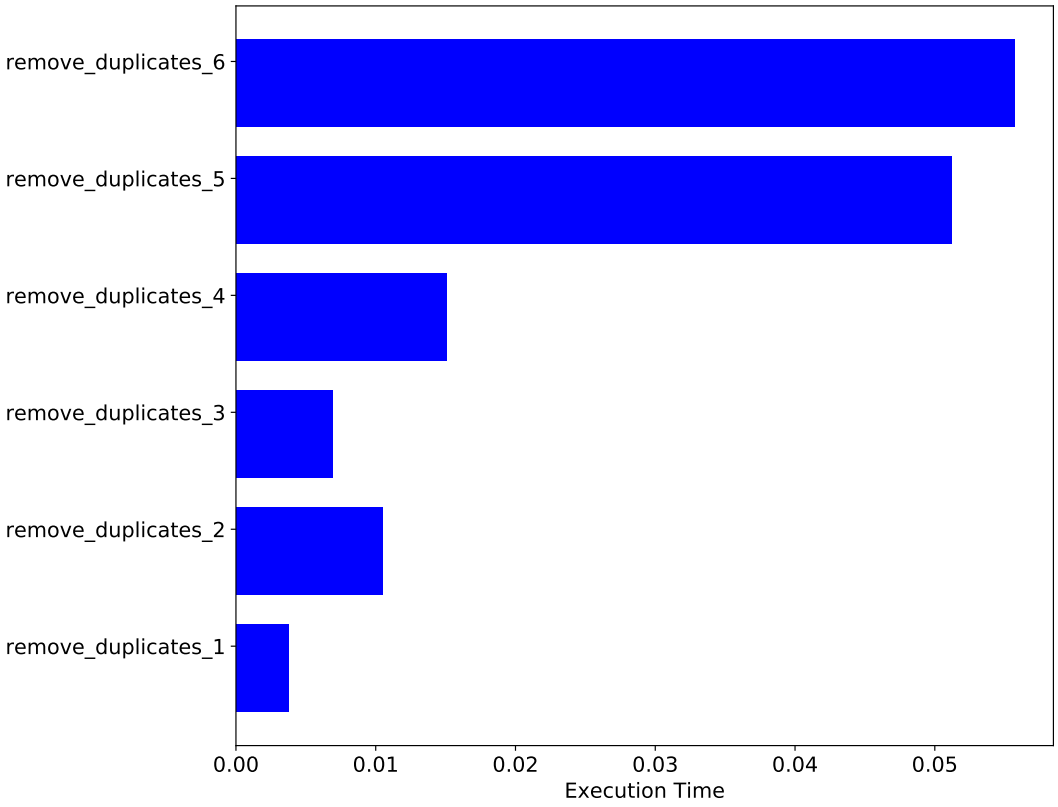


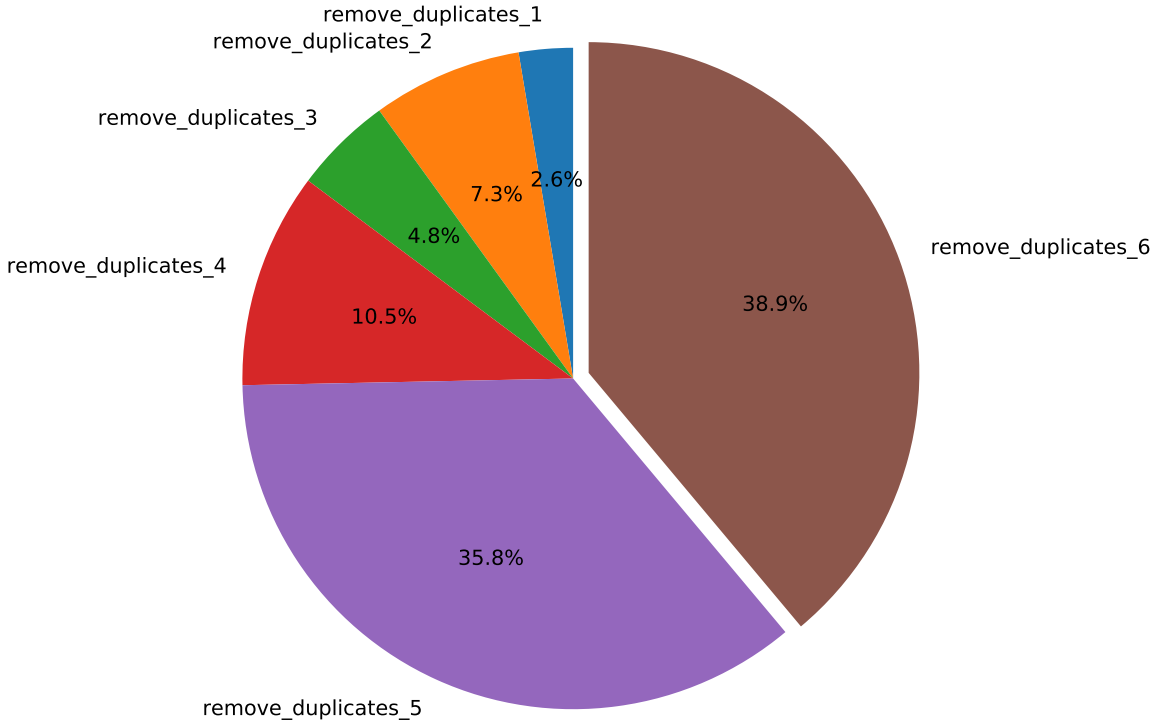
TIMES FOR THE FIRST 98000 NUMBERS



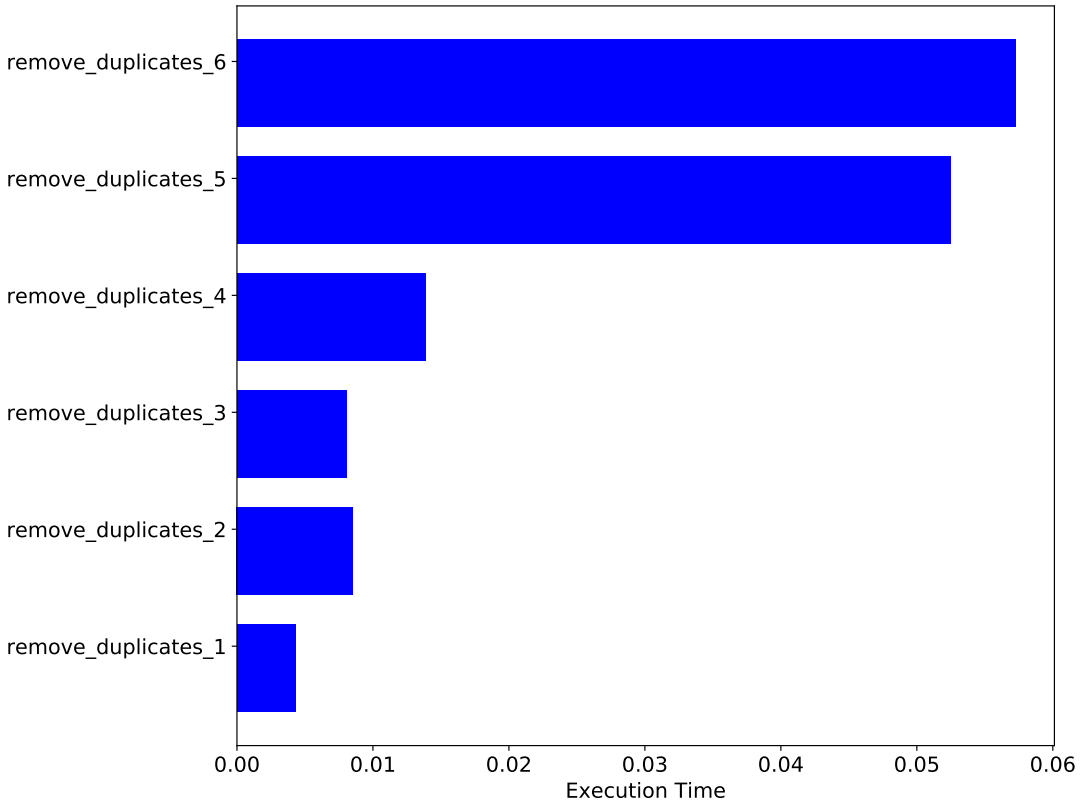


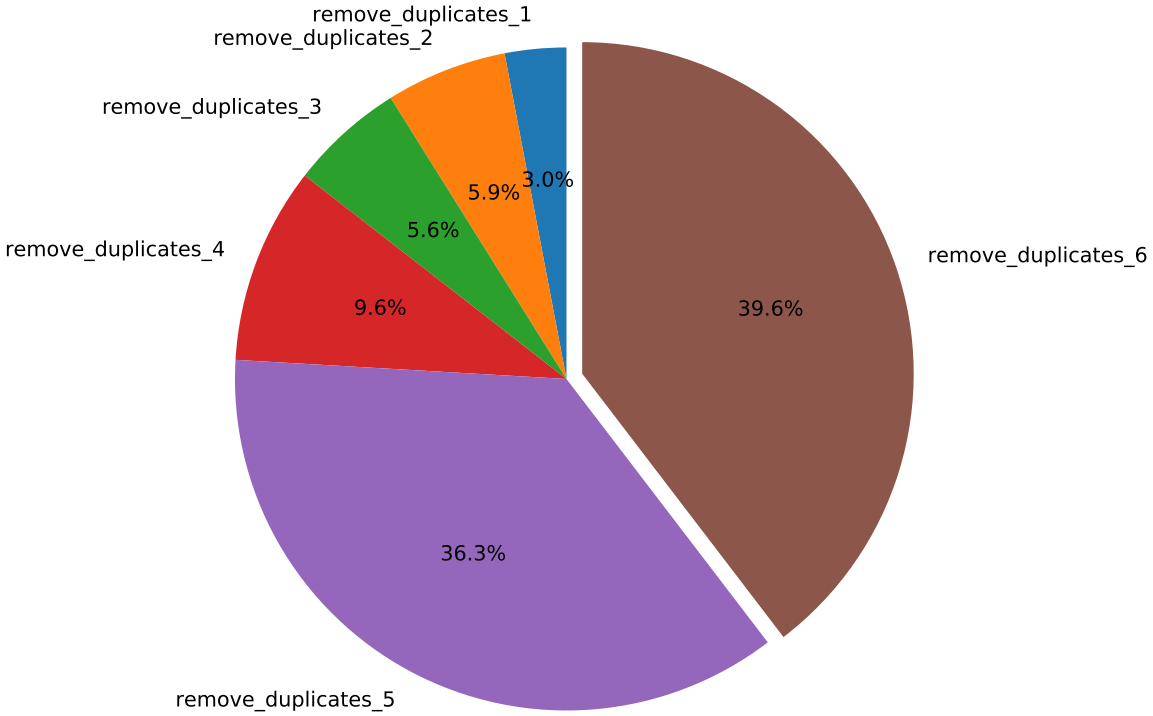
TIMES FOR THE FIRST 100000 NUMBERS



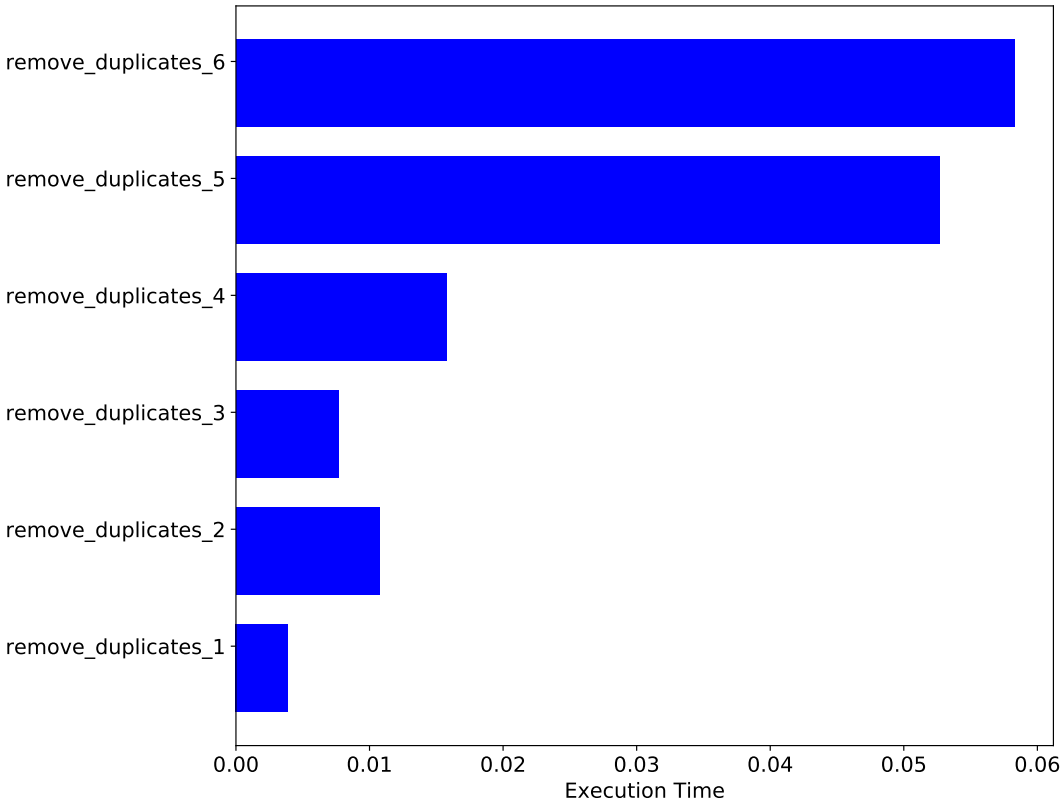


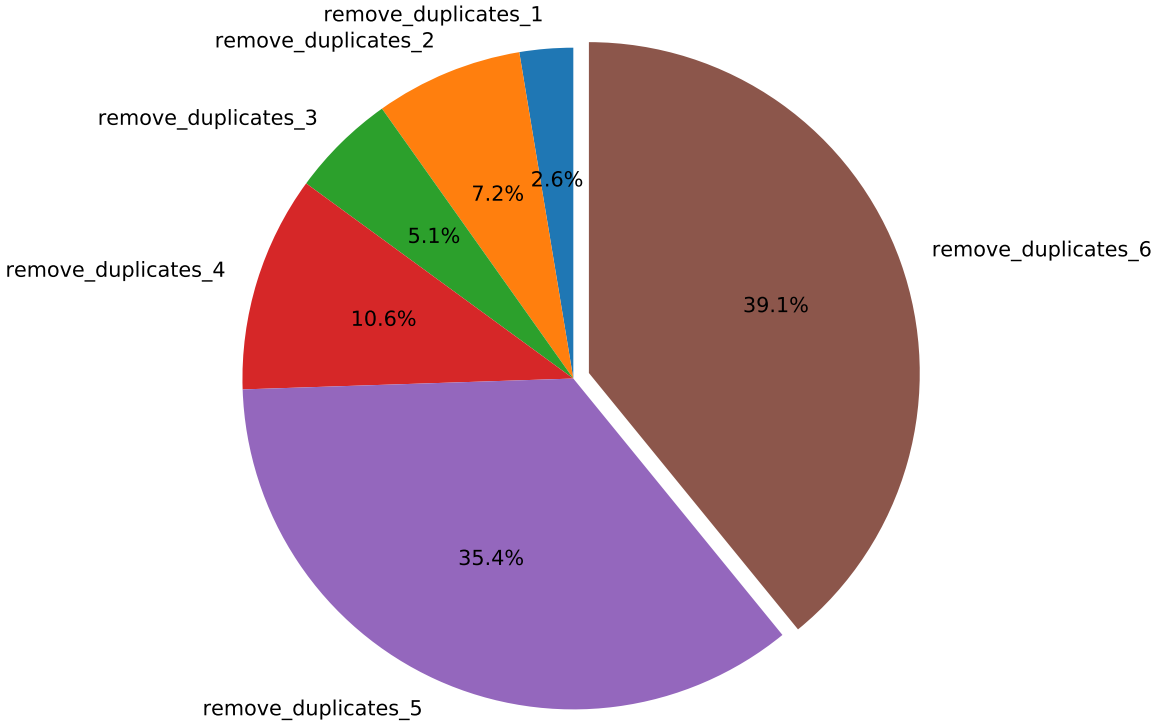
TIMES FOR THE FIRST 102000 NUMBERS



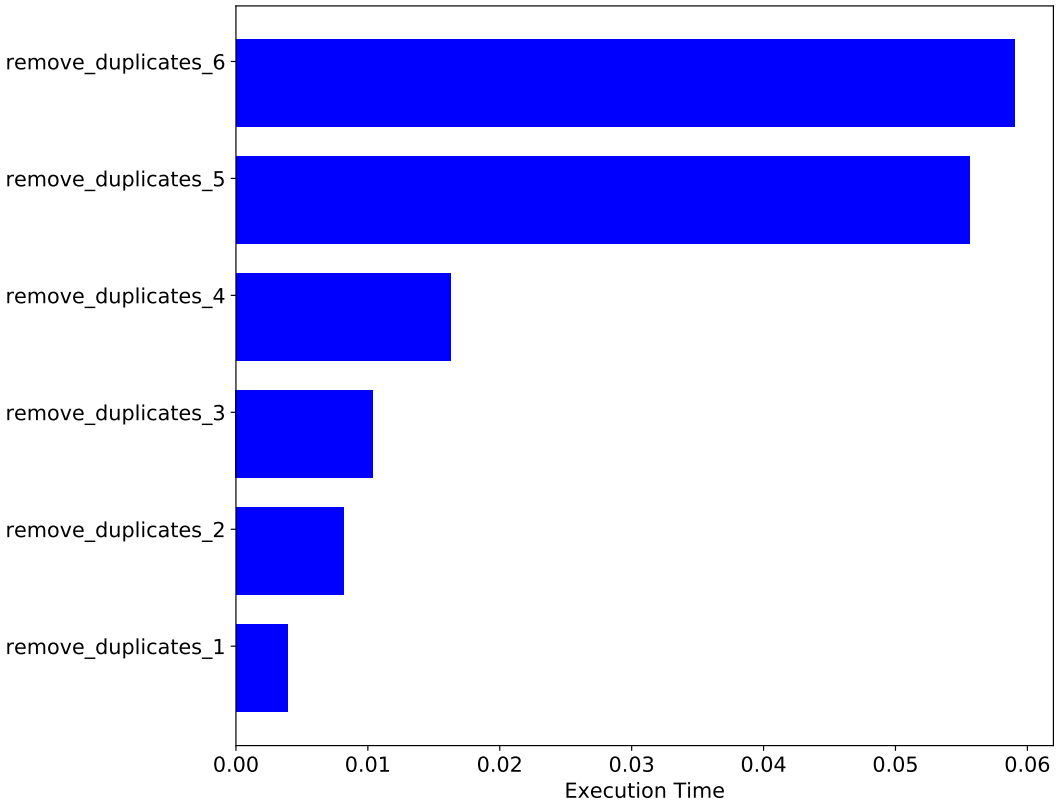


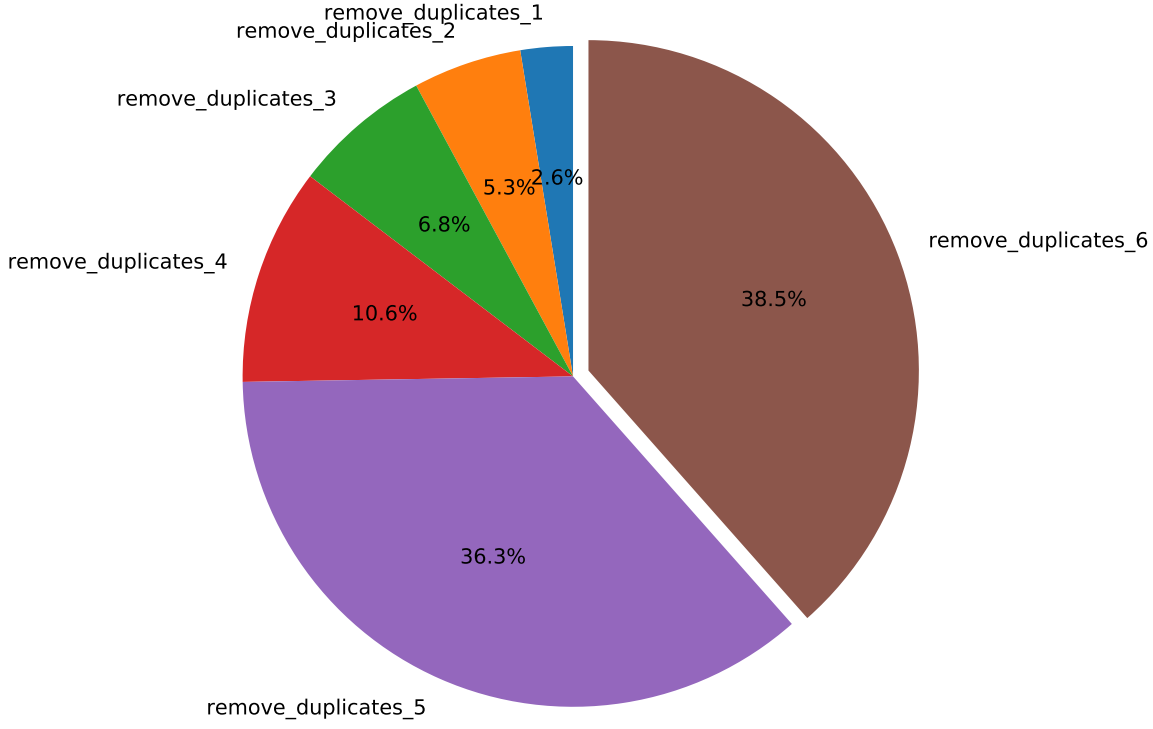
TIMES FOR THE FIRST 104000 NUMBERS



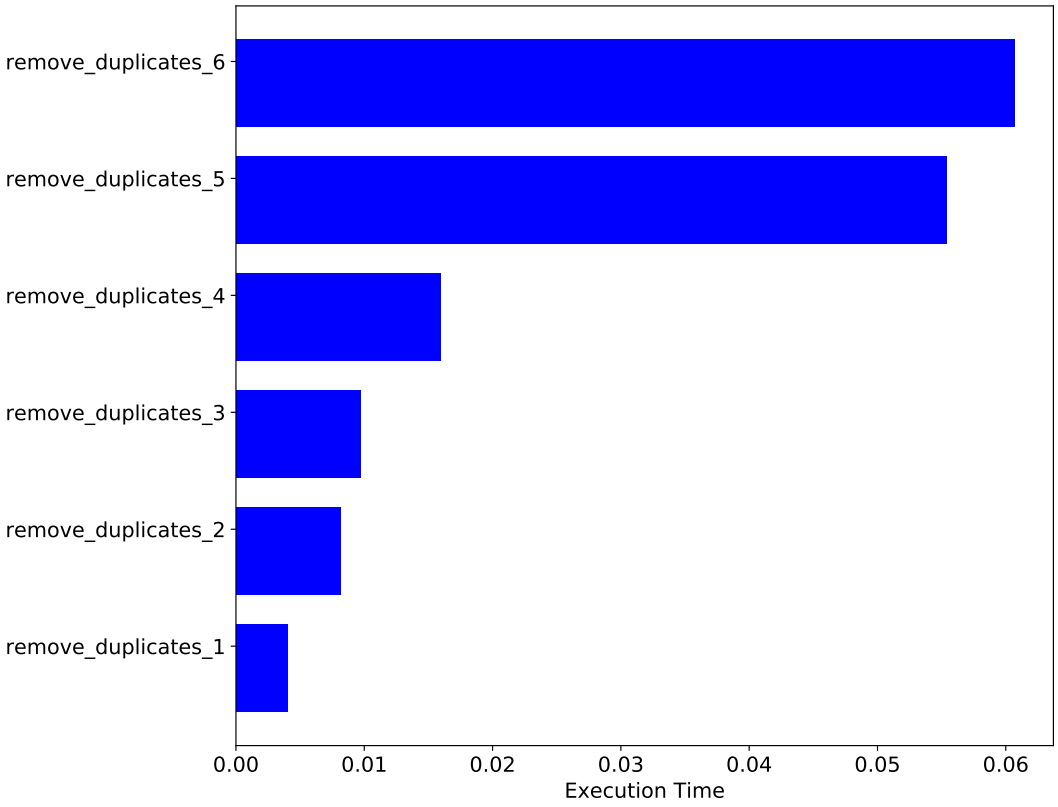


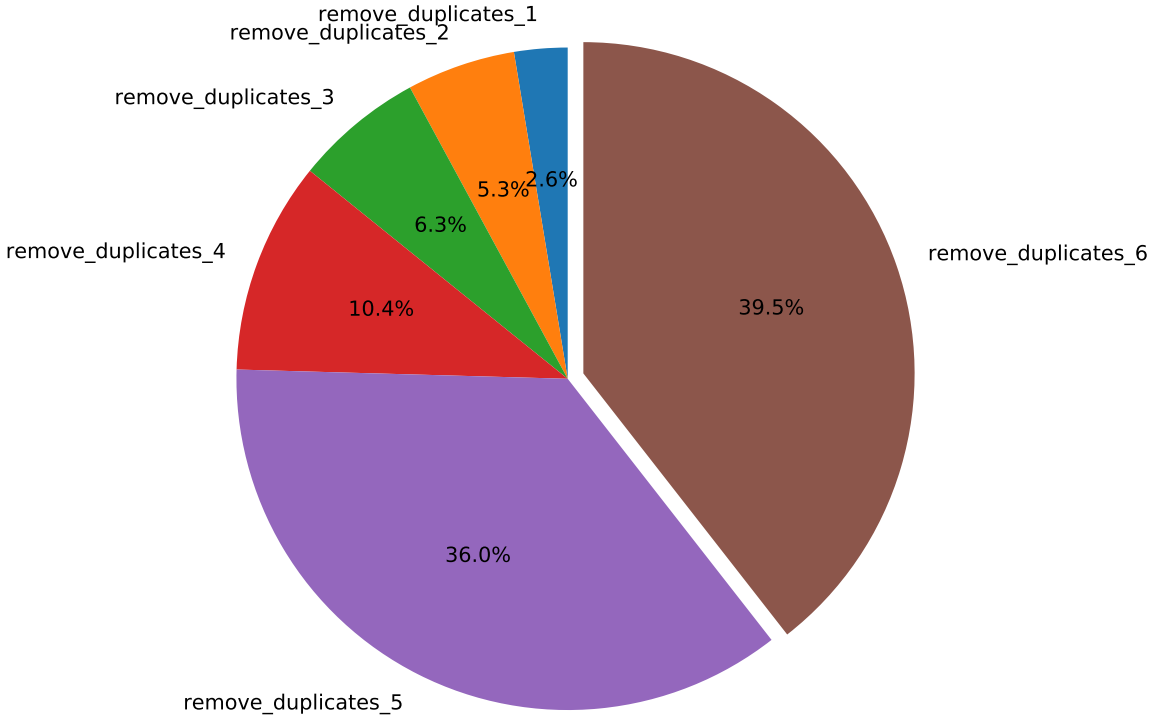
TIMES FOR THE FIRST 106000 NUMBERS



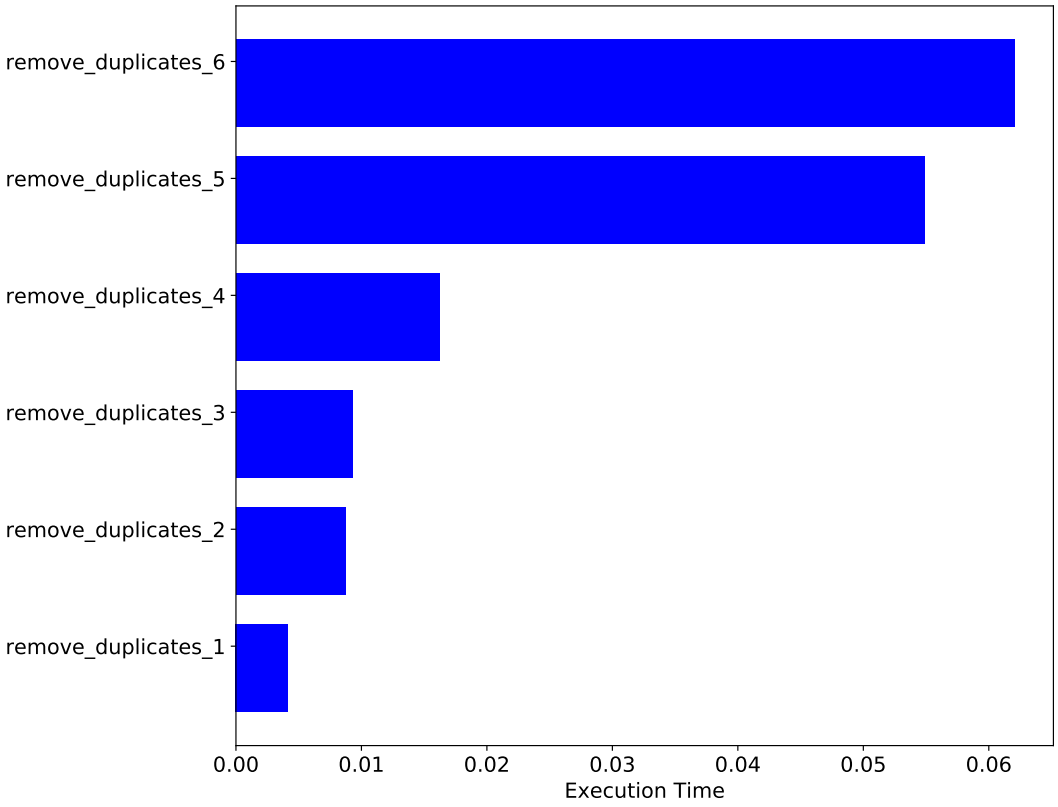


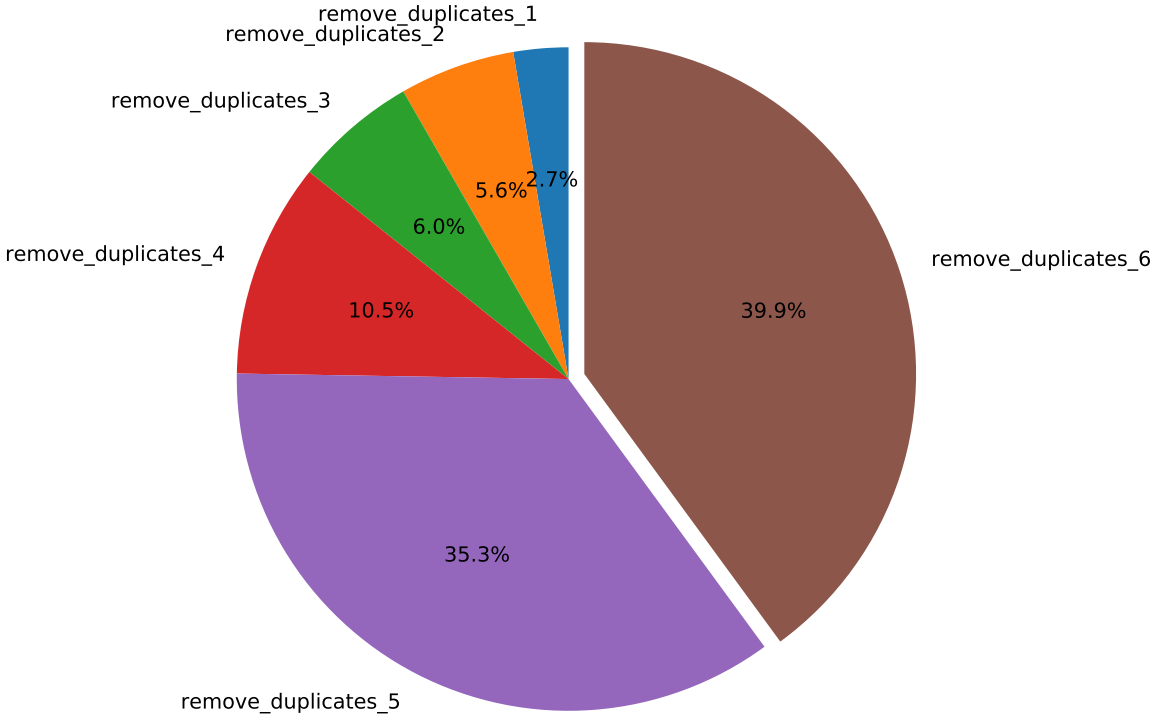
TIMES FOR THE FIRST 108000 NUMBERS



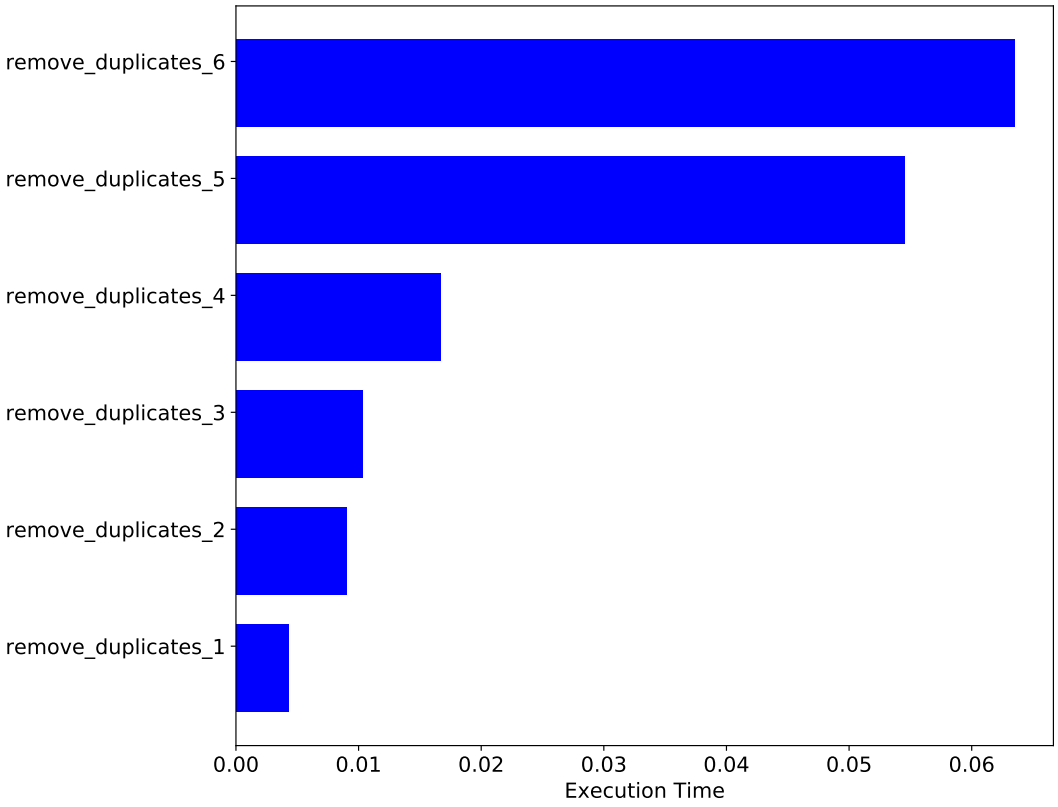


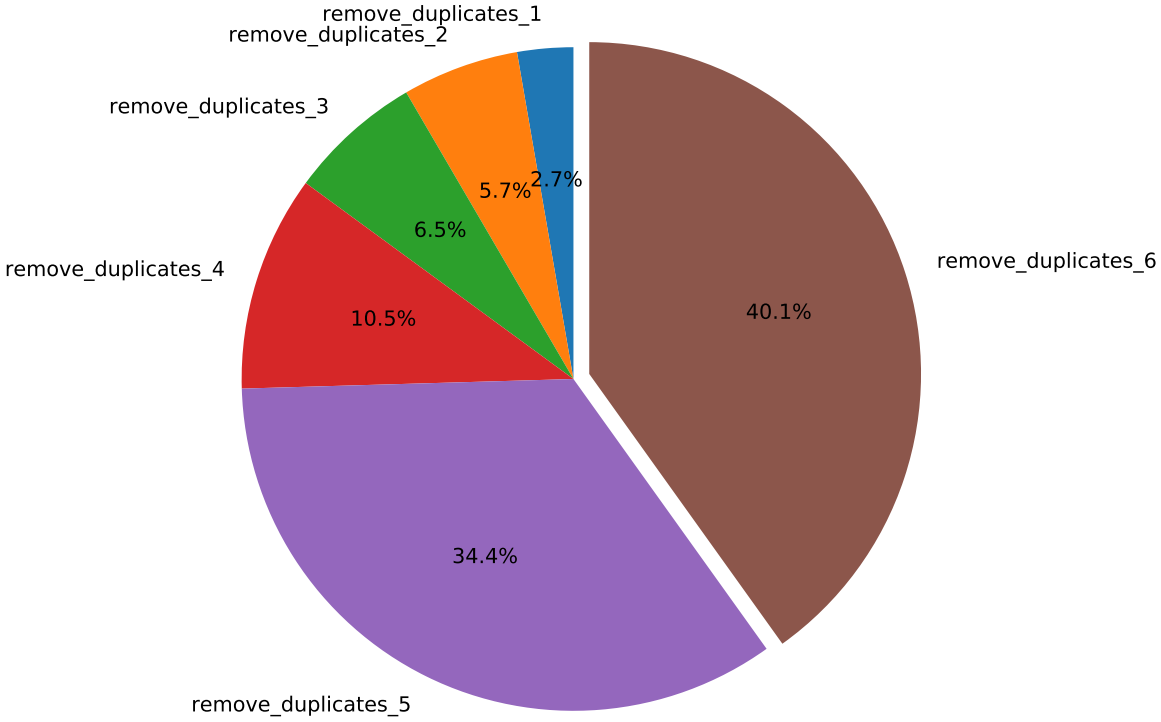
TIMES FOR THE FIRST 110000 NUMBERS



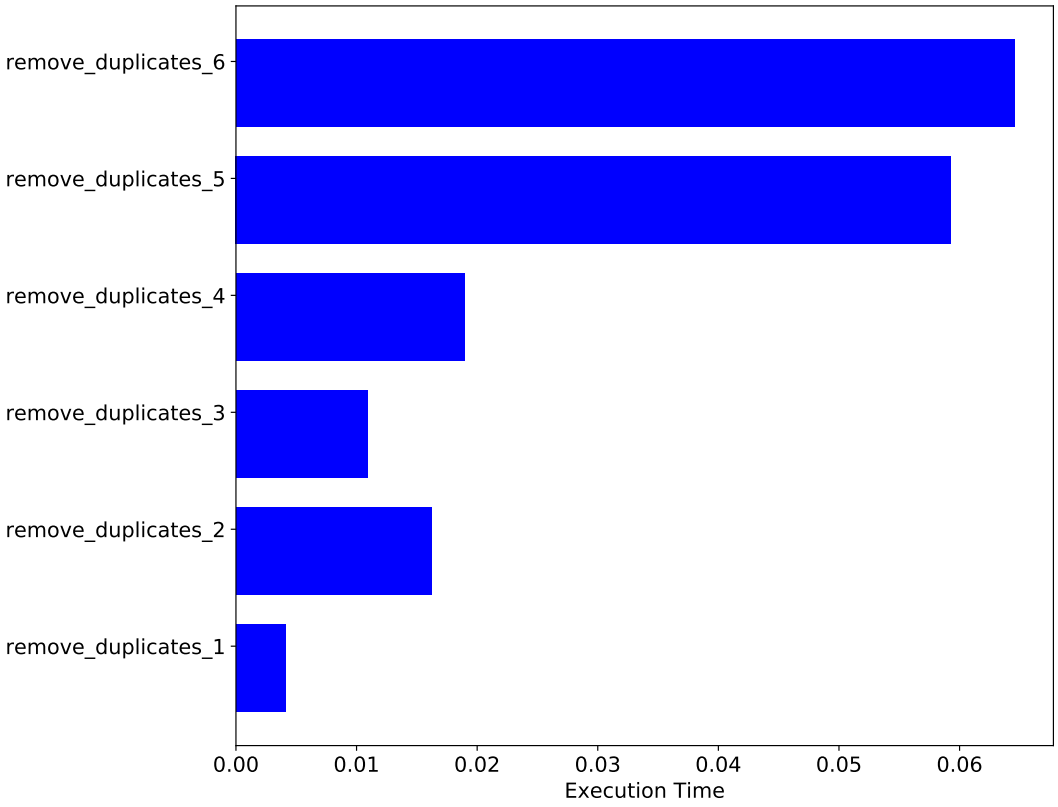


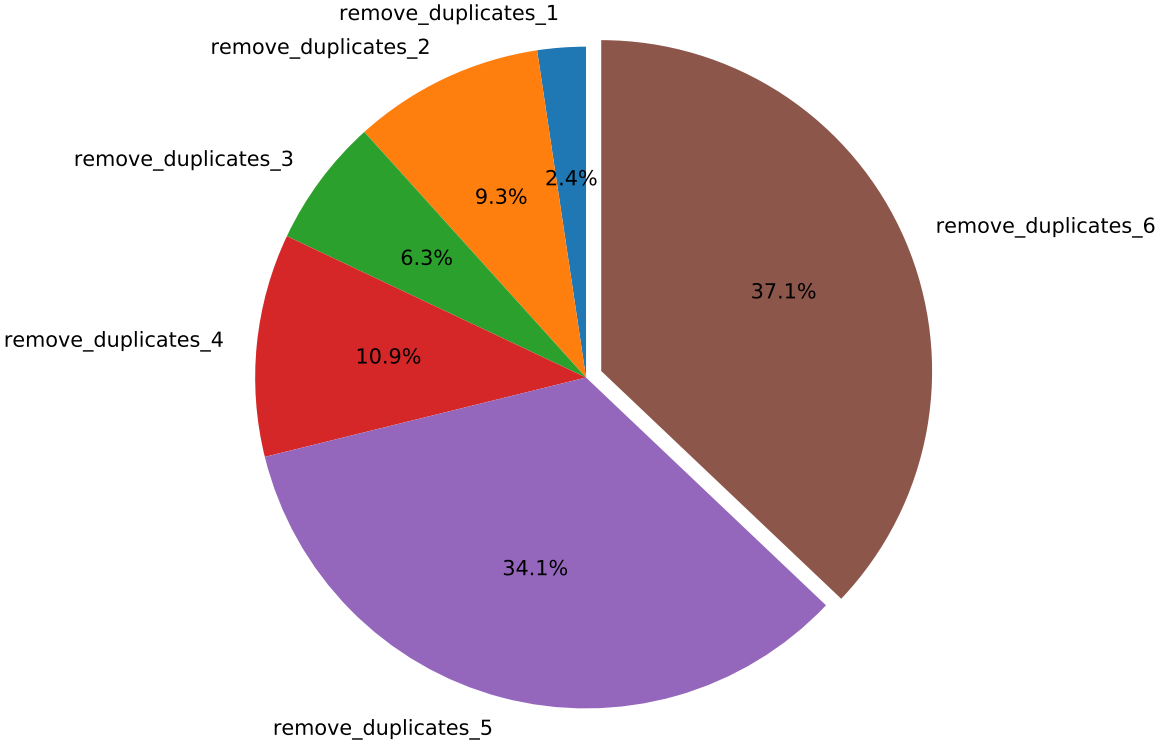
TIMES FOR THE FIRST 112000 NUMBERS



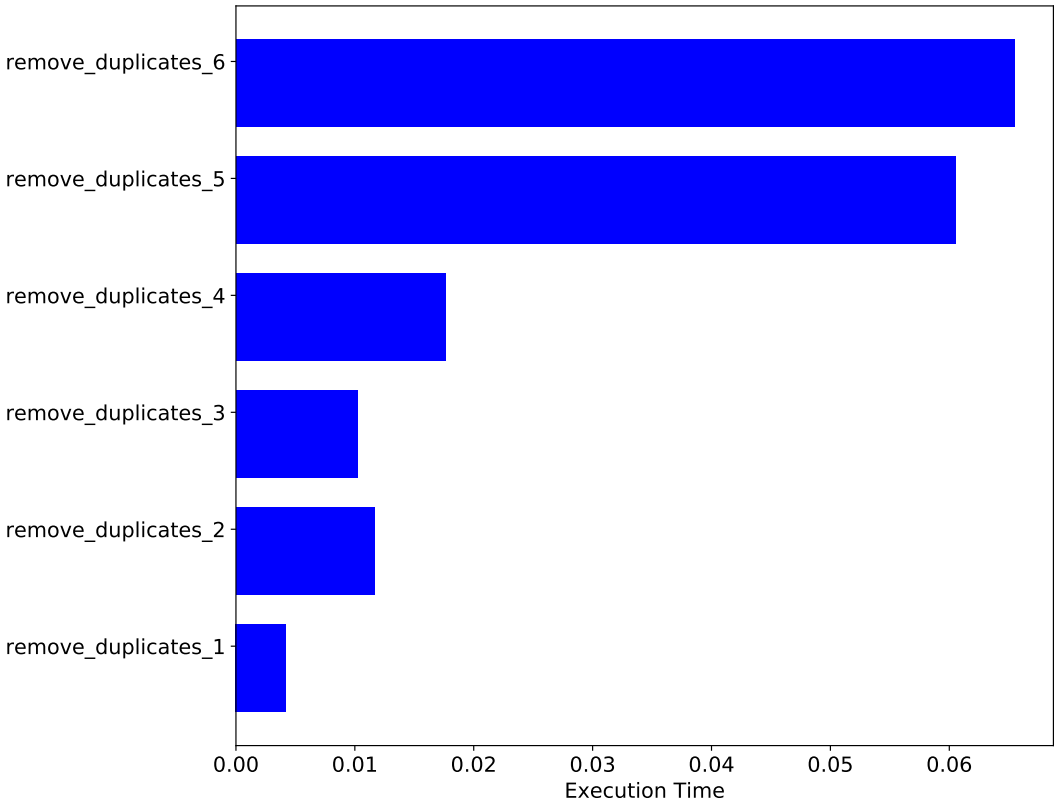


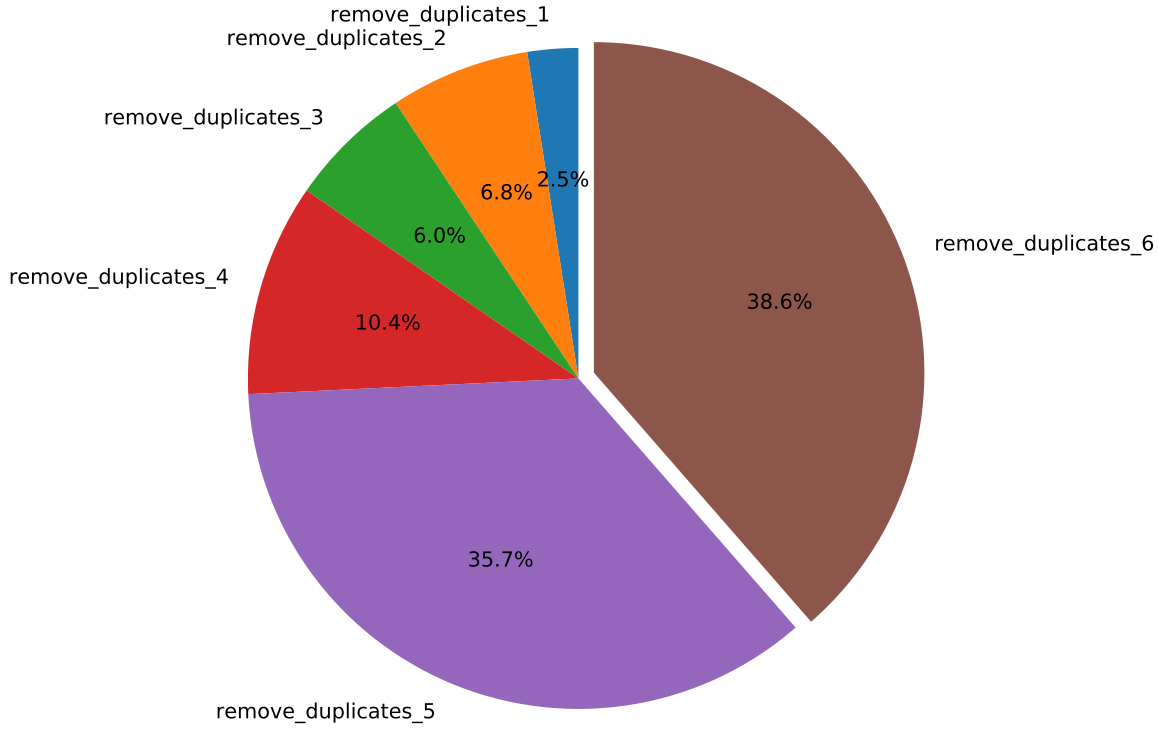
TIMES FOR THE FIRST 114000 NUMBERS



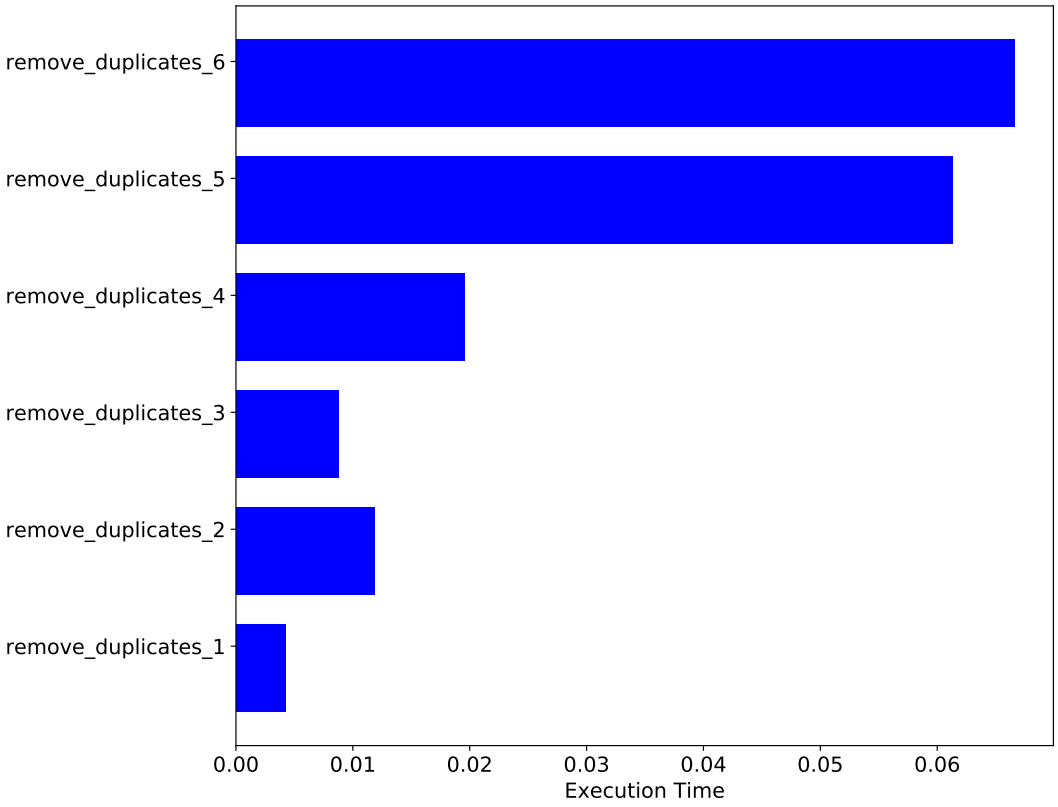


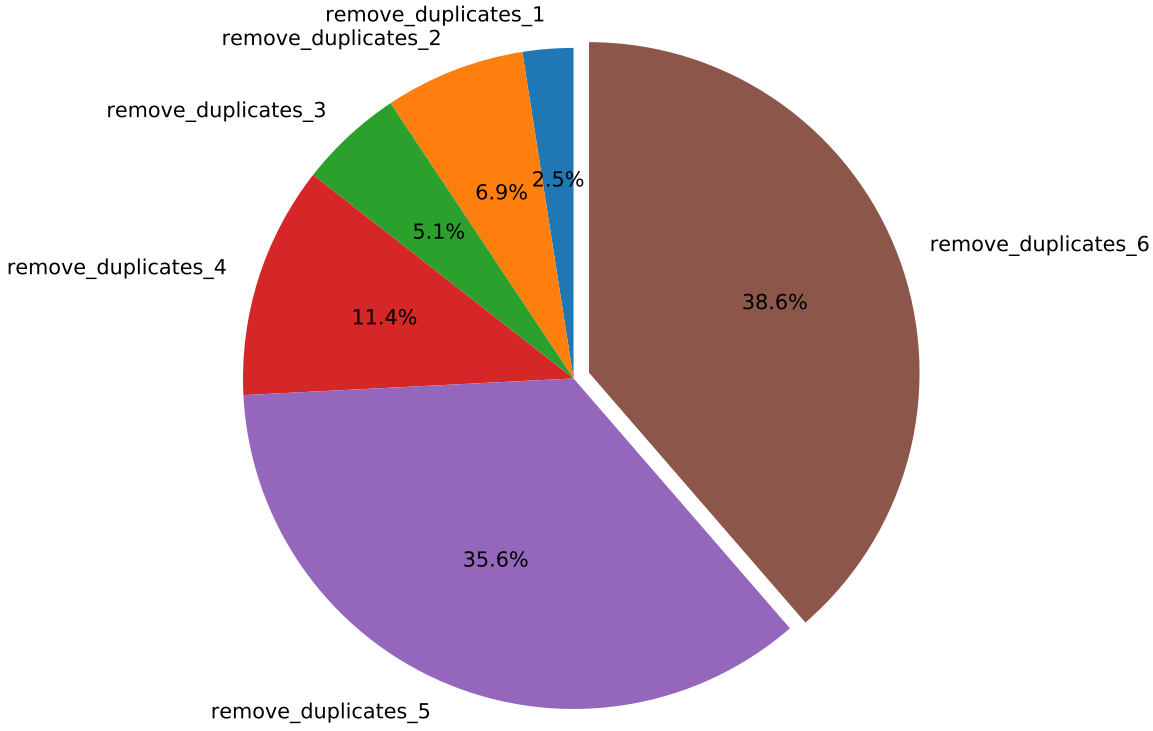
TIMES FOR THE FIRST 116000 NUMBERS



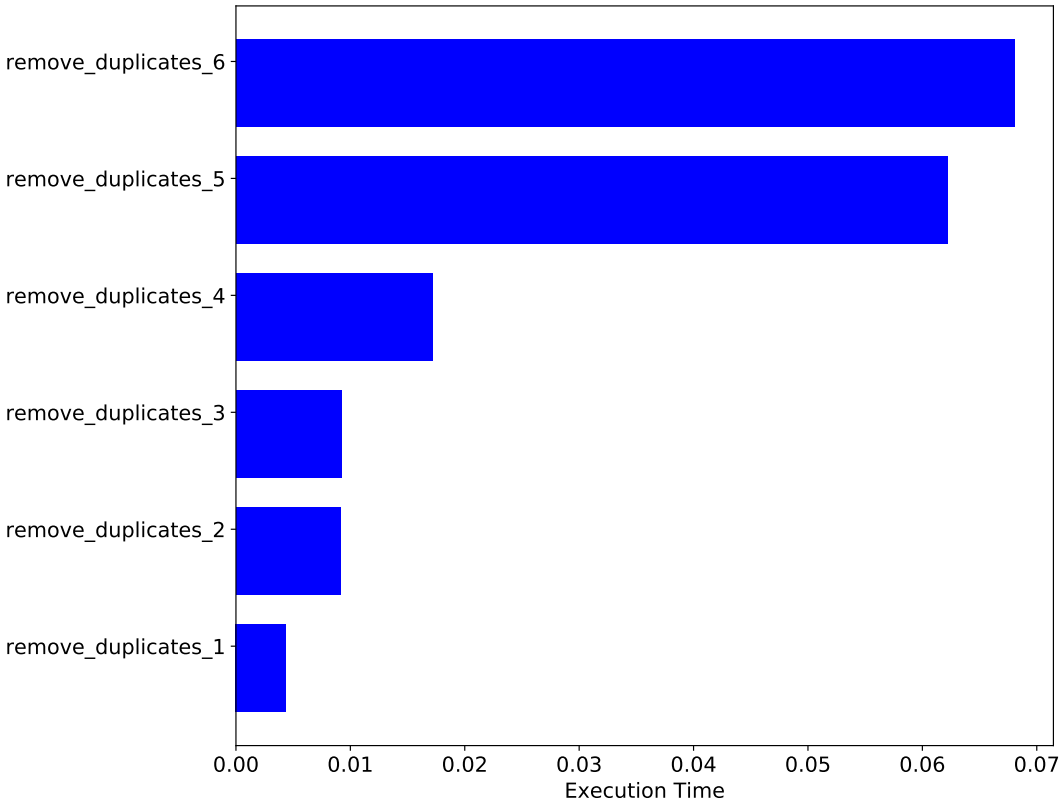


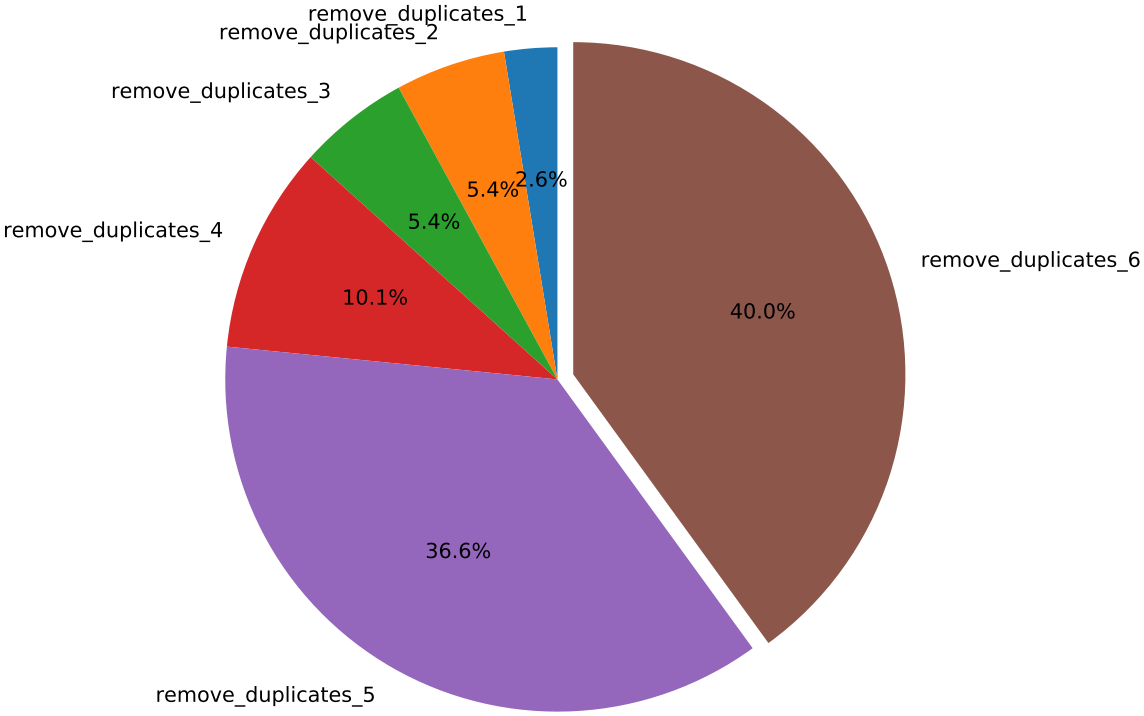
TIMES FOR THE FIRST 118000 NUMBERS



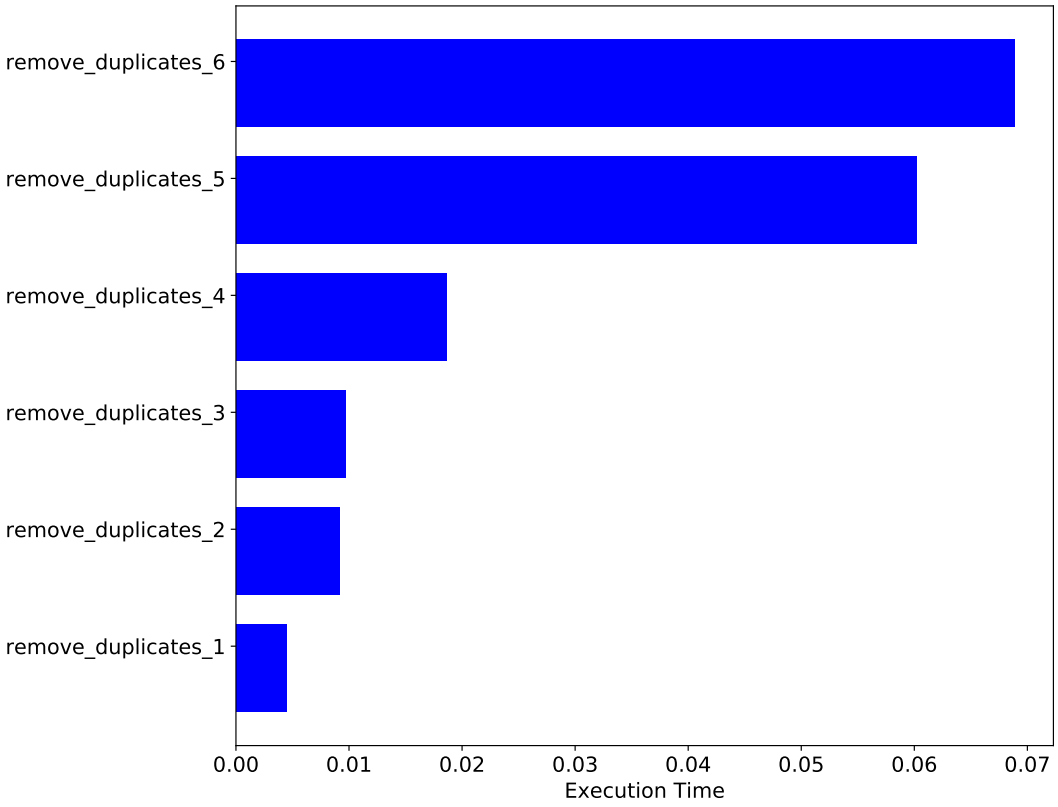


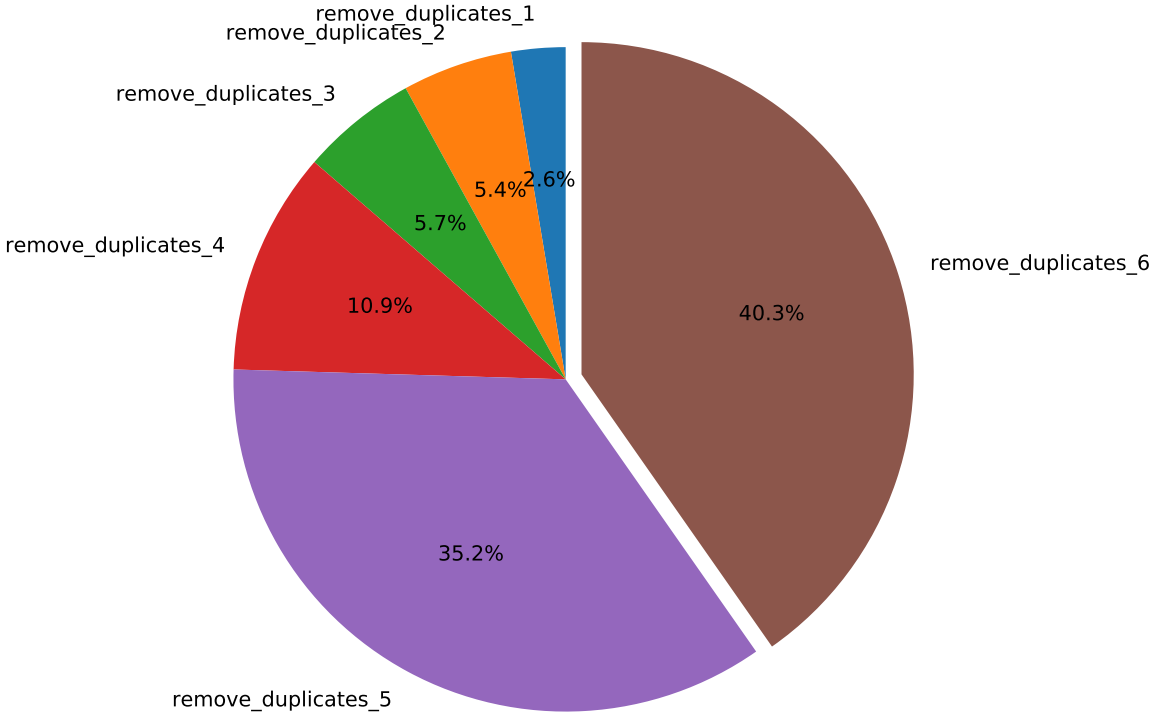
TIMES FOR THE FIRST 120000 NUMBERS



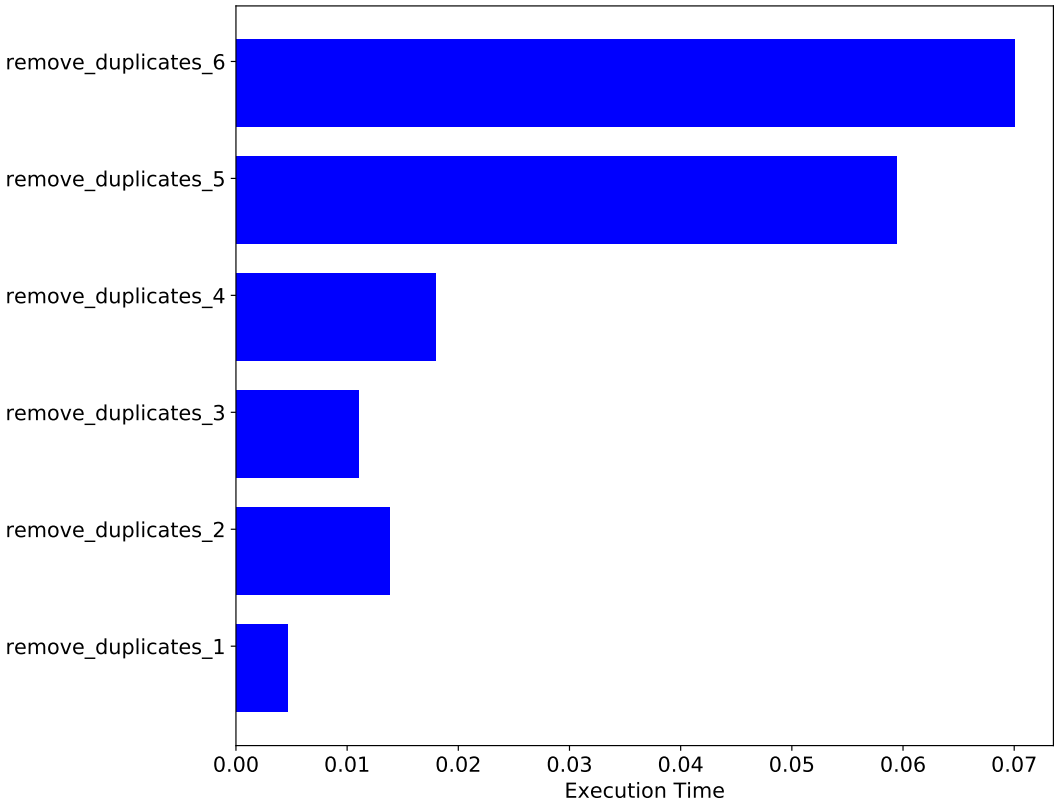


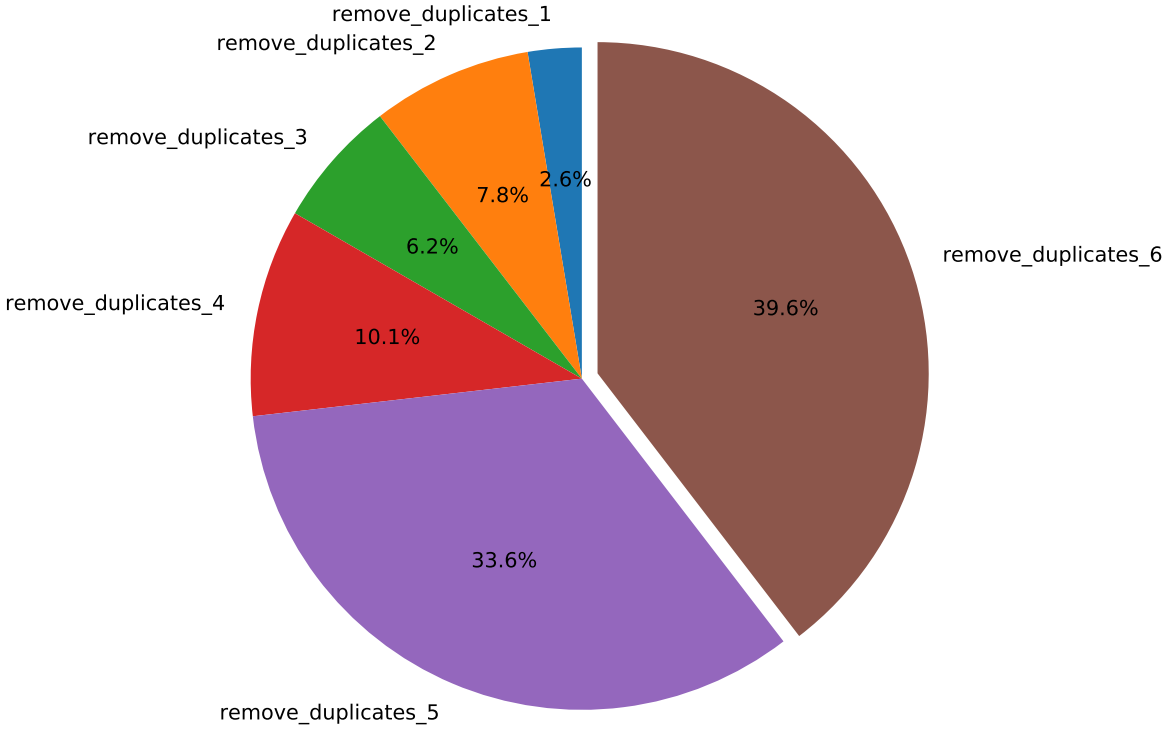
TIMES FOR THE FIRST 122000 NUMBERS



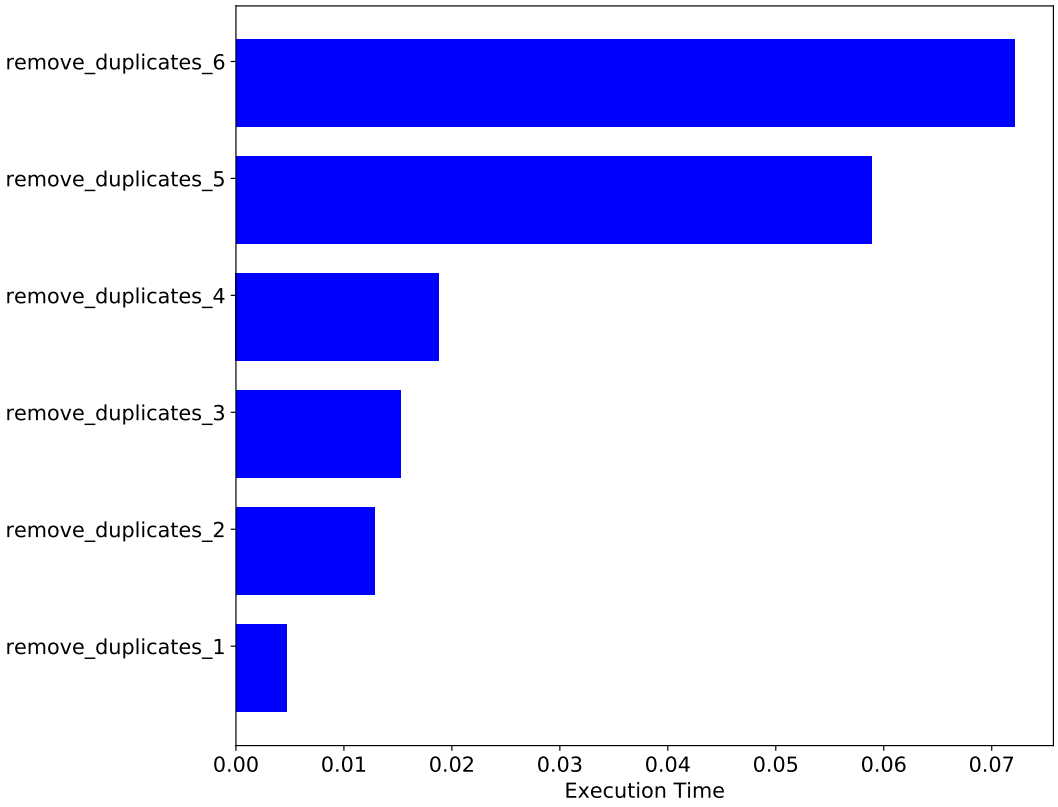


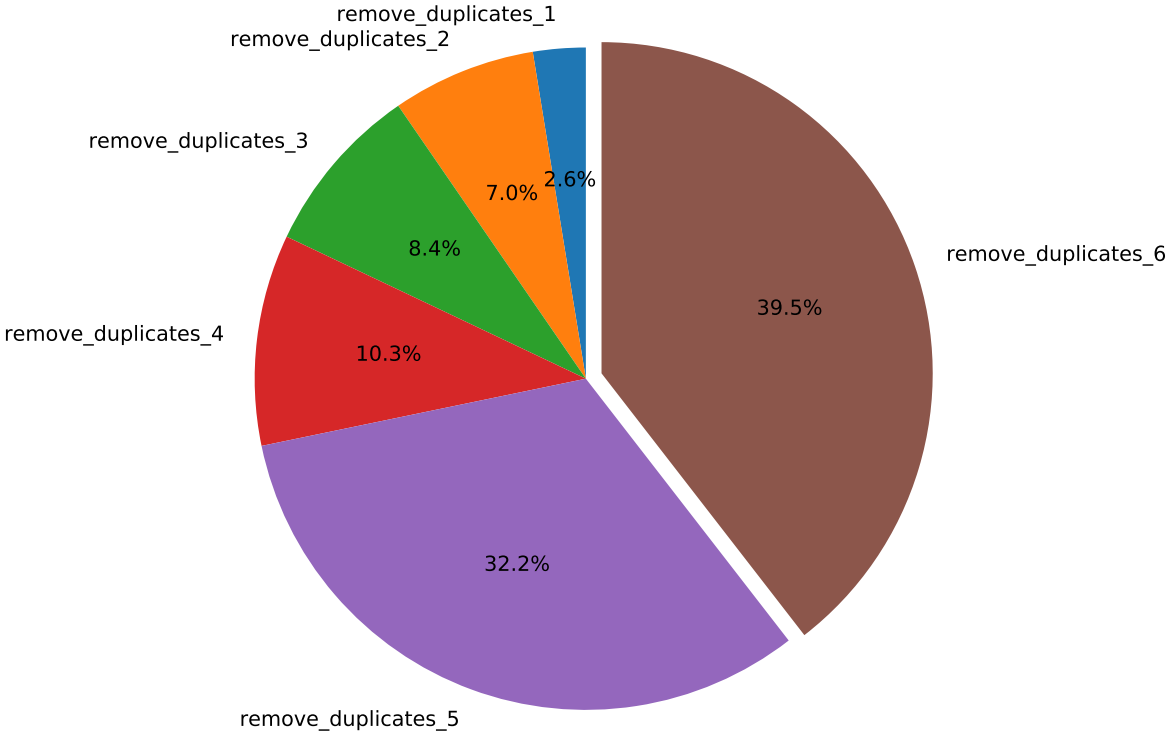
TIMES FOR THE FIRST 124000 NUMBERS



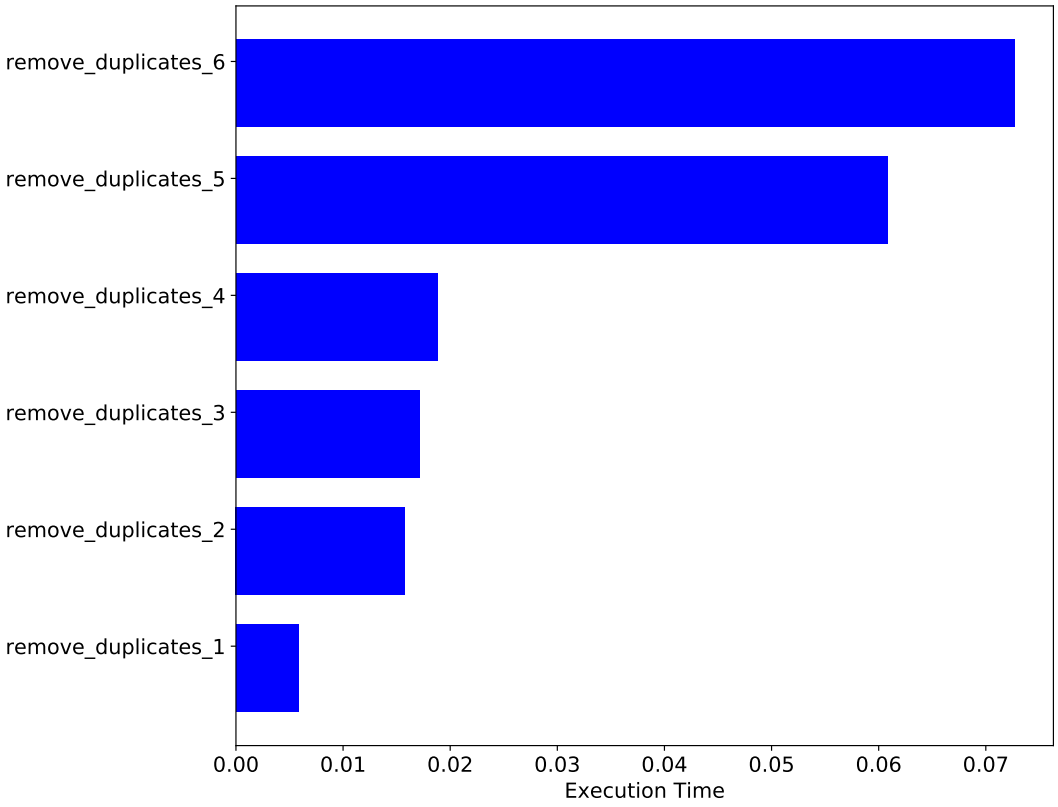


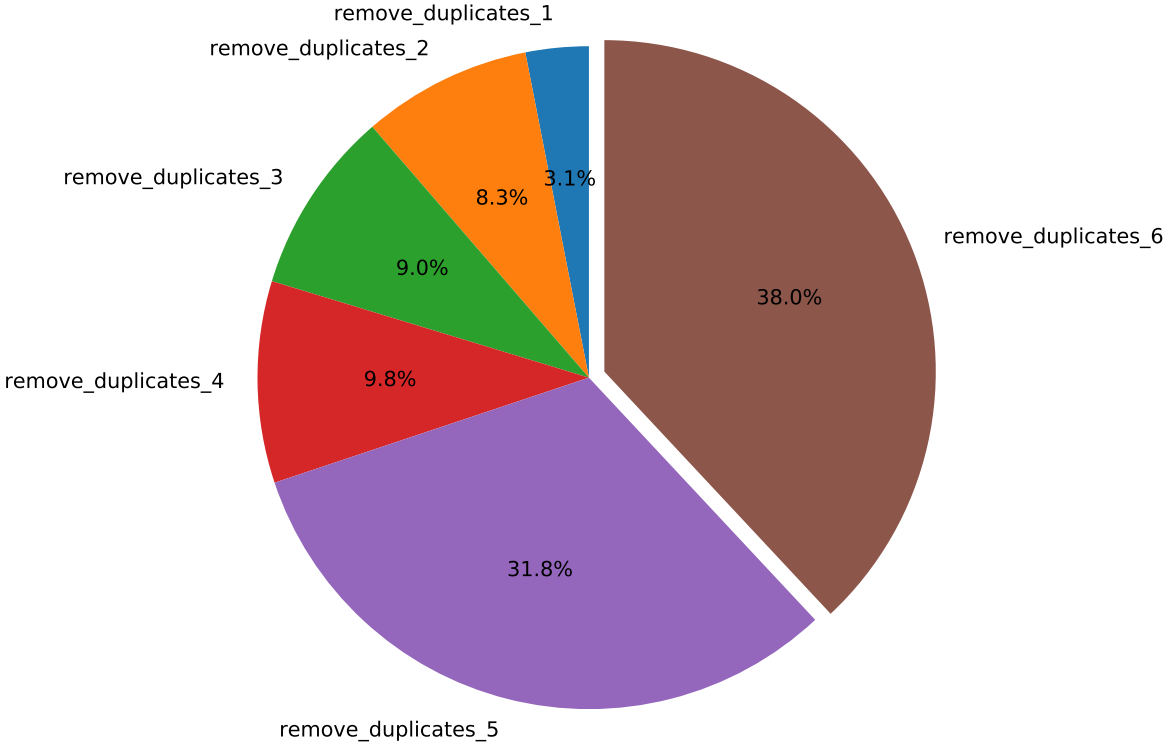
TIMES FOR THE FIRST 126000 NUMBERS



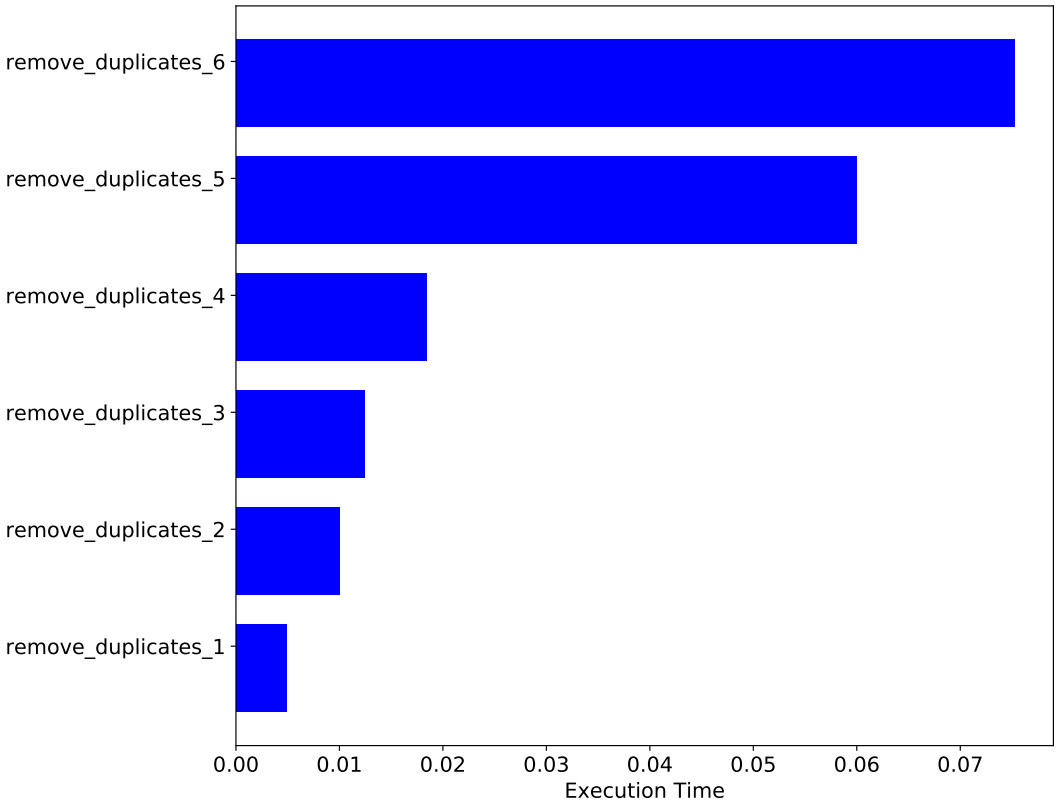


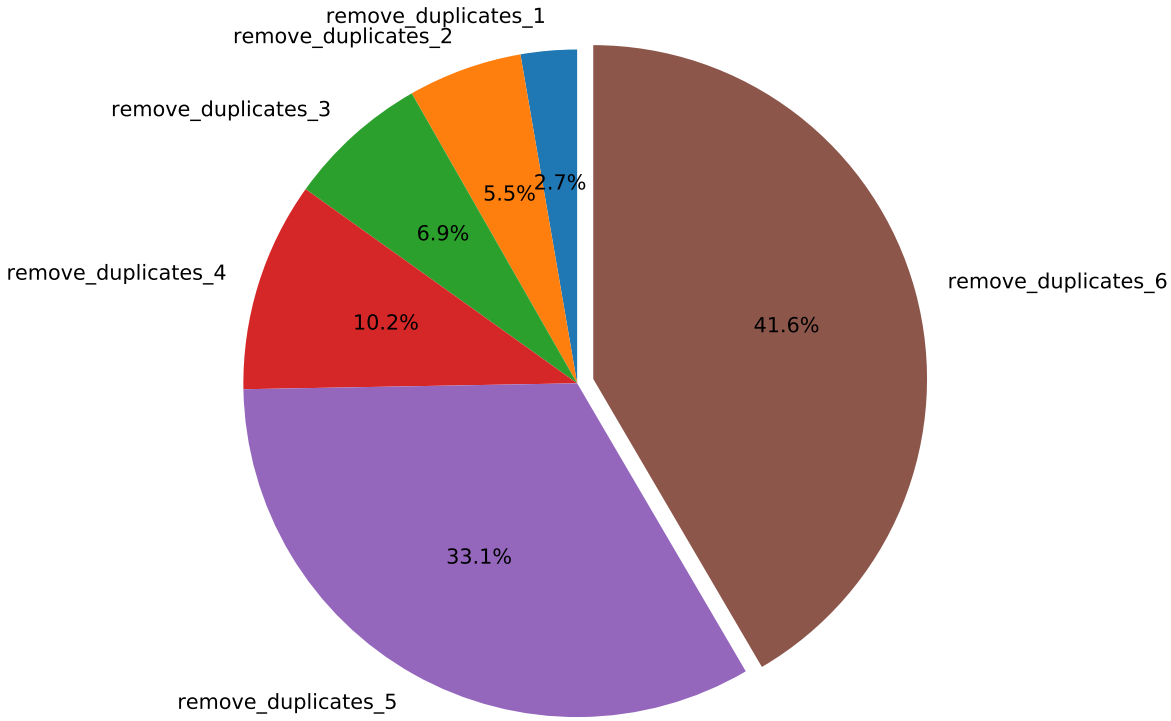
TIMES FOR THE FIRST 128000 NUMBERS



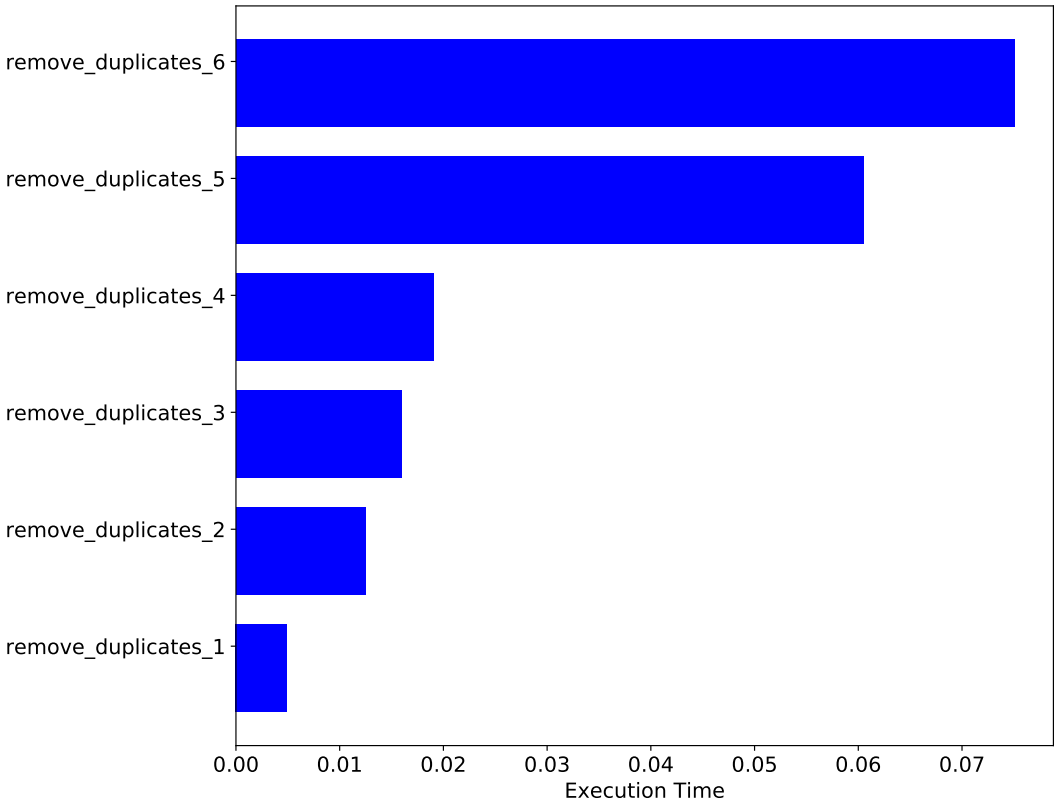


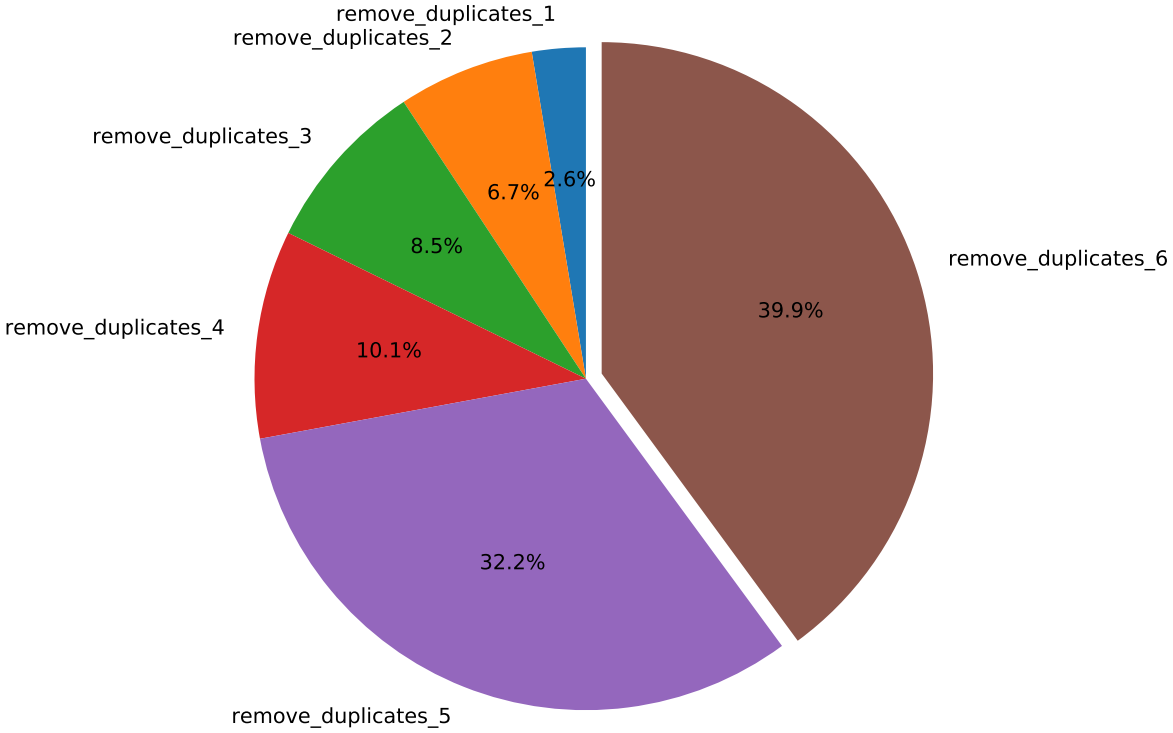
TIMES FOR THE FIRST 130000 NUMBERS



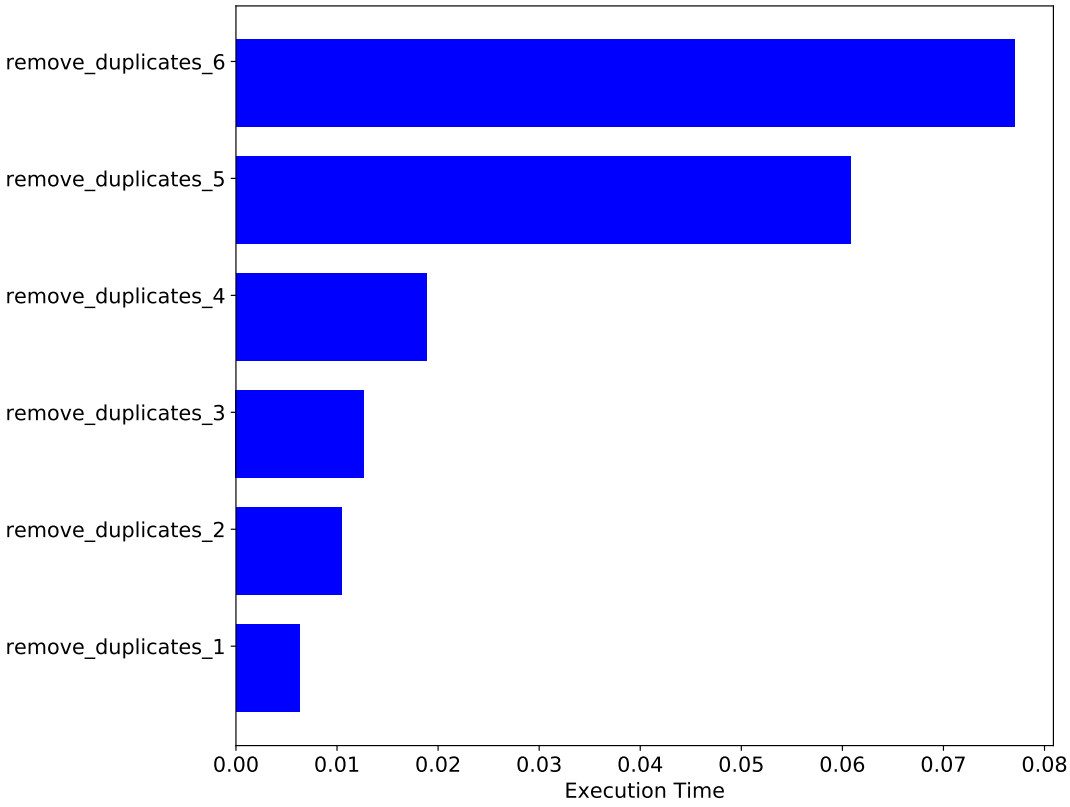


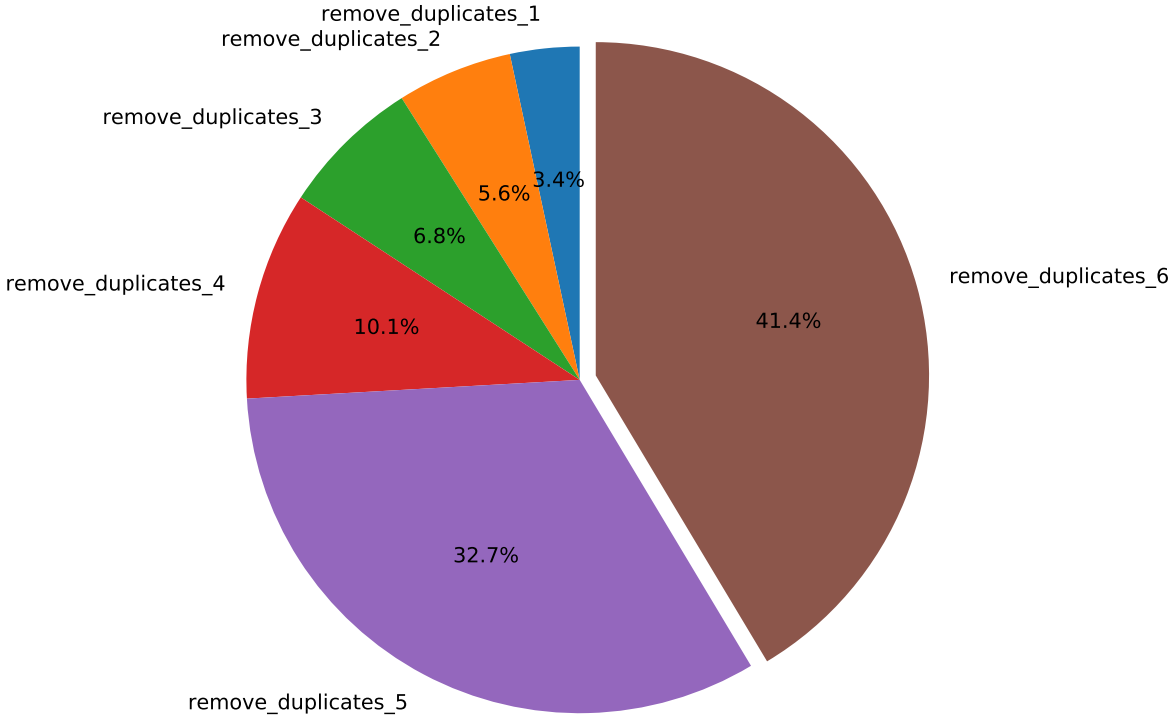
TIMES FOR THE FIRST 132000 NUMBERS



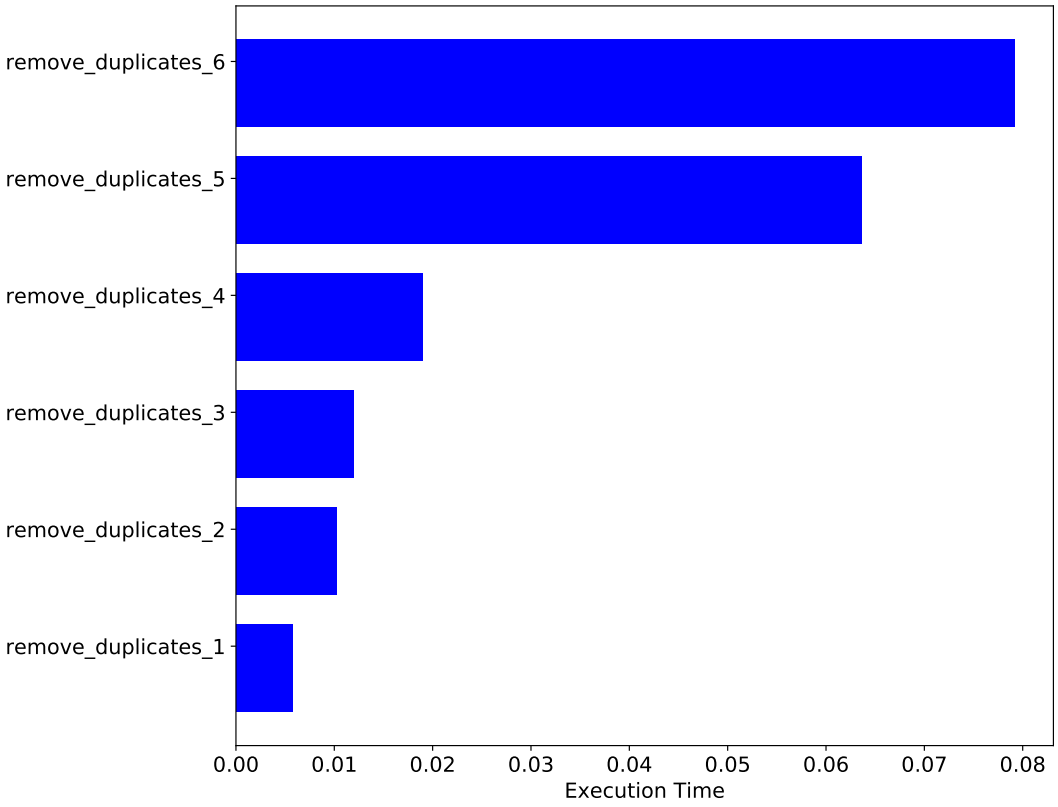


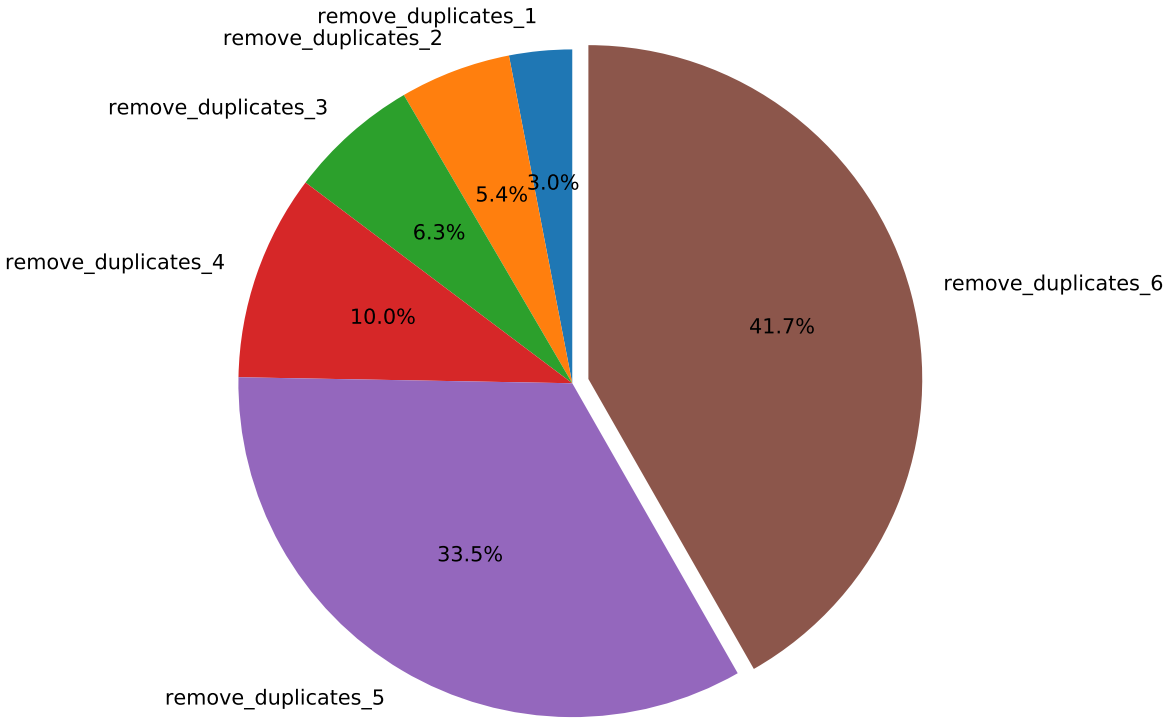
TIMES FOR THE FIRST 134000 NUMBERS



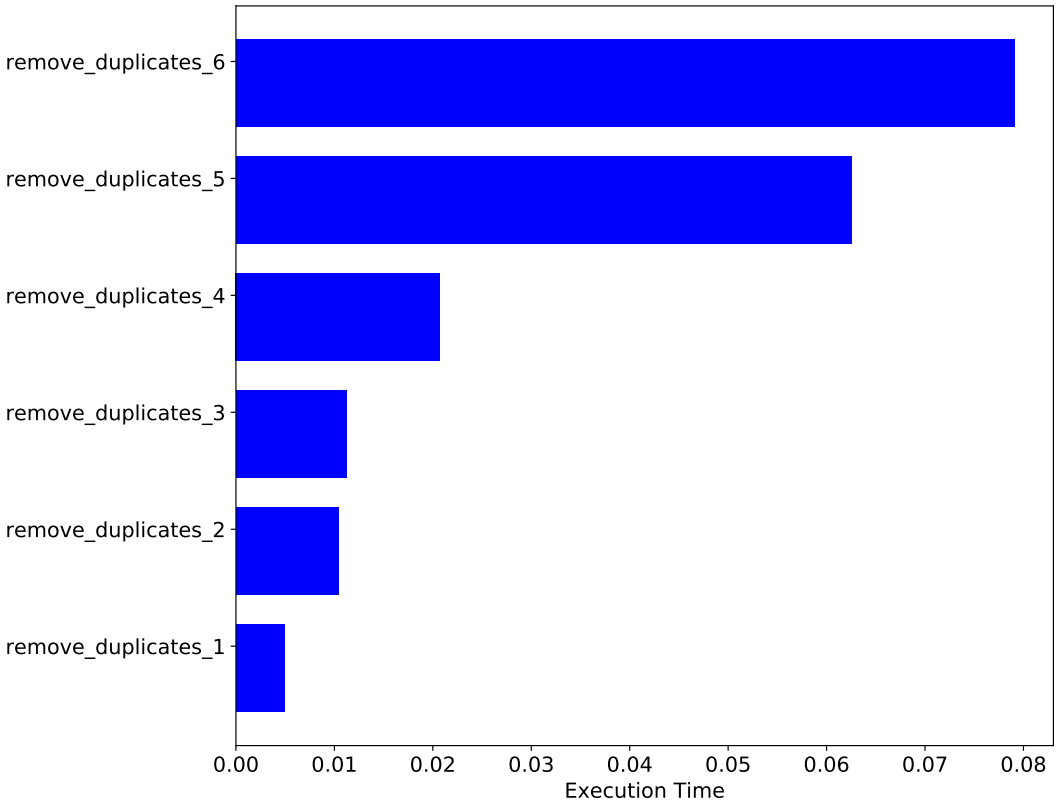


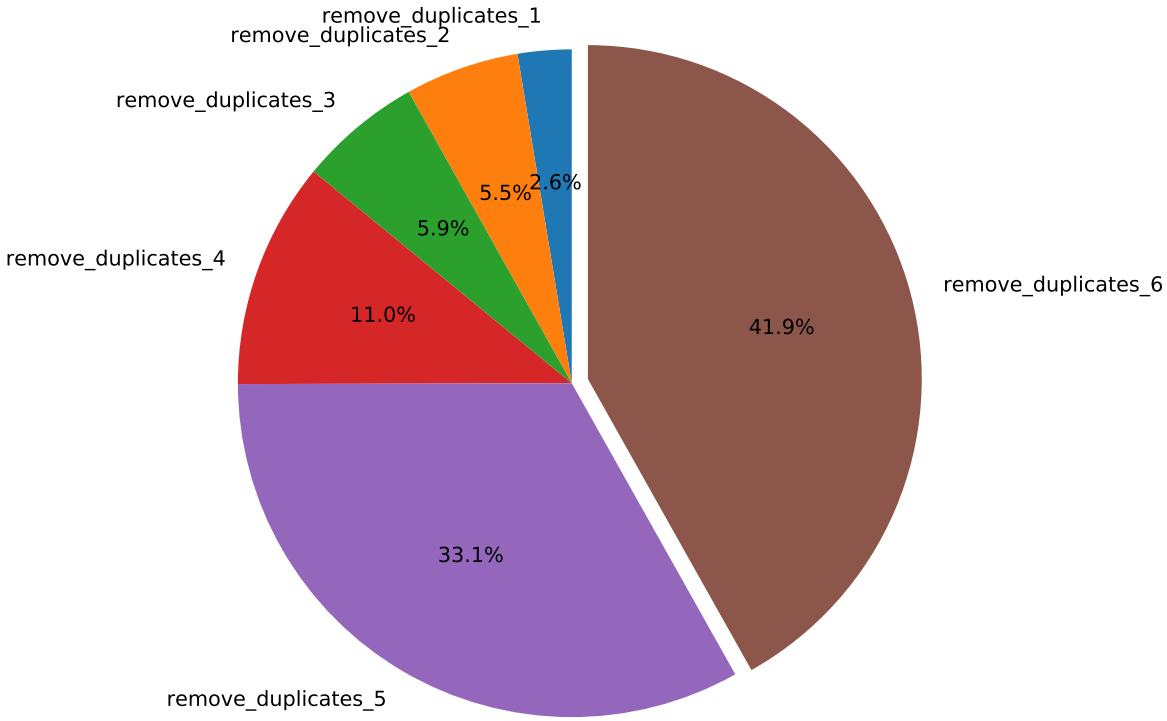
TIMES FOR THE FIRST 136000 NUMBERS



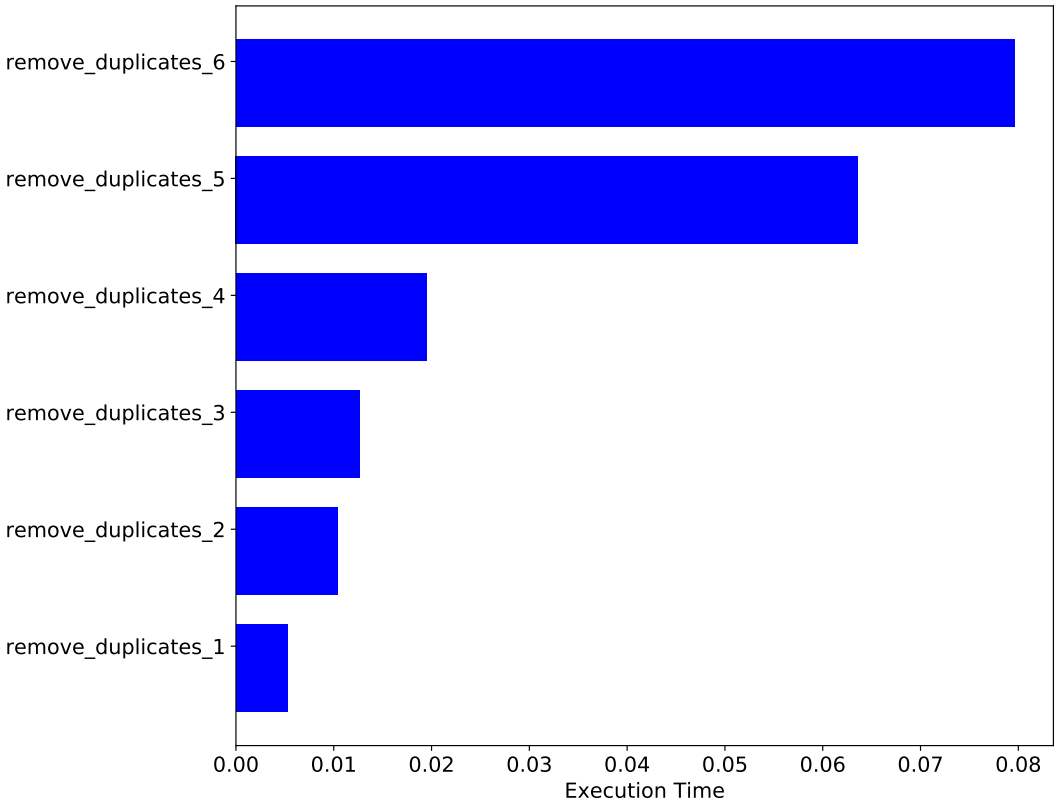


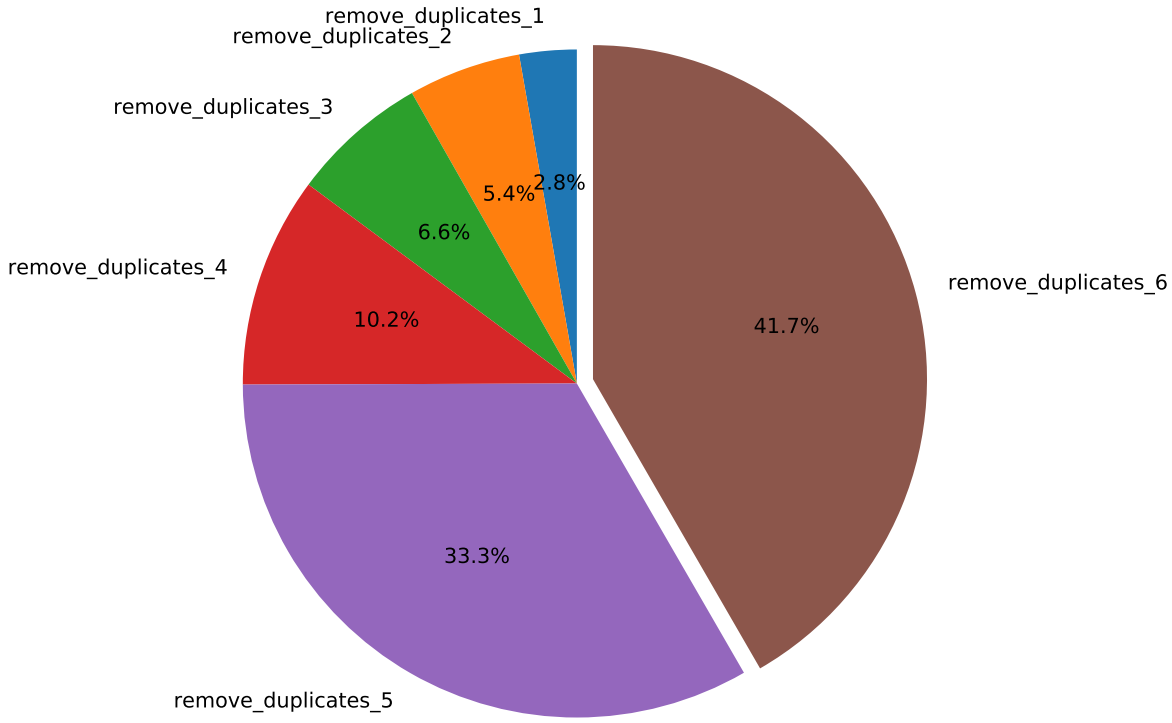
TIMES FOR THE FIRST 138000 NUMBERS



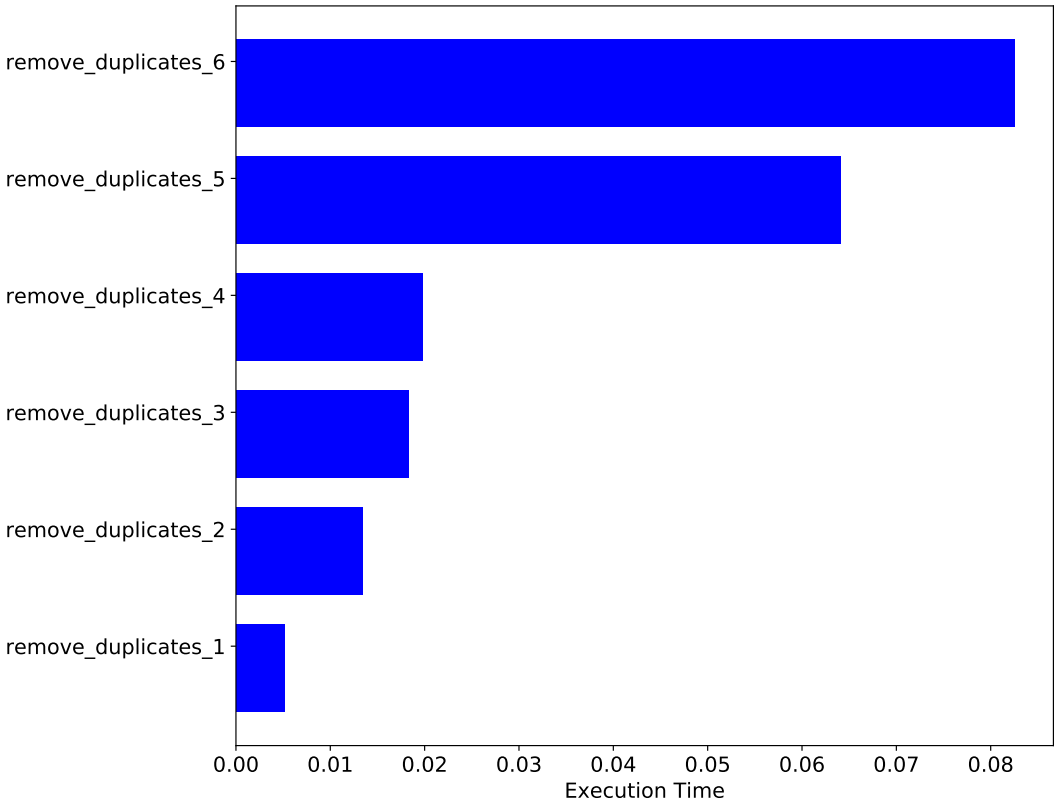


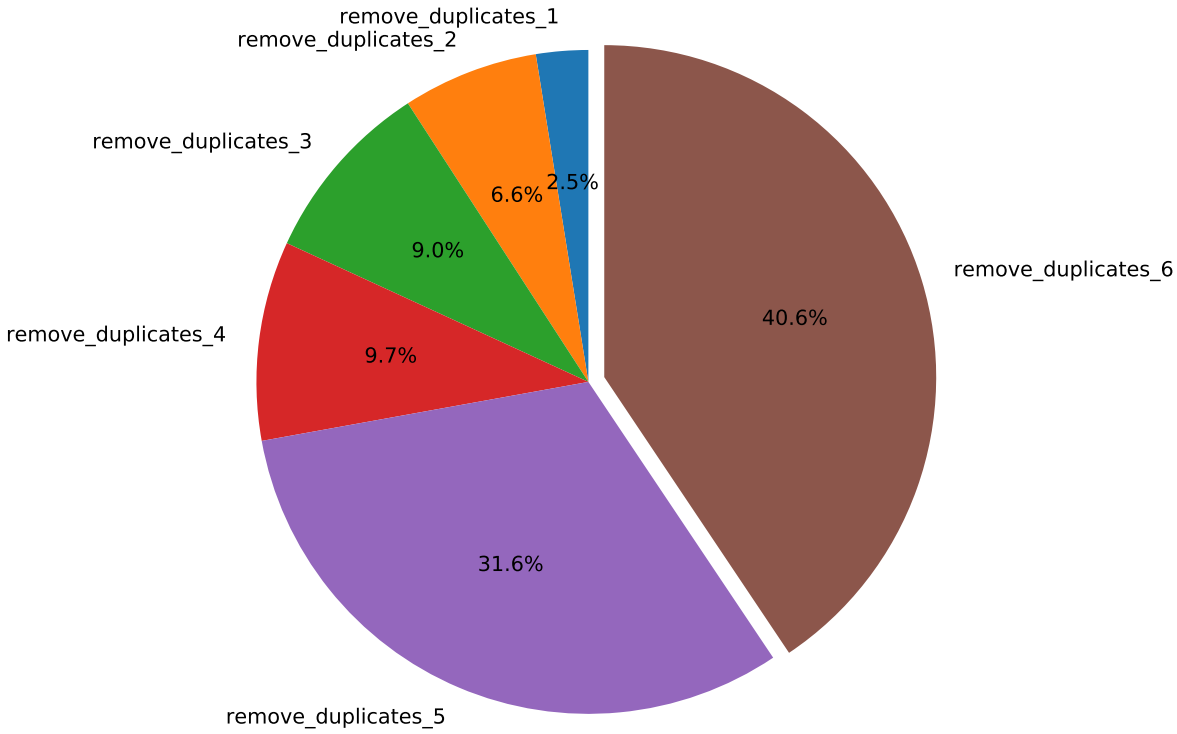
TIMES FOR THE FIRST 140000 NUMBERS



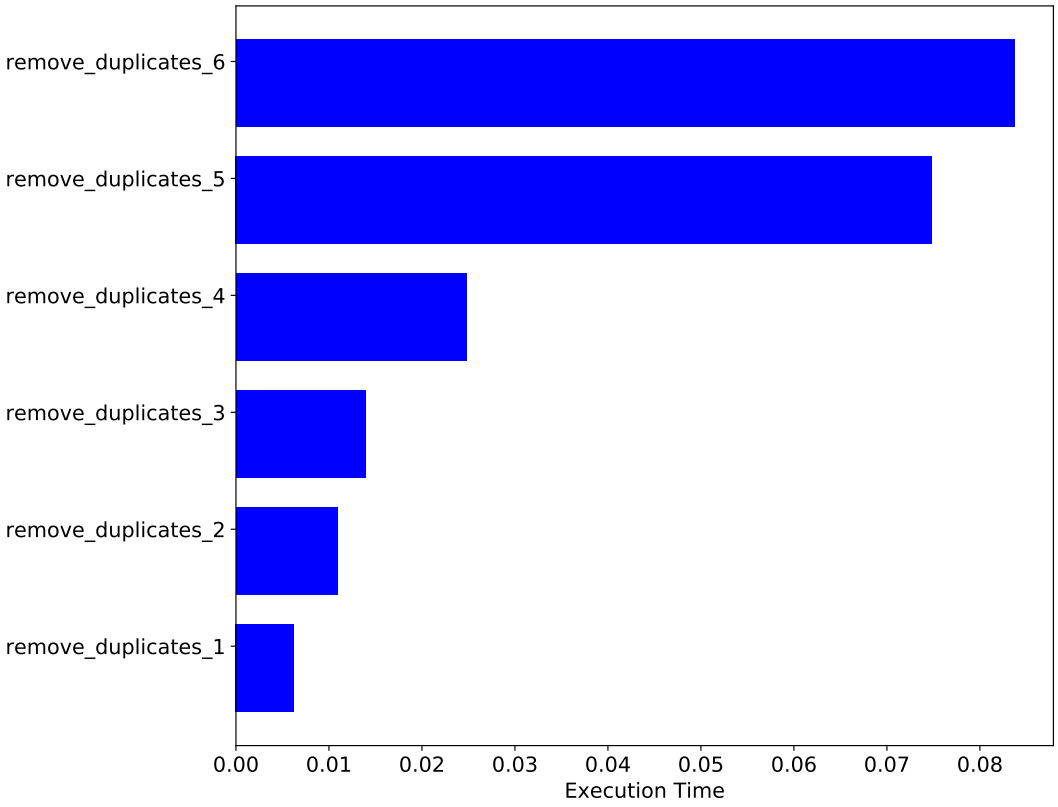


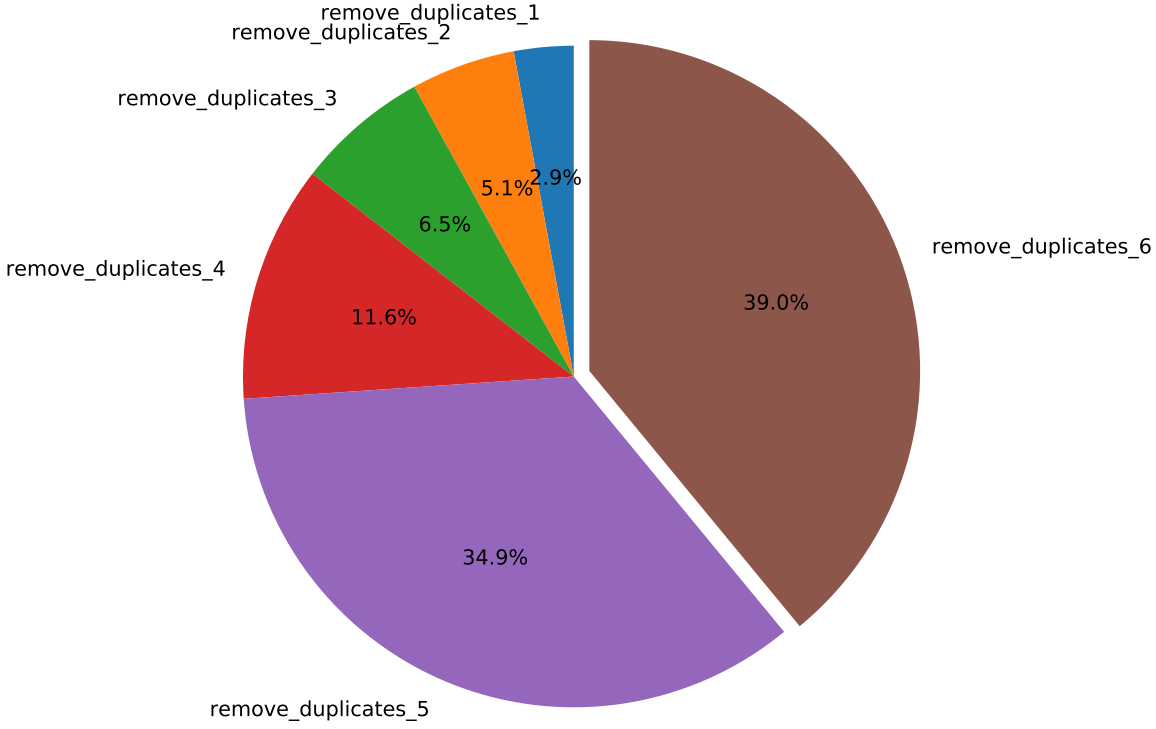
TIMES FOR THE FIRST 142000 NUMBERS



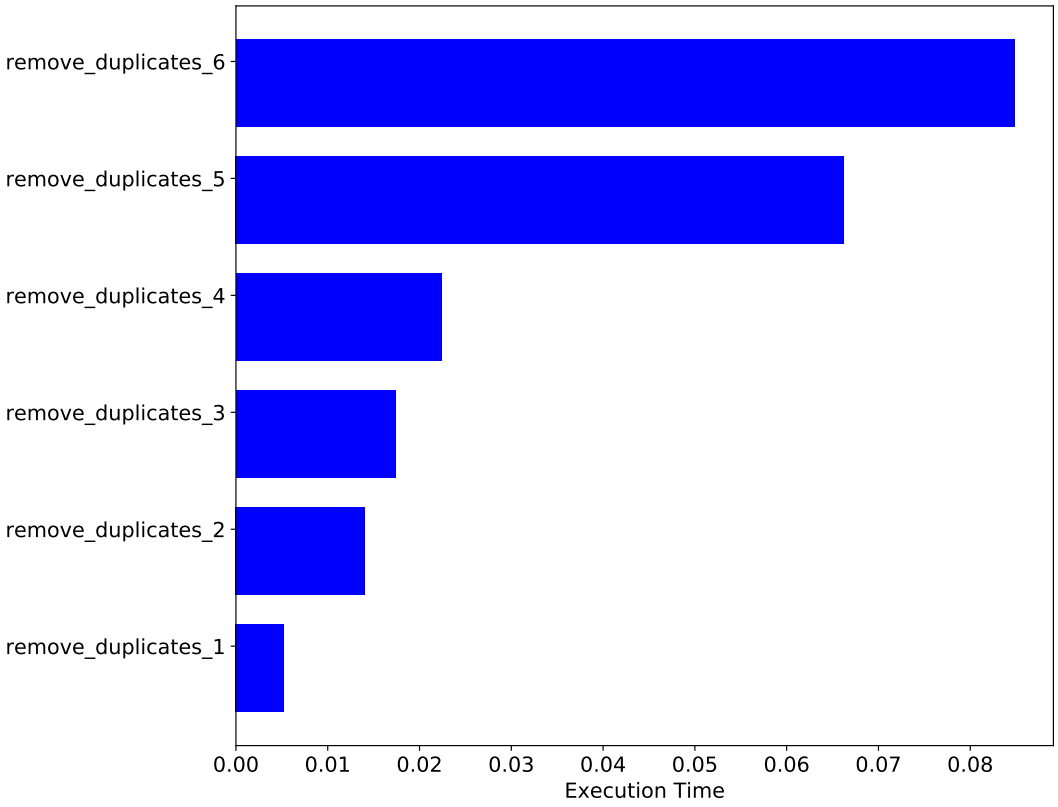


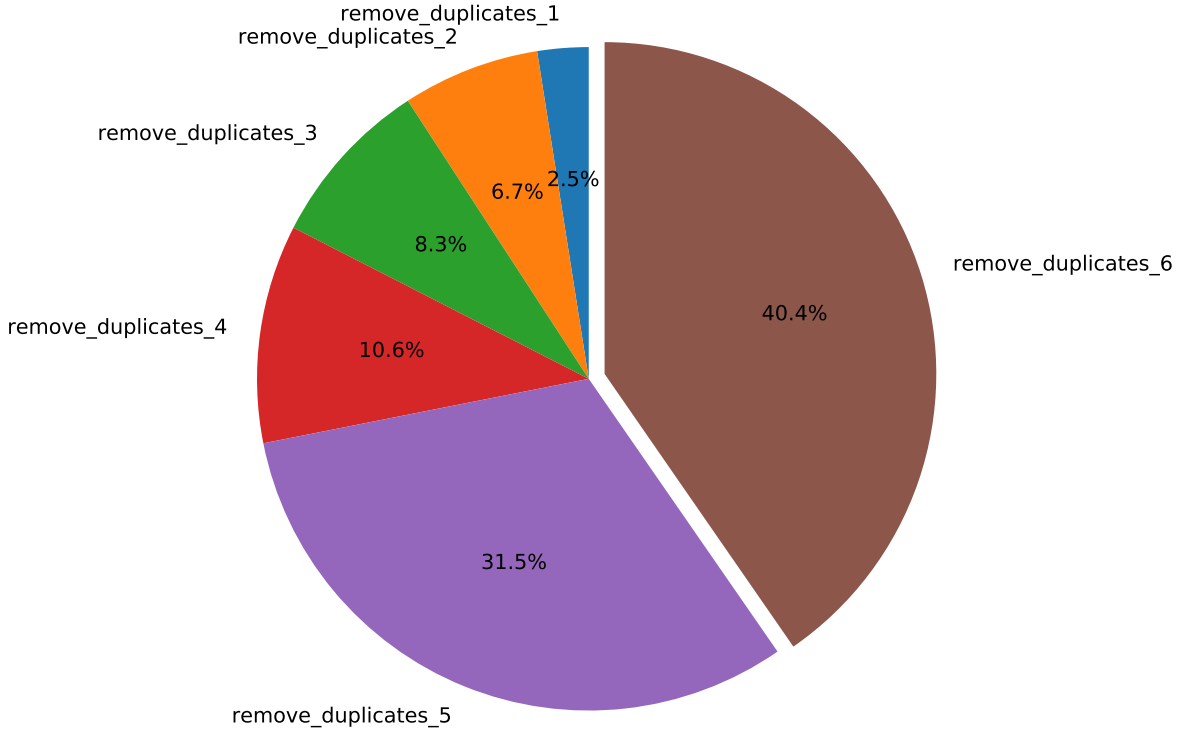
TIMES FOR THE FIRST 144000 NUMBERS



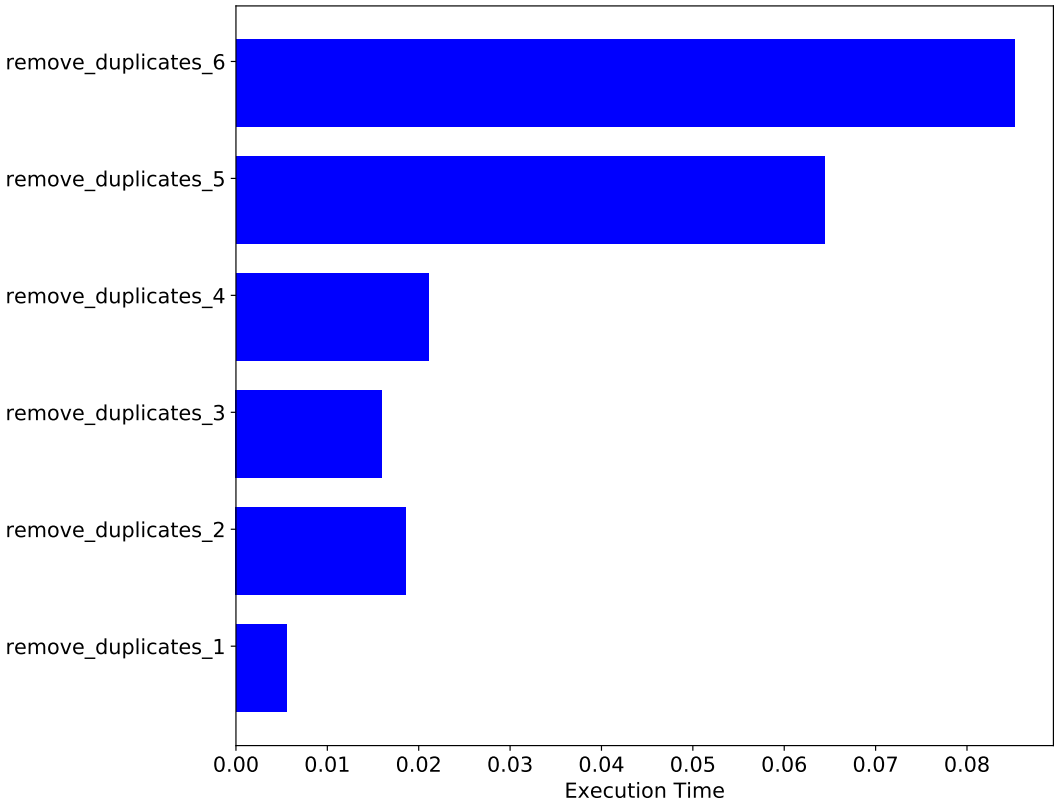


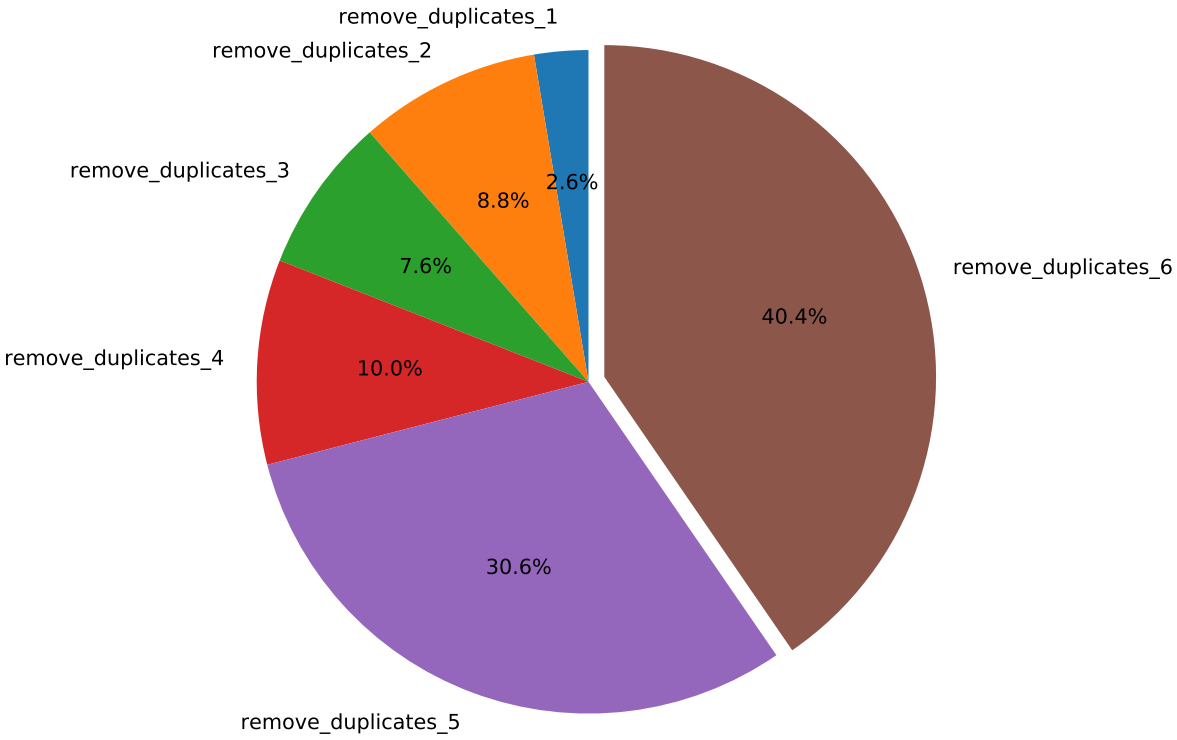
TIMES FOR THE FIRST 146000 NUMBERS



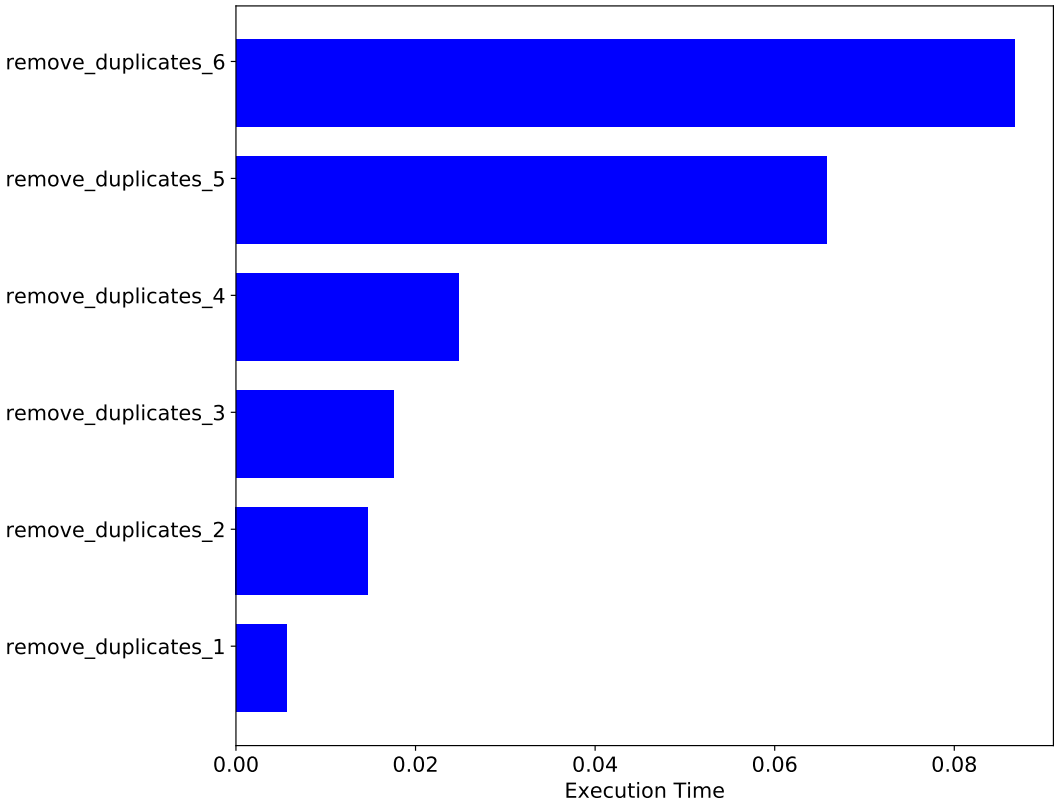


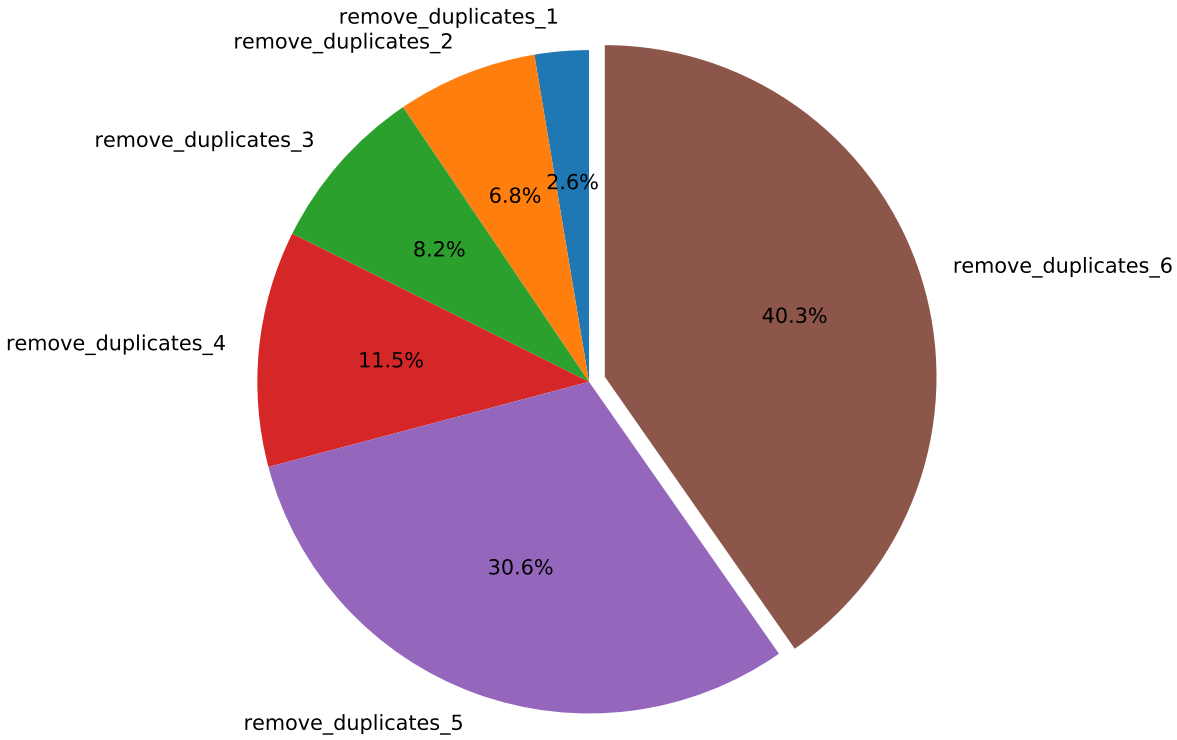
TIMES FOR THE FIRST 148000 NUMBERS



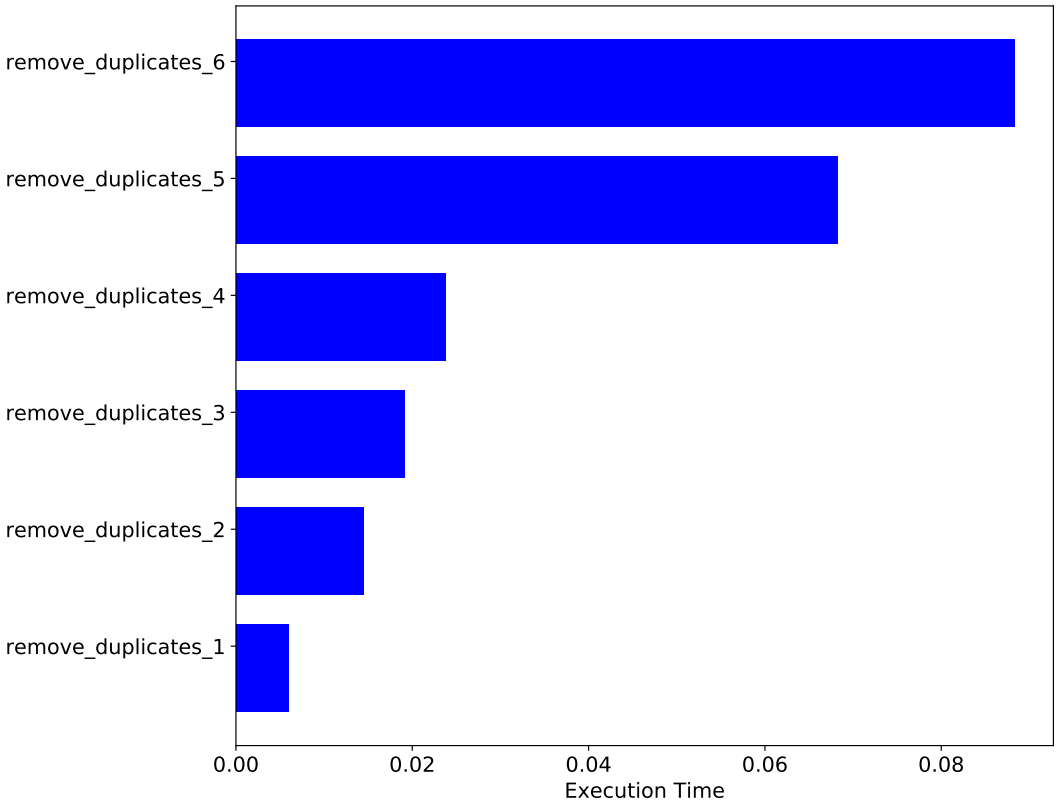


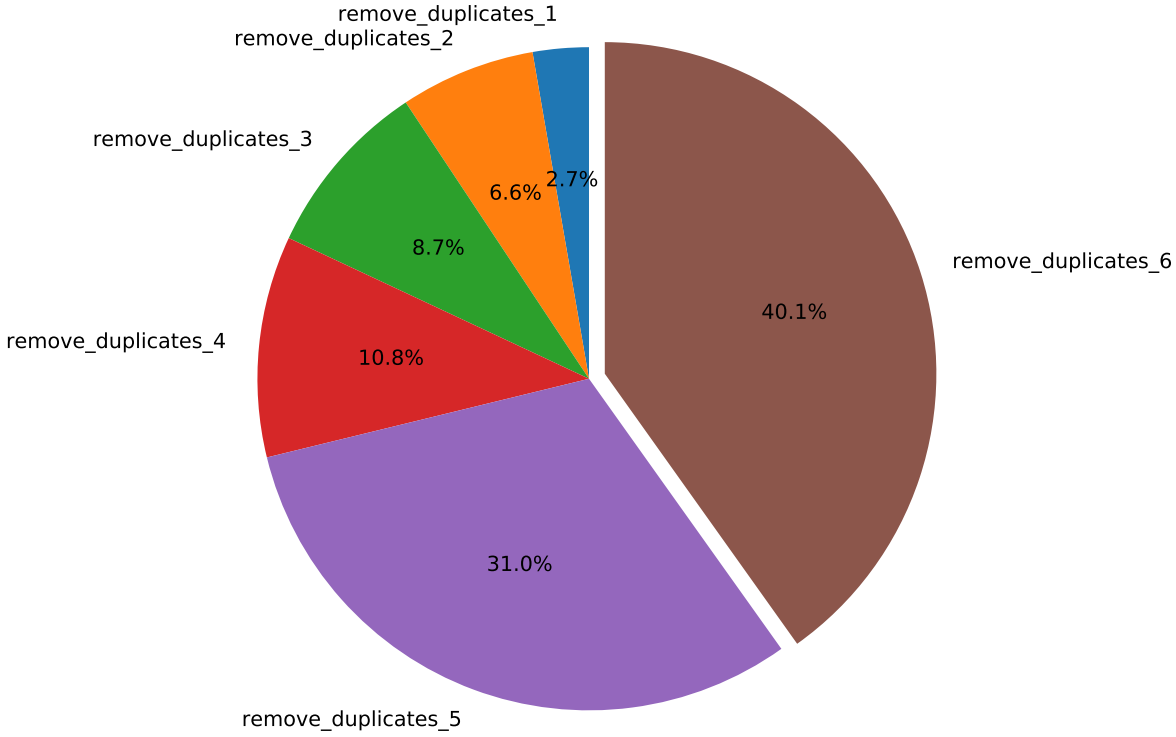
TIMES FOR THE FIRST 150000 NUMBERS



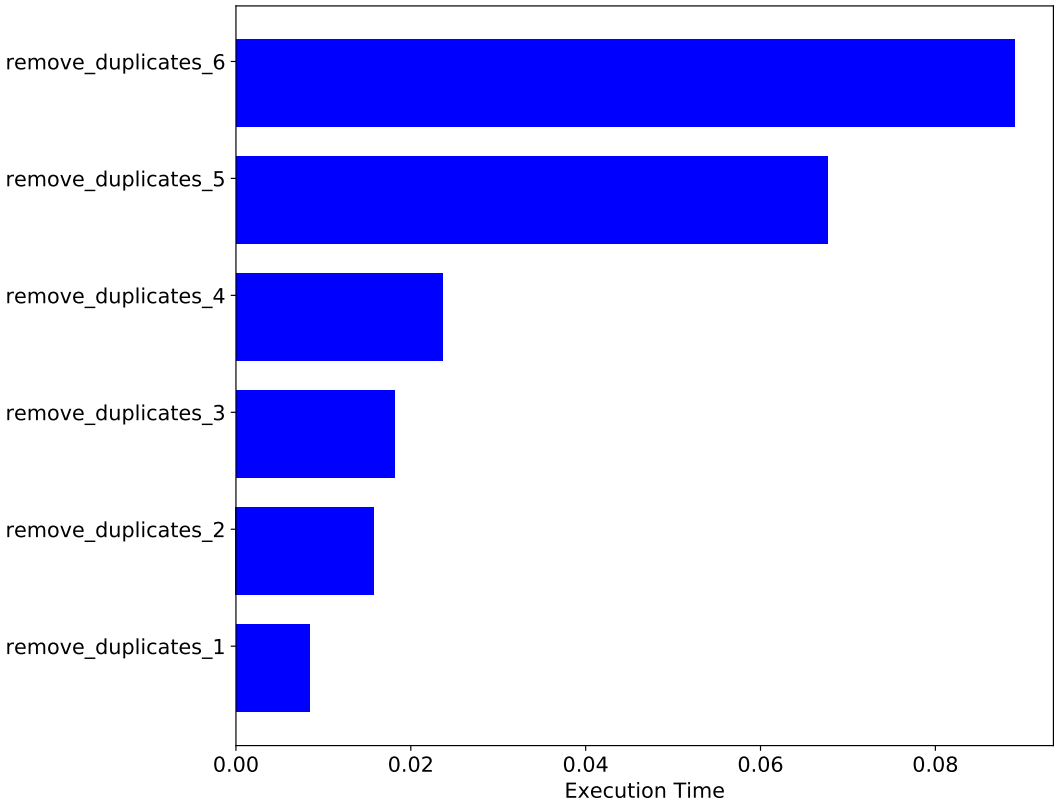


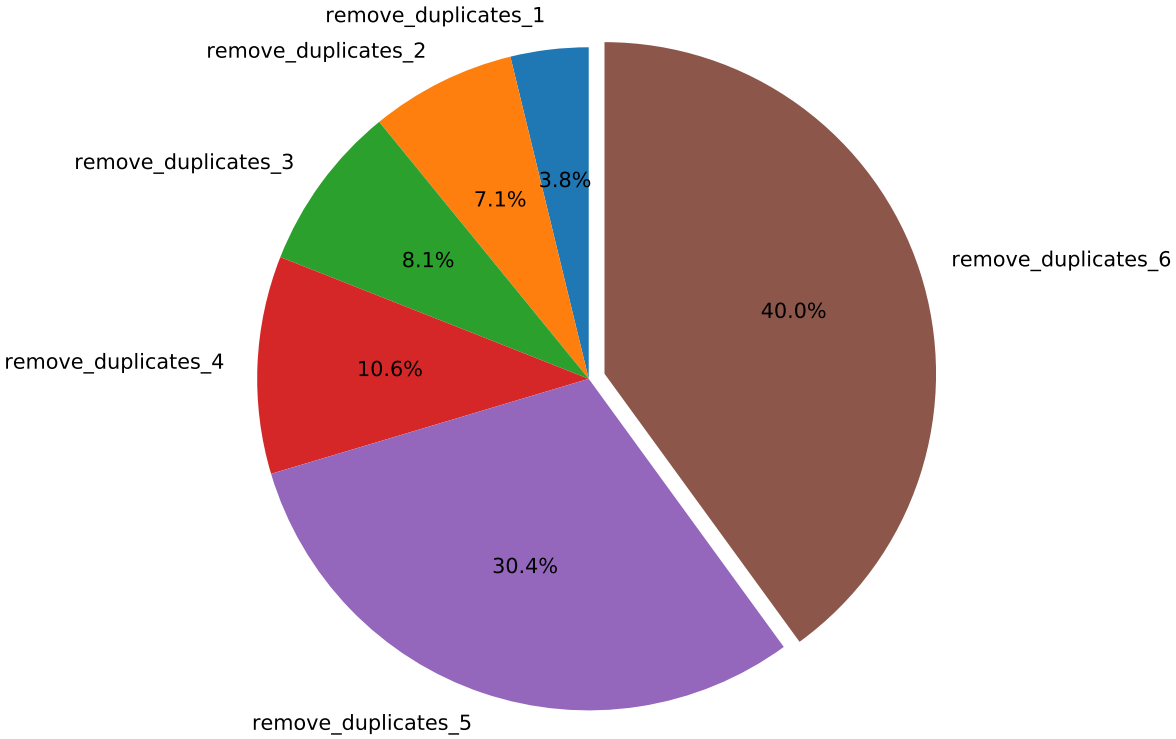
TIMES FOR THE FIRST 152000 NUMBERS



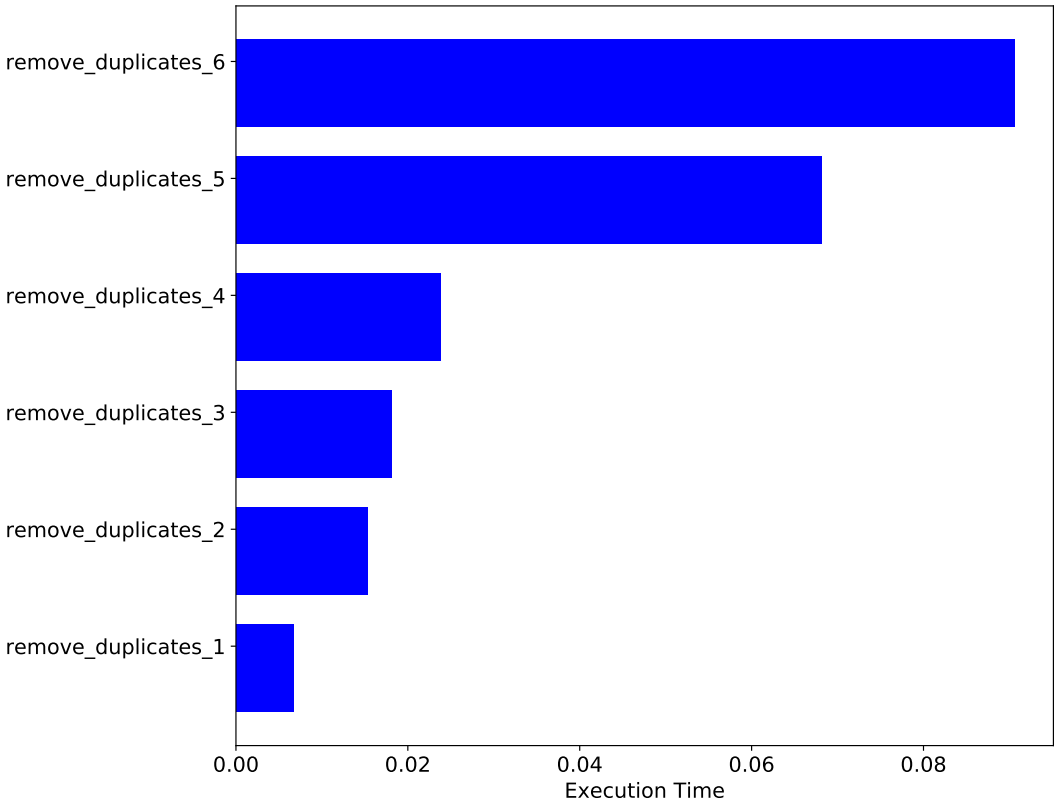


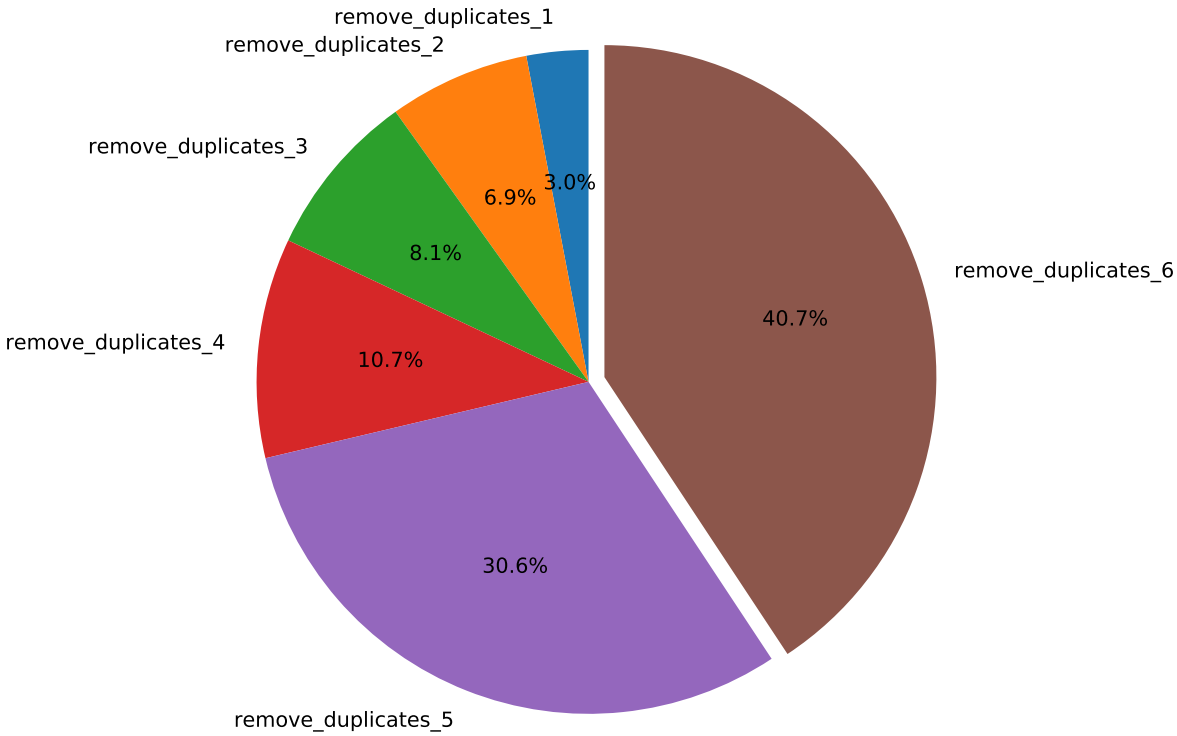
TIMES FOR THE FIRST 154000 NUMBERS



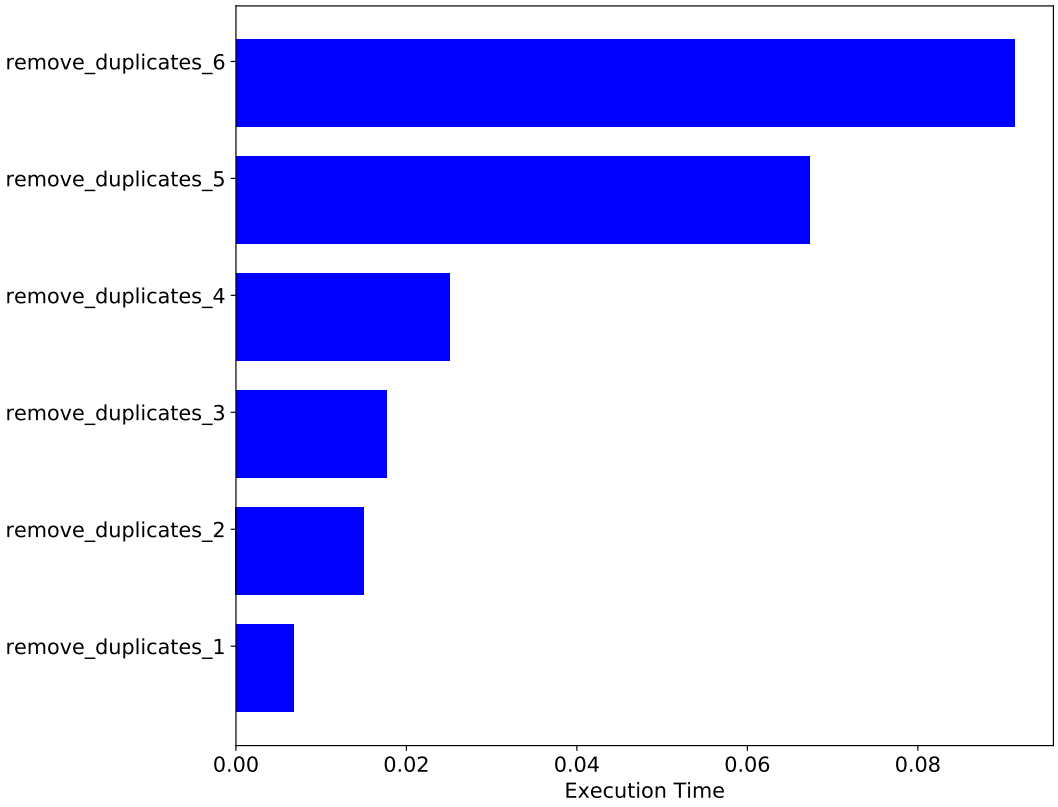


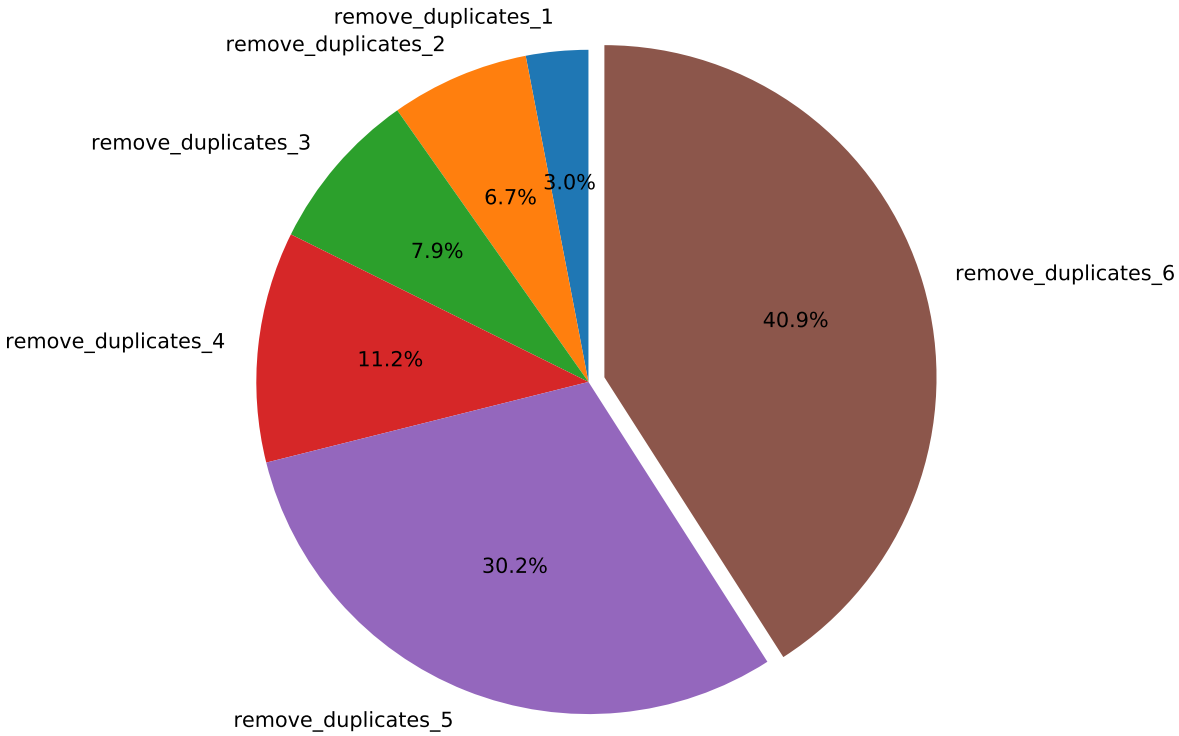
TIMES FOR THE FIRST 156000 NUMBERS



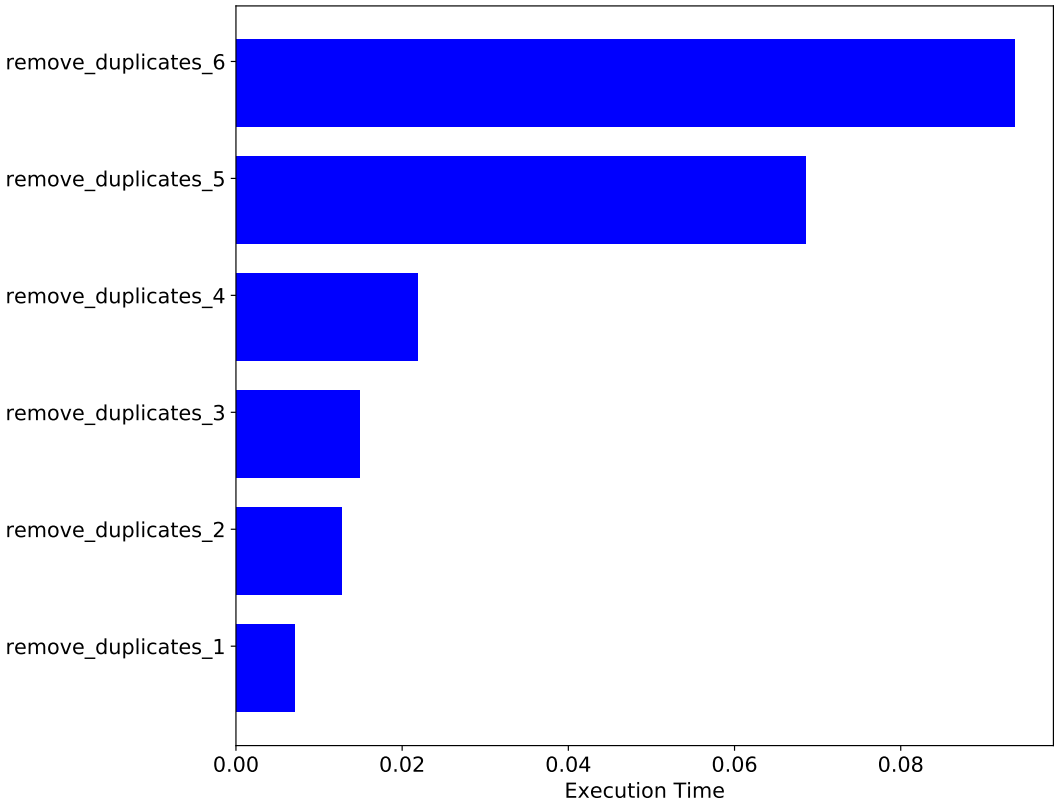


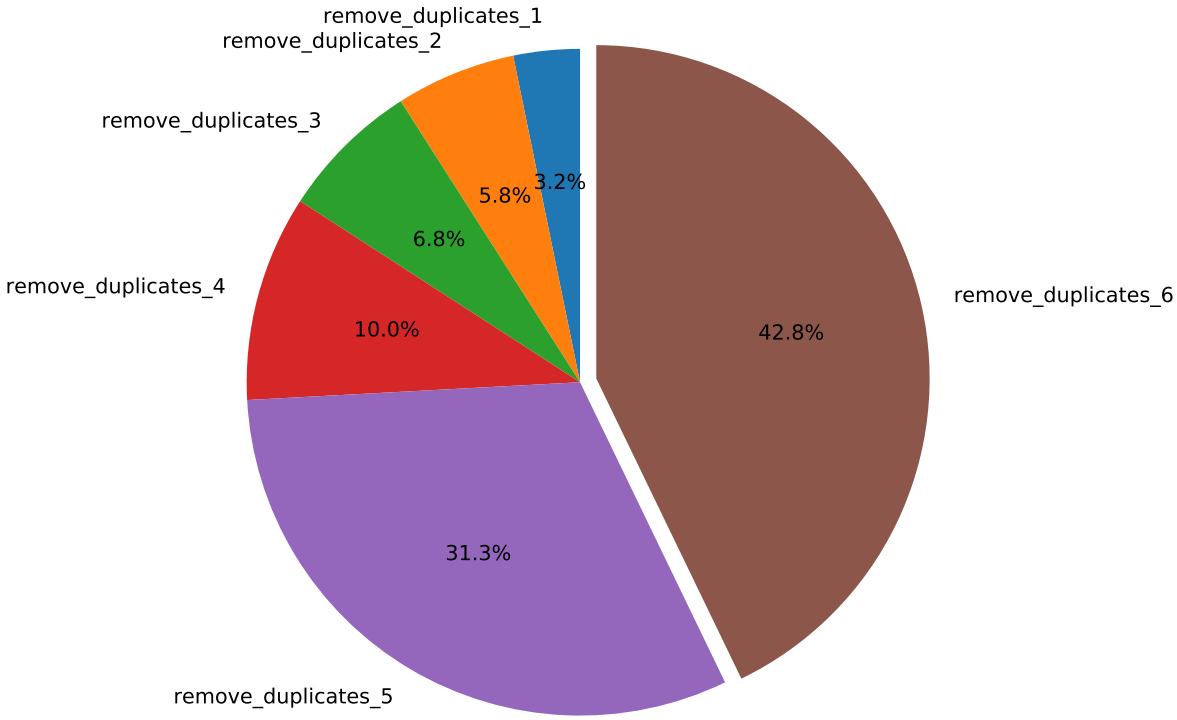
TIMES FOR THE FIRST 158000 NUMBERS



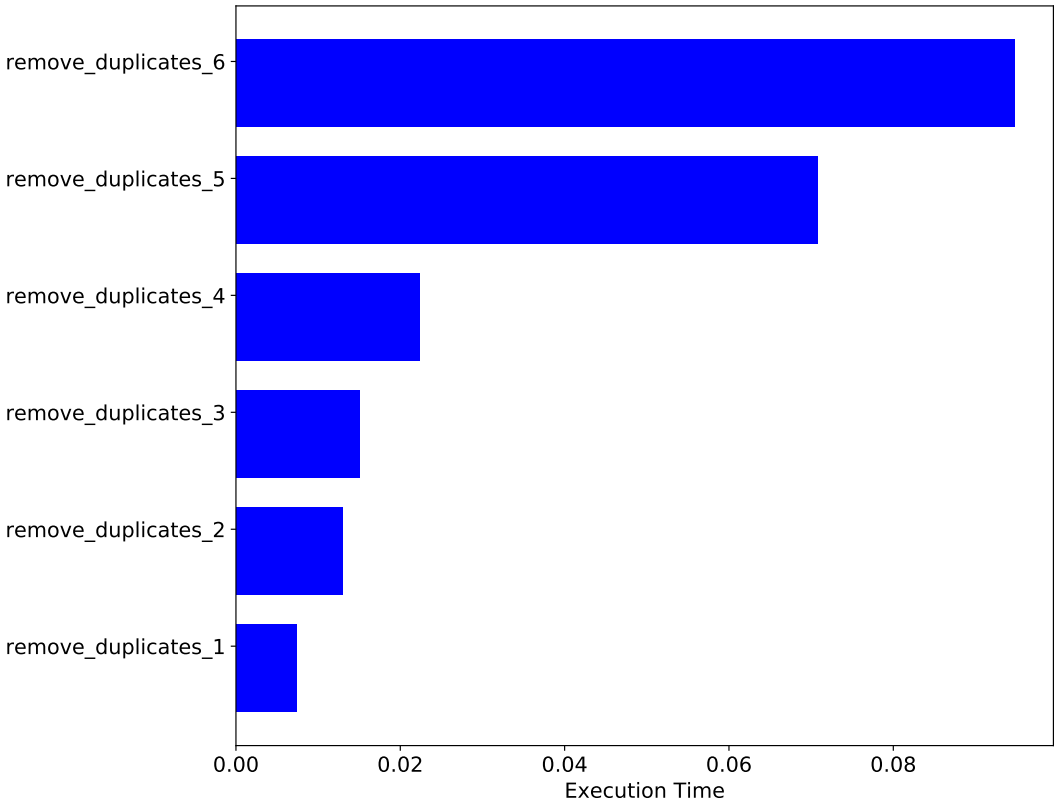


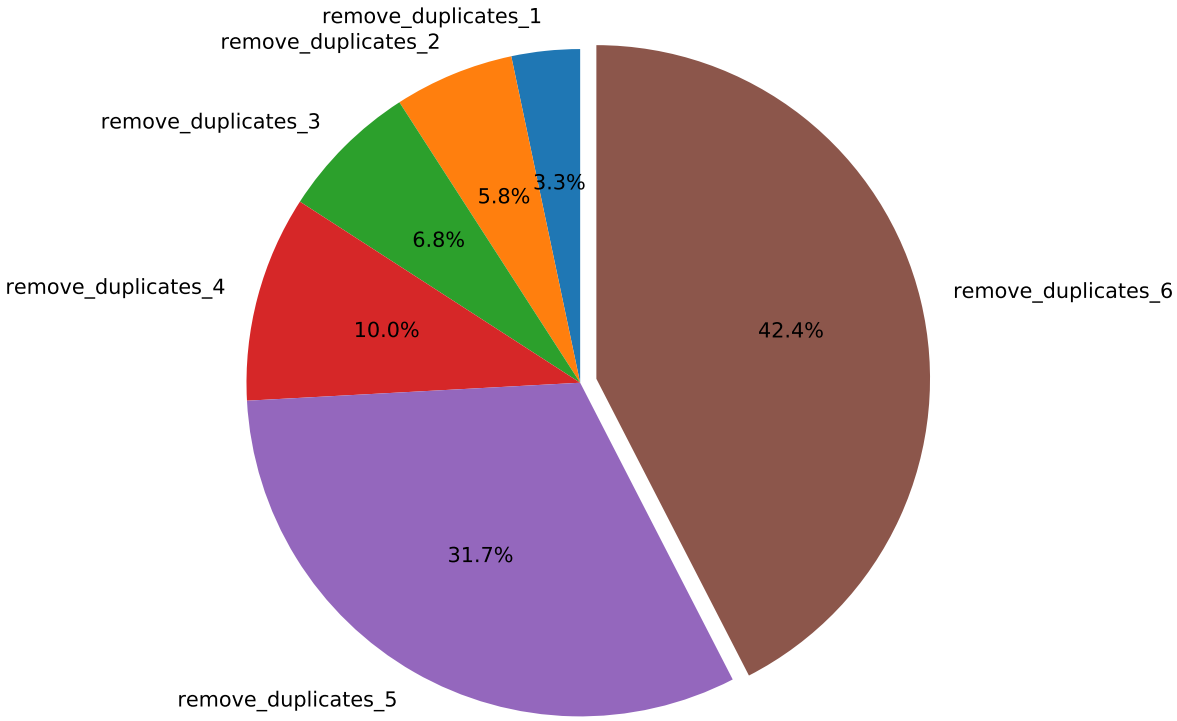
TIMES FOR THE FIRST 160000 NUMBERS



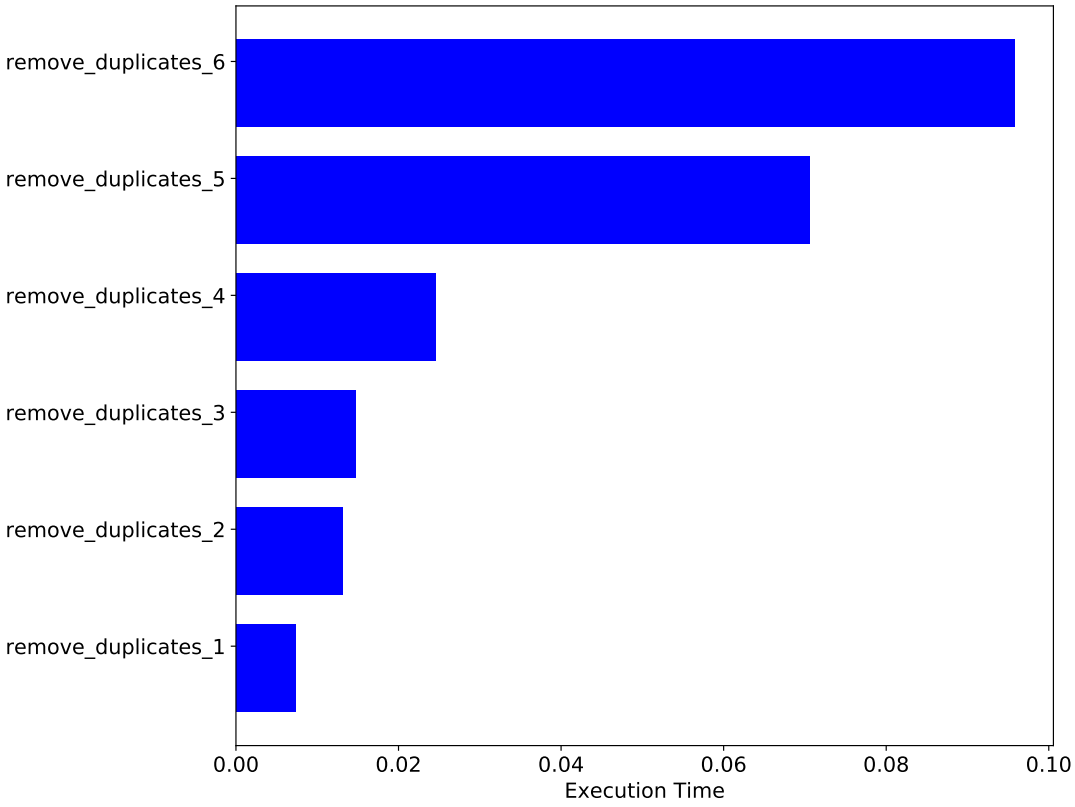


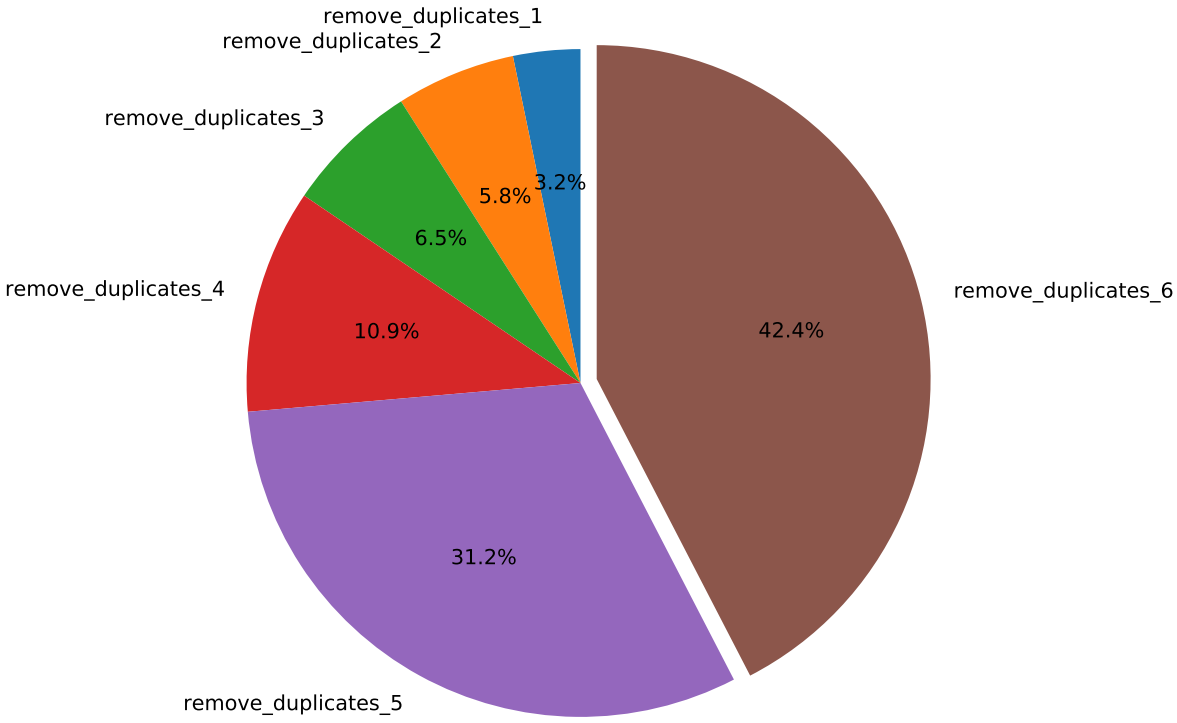
TIMES FOR THE FIRST 162000 NUMBERS



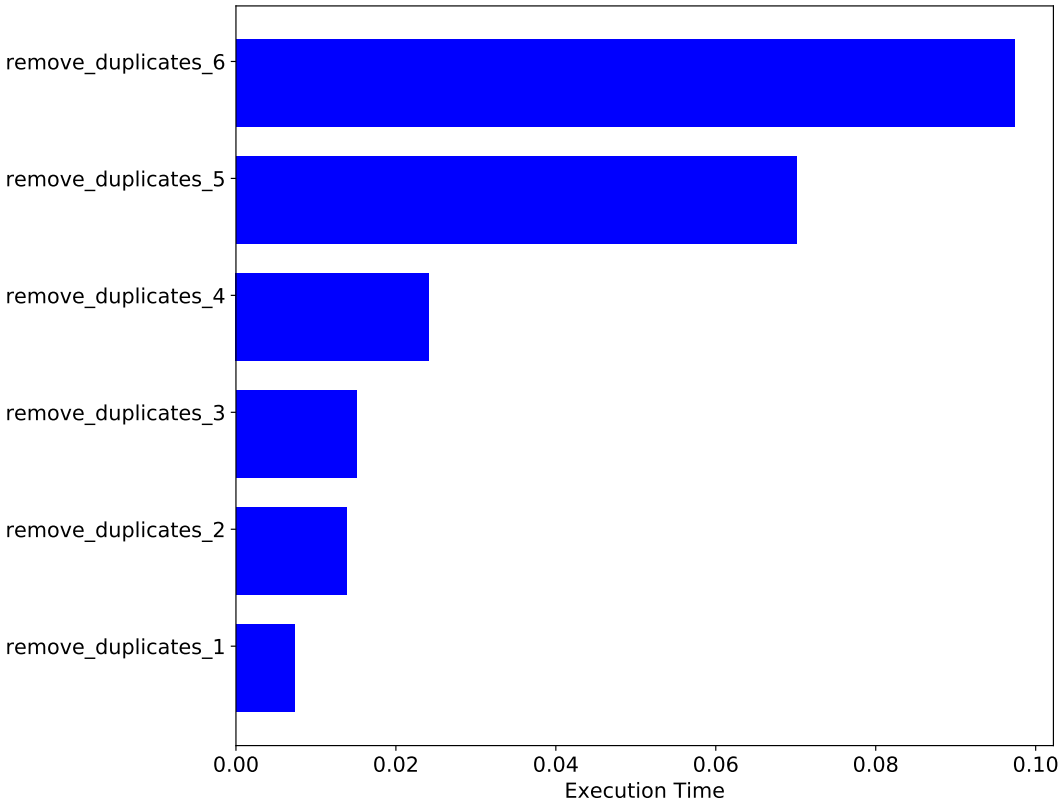


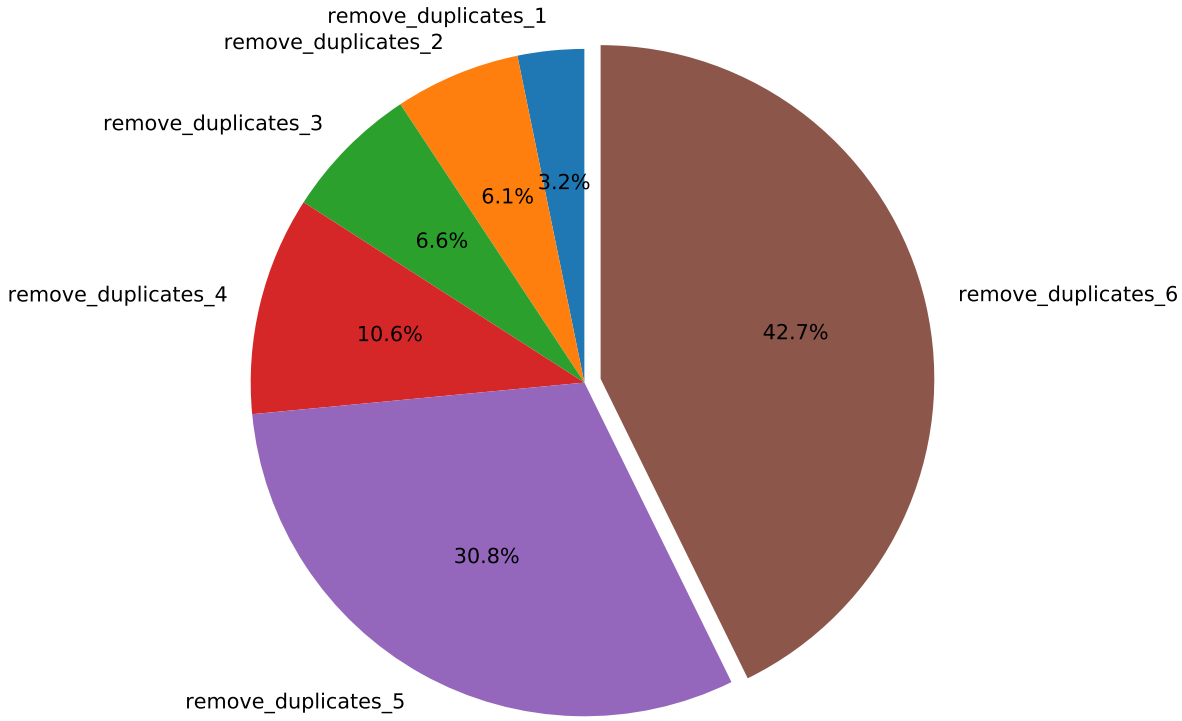
TIMES FOR THE FIRST 164000 NUMBERS



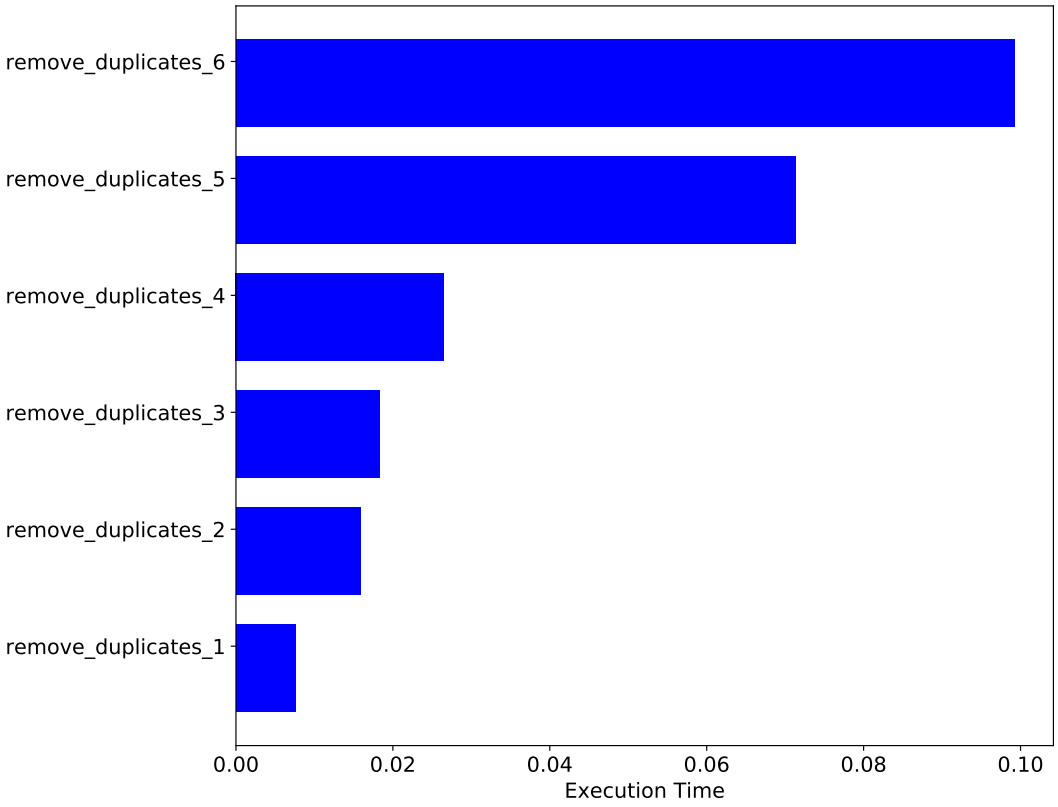


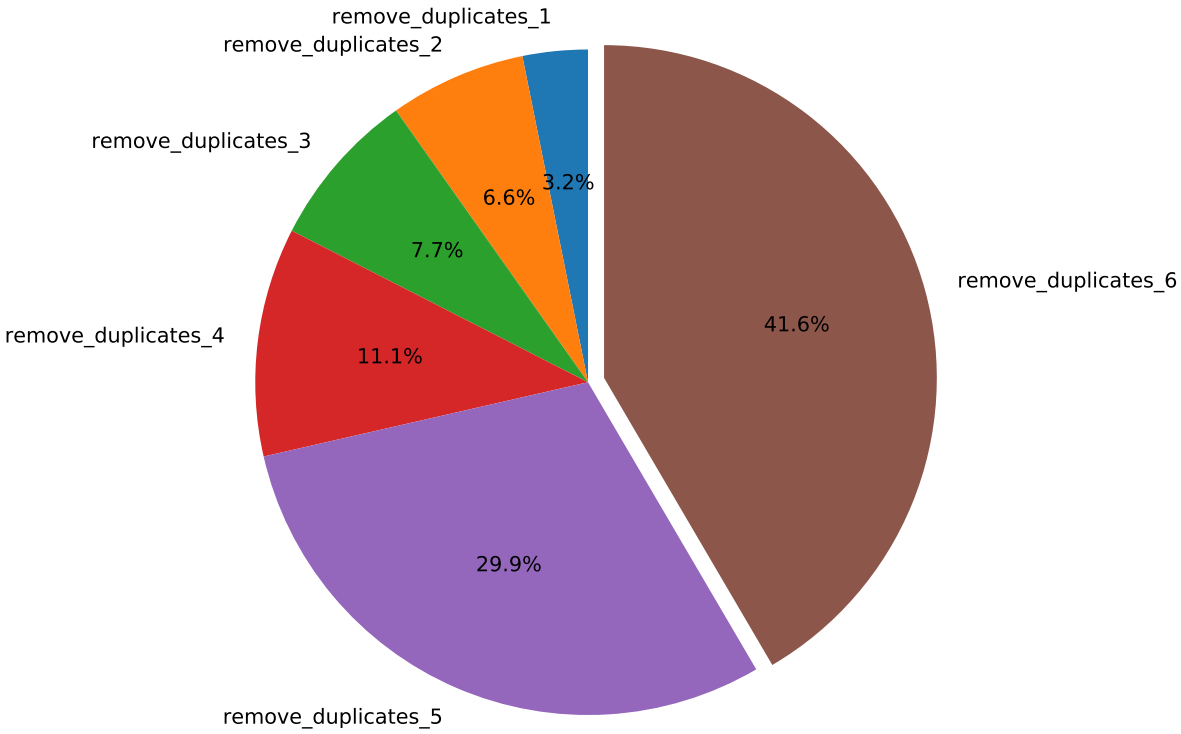
TIMES FOR THE FIRST 166000 NUMBERS



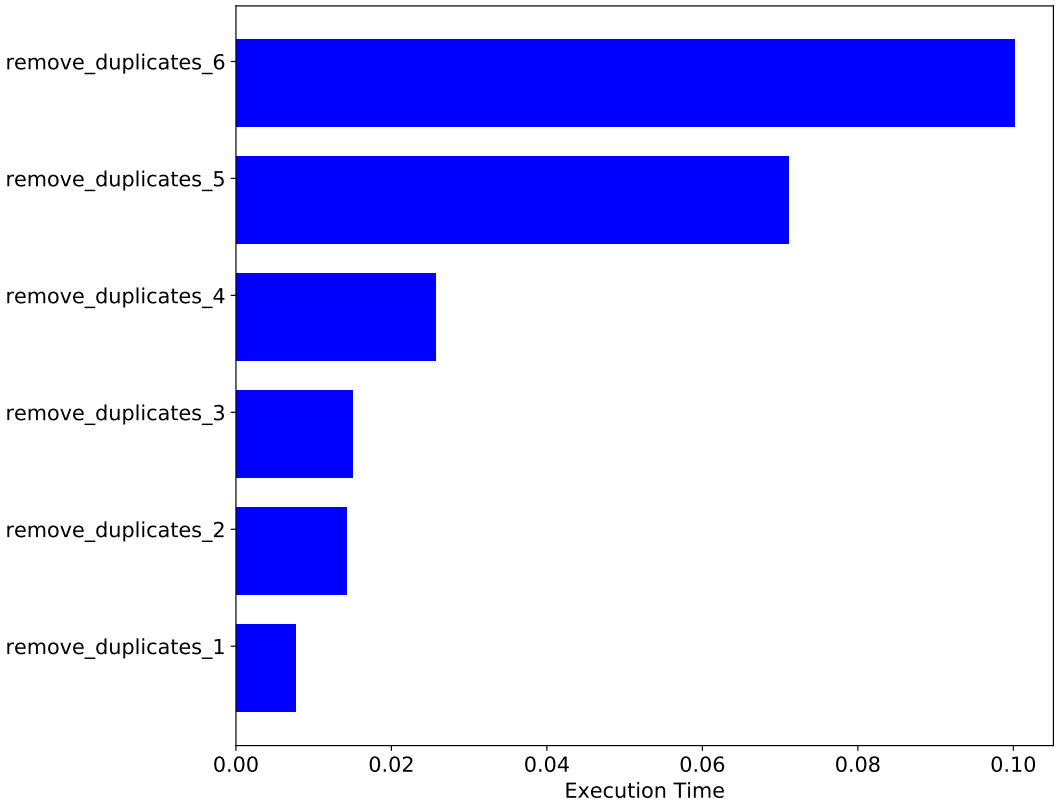


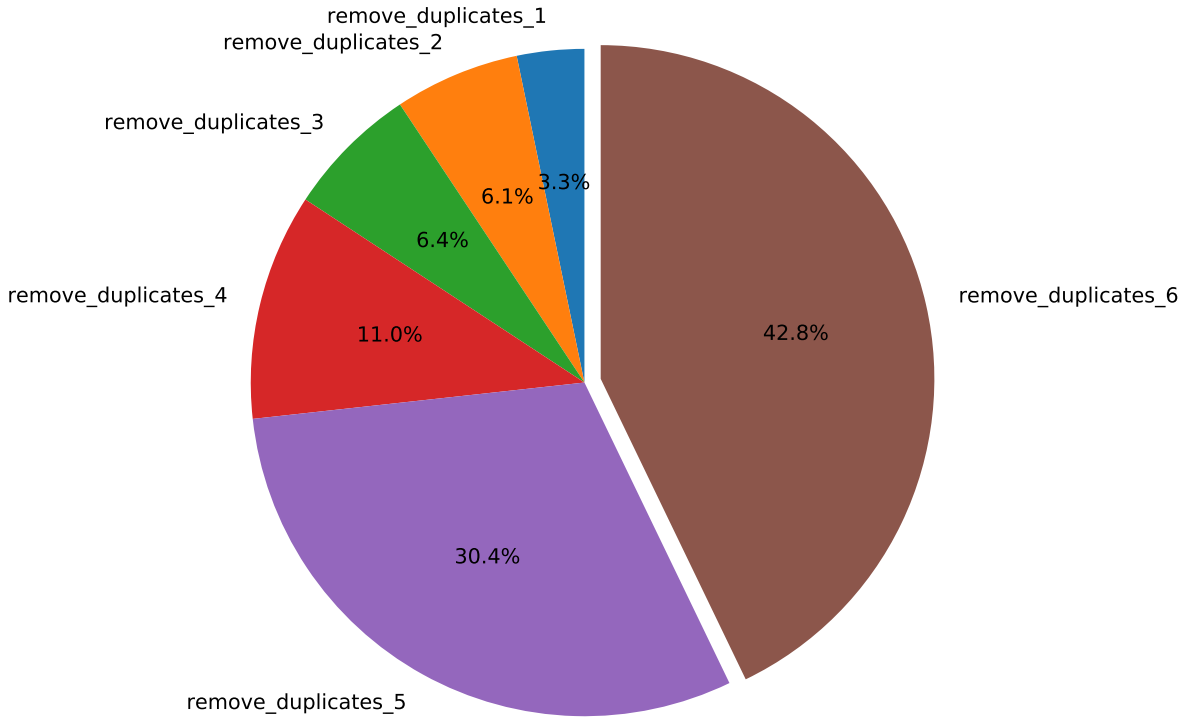
TIMES FOR THE FIRST 168000 NUMBERS



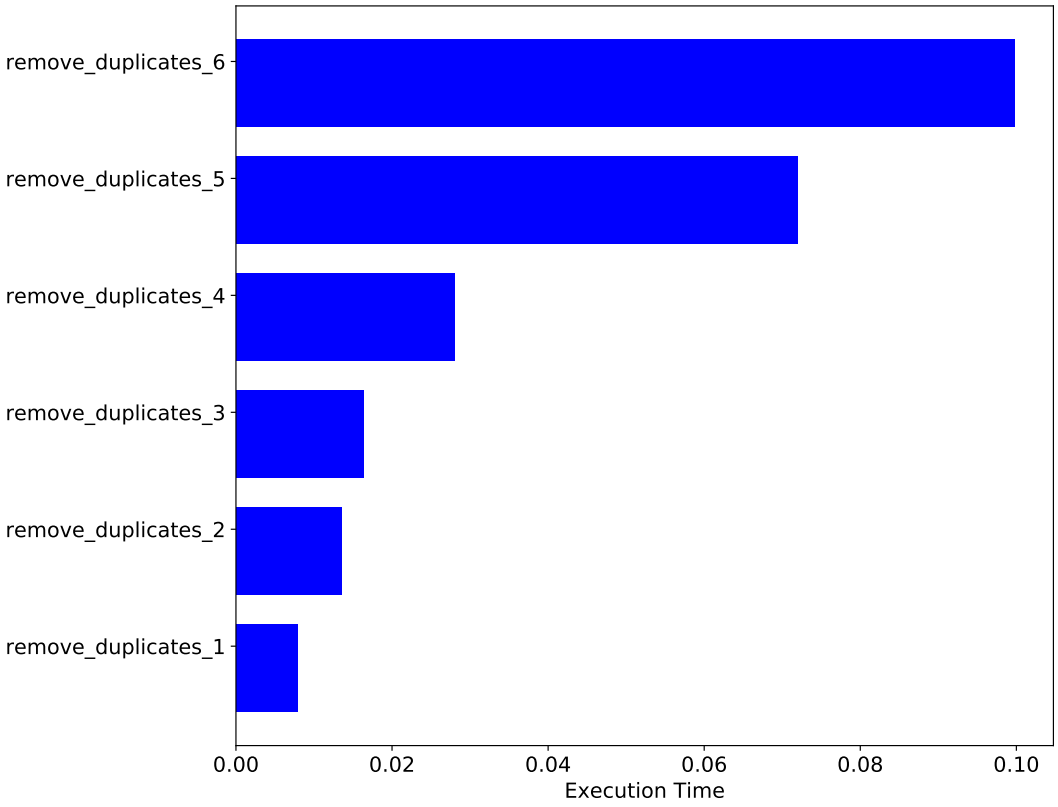


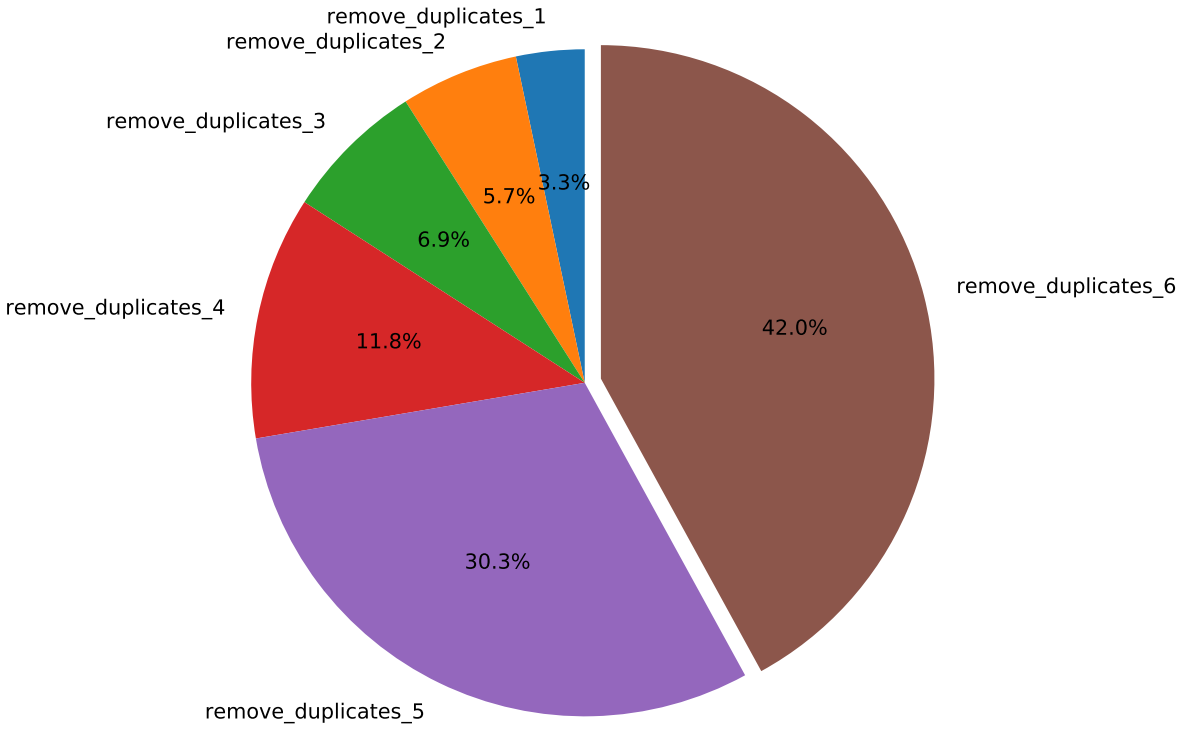
TIMES FOR THE FIRST 170000 NUMBERS



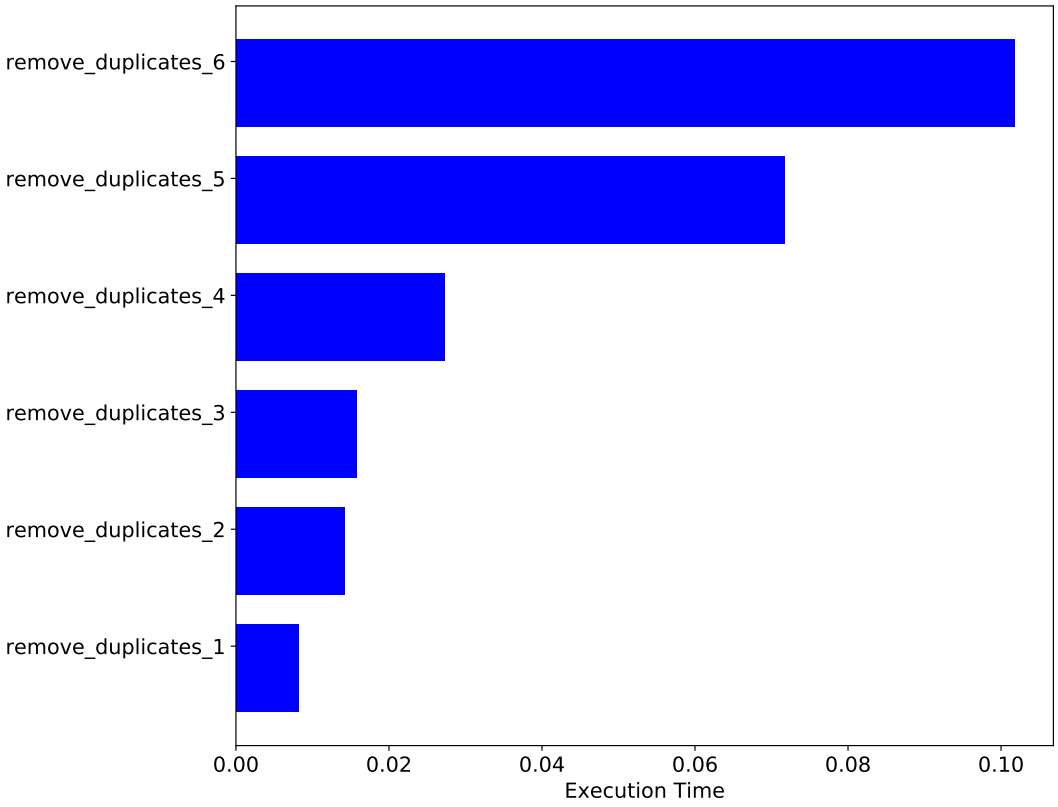


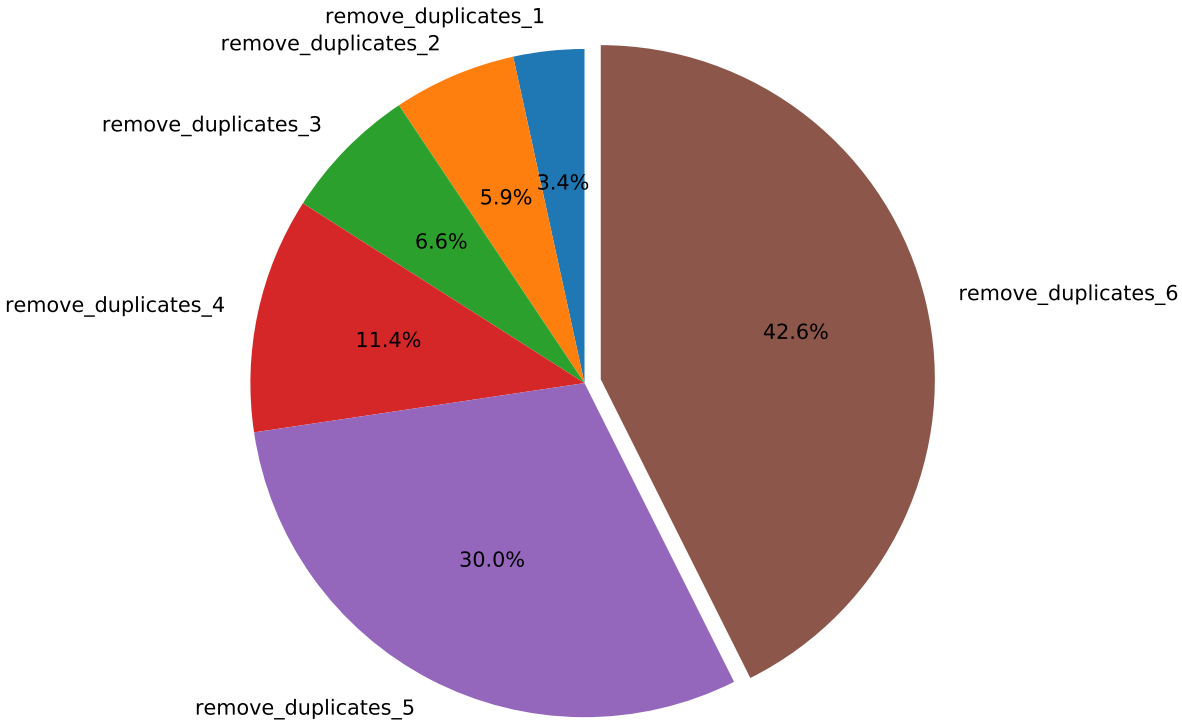
TIMES FOR THE FIRST 172000 NUMBERS



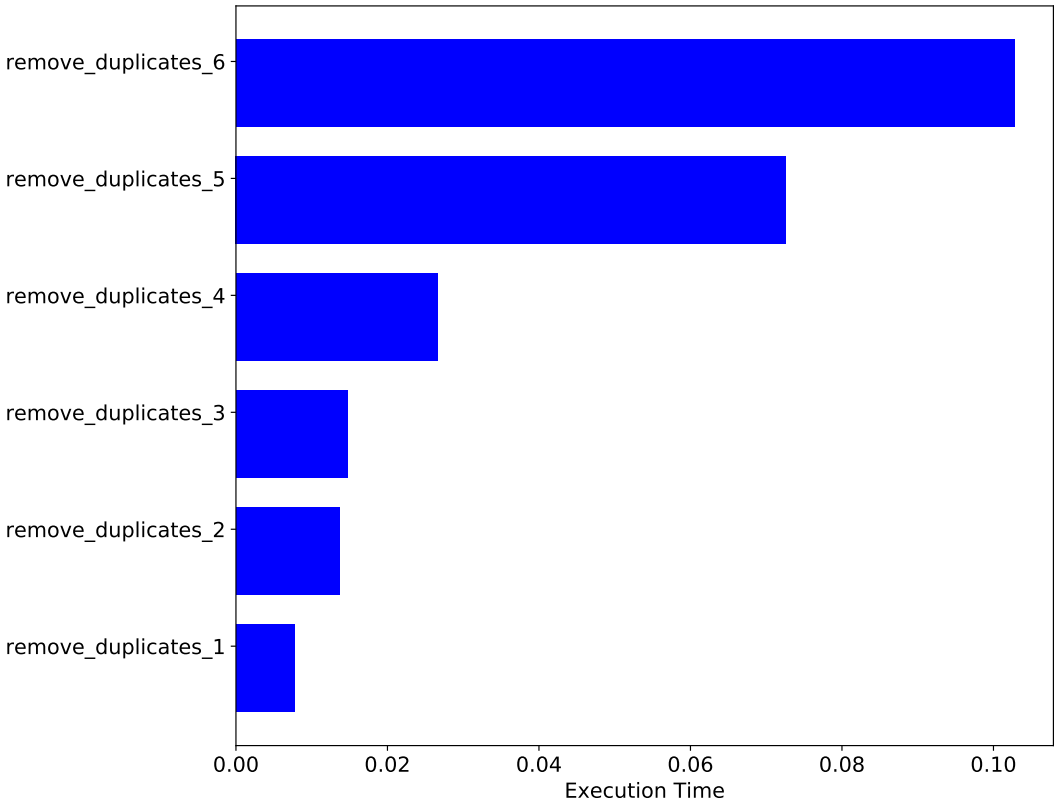


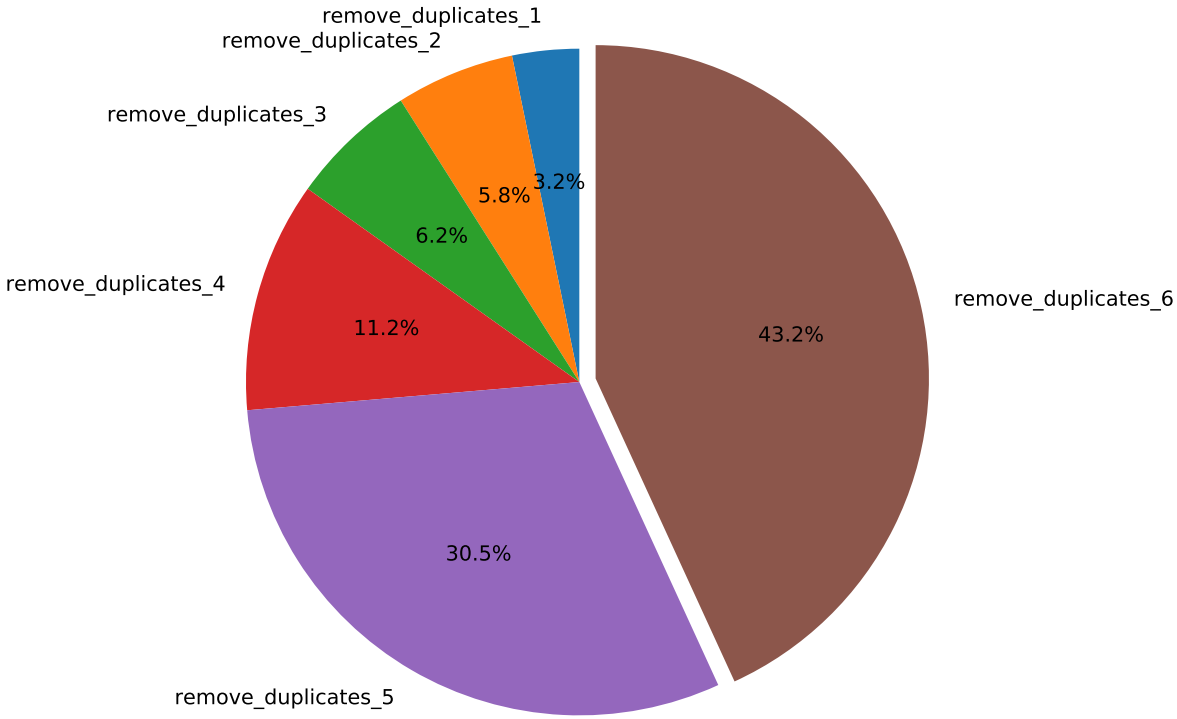
TIMES FOR THE FIRST 174000 NUMBERS



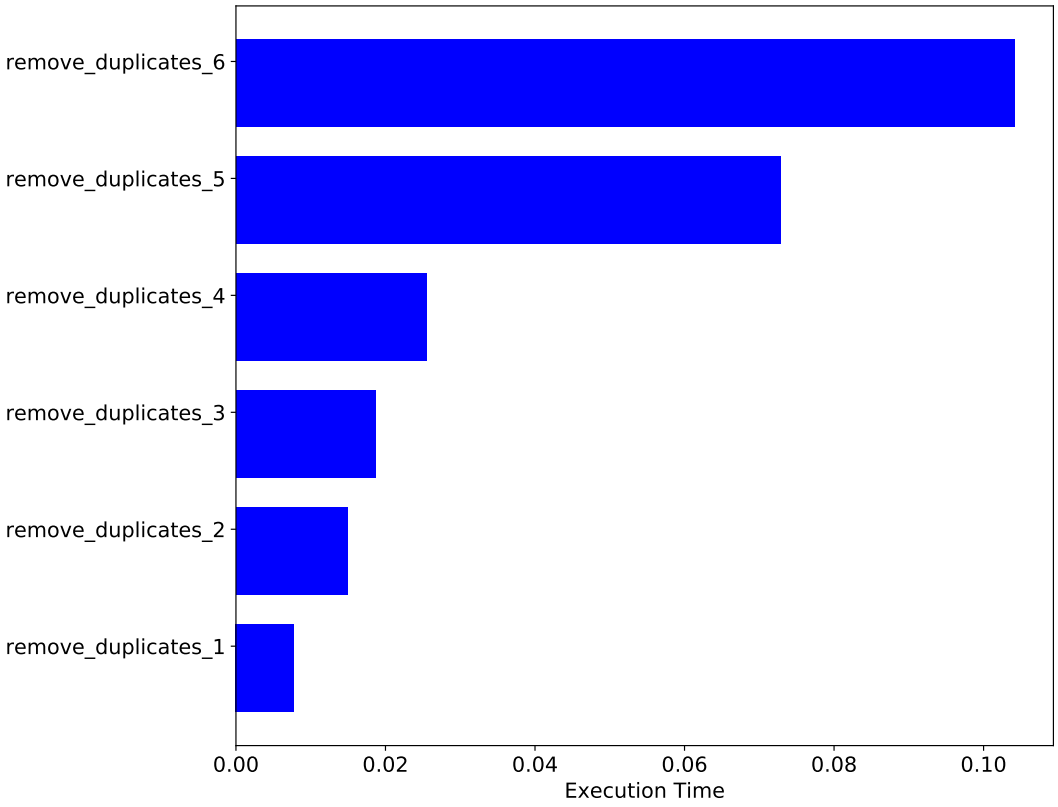


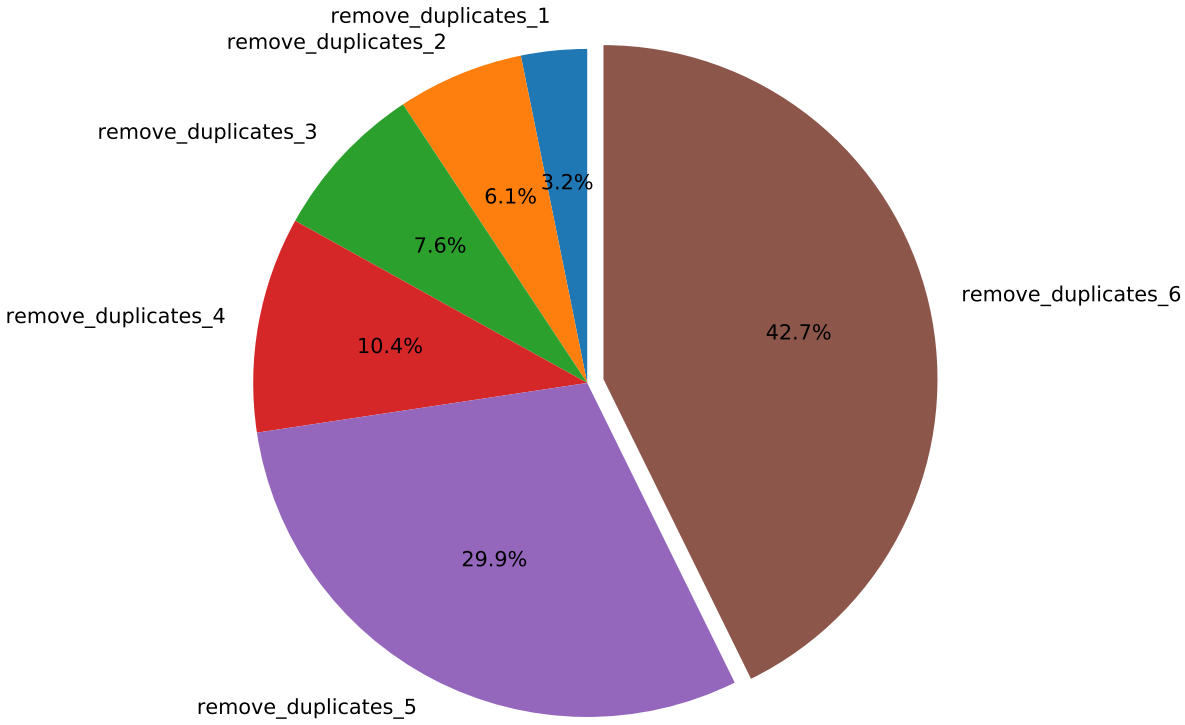
TIMES FOR THE FIRST 176000 NUMBERS



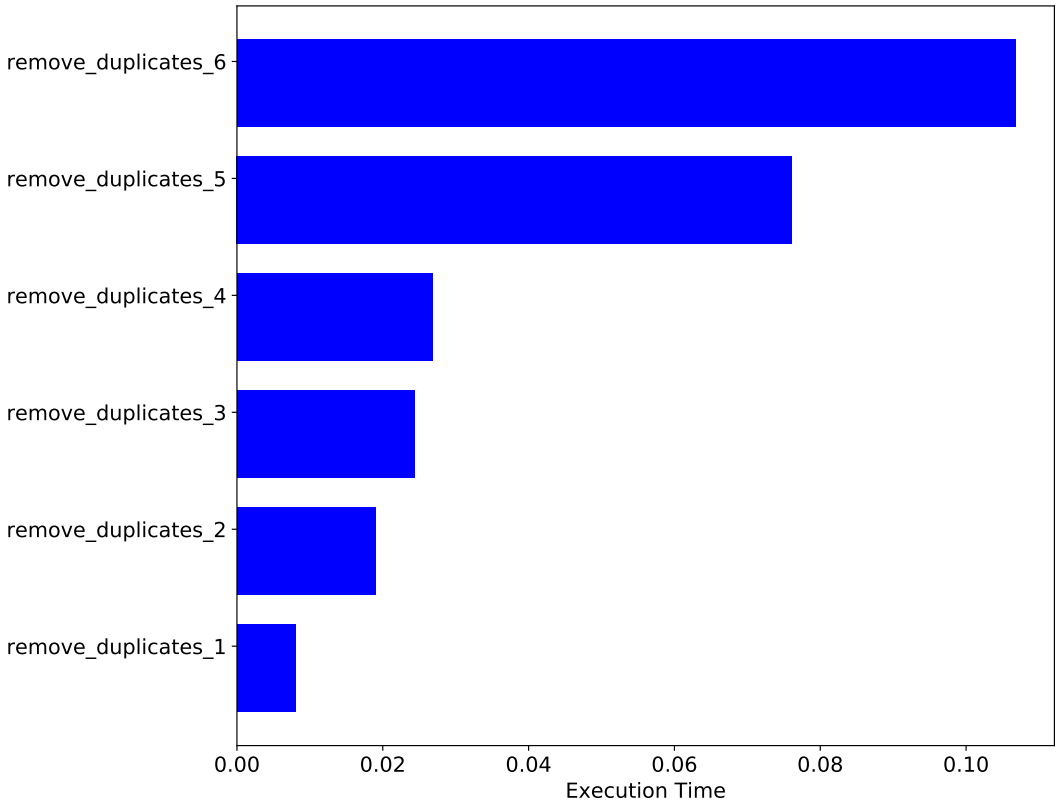


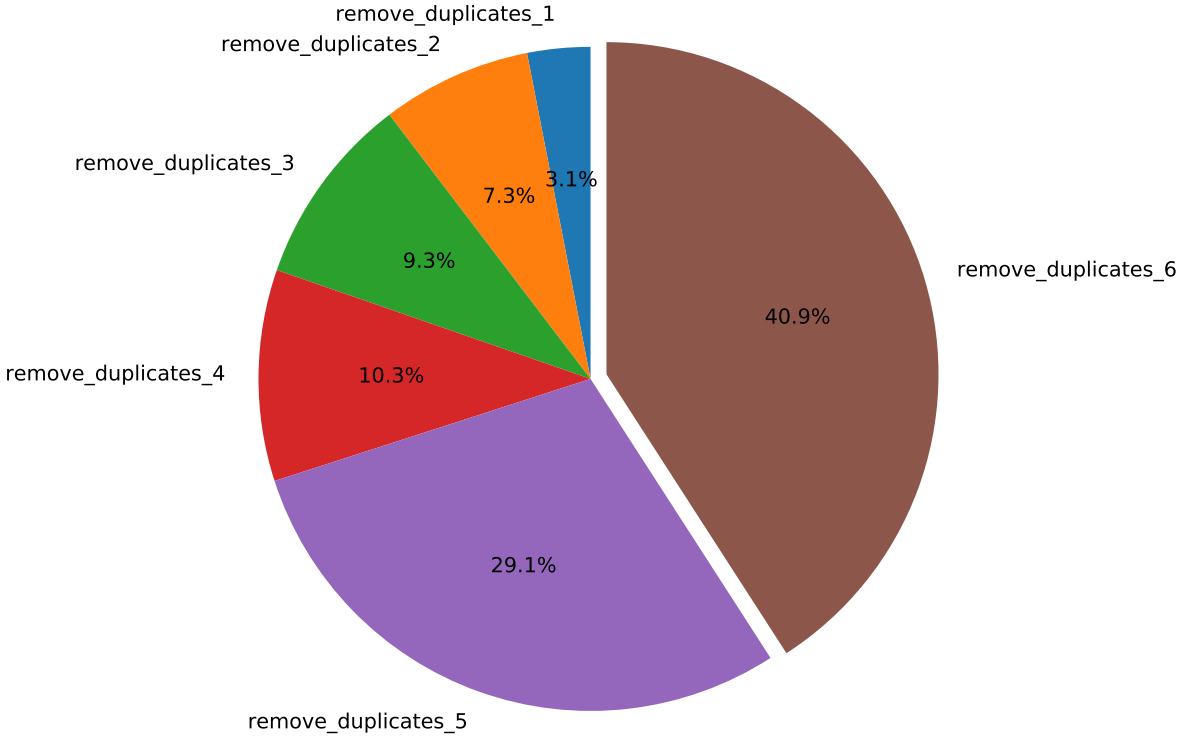
TIMES FOR THE FIRST 178000 NUMBERS



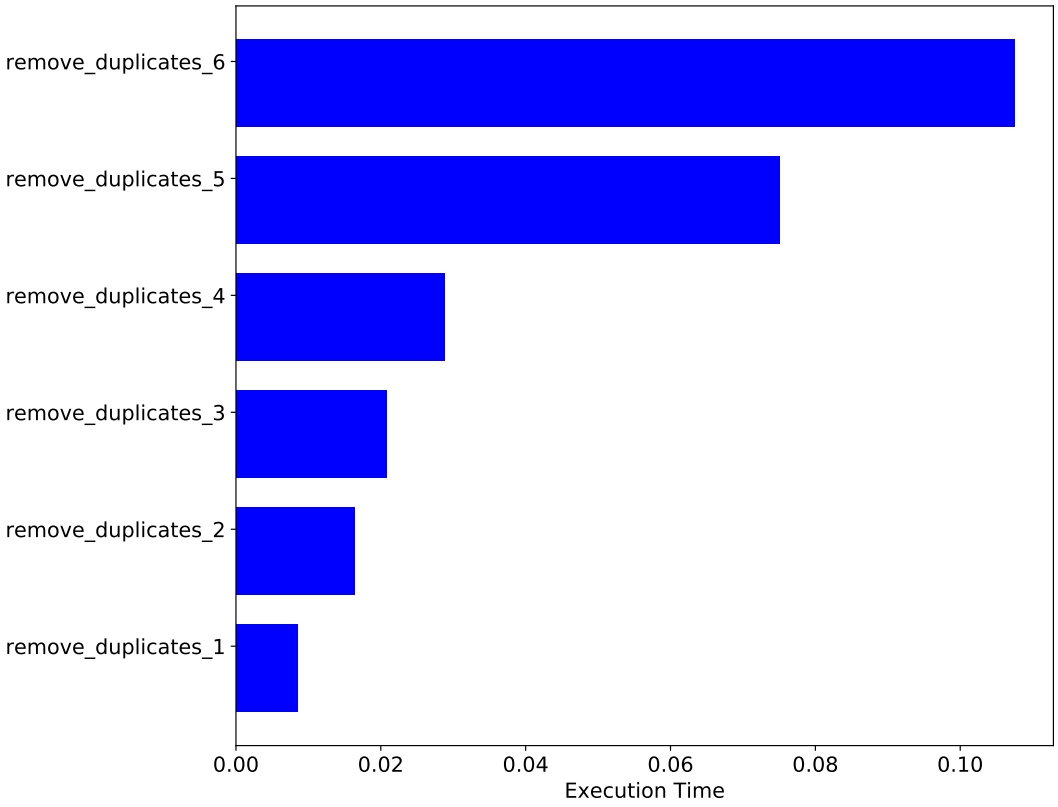


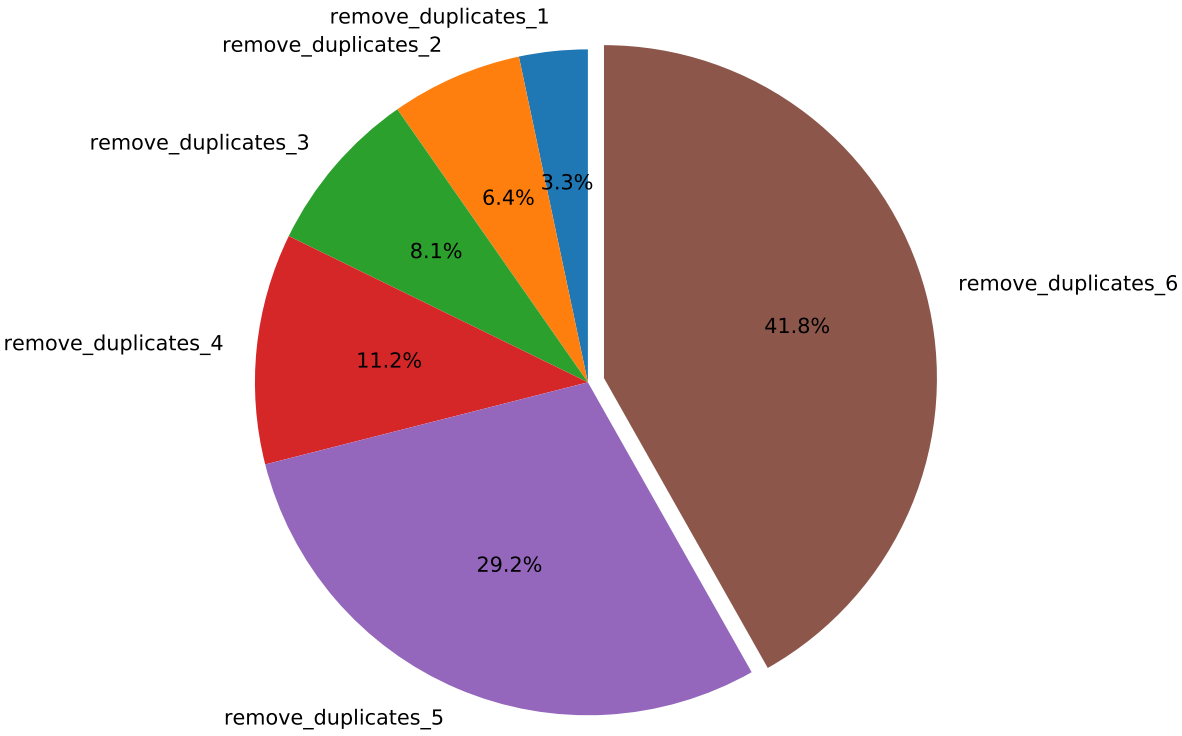
TIMES FOR THE FIRST 180000 NUMBERS



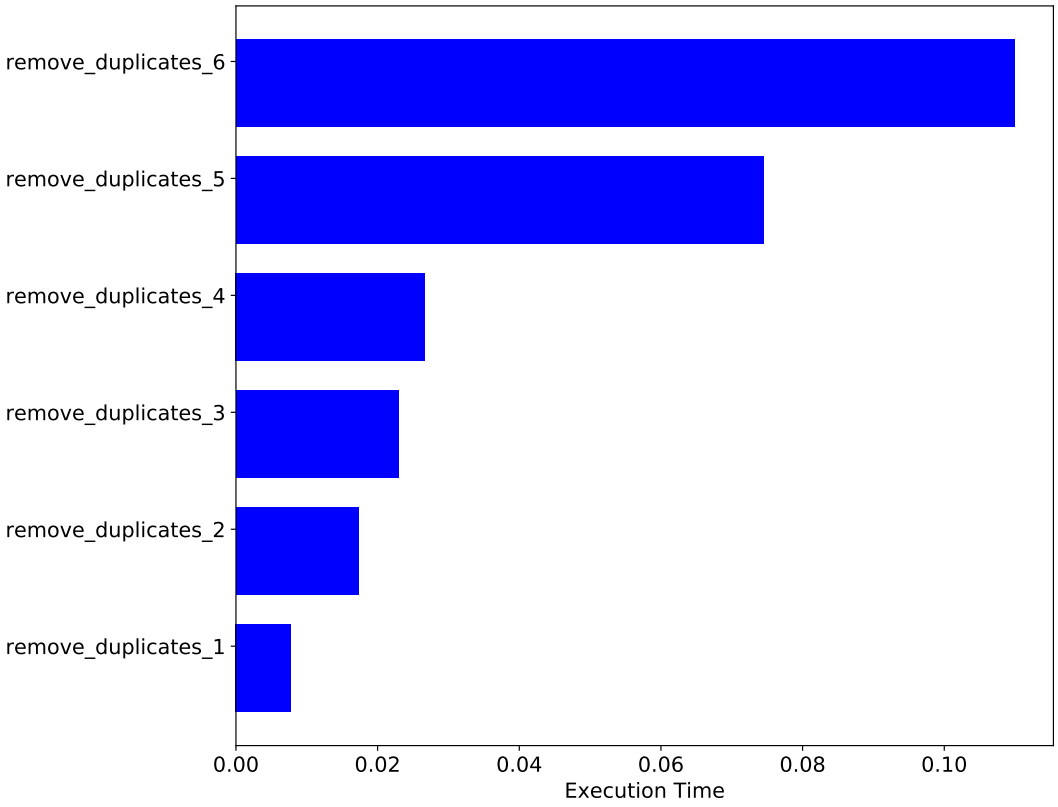


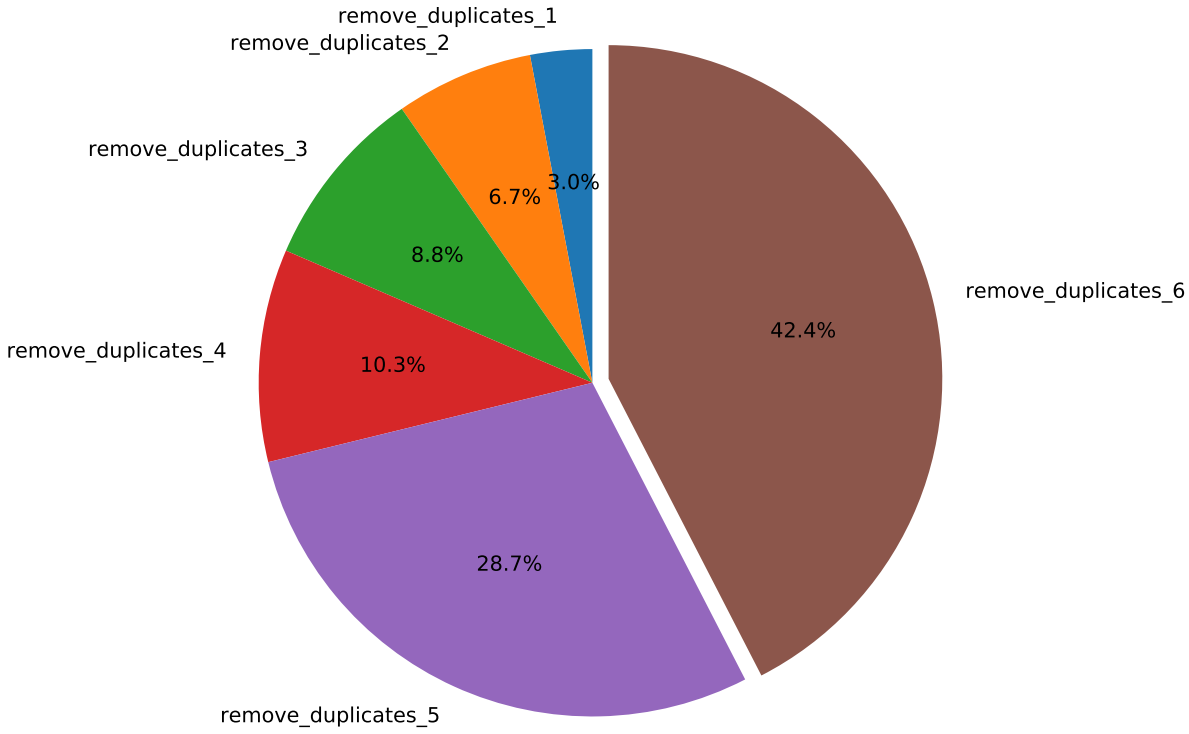
TIMES FOR THE FIRST 182000 NUMBERS



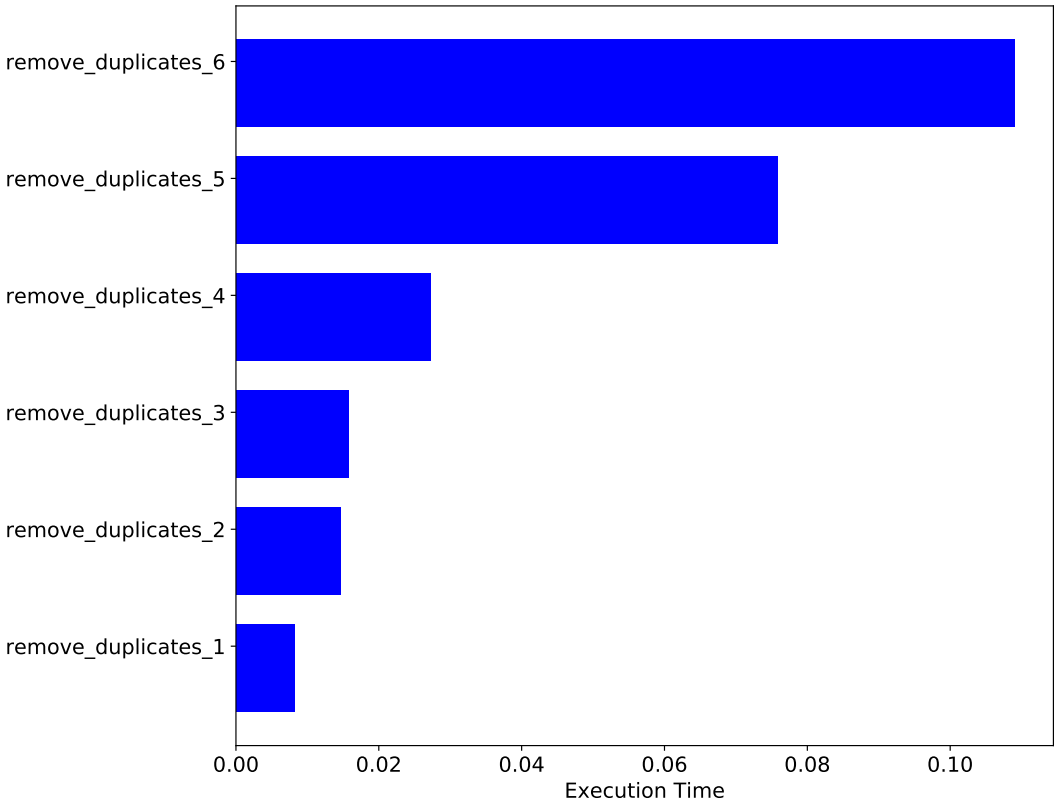


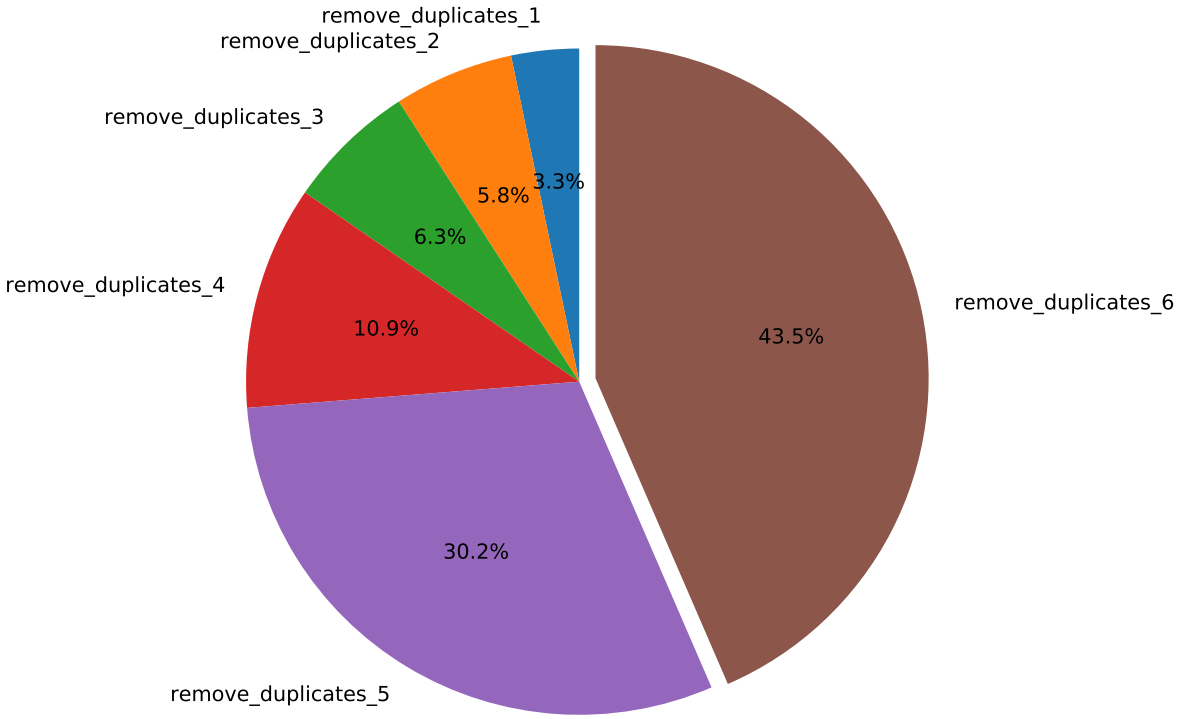
TIMES FOR THE FIRST 184000 NUMBERS



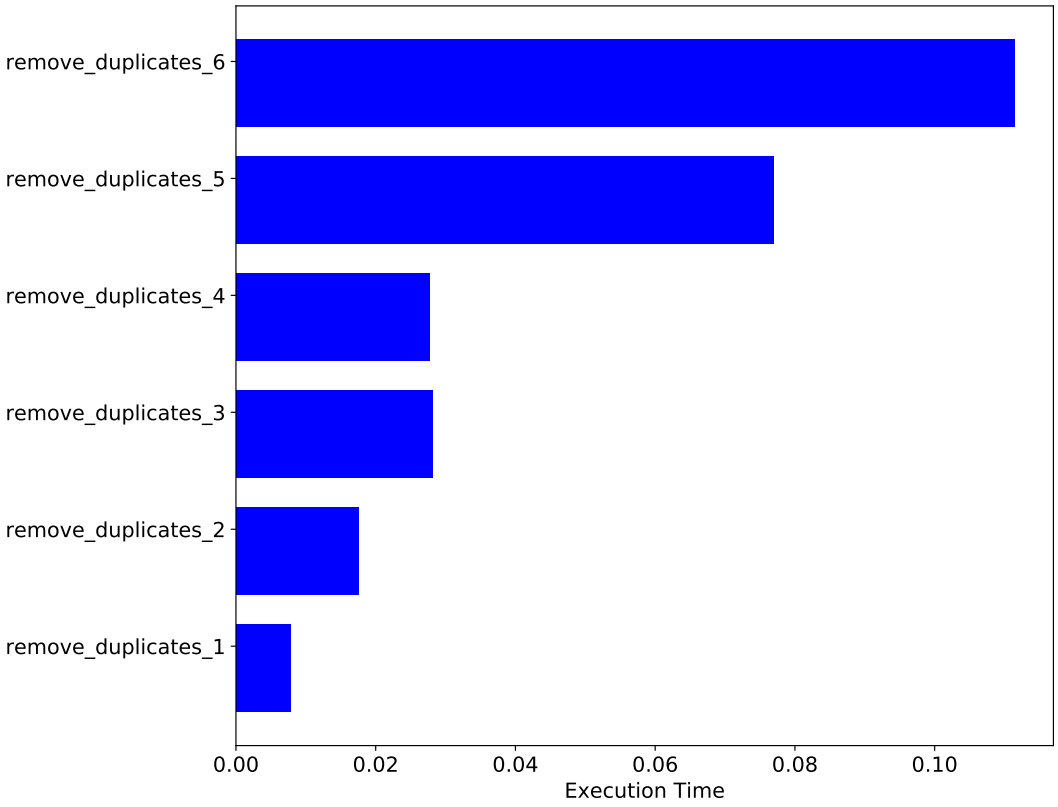


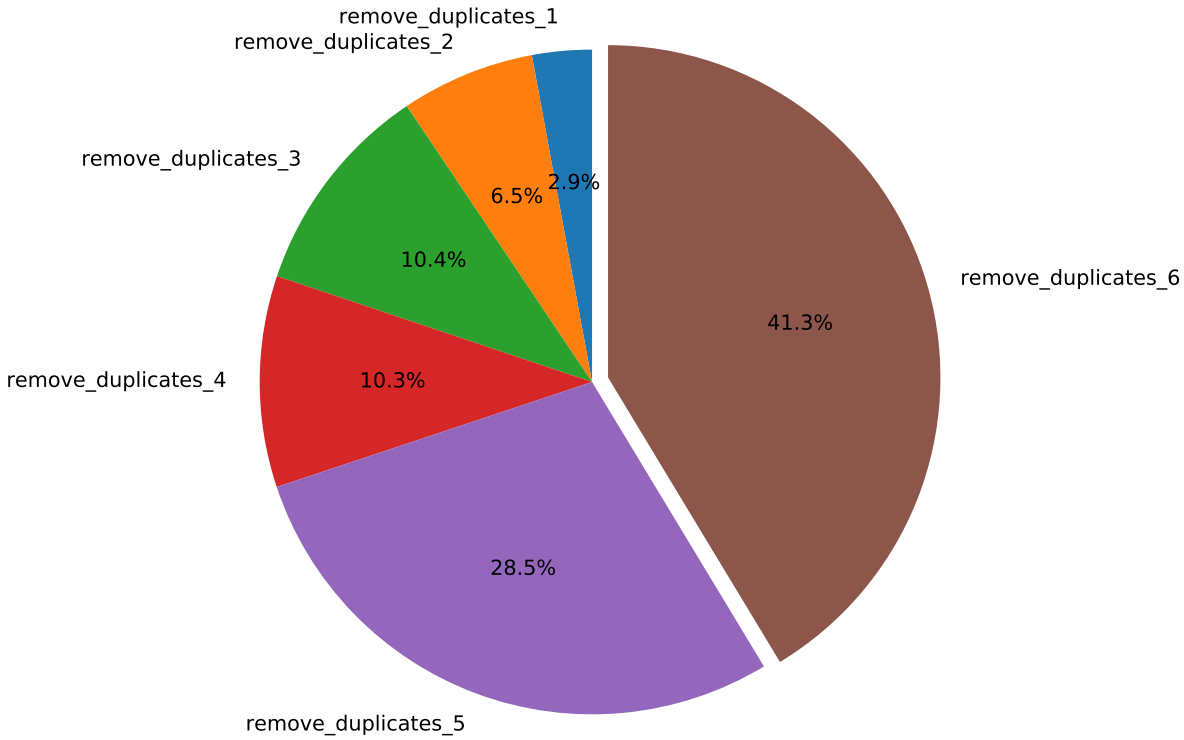
TIMES FOR THE FIRST 186000 NUMBERS



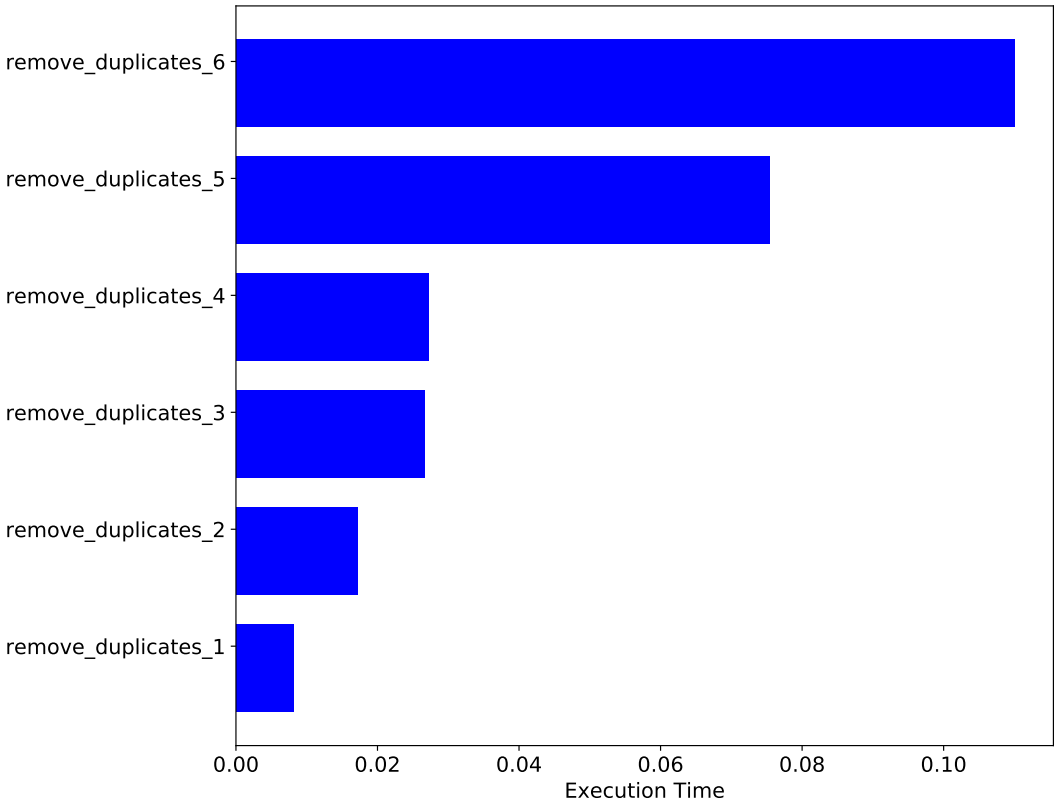


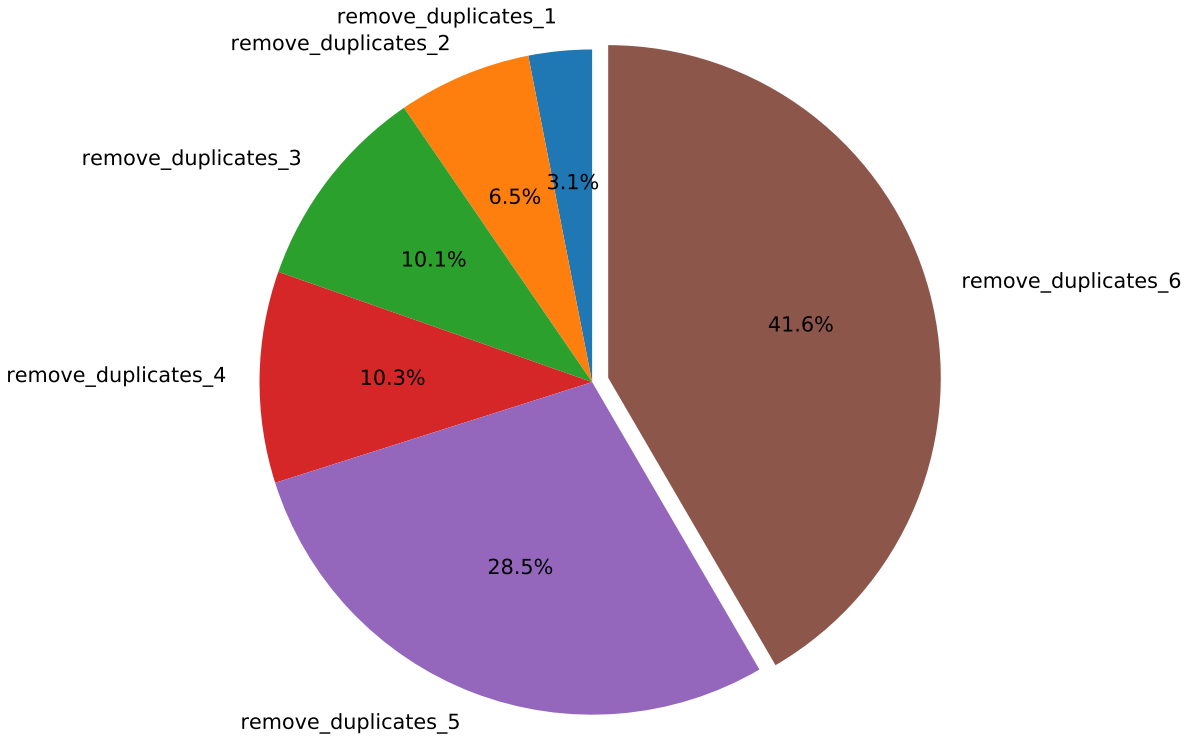
TIMES FOR THE FIRST 188000 NUMBERS



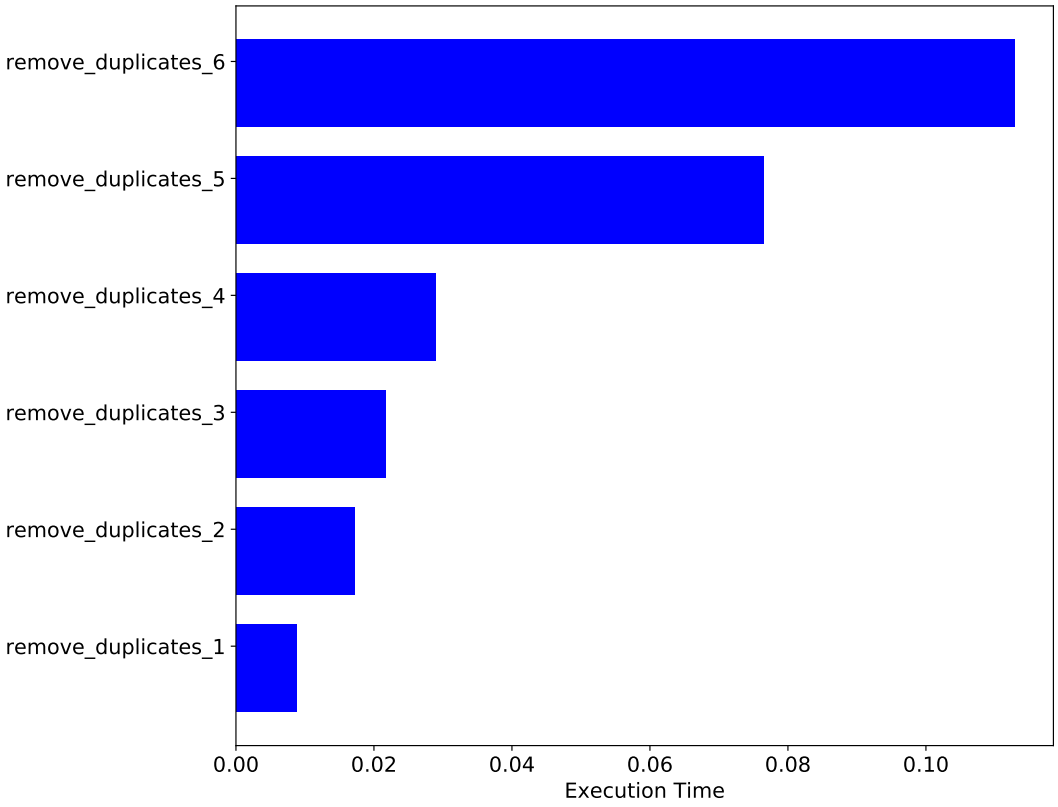


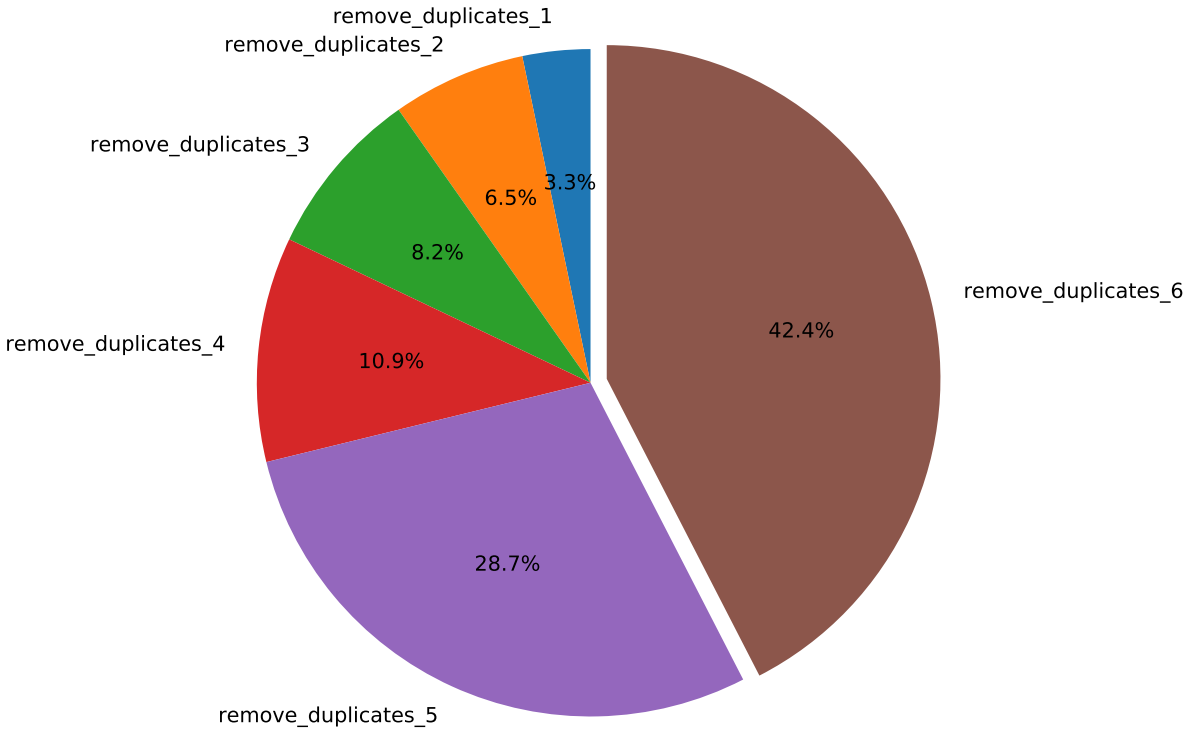
TIMES FOR THE FIRST 190000 NUMBERS



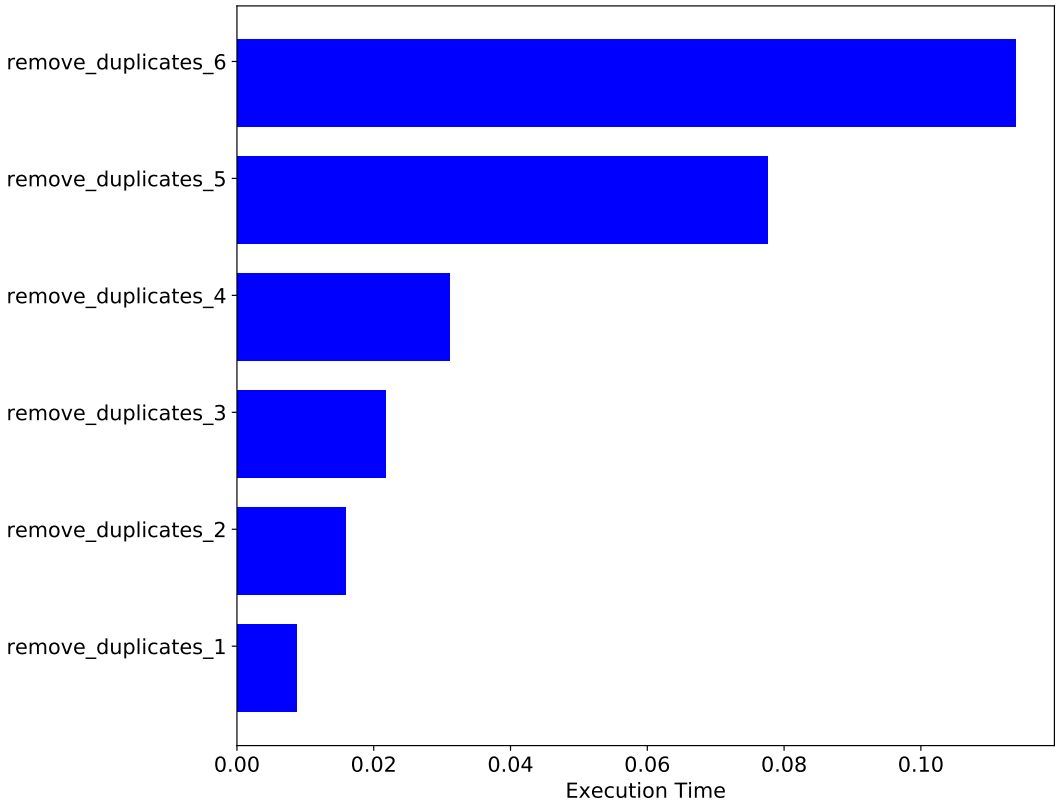


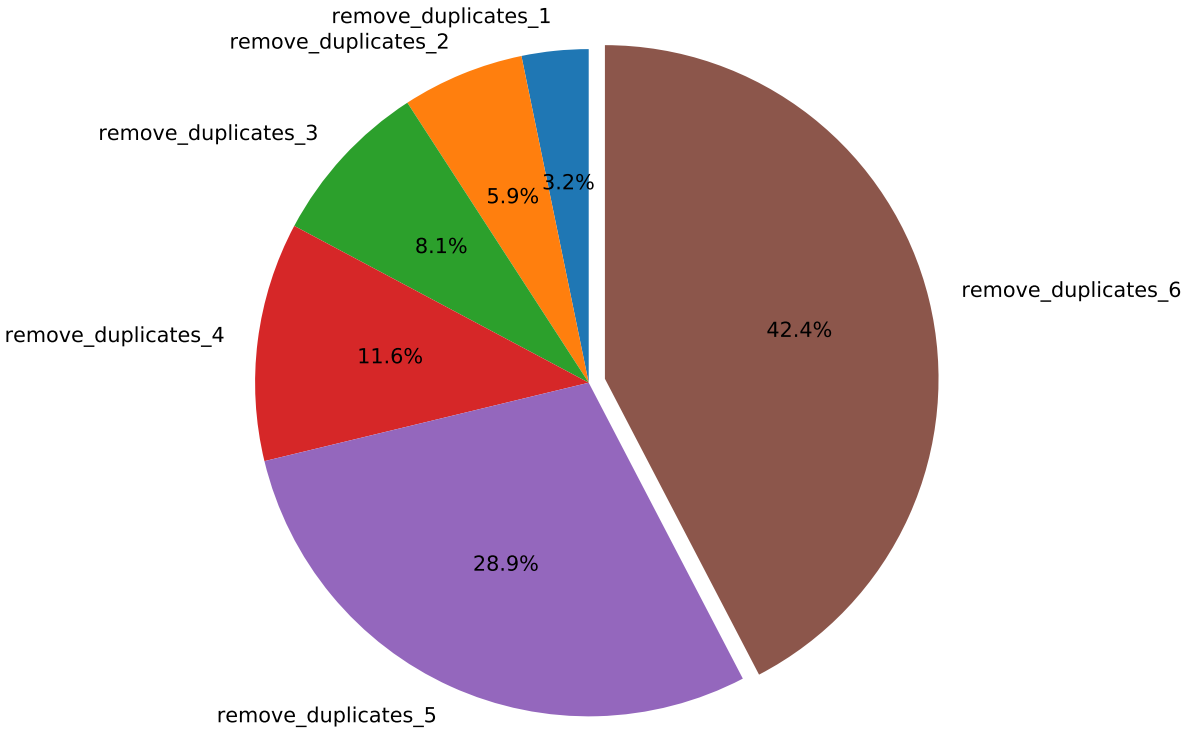
TIMES FOR THE FIRST 192000 NUMBERS



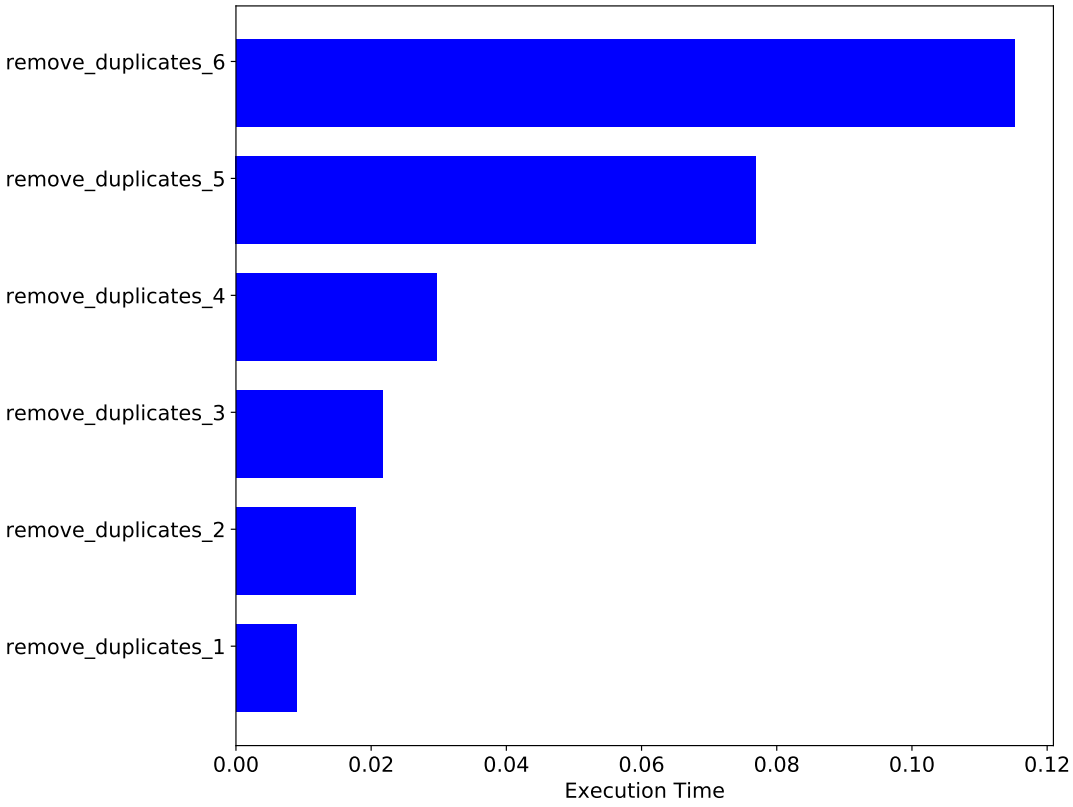


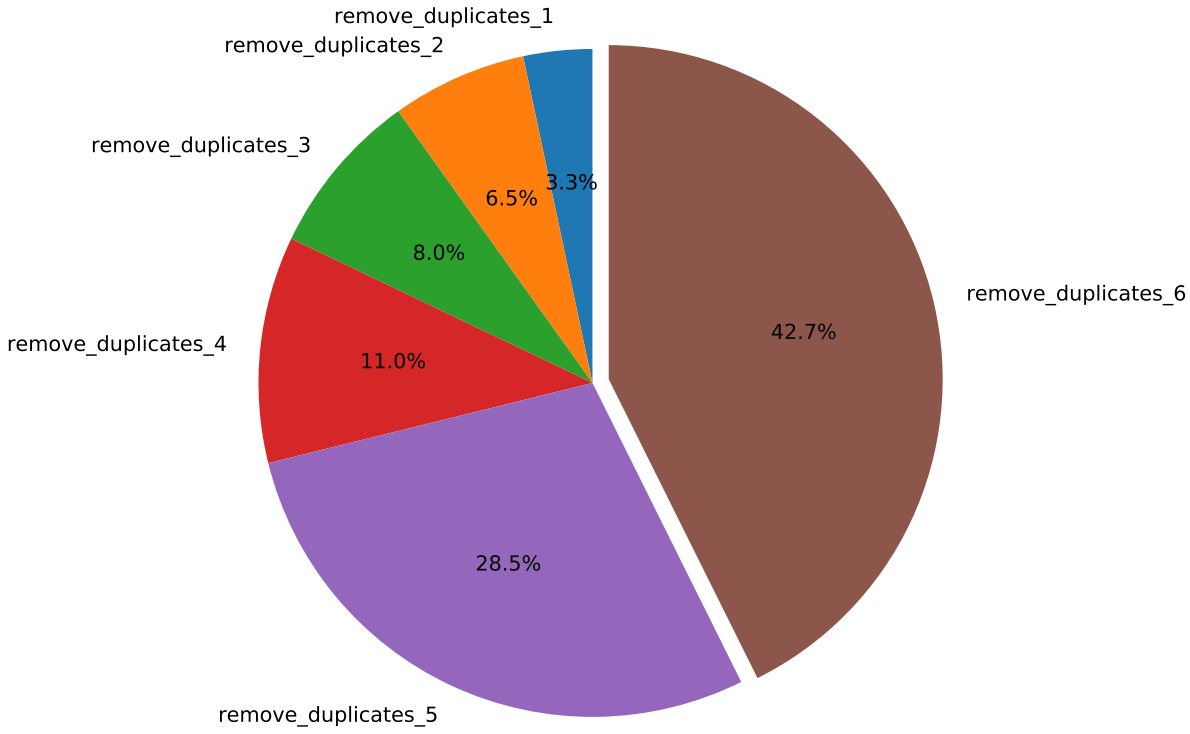
TIMES FOR THE FIRST 194000 NUMBERS



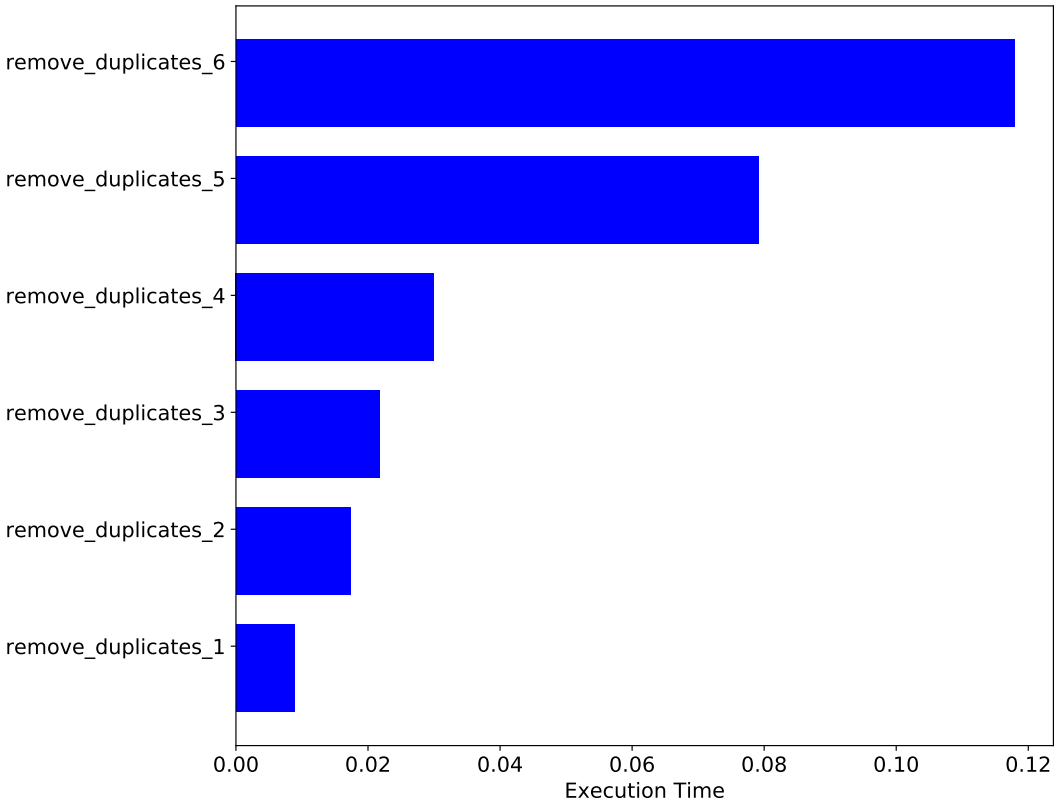


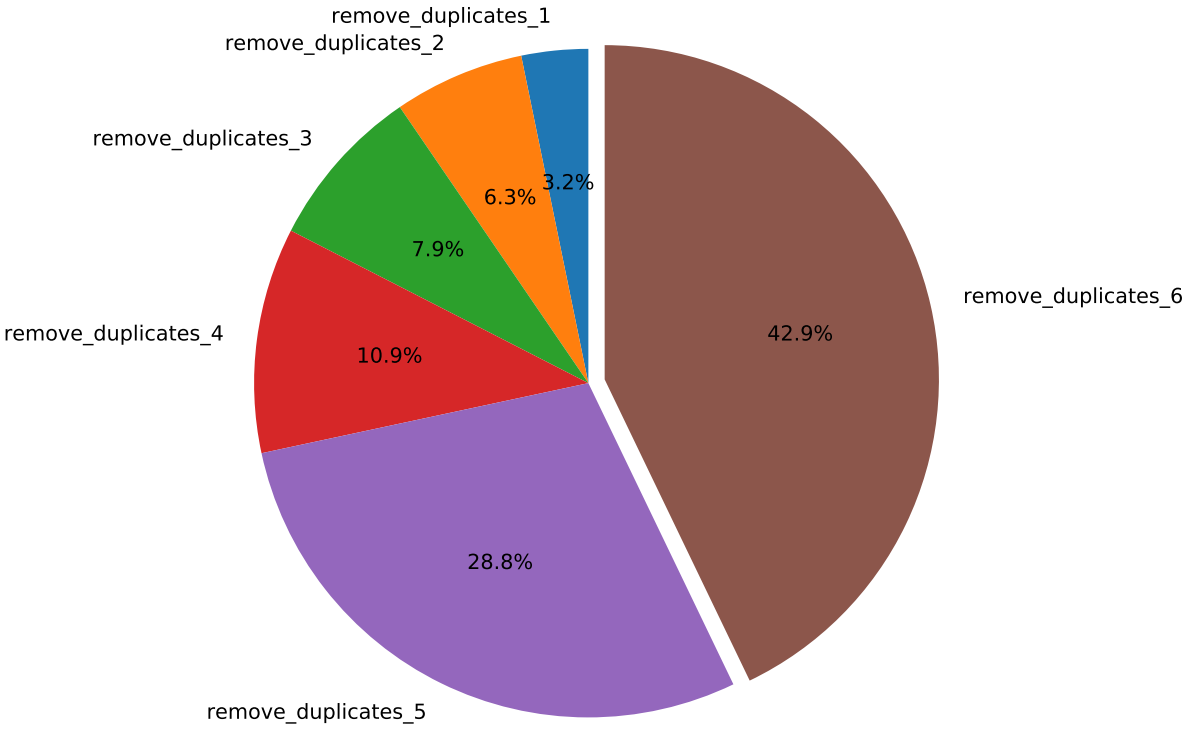
TIMES FOR THE FIRST 196000 NUMBERS



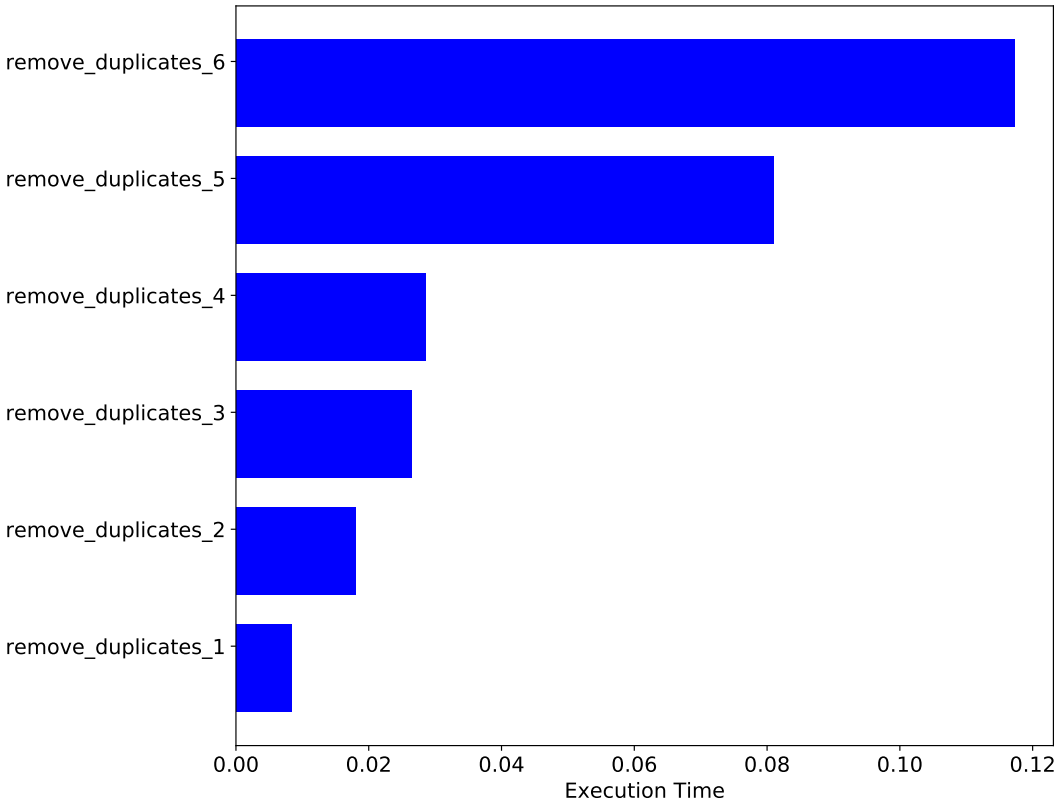


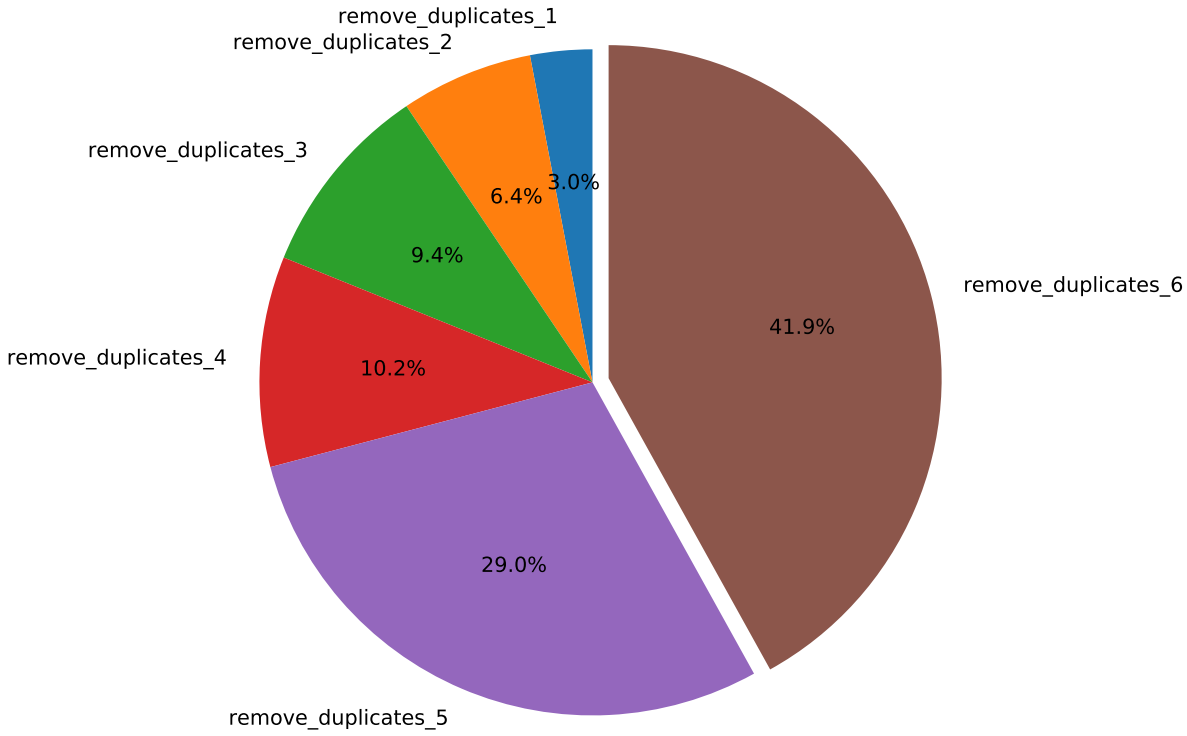
TIMES FOR THE FIRST 198000 NUMBERS





TIMES FOR THE FIRST 200000 NUMBERS





7. Bibliografía

Referencias

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] Git. *Sobre el Control de Versiones*. [*Una breve historia de Git*]. https://git-scm.com/book/es/v2/Inicio—Sobre-el-Control-de-Versiones-Una-breve-historia-de-Git

[3] Wikipedia *Análisis de rendimiento de software*. https://es.wikipedia.org/wiki/An%C3%A1lisis_de_rendimiento_de_software

[4] Matplotlib *Generación de gráficos con Matplotlib* https://matplotlib.org/