



uhu.es

**Escuela Técnica Superior de Ingeniería
Universidad de Huelva**

Grado en Ingeniería Informática

Trabajo Fin de Grado

Análisis de imágenes aplicados al estudio
del tráfico urbano

Autor : Ricardo Manuel Ruiz Díaz

Tutor : Gonzalo A. Aranda Corral

Curso 2020-2021

Índice general

1. Introducción al análisis de datos	9
1.1. Data Science o Ciencia del Dato	9
1.2. Definición del problema	10
2. Extracción de Datos	13
2.1. Información sobre las imágenes a extraer	13
2.2. Extracción de Imágenes	17
3. Procesamiento y Selección de Características	21
3.1. Procesamiento de las Imágenes	21
3.2. Selección de características	38
3.3. Automatización de los procesos de extracción de imágenes y cálculo de la ocupación	50
3.4. Preparación de los Datos	51
4. Aplicación de Algoritmos	57
4.1. Análisis exploratorio	57
4.2. Algoritmos aplicados	59
5. Visualización de Resultados	67
5.1. Evolución de la ocupación de un día, por horas	67
5.2. Comportamiento de cada día de la semana	70
5.3. Representación visual de la ocupación de la ciudad	72
5.4. Estudio sobre las cámaras no disponibles	73
6. Conclusiones	77
A. Resultado de Análisis Exploratorio	81
B. Código	135

Índice de figuras

1.1.	Ciclo de vida	9
2.1.	Mapa de Sevilla con la localización de las cámaras	13
2.2.	Imagen tipo a analizar	14
2.3.	Ejemplo de cámara con varias escenas	15
2.4.	Ejemplo de imagen con ruido	15
2.5.	Ejemplo de imagen nocturna	16
2.6.	Ejemplo de cámara fuera de servicio	16
2.7.	Estructura de carpetas	18
3.1.	Recolección de imágenes anteriores	22
3.2.	Estructura de las imágenes	22
3.3.	Cálculo de mediana a nivel de píxel	23
3.4.	Concatenación de matrices finales	23
3.5.	Ejemplo de matrices finales por colores (<i>Red, Green, Blue</i>)	24
3.6.	Ejemplo de imagen de fondo	25
3.7.	Imagen de estudio y su fondo calculado	25
3.8.	Diferencia en Blanco y Negro	26
3.9.	Diferentes <i>Threshold</i> aplicados al resultado	27
3.10.	Resultado final con <i>Threshold</i> a 70	27
3.11.	Ejemplo de ruido producido por el movimiento de la cámara	28
3.12.	Función <i>encuadrarImagen</i>	28
3.13.	Ejemplo de eliminación de ruido con <i>encuadrarImagen</i>	29
3.14.	Ejemplo de resta sin ruido	30
3.15.	Ejemplo de imagen resultante con ruido	31
3.16.	Ejemplo de selección con <i>roipoly(I)</i>	31
3.17.	Resultado de selección con <i>roipoly(I)</i>	32
3.18.	Selección de <i>roipoly(I)</i> de varias zonas	32
3.19.	Empleo de plantillas extraídas con la función <i>roipoly()</i>	33
3.20.	Proceso de creación de plantilla automática	34
3.21.	Ejemplo de creación de plantilla automática	35
3.22.	Ejemplo de creación de plantilla automática con exceso de ruido	35
3.23.	Ejemplo de aplicación de cierre morfológico	36
3.24.	Empleo de plantillas extraídas automáticamente	36
3.25.	Comparación de métodos de extracción de zonas de interés (1)	37
3.26.	Resultado del cálculo de la ocupación	38
3.27.	Ejemplo de cámara con diferentes plantillas	40
3.28.	Ejemplo de plantillas, imagen de estudio e imagen de fondo	44

3.29. Problema de profundidad	45
3.30. División de imagen en franjas	45
3.31. Ejemplo de Elección de plantillas	46
3.32. Ejemplo de Elección de plantillas (2)	46
3.33. Resultado final de la preparación de datos	51
3.34. Representación de la cámara 1	53
3.35. Histograma de la cámara 1	53
3.36. Comparación de imagen normal con sus outliers	54
3.37. BoxPlot de Ejemplo	54
4.1. Resumen	58
4.2. Ejemplo de Variables	58
4.3. Mapa de calor del resultado de la Correlación de Pearson	59
4.4. Resultado de relación lineal	60
4.5. Ejemplo cámara 5	60
4.6. Gráfica con el método de Elbow a nuestros datos	61
4.7. Gráfica con el método average silhouette a nuestros datos	62
4.8. Clasificación por colores según <i>Kmeans</i>	63
4.9. Clasificación de vías rápidas según <i>Kmeans</i>	64
4.10. Diagrama de Voronoi	65
4.11. Mapa con el Diagrama de Voronoi	65
4.12. Mapa con el Diagrama de Voronoi Ampliado	66
4.13. Dirección de la ciudad	66
5.1. Ejemplo de la cámara 23	67
5.2. Evolución del día de la cámara 23	68
5.3. Evolución del día de la cámara 23 un domingo	68
5.4. Ejemplo de la evolución cuando está no disponible	69
5.5. Evolución del día de la ciudad	69
5.6. Superposición de gráficas	70
5.7. Caracterización de todos los lunes de junio	70
5.8. Comportamiento medio de todos los lunes de la cámara 23	71
5.9. Comportamiento medio de todos los domingos de la cámara 1	71
5.10. Comportamiento medio de cada día de la semana	71
5.11. Mapa en un instante dado de la ciudad	72
5.12. Mapa aumentado en un instante dado de la ciudad	73
5.13. Cámaras con más imágenes perdidas	74
5.14. Cámara que más tiempo ha estado no disponible	74
5.15. Mapa con la posición de las cámaras con más fallos	75

Resumen

El objetivo de este proyecto es estudiar la ocupación de las carreteras de Sevilla. Para dicho cálculo nos ayudaremos de unas imágenes extraídas de la web del Ayuntamiento de Sevilla. El proceso empieza con la extracción del fondo utilizando el resultado de la mediana de X imágenes consecutivamente anteriores. Una vez tengamos este fondo, calculamos la diferencia de la imagen actual con la imagen de fondo y con ello obtenemos como resultado los píxeles distintos entre las dos imágenes. Estos píxeles distintos indican la ocupación de vehículos que tenemos en la carretera. Pero además, gracias a unas plantillas calculadas previamente, identificamos la zona de carretera y con ello eliminamos todo el ruido externo a las carreteras. Una vez terminado este proceso ya tendríamos la ocupación de una carretera gracias a una imagen de estudio. Este proceso se automatizará de manera que iremos guardando el resultado de estas imágenes en una base de datos. Cuando tengamos un número de imágenes significativo, podremos realizar un estudio de cómo evoluciona la circulación en la ciudad de Sevilla gracias a todas las cámaras distribuidas por la ciudad.

Abstract

The goal of this project is study the occupation of the roads in Seville. For this calculation we will use some images taken from the website of the Seville City Council. The process begins with the extraction of the background using the result of the median of X consecutively previous images. Once we have this background, we calculate the difference of the current image with the background image and with this we obtain the different pixels between the two images. These different pixels indicate the occupancy of vehicles that we have on the road. But also, thanks to previously calculated templates, we identify the road area and we can eliminate all external noise from the roads. Once this process is finished we would already have the occupation of a road thanks to a study image. This process will be automated so that we will save the results of these images in a database. When we have a significant number of images, we will be able to carry out a study of how the circulation in the city of Seville evolves with the help to all the cameras distributed throughout the city.

Capítulo 1

Introducción al análisis de datos

1.1. Data Science o Ciencia del Dato

Data Science o Ciencia del Dato, es la ciencia centrada en el estudio de grandes cantidades de datos. Se encarga de extraer información, comprender la realidad y descubrir patrones con los que tomar decisiones.

Para conseguir que esta información, ya sea no estructurada o estructurada, se convierta en contenido de valor, la Ciencia del Dato combina herramientas matemáticas, estadísticas y/o informáticas.[2]

Todo proyecto de análisis de datos tiene un ciclo de vida que describe los pasos por los que debe pasar dicho proyecto para una correcta realización de éste. El ciclo de vida consta de cinco pasos: [4]

1. Extracción de datos.
2. Procesamiento y selección de características.
3. Aplicación de algoritmos.
4. Visualización de datos.
5. Conclusiones.



Figura 1.1: Ciclo de vida

1.2. Definición del problema

Poco a poco, podemos ir viendo como la circulación con vehículos convencionales (coches, autobuses, etc ...) en las ciudades va empeorando debido a la mayor cantidad de vehículos y que las vías no están preparadas para este aumento de vehículos y como consecuencia se producen atascos, por lo que los trayectos cada vez son más largos. Según estudios realizados, Sevilla es la tercera gran capital con más atascos después de Barcelona y Madrid [5] .

La finalidad de este proyecto es analizar la circulación de la ciudad de Sevilla a través de las imágenes para ver la situación actual de la ciudad y poder ver cómo afectan este tipo de situaciones.

Cómo se ha comentado anteriormente, todo proyecto de análisis de datos consta de 5 pasos esenciales. A continuación, vamos a explicar las acciones realizadas en cada paso:

Extracción de datos

Los datos necesarios para poder realizar este proyecto son imágenes. Estas imágenes son extraídas desde la web del Ayuntamiento de Sevilla la cual es pública y cualquiera puede acceder.

En esta sección explicaremos el *script* creado para extraer las imágenes de manera automatizada de todas las cámaras de la web. Además este script no sólo descarga las imágenes actual de cada cámara, sino que también elimina las imágenes innecesarias, es decir, en cada cámara sólo se necesitan 10 imágenes para el cálculo de la ocupación de una imagen actual. Estas imágenes son la imagen a analizar y 9 imágenes más para calcular el fondo. De manera que cada vez que se realiza la extracción de una imagen, que será la imagen de estudio, borrará la imagen más antigua, con lo que tenemos las 9 imágenes más recientes para calcular este fondo.

Procesamiento y selección de características

Una vez que se tiene las 10 imágenes de cada cámara, el primer paso a realizar será calcular el fondo.

Para extraer el fondo de la imagen a estudiar, se utilizará el cálculo resultante de la mediana del mismo píxel de las 9 imágenes anteriores a esta, es decir, tenemos el valor de todos los píxeles situados en la misma posición de cada imagen y con este vector calculamos la mediana. Este resultado devuelve el valor situado en la posición central en el vector de valores ordenados.

Inmediatamente después, realizando una resta entre la imagen de estudio y la imagen de fondo, devolverá una imagen donde resaltarán los píxeles que han sufrido modificaciones, en nuestro caso, los píxeles de carretera que están ocupados. Pero, en esta imagen también aparecerá cierto ruido procedente de las aceras, árboles, edificios, etc.

Para eliminar este ruido, se han generado unas plantillas que identifican qué zona de la imagen es carretera y qué zona de la imagen no lo es.

Superponiendo esta plantilla con la imagen resultante de la diferencia, se podrá tapar todo el ruido que esté fuera de esta carretera, quedando así sólo los píxeles dentro de ella que es nuestro objetivo.

Todo este proceso explicado tanto en el primer punto como en este, se automatizará en un servidor de manera que se esté ejecutando automáticamente y vaya guardando los resultados en una base de datos para su posterior estudio.

Pero, si se observa bien, ahora mismo los datos que estamos guardando son píxeles ocupados. Este número de píxeles ocupados de una cámara no es comparable con el de otras cámaras, ya que las resoluciones son distintas y con ello la cantidad de píxeles en cada imagen varía. Para solucionar este problema, se ha normalizado los valores, tomando los máximos y mínimos obtenidos por cada cámara, teniendo así una medida comparable entre cada cámara.

Aplicación de algoritmos

Llegados a este punto, ya se tienen los datos filtrados para su estudio.

Primero se realizará un análisis exploratorio de todas las variables (cámaras) para ver los valores con los que se va a trabajar (rango, tipo, histograma, porcentaje de valores nulos o *missing values* , etc).

A continuación, aplicaremos el algoritmo *Kmenas* para ver la similitud entre cada cámara, y se representará en un mapa todas las cámaras marcadas con un color en función de la clasificación obtenida este algoritmo.

Para finalizar este punto, se realizará un diagrama de Voronoi para observar el radio de influencia de cada cámara sobre un mapa de la ciudad de Sevilla, y con ello poder estudiar la necesidad de implementar más cámaras en qué zonas o quitar cámaras debido a que existan demasiadas en una misma zona.

Visualización de datos

En esta sección, se representará los valores filtrados de cada cámara, y con ellos podremos observar gráficamente cómo evoluciona a lo largo del día la ocupación en una cámara, comportamiento de cada día de la semana, etc.

Después se representará en un mapa todas las cámaras, pero en este caso el color de ellas indicará la ocupación actual en la carretera.

Para finalizar, realizaremos un estudio de estas calidad de estas cámaras, debido a que hay veces que las cámaras no están disponibles como consecuencia de fallos en ellas. Esto produce pérdidas de datos. Estos fallos son casi imprevisibles, pero lo que si se puede mejorar es el tiempo de reacción en el momento falla una cámara o se detecta que va a fallar. Por lo que se ha realizado un estudio de las cámaras que más tiempo se han llevado en no disponible.

Conclusiones

Por último, en este apartado se realizará unas conclusiones del proyecto. Se hablará de la veracidad de los datos obtenidos cómo de posibles mejoras de este sistema.

Por último, se ha de comentar que debido a la complejidad del propio problema, haremos más énfasis en el análisis de las imágenes dentro de este proyecto, es decir, en los dos primeros puntos.

También, comentar que este proyecto puede servir como punto de inicio para otros proyectos que encuentren alguna mejora a las partes de este o que realicen un estudio en el que se haga más hincapié en mejoras de la circulación o puntos críticos de ésta.

Capítulo 2

Extracción de Datos

2.1. Información sobre las imágenes a extraer

Las imágenes que se van a analizar son extraídas desde una página web creada por el ayuntamiento de Sevilla, <http://trafico.sevilla.org>, en la cual se puede ver el estado actual de las carreteras de esta ciudad. Esta web da acceso a ver la última imagen captada por cada cámara con un refresco de cinco minutos, es decir, que cada cinco minutos se van actualizando todas las cámaras con las nuevas imágenes. La ciudad consta actualmente con un total de 66 cámaras situadas de la siguiente forma:

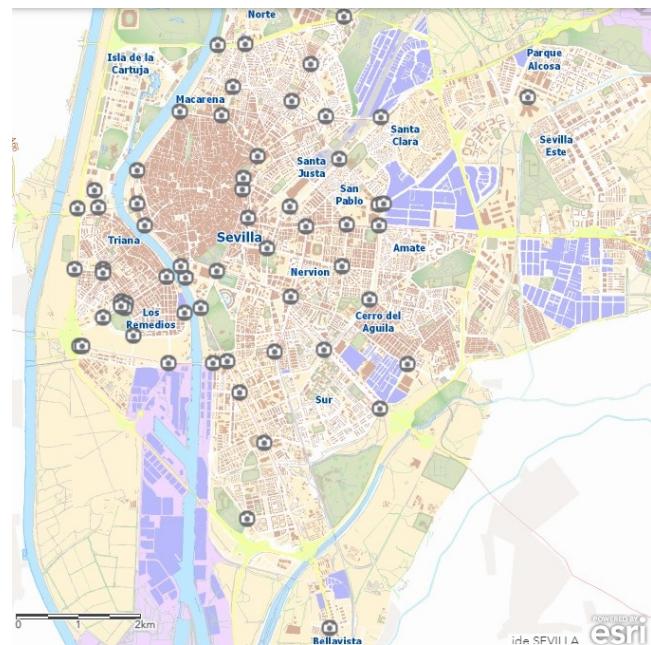


Figura 2.1: Mapa de Sevilla con la localización de las cámaras

Una vez que ya se sabe cuál va a ser nuestra fuente de datos, a continuación se mostrará un ejemplo de como sería una imagen de una cámara de tráfico a analizar.



Figura 2.2: Imagen tipo a analizar

Pero se debe lidiar con una serie de problemas:

- Posición de las cámaras.
- Imágenes con ruido.
- Imágenes nocturnas.
- Cámaras fuera de servicio.

2.1.1. Posición de las cámaras

Las cámaras que se van a analizar no son estáticas, es decir, que la misma cámara puede apuntar a varias zonas, por ejemplo, una cámara que esté en una rotonda puede apuntar a todas las salidas de la misma. El problema es que estos cambios no son periódicos ni con previo aviso.



Figura 2.3: Ejemplo de cámara con varias escenas

En la imagen 2.3, se observa como una misma cámara *Luis Montoto - Luis de Morales - Kansas City* apunta a varias escenas.

2.1.2. Imágenes con ruido

Las imágenes que se van a analizar no son perfectas, es decir, contienen ruido como reflejos producidos por la luz del sol, manchas en la propia lente, e incluso se mueven ya que estas cámaras oscilan debido a la altura a la que están situadas.



Figura 2.4: Ejemplo de imagen con ruido

2.1.3. Imágenes nocturnas

Un problema con el que se tiene que lidiar, son con las imágenes nocturnas, ya que por la noche es más difícil detectar los vehículos por la calidad de las imágenes y además las luces producen reflejos que crean diferencias más significativas a la hora de realizar los cálculos.



Figura 2.5: Ejemplo de imagen nocturna

2.1.4. Cámaras fuera de servicio

Otro de los problemas a los que se debe enfrentar nuestro algoritmo, es detectar cuando una cámara está fuera de servicio para que durante este periodo se tenga constancia de que dicha cámara esté fuera de servicio.



Figura 2.6: Ejemplo de cámara fuera de servicio

2.2. Extracción de Imágenes

El primer paso necesario para la realización de este proyecto, es el proceso de extracción de imágenes desde la web mencionada anteriormente. Para ello, utilizaremos un script desarrollado en *Python* que consta de las siguientes partes:

2.2.1. Librerías necesarias

Para el correcto funcionamiento de este script hemos tenido que utilizar las siguientes librerías: [6]

- *requests* : permite enviar solicitudes *HTTP* con *Python* sin necesidad de tanta labor manual, haciendo que la integración con los servicios web sea mucho más fácil. No es necesario agregar manualmente consultas a las URLs o convertir información a formularios para realizar una solicitud POST. Además, hace más fácil la extracción de partes de la página web, como imágenes, secciones, cabeceras, etc.
- *os* : esta librería permite realizar operaciones dependientes del Sistema Operativo como crear una carpeta, listar contenidos de una carpeta, finalizar un proceso, etc.
- *datetime* : Para manejar fechas en *Python* se suele utilizar esta la librería, que incorpora los tipos de datos *date*, *time* y *datetime* para representar fechas y contiene funciones para manejarlas. Algunas de estas funciones nos ayuda a acceder a los distintos componentes de una fecha, convertir cadenas de *strings* a *datetime*, comparar fechas, extraer la fecha actual, etc.

2.2.2. Funciones Creadas

Se ha creado la función *extraerHora()* la cual, gracias a la librería *datetime*, devuelve la hora del sistema, en el formato deseado, en el momento que se llama a la función.

Gracias a que las URLs donde encontramos las imágenes de las cámaras son del formato *http://trafico.sevilla.org/camaras/cam + NumeroCam + .jpg*, se ha podido automatizar la extracción de estas. Para ello, se ha creado la función *descargaCamara(numeroCam)*, que dado el numero pasado por parámetro, accede a esa web y obtiene la imagen gracias a la librería *requests* devolviendo *True*, en caso satisfactorio o *False* en caso contrario.

La imagen es nombrada por la cámara a la que pertenece seguido de la fecha actual del sistema, extraída gracias a la función *extraerHora()*.

Además, esta función crea, si fuera necesario, la estructura en carpetas donde deben estar guardadas cada imagen, separadas por cámaras.

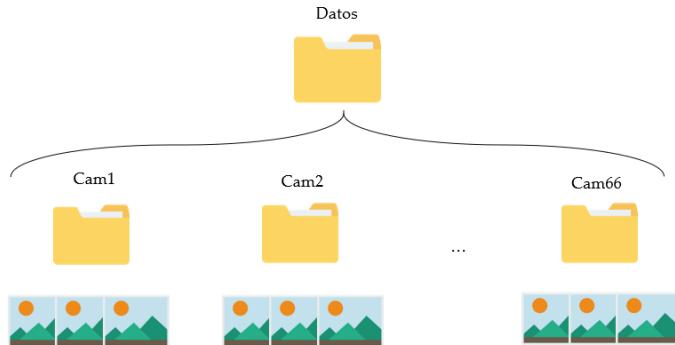


Figura 2.7: Estructura de carpetas

Puede ocurrir que a la hora de extraer una imagen, justo sea el momento el que se actualice la cámara y con ello no exista ninguna imagen en la URL. Esto conlleva que se descargue una imagen vacía, es decir, sin tamaño ninguno por lo que esta función controla que existan este tipo de imágenes y las elimina.

```

1   def descargaCamara(numero):
2       url = "http://trafico.sevilla.org/camaras/cam"+numero+".jpg"
3       page = get(url)
4       if page == 200:
5           path = "./Datos/cam" + numero + "/" + extraerHora() + "_cam" + numero + ".jpg"
6           image = page.content
7           save(image, path)
8           file_size = getSize(path)
9           if file_size == 0:
10              remove(path)
11         end
12       return True
13     else:
14       return False
15   end
16 end
  
```

Este script está creado para que funcione de manera automatizada, y tal y como está explicado hasta el momento, sólo es capaz de descargar imágenes sin parar y con ello sobrecarga de imágenes el servidor o dispositivo donde se ejecute.

Para solucionar esto, se ha creado la función *limpiaCarpeta(camara)*, la cual se encarga de que en cada carpeta no haya más de 10 imágenes, que son las necesarias para realizar el cálculo de la ocupación (9 imágenes para calcular el fondo y 1 imagen que es la de estudio). Para ello siempre elimina la imagen más antigua quedando siempre las 10 más recientes.

2.2.3. Main

En esta sección es donde se pone en común todo lo explicado anteriormente (las librerías y las funciones creadas).

Una vez importadas todas las librerías necesarias, lo único que hace es un bucle en el que se descarga la imágenes de todas las cámaras. Si alguna descarga fuera fallida, o no existiera, se informa de que existe un error en la cámara X y sino, es decir, que la descarga fuera correcta, se invoca a la función *limpiaCarpeta()* para que elimine la imagen más antigua.

```
1  for i in range(1,numCamaras):
2      d = descargaCamara(i)
3      if not d:
4          print("Descarga_fallida_en:", i)
5      else:
6          limpiaCarpeta(i)
7      end
8  end
```

Capítulo 3

Procesamiento y Selección de Características

La finalidad de este capítulo será dada una imagen, calcular la ocupación de la carretera que se observa en ella, Automatizar el procedimiento de extracción de datos y cálculo de la ocupación y por último, preparación de los datos antes de su estudio, ya que hasta ese punto, la ocupación consiste en dar el número de píxeles ocupados en la imagen.

3.1. Procesamiento de las Imágenes

Es el proceso de calcular el porcentaje de ocupación en una carretera. Se basa en un proceso de análisis y procesamiento de imágenes. Para poder obtener la ocupación de la imagen en cuestión, es necesario obtener primero el fondo de esta imagen. Una vez se tenga este fondo, se resta a la imagen de fondo nuestra imagen de estudio y como resultado se obtiene una imagen con la diferencia de estas dos. Esta imagen es necesaria convertirla a una imagen binaria y con ello, se puede observar tanto los píxeles que representan los vehículos que nos interesa como cierto ruido que corresponde al movimiento de la cámara, ya que esta oscila, tiene reflejos causados por la luz del sol, se pueden observar personas que se mueven por la acera incluso el movimiento de los árboles. Para poder desprendernos de este ruido y además saber que parte de la imagen es carretera para poder obtener los píxeles ocupados, se debe calcular la zona de interés, es decir, sólo la parte de carretera que se pueda ver en la imagen.

Por lo que se ha expuesto anteriormente, existen tres puntos clave a analizar antes de calcular la ocupación de las imágenes:

1. Calcular la imagen de fondo.
2. Restar la imagen de estudio con el fondo calculado.
3. Calcular / Extraer las zonas de interés de las cámaras.

3.1.1. Calcular la imagen de fondo

Para calcular la imagen de fondo de nuestra imagen de estudio es necesario tomar las X imágenes consecutivamente anteriores a esta para que el fondo

22 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

sufran el menor número de modificaciones (reflejos de luces, cambios de escena, distintas alturas del sol y con ello distintos niveles de luz, etc).

En este caso, se toma como referencia la imagen de estudio y se recorre hacia atrás hasta que se guarde X fotos anteriores y se van guardando en una estructura o también llamado *Struct*.

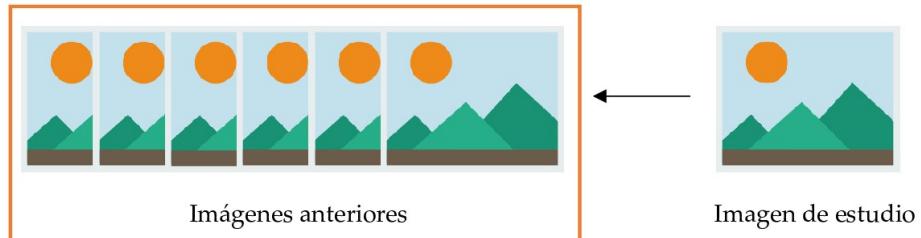


Figura 3.1: Recolección de imágenes anteriores

Antes de continuar con el cálculo del fondo se debe especificar que cada imagen está formada por tres matrices en función del color que represente (*Red*, *Green* y *Blue*). Estás matrices están formadas por tantos píxeles como resolución tenga la imagen y cada uno de ellos indica la intensidad de color que tiene el píxel en cuestión.

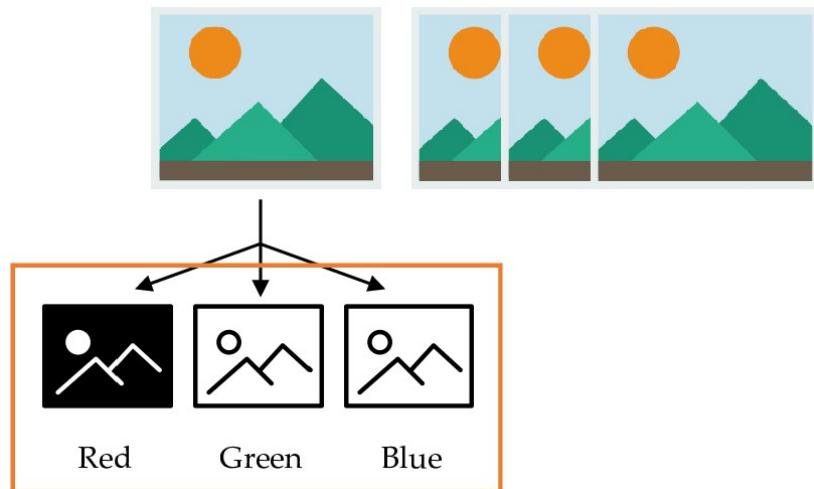


Figura 3.2: Estructura de las imágenes

Una vez obtenidas las X imágenes anteriores a nuestra imagen de estudio, y habiendo comprendido las partes en las que se puede dividir una imagen en color, se procede al cálculo del fondo.

Para ello se ha procedido a calcular la mediana¹ entre matrices que representen el mismo color de cada imagen a nivel de píxel. Se obtiene como resultado

¹Se debe tener en cuenta que el cálculo de la mediana debe ser aplicado sobre vectores que contengan un número impar de elementos, ya que sino, cogería la media entre los dos elementos centrales del vector una vez ordenados.

tres matrices (*Red*, *Green*, *Blue*) donde cada píxel perteneciente a estas matrices, es el resultado de la mediana entre los los píxeles situados en la misma posición de cada matriz que representan el mismo color.

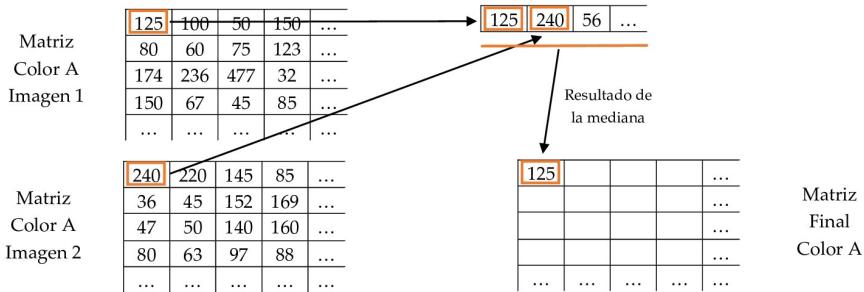


Figura 3.3: Cálculo de mediana a nivel de píxel

Por último se debe concatenar estas tres matrices y se obtendría la imagen de fondo.

<table border="1"> <tr><td>125</td><td>86</td><td>12</td><td>...</td></tr> <tr><td>24</td><td>89</td><td>52</td><td>...</td></tr> <tr><td>32</td><td>12</td><td>30</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> </table>	125	86	12	...	24	89	52	...	32	12	30	<table border="1"> <tr><td>47</td><td>89</td><td>36</td><td>...</td></tr> <tr><td>45</td><td>56</td><td>125</td><td>...</td></tr> <tr><td>250</td><td>245</td><td>230</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> </table>	47	89	36	...	45	56	125	...	250	245	230	<table border="1"> <tr><td>68</td><td>74</td><td>23</td><td>...</td></tr> <tr><td>12</td><td>35</td><td>69</td><td>...</td></tr> <tr><td>45</td><td>23</td><td>98</td><td>...</td></tr> <tr><td>...</td><td>...</td><td>...</td><td>...</td></tr> </table>	68	74	23	...	12	35	69	...	45	23	98
125	86	12	...																																															
24	89	52	...																																															
32	12	30	...																																															
...																																															
47	89	36	...																																															
45	56	125	...																																															
250	245	230	...																																															
...																																															
68	74	23	...																																															
12	35	69	...																																															
45	23	98	...																																															
...																																															
Matriz Final Color A	Matriz Final Color B	Matriz Final Color C																																																

Figura 3.4: Concatenación de matrices finales

Una vez entendido cómo se va a extraer la imagen de fondo, se procede a explicar como se ha implementado en el código. Primero se ha creado tres matrices (una por cada color) y cada una de ellas tiene tres dimensiones (*numPixelAlto X numPixelAncho X numeroImagenesFondo*). Cada matriz de color creada contiene tantas filas y columnas como resolución tenga la imagen y como tercera dimensión, tantas como número de imágenes tenga el *Struct*. El primer paso es hacer un recorrido por el *Struct* descomponiendo las imágenes en sus tres matrices de color e ir asignándolas a la matriz que le corresponda según el color que represente.

Cuando finalice este paso se ha conseguido obtener tres matrices donde cada una de ellas contiene todas las matrices que indican el mismo color de cada imagen.

Además, se ha creado tres matrices finales (una por cada color) donde se irá insertando el valor final de cada píxel.

A continuación se realiza un recorrido por cada píxel de las matrices. En cada paso de este bucle, se asigna en cada posición de las matrices finales el

24 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

resultado de la mediana del *array* compuesto por los píxeles correspondiente a la misma posición y que indican el mismo color. Para terminar, se debe concatenar estas tres matrices para crear la imagen de fondo.

```
1  def calculaFondo(imagenStruct , numImagenes):
2      [f , c , m] = imagenStruct [1]. size ()
3
4      r = zeros(f ,c ,numImagenes)
5      g = zeros(f ,c ,numImagenes)
6      b = zeros(f ,c ,numImagenes)
7
8      for i in range(1 ,numImagenes):
9          I = imagenStruct [i]
10         r (:,:,i) = I (:,:,1)
11         g (:,:,i) = I (:,:,2)
12         b (:,:,i) = I (:,:,3)
13     end
14
15     ifinal_r = zeros(f ,c )
16     ifinal_g = zeros(f ,c )
17     ifinal_b = zeros(f ,c )
18
19     for fila in range(1 ,f ):
20         for columna in range(1 ,c ):
21             ifinal_r (fila ,columna) = mediana(r (fila ,columna ,:))
22             ifinal_g (fila ,columna) = mediana(g (fila ,columna ,:))
23             ifinal_b (fila ,columna) = mediana(b (fila ,columna ,:))
24         end
25     end
26
27     ifondo = concatenar(ifinal_r ,ifinal_g ,ifinal_b )
28     return ifondo
29 end
```

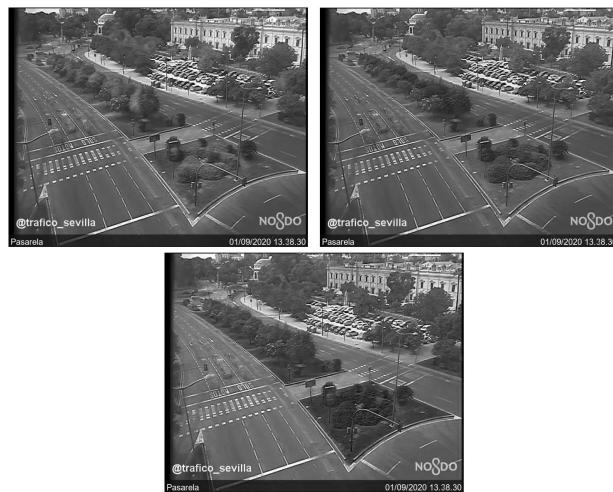


Figura 3.5: Ejemplo de matrices finales por colores (*Red, Green, Blue*)



Figura 3.6: Ejemplo de imagen de fondo

3.1.2. Restar la imagen de estudio con el fondo calculado

Una vez llegado a este punto se tiene las dos imágenes siguientes, donde una de ellas es nuestra imagen de estudio y la otra sería la imagen de fondo calculada mediante el proceso explicado anteriormente.



Figura 3.7: Imagen de estudio y su fondo calculado

A continuación se debe realizar la siguiente operación:

$$\text{Resultado} = \text{Imagen de estudio} - \text{Imagen de fondo}$$

En la que gracias a que las dos imágenes tienen la misma resolución, es decir, la misma cantidad de píxeles, es posible hacer esta resta sin tener que reescalar ni realizar ningún otro tipo de operación.

Como resultado se obtendrá una imagen compuesta por tres matrices en la que se podrá observar cuáles son los puntos donde hay diferencias entre esas dos imágenes.

Para simplificar el proceso se aplica la función *rgb2gray* a esta imagen resultante. Esta función devuelve una imagen en blanco y negro compuesta por sólo una matriz y esta contiene la media entre las tres matrices anteriores.



Figura 3.8: Diferencia en Blanco y Negro

Se puede observar en la imagen 3.8 que los puntos de interés, es decir, los vehículos, tienen un color más claro. Esto es debido a que al realizar la resta, los elementos no existentes en la imagen de fondo adquieren un valor más alto y los elementos que están en las dos imágenes o sufren pocas modificaciones tienen un valor más bajo.

Gracias a que se sabe que el valor de los píxeles que componen la imagen varían entre 1 - 256, se debe realizar una operación de filtrado de manera que los píxeles que sean inferior a un *Threshold* fijado se pongan a 0 y los valores que sean mayor a dicho *Threshold* se pongan a 1. De manera que se tenga una imagen binaria siendo los puntos a 1 los puntos de interés y a 0 el resto de la imagen.

Pero fijar el valor del *Threshold* es una tarea muy delicada ya que si se opta por un valor del *Threshold* muy bajo la imagen resultando adquiere mucha información innecesaria y si se opta por un valor muy alto, se pierde toda la información de la imagen. A continuación se puede ver el efecto que produce la operación de filtrado con varios valores de *Threshold* desde un valor bajo, en este caso 10, hasta un valor alto, que en este caso es 120.

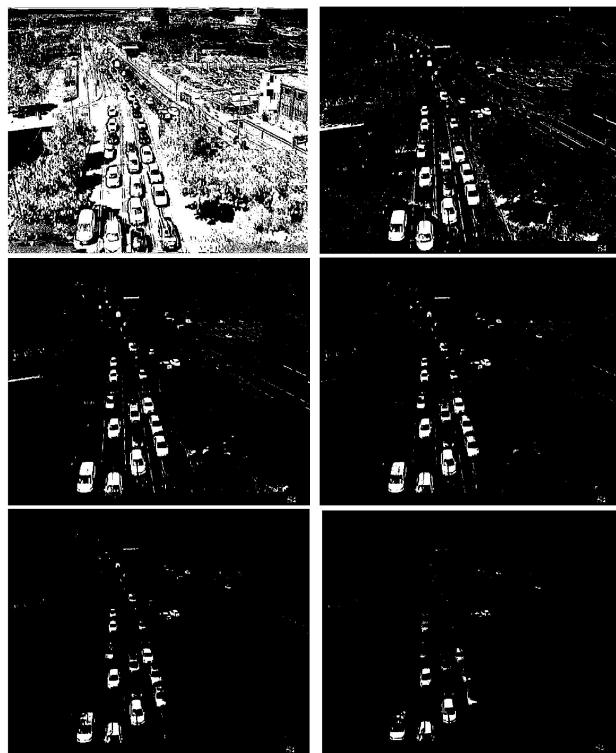


Figura 3.9: Diferentes *Threshold* aplicados al resultado

Tras varias pruebas con diferentes imágenes y cámaras se ha optado que el mejor valor posible para el *Threshold* es 70. Ya que con este valor se obtiene el menor ruido posible sin perder demasiada información necesaria.



Figura 3.10: Resultado final con *Threshold* a 70

28 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

En la imagen 3.10 se obtiene un buen resultado, pero si se realiza esta operación en otra imagen sin tener en cuenta el movimiento de la cámara el resultado podría ser el siguiente:



Figura 3.11: Ejemplo de ruido producido por el movimiento de la cámara

En la imagen 3.11 se puede ver el ruido producido por el movimiento oscilatorio de la cámara, es decir, ya que la cámara se balancea debido al viento, la posición de los edificios y de los límites de la carretera pueden variar de la imagen de fondo con respecto la imagen de estudio por lo que al realizar la resta, nos resaltaría como diferencia significativa.

Para poder suprimir este ruido mencionado, se ha diseñado la función *encuadrarImagen(I, iFondo, numPixelos)*. Lo que realiza esta función es ir superponiendo la imagen de estudio sobre la imagen de fondo, la resta y almacena ese resultado, a continuación desplaza la imagen de estudio lateralmente y/o horizontalmente, la vuelve a restar, y así sucesivamente. La función realiza desplazamientos a nivel de filas y columnas. El número de desplazamientos viene definido como *numPixelos* se le pase por parámetro, es decir, si por ejemplo se le pasa como valor 4, va a realizar un máximo de cuatro desplazamientos laterales y cuatro horizontales, sería un total de 16 desplazamientos. Siendo el mejor resultado para esta función el que contenga el menor numero de puntos de interés.

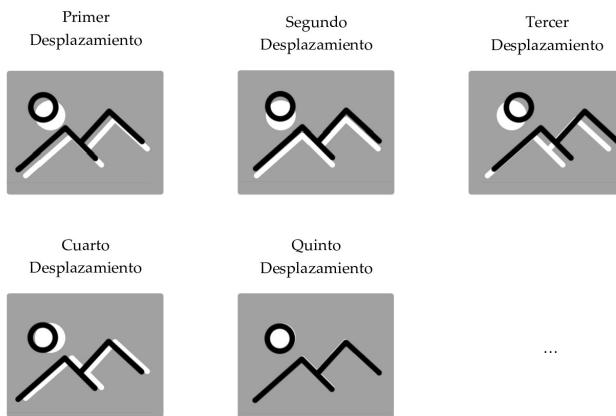


Figura 3.12: Función *encuadrarImagen*

Tras aplicar esta función, la imagen resultante es la siguiente:



Figura 3.13: Ejemplo de eliminación de ruido con *encuadrarImagen*

Por lo que la función *encuadrarImagen* engloba el proceso de restar la imagen de estudio con la imagen de fondo, aplicarle la función *rgb2gray*, realizar la operación de filtrado con el *Threshold* fijado, además de suprimir el movimiento oscilatorio de la cámara.

Para poder implementar esta función, lo primero que se ha realizado ha sido recortar la imagen de fondo *numPixelles* por cada lado, es decir, tanto lateralmente como horizontalmente, de manera que la imagen resultante tiene *numPixelles * 2* filas y columnas menos.

A partir de aquí, se realiza un bucle, en el que en cada iteración, a la imagen de estudio se va eliminando desde 0 hasta *numPixelles * 2* por cada lado tanto en filas, como en columnas y entre cada iteración se realiza la resta y se guarda el resultado.

Por ejemplo, en la primera iteración por la izquierda no se elimina ninguna columna pero por la derecha se elimina *numPixelles * 2* y lo mismo ocurre en las filas, es decir, no se elimina ninguna fila por arriba, pero por abajo se elimina *numPixelles * 2* filas, una vez que se tenga recortada nuestra imagen de estudio, se realiza la resta con la imagen de fondo y se guarda el resultado si fuera mejor que el anterior. Este proceso va incrementando las filas y/o columnas que se eliminan por un lado y por el otro va reduciendo las filas y/o columnas que se eliminan, quedando la imagen siempre con el mismo tamaño.

```

1  def encuadrarImagen(I, ifondo, numPixelles):
2      [f, c, m] = ifondo.size()
3      ifondo2 = ifondo(numPixelles:f-numPixelles, numPixelles:c-
4          numPixelles,:)

```

30 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

```
5     desplazamiento = numPixelas * 2
6     maximo = sum(ifondo2 (:))
7
8     for i in range(1, desplazamiento):
9         for j in range(1, desplazamiento):
10             Iprueba = I(i:(f-(desplazamiento-i)),j:(c-
11                             desplazamiento-j)),:,:)
12             res = Iprueba - ifondo2
13             res2 = rgb2gray(res)
14             res3 = res2 > Threshold
15             if(sum(res3 (:)) < maximo)
16                 mejorResta = res3
17                 maximo = sum(res3 (:))
18             end
19         end
20     end
21
22     return mejorResta
```

Igualmente, se aplica una operación de filtrado para eliminar todas las agrupaciones menores a un tamaño X , en este caso 15. Con esto se consigue eliminar puntos que aportan información innecesaria para nuestro estudio como coches muy al fondo o incluso ruido de la propia imagen.

```
1     imagenResultante = bwareaopen(mejorResta, 15)
```



Figura 3.14: Ejemplo de resta sin ruido

3.1.3. Calcular / Extraer zonas de interés de las cámaras

El resultado obtenido en la imagen 3.14 parece bastante bueno para continuar con el cálculo de la ocupación, en contraposición, se puede obtener como resultado final lo expuesto en la imagen 3.15 en la que se observa la carretera vacía. Además, se han tomado como puntos de interés mucha información complementaria la cual hará que este cálculo de ocupación sea erróneo.



Figura 3.15: Ejemplo de imagen resultante con ruido

En el momento que se consiga reconocer en una imagen qué píxeles pertenecen a la carretera, se podrá tomar la información referente a estos píxeles de la imagen binaria. De modo que se evita todo el ruido contenido en la misma debido a parkings cercanos a la carretera, personas esperando en paradas de autobuses, reflejos del sol que no tapan la carretera, ...

Para poder detectar que parte de la imagen es carretera, se ha desarrollado dos algoritmos diferentes. La diferencia fundamental entre ellos es el hecho de que uno lo realiza un humano y la otra lo realiza automáticamente el sistema.

- Extracción de plantilla de forma manual.
- Extracción de plantilla de forma automática.

Extracción de plantilla de forma manual

El proceso de extracción de plantillas de forma manual es muy sencilla. Lo primero que se debe hacer es localizar cada una de las posibles escenas que tiene la cámara, es decir, coger una imagen de cada escena de la misma cámara.

En el momento que se tengan localizadas, gracias a la función $roipoly(I)$ se puede seleccionar la zona de interés en cada imagen.



Figura 3.16: Ejemplo de selección con $roipoly(I)$

32 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

Como se puede observar en la imagen 3.16 se ha seleccionado el contorno ² de la carretera siendo el interior de esta forma la zona de interés.

Esta función nos devuelve la siguiente matriz binaria:



Figura 3.17: Resultado de selección con $roipoly(I)$

Este proceso se debe realizar con cada escena de cada cámara, lo que conlleva un tiempo tanto de búsqueda de cada escena, como la selección de cada zona. Además se debe tener en cuenta que esta herramienta sólo deja seleccionar una zona por imagen por lo que se debe tener cuidado a la hora de seleccionar por ejemplo una imagen con dos carreteras ya que las dos deben pertenecer a la misma forma (imagen 3.18).

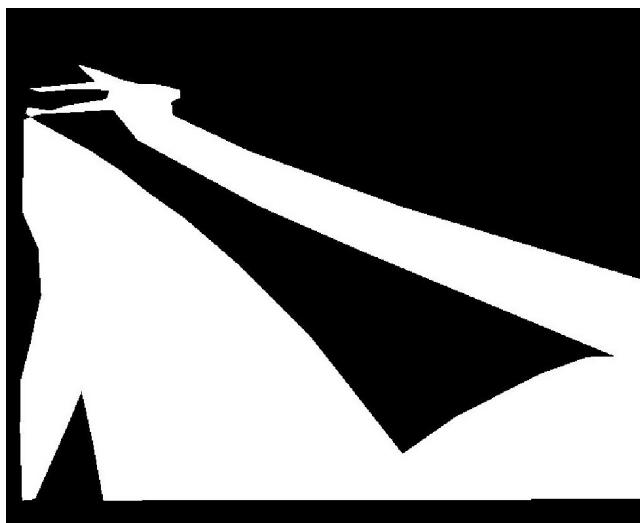


Figura 3.18: Selección de $roipoly(I)$ de varias zonas

²Se debe tener en cuenta el hecho de dejar un margen entre la carretera y la línea que delimita la zona de interés marcada por nosotros, por el hecho de que las cámaras oscilan y con este margen no se pierde nada de información y si se cogiera ruido, en este caso, de la acera sería casi inapreciable

Un ejemplo de uso de esta plantilla sería la imagen 3.19 en la que se puede observar como se ha recortado perfectamente tanto la imagen de estudio y la imagen de fondo. Con ello se elimina todo el ruido externo a la carretera y ya se sabe el total de píxeles que pertenecen a ella.

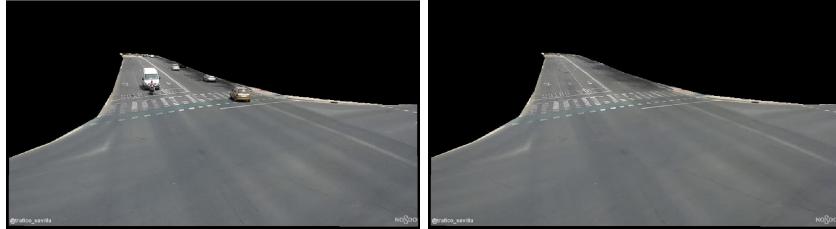


Figura 3.19: Empleo de plantillas extraídas con la función *roipoly()*

Para intentar hacer este proceso un poco más liviano, se ha automatizado, pero para ello necesitamos un conjunto de imágenes de cada cámara, es decir, una semana de capturas donde dentro estén todas las escenas posibles.

Este algoritmo de automatización recorre todas las imágenes de la carpeta que contiene el conjunto de imágenes de la cámara que queremos crear la plantilla e indica si hay diferencia significativa para sacar la plantilla de cada imagen.

El procedimiento que sigue este algoritmo, comienza siempre obligando a que realices la plantilla de la primera imagen de cada cámara (si está disponible). A partir de ahí, con las siguientes imágenes, se llama a la función *plantillaNecesaria(ImagenActual, ImagenAnterior, cam)*, la cual extrae todas las plantillas que se haya recortado de la cámara *cam* y realiza la resta entre la *ImagenActual* y *ImagenAnterior*. Teniendo todas las plantillas de esta cam y la imagen resultante de la diferencia entre las imágenes, superponemos a cada plantilla esta imagen resultante, de esta manera se debe obtener como resultado tantas imágenes como plantillas tenga esta cam, donde en estas imágenes los puntos que resaltan son la diferencia de las dos imágenes pero sin tener en cuenta la carretera, es decir, el entorno. A partir de aquí, se queda con la imagen que minimice esta diferencia.

Llegados a este punto, se pueden dar varias situaciones:

1. Si las dos imágenes tienen la misma escena pero no hay plantilla de esta escena, la diferencia entre ellas sería significativa debido a la carretera, ya que los coches si varían y no habría plantilla que los cubra, y el mismo algoritmo te recomendaría hacer la plantilla de esta imagen.
2. Si las dos imágenes tienen la misma escena y hay plantilla de esta escena, la diferencia entre ellas no sería significativa ya que la escena es la misma, y la plantilla cubre la carretera.
3. Si las imágenes son de escenas distintas, es decir, es un cambio de escena, esta operación daría como resultado una diferencia significativa, ya que si hubiera o no plantilla de esta escena, no podría cubrir todo, además de la diferencia del entorno, por lo que en este caso se dejaría la decisión de extraer o no la plantilla al usuario.

```

1   myPath = "D:\Imagenes_TFG\cam" + numCam
2   allPhotos = listdir(path)
3   dif = 100
4   numPlantillas = 1
5
6   for k in range(0, len(allPhotos)):
7       P = imread(allPhotos[k])
8       disponible = camDisponible(P)
9       if (disponible):
10           if (k > 1):
11               pAnterior = imread(allPhotos[k-1])
12               if(camDisponible(pAnterior)):
13                   dif = plantillaNecesaria(P, pAnterior, "cam" +
14                     str(cont))
15               else:
16                   dif = 100
17               end
18               if(dif > 2):
19                   I_Plantilla = roipoly(P)
20                   save(I_Plantilla)
21               end
22           end
23       end

```

Extracción de plantilla de forma automática

La creación de plantillas de forma automática básicamente consiste en el hecho de ir superponiendo una serie de imágenes resultantes de la diferencia entre el fondo y la imagen de estudio, es decir, se tiene la imagen de estudio, se calcula el fondo y se guarda la diferencia entre estas (imagen 3.14), se coge la siguiente imagen de estudio y el resultado de la diferencia con su fondo se suma al resultado de la primera imagen, y así sucesivamente. De manera que consiga rellenarse el mayor numero de píxeles de carretera posible. Para este método también es necesario tener un conjunto de imágenes de cada cámara.

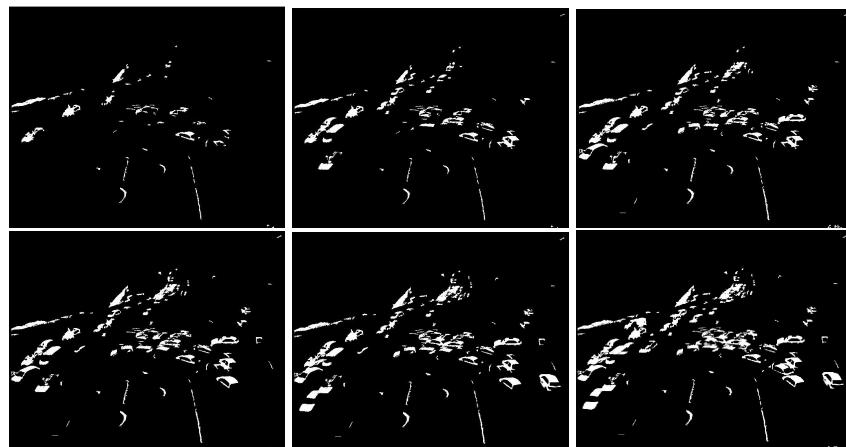


Figura 3.20: Proceso de creación de plantilla automática

Se puede ir observando en la imagen 3.20 como poco a poco se va rellenando las zonas de interés con cada imagen que se va superponiendo, hasta que se llegue a la siguiente plantilla.

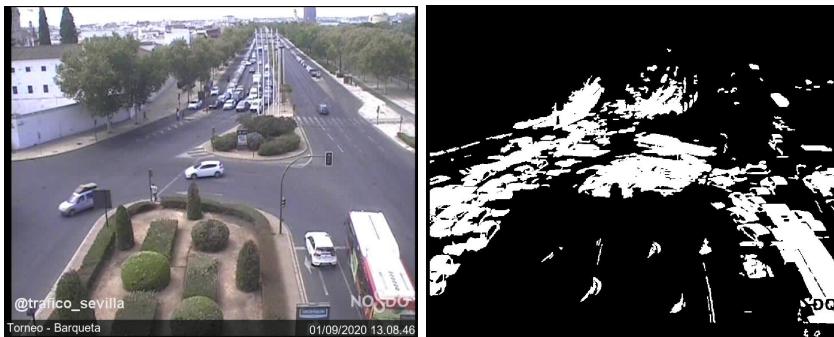


Figura 3.21: Ejemplo de creación de plantilla automática

No se debe suponer que mientras más imágenes se analicen y se superpongan se obtendrá mejor resultado ya que se consiguen llenar más los píxeles de carretera, pero también es posible que se añada a nuestra plantilla ruido como reflejos, personas esperando, u otro tipo de ruidos que afectaría de forma negativa nuestro resultado, como en el siguiente caso en el que se superpusieron 75 imágenes.



Figura 3.22: Ejemplo de creación de plantilla automática con exceso de ruido

Tras varias pruebas se llegó a la conclusión que el mejor resultado en relación píxeles de interés / ruido son 50 imágenes. En la imagen 3.21 se puede observar que rellena bastante bien la carretera, pero se pueden diferenciar los carriles ya que entre coches hay huecos y no queda del todo bien. Para solucionar este fallo hemos utilizando la función *imclose* la cual realiza un cierre morfológico.

La operación de cierre morfológico dilata una imagen y, a continuación, erosiona la imagen dilatada, utilizando el mismo elemento de estructuración para ambas operaciones. El cierre morfológico es útil para llenar pequeños huecos de una imagen conservando la forma y el tamaño de los objetos de la imagen.

Tras aplicar la función de cierre morfológico a la imagen 3.21 nos queda el siguiente resultado final de la creación de plantillas automáticas.

36 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS



Figura 3.23: Ejemplo de aplicación de cierre morfológico

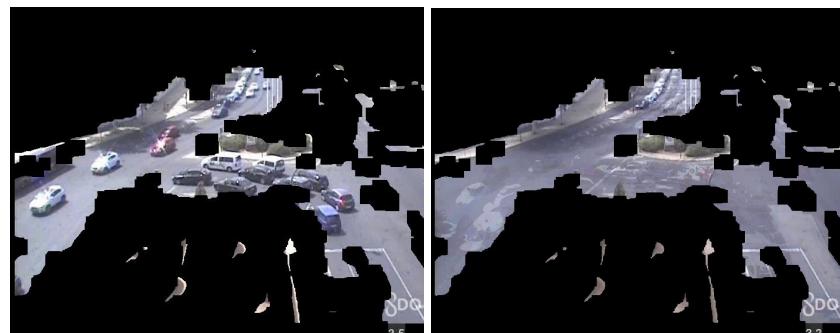


Figura 3.24: Empleo de plantillas extraídas automáticamente

```
1 allPhoto = listdir(path)
2 P = imread(allPhoto[0])
3 [f, c, m] = P.size()
4 plantilla = logical(f,c)
5 for i in range(numImagenes, 50):
6     I = imread(allPhoto[i])
7     ifondo = calculaFondo(I)
8     imagenResultante = encuadrarImagen(I, ifondo, numPixel);
9     imagenFiltrada = bwareaopen(imagenResultante,15);
10    plantilla = plantilla + imagenFiltrada;
11 end
12 cierreMorfologico = imclose(plantilla);
13 save(plantilla)
```

Diferencias

Tras analizar los resultados obtenidos, se ha optado por usar el método de extracción de plantilla de forma manual, ya que los resultados obtenidos con el método automático no satisfacen un mínimo de calidad para todas las cámaras.

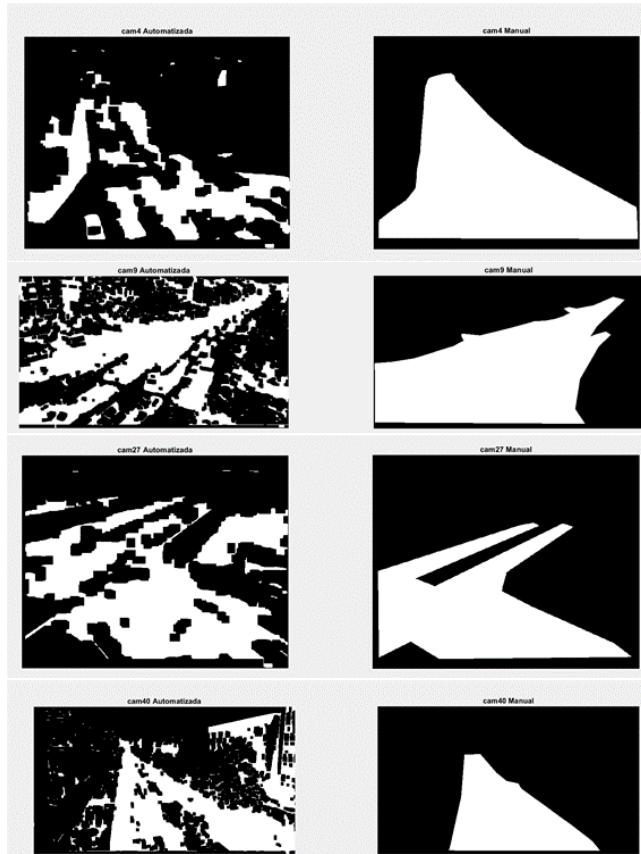


Figura 3.25: Comparación de métodos de extracción de zonas de interés (1)

Aunque la extracción de plantillas de forma manual, como su propio nombre indica, es necesario que un humano este presente para cada cámara, que aproximadamente se tarda unos 10 - 15 minutos por cámara, para la extracción de plantillas de manera automatizada, también es necesario el trabajo previo por un humano para clasificar las imágenes de una cámara para que no existan cambios de escena durante el procedimiento del cálculo, ya que si esto ocurriera destruye totalmente estas plantillas.

Los resultados obtenidos por la extracción automática, como hemos comentado anteriormente, no nos sirve para nuestro estudio como podemos ver en la imagen 3.25. Ya que no rellena la carretera que es nuestra mayor objetivo, además de adquirir mucha información externa a esta, proveniente de reflejos o cambios de luces de edificios, árboles, movimiento de personas por la acera, ...

3.2. Selección de características

En esta sección se extraerá la información sobre la ocupación de las imágenes con la ayuda de las técnicas explicadas en el capítulo anterior.

Se debe aclarar que, aunque se ha usado *Matlab* [7] para desarrollar el prototipo del mecanismo de extracción de datos y para el procesamiento y selección de características del proyecto, para desarrollar la versión final que se automatizará en el servidor, se ha utilizado el lenguaje *Python* [8].

Se partirá de una carpeta donde estarán todas las cámaras con sus respectivas imágenes, con la estructura vista en la imagen 2.7, y como resultado final de cada imagen analizada, se obtendrá una tupla con la cámara a la que pertenece, la fecha y la hora y la ocupación de esta. Esta tupla se podrá añadir a un archivo *.csv*, el cual si no está creado se crea automáticamente, o también se podrá añadir este resultado una base de datos (esta técnica es la que está activa en el script). Al final el objetivo es tener un contenedor, el cual incrementando con cada imagen analizada, creando así un histórico.

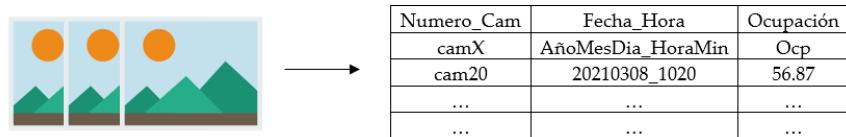


Figura 3.26: Resultado del cálculo de la ocupación

3.2.1. Contenedor de datos

El primer paso que se realiza antes de nada, es elegir dónde se van a guardar los datos extraídos, ya sea en una base de datos creada con anterioridad o en un archivo *.csv*.

Conexión a la base de datos

Si se opta por la opción de la base de datos, se debe tener en cuenta de que esta base de datos tiene que estar creada para que la conexión de este script sea correcta.

Como primer paso, gracias a la librería *MySQLDb*, se tiene que realizar la conexión a esta base de datos con la función *connect()*, a la que se le pasa por parámetros la dirección del servidor, usuario, contraseña y base de datos a la que quieras conectarte ya que en un mismo servidor podemos tener varias bases de datos.

¹ db = MySQLDb.connect("localhost", "user", "pass", "db_name")
² cursor = db.cursor()

Una vez que se establezca la conexión con el servidor exitosamente, se tiene que asegurar que las tablas donde se inserten los datos existan. Esto se realiza con la consulta *CREATE TABLE IF NOT EXIST*, la cual crea la tabla si no existe.

En este caso, se van a tener dos tablas, una para los datos extraídos satisfactoriamente, llamada *Datos*, y otra para las imágenes que den algún tipo de error, llamada *Log*. Además, predefinimos las querys a realizar (los *INSERT*) de tal manera que sólo se deberán añadir los parámetros, lo que hace esta operación más sencilla.

```

1   cursor . execute ( "CREATE _TABLE _if _not _exists _Datos ( Camara _TEXT ,_
2   Date _TEXT , _Ocupacion _TEXT ) " )
3
4   cursor . execute ( "CREATE _TABLE _if _not _exists _Log ( Camara _TEXT ,_
5   Date _TEXT , Descripcion _TEXT ) " )
6
7   mySql_insert_Datos = " INSERT _INTO _Datos _VALUES ( %s , %s , %s ) "
8   mySql_insert_Log = " INSERT _INTO _Log _VALUES ( %s , %s , %s ) "

```

Creación / Lectura archivo .csv

En el caso de que se elija la opción de guardar los datos en un fichero. Se debe estar seguro que el archivo exista. Por esto, encapsulamos el procedimiento en un *Try-Catch*.

El funcionamiento es el siguiente, gracias a la función *obtieneFichero()*, intenta leer el fichero llamado *Datos.csv*, si este existe, devuelve los datos que contenga y se guardan en la variable *datos*, si en otro caso, este archivo no existe, da un error y salta la excepción, la cual llama a la función *crearFichero()* y vuelve a llamar a la función *obtieneFichero()* para recoger los datos, en este caso el fichero sólo contiene el nombre de las columnas.

```

1   def obtieneFichero () :
2       datos = open ( "./ Datos . csv " )
3       return datos
4
5   def crearFichero () :
6       myData = [ "Cam" , "Date" , "Ocp" ]
7       myFile = createFile ()
8       myFile . insert ( myData )
9       save ( myFile , "./ Datos . csv " )
10
11  def main () :
12      try :
13          datos = obtieneFichero ()
14      except :
15          crearFichero ()
16          datos = obtieneFichero ()

```

3.2.2. Extracción de plantillas

Una vez esté el sistema conectado a la base de datos o haya cargado el fichero, el siguiente paso es recorrer cada una de las carpetas, las cuales contienen las imágenes de cada cámara. Para ello se realiza un bucle *for* desde 1 hasta el total de cámaras que se tenga, en este caso 66.

En cada iteración, y gracias a que la estructura de las carpetas siguen la directriz *./Datos/cam + iteracionActual*, se tiene acceso de manera automatizada a cada cámara.

Con la librería *os*, se puede listar todos los objetos que se encuentren en una carpeta. Esto es necesario para saber cuantas imágenes contiene la carpeta y si son las mínimas necesarias para realizar el análisis. Se debe tener en cuenta que cuando se obtenga todos los elementos de la carpeta, esta lista debe estar ordenada de menor a mayor para que en la posición 0 esté la imagen más antigua y por consecuente en la última posición se encuentre la imagen más nueva, que sería la de estudio.

Una vez que se compruebe que en la carpeta hay un mínimo de imágenes para realizar el estudio, el siguiente paso es obtener las plantillas pertenecientes a nuestra cámara actual gracias a la función *getPlantillas()*, a la que se le pasa por parámetro la concatenación del string *cam* y el número de la iteración actual. Estas plantillas como explicamos en el capítulo anterior, muestran la zona de estudio de una imagen, y una misma cámara puede tener varias plantillas.

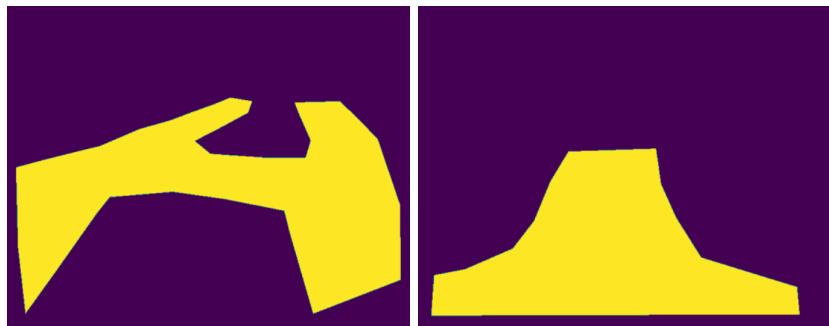


Figura 3.27: Ejemplo de cámara con diferentes plantillas

La función mencionada antes, *getPlantillas()*, es la encargada de acceder a la carpeta donde tenemos ya guardadas todas la plantillas extraídas de forma manual pertenecientes a la cámara actual que se encuentran en el directorio *./Plantillas/cam + iteracionActual*. Esta función lee todos los elementos de esa ruta y al tener todos el mismo tamaño, va concatenando todas las plantillas. Con lo que tiene que devolver una matriz de tres dimensiones, tantas filas y columnas como resolución tenga la imagen, y como tercera dimensión tantas plantillas como tenga la cámara de estudio actual.

```

1  def getPlantillas(cam):
2      carpeta = "./ Plantillas/" + cam
3      listPlantillas = listdir(carpeta)
4      plantillas = []
5      for i in range(0,len(listPlantillas)):
6          variable = load(carpeta + "/" + listPlantillas[i])
7          plantillas.add(variable)
8      end
9  return plantillas

```

El siguiente punto a realizar no es necesario para el correcto funcionamiento de este script, pero si para su eficiencia. Este punto sería redimensionar las imágenes grandes, de manera que a la hora de realizar operaciones con estas sean menos costosas y más rápidas de realizar. Además, como la imagen a analizar tiene la misma dimensión que las plantillas, asignamos *True* o *False* a una variable para saber si mas adelante se debe redimensionar tanto la imagen de estudio como el fondo calculado.

3.2.3. Lectura de imagen y creación de fondo

Llegado este momento, es necesario encapsular todo el proceso que ocurre a continuación en un *Try-Catch* para evitar que cualquier error pare el proceso. Además, se puede recoger el error y avisar para que se revise y se solucione.

Lo siguiente sería leer la imagen de estudio, que gracias a tener ordenado la variable que contiene el listado de la carpeta, sabemos que es la última posición, y gracias a que creamos una variable para redimensionar las plantillas, sabemos si debemos redimensionar esta imagen de estudio para que cuando se realice las operaciones entre las plantillas y la imagen no ocurra ningún error debido a que tienen diferentes dimensiones.

También gracias al nombre de la imagen, se puede sacar la información sobre la fecha y hora a la que se tomó, y poder a nuestro contenedor de datos.

Debido a un asunto comentado en un capítulo anterior, cabe la posibilidad de que existan cámaras que en un momento determinado estén no disponibles. Para poder detectar estos casos, se ha creado una función la cual resta la imagen de estudio con la imagen tipo. Dicha imagen tipo es la que se muestra cuando la cámara no esta disponible (Imagen 2.6) que se ha guardado previamente en una carpeta para poder utilizarla siempre para esta función.

Si existe diferencia entre ellas sabemos que la cámara está disponible y se continua con el procedimiento, pero en caso contrario, que no exista diferencia entre ellas, se opta por devolver que la cámara esta no disponible por lo que no se realiza ningún tratamiento sobre ella para el calculo de la ocupación.

```

1  def camDisponible(iEstudio):
2      disp = False
3      noDisponible = imread(' ./ No_Disponible/NO_DISPONIBLE.jpg ')
4      dif = iEstudio - noDisponible
5      if sum(dif [:]) == 0:
6          disponible = False
7      else:
8          disponible = True

```

```

9      end
10     return disponible

```

En este punto se sabe si la cámara esta disponible o no, pero puede ocurrir que la cámara acaba de ser reparada y las fotos anteriores, con las que se debe calcular el fondo, no están disponibles. Por lo que hemos creado una función llamada *isFondoDisponible()*, a la que le pasamos la cam actual y la lista de las imágenes para el fondo, la cual, revisa estas imágenes y comprueba si hay alguna no disponible. El funcionamiento es básicamente ir leyendo todas las fotos una a una y llamar a la función *camDisponible()* pasándole la imagen leída, y si hay alguna que devuelva que no esta disponible, anulamos el proceso ya que el fondo no se podría calcular correctamente.

```

1  def isFondoDisponible(cam, listImagenes):
2      disponible = True
3      encontrado = False
4      i = 0
5      while i < (len(listImagenes)) and encontrado == False:
6          img = imread("./Datos/" + cam + "/" + listImagenes[i])
7          disp = camDisponible(img)
8          if disp:
9              i += 1
10         else:
11             encontrado = True
12             disponible = False
13         end
14     end
15     return disponible

```

Una vez que se sepa que la cam está disponible y que en las imágenes que se utilizarán para calcular el fondo no hay ninguna que no esté disponible, se pasa a calcular el fondo de la manera que se explico en el capítulo anterior. Pero se ha añadido una mejora para mejorar el servicio del sistema.

Se ha descubierto un patrón que siguen todas las cámaras cuando van a dejar de estar disponible, este patrón es que muestra la misma imagen, es decir, no actualiza la imagen y muestra la última imagen que tenga en el buffer y no se actualiza. Este error se puede detectar en poco tiempo por nuestra parte pero, en el caso del sistema del Ayuntamiento de Sevilla, hay casos en que se ha llevado días mostrando la misma imagen, es decir, sin actualizar la cámara ya que estaría fallando y no se habían dado cuenta de que estaba la cámara averiada. Esto produce que se obtengan datos erróneos durante el tiempo que la cámara este averiada y no se haya puesto en no disponible. Por lo que hemos implementado un servicio en nuestra función *calculaFondo()*, la cual cuando descubre que en el fondo existen tres o más imágenes exactamente iguales avisa por correo electrónico a la dirección que nosotros queramos indicando que la cámara *X* es posible que sufra alguna avería, que debe ser revisada.

El aviso por correo electrónico está implementado gracias a la librería *smptlib* y se realiza llamando a la función *enviaCorreo()* a la que se le debe pasar por parámetro el asunto y el mensaje. Esta función establece una conexión *smpt* con el servidor del correo electrónico, en este caso gmail, y consigue enviar un

correo electrónico a la dirección deseada.

```

1  def enviaCorreo(asunto , mensaje):
2      MY_ADDRESS = "*****"
3      PASSWORD = "*****"
4
5      s = smtplib.SMTP(host='smtp.gmail.com' , port=587)
6      s.starttls()
7      s.login(MY_ADDRESS, PASSWORD)
8
9      message = mensaje
10
11     msg[ 'From ']=MY_ADDRESS
12     msg[ 'To ']= "AnalisisCamarasSevilla@gmail.com"
13     msg[ 'Subject ']= asunto
14
15     s.send_message(msg , message)
16     s.quit()

```

Por último, calculamos el fondo y gracias a la misma variable con la que redimensionamos, si hubiera sido necesario, la imagen de estudio, decidimos si fuera necesario redimensionar o no la imagen de fondo calculada.

```

1  try:
2      iEstudio = imread(carpeta+ "/" +listImagenes[-1])
3      if redimensionar:
4          iEstudio = redimensionaImagen(iEstudio)
5      end
6
7      nombre = listImagenes[-1].split("_")
8      fecha_Hora = nombre[0]+nombre[1]
9
10     disponible = camDisponible(iEstudio)
11
12     if disponible:
13         if isFondoDisponible(cam, listImagenes):
14             fondo = calculaFondo(cam, listImagenes)
15             if redimensionar:
16                 fondo = redimensionaImagen(fondo)
17             end

```

3.2.4. Cálculo de la ocupación

Llegados a este punto, se tiene todas las plantillas que pertenecen a la cámara, el fondo calculado y la imagen de estudio. El siguiente paso sería calcular la ocupación, para ello se ha creado la función *getOcupacion()*, a la que se le debe pasar por parámetro las plantillas, el fondo calculado y la imagen de estudio.

Esta función debe devolver el numero de píxeles distintos en función del fondo y sólo de la zona de interés demarcada por las plantillas.



Figura 3.28: Ejemplo de plantillas, imagen de estudio e imagen de fondo

```
1     ocupacion = getOcupacion(plantillas, fondo, iEstudio)
```

El primer paso que realiza esta función es llamar a la función *encuadraImagen()* para que realice la resta entre la imagen de fondo y la imagen de estudio de la manera que explicamos en el capítulo anterior.

Como se pudo ver en ese mismo capítulo, se le tuvo que recortar a la imagen un número *X* de píxeles por cada lado para poder aplicar la función *encuadraImagen()*. Como consecuencia también se le deben quitar el mismo numero de píxeles y de la misma forma (por cada lado) a las plantillas para que a la hora de realizar operaciones entre ellas tengan el mismo tamaño.

Llegados a este momento, se encontró con el problema de que en la imagen no hay profundidad, es decir, que un píxel en la parte de arriba de la imagen tiene el mismo peso que uno en la parte inferior, cuando esto no es real porque en la imagen 3.29, podemos ver que al fondo un coche tiene 10 píxeles y, en cambio, en la parte inferior, que es cuando el coche está más cerca de la cámara, el mismo coche esta constituido por 100 píxeles.



Figura 3.29: Problema de profundidad

Para solucionar este problema, se encontró la solución de dividir la imagen en franjas y darle un valor a cada píxel en función de la franja a la que pertenezca.

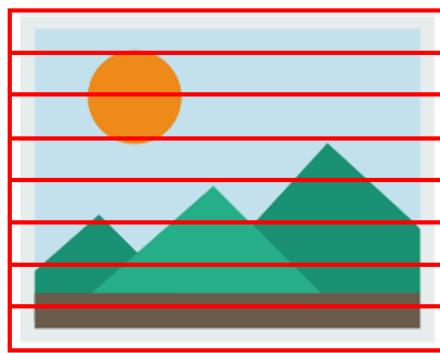


Figura 3.30: División de imagen en franjas

De esta forma podemos incrementar el valor de un píxel en la zona superior de la imagen y/o decrementar su valor cuando este píxel pertenezca a la parte inferior de esta y además de manera gradual con la simple operación de multiplicar por una constante en cada franja.

Cuando se llega a esta situación, se tiene una imagen resultante de la diferencia entre las dos imágenes y una serie de plantillas que pertenecen a la cámara pero sólo una de estas plantillas es la que encaja perfectamente en la visión de esta imagen.

Para solventar esta elección, se realiza una operación *AND* entre la imagen de estudio y cada plantilla. De manera que como resultado tendremos tantas imágenes lógicas como plantillas tengamos y en las que aparecerán sólo los píxeles que estén resaltados en las dos imágenes (imagen de fondo e imagen de estudio).

Esta operación se realiza para quedarnos sólo con los píxeles que estén en la zona de interés existentes en la imagen de estudio. Por ejemplo, se tiene la siguiente imagen de estudio y las siguientes plantillas.

46 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

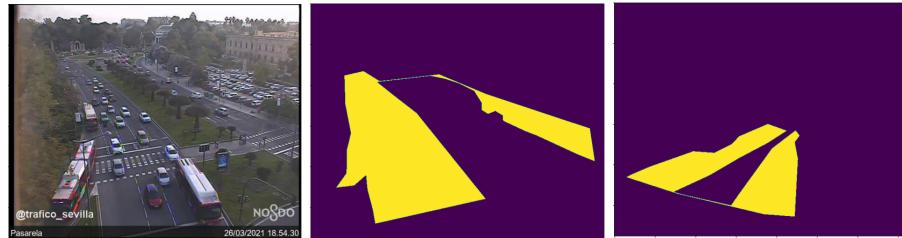


Figura 3.31: Ejemplo de Elección de plantillas

Al realizar esta operación obtenemos los siguientes resultados:

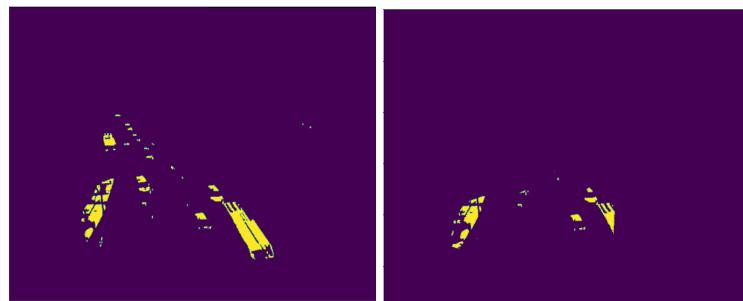


Figura 3.32: Ejemplo de Elección de plantillas (2)

Se puede observar, que en la plantilla que calza correctamente con la imagen de estudio se pueden ver todos los vehículos, en cambio, en la otra sólo se pueden ver los vehículos que coinciden con la plantilla que está extraída desde otro punto de vista de la cámara.

Simplemente se debe elegir la imagen resultante que maximice el número de píxeles que se muestran en la imagen, con esta forma, siempre se elegiría la plantilla que mejor case con la imagen de la cámara actual ya que recogerá el número máximo de coches.

Esta forma de elegir plantillas siempre funciona correctamente ya que en todas las cámaras de este estudio, los puntos de vistas captados desde una cámara siempre tienen ángulos opuestos. Con lo que siempre tendrá con ciertas plantillas un resultado muy bajo y con otra un resultado muy alto.

En el único caso que esto no se cumple sería si nos encontrásemos en una rotonda, pero en este caso, todos los puntos de vista tienen la misma forma, por lo que elegir una plantilla u otra tendría el mismo resultado o uno muy similar.

Una vez llegados a este punto, se tiene el numero de píxeles ocupados, que es el sumatorio de la imagen elegida.

```

1  def getOcupacion(plantillas , fondo , iEstudio):
2      numPixel = 3
3      diffFiltrado = encuadraImagen(iEstudio , fondo , numPixel)
4
5      ocupacion = []
6      for i in range(0, len(plantillas)):
7          I_Plantilla = plantillas [ i ];

```

```

8     I_Recortada = I_Plantilla AND diffFiltrado
9
10    [ fila , columnas , matriz ] = I_Recortada . size ()
11
12    intervalo = round( fila / numeroFranjas );
13
14    I_Recortada [ 1 : intervalo , : ] = I_Recortada [ 1 : intervalo , : ]
15        * 50;
16    I_Recortada [ intervalo + 1 : 2 * intervalo , : ] =
17        I_Recortada [ intervalo + 1 : 2 * intervalo , : ] * 40;
18    I_Recortada [ 2 * intervalo + 1 : 3 * intervalo , : ] =
19        I_Recortada [ 2 * intervalo + 1 : 3 * intervalo , : ] *
20            30;
21
22    .
23    .
24    .
25
26    ocupacion . add ( sum ( I_Recortada [ : ] ) )
27
28    return max ( ocupacion )

```

3.2.5. Inserción de datos

En este apartado, se podrá ver la inserción en la base de datos y/o en el archivo *.csv* de los datos extraídos. Pero cabe destacar que hay cuatro formas de acabar este proceso de extracción de datos desde una imagen, y en función de cada una, los datos a insertar cambian.

1. Análisis Correcto
2. Fondo no disponible
3. Cámara no disponible
4. Error inesperado

Análisis correcto

Este caso se da cuando todo el proceso ha salido correctamente y hemos llegado a la función de *calculaOcupacion()* y esta ha devuelto un valor. En tales circunstancias, sólo debemos insertar a la variable *datos* los valores extraídos (nombre, fecha y ocupación) o, en el caso de la base de datos, ejecutar la función *execute()*, la cual ejecuta una consulta, la cual predefinimos al principio, a la cual se le debe añadir estos valores extraídos.

```

1     datoInsetar = [cam, str (fecha_Hora), str (ocupacion)]
2     datos.append (datoInsetar)
3
4     recordDatos = (cam,str(fecha_Hora),str(ocupacion))
5     cursor.execute(mySql_insert_Datos, recordDatos)

```

Fondo no disponible

Se llega a este supuesto cuando a la hora de calcular el fondo, existe alguna imagen en la que la cámara no esté disponible (Imagen 2.6). Dado este caso, en vez de añadir el valor de la ocupación, insertamos el *string* "Fondo NO Disponible"

```

1  datoInsetar = [cam, fecha_Hora , "Fondo_NO_Disponible"]
2  datos.append(datoInsetar)
3
4  recordDatos = (cam,str(fecha_Hora ),"Fondo_NO_Disponible")
5  cursor.execute(mySql.insert_Datos , recordDatos)

```

Cámara no disponible

Este supuesto es similar al anterior ya que tenemos una imagen no disponible (Imagen 2.6), pero en vez de tener una imagen con la que se calcula el fondo, sería nuestra imagen de estudio.

Un caso es conseciente de otro, ya que supongamos que la cámara está no disponible, y en la siguiente captura esta ya disponible, pero tenemos la imagen anterior no disponible para el cálculo del fondo.

En esta situación, añadimos el *string* " Cam NO disponible ".

```

1  datoInsetar = [cam, fecha_Hora , "Cam_NO_Disponible"]
2  datos.append(datoInsetar)
3
4  recordDatos = (cam,str(fecha_Hora ),"Cam_NO_Disponible")
5  cursor.execute(mySql.insert_Datos , recordDatos)

```

Error inesperado

A este último supuesto se llega cuando en cualquier punto de la ejecución cuando hay un error / excepción, ya que gracias al *Try-Catch* se tiene controlado.

En este punto enviamos un correo electrónico de la misma manera que se explicó anteriormente pero con el mensaje y el asunto cambiado. Además se inserta en la base de datos la tupla formada por la cámara, la fecha y la hora y un *string* con el mensaje de Error pero esta tupla no se inserta en la tabla de *Datos*, sino en una tabla que se creó para ir revisando estos mismos fallos y tenerlos registrados pero fuera de la tabla principal de datos, esta tabla se llama *Log*, la cual se mencionó anteriormente.

```

1  except:
2      mensaje = "Ha ocurrido un error inesperado en la ejecucion del script"
3      asunto = "_Revisar_"
4      enviaCorreo(asunto , mensaje)
5

```

```
6     recordLog = (cam, str(fecha_Hora), "Error")
7     cursor.execute(mysql_insert_Log, recordLog)
```

Guardar datos extraídos

Por último, no se debe olvidar guardar los datos, ya que las actualizaciones realizadas en el fichero *.csv* están en memoria, por lo que debemos llamar a la función *guardaFichero()*, a la que debemos pasarle la variable *datos*, y este mismo se encarga de guardarlo en fichero, sobrescribiendo el existente.

```
1 def guardarFichero(Datos):
2     save(Datos, "./Datos.csv")
```

En el caso que se esté utilizando la base de datos, ya que ahora mismo las modificaciones son inconsistentes, se debe comitear gracias a la función *commit()* y cerrar la conexión con esta base de datos.

```
1 db.commit()
2 db.close()
```

3.3. Automatización de los procesos de extracción de imágenes y cálculo de la ocupación

Este proceso de extracción de imágenes y procesamiento de estas, se ha automatizado en un servidor con el sistema operativo *Ubuntu*, en el cual, nos hemos ayudado de la función integrada en él llamada *crontab*. [9]

Pero primero, debemos crear un script *.sh*, el cuál sea un ejecutable para poder llamarlo. En este script las únicas funciones que nos encontramos son la de hacer que el directorio actual sea donde se encuentran los ficheros *calculaOcupacion.py* y *extraeImagenes.py* y las ejecuta, de manera que primero se descarga las imágenes y/o las actualiza en el caso de que haya diez imágenes por cada cámara, y luego ejecuta el script *calculaOcupacion.py* el cual realiza el proceso explicado anteriormente.

```

1   cd /"Ruta_donde_se_encuentren_los_Scripts"
2   /usr/local/bin/python3.6 ExtraeImagenes.py
3   /usr/local/bin/python3.6 CalculaOcupacion.py

```

Una vez creado el script *.sh* mencionado, el siguiente paso sería introducir en el servidor los siguientes elementos:

1. Script *.sh*, llamado en este caso *Ejecuta.sh*.
2. Script *extraImatgenes.py*.
3. Script *calculaOcupacion.py*.
4. Carpeta con todas las plantillas extraídas con la estructura necesaria (Imagen 2.7) llamada *Plantillas*.
5. Carpeta con la imagen, la cual tomamos como referencia para evaluar si una imagen está o no disponible, esta carpeta debe ser llamada *No_Disponible*.

Cuando se tenga todos estos archivos en el servidor, se debe llamar a la función *crontab* de la siguiente forma

```

1   crontab -e

```

Esta llamada, nos abrirá un editor de texto, y se le debe añadir la siguiente línea

```

1   # m h dom mon dow command
2   */10 * * * * /"Ruta_de_los_Scripts"/Ejecuta.sh

```

Esto que se ha insertado significa que cada 10 minutos (*/10), de todas las horas, de todos los días, de todos los meses (insertando en las posiciones que se quiera indicar todas el carácter *), ejecute el script *Ejecuta.sh*.

Para comprobar que se ha insertado correctamente debemos ejecutar :

```
1      crontab -l
```

Y ahí debemos comprobar que todo está correcto. Incluso ver si hay más añadidas por equivocación.

Este proceso que hemos añadido, lo que realizará será ejecutar el script *Ejecuta.sh* cada 10 minutos, de manera que sólo dejará de ejecutarlo cuando nosotros eliminemos esta automatización.

La primera idea era ejecutar este script cada 5 minutos para tener la mayor cantidad de datos posibles, pero debido a las limitaciones del servidor no podemos ponerlo cada menos tiempo ya que se pisan las ejecuciones.

3.4. Preparación de los Datos

El primer paso antes de empezar a analizar los datos, es preparar los datos extraídos y estructurarlos de manera que sea más sencillo utilizarlos en *Python*.



Numero_Cam	Fecha_Hora	Ocupación
cam1	20210308_1020	113487
cam2	20210308_1020	432312
cam3	20210308_1020	234523
cam4	20210308_1020	Cam NO Disponible
cam5	20210308_1020	Fondo No Disponible
...

Fecha_Hora	Ocupa_cam1	Ocupa_cam2	Ocupa_cam3	Ocupa_cam4	Ocupa_cam5	...
20210308_1020	0.5	0.3	0.9	NaN	NaN	...
...

Fecha_Hora	Ocupa_cam1	Ocupa_cam2	Ocupa_cam3	Ocupa_cam4	Ocupa_cam5	...
20210308_1020	NaN	NaN	NaN	-1	-2	...
...

Figura 3.33: Resultado final de la preparación de datos

Como se puede observar en la imagen 3.33, se ha realizado una gran transformación de los datos leídos de la base de datos o del archivo *.csv*. Se deben realizar los siguientes pasos:

1. Identificar los valores "Cam NO Disponible" y sustituirlos por el valor *-1* e identificar los valores "Fondo NO Disponible" y sustituirlos por el valor *-2*, además, separar el *dataframe* original en dos, uno para cámaras disponibles y otros para cámaras no disponibles para poder hacer también un estudio de estos fallos.
2. Por cada *dataframe*, diferenciar entre los valores de cada cámara, y en el *dataframe* de cámaras disponibles, normalizar los datos pero únicamente con los datos de la misma cámara.

3. Concatenar los valores de cada cámara en un mismo dataframe teniendo tantas filas como instantes de tiempo (fecha y hora) existan y tantas columnas como cámaras sean.
4. Añadir las columnas *Ocupa_ciudad*, que hace referencia a la media de todas las cámaras en esa misma fila, es decir, en ese instante de tiempo y añadir la columna *Day* la cual indica a qué día de la semana pertenece ese instante de tiempo al *dataframe* de cámaras disponibles.

3.4.1. Identificar valores y separarlos en dos *dataframes*

Esta parte del filtrado es muy sencilla, ya que *Pandas* hace que la modificación de valores en un *dataframe* sea fácil, basta sólo realizar una operación booleana sobre la columna *Ocupación* que sean == a los valores deseados e igualarlo al valor que se quiere. Además, gracias a esta modificación, todos los valores que vamos a tener en la columna *Ocupacion* de nuestro *dataframe* original, *datosBrutos*, son de tipo numérico, por lo que le indicamos al *dataframe* que estos valores de la columna *Ocupacion* sean de tipo *float*.

```

1   datosBrutos.loc[datosBrutos['Ocupacion'] == "Cam_NO_Disponible"
                  , 'Ocupacion'] = '-1'
2   datosBrutos.loc[datosBrutos['Ocupacion'] == "Fondo_NO_Disponible", 'Ocupacion'] = '-2'
3
4   datosBrutos["Ocupacion"] = datosBrutos["Ocupacion"].astype(
                                float)

```

Gracias a que todos los valores son numéricos, es muy sencillo separar estos valores para cámaras disponibles (valores iguales a 0 o positivos) y cámaras no disponibles (valores negativos).

```

1   datosNoDisponibles = datosBrutos[datosBrutos["Ocupacion"] < 0]
2   datos = datosBrutos[datosBrutos["Ocupacion"] >= 0]

```

3.4.2. Normalizar los valores

El fin de esta sección es normalizar los valores entre 0 y 1 para que sean comparables entre las cámaras. Los valores ahora mismo representan el número de píxeles ocupados, pero hay imágenes que tienen mucha más resolución que otras y, por tanto, estas imágenes tienen mayor número de píxeles. Esta variación de escala hace que no se pueda comparar dos imágenes de cámaras con distinta resolución, así que, hay que normalizar.

Para ello vamos a utilizar la siguiente fórmula:

$$\text{ValorNormalizado} = (\text{ValorActual} - \text{ValorMínimo}) / (\text{ValorMáximo} - \text{ValorMínimo})$$

Cuando se aplicó esta fórmula se vió que la mayoría de los resultados eran demasiado bajos, en torno a 0, esto es debido a los outliers o valores atípicos, ya que estos valores hacen que la normalización no dé los resultados esperados.

Si se mira una representación de los puntos de una cámara cualquiera vemos lo siguiente:

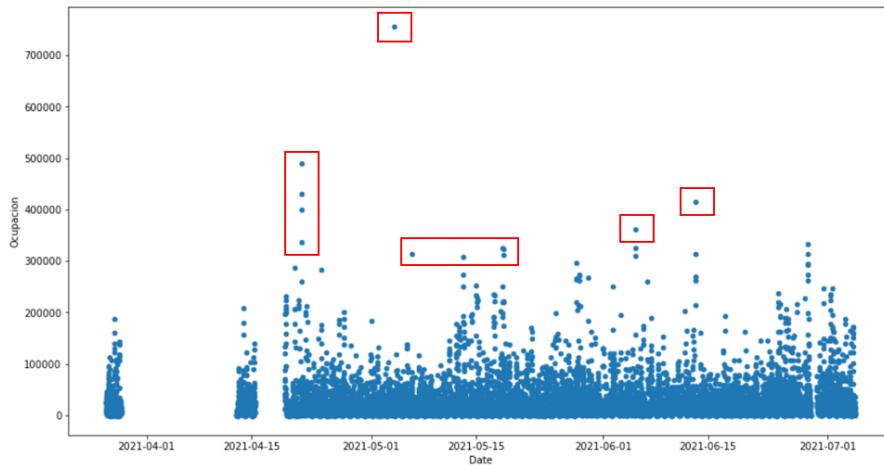


Figura 3.34: Representación de la cámara 1

Se puede observar como los outliers son muchísimo más grandes que la gran mayoría de los puntos, y para normalizar, se coge el máximo valor de la cámara, y como resultado da que todos los valores son muy bajos excepto estos outliers.

Gracias al siguiente histograma, se confirma lo anteriormente dicho, que los outliers son un porcentaje mínimo del total de los datos recogidos.

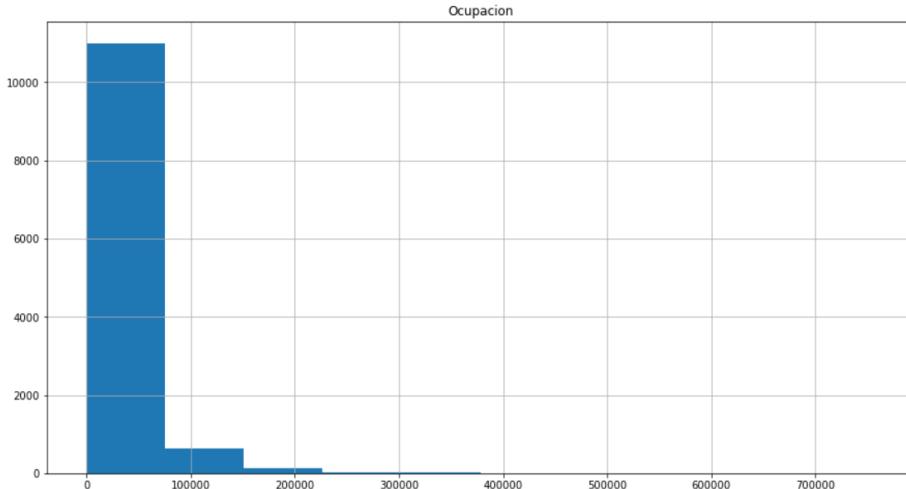


Figura 3.35: Histograma de la cámara 1

Estos outliers son producidos por cámaras con una captura errónea en un momento dado, o justo una luz alumbría la cámara, o que el operario de la cámara haga zoom a una zona nueva, por lo que en un instante dado todo es

54 CAPÍTULO 3. PROCESAMIENTO Y SELECCIÓN DE CARACTERÍSTICAS

nuevo y hace que haya mucha diferencia entre unas y otras. No se ha podido controlar este tipo de outliers.



Figura 3.36: Comparación de imagen normal con sus outliers

Para intentar mejorar esta normalización, se ha pensado que en vez de normalizar por el valor máximo de la cámara, se debe coger el *Bigote Superior* [12], el cual delimita los valores típicos y los atípicos.

En la imagen siguiente se puede ver un diagrama de caja, donde se muestra las diferentes partes que tiene, dónde se concentra la mayor parte de los datos y a partir de donde se puede considerar un outliers.

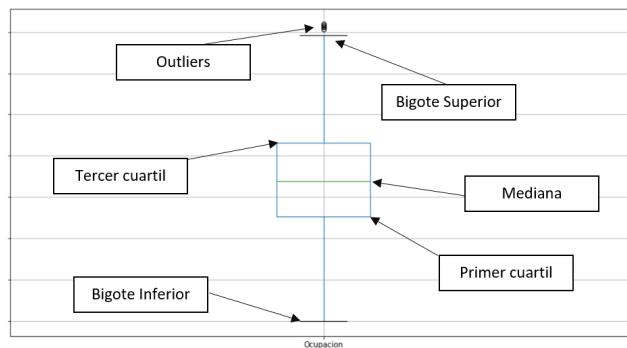


Figura 3.37: BoxPlot de Ejemplo

El *Bigote Superior* se calcula e la siguiente manera:

$$\text{Bigote Superior} = \text{Tercer Cuartil} + (1.5 * \text{Rango Intercuartílico})$$

Al aplicar esta mejora a la normalización, da un mejor resultado para poder compararlo entre las cámaras, ya que un imagen con un valor 0.3 de ocupación está alrededor del 30 % de ocupación.

El único problema que tiene esta mejora es que cuando existe un outlier, su valor es muy superior a 1 y durante 4 o 5 imágenes siguientes (el tiempo en que esta imagen se utiliza para calcular el fondo) nos da un valor superior al real.

Si se hubiera optado por la realización de la normalización con el método tradicional, este problema no hubiera salido pero a la hora de la representación se verían de la misma manera.

3.4.3. Añadir las nuevas columnas

Una vez normalizados los valores y concatenado las columnas teniendo ya el formato de la imagen 3.33, el siguiente paso es calcular el valor de las nuevas columnas *Ocupa_ciudad* y *Day*.

Para calcular la columna *Ocupa_ciudad* es tan sencillo como sumar todos los valores de la fila y dividirlos por el número de cámaras. Y para calcular la columna *Day*, la cual nos indica el día de la semana, existe la función *dt.dayofweek*, perteneciente a las variables de tipo *datetime*, el cuál devuelve un valor entre 0 y 6 donde 0 representa el lunes y 6 representa el domingo.

```
1   dfDisponible [”Ocupa_Ciudad”] = dfDisponible [list (dfDisponible)
2           ].sum (axis=1) / len (list (dfDisponible))
3
3   dfDisponible [”Day”] = dfDisponible [”Date”].dt.dayofweek
```

Capítulo 4

Aplicación de Algoritmos

Una vez que se hayan procesado todas las imágenes, identificado las características, almacenado estos resultados y procesados los datos de manera que sean más fácil entenderlos, llega el momento de aplicar los algoritmos para ver la calidad de nuestros datos extraídos y, además, poder sacar conclusiones de estos datos.

En este capítulo pasaremos por los siguientes pasos:

1. Análisis exploratorio.
2. Algoritmos aplicados.

4.1. Análisis exploratorio

El análisis exploratorio de datos es un paso previo e imprescindible a la hora de comprender los datos con los que se va a trabajar. Es un proceso altamente recomendable para una correcta metodología de investigación ya que con esto, se sabrá de un breve vistazo los valores y tipos de datos que contiene el dataframe.

Para ello, hemos utilizado la función *ProfileReport(dataframe)* perteneciente al paquete de *pandas_profiling* [13] el cual genera un archivo *.html* con un resumen general del *dataframe* y un análisis de cada variable.

Se ha de comentar, que debido a la cantidad de datos y variables (columnas) que tiene nuestro dataframe, se ha realizado el análisis mínimo ya que el cálculo de relaciones entre variables es una carga de trabajo demasiado grande para el equipo donde se ejecutó.

```
1 profile = ProfileReport(dfDisponible, minimal = True)
2 profile.to_file(output_file="Analisis_Exploratorio.html")
```

Este tipo de análisis está dirigido a analistas que no han tocado nada de los datos hasta este momento, en nuestro caso, como nosotros hemos implementado toda la estructura y normalizado los valores, ya tenemos una idea de los tipos y el rango que van a tener, pero aún así podemos sacar algunas conclusiones en claro.

Resumen del dataframe

Este apartado del informe, realiza un breve resumen del dataframe completo, en el que se puede ver:

- Número de columnas.
- Número de filas.
- Cuántos valores nulos o *Missing Values* tenemos y porcentaje del total.
- Cuántas filas duplicadas tenemos.
- Qué tipos de variables hay en el dataframe.

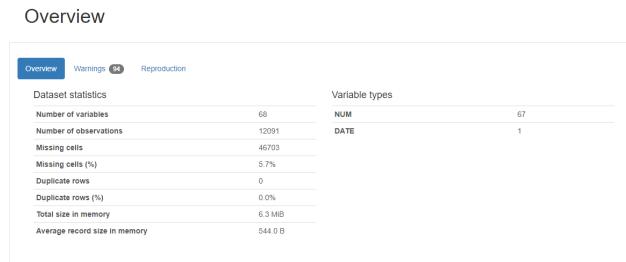


Figura 4.1: Resumen

Análisis de cada variable

En este apartado del informe, detalla más los datos de cada variable (columna). En este apartado podemos ver los siguientes puntos:

- Valores distintos y su porcentaje del total de valores.
- Valores nulos o *missing values*.
- Algunos valores estadísticos como la media, mínimo o máximo.
- Histograma correspondiente a cada variable para ver como se distribuyen los datos.

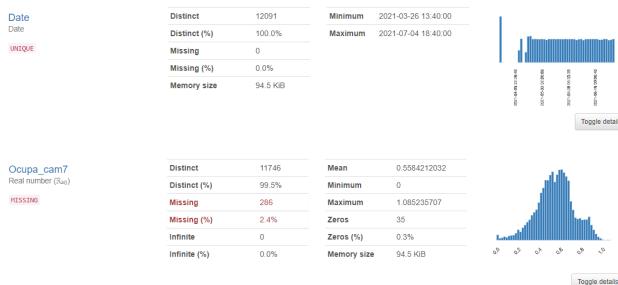


Figura 4.2: Ejemplo de Variables

Cómo podemos ver en estos ejemplos, este análisis ha servido para ver que el histograma de la cámara sigue una distribución normal, al igual que el número de valores nulos es casi inapreciable.

Se puede ver completo el análisis exploratorio en el *APÉNDICE A*.

4.2. Algoritmos aplicados

En este apartado, explicaremos los algoritmos utilizados y comentaremos los resultados obtenidos por cada uno de ellos.

4.2.1. Correlación de Pearson

La correlación lineal es un método estadístico que permite cuantificar la relación lineal existente entre dos variables siempre y cuando sean cuantitativas y continuas. Existen varios coeficientes de correlación lineal (Pearson, Spearman y Kendall). [14]

En este caso, se utilizará el coeficiente de Pearson. Esta correlación, es independiente de la escala de medida de las variables. [16]

En pandas, es muy sencillo calcular la correlación de pearson. [15]

```
1   pearson = dataFrame.corr(method = "pearson")
```

Este método devuelve una matriz de *numColumnas X numColumnas*, donde la intersección entre ellas indica la relación lineal entre estas, cuantificado entre 0 y 1. Además esta función tiene la posibilidad de mostrar un mapa de calor entre las variables, donde el amarillo es el valor máximo y el azul oscuro es el valor mínimo.

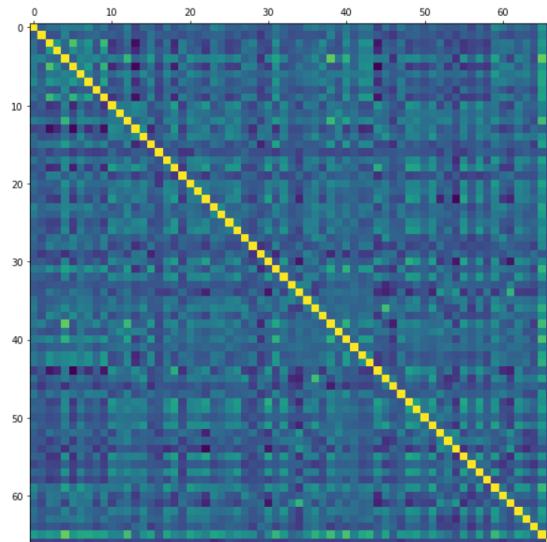


Figura 4.3: Mapa de calor del resultado de la Correlación de Pearson

En este caso, se ha decidido que a partir de 0.6, existe una relación lineal fuerte entre dos variables, y existen varias relaciones reseñables, pero vamos a fijarnos en las relaciones existentes entre *Ocupa_ciudad*, que es la ocupación media de la ciudad en ese instante, y todas las cámaras, de manera que se va a buscar las cámaras que tengan relación con el comportamiento global de la ciudad ya sea cómo causante o cómo consecuencia.

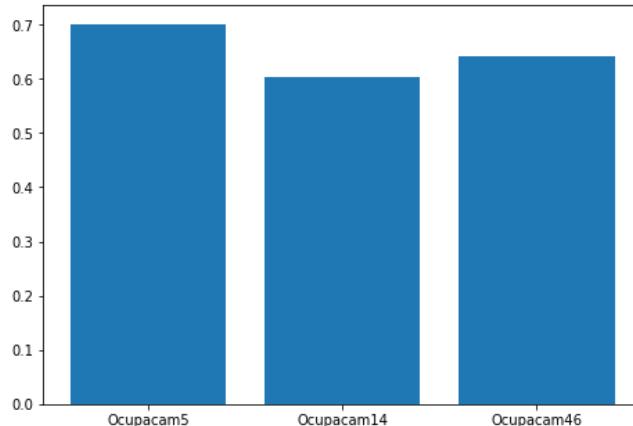


Figura 4.4: Resultado de relación lineal

Con todas se puede sacar la misma conclusión, pero se va a hacer más hincapié en la cámara 5, ya que es la que muestra la relación más fuerte. Se puede obtener que cuando la ocupación que muestra esta cámara es alta influye mucho en la ocupación total de la ciudad.



Figura 4.5: Ejemplo cámara 5

Este resultado tiene lógica ya que es una de las arterias principales de Sevilla y desde aquí deriva al interior de la ciudad y es la avenida en la que se encuentra

la entrada al centro de la ciudad. Además si se mira el mapa de la ciudad, un porcentaje muy alto de cámaras están situadas cerca de esta zona, por lo que si esta avenida está vacía o con la máxima ocupación se ve reflejado en las cámaras colindantes.

4.2.2. Algoritmo K-means

El algoritmo *K-means* es un algoritmo de clasificación no supervisado, que agrupa objetos en K grupos (clusters), basándose en sus características. [18]

Este algoritmo funciona preseleccionando un valor de K . Para encontrar el número de clusters en los datos, deberemos ejecutar el algoritmo para un rango de valores, ver los resultados y comparar características de los grupos obtenidos. En general, no hay un modo exacto de determinar el valor de K , pero se puede estimar con una precisión aceptable. [19]

Valor de K

Determinar el número óptimo de clusters es uno de los pasos más complicados a la hora de aplicar métodos de clustering. Se van a implementar las siguientes técnicas [20]:

1. Método Elbow.
2. Método average silhouette.

Método Elbow También conocido como el método del codo, usa como métrica de comparación la distancia media entre los puntos de datos y su centroide. Cómo el valor de la media siempre disminuirá a medida que aumentemos el valor de k , deberemos encontrar el "*punto codo*", donde la tasa de descenso tiene la bajada más pronunciada. [19]

Al aplicar el algoritmo a nuestro conjunto de datos nos sale la siguiente gráfica:

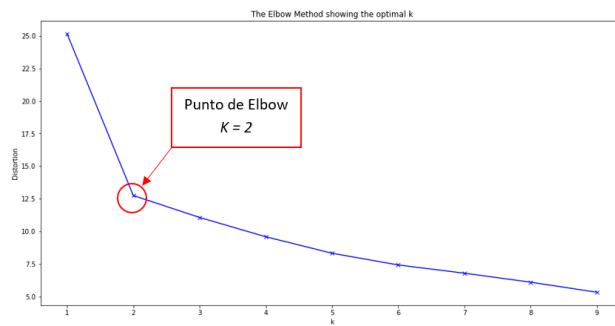


Figura 4.6: Gráfica con el método de Elbow a nuestros datos

En la gráfica anterior podemos distinguir perfectamente el número de clusters con el que se pronuncia la caída de la tasa de descenso, es decir, *el codo de Elbow* y es con K igual a 2.

Método average silhouette El método de average silhouette considera como número óptimo de clusters aquel que maximiza la media del *silhouette coefficient* de todas las observaciones. Este coeficiente cuantifica cómo de buena es la asignación que se ha hecho de una observación comparando su similitud con el resto de observaciones de su cluster frente a las de los otros clusters y su valor puede estar entre -1 y 1.

Al aplicar este método incluido en la librería *sklearn*, ofrece el siguiente resultado:

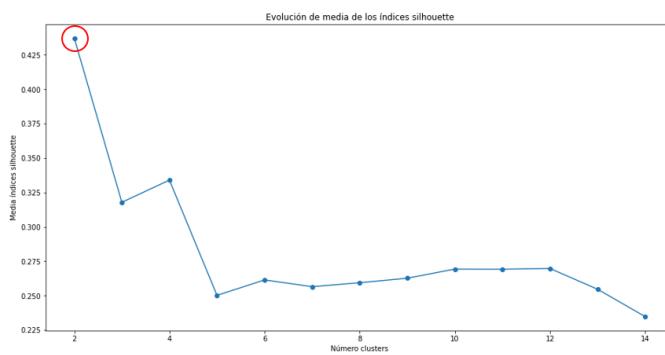


Figura 4.7: Gráfica con el método average silhouette a nuestros datos

Como se puede observar, el valor de K que maximiza este coeficiente es cuando el valor es igual a 2, lo cual nos sirve para ratificar nuestra primera impresión utilizando el método *Elbow*.

Ejecución del algoritmo

Una vez elegido el valor de K , que en nuestro caso es 2, el siguiente paso es ejecutar el algoritmo y ver los resultados.

Y de nuevo, gracias a la librería *sklearn*, es muy simple la ejecución del algoritmo. Debemos de tener en cuenta los siguientes puntos antes de ejecutar el algoritmo:

1. Se debe eliminar los valores nulos.
2. Se debe eliminar las columnas que nos distorsione los resultados como la columna *Day*.
3. Se debe realizar un *resample* de los datos a la media por día, debido a la cantidad de datos que se tiene.
4. Por último, hay que transponer el *dataframe* para que tome como variables las cámaras en vez de las fechas.

```

1  outNa = DataFrame.resample("d").mean().dropna().drop(columns='
2   Day').T
3  kmeans = KMeans(n_clusters=2).fit(outNa)

```

A continuación se va a representar todas las cámaras en un mapa para ver la clasificación que ha realizado el algoritmo *Kmeans* y ver qué cámaras se parecen a qué otras.

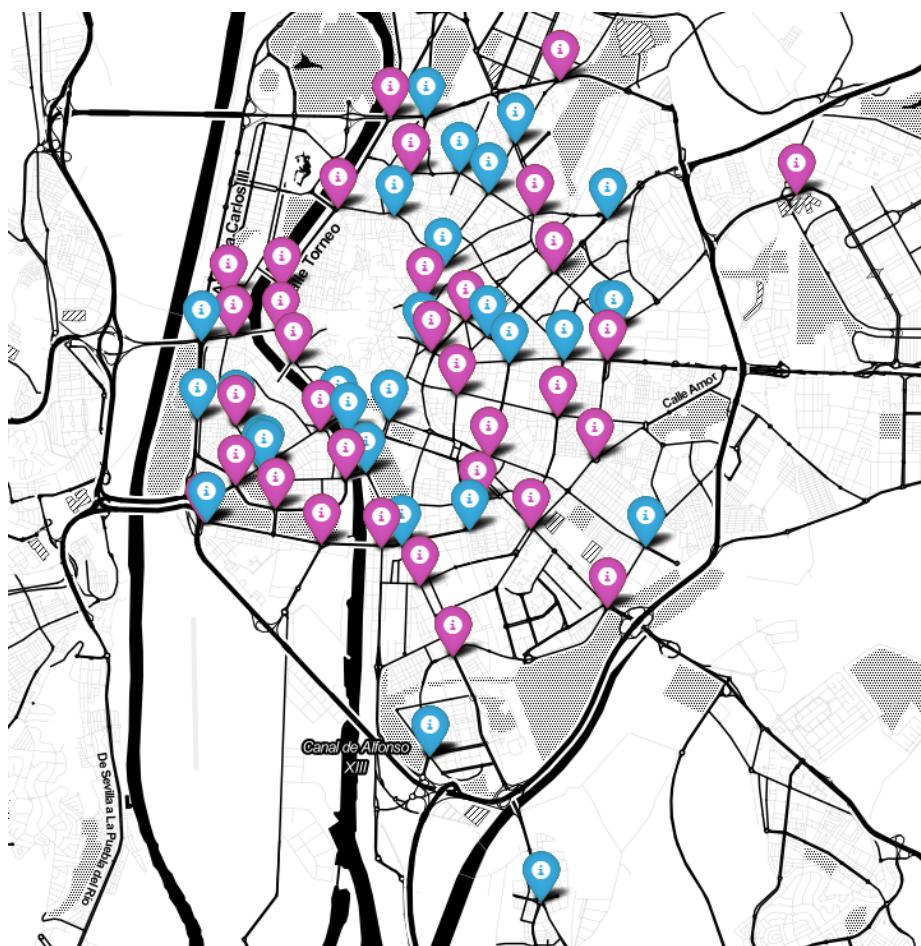


Figura 4.8: Clasificación por colores según *Kmeans*

Si observamos el mapa, ha clasificado de la misma manera la mayoría de las vías rápidas, que no significa vías con más tránsito. Cuando se habla de vías rápidas nos referimos a vías de entrada a la ciudad.



Figura 4.9: Clasificación de vías rápidas según *Kmeans*

4.2.3. Diagrama de Voronoi

El funcionamiento del diagrama de Voronoi es el siguiente, realiza una división del plano indicado, en tantas áreas como nodos se hallan colocados, de tal manera que cada uno de esos puntos posee un área que englobe todo el espacio que esté más cerca de ese punto que de otro cualquiera. Todas las partes del plano en las que la distancia entre dos puntos es idéntica, conforman la frontera entre ambos puntos. [21]

En nuestro caso, los nodos colocados en el plano, son las coordenadas de cada cámara (latitud y longitud).

Calcularlo en *Python*, es una operación muy sencilla. [22]

Esta función devuelve las posiciones de cada polígono y los vértices que contiene. Si mostramos el diagrama calculado anteriormente obtenemos el siguiente mapa:

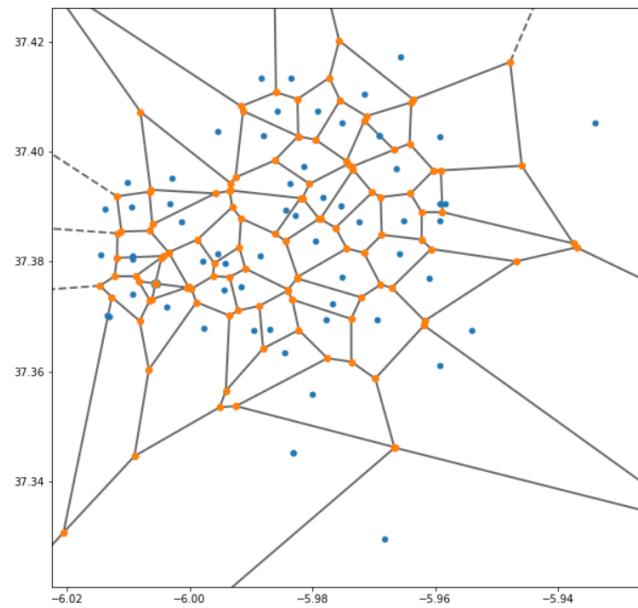


Figura 4.10: Diagrama de Voronoi

Además, gracias a una función encontrado por internet [23], se puede pintar este diagrama sobre el mapa de Sevilla y se pueden identificar mejor las zonas.

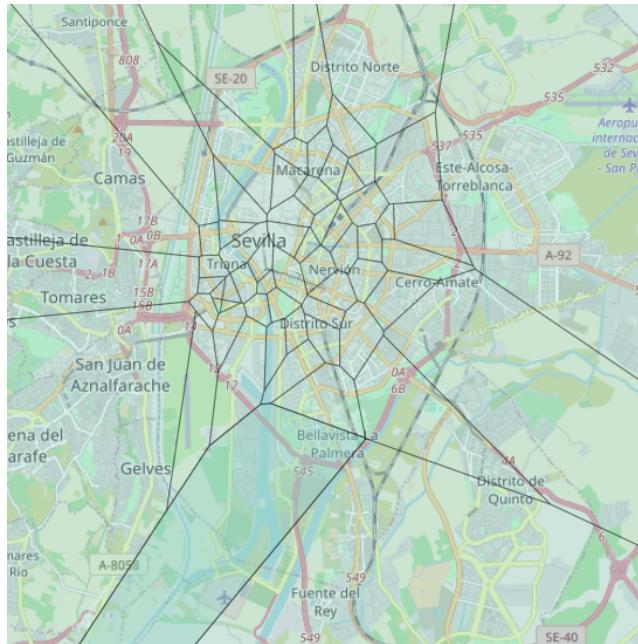


Figura 4.11: Mapa con el Diagrama de Voronoi

Gracias a este diagrama, se puede ver donde existen cámaras de más o es necesario poner más cámaras debido a que influyen en una zona muy amplia.

Si se amplía la imagen observando sólo lo que es la ciudad de Sevilla, a priori parece que hay una aglomeración de cámaras en la siguiente zona.



Figura 4.12: Mapa con el Diagrama de Voronoi Ampliado

Pero si se observa bien, una cámara tiene influencia sobre la ocupación en demasiadas calles a la vez, por lo que la medida no es del todo representativa.

Además como se puede observar en la imagen 4.12, la cámara 5, la cuál según la correlación de Pearson, es la que más relación tiene con el comportamiento de la ciudad, parece que no tiene mucha relevancia por la situación geográfica de esta cámara, pero debido a la dirección de la circulación de entrada a la ciudad, si se entiende la relación.



Figura 4.13: Dirección de la ciudad

Al mirar la imagen 4.13, se observa que la entrada de la mayoría de los vehículos a la zona céntrica de la ciudad, es decir, donde se encuentra la mayor parte de las cámaras, se realiza a través de esta avenida, por lo que esta relación entre la cámara 5 y la ocupación general de la ciudad tiene justificación.

Capítulo 5

Visualización de Resultados

La visualización de datos es la presentación de datos en formato gráfico. Permite ver los datos de forma visual, de modo que se pueden captar conceptos o identificar patrones más fácilmente.

Este apartado se visualizarán las siguientes gráficas:

1. Evolución de la ocupación de un día, por horas.
2. Comportamiento de cada día de la semana.
3. Representación visual de la ocupación actual de la ciudad.
4. Cámaras NO disponibles.

5.1. Evolución de la ocupación de un día, por horas

En este apartado, veremos como evoluciona la ocupación en la cámara 23 a lo largo de las horas de un día. A continuación podemos ver un ejemplo de esta cámara.



Figura 5.1: Ejemplo de la cámara 23



Figura 5.2: Evolución del día de la cámara 23

Podemos observar como desde las 24 horas hasta las 6 de la mañana, la ocupación es mínima ya que debido a que es un día entre semana, por las noches no suele haber mucho tráfico, pero a partir de esa hora, la ocupación empieza a aumentar considerablemente debido a la mayor cantidad de personas que entran a trabajar. A partir de las 8, la ocupación desciende un poco pero se mantiene, excepto un pico producido sobre las 12 de la mañana, que suele ser la hora habitual en que la afluencia de gente aumenta para ir en dirección al centro comercial para compras o otros motivos.

Por último podemos ver cómo desde las 15 horas hasta las 18 horas se mantiene la ocupación, pero a partir de ahí desciende hasta un 10 % aproximadamente debido a que la mayoría de los trabajadores ya se han ido y por esa hora no suele haber mucho tráfico por la zona. Exceptuando el pico que se produce entre las 22 horas y las 24 aproximadamente que puede ser producido por la salida de la gente del centro comercial hacia sus viviendas tras cenar o motivos similares.

A continuación, se mostrará la gráfica que muestra la evolución de la cámara un domingo.



Figura 5.3: Evolución del día de la cámara 23 un domingo

En este caso, se puede observar cómo el comportamiento no es en nada parecido a la gráfica 5.2, ya que en este caso si hay mucho más tráfico en la madrugada del sábado al domingo, y en las horas siguientes sigue una media de un 20 % de ocupación. Excepto a partir de las 22 horas, que se puede suponer que es cuando la gente se acerca a la zona para cenar.

Si por cualquier motivo, la cámara estuviera fuera de servicio, se observaría en la gráfica de la siguiente manera.

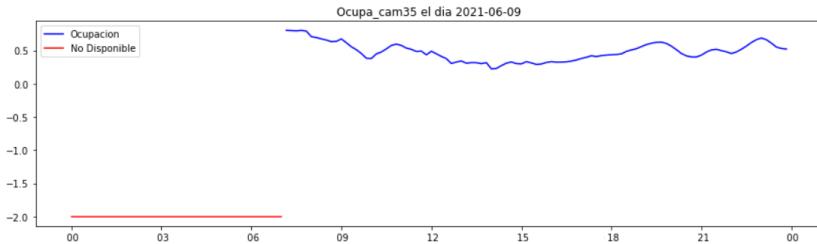


Figura 5.4: Ejemplo de la evolución cuando está no disponible

Asimismo, también se puede ver esta evolución en el comportamiento global de la ciudad.

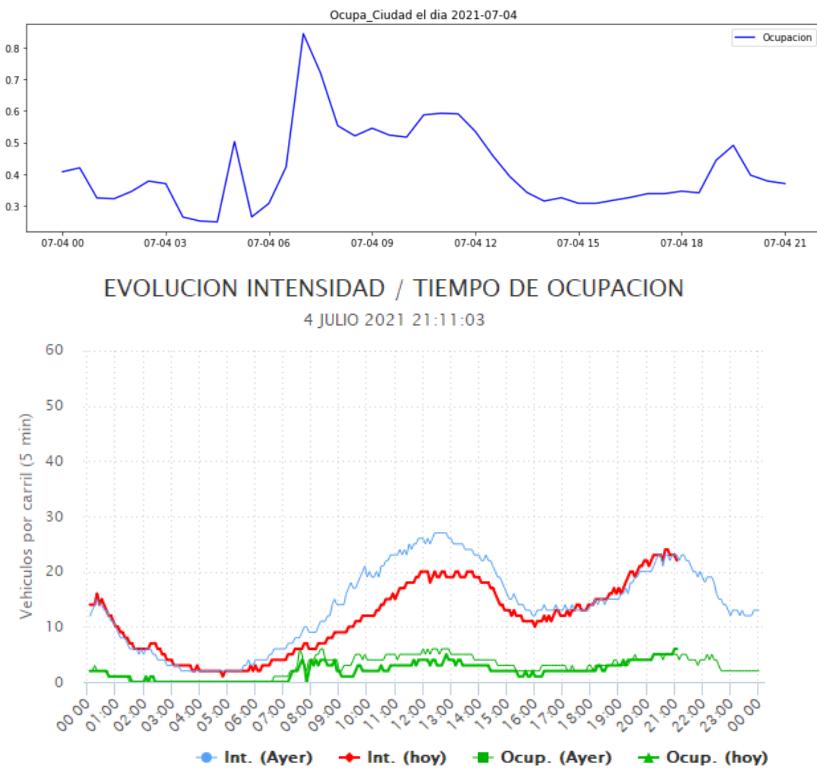


Figura 5.5: Evolución del día de la ciudad

En la imagen anterior, podemos observar en la parte de arriba, una gráfica que muestra la evolución de la ocupación en la ciudad extraída con nuestro proyecto, en la parte de abajo, la evolución de la ocupación de la ciudad pero esta gráfica es extraída desde la web oficial del ayuntamiento de Sevilla.

Si observamos las dos gráficas, la línea verde oscura, que es la que indica la ocupación extraída del gráfico del anterior, y la línea azul, que pertenece a la gráfica extraída por nuestro algoritmo, siguen la misma tendencia.

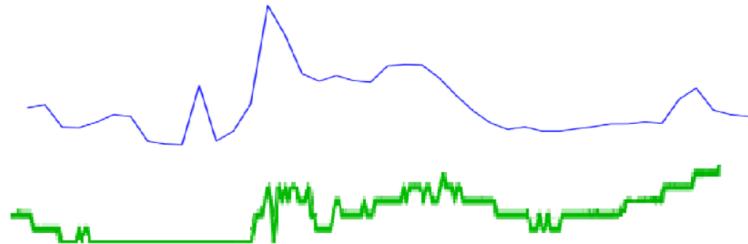


Figura 5.6: Superposición de gráficas

5.2. Comportamiento de cada día de la semana

En este apartado se analizará cada día de la semana. Pero primero, para validar que nuestro sistema tiene un comportamiento uniforme, se va a caracterizar todos los lunes en una misma gráfica.

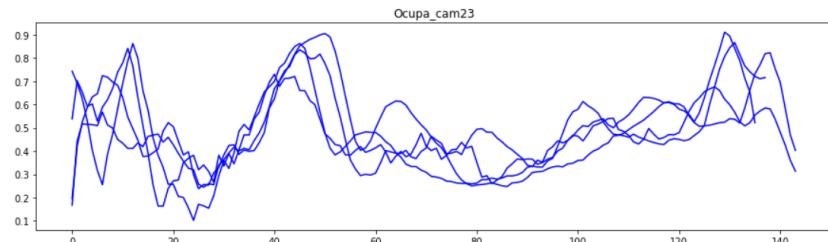


Figura 5.7: Caracterización de todos los lunes de junio

En la gráfica 5.7 se puede observar que todos los lunes en esta cámara siguen la misma tendencia, con lo que se demuestra que no tiene un comportamiento errático y valida lo anteriormente expuesto.

Antes de continuar con el proceso, se debe indicar que hemos filtrado el *dataframe* de datos para sólo obtener los datos que se hayan obtenido durante las horas de sol, ya que si se analiza el comportamiento medio de los días de la semana con las horas nocturnas, esta media bajaría considerablemente ya que existen muchos registros con valores bajos de ocupación por la noche, y se quiere hacer hincapié en las horas diurnas. Además, la diferencia entre los fines de semana y los días entre semana no sería tan notable debido a que por la noche, los fines de semana tienen mayor actividad.

Primero se verá el comportamiento de la cámara 23 de todos los lunes y de todos los domingos para comprobar si en este caso, la ocupación que tiene los fines de semana es inferior a los días entre semana.

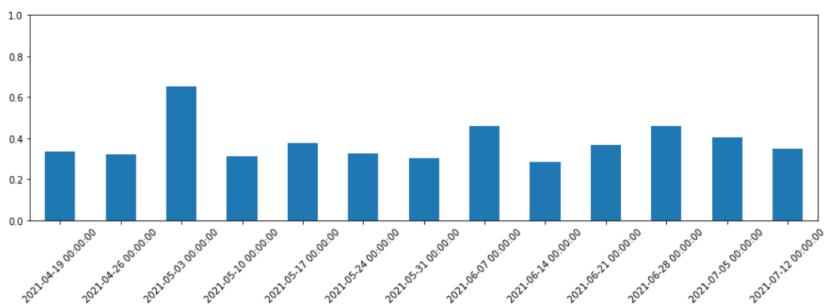


Figura 5.8: Comportamiento medio de todos los lunes de la cámara 23

En esta primera gráfica, se puede observar que, exceptuando el día 19 de abril, el comportamiento de los lunes de esta cámara tiene una media de un 40% de ocupación, con un comportamiento estable y cíclico que sigue un patrón de que cada 3 lunes existe un pico de ocupación.

En la siguiente gráfica, que muestra todos los domingos de la cámara 23, sigue un comportamiento más estable, debido a que tiene muy poca ocupación durante este día, aproximadamente un 20%.

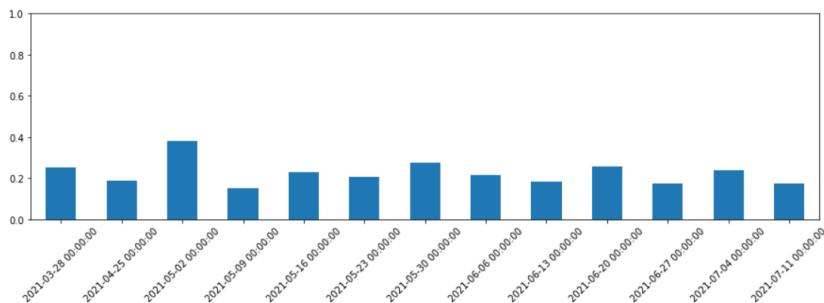


Figura 5.9: Comportamiento medio de todos los domingos de la cámara 1

Por último, se muestra una gráfica en el que se compara la media de ocupación por día siendo la primera columna el lunes y la última el domingo.

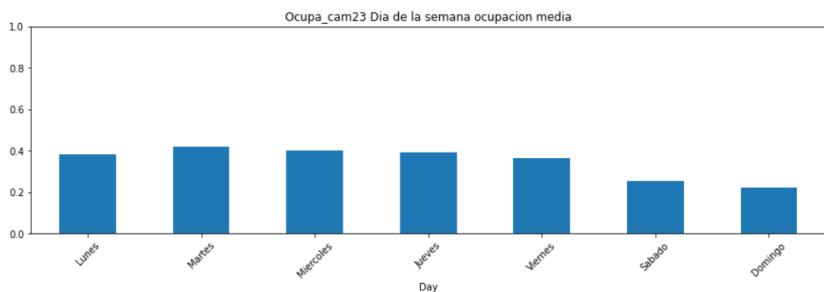


Figura 5.10: Comportamiento medio de cada día de la semana

Se puede contemplar que los días entre semana tienen aproximadamente la misma ocupación, pero se puede observar que a partir del miércoles, la ocupación

va descendiendo hasta que llegamos al domingo, que la ocupación es casi la mitad que podemos observar en los días entre semana.

5.3. Representación visual de la ocupación de la ciudad

En este punto se va a mostrar un mapa de la ciudad de Sevilla en el que se puede ver la ocupación en momento dado en cada cámara. Cada cámara puede ser de varios colores que indica su ocupación o su disponibilidad. Los colores y su significado son los siguientes:

1. Color Rojo, ocupación por encima del 70 %.
2. Color Naranja, ocupación entre el 70 % y el 30 %.
3. Color Verde, ocupación por debajo del 30 %.
4. Color Negro, este color representa que la cámara no está en funcionamiento, no disponible, o que el cálculo de la ocupación de esta cámara haya dado un error.



Figura 5.11: Mapa en un instante dado de la ciudad

Se puede observar la ocupación de cada cámara, y según nuestra columna *Ocupa_Ciudad*, la ciudad en este instante tiene un 32,0668 % de ocupación, que se corresponde con lo que podemos ver en la imagen 5.11.

Este mapa es interactivo, con lo que podemos acercarnos a cada punto para analizar mejor la ocupación como en la siguiente imagen.



Figura 5.12: Mapa aumentado en un instante dado de la ciudad

5.4. Estudio sobre las cámaras no disponibles

En esta sección, gracias al filtrado que hicimos anteriormente, podemos hacer un estudio de las cámaras que fallan y se encuentran no disponible. Podremos saber:

1. Cámara que más tiempo ha estado en No disponible.
2. Tiempo medio de una cámara fuera de servicio.

Cámara que más tiempo ha estado en No disponible

Gracias al dataset *datosNoDisponible*, se ha podido sacar la siguiente gráfica:

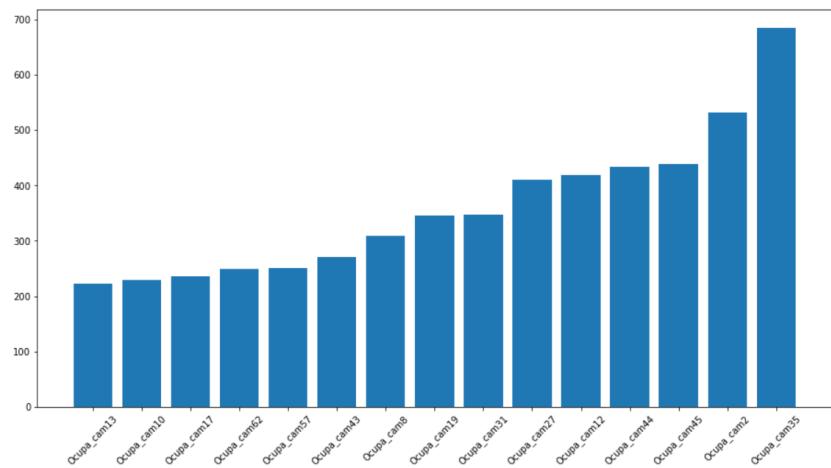


Figura 5.13: Cámaras con más imágenes perdidas

Donde podemos ver las cámaras que mas tiempo han estado en *No Disponible*, con la correspondiente pérdida de información.

La cámara que más tiempo ha estado no disponible es la cámara *Ocupa.cam35*, con un total de 690 imágenes perdidas. Incluso podemos ver en la siguiente gráfica que ha estado incluso varios días inactiva.

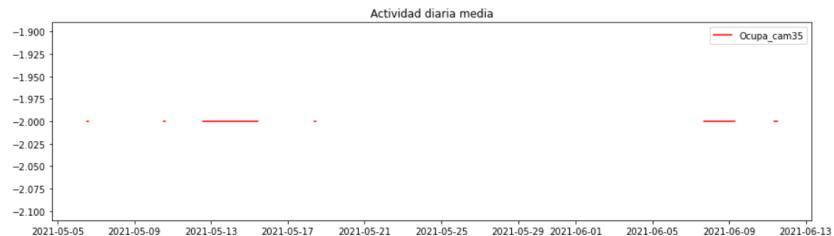


Figura 5.14: Cámara que más tiempo ha estado no disponible

Y si se toma una imagen cada 6 minutos, esta cámara ha estado un total de 2,85 días fuera de servicio.

El problema que tiene estas cámaras que de un momento a otro fallan, no es el tiempo de reparación, sino el tiempo en que el operario se da cuenta de que la cámara falla. Con el sistema que mencionamos anteriormente, estos fallos se detectarían en 4 imágenes, es decir en 24 minutos.

A continuación se podrá ver un mapa con estas cámaras que más tiempo han estado en esa situación:



Figura 5.15: Mapa con la posición de las cámaras con más fallos

Tiempo medio de una cámara fuera de servicio

Si analizamos el total de las cámaras, en estos 3 meses en los que se han tomado registro de los datos, se han perdido 9516 imágenes, que es un total de 146 imágenes por cámara, lo que hace un tiempo medio de 24,3 horas fuera de servicio cada cámara.

10 más tiempo fuera de servicio pintar en un mapa estas cámaras

Capítulo 6

Conclusiones

En este proyecto empezó con el objetivo de conseguir medir de manera aproximada la ocupación en las carreteras de Sevilla. Una vez que consiguiésemos esto, hay infinitos estudios que se pueden hacer gracias a estos datos, siendo este punto en el que se quería hacer más hincapié en el proyecto. Pero, mediante íbamos avanzando en él, vimos que la extracción de imágenes, su análisis y su automatización en un servidor, cuyos recursos de hardware eran muy limitados, era al final la parte más tediosa y larga del proyecto, ya que se ha invertido la mayor parte del tiempo en este punto.

Tras mucho esfuerzo, se ha conseguido y se ha realizado un breve análisis de estos datos extraídos para comprobar que estos son válidos.

Este proyecto tiene muchos puntos fuertes, de los cuales vamos a mencionar algunos a continuación:

1. El algoritmo para la extracción de fondo gracias a la mediana de una serie de fotos, es un algoritmo sencillo de entender y con unos resultados bastante buenos.
2. La eficiencia de sus algoritmos para su ejecución en cualquier equipo. Además, gracias a su implementación en *Python*, no se necesita ningún programa de pago.
3. La implementación de las plantillas manuales, que no sólo sirve para identificar las zonas de carretera sino que elimina todo el ruido que no está encima de las carreteras (peatones, parkings ,...).
4. El mecanismo implementado para eliminar el ruido producido por la oscilación de las cámaras.
5. La detección de errores en las cámaras, incluso mejoramos al servicio del ayuntamiento de Sevilla, ya que identificamos esto antes que ellos.

Pero, también debemos mencionar que hay muchas cosas que mejorar, algunas de las cuales, nosotros no tenemos manera para hacerlo:

1. Como pudimos observar en el *Diagrama de Voronoi*, es necesario tener más cámaras en la ciudad para poder tener una imagen más real de la situación de las carreteras de Sevilla, ya que hay muchas cámaras que tienen una zona de influencia muy grande.

2. La mayoría de las cámaras están situadas en semáforos, por lo que tampoco vemos una medida real de estas carreteras, porque por muy poco tráfico que haya, si un semáforo está en rojo, normalmente hay como mínimo 2 o 3 filas de coche y este algoritmo te dará una ocupación de un 30 o 40 % cuando en realidad está vacía, sólo que al tomar la foto desde un semáforo en rojo, da una sensación falsa de ocupación.
3. Debido a las limitaciones del equipo donde se ejecutan los algoritmos, se toman imágenes cada 10 minutos, cuando en realidad se podrían tomar las imágenes cada 5 minutos. Con ello podríamos tener una medida más real y menos discreta de la situación para ver cómo evoluciona la ocupación.
4. Al optar por las plantillas manuales, debemos estar pendientes de si se implementan nuevas cámaras o incluso, si cambian la cámara, es decir, la mejoran o simplemente cambia la resolución de esta, entonces las plantillas ya no sirven, y con ello, la creación de nuevas plantillas.
5. El sistema en general es muy susceptible a cambios, ya sea en las cámaras o en la web del ayuntamiento de Sevilla.
6. El cálculo de la ocupación es mejorable, ya que es muy susceptible a cambios de vistas nuevas, o incluso cuando el operario hace zoom, como hemos visto anteriormente.

En definitiva, este trabajo se puede considerar cómo la base para futuros proyectos, ya sea para solucionar alguno de los fallos vistos anteriormente o como para implementar ideas nuevas, o realizar el mismo análisis en distintas ciudades. Pero cumple con nota las expectativas previstas de este.

Bibliografía

- [1] ARANDA CORRAL, GONZALO A.
- [2] UNIVERSIDAD COMPLUTENSE DE MADRID, *¿Qué es Data Science?* , <https://www.masterdatascienceucm.com/que-es-data-science/>
- [3] WIKIPEDIA, *Ciencia de datos*, https://es.wikipedia.org/wiki/Ciencia_de_datos
- [4] NORTHEASTERN UNIVERSITY, *Understanding a Project Lifecycle* , <https://www.northeastern.edu/graduate/blog/data-analysis-project-lifecycle/>
- [5] SÁNCHEZ AMENEIRO, ANA, *Sevilla es la tercera gran capital con más atascos después de Barcelona y Madrid*, https://www.diariodesevilla.es/sevilla/Sevilla-tercera-capital-atascos-ranking-TomTom-Barcelona-Madrid_0_1360964109.html
- [6] PYPI, *Información sobre librerías* , <https://pypi.org/>
- [7] MARÍN SANTOS, DIEGO, *Sistemas de Percepción y Visión por computador*.
- [8] LUDA, MARCELO, *De Matlab a Python* , <https://marceluda.github.io/python-para-fisicos/intro/de-matlab-a-python/>
- [9] CRESPO, ADRIÁN, *Cómo utilizar Cron y Crontab en Ubuntu* , <https://www.redeszone.net/2017/01/09/utilizar-cron-crontab-linux-programar-tareas/>
- [10] Wikipedia, *Segmentación (procesamiento de imágenes)* , [https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_\(procesamiento_de_im%C3%A1genes\)#:~:text=La%20segmentaci%C3%B3n%20en%20el%20campo,grupos%20de%20p%C3%ADxeles\)%20u%20objetos.&text=Cada%20uno%20de%20los%20p%C3%ADxeles,la%20intensidad%20o%20la%20textura.](https://es.wikipedia.org/wiki/Segmentaci%C3%B3n_(procesamiento_de_im%C3%A1genes)#:~:text=La%20segmentaci%C3%B3n%20en%20el%20campo,grupos%20de%20p%C3%ADxeles)%20u%20objetos.&text=Cada%20uno%20de%20los%20p%C3%ADxeles,la%20intensidad%20o%20la%20textura.)
- [11] LUIS, *Aprendiendo Latex*, <http://minisconlatex.blogspot.com/2012/09/hyperlinks-con-latex.html>

- [12] R CODER *Box Plot en R*, <https://r-coder.com/boxplot-en-r/>
- [13] PANDAS PROFILING, <https://pandas-profiling.github.io/pandas-profiling/docs/master/index.html>
- [14] AMAT RODRIGO, JOAQUÍN, *Correlación lineal con Python*, <https://www.cienciadedatos.net/documentos/pystats05-correlacion-lineal-python.html>
- [15] PANDAS, *Función pandas.DataFrame.corr*, <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html>
- [16] WIKIPEDIA, *Coeficiente de correlación de Pearson*, https://es.wikipedia.org/wiki/Coeficiente_de_correlaci%C3%B3n_de_Pearson
- [17] MANUAL DE LATEX, *Crear tablas en LaTeX*, <https://manualdelatex.com/tutoriales/tablas>
- [18] UNIVERSIDAD DE OVIEDO, *El algoritmo k-means aplicado a clasificación y procesamiento de imágenes*, https://www.unioviedo.es/comprnum/laboratorios_py/kmeans/kmeans.html
- [19] APRENDE MACHINE LEARNING, *K-Means con Python paso a paso*, <https://www.aprendemachinelearning.com/k-means-en-python-paso-a-paso/>
- [20] AMAT RODRIGO, JOAQUÍN, *Clustering con Python*, <https://www.cienciadedatos.net/documentos/py20-clustering-con-python.html>
- [21] EL ECONOMISTA, *Así es el Diagrama de Voronoi*, <https://www.eleconomista.es/politica/noticias/10304109/01/20/Asi-es-el-Diagrama-de-Voronoi-el-metodo-matematico-para-dividir-el-mundo.html>
- [22] SCIPY, *Función scipy.spatial.Voronoi*, <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html>
- [23] ROSSANT, CYRILLE, *Computing the Voronoi diagram of a set of points*, <https://ipython-books.github.io/145-computing-the-voronoi-diagram-of-a-set-of-points/>
- [24] FOLIUM DOCUMENTATION, *Quickstart*, <https://python-visualization.github.io/folium/quickstart.html>

Apéndice A

Resultado de Análisis Exploratorio

En este anexo, mostraremos el análisis exploratorio completo del *dataframe* de datos disponibles.

Pandas Profiling Report 

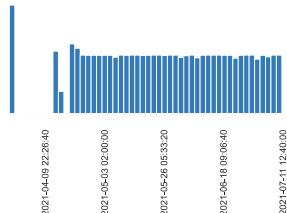
Overview

Overview	Warnings 94	Reproduction
Dataset statistics		
<hr/>		
Number of variables	68	
Number of observations	12938	
Missing cells	49450	
Missing cells (%)	5.6%	
Duplicate rows	0	
Duplicate rows (%)	0.0%	
Total size in memory	6.7 MiB	
Average record size in memory	544.0 B	
Variable types		
<hr/>		
NUM	67	
DATE	1	

Variables

Date	
Date	
	UNIQUE
Distinct	12938
Distinct (%)	100.0%
Missing	0
Missing (%)	0.0%
Memory size	101.1 KiB

Pandas Profiling Report



[Toggle details](#)

Ocupa_cam1Real number ($\mathbb{R}_{\geq 0}$)

MISSING

ZEROS

Distinct	10013
----------	-------

Distinct (%)	79.4%
--------------	-------

Missing	321
---------	-----

Missing (%)	2.5%
-------------	------

Infinite	0
----------	---

Infinite (%)	0.0%
--------------	------

Mean	0.3407724896
------	--------------

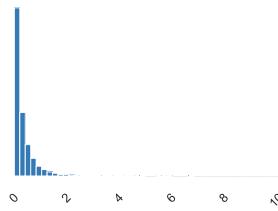
Minimum	0
---------	---

Maximum	10.38282655
---------	-------------

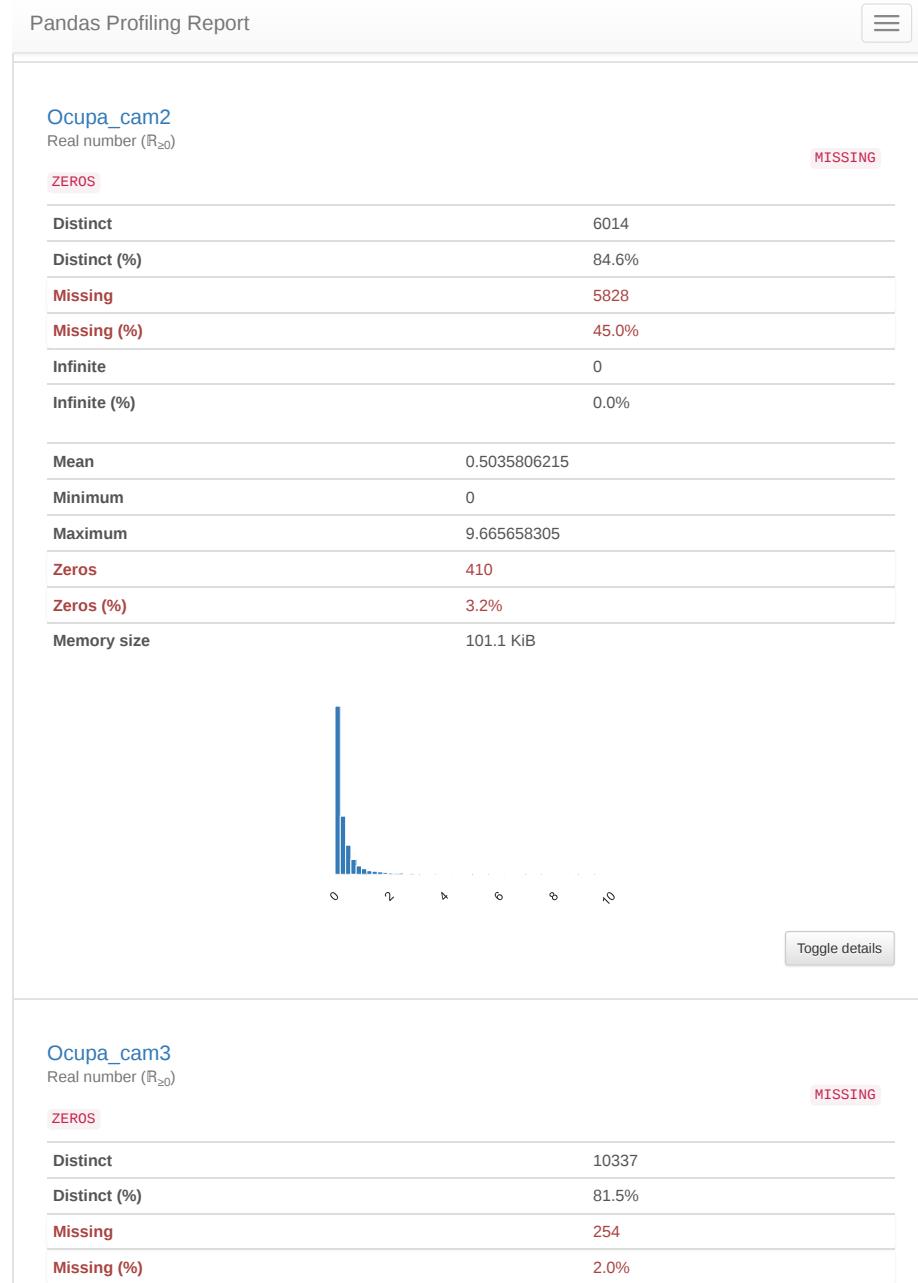
Zeros	528
-------	-----

Zeros (%)	4.1%
-----------	------

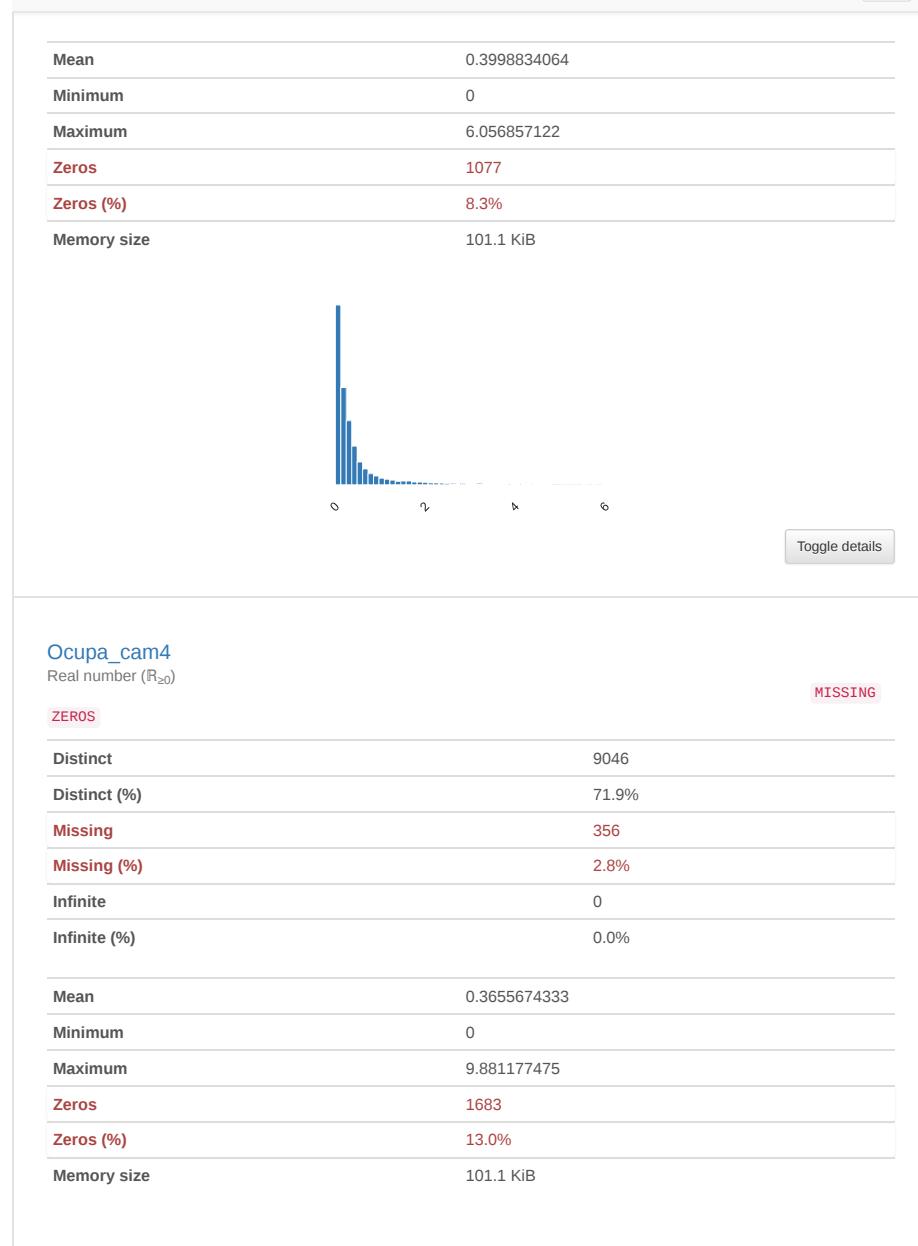
Memory size	101.1 KiB
-------------	-----------

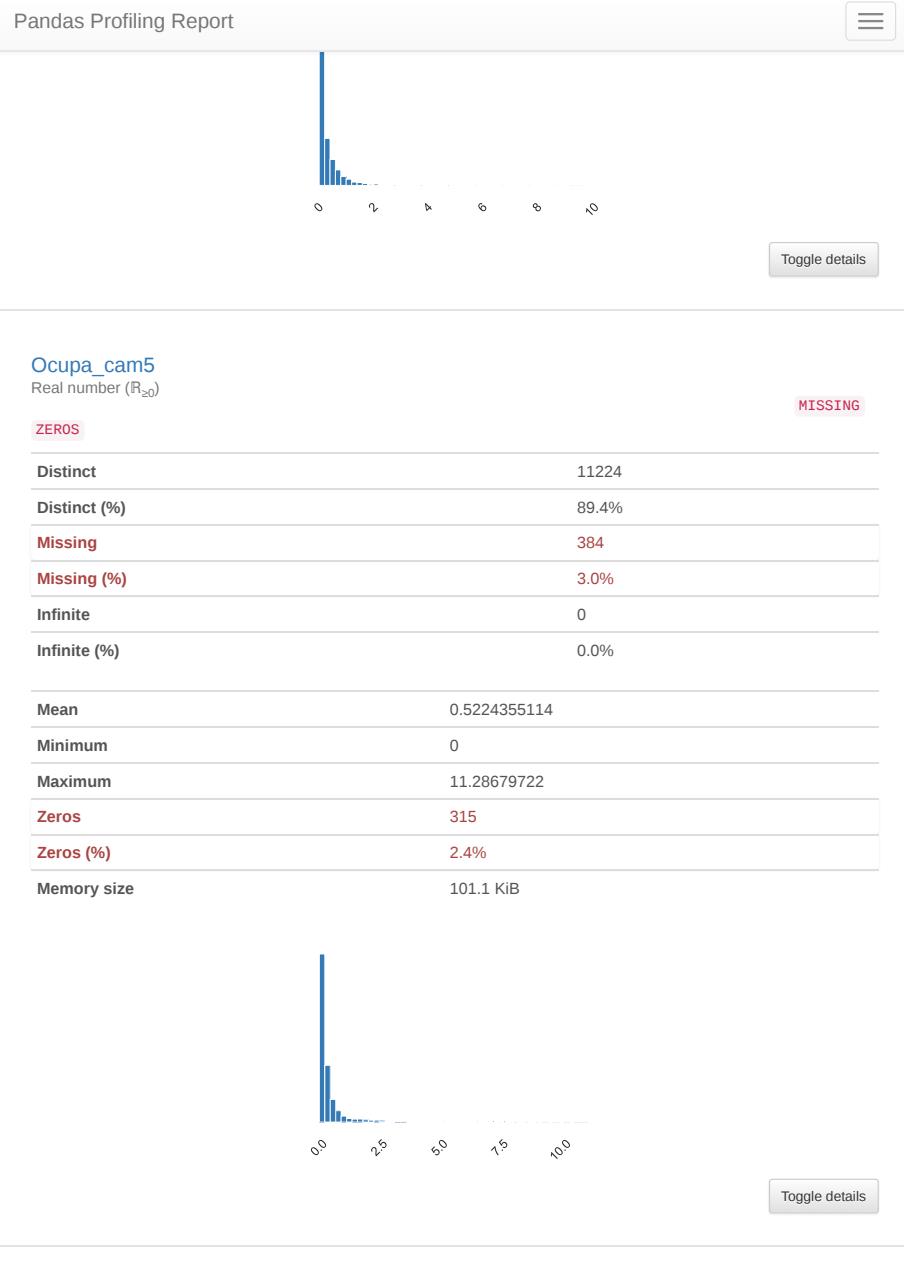


Pandas Profiling Report

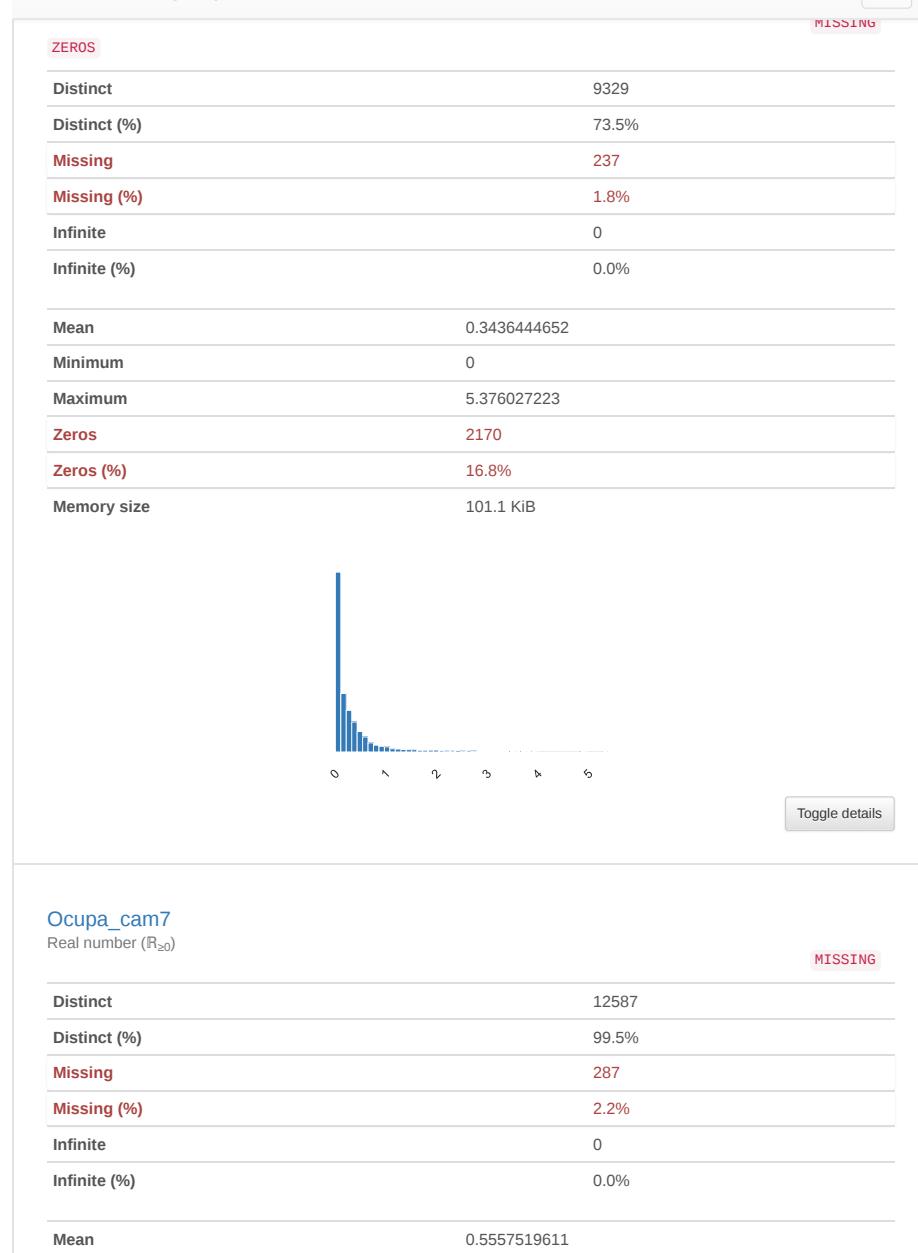


Pandas Profiling Report

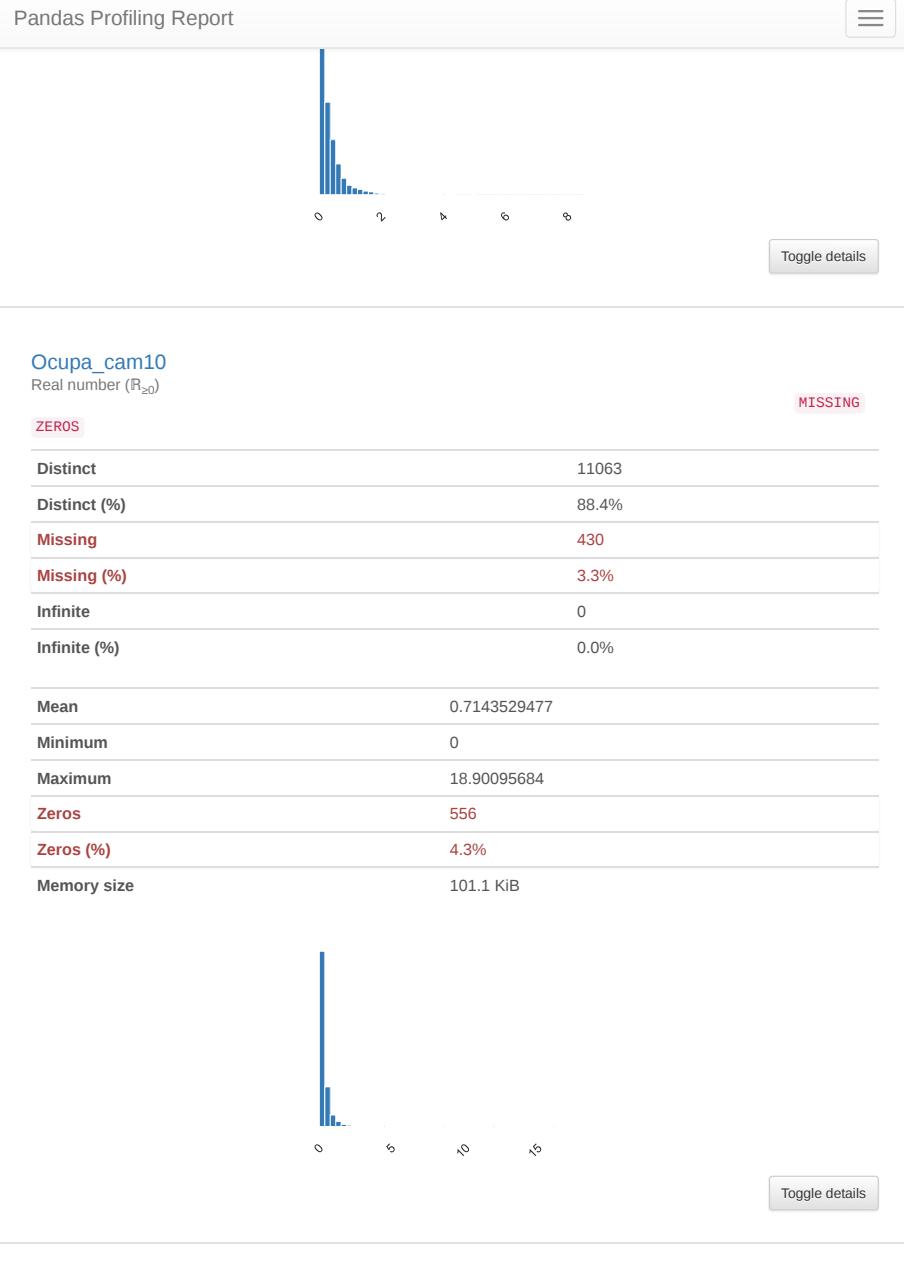


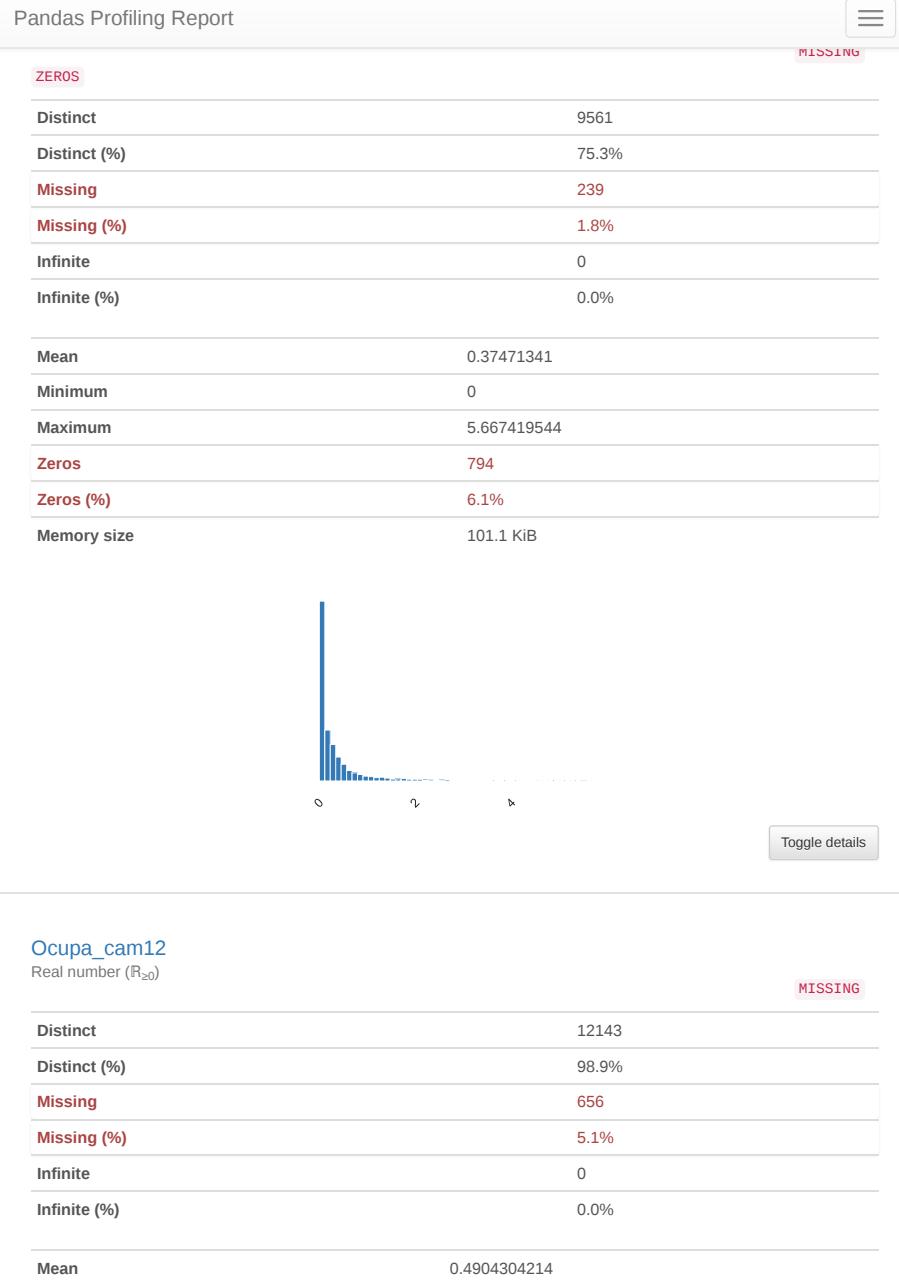


Pandas Profiling Report

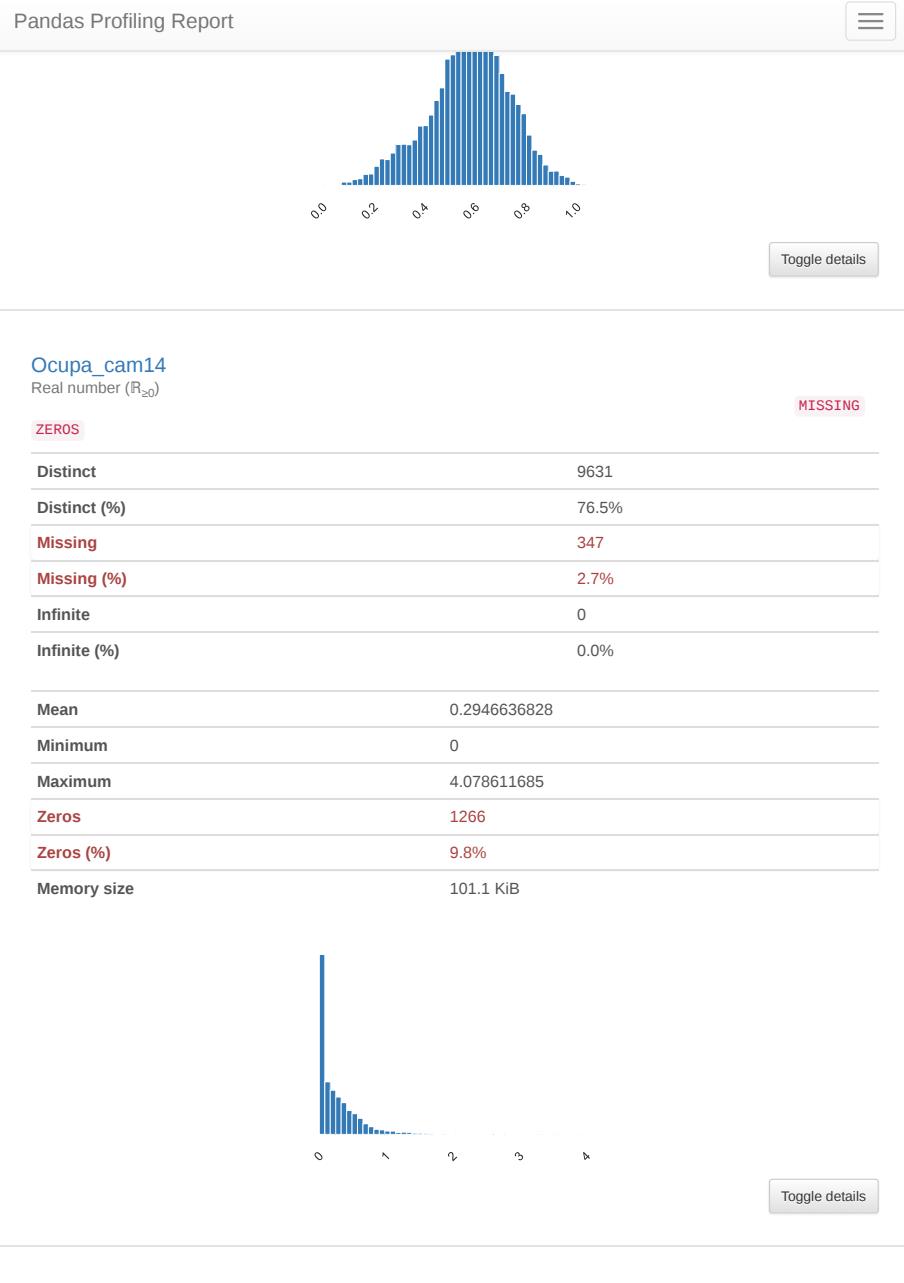




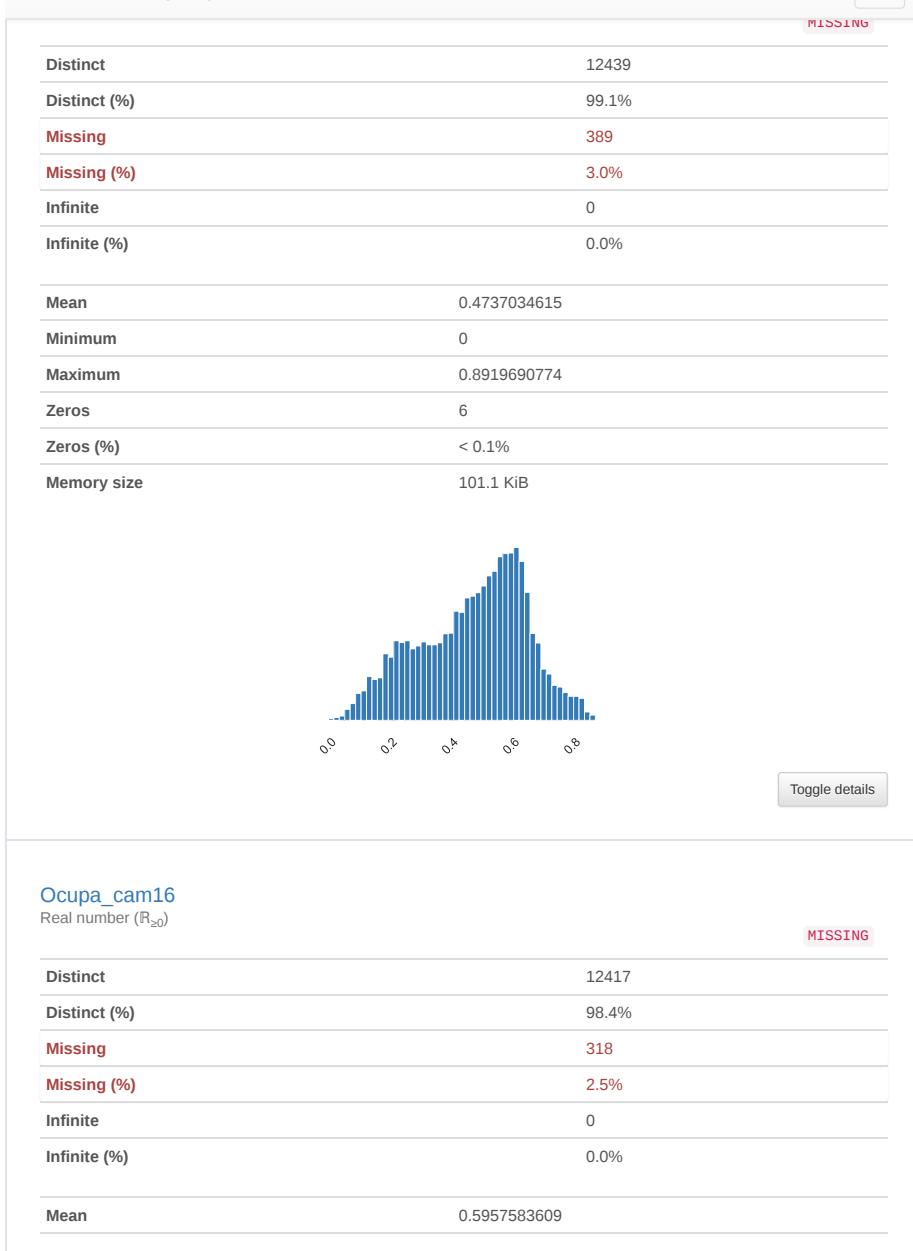


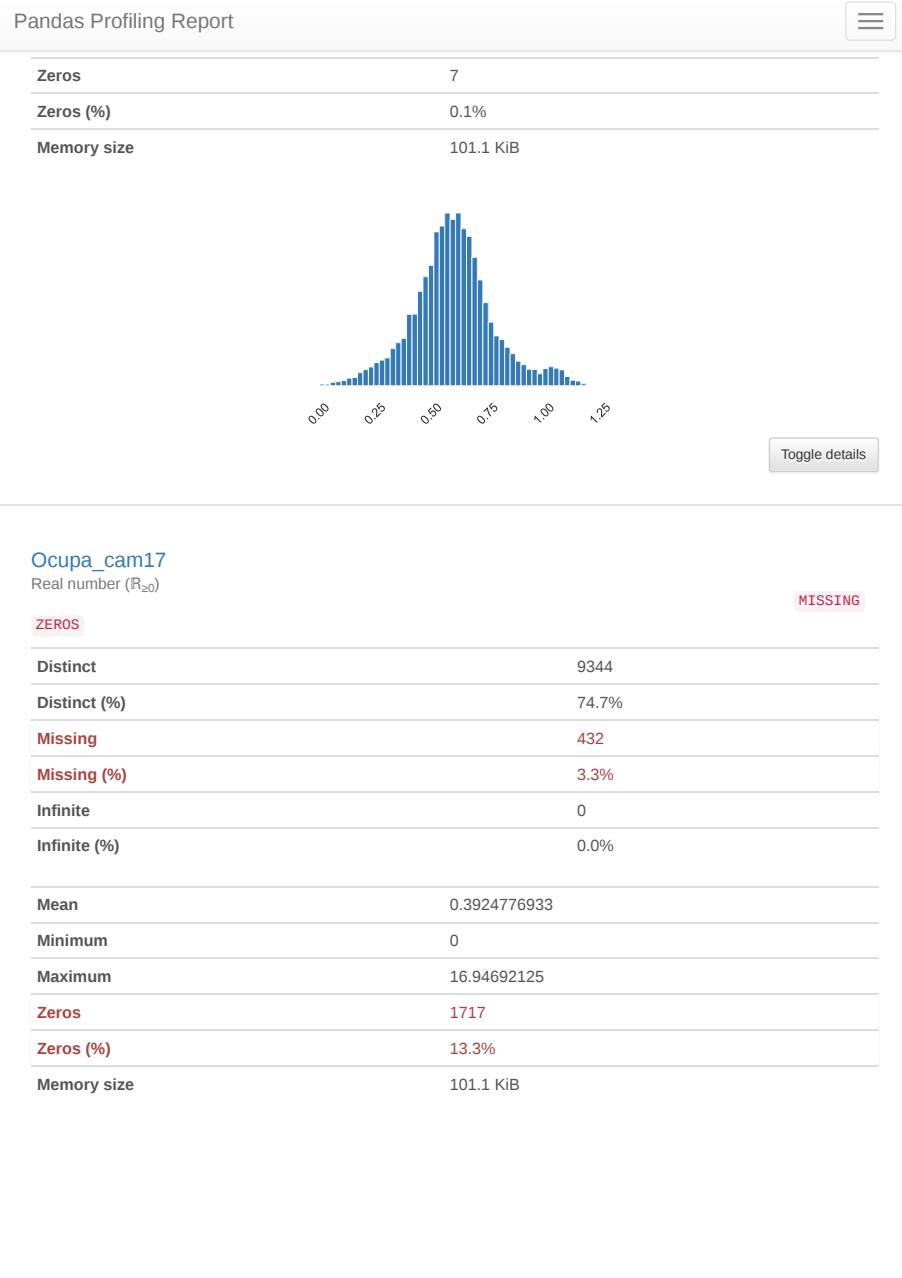






Pandas Profiling Report

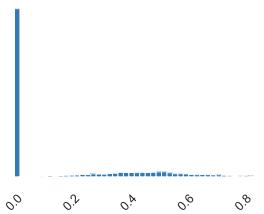




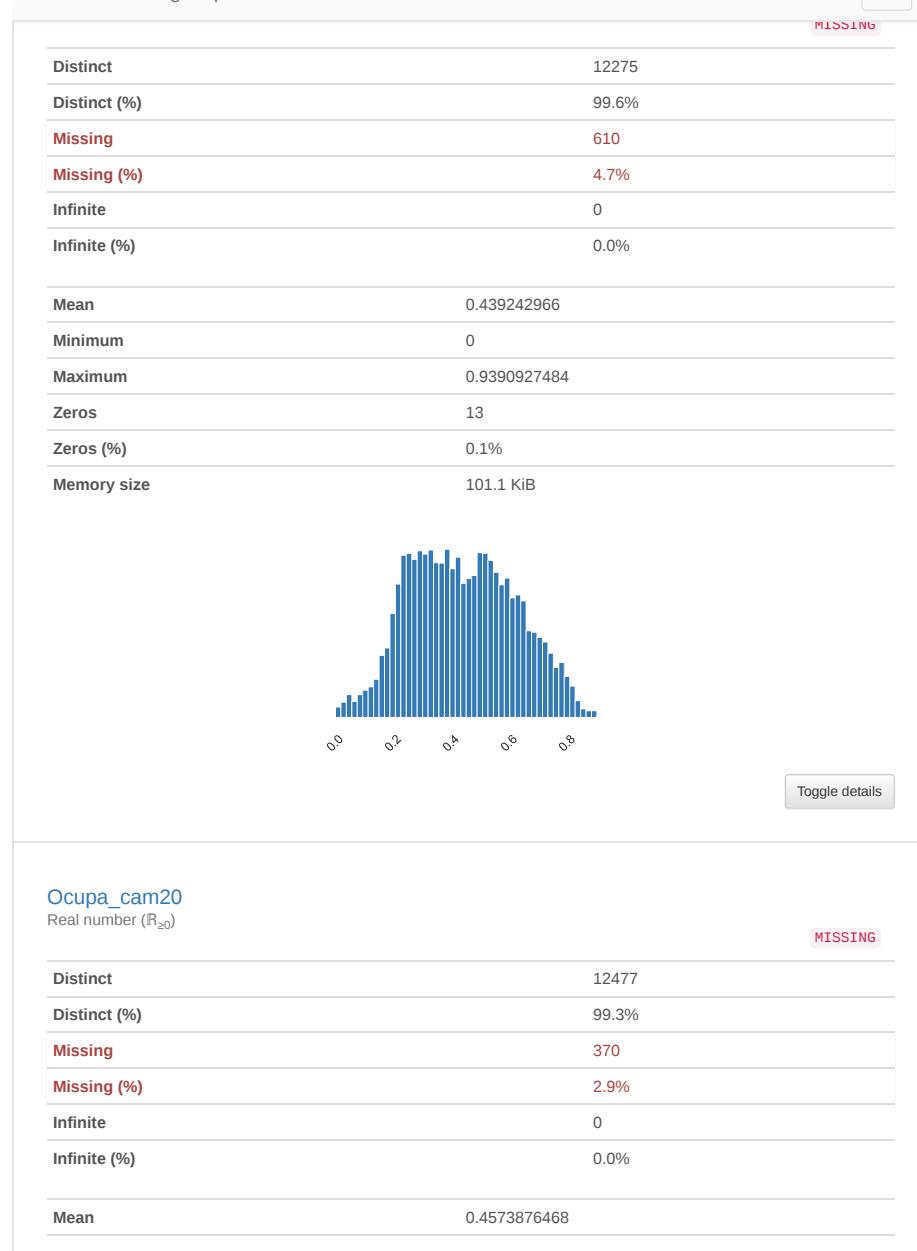
Pandas Profiling Report

[Toggle details](#)**Ocupa_cam18**Real number ($\mathbb{R}_{\geq 0}$)[MISSING](#)

Distinct	5029
Distinct (%)	40.3%
Missing	444
Missing (%)	3.4%
Infinite	0
Infinite (%)	0.0%
Mean	0.1881336399
Minimum	0
Maximum	0.9509154101
Zeros	4
Zeros (%)	< 0.1%
Memory size	101.1 KiB

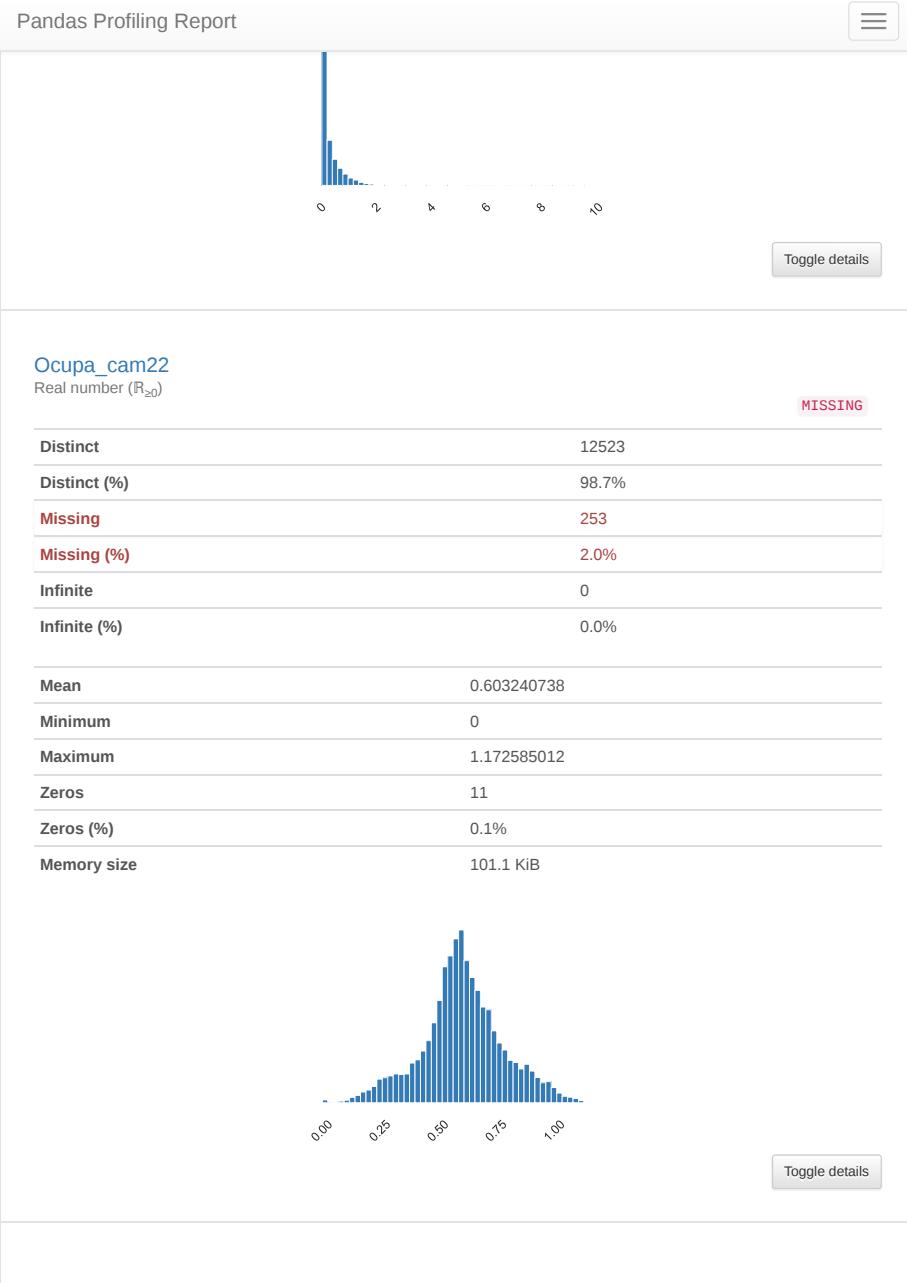
[Toggle details](#)

Pandas Profiling Report

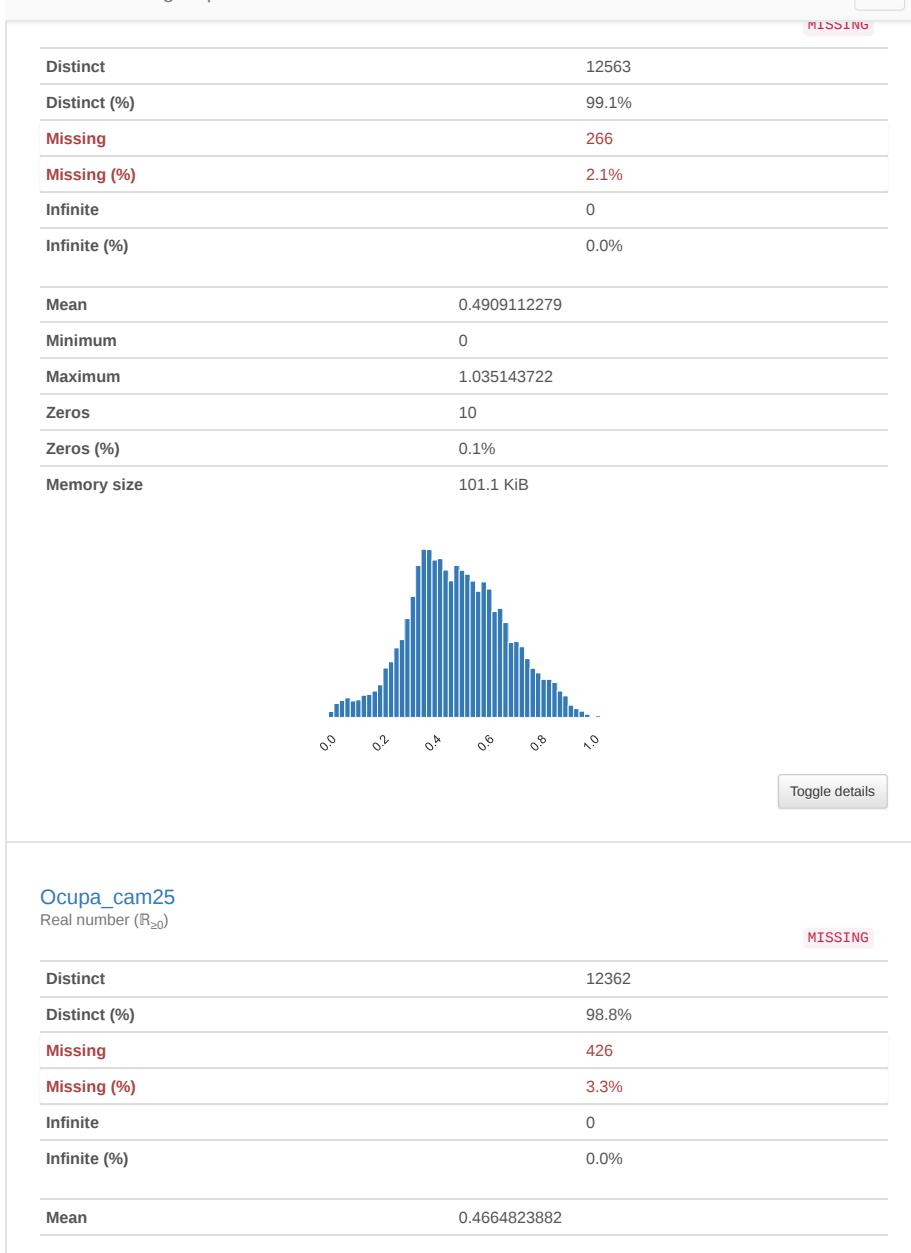


Pandas Profiling Report





Pandas Profiling Report

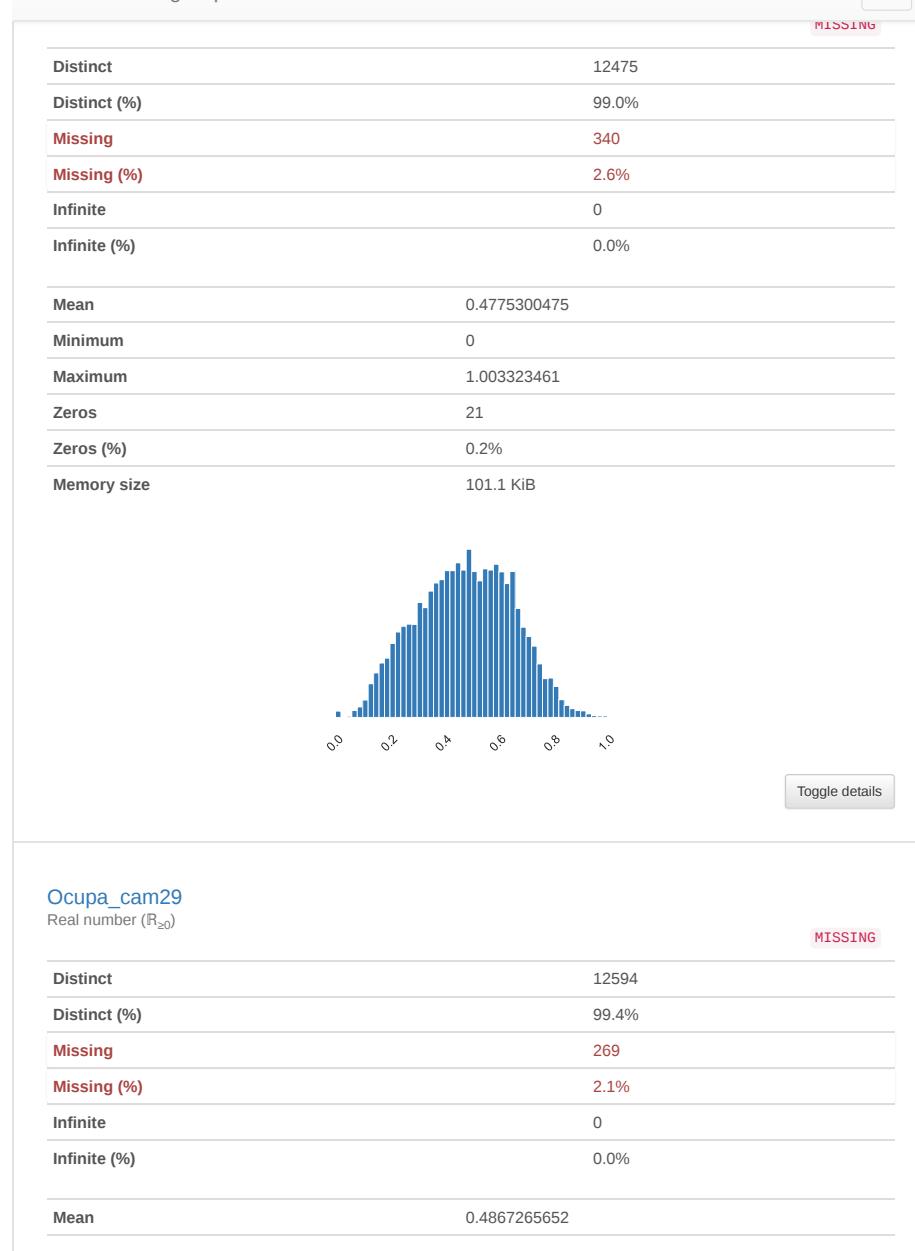




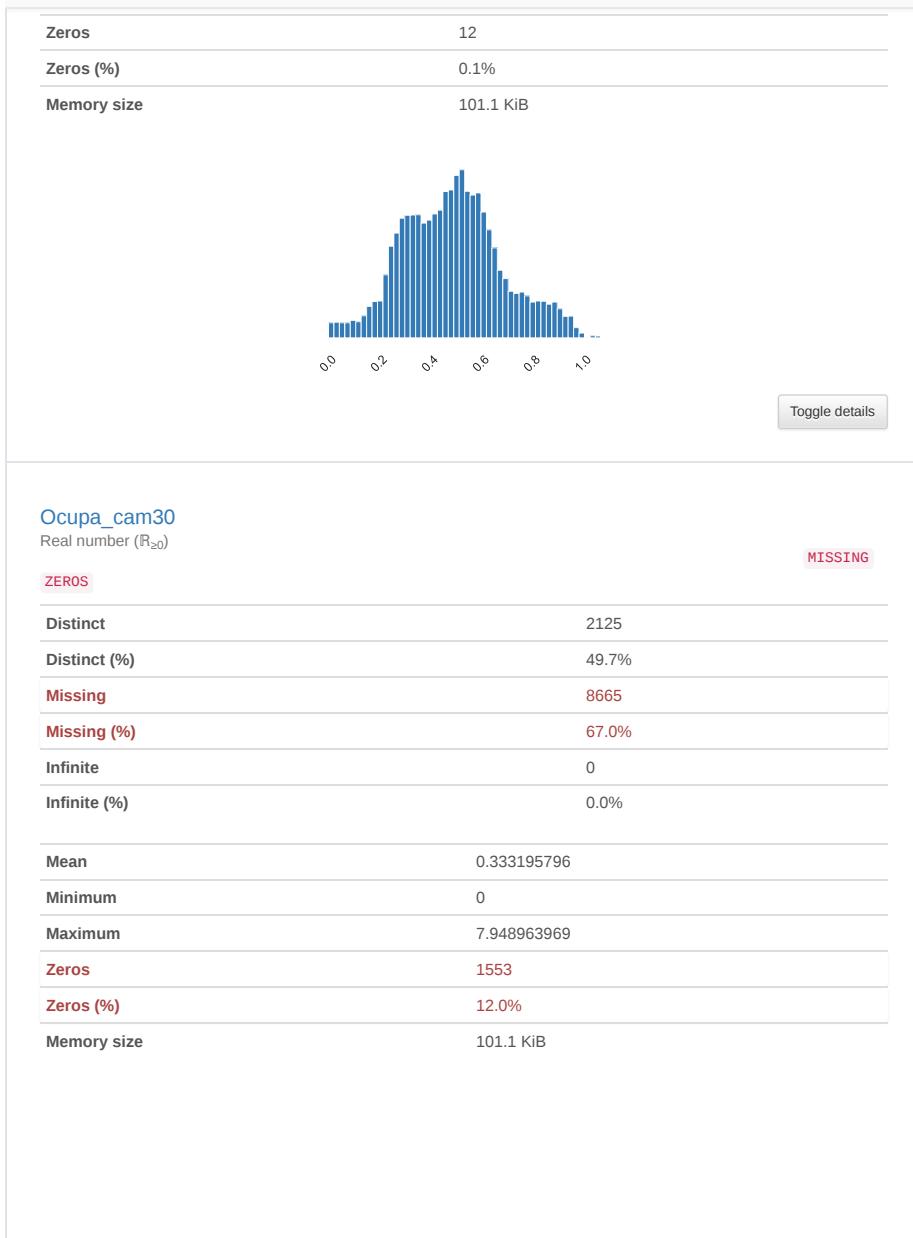
Pandas Profiling Report

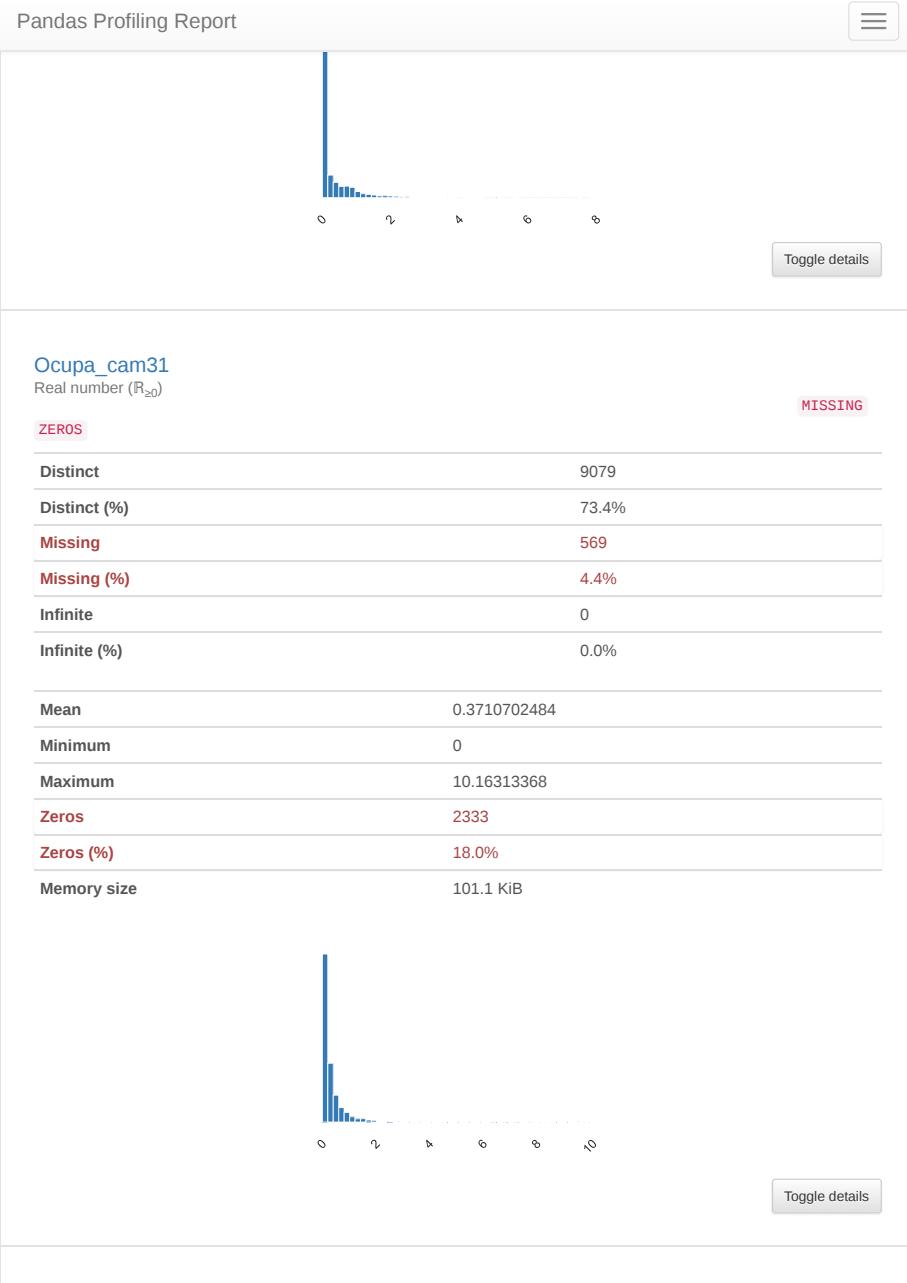
[Toggle details](#)**Ocupa_cam27**Real number ($\mathbb{R}_{\geq 0}$)[MISSING](#)**ZEROS****Distinct** 9754**Distinct (%)** 79.2%**Missing** 618**Missing (%)** 4.8%**Infinite** 0**Infinite (%)** 0.0%**Mean** 0.3729418531**Minimum** 0**Maximum** 9.872306984**Zeros** 670**Zeros (%)** 5.2%**Memory size** 101.1 KiB[Toggle details](#)

Pandas Profiling Report

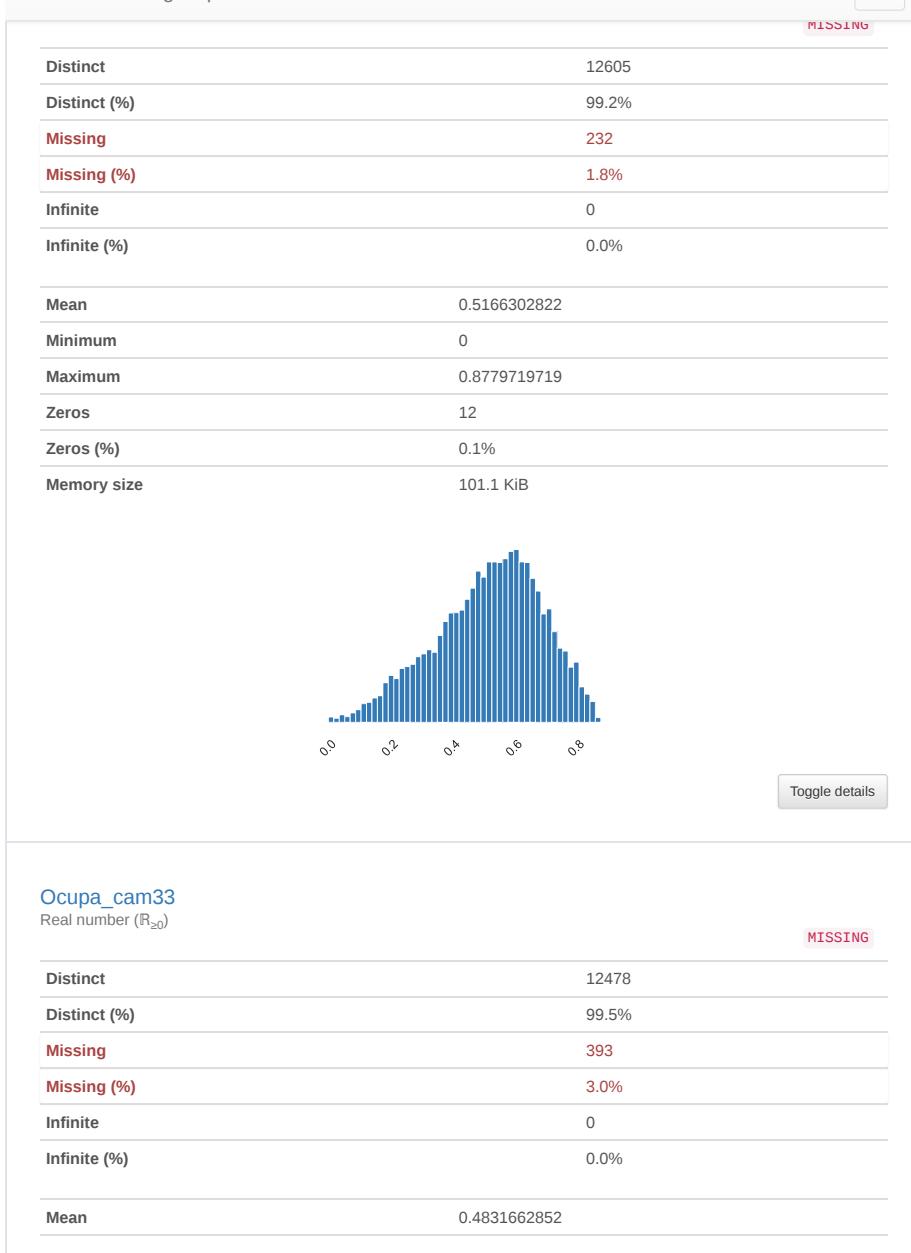


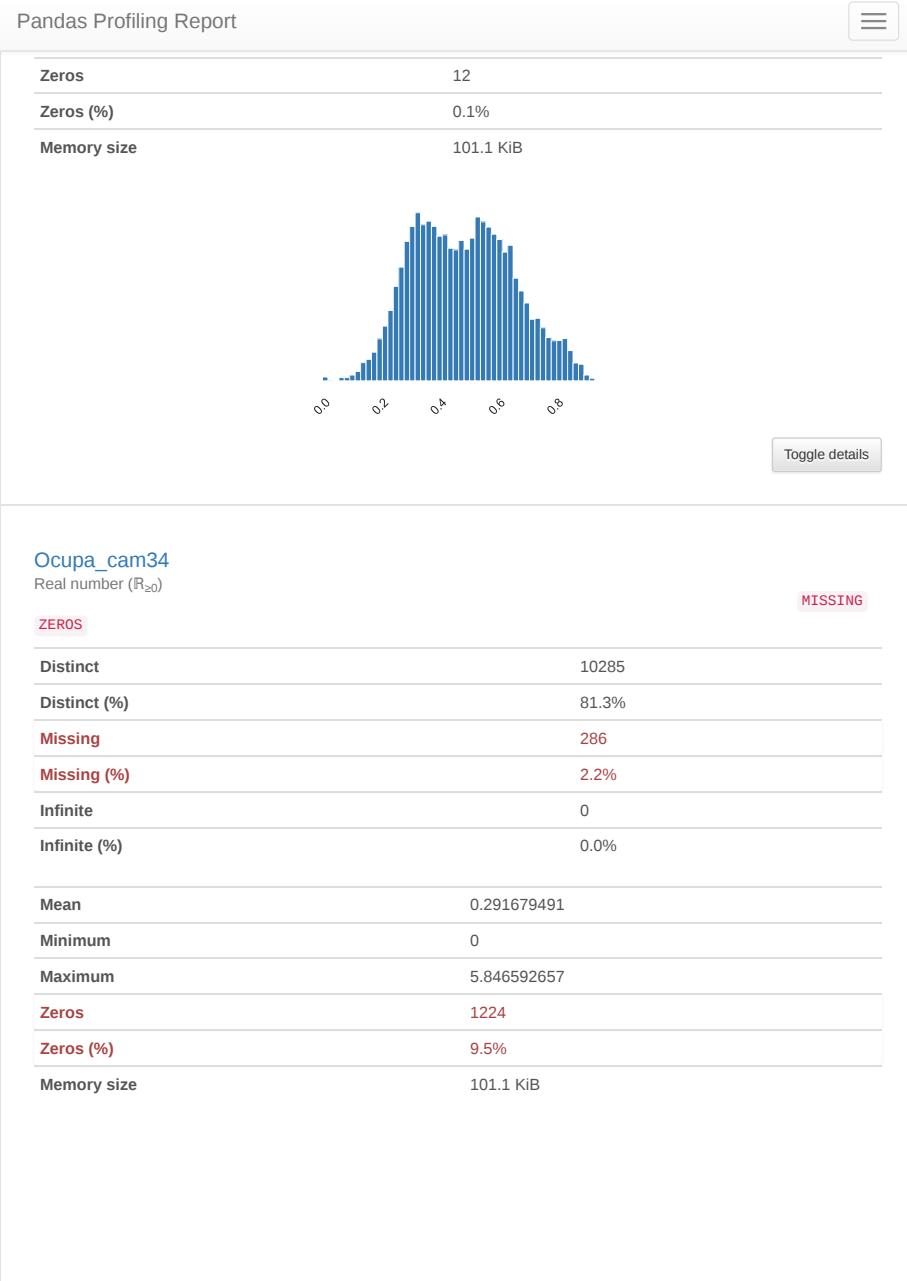
Pandas Profiling Report



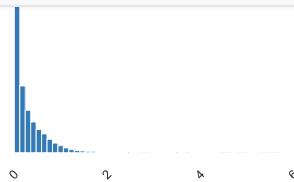


Pandas Profiling Report

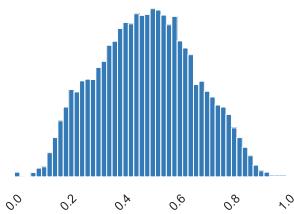




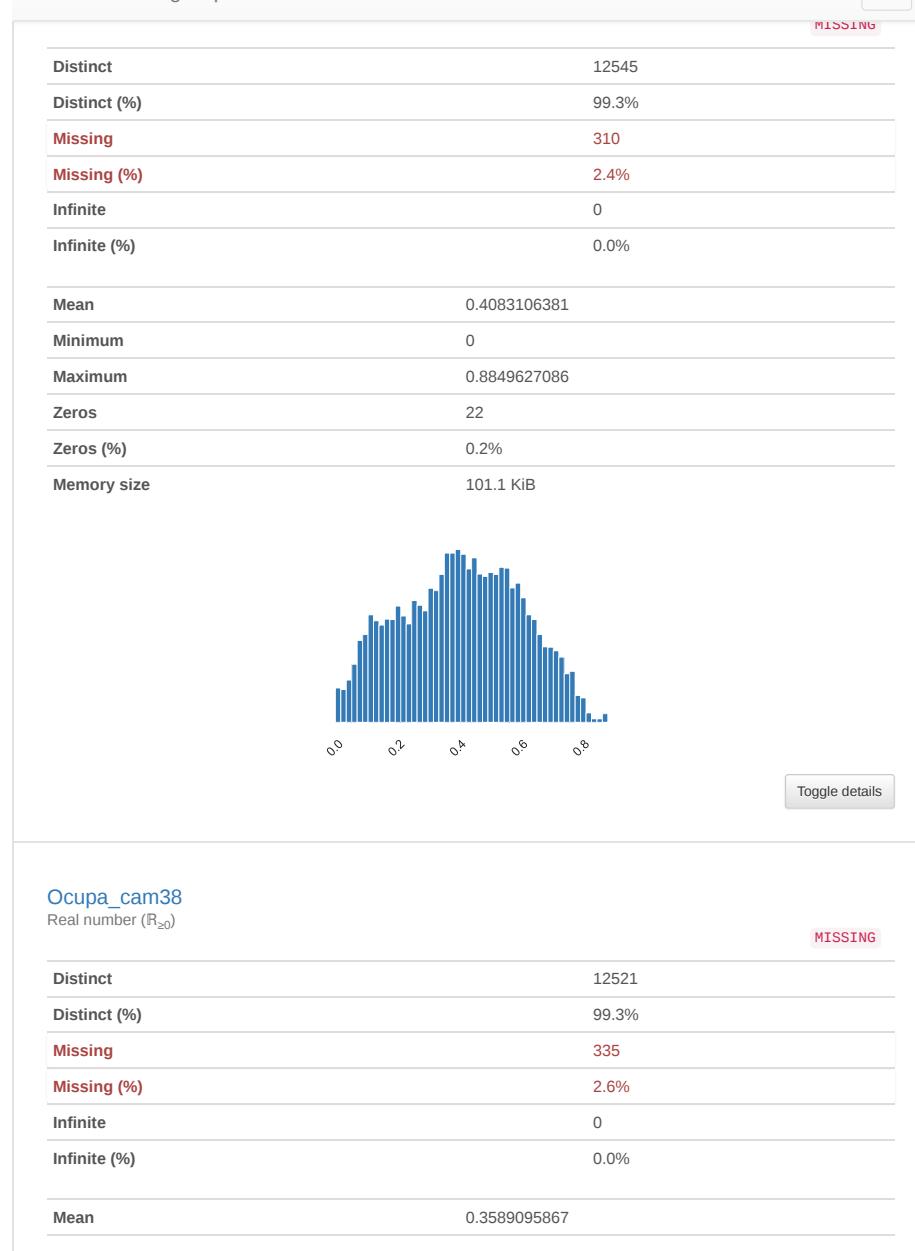
Pandas Profiling Report

[Toggle details](#)**Ocupa_cam35**Real number ($\mathbb{R}_{\geq 0}$)**MISSING**

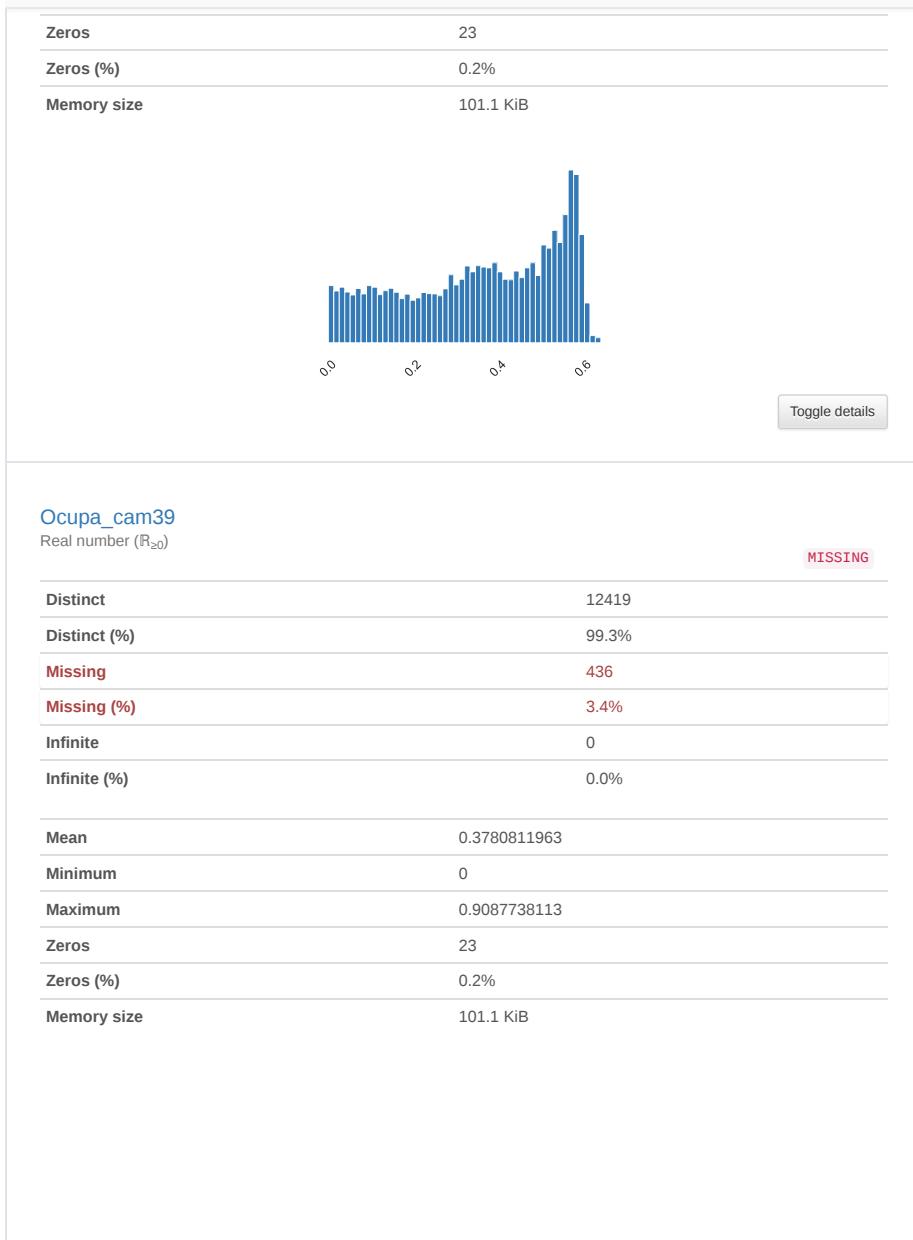
Distinct	11938
Distinct (%)	99.3%
Missing	920
Missing (%)	7.1%
Infinite	0
Infinite (%)	0.0%
Mean	0.4850186712
Minimum	0
Maximum	0.9955271919
Zeros	14
Zeros (%)	0.1%
Memory size	101.1 KiB

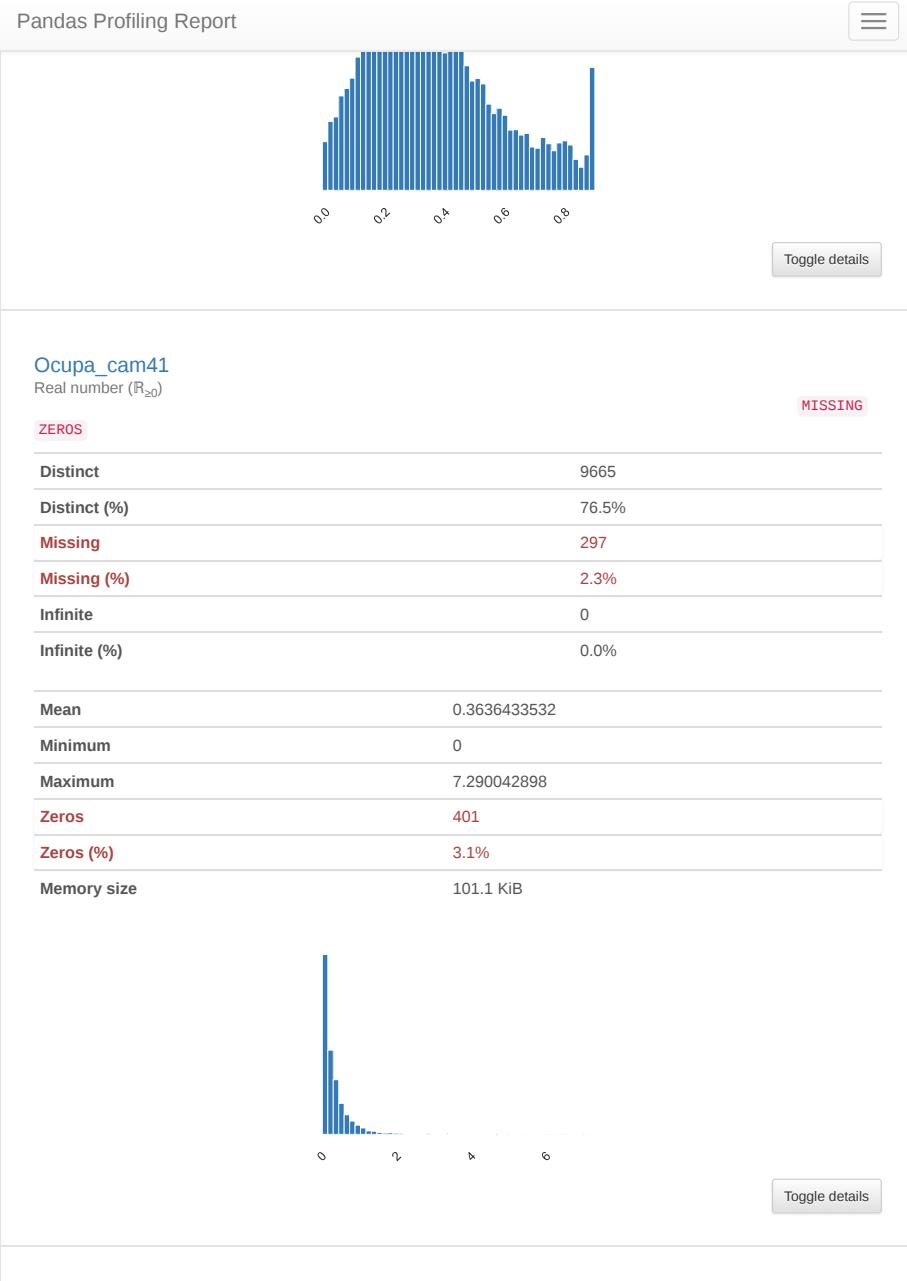
[Toggle details](#)

Pandas Profiling Report

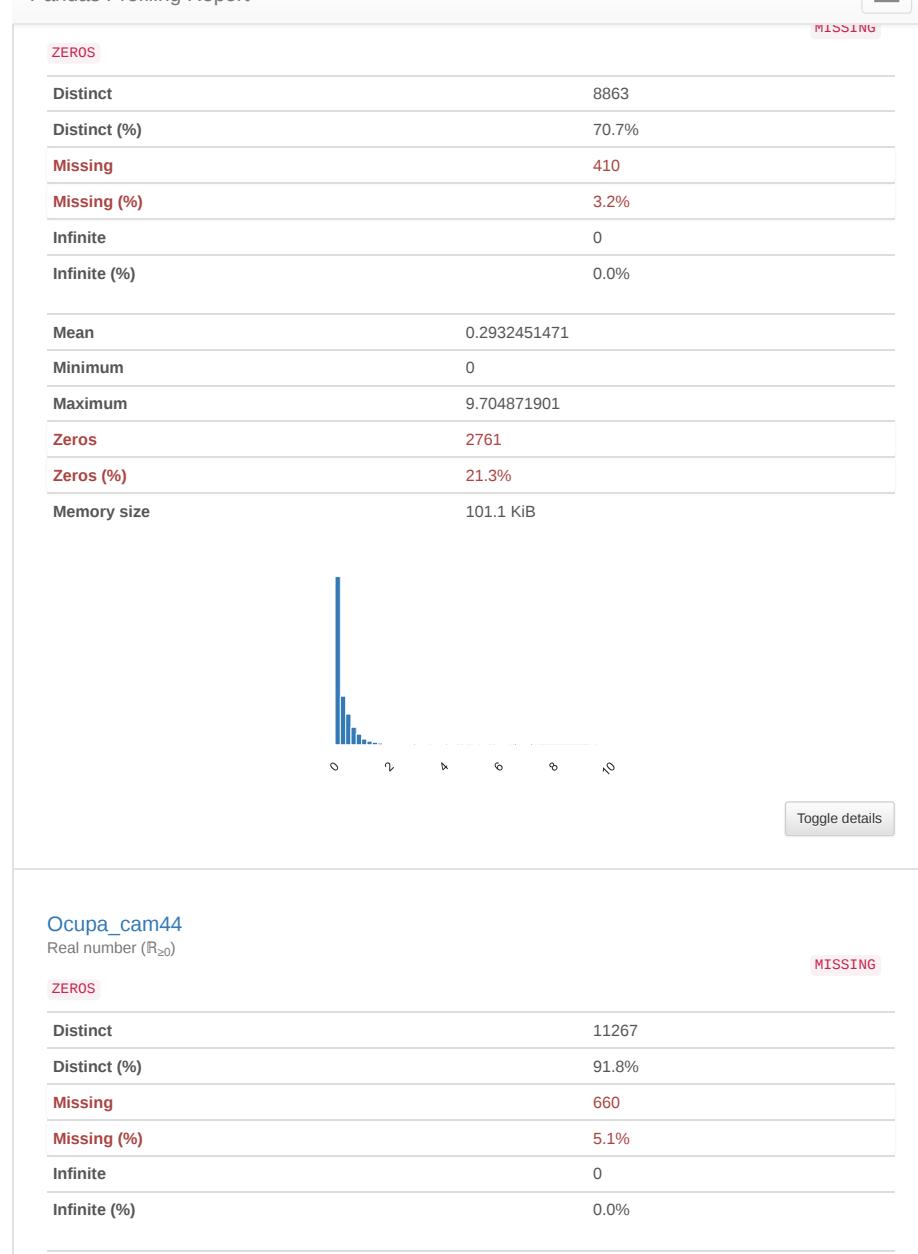


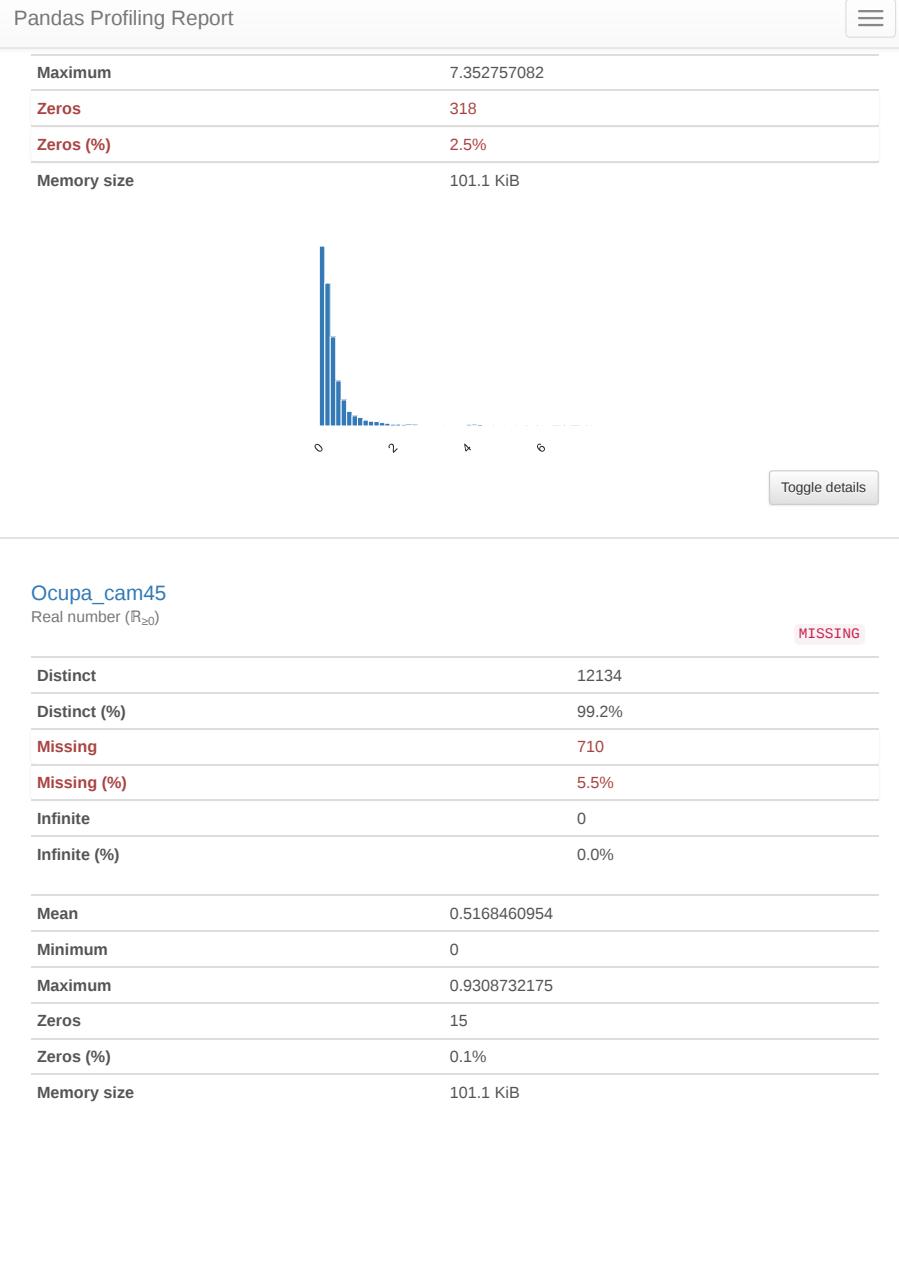
Pandas Profiling Report



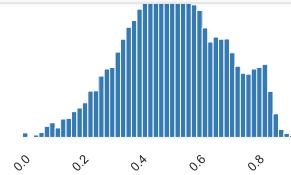


Pandas Profiling Report





Pandas Profiling Report

[Toggle details](#)**Ocupa_cam46**Real number ($\mathbb{R}_{\geq 0}$)[MISSING](#)[ZEROS](#)

Distinct	10229
----------	-------

Distinct (%)	82.3%
--------------	-------

Missing	515
---------	-----

Missing (%)	4.0%
-------------	------

Infinite	0
----------	---

Infinite (%)	0.0%
--------------	------

Mean	0.398553939
------	-------------

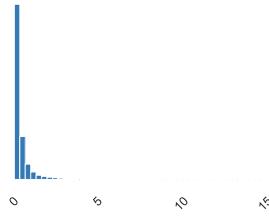
Minimum	0
---------	---

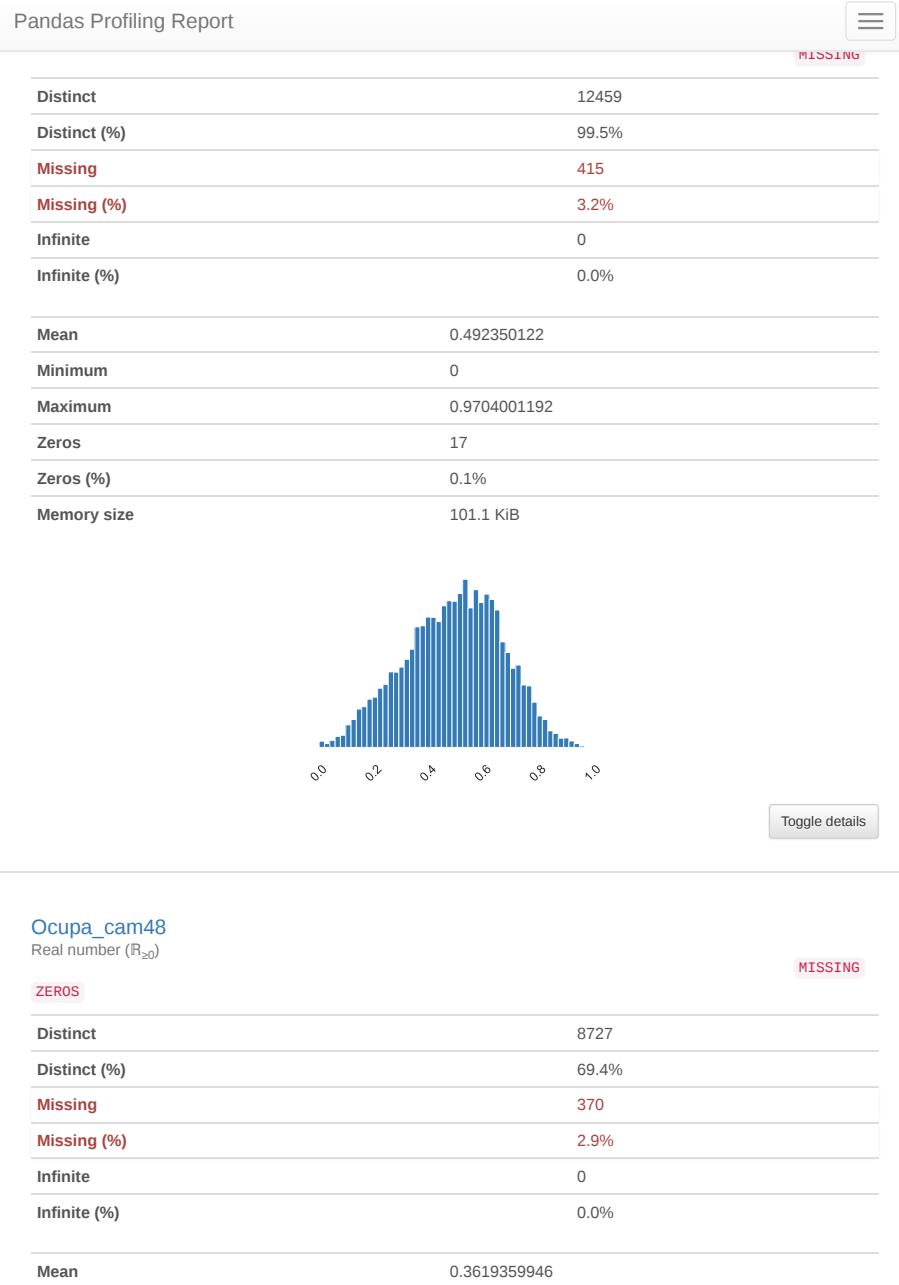
Maximum	16.26980093
---------	-------------

Zeros	915
-------	-----

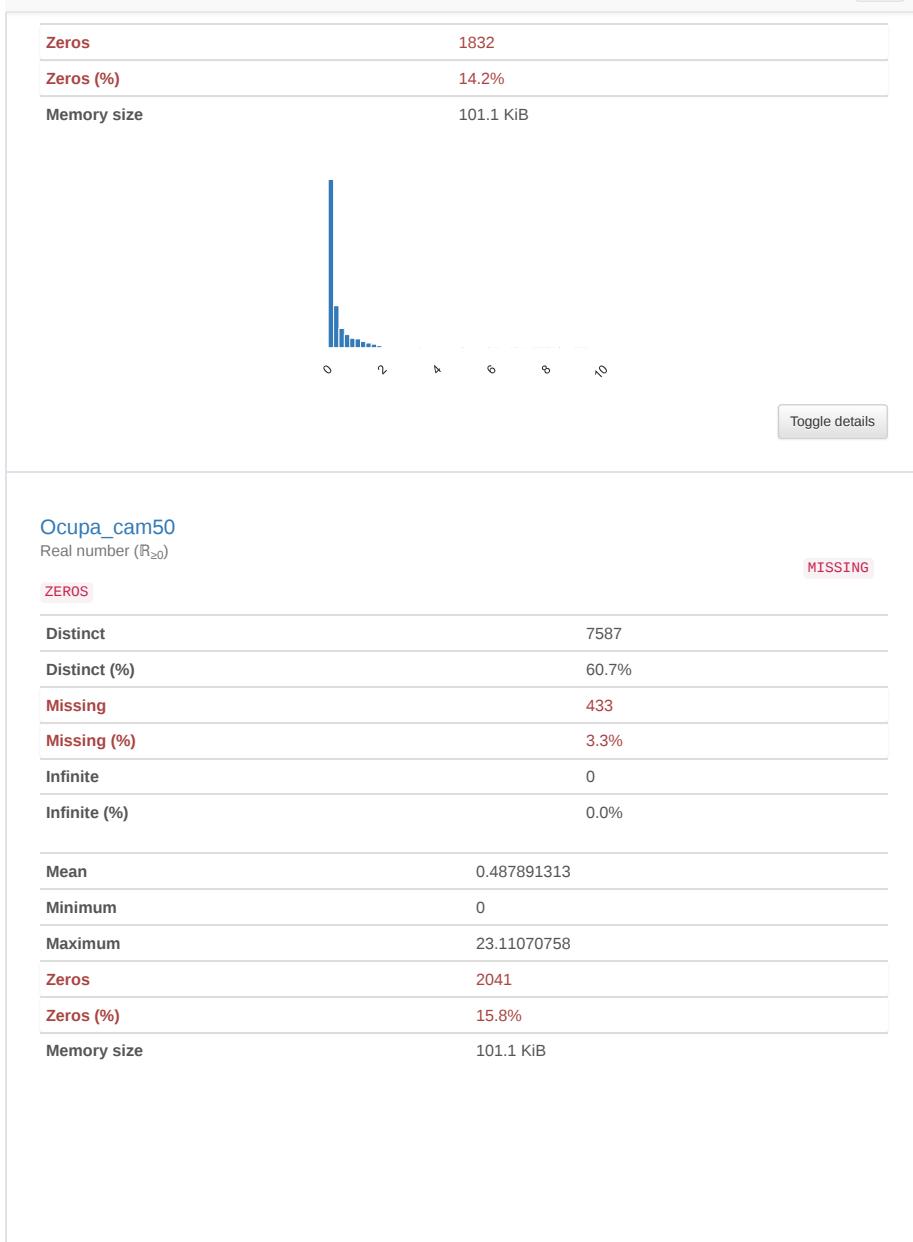
Zeros (%)	7.1%
-----------	------

Memory size	101.1 KiB
-------------	-----------

[Toggle details](#)

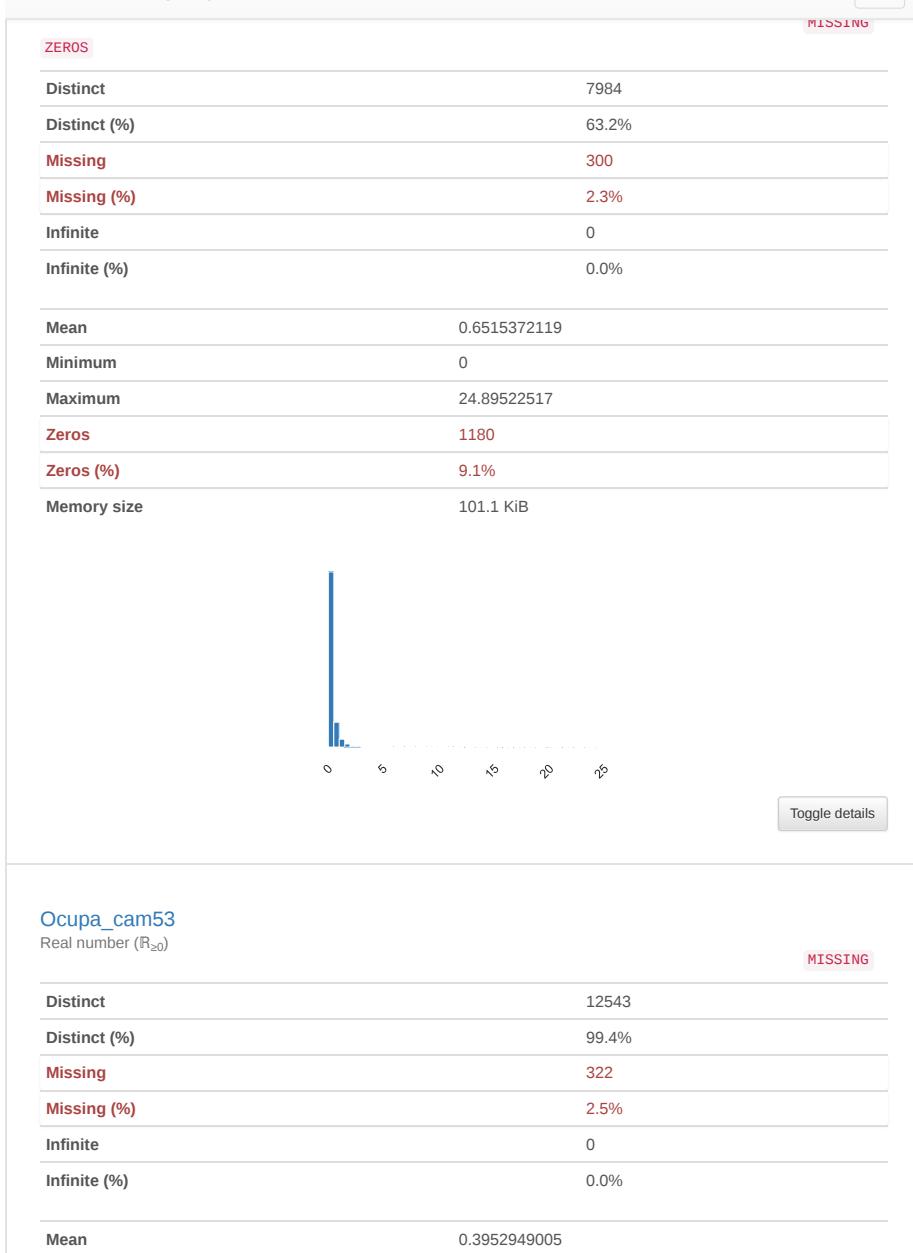


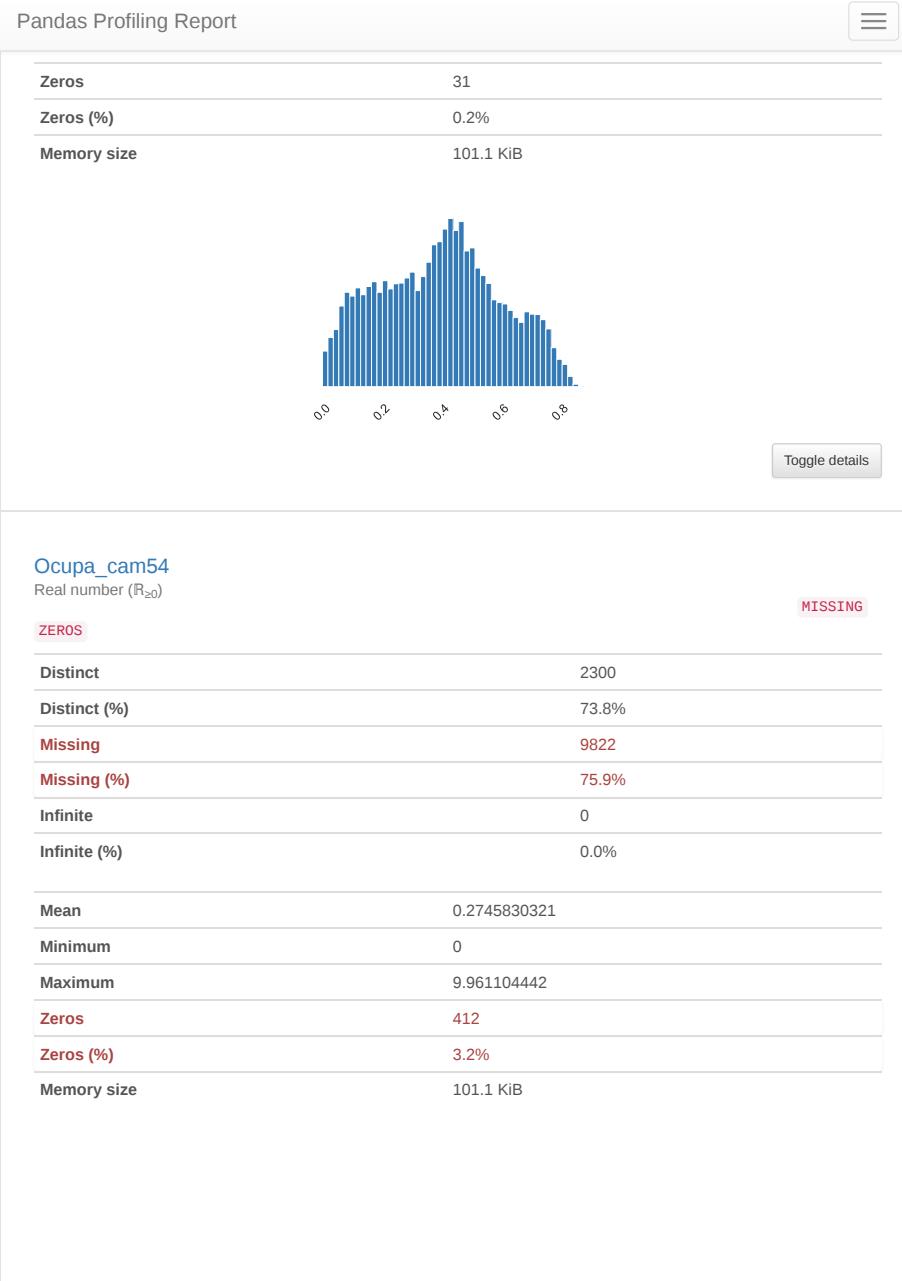
Pandas Profiling Report





Pandas Profiling Report



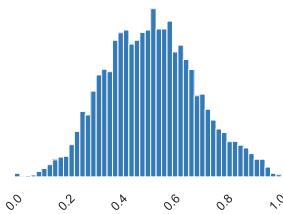


Pandas Profiling Report

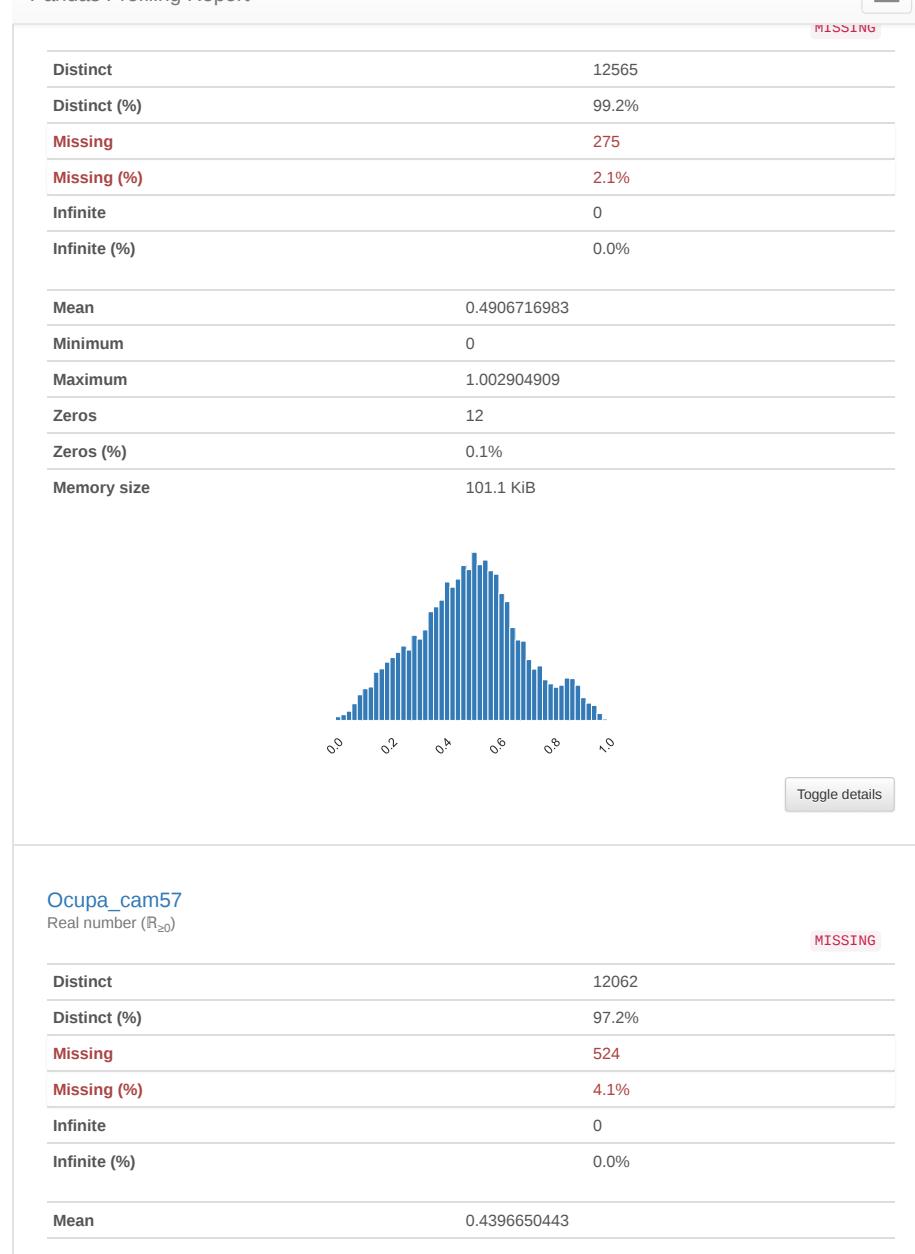
[Toggle details](#)**Ocupa_cam55**Real number ($\mathbb{R}_{\geq 0}$)

MISSING

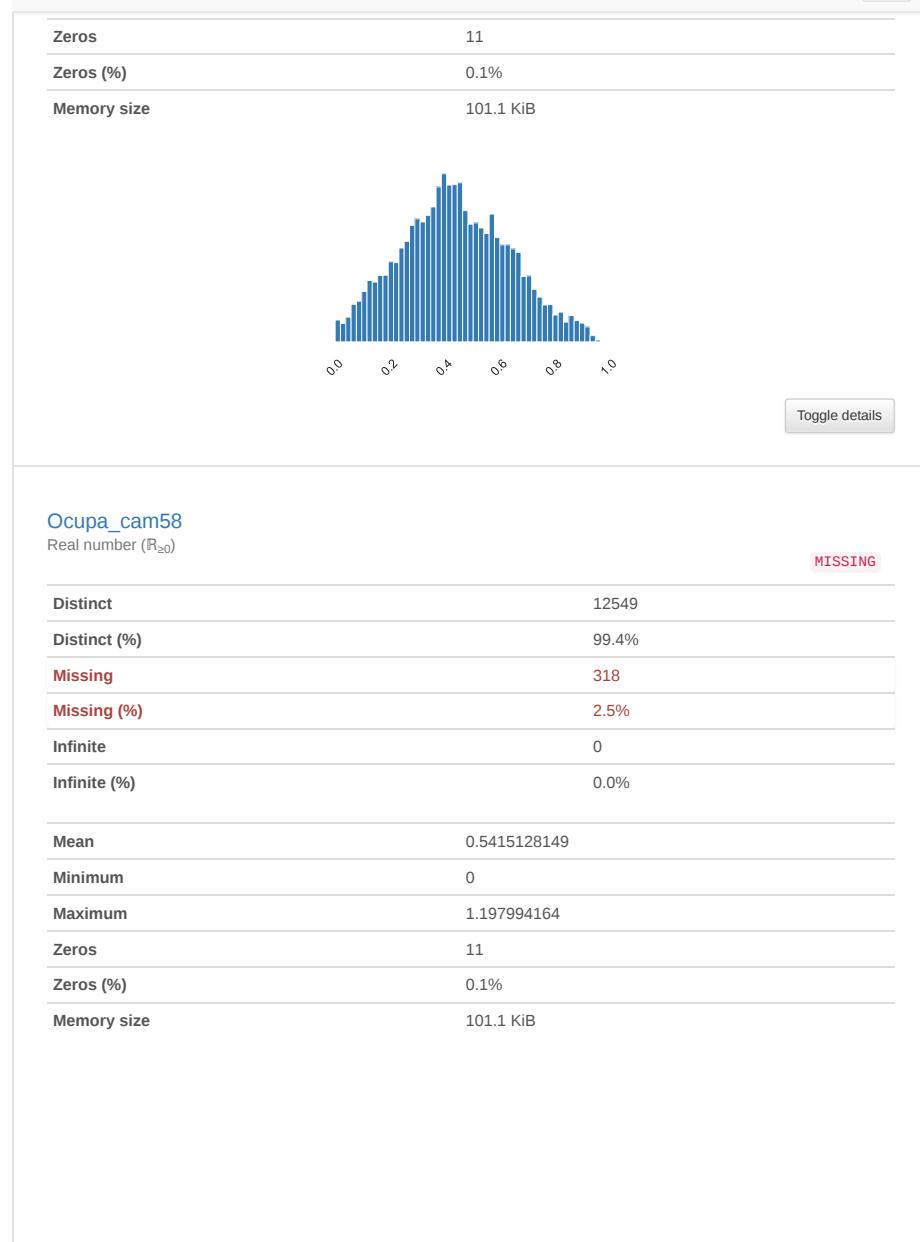
Distinct	12398
Distinct (%)	99.5%
Missing	473
Missing (%)	3.7%
Infinite	0
Infinite (%)	0.0%
Mean	0.5199983942
Minimum	0
Maximum	1.034438061
Zeros	12
Zeros (%)	0.1%
Memory size	101.1 KiB

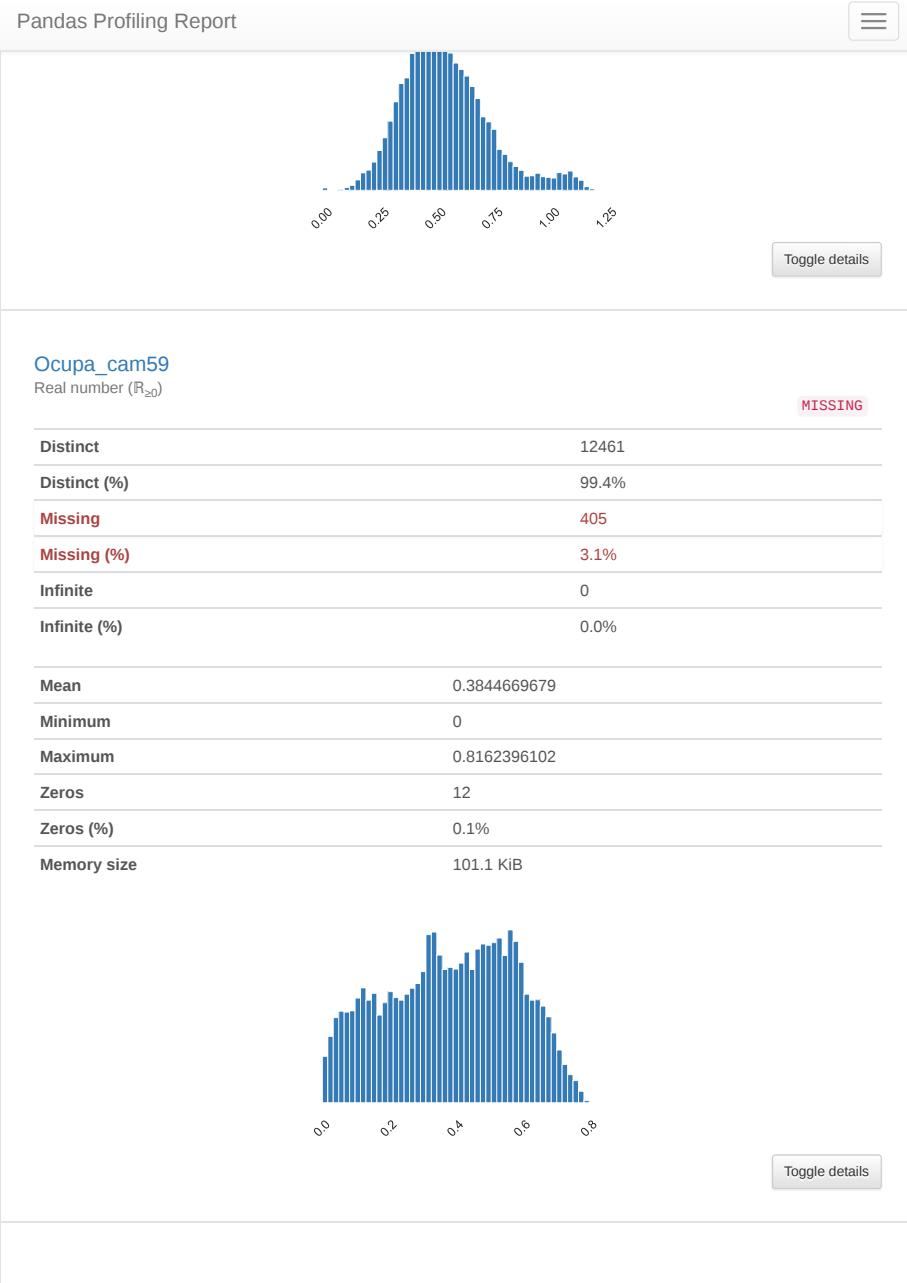
[Toggle details](#)

Pandas Profiling Report



Pandas Profiling Report





Pandas Profiling Report



MISSING

ZEROS

Distinct	12256
----------	-------

Distinct (%)	97.4%
--------------	-------

Missing	352
---------	-----

Missing (%)	2.7%
-------------	------

Infinite	0
----------	---

Infinite (%)	0.0%
--------------	------

Mean	0.3765580877
------	--------------

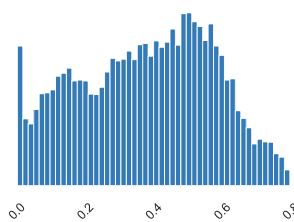
Minimum	0
---------	---

Maximum	0.7953733242
---------	--------------

Zeros	279
-------	-----

Zeros (%)	2.2%
-----------	------

Memory size	101.1 KiB
-------------	-----------



Toggle details

Ocupa_cam61

Real number ($\mathbb{R}_{\geq 0}$)

MISSING

Distinct	12253
----------	-------

Distinct (%)	99.2%
--------------	-------

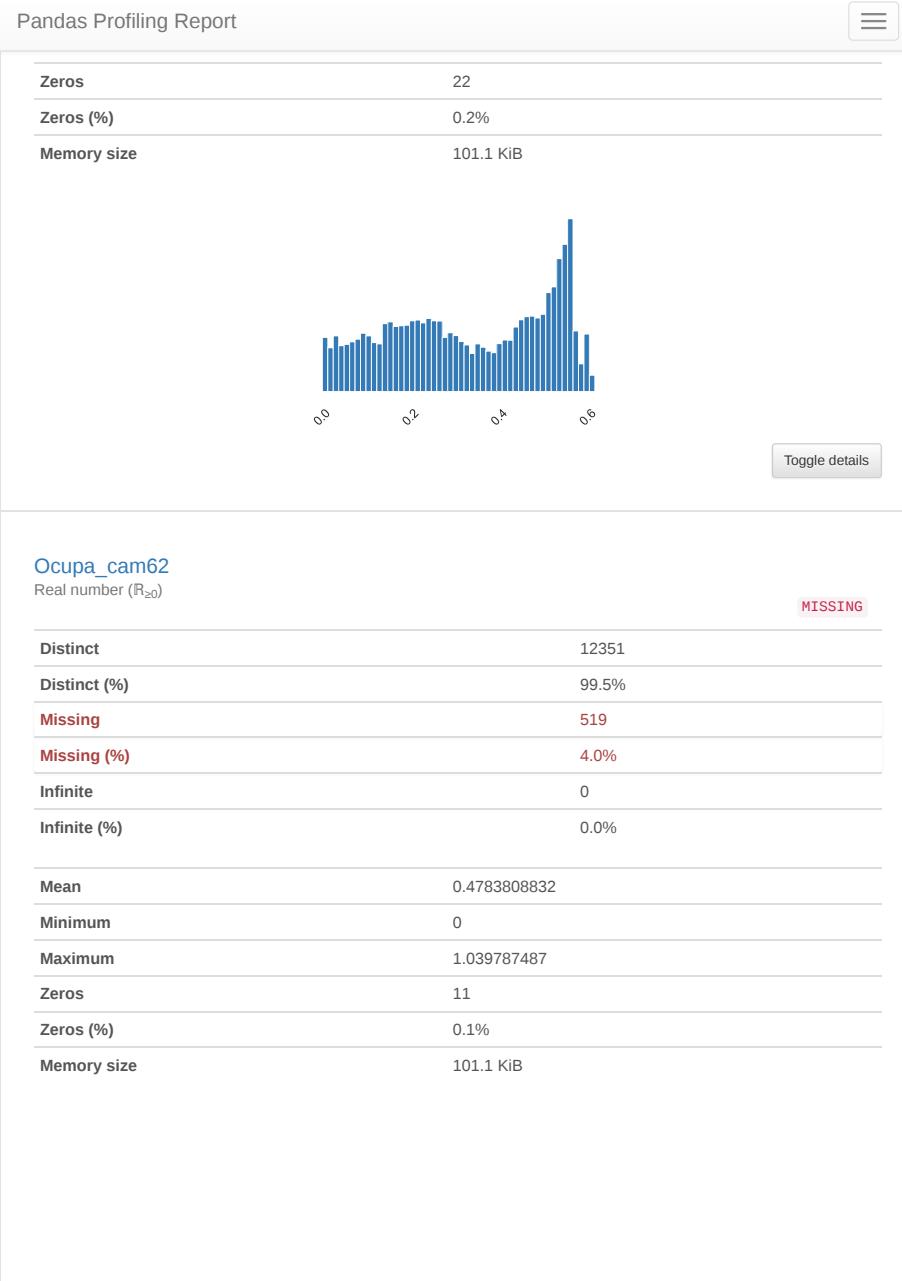
Missing	583
---------	-----

Missing (%)	4.5%
-------------	------

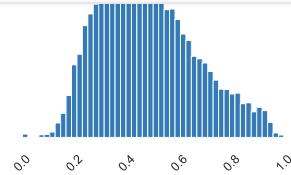
Infinite	0
----------	---

Infinite (%)	0.0%
--------------	------

Mean	0.3323064051
------	--------------

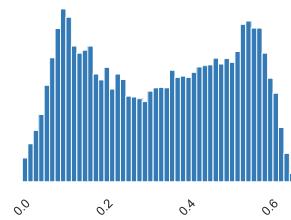


Pandas Profiling Report

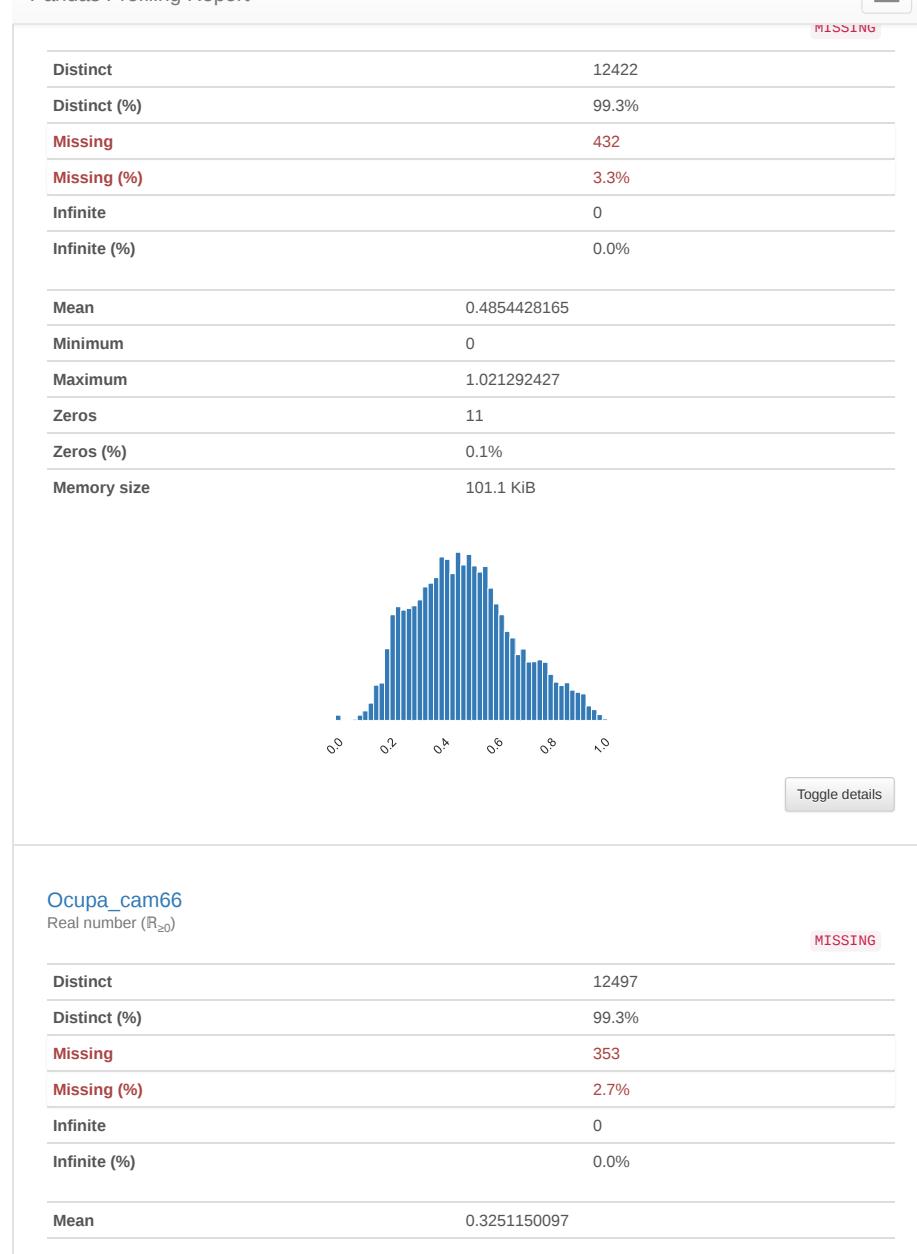
[Toggle details](#)**Ocupa_cam64**Real number ($\mathbb{R}_{\geq 0}$)

MISSING

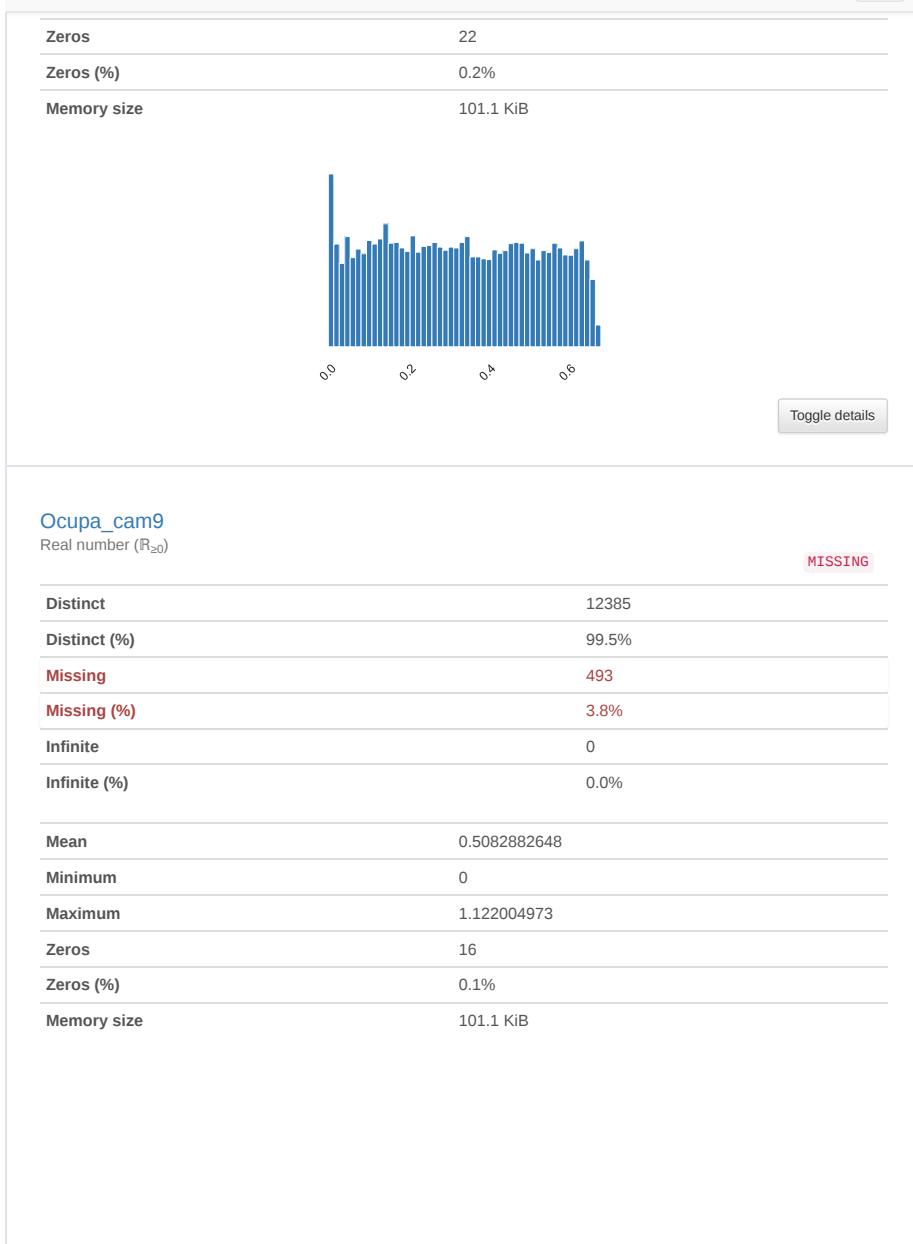
Distinct	12629
Distinct (%)	99.7%
Missing	267
Missing (%)	2.1%
Infinite	0
Infinite (%)	0.0%
Mean	0.3314207381
Minimum	0
Maximum	0.6587019007
Zeros	11
Zeros (%)	0.1%
Memory size	101.1 KiB

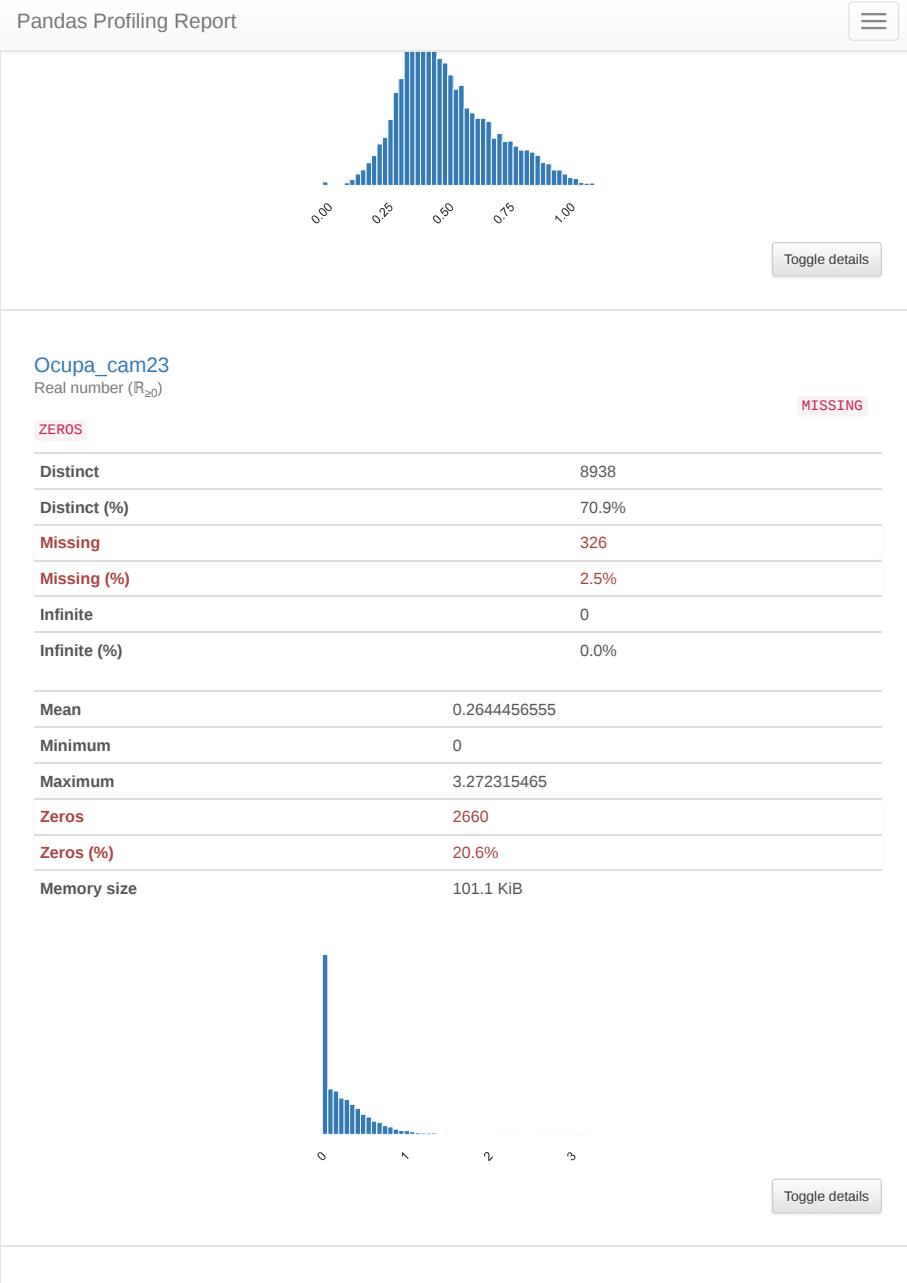
[Toggle details](#)

Pandas Profiling Report

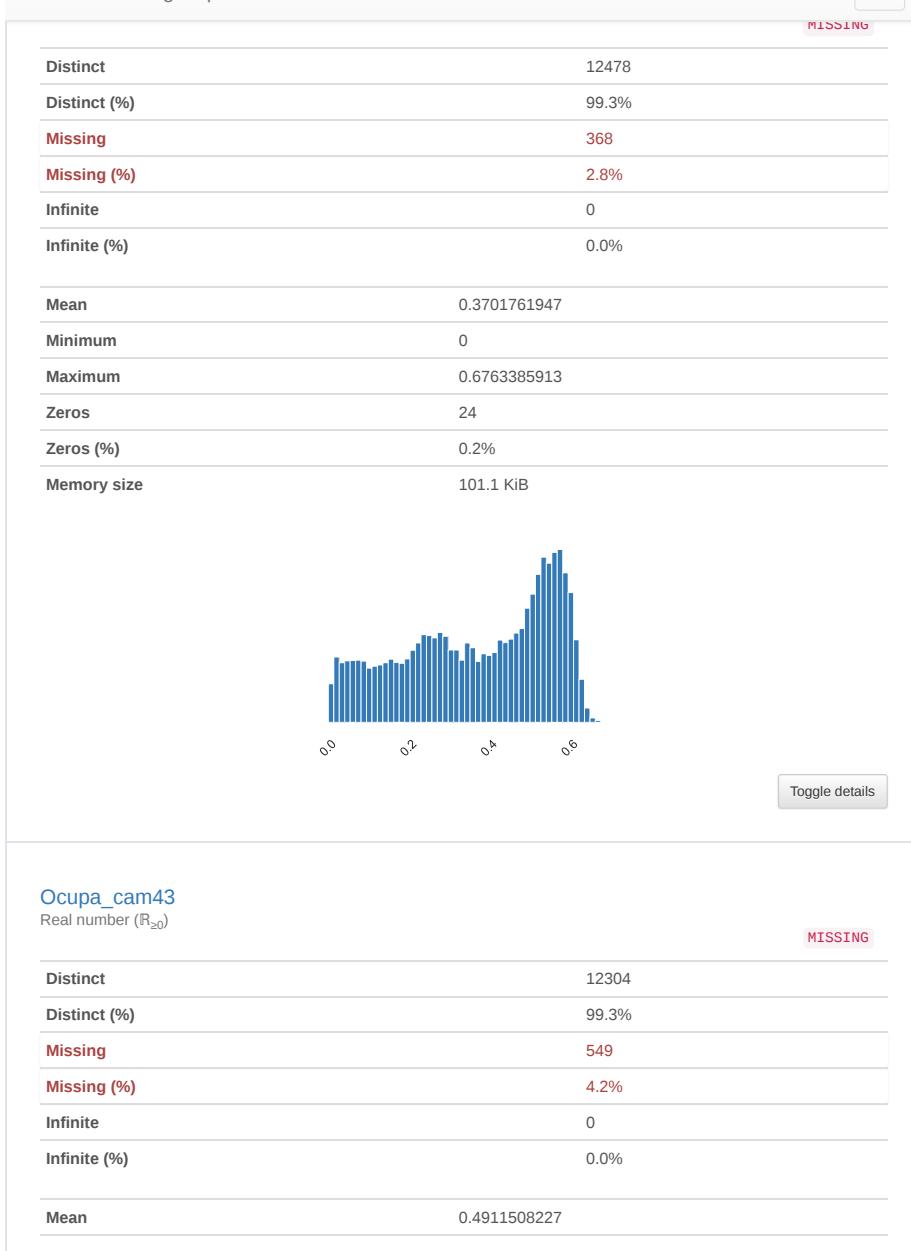


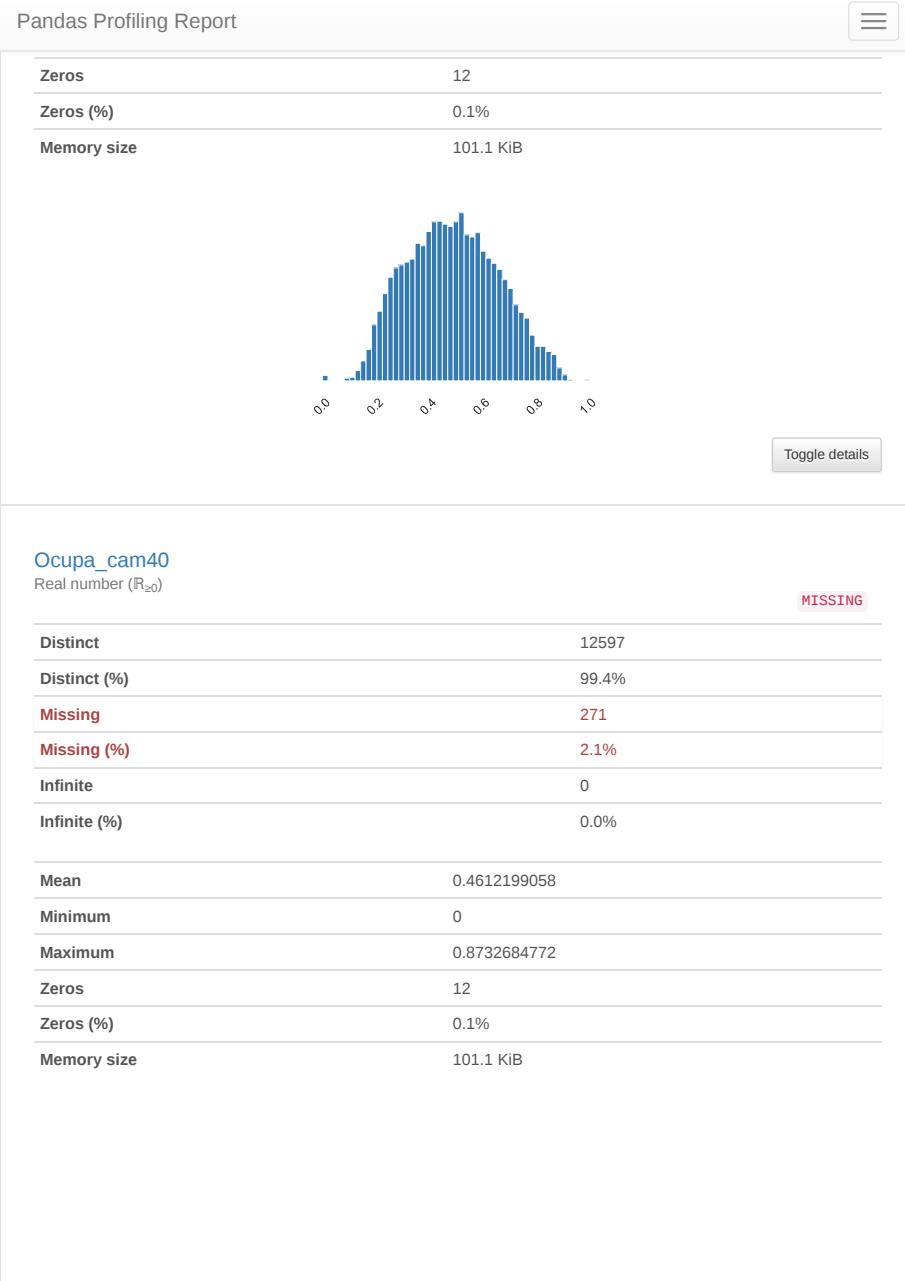
Pandas Profiling Report



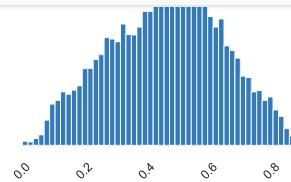


Pandas Profiling Report



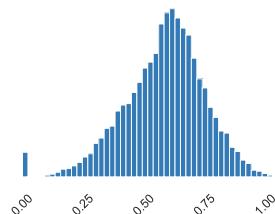


Pandas Profiling Report

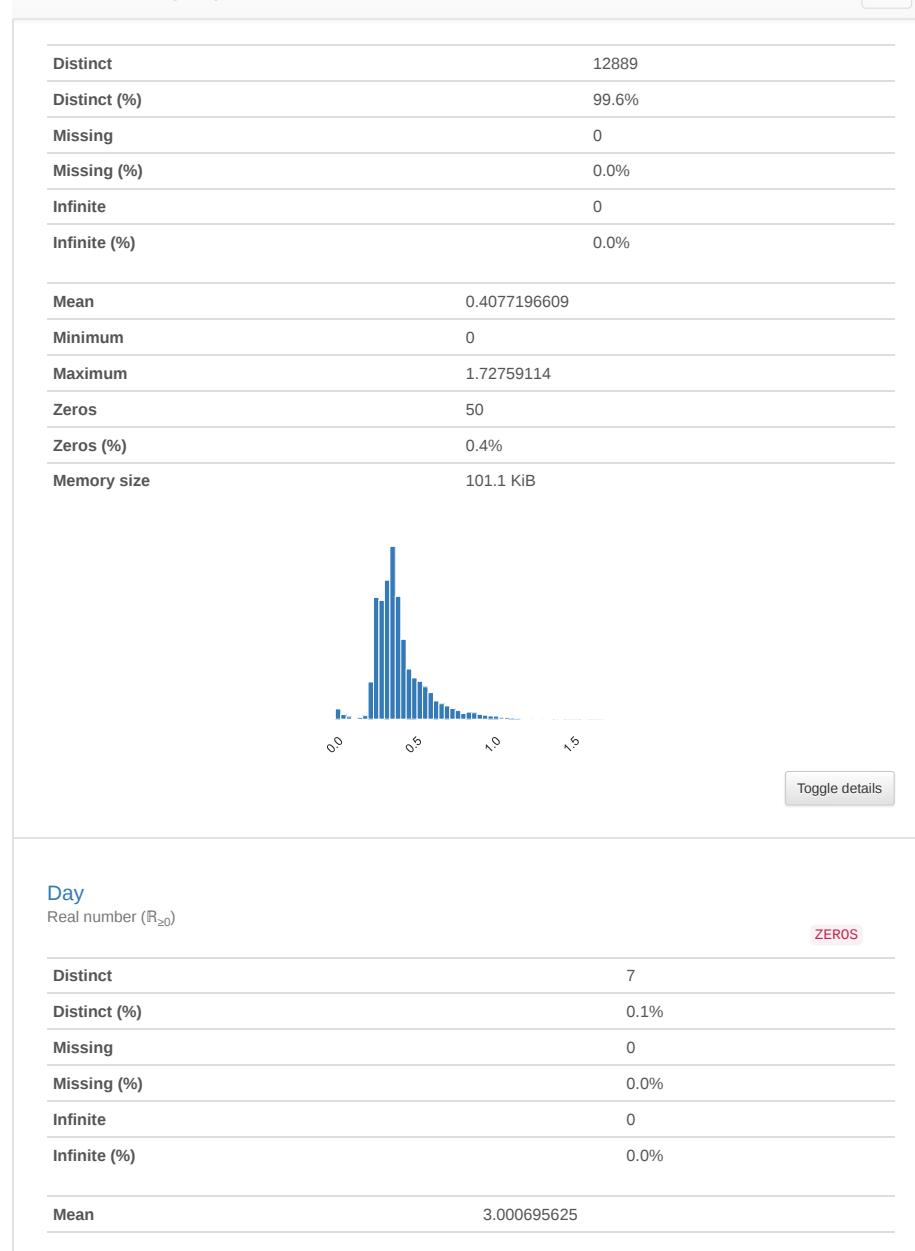
[Toggle details](#)**Ocupa_cam49**Real number ($\mathbb{R}_{\geq 0}$)

MISSING

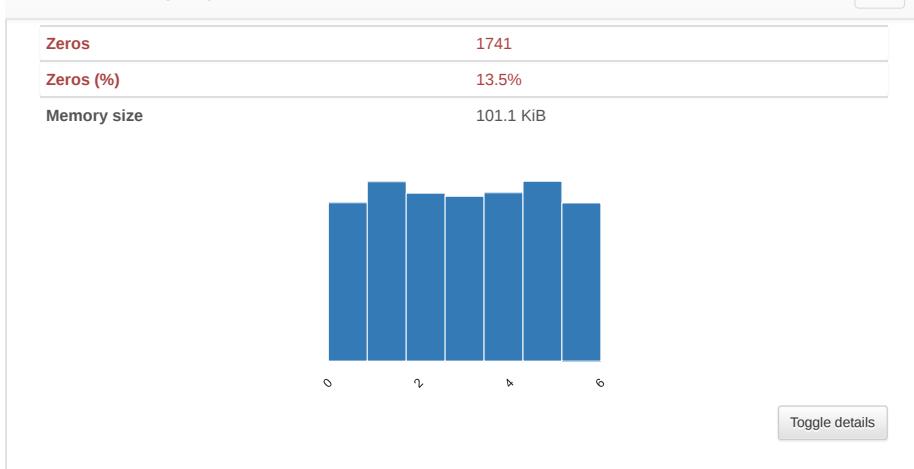
Distinct	12423
Distinct (%)	98.9%
Missing	371
Missing (%)	2.9%
Infinite	0
Infinite (%)	0.0%
Mean	0.5836603811
Minimum	0
Maximum	1.119790663
Zeros	119
Zeros (%)	0.9%
Memory size	101.1 KiB

[Toggle details](#)

Pandas Profiling Report



Pandas Profiling Report



Apéndice B

Código

Para su descarga y/o visualización del código, se pone a disposición el siguiente repositorio en *GitHub*, https://github.com/RiickyR91/Analisis_Trafico, donde además del código, se puede tener acceso a un dataset para realizar pruebas.