

**Reconocimiento de patrones**

**Proyecto**

Determinar el tipo de fenómeno natural

Profesor:

Dr. Ramón Soto de la Cruz

**Alumno:**

Raúl Francisco Pérez Rodríguez

**Diciembre 2017**

**Contenido**

Introducción

Descripción de los datos

Limpieza de los datos

Valores faltantes

Imputación

Valores atípicos

Clustering

Dendrogramas

K-Means

ISODATA

Clasificación

K-Vecinos Próximos

Teorema de Bayes

Árbol de decisión

Máquinas de vectores de soporte

Redes Neuronales

**Introducción**

**Descripción de los datos**

Los datos fueron conseguidos de la página web kaggle.

* <https://www.kaggle.com/juliansimon/weather_madrid_lemd_1997_2015.csv>
* https://www.kaggle.com/grubenm/austin-weather

La unión de ambos conjuntos de datos después de hacer una conversión de medidas, da una cantidad de 8131 registros y 10 variables de información las cuales son:

1. TemperaturaMaxC
2. TemperaturaPromC
3. TemperaturaMinC
4. HumedadMax
5. HumedadProm
6. HumedadMin
7. VientoMaxKMXH
8. VientoPromKMXH
9. PrecipitacionMM
10. Eventos

**TemperaturaMaxC:** Describe la temperatura máxima en grados centígrados.

**TemperaturaPromC:** Describe la temperatura promedio en grados centígrados.

**TemperaturaMinC:** Describe la temperatura mínima en grados centígrados.

**HumedadMax:** Describe el porcentaje de humedad máxima.

**HumedadProm:** Describe el porcentaje de humedad promedio.

**HumedadMin:** Describe el porcentaje de humedad mínima.

**VientoMaxKMXH:** Describe la velocidad del viento máximo dado por kilómetros por hora.

**VientoPromKMXH:** Describe la velocidad del viento promedio dado por kilómetros por hora.

**PrecipitacionMM:** Describe la precipitación está dado por milímetros.

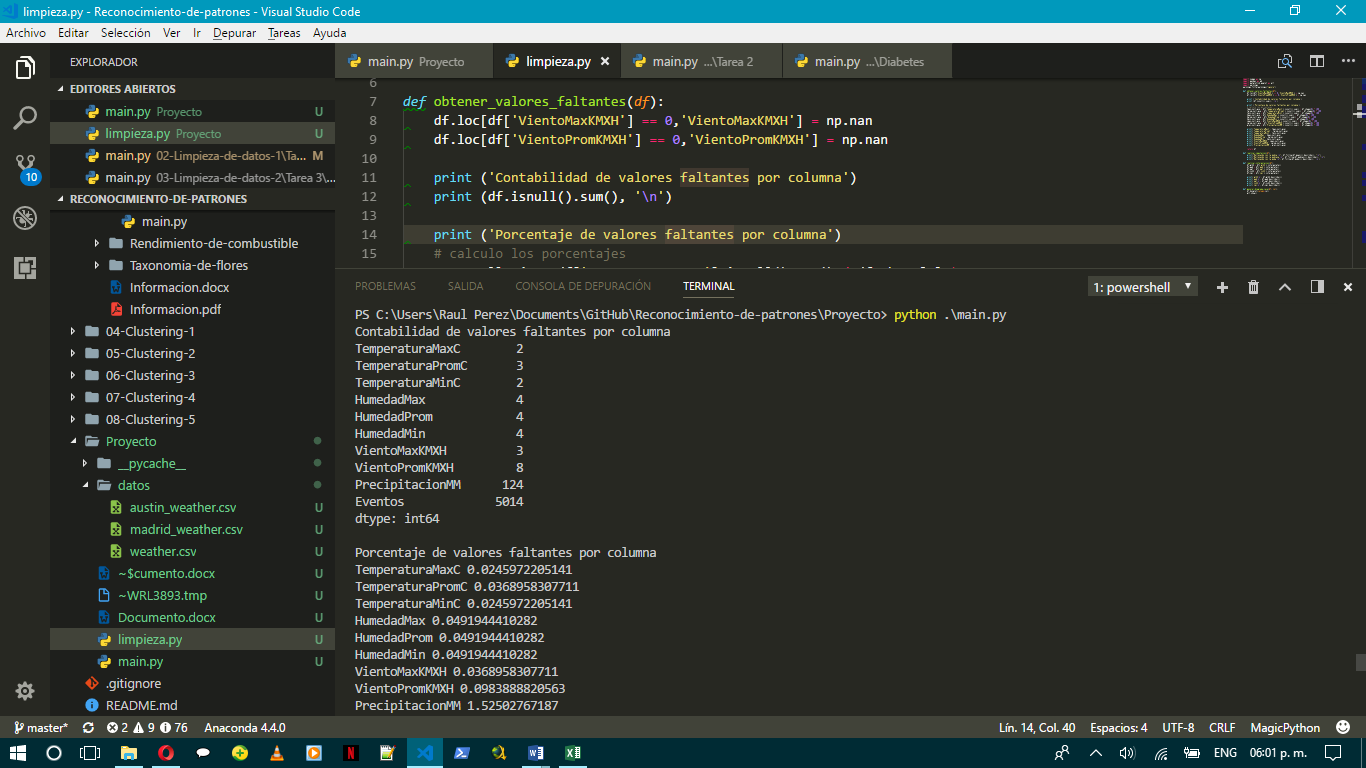
**Eventos:** Describe el evento o eventos sucedidos en el día, se representa con un valor entero. (Nada = 0, Neblina = 1, Nieve = 2, Lluvia = 3, Neblina-Lluvia = 4, Tornado = 5, Lluvia-Nieve = 6, Lluvia-Tormenta = 7, Neblina-Lluvia-Tormenta = 8, Lluvia-Granizo-Tormenta).

**Limpieza de los datos**

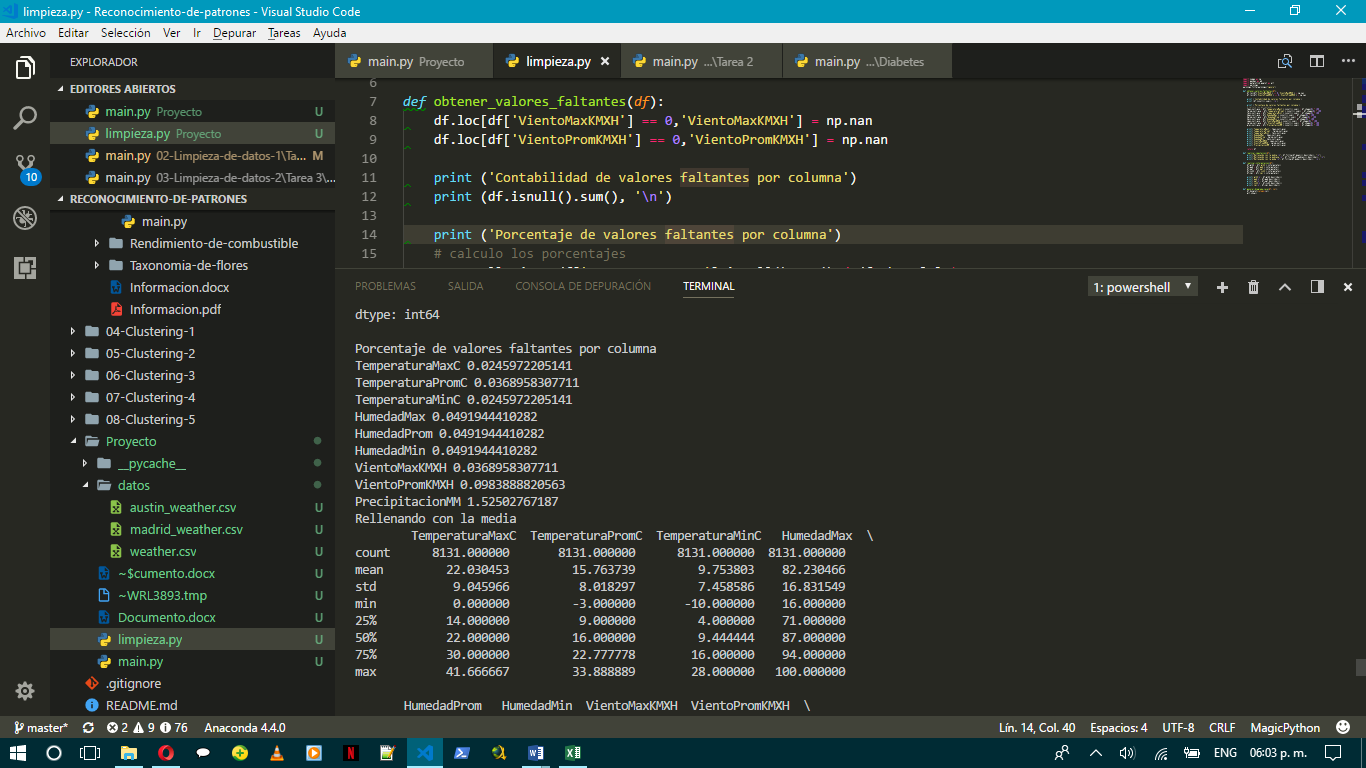
*El código relacionado a esta parte está contenido en el archivo* ***limpieza.py*** *dentro de la carpeta del proyecto.*

**Valores faltantes**

Después de analizar los datos se determinó la cantidad de valores faltantes para cada variable en el conjunto de datos, los cuáles son siguiente:



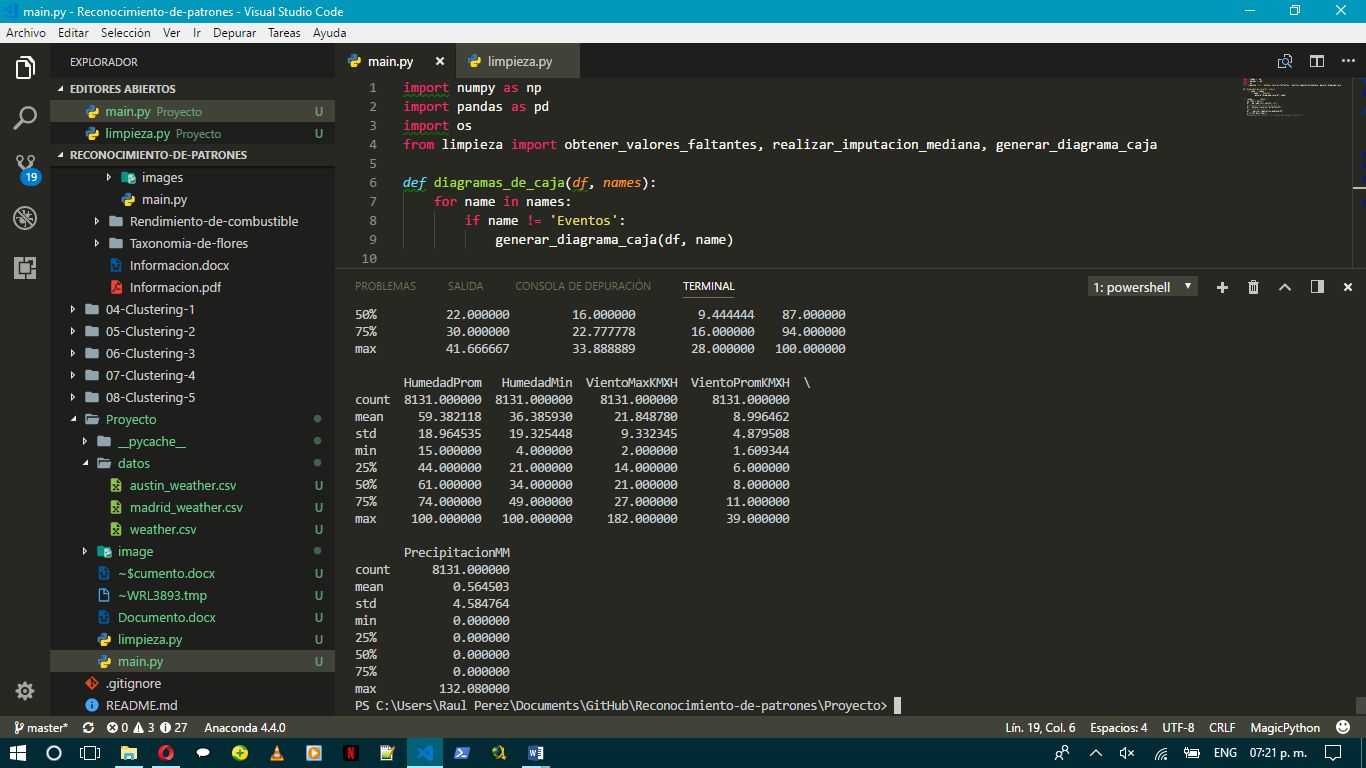
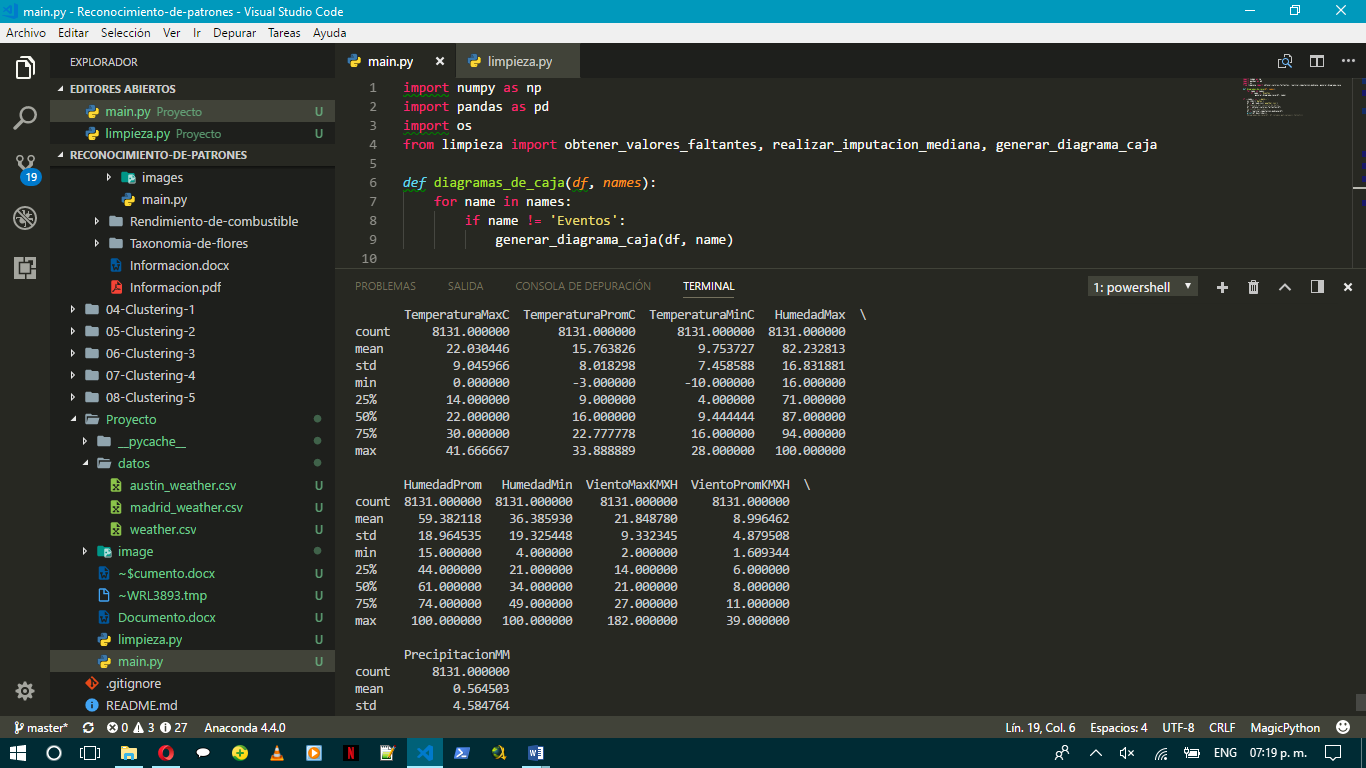
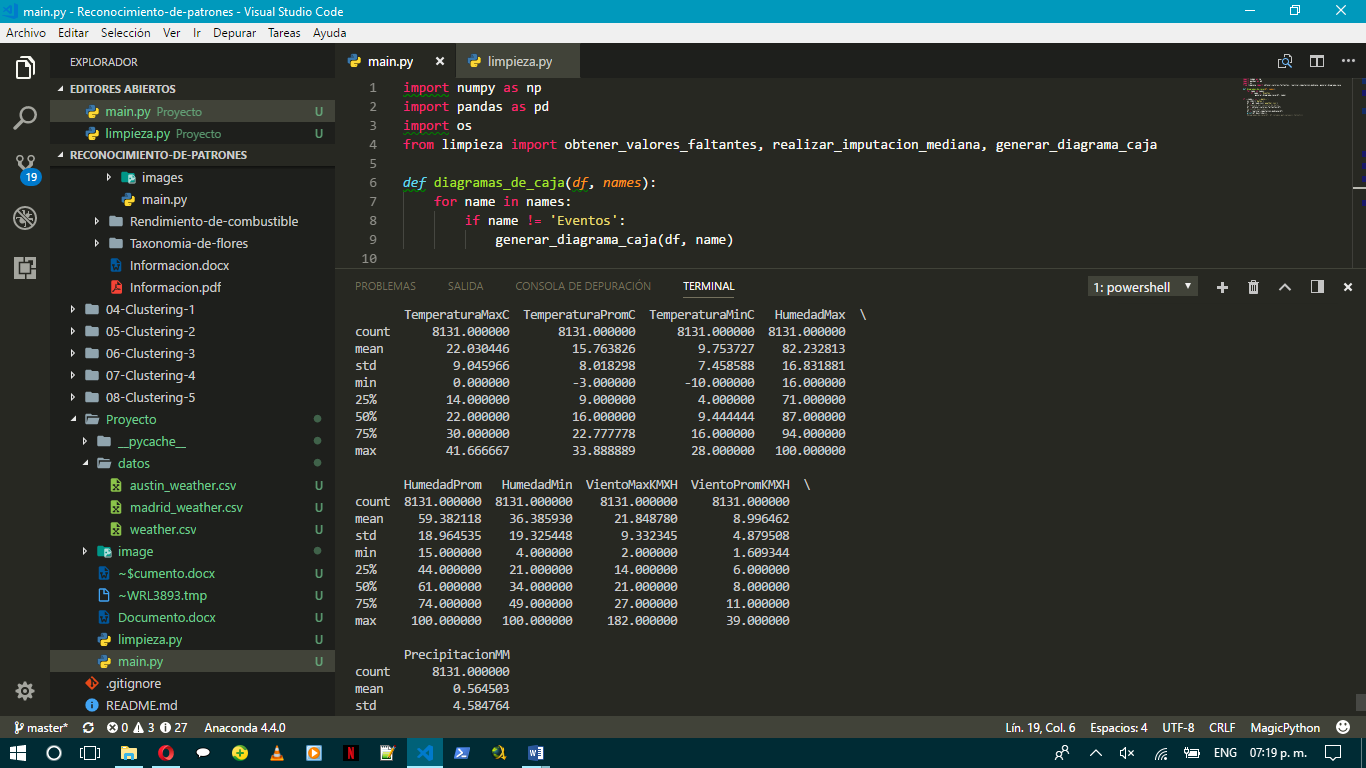
Una vez teniendo el número de valores faltantes se determinó el porcentaje del mismo.



Únicamente la variable de **PrecipitacionMM** tiene un grado de impacto manejable, las otras variables tienen un grado de impacto trivial.

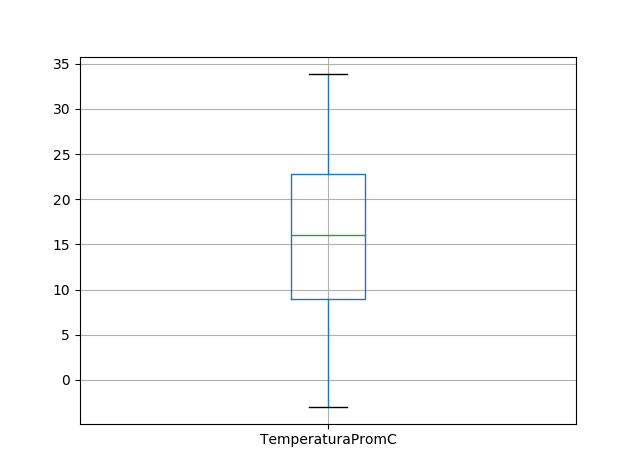
**Imputación**

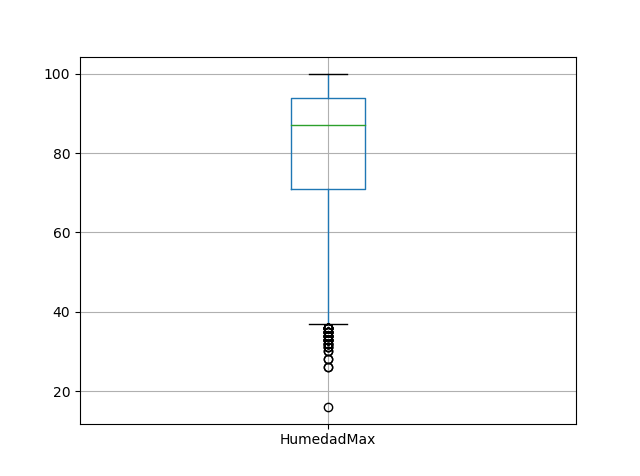
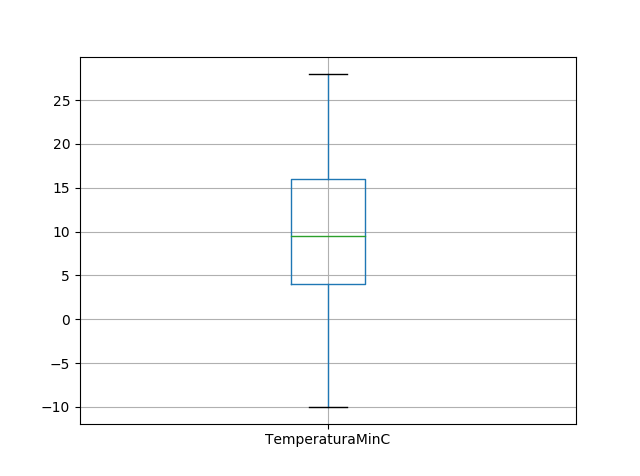
Después se determinó usar imputación con la mediana para el rellenado de los datos faltantes ya que interpolación no tendría sentido ya que los datos ni siquiera tiene una variable de fecha.

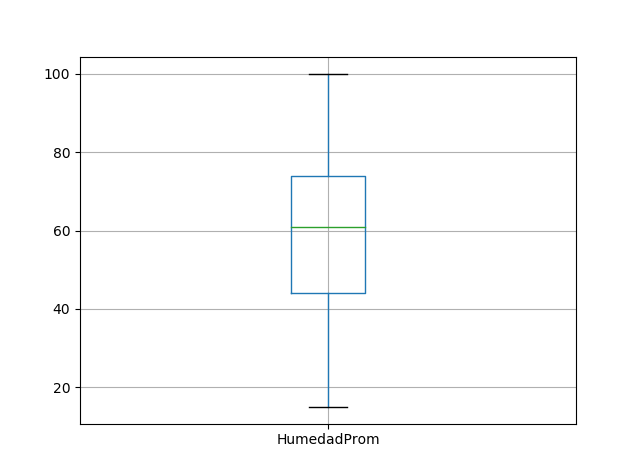
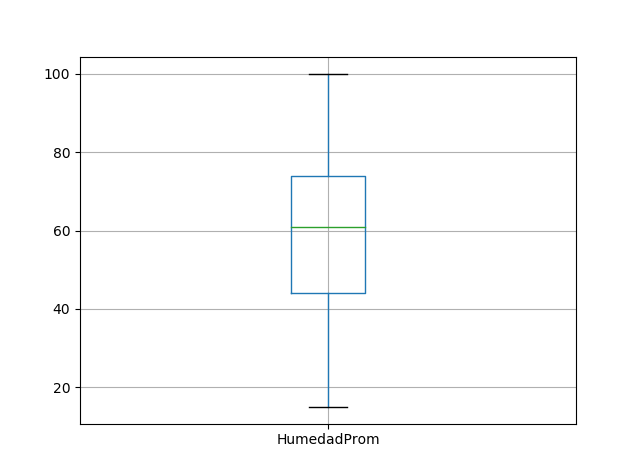


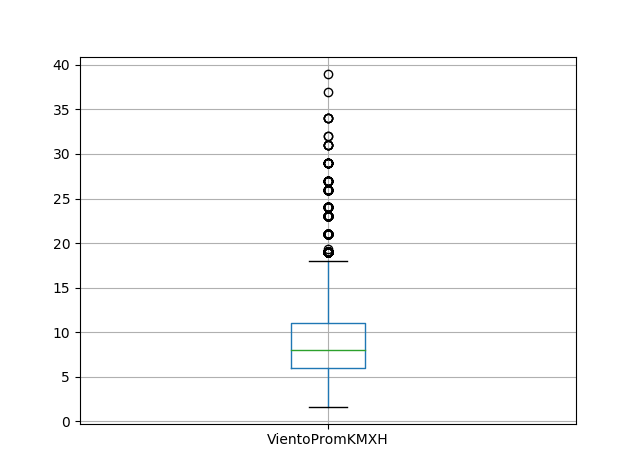
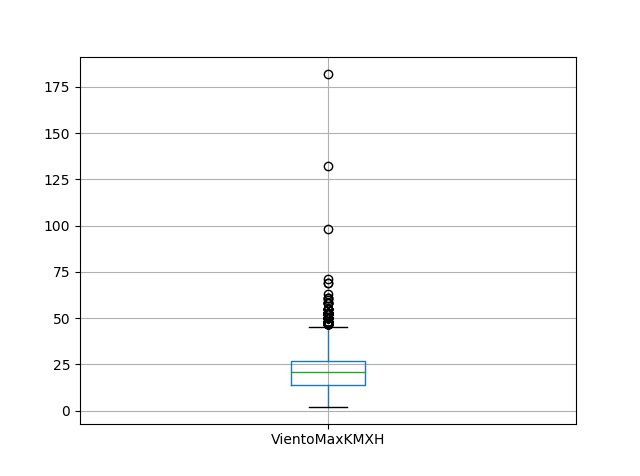
**Valores atípicos**

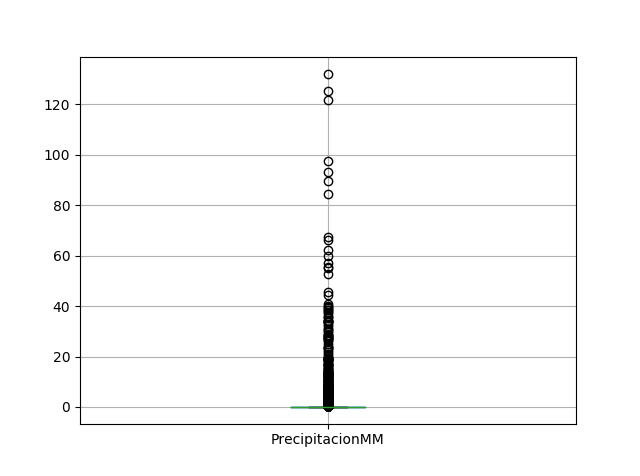
Para el análisis de los valores atípicos se usó diagramas de cajas para cada variable numérica en el conjunto de datos.











Los valores atípicos dados por las variables se consideran normales, ya que el conjunto de datos es la unión de mediciones del clima de dos ciudades diferentes y por lo tanto **HumedadMin** contiene valores atípicos. En el caso de las otras variables las de **VientoMaxKMXH**, **VientoPromKMXH** y **PrecipitacionMM**.También son valores atípicos previstos, ya que son las situaciones de las lluvias posiblemente provocadas por tormentas, huracanes, tornados, etc.

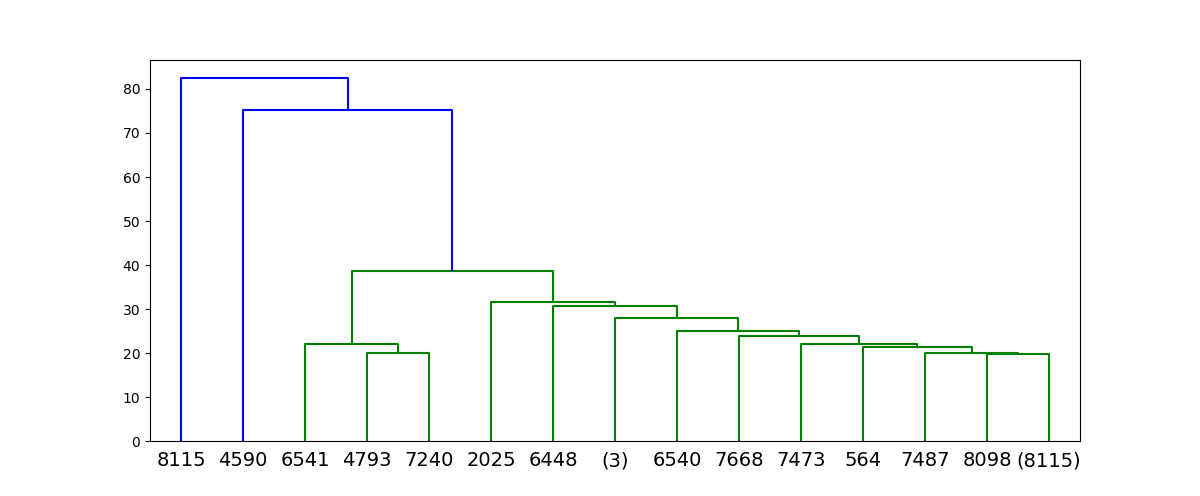
**Clustering**

**Dendrogramas**

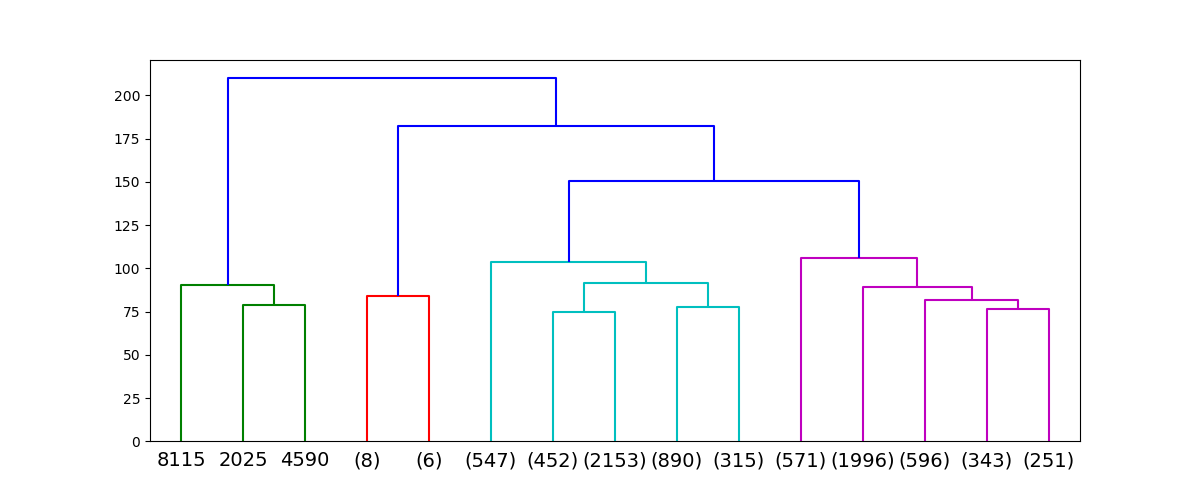
*El código relacionado a esta parte está contenido en el archivo* ***dendrograma.py*** *dentro de la carpeta del proyecto.*

Los dendrogramas son útiles para una primera impresión para saber el número de grupos que puede haber en el conjunto de datos. Se usaran los métodos de singler, complete y centroid para visualizar los últimos 15 clusters formados del conjunto de datos.

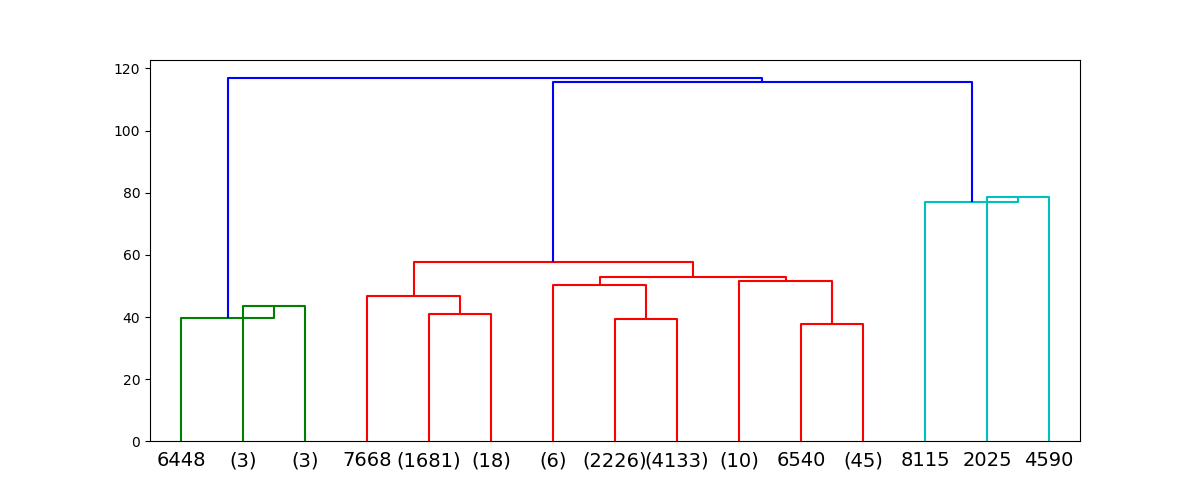
**Dendrograma método singler**



**Dendrograma método complete**



**Dendrograma método centroid**



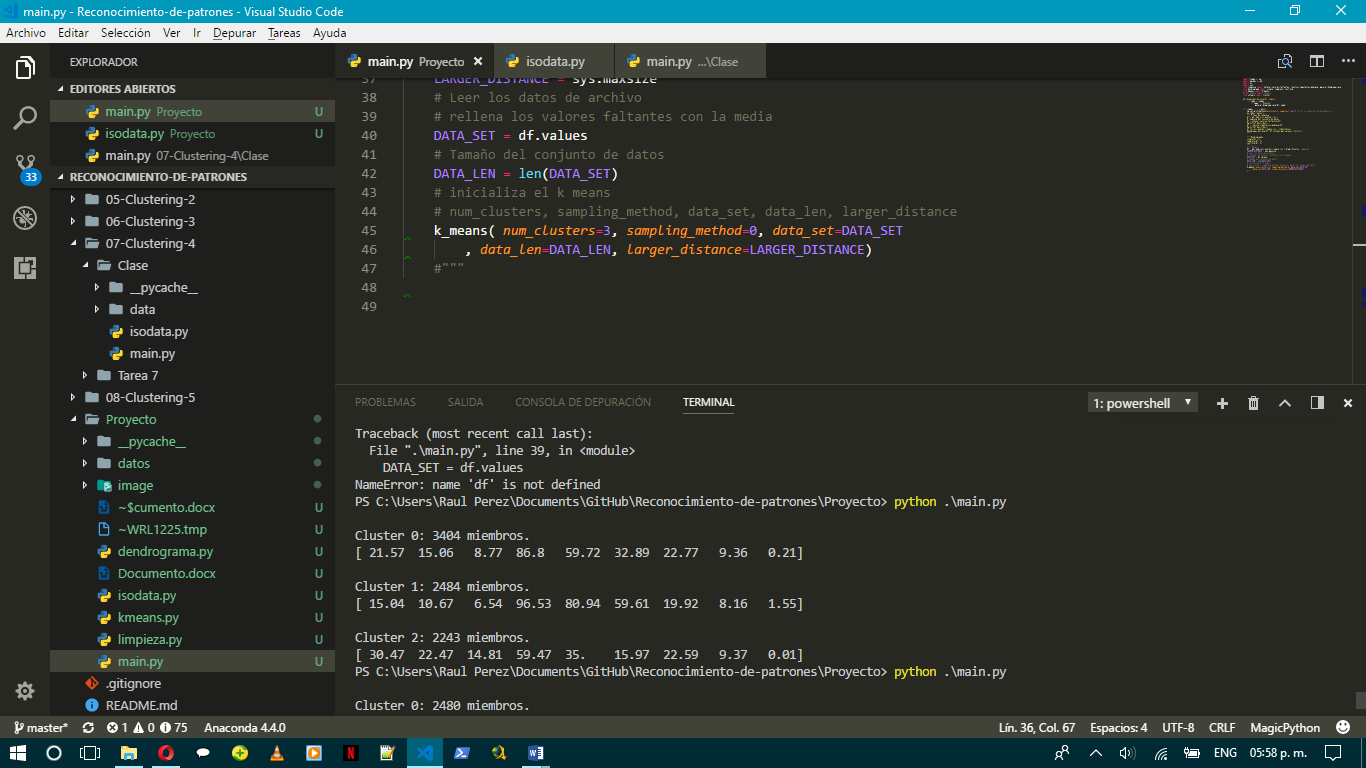
Usando dendrogramas y analizando los resultados con los tres métodos se pueden observar la formación de tres o cuatro grupos en el conjunto de datos. Como los datos son de un conjunto de mediciones del clima, se puede llegar a concluir que existen uno o dos grupos contiene a gran parte de las mediciones que son posiblemente de los tiempos normales del día, es decir, cuando no ocurre un fenómeno natural de gran intensidad ya se por ejemplo un huracán. Los otros grupos que contienen pocas mediciones pueden ser de mediciones de cuando está ocurriendo un fenómeno natural de gran intensidad.

**K-means**

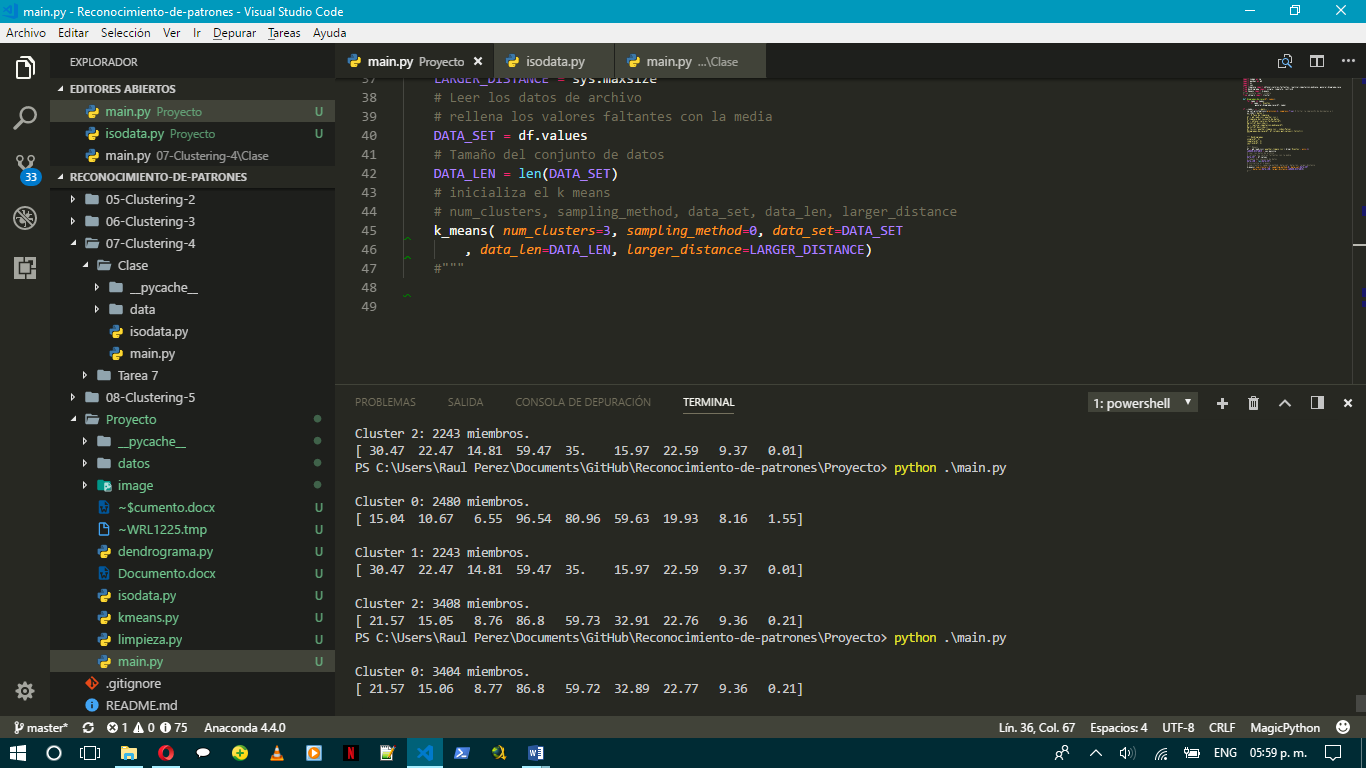
*El código relacionado a esta parte está contenido en el archivo* ***kmeans.py*** *dentro de la carpeta del proyecto.*

K-means es un método de agrupamiento que tiene como objetivo la división de un conjunto de n observaciones en k grupos en el que cada observación pertenece al grupo cuyo valor medio es más cercano. Como utilizando dendrogramas en el conjunto de datos se concluyó que posiblemente existan cuatro diferentes agrupaciones de datos, se usara k-means con k igual a 3 y 4 con una forma de muestreo aleatoria.

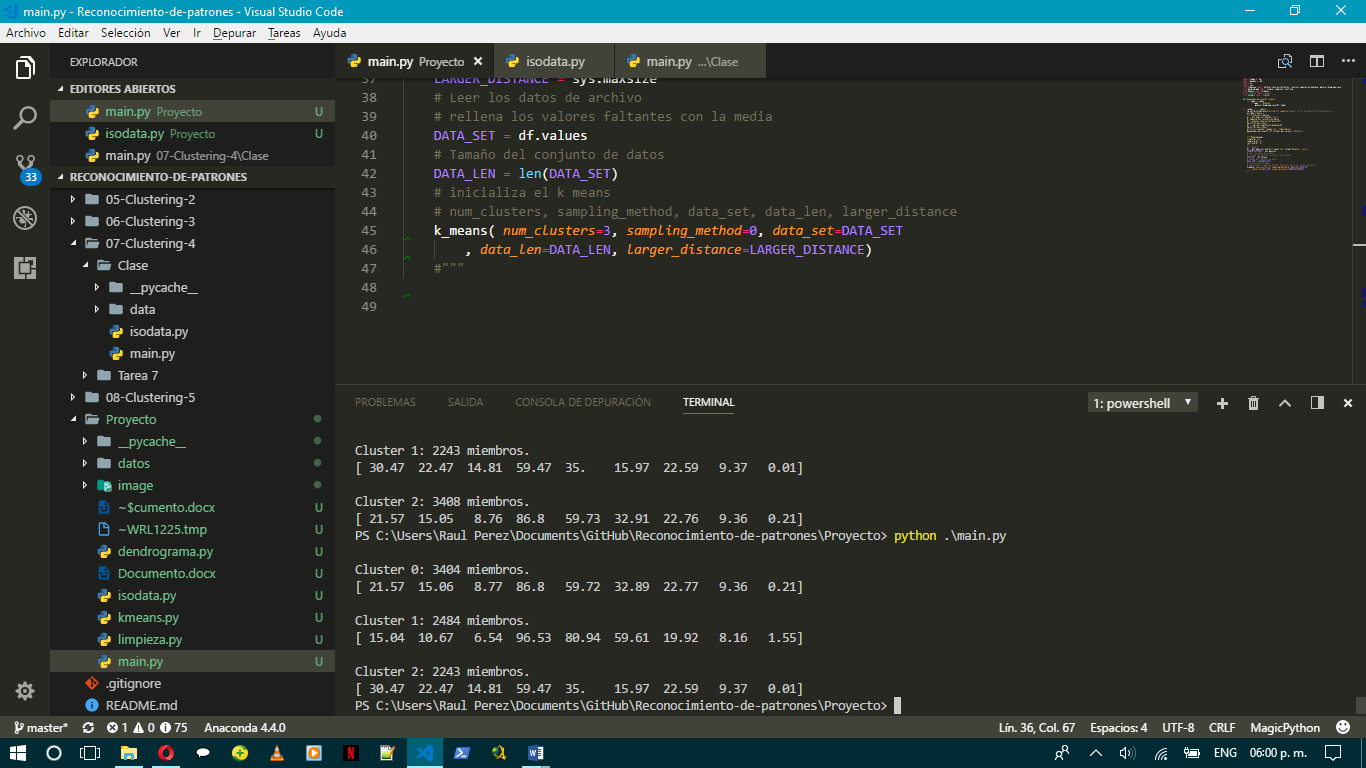
**Primer resultado k = 3**



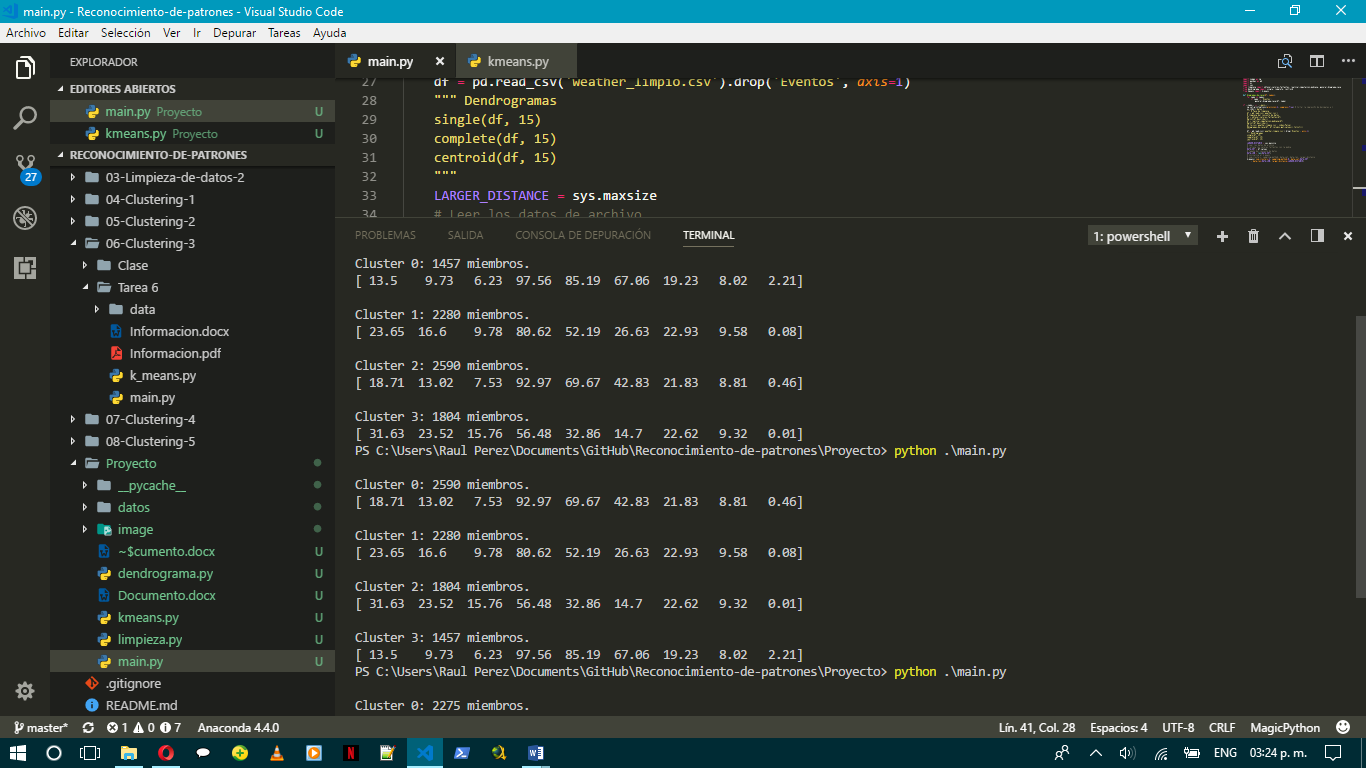
**Segundo resultado k = 3**



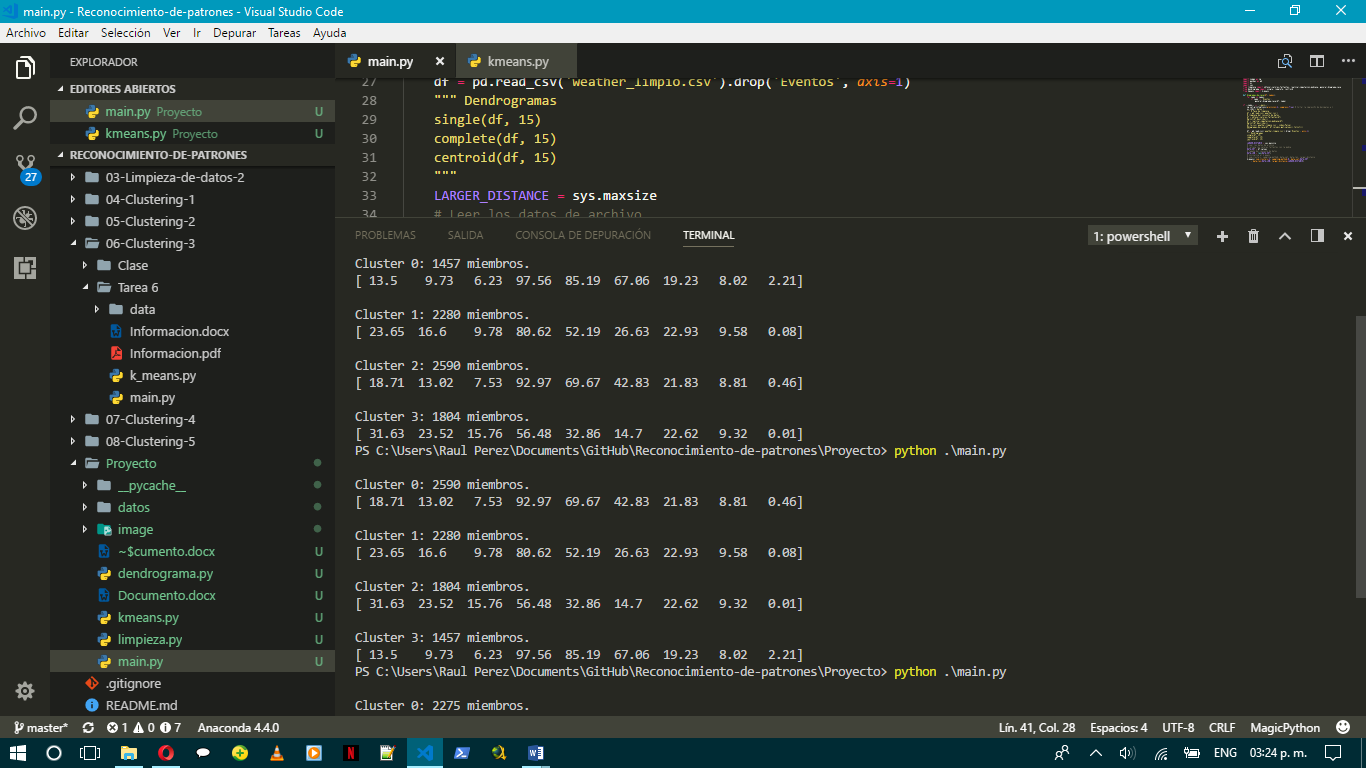
**Tercer resultado k = 3**



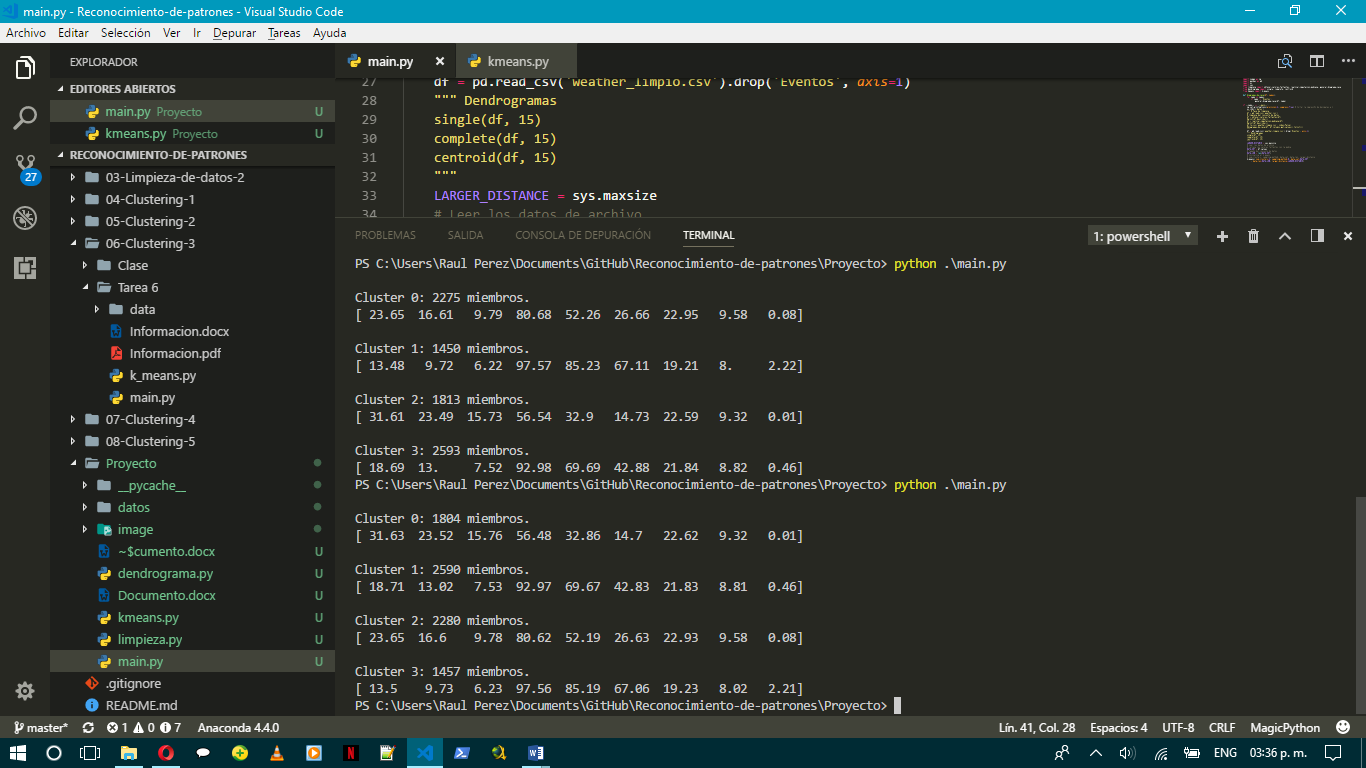
**Primer resultado k = 4**



**Segundo resultado k = 4**



**Tercer resultado k = 4**



Se puede observar tanto con k igual a 3 como k igual a 4, que los grupos que se forman son iguales (k = 4) o casi iguales (k = 3) en los 3 resultados que se hicieron de cada uno. Por lo tanto, se puede pensar que es posible que existan tres o cuatro grupos diferentes como se había concluido en los dendrogramas.

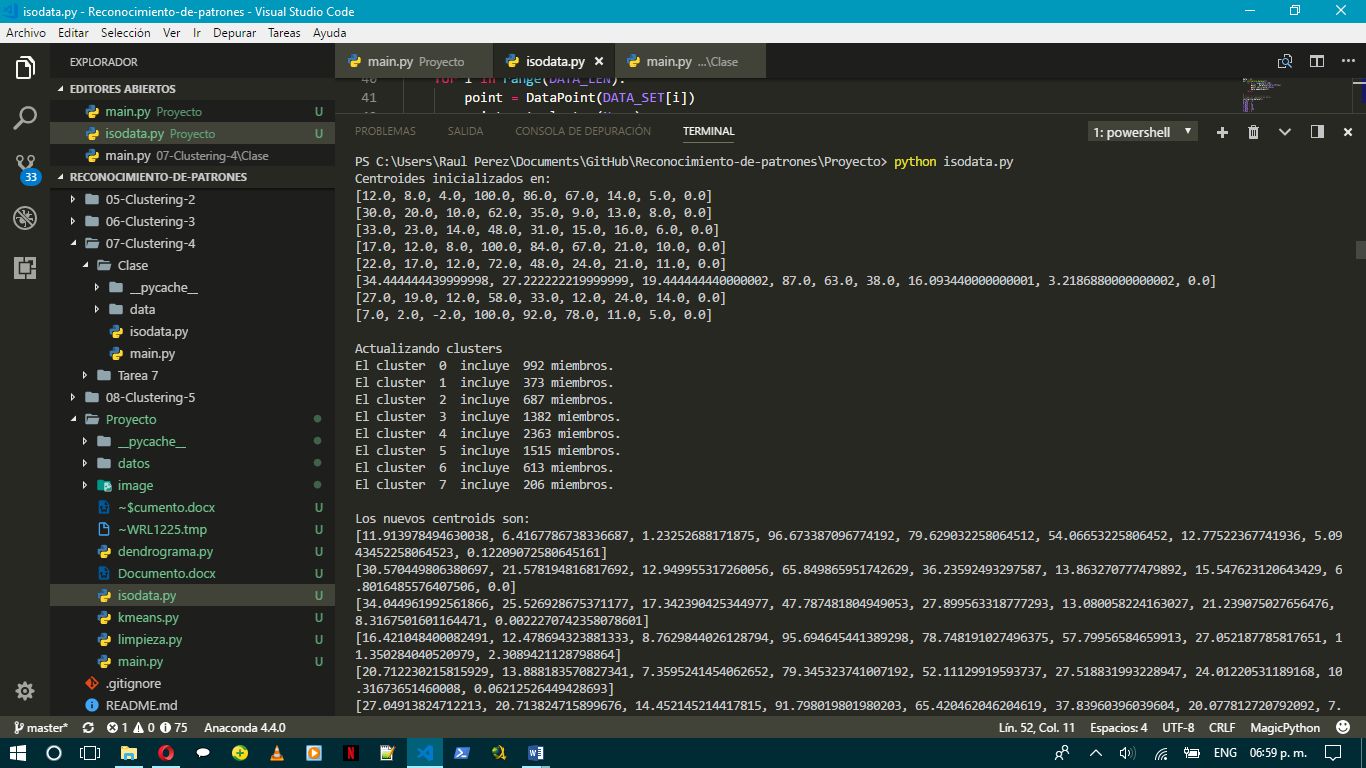
**ISODATA**

*El código relacionado a esta parte está contenido en el archivo* ***isodata.py*** *dentro de la carpeta del proyecto.*

El método ISODATA es un algoritmo similar al k-means. Su objetivo es particionar un conjunto de datos en subconjuntos. Sin embargo, a diferencia de k-means, el método ISODATA  maneja una serie de heurísticas con tres objetivos que le permiten optimizar el número de clusters:

* Eliminar clusters con poco ejemplares.
* Unir clusters muy cercanos.
* Dividir clusters dispersos

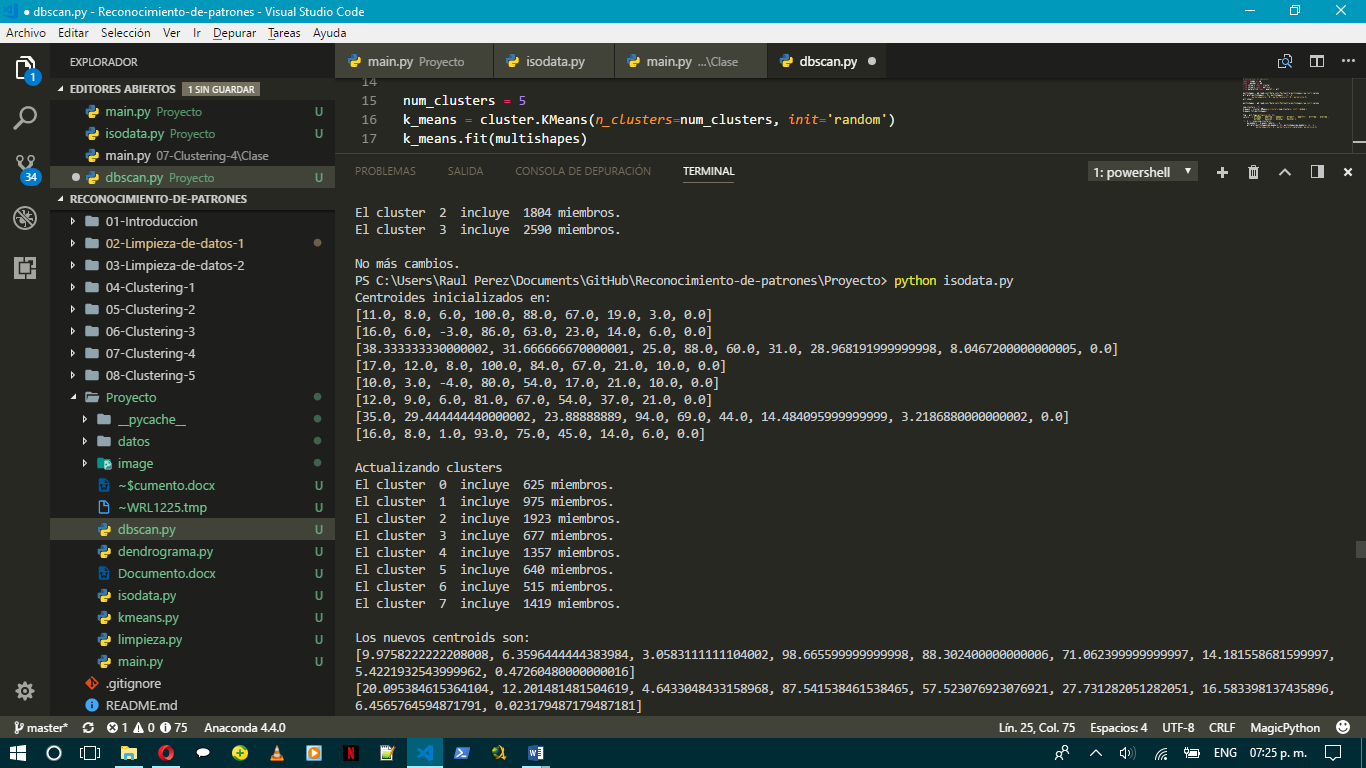
**Centroides iniciales resultado uno**



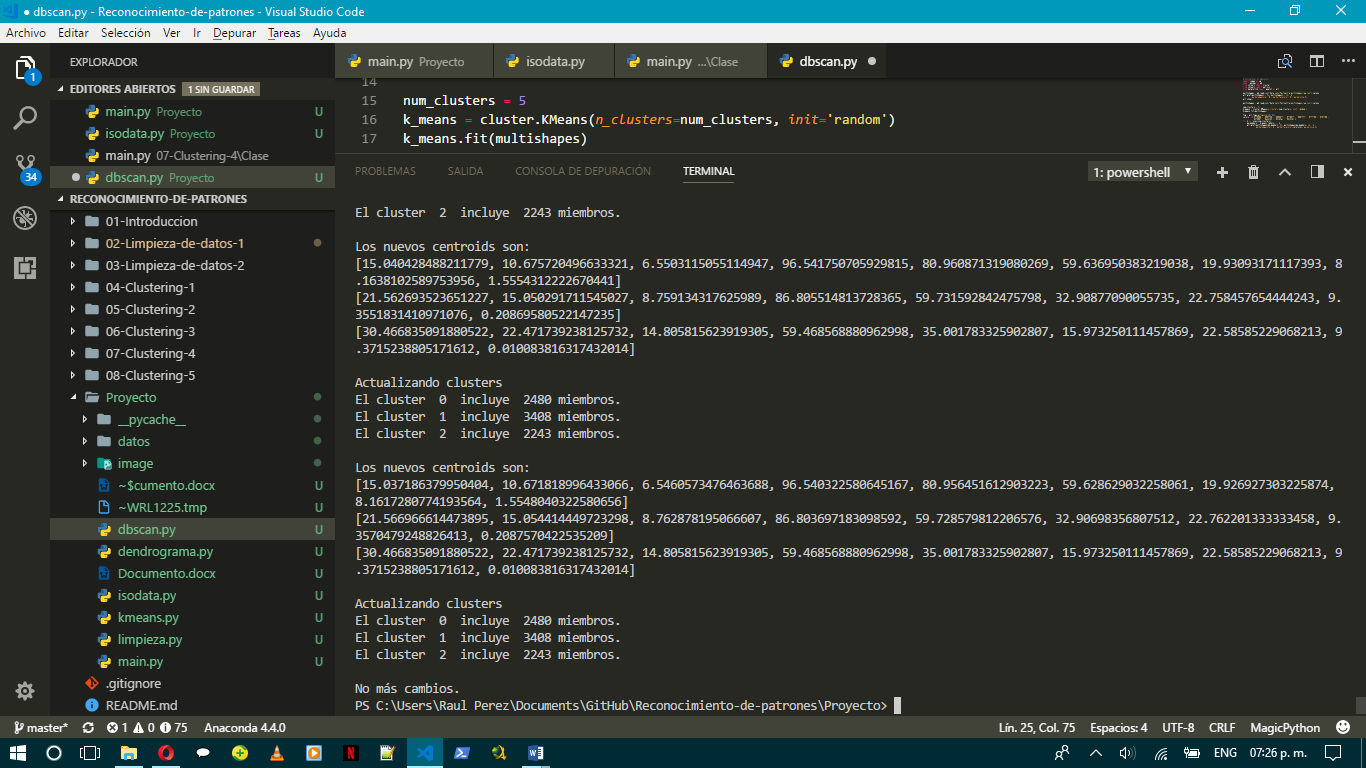
**Centroides finales resultado uno**



**Centroides iniciales resultado dos**



**Centroides finales resultado dos**



Realizando varias ejecuciones del ISODATA con el conjunto de datos, se observó que con una k inicial de 8 y con una forma de muestreo aleatorio, al final de las iteraciones se formaban grupos de 3 y 4 clusters. Mismos que se observaban en los demás técnicas de clustering.

**Clasificación**

**K-Vecinos Próximos**

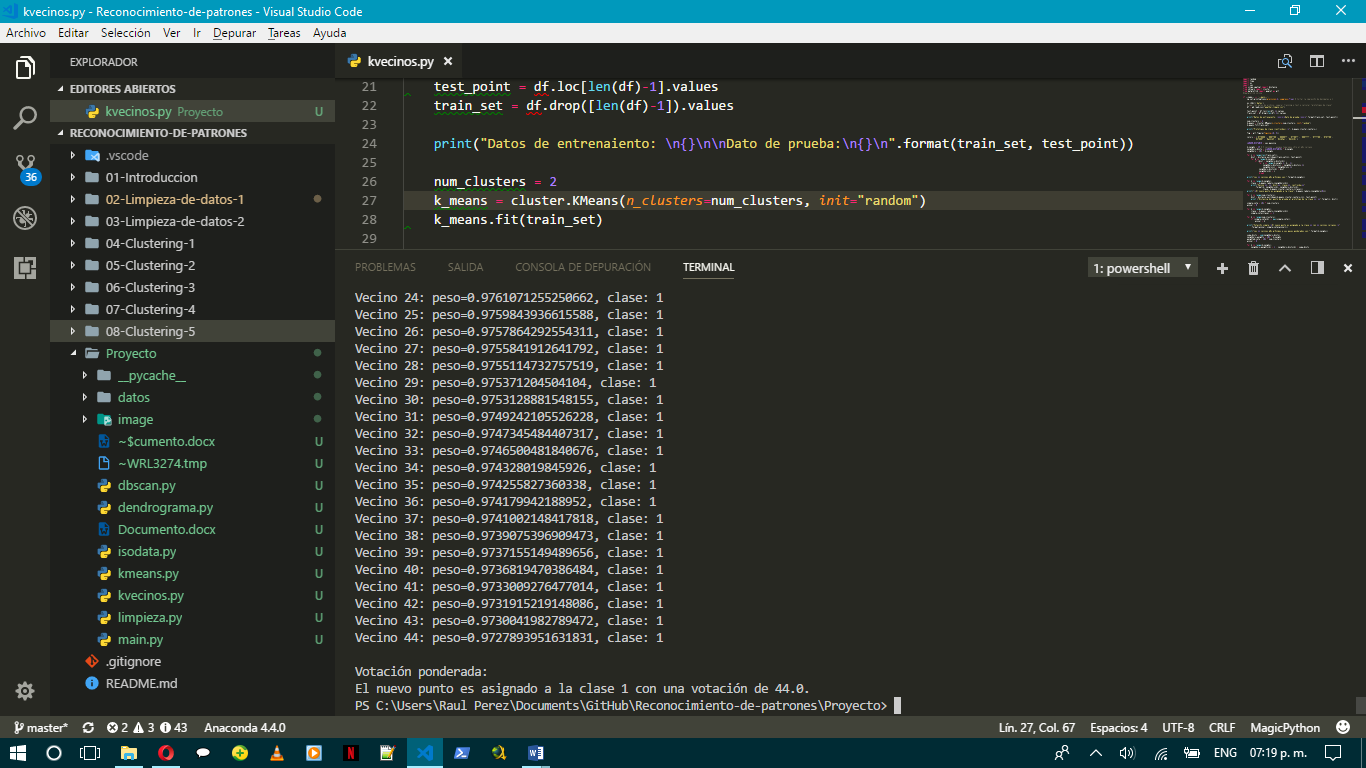
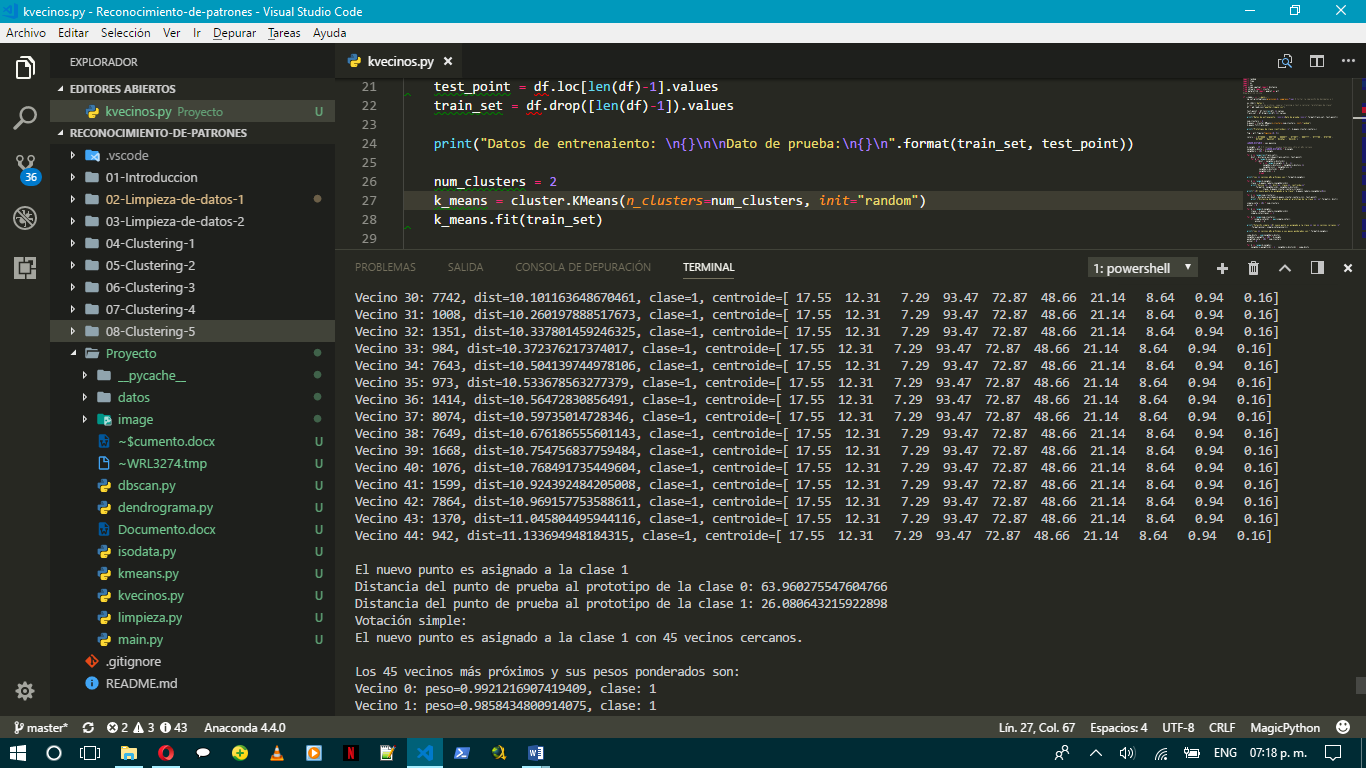
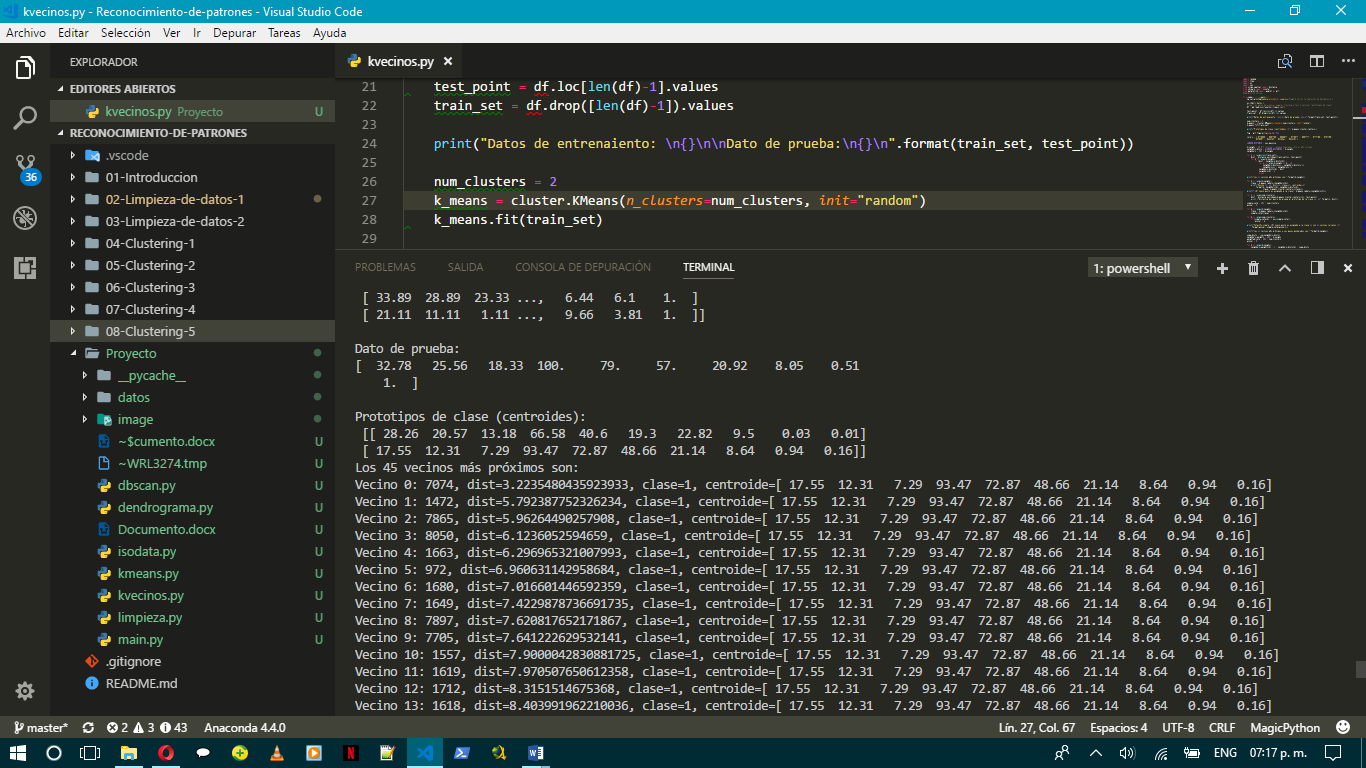
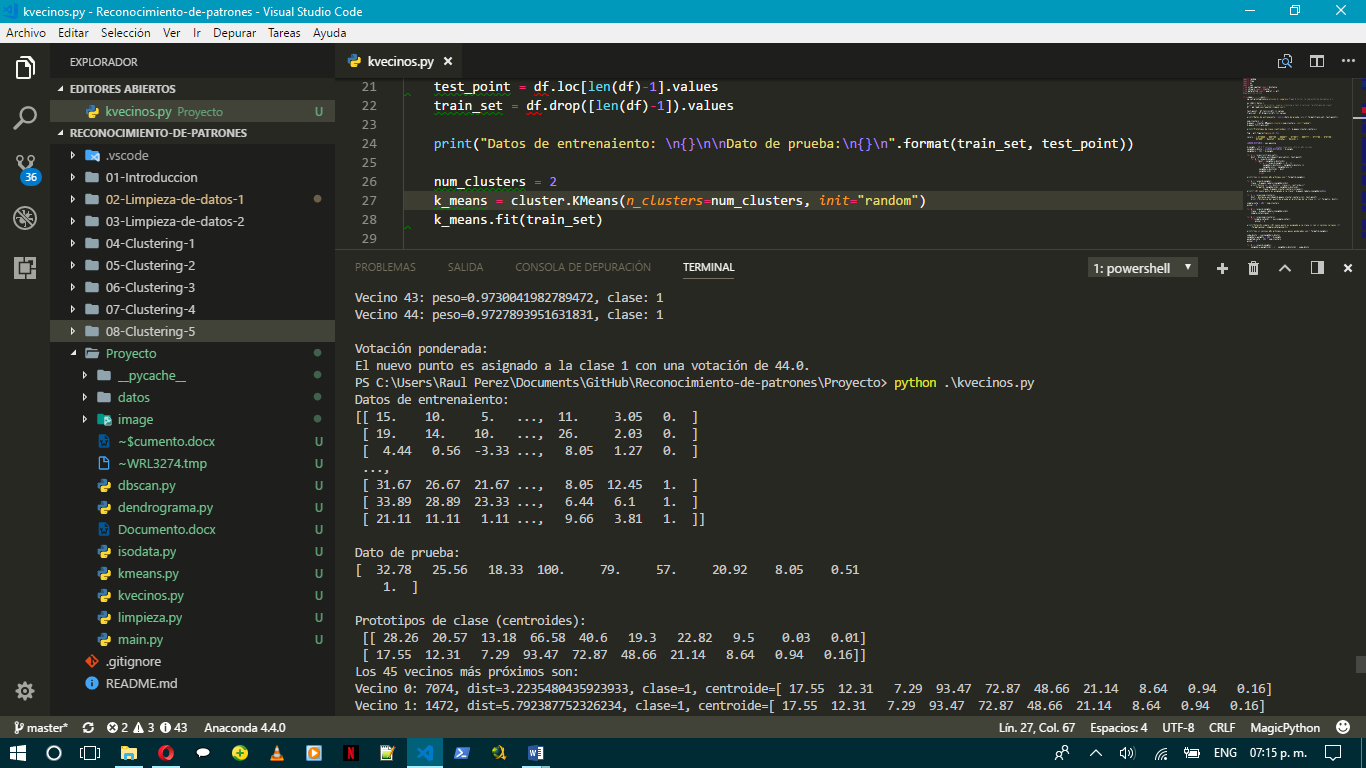
*El código relacionado a esta parte está contenido en el archivo* ***kvecinos.py*** *dentro de la carpeta del proyecto.*

*Debido a que el conjunto de datos contiene varios eventos dependiendo de lo sucedido en el día, se cambiara sus valores a sucedió o no sucedió un evento en el día, para que de esa manera haya dos clases.*

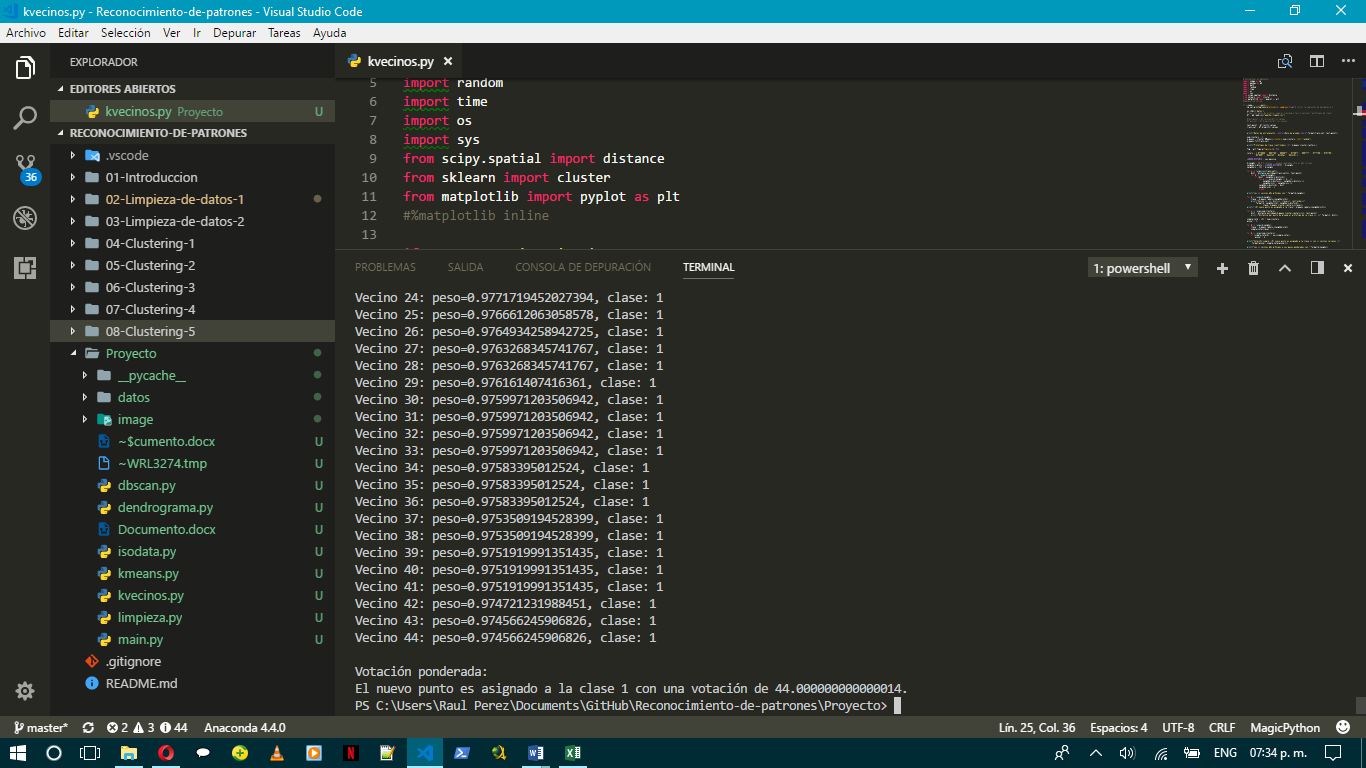
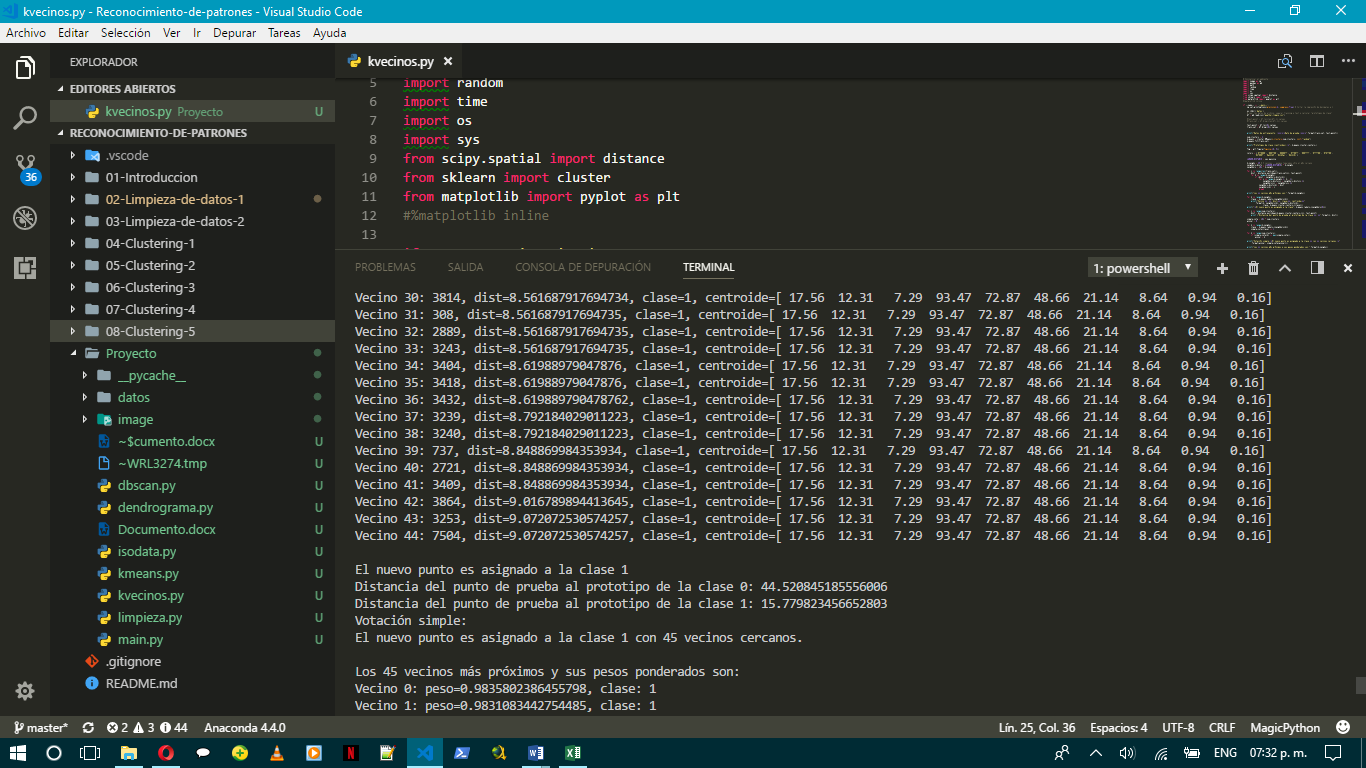
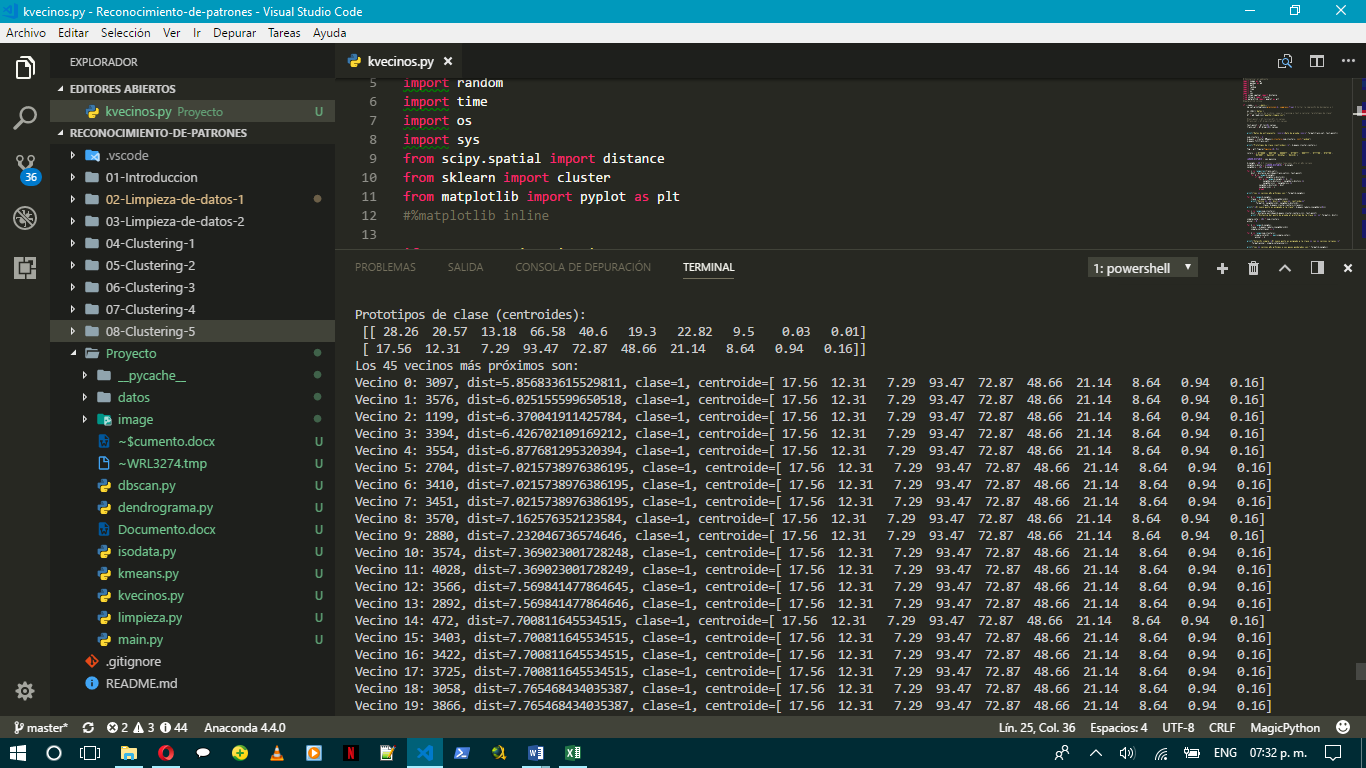
El método de k**-vecinos próximos** es un método de aprendizaje vago (lazy learning). Esto significa que el aprendizaje no conduce a una generalización: no existe una fase de entrenamiento (o es muy breve) y en su lugar el método mantiene todos los datos disponibles y los emplea para realizar la clasificación.

Para la ejecución del método se tomaron 45 debido a que correspondía con la cantidad sugerida: donde es la cantidad de datos. Se tomó la última medición del conjunto de datos, correspondiente a un dato del tipo de evento con valor 1 (sucedió) y la primera medición que corresponde a un evento de no sucedió*.*

**Última medición**



**Primera medición**



Se puede observar que los dos grupos formados por el método, se encuentran mesclados mediciones de eventos de sucedió o no sucedió. Ya que tanto la primera como la segunda medición caen en el mismo grupo.

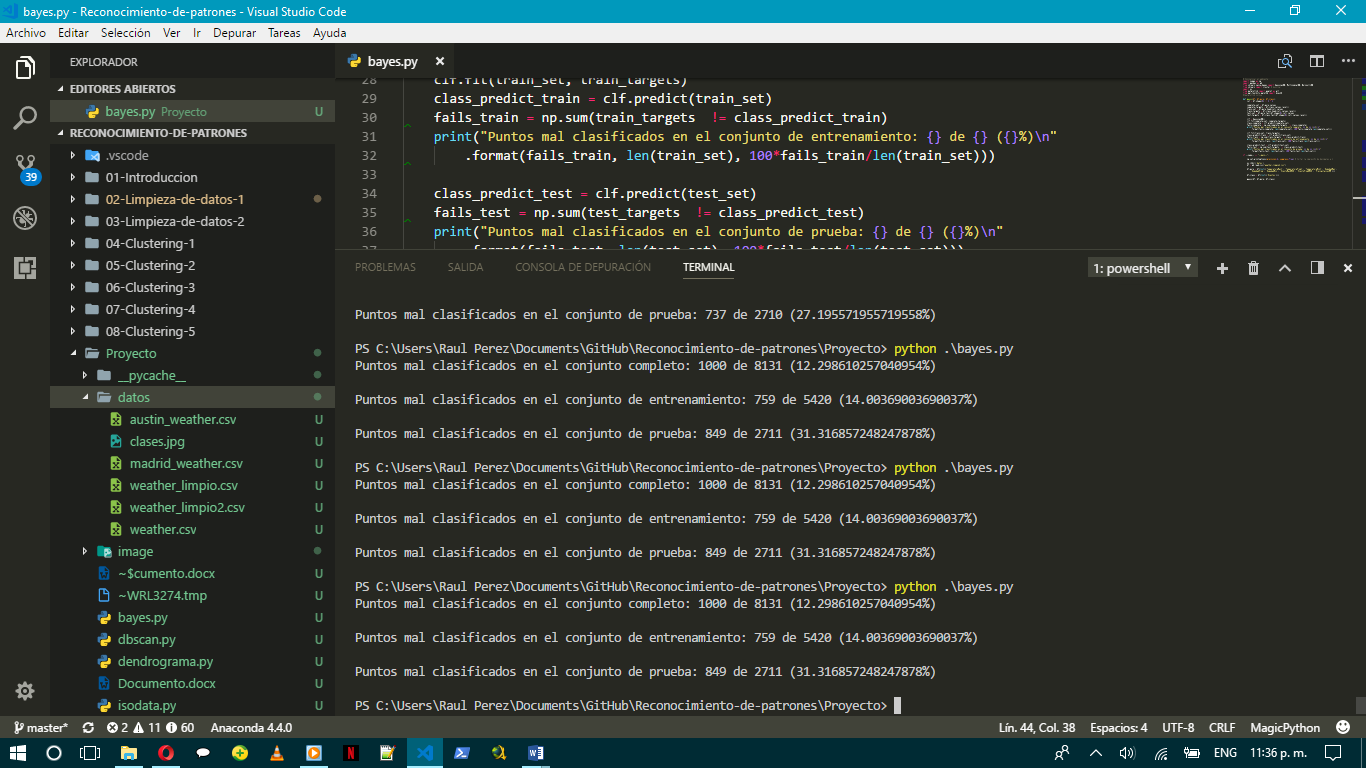
**Teorema de Bayes**

*El código relacionado a esta parte está contenido en el archivo* ***bayes.py*** *dentro de la carpeta del proyecto.*

En teoría de la probabilidad y minería de datos, un clasificador Bayesiano ingenuo es un clasificador probabilístico fundamentado en el teorema de y algunas hipótesis simplificadoras adicionales. Es a causa de estas simplificaciones, que se suelen resumir en la hipótesis de independencia entre las variables predictoras, que recibe el apelativo de ingenuo.

Para el clasificador de Bayes se dividieron los datos en datos de entrenamiento y datos de prueba para poder hacer las pruebas con la distribución Gaussiana.

**Modelo gaussiano**



Con la distribución Gaussiana se obtuvo un error de 12.29% para el conjunto de datos completos, es un error razonable teniendo en cuanta que hay 10 tipos de clases en el conjunto de datos.

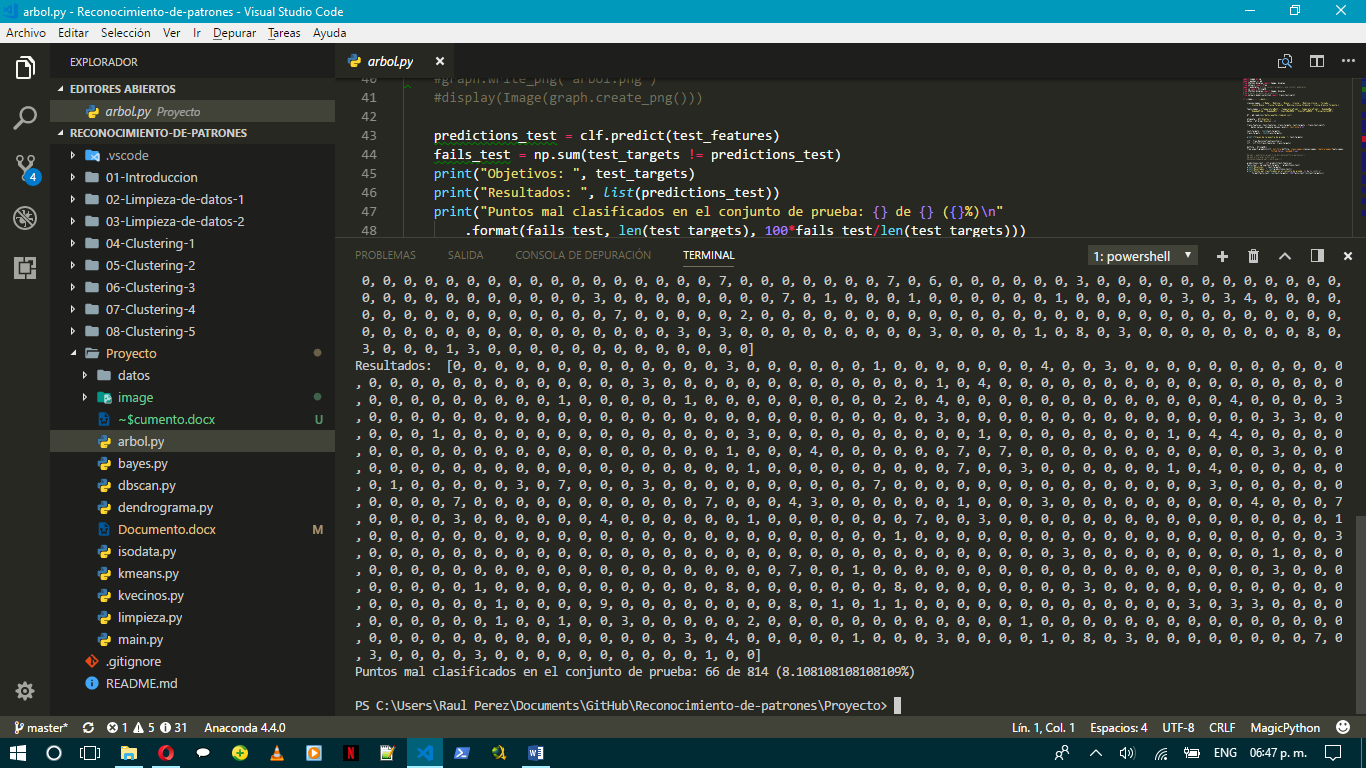
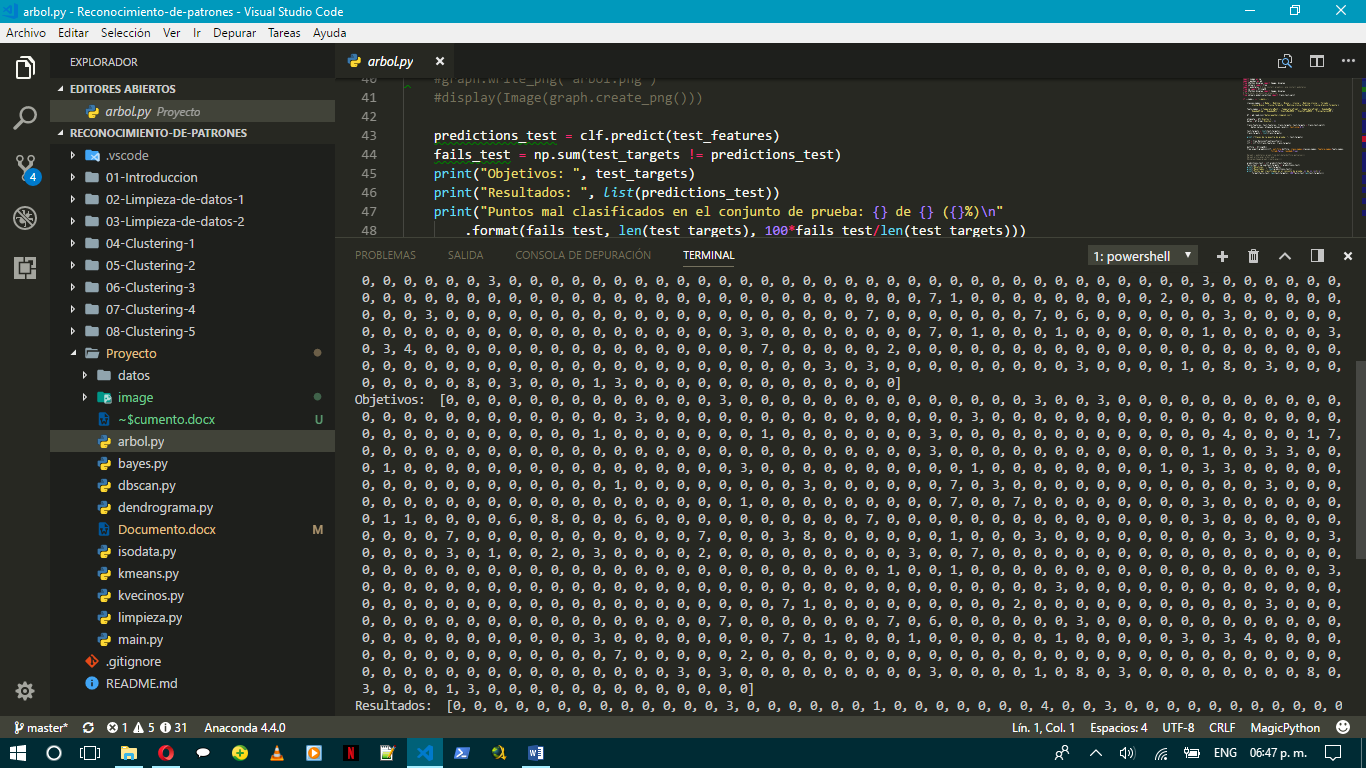
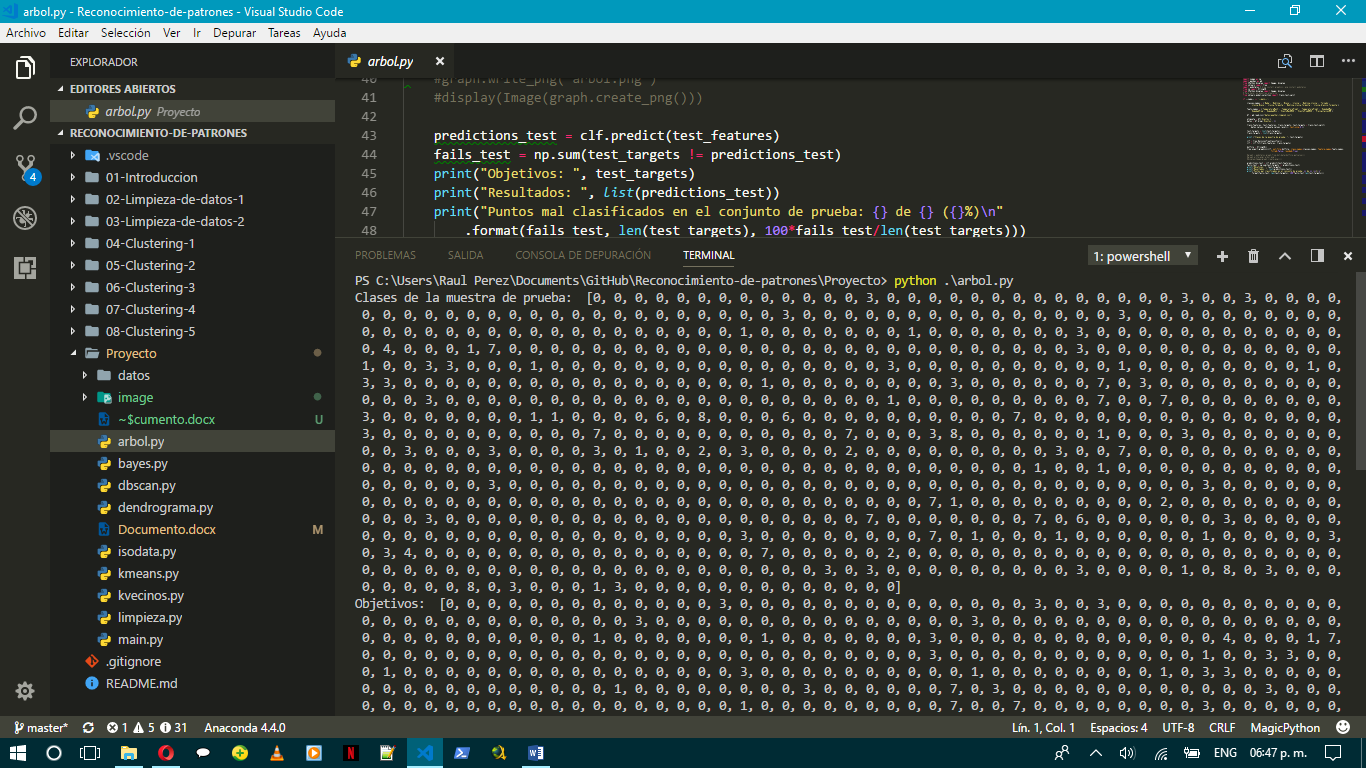
**Árbol de decisión**

*El código relacionado a esta parte está contenido en el archivo* ***arbol.py*** *dentro de la carpeta del proyecto.*

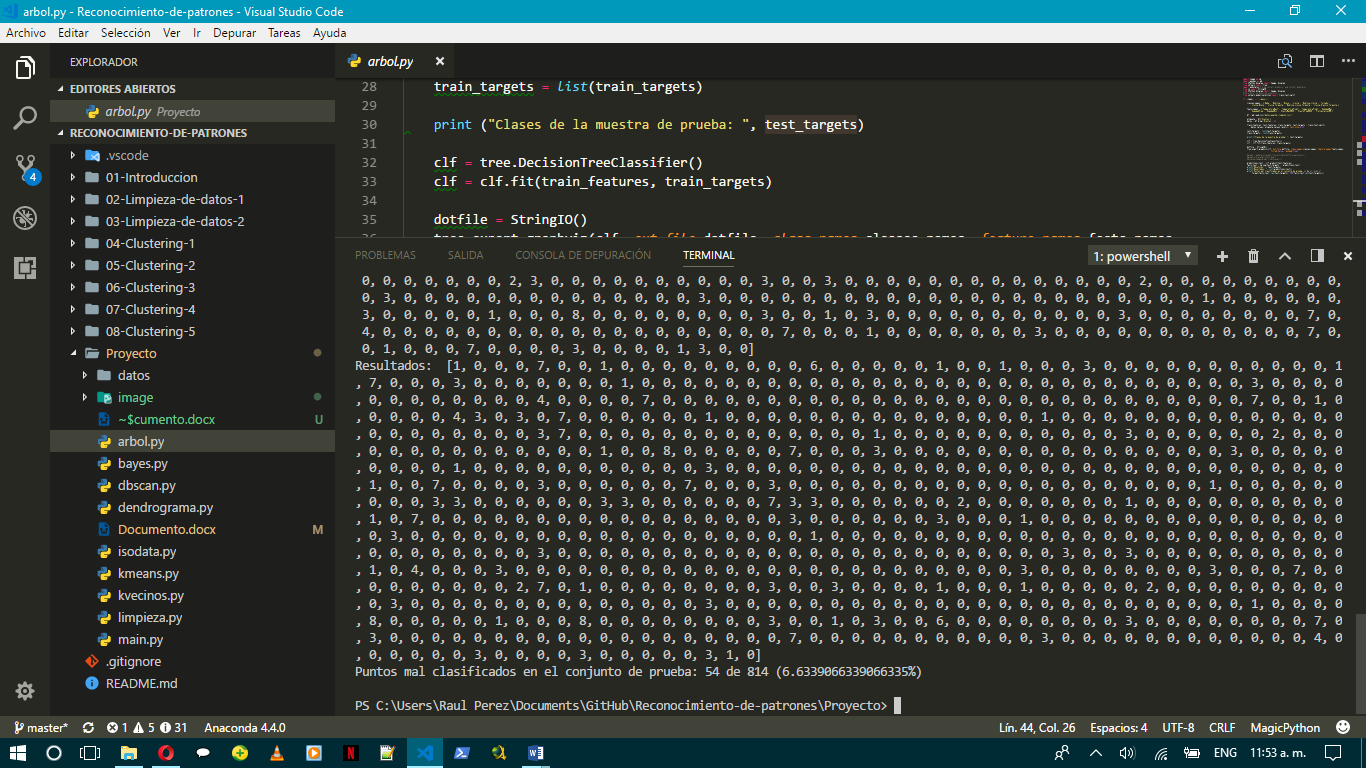
Los [árboles de decisión](https://en.wikipedia.org/wiki/Decision_tree) son uno de los métodos de clasificación más antiguos y robustos y, por lo tanto, de las más utilizados. Los árboles de decisión son un método no paramétrico de aprendizaje supervisado utilizado para clasificación y regresión. El objetivo es generar un conjunto de reglas de decisión simples a partir de las características de los datos de entrenamiento. En un árbol de decisión, los datos de entrenamiento son divididos en cada nodo utilizando reglas de decisión para obtener grupos de elementos similares. Este agrupamiento equivale a una operación de decisión tipo *switch*(caso). Cada nodo interno en el árbol representa una evaluación sobre un atributo, los arcos representan los posibles resultados de la evaluación y las hojas representan una decisión. Las trayectorias, desde el nodo raíz a las hojas son las reglas de decisión.

*El árbol de decisión generado es tan grande que no se vería en el documento, si se desea ver se encuentra en la carpeta de* ***image*** *con el nombre de* ***arbol.png.***

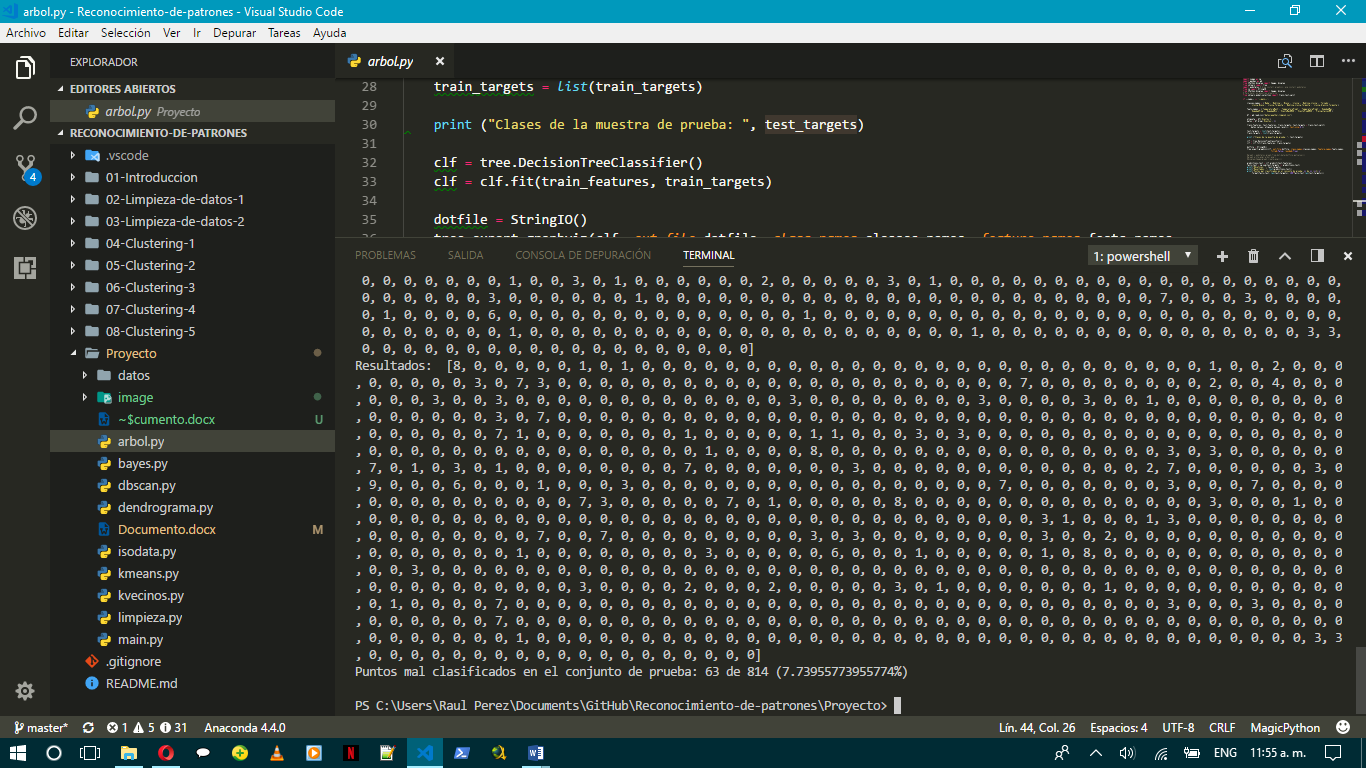
**Primera ejecución**



**Segunda ejecución**



**Tercera ejecución**



Los resultados después de tres ejecuciones del árbol de decisión fueron distintos pero todos dieron mejor resultado que el Bayes que fue de 12.29%. El mejor resultado del árbol de decisión fue de 6.63% mucho mejor que el de Bayes.

**Máquinas de vectores de soporte**

*El código relacionado a esta parte está contenido en el archivo* ***vectores.py*** *dentro de la carpeta del proyecto.*

Las máquinas de vectores de soporte (SVM) se basan en la idea de definir planos de decisión que separen a los objetos pertenecientes a diferentes clases. El objetivo de clasificación es encontrar el hiperplano que separe mejor las regiones en el espacio de características ocupadas por cada una de las clases.

**SVM lineal**

A continuación, se presentan los resultados de la aplicación de SVM lineal al conjunto de datos utilizando diferentes valores de penalización.

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.1 | 7.57% | 6.51% |
| 2 | 0.1 | 10.90% | 12.77% |
| 3 | 0.1 | 9.81% | 10.68% |
| 4 | 0.1 | 8.41% | 7.98% |
| 5 | 0.1 | 10.08% | 8.47% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.5 | 8.40% | 7.86% |
| 2 | 0.5 | 10.91% | 12.89% |
| 3 | 0.5 | 8.10% | 8.10% |
| 4 | 0.5 | 8.20% | 8.10% |
| 5 | 0.5 | 9.85% | 9.70% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1 | 11.35% | 10.07% |
| 2 | 1 | 6.98% | 8.35% |
| 3 | 1 | 7.47% | 7.73% |
| 4 | 1 | 7.83% | 8.23% |
| 5 | 1 | 9.15% | 9.09% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 20 | 18.62% | 18.79% |
| 2 | 20 | 7.99% | 9.45% |
| 3 | 20 | 7.73% | 8.10% |
| 4 | 20 | 7.38% | 7.24% |
| 5 | 20 | 7.58% | 7.61% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 100 | 7.92% | 7.49% |
| 2 | 100 | 8.59% | 9.95% |
| 3 | 100 | 7.61% | 7.61% |
| 4 | 100 | 7.83% | 7.86% |
| 5 | 100 | 8.86% | 8.59% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 500 | 8.41% | 7.49% |
| 2 | 500 | 14.78% | 16.95% |
| 3 | 500 | 9.17% | 10.44% |
| 4 | 500 | 8.52% | 8.10% |
| 5 | 500 | 8.26% | 7.61% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1000 | 7.50% | 7.12% |
| 2 | 1000 | 6.97% | 8.47% |
| 3 | 1000 | 9.74% | 9.70% |
| 4 | 1000 | 13.20% | 11.17% |
| 5 | 1000 | 7.50% | 7.24% |

A partir de estos resultados podemos observar que, en este caso, no hay un valor de C claramente superior, el más constante fue con C=100, pero este no obtuvo el mejor porcentaje. El mejor porcentaje obtenido para el conjunto de prueba fue de un 6.51%, con un valor de C=0.1, el peor porcentaje obtenido fue un 18.79% obtenido con un valor de C=20.

**SVM kernel RBF**

A continuación, mostramos los resultados de varias corridas utilizando una máquina de vectores de soporte con función kernel RBF (función de base radial)

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.1 | 0.01% | 10.93% |
| 2 | 0.1 | 0.01% | 9.95% |
| 3 | 0.1 | 0.01% | 11.54% |
| 4 | 0.1 | 0.01% | 12.77% |
| 5 | 0.1 | 0.01% | 10.81% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.5 | 0.01% | 10.93% |
| 2 | 0.5 | 0.01% | 9.95% |
| 3 | 0.5 | 0.01% | 11.54% |
| 4 | 0.5 | 0.01% | 12.77% |
| 5 | 0.5 | 0.01% | 10.81% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1 | 0.01% | 10.68% |
| 2 | 1 | 0.01% | 9.95% |
| 3 | 1 | 0.01% | 11.42% |
| 4 | 1 | 0.01% | 12.65% |
| 5 | 1 | 0.01% | 10.68% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 20 | 0.01% | 10.93% |
| 2 | 20 | 0.01% | 10.19% |
| 3 | 20 | 0.01% | 12.03% |
| 4 | 20 | 0.01% | 12.40% |
| 5 | 20 | 0.01% | 10.81% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 100 | 0.01% | 10.93% |
| 2 | 100 | 0.01% | 10.19% |
| 3 | 100 | 0.01% | 11.91% |
| 4 | 100 | 0.01% | 12.40% |
| 5 | 100 | 0.01% | 10.81% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 500 | 0.01% | 10.93% |
| 2 | 500 | 0.01% | 10.19% |
| 3 | 500 | 0.01% | 11.91% |
| 4 | 500 | 0.01% | 12.40% |
| 5 | 500 | 0.01% | 10.81% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1000 | 0.01% | 10.93% |
| 2 | 1000 | 0.01% | 10.19% |
| 3 | 1000 | 0.01% | 11.91% |
| 4 | 1000 | 0.01% | 12.40% |
| 5 | 1000 | 0.01% | 10.81% |

A partir de estos resultados podemos observar que en cuanto a los datos de entrenamiento todos tuvieron el mismo resultado, pero con los de prueba el valor de C con mejores resultados fue C=1, pero cabe recalcar que independientemente del valor de C los porcentajes son muy similares. El mejor resultado obtenido fue 9.95% con valor de C= 0.1, 0.5, 1.

**SVM kernel RBF (Gamma)**

El otro parámetro de la función kernel RBF es . A continuación, mostramos variaciones del valor de   en ejercicios de clasificación sobre los datos, utilizando C=1.

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.1 | 2.30% | 9.33% |
| 2 | 0.1 | 2.14% | 11.42% |
| 3 | 0.1 | 2.26% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 0.5 | 0.17% | 9.58% |
| 2 | 0.5 | 0.17% | 11.42% |
| 3 | 0.5 | 0.20% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1 | 0.04% | 9.58% |
| 2 | 1 | 0.04% | 11.42% |
| 3 | 1 | 0.05% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 20 | 0.01% | 9.58% |
| 2 | 20 | 0.01% | 11.42% |
| 3 | 20 | 0.01% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 100 | 0.00% | 9.58% |
| 2 | 100 | 0.00% | 11.42% |
| 3 | 100 | 0.00% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 500 | 0.00% | 9.58% |
| 2 | 500 | 0.00% | 11.42% |
| 3 | 500 | 0.00% | 9.45% |

|  |  |  |  |
| --- | --- | --- | --- |
| **Corrida** | **Valor de C** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 1000 | 0.00% | 9.58% |
| 2 | 1000 | 0.00% | 11.42% |
| 3 | 1000 | 0.00% | 9.45% |

A partir de estos resultados podemos observar que en cuanto a los datos de entrenamiento con una C mayor a 100 no se obtiene un porcentaje de error, en cuanto a los de prueba los resultados fueron iguales o casi iguales independientemente del valor de C y del porcentaje de error en el conjunto de entrenamiento. El mejor resultado fue obtenido con un valor de C=0.1, con un porcentaje de 9.33%.

**SVM Sigmoide con C=1**

|  |  |  |
| --- | --- | --- |
| **Corrida** | **Puntos de entrenamiento mal clasificados** | **Puntos de prueba mal clasificados** |
| 1 | 10.55% | 11.79% |
| 2 | 10.82% | 9.33% |
| 3 | 10.56% | 11.67% |

**Conclusión**

Con el SVM Lineal se obtuvieron los mejores resultados y el mejor porcentaje con un 6.51% con los datos de prueba. En cuanto al conjunto de entrenamiento con un SVM Kernel, se obtuvieron porcentajes de error de 0.00%, pero no obtuvieron el mejor porcentaje en el conjunto de prueba.

**Redes Neuronales**

*El código relacionado a esta parte está contenido en el archivo* ***redes.py*** *dentro de la carpeta del proyecto.*

**Perceptrón**

El [perceptrón](https://en.wikipedia.org/wiki/Perceptron) es un modelo de neurona obtenido al combinar el modelo de la neurona de McCulloch-Pitts con una extensión de la regla de Hebb. La regla de entrenamiento consta de los siguientes pasos:

1. Inicializar los pesos con valores aleatorios pequeños.
2. Para cada vector de entrenamiento Xj con valor de clase Yj (el valor de salida esperado):
   1. Calcular el valor de salida Y\*j de la función de activación
   2. Actualizar los pesos de acuerdo a la regla de Hebb modificada:



Como en el conjunto de datos existen 9 tipos de clases, se usara el conjunto de datos usado para el k vecinos próximos, donde solo existen dos clases: sucedió un evento o no.

A continuación se presentan los resultados del perceptrón para la mitad del conjunto de datos.

|  |  |
| --- | --- |
| **Corrida** | **Resultado** |
| 1 | 368 vectores mal clasificados de 4064 (9.06%) |
| 2 | 363 vectores mal clasificados de 4064 (8.93%) |
| 3 | 2030 vectores mal clasificados de 4064 (49.95%) |
| 4 | 367 vectores mal clasificados de 4064 (9.03%) |
| 5 | 357 vectores mal clasificados de 4064 (8.78%) |

Como se puede observarse, el porcentaje de error en las corridas presentadas, va desde 8.78% hasta 49.95%. Esta variabilidad se debe a que el perceptrón solo toma en cuenta si la clasificación fue correcta o no, pero no se toman en cuenta los aciertos que estuvieron cerca de fallar y se magnifican los errores cometidos por un margen pequeño o corrige los pesos cada que encuentra un vector que se clasifica erróneamente, lo que lo hace susceptible a valores atípicos.

Ahora se realizaron pruebas bajo tres rondas de entrenamiento. Los resultados fueron los siguientes.

Corrida 1:

* **Ronda 1:** 
  + 383 vectores mal clasificados de 4064 (9.42%)
* **Segunda ronda:**
  + 612 vectores de entrenamiento mal clasificados de 4064 (15.06%)
* **Tercera ronda con todos los datos:** 
  + 480 vectores de entrenamiento mal clasificados de 8128 (5.91%)
  + 1671 vectores mal clasificados de 4064 (41.12%)

Corrida 2:

* **Ronda 1:** 
  + 723 vectores mal clasificados de 4064 (17.79%)
* **Segunda ronda:**
  + 639 vectores de entrenamiento mal clasificados de 4064 (15.72%)
* **Tercera ronda con todos los datos:** 
  + 480 vectores de entrenamiento mal clasificados de 8128 (5.91%)
  + 1691 vectores mal clasificados de 4064 (41.61%)

Corrida 3:

* **Ronda 1:** 
  + 321 vectores mal clasificados de 4064 (7.90%)
* **Segunda ronda:**
  + 597 vectores de entrenamiento mal clasificados de 4064 (14.69%)
* **Tercera ronda con todos los datos:** 
  + 480 vectores de entrenamiento mal clasificados de 8128 (5.91%)
  + 1698 vectores mal clasificados de 4064 (41.78%)

Al observar los resultados, el mejor resultado de las tres corridas con todos los datos fue de 41.12%. Este resultado es muy inferior a otros métodos ya vistos.

### Adaline y la regla delta

### A continuación se presentan los resultados de las ejecuciones de la implementación de este.

|  |  |  |
| --- | --- | --- |
| Corrida | Iteraciones | Resultado |
| 1 | 5 | 7261 vectores mal clasificados de 8078 (89.89%) |
| 2 | 10 | 817 vectores mal clasificados de 8078 (10.11%) |
| 3 | 15 | 817 vectores mal clasificados de 8078 (10.11%) |

### Como se observa, a partir de las 10 iteraciones ya no hay una mejora en el porcentaje de aciertos. El mejor resultado fue un 10.11%, mucho mejor que el perceptrón.

### Redes feed-forward y el algoritmo backpropagation

### A continuación se presentan los resultados de las ejecuciones de la implementación de este con números diferentes de neuronas en las capas ocultas.

### Capas ocultas = (2, 2, 1)

|  |  |
| --- | --- |
| Corrida | Resultado |
| 1 | 138 vectores mal clasificados de 4064 (3.39%) |
| 2 | 429 vectores mal clasificados de 4064 (10.55%) |
| 3 | 377 vectores mal clasificados de 4064 (9.27%) |
| 4 | 129 vectores mal clasificados de 4064 (3.17%) |
| 5 | 410 vectores mal clasificados de 4064 (10.088%) |

### Capas ocultas = (3, 3, 1)

|  |  |
| --- | --- |
| Corrida | Resultado |
| 1 | 140 vectores mal clasificados de 4064 (3.44%) |
| 2 | 136 vectores mal clasificados de 4064 (3.34%) |
| 3 | 141 vectores mal clasificados de 4064 (3.46%) |
| 4 | 135 vectores mal clasificados de 4064 (3.32%) |
| 5 | 129 vectores mal clasificados de 4064 (3.17%) |

### Capas ocultas = (4, 4, 2)

|  |  |
| --- | --- |
| Corrida | Resultado |
| 1 | 406 vectores mal clasificados de 4064 (9.99%) |
| 2 | 112 vectores mal clasificados de 4064 (2.75%) |
| 3 | 415 vectores mal clasificados de 4064 (10.21%) |
| 4 | 127 vectores mal clasificados de 4064 (3.125%) |
| 5 | 416 vectores mal clasificados de 4064 (10.23%) |