



Facultad de Ingeniería

Escuela de Ingeniería en Bioinformática

---

# Covid-19, ETL, Bases de Datos NoSql y Aprendizaje Automático

## Base de Datos Avanzada 2020-1

---

*Alumno:*

Claudio Quevedo G

*Profesor:*

Alejandro Valdés

# Índice

<b>1. Introducción</b>	<b>6</b>
<b>2. Problemática</b>	<b>8</b>
2.1. ¿Qué es el Covid19? . . . . .	8
2.2. ¿Qué significan los diferentes nombres? . . . . .	8
2.3. Repositorios Covid-19 de los estados de Estados Unidos . . . . .	9
2.4. ¿Porqué involucrar la política? . . . . .	9
2.5. Influencia de la densidad poblacional . . . . .	9
2.6. ¿Se debería tomar en cuenta la temperatura? . . . . .	10
2.7. ¿Los niveles de empleabilidad y pobreza podrían afectar? . . . . .	10
2.8. ¿Afectaría el nivel de ruralidad de los estados? . . . . .	10
2.9. ¿Podría afectar la velocidad y nivel de cobertura de internet en los distintos estados? . . . . .	10
<b>3. Pre-procesamiento y ETL</b>	<b>12</b>
<b>4. Bases de Datos NoSQL</b>	<b>19</b>
4.1. ¿Que son las BD's NoSQL? . . . . .	19
4.2. ¿Cómo funciona una base de datos NoSQL? . . . . .	19
4.3. ¿Por qué debería usar una base de datos NoSQL? . . . . .	20
4.4. Tipos de bases de datos NoSQL . . . . .	20
4.5. Algoritmo de Aprendizaje Automático K-means. . . . .	21

4.6. MongoDB . . . . .	21
4.6.1. Instalación MongoDB en Ubuntu 18.04 . . . . .	22
4.6.2. Implementación . . . . .	23
4.6.3. Problemas . . . . .	27
4.7. Neo4J . . . . .	27
4.7.1. Instalación de Neo4j en Ubuntu 18.04 . . . . .	28
4.7.2. Implementación . . . . .	29
4.7.3. Problemas . . . . .	33
4.8. Apache Cassandra . . . . .	35
4.8.1. Instalación de Apache Cassandra en Ubuntu 18.04 . . . . .	35
4.8.2. Implementación . . . . .	36
4.8.3. Problemas . . . . .	40
<b>5. Conclusiones</b>	<b>41</b>

# Índice de figuras

1.	Set de datos de migración entre estados . . . . .	14
2.	Set de datos pre-procesados . . . . .	18
3.	Ejemplo de MongoDB . . . . .	23
4.	Resultado script mongo_kmeans.py . . . . .	27
5.	Ejemplo de Neo4j . . . . .	29
6.	Resultado script neo4j_sd.py . . . . .	31
7.	Resultado script neo4j_kmeans.py . . . . .	33
8.	Ejemplo de Apache Cassandra . . . . .	37
9.	Resultado script cassandra_kmeans.py . . . . .	40

## Índice de cuadros

# 1. Introducción

Los coronavirus son una familia de virus que pueden causar enfermedades como el resfriado común, el síndrome respiratorio agudo grave (SARS) y el síndrome respiratorio de Oriente Medio (MERS). En 2019 se identificó un nuevo coronavirus y el 31 de diciembre fue notificado por primera vez la enfermedad por coronavirus (COVID-19) en Wuhan, China. En marzo de 2020 la Organización Mundial de la Salud (OMS) declaró que este brote de COVID-19 es una pandemia. Los signos y síntomas de la enfermedad pueden aparecer entre dos y catorce días después de la exposición al virus (período de incubación), entre los síntomas más comunes se incluyen fiebre, tos seca y cansancio, otros síntomas que pueden ser dificultades respiratorias, dolor musculares y de garganta, la gravedad de estos síntomas puede ser de leve a extrema, aunque se han detectado casos de personas asintomáticas, personas con afecciones crónicas y adultos mayores son considerados un grupo de riesgo de enfermarse gravemente con la enfermedad. A la fecha se sabe que el virus se propaga de persona a persona mediante contacto estrecho y se transmite mediante las gotitas respiratorias que se liberan cuando una persona habla, tose o estornuda, también se indica que puede propagarse por medio de las superficies[1].

Al no existir una vacuna para el tratamiento de la enfermedad, el número de casos confirmados y de muertes por COVID-19 se han disparado a nivel mundial convirtiéndose en un grave problema de salud pública e impactando duramente en la economía global.

Es a partir de esta relevancia que surge la necesidad de hacer un estudio que podría ser importante para determinar desde responsabilidades políticas hasta si hay influencia en la densidad de población, temperatura ambiental, nivel de acceso y conectividad a internet, nivel de pobreza y empleabilidad en cuanto a la cantidad de personas que están infectadas, fallecidas y recuperadas en los distintos estados de Estados Unidos.

Se integraron distintas bases de datos, la primera de ellas es un repositorio de datos operado por el Centro de Ciencias e Ingeniería de Sistemas de la Universidad Johns Hopkins (CSSE)[2], la segunda corresponde a la base de datos perteneciente a Civil Services USA[3], la tercera fué adquirida desde WorldPopulationReview[4], la cuarta desde United States Census[5], la quinta desde Rural Research Brief[6], la sexta desde StateScape Policy tracking and Analysis (The Governors)[7] y por último desde BroadbandNow[8].

La Extracción, Transformación y Carga de datos (ETL - Extract, Transform, Load) es el proceso que permitirá tomar estos datos desde sus múltiples fuentes, reformatearlos y lim-

piarlos, y cargarlos. Estos datos serán inicialmente insertados en distintas bases de datos no relacionales o bases de datos NoSQL (también conocidas como "no solo SQL") las cuales no son tabulares y almacenan datos de manera diferente a las tablas relacionales. Proporcionan esquemas flexibles y se escalan fácilmente con grandes cantidades de datos y altas cargas de usuarios[11].

Las bases de datos NoSql a utilizar son: MongoDB, Neo4j y Apache Cassandra. Por otro lado el lenguaje de programación con el que se harán las implementaciones será Python.

Para poder llevar a cabo el estudio se utilizarán algoritmos de Aprendizaje Automático. Los algoritmos de Aprendizaje Automático utilizan estadísticas para encontrar patrones en ciertas cantidades de datos. Si se puede almacenar digitalmente, se puede alimentar a un algoritmo de aprendizaje automático[9].

Para este estudio se utilizará el algoritmo de aprendizaje K-means, el cuál es un algoritmo de clasificación no supervisada (clusterización) que agrupa objetos en k grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster. Se suele usar la distancia cuadrática. Consta de 3 pasos: inicialización, asignación objetos a los centroides y actualización de centroides[10].

El objetivo de este proyecto es:

1. Investigar sobre diferentes tipos de BDs NoSQL.
2. Instalar y poner en marcha las diferentes BDs NoSQL.
3. Utilizar los mecanismos de gestión de datos de las diferentes BDs NoSQL.
4. Aplicar procedimientos ETL y generar de datos para las diferentes BDs NoSQL.

## 2. Problemática

La enfermedad del coronavirus, una emergencia de salud mundial. Las zonas urbanas son la zona cero de la pandemia del COVID-19, con un 90 % de los casos comunicados. Las ciudades están sufriendo las peores consecuencias de la crisis, muchas de ellas con sistemas de salud sobrecargados, servicios de agua y saneamiento inadecuados y otros problemas.

### 2.1. ¿Qué es el Covid19?

Los coronavirus (CoV) son una gran familia de virus que causan enfermedades que van desde el resfriado común hasta enfermedades más graves, como el síndrome respiratorio agudo severo (SARS-CoV).

El nuevo coronavirus de 2019 es una nueva cepa que no se había visto en humanos hasta ahora y que ha causado neumonía viral. Se relacionó por primera vez con el mercado de la ciudad de mariscos del sur de China de Wuhan, que es un mercado mayorista de mariscos y animales vivos en diciembre de 2019.

El virus ahora se ha detectado en varias áreas de China, junto con países de Asia, América del Norte y del Sur, Europa, África y Oceanía.

### 2.2. ¿Qué significan los diferentes nombres?

Es posible que hayas notado que circulan diferentes nombres relacionados con el nuevo coronavirus. Bueno, partiendo por COVID-19: este es el nombre de la enfermedad causada por el coronavirus. Esto es simplemente la abreviatura de la enfermedad por coronavirus 2019. La Organización Mundial de la Salud anunció este nombre el 11 de febrero de 2020. Luego está SARS-CoV-2- coronavirus grave relacionado con el síndrome respiratorio agudo 2. Este es el nombre del virus, no la enfermedad que resulta de él. La Organización Mundial de la Salud enfatiza que si bien los virus están relacionados, COVID-19 es diferente del brote de SARS de 2003. También está Nuevo coronavirus 2019 (nCoV-19): esto se usó inicialmente al comienzo del brote. Se refiere al virus, que es una forma novedosa del coronavirus que se vio por primera vez en 2019[12]. Y finalmente coronavirus: es posible que vea o escuche sobre el virus denominado coronavirus. Esto no es inexacto, ya que es una cepa novedosa de un coronavirus.



## 2.3. Repositorios Covid-19 de los estados de Estados Unidos

Este repositorio de datos operado por la Universidad Johns Hopkins (CSSE)[2] entrega la información de la población, cantidad de infectados, cantidad de fallecidos, la cantidad de recuperados y la migración en cada estado de Estados Unidos en una respectiva fecha, que para este proyecto se utilizaron los datos del 26-05-2020.

## 2.4. ¿Porqué involucrar la política?

Los dos partidos políticos principales e históricamente más grandes de los Estados Unidos son los republicanos y los demócratas y, después de cada elección, tienen los puestos mayoritarios en la cámara de representantes y el senado, así como el mayor número de gobernadores. Aunque ambas partes tienen buenas intenciones para los ciudadanos estadounidenses, tienen diferencias claras que se manifiestan en sus comentarios, decisiones e historia. Estas diferencias son principalmente caminos ideológicos, políticos, sociales y económicos para hacer que Estados Unidos tenga éxito y que el mundo sea un lugar mejor para ellos. Dichas diferencias en los partidos políticos podrían ser responsables en la toma de decisiones llevadas a cabo durante la pandemia en los distintos estados.

La forma en la que se trabajó esta información es con tipo de dato booleano, específicamente 0 si es demócrata o 1 si es republicano en cada estado[7].

## 2.5. Influencia de la densidad poblacional

Como hipótesis personal pienso que sí podría influir éste parámetro en el aumento o disminución de probabilidad de contagio que puede sufrir un estado, puesto que si hay una mayor concentración de personas en una determinada zona, deberían aumentar la cantidad de contagios, por el contrario si hay una baja concentración de personas en una zona muy grande, deberían haber menos contagios. Esto se corroborará mas adelante en el análisis de resultados.

Los datos están en persona por milla cuadrada[4], 1 milla equivale a 1.6km (este dato es decimal).

## **2.6. ¿Se debería tomar en cuenta la temperatura?**

Como nombré anteriormente, esta enfermedad ataca al sistema respiratorio principalmente, por lo que considero que las temperaturas estarían directamente involucradas, de igual forma se corroborará mas adelante en el análisis de resultados.

Se tienen las temperaturas promedio anual de cada estado[4], están en formato Fahrenheit, por lo cuál en el pre-procesamiento de datos se cambió a grados Celsius.

## **2.7. ¿Los niveles de empleabilidad y pobreza podrían afectar?**

Considero que es muy probable, debido a que con una situación económica poco estable obligaría a las personas a tener que abandonar la cuarentena e ir a sus puestos de trabajos, sobretodo aquellas personas que tienen sus trabajos de forma particular y no con un contrato. Al no cumplir con la cuarentena debería aumentar los niveles de contagio de esta enfermedad, esto se verá en el análisis de datos.

La forma en que se procesó los datos[5] es primero por la cantidad de personas con empleo seguro (contrato) y en segundo lugar por el porcentaje de pobreza que tiene cada estado.

## **2.8. ¿Afectaría el nivel de ruralidad de los estados?**

Es interesante este punto, puesto que ya se esta revisando si influiría la densidad poblacional en la cantidad de contagios que sufre cada estado, por lo tanto sí debería influir el nivel de ruralidad[6], a mayor nivel de ruralidad menor nivel de urbanismo. Esto quiere decir que hay menos contacto entre personas, por lo que debería haber menos densidad poblacional y menos contagios.

## **2.9. ¿Podría afectar la velocidad y nivel de cobertura de internet en los distintos estados?**

Sinceramente dudo que afecte de forma importante, de igual forma no es algo que se deba descartar totalmente, debido a la pandemia muchos están trabajando y estudiando

de forma online, por lo que una buena o mala calidad en la navegación podría afectar en que una persona tenga que abandonar la cuarentena para poder responder a las distintas responsabilidades que el modo online conlleva.

### 3. Pre-procesamiento y ETL

Los registros se encuentran en diferentes formatos en las distintas bases de datos, por lo que es necesario transformarla y ordenarla para que la data se le parezca (dentro de ello se deben sacar las comillas dobles que hay en el archivo estados.csv). Para realizar este parseo se a utilizado el editor de texto VI y expresiones regulares. En primer lugar se abre el documento en la terminal, escribiendo: `vi estados.csv`, una vez dentro del documento se escribe la siguiente expresión regular simple: « `:%s/"/g` ». A través de esta expresión lo primero que realizo es que en todas las filas se eliminen todas las comillas que se encuentren en el documento.

Para extraer el nombre del estado y su respectiva población se realizará a través de comando bash, en el cual debe imprimirse sólo la columna 1 y la columna 10 del archivo parseado anteriormente: « `awk 'BEGIN{OFS=FS=","}{print $1,$10}' estados.csv` », la cuál se guardará como **estados\_parsed.csv**.

Al realizar este paso, se observa que hay un error, como se puede ver en el siguiente ejemplo no aparece la población, sino más bien un link de una página web en el estado de Minnesota:

```
Kentucky 4395295
Louisiana 4625470
Maine 1328302
Maryland 5928814
Massachusetts 6692824
Michigan 9895622
Minnesota http://www.stpaul.gov
```

A través del análisis de datos se observa que el error se produce debido a que los números se encuentran con coma, por ejemplo el 10000, lo tienen como 10,000 (esto se logró encontrar utilizando el editor de texto VI y buscando el estado de Minnesota), se cambió la “,” por un “.” y de este modo se soluciona este problema.

Otro problema es la cantidad de estados, EEUU tiene 50 estados, pero en los datos de Covid-19[2] aparecen 58, esto es debido a que están los estados libres asociados (Puerto Rico por ejemplo), se puede hacer 2 cosas, buscar manualmente la población de esos 8 estados faltantes o se pueden descartar y tomar en consideración solo los 50 estados correspondientes. Para solucionar este problema, me base en la información de los estados de Servicios

Civiles[3], por lo tanto solo consideraré los 50 estados.

En la información de COVID-19[2], habían muchas columnas con datos vacíos en contagiados, muertos y recuperados, al utilizar los 50 estados (en Python) logré notar que la información que faltaba era de los estados libres asociados, por lo tanto confirmo que la acción correcta fue dejarlos de lado.

Para la información de los datos de migración entre estados se utilizará lo que entrega United States Census[5], en nuestro caso nos interesa el total, para obtener la tasa de migración de dichos estados. No es fácil que la máquina interprete los datos entregados en formato excel, así que se procedió a seleccionar los datos de relevancia, esto quiere decir que serían los datos desde A10 hasta I78, el cual se traspasó a un archivo csv llamado **migracion.csv**.

Luego se hizo un cambio en el formato de los números, puesto que estaban con comas (109,301), se realizó con el mismo paso anterior utilizando el editor de texto VI: « :%s/,//g ». También se cambió el formato de los datos (tenían saltos de líneas innecesarios y tabulaciones innecesarias), esto se realizó con VI: « :%s/\t;/g », el cuál cambia tabulaciones por punto y coma. Luego se detecta el patrón “;;;;;;;;” y se elimina también con VI: « :%s/;;;;;;;;//g ». Y finalmente se eliminan saltos de línea innecesarios con VI: « :%s/\n//g ».

```

United States;2;323531965;+/- 28863;278079469;+/- 191887;35921585;+/- 160760;7571282;+/- 77765
Alabama;4832358;+/- 3328;4173026;+/- 19530;534914;+/- 18557;109301;+/- 9827
Alaska;727164;+/- 1214;608707;+/- 7474;81229;+/- 5883;32099;+/- 4643
Arizona;7090137;+/- 4507;5887700;+/- 28309;888452;+/- 25934;273714;+/- 15128
Arkansas;2975961;+/- 3093;2539294;+/- 19535;354774;+/- 17489;73179;+/- 7023
California;39114889;+/- 11156;34232867;+/- 55636;4092388;+/- 50349;501023;+/- 17029
Colorado;5633029;+/- 3632;4626834;+/- 24416;728730;+/- 20205;239153;+/- 15015
Connecticut;3538203;+/- 3043;3102312;+/- 17295;320248;+/- 14157;84718;+/- 7195
Delaware;958770;+/- 1413;853523;+/- 7519;67984;+/- 6209;34451;+/- 4392
District of Columbia ;693798;+/- 1873;568201;+/- 6930;68603;+/- 5579;47555;+/- 4011
Florida;21092877;+/- 8313;17863603;+/- 57744;2420485;+/- 51354;587261;+/- 22336
Georgia;10410462;+/- 5598;8909803;+/- 34965;1174523;+/- 34817;274273;+/- 15590
Hawaii;1403653;+/- 1798;1211015;+/- 10611;125172;+/- 9102;54074;+/- 5497
Idaho;1734756;+/- 1843;1438727;+/- 13783;209540;+/- 11848;79765;+/- 7651
Illinois;12599244;+/- 4929;11029483;+/- 28667;1295596;+/- 27638;206620;+/- 12209
Indiana;6612948;+/- 4582;5614432;+/- 25211;821113;+/- 19953;145256;+/- 9569
Iowa;3119306;+/- 2428;2662849;+/- 14455;371285;+/- 12931;74199;+/- 4846
Kansas;2874208;+/- 2798;2392841;+/- 17716;371025;+/- 16909;95748;+/- 7841
Kentucky;4414714;+/- 3279;3744188;+/- 18273;547535;+/- 17640;110456;+/- 8385
Louisiana;4603081;+/- 3921;4033845;+/- 20655;474843;+/- 19702;81355;+/- 7131
Maine;1324095;+/- 1563;1143612;+/- 8051;142453;+/- 7284;34706;+/- 3872
Maryland;5974521;+/- 4649;5133916;+/- 25668;630955;+/- 20758;163216;+/- 10535
Massachusetts;6834364;+/- 3940;5970065;+/- 27037;653940;+/- 24452;144042;+/- 7468
Michigan;9891083;+/- 5094;8562101;+/- 27627;1144840;+/- 25036;139276;+/- 9581
Minnesota;5543490;+/- 3354;4781432;+/- 18432;635928;+/- 16956;100270;+/- 6150
Mississippi;2955717;+/- 2710;2582347;+/- 16553;297788;+/- 14562;68000;+/- 7708
Missouri;6053109;+/- 4138;5161553;+/- 20177;720626;+/- 18922;152326;+/- 8935
Montana;1051757;+/- 1381;891554;+/- 8487;113693;+/- 6464;43375;+/- 4560
Nebraska;1905843;+/- 2229;1602056;+/- 12636;239853;+/- 10425;54579;+/- 5668
Nevada;3003761;+/- 2581;2495398;+/- 19156;366629;+/- 16899;127362;+/- 9279
New Hampshire;1344750;+/- 1556;1170412;+/- 9723;119936;+/- 8415;48367;+/- 4520
New Jersey;8815804;+/- 5090;7870909;+/- 27774;722620;+/- 23628;159846;+/- 11726
New Mexico;2072546;+/- 2647;1797834;+/- 15105;204099;+/- 13898;60068;+/- 6617
New York;19329704;+/- 7489;17327068;+/- 36048;1614505;+/- 32857;254447;+/- 9994
North Carolina;10274540;+/- 4937;8723401;+/- 36579;1178884;+/- 32327;318681;+/- 14755
North Dakota;749825;+/- 1391;623183;+/- 7019;89228;+/- 6522;33735;+/- 4338
Ohio;11555163;+/- 5668;9892715;+/- 29438;1426959;+/- 27610;198026;+/- 10200
Oklahoma;3895983;+/- 2671;3226343;+/- 19150;548493;+/- 16807;106943;+/- 7091
Oregon;4152756;+/- 2818;3438653;+/- 21057;554097;+/- 19988;140627;+/- 8621
Pennsylvania;12674699;+/- 5544;11119060;+/- 29386;1239854;+/- 27201;253520;+/- 11151
Rhode Island;1047975;+/- 1870;928114;+/- 8740;78525;+/- 6998;32841;+/- 3919
South Carolina;5029034;+/- 4100;4309043;+/- 19661;511153;+/- 17925;186505;+/- 10214
South Dakota;871271;+/- 1414;740970;+/- 7483;100177;+/- 7064;26775;+/- 3764
Tennessee;6692936;+/- 4573;5706212;+/- 25189;748791;+/- 24332;209377;+/- 12422
Texas;28333482;+/- 10498;23992374;+/- 61744;3575604;+/- 57916;563945;+/- 24840
Utah;3114717;+/- 3345;2583764;+/- 17461;408033;+/- 13986;100824;+/- 7505
Vermont;621681;+/- 937;542177;+/- 5046;50818;+/- 4088;26306;+/- 3523
Virginia;8424340;+/- 5001;7142767;+/- 30604;947475;+/- 29018;274003;+/- 13094
Washington;7453439;+/- 4565;6101533;+/- 28433;1028654;+/- 26957;260906;+/- 14324
West Virginia;1787630;+/- 2061;1585832;+/- 8806;156420;+/- 8563;43841;+/- 4205
Wisconsin;5751525;+/- 3027;4960260;+/- 19314;659159;+/- 17572;114613;+/- 6968
Wyoming;570867;+/- 1239;479561;+/- 7122;62957;+/- 6074;25734;+/- 3856

```

Figura 1: Set de datos de migración entre estados

NOTA: aún falta hacer el cálculo de la tasa de migración, solo tenemos el número de migraciones totales por estado en el año 2018.

Considerando que los datos que se presentan en el archivo **partido\_politico.csv**[7] se encuentran en español, se extraerá lo necesario utilizando awk, mediante las siguientes sentencias: « awk 'BEGIN{OFS=FS="\t"}{print \$2}' partido\_politico.csv », para obtener estado en español. Y « awk 'BEGIN{OFS=FS="\t"}{print \$4}' partido\_politico.csv », para obtener partido político en español.

Una vez que se obtienen los campos descritos con anterioridad (nombre de estado y orientación política), se traducirá a inglés el resultado completo, de esta forma se logrará una agrupación adecuada de la nueva información, con lo obtenido de manera previa, resultando el archivo **pp\_datasets.csv**.

A partir del archivo **densidad\_poblacion.csv**[4] se extraerá los datos de densidad para los estados correspondientes. Dado que estos datos se encuentran con los estados desordenados, además de extraer la información, será necesario ordenar la misma. Para esto se utilizó la línea de comando que se muestra a continuación, implementando las herramientas AWK, SORT y TAIL de bash: « tail -n +3 densidad\_poblacion.csv|sort -k1|awk 'BEGIN{OFS=FS="\t";print "state\tdensity"}{print \$1,\$2}' », la información que entrega este comando recibe el nombre de **pd\_datasets.csv**.

Sumado a lo anterior, se trabajará con el archivo **temperaturas.csv**[4], con este se extraerán los datos de temperatura por estado, transformando de grados Fahrenheit a Celsius, además de ordenar estos datos por nombre de estado. Esto, de igual manera que en el punto anterior, utilizando AWK, SORT y TAIL de bash: « tail -n +3 temperaturas.csv|sort -k1|awk 'BEGIN{OFS=FS="\t";print "state\tdtemperature"}{print \$1,\$2-32\*0.5556}' », quedando como resultado el archivo **tcelsius\_datasets.csv**, que tiene los estados y las temperaturas promedio de lo que va del año 2020 en grados Celsius de dichos estados.

Extracción de la cantidad de empleo por estado a partir del archivo **nivel\_empleo.csv**[5], eliminando los datos que no tienen relación con la problemática de este proyecto, esto lo realicé por medio de las herramientas AWK y TAIL de bash: « tail -n +13 nivel\_empleo.csv| awk 'BEGIN{OFS=FS="\t"}{print \$2,\$NF}' », quedando como **resultado\_ne\_datasets.csv**, que tiene lo requerido en este punto.

El pre-procesamiento en este punto es sobre el archivo **porcentaje\_pobreza.csv**[5]. Para comenzar, se cambia el formato de cada uno de los datos. Esto debido a que después de cada nombre de estado, proseguía una serie de puntos (“.”), los cuales perjudicarán al script **pre\_process.py** (ya se hablará de este script más adelante). Para esto utilicé el siguiente

comando, por medio de la herramienta VI: « : %s/\.\{1,\}\t/\t/g », sumado a lo anterior, se elimina la línea “District of Columbia”, debido a que este no corresponde a un estado. Finalmete, se utiliza la herramienta SORT para ordenar los datos por estado: « sort -k1 porcentaje\_pobreza.csv », el archivo resultante corresponde a **ppobreza\_datasets.csv**, el cual contiene la información expuesta con anterioridad.

En cuanto al apartado “niveles de ruralidad por estado”, se trabajará sobre el archivo **poblacion\_rural.csv**[6]. En este se extraen las columnas que contienen los estados y porcentaje de población rural en dicho estado; además de realizar cambios de formato de la información por medio de la herramienta VI, se procede a la eliminación de la línea “District of Columbia”, el cual, como se mencionó en el punto anterior, no es un estado. Las herramientas que se requieren para lograr esta tarea son AWK, SORT y TAIL: « tail -n +2 poblacion\_rural.csv|sort -k1|awk ‘BEGIN{OFS=FS=“\t”;print “state\t rural\_percentage”}{print \$1,\$3}’ », el archivo resultante se llamará **prural\_datasets.csv**, el cual posee los datos de estados y el porcentaje de ruralidad por estado.

Finalmente, se trabajará sobre el archivo **conectividad.csv**[8] y se extraerá la información correspondiente al promedio de velocidad de internet y el porcentaje de cobertura por cada uno de los estados. Al igual que en la mayoría de los casos expuestos, las herramientas utilizadas son AWK, SORT y TAIL de bash: « tail -n +2 conectividad.csv|sort -k1|awk ‘BEGIN{OFS=FS=“\t”;print “state\t ASpeed\_Mbps\t BCoverage\_percent”}{print \$1,\$3,\$4}’ », **cv\_datasets.csv** corresponde al nombre del archivo resultante, el cual contiene el estado, el promedio de velocidad de internet y el porcentaje de cobertura por cada uno de los estados.

Para realizar el pre-procesamiento final de datos y obtener los atributos definidos anteriormente para la problemática, se realizó un script en Python llamado **pre\_process.py**[13], el cual lee los archivos creados en los pasos anteriores, los cuales serían: **reporte\_covid19.csv**, **estados\_parsed.csv**, **migracion.csv**, **pp\_datasets.csv**, **pd\_datasets.csv**, **tcelsius\_datasets.csv**, **ne\_datasets.csv**, **ppobreza\_datasets.csv** y **cv\_datasets.csv**. Luego de la lectura de estos archivos el script hace la selección, el cálculo de tasa migratoria de cada estado y la unión de columnas de datos separadas por “;” respecto a un estado.



```

#Script: pre_process.py
import csv
data_covid = []
data_estados = []
data_migracion = []
data_nivel_empleo = []
with open('reporte_covid19.csv') as File:
    reader = csv.reader(File, delimiter=',', quotechar=';', quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        data_covid.append(row)
with open('estados_parsed.csv') as File:
    reader = csv.reader(File, delimiter=',', quotechar=';', quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        data_estados.append(row)
with open('migracion.csv') as File:
    reader = csv.reader(File, delimiter=';', quotechar=';', quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        data_migracion.append(row)
data_hito1 = []
for row in data_estados:
    for row2 in data_covid:
        for row3 in data_migracion:
            if (row[0] == row2[0] and row[0] == row3[0]):
                tasa_migracion=int(row3[7])/int(row[1])
                data_hito1.append([row[0],row[1],row2[5],
                    row2[6],row2[7],tasa_migracion])
#script incompleto, faltan: pp_datasets.csv,
pd_datasets.csv, tcelsius_datasets.csv, ne_datasets.csv,
ppobreza_datasets.csv y cv_datasets.csv.
print("state",";", "population",";", "infected",";", "death",
";", "recovered",";", "migration",";", "politic",";", "density",
";", "temperature",";", "employment",";", "poverly",
";", "netspeed",";", "netcoverage")
for row in data_hito2_6:
    print(row[0], ";", row[1], ";", row[2], ";", row[3], ";",
row[4], ";", row[5], ";", row[6], ";", row[7], ";", row[8], ";",
row[9], ";", row[10], ";", row[11], ";", row[12])

```

El script **pre\_process.py** lo ejecuté por medio de la terminal utilizando python3, de la siguiente forma: « python3.6 pre\_process.py », resultando lo que se muestra en la Figura 2.

```
cquevedo@cquevedo:~/bd_avanzada$ cat datasets
state ; population ; infected ; death ; recovered ; migration ; politic ; density ; temperature ; employment ; poverty ; netspeed ; netcoverage
Alabama ; 4833722 ; 14986 ; 566 ; 7951 ; 0.022612181668701676 ; 1 ; 96.9221 ; 45.0208 ; 4455590 ; 16.0 ; 33.7 ; 81.2
Alaska ; 735132 ; 408 ; 10 ; 361 ; 0.043664267097609684 ; 1 ; 1.2863 ; -2.0000 ; 718584 ; 12.2 ; 27.9 ; 78.2
Arizona ; 6626624 ; 16575 ; 807 ; 4204 ; 0.04130519552640893 ; 1 ; 64.9549 ; 42.5208 ; 6318657 ; 14.4 ; 33.9 ; 86.7
Arkansas ; 2959373 ; 6029 ; 117 ; 4249 ; 0.02472787310014655 ; 1 ; 58.4030 ; 42.6208 ; 2728483 ; 15.5 ; 25.0 ; 76.7
California ; 38332521 ; 96400 ; 3769 ; 92631.06 ; 0.013070442197109865 ; 0 ; 256.3728 ; 41.6208 ; 39623408 ; 12.5 ; 29.0 ; 94.5
Colorado ; 5268367 ; 24256 ; 1333 ; 3666 ; 0.04539414205578313 ; 0 ; 56.4012 ; 27.3208 ; 6317666 ; 8.9 ; 40.9 ; 90.4
Connecticut ; 3596080 ; 40873 ; 3742 ; 7127 ; 0.023558430290761052 ; 0 ; 735.8695 ; 31.2208 ; 4071818 ; 9.9 ; 41.8 ; 99
Delaware ; 925749 ; 8965 ; 332 ; 4693 ; 0.037214190887594804 ; 0 ; 504.3073 ; 37.5208 ; 1052950 ; 9.6 ; 44.9 ; 97.5
Florida ; 19552860 ; 51746 ; 2252 ; 49494.012 ; 0.030034532032654045 ; 1 ; 410.1259 ; 52.9208 ; 21640391 ; 13.6 ; 41.2 ; 94.9
Georgia ; 9992167 ; 43400 ; 1848 ; 41552.013 ; 0.027448800645545657 ; 1 ; 186.6726 ; 45.7208 ; 10039531 ; 14.7 ; 39.1 ; 90.6
Hawaii ; 1404054 ; 643 ; 17 ; 591 ; 0.03851276375410063 ; 0 ; 219.9424 ; 52.2208 ; 1453877 ; 9.5 ; 22.5 ; 96
Idaho ; 1612136 ; 2626 ; 79 ; 1755 ; 0.049477835616846225 ; 1 ; 22.0970 ; 26.6208 ; 1611623 ; 11.3 ; 25.6 ; 82
Illinois ; 12882135 ; 112017 ; 4885 ; 107132.017 ; 0.016039266783029367 ; 0 ; 228.0246 ; 34.0208 ; 14299174 ; 10.9 ; 40.4 ; 92.3
Indiana ; 6570902 ; 31715 ; 1984 ; 29731.018 ; 0.02210594527204941 ; 1 ; 188.2809 ; 33.9208 ; 7188014 ; 11.8 ; 36.7 ; 85.7
Iowa ; 3090416 ; 17557 ; 456 ; 9377 ; 0.024009389027237756 ; 1 ; 56.9284 ; 30.0208 ; 3575063 ; 9.2 ; 24.7 ; 83.7
Kansas ; 2893957 ; 9125 ; 207 ; 493 ; 0.03308549505054843 ; 0 ; 35.5968 ; 36.5208 ; 3202927 ; 10.8 ; 39.9 ; 81.9
Kentucky ; 4395295 ; 8571 ; 391 ; 3102 ; 0.025130508873693346 ; 0 ; 113.9566 ; 37.8208 ; 4203140 ; 15.3 ; 30.5 ; 85.3
Louisiana ; 4625470 ; 37809 ; 2691 ; 28700 ; 0.017588482900116095 ; 0 ; 107.5174 ; 48.6208 ; 4579023 ; 19.8 ; 35.1 ; 84.6
Maine ; 1328302 ; 2074 ; 78 ; 1290 ; 0.026128094364082866 ; 0 ; 43.6336 ; 23.2208 ; 1432474 ; 12.6 ; 21.2 ; 89.7
Maryland ; 5928814 ; 47152 ; 2302 ; 3329 ; 0.02752928325968735 ; 1 ; 626.6735 ; 36.4208 ; 6206853 ; 7.1 ; 51.3 ; 96.7
Massachusetts ; 6692824 ; 93271 ; 6416 ; 86855.025 ; 0.021521856842492797 ; 1 ; 894.4359 ; 30.1208 ; 8643460 ; 9.8 ; 43.5 ; 97.6
Michigan ; 9895622 ; 54881 ; 5241 ; 33168 ; 0.01407450688799552 ; 0 ; 177.6650 ; 26.6208 ; 10241015 ; 11.0 ; 28.7 ; 88.3
Minnesota ; 5420380 ; 21315 ; 890 ; 14816 ; 0.01849870304296009 ; 0 ; 71.5922 ; 23.4208 ; 7020027 ; 8.7 ; 36.8 ; 88.1
Mississippi ; 2991207 ; 13458 ; 635 ; 9401 ; 0.02273329796299621 ; 1 ; 63.7056 ; 45.6208 ; 2478495 ; 19.8 ; 25.2 ; 70.2
Missouri ; 6044171 ; 12476 ; 689 ; 11787.029 ; 0.025202132765601768 ; 1 ; 89.7453 ; 36.8208 ; 6586786 ; 11.9 ; 38.5 ; 80.1
Montana ; 1015165 ; 479 ; 16 ; 441 ; 0.042727044372097144 ; 0 ; 7.4668 ; 24.9208 ; 1105698 ; 10.4 ; 20.3 ; 69.2
Nebraska ; 1868516 ; 12362 ; 147 ; 12215.031 ; 0.029209811422540668 ; 1 ; 25.4161 ; 31.0208 ; 2225692 ; 10.4 ; 27.1 ; 82.4
Nevada ; 2790136 ; 7956 ; 394 ; 359 ; 0.04564723726728733 ; 0 ; 28.5993 ; 32.1208 ; 3056642 ; 12.2 ; 34.3 ; 91.8
New Hampshire ; 1323459 ; 4197 ; 210 ; 2204 ; 0.03654589979742478 ; 1 ; 153.1610 ; 26.0208 ; 1616350 ; 6.4 ; 37.4 ; 94.3
New Jersey ; 8899339 ; 155092 ; 11147 ; 25253 ; 0.017961558718012652 ; 0 ; 1215.1985 ; 34.9208 ; 9859037 ; 9.1 ; 52.0 ; 99
New Mexico ; 2085287 ; 7026 ; 320 ; 2464 ; 0.028805627235004103 ; 0 ; 17.2850 ; 35.6208 ; 1713657 ; 18.7 ; 30.0 ; 80.3
New York ; 19651127 ; 362764 ; 29229 ; 64280 ; 0.0129482141151549731 ; 0 ; 412.5218 ; 27.6208 ; 22147985 ; 11.8 ; 45.2 ; 97.8
North Carolina ; 9848060 ; 24057 ; 790 ; 14954 ; 0.03235977441242235 ; 0 ; 218.2710 ; 41.2208 ; 9868070 ; 14.1 ; 42.4 ; 93.1
North Dakota ; 723393 ; 2457 ; 54 ; 1551 ; 0.04663440204702009 ; 1 ; 11.0393 ; 22.6208 ; 944423 ; 10.9 ; 28.6 ; 92.2
Ohio ; 11570808 ; 32477 ; 1987 ; 30490.039 ; 0.017114275856966947 ; 1 ; 287.5040 ; 32.9208 ; 12502230 ; 12.8 ; 32.1 ; 91.9
Oklahoma ; 3850568 ; 6090 ; 313 ; 4714 ; 0.02777330513316477 ; 1 ; 57.6546 ; 41.8208 ; 3679022 ; 14.0 ; 40.2 ; 73.3
Oregon ; 3930065 ; 3949 ; 148 ; 1376 ; 0.035782359833743206 ; 0 ; 44.8086 ; 30.6208 ; 4399279 ; 10.6 ; 39.1 ; 89.7
Pennsylvania ; 12773801 ; 71925 ; 5146 ; 40911 ; 0.019846872516645592 ; 0 ; 286.5454 ; 31.0208 ; 14219391 ; 11.5 ; 41.4 ; 94.7
Rhode Island ; 1051511 ; 14065 ; 608 ; 1119 ; 0.031232198236632806 ; 0 ; 1021.4313 ; 32.3208 ; 1182747 ; 10.3 ; 46.7 ; 98.2
South Carolina ; 4774839 ; 10178 ; 440 ; 6063 ; 0.03905995573882177 ; 1 ; 173.3176 ; 44.6208 ; 4840298 ; 14.1 ; 39.3 ; 88.2
South Dakota ; 844877 ; 4586 ; 50 ; 3415 ; 0.03169100354252749 ; 1 ; 11.9116 ; 27.4208 ; 1004981 ; 11.8 ; 26.8 ; 85.3
Tennessee ; 6495978 ; 20535 ; 338 ; 13073 ; 0.03223179019387073 ; 1 ; 167.2749 ; 39.8208 ; 6780993 ; 12.6 ; 36.6 ; 89.9
Texas ; 26448193 ; 56409 ; 1533 ; 35292 ; 0.02132262873308585 ; 1 ; 112.8204 ; 47.0208 ; 27507052 ; 13.7 ; 46.9 ; 86.9
Utah ; 2900872 ; 8521 ; 98 ; 5218 ; 0.0347564456480672 ; 1 ; 39.9430 ; 30.8208 ; 3361622 ; 7.9 ; 37.2 ; 94.9
Vermont ; 626630 ; 962 ; 54 ; 843 ; 0.04198011585784275 ; 1 ; 68.1416 ; 25.1208 ; 736384 ; 9.6 ; 22.4 ; 86.1
Virginia ; 8260405 ; 37727 ; 1208 ; 5145 ; 0.0331706496231117 ; 0 ; 218.4404 ; 37.3208 ; 8712574 ; 10.7 ; 48.7 ; 89.9
Washington ; 6971406 ; 20065 ; 1070 ; 18995.053 ; 0.037425162155238126 ; 0 ; 117.3273 ; 30.5208 ; 7474534 ; 10.3 ; 41.0 ; 94.2
West Virginia ; 1854304 ; 1774 ; 72 ; 1135 ; 0.023642833106114206 ; 1 ; 73.9691 ; 34.0208 ; 1466658 ; 17.2 ; 29.9 ; 75.2
Wisconsin ; 5742713 ; 15584 ; 514 ; 9207 ; 0.019957988497771 ; 0 ; 108.0496 ; 25.3208 ; 6787539 ; 9.5 ; 37.4 ; 85.3
Wyoming ; 582658 ; 843 ; 12 ; 575 ; 0.04416656082985216 ; 1 ; 5.8400 ; 24.2208 ; 591087 ; 11.4 ; 26.8 ; 74.5
```

Figura 2: Set de datos pre-procesados

Las 13 columnas con la información nombrada anteriormente generadas a partir del script **pre\_process.py**.

## 4. Bases de Datos NoSQL

### 4.1. ¿Que son las BD's NoSQL?

Las bases de datos NoSQL están diseñadas específicamente para modelos de datos específicos y tienen esquemas flexibles para crear aplicaciones modernas. Las bases de datos NoSQL son ampliamente reconocidas por su facilidad de desarrollo, funcionalidad y rendimiento a escala[14].

Las bases de datos NoSQL (acrónimo de Not Only SQL) es la innovación de almacenamiento de datos, de hecho grandes empresas están aprovechando sus ventajas en cuanto al rendimiento obtenido. Es el caso de empresas como AirFrance, KPMG Francia y Ciudad de Chicago, quienes se inclinaron hacia el uso de una base de datos NoSQL, cada vez más popular, dentro de la arquitectura de sus proyectos de Big Data.

Gran parte de la información generada hoy en la web es no estructurada, por lo que el modelo relacional resulta lento y no es el más apropiado para grandes cantidades de datos.

### 4.2. ¿Cómo funciona una base de datos NoSQL?

Las bases de datos NoSQL utilizan una variedad de modelos de datos para acceder y administrar datos. Estos tipos de bases de datos están optimizados específicamente para aplicaciones que requieren un gran volumen de datos, baja latencia y modelos de datos flexibles, que se logran al relajar algunas de las restricciones de consistencia de datos de otras bases de datos[14].

En una base de datos relacional, un registro de libro a menudo se desarma (o "normaliza") y se almacena en tablas separadas, y las relaciones se definen por restricciones de clave primaria y externa. El modelo relacional está diseñado para permitir que la base de datos imponga integridad referencial entre tablas en la base de datos, normalizada para reducir la redundancia y, en general, optimizada para el almacenamiento.

En una base de datos NoSQL, un registro de libro generalmente se almacena como un documento JSON. En este modelo, los datos están optimizados para un desarrollo intuitivo

y escalabilidad horizontal.

### 4.3. ¿Por qué debería usar una base de datos NoSQL?

Las bases de datos NoSQL son ideales para muchas aplicaciones modernas, como dispositivos móviles, web y juegos, que requieren bases de datos flexibles, escalables, de alto rendimiento y altamente funcionales para proporcionar excelentes experiencias de usuario[14].

1. Flexibilidad: las bases de datos NoSQL generalmente proporcionan esquemas flexibles que permiten un desarrollo más rápido e iterativo.
2. Escalabilidad: las bases de datos NoSQL generalmente están diseñadas para escalar utilizando clústeres distribuidos de hardware en lugar de escalar agregando servidores caros y robustos. Algunos proveedores de la nube manejan estas operaciones detrás de escena como un servicio totalmente administrado.
3. Alto rendimiento: la base de datos NoSQL está optimizada para modelos de datos específicos y patrones de acceso que permiten un mayor rendimiento que intentar lograr una funcionalidad similar con bases de datos relacionales.
4. Altamente funcional: las bases de datos NoSQL proporcionan API y tipos de datos altamente funcionales diseñados específicamente para cada uno de sus respectivos modelos de datos.

### 4.4. Tipos de bases de datos NoSQL

Hay varios tipos de bases de datos NoSQL[14], como por ejemplo:

1. Key-value: las bases de datos de valores clave son altamente particionables y permiten el escalado horizontal a escalas que otros tipos de bases de datos no pueden lograr.
2. Document: en el código de la aplicación, los datos se representan a menudo como un objeto o documento similar a JSON porque es un modelo de datos eficiente e intuitivo para los desarrolladores.
3. Graph: el propósito de una base de datos gráfica es facilitar la creación y ejecución de aplicaciones que funcionen con conjuntos de datos altamente conectados.

4. In memory: las aplicaciones de juegos y tecnología publicitaria tienen casos de uso como tablas de clasificación, almacenes de sesiones y análisis en tiempo real que requieren tiempos de respuesta de microsegundos y pueden tener grandes picos en el tráfico en cualquier momento.
5. Search: muchos registros de salida de aplicaciones para ayudar a los desarrolladores a solucionar problemas.

## 4.5. Algoritmo de Aprendizaje Automático K-means.

Antes de dar lugar a las bases de datos NoSQL que se ocuparon en este proyecto, primero hay que profundizar un poco en el algoritmo K-means.

K-means es un algoritmo de clasificación no supervisada (clusterización) que agrupa objetos en  $k$  grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster[15].

Para el desarrollo de los scripts utilicé las librerías que proporciona SKLEARN[16] (scikit-learn). El algoritmo K-means a grandes rasgos funciona de la siguiente forma:

1. Inicialización: una vez escogido el número de grupos,  $k$ , se establecen  $k$  centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente. Para este caso se utilizó  $k = 4$ .
2. Asignación objetos a los centroides: cada objeto de los datos es asignado a su centroide más cercano.
3. Actualización centroides: se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.
4. Se repiten los pasos 2 y 3 hasta que los centroides no se mueven.

## 4.6. MongoDB

El modelo de documento de MongoDB es simple para que los desarrolladores lo aprendan y lo usen, al tiempo que proporciona todas las capacidades necesarias para cumplir con los

requisitos más complejos a cualquier escala. Ofrecen controladores para más de 10 lenguajes, y la comunidad ha creado docenas más. MongoDB es una base de datos de documentos con la escalabilidad y flexibilidad en las consultas e indexaciones[17].

MongoDB almacena datos en documentos flexibles, similares a JSON, lo que significa que los campos pueden variar de un documento a otro y la estructura de datos se puede cambiar con el tiempo. Es de uso gratuito. Las versiones publicadas antes del 16 de octubre de 2018 se publican bajo la AGPL.

Algunas empresas que usan MongoDB son: Forbes, Bosch y Metlife.

#### 4.6.1. Instalación MongoDB en Ubuntu 18.04

1. Importar la clave pública utilizada por el sistema de gestión de paquetes.  
Desde la terminal de Ubuntu se escribe: « `wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc | sudo apt-key add -` »
2. Crear un archivo de lista para MongoDB, utilizaré el editor de texto VI.  
« `vi /etc/apt/sources.list.d/mongodb-org-4.2.list` », y luego escribiré: « `echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.2 multiverse"| sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list` », finalmente guardo.
3. Recargar la base de datos del paquete local.  
« `sudo apt-get update` »
4. Instalar los paquetes MongoDB.  
« `sudo apt-get install -y mongodb-org` »
5. Iniciamos el servicio de MongoDB.  
« `systemctl start mongod.service` »
6. Entramos al shell de MongoDB.  
« `cquevedo@cquevedo: /bd_avanzada$ mongo`  
MongoDB shell version v4.2.8  
connecting to: mongodb://127.0.0.1:27017/?  
compressors=disabled&gssapiServiceName=mongodb  
Implicit session: session id: UUID("59be4712-00c1-4d1c-9589-2e58ef0dc859")  
MongoDB server version: 4.2.8 »

### 4.6.2. Implementación

La implementación se hará con Python (versión 3.6). Para obtener los drivers[18] que controlarán la conexión entre la base de datos MongoDB y Python se instala el siguiente paquete en la terminal: « python3.6 -m pip install pymongo ».

Desarrollé un script llamado **mongo\_sd.py**, el cuál esta hecho para cargar el set de datos que quedó despues de ejecutar el script **pre\_process.py** llamado "datasets". El script se conecta de forma local a MongoDB, al cliente "BDAProjectz crea la base de datos "states", solo por ser un ejercicio se borra la base de datos inicial para que no haya duplicamiento en el testeo de código (o cada vez que se ejecute iba a guardar más datos y más datos, y eso no iba a servir para usarlo mas adelante en el script de Aprendizaje Automático). Se utiliza `insert_one()` para ir insertando los datos de estado uno por uno.

```
> show dbs
BDAProject      0.000GB
admin           0.000GB
config          0.000GB
local           0.000GB
test-namverwan  0.000GB
> use BDAProject
switched to db BDAProject
> db.states.find({'date':'26-05-2020'})
{ "_id" : ObjectId("5f1cb05c7c550acde6d4aab4"), "date" : "26-05-2020", "state" : "Alabama ", "population" : 4833722, "infected" : 14986, "death" : 566, "recovered" : 7951, "migration" : 0.022612181668701683, "politic" : 1, "density" : 96.9221, "temperature" : 45.0208, "employment" : 4455590, "poverty" : 16, "netspeed" : 33.7, "netcoverage" : 81.2 }
{ "_id" : ObjectId("5f1cb05c7c550acde6d4aab5"), "date" : "26-05-2020", "state" : "Alaska ", "population" : 735132, "infected" : 408, "death" : 10, "recovered" : 361, "migration" : 0.043664267097609684, "politic" : 1, "density" : 1.2863, "temperature" : -2, "employment" : 718584, "poverty" : 12.2, "netspeed" : 27.9, "netcoverage" : 78.2 }
{ "_id" : ObjectId("5f1cb05c7c550acde6d4aab6"), "date" : "26-05-2020", "state" : "Arizona ", "population" : 6626624, "infected" : 16575, "death" : 807, "recovered" : 4204, "migration" : 0.04130519552640983, "politic" : 1, "density" : 64.9549, "temperature" : 42.5208, "employment" : 6318657, "poverty" : 14.4, "netspeed" : 33.9, "netcoverage" : 86.7 }
```

Figura 3: Ejemplo de MongoDB  
Ejemplo de consulta a MongoDB en terminal de Ubuntu 18.04.

El código Python para la inserción de datos a MongoDB es el siguiente:

```
import pandas as pd
import pymongo
from pymongo import MongoClient
import pprint
client = MongoClient('localhost',27017)
db = client['BDaproject']
states = db['states']
states.drop()
datasets = pd.read_csv('datasets',sep=';')
records = []
for i in range(0, 50):
    records.append([datasets.values[i,j] for j in range(0, 13)])
for data in records:
    a = states.insert_one({'date':'26-05-2020',
        'state':data[0], 'population':data[1],
        'infected':data[2], 'death':data[3],
        'recovered':data[4], 'migration':data[5],
        'politic':data[6], 'density':data[7],
        'temperature':data[8], 'employment':data[9],
        'poverty':data[10], 'netspeed':data[11],
        'netcoverage':data[12]}).inserted_id
print("\n>>_Datos_guardados_correctamente_<<")
```

Ya teniendo los datos guardados en MongoDB, procedí a desarrollar un segundo script llamado **mongo\_kmeans.py**, en donde utilizo el algoritmo K-means.

El código del script **mongo\_kmeans.py** se conecta a MongoDB de la misma forma que en el script que inserta los datos, el cambio es que en vez de usar `insert_one()` utiliza `find()` para hacer la consulta, para este caso se usó `find('date':'26-05-2020')`, debido a que los datos de Covid-19 son de esa fecha.

Luego hay una función llamada `find_clusters()` que será la encargada de hacer el proceso descrito anteriormente para encontrar los clusters (con  $k = 4$ ). Claro, antes de enviar los datos a dicha función se debe transformar a formato Dataframe (para eso usamos la librería pandas), una vez transformados se llama la función. Posterior a la ejecución de la función se imprimen los clusters formados. El código es el siguiente:



```

import pandas as pd
import pymongo
from pymongo import MongoClient
import pprint
from sklearn.metrics import pairwise_distances_argmin
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
        labels = pairwise_distances_argmin(X, centers)
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels
client = MongoClient('localhost',27017)
db = client['BDAProject']
states = db['states']
records = []
for states in states.find({'date':'26-05-2020'}):
    records.append([states['state'],states['population'],
                    states['infected'],states['death'],
                    states['recovered'],states['migration'],
                    states['politic'],states['density'],
                    states['temperature'],states['employment'],
                    states['poverty'],states['netspeed'],states['netcoverage']])
n_clusters = 4
df = DataFrame(records,columns=['state','population','infected',
                                'death','recovered','migration','politic','density',
                                'temperature','employment','poverty','netspeed','netcoverage'])
X = np.array(df[['population','infected','death','recovered',
                 'migration','politic','density','temperature','employment',
                 'poverty','netspeed','netcoverage']])
centroids, labels = find_clusters(X, n_clusters)
cluster1 = []
cluster2 = []

```

```
cluster3 = []
cluster4 = []
for cluster , state in zip(labels , records):
    if (cluster == 0):
        cluster1.append(state[0])
    elif (cluster == 1):
        cluster2.append(state[0])
    elif (cluster == 2):
        cluster3.append(state[0])
    else:
        cluster4.append(state[0])
```

Luego de ejecutar el script, el resultado que entrega se muestra en la figura 4.

```
cquevedo@cquevedo:~/bd_avanzada$ python3.6 mongo_kmeans.py
Grupo 1
Georgia , Illinois , Michigan , New Jersey , North Carolina , Ohio , Pennsylvania , Virginia .

Grupo 2
Alabama , Arizona , Colorado , Connecticut , Indiana , Kentucky , Louisiana , Maryland , Massachusetts , Minnesota ,
Missouri , Oklahoma , Oregon , South Carolina , Tennessee , Washington , Wisconsin .

Grupo 3
Alaska , Arkansas , Delaware , Hawaii , Idaho , Iowa , Kansas , Maine , Mississippi , Montana , Nebraska , Nevada ,
New Hampshire , New Mexico , North Dakota , Rhode Island , South Dakota , Utah , Vermont , West Virginia , Wyoming .

Grupo 4
California , Florida , New York , Texas .
```

Figura 4: Resultado script mongo\_kmeans.py  
Clusterización de los estados de Estados Unidos en 4 grupos.

#### 4.6.3. Problemas

Un "problema" que tuve con MongoDB es el guardado de datos con diferenciación de fechas, en el caso que quisiera insertar nuevos datos Covid-19 de otros días. Dejo entre comillas la palabra problema por que mas que eso era una mala interpretación de la base de datos (es difícil dejar de pensar de forma relacional).

La solución fue bastante simple, le agregué una nueva columna al documento llamado fecha y le asigne la fecha correspondiente a los datos Covid-19, eso se puede ver en el script **mongo\_sd.py**, en la línea que dice `insert_one('date': '26-05-2020', ...)`.

#### 4.7. Neo4J

Los desarrolladores la definen como "Neo4j es una base de datos de grafos nativa, construida desde cero para aprovechar no solo datos sino también relaciones de datos. Neo4j conecta los datos tal como están almacenados, permitiendo consultas nunca antes imaginadas, a velocidades que nunca se creyeron posibles" [19].

A diferencia de las bases de datos tradicionales, que organizan los datos en filas, columnas y tablas, Neo4j tiene una estructura flexible definida por las relaciones almacenadas entre

los registros de datos.

Con Neo4j, cada registro de datos o nodo almacena punteros directos a todos los nodos a los que está conectado. Debido a que Neo4j está diseñado en torno a esta optimización simple pero potente, realiza consultas con conexiones complejas en órdenes de magnitud más rápidas y con mayor profundidad que otras bases de datos[20].

Las bases de datos orientadas a grafos se utilizan para extraer valor de la relación entre los datos. Neo4j es líder en este tipo de bases de datos y la utilizan empresas como eBay, Walmart, Telnor, UBS, Cisco, HP o Lufthansa para ofrecer mejor servicio a sus clientes o analizar sus datos.

#### 4.7.1. Instalación de Neo4j en Ubuntu 18.04

1. Importar la clave pública utilizada por el sistema de gestión de paquetes.  
Desde la terminal de Ubuntu se escribe: « `wget -O - https://debian.neo4j.com/neotechnology.gpg.key | apt-key add -` »
2. Crear archivo con información del repositorio, utilizaré el editor de texto VI.  
« `/etc/apt/sources.list.d/neo4j.list` », y luego escribiré: « `echo 'deb https://debian.neo4j.com stable latest' | tee -a /etc/apt/sources.list.d/neo4j.list` », finalmente guardo.
3. Recargar la base de datos del paquete local.  
« `sudo apt-get update` »
4. Instalar los paquetes Neo4j.  
« `sudo apt install neo4j` »
5. Iniciamos el servicio de Neo4j.  
« `systemctl start mongod.service` »
6. Entramos al shell de Neo4j.  
« `cquevedo@cquevedo: /bd_avanzada$ cypher-shell`  
Connected to Neo4j 4.1.0 at neo4j://localhost:7687.  
Type :help for a list of available commands or :exit to exit the shell.  
Note that Cypher queries must end with a semicolon.  
`@neo4j` »

### 4.7.2. Implementación

La implementación se hará con Python (versión 3.6). Para obtener los drivers que controlarán la conexión entre la base de datos Neo4j y Python se instala el siguiente paquete en la terminal: « python3.6 -m pip install py2neo ».

Desarrollé un script llamado **neo4j\_sd.py**, el cuál esta hecho para cargar el set de datos que quedó despues de ejecutar el script `pre_process.py` llamado "datasets". El script se conecta de forma local a Neo4j usando la función `Graph()`, de igual forma que en MongoDB solo por ser un ejercicio se borra la base de datos inicial para que no haya duplicamiento en el testeo de código, para llevar acabo eso se utilizó la sentencia "MATCH (n) DETACH DELETE n".

En primera instancia se crea el nodo "Country" con nombre "United States". Luego hay un for el cual va leyendo los datos y creando un nodo por cada línea (un nodo por cada estado llamado "State"), a medida que va creando un nodo nuevo de un estado, va creando simultáneamente la relación entre "Country" y "State". Una vez creado todos los nodos "State" con sus atributos y sus respectivas relaciones a "Country", se guarda en la base de datos Neo4j.

```
@neo4j> MATCH (state:State) RETURN state.population,state.infected,state.death,state.recovered LIMIT 5;
+-----+-----+-----+-----+
| state.population | state.infected | state.death | state.recovered |
+-----+-----+-----+-----+
| 4833722          | 14986          | 566         | 7951.0          |
| 735132           | 408            | 10          | 361.0           |
| 6626624          | 16575          | 807         | 4204.0          |
| 2959373          | 6029           | 117         | 4249.0          |
| 38332521         | 96400          | 3769        | 92631.06        |
+-----+-----+-----+-----+
5 rows available after 39 ms, consumed after another 2 ms
```

Figura 5: Ejemplo de Neo4j  
Ejemplo de consulta a Neo4j en cypher-shell.

El código Python para la inserción de datos a Neo4j es el siguiente:

```
import pandas as pd
from py2neo import Graph, Node, Relationship
datasets = pd.read_csv('datasets',sep=';')
records = []
for i in range(0, 50):
    records.append([datasets.values[i,j] for j in range(0, 13)])
```

```

g = Graph()
g.run("MATCH_(n)_DETACH_DELETE_n")
tx = g.begin()
country = Node("Country", name="United_States")
tx.create(country)
for data in records:
    state = Node("State", name=data[0], population=data[1],
    infected=data[2], death=data[3], recovered=data[4],
    migration=data[5], politic=data[6], density=data[7],
    temperature=data[8], employment=data[9], poverly=data[10],
    netspeed=data[11], netcoverage=data[12])
    country_state = Relationship(country, "HAS", state)
    tx.create(country_state)
tx.commit()

```

Ya teniendo los datos guardados en Neo4j, procedí a desarrollar un segundo script llamado `neo4j_kmeans.py`, en donde utilizo el algoritmo K-means.

Pero antes, un ejemplo de que se guardaron correctamente los datos se puede visualizar en la Figura 6.

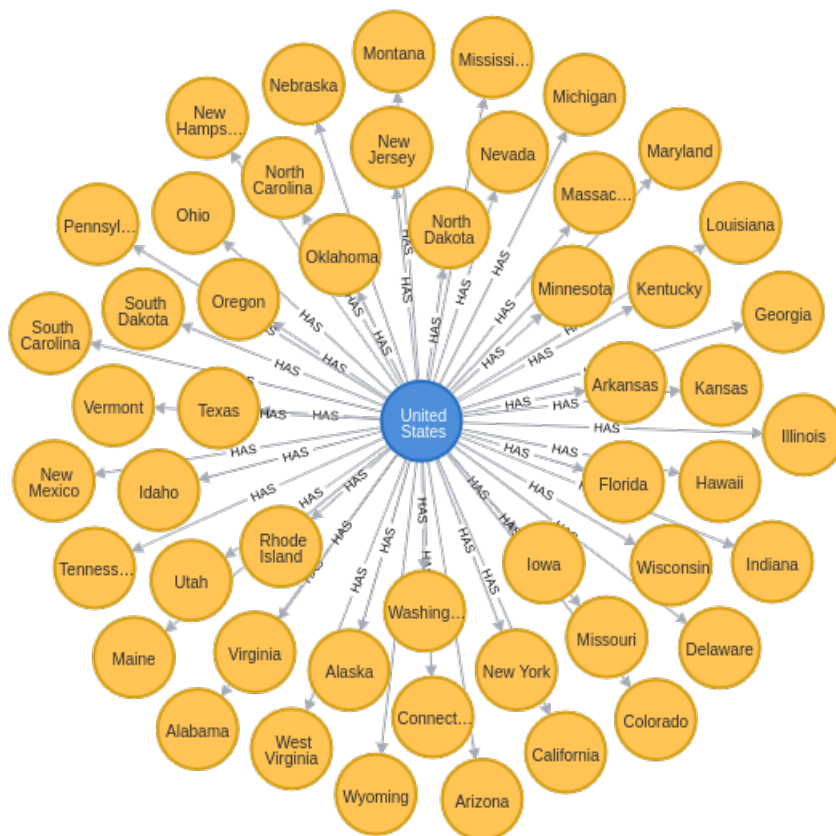


Figura 6: Resultado script `neo4j_sd.py`  
Visualización de nodos Neo4j.

El código del script **`neo4j_kmeans.py`** se conecta a Neo4j (cypher-shell) de la misma forma que en el script que inserta los datos, usando la función `Graph()`. Luego en la extracción de datos me recuerda a las bases de datos relacionales (SQL), puesto que para consultar los datos necesarios para ejecutar el algoritmo K-means debo hacerlo mediante un "Query" muy similar a un "Select \* from" de MySQL o PostgreSQL por ejemplo, por supuesto la sintaxis cambia un poco (en vez de "Select." es "Match").

Al igual que en los scripts de MongoDB hay una función llamada `find_clusters()` que

será la encargada de hacer el proceso descrito anteriormente para encontrar los clusters (con  $k = 4$ ). Posterior a la ejecución de la función se imprimen los clusters formados. El código es el siguiente:

```
import pandas as pd
from py2neo import Graph, Node, Relationship
from sklearn.metrics import pairwise_distances_argmin
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
g = Graph()
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
        labels = pairwise_distances_argmin(X, centers)
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels
X = g.run("MATCH (state:State)_RETURN state.population, "+
"state.infected, state.death, state.recovered, state.migration, "+
"state.politic, state.density, state.temperature, "+
"state.employment, state.poverly, state.netspeed, "+
"state.netcoverage_LIMIT_50").to_ndarray()
records = g.run("MATCH (state:State)_RETURN state.name").data()
n_clusters = 4
centroids, labels = find_clusters(X, n_clusters)
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []
for cluster, state in zip(labels, records):
    if (cluster == 0):
        cluster1.append(state['state.name'])
    elif (cluster == 1):
        cluster2.append(state['state.name'])
    elif (cluster == 2):
```



```

        cluster3.append(state['state.name'])
    else:
        cluster4.append(state['state.name'])

```

Uno de los aspectos interesantes es que retorna la consulta como arreglo Numpy (lo cual en este caso me ahorra varias lineas de código en las que debia transformar los datos).

Luego de ejecutar el script, el resultado que entrega se muestra en la figura 6.

```

cquevedo@cquevedo:~/bd_avanzada$ python3.6 neo4j_kmeans.py
Grupo 1
Georgia , Illinois , Michigan , New Jersey , North Carolina , Ohio , Pennsylvania , Virginia .

Grupo 2
Alabama , Arizona , Colorado , Connecticut , Indiana , Kentucky , Louisiana , Maryland , Massachusetts , Minnesota ,
Missouri , Oklahoma , Oregon , South Carolina , Tennessee , Washington , Wisconsin .

Grupo 3
Alaska , Arkansas , Delaware , Hawaii , Idaho , Iowa , Kansas , Maine , Mississippi , Montana , Nebraska , Nevada ,
New Hampshire , New Mexico , North Dakota , Rhode Island , South Dakota , Utah , Vermont , West Virginia , Wyoming .

Grupo 4
California , Florida , New York , Texas .

```

Figura 7: Resultado script neo4j\_kmeans.py  
Clusterización de los estados de Estados Unidos en 4 grupos.

### 4.7.3. Problemas

Tuve un problema con la contraseña de cypher-shell, para solucionarlo seguí un tutorial de la misma página de Neo4j para poder recuperar el acceso, al parecer es algo mas o menos común, sobretodo en la versión pública.

1. Editar archivo neo4j.conf, comentando la línea que dice:  
« dbms.connectors.default\_listen\_address=your\_configuration ».
2. Detener el servicio de neo4j: « bin/neo4j stop ».
3. Editar archivo neo4j.conf, la siguiente línea:  
« dbms.security.auth\_enabled=false ».
4. Iniciando nuevamente el servicio de neo4j: « bin/neo4j start ».

5. Ingresando a neo4j, a la base de dato system para editar la contraseña:  
« bin/cypher-shell -d system  
neo4j@system>ALTER USER neo4j SET PASSWORD 'mynewpass';  
neo4j@system>:exit »

Luego al instalar las librerías requeridas para poder hacer consultas con Python en Neo4j, las librerías iniciales no me funcionaban, específicamente las que se instalan al ejecutar el siguiente comando: « python3.6 -m pip install neo4j ».

Continué probando con otras opciones, la cual sería neomodel, que se instala con el siguiente comando: « python3.6 -m pip install neomodel ». Pero tampoco me funcionó, al ejecutar el script me decía que la librería no existía ("librería neo4j.v1 no existe"). Y claro, al ejecutar el comando « python3.6 -m pip list » no salía ninguna llamada neo4j.v1.

Finalmente opté por la última opción que me quedaba PY2NEO[21], instalé esta librería con el siguiente comando: « python3.6 -m pip install py2neo ». Al probar el script, este sí funcionaba, por lo que me quede con py2neo.

Otro problema es que necesita de Java11, por lo que hubo conflicto cuando se trabajo con la base de datos que se hablará a continuación, Apache Cassandra que necesita de Java8. Para solucionar este problema se realizó los siguientes pasos:

1. Entrar como usuario root al sistema: « sudo su ».
2. Ejecutar el siguiente comando: « update-alternatives --config java ».  
root@cquevedo:/home/cquevedo/bd\_avanzada\$ update-alternatives --config java  
There are 2 choices for the alternative java (providing /usr/bin/java).

Selection	Path	Priority	Status
0	/usr/lib/jvm/java-11-openjdk-amd64/bin/java	1111	auto mode
1	/usr/lib/jvm/adoptopenjdk-8-hotspot-amd64/bin/java	1081	manual mode
2	/usr/lib/jvm/java-11-openjdk-amd64/bin/java	1111	manual mode

Press <enter> to keep the current choice[\*], or type selection number:  
En el que se seleccionó el número 2.

3. Ejecutar comando: « sudo systemctl restart neo4j.service ».

## 4.8. Apache Cassandra

La base de datos Apache Cassandra es la elección correcta cuando se necesita escalabilidad y alta disponibilidad sin comprometer el rendimiento. La escalabilidad lineal y la probada tolerancia a fallas en hardware básico o infraestructura de nube lo convierten en la plataforma perfecta para datos de misión crítica. El soporte de Cassandra para replicar en múltiples centros de datos es el mejor en su clase, brindando una latencia más baja para sus usuarios y la tranquilidad de saber que puede sobrevivir a las interrupciones regionales[22].

Es una base de datos distribuída, esto significa que los datos se replican automáticamente en múltiples nodos para tolerancia a fallas. Se admite la replicación en múltiples centros de datos. Los nodos fallidos se pueden reemplazar sin tiempo de inactividad.

### 4.8.1. Instalación de Apache Cassandra en Ubuntu 18.04

1. Crear archivo con información del repositorio, utilizaré el editor de texto VI.  
« vi /etc/apt/sources.list.d/cassandra.sources.list », y luego escribiré:  
« deb https://downloads.apache.org/cassandra/debian 311x main », finalmente guardo.
2. Importar la clave pública utilizada por el sistema de gestión de paquetes.  
Desde la terminal de Ubuntu se escribe:  
« curl https://downloads.apache.org/cassandra/KEYS | apt-key add - »
3. Recargar la base de datos del paquete local.  
« sudo apt-get update »
4. Instalar los paquetes de Apache Cassandra.  
« sudo apt install cassandra »
5. Iniciamos el servicio de cassandra.  
« sudo systemctl start cassandra.service »
6. Entramos al shell de cassandra.  
« cquevedo@cquevedo: /bd\_avanzada\$ cqlsh  
Connected to Test Cluster at 127.0.0.1:9042.  
cqlsh 5.0.1 | Cassandra 3.11.7 | CQL spec 3.4.4 | Native protocol v4  
Use HELP for help.  
cqlsh\$ »

### 4.8.2. Implementación

La implementación se hará con Python (versión 3.6). Para obtener los drivers[23] que controlarán la conexión entre la base de datos Apache Cassandra y Python se instala el siguiente paquete en la terminal: « `python3.6 -m pip install cassandra-driver` ».

Desarrollé un script llamado **cassandra\_sd.py**, el cuál está hecho para cargar el set de datos que quedó después de ejecutar el script `pre_process.py` llamado "datasets". En primera instancia se conecta con la base de datos `united_states` de forma local a `cqlsh` usando la función `Cluster()`. Luego se utiliza con la función `execute('USE united_states')`. Después, de igual forma que en MongoDB y Neo4j solo por ser un ejercicio se borra la base de datos inicial para que no haya duplicamiento en el testeo de código, para llevar a cabo eso se utilizó la función `execute('TRUNCATE state')`, la cuál eliminó todas las líneas de la tabla "state".

Debo dejar en claro que en Apache Cassandra debo crear la base de datos primero por `cqlsh`, por lo que creé un keyspace llamado "United\_States" con el siguiente comando `cqlsh`: « `CREATE KEYSPACE United_States WITH REPLICATION = {'class': 'SimpleStrategy', 'replication_factor': 3};` ».

Y luego creé una tabla llamada "state" con el siguiente comando `cqlsh`: « `CREATE TABLE state(name text, population int, infected int, death int, recovered int, migration float, politic int, density float, temperature float, employment int, poverty float, netspeed float, netcoverage float, PRIMARY KEY(name));` ».

En la Figura 8 se puede observar una consulta simple usando `cqlsh` de Apache Cassandra, que, si se compara con las demás, esta es la única de las 3 bases de datos que entrega la consulta de forma desordenada (cuando es una consulta simple, se puede ordenar usando "order by", lo que me recuerda a las bases de datos relacionales).

```
cqlsh:united_states> select name,population,infected,death,recovered from state;
```

name	population	infected	death	recovered
Kansas	2893957	9125	207	493
Arizona	6626624	16575	807	4204
Hawaii	1404054	643	17	591
Iowa	3090416	17557	456	9377
South Carolina	4774839	10178	440	6063
Arkansas	2959373	6029	117	4249
New Jersey	8899339	155092	11147	25253
Illinois	12882135	112017	4885	107132
Texas	26448193	56409	1533	35292
New Mexico	2085287	7026	320	2464
Nevada	2790136	7956	394	359
Alabama	4833722	14986	566	7951
Minnesota	5420380	21315	890	14816
Connecticut	3596080	40873	3742	7127
Montana	1015165	479	16	441
Wyoming	582658	843	12	575
South Dakota	844877	4586	50	3415
Oregon	3930065	3949	148	1376
Colorado	5268367	24256	1333	3666
North Carolina	9848060	24057	790	14954
West Virginia	1854304	1774	72	1135
Wisconsin	5742713	15584	514	9207
Pennsylvania	12773801	71925	5146	40911
Delaware	925749	8965	332	4693
Maryland	5928814	47152	2302	3329
Louisiana	4625470	37809	2691	28700
North Dakota	723393	2457	54	1551
Alaska	735132	408	10	361
Utah	2900872	8521	98	5218
New Hampshire	1323459	4197	210	2204
Missouri	6044171	12476	689	11787
New York	19651127	362764	29229	64280
Virginia	8260405	37727	1208	5145
Ohio	11570808	32477	1987	30490
Mississippi	2991207	13458	635	9401
Idaho	1612136	2626	79	1755
Nebraska	1868516	12362	147	12215
Florida	19552860	51746	2252	49494
Tennessee	6495978	20535	338	13073
Indiana	6570902	31715	1984	29731
Washington	6971406	20065	1070	18995
Maine	1328302	2074	78	1290
Georgia	9992167	43400	1848	41552
Oklahoma	3850568	6090	313	4714
Massachusetts	6692824	93271	6416	86855
Rhode Island	1051511	14065	608	1119
California	38332521	96400	3769	92631
Vermont	626630	962	54	843
Kentucky	4395295	8571	391	3102
Michigan	9895622	54881	5241	33168

Figura 8: Ejemplo de Apache Cassandra  
Ejemplo de consulta a Apache Cassandra a través de cqlsh.

El código Python para la inserción de datos a Apache Cassandra es el siguiente:

```
import pandas as pd
from cassandra.cluster import Cluster
cluster = Cluster()
session = cluster.connect('united_states')
session.execute('USE_united_states')
session.execute('TRUNCATE_state')
```

```

datasets = pd.read_csv('datasets', sep=';')
records = []
for i in range(0, 50):
    records.append([datasets.values[i, j] for j in range(0, 13)])
for data in records:
    session.execute("INSERT INTO state(name, population, infected, "+
"death, recovered, migration, politic, density, temperature, employment, "+
"poverty, netspeed, netcoverage) VALUES(%(name)s, %(population)s, "+
"%(infected)s, %(death)s, %(recovered)s, %(migration)s, %(politic)s, "+
"%(density)s, %(temperature)s, %(employment)s, %(poverty)s, "+
"%(netspeed)s, %(netcoverage)s)",
    {'name': data[0], 'population': data[1], 'infected': data[2],
'death': data[3], 'recovered': int(data[4]), 'migration': data[5],
'politic': data[6], 'density': data[7], 'temperature': data[8],
'employment': data[9], 'poverty': data[10], 'netspeed': data[11],
'netcoverage': data[12]})

```

A simple vista se puede ver que es un poco mas compleja la forma en que inserta los datos, respecto a MongoDB y a Neo4j.

Ya teniendo los datos guardados en Cassandra, procedí a desarrollar un segundo script llamado **cassandra\_kmeans.py**, en donde utilizo el algoritmo K-means.

El código del script **cassandra\_kmeans.py** se conecta Apache Cassandra (cqlsh) de la misma forma que en el script que inserta los datos, usando la función **Cluster()**. Luego en la extracción de datos, Apache Cassandra también me recuerda a las bases de datos relacionales (SQL), puesto que para consultar los datos necesarios para ejecutar el algoritmo K-means debo hacerlo mediante un "Query" muy similar a un "Select \* from" de MySQL o PostgreSQL por ejemplo, de hecho la sintaxis no cambia, al menos en este ensayo.

Al igual que en los scripts de MongoDB y Neo4j hay una función llamada **find\_clusters()** que será la encargada de hacer el proceso descrito anteriormente para encontrar los clusters (con  $k = 4$ ). Posterior a la ejecución de la función se imprimen los clusters formados. El código es el siguiente:

```

import pandas as pd
from cassandra.cluster import Cluster
from sklearn.metrics import pairwise_distances_argmin
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]
    while True:
        labels = pairwise_distances_argmin(X, centers)
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers, labels
cluster = Cluster()
session = cluster.connect('united_states')
session.execute('USE_united_states')
rows = session.execute('select_name, population, infected, "+
"death, recovered, migration, politic, density, temperature, "+
"employment, poverty, netspeed, netcoverage_from_state')
records = []
for data in rows:
    records.append([data[0], data[1], data[2],
                    data[3], data[4], data[5], data[6], data[7],
                    data[8], data[9], data[10], data[11], data[12]])
n_clusters = 4
df = DataFrame(records, columns=['state', 'population', 'infected',
'death', 'recovered', 'migration', 'politic', 'density',
'temperature', 'employment', 'poverty', 'netspeed', 'netcoverage'])
X = np.array(df[['population', 'infected', 'death', 'recovered',
'migration', 'politic', 'density', 'temperature', 'employment',
'poverty', 'netspeed', 'netcoverage']])
centroids, labels = find_clusters(X, n_clusters)
cluster1 = []
cluster2 = []
cluster3 = []
cluster4 = []

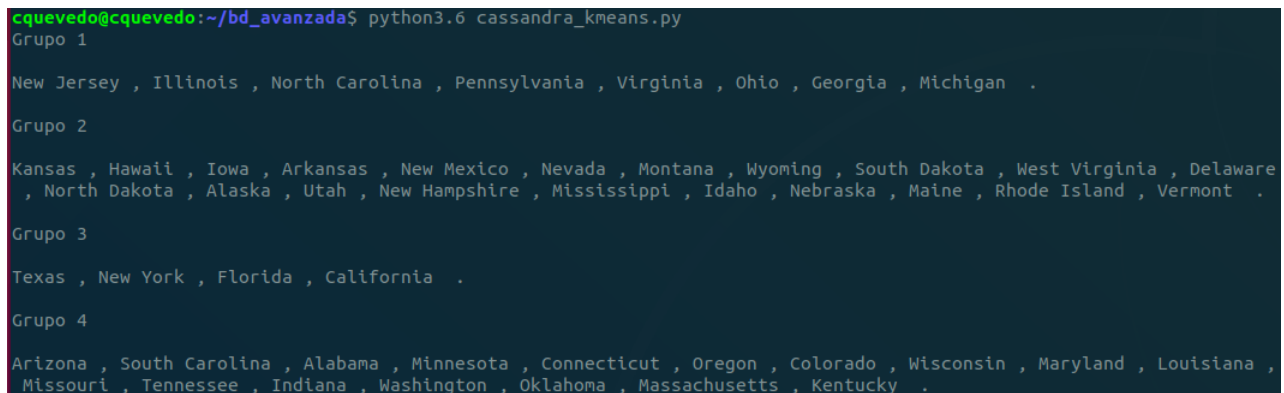
```

```

for cluster , state in zip(labels , records ):
    if (cluster == 0):
        cluster1.append( state [0])
    elif (cluster == 1):
        cluster2.append( state [0])
    elif (cluster == 2):
        cluster3.append( state [0])
    else :
        cluster4.append( state [0])

```

En la Figura 9 se puede observar el resultado que entrega el script **cassandra\_kmeans.py**, al hacer la consulta sin utilizar “order by“ los grupos resultantes estan desordenados (en cuanto al número del grupo, no a los integrantes de cada grupo, por que esos no deberían cambiar ya que se está trabajando con la mismo set de datos).



```

cquevedo@cquevedo:~/bd_avanzada$ python3.6 cassandra_kmeans.py
Grupo 1
New Jersey , Illinois , North Carolina , Pennsylvania , Virginia , Ohio , Georgia , Michigan .

Grupo 2
Kansas , Hawaii , Iowa , Arkansas , New Mexico , Nevada , Montana , Wyoming , South Dakota , West Virginia , Delaware , North Dakota , Alaska , Utah , New Hampshire , Mississippi , Idaho , Nebraska , Maine , Rhode Island , Vermont .

Grupo 3
Texas , New York , Florida , California .

Grupo 4
Arizona , South Carolina , Alabama , Minnesota , Connecticut , Oregon , Colorado , Wisconsin , Maryland , Louisiana , Missouri , Tennessee , Indiana , Washington , Oklahoma , Massachusetts , Kentucky .

```

Figura 9: Resultado script cassandra\_kmeans.py  
Clusterización de los estados de Estados Unidos en 4 grupos.

### 4.8.3. Problemas

Un problema es que necesita de Java8 (se había comentado antes), por lo que hubo conflicto cuando se trabajo con la base de datos anterior, Neo4j que necesita de Java11. Para solucionar este problema se realizó los siguientes pasos:

1. Entrar como usuario root al sistema: « sudo su ».
2. Ejecutar el siguiente comando: « update-alternatives –config java ».  
root@cquevedo:/home/cquevedo/bd\_avanzada\$ update-alternatives –config java



There are 2 choices for the alternative java (providing /usr/bin/java).

Selection Path Priority Status

---

```
0 /usr/lib/jvm/java-11-openjdk-amd64/bin/java 1111 auto mode
1 /usr/lib/jvm/adoptopenjdk-8-hotspot-amd64/bin/java 1081 manual mode
2 /usr/lib/jvm/java-11-openjdk-amd64/bin/java 1111 manual mode
```

Press <enter> to keep the current choice[\*], or type selection number:

En el que se seleccionó el número 1.

3. Ejecutar comando: « sudo systemctl restart cassandra.service ».

Como bien nombré anteriormente, Apache Cassandra es un base de datos distribuída, esto quiere decir que se aprovecha mas con más nodos conectados haciendo alguna tarea, en este caso solo se utilizó el servicio local, sin ningún otro nodo, por lo que no podría decir que lo aprendí de forma correcta o no, de momento me quedo con que funciona.

Otro "problema.<sup>es</sup> que se debe crear la base de datos o keyspace y las tablas antes de empezar a utilizarlas, en eso pierde versus MongoDB y Neo4j. Aunque no es tan problemático, pero si tengo que trabajar así, entonces por conveniencia propia me quedaría con una base de datos relacional como PostgreSQL.

## 5. Conclusiones

Se utilizó ETL para hacer el pre-procesamiento de datos, esto se hizo con la utilización de herramientas bash como SORT, AWK y TAIL. También se desarrolló un script llamado **pre\_process.py** con el que transformé algunos datos (como los de migración) y uní la información de las distintas bases de datos investigadas. Finalmente en este punto, VI junto a expresiones regulares, combinados son una herramienta poderosa.

Se logró poner en marcha las 3 bases de datos no relacionales (MongoDB, Neo4j y Apache Cassandra), junto a esto se logró utilizar estas bases de datos junto a Python sin muchos problemas, excepto por Neo4j. Por temas de tiempo, paro estudiantil (online) y pandemia mundial de Covid-19 (que está en la problemática de este estudio) que no se hizo ninguna comparación tangible entre las 3 bases de datos. De modo personal encuentro que las bases de datos no relacionales (NoSQL) son una herramienta bastante robusta en cuanto a la flexibilidad que estas poseen, por ejemplo un error que yo tuve es que se me olvidó integrar el

porcentaje de ruralidad en el script `pre_process.py`, pero basto agregar algunas líneas más a los scripts `*_kmeans.py` y `*sd.py` para agregar una nueva columna, con excepción de Apache Cassandra, que tuve que modificarla de forma muy similar a una base de datos relacional, esta última fue la que menos me gustó (pero debe ser por que realmente no le pude sacar todo su potencial). Si tuviese que elegir una para continuar trabajando sería MongoDB, más que nada por que su curva de aprendizaje es bastante más exponencial que Neo4j, pero no descartaría por ningún motivo Neo4j.

No pude hacer una comparación en cuanto al tiempo que tardaban en insertar o en extraer los datos, básicamente por dos motivos. El primer motivo es por que mi set de datos era en realidad bastante pequeño (solo 14 columnas y 50 filas) y el segundo motivo es por que se tuvo que sacar en los requisitos debido al paro estudiantil junto a la pandemia mundial.

En cuanto a qué se puede concluir acerca de los resultados[13] que entregó la clusterización que se hizo mediante el algoritmo no supervisado K-means, es lo siguiente:

- Grupo de estados 1 (Georgia, Illinois, Michigan, New Jersey, North Carolina, Ohio, Pennsylvania y Virginia).
  1. Poseen el segundo lugar en cuanto a número de habitantes.
  2. La mayoría de gobernadores son demócratas.
  3. Poseen el segundo lugar en cantidad de infectados, fallecidos y recuperados.
  4. Tiene el segundo lugar con menor ruralidad en sus estados.
- Grupo de estados 2 (Alabama, Arizona, Colorado, Connecticut, Indiana, Kentucky, Louisiana, Maryland, Massachusetts, Minnesota, Missouri, Oklahoma, Oregon, South Carolina, Tennessee, Washington y Wisconsin).
  1. Mitad de gobernadores son demócratas y mitad son republicanos.
  2. Poseen las densidades de población mas altas.
  3. Poseen el tercer lugar en cantidad de infectados, fallecidos y recuperados.
  4. Se encuentran los estados con mayor porcentaje de pobreza.
- Grupo de estados 3 (Alaska, Arkansas, Delaware, Hawaii, Idaho, Iowa, Kansas, Maine, Mississippi, Montana, Nebraska, Nevada, New Hampshire, New Mexico, North Dakota, Rhode Island, South Dakota, Utah, Vermont, West Virginia y Wyoming).
  1. La mayoría de gobernadores son republicanos.
  2. Poseen las temperaturas promedio anual más bajas.

3. Tienen la menor cantidad de infectados, fallecidos y recuperados.
  4. Poseen un alto nivel de ruralidad en sus estados.
- Grupo de estados 4 (California, Florida, New York y Texas).
    1. Mitad de gobernadores son demócratas y mitad son republicanos.
    2. Poseen la mayor cantidad de habitantes en cada estado.
    3. Tienen la mayor cantidad de infectados, fallecidos y recuperados.
    4. Poseen un alto nivel de empleabilidad.
    5. Tienen temperaturas altas.
    6. Poseen un bajo nivel de ruralidad en sus estados.

Detalle importante, no olvidar iniciar el servicio antes de intentar conectarse a la base de datos o de ejecutar un script.

## Referencias

- [1] Organización Mundial de la Salud  
<https://www.who.int/es/health-topics/coronavirus>
- [2] Universidad Johns Hopkins (CSSE)  
[https://github.com/CSSEGISandData/COVID-19/blob/master/csse\\_covid\\_19\\_data/csse\\_covid\\_19\\_daily\\_reports\\_us/05-18-2020.csv](https://github.com/CSSEGISandData/COVID-19/blob/master/csse_covid_19_data/csse_covid_19_daily_reports_us/05-18-2020.csv)
- [3] Civil Services USA  
<https://github.com/CivilServiceUSA/us-states/blob/master/data/states.csv>
- [4] WorldPopulationReview  
<https://worldpopulationreview.com/state-rankings/state-densities>
- [5] Census  
<https://www.census.gov/data/tables/2017/econ/susb/2017-susb-annual.html>
- [6] Rural Research Brief  
[http://www.ruralhome.org/storage/research\\_notes/Rural\\_Research\\_Note\\_Rurality\\_web.pdf](http://www.ruralhome.org/storage/research_notes/Rural_Research_Note_Rurality_web.pdf)
- [7] Policy tracking and Analysis  
<https://web.archive.org/web/20100407124510/http://statescape.com/resources/Governors/Governors.asp>
- [8] BroadBandNow  
<https://broadbandnow.com/report/us-states-internet-coverage-speed-2018/>
- [9] Machine Learning  
<https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart>
- [10] Algoritmo K-means  
[https://www.unioviedo.es/compnum/laboratorios\\_py/kmeans/kmeans.html](https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html)
- [11] Bases de datos NOSQL  
<https://www.mongodb.com/nosql-explained>
- [12] Covid-19  
<https://www.europeanlung.org/en/covid-19/what-is-covid-19/>
- [13] Scripts y documentos  
<https://github.com/CloudIO-bioinformatic/bda>
- [14] What is NoSQL?  
<https://aws.amazon.com/nosql/>

- [15] Algoritmo K-means  
[https://www.unioviedo.es/compnum/laboratorios\\_py/kmeans/kmeans.html](https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html)
- [16] Sklearn K-means  
<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>
- [17] MongoDB  
<https://www.mongodb.com/es>
- [18] PyMongo  
<https://api.mongodb.com/python/current/tutorial.html>
- [19] Neo4j Oficial  
<https://neo4j.com>
- [20] Neo4j No Oficial  
[https://www.albertcoronado.com/2016/04/22/desarrollo-de-aplicaciones-con-la-base-de-#:~:text=Neo4j%20es%20líder%20en%20este,analizar%20sus%20datos\(BI\).](https://www.albertcoronado.com/2016/04/22/desarrollo-de-aplicaciones-con-la-base-de-#:~:text=Neo4j%20es%20líder%20en%20este,analizar%20sus%20datos(BI).)
- [21] Py2Neo  
<https://py2neo.org/v5/database.html>
- [22] Apache Cassandra  
<https://cassandra.apache.org>
- [23] DataStax Python Driver  
[https://docs.datastax.com/en/developer/python-driver/3.24/getting\\_started/](https://docs.datastax.com/en/developer/python-driver/3.24/getting_started/)
- [24] CQL API y CQLSH  
<https://code.tutsplus.com/es/articles/getting-started-with-cassandra-using-cql-api-a>