PRIMERA PRACTICA ACCESO A DATOS

Alejandro López Abad | Rubén García-Redondo Marín

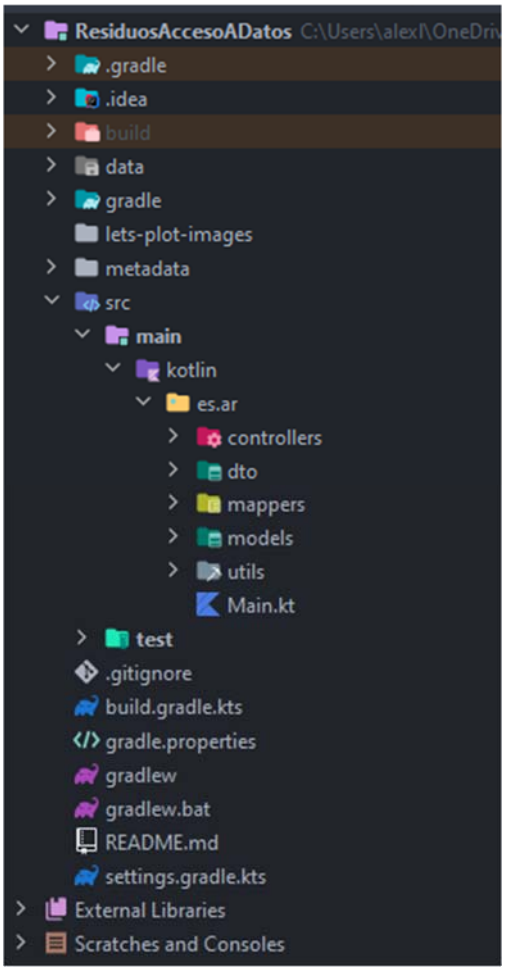
Contenido

Introducción

Primera practica de Acceso a Datos sobre el reciclaje y la limpieza de Madrid, en esta práctica trabajaremos con dos ficheros CSV los cuales tendremos que procesar y trabajar con esa información

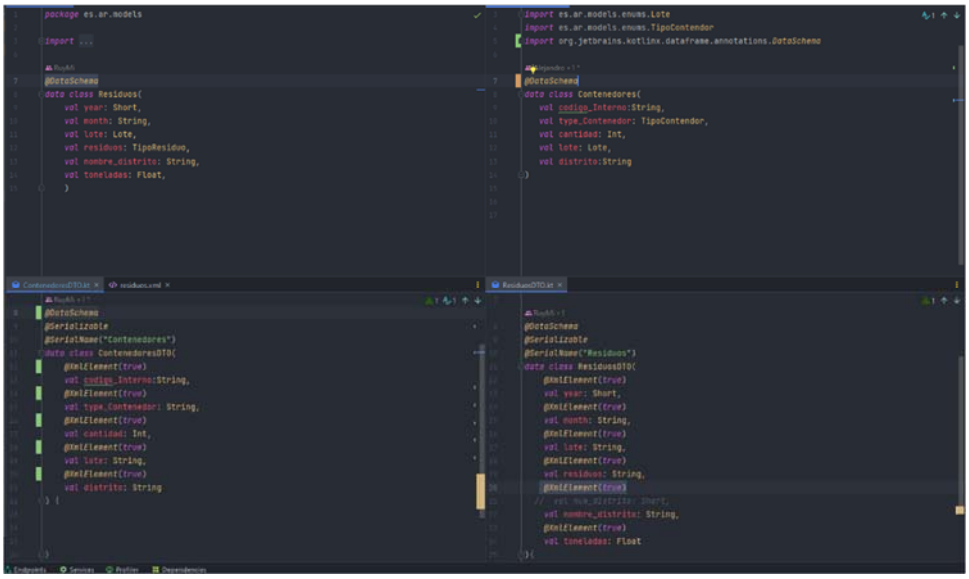
Estructura del proyecto y sus clases

El proyecto está distribuido en paquete varios paquetes, donde guardaremos las clases



Para este proyecto necesitaremos los modelos, los cuales son Residuos y Contenedores, que serán las listas de objetos que guardaremos una vez hayamos leído los dos CSV.

Para esto también necesitaremos sus respectivos DTO con los que haremos la transformación de la información para poder operar con ella de una forma más sencilla.



Aquí se puede observar las 4 clases mencionadas y cuáles son sus variables, como se puede ver algunas variables son ENUM por lo que para poder transformar esa información del tipo Enum que hemos creado a un String, para llevar esto acabo creamos una clase llamada ParseTipo la cual se encargar como su nombre indica de parsear el tipo del enum a un String:

El primer método que creamos será `stringLoteToTypeLote()` el cual recibe un String del CSV y lo cambia por el valor asignado en el ENUM

```
package es.ar.models.enums

Alejandro
enum class Lote(val num_lote:String) {
    UNO( num_lote: "1"),
    DOS( num_lote: "2"),
    TRES( num_lote: "3")
}
```

```kotlin
/**
 * Convierte una cadena en un tipo de Lote.
 *
 * @param campo -> Campo a transformar
 * @return El tipo de Lote
 */
RuyMi
fun stringLoteToTypeLote(campo: String): Lote {
    var type: Lote = Lote.UNO
    when (campo) {
        "1" -> type = Lote.UNO
        "2" -> type = Lote.DOS
        "3" -> type = Lote.TRES
    }
    return type
}
```

El segundo método sería igual que el primero, pero con Residuos, `stringResiduoToTypeResiduo()` este método leería un String del campo que le digamos y lo parsearia al tipo correspondiente de residuo, algo a destacar es el dato CAMA_CABALLO, ya que no aparece en los pdf proporcionados por la comunidad de Madrid donde explican los datos que vienen en los CSV

```kotlin
/**
 * Hemos decidido crear este enum para poder clasificar todos los tipos de residuos que hay,
 * ya que para la posterior filtración de datos no será más facil de esta manera
 */
Alejandro
enum class TipoResiduo(tipo:String) {
    RESTO( tipo: "RESTO"),
    ENVASES( tipo: "ENVASES"),
    VIDRIO( tipo: "VIDRIO"),
    ORGANICA( tipo: "ORGANICA"),
    PAPEL_CARTON( tipo: "PAPEL Y CARTON"),
    PUNTOS_LIMPIOS( tipo: "PUNTOS LIMPIOS"),
    CARTON_COMERCIAL( tipo: "CARTON COMERCIAL"),
    VIDRIO_COMERCIAL( tipo: "VIDRIO COMERCIAL"),
    PILAS( tipo: "PILAS"),
    ANIMALES_MUERTOS( tipo: "ANIMALES MUERTOS"),
    RCD( tipo: "RCD"),
    CONTENEDORES_ROPA_USADA( tipo: "CONTENEDORES DE ROPA USADA"),
    //Hemos encontrado en el CSV otro tipo de residuo que es CAMA DE CABALLO
    CAMA_CABALLO( tipo: "CAMA DE CABALLO")

}
```

```kotlin
/**
 * Convierte una cadena en un tipo de residuo.
 *
 * @param campo -> Campo a transformar
 * @return El tipo de Residuo
 */
Alejandro
fun stringResiduoToTypeResiduo(campo: String): TipoResiduo {
    var type: TipoResiduo = TipoResiduo.ENVASES
    when (campo) {
        "RESTO" -> type = TipoResiduo.RESTO
        "ENVASES" -> type = TipoResiduo.ENVASES
        "VIDRIO" -> type = TipoResiduo.VIDRIO
        "ORGANICA" -> type = TipoResiduo.ORGANICA
        "PAPEL Y CARTON" -> type = TipoResiduo.PAPEL_CARTON
        "PUNTOS LIMPIOS" -> type = TipoResiduo.PUNTOS_LIMPIOS
        "CARTON COMERCIAL" -> type = TipoResiduo.CARTON_COMERCIAL
        "VIDRIO COMERCIAL" -> type = TipoResiduo.VIDRIO_COMERCIAL
        "PILAS" -> type = TipoResiduo.PILAS
        "ANIMALES MUERTOS" -> type = TipoResiduo.ANIMALES_MUERTOS
        "RCD" -> type = TipoResiduo.RCD
        "CONTENEDORES DE ROPA USADA" -> type = TipoResiduo.CONTENEDORES_ROPA_USADA
        "CAMA DE CABALLO" -> type = TipoResiduo.CAMA_CABALLO
    }
    return type
}
```

El tercer método es igual a los dos anteriores y se encarga de leer el String del CSV y pasarlo al Tipo de Contenedor

```kotlin
Alejandro
enum class TipoContendor(val tipo:String) {
    ORGANICA( tipo: "ORGANICA"),
    RESTO( tipo: "RESTO"),
    ENVASES( tipo: "ENVASES"),
    VIDRIO( tipo: "VIDRIO"),
    PAPEL_CARTON( tipo: "PAPEL-CARTON")
}
```

```kotlin
/**
 * Convierte una cadena en un tipo de contenedor.
 *
 * @param campo -> Campo a transformar
 * @return El tipo de contenedor
 */
Alejandro
fun stringContenedorToTypeContenedor(campo: String): TipoContendor {
    var type: TipoContendor = TipoContendor.ENVASES
    when (campo) {
        "ORGANICA" -> type = TipoContendor.ORGANICA
        "RESTO" -> type = TipoContendor.RESTO
        "ENVASES" -> type = TipoContendor.ENVASES
        "VIDRIO" -> type = TipoContendor.VIDRIO
        "PAPEL_CARTON" -> type = TipoContendor.PAPEL_CARTON
    }
    return type
}
```

Tenemos otra clase que se llama ParseFloat que lo que hace es que convierte una cadena(String) a un tipo Float

```kotlin
/**
 * Convierte una cantidad de tipo String en Float
 *
 * @param valor -> Valor a cambiar
 * @return La cantidad transformada
 */
RuyMi
fun stringToFloat(valor: String): Float{
        val res = valor.replace( oldValue: ",",  newValue: ".")
        return res.toFloat()
}
```

Realización de las consultas

Para las consultas utilizamos DataFrames ya que nos daba facilidades a la hora del filtrado y encontrar los máximos, mínimos, medias... Para algunas consultas nos resultó complicado entender que pedía, aunque su realización era más fácil de lo que se planteaba. A continuación, pondremos las consultas pedidas para el resumen general.

1. Número de contenedores de cada tipo que hay en cada distrito

En esta consulta hemos entendido que había que encontrar la cantidad de contenedores y agruparlos por tipo y después agruparlos por distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun numContenedoresByTipoByDistrito(listaContenedores: DataFrame<Contenedores>): String {
    return listaContenedores
            .groupBy( ...cols: "distrito", "type_Contenedor")
            .count().sortBy( ...cols: "distrito").html()
}
```

distrito	type_Contenedor	count
ARGANZUELA	ENVASES	336
ARGANZUELA	VIDRIO	332
ARGANZUELA	RESTO	1
BARAJAS	ENVASES	313
BARAJAS	ORGANICA	129
BARAJAS	RESTO	196
BARAJAS	VIDRIO	161
CARABANCHEL	ENVASES	1815
CARABANCHEL	RESTO	1863
CARABANCHEL	VIDRIO	548
CARABANCHEL	ORGANICA	1095
CENTRO	ENVASES	329
CENTRO	VIDRIO	161
CHAMARTIN	ENVASES	478
CHAMARTIN	ORGANICA	91
CHAMARTIN	RESTO	135
CHAMARTIN	VIDRIO	341
CHAMBERI	ENVASES	272
CHAMBERI	VIDRIO	265
CIUDAD LINEAL	ENVASES	882

... showing only top 20 of 77 rows
DataFrame [77 x 3]

2. Media de contenedores de cada tipo que hay en cada distrito

En esta consulta hemos entendido que había que sacar la media de los contenedores que recogían por cada tipo agrupado por distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun mediaContenedoresByTipoByDistrito(listaContenedores: DataFrame<Contenedores>): String {
    return listaContenedores
            .groupBy( ...cols: "distrito", "type_Contenedor")
            .mean().sortBy( ...cols: "distrito").html()
}
```

distrito	type_Contenedor	cantidad
ARGANZUELA	ENVASES	1,0
ARGANZUELA	VIDRIO	1,0
ARGANZUELA	RESTO	1,0
BARAJAS	ENVASES	1,0
BARAJAS	ORGANICA	1,0
BARAJAS	RESTO	1,0
BARAJAS	VIDRIO	1,0
CARABANCHEL	ENVASES	1,0
CARABANCHEL	RESTO	1,0
CARABANCHEL	VIDRIO	1,0
CARABANCHEL	ORGANICA	1,0
CENTRO	ENVASES	1,0
CENTRO	VIDRIO	1,0
CHAMARTIN	ENVASES	1,0
CHAMARTIN	ORGANICA	1,0
CHAMARTIN	RESTO	1,0
CHAMARTIN	VIDRIO	1,0
CHAMBERI	ENVASES	1,0
CHAMBERI	VIDRIO	1,0
CIUDAD LINEAL	ENVASES	1,0

... showing only top 20 of 77 rows
DataFrame [77 x 3]

3. Media de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito.

Esta siguiente consulta hemos obtenido la media de la cantidad de toneladas en el año agrupadas por cada tipo de basura y agrupadas por cada distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun mediaToneladasByTipoByDistrito(listaResiduos: DataFrame<Residuos>): String {
    return listaResiduos
            .groupBy( ...cols: "nombre_distrito", "residuos", "year") GroupBy<Residuos, Residuos>
            .aggregate { mean( ...columns: "toneladas") into "Media_Toneladas" } DataFrame<Residuos>
            .sortBy( ...cols: "nombre_distrito").html()
}
```

nombre_distrito	residuos	year	Media_Toneladas
Arganzuela	RESTO	2021	1894,020844
Arganzuela	ENVASES	2021	313,039806
Arganzuela	VIDRIO	2021	275,680002
Arganzuela	ORGANICA	2021	617,755000
Arganzuela	CARTON_COMERCIAL	2021	25,528334
Arganzuela	RCD	2021	53,619092
Arganzuela	PUNTOS_LIMPIOS	2021	72,271667
Barajas	RESTO	2021	818,626668
Barajas	ENVASES	2021	102,462500
Barajas	VIDRIO	2021	107,181669
Barajas	ORGANICA	2021	346,575002
Barajas	RCD	2021	24,888750
Barajas	PUNTOS_LIMPIOS	2021	34,290000
Barajas	CARTON_COMERCIAL	2021	5,610000
Carabanchel	RESTO	2021	4396,960815
Carabanchel	ENVASES	2021	510,834587
Carabanchel	VIDRIO	2021	281,058333
Carabanchel	ORGANICA	2021	1118,248332
Carabanchel	CARTON_COMERCIAL	2021	28,246667
Carabanchel	RCD	2021	54,161818

... showing only top 20 of 148 rows
DataFrame [148 x 4]

4. Máximo, mínimo, media y desviación de toneladas anuales de recogidas por cada tipo de basura agrupadas por distrito.

En esta consulta hemos obtenido el máximo, mínimo, la media y desviación de los datos de toneladas agrupadas por distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun estadisticasByTipoByDistrito(listaResiduos: DataFrame<Residuos>): String {
    return listaResiduos
            .groupBy( ...cols: "nombre_distrito", "residuos", "year")
            .aggregate { this: AggregateGroupedDsl<Residuos>  it: AggregateGroupedDsl<Residuos>
                min( ...columns: "toneladas") into "Mínimo_Toneladas"
                max( ...columns: "toneladas") into "Máximo_Toneladas"
                mean( ...columns: "toneladas") into "Media_Toneladas"
                std( ...columns: "toneladas") into "Desviación_Toneladas" ^aggregate
            }.sortBy( ...cols: "nombre_distrito").html()
}
```

nombre_distrito	residuos	year	Mínimo_Toneladas	Máximo_Toneladas	Media_Toneladas	Desviación_Toneladas
Arganzuela	RESTO	2021	1568,8	2193,0	1894,020844	147,384054
Arganzuela	ENVASES	2021	0,2	385,8	313,039806	99,780040
Arganzuela	VIDRIO	2021	202,5	352,6	275,680002	45,151989
Arganzuela	ORGANICA	2021	402,5	693,6	617,755000	87,791320
Arganzuela	CARTON_COMERCIAL	2021	20,3	34,0	25,528334	3,829763
Arganzuela	RCD	2021	30,6	142,4	53,619092	33,159174
Arganzuela	PUNTOS_LIMPIOS	2021	59,5	88,0	72,271667	10,631816
Barajas	RESTO	2021	658,3	958,9	818,626668	71,678440
Barajas	ENVASES	2021	31,6	193,8	102,462500	54,484873
Barajas	VIDRIO	2021	59,1	152,2	107,181669	31,119079
Barajas	ORGANICA	2021	144,5	447,9	346,575002	97,545376
Barajas	RCD	2021	8,8	88,4	24,888750	23,262916
Barajas	PUNTOS_LIMPIOS	2021	19,8	43,5	34,290000	6,313676
Barajas	CARTON_COMERCIAL	2021	1,2	18,5	5,610000	5,128978
Carabanchel	RESTO	2021	3449,9	5138,9	4396,960815	386,480146
Carabanchel	ENVASES	2021	228,1	819,2	510,834587	235,494824
Carabanchel	VIDRIO	2021	218,8	331,5	281,058333	35,296405
Carabanchel	ORGANICA	2021	724,4	1242,7	1118,248332	145,544228
Carabanchel	CARTON_COMERCIAL	2021	23,8	33,6	28,246667	2,946163
Carabanchel	RCD	2021	25,0	71,4	54,161818	13,525255

... showing only top 20 of 148 rows
DataFrame [148 x 7]

5. Suma de todo lo recogido por distrito.

En esta siguiente consulta hemos obtenido la suma de los datos de toneladas agrupadas por distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun sumaByDistrito(listaResiduos: DataFrame<Residuos>): String {
    return listaResiduos
            .groupBy( ...cols: "nombre_distrito") GroupBy<Residuos, Residuos>
            .aggregate { sum( ...columns: "toneladas") into "Toneladas_Totales" } DataFrame<Residuos>
            .sortBy( ...cols: "nombre_distrito").html()
}
```

nombre_distrito	Toneladas_Totales
Arganzuela	43351,922
Barajas	18499,561
Carabanchel	83218,828
Centro	74160,461
Chamartín	50307,875
Chamberí	47037,707
Ciudad Lineal	68199,734
Fuencarral - El Pardo	86267,578
Hortaleza	60016,137
Latina	72393,133
Moncloa - Aravaca	59491,098
Moratalaz	28804,938
Puente de Vallecas	76623,203
Retiro	40118,055
Salamanca	53657,559
San Blas - Canillejas	51812,082
Sin Distrito	153,810
Tetuán	53028,875
Usera	42565,445
Vicálvaro	29657,307

... showing only top 20 of 22 rows
DataFrame [22 x 2]

6. Por cada distrito obtener para cada tipo de residuo la cantidad recogida.

En esta consulta hemos obtenido la cantidad recogida agrupada por cada tipo de residuo y agrupado por distrito. A continuación, pondremos el método que realiza la consulta y su resultado.

```
private fun numContenedoresByTipoDistrito(listaContenedores: DataFrame<Contenedores>, distrito: String): String {
    var cambioDistrito : String = distrito.replace( oldValue: "í", newValue: "i").uppercase(Locale.getDefault())
    return listaContenedores
        .filter { it["distrito"] == cambioDistrito } DataFrame<Contenedores>
        .groupBy( ...cols: "distrito", "type_Contenedor") GroupBy<Contenedores, Contenedores>
        .count().sortBy( ...cols: "distrito").html()
}
```

nombre_distrito	residuos	Total_Toneladas
Arganzuela	RESTO	22728,250000
Arganzuela	ENVASES	8139,034668
Arganzuela	VIDRIO	3308,160156
Arganzuela	ORGANICA	7413,060059
Arganzuela	CARTON_COMERCIAL	306,339996
Arganzuela	RCD	589,809998
Arganzuela	PUNTOS_LIMPIOS	867,260010
Barajas	RESTO	9823,520508
Barajas	ENVASES	2459,099854
Barajas	VIDRIO	1286,180054
Barajas	ORGANICA	4158,900391
Barajas	RCD	298,665009
Barajas	PUNTOS_LIMPIOS	411,480011
Barajas	CARTON_COMERCIAL	61,709999
Carabanchel	RESTO	52763,527344
Carabanchel	ENVASES	12260,029297
Carabanchel	VIDRIO	3372,700195
Carabanchel	ORGANICA	13418,979492
Carabanchel	CARTON_COMERCIAL	338,959991
Carabanchel	RCD	595,779907

... showing only top 20 of 148 rows
DataFrame [148 x 3]

Ahora pondremos las consultas realizadas por un distrito concreto. Antes de realizarse estas consultas, se filtra que el distrito metido no es inventado.

1. Media de contenedores de cada tipo que hay en cada distrito

De esta consulta hemos obtenido la media de la cantidad de contenedores por cada tipo que hay en dicho distrito. A continuación, pondremos el método que realiza la consulta y su resultado para el distrito “Centro”.

```
private fun numContenedoresByTipoDistrito(listaContenedores: DataFrame<Contenedores>, distrito: String): String {
    return listaContenedores
        .filter { it["distrito"] == distrito.replace( oldValue: "í",  newValue: "i").uppercase(Locale.getDefault()) } Da
        .groupBy( ...cols: "distrito", "type_Contenedor") GroupBy<Contenedores, Contenedores>
        .count().sortBy( ...cols: "distrito").html()

}
```

distrito	type_Contenedor	count
CENTRO	ENVASES	329
CENTRO	VIDRIO	161

DataFrame [2 x 3]

2. Total de toneladas recogidas en ese distrito por residuo

Esta consulta la hemos realizado obteniendo el total de las toneladas recogidas en el distrito metido agrupado por residuo. A continuación, pondremos el método que realiza la consulta y su resultado para el distrito “Centro”.

```
private fun totalToneladasByResiduoDistrito(listaResiduos: DataFrame<Residuos>, distrito2: String): String {
    return listaResiduos
        .filter { it["nombre_distrito"] == distrito2 } DataFrame<Residuos>
        .groupBy( ...cols: "nombre_distrito", "residuos") GroupBy<Residuos, Residuos>
        .aggregate { sum( ...columns: "toneladas") into "Toneladas_Totales" } DataFrame<Residuos>
        .sortBy( ...cols: "nombre_distrito").html()
}
```

nombre_distrito	residuos	Toneladas_Totales
Centro	RESTO	45000,16016
Centro	ENVASES	9588,02051
Centro	VIDRIO	7042,54004
Centro	VIDRIO_COMERCIAL	513,33997
Centro	ORGANICA	9087,28027
Centro	CARTON_COMERCIAL	1305,96997
Centro	RCD	1623,15991

DataFrame [7 x 3]

3. Máximo, mínimo, media y desviación por mes por residuo.

En esta consulta hemos obtenido el valor máximo, mínimo, la media y desviación de cada uno de los residuos agrupados por mes. A continuación, pondremos el método que realiza la consulta y su resultado para el distrito “Centro”.

```
private fun estadisticaByMesByResiduo(listaResiduos: DataFrame<Residuos>, distrito2: String): String {
    return listaResiduos
            .filter { it["nombre_distrito"] == distrito2 } DataFrame<Residuos>
            .groupBy( ...cols: "month", "nombre_distrito", "residuos") GroupBy<Residuos, Residuos>
            .aggregate { this: AggregateGroupedDsl<Residuos>  it: AggregateGroupedDsl<Residuos>
                max( ...columns: "toneladas") into "Máximo_Toneladas"
                min( ...columns: "toneladas") into "Mínimo_Toneladas"
                mean( ...columns: "toneladas") into "Media_Toneladas"
                std( ...columns: "toneladas").toString().replace( oldValue: "NaN",  newValue: "0") into "Desviación_Toneladas"
            }.html()
}
```

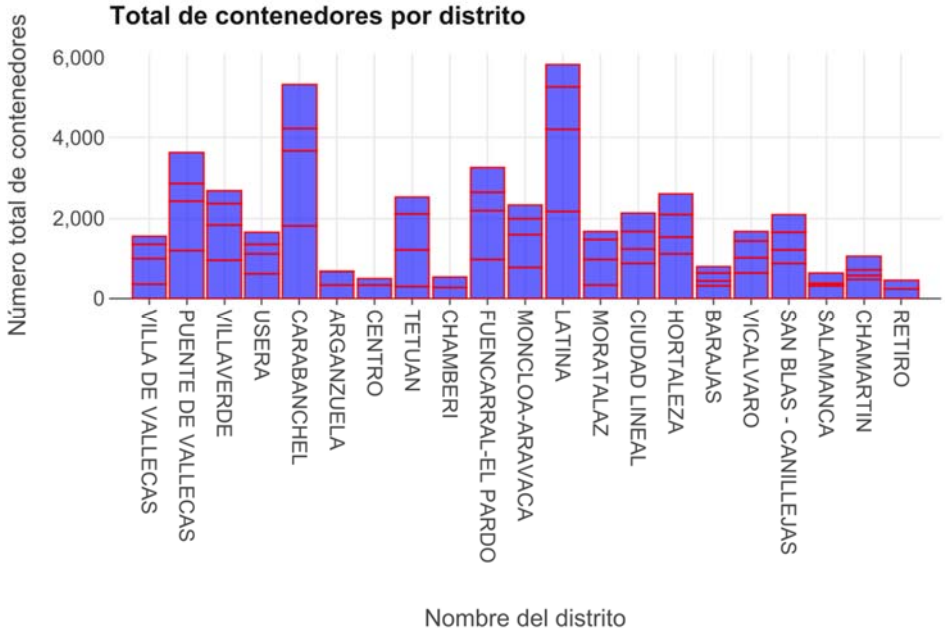
month	nombre_distrito	residuos	Máximo_Toneladas	Mínimo_Toneladas	Media_Toneladas	Desviación_Toneladas
enero	Chamberí	RESTO	2403,3	2403,3	2403,340088	0
enero	Chamberí	ENVASES	269,0	212,9	240,945007	39.605057719598186
enero	Chamberí	VIDRIO	225,3	225,3	225,279999	0
enero	Chamberí	ORGANICA	523,8	523,8	523,760010	0
enero	Chamberí	CARTON_COMERCIAL	62,2	62,2	62,200001	0
enero	Chamberí	RCD	34,8	34,8	34,830002	0
febrero	Chamberí	RESTO	2165,6	2165,6	2165,620117	0
febrero	Chamberí	ENVASES	324,0	257,3	290,629990	47.13574062339763
febrero	Chamberí	VIDRIO	227,1	227,1	227,059998	0
febrero	Chamberí	ORGANICA	757,8	757,8	757,760010	0
febrero	Chamberí	CARTON_COMERCIAL	64,8	64,8	64,800003	0
marzo	Chamberí	RESTO	2351,5	2351,5	2351,459961	0
marzo	Chamberí	ENVASES	355,2	307,3	331,214996	33.891622843266475
marzo	Chamberí	VIDRIO	252,1	252,1	252,139999	0
marzo	Chamberí	ORGANICA	828,1	828,1	828,119995	0
marzo	Chamberí	CARTON_COMERCIAL	75,7	75,7	75,699997	0
marzo	Chamberí	RCD	85,7	85,7	85,730003	0
abril	Chamberí	RESTO	2215,4	2215,4	2215,399902	0
abril	Chamberí	ENVASES	336,0	280,0	308,024994	39.56263303906191
abril	Chamberí	VIDRIO	225,6	225,6	225,600006	0

... showing only top 20 of 71 rows
DataFrame [71 x 7]

Gráficos

Para la parte de Graficas hemos utilizado la dependencia sugerida por nuestro profesor, Lets-Plot, la cual nos da muchas facilidades ya que hemos trabajado las consultas con Dataframes, la parte de las gráficas fue complicada de entender, pero fácil de realizar ya que, una vez terminadas las consultas, Lets-Plot da muchas facilidades para crear las Gráficas.

La primera grafica pedida es con el total de contenedores por distrito, a continuación, mostrare como es la gráfica y posteriormente como hicimos el método respectivo para crear la gráfica Estas graficas deberán ser guardadas en el mismo directorio que el HTML que generemos para que este pueda acceder a ellas.



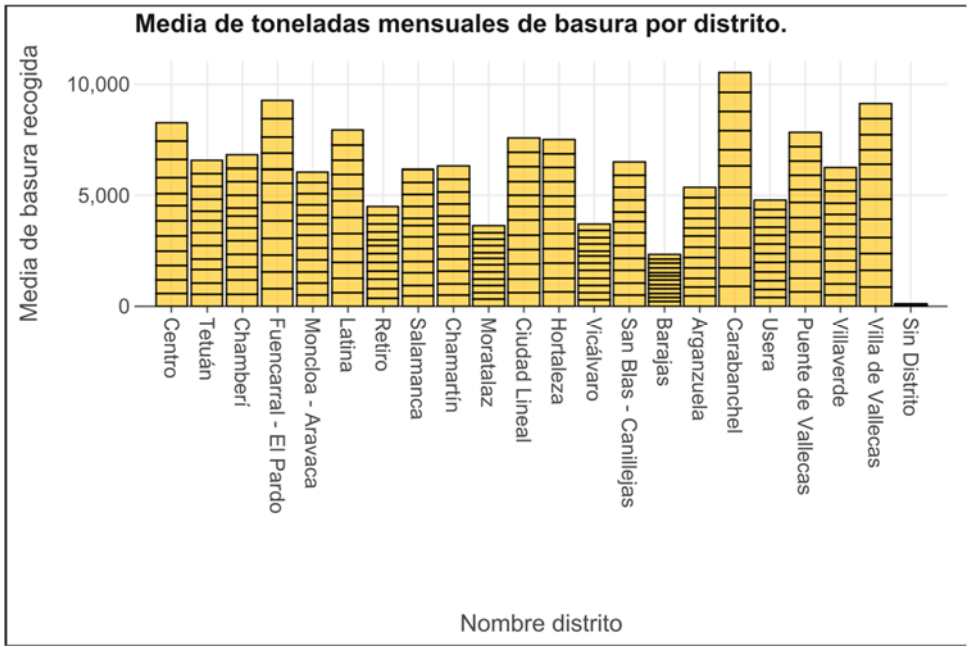
```
*/
Rubén García-Redondo Marín +1
private fun graficoTotalContenedoresDistrito(listaContenedores: DataFrame<Contenedores>, pathFinal: String) {
    val consulta = listaContenedores
        .groupBy( ...cols: "distrito", "type_Contenedor")
        .aggregate { sum( ...columns: "cantidad") into "TotalContenedores" }.toMap()

    val fig: Plot = letsPlot(data = consulta) + geomBar(
        stat = identity,
        alpha = 0.6,
        fill = Color.BLUE,
        color = Color.RED
    ) { this: BarMapping
        x = "distrito"
        y = "TotalContenedores"
    } + labs(
        x = "Nombre del distrito",
        y = "Número total de contenedores",
        title = "Total de contenedores por distrito"
    )

    val path = pathFinal + File.separator + "images"
    if (!Paths.get(path).exists()) {
        Files.createDirectory(Paths.get( first: pathFinal + File.separator + "images" + File.separator))
    }

    ggsave(fig, path = path + File.separator, filename = "total_contenedores_distrito.png")
}
```

La segunda Grafica será la Media de Toneladas mensuales de basura por distrito, al igual que antes mostrare la gráfica y posteriormente el método que la crea

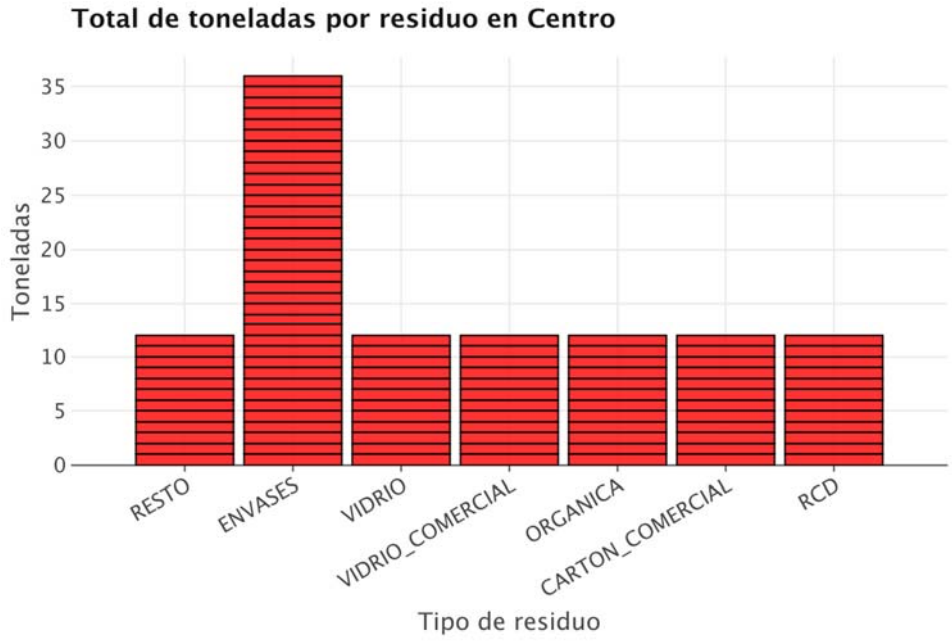


```
private fun graficoMediaToneladasDistrito(listaResiduos: DataFrame<Residuos>, pathDestino: String) {
    val consulta = listaResiduos
        .groupBy( ...cols: "nombre_distrito", "month")
        .aggregate { mean( ...columns: "toneladas") into "media" }.toMap()

    val fig: Plot = letsPlot(data = consulta) + geomBar(
        stat = identity,
        alpha = 0.6,
        fill = Color.ORANGE,
        color = Color.BLACK
    ) { this: BarMapping
        x = "nombre_distrito"
        y = "media"
    } + labs(
        x = "Nombre distrito",
        y = "Media de basura recogida",
        title = "Media de toneladas mensuales de basura por distrito."
    )
    val path = pathDestino + File.separator + "images"
    if (!Paths.get(path).exists()) {
        Files.createDirectory(Paths.get( first: pathDestino + File.separator + "images" + File.separator))
    }
    ggsave(fig, path = path + File.separator, filename = "media_toneladas_distrito.png")
}
```

Estas dos graficas corresponde a la opción RESUMEN del proyecto en la que se debe tomar la información de los contenedores y de la recogida, procesarla generando un resumen.html con los resultados de las consultas y sus graficas

A continuación, os enseñare las gráficas que pertenecen a la opción de RESUMEN DISTRITO. El primer grafico es el total de toneladas recogidas por residuo en ese distrito.



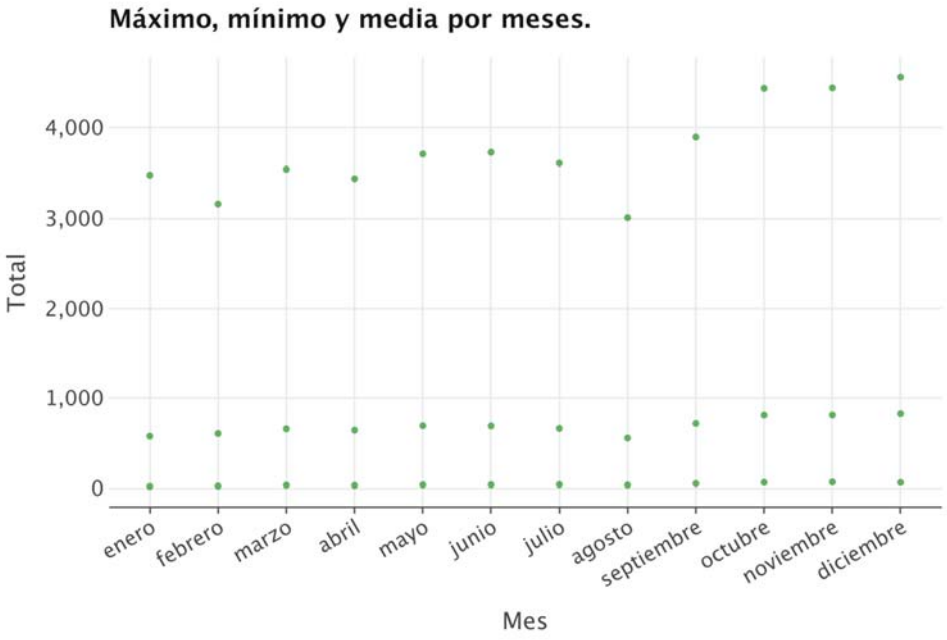
El método para la creación de esta grafica sería el siguiente

```
Rubén García-Redondo Marín + 1
private fun graficoTotalToneladasResiduoDistrito(listaResiduos: DataFrame<Residuos>, distrito2: String, pathFinal: String) {
    val res = listaResiduos
        .filter { it["nombre_distrito"] == distrito2 } DataFrame<Residuos>
        .groupBy( ...cols: "residuos", "toneladas") GroupBy<Residuos, Residuos>
        .aggregate { this: AggregateGroupedDsl<Residuos> it: AggregateGroupedDsl<Residuos>
            count() into "totalToneladas"
        } DataFrame<Residuos>
        .toMap()

    val fig: Plot = letsPlot(data = res) + geomBar(
        stat = identity,
        alpha = 0.8,
        fill = Color.RED,
        color = Color.BLACK,
    ) { this: BarMapping
        x = "residuos"
        y = "totalToneladas"
    } + labs(
        x = "Tipo de residuo",
        y = "Toneladas",
        title = "Total de toneladas por residuo en $distrito2"
    )

    val path = pathFinal + File.separator + "images"
    if (!Paths.get(path).exists()) {
        Files.createDirectory(Paths.get( first: pathFinal + File.separator + "images" + File.separator))
    }
    ggsave(fig, path = path + File.separator, filename = "toneladas_tipo_$distrito2.png")
}
```

Todas las gráficas anteriores son geomBar que son gráficos de barras, pero para la última grafica hemos utilizado geomPoint que es un gráfico con puntos



Y su método sería el siguiente

```
Rubén García-Redondo Marín +1
private fun graficoMaxMinMediaPorMeses(listaResiduos: DataFrame<Residuos>, distrito2: String, pathFinal: String) {
    val res = listaResiduos.filter { it["nombre_distrito"] == distrito2 } DataFrame<Residuos>
        .groupBy( ...cols: "nombre_distrito", "month") GroupBy<Residuos, Residuos>
        .aggregate { this: AggregateGroupedDsl<Residuos> it: AggregateGroupedDsl<Residuos>
            max( ...columns: "toneladas") into "Máximo"
            min( ...columns: "toneladas") into "Mínimo"
            mean( ...columns: "toneladas") into "Media"
        }.toMap()

    val fig: Plot = letsPlot(data = res)  + geomPoint(
        stat = identity,
        alpha = 0.6,
        fill = Color.DARK_BLUE,
        color = Color.DARK_GREEN
    ) { this: PointMapping
        x = "month"
        y = "Mínimo"
    } + geomPoint(
        stat = identity,
        alpha = 0.6,
        fill = Color.YELLOW,
        color = Color.DARK_GREEN
    ) { this: PointMapping
        x = "month"
        y = "Media"
    } + geomPoint(
        stat = identity,
        alpha = 0.6,
        fill = Color.RED,
        color = Color.DARK_GREEN
    ) { this: PointMapping
        x = "month"
        y = "Máximo"
    } + labs(
        x = "Mes",
        y = "Total",
        title = "Máximo, mínimo y media por meses."
    )
```

```
        title = "Máximo, mínimo y media por meses."
    )
    val path = pathFinal + File.separator + "images"
    if (!Paths.get(path).exists()) {
        Files.createDirectory(Paths.get( first: pathFinal + File.separator + "images" + File.separator))
    }
    ggsave(fig, path = path + File.separator, filename = "estadisticas_mensual_$distrito2.png")
```

Justificación Tecnológica

En esta práctica hemos utilizado Kotlin ya que con lo aprendido en clase este lenguaje nos daba muchas más facilidades que java a la hora de leer CSV, crear modelos y utilizar DataFrames, aparte de la utilización de las Gráficas con Lets-Plot, decidimos Kotlin por la sencillez del lenguaje y la comodidad a la hora de programar. Hemos utilizado DataFrames por encima de Streams u otra forma de filtrado ya que con los DataFrames podíamos utilizar métodos propios de DataFrames que nos facilitabas las consultas, ya sea así filtrando las listas para sacar los elementos que queríamos o simplemente para calcular los max,min,media,std , ya que con los DataFrames se puede hacer metiendo esos métodos dentro de un aggregate , y no daría ningún problema, otra de las razones principales de utilizar kotlin y a su vez DataFrames ,es la librería de Lets-plot ya que con lo que vimos en clase podíamos realizar muchos tipos diferentes de gráficos una vez hubiéramos conseguido la consulta del DataFrames el grafico se haría de forma casi inmediata, lo único complicado de esta parte fue entender exactamente cómo funciona la librería , una vez creado un gráfico , podías hacer 100 más de forma sencilla. Como IDE para programar utilizamos Intellij, ya que es el segundo año que llevamos trabajando con este IDE, y gracias a que tenemos la versión Ultimate, el IDE está repleto de ayudas, para que a la hora de programar todo sea más ameno y claro.