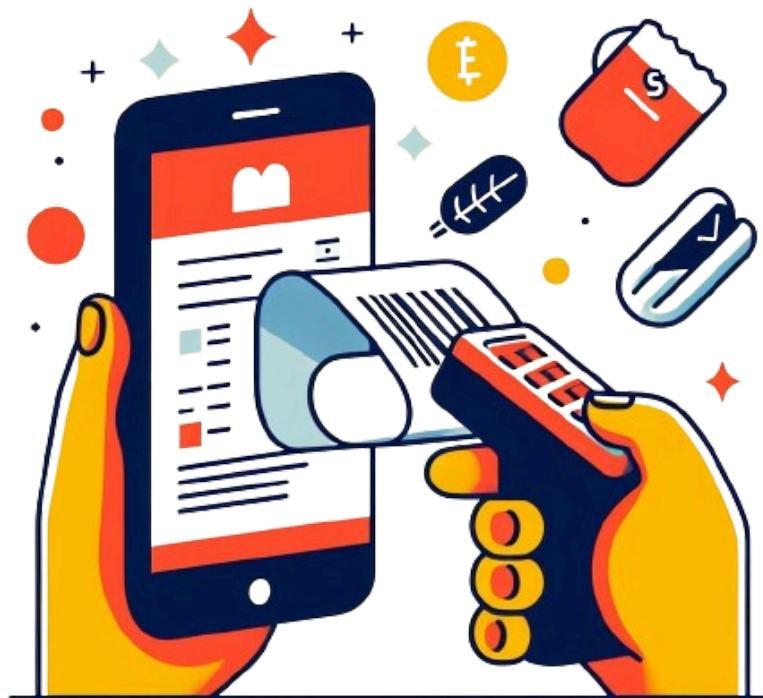


Ticket Scan

Automatització i anàlisi de tiquets de supermercats



Imatge generada amb intel·ligència artificial a partir de la descripció del projecte.

Jaume Juan i Oriol Torrent

INS Sa Palomera
Curs d'Especialització en Intel·ligència Artificial i Big Data

31-05-2024

Índex

0. Pluja de Idees Inicial i Proposta Final del Projecte

- 0.1 Idea
- 0.2 Datasets
- 0.3 Idea Final
- 0.3 Plantejament inicial i desenvolupament
- 0.4 Objectius personals i motivació
- 0.5 Requisits tècnics
 - 0.5.1 Tecnologies utilitzades
 - 0.5.2 Necessitats de hardware
 - 0.5.2.1 Requisits mínims

1. Resum i Abstracte

- 1.1 A alt nivell
- 1.2 A baix nivell

2. Motivació i Objectius del Projecte

3. Introducció

4. Metodologia General

5. Anàlisis del Problema

6. Extracció / Mineria de dades

- 6.1 Dades dels Supermercats (establiments)
- 6.2 Dades dels productes
 - 6.2.1 Data Lake local
 - 6.2.1.1 web to raw script
 - 6.2.1.2 raw to bronze script
 - 6.2.1.3 bronze to silver script
 - 6.2.1.4 silver to gold script

7. Plantilla tiquet Mercadona

8. Generació de tiquets

- 8.1 Exploració i preparació de les dades
- 8.2 Creació del tiquet
 - 8.2.1 Preparació del dataset
 - 8.2.2 selecció de n productes
- 8.3 Renderització de la plantilla i generació del tiquet
 - 8.3.1 Creació de la plantilla HTML
 - 8.3.2 Creació de la imatge

9. Dataset train/test

- 9.1 Dataset entrenament reconeixement d'imatges
 - 9.1.1 Coco
 - 9.1.2 Yolo
- 9.3 Visualització i comprovació dels tiquets
- 9.3 Dataset reconeixement de text

10. Xarxes Neuronals

- 10.1 Fonaments Teòrics
 - 10.1.1 YOLO
 - 10.1.2 Tesseract
- 10.2 Entrenament de les xarxes neuronals
 - 10.2.1 Detecció Imatges (YOLO)
 - 10.2.2 Detecció Text (Tesseract)
- 10.3 Test de les xarxes neuronals
 - 10.3.1 Detecció Imatges (YOLO)
 - 10.3.2 Detecció Text (Tesseract)
- 10.4 Funcionalitats addicionals

11. Aplicació

- 11.1 Arquitectura General
- 11.2 Frontend
- 11.3 Backend
- 11.4 Funcionament

12. ETL

- 12.1 Descripció del procés ETL
- 12.2 Esquema ETL
- 12.3 Arquitectura Medallion
- 12.4 Azure Data Lake
- 12.5 Azure SQL Server
- 12.6 Azure Databricks
- 12.7 Azure Data Factory
- 12.8 Apache Spark

13. Dashboard Power BI

- 13.1 Pantalles
- 13.1.1 Inici
- 13.1.2 Tiquets
- 13.1.3 Productes
- 13.1.4 Clients

14. Conclusions

- 14.1 Aprendentatges
- 14.2 Possibles Millores
 - 14.2.1 Mineria de Dades
 - 14.2.2 Plantilla
 - 14.2.3 Power BI
 - 14.2.4 Generador Tiquets
 - 14.2.5 ETL
 - 14.2.6 Aplicació Web
 - 14.2.7 NN imatges
 - 14.2.8 NN altres

0. Pluja de Idees Inicial i Proposta Final del Projecte

0.1 Idea

La nostra premissa inicial era la següent:

- Podem identificar números o lletres d'imatges analitzant com estan col·locats els seus píxels, comparant-los amb els d'un conjunt de dades on estiguin els píxels de la imatge codificats juntament amb la seva etiqueta?

Vam desenvolupar una idea al voltant de construir un lector de números. On l'usuari pogués introduir un valor entre 0 i 9, i nosaltres identificaríem el valor escrit. Però al llarg de les setmanes, a mesura que hem investigat i recopilat informació, hem vist que era una idea bastant bàsica i molt treballada ja per altres persones a internet. Arran d'això i d'haver parlat amb els professors, la nostra idea ha anat evolucionant.

Hem valorat diverses idees durant aquest procés. Però sempre tenim com a referència principal fer alguna cosa relacionada amb la detecció/lectura/classificació d'imatges. A continuació deixem algunes idees que van sortir al respecte:

- Transformació de text escrit a mà a text codificat per ordinador.
 - IA: Crear el model capaç de detectar el text codificat a partir d'una imatge.
 - Big Data: Crear la infraestructura necessària per poder pujar una imatge i tractar-la fins a poder-la processar pel model.
- Detecció de la informació en factures per traspassar la informació cap a una base de dades.
 - IA: Crear el model / xarxa neuronal que donada la imatge d'una factura, en tregui la informació primordial.
 - Big Data: Crear la infraestructura necessària per guardar la imatge, relacionar-la amb les seves dades un cop passades pel model. Això pot implicar crear diferents capes i ETL's per arribar a tenir una arquitectura llesta per fer un anàlisi de dades posterior. (data lake, data warehouse, agregacions, etc).
- Detecció de diferents objectes (cotxe, arbre, autobús, etc) en imatges per veure si el nostre model podria passar un reCAPTCHA de classificació d'imatges (indicar a quines imatges surt 'x' objecte).
 - IA i Big Data: (ho hauríem de pensar, però seria semblant als dos apartats anteriors).

0.2 Datasets

Per dur a terme alguna d'aquestes idees, obviament necessitaríem diferents conjunts de dades. Aquestes són les nostres propostes:

- Transformació de text:
 - Nosaltres mateixos plantejaríem diferents textos que diferents voluntaris haurien d'escriure a mà el que hi ha representat. Així, tindrem diferents tipografies i mostres per crear el conjunt de dades.
 - Patim el risc de no tenir suficients tipologies de text i que la xarxa neuronal només sàpiga interpretar aquells textos (over-fitting).
 - <https://www.nist.gov/srd/nist-special-database-19>
 - Conjunt de dades amb mostres semblants al que comentem al punt anterior (però amb anglès).
- Detecció de la informació d'una factura:
 - Hi ha conjunts de dades amb informació de factures, però creiem que seria millor opcio generar-les nosaltres, com proposem en el següent punt.:
 - <https://universe.roboflow.com/browse/logistics/invoice>
 - <https://www.kaggle.com/discussions/general/263960>
 - Creiem que la millor opcio seria buscar diferents plantilles de factures i amb un petit script generar-ne moltes.
 - Hauríem de tenir informació de mostra (real o almenys que ho sembli) de diferents empreses en un format digital (csv, bbdd, json, etc) per poder generar aquestes factures finals falses amb un script.
 - Informació que necessita una factura:
 - <https://getquipu.com/blog/datos-obligatorios-factura/>
 - <https://www.holded.com/es/blog/facturas-ejemplos>
 - Crear factures amb Python:
 - https://www.youtube.com/watch?v=_txlkMXtPA
 - Exemples de plantilles:
 - <https://templatesjungle.com/best-simple-free-html-invoice-templates/>
 - Com comentem al punt principal, podríem fer servir aquest conjunt de dades:
 - <https://www.kaggle.com/datasets/himanshu007121/invoice-data>
 - Si necessitem més factures, podem fer inverse sampling d'aquest dataset per generar més dades amb diferents plantilles de factures per entrenar la xarxa neuronal.
 - Dimensió: tan gran com siguem capaços de generar nosaltres.
 - Detecció objectes:
 - Hauríem de tenir diferents conjunts de dades amb la corresponent imatge i si hi apareix l'objecte i quins objectes hi ha a dins.

- <https://es.shaip.com/blog/22-open-source-image-data-for-computer-vision/>
 - Dimensió: No la sabem exacta, però seria el més gran.

0.3 Idea Final

Segons el que hem estat discutint a classe juntament amb els professors i el altres companys, hem decidit que la idea de les factures era prou bona, però massa extensa.

Hem decidit pivotar a una idea semblant però analitzant tiquets de compra en comptes de factures. Els tiquets, són més homogenis i més fàcils d'examinar per una xarxa neuronal.

Crearem tiquets de mostra (igual que anàvem a fer amb les factures i com s'especifica en el punt anterior) però amb les dades que ens ha proporcionat en (*professor X*) dels preus dels supermercats (així també li donem una utilitat extra a aquestes dades). D'aquesta manera tindrem un *Ground Truth* dels preus de productes prou gran i ajustat a la realitat.

0.3 Plantejament inicial i desenvolupament

- Fase d'exploració de les tecnologies.
 - Si no ho fem a classe, faríem el típic exercici de classificació de números amb el conjunt de dades 'mnist' per crear i provar alguns models:
 - provar i explicar superficialment: knn, svc, perceptron, gaussianNB (bayes) i provar-los contra aquest conjunt de dades.
 - Passar a fer-ho amb xarxes neuronals:
 - Explicació bàsica del concepte NN.
 - Fonaments matemàtics.
 - Comparar i analitzar els resultats.
 - Crear el dataset de tiquets a partir del .csv de productes dels supermercats. Crear diferents tiquets per els diferents supermercat. Agafar tiquets reals de mostra i fer un mockup semblant.
- Fase implementació de les tecnologies al nostre problema.
 - Entrenar la xarxa neuronal amb el conjunt de dades. Comprovant la seva fiabilitat amb train/test.
 - Veure d'alguna manera, donada una imatge de test, d'on està traient la informació la xarxa neuronal.
 - Crear el conjunt de dades final de tiquets VS. informació que hi ha.
- Fase construcció producte final.
 - Crear l'arquitectura necessària (Azure) per guardar els resultats.
 - Escanejar tots els tiquets.
 - Crear un script que agafi totes les imatges d'una carpeta, les passi pel model i guardi la informació al cloud (ETL, capes, agregacions, per anàlisis de dades).

- Fer anàlisis de dades amb la informació extreta (analitzar vendes i compres de l'empresa, entre altres possibles mètriques).

0.4 Objectius personals i motivació

- Poder identificar els principals paràmetres del tiquet com:
 - Import: total i per producte unitari.
 - Productes que s'han comprat.
 - Data d'emissió.
 - Nom i ubicació del supermercat.
- A part de ser un repte per nosaltres. També pot tenir un ús útil al món real.
 - Confirmació dels productes comprats i cobrats per part del supermercat per persones cegues a través d'enumerar per l'altaveu les característiques del tiquet.
 - Ajudar a la gestió de gastos que fem al supermercat.
 - Un cop escanejat el tiquet; podem guardar que ha comprat cada client i el que li ha costat. Amb això podem tenir un millor control de què comprem i quan ens gastem al supermercat.
 - Si tenim la informació dels tiquets en una base de dades, podem fer un anàlisis d'aquesta informació.
 - Actualment, els supermercats ja està implementant el tiquet electrònic i ja no tindran un ús tan real. Malgrat això, creiem que el projecte és prou interessant.

0.5 Requisits tècnics

0.5.1 Tecnologies utilitzades

- Azure
- Jupyter Notebook
- Python (Amb diverses llibreries Pandas, Numpy...)
- Llibreries específiques per crear models neuronals.
- etc.

0.5.2 Necessitats de hardware

El hardware necessari sempre intentarem tenir el millor que puguem aconseguir, ja que un bon hardware facilita les tasques i redueix el temps necessari per processar-les. Tot i això hem establert uns requisits mínims per al projecte que la nostre màquina ha de complir.

0.5.2.1 Requisits mínims

- **CPU:** Necessitem un processador modern amb almenys 4 nuclis.
- **GPU:** No és obligatòria però si la tenim molt millor, ens permet augmentar la velocitat del procés d'entrenament.

- **Memòria RAM:** Es necessita com a mínim 8GB de memòria RAM en el nostre cas.
- **Emmagatzematge:** És necessari emmagatzematge suficient per el codi, les dades, i els models entrenats. Si pot ser una memòria SSD millor ja que ens donà un accés més ràpid a les dades. Per el nostre projecte tenim un pronòstic de 50Gb ja que hi haurà moltes imatges i tenen un pes gran.
- Com és lògic necessitem connexió a internet.

1. Resum i Abstracte

1.1 A alt nivell

Com ja hem comentat a la [proposta](#); el nostre projecte es basa en crear una aplicació a partir d'una xarxa neuronal capaç de detectar la informació de tiquets de supermercat (productes i preus), guardar i tractar aquesta informació al cloud de Azure a través de diferents procediments i finalment presentar i fer un anàlisis de les dades.

1.2 A baix nivell

El projecte "Ticket Scan" és una aplicació web que automatitza el procés d'escaneig i anàlisi de tiquets del supermercat Mercadona. El seu objectiu és extreure informació detallada dels tiquets, com productes i preus, utilitzant una combinació de tècniques avançades de processament d'imatges i reconeixement de text. Per entrenar la xarxa neuronal, s'ha generat una plantilla visual de supermercat i s'han fet servir dades de productes i preus de supermercats en línia per generar tiquets de mostra.

El procés d'anàlisi es basa en l'arquitectura YOLO (You Only Look Once), que permet identificar i localitzar objectes en imatges de manera eficient. A més, s'usa el model Tesseract per reconèixer el text de les diferents seccions dels tiquets detectades per YOLO i així obtenir informació precisa sobre cada element.

Una cop extreta la informació, l'aplicació la presenta la informació de manera clara i accessible al Frontend (client), on també es fa servir el model gTTS per convertir el text en audio, permetent a persones amb discapacitat visual verificar les seves compres de manera auditiva.

A més de l'anàlisi individual dels tiquets, el projecte inclou una part d'anàlisi estadístic que permet als usuaris aconseguir *insights* sobre els seus hàbits de compra i comparar-los amb altres clients. Aquestes estadístiques es visualitzen mitjançant Power BI, proporcionant una representació gràfica interactiva de les dades. Per realitzar aquest anàlisi, s'ha implementat l'arquitectura Medallion implementada amb Azure Data Factory i Azure Databricks, que garanteix una transformació i emmagatzematge eficients de les dades, permetent una manipulació àgil i fàcil de les dades.

En resum, "Ticket Scan" ofereix una solució integral per a la gestió i anàlisi de tiquets de supermercat, en aquest cas, del supermercat Mercadona.

2. Motivació i Objectius del Projecte

La principal motivació ha sigut fer un projecte transversal amb el qual, assolir tant la part de big data com d'intel·ligència artificial. Els aspectes que desde uns inicis volíem tocar eren els següents:

- Treballar amb dades reals.
 - Fer servir apis per recol·lectar informació.
 - Tractament de json, csv i altres formats de big data com parquet.
- Treballar amb tecnologies modernes.
 - BBDD al núvol.
 - Creació de scripts per tractar les dades (ETL).
- Construir un model predictiu.
 - Entrenament de xarxes neuronals.
 - Avaluació dels resultats.
- Bona presentació dels resultats.
 - Creació d'una web per mostrar els resultats i interactuar amb l'aplicació.

3. Introducció

Quan volem abordar un projecte amb intel·ligència artificial, hi ha dos grans aspectes a considerar: les dades i els models.

Quan programem amb intel·ligència artificial, a diferència de la programació tradicional, no indiquem les instruccions exactes que el programa ha de seguir. És a dir, programem d'una manera estocàstica en comptes d'una manera determinista.

En lloc d'indicar pas a pas com s'ha d'executar el programa en funció del input, proporcionem una sèrie de dades d'entrenament al model, el qual, mitjançant alguns processos matemàtics, generarà un model entrena't que podem fer servir posteriorment per fer inferència. Entenem model com: algorisme estocàstic / caixa negra que converteix inputs en outputs.

Resumint; quan tinguem un nou input, generarem una resposta en funció de les dades d'entrenament que havíem proporcionat i no en funció d'un procés determinista.

Per aquest motiu, que les dades siguin confiables, variades i representatives del món real, també anomenat: *ground truth* és un aspecte clau per crear un bon algorisme d'intel·ligència artificial.

En el nostre cas, generarem tiquets amb dades aleatòries extretes del món real, en format .png i guardarem tota la informació que hi apareix (import total, productes, preus unitaris dels productes, ubicació, etc.).

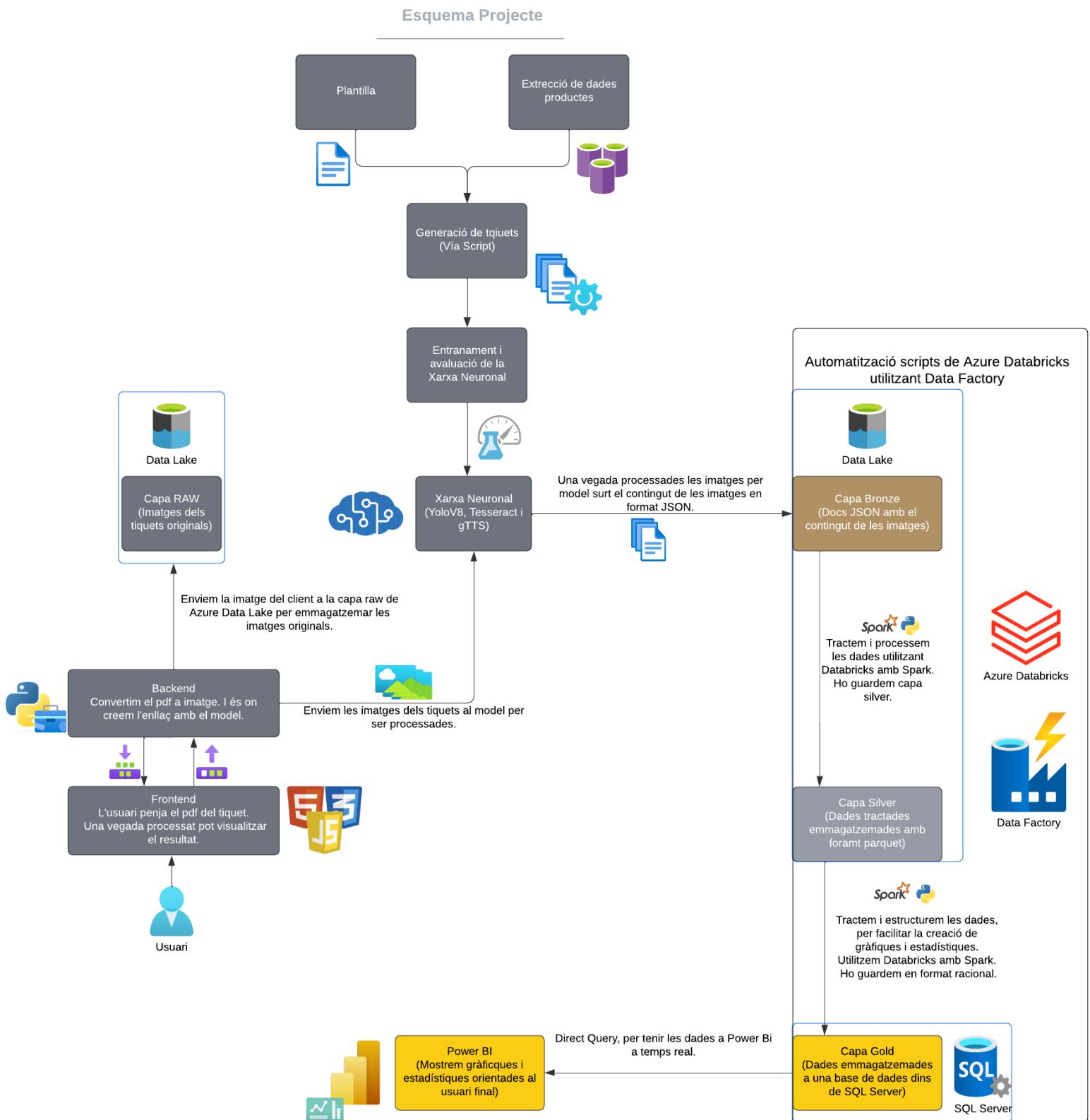
Amb aquestes dades, entrenarem una xarxa neuronal que relacioni la imatge amb les seves dades per tal que en un futur, sigui capaç d'extreure la informació detallada d'un tiquet a partir de la imatge.

4. Metodologia General

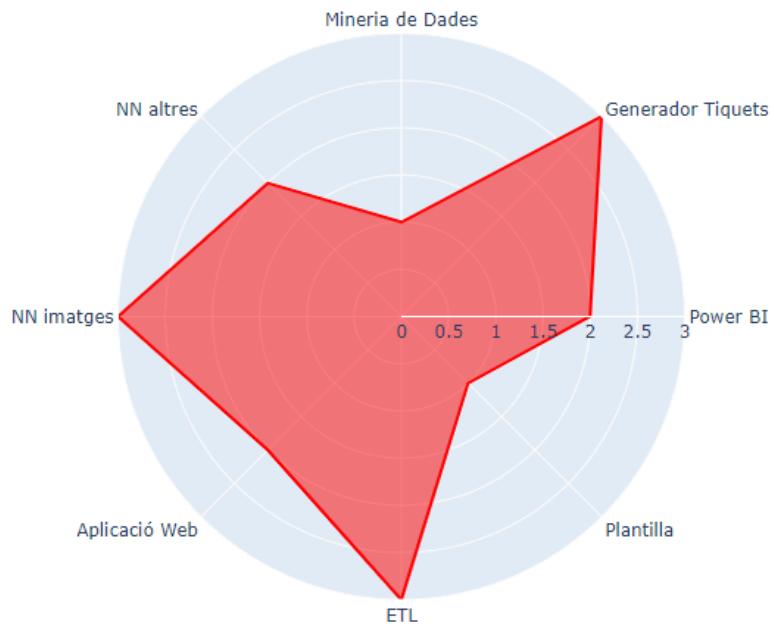
Per assolir el que expliquem a la introducció, hem seguit els passos que es presenten a continuació. Cada punt, és una petita explicació dels apartats que es desenvolupen més endavant en aquest mateix document.

- **Anàlisis del Problema:** El primer que hem de fer és saber a què ens enfrontem, quines eines tenim, quina informació inicial tenim i com ho podem solucionar.
- **Extracció / mineria de dades:** Extraiem les dades de productes i establiments de la web.
- **Creació d'una plantilla:** Elaboració d'una plantilla amb HTML i CSS que servirà com a base per generar els tiquets.
- **Creació d'un script de Python per crear imatges:** A partir de la plantilla i les dades extretes, creem un script per crear imatges de tiquets de forma programàtica.
- **Creació del dataset:** Relacionant les imatges generades amb l'script amb la informació que hi conté (productes, descripcions, preus, ...) i on està situada (coordenades / bounding boxes).
- **Entrenar la xarxa neuronal:** Perquè el model aprengui a detectar on està la informació (cada descripció, preu, total, ...) del tiquet i posteriorment, una altre xarxa neuronal que sàpiga identificar quin text hi ha en cada quadre que ha detectat la xarxa neuronal.
- **Creació d'una aplicació:** Programació d'una interfície i una api perquè l'usuari pugui enviar els seus tiquets a la nostre xarxa neuronal i aquesta li retorna la informació que hi ha i executi els triggers i processos per inicialitzar la ETL.
- **Creació de diferents pipelines/ETL:** Per agafar aquesta informació que surt del model neuronal i a través d'una arquitectura de big data, arribar a transformar-la i guardar-la de la millor manera possible.
- **Dashboard final:** Presentar la informació extreta i ja procesada de la xarxa neuronal mitjançant Power BI.

Tots aquests apartats, a grans trets, es poden representar amb aquest esquema:



Aquest seria l'abast que han tingut les diferents àrees:



En l'apartat de [conclusions](#), comentem cada punt i proposem diferents millors que podem fer en cada àmbit.

Tot el codi relacionat, està guardat en aquest repositori de GitHub. A mida que anem explicant els trets més importants en aquesta memòria, anirem enllaçant als diferents apartats del repositori.

- <https://github.com/jaumejp/analisis-tiquets>

5. Anàlisis del Problema

Per resoldre el problema, necessitem entrenar una xarxa neuronal amb imatges de tiquets i per això necessitem: tenir-los o bé crear-los. Nosaltres hem optat per la segona opció. Crear-los nosaltres mateixos.

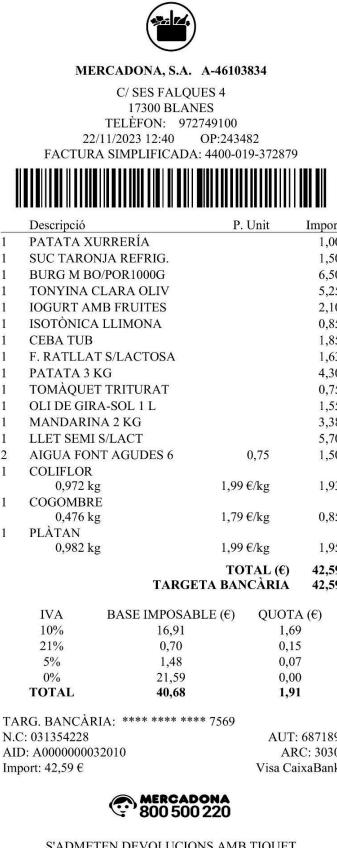
Si volguessim ser capaços de detectar informació de tots els supermercats, necessitariem tenir tiquets de mostra de tots. Per això simplificarem el problema agafant tiquets i productes només del mercadona. Així, només hem de maquetar el tiquet del mercadona amb els seus productes.

Podem trobar diferents mostres de tiquets en aquesta en aquesta part del repositori:

- https://github.com/jaumejp/analisis-tiquets/tree/main/generador-tiquets/exemples_tiquets

Quan anem a comprar al mercadona, aquests ens donen l'opció de generar un tiquet digital, el qual ens arriba en format pdf al correu electrònic.

Aquest és el tiquet:



Com podem observar, hi ha la següent informació:

- El podem subdividir en 4 parts:
 - Capçelera.

- Secció de productes.
- Secció amb els ivas.
- Footer.
- A la secció de productes, tenim 2 tipus de productes:
 - Preu per unitats (1)
 - Preu per €/kg (2)
 - Tenim quantitat, descripció, preu unitari i preu total.

1	MANDARINA 2 KG		3,38
1	LLET SEMI S/LACT	1	5,70
2	AIGUA FONT AGUDES 6	0,75	1,50
1	COLIFLOR 0,972 kg	1,99 €/kg	1,93
1	COGOMBRE 0,476 kg	1,79 €/kg	0,85
1	PLÀTAN 0,982 kg	1,99 €/kg	1,95
TOTAL (€)			42,59

Els productes unitaris (1):

- Si tenen només 1 unitat, el preu unitari no es veu reflexat al tiquet.
- Si tenen més de 1 unitat, si que s'hi veu reflexat.

Els productes de preu al kilo (2):

- A la primera columna (unitats) sempre hi ha el número 1.
- La quantitat són els Kg que hem comprat que està en una segona línia.

El següent pas serà aconseguir aquesta informació dels productes:

- Mandarina 2kg = 3,38€
- Llet semi s/lact = 5,70€
- etc

6. Extracció / Mineria de dades

Per aconseguir la informació dels productes, hem considerat tres opcions:

- Fer servir el conjunt de dades que un professor del institut ens podia proporcionar.
 - Problemes: noms amb castella, no hi ha l'iva.
 - Avantatges: diferents supermercats, moltes dades i de molts dies diferents.
- Fer servir el conjunt de dades de [internet](#).
 - Problemes: no són preus actuals i tampoc hi ha l'iva.
 - Avantatges: diferents supermercats i moltes dades.
- Crear nosaltres mateixos un conjunt de dades a partir de les dades que publiquen els supermercats a les botigues en línia.
 - Problemes: Més difícil de scrapejar.
 - Solució: com que ens centrem només amb el mercadona, no serà un problema.
 - Avantatges: dades estructurades a la nostre manera i amb català.

Hem cregut que la millor opció és la de crear nosaltres el dataset ja que parlem d'un projecte educatiu on s'han de demostrar les habilitats adquirides al llarg del curs.

Hem extret 2 tipus de informacions diferents: de productes i de establiments. I, ho hem fet de la següent manera:

6.1 Dades dels Supermercats (establiments)

Mercadona online té aquesta web amb tots els supermercats.

- <https://info.mercadona.es/es/supermercados>

Si fem una cerca i mirem a la network, veiem que no fa crides cada cop que fem una nova cerca:

The screenshot shows a browser window with the Mercadona website loaded. The address bar shows the URL: info.mercadona.es/es/supermercados?s=Barcelona. The developer tools Network tab is open, displaying a list of network requests. The requests include several pings to 'js?tid=G-0XCSCX...', which corresponds to the 'ping' entries in the table below. The main content of the page shows a search bar with 'Barcelona' and a results section for 'Barcelona - Avda. Meridiana, 326'.

Name	Status	Type	Initiator	Size	Time	Waterfall
collect?v=2&tid=G-0XCSCXMPGV>m=...	204	ping	js?tid=G-0XCSCX...	17 B	319 ...	
ga-audiences?v=1&t=sr&sf_r=1&r=4...	200	gif	js?tid=G-0XCSCX...	63 B	243 ...	
collect?v=2&tid=G-0XCSCXMPGV>m=...	204	ping	js?tid=G-0XCSCX...	17 B	69 ms	

Si examinem els recursos de la web i anem a:

- Sources > estaticos > cargas

```

1 var dataJson = [
2   {
3     "id": 883185314342,
4     "pv": "ES",
5     "p": "A CORUÑA",
6     "lc": "A CORUÑA",
7     "dr": "AVDA. DE OZA, 132",
8     "cp": "15000",
9     "tf": "+34 981383071",
10    "f1": "C*****",
11    "f2": "C*****",
12    "fa": "14/04/24-C#21/04/24-C#28/04/24-C#01/05",
13    "pk": "$",
14    "l1": "43.353425924715,
15    "l2": "-8.393176181545,
16    "fap": "24/11/2011"
17  },
18  {
19    "id": 883185313520,
20    "pv": "ES",
21    "p": "A CORUÑA",
22    "lc": "A CORUÑA",
23    "dr": "C/ RONDA OUTEIRO, 217",
24    "cp": "15007",
25    "tf": "+34 981277366",
26    "f1": "C*****",
27    "f2": "C*****",
28    "fa": "14/04/24-C#21/04/24-C#28/04/24-C#01/05",
29    "pk": "$",
30    "l1": "43.35827753458,
31    "l2": "-8.419614926424,
32    "fap": "09/11/2006"
33  },
34  ...
35 ]
  
```

Veiem que hi ha un objecte que fa servir el frontend per mostrar la informació que se li demana. Si examinem aquest objecte desde la consola, viem tota la informació:

```

0: {id: 883185314342, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Avda. de Oza, 132", ...}
1: {id: 883185313520, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "C/ Ronda Outeiro, 217", ...}
2: {id: 883185312329, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Lugar de Montepetalo, 63", ...}
3: {id: 883185313502, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Rúa José Pascual López Cortón, 10", ...}
4: {id: 883185313510, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Rúa Pandaderas, 28", ...}
5: {id: 883185313512, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Avda. de Alfonso, 1", ...}
6: {id: 883185314405, pv: "ES", p: "A Coruña", lc: "A Coruña", dr: "Avda. de Alfonso, 1", ...}
7: {id: 883185313609, pv: "ES", p: "A Coruña", lc: "Arteixo", dr: "Avda. Diputación, S/N (p.i. de Sabón)", ...}
8: {id: 883185315462, pv: "ES", p: "A Coruña", lc: "Betanzos", dr: "Rúa Fraga Iribarre, 7", ...}
9: {id: 883185316124, pv: "ES", p: "A Coruña", lc: "Boiro", dr: "Rúa Principal, 75", ...}
10: {id: 883185316124, pv: "ES", p: "A Coruña", lc: "Carballo", dr: "Avenida, Revolta, S/N", ...}
11: {id: 883185315586, pv: "ES", p: "A Coruña", lc: "Cee", dr: "Lc. Vilari de Toba, S/N", ...}
12: {id: 883185315704, pv: "ES", p: "A Coruña", lc: "Culleredo", dr: "Avda. de Alvedro, 5", ...}
13: {id: 883185316274, pv: "ES", p: "A Coruña", lc: "Fene", dr: "Rúa Áncoras, S/N", ...}
14: {id: 883185313344, pv: "ES", p: "A Coruña", lc: "Gondomar", dr: "Avda. de Montero, 1", ...}
15: {id: 883185316509, pv: "ES", p: "A Coruña", lc: "Pontevedra", dr: "Avda. Pedro Barreiro, S/N", ...}
16: {id: 883185314724, pv: "ES", p: "A Coruña", lc: "Narón", dr: "Avda. Do Mar, 52", ...}
17: {id: 883185313292, pv: "ES", p: "A Coruña", lc: "Noia", dr: "C/ Pedra Santaña, 41", ...}
18: {id: 883185315510, pv: "ES", p: "A Coruña", lc: "Oleiros", dr: "Avda. Das Mariñas, 281", ...}
19: {id: 883185313758, pv: "ES", p: "A Coruña", lc: "Oleiros", dr: "Avda. Ramón Núñez Montero, 52", ...}
20: {id: 883185313248, pv: "ES", p: "A Coruña", lc: "Ribreira", dr: "R. Xohana Torres, 12", ...}
21: {id: 883185313508, pv: "ES", p: "A Coruña", lc: "Santiago de Compostela", dr: "Rúa Castela E Lédo, 4", ...}
22: {id: 883185314096, pv: "ES", p: "A Coruña", lc: "Santiago de Compostela", dr: "Rúa María Casares, 5", ...}
23: {id: 883185316026, pv: "ES", p: "Alacant", lc: "Alacant/Alicante", dr: "Avda. Antonio Ramos Carratilla, 175", ...}
24: {id: 883185312530, pv: "ES", p: "Alacant", lc: "Alacant/Alicante", dr: "Avda. de La Condomina, S/N", ...}
  
```

Copiem aquest objecte en un Notebook i tractem les dades.

Aquest procés, es pot veure en el següent Notebook.

- https://github.com/jaumejp/analisis-tiquets/blob/main/generador-tiquets/dades/estabili_ments/collect_establishments.ipynb

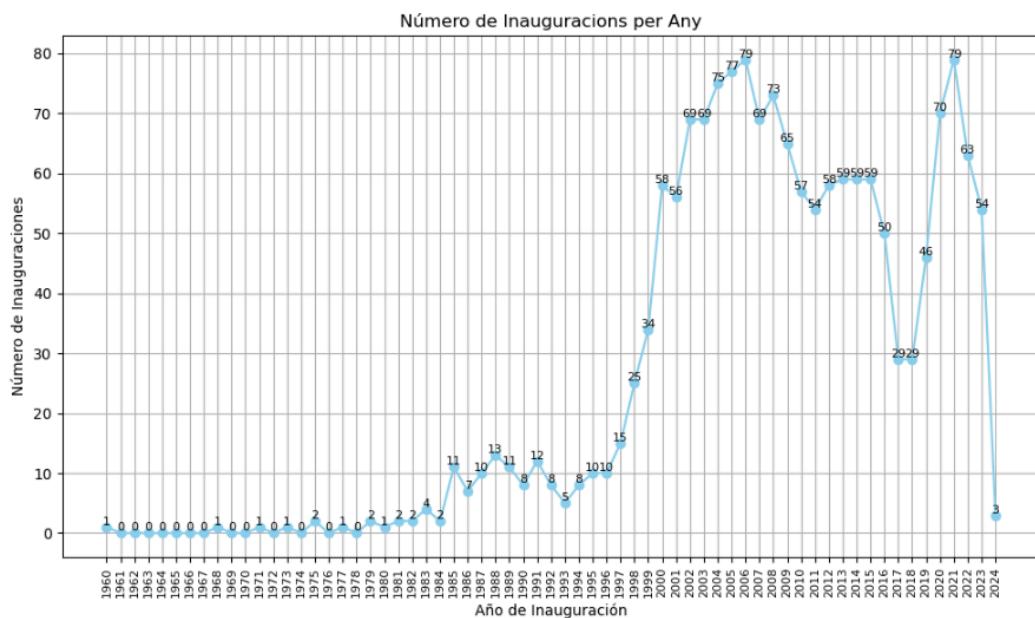
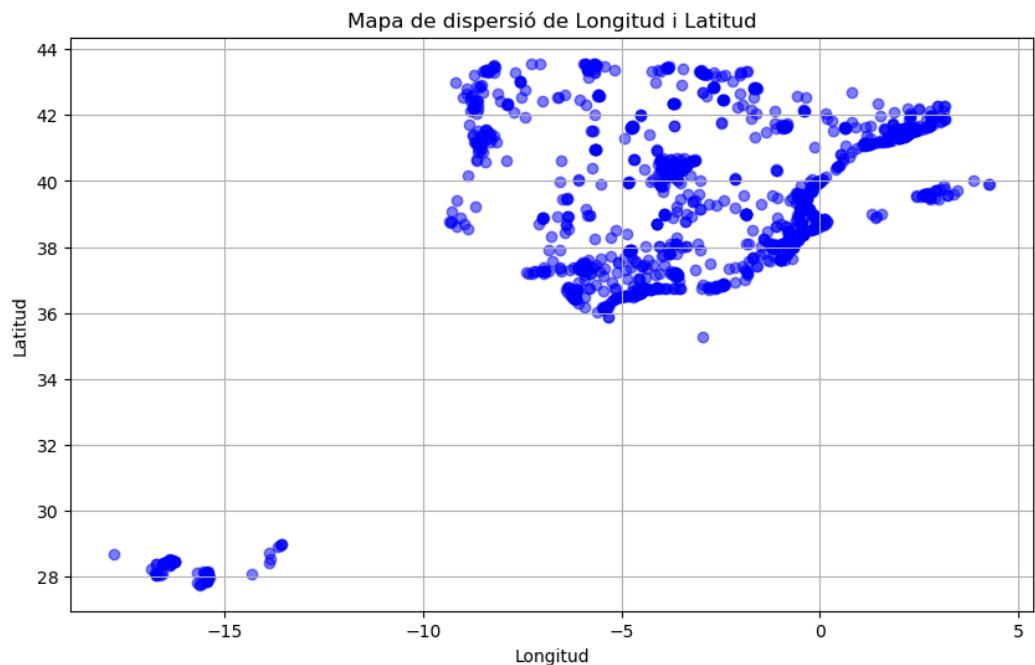
Però, resumidament fem el següent:

- Transformem l'anomenat objecte en un dataframe de pandas.
- Fem modificacions per tenir les dades més estructurades.
- Fem algunes visualitzacions per poder tenir una referència general de com són les dades dels establiments del mercadona.
- Guardem el dataframe en format csv en local.

Algunes visualitzacions:

In [9]:

```
mostrar_mapa(df.longitud, df.latitud)
```



6.2 Dades dels productes

Botiga online de Mercadona:

- <https://tienda.mercadona.es/>

Si anem a la pestanya *categorias* i examinem les peticions que fa aquesta pàgina, veiem:

The screenshot shows the Mercadona online store interface. On the left, there's a sidebar with various food categories like 'Aceite, especias y salsas', 'Aperitivos', and 'Carne'. The main content area displays two products: 'Aceite de oliva' (Botella 1 L) and 'Aceite de oliva virgen extra' (Garrucha 3 L). The DevTools Network tab is open, showing a list of requests. One request is highlighted with a red circle and the number '3', pointing to the URL: https://tienda.mercadona.es/api/categories/112?lang=es&wh=3951. The request method is GET, status code is 200 OK (from disk cache), and the response headers include 'Content-Type: application/json', 'Cross-Origin-Opener-Policy: same-origin', and 'Expires: Sat, 13 Apr 2024 09:53:18 GMT'.

1. Pestanya que mostra les categories.
2. Si Anem a Inspeccionar > Network > Fetch/XHR
3. Veiem que fa una crida a una api per cada categoria.

Les crides a la api són les següents:

Totes les categories

- <https://tienda.mercadona.es/api/categories/?lang=ca>

JSON

```
{  
    "next": null,  
    "count": 26,  
    "results": [  
        {  
            "id": 12,  
            "name": "Oli, espècies i salses",  
            "order": 7,  
            "layout": 2,  
            "published": true,  
            "categories": [  
                {  
                    "id": 112,  
                    "name": "Oli, vinagre i sal",  
                    "order": 7,  
                    "layout": 1,  
                    "published": true,  
                    "is_extended": false  
                },  
                // Totes les subcategories  
            ],  
            "is_extended": false  
        },  
        // Totes les altres categories principals
```

Si anem fent requests per totes les id, tindrem la informació de cada categoria principal. Ho provem amb:

- nom: Oli, espècies i salses
- id 12: <https://tienda.mercadona.es/api/categories/12?lang=ca>

JSON

```
{  
    "id": 12,  
    "name": "Oli, espècies i salses",  
    "layout": 2,  
    "categories": [  
        {  
            "id": 112,  
            "name": "Oli, vinagre i sal",  
            "layout": 1,  
            "products": [  
                {  
                    "id": "19733",  
                    "limit": 1000,  
                    "badges": {  
                        "is_water": false,  
                        "requires_age_check": false  
                    },  
                    "packaging": "Paquet",  
                    "thumbnail": "https://prod-  
mercadona.imgix.net/20190521/33/19733/vlc1/19733_00_02.jpg?  
fit=crop&h=206&w=206",  
                    "display_name": "Sal gruixuda Hacendado",  
                    "bunch_selector": false,  
                    "price_instructions": {  
                        "iva": 10,  
                        "is_new": false,  
                        "is_pack": false,  
                        "pack_size": null,  
                        "unit": "unid"  
                    }  
                }  
            ]  
        }  
    ]  
}
```

Veiem que cada categoria té subcategories associades. Per tant, també podem fer requests per cada id de sub categoria:

- nom: Oli, vinagre i sal
- id 112: <https://tienda.mercadona.es/api/categories/112?lang=ca>

```

{
  "id": 112,
  "name": "Oli, vinagre i sal",
  "order": 7,
  "layout": 1,
  "published": true,
  "categories": [
    {
      "id": 420,
      "name": "Oli d'oliva",
      "order": 7,
      "layout": 2,
      "products": [
        {
          "id": "4240",
          "slug": "aceite-oliva-04o-hacendado-botella",
          "limit": 999,
          "badges": {
            "is_water": false,
            "requires_age_check": false
          },
          "packaging": "Ampolla",
          "published": true,
          "share_url": "https://tienda.mercadona.es/product/4240/aceite-oliva-04o-hacendado-botella",
          "thumbnail": "https://prod-mercadona.imgix.net/images/1a84cd7052b68873985104ac24b87043.jpg?fit=fill&w=200&h=200"
        }
      ]
    }
  ]
}

```

Un cop fet aquest anàlisis, només ens falta de forma periòdica, fer crides a les apis i transformar la informació per obtenir la informació dels productes.

Ho farem amb una estructura i arquitectura Medallion en un data lake en local.

6.2.1 Data Lake local

Farem una arquitectura de Data Lake bàsica en local per guardar aquests fitxers i poder-los tractar.

Si tenim temps ens agradaria unificar-ho amb el data lake de productes escanejats pels usuaris. De moment, tenim aquest en local.

L'hem estructurat de la següent manera:

Dins de la carpeta:

- https://github.com/jaumejp/analisis-tiquets/tree/main/generador-tiquets/dades/productes/dataset_propi/DataLake

Hi haurà tota la estructura de carpetes i fitxers que posteriorment s'hauria de passar al núvol i automatizar els scripts per fer-ho correctament.



Ara, procedirem a explicar cada capa i el corresponent script què s'encarrega de traspasar la informació:

6.2.1.1 web to raw script

- https://github.com/jaumejp/analisis-tiquets/blob/main/generador-tiquets/dades/productes/dataset_propi/DataLake/0-web-to-raw.ipynb
- Dins la capa (directori) creem una carpeta amb el dia que s'està executant l'script.
- Cridem l'endpoint que retorna totes les categories (És el primer json dels tres que hem comentat anteriorment) i, es guarda la resposta en local dins aquesta carpeta del dia actual.

- A més, creem una carpeta anomenada categories dins la carpeta que acabem de crear amb el dia actual.
- Dins la carpeta categories, per cada sub categoria que existeix de la categoria en qüestió, creem una carpeta (amb el nom: *id#nom*) per guardar la seva informació.
- Dins d'aquesta carpeta de subcategoria (*id#nom*), guardem el .json de fer la crida.
- Creem una carpeta subcategories on es guarden totes les subcategories finals (aquest seria el tercer json dels tres anomenats anteriorment).

Bàsicament, cridem tots els endpoints de les categories i guardem els fitxers en format .json (raw) a una carpeta diferent cada un.

El següent pas és passar d'aquesta capa raw, a una capa amb un format més estructurat. (transformem de json a csv).

6.2.1.2 raw to bronze script

- https://github.com/jaumejp/analysis-tiquets/blob/main/generador-tiquets/dades/productes/dataset_propi/DataLake/1-raw_to_bronze.ipynb

En aquesta capa (directori) tindrem per cada dia un fitxer csv amb tota informació ingestada d'aquell dia en concret. Per fer-ho:

- Llegeix totes les carpetes que s'han ingestats de la capa raw.
 - Llista de dies que tenim ingestats.
- Llegeix totes els fitxers que ja existeixen a la capa bronze
 - Llista de dies (que NO hem de procesar).
- Itera per tots els dies de la capa raw i si ja el tenim tractat (llista de la capa bronze), se'l salta.

Comentar, que aquesta lògica de llegir-los tots i saltar-se els que ja existeixen, la podem millorar o canviar tinguent un fitxer en la carpeta on cada cop que s'executa el canvi de capa s'hi modifica el contingut per el dia actual. Així, quan es torna a executar de nou, tenim l'últim dia ingestat i sabem desde on hem d'executar l'script.

- Per cada dia que falta tractar: De la capa raw, llegir-ne la informació i concatenar-la per crear un fitxer amb nom: *dd-mm-yyyy.csv* (dia que es van ingestar els productes) i guardar-lo a la capa bronze.

6.2.1.3 bronze to silver script

- https://github.com/jaumejp/analysis-tiquets/blob/main/generador-tiquets/dades/productes/dataset_propi/DataLake/2-bronze_to_silver.ipynb

En aquesta capa guardem tots els dies en un mateix fitxer csv.

- Mirem si existeix ja un fitxer a la capa silver.
 - En el cas negatiu, llegirem tots els fitxers de la capa bronze i concatenar-los.
 - En el cas positiu:

- Llegir el fitxer, mirarem la última data de ingestà.
- Llegim tots els dies ingestats a la capa bronze en format .csv i concatenar els que faltin.

6.2.1.4 silver to gold script

- https://github.com/jaumejp/analisis-tiquets/blob/main/generador-tiquets/dades/productes/dataset_propi/DataLake/3-silver_to_gold.ipynb

Serveix per deixar les dades el màxim de preparades per llegir-les desde l'script que genera els tiquets.

- Ens quedem només amb les columnes més interessants.
- Reduïm les descripcions dels productes perquè entrin dins el tiquet, ja que n'hi ha de molt grans.
- Convertir aquestes descripcions a majúscula, ja que a els tiquets estan així.
- En funció del producte que és, calculem el preu total.
- Si el preu del producte és més gran que 100, fem que sigui un valor random entre 80 i 100. Així no queden preus super grans.
- Al final, queda així:

Out[17]:	category	subcategory	name	iva	bulk_price	unit_price	units	format	image	price	date
0	Fruita i verdura	Verdura	Tomàquet sec	4	22.38	1.79	NaN	kg	https://prod-mercadona.imgix.net/20190521/49/6...	22.38	07/05/2024
1	Fruita i verdura	Verdura	Branca de tomàquets	4	1.79	1.22	NaN	ud	https://prod-mercadona.imgix.net/20190521/87/6...	1.22	07/05/2024
2	Fruita i verdura	Verdura	Col llombarda	4	1.20	1.39	NaN	kg	https://prod-mercadona.imgix.net/20190521/51/6...	1.20	07/05/2024
3	Fruita i verdura	Verdura	Coliflor	4	1.89	2.78	NaN	kg	https://prod-mercadona.imgix.net/20190521/20/6...	1.89	07/05/2024
4	Fruita i verdura	Verdura	Col de cabdell llisa	4	1.00	1.98	NaN	ud	https://prod-mercadona.imgix.net/20190521/36/6...	1.98	07/05/2024
5	Fruita i verdura	Verdura	Col de cabdell llisa	4	1.30	0.96	NaN	ud	https://prod-mercadona.imgix.net/20190521/46/6...	0.96	07/05/2024
6	Fruita i verdura	Verdura	Col cabdell	4	1.15	1.48	NaN	kg	https://prod-mercadona.imgix.net/20190521/38/6...	1.15	07/05/2024
7	Fruita i verdura	Verdura	Col cabdell	4	1.45	0.97	NaN	kg	https://prod-mercadona.imgix.net/20190521/57/6...	1.45	07/05/2024
8	Fruita i verdura	Verdura	Col llombarda	4	1.45	0.71	NaN	kg	https://prod-mercadona.imgix.net/20190521/58/6...	1.45	07/05/2024
9	Fruita i verdura	Verdura	Pebrot vermell	4	2.29	0.89	NaN	kg	https://prod-mercadona.imgix.net/20190521/10/6...	2.29	07/05/2024

7. Plantilla tiquet Mercadona

Tinguent com a referència alguns tiquets del mercadona:

- https://github.com/jaumejp/analisis-tiquets/tree/main/generador-tiquets/exemples_tiquets/mercadona

I, el que prèviament hem comentat a la secció [Anàlisis del problema](#), hem maquetat una plantilla amb HTML i CSS.

La plantilla final es troba aquí:

- <https://github.com/jaumejp/analisis-tiquets/tree/main/generador-tiquets/templates/mercadona>

Mitjançant la llibreria [jinja2](#), la qual permet passar objectes de python i variables a dins la plantilla per renderitzar-la. Generem el tiquet dinàmicament i tindrem un nou HTML amb tota la informació carregada.

Finalment, amb la llibreria [html2image](#) passarem aquest fitxer HTML a png.

Per explicar com la plantilla està composada i pensada. Explicarem els blocs de codi més interessants o rellevants de la mateixa.

Podem dividir el tiquet en els següents blocs:

1. Capçalera (header).
 - a. Logo.
 - b. Informació.
 - c. Codi de barres.
2. Secció amb productes (main).
 - a. Unitats.
 - b. Descripció.
 - c. Preu unitari.
 - d. Import.
 - e. Total.
3. Secció dels ivas (main).
 - a. Diferents apartats.
4. Peu de pàgina (footer).
 - a. Informació vària de les targets.
 - b. Diferents logos.
 - c. Missatge de devolució de productes



Pel que fa al bloc 1, només hi ha a destacar el codi del codi de barres.

El codi de bares o barcode, està compost per un contingut `<section>` el qual tindrà subcontinguts a dins ``. Aquests estan orientats horitzontalment amb la propietat de CSS `display: flex;` i centrats al mig amb `justify-content: center;`

Els continguts `span` de dins, tenen la classe `.stripe` la qual els hi aplica una altura, amplada, marge esquerre i separació esquerre per defecte. Aquest marge i separació es el que genera l'efecte de barres verticals. Finalment, amb les classes `.s1-4`, fem modificadors perquè tinguin amplades i separacions horitzontals diferents (randoms).

Com hem comentat anteriorment, amb jinja2 se li podem passar paràmetres, llavors nosaltres passarem un array de n valors aleatoris del 1-4 per crear el codi de barres.

```
<section class="mt-10 mb-10 barcode">
    {% for rand in rand_one_to_four %}
        <span class="stripe s-{{ rand }}></span>
    {% endfor %}
</section>
```

`rand_one_to_four` és l'array que li passem a la plantilla per crear el barcode. Aquest array conté valors de 1 al 4 aleatoris, els quals, cadascun representa una mida de barra diferent.

El següent bloc, la secció on es mostren els productes (2). El més destacable es que és una taula posicionada amb posicions absolutes.

Fins ara les posicions dels blocs eren relatives, això significa que es van col·locant un sota l'altre automàticament.

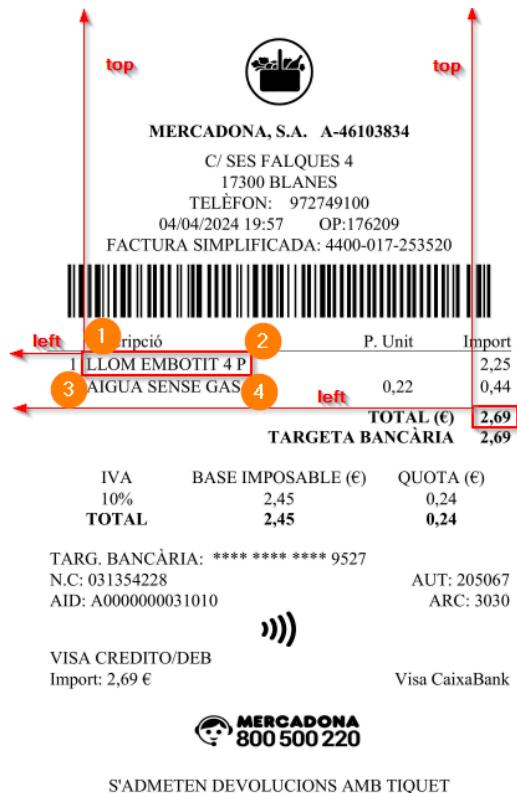
Si amb css, indiquem la propietat: *position:absolute*; li podem posar també les propietats *top* i *left* les quals indiquen a quants px s'ha de situar desde l'extrem superior esquerre.

```
.row-1 {  
    border-bottom: 1px solid black;  
  
    position: absolute;  
    top: 290px;  
    left: 25px;  
  
    width: 450px;  
    height: 20px;  
  
    display: flex;  
    align-items: center;  
}
```

Això indica que la classe *.row-1* estarà posicionada a una altura de 290px de la part superior de la pantalla i 25px a la esquerra.

Això és un aspecte fonamental per generar els tiquets correctament, ja que per tenir els *bounding boxes* (coordenades d'on es troba el text) necessitem la informació d'on estàn posicionats els elements en px.

Per altre banda, també hem fixat la propietat *width* i *height* de cada element per poder localitzar-ne tots els vèrtex del rectangle que envolta cada element.



Per cada element, el bounding box, estarà format per les quatre coordenades següents (x,y):

1.
 - a. x = left
 - b. y = top
2.
 - a. x = left + width del rectangle
 - b. y = top
3.
 - a. x = left
 - b. y = top + height del rectangle
4.
 - a. x = left + width del rectangle
 - b. y = top + height del rectangle

Llavors, per cada element que vulguem identificar, necessitem tenir aquestes 4 característiques (top, left, width, height).

Podríem localitzar tants elements (espai on es representa una informació) com volguessim. Nosaltres ens hem quedat només amb els que representen els productes

1	Quantitat	RINA 2 KG		3,38
1	LLET SEMI	Descripció	Preu unitari ud	5,70
2	AIGUA FONT AGUDES 6		0,75	1,50
1	COLIFLOR	0,972 kg	1,99 €/kg	1,93
1	COGOMB	Preu unitari kg	1,79 €/kg	Import
		0,476 kg		0,85
1	PLÀTAN	Quantitat	1,99 €	Total tiquet
		0,982 kg		1,95
			TOTAL (€)	42,59

Ara passem a examinar i descriure les propietats top, left, width i height de cada un d'aquests elements per posteriorment, poder entendre com buscar-ne els bounding box.

La capçalera d'aquesta taula;

```

<table>
  <thead>
    <tr class="row-1">
      <th class="col-1"></th>
      <th class="col-2">Descripció</th>
      <th class="col-3">P. Unit</th>
      <th class="col-4">Import</th>
    </tr>
  </thead>

```

Descripció	P. Unit	Import

té la classe *row-1* la cual posciona a

- top: 290px
- left: 25px
- width: 450px
- height: 20px

Aquestes mesures, han sigut posades en funció del que hi havia a la capçalera, si hi hagués hagut més contingut, i per tant, tots els components haguessin arribat més avall, hagués sigut un valor més gran que 290.

Si fixem que comencem a 290px i que cada element tindrà una altura (*height*) de 20px, podem calcular a quina altura (*top*) ha d'estar cada element.

Només hauríem de calcular a quina fila ens trobem (n), multiplicar-ho per 20px i sumar-li 290px.

Això ho aconseguim assignant un contador inicialitzat a 1, que itera per tots els productes que li passem (el que comentàvem que amb jinja2 es pot passar la informació a renderitzar) i es va incrementant en 1 si és un producte unitari i 2 si es un producte a pes.

Fins ara tenim:

- height: tots fixes a 20px
- top: el podem calcular.

Només falta les coordenades x les quals necessitarem fixar un left i width per cada un dels sis elements.

Ho fem de la següent manera:

Dividim la secció de productes amb quatre columnes:

	Descripció	P. Unit	Import
1	PATATA XURRERÍA		1,00
1	SUC TARONJA REFRIG.		1,50
1	BURG M BO/POR1000G		6,50
1	TONYINA CLARA OLIV		5,25
1	IOGURT AMB FRUITES		2,10
1	ISOTÒNICA LLIMONA		0,85
1	CEBA TUB		1,85
1	F. RATLLAT S/LACTOSA		1,63
1	PATATA 3 KG		4,30
1	TOMÀQUET TRITURAT		0,75
1	OLI DE GIRA-SOL 1 L		1,55
1	MANDARINA 2 KG		3,38
1	LLET SEMI S/LACT		5,70
2	AIGUA FONT AGUDES 6	0,75	1,50
1	COLIFLOR		
	0,972 kg	1,99 €/kg	1,93
1	COGOMBRE		
	0,476 kg	1,79 €/kg	0,85
1	PLÀTAN		
	0,982 kg	1,99 €/kg	1,95
		TOTAL (€)	42,59
		TARGETA BANCÀRIA	42,59

Per cada fila, si tenim un producte unitari, només necessitem una fila , si és un producte per pes, en necessitarem dues.

Això ho aconseguim posant un condicional per evaluar si es tracta d'un producte unitari ho posem tot en una línia o altrament, ho posem amb dues línies (deixant els espais en blanc on corresponen a la primera línia).

Fixem left i width per cada element:

```
.units {  
    position: absolute;  
    left: 25px;  
  
    width: 40px;  
    height: 20px;  
  
    display: flex;  
    align-items: center;  
}  
.description {  
    position: absolute;  
    left: 65px;  
  
    width: 200px;  
    height: 20px; /* In function of products */  
  
    display: flex;  
    align-items: center;  
}  
.p-unit {  
    position: absolute;  
    left: 307.5px;  
    width: 70px;  
    height: 20px;  
  
    display: flex;  
    align-items: center;  
    justify-content: end;  
}  
.import {  
    position: absolute;  
    left: 430px;  
  
    width: 45px;  
    height: 20px;  
  
    display: flex;  
    align-items: center;  
    justify-content: end;  
}
```

Al codi quedaría així:

```

<tr>
  <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="units">1</td>
  <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="description">{{ product.descripcio }}</td>
  <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="p-unit"></td>
  <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="import"></td>
</tr>

```

Només puntualitzar que quan estem fent un producte de preu al kg i estem fent la segona línia, el preu s'ubica a la columna de descripció i per tant li hem de modificar el width i left perquè encaixi. Li hem donat aquests valors:

```
<td style="top: calc(290px + {{ (count.value *20) | int }}px); left: 99px; width: 166px" class="description">{{ product.quantitat_kg }} kg</td>
```

Per poder localitzar tots els elements amb python, tota aquesta informació, queda recollida en aquests objectes:

```

ud = {
    'quantitat_ud': {'left': 25, 'width': 40},
    'descripcio': {'left': 65, 'width': 200},
    'preu_unitari': {'left': 307.5, 'width': 70},
    'import': {'left': 430, 'width': 45},
}

kg = {
    'quantitat_kg': {'left': 99, 'width': 166},
    'descripcio': {'left': 65, 'width': 200},
    'preu_unitari': {'left': 307.5, 'width': 70},
    'import': {'left': 430, 'width': 45},
}

total = {
    'total': {'left': 430, 'width': 45},
}

```

Els quals són els que es fan servir per calcular les coordenades de cada element (blobs)

```

def _coords(self, format_dict, count, key):
    left = format_dict[key]['left']
    top = self.top_offset + (count *self.item_height)
    width = format_dict[key]['width']

    return {
        '1': (left, top),
        '2': (left + width, top),
        '3': (left, top + self.item_height),
        '4': (left + width, top + self.item_height),
    }

```

Aquesta lògica està més explicada i detallada a la secció que es comenta com s'ha creat el [dataframe per entrenar la xarxa neuronal](#).

Finalment, per la secció dels ivas i el footer. El contenidor general es posiciona de la mateixa manera (fent servir el contador per donar la posició top).

Els elements de dins cada contenidor no se'ls hi aplica aquesta lògica ja que no necessitem tenir-ne la informació (es posiciona automàticament amb les regles generals de CSS).

Si volguessin predir-les amb la xarxa neuronal, hauríem de posicionar-les de la mateixa manera per obtenir-ne els bounding box (coordenades).

Com podem veure en aquesta secció, crear la plantilla d'aquesta manera es fonamental pel nostre projecte ja així podem trobar els bounding boxes i localitzar la informació. Cosa imprescindible perquè la xarxa neuronal aprengui a identificar on està cada element del tiquet.

8. Generació de tiquets

Com hem comentat, el ground truth és molt important perquè serà el que aprengui la xarxa neuronal, nosaltres hem seguit el següent procediment per generar-lo:

La primera part ha sigut explorar com crear un tiquet a partir del [dataframe de productes](#) i la [plantilla](#).

En aquest apartat de la memòria ho explicarem per sobre, però per més detall sobre el codi, es pot visitar el Notebook següent:

- https://github.com/jaumejp/analisis-tiquets/blob/main/generador-tiquets/generador_tiquets_documentacio.ipynb

8.1 Exploració i preparació de les dades

Com que les dades han sigut creades per nosaltres, no hem tingut massa problema. Examinem la llargada del dataset, el tipus de dades, i fem algun petit canvi (que l'iva sigui un string, etc). Està més detallat al Notebook.

Necessitem que l'iva sigui un string

- Perquè en el moment de fer els càlculs de cada preu, es fa servir l'objecte:

```
ivas = {
    '0' : {'d': 0.000, 'i': 1.000},
    '4' : {'d': 0.040, 'i': 1.040},
    '10' : {'d': 0.100, 'i': 1.100},
    '21' : {'d': 0.210, 'i': 1.210},
}
```

- on:
 - Les claus: 0, 4, 10, 21 són els possibles ivas que podem tenir al dataset.
 - Els valors de cada clau són un diccionari per afegir o treure l'iva a un producte.
 - d: decrementar.
 - Si tenim un preu de 100€ iva inclòs, a un iva del 21%. Si dividim $100 * 0.21 = 21$ € que d'aquests 100€ pertanyen a l'iva.
 - Si volem saber el preu sense iva d'aquests 100€, hem de dividir per 1.21.
 - i: incrementar.
 - Si tenim un preu sense iva de 100€ i volem sumar-li l'iva, hem de multiplicar $100 * 1.21 = 121$ €

- Llavors, tinguent aquest objecte, quan calculem els preus amb iva del tiquet o els ivas total que han d'aparèixer al tiquet, accedirem a aquest objecte `ivas` per la clau que pertoqui.

```
In [9]: df['iva'] = df['iva'].astype(str)
df.dtypes
```

```
Out[9]: name          object
category        object
subcategory     object
price           float64
iva             object
format          object
ingested_day   object
dtype: object
```

8.2 Creació del tiquet

8.2.1 Preparació del dataset

Assignem una sèrie de pesos a cada producte en funció de si nosaltres creiem que es ven més o menys. La nostre intuïció ens diu que per categories com: arròs, ous, postres, xarcuteria i congelats es venen més i per tant han de sortir més.

També posem un pes major als productes que van amb €/kg, ja que n'hi ha menys i així es veuràn representats igualment. Si no posem suficients productes, la xarxa neuronal no els aprendrà a diferenciar.

Agafarem només productes del mateix dia perquè el tiquet sigui més coherent.

Canviem la columna price per *p_u_sense_iva*, ja que és la informació que representa.

El dataset final amb tots els productes té aquesta forma:

Out[21]:		name	category	subcategory	p_u_sense_iva	iva	format	ingested_day	category_weight
0	RAÍM NEGRE SENSE	Fruita i verdura	Fruita		2.63	4	ud	16-04-2024	0.225
1	RAÍM NEGRE AMB	Fruita i verdura	Fruita		345.51	4	ud	16-04-2024	0.225

8.2.2 selecció de n productes

Per fer un tiquet, hem de fer una selecció de n productes d'aquest dataframe.

Fem un random sample d'aquest dataset:

```
In [26]: np.random.seed(3)
def shop(n_products):
    # Fem el tiquet del primer dia que tinguem: (dfs[0])
    return dfs[0].sample(n=n_products, weights='category_weight')

tiquet = shop(15)
tiquet
```

Out[26]:		name	category	subcategory	p_u_sense_iva	iva	format	ingested_day	category_weight
36839	BAIETA PAL SUAU	Neteja i llar	Estris de neteja i calçat		0.70	21	ud	16-04-2024	0.010
42075	SECRET DE PORC	Carn		Porc	2.36	10	ud	16-04-2024	0.100
9727	AIGUA MINERAL	Aigua i refrescos		Aigua	0.20	10	ud	16-04-2024	0.100
30922	SORRA PER GAT	Mascotes		Gat	3.00	21	ud	16-04-2024	0.015
44856	FORMATGE RATLLAT	Xarcuteria i formatges		Formatge tall, ratllat i en porcions	1.20	4	ud	16-04-2024	0.100

En aquest cas, `dfs`, és un array de datasets on cada dataset són els productes d'un dia. En el moment de fer l'script final de tiquet, agafarem un dia random d'aquesta llista.

En el Notebook d'exploració, sempre agafem el primer dia i posem una *llavor* al random perquè sempre surti el mateix tiquet.

Creem unitats aleatòries per cada producte

Explicació detallada al Notebook, però a grans trets fem el següent: número entre 1-5 per els productes unitaris (donant més prioritat al 1, ja que creiem que és més comú comprar una unitat que més de una). I, un float aleatori entre 0.1 i 1.5 per els productes €/Kg, que signifiquen els kilograms comprats.

Generem les quantitats de compra de cada producte

- Creació del atribut `quantitat`:
 - Si és producte unitari, agafem una quantitat aleatoria de 1 a 5, assignant un 70% de probabilitats de ser 1 i un 30% de probabilitats de ser 2-5.
 - Això ho fem perquè creiem que normalment sempre es compra 1 unitat de cada producte i menys sovint, més unitats.
- Es crea un array amb els números [1,x] on $x = 2, 3, 4, 5$ i s'escau un dels dos nombres amb la probabilitat especificada anteriorment.
 - Si el format és d'un producte amb pes, agafem un numero aleatori entre 0.1 (significant 100g → 0.1kg) i 1.5 (significant 1500g → 1.5kg) i com que el preu és €/kg , ja tindriem el preu final.

```
round(np.random.uniform(0.1, 1.5), 3)
```

```
In [31]: np.random.seed(2334)

tiquet['qty'] = tiquet.apply(lambda x: round(np.random.uniform(0.1, 1.5), 3) if x.format == 'kg' else np.ra
```

Afegim els preus corresponents

Calculem el preu del producte sense iva multiplicant el preu unitari sense iva per la quantitat aleatòria que hem generat.

Afegim el preu amb iva. Explicació del Notebook:

Afegir preu amb iva (unitari i total)

- Per calcular el preu final de cada producte amb iva s'ha de multiplicar el preu sense iva per l'atribut `'i'` de l'objecte de `ivas` anomenat anteriorment.
 - Si tenim 5€ sense iva i un iva del 10% → $5 * 1.1 = 5.5\text{€}$ amb iva.

Per calcular el preu amb iva, fem el següent proccés:

Com hem vist al principi, necessitem tenir aquest objecte per fer càlculs amb els preus (sumar i restar iva):

Creem l'objecte `ivas`

- Amb tots els ivas del conjunt de dades, ja que pot variar (noves lleis, canvis, etc).
- Fem que cada valor que farem servir per fer els càlculs tingui 3 decimals perquè al tiquet hi ha una presició de 2 decimals i així no perdem informació al fer els arrodoniments.

```
In [33]: df.iva.unique()  
  
Out[33]: array(['4', '10', '21'], dtype=object)  
  
In [34]: ivas = { iva: { 'd': int(iva) / 100, 'i': (int(iva) +100) / 100 } for iva in df.iva.unique() }  
ivas  
  
Out[34]: {'4': {'d': 0.04, 'i': 1.04},  
          '10': {'d': 0.1, 'i': 1.1},  
         '21': {'d': 0.21, 'i': 1.21}}
```

Per cada registre, mirem el preu sense iva i el iva corresponent. Després anem a aquest objecte que hem creat i multipliquem el valor de 'i' per augmentar l'iva.

Calculem el preu unitari amb iva

```
In [36]: tiquet['p_u_amb_iva'] = tiquet.apply(lambda x: x.p_u_sense_iva *ivas[x.iva]['i'], axis=1)
```

Calculem els ivas desglossats del tiquet

Seria aquesta part del tiquet:

IVA	BASE IMPOSABLE (€)	QUOTA (€)
10%	2,45	0,24
TOTAL	2,45	0,24

Aquesta informació, la guardem en l'objecte `ivas_list`, la qual passarem a la template.

```
In [40]: ivas_list = []
def get_total(iva):
    return (tiquet[tiquet['iva'] == iva]['preu_sense_iva'] *ivas[iva]['d']).sum()

for iva in tiquet.iva.unique():
    total = get_total(iva)
    ivas_list.append({
        "valor": f'{iva}%',
        "base_imposable": format(total, ',.2f').replace(".", ","),
        "quota": format(total *ivas[iva]['d'], ',.2f').replace(".", ",")
    })

ivas_list
```

Out[40]: [{}{'valor': '21%', 'base_imposable': '44,01', 'quota': '9,24'}, {}{'valor': '10%', 'base_imposable': '3,48', 'quota': '0,35'}, {}{'valor': '4%', 'base_imposable': '0,47', 'quota': '0,02'}]

```
# Create context per les dades que escriurem
context = {
    'nom': f'MERCADONA, S.A. {random_letter()}{random_num(8)}',
    'adressa': rand_establishment.direccion.values[0],
    'ciutat': f'{rand_establishment.localidad.values[0]} {rand_establishment.codigo_postal.values[0]}',
    'telf': f'{rand_establishment.telefono.values[0]}',
    'data': f'{get_random_date()}',
    'op': f'{random_num(5)}',
    'factura_simplificada': f'{random_num(4)}-{random_num(3)}-{random_num(5)}',
    'rand_one_to_four': [np.random.randint(1, 5) for _ in range(60)],
    'products': products,
    'total_amb_IVA': total_amb_iva,
    'pagat_targeta_bancaria': total_amb_iva,
    'ivas': ivas_list,
    'total_sense_IVA': format(no_iva, ',.2f').replace(".", ","),
    'iva_total': format(iva_total, ',.2f').replace(".", ","),
    'targeta_bancaria': f'***** ***** ***** {random_num(4)}',
    'NC': random_num(9),
    'AUT': random_num(6),
    'AID': f'{random_letter()}00000000{random_num(5)}',
    'ARC': random_num(4),
    'targeta_1': targeta_1,
    'targeta_2': targeta_2,
    'total_amb_IVA': total_amb_iva,
    'logo_bottom': logo_bottom,
    'logo_top': logo_top,
    'logo_contact_less': logo_contact_less,
    'height': container_height,
}
```

context és l'objecte que passarem a la plantilla.

D'aquesta manera, renderitzem l'objecte a la plantilla:

```

<table class="ivas-table w-100" style="width: 100%;">
  <thead>
    <tr class="ivas-title">
      <th>IVA</th>
      <th>BASE IMPOSABLE (€)</th>
      <th>QUOTA (€)</th>
    </tr>
  </thead>
  <tbody>
    {% for iva in ivas %}
    <tr>
      <td>{{ iva.valor }}</td>
      <td>{{ iva.base_imposable }}</td>
      <td>{{ iva.quota }}</td>
    </tr>
    {% endfor %}
    <tr>
      <td><b>TOTAL</b></td>
      <td><b>{{ total_sense_IVA }}</b></td>
      <td><b>{{ iva_total }}</b></td>
    </tr>
  </tbody>
</table>

```

Productes del tiquet

Igual que l'objecte de ivas, necessitem l'objecte *products*, per poder-lo iterar i mostrar-los. Aquesta llista d'objectes, la creem així:

```
In [48]: products = tiquet[['name', 'qty', 'p_u_amb_iva', 'preu_amb_iva', 'format']].rename(columns={
    'name': 'descripcio',
    'qty': 'quantitat',
    'p_u_amb_iva': 'preu_unitari',
    'preu_amb_iva': 'import',
})
products = products.to_dict(orient='records')
products
```

```
Out[48]: [{"descripcio": "BAIETA PAL SUAU",
  "quantitat": "3",
  "preu_unitari": "0,85",
  "import": "2,54",
  "format": "ud"}, {"descripcio": "SECRET DE PORC",
  "quantitat": "1",
  "preu_unitari": "2,60",
  "import": "2,60",
  "format": "ud"}, {"descripcio": "AIGUA MINERAL",
  "quantitat": "2",
  "preu_unitari": "0,22",
  "import": "0,44",
  "format": "ud"}, {"descripcio": "SORRA PER GAT",
  "quantitat": "3",
  "preu_unitari": "3,63",
  "import": "10,89",
```

Fins aquest moment, no diferenciamavem la quantitat_ud i quantitat_kg, tot era quantitat. Quan vam passar a entrenar el model, ens vam adonar que necessitavem separar-ho. Per fer-ho, ho hem fet així:

Hem de separar amb dos classes la quantitat_ud i quantitat_kg

- el `**p` → crea un diccionari igual
- després crem un nou atribut `quantitat_ud` o `quantitat_kg` en funció de si el format és kg o ud amb el mateix valor del previ atribut `quantitat`.
- finalment ens quedem amb tots els atributs menys el de `quantitat` perquè es el que volem substituir per `_ud` o `_kg`

```
In [49]: products = [{**p, 'quantitat_ud' if p['format'] == 'ud' else 'quantitat_kg': p['quantitat']} for p in products]
products = [{k: v for k, v in p.items() if k != 'quantitat'} for p in products]
products
```

```
Out[49]: [{"descripcio": "BAIETA PAL SUAU",
  "preu_unitari": "0,85",
  "import": "2,54",
  "format": "ud",
  "quantitat_ud": "3"}, {"descripcio": "SECRET DE PORC",
  "preu_unitari": "2,60",
  "import": "2,60",
  "format": "ud",
```

Calcular les coordenades

El càlcul de les coordenades de cada element, està implementada a la classe `BlobGenerator.py` i ven explicada al Notebook i a la secció d'explicació dels [datasets de train/test](#).

El dataset amb les coordenades queda així:

In [52]:	tiquets						
Out[52]:	ticket_id	etiqueta	value	b_1	b_2	b_3	b_4
0	000001.png.png	quantitat_ud	3	(25, 310)	(65, 310)	(25, 330)	(65, 330)
1	000001.png.png	descripcio	BAIETA PAL SUAU	(65, 310)	(265, 310)	(65, 330)	(265, 330)
2	000001.png.png	preu_unitari	0,85	(307,5, 310)	(377,5, 310)	(307,5, 330)	(377,5, 330)
3	000001.png.png	import	2,54	(430, 310)	(475, 310)	(430, 330)	(475, 330)
4	000001.png.png	quantitat_ud	1	(25, 330)	(65, 330)	(25, 350)	(65, 350)
5	000001.png.png	descripcio	SECRET DE PORC	(65, 330)	(265, 330)	(65, 350)	(265, 350)
6	000001.png.png	import	2,60	(430, 330)	(475, 330)	(430, 350)	(475, 350)
7	000001.png.png	quantitat_ud	2	(25, 350)	(65, 350)	(25, 370)	(65, 370)
8	000001.png.png	descripcio	AIGUA MINERAL	(65, 350)	(265, 350)	(65, 370)	(265, 370)
9	000001.png.png	preu_unitari	0,22	(307,5, 350)	(377,5, 350)	(307,5, 370)	(377,5, 370)
10	000001.png.png	import	0,44	(430, 350)	(475, 350)	(430, 370)	(475, 370)
11	000001.png.png	quantitat_ud	3	(25, 370)	(65, 370)	(25, 390)	(65, 390)
12	000001.png.png	descripcio	SORRA PER GAT	(65, 370)	(265, 370)	(65, 390)	(265, 390)

També necessitem calcular l'altura del tiquet. Sabent les altures del header i footer (que són fixes) i la dels productes en funció de quants n'hi ha de cada tipus, calculem la altura (ho necessitem per passar de html a png correctament).

```

# blank row
count = count + 1

# Total (€)
tiquets.append({
    'ticket_id': '000001.png',
    'etiqueta': 'total',
    'value': total_amb_iva,
    'blob': coords(total, count, 'total'),
})
count = count + 1

# Targeta bancària
count = count + 1

top = 290 + (count * 20) + 20

# Height del footer => 328px
# Padding top i bottom => 25px

container_height = top + 328 + 25

container_height

```

Per cada element augmentem el contador + 1 (productes unitaris) + 2 €/kg, etc.

8.3 Renderització de la plantilla i generació del tiquet

8.3.1 Creació de la plantilla HTML

En funció de les dades pre processades en els apartats anteriors, amb jinja2, carreguem la template que hem creat i li passem totes les dades que volem que hi surtin.

```
In [74]: targeta_1, targeta_2 = random_card()

# Create context per les dades que escriurem
context = {
    'nom': f"MERCADONA, S.A. {random_letter()}-{random_num(8)}",
    'adressa': rand_establishment.direccion.values[0],
    'ciutat': f'{rand_establishment.localidad.values[0]} {rand_establishment.codigo_postal.values[0]}',
    'telf': f'{rand_establishment.telefono.values[0]}',
    'data': f'{get_random_date()}',
    'op': f'{random_num(5)}',
    'factura_simplificada': f'{random_num(4)}-{random_num(3)}-{random_num(5)}',
    'rand_one_to_four': [np.random.randint(1, 5) for _ in range(60)],
    'products': products,
    'total_amb_IVA': total_amb_iva,
    'pagat_targeta_bancaria': total_amb_iva,
    'ivas': ivas_list,
    'total_sense_IVA': format(no_iva, ',.2f').replace(".", ","),
    'iva_total': format(iva_total, ',.2f').replace(".", ","),
    'targeta_bancaria': f"***** **** * {random_num(4)}",
    'NC': random_num(9),
    'AUT': random_num(6),
    'AID': f'{random_letter()}{random_num(5)}',
    'ARC': random_num(4),
    'targeta_1': targeta_1,
    'targeta_2': targeta_2,
    'total_amb_IVA': total_amb_iva,
    'logo_bottom': logo_bottom,
    'logo_top': logo_top,
    'logo_contact_less': logo_contact_less,
    'height': container_height,
}
```

```
In [76]: # Renderitzar la template
reportText = template.render(context)
reportText[0:100]
```

```
Out[76]: '<!DOCTYPE html>\n<html lang="en">\n<head>\n    <meta charset="UTF-8">\n    <meta name="viewport" content='
```

8.3.2 Creació de la imatge

A partir d'aquest html, amb la llibreria html2Image, creem una captura de pantalla del html.

```
In [79]: # Ho convertim a imatge (fer-ho directament després)
from html2image import Html2Image

htmlimg = Html2Image()

htmlimg.output_path = "./exports/mercadona/"

filesize = (500, container_height)

htmlimg.screenshot(
    html_str=html_content,
    save_as="export_1.png",
    size=filesize
)
```

9. Dataset *train/test*

Com hem comentat a la [introducció](#), quan programem en intel·ligència artificial no donem un seguit d'instruccions al programa, sinó que anem ensenyant casos de mostra/prova i el model (algoritme que va ajustant els pesos) es va entrenant per solucionar aquell problema en concret. Per tant, hem de definir diferents estructures per tenir les dades preparades per assolir-ho.

Per entrenar una xarxa neuronal de detecció de imatges, necessitem tres coses de cada tiquet o imatge:

- Bounding-box (coordenades en píxels) de cada una de les zones on hi ha informació rellevant.
- Etiqueta identificant de quin tipus d'informació es tracta (unitats, descripció, quantitat unitats, quantitat kg, import i total).
- El contingut de la informació (text que hi ha dins aquest bounding box).

9.1 Dataset entrenament reconeixement d'imatges

Com hem comentat en la secció de [generació del script](#). Tenim una llista de productes la qual passarem a la plantilla i aquests seran pintats allà amb unes coordenades concretes gràcies al CSS.

```
In [48]: products = tiquet[['name', 'qty', 'p_u_amb_iva', 'preu_amb_iva', 'format']].rename(columns={  
    'name': 'descripcio',  
    'qty': 'quantitat',  
    'p_u_amb_iva': 'preu_unitari',  
    'preu_amb_iva': 'import',  
})  
  
products = products.to_dict(orient='records')  
  
products
```



```
Out[48]: [{"descripcio": "BAIETA PAL SUAU",  
           "quantitat": "3",  
           "preu_unitari": "0,85",  
           "import": "2,54",  
           "format": "ud"},  
          {"descripcio": "SECRET DE PORC",  
           "quantitat": "1",  
           "preu_unitari": "2,60",  
           "import": "2,60",  
           "format": "ud"},  
          {"descripcio": "AIGUA MINERAL",  
           "quantitat": "2",  
           "preu_unitari": "0,22",  
           "import": "0,44",  
           "format": "ud"},  
          {"descripcio": "SORRA PER GAT",  
           "quantitat": "3",  
           "preu_unitari": "3,63",  
           "import": "10,89",  
           "format": "ud"}]
```

Tal com expliquem a la secció de [generació de la plantilla](#). Tenim una sèrie de bucles i condicionals que van augmentant un contador per donar la propietat de css top. Juntament amb la propietat height (serien la coordenada y). Per la coordenada x, tenim les propietats de css left i width.

A termes generals, el bucle és el següent:

```
{% set count = namespace(value=1) %}
{% for product in products %}
    {% if product.format == "ud" %}
        <tr>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="units">{{ product.quantitat_ud }}</td>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="description">{{ product.descripcio }}</td>
            {% if product.quantitat_ud == '1' %}
                <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="p-unit"></td>
            {% else %}
                <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="p-unit">{{ product.preu_unitari }}</td>
            {% endif %}
                <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="import">{{ product.import }}</td>
        </tr>
        {% set count.value = count.value + 1 %}
    {% else %}
        <tr>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="units">1</td>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="description">{{ product.descripcio }}</td>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="p-unit"></td>
            <td style="top: calc(290px + {{ (count.value *20) | int }}px)" class="import"></td>
        </tr>
        {% set count.value = count.value + 1 %}
    {% endif %}
    {% endfor %}
```

Inicialitzem el contador en 1, ja que al fer el primer element, la propietat top és: 290px + 20px. (perquè 290px és la posició de la capçalera de la taula).

Si el producte és unitari, només hem d'aumentar el contador en 1. Altrament, si és un producte a pes, tindrem dues línies, per tant les hem de fer per separat i sumar 2.

Un cop hem pintat tots els productes, hi ha més espais que hem de considerar:

- Espai en blanc
- Total

Cada cop que fem una fila de la taula, hem d'incrementar el contador en 1 perquè el càlcul de: style="top: calc(290px + {{ (count.value *20) | int }}px) sigui consistent i correcte.

Llavors, entenent això, podem reproduir el bucle que es fa a la plantilla amb python per obtenir totes les coordenades per tenir-les codificades.

Iterem per cada objecte de la llista products:

```
[{'descripcio': 'BAIETA PAL SUAU',
 'preu_unitari': '0,85',
 'import': '2,54',
 'format': 'ud',
 'quantitat_ud': '3'},
 {'descripcio': 'SECRET DE PORC',
 'preu_unitari': '2,60',
 'import': '2,60',
 'format': 'ud',
 'quantitat_ud': '1'},
 {'descripcio': 'AIGUA MINERAL',
 'preu_unitari': '0,22',
 ...]
```

Per cada element de la llista, mirem si és producte unitari o kg (atribut format), en funció d'això, sabem que tindrà un width i left específic (coordenada x), per la coordenada y, ho calcularem en funció del counter.

En funció de si es producte unitari o producte pes, iterem per l'objecte *ud* o *kg*, els quals guarden totes les propietats que necessitem i per cada un, cridem la funció *coords()*, la qual en calcula els bounding boxes.

```
for product in products:
    if product['format'] == 'ud':
        for key in ud.keys():
            tiquets.append({
                'ticket_id': '000001.png',
                'etiqueta': key,
                'value': product[key],
                'blob': coords(ud, count, key),
            })
            count = count + 1
    else:
        for key in kg.keys():
            tiquets.append({
                'ticket_id': '000001.png',
                'etiqueta': key,
                'value': product[key],
                'blob': coords(kg, count if key == 'descripcio' else count +1, key),
            })
            count = count + 2
```

El tret rellevant d'aquesta part del codi és que els objectes *ud* i *kg*, tenen les mateixes claus que els objectes dins la llista *products*:

```

ud = {
    'quantitat_ud': {'left': 25, 'width': 40},
    'descripcio': {'left': 65, 'width': 200},
    'preu_unitari': {'left': 307.5, 'width': 70},
    'import': {'left': 430, 'width': 45},
}

kg = {
    'quantitat_kg': {'left': 99, 'width': 166},
    'descripcio': {'left': 65, 'width': 200},
    'preu_unitari': {'left': 307.5, 'width': 70},
    'import': {'left': 430, 'width': 45},
}

[{'descripcio': 'BAIETA PAL SUAU',
  'preu_unitari': '0,85',
  'import': '2,54',
  'format': 'ud',
  'quantitat_ud': '3'},
 {'descripcio': 'SECRET DE PORC',
  'preu_unitari': '2,60',
  'import': '2,60',
  'format': 'ud',
  'quantitat_ud': '1'},
 {'descripcio': 'AIGUA MINERAL',
  'preu_unitari': '0,22',
  'import': '0,22',
  'format': 'kg',
  'quantitat_kg': '1'}]

```

Per tant, quan fem el bucle `for key in ud.keys():`: cada clau dels diccionaris ud i kg, correspon també amb les claus dels objectes de `products`. Llavors, podem crear l'objecte que conté les coordenades fàcilment.

A la funció `coords`, se li pasa el diccionari i la clau que estem examinant i va a buscar les coordenades per calcular cada bounding box als objectes comentats anteriorment.

```

def coords(d, count, key):
    left = d[key]['left']
    height = 20
    top = 290 + (count * 20)
    width = d[key]['width']

    return {
        '1':(left, top),
        '2':(left + width, top),
        '3':(left, top + height),
        '4':(left + width, top + height),
    }

```

Finalment, només destacar, que ho estem guardant tot en una llista anomenada `tiquets` a la qual hi anem afegint objectes amb tota la informació. Només caldrà separar el camp `blob` en tots els diferents objectes que ha creat a dins ('1', '2', '3', '4').

Separem els blobs en tots els vèrtex que hi ha

```
tiquets[['b_1', 'b_2', 'b_3', 'b_4']] = tiquets.blob.apply(lambda b: pd.Series([b['1'], b['2'], b['3'], b['4']]))
tiquets.drop('blob', axis=1, inplace=True)
tiquets
```

In [52]:

tiquets

Out[52]:

	ticket_id	etiqueta	value	b_1	b_2	b_3	b_4
0	000001.png.png	quantitat_ud	3	(25, 310)	(65, 310)	(25, 330)	(65, 330)
1	000001.png.png	descripcio	BAIETA PAL SUAU	(65, 310)	(265, 310)	(65, 330)	(265, 330)
2	000001.png.png	preu_unitari	0,85	(307,5, 310)	(377,5, 310)	(307,5, 330)	(377,5, 330)
3	000001.png.png	import	2,54	(430, 310)	(475, 310)	(430, 330)	(475, 330)
4	000001.png.png	quantitat_ud	1	(25, 330)	(65, 330)	(25, 350)	(65, 350)
5	000001.png.png	descripcio	SECRET DE PORC	(65, 330)	(265, 330)	(65, 350)	(265, 350)
6	000001.png.png	import	2,60	(430, 330)	(475, 330)	(430, 350)	(475, 350)
7	000001.png.png	quantitat_ud	2	(25, 350)	(65, 350)	(25, 370)	(65, 370)
8	000001.png.png	descripcio	AIGUA MINERAL	(65, 350)	(265, 350)	(65, 370)	(265, 370)
9	000001.png.png	preu_unitari	0,22	(307,5, 350)	(377,5, 350)	(307,5, 370)	(377,5, 370)

Finalment, ho hem posat tot el codi en una classe BlobGenerator.py

- https://github.com/jaumejp/analisis-tiquets/blob/main/generador-tiquets/script_generacion/clases/BlobGenerator.py

Per buscar les coordenades dels productes només tinguent en compte el quadre on estan, s'ha de reaprofitar el codi perquè el concepte és el mateix.

```
In [90]: # Calcular nous blobs de tots els productes (eliminant la part superior i inferior)
# el count ha de començar a 0, el top_offset també a 0 i hem de restar 25 a cada left
```

```
In [91]: ud_cropped = {
    'quantitat_ud': {'left': 25-25, 'width': 40},
    'descripcio': {'left': 65-25, 'width': 200},
    'preu_unitari': {'left': 307.5-25, 'width': 70},
    'import': {'left': 430-25, 'width': 45},
}

kg_cropped = {
    'quantitat_kg': {'left': 99-25, 'width': 166},
    'descripcio': {'left': 65-25, 'width': 200},
    'preu_unitari': {'left': 307.5-25, 'width': 70},
    'import': {'left': 430-25, 'width': 45},
}

total_cropped = {
    'total': {'left': 430-25, 'width': 45},
}

# Instància
ticket_generator = BlobGenerator(initial_count=0, top_offset=0, ud=ud_cropped, kg=kg_cropped, total=total_cropped)

tiquets = ticket_generator.generate_blobs(products, '001')
```

```
In [92]: tiquets.head(10)
```

	ticket_id	etiqueta	value	b_1	b_2	b_3	b_4
0	001.png	quantitat_ud	3	(0, 0)	(40, 0)	(0, 20)	(40, 20)
1	001.png	descripcio	BAIETA PAL SUAU	(40, 0)	(240, 0)	(40, 20)	(240, 20)

Com comentem en la última secció del [notebook](#), creiem que podríem necessitar més tipus de datasets, per això també vam generar els següents:

- Detecció de diferents zones:
 - https://github.com/jaumejp/analisis-tiquets/tree/main/generador-tiquets/script_general/exports/data
 - **bbx_area_products_in_ticket.csv**: Dataset que hem explicat, coordenades dels productes respecte tot el tiquet.

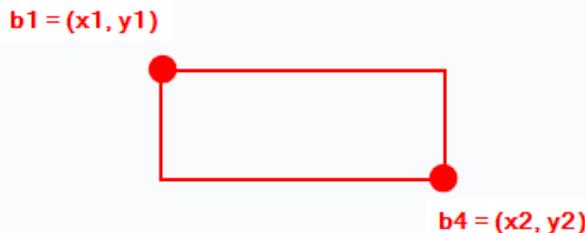


- **bbx_products_in_area.csv**: Dataset amb les coordenades de només la secció dels productes respecte tot el tiquet.

Tot aquest procés ha sigut per generar les coordenades amb format COCO, per entrenar el model les necessitem amb format YOLO.

9.1.1 Coco

El format coco significa tenir les coordenades així:



Exemple:

1	ticket_id	etiqueta	value	b_1	b_2	b_3	b_4
2	000001	quantitat_ud	1	(25, 310)	(65, 310)	(25, 330)	(65, 330)
3	000001	descripcio	DESODORANT CURA	(65, 310)	(265, 310)	(65, 330)	(265, 330)
4	000001	import	1,69	(430, 310)	(475, 310)	(430, 330)	(475, 330)
5	000001	quantitat_ud	1	(25, 330)	(65, 330)	(25, 350)	(65, 350)
6	000001	descripcio	ACTIVADOR	(65, 330)	(265, 330)	(65, 350)	(265, 350)
7	000001	import	2,04	(430, 330)	(475, 330)	(430, 350)	(475, 350)
8	000001	quantitat_ud	3	(25, 350)	(65, 350)	(25, 370)	(65, 370)
9	000001	descripcio	MIG CONILL	(65, 350)	(265, 350)	(65, 370)	(265, 370)

9.1.2 Yolo

Aquest format consisteix en tenir el bounding box de la següent manera:

- label_id, x_center, y_center, width, height

on:

- label_id és quina etiqueta tenim (descripció, import, ...)
- x_center i y_center són les coordenades del centre del bounding box (rectangle).
- width i height són l'ample i llarg del bbox.

Tot amb format relatiu (només les mesures, el label_id evidentment no) al tiquet, es a dir, haurem de dividir aquestes mesures per la amplada i altura del tiquet i expressar-ho en un fitxer .txt

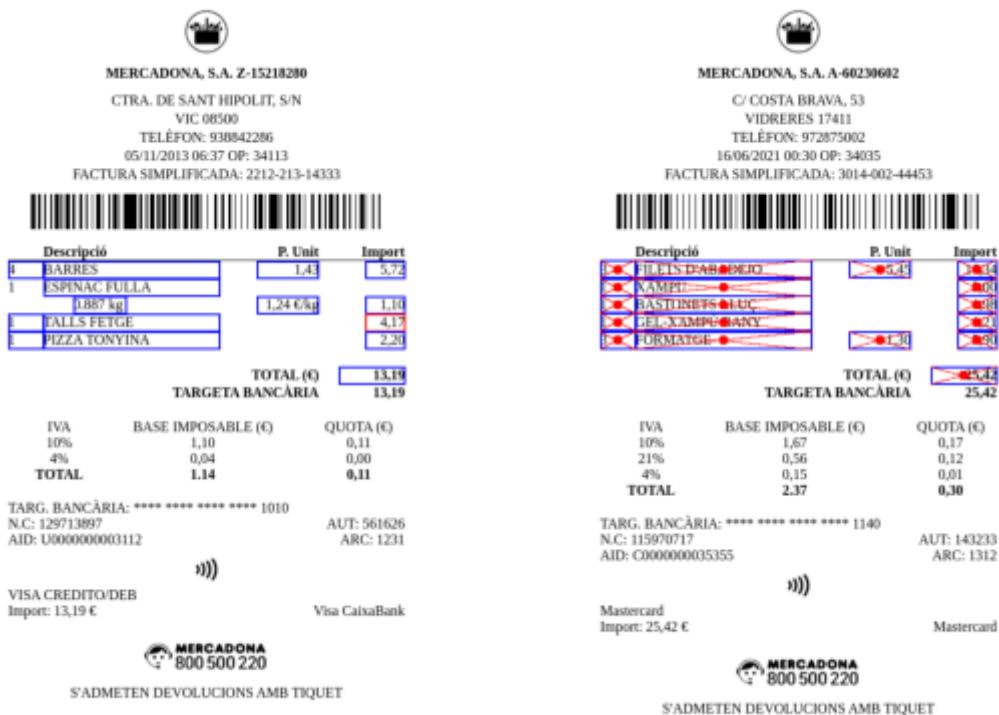
Exemple:

```
1 0.09 0.37959667852906287 0.08 0.02372479240806643
0 0.33 0.37959667852906287 0.4 0.02372479240806643
3 0.905 0.37959667852906287 0.09 0.02372479240806643
1 0.09 0.4033214709371293 0.08 0.02372479240806643
0 0.33 0.4033214709371293 0.4 0.02372479240806643
3 0.905 0.4033214709371293 0.09 0.02372479240806643
```

9.3 Visualització i comprovació dels tiquets

- https://github.com/jaumejp/analisis-tiquets/blob/main/ocr-tiquets/bbox_detection/data_set_exploration.ipynb

COCO vs YOLO



En aquest apartat vam tenir bastants problemes perquè teniem un bug al script que no guardava bé les dades. Això ens va fer recordar el que sempre es comenta d'entendre bé cada línia de codi. Cosa que hem intentat fer en tot moment al projecte fent els Notebooks pas a pas i explicant al màxim cada tros de codi i el perquè.

9.3 Dataset reconeixement de text

Un cop fem la inferència de les coordenades del tiquet, necessitarem saber que hi posa a dins. Per fer aquesta part agafem un tiquet i en retallem tots els elements per els seus bounding boxes i els guardem (retalls que tenen cada informació).

Ex: 1, BARRES, ESPINAC FULLA, 0.887€/kg, etc. Aquest Notebook explicatiu es pot trobar a:

- https://github.com/jaumejp/analisis-tiquets/blob/main/ocr-tiquets/text_detection/test_text_esseract.ipynb

A grans trets, per cada bbox, fem un retall i el guardem en disc, al mateix temps, afegim una columna al dataframe on està tota la informació amb la id del retall (nom amb què el guardem):

```
In [4]: image_root = '../../../../../generador-tiquets/script_general/exports/data/complexity_5/coco/tickets/'

def crop_elements(row, img):
    cropped_img = img.crop((row.b_1[0], row.b_1[1], row.b_4[0], row.b_4[1]))
    image_name = f"cut_{row.name}"
    cropped_img.save(f'./cropped_tiquets/{image_name}.png')

    return image_name

for idx, (ticket_id, grouped_df) in enumerate(df.groupby('ticket_id')):
    img = Image.open(f'{image_root}/{ticket_id}.png')

    # Crea tots els noms d'aquell tiquet en un array
    names = grouped_df.apply(lambda row: crop_elements(row, img), axis=1)

    # Afegeix aquest array a la columna cutted_img_name al ticket_id que correspon al grup
    condition = df['ticket_id'] == ticket_id
    df.loc[condition, 'cutted_img_name'] = names

df.head()
```

```
Out[4]:   ticket_id  etiqueta      value    b_1    b_2    b_3    b_4  cutted_img_name
0     000001  quantitat_ud        1  (25, 310)  (65, 310)  (25, 330)  (65, 330)       cut_0
1     000001    descripcio  SALSITXES AMB  (65, 310)  (265, 310)  (65, 330)  (265, 330)       cut_1
2     000001      import        0,79  (430, 310)  (475, 310)  (430, 330)  (475, 330)       cut_2
3     000001  quantitat_ud        2  (25, 330)  (65, 330)  (25, 350)  (65, 350)       cut_3
4     000001    descripcio  IOGURT NATURAL  (65, 330)  (265, 330)  (65, 350)  (265, 350)       cut_4
```

10. Xarxes Neuronals

El model principal que hem estat treballant és YOLO (you only look once) per detectar on estan els elements del tiquet (bounding boxes) i la etiqueta corresponent per saber de que es tracta.

Exemple; a grans trets el que ens retorna le model és:

- A la coordenada (x1, y1, x2, y2) hi ha una descripció.
- A la coordenada (x1, y1, x2, y2) hi ha una altre descripció.
- A la coordenada (x1, y1, x2, y2) hi ha un preu unitari.
- A la coordenada (x1, y1, x2, y2) hi ha un import.
- A la coordenada (x1, y1, x2, y2) hi ha una altre descripció.
- A la coordenada (x1, y1, x2, y2) hi ha un altre import.
- ...
- Així amb tot els elements que hagim detectat.

Un cop detectades les coordenades i etiquetes del que tenim, falta saber que hi posa a dins. Per fer això, hem fet servir el model de Tesseract, que segons hem estat investigant i comprovat al notebook de la secció [anterior](#), és el millor per detectar text.

Finalment, hem fet servir el model gTTS per convertir el text de tots els productes a veu per poder-lo reproduir amb so.

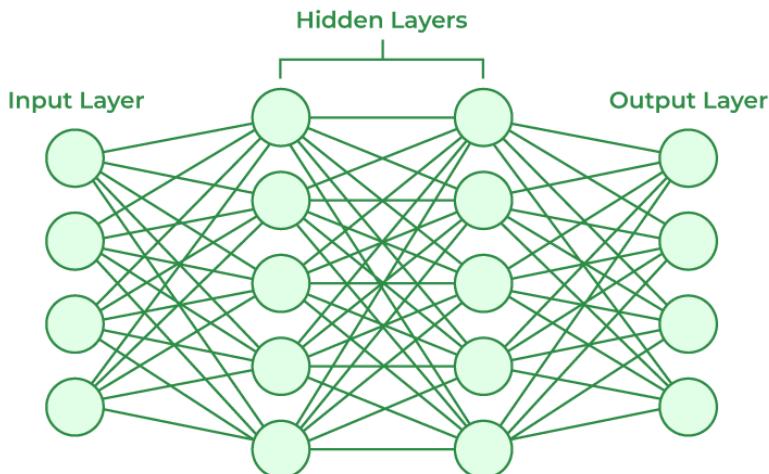
10.1 Fonaments Teòrics

Les xarxes neuronals intenten simular com aprenen les persones:

- Quan un nen petit veu un gos per primera vegada en una imatge, no sap que és. I, a mesura d'anar-ne veient de diferents i examinar les característiques que el composen, al final sap diferenciar-lo d'altres animals.

El mateix procés seguim amb les xarxes neuronals:

- Ensenyem una sèrie de casos de exemple a la xarxa neuronal per ajustar-ne els pesos i biaixos. Per fer-ne una idea bàsica, seria com una regressió lineal però amb multitud de dimensions.



En primer lloc, hi ha una sèrie de neurones connectades entre si. Les connexions poden ser més o menys potents en funció del valor del pes de la conxció que tenen. A més, poden estar més o menys activades en funció del biaix que se li aplica a la neurona.

El procés comença codificant el input que li volem passar, per exemple, si tractem imatges, a cada neurona podríà entrar un píxel (el valor 0-255). Si estiguéssim examinant imatges en color es complicaria, però per mantenir l'exemple fàcil, imaginem una imatge en blanc i negre.

Un cop entra cada input a la primera capa de neurones, es combinen tots els valors d'entrada amb els pesos que van connectats a la neurona següent, després s'aplica el biaix i una funció d'activació (que serveix per passar de una combinació lineal a una no lineal). Aquest procés es va repetint fins arribar al final de la xarxa neuronal. Això s'anomena fit-forward.

Al final examinarem el valor que ens dona i anirem ajustant els pesos i biaixos per millorar la sortida. Inicialment, les connexions entre neurones són inventades i, per tant, la xarxa no sap predir res. Però a mida d'anar-li ensenyant imatges, i ajustant les connexions, el cervell comença a funcionar i amb suficients imatges i temps d'entrenament pot arribar a funcionar molt bé. Aquest procés invers s'anomena backpropagation.

Cal destacar que no es comença amb valors (pesos i biaixos) aleatoris. Es comença amb valors que ha estat pre entrenats. Ho expliquem amb la següent analogia:

- Si volem ensenyar a una persona a llegir radiografies, que serà millor? Tenir una persona que no sap res de medicina però sap entendre el món real (sap detectar patrons, com es componen les formes, etc). O, un bebe recent nascut que no sap interpretar res? Si especialitzem al bebe només a representar radiografies, ens costarà molt més que ho aprengui i que ho faci bé. Però si ho aconseguim, al final segurament ho farà millor que el nostre humà adult que prèviament ja sabia interpretar formes, patrons, etc.

10.1.1 YOLO

Hem utilitzat YOLO com ha xarxa neuronal per identificar les dades del tiquet, és un model de visió per ordinador que permet detectar objectes i segmentar imatges en temps real. Utilitza una xarxa neuronal convolucional per analitzar una imatge i identificar diferents objectes que hi apareixen, com cotxes, persones o animals.

El que fa especial a YOLO és la seva capacitat per detectar múltiples objectes en una sola passada en el nostre cas detectem multiples elements com descripció, import, preu unitari, etc, cosa que el fa molt eficient i adequat per a aplicacions en temps real, com ara la conducció autònoma, la vigilància o la interacció amb els usuaris en aplicacions de realitat augmentada.

A més, proporciona coordenades precises dels objectes detectats, el que el fa molt útil per a tasques de seguiment i localització.

10.1.2 Tesseract

Quan tenim els bbox de cada camp del tiquet predictos amb el model YOLO, fem servir el model Tesseract per identificar el valor de cada camp. Tallem per el tiquet per les coordenades que ens ha indicat YOLO i així obtenim només la zona que volem identificar.

Hem realitzat diverses proves abans de posar-ho en 'producció', i hem observat que el model no detecta els valors enters únics. Per solucionar-ho, hem provat diverses coses com treure el zoom, centrar els números, etc. Finalment, hem buscant més informació sobre el model, hem descobert que hi ha diferents configuracions. Concretament, 14 (PSM 0 a PSM13).

Cada una configuració està orientada a un tipus de reconeixement diferent, a continuació deixem una petita descripció de cada una.

Nosaltres hem utilitzat la 7, ja que li donem només una única línia de text. I en aquesta línia hi pot trobar text o números.

- 0: Orientació i detecció de script (OSD) només.
- 1: Segmentació automàtica de pàgines amb OSD.
- 2: Segmentació automàtica de pàgines sense OSD ni OCR.
- 3: Segmentació totalment automàtica de pàgines sense OSD.
- 4: Suposa una única columna de text de mides variables.
- 5: Suposa un únic bloc uniforme de text alineat verticalment.
- 6: Assumeix un únic bloc uniforme de text.
- 7: Tracta la imatge com una única línia de text.**
- 8: Tracta la imatge com una única paraula.
- 9: Tracta la imatge com una única paraula dins d'un cercle.
- 10: Tracta la imatge com un únic caràcter.
- 11: Recerca de text dispers. Trobar la major quantitat de text possible sense un ordre específic.
- 12: Recerca de text dispers amb OSD.

13: Tracta el text com una única línia, sense utilitzar mètodes específics de Tesseract.

10.2 Entrenament de les xarxes neuronals

Per aquest projecte, hem hagut de fer servir dos tipus de *approach* o maneres d'afrontar el problema.

Per la detecció de imatges, hem partit del model YOLO ja pre entrenat, però l'hem hagut d'ajustar amb les nostres imatges perquè aprengués a solucionar el nostre en concret. Això se'n diu fer un *fine tuning* o ajustament fi. Consisteix bàsicament en que la xarxa ja ha sigut pre entrenada amb moltes imatges i ja ha après a detectar els patrons comuns del nostre món (tots els pesos i biaixos interiors ja s'han ajustat) i nosaltres el que fem es ajustar els pesos i biaixos finals (de la última capa) perquè s'ajusti al nostre problema (detectar x elements del tiquet).

Per altre banda, per fer l'extracció de text de les imatges (saber que hi posa) i convertir el text a so, hem fet servir els models tal com venen per defecte.

10.2.1 Detecció Imatges (YOLO)

Com és normal, primer hem hagut de generar totes les dades d'entrenament. Això és el que fa el script generador de tiquets.

Després, hem fet el fine tuning amb aquestes imatges i això ens ha generat uns nous pesos i biaixos del model raw.

Per fer-ho, necessitem tenir el següent fitxer de configuració:



```
Code Blame 17 lines (14 loc) • 319 Bytes

1 # dataset root dir (absolute)
2 path: /home/jaume/tiquets/ocr-tiquets/bbox_detection/data_yolo/
3
4 # train images (relative to 'path')
5 train: images/train
6
7 # val images (relative to 'path')
8 val: images/val
9
10 # classes
11 names:
12   0: descripcio
13   1: quantitat_ud
14   2: quantitat_kg
15   3: import
16   4: total
17   5: preu_unitari
```

10.2.2 Detecció Text (Tesseract)

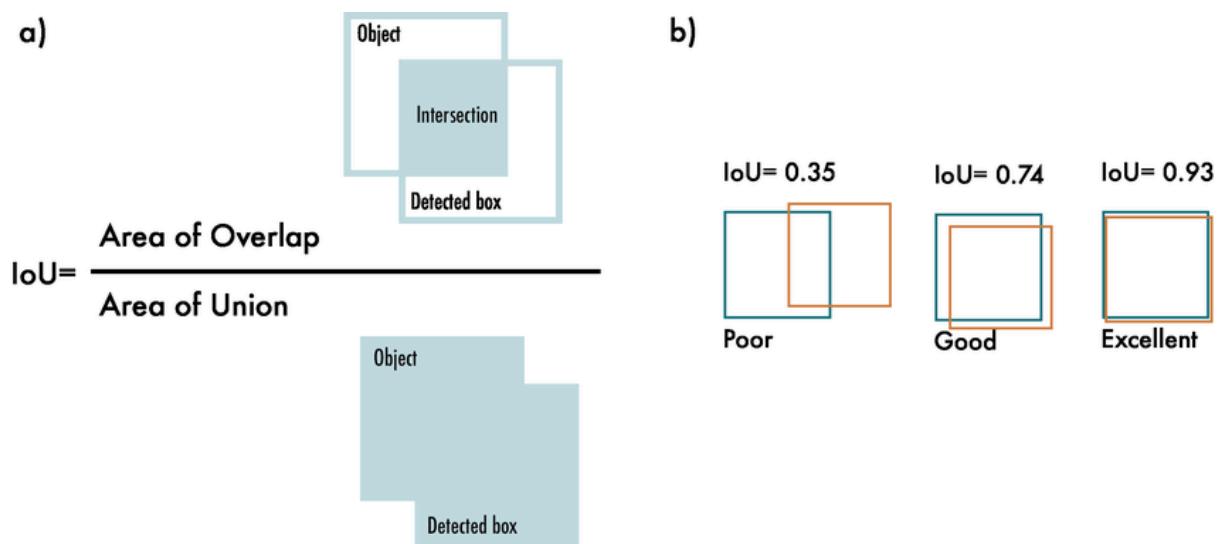
En aquest cas no hem de fer entrenament en si. Simplement apliquem els mètodes de la llibreria que ja estan implementats.

10.3 Test de les xarxes neuronals

10.3.1 Detecció Imatges (YOLO)

Primer de tot, per evaluar el model necessitem fer inferència amb tiquets de test (tiquets que el model no hagi vist prèviament). En el nostre cas és molt fàcil perquè simplement hem de generar nous tiquets aleatoris.

Un cop feta la inferència el model ens retorna una llista d'elements que ha trobat i a quines coordenades. Nosaltres podem evaluar com de bona és aquesta predicció amb l'anomenat: IoU (intersection over union). A grans trets és la divisió de la àrea que forma la intersecció entre l'àrea de les dues parts. Per tant, com més proper sigui el IoU a 1, millor serà la detecció.



Per relacionar cada detecció, farem servir el hungarian algorithm implementat amb el mètode *linear sum assignment* de SciPy de Python, el qual l'implementa de forma fàcil.

Aquest mètode bàsicament, resol el problema d'assignació de variables de forma polinòmica. El qual té un millor rendiment que mirar tots amb tots, que seria un algoritme quadràtic.

Per fer-ho, creem una matriu de IoU on hi ha el IoU per totes les possibles combinacions entre les combinacions reals i les predites. Finalment, restem 1 a la matriu (valor màxim teòric que pot aparèixer a la matriu) per convertir el problema en una assignació de mínims i no de màxims i el mètode ens retornarà quin index va amb quin.

Es pot veure implementat aquí:

- https://github.com/jaumejp/analisis-tiquets/blob/main/ocr-tiquets/bbox_detection/3_evaluate_model.ipynb

Un altre problema que ens trobem es saber quins productes van amb quins. És a dir, la xarxa ens haurà dit el que hi ha, però nosaltres volem saber concretament quin import va amb cada descripció, juntament amb el preu unitari. Per solucionar-ho també ho hem fet amb el linear sum assignment (en general podem dir que es fa minimitzant per la altura de les y, però ha sigut un procés una mica més complex. Aquest algoritme i la seva explicació es pot trobar a:

- https://github.com/jaumejp/analisis-tiquets/blob/main/ocr-tiquets/bbox_detection/4_relate_products.ipynb

Finalment, hem analitzat els valors de les inferències del primer model entrenat. Hem trobat els següents problemes:

- Em pogut veure que al dataframe predit hi ha varis valors més comparant-ho amb el dataframe real.
 - Si entrem més en detall hem vist que aquests camps extres són els tiquets que tenen productes que són de unitats kg.
- Ho hem revisat en detall i no es tracta de una línia més sinó que les unitats dels preus en kg, els detecta com descripcions.

A la imatge següent, hem pintat els bbox predictius, es pot veure que el 1.024 kg té la mateixa mida que les descripcions, per això és normal que la xarxa neuronal es confongui i ho detecti malament.



bbox predictius

1	ESPINACS TALLATS		
	0,824 kg	3,46 €/kg	2,85
TOTAL (€)			2,85

bbox usats per entrenar el model

Solució

- Afegirem al set de train més tiquets amb productes de €/kg. (No funciona)
- Acotarem la mida del bbox perquè es pugui diferenciar de les descripcions.

	descripcio	0	0	0	0	0
True label	2162	0	0	0	0	0
	import	0	1883	0	0	0
	preu_unitari	0	0	831	0	0
	quantitat_kg	0	0	0	249	0
	quantitat_ud	0	0	0	0	1913
	total	0	279	0	0	121
Predicted label						

Veiem que efectivament quan es un total es pensa que es un import i viceversa.

Al Notebook hi ha el codi de comprovació de que els tiquets més llargs són els que més encerta i els curts els que menys, però comentem els resultats aquí també ja que al millorar el model, després els outputs de les execucions han canviat.

Productes que hi ha en els tiquets que la etiqueta import i total ha sigut ben diferenciada:

- Per saber el total de productes, sumem totes les descripcions que hi ha en el df original.
- Agrupem el df original per ticket_id (on hi ha tots els registres de tots els tiquets per separat df).
- Si el ticket_id està en la llista dels tiquets correctes, sumem tots els productes que hi havia en aquell tiquet.

```
In [95]: # No ho fem amb dict comprehension per no sobrecomplicar la sintaxis.
total_productes_tickets = []
for ticket_id, grouped_df in df.groupby('ticket_id'):
    if ticket_id in df_correct.ticket_id.unique():
        total_productes_tickets.append({
            "ticket_id": ticket_id,
            "products": grouped_df['etiqueta'].value_counts()['descripcio']
        })

# Mirem quants quants cops tenim 'x' productes:
count = {}

for n_productes in total_productes_tickets:
    total = n_productes['products']
    if total in count:
        count[total] += 1
    else:
        count[total] = 1

ordenat = dict(sorted(count.items(), key=lambda item: item[1], reverse=True))

# Imprimir el recuento de cada cantidad de productos
for qty, cops in ordenat.items():
    print(f"Número de productes: {qty}, Surt: {cops}")
```

Número de productes: 9, Surt: 29
Número de productes: 10, Surt: 23
Número de productes: 8, Surt: 19
Número de productes: 7, Surt: 15
Número de productes: 6, Surt: 14
Número de productes: 2, Surt: 8
Número de productes: 5, Surt: 6
Número de productes: 1, Surt: 3
Número de productes: 3, Surt: 2
Número de productes: 4, Surt: 2

Veiem com si que la majoria que estan bé, tenen més productes.

Ara mirem els que estan malament i hauria de ser al revés.

```
In [97]: total_productes_tickets = []
for ticket_id, grouped_df in df.groupby('ticket_id'):
    if ticket_id in df_wrong.ticket_id.unique():
        total_productes_tickets.append({
            "ticket_id": ticket_id,
            "products": grouped_df['etiqueta'].value_counts()['descripcio']
        })
# Mirem quants quants cops tenim 'x' productes:
count = {}

for n_productes in total_productes_tickets:
    total = n_productes['products']
    if total in count:
        count[total] += 1
    else:
        count[total] = 1

ordenat = dict(sorted(count.items(), key=lambda item: item[1], reverse=True))

# Imprimir el recuento de cada cantidad de productos
for qty, cops in ordenat.items():
    print(f"Número de productes: {qty}, Surt: {cops}")
```

Número de productes: 2, Surt: 43
Número de productes: 5, Surt: 37
Número de productes: 3, Surt: 37
Número de productes: 6, Surt: 35
Número de productes: 4, Surt: 34
Número de productes: 1, Surt: 30
Número de productes: 7, Surt: 26
Número de productes: 9, Surt: 17
Número de productes: 8, Surt: 13
Número de productes: 10, Surt: 7

Hem provat de entrenar el model amb més imatges i més epoch. Ho hem fet amb 100k imatges i 38 epoch.

I aquest és el resultat:

		descripcio	2757	0	0	0	0	0
True label	import	0	2447	0	0	0	0	310
	preu_unitari	0	0	1044	0	0	0	0
	quantitat_kg	0	0	0	329	0	0	0
	quantitat_ud	0	0	0	0	2428	0	0
	total	0	310	0	0	0	0	190
	Predicted label	descripcio	import	preu_unitari	quantitat_kg	quantitat_ud	total	

Tal com es pot veure, el model continua fallant, el que indica que no és perquè el model no estigui prou entrenat.

La solució bona és fer el bounding box del total més gran. Ho podem argumentar dient que com que es un total, pot pujar a més xifres així que ha de ser més llarg.

1	ARROS AMB LLET	1,32
1	MONGETA TENDRA	
	0,707 kg	1,09 €/kg
1	TUBETS HACENDADO	0,77
TOTAL (€)		110,28

Per fer els canvis, canviem les coordenades a la plantilla:

Canviem les coordenades als objectes que generen les coordenades (és la mateixa informació que hi ha al css):

```
.import {
  position: absolute;
  left: 430px;

  width: 45px;
  height: 20px;

  display: flex;
  align-items: center;
  justify-content: end;
}

.total {
  position: absolute;
  left: 400px;

  width: 75px;
  height: 20px;

  display: flex;
  align-items: center;
  justify-content: end;
}
```

Abans:

```
total = {
  'total': {'left': 430, 'width': 45},
}
```

Després:

```
total = {
  'total': {'left': 400, 'width': 75},
}
```

Ho canviem a tots els scripts que generen els tiquets, tornem a generar una mostra de tiquets, tornem a entrenar el model amb les noves dades i tornem a evaluar.

Hem entrenat el model amb 1000 tiquets i 25 epoch, hem fet la avaluació del model amb 500 tiquet com es pot veure ara comet un error:

	descripcio	0	0	1	0	0
True label	descripcio	2756	0	0	1	0
	import	0	2757	0	0	0
	preu_unitari	0	0	1044	0	0
	quantitat_kg	0	0	0	329	0
	quantitat_ud	0	0	0	0	2428
	total	0	0	0	0	500
	descripcio	import	preu_unitari	quantitat_kg	quantitat_ud	total
Predicted label						

L'error és que FESTUC TORRAT ha detectat el bbox molt petit i això ha fet que l'associes a la classe "quantitat kg", per això hem decidit que la solució és entrenar el model amb més tiquets per evitar aquest error.



Ara entrenem el model amb 5000 tiquets i 15 epoch. Avaluem el model amb 1000 tiquets, per estar més segurs del correcta funcionament del model. Com podem veure no ha fallat amb cap inferència.

True label	descripcio	0	0	0	0	0
import	0	5409	0	0	0	0
preu_unitari	0	0	2021	0	0	0
quantitat_kg	0	0	0	569	0	0
quantitat_ud	0	0	0	0	4840	0
total	0	0	0	0	0	1000

Predicted label

Un cop havent entrenat la xarxa neuronal amb els tiquets generats per nosaltres a través de la plantilla. Fem inferència i test amb tiquets reals del mercadona (no copies). En algunes ha funcionat bé i en d'altres pitjor. Una cosa que no hem tingut amb compte és que pot ser que els supermercats de diferents poblacions tinguin plantilles lleugerament diferents.

Al supermercat de blanes ho fa bé:


MERCADONA, S.A. A-46193834
C/SUS FAUQUES 4
17300 BLANES
TELEFON: 972499300
22/11/2023 12:46 OP:243482
FACTURA SIMPLIFICADA: 4400-019-372079

Descripció	P. Unit	Import
1 PATATA XURRERA		1,00
1 SUC TARONIA REFRIG.		1,50
1 BUBI M BO POB 1000G		6,50
1 TONYINA CLARA OLIV		3,20
1 YOGURT AMB FRUITES		2,10
1 NUTRONICA LIMONNA		0,80
1 CEBÀ TUB		1,00
1 F. RATLLAT STACIOSA		1,60
1 PATATA 1 KG		4,80
1 TOMAGETE TRITURAT		0,75
1 OLI DE GIBRA-SOL 1L		1,50
1 MANDARINA 2 KG		3,80
1 LLLET SEMI-SILACT		5,70
2 AGUA FONTCAGÜERA	0,75	1,50
1 COLIFOR	0,972 kg	1,90
1 EGOMERRE	0,676 kg	1,00
1 PLATAN	0,982 kg	1,00
	TOTAL (€)	42,58
	TARGETA BANCÀRIA	42,58

IVA	BASE IMPONSABLE (€)	QUOTA (€)
10%	16,31	1,60
21%	0,70	0,15
5%	1,48	0,07
0%	21,29	0,00
TOTAL	48,68	1,90

TARG. BANCÀRIA: ***** 7569
N.C. 031354238
AID: A00000000032600
Import: 42,59 €

AUT: 667189
ABC: 3030
Visa CatalaBank



S'ADMETEN DEVOLUCIONS AMB TIQUET

Però al de Banyoles no (no localitza les unitats):


MERCADONA, S.A. A-46193834
AVDA. DE LA FARIGA 40
17202 BANYULES
TELEFON: 972563880
06/04/2024 11:32 OP:387182
FACTURA SIMPLIFICADA: 3949-019-136225

Descripció	P. Unit	Import
1 AGUA FONTCAGÜERA	0,75	1,50
1 BUBU M BO POB 1000G		6,50
1 TONINA CLARA OLIV		3,20
1 YOGURT AMB FRUITES		2,10
1 NUTRONICA LIMONNA		0,80
1 CEBÀ TUB		1,00
1 F. RATLLAT STACIOSA		1,60
1 PATATA 1 KG		4,80
1 TOMAGETE TRITURAT		0,75
1 OLI DE GIBRA-SOL 1L		1,50
1 MANDARINA 2 KG		3,80
1 LLLET SEMI-SILACT		5,70
2 AGUA FONTCAGÜERA	0,75	1,50
1 COLIFOR	0,972 kg	1,90
1 EGOMERRE	0,676 kg	1,00
1 PLATAN	0,982 kg	1,00
	TOTAL (€)	76,18
	TARGETA BANCÀRIA	76,18

IVA	BASE IMPONSABLE (€)	QUOTA (€)
0%	17,81	0,00
0%	1,20	0,00
0%	4,40	4,14
21%	4,46	0,94
TOTAL	64,96	5,18

TARG. BANCÀRIA: ***** 9527
N.C. 047269222
AID: A0000000003101818

AUT: 679960
ABC: 3030

VISA CREDITODER
Import: 76,18 €



S'ADMETEN DEVOLUCIONS AMB TIQUET

Si ho fem amb un tiquet que no té res a veure, com podríà ser el del Lidl, pasa el següent:

Còpia



No funciona correctament.

10.3.2 Detecció Text (Tesseract)

Un cop tenim el [dataset dels retalls generat](#).

Per cada columna 'cutted_img_name', obrim la imatge, la passem pel model i veiem si coincideix amb la columna 'value'.

Per passar-li al model fem servir el metode `pytesseract.image_to_string()`

```
In [12]: def make_inference(row):
    filename = f"./cropped_tiquets/{row.cutted_img_name}.png"
    img = np.array(Image.open(filename))
    text = pytesseract.image_to_string(img, config="--psm 7")
    pbar.update()
    return text

df['predicted_raw_pytesseract'] = df.apply(lambda row: make_inference(row), axis = 1)
df.head()
```

100% | 2140/2141 [01:56<00:00, 18.74it/s]

Per evaluar el model, fem servir les mètriques WER i CER, també explicades al Notebook i el resultat és un 10% d'error aproximadament ja que a vegades posa caràcters a les frases que no hi són. Si els treiem, l'error baixa gairabé a 0.



10.4 Funcionalitats addicionals

Per fer la conversió de texto a audio ho hem fet amb el model gTTS i l'hem implementat de la següent manera:

```
63  def text_to_speech(text, lang = 'ca'):
64      # return fake_audio
65
66      output = gTTS(text=text, lang=lang, slow=False)
67
68      output.save("output.mp3")
69
70      # Llegim el fitxer i el convertim a base64, després el borrem del servidor.
71      with open("output.mp3", "rb") as audio_file:
72          base64_audio = base64.b64encode(audio_file.read()).decode('utf-8')
73
74      os.remove("output.mp3")
75
76      return base64_audio
77
```

El codifiquem en base64, per poder-lo enviar al frontend perquè pugui ser reproduït per el navegador.

11. Aplicació

Aquesta part del projecte consisteix en apropar l'usuari final a la nostre aplicació. perquè els usuaris puguin processar tiquets de forma fàcil.

11.1 Arquitectura General

A termes general, l'aplicació està formada per dues parts, el client i el servidor, els quals es comuniquen a través de http. El codi es pot trobar a:

- <https://github.com/jaumejp/analisis-tiquets/tree/main/escaner-tiquets>

11.2 Frontend

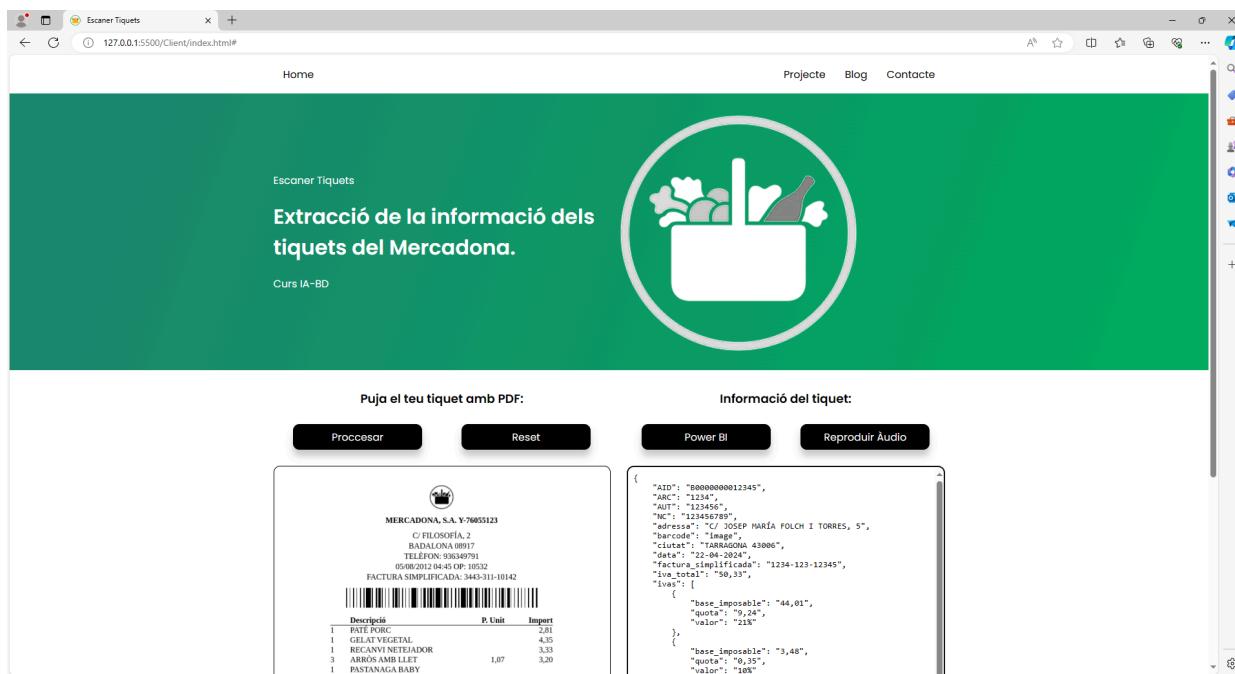
- Programada amb HTML, CSS i JavaScript.
- Té un disseny amigable i interactiu per fer tot el procés.
- Permet pujar un PDF i enviar-lo al servidor per ser processat.
- Mostrarà la imatge en png i la informació que en treu la xarxa neuronal.
- Permet reproduir amb veu els productes que hi ha al tiquet.

11.3 Backend

- Programat amb Python i el framework Flask.
- Rep els PDF's del Frontend i el converteix a imatge.
- Extreu la informació d'aquesta imatge. (executant els models neuronals).
- Genera un audio amb els productes i l'envia cap al client.
- Retorna la imatge i tota la informació al frontend per informar al usuari.
- Guarda la imatge i la informació al Data Lake.
- La lògica principal està a:
 - <https://github.com/jaumejp/analisis-tiquets/blob/main/escaner-tiquets/Server/main.py>

11.4 Funcionament

Si mirem la interfície, veiem:



Tenim dos elements principals:

- Esquerra: secció per pujar un PDF.
- Dreta: mostra el resultat del model neuronal.

El procés per enviar el PDF al servidor comença amb la necessitat d'emplenar l'input de pdf i el id del usuari. Aquesta id, idealment s'hauria de obtenir fent alguna mena de *login* de l'aplicació.

Un cop hem empletat els dos inputs, podem clicar al botó *Procesar*, el qual llegeix el pdf que hem pujat, el converteix a base64 i després a string, per poder fer un fetch al servidor.

Un cop arriba la petició a l'api, es torna a codificar el string del pdf en base64 per convertir-lo a imatge. Un cop tenim la imatge al servidor, fem el següent:

- Retornar-la al client per mostrar el resultat de la conversió al frontend.
- Passar la imatge per la xarxa neuronal.
- Guardar aquesta imatge a la cap raw.

A part de retornar la imatge del tiquet, també retornem la informació que hi ha un cop l'hem examinat amb la xarxa neuronal.

A aquesta resposta, també s'inclou un audio codificat en base64 per poder reproduir en audio la informació del tiquet al Frontend.

Per tant el frontend ja rep un objecte amb aquests tres paràmetres i podem tractar la informació per renderitzar-la i mostrar-la al client.

```

fetch(endpoint, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ pdfData: base64pdf })
})
.then(response => {
  if (!response.ok) {
    throw new Error("Error al enviar el pdf al servidor");
  }
  return response.json();
})
.then(data => {

  inputsContainer.classList.add('d-none');

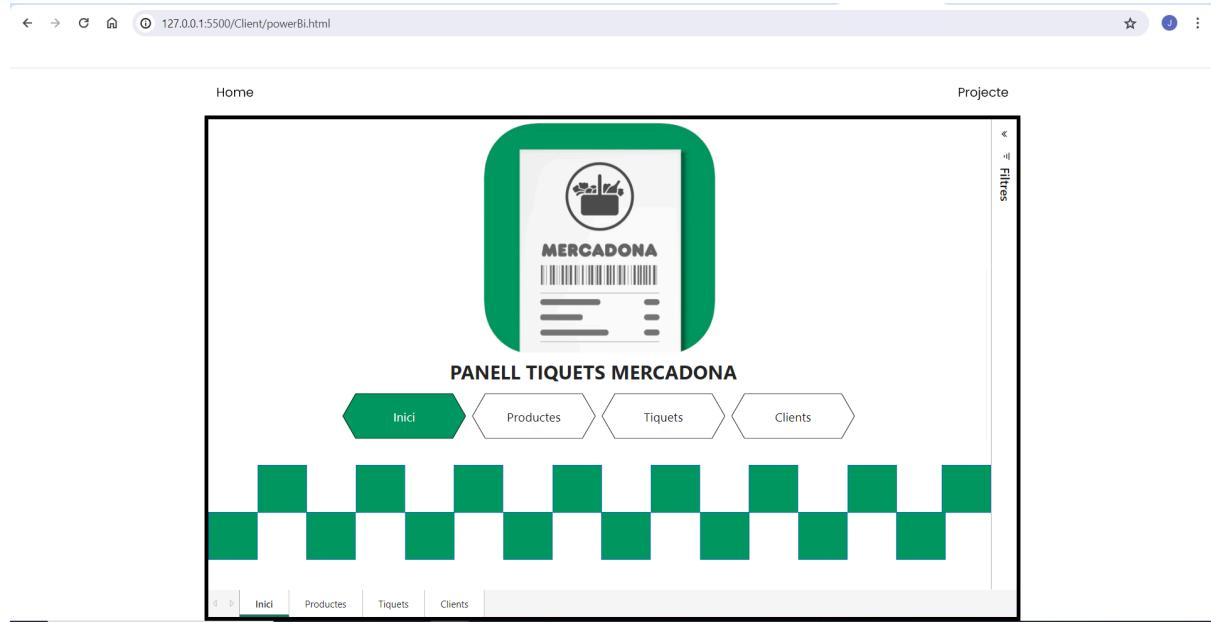
  const { image, ticket, audio } = data;

  audioBase64.innerHTML = audio
  audioBtn.classList.remove('btn-disabled')
  img.src = `data:image/jpeg;base64, ${image}`;

  const jsonString = JSON.stringify(ticket)
  textareaText(textarea, jsonString);
})
.catch(error => {
  console.error('Error:', error);
});

```

Si fem click al botó de Power BI:

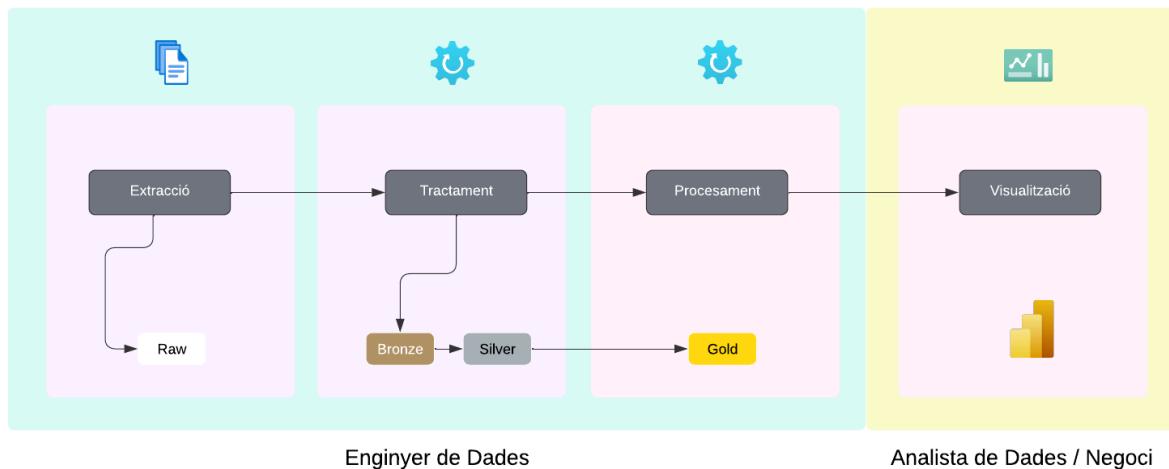


Vídeo demostració, funcionament.

https://drive.google.com/file/d/1BaMewju_znPkY_a6No09aFJT8NsXojDo/view?usp=sharing

12. ETL

La ETL (Extract, transform and load) té com a objectiu principal el tractament de les dades extretes d'un tiquet per poder-les utilitzar posteriorment per generar gràfiques i estadístiques que permetin extreure conclusions. En resum, la ETL el que fa és obtenir les dades d'entrada, processar-les de manera correcta i emmagatzemar-les en un format estructurat per a futurs usos.



El primer pas és l'extracció o generació de dades. Aquestes, poden ser generades per màquines, events de pàgines web o simplement extretes de apis. Seguidament es guardaran en les capes bronze i raw i seran tractades i transformades per ser finalment visualitzades.

El procés de tractament podrà ser la neteja o formateig de les dades. I, el de processament el fet de crear agregacions o relacionar les dades entre si. Aquest processament té més valor pel negoci ja que permet als analistes de dades transformar les dades en coneixement i informació real per la presa de decisions. Aquest procés final de agregar les dades també és crucial per fer servir totes les dades o *Big Data* per alimentar models predictius que ajudin a les empreses a prendre decisions.

En els següents apartats veurem amb quines tecnologies i arquitectura hem implementat.

Per a una explicació més tècnica i detallada sobre que realitzen els scripts per processar i tractar les dades, hi ha la informació molt detallada als notebooks de documentació.

Per altre banda, hi ha els notebooks de producció que son els que utilitzem a l'entorn real i estan orientats a una millor execució.

Aquestes dos documentacions estan a:

- <https://github.com/jaumejp/analisis-tiquets/tree/main/etl-tiquets/Scripts>

12.1 Descripció del procés ETL

A continuació, expliquem de manera senzilla com funciona la ETL:

Extracció:

1- Quan el model analitza el tiquet, genera un document en format JSON amb les dades que ha extret. Aquest document s'emmagatzema a la capa Bronze del nostre Data Lake. A més, també es guarda la imatge a la capa raw.

Tractament i Processament:

2- A continuació, utilitzem scripts que fan servir Spark i Python per processar aquest document. Aquest procés implica netejar i estructurar les dades per a un millor ús.

3- Un cop les dades estan processades, les guardem en format Parquet a la capa Silver del Data Lake. Aquí, realitzem un altre pas de processament per organitzar les dades de manera que siguin més fàcils de comprendre i utilitzar per crear gràfiques i estadístiques.

4- Les dades ja processades i organitzades es guarden en una base de dades SQL a Azure SQL Server. Aquesta base de dades és la nostra capa Gold, on les dades estan emmagatzemades de manera estructurada i eficient.

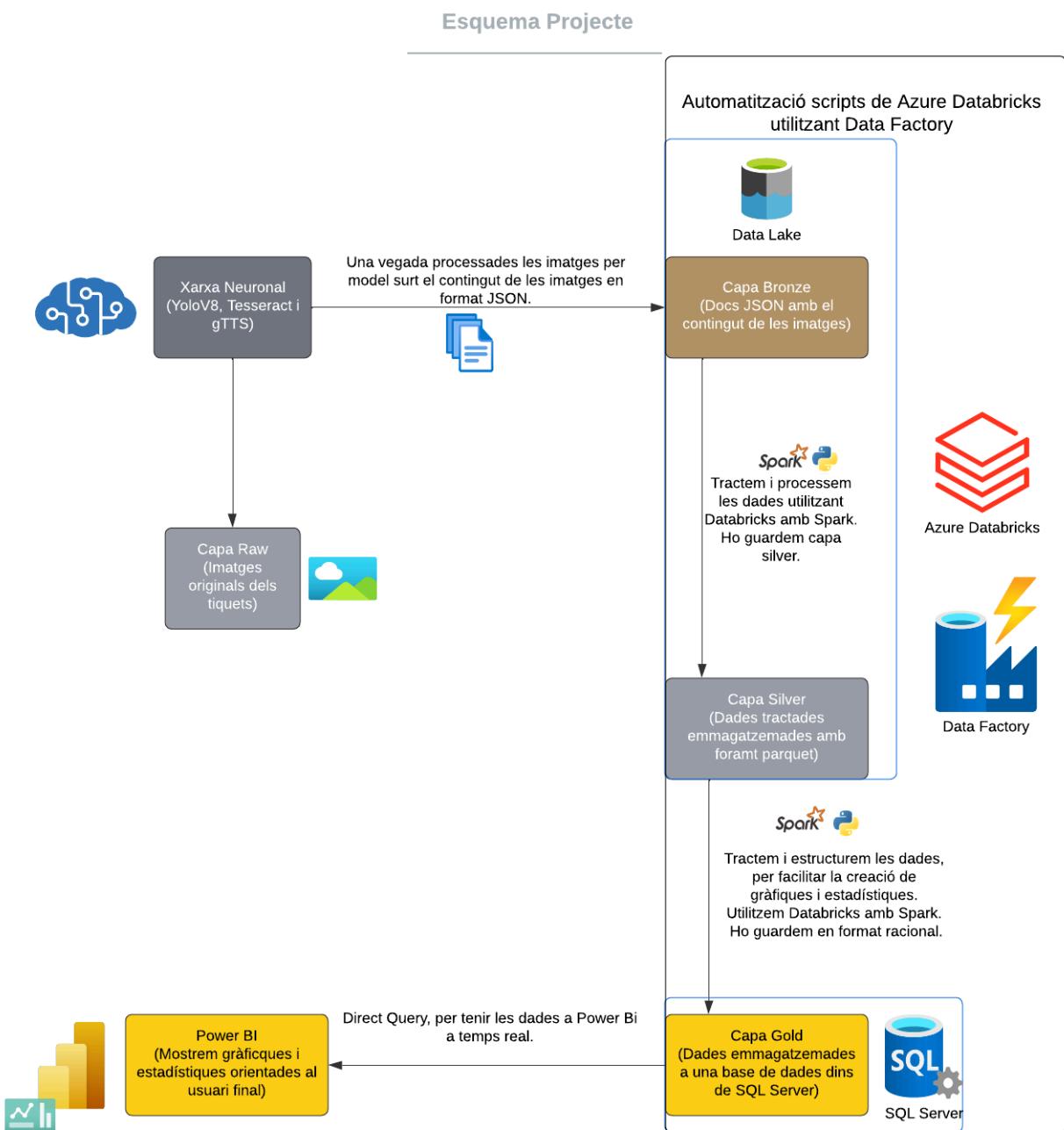
5- Per dur a terme tot aquest procés, fem servir Databricks per executar els scripts i Data Factory per automatitzar les tasques i assegurar-nos que tot funcioni de manera automàtica sense haver de fer-ho manualment.

Visualització de dades:

6- Un cop les dades estan a la capa Gold, ja podem utilitzar-les per realitzar anàlisis i estadístiques que ens ajudaran a prendre decisions. En el nostre cas ho fem amb Power Bi.

12.2 Esquema ETL

Per entendre millor la ETL i el procés explicat anteriorment, hem realitzat un esquema general que mostra les diferents capes i eines utilitzades a cada pas.



La ETL comença al [backend](#) de la aplicació. L'aplicació rep les dades de l'usuari i les processa. Un cop processades, penja la imatge que ha fet servir a la capa raw i el resultat del model neuronal a la capa bronze. Ho fem amb la llibreria: `azure.storage.blob` i la funció `BlobServiceClient` i ho fa el següent codi:

```
def save_json_bronze(json_string, user_id):
    # pip install azure-mgmt-storage
    # pip install azure-storage-blob

    # Container d'azure on ho penjarem.
    container_name = 'bronze'

    # Convertim el document a JSON si és necessari - ES POT BORRAR?
    if isinstance(json_string, dict):
        json_string = json.dumps(json_string)

    # Agafem la data d'avui i la guardem en el format que ens interessa
    current_datetime = datetime.now()
    timestamp = current_datetime.strftime("%d%m%Y_%H%M%S")
    current_datetime_formated = current_datetime.strftime("%d%m%Y")

    # Creem el path on hem de guardar el json
    # /id usuari/data formatejada/timestamp
    path = f"{user_id}/{current_datetime_formated}/{timestamp}.json"

    # Connectem al servei de blob d'Azure
    blob_service_client = BlobServiceClient.from_connection_string(connection_string)
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=path)

    # Pujem el fitxer JSON al blob d'Azure
    blob_client.upload_blob(json_string, overwrite=True)

    print('Pujat a Azure Data Lake - Bronze')

def save_img_raw(image_path, user_id):
    # Container d'azure on ho penjarem.
    container_name = 'raw'

    # Agafem la data d'avui i la guardem en el format que ens interessa
    current_datetime = datetime.now()
    timestamp = current_datetime.strftime("%d%m%Y_%H%M%S")
    current_datetime_formated = current_datetime.strftime("%d%m%Y")

    # Creem el path on hem de guardar el json
    # /id usuari/data formatejada/timestamp
    path = f"{user_id}/{current_datetime_formated}/{timestamp}.png"

    # Connectem al servei de blob d'Azure
    blob_service_client = BlobServiceClient.from_connection_string(connection_string)
    blob_client = blob_service_client.get_blob_client(container=container_name, blob=path)

    # Pujem el fitxer JSON al blob d'Azure
    blob_client.upload_blob(image_path, overwrite=True)

    print('Pujat a Azure Data Lake - Raw')
```

12.3 Arquitectura Medallion

Hem utilitzat l'arquitectura Medallion, que es basa en processar les dades utilitzant capes. Aquesta arquitectura permet transformar i millorar les dades a mesura que passen per les capes, des de la seva forma inicial “brut” fins la seva presentació final en un dashboard.

A continuació explicarem les capes, en aquest cas orientades a que realitza cada capa en el nostre projecte

Capa RAW:

Aquesta capa actua com el magatzem inicial de les nostres dades. Aquí emmagatzemem les imatges originals que rebem dels clients. Aquestes imatges són importants per diverses raons, com ara entrenar nous models o detectar canvis en els patrons dels tiquets. Organitzem les imatges classificades segons l'ID del client i la data d'escaneig, permetent un accés fàcil i ordenat.

The screenshot shows a file browser interface with the following details:
Location: raw / 2 / 21052024
Search blobs by prefix (case-sensitive)
Name
[.]
21052024_20051716314680.png
21052024_20051716314693.png
21052024_20051716317301.png

Capa BRONZE:

A la capa Bronze, els documents JSON generats després del processament de les imatges dels tiquets són emmagatzemats. Aquests JSON contenen la informació extreta dels tiquets, però en un format sense processar, tal com el model les ha proporcionat. Posteriorment, aquests JSON són tractats i enviats a la capa següent, la Silver. Igual que a la capa RAW, organitzem els JSON segons l'ID del client i la data del tiquet.

bronze/id_client/data/json

The screenshot shows a file browser interface with the following details:
Location: bronze / 1 / 21052024
Search blobs by prefix (case-sensitive)
Name
[.]
21052024_20051716314645.json
21052024_20051716314659.json
21052024_20051716314670.json
21052024_20051716317284.json

Capa SILVER:

En aquesta fase, els fitxers Parquet arriben després de processar els JSON. Aquests fitxers estan emmagatzemats en format Parquet dins del contenidor Silver del nostre Data Lake. Aquest format Parquet permet una emmagatzematge més eficient i optimitzat. Les dades en aquesta capa estan refinades i són més estructurades que les de la capa Bronze. Tractem aquests Parquet i els enviem a la capa Gold. Emmagatzemem els Parquet particionats de dues maneres diferents: una partició per ID del client, que agrupa tots els tiquets del mateix client, i una partició per dia, que agrupa tots els tiquets d'una mateixa data.

Partició clients:

/silver/clients/id_client=x/Parquet

Location: silver / clients / id_client=4

Search blobs by prefix (case-sensitive)

Name
<input type="checkbox"/> [..]
<input type="checkbox"/> _committed_2439531566078366682
<input type="checkbox"/> _committed_2766831015800996423
<input type="checkbox"/> _committed_6509451233396616621

Partició data:

/silver/dies/data=dd/mm/YYYY/Parquet

Location: silver / dies / data=22-04-2024

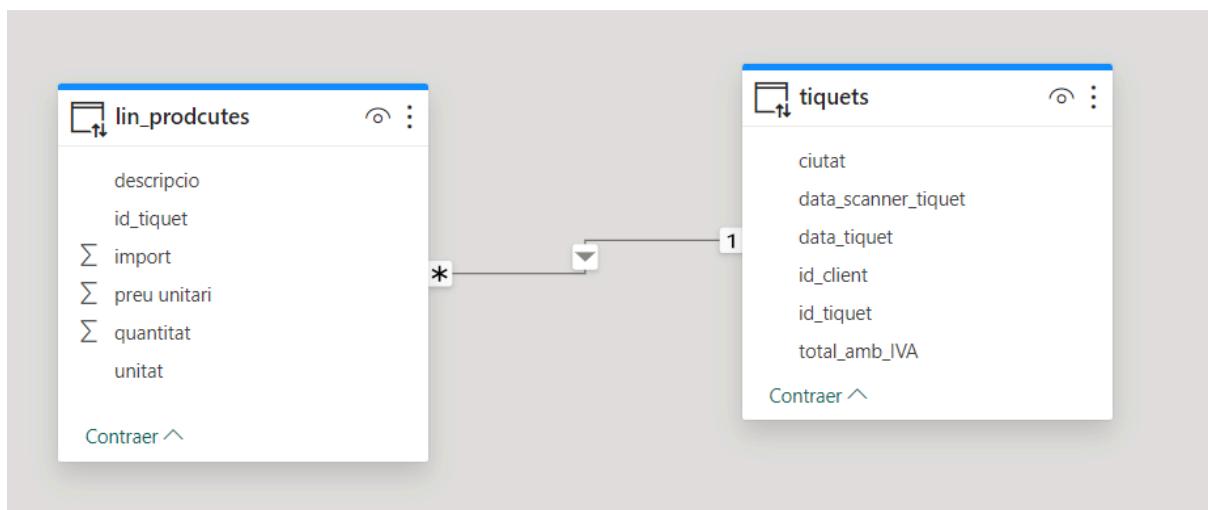
Search blobs by prefix (case-sensitive)

Name
<input type="checkbox"/> [..]
<input type="checkbox"/> _committed_2792846715504749062
<input type="checkbox"/> _committed_2954772228773187216
<input type="checkbox"/> _committed_3477434496188429783
<input type="checkbox"/> _committed_5205904188148232726
<input type="checkbox"/> _committed_6160802361299843110
<input type="checkbox"/> _committed_7308822718026646640

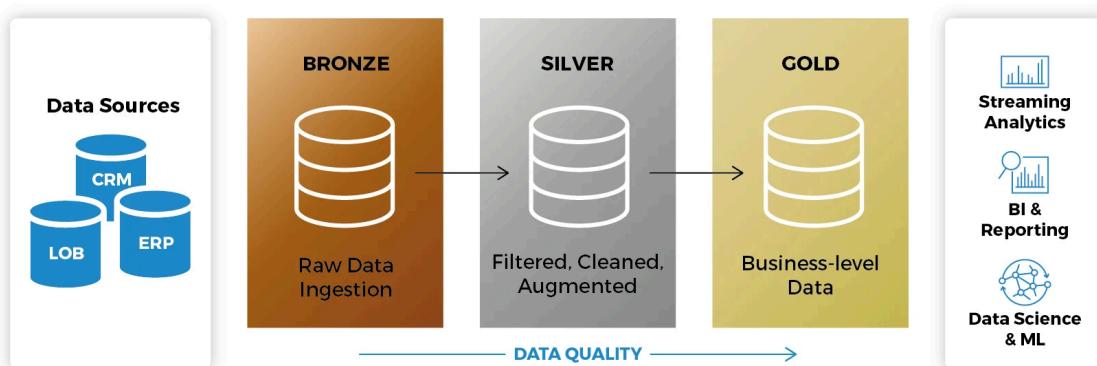
Capa GOLD:

Finalment, a la capa Gold, les dades estan emmagatzemades de manera estructurada i eficient. Aquesta capa consta de dues taules: una per als tiquets i una altra per a les línies dels tiquets. Hem optat per aquesta estructura per proporcionar una major flexibilitat i optimitzar-ne el funcionament. Les dues taules estan relacionades entre si mitjançant l'identificador del tiquet, permetent-nos accedir fàcilment a les dades del tiquet i les seves línies de productes associades. Aquest disseny ens proporciona una gestió més eficaç i una millor organització de la informació, facilitant-ne l'anàlisi i l'extracció de conclusions.

Estructura de la base de dades per facilitar la comprensió:



Esquema visual de l'arquitectura Medallion:



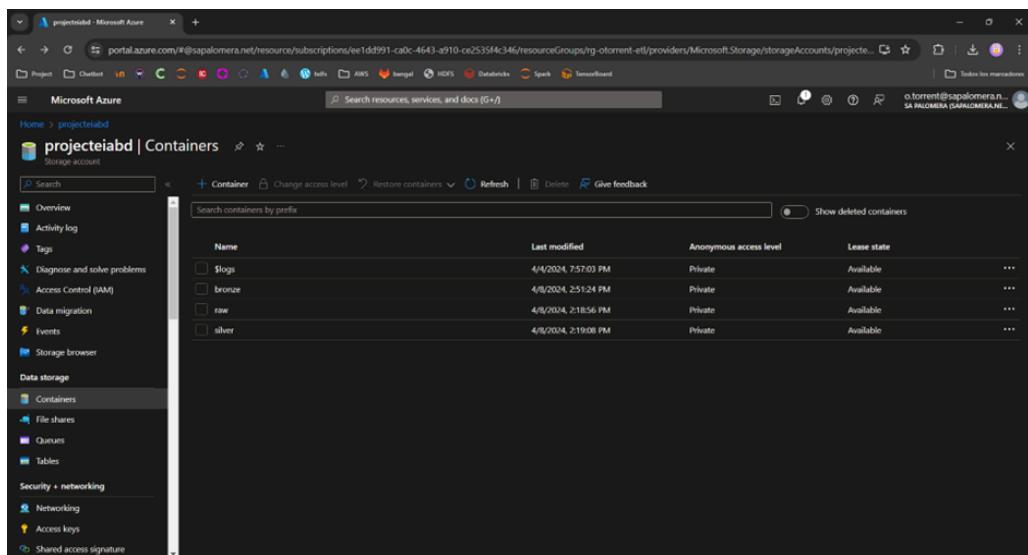
A continuació expliquem detalladament les tecnologies utilitzades en el procés i quina part realitzen de la ETL. Les tecnologies són Azure Data Lake, Azure SQL Server, Azure Databricks, Apache Spark i Python.

12.4 Azure Data Lake

Amb ADL (Azure Data Lake) hem pogut crear el nostre DL (Data Lake) per poder emmagatzemar les dades, organitzades per containers i directoris. Hem actualitzat el DL a Gen2 ja que sinó no permet la funcionalitat de crear directoris. I aquesta era una funcionalitat necessària en el nostre cas per poder tenir les dades emmagatzemades de forma organitzada i facilitar a la hora de realitzar accions amb aquestes dades.

Una alternativa pot ser Snowflake Data Cloud però hem preferit Azure Data Lake ja que té una gran integració amb l'ecosistema Azure i apart permet una gran flexibilitat a l'emmagatzamatge de dades respecte a Snowflake.

A continuació es pot veure una captura de pantalla, on es veu el Data Lake que hem creat (projecteabd) amb els seus containers (bronze, raw, silver).



The screenshot shows the Azure Storage account interface for the 'projecteabd' storage account. On the left, there's a navigation sidebar with options like Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser, Data storage (Containers, File shares, Queues, Tables), Security + networking (Networking, Access keys, Shared access signature), and Help. The main area is titled 'Containers' and shows a list of four containers: \$logs, bronze, raw, and silver. Each container has columns for Name, Last modified, Anonymous access level, and Lease state. All containers are listed as Private and Available.

Name	Last modified	Anonymous access level	Lease state
\$logs	4/4/2024, 7:57:03 PM	Private	Available
bronze	4/8/2024, 2:51:24 PM	Private	Available
raw	4/8/2024, 2:18:56 PM	Private	Available
silver	4/8/2024, 2:19:08 PM	Private	Available

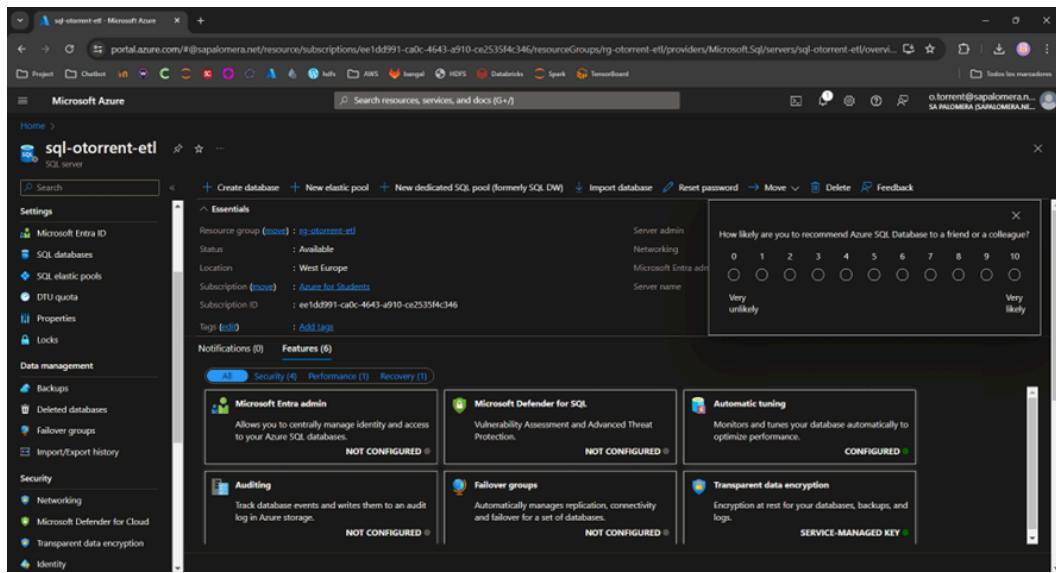
En el DL (Data Lake) tenim diferents capes (containers) de processament, amb això ens referim amb les capes raw, bronze, silver. Falta la capa Gold, ja que no s'emmagatzema dins del DL. Aquestes capes les hem explicat anteriorment.

12.5 Azure SQL Server

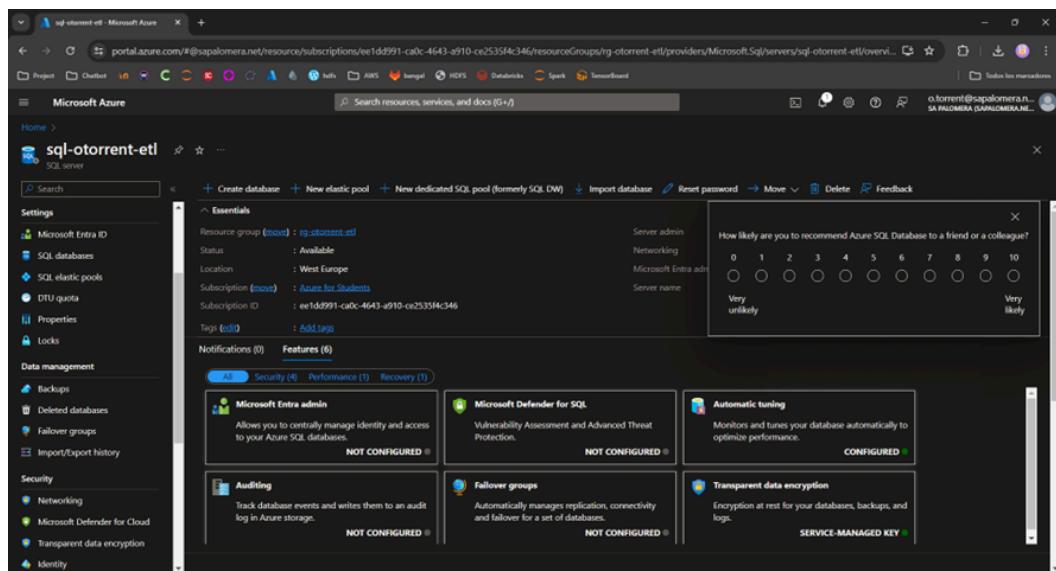
Hem utilitzat Azure SQL Server per crear la nostra base de dades relacional. Necessitem aquesta base de dades per emmagatzemar les dades tractades i processades, aquesta és la capa anomenada Gold. Les dades dins d'aquesta capa estan estructurades i organitzades de manera que es poden utilitzar per crear gràfiques o realitzar anàlisis entre altres amb més facilitat.

Hem utilitzat aquesta eina perquè ofereix una integració amb l'ecosistema Azure. Garanteix una alta disponibilitat, ja que es pot accedir a les dades des de qualsevol lloc amb connexió a internet. Aquesta característica és important ja que hem accedit en diferent localitzacions, equips i persones.

A continuació deixo una captura de com es veu SQL Server



I també una captura de la base de dades que tenim dins d'aquest servidor SQL.



12.6 Azure Databricks

Hem optat per Azure Databricks perquè és una eina extremadament potent que ofereix una àmplia gamma de funcionalitats per al processament i anàlisi de dades a gran escala.

Azure Databricks ens permet executar scripts al núvol, garantint que sempre disposem dels recursos necessaris per a les nostres tasques, evitant així qualsevol problema de capacitat. A més, ens permet escalar recursos de hardware segons les necessitats del moment, assegurant una execució eficient i sense interrupcions.

Un dels avantatges clau de Azure Databricks és que elimina la necessitat de descarregar i configurar eines addicionals, ja que aquestes ja estan integrades dins de la plataforma. Això simplifica molt la configuració inicial i el manteniment.

Hem utilitzat Azure Databricks perquè forma part de l'ecosistema Azure i permet una integració fluida amb la resta d'eines d'Azure, fet que és crucial per a nosaltres. Aquesta integració facilita la creació i automatització de fluxos de treball de dades complexos mitjançant Azure Data Factory.

The screenshot shows the Azure Databricks workspace interface. On the left, there is a sidebar with various navigation options: New, Workspace, Recents, Catalog, Workflows, Compute, Data Engineering, Job Runs, Machine Learning, Playground, Experiments, Features, Models, Serving, Partner Connect, and Collapse menu. The 'New' button is highlighted. The main area is titled 'Projecte' and shows a list of items under 'Workspace'. The list includes:

Name	Type	Owner	Created at
Bronze-to-Silver-Prod	Notebook	Juan Jaume	2024-05-27 20:13:15
Silver-to-Gold-Prod	Notebook	Juan Jaume	2024-05-27 20:13:15

At the top right, there is a search bar with the placeholder 'Search data, notebooks, recent, and more...', a 'CTRL + P' keyboard shortcut key, and a 'tickets_project' dropdown. There are also 'Share' and 'Create' buttons.

12.7 Azure Data Factory

Utilitzem Azure Data Factory perquè ens permet automatitzar els scripts de Databricks i configurar triggers per a l'execució periòdica. Aquesta eina ens ofereix una gran flexibilitat i escalabilitat a la nostre ETL, ja que podem afegir noves funcionalitats de manera més eficient i adaptar-nos a les necessitats canviants del projecte.

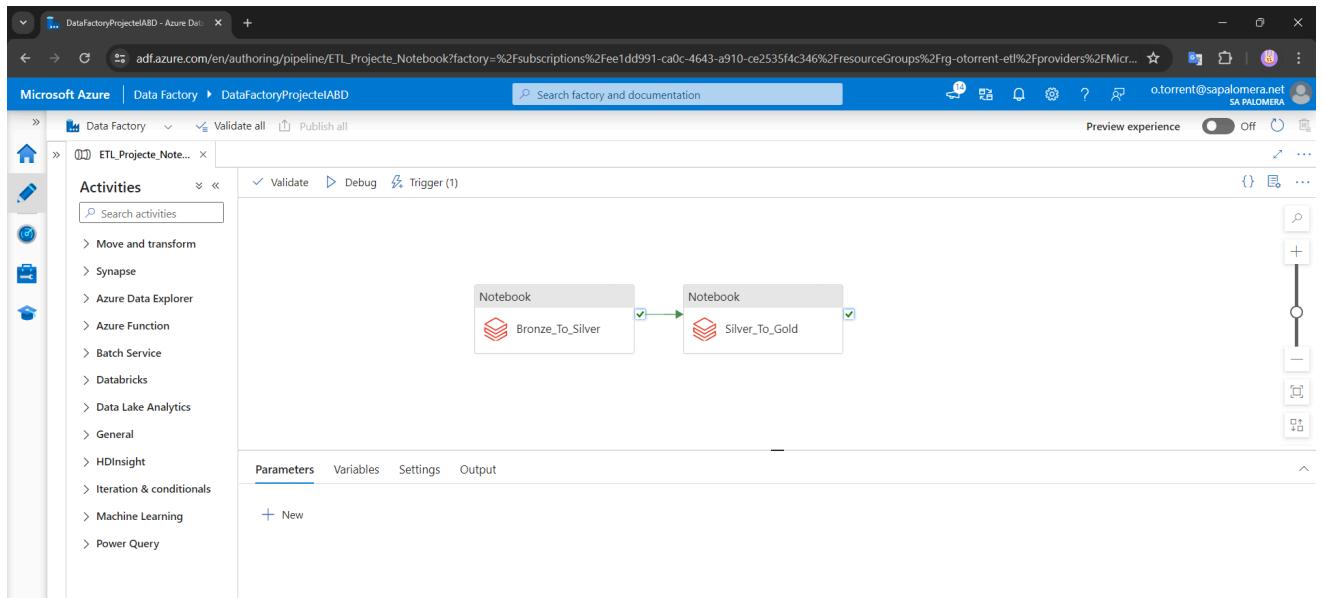
Permet una integració fàcil amb altres elements de l'ecosistema d'Azure. També ens permet monitoritzar i gestionar fàcilment els nostres processos ETL, assegurant que tot funcioni correctament i detectant qualsevol problema de manera proactiva.

Actualment, el trigger està configurat per executar-se un cop al dia, però es pot modificar fàcilment segons les necessitats del projecte.

Hem realitzat un manual per realitzar la configuració entre Azure Data Factory i Azure Databricks, a continuació deixem l'enllaç:

[Manual Azure Data Factory amb Azure Databricks](#)

A continuació deixem una captura de com ha quedat per el moment Azure Data Factory:



12.8 Apache Spark

Hem triat Apache Spark per importar i exportar dades perquè és una eina super potent i fàcil d'utilitzar. Amb Spark, podem treballar amb enormes quantitats de dades de manera eficient i escalable. Lu més important de Spark és la seva capacitat per processar dades en temps real amb streaming o en lots. Això ens dóna molta flexibilitat i agilitat per gestionar les dades.

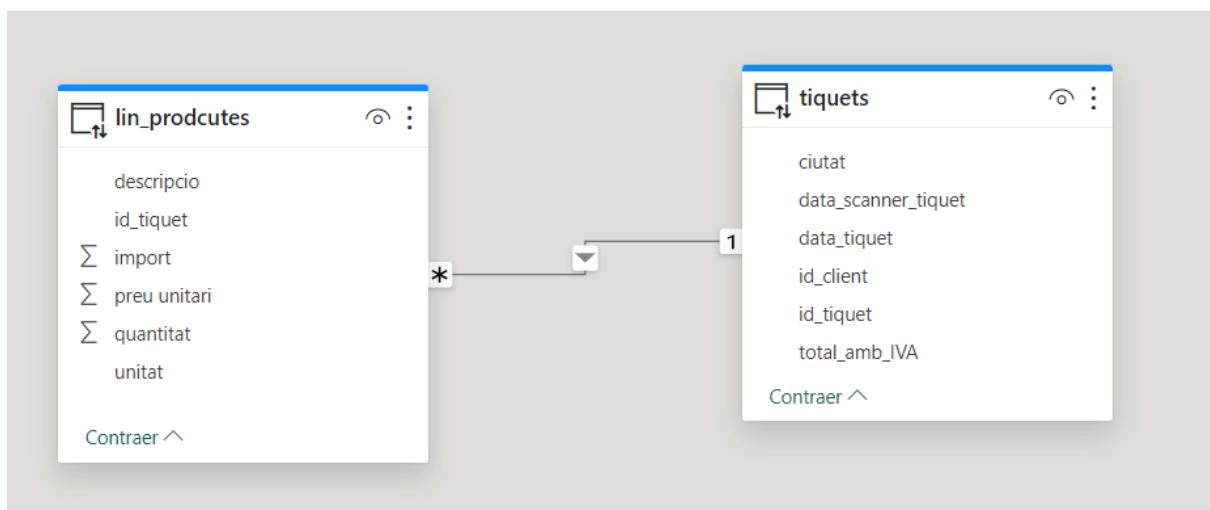
Encara que, per temes de temps i complexitat, no hem pogut desenvolupar la part de streaming, tot està preparat per fer-ho en un futur.

13. Dashboard Power BI

Fem servir Power BI per crear visualitzacions interactives i panells de control que presenten estadístiques detallades sobre els habits de compra dels usuaris. Power BI és una potent eina d'anàlisi i visualització de dades. Mitjançant Direct Query, extraurem les dades directament del nostre servidor SQL. Això ens permetrà obtenir una visió en temps real de les dades emmagatzemades al nostre sistema.

Hem utilitzat Power BI ja que és una eina que té un gran potencial i ús al món real, i ens ha interessat poder aprendre més sobre aquesta eina tan coneguda.

Dins de Power BI hem creat la relació entre les dos taules, d'aquesta manera podem generar moltes altres estadístiques.



A continuació deixem l'enllaç del document de Power BI:

- <https://github.com/jaumejp/analisis-tiquets/tree/main/etl-tiquets/PowerBi>

13.1 Pantalles

Si accediu amb el correu de microsoft del institut (@sapalomera.net) al següent link, es pot veure el power BI en directe.

- https://app.powerbi.com/groups/f7d22044-561c-4216-aef2-204f22a81454/reports/a1edcf2f-489e-4165-92b2-75f27c22f8c0?ctid=f885f0ed-d49b-47d8-a57b-46659c27da85&pbi_source=linkShare

A continuació, explicarem les diferents pantalles i informació que s'hi veu representada.

13.1.1 Inici

Aquesta pantalla és la primera que veurà l'usuari. És la secció inicial on seleccionem quina secció de gràfiques volem veure. Podem seleccionar productes, tiquets i clients.



13.1.2 Tiquets

La secció de tiquets consta de 4 indicadors principals (situats a la esquerra de les pantalles).

- Número de tiquets.
- Número de ciutats.
- Import total.
- Mitjana dels imports dels tiquets.

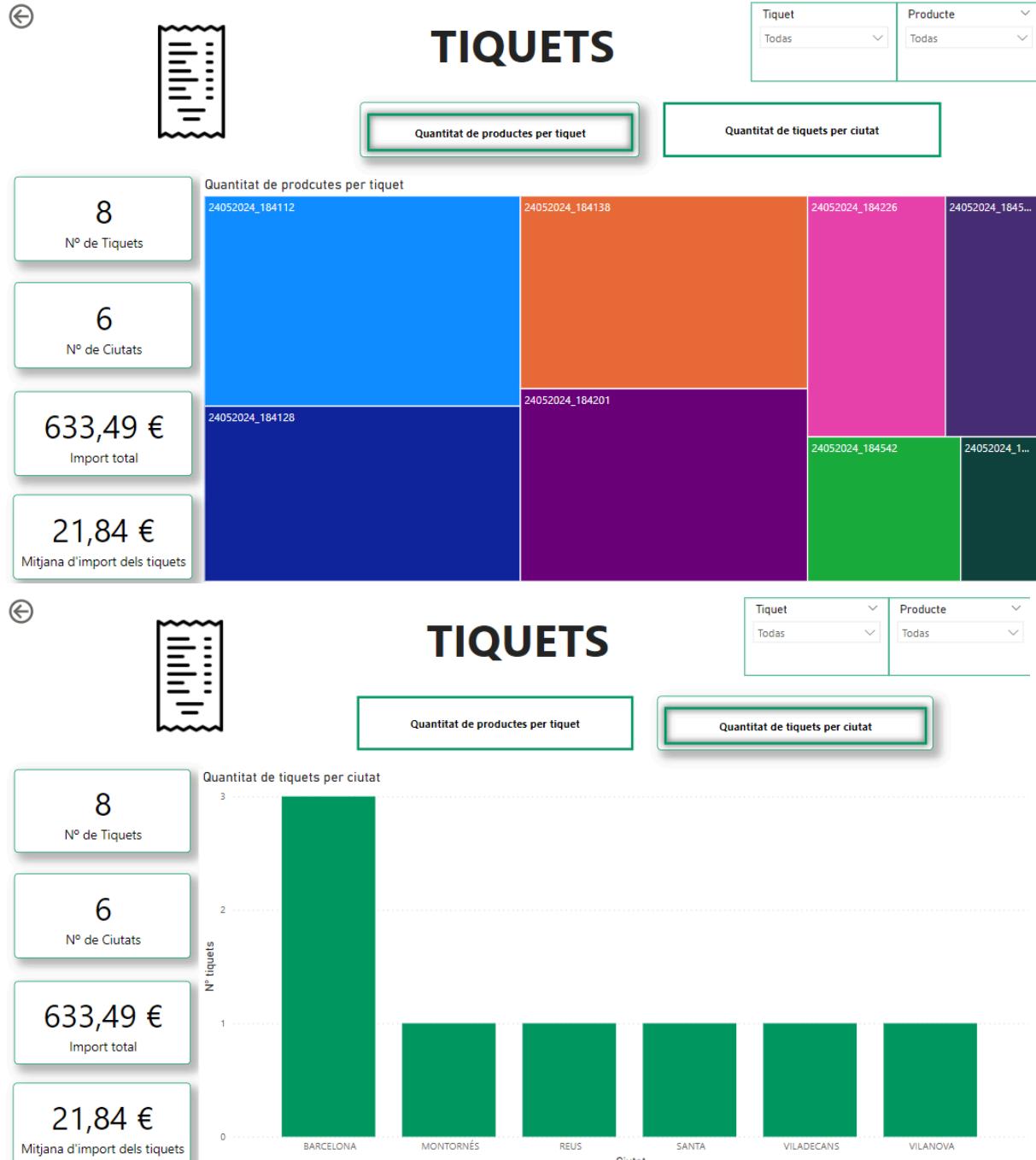
A més a la part superior dreta, tenim dos filtres:

- Tiquet (per veure les dades d'un tiquet)

- Producte (nom del producte).

Finalment, tenim dos pantalles les quals s'hi accedeixen pels botons:

- Quantitat de productes per tiquet.
 - Podem veure la quantitat de productes que hi ha per cada tiquet.
- Quantitat de tiquets per ciutat.
 - Podem veure cada ciutat quants tiquets té.



13.1.3 Productes

La secció de productes també consta de 4 indicadors principals (situats a la esquerra de les pantalles).

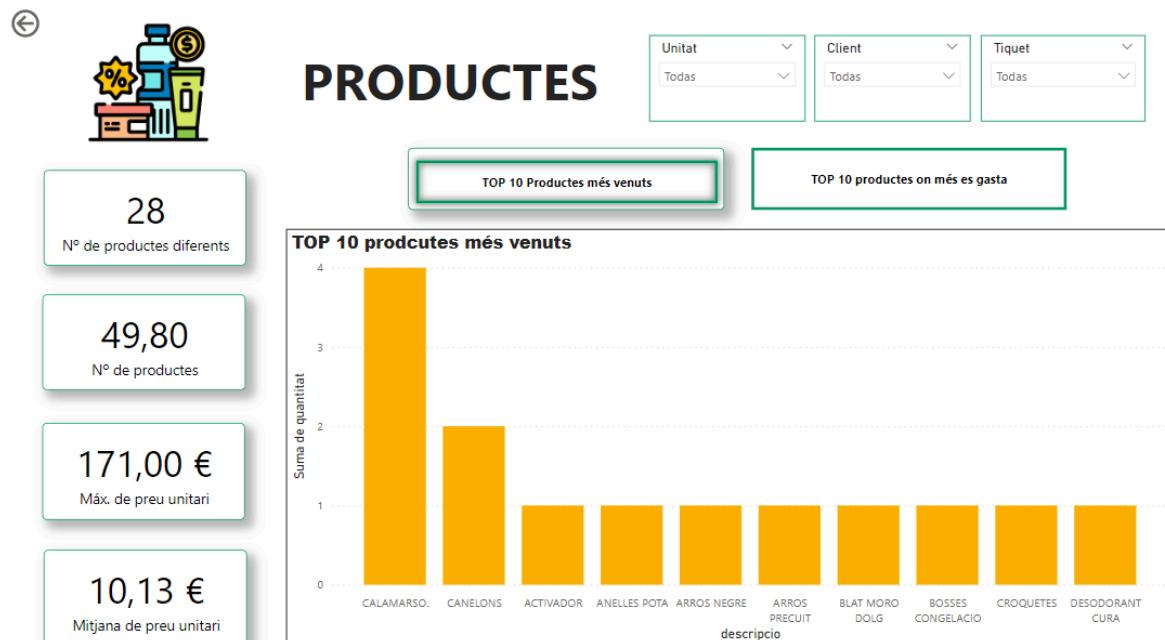
- Número de productes diferents
- Número de productes totals
- El preu unitari més gran
- Mitjana dels preus unitaris dels productes.

A més a la part superior dreta, en aquest cas tenim tres filtres:

- Unitat (tipus de unitat dels productes kg o unitats)
- Client (número de client).
- Tiquet (per veure les dades d'un tiquet)

Finalment, tenim dos pantalles les quals s'hi accedeixen pels botons:

- Top 10 productes més venuts
 - Podem veure els 10 productes més venuts.
- Top 10 productes on més es gasta
 - Podem veure els 10 productes que la gent gasta més diners.





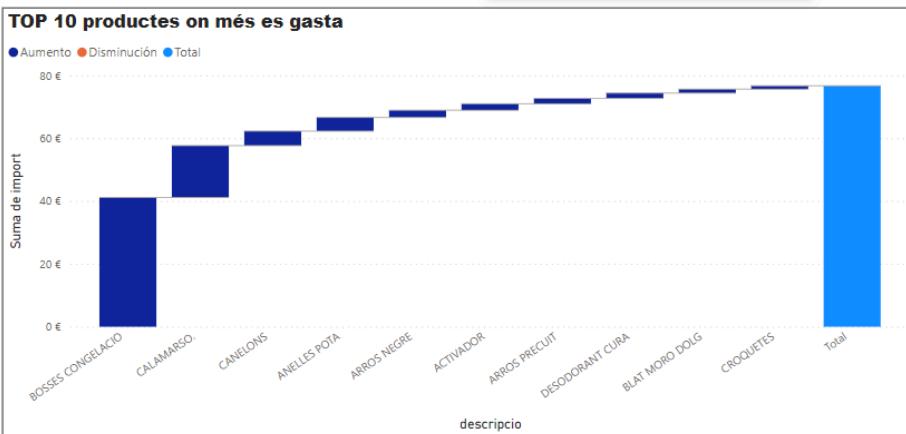
PRODUCTES

Unitat	ud
Client	Todas
Tiquet	Todas



TOP 10 Productes més venuts

TOP 10 productes on més es gasta



13.1.4 Clients

La secció de clients consta de 2 indicadors principals (situats a la part superior dreta de la pantalla).

- Número de clients.
- Número de ciutats.

A més a la part superior dreta, en aquest cas tenim un filtre:

- Client (número de client).
- Ciutat (nom de ciutat).

Finalment, a la part inferior tenim tres gràfiques:

- Número de clients per ciutat.
 - Podem veure quants clients té cada ciutat.
- Import per client.
 - Podem veure el total d'import que ha gastat cada client.
- Mapa de ciutats dels clients.
 - Podem veure un mapa amb les ciutats des de on compren els clients.



CLIENTS

10

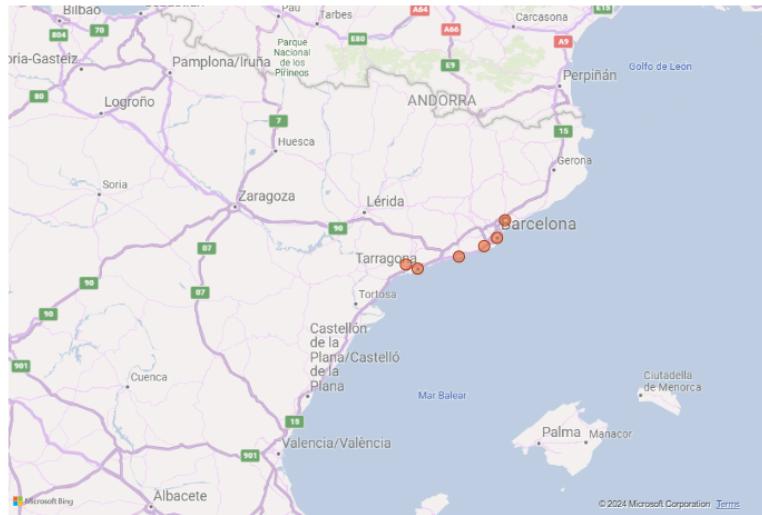
Nº de Clients

8

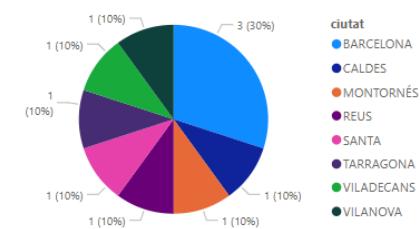
Nº de Ciutats

Ciutat	Client
Todas	Todas

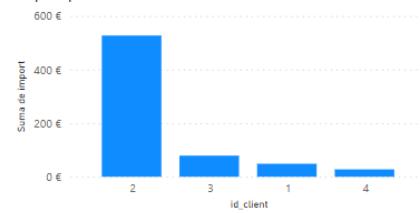
ciutat



Nº de clients per ciutats



Import per client



14. Conclusions

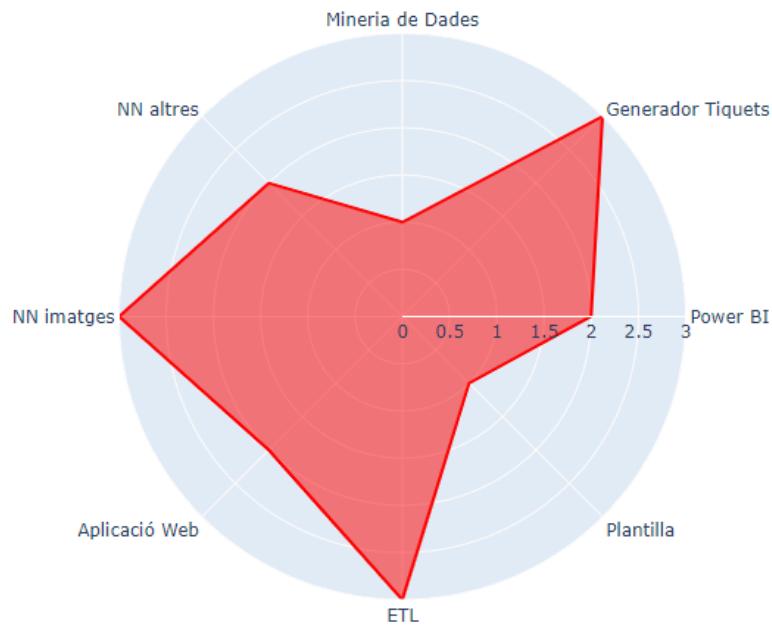
14.1 Aprendentatges

Els principals aprenentatges han estat tres:

- Millora de la programació: Gràcies a haber de pensar i implementar algoritmes i estructures de dades complexes, ha millorat molt la nostra lògica de programació.
 - Generació de les coordenades.
 - Relacionar els diferents bounding boxes que surten de la xarxa neuronal (els que són del mateix producte).
- Xarxes neuronals: Gràcies al haver de fer l'ajustatge fi la xarxa neuronal amb les nostres imatges, hem vist tots els errors que es podien anar generant i com es comporta la xarxa en funció de com preparam els tiquets.
- Big data: Aplicació d'una arquitectura real amb DataFactory i Databricks al núvol amb Azure.

14.2 Possibles Millores

Segons el diagrama d'aranya que representa el projecte:



Considerem les següents millores

14.2.1 Minería de Dades

- Actual: Nivell 1.
- Millores:

- Integrar-ho amb el data lake existent de azure.
- Fer les ingestes automàtiques.

14.2.2 Plantilla

- Actual: Nivell 1.
- Millores:
 - Fer plantilles d'altres supermercats.

14.2.3 Power BI

- Actual: Nivell 2.
- Millores:
 - Més estadístiques.
 - Pantalles per usuaris independents.

14.2.4 Generador Tiquets

- Actual: Nivell 3.
- Millores:
 - Afegir més padding random als tiquets, perquè la xarxa aprengui a detectar millor (així no estan els productes allà mateix).
 - Plantilles d'altres supermercats.
 - Quan generem els tiquets estaria bé guardar el nom del producte amb la categoria amb una taula perquè després quan fem inferència i detectem els productes, poder anar a buscar de quina categoria és i poder fer estadístiques al respecte.

14.2.5 ETL

- Actual: Nivell 3.
- Creiem que falta alguna millora, però està casi al nivell 3 ja que un entorn productiu es faria servir algún model similar.

14.2.6 Aplicació Web

- Actual: Nivell 2.
- Millores:
 - Fer autenticació d'usuaris amb login.
 - Desplegament al núvol.

14.2.7 NN imatges

- Actual: Nivell 3.
- Creiem que hem cobert tots els possibles fallos ja que dona 100% d'encert.

14.2.8 NN altres

- Actual: Nivell 2.
- Millores:
 - Explorar altres models.
 - Aconseguir millors resultats amb un fine tuning.