O presente relatório apresenta o desenvolvimento da ferramenta ETL para extração de informações de um CSV de uma base com informações de pedidos e itens do pedido. A partir disso, era necessário calcular a receita e impostos dos produtos, para cada pedido. Como item adicional também foi realizado o cálculo da média mensal de receita e impostos. O projeto foi criado inteiramente em Ocaml e com o dune para organizar as dependências e módulos do projeto. Para esse projeto, foram utilizadas LLMs em algumas partes de sua execução e seu uso será detalhado à frente.

A partir dos arquivos csv disponibilizados, foi utilizada a biblioteca “csv” que traz módulos que permitem a leitura do arquivo csv, transformando em um Csv.t, uma matriz com valores (trata-se de uma lista que representa as linhas, cada linha tem uma lista que representa as colunas e cada elemento dessa última lista são strings). Foi criado um arquivo “readData.ml” para fazer a chamada das funções Csv.load dos dois arquivos.

Com os Csv.t partiu-se para a criação das helper functions, para transformar cada uma das linhas em “records” que são tipos customizados do Ocaml, semelhantes a um dicionário/HashMap porém os campos por padrão são imutáveis. Desse modo, o objetivo das funções helpers é facilitar a manipulação dos dados a partir dos seus campos, nas etapas seguintes.

Antes de criar as funções, foram criados tipos customizados, para receber as informações de forma apropriada e facilitar a rastreabilidade. Os order e order_item ficaram da seguinte forma:

```
(** Represents an order record with its attributes *)
type order = {
   id : int;
   client_id : int;
   order_date : datetime;
   status: order_status;
   origin: order_origin;
}
(** Represents an item within an order *)
type order_item = {
   order_id : int;
   product_id : int;
   quantity : int;
   price : float;
   tax : float;
}
```

Em que cada um desses campos corresponde a uma das colunas do csv original.

Com os tipos declarados, foi utilizada a seguinte estratégia para converter as informações de Csv.t para uma lista de records:

1. Em cada Csv.t foi separado o header (cabeçalho) dos dados
2. Foi utilizada a função associate da biblioteca csv, para gerar listas de tuplas, em que o primeiro elemento é o nome da coluna e o segundo o valor.
3. Em cada uma das linhas (através de um map) foi utilizada uma função para preencher um record inicialmente com informações inválidas (unknow ou -1) e substituindo pelos valores que foram encontrados.

A função de preenchimento do record funciona de forma semelhante a uma de somar valores em uma lista. Tem-se um acumulador - no caso em questão, o dicionário inicializado com valores inválidos - e a partir do pattern matching (semelhante a uma série de if else

encadeados) é possível analisar qual o campo do valor em questão e atualizar o record (na prática é criado um novo, com o valor alterado). Utilizando a função List.fold_left então, podemos aplicar essa função (que recebe o record e o valor a ser atualizado e retorna um record com o campo atualizado) e aplicar para cada um dos elementos da linha.

Essa estratégia foi aplicada em ambos os helpers dos arquivos csv e com isso, foi possível chegar a uma lista de records que representam os dados originais. Nessa etapa, foi utilizado o claude 3.7 para corrigir alguns problemas que impedia a compilação, bem como criar uma main e funções de print para garantir que os dados de fato estavam sendo carregados adequadamente.

Com as listas de records, seguiu-se para a etapa de filtro e transformação dos dados. Foi requisitado que os pedidos sejam filtrados para um determinado tipo de origem e status. Para isso, adotou-se a seguinte estratégia:

1. Foi criada uma função com pattern matching que abre o record de order e checa se ele é igual aos que foram passados de antemão, como argumento da função.
2. Caso ele seja igual, o id desse order é adicionado como cabeça da lista que é passada como argumento (o acumulador)
3. Caso contrário ele mantém o acumulador inalterado
4. Desse modo a função de filter se resumiu a fazer a aplicação dessa função para cada elemento da lista de orders e uma lista vazia como acumulador inicial.

A função retorna uma lista com ids de orders que atendem aos requisitos.

Com essa lista de ids filtrados, seguiu-se para a próxima etapa, a transformação desse dado, com os cálculos para retornar as informações agregadas. Aqui, novamente foi necessário a criação de um novo tipo customizado, um record que represente o dado de saida dessa transformação, com o order id, receita total e total de impostos.

Com o tipo adequado, foi criada uma função que é composta por uma série de funções, que serão detalhadas à frente. Por hora, vamos a ideia geral por trás da implementação. Para cada um dos order ids na lista filtrada, precisamos selecionar os order items que correspondem àquele order id, então a primeira etapa é filtrar os order items que tem exatamente o mesmo order id do iésimo elemento da lista. Com os order items filtrado, podemos criar uma lista com os amounts de cada order item, fazendo um map da multiplicação entre preço e quantidade, e uma lista de imposto, multiplicando o preço pela quantidade e a taxa. A partir dessas informações, podemos realizar uma soma dos amounts e das taxes para chegar aos valores agregados.

Para implementar isso, foram criadas as seguintes funções:

1. transform_orders_to_agg_info: responsável por receber a lista de ids de interesse e todos os order items, realizar o map para aplicar a função de agregação sobre cada um deles.
2. agg_order_info_of_id: recebe os order items e um id, filtra os order items com aquele id, através de uma função de pattern mattching, em seguida realiza os maps para gerar as listas de amounts e taxes que em seguida são colapsadas para o somatório usando o fold_left.

Com isso, foi realizada a transformação de volta para Csv.t e salvo em um arquivo csv novamente. Para essa etapa, novamente foi utilizada a LLM para criar uma main de teste e pequenos ajustes que impedia a build.

Para o agregado mensal, foi realizado o mesmo processo, porém ao invés de aplicar sobre os ids, foi aplicado sobre um conjunto “ano-mês”. Para essa etapa e usando como referência os códigos já desenvolvidos, a LLM criou funções para determinar o range de tempo dos dados, buscando a menor e a maior data e criando uma lista de “ano-mês” e em

seguida aplicou a estratégia citada de filtrar os order ids baseado na data e em seguida aplicando a função de agregação criada anteriormente para determinar a média de receita e imposto.

Por fim, vale ressaltar que o código apresenta docstrings de todas as funções criadas (criado a partir da LLM).