The Business Game – NTTDATA X LUISS

Simone Lu, Lorenzo Antolini, Enrico Romano, Luca Schisano

Algorithmic approach

Index

Data Exploration

Price Distribution in the product category

To identify the main business areas of interest for the company, the starting categories of products have been grouped with the following function in macro-categories:

From product category to Label

```
def label_states(row):
    if row['product_category_name'] in ["book", "comics", "sportoutdoors",
"musicinstruments", "agriculturesuppliers", "modelhobbybuilding",
"painting",
"fabric", "film&photography", "camera&photo", "cdvinyl", "wallart",
"event&partysupplies"]:
        return 'hobby'
    if row['product_category_name'] in ["food", "dietsportsnutrition"]:
        return 'food'
```

```
    if row['product_category_name'] in ["homeaccessories", "furniture",
"officeproducts", "kitchen&dining", "seasonaldecor", "ceilingfans",
"businessoffice",
"bedroomdecor", "sofa", "bakeware", "lawngarden",
"mattresses&pillows","toolshomeimprovement"]:
        return 'forniture'
    if row['product_category_name'] in ["fragrance", "beauty&personalcare",
"wellness&relaxation", "oralcare"]:
        return 'beauty'
    if row['product_category_name'] in ["automotive"]:
        return 'automotive'
    if row['product_category_name'] in ["computeraccessories",
"cellphones", "television&video", "homeappliances", "hardware",
"homelighting",
"lightbulbs", "videogameconsole", "homeaudio", "coffeemachines",
"headphones", "monitors", "computerstablets", "dvd", "videogame"]:
        return 'eletronic'
    if row['product_category_name'] in ["toysgames", "kids",
"kidsfashion"]:
        return 'kids'
    if row['product_category_name'] in ["petsupplies", "petfood"]:
        return 'pet'
    if row['product_category_name'] in ["underwear", "fashion&shoes",
"men'sfashion", "woman'sfashion"]:
        return 'clothes'
    if row['product_category_name'] in ["handbags&accessories", "luggage",
"watches", "jewelry"]:
        return "accessories"
    if row['product_category_name'] in ["homeemergencykits", "firesafety",
"healthhousehold", "medicalsupplies", "homesecuritysystems",
"cleaningsupplies","safetyapparel"]:
```

```
        return 'safety'
    else:
        return "other"
df_final['label'] = df_final.apply(lambda row: label_states(row), axis=1)
```

The distribution of prices of the various product categories within the dataset was analysed for marketing analysis. Histogram and KDE analysis have been used for this purpose.

- **Histogram**

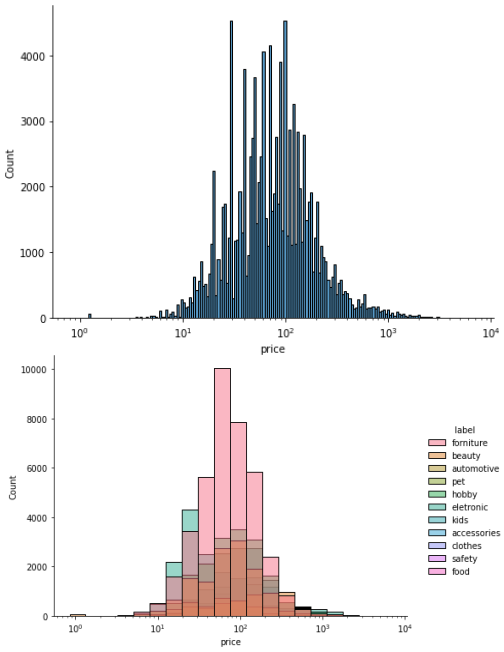
It divides the value range of continuous variables into discrete bins and shows how many values exist in each bin.

Focusing the attention on the price distribution for the category of product different points can be noticed:

1) home furniture goods are the most common product sold around the 80€ range.

2) the most common goods in the 20€ - 50€ range are kids-related products.

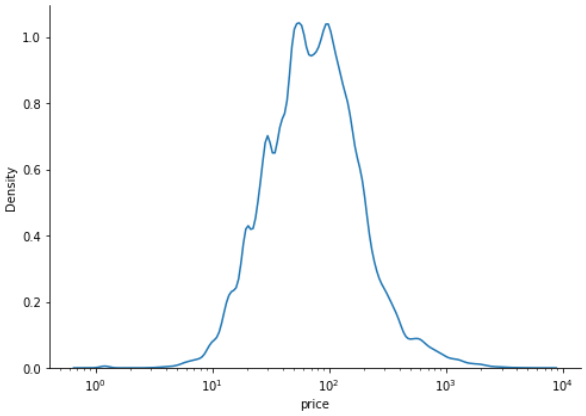
3) home furniture goods represent also the main products sold in the e-commerce.



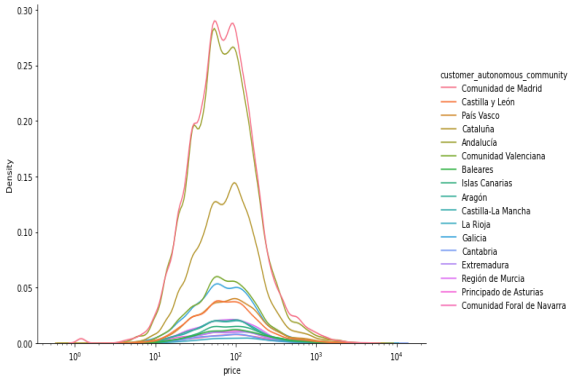
KDE

KDE plots can be used for visualizing the distribution of variables as well. They are quite similar to the histograms. However, a KDE plot represents the distribution using a continuous probability density curve rather than with discrete bins.

The graph highlights the density distribution of the variable price in the dataset. As supposed before, the density distribution of the price variable tends towards values close to 80€.



Looking deeply at the distribution of the price by analysing the results focusing on the autonomous community of the customer. The earlier analysis is strongly influenced by the fact that most orders come from the three most populous population centres in Spain (Madrid, Barcelona, and Andalusia). In fact, if we compare the distributions for these three autonomous communities, we notice that there is a very strong similarity to the total distribution curve.



Clustering Technique: RFM analysis

For the task of identifying groups of customers who have similar purchasing behaviours, hence clustering.

We have proceeded with RFM analysis, more than an algorithm it is a series of statistical calculations that allows us to classify customers based on three metrics:

R - How recently did the customer purchase?

F - How often do they purchase?

M - How much do they spend?

```
RFM_table = RFM_data.groupby('customer_id').agg({'ts_order_purchase':
```

```
                                    lambda x: (NOW -x.max()).days,'order_id':
                                    lambda x: len(x), 'transaction_value':
                                    lambda x: x.sum()})
```

We created a table by grouping the customer id, so avoid the repetition of the customer's NAME/ID. With lambda, which is a function that takes one argument, and that argument will be applied to the rest of the dataset by just recalling them.

For the Recency we calculated the last day of the transaction recorded in order to use it to calculate the Recency (example: Last recorded trans. - day x).

For the frequency, we just summed the number of transactions to know how many time the customer ordered.

For how much a customer spend we did x.sum(), so a summation of the transactions.

We have used **Quintiles** — Make four equal parts based on available values — to calculate the RFM score.

```
quantiles = RFM_table.quantile(q=[0.25,0.5,0.75])
```

Then in order to attribute the RFM score, we defined two functions, one for Recency and the other for Frequency and Monetary. So, we attributed than value to a quintile with function that contains if else condition.

```
def R_Class(x,p,d):
        if x <= d[p][0.25]:
                    return 4
        elif x <= d[p][0.50]:
                   return 3
        elif x <= d[p][0.75]:
                   return 2
        else:
                    return 1
```

```
def FM_Class(x,p,d):
    if x <= d[p][0.25]:
        return 1
    elif x <= d[p][0.50]:
        return 2
    elif x <= d[p][0.75]:
        return 3
    else:
        return 4
```

Finally based on the definition of quintiles. We can classify a customers based on the score by just unify the quartiles not numerically.

```
RFM_Segment['RFMClass'] = RFM_Segment.R_Quartile.map(str) \
 + RFM_Segment.F_Quartile.map(str) \
 + RFM_Segment.M_Quartile.map(str)
```

So, if RFM score is 444, it means that this customer belongs to Recency quintile 4 of Recency, Frequency and Monetary.

Finally, we classified the customers base into four classes: BFF, Lost, Attention seaker and Loyal.

Hierarchical clustering

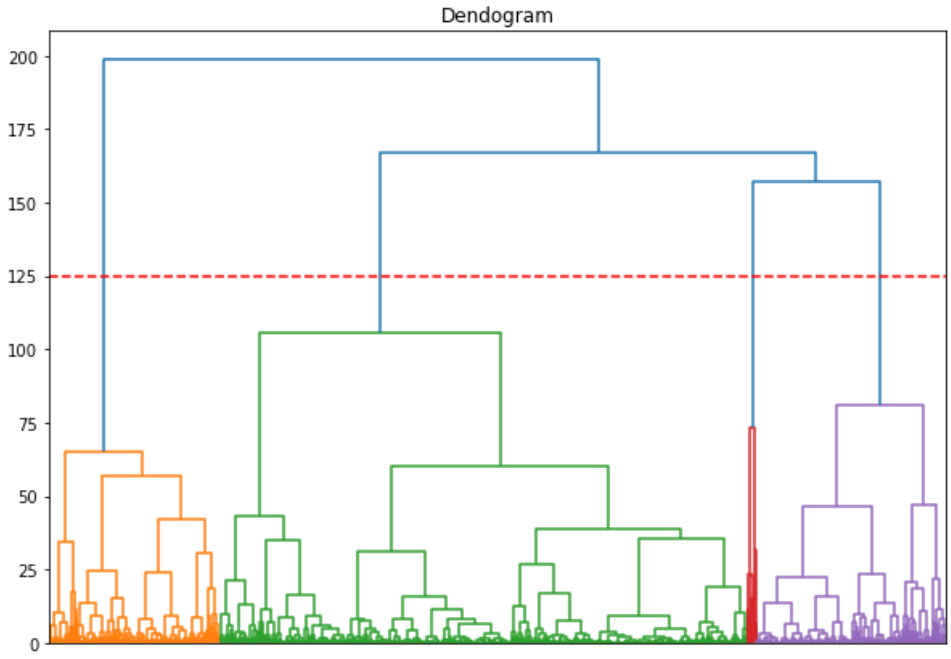
This algorithm tries to identify different group of people based on some features and the similarity among them. It's a bottom-up based method that starts with every observation as a single cluster and then merge if there are similarities between them. It uses a dendrogram to identify the right number of clusters.

The first step is: select the features that I want to use to cluster my data and I have used product_category_name' (changed in a numerical category thanks to hot coding), 'review_score' and 'price'. The customers_id replaces the index column to trace the ID of every customer. I filter my dataset for one-chosen community, and then plot every single customer's feature in a 3D graph, subsequently cluster the customers, based on those features, in how many clusters I decide to give to the 'cluster3D' function.

```
def cluster3D(Comunity, number_of_cluster):
    df = df_final[df_final['customer_autonomous_community']==Comunity]
    df.reset_index(inplace=True,drop=True)
    df = df[['customer_id','product_category_name', 'review_score','price']]
    df = df.set_index('customer_id')

    labelencoder_X_1 = LabelEncoder()
    df['product_category_name'] = labelencoder_X_1.fit_transform(df['product_category_name'])

    X3 = df[['product_category_name','price','review_score']]

    x = X3.iloc[:,0]
    y = X3.iloc[:,1]
    z = X3.iloc[:,2]

    hc3 = AgglomerativeClustering(n_clusters = number_of_cluster, affinity = 'euclidean', linkage = 'ward')
    y_hc3 = hc3.fit_predict(X3)

    fig = plt.figure(figsize=(5,5))
    ax = Axes3D(fig)

    ax.scatter(x,y,z,s=40,c = y_hc3,marker='o',alpha=1)

    ax.set_xlabel('product_category_name')
    ax.set_ylabel('price')
    ax.set_zlabel('review_score')
```

Eventually I filter the dataset for every customer that have made more than 2 orders in the dataset and made some cluster based on just two features: first category product and price secondly price and review score.

Then I go back to analyse the main 3 features that I have used in the 'cluster3D' function but this time since they have scale-value different among them I decide to standardize the value thanks to 'StandardScaler' function.

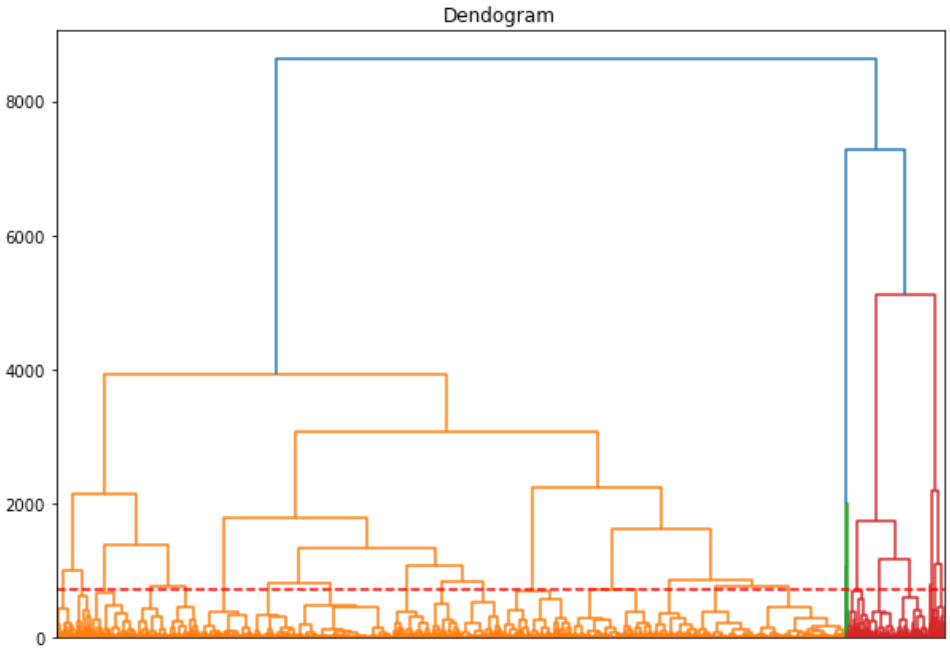


I plot the dendrogram and see what the best number of clusters within the dataset is: in this case 4. I compute the cluster prediction and see how customers are spread into clusters.

In the last study every feature contributes to the same weight in defining the clusters, but I want to give more weight to the price features and less to the others two variables because the feature price is the most important, in our opinion, to cluster different type of customers.

```
data_weighted['product_category_name'] = data_weighted['product_category_name']*20
data_weighted['price'] = data_weighted['price']*60
data_weighted['review_score'] = data_weighted['review_score']*20
```

So, I repeat the study this time with the dataset weighted and I point out that the best number of clusters decrease to 3.



We decided to not use this method for implementing a market strategy because both the few categories used in the process of this method and because the incapability in terms of computation power to process and develop a dendrogram with a huge amount of observation.

Recommendation System

In order to help the user to find out information about the product, recommendation systems where developed. These systems create a similarity between the user and items and exploits the similarity between user/item to make recommendations.

A first step of our analysis is the features selection; in order to create a recommendation system, three variables are selected: customer_id, product_id and review_score. Then, we count the number of unique values inside the columns and checking for duplicates.

After this step of feature selection, an exploratory data analysis is performed. So, through the histogram and the joinplot, we displayed the most common review. Moreover, we discovered the thirty best-selling products.

Then, we move on to the fundamental step of the recommendation system: the creation of the algorithm. We have chosen the collaborative filtering. Collaborative filtering is commonly used for recommender systems. These techniques aim to fill in the missing entries of a user-item association matrix. We are going to use collaborative filtering (CF) approach. It is based on the idea that the best recommendations come from people who have similar tastes. In other words, it uses historical item ratings of like-minded people to predict how someone would rate an item. Collaborative filtering has two sub-categories that are generally called:

- Model based approach: These methods are based on machine learning and data mining techniques. The goal is to train models to be able to make predictions:
  - SVD.
- Memory based (item based) approach:
  - A basic collaborative filtering algorithm. (**KNNBasic model**)
  - A basic collaborative filtering algorithm, taking into account the mean ratings of each user. (**KNNWithMeans model**)
  - A basic collaborative filtering algorithm, taking into account the z-score normalization of each user (**KNNWithZScore model**)
  - A basic collaborative filtering algorithm taking into account a baseline rating. (**KNNBaseline model**).

The main difference between these approaches is that for item based models, **algo.get_neighbors(product_inner_index , k = number_of_nearest_neighbors)** will produce the nearest neighbors to a product, if purchased by user then the neighbors of that product are to be recommended to the user, however for the model-based methods **algo.predict(user_id, product_id)** can be used to create a list of recommended items.

For this analysis, we have performed only three of these algorithms: SVD, KNNBasic and KNNwithMeans. All these algorithms are in the “Surprise" library.

Firstly, we grouped the dataset by product_id and considered only users that have made more than 3 reviews, because we consider them as ‘” loyal” customers. Then, we counted the average value of the reviews left by each customer. From this new filtered dataset, we built our collaborative filtering algorithms.

To begin with, we split the dataset into train and test and compute the RMSE for each algorithm. From this, we can notice that the value is almost the same. So, we expect similar outcomes.

```
Item-based Model : Test Set
KNNBasic Model 1.3442786520929786
KNNWithMeans Model 1.3479967347748536


Model-based Model : Test Set
SVD Model 1.3556817351779884
```

Then, we have created a function that choose a random user from the test set predict the recommended products for him with each of the computed methods with the little loop in the figure below.

```
items_purchased = trainset.ur[trainset.to_inner_uid(uid)]


print("Choosen User has purchased the following items ")
for items in items_purchased[0]:
    print(algo_KNNBasic.trainset.to_raw_iid(items))



#getting K Neareset Neighbors for first item purchased by the choosen user
KNN_Product = algo_KNNBasic.get_neighbors(items_purchased[0][0], 15)

recommendedation_lits = []
for product_iid in KNN_Product:
    if not product_iid in items_purchased[0]: #user already has purchased the item
        purchased_item = algo_KNNBasic.trainset.to_raw_iid(product_iid)
        recommendedation_lits.append(purchased_item)

recommendedation_list=pd.DataFrame(recommendedation_lits,columns=['recommended product'])

print("Recommended items for user " + str(uid) + " by KNNBasic \n"  , recommendedation_lits)
```

Firsly, we defined the list of items purchased from the random user, (in the example we used the KNNBasic), then we computed the KNN for first item purchased by this user, and started a for-loop in order to identify the recommendation list. In conclusion, we printed this list.

So, as said before, for item-based model (KNNwithMeans, KNNbasic), we had the nearest neighbours to a product as outcome. While for model based (SVD) we had a list of recommended items.

Last part of the recommendation systems is model evaluation. To compare each model prediction a Predictions Dataframe needs to be created. So, we create a function that return to us some crucial information such as the number of items, the user id, the expected review, the actual review and the error.

```
def get_Iu(uid):
    """ return the number of items rated by given user
    args:
      uid: the id of the user
    returns:
      the number of items rated by the user
    """
    try:
        return len(trainset.ur[trainset.to_inner_uid(uid)])
    except ValueError: # user was not part of the trainset
        return 0

def get_Ui(iid):
    """ return number of users that have rated given item
    args:
      iid: the raw id of the item
    returns:
      the number of users that have rated the item.
    """
    try:
        return len(trainset.ir[trainset.to_inner_iid(iid)])
    except ValueError:
        return 0
```

With these def functions, we define two functions that return us: the number of items rated by a given user and the number of users that have rated given item.

Subsequently, we create three datasets for each model:

- Best predictions dataset.
- Worst predictions dataset.

- Main dataset with all the predictions done.

In particular from the last dataset, we obtain some fundamental information about the performance of the models. As a result, we have that all the models have basically the same performance and also the predictions have the same error rate, as we expected from the RMSE calculation.

Time Delivery Prediction

In order to predict an accurate date of delivery, with the following code, we will try to predict a delivery time, with this predicted delivery time, we can know when the package will reach the customer.

Firstly, we merged together customers, orders, order_items and products in order to consider some important features for the delivery prediction:

```
merge_df=customers.merge(orders,on='customer_id',how='inner')
```

```
merge_df=merge_df.merge(order_items,on='order_id',how='inner')
```

```
merge_df=merge_df.merge(products,on='product_id',how='inner')
```

We have extracted the month from the ts_order_purchase variable, because it represents our main interest. Also, we have transformed the other orders variables from object to datetime. Looking at our data, it's easy to see all the "id" related fields are not relevant in predicting delivery time, thus all of these will be removed. as for the purchase timestamp, we've already extracted the month into the 'purchase month' column, which is the relevant parameter we need. After, we create our target variable: **delivery time**.

```
param_df['delivery_time']=(merge_df['ts_order_delivered_customer']-merge_df['ts_order_purchase']).dt.days
```

After this step, we started to predict delivery time. In order to do this, we have chosen three main algorithms: XGBoost, RandomForest and Artificial Neural Network. As a first step, we have done some feature engineering, so we removed the NaN values, encoded categorical

variables and select our X and Y. Then, we have done a standardization of X and Y and we splitted data in train, test and validation set.

```
X=preprocessor.fit_transform(X)

X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,random_state=0)
scaler = MaxAbsScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
```

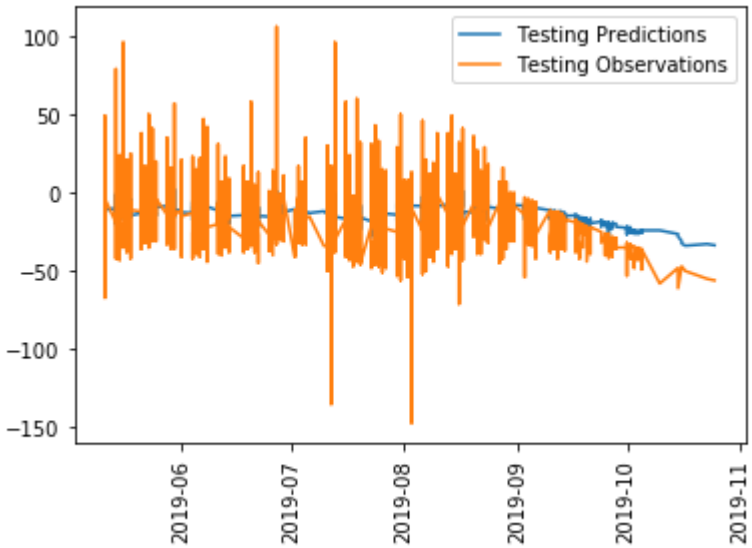
As evidenced, attempts to predict delivery time with two powerful ensemble methods: XGB and Random Forest, have shown very unreliable results. Furthermore, current attempts to either use additional features as well as reducing our models' complexity have shown similarly poor results. Looking at our target data gives us a strong context for the current inability to predict the time of delivery, while the mean delivery time is a reasonable time of twelve days, the standard deviation is almost 63 for XGboost and 81 for RandomForest. Furthermore, we see some very extreme outliers, with the extreme maximum of 209 days, well over half a year. A last attempt at improving our model by cutting off highly delayed deliveries would still maintain a high standard deviation perpetuating poor model performance. This record shows that this Spanish E-Commerce's delivery system was inefficient and thus makes any hopes to properly predict the delivery time rather tantalising. We used the LSTM algorithm to predict the trend of difference between the arrival date and estimated arrival date of the previous order.

```
time = []
i=0
while i < len(df_final):
    time_1 = datetime.datetime.strptime(df_diff['estimated'][i],'%d/%m/%Y %H:%M')
    time_2 = datetime.datetime.strptime(df_diff['delivered'][i],'%d/%m/%Y %H:%M')
    time_interval = time_2 - time_1    #differenza tra il delivered e l'estimato
    time.append(time_interval)

    i=i+1
```

We try to predict how will performance our company in the feature deliveries with an LSTM that look back in the last five orders. We split the dataset into train (70%) and test (30%) and scaled the difference values. The architecture of our model is made up four layers.

```
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_5.shape[1],1)))
model.add(LSTM(20, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(loss='mse',
              optimizer='adam',
              metrics=['mean_absolute_error'])
```

As cost function I have used a mean absolute error. At the end the LSTM predict the future trend of the next orders as negative trend, so we understand that the company needs to change something in order to accomplish the goal of delivery object in time with the estimated delivery.



In order to find ways to optimise delivery times, we are focusing on logistic regression. Our goal here is to find deliveries that have suffered and to suggest solutions to improve the customer experience. It is natural that this requirement requires us to analyse the dataset of deliveries and their columns. In order to do a better analysis, is important to have the zip codes of the Spanish cities. So, we created this variable and add it to our delivery's dataset. We have noticed that there is a good chance of gaining insights by comparing the columns "ts_order_estimated_delivery" and "ts_order_delivered_customer", which represent the

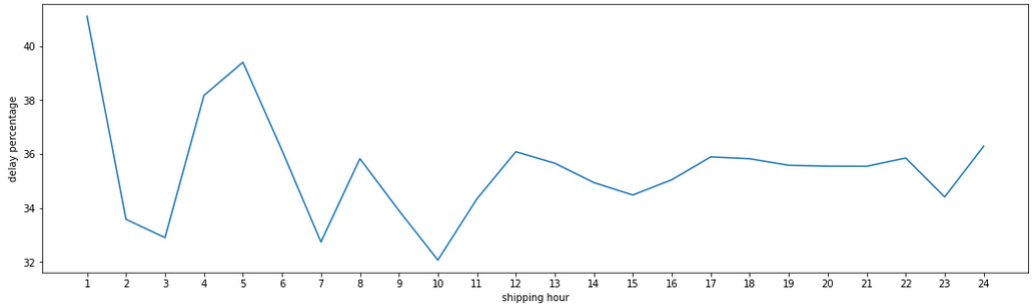
estimated time the product arrived at the customer's address, and the time the product actually took to arrive. Based on this, we created a new column called "diff ". which represents the difference between the estimated date and the actual date. We have a hypothesis. Maybe these delays are due to some process in the delivery of the product to the carrier? The "ts_order_delivered_carrier" column can help us verify this.

In order to do this verification, we have transformed our variable in a format that can be worked with mathematical operations. Then, we checked deliveries to the carrier/post office that occurred after 6pm and the deliveries that caused the most delivery delays to the end customer. We obtain a dataset, in which in the left column we see the time and in the right the number of delayed deliveries. We can see a pattern, most of the delays happened after midday and before midnight. Then, we repeat the same analysis in order to find a pattern also in the early delivered goods. Knowing that in this dataset there are many earlier (or on time) deliveries than late ones, direct comparison did not help us. So, we have calculated the delay proportion between the schedules and plot it. To do this we'll need to create a small repeating loop.

```
percentage_delay_shipping_time = {}


for i in range(24):
    hour = i+1
    if hour < 24:
        percentage_delay = len(deliveries[(deliveries['shipping_time'] == hour) &
(deliveries['diff'] < 0)])/len(deliveries[(deliveries['shipping_time'] == hour)])
        percentage_delay_shipping_time[hour] = percentage_delay
    else:
```

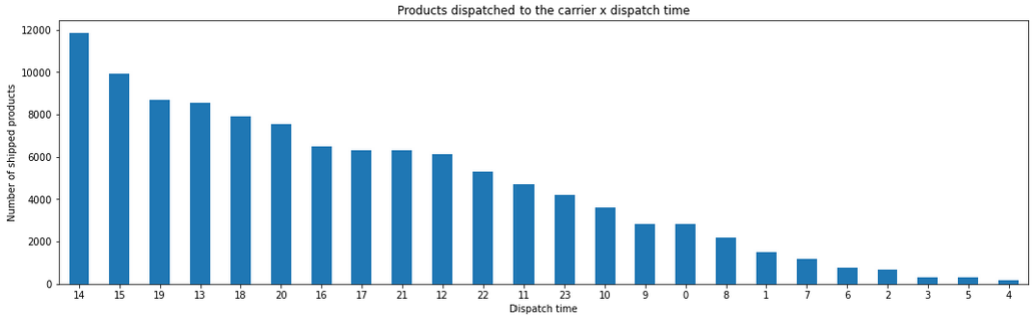
percentage_delay = len(deliveries[(deliveries['shipping_time'] == 0) & (deliveries['diff'] < 0)])/len(deliveries[(deliveries['shipping_time'] == 0)])

percentage_delay_shipping_time[hour] = percentage_delay



Apparently, shipping times with the lowest delay percentage are from 9am to 10 am and 3 a.m.. Of course, is impossible for a seller to ship a product at 3 a.m., so we do not consider this time hour.

Now, we checked out in which of these times, sellers of the Spanish e-commerce ship the most products.



Our sellers have been dispatching most products at 2pm, 3pm, which are not the best performing times for delivery without delay. There is likely to be a reduction in the delays that exist today if sellers are directed to make deliveries to the carrier/post office at the times of 9am, 10am . But this hypothesis needs to be tested. So, we created a test statistic for that hypothesis. Our stat test is equal to 0.76. The hypothesis would already be nullified if the difference between the averages of group A (deliveries at 2pm) and group B (deliveries at 10am) were equal to zero. There is indeed a mathematical difference between the two groups,

but now we will verify that this is not due to chance by calculating Test Statistics and the Significance Level, represented in the p-value, which should result in a number below 0.5 to confirm our hypothesis.

We will need to create a large group with a slice of the dataset, including the difference values of estimated x actual delivery for products dispatched 2pm and those dispatched at 10am.

```
p_value
```

```
0.0
```

With p_value equal to 0.0, we can say that it is statistically relevant the difference in the outcome in delivery delays between groups A and B, and these differences are not given by chance. That is, the hypothesis has statistical power.

In conclusion, there is a good opportunity to reduce the number of late deliveries by simply switching the most frequent times of dispatching goods to the carrier. In the interval that this data was generated Spanish e-commerce sellers were using the 2pm, 3pm times to ship most goods, but could have had fewer delays if they used the 9am, 10am.