Nir Ben Eliahu – MADGROWTH

Assignment 1: Olist Store

Technological tools

This assignment is explored and analyzed using MSSQL and Jupyter Notebook. The Jupyter notebook is the frontend of the work and contains all of the codes and queries that used to perform the exploratory data analysis (EDA).

The combining of MSSQL and jupyter notebook is described in my linkdin article click here to read my article on linkdin (https://www.linkedin.com/pulse/combining-jupyter-sql-r-nir-ben-eliahu).

About the Dataset

2016-2018 performance data of a Brazilian e-commerce marketplace called Olist Store, with information from 100K orders.

Tasks

Your assignment is to research this data and analyze it using SQL/Excel/Python or any other tool you prefer. You are then asked to summarize your top 3 insights from your research, detail your thought process for each insight and present the business logic and the queries you wrote to reach this conclusion.

Strategy to start the analysis

A good place to start the analysis is the orders table.

The data is about an e-commerce store, and its success, performence, and target of customers can be described ny the orders history. the orders can provide information like:

1. orders status
2. the num of orders
3. the evolution of number of order by time
4. the customers prefered order time

Combining the orders with the customers and the sellers

Combining all of this data will provide information on the orders by state and city - the target customers:

1. which state and city have the top buying customers?
2. which state and city have the top seller?

Products that are orderd from the store

After learning about the store performence, the products data holds information the store's potential:

1. what are the best selles products?
2. what are the largest product category in terms of num of different products?
3. are the best sellers product also has the largest categories?

summary insights

1. Orders

- Olist Store has 97% rate of delivering its orders. Only 0.6% of the orders were canceled and only 0.6% of the orders could not be completed due to unavailable in the products.
- From October 2016 to December 2016 the store is probebly in the lunching phase, with less the 500 orders per month.
- From December 2016 to October 2017 the store is perfoming very well reaching 4,000 orders per month.
- November 2017 is the month with the highest number of orders (black firday?)
- In 2018 the store peeks to over 7,000 orders per month.
- the store (E-commerce) has a growing trend along the time. people buy more things online than before.
- Monday, Tuesday,and Wednesday are the most prefered days for brazilian's customers
- customers tend to buy more at afternoons and in the mornings
- from 2017 until 2018 the orders distribution between states is witout change, with SP (São Paulo?) as the state with the highest number of buying customers.
- SP (São Paulo?) num of orders also increases in time and its cut from the entire orders increases.

2. Products

- There are 32,951 products and 73 product categories
- The 10 largest product categories are: bed_bath_table, sports_leisure, furniture_decor, health_beauty, housewares, auto, computers_accessories, toys, watches_gifts, telephony
- Increase of orders over time from categories:
  - watches_gifts
  - health_beauty
  - housewares
- Decrease of orders over time from categories:
  - telephony
  - sports_leisure
  - garden_tools
- no-change of orders over time from categories:
  - furniture_decor
  - bed_bath_table
  - auto

3. Customers

There are 96,096 unique customers. among them, 3,345 are returning customers, 3.5% The states with the highest orders are: SP, RJ, MG, RS, PR, SC, BA, DF, ES, GO

Preperation

Loading Python relevant libraries

In [1]:
```
# Import python libraries
import pandas as pd
import numpy as np

# Import Date manipulation libraries
from datetime import datetime, timedelta
from matplotlib.dates import DateFormatter
import matplotlib.dates as mdates
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

# Import visualization libraries
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
%matplotlib inline

# Import sql Extensions librarys
import sqlalchemy
import pyodbc
from sqlalchemy_utils import database_exists, create_database
```

Specific Notebook Parameters

In [2]:
```
# Set the graphics settings for the visualizations
sns.set(font_scale=1.5)
sns.set_style("ticks")
```

Define functions to create visualizations

Define a line chart function

In [3]:
```
def line_chart_month(x_date, y, line_color, width, Line_Label):

    plt.rc('figure', figsize=(12, 7))   # this is to overwrite default aspect of graph t
    fig, ax = plt.subplots()

    monthly_locator = mdates.MonthLocator()
    ax.xaxis.set_minor_locator(monthly_locator)
    ax.plot(x_date, y, color = line_color, lw = width, label=Line_Label)

    ax.get_yaxis().set_major_formatter(
    matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

    # Ensure ticks fall once every other month (interval=2)
    ax.xaxis.set_major_locator(mdates.MonthLocator(interval=2))

    ax.margins(x=0, y=0)

    fig.autofmt_xdate()

    return ax
```

Define a rectangle line to mark areas on the chart, function

In [4]:
```
def add_rectangle_datetime_x_axis(ax, date_list, date_start_index, date_delta, y_max_val
    from datetime import datetime, timedelta
    import matplotlib.dates as mdates
    from matplotlib.patches import Rectangle

    # Create rectangle x coordinates
    startTime_1 = date_list[date_start_index]
    endTime_1 = startTime_1 + timedelta(days = date_delta)
    # convert to matplotlib date representation
    start_1 = mdates.date2num(startTime_1)
    end_1 = mdates.date2num(endTime_1)
    width_1 = end_1 - start_1

    # Create rectangle y coordinates
    hight_1 = y_max_value

    # Plot rectangle
    rect1 = Rectangle((start_1, 0), width_1, hight_1, edgecolor=color, facecolor="none",
    ax.add_patch(rect1)

    return ax.add_patch(rect1)
```

Define a pie chart function

In [5]:
```
def pie_chart_function(df, categorial_column, values_column, title, palette):

    colors = sns.color_palette(palette)

    Categories = df[categorial_column].tolist()
    Values = df[values_column].tolist()
    percent = df[values_column]*100./df[values_column].sum()

    patches, texts = plt.pie(Values, colors=colors, radius=1.2)
    plt.title(title)

    labels = ['{0} - {1:1.2f} %'.format(i,j) for i,j in zip(Categories, percent)]

    sort_legend = True
    if sort_legend:
        patches, labels, dummy =  zip(*sorted(zip(patches, labels, Values),key=lambda x:

    plt.legend(patches, labels, loc='center left', bbox_to_anchor=(1.2, 0.5),fontsize=18
    return plt.show()
```

Define a stacked area plot in percents function

In [6]:
```
def stacked_area_plot_percents_month(pivot_df, title):
    from matplotlib.dates import DateFormatter
    import matplotlib.dates as mdates

    colors = sns.color_palette('bright')

    pivot_df = pivot_df.divide(pivot_df.sum(axis=1), axis=0)
    pivot_df = pivot_df.mul(100)

    ax = pivot_df.plot(kind='area', stacked=True, title=title, color = colors)
    ax.margins(0, 0) # Set margins to avoid "whitespace"

    ax.tick_params(bottom = False)


    # Ensure ticks fall once every other month (interval=2)
    monthly_locator = mdates.MonthLocator()
    ax.xaxis.set_minor_locator(monthly_locator)
    ax.xaxis.set_major_locator(mdates.MonthLocator(interval=2))

    fig.autofmt_xdate()

    ax.legend(loc='center left', bbox_to_anchor=(1.1, 0.5),
              fancybox=True, shadow=True, ncol=1)

    return ax
```

Establish a connection between jupyter and MSSQL

Create a connection througe an Open Database Connectivity (ODBC), Define an API for accessing the MSSQL database

In [7]:
```
SERVER = "DESKTOP-NIR\SQLEXPRESS"
DATABASE = "Olist_Store"
DRIVER = "SQL Server"
USERNAME = "nir"
PASSWORD = "nir123"
DATABASE_CONNECTION = f'mssql://{USERNAME}:{PASSWORD}@{SERVER}/{DATABASE}?driver={DRIVER
```

In [8]:
```
engine = sqlalchemy.create_engine(DATABASE_CONNECTION)

# if the database is does not exist, sqlalchemy will create the database
if not database_exists(DATABASE_CONNECTION):
    create_database(DATABASE_CONNECTION)
else:
    engine.connect()

# the connection command and variable
connection = engine.connect()
```

Load the extensions required to run SQL queries in jupyter

In [9]:
```
%load_ext sql
%sql mssql+pyodbc://nir:nir123@mssql
#parameters: username=nir password=nir123
```

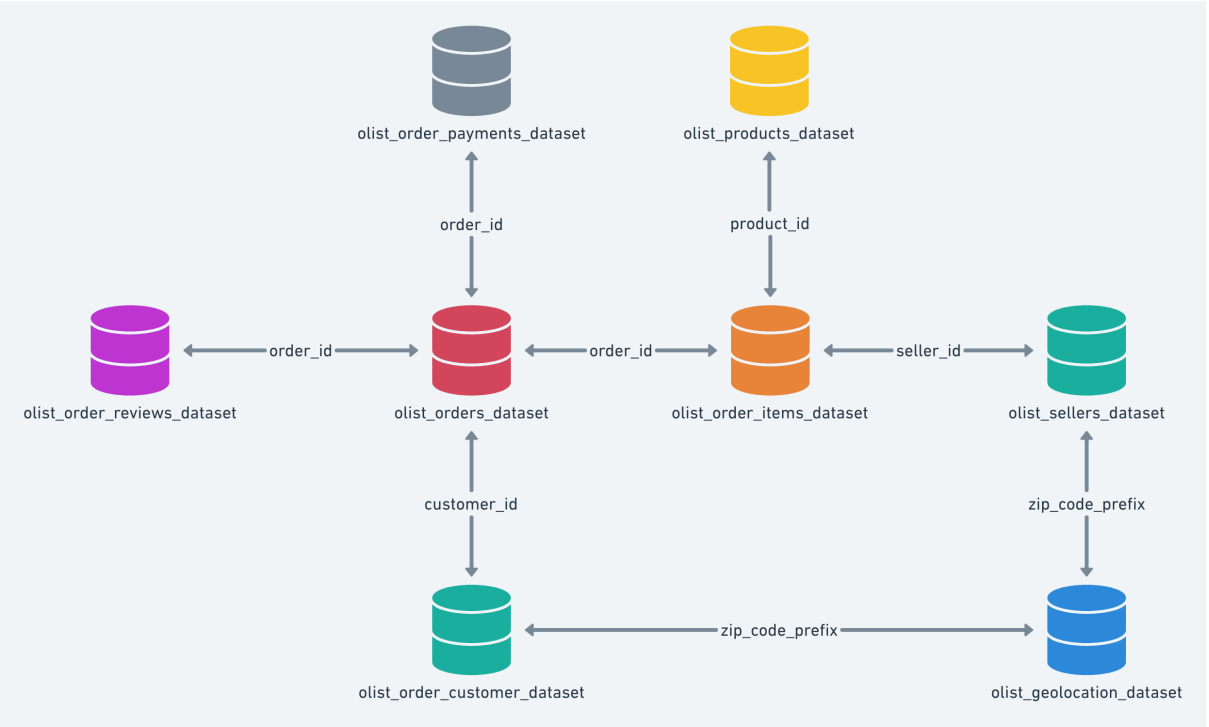
Select the Olist_Store database in MSSQL

In [10]:
```
%%sql
USE "Olist_Store"
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[10]:
```
[]
```

Datasets relationships map:



Loading the Data

Load the data from the local directory

In [11]:
```
customers = pd.read_csv("olist_customers_dataset.csv")
products = pd.read_csv("olist_products_dataset.csv")
orders = pd.read_csv("olist_orders_dataset.csv")
sellers = pd.read_csv("olist_sellers_dataset.csv")

order_items = pd.read_csv("olist_order_items_dataset.csv")
order_payments = pd.read_csv("olist_order_payments_dataset.csv")
order_reviews = pd.read_csv("olist_order_reviews_dataset.csv")

product_category_name_translation = pd.read_csv("product_category_name_translation.csv")

geolocation = pd.read_csv("olist_geolocation_dataset.csv")
```

Understanding the Data

Sampling some rows

In [12]:
```
customers.sample(3)
```

Out[12]:

	customer_id	customer_unique_id	customer_zip_code_prefix	cu
3150	7bd51ad546d78d7678edfe06a435e86f	826df974b989d9b10fba85c7d3020da8	13484	
34766	b79bc49c04e9d0f44712b00179d4a909	ada60145ed9a080e6e3114fe436815e6	20210	r
45115	1755fad7863475346bc6c3773fe055d3	a5314ac290a8b141491e987ae37aa7cc	13454	sa

In [13]:
```
products.sample(3)
```

Out[13]:

	product_id	product_category_name	product_name_lenght	product_descriptio
26843	5bf0a3751ac1a1c1253ef97a027b8c13	alimentos	33.0	
8039	b3e62451b0073eb78f48426d3832b8d2	utilidades_domesticas	55.0	
2734	2f2155ee7545c9ca31ac68685224bc4a	utilidades_domesticas	41.0	

In [14]:
```
orders.sample(3)
```

Out[14]:

	order_id	customer_id	order_status	order_purchase_
20415	5c7932e52b3c1515dbad06328d81d6fa	7dbc7e3964ce88c85cfdcdc61320d724	delivered	2017-12-
62772	20aea91835c11696daa9520693503694	649ccc91c147ec069e6df8ba5ca3ddfc	delivered	2017-09-
74416	3dc3012f8b35d3e48e9a5537d549986f	3f4e232392aebfd6eba1b4e785120bf4	delivered	2017-04-

In [15]:
```
sellers.sample(3)
```

Out[15]:

	seller_id	seller_zip_code_prefix	seller_city	seller_state
259	dff87e4de60c9736ce8df835951b09bc	3172	sao paulo	SP
1880	5f1dc28029d2c244352a68107ec2b542	5126	sao paulo	SP
1912	cc63f0dd2acba93ffed4fe9f8e0321fa	15025	sao jose do rio preto	SP

In [16]:
```
order_items.sample(3)
```

Out[16]:

	order_id	order_item_id	product_id	
11829	1adbc05f8bb9f7f40e1566992a9b5e8d	1	602bd303d85e0e535a0767b9f1d85f91	9b1585752613ec
48982	6f4a1ccdb6d8b5f5249737581a50b1ea	1	1a986b4015f20a3c26d8573a46dcf65e	4cf490a5825928
28406	40a9be4510459ae28841128e5938153f	1	695e7998d136c389e5b287c1ffe4b62a	165fc07beebdcl

In [17]:
```
order_payments.sample(3)
```

Out[17]:

	order_id	payment_sequential	payment_type	payment_installments	payment
65146	d56e764f8fe72cb1656efbbc36d504d1	1	credit_card	8	
16894	5f4fb6276021ebd35972ce9dcccfed59	1	credit_card	1	
66976	2ba6f4043547bc847cc055c975cb57cf	1	credit_card	10	

In [18]:
```
order_reviews.sample(3)
```

Out[18]:

	review_id	order_id	review_score	review_commen
71692	1fcd175bc4c3cf106e4f26cb6a1aa41b	896128ba70225c31859100341b20b186	5	
25745	e72754e4691488167f0cec3841504c6f	1b37c11d1b330f2f5a3ad3676bb9c07d	5	
98168	872073acf20af03791f23530c94845bc	8b1956ed3dc60c9bfa629f6e215fd306	3	

In [19]:
```
product_category_name_translation.sample(3)
```

Out[19]:

	product_category_name	product_category_name_english
61	musica	music
68	fraldas_higiene	diapers_and_hygiene
25	construcao_ferramentas_construcao	construction_tools_construction

In [20]:
```
geolocation.sample(3)
```

Out[20]:

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
751088	64003	-5.080877	-42.819395	teresina	PI
941197	90240	-29.993219	-51.193760	porto alegre	RS
472225	24210	-22.902421	-43.134019	niteroi	RJ

Data Dictionary

1. customers:
   - **customer_id** - key to the orders dataset. Each order has a unique customer_id.
   - **customer_unique_id** - unique identifier of a customer.
   - **customer_zip_code_prefix** - first five digits of customer zip code
   - **customer_city** - customer city name
   - **customer_state** - customer state
2. products:
   - **product_id** - unique product identifier
   - **product_category_name** - root category of product, in Portuguese.
   - **product_name_lenght** - number of characters extracted from the product name.
   - **product_description_lenght** - number of characters extracted from the product description.
   - **product_photos_qty** - number of product published photos
   - **product_weight_g** - product weight measured in grams.
   - **product_length_cm** - product length measured in centimeters.
   - **product_height_cm** - product height measured in centimeters.
   - **product_width_cm** - product width measured in centimeters.
3. orders:
   - **order_id** - unique identifier of the order.

- **customer_id** - key to the customer dataset. Each order has a unique customer_id.
- **order_status** - Reference to the order status (delivered, shipped, etc).
- **order_purchase_timestamp** - Shows the purchase timestamp
- **order_approved_at** - Shows the payment approval timestamp.
- **order_delivered_carrier_date** - Shows the order posting timestamp. When it was handled to the logistic partner.
- **order_delivered_customer_date** - Shows the actual order delivery date to the customer.
- **order_estimated_delivery_date** - Shows the estimated delivery date that was informed to customer at the purchase moment.

4. sellers:
   - **seller_id** - seller unique identifier
   - **seller_zip_code_prefix** - first 5 digits of seller zip code
   - **seller_city**- seller city name
   - **seller_state**- seller state
5. order_items:
   - **order_id**- order unique identifier
   - **order_item_id** - sequential number identifying number of items included in the same order.
   - **product_id**- product unique identifier
   - **seller_id**- seller unique identifier
   - **shipping_limit_date**- Shows the seller shipping limit date for handling the order over to the logistic partner.
   - **price** - item price
   - **freight_value**- item freight value item (if an order has more than one item the freight value is splitted between items)
6. order_payments:
   - **order_id**- order unique identifier
   - **payment_sequential** - a customer may pay an order with more than one payment method. If he does so, a sequence will be created to
   - **payment_type**- method of payment chosen by the customer.
   - **payment_installments**- number of installments chosen by the customer.
   - **payment_value**- transaction value.
7. order_reviews:
   - **review_id** - unique review identifier
   - **order_id** - unique order identifier
   - **review_score** - Note ranging from 1 to 5 given by the customer on a satisfaction survey.
   - **review_comment_title** - Comment title from the review left by the customer, in Portuguese.
   - **review_comment_message** - Comment message from the review left by the customer, in Portuguese.
   - **review_creation_date** - Shows the date in which the satisfaction survey was sent to the customer.
   - **review_answer_timestamp** - Shows satisfaction survey answer timestamp.
8. product_category_name_translation:
   - **product_category_name** - category name in Portuguese
   - **product_category_name_english** - category name in English
9. geolocation:
   - **geolocation_zip_code_prefix** - first 5 digits of zip code
   - **geolocation_lat** - latitude
   - **geolocation_lng** - longitude
   - **geolocation_city** - city name
   - **geolocation_state** - state

Cleanup

Duplicates Rows

In [21]:
```
customers[customers.duplicated()].head()
```

Out[21]:

customer_id	customer_unique_id	customer_zip_code_prefix	customer_city	customer_state

In [22]:
```
products[products.duplicated()].head()
```

Out[22]:

product_id	product_category_name	product_name_lenght	product_description_lenght	product_photos_qty

In [23]:
```
orders[orders.duplicated()].head()
```

Out[23]:

order_id	customer_id	order_status	order_purchase_timestamp	order_approved_at	order_delivered_carrier

In [24]:
```
sellers[sellers.duplicated()].head()
```

Out[24]:

seller_id	seller_zip_code_prefix	seller_city	seller_state

In [25]:
```
order_items[order_items.duplicated()].head()
```

Out[25]:

order_id	order_item_id	product_id	seller_id	shipping_limit_date	price	freight_value

In [26]:
```
order_payments[order_payments.duplicated()].head()
```

Out[26]:

order_id	payment_sequential	payment_type	payment_installments	payment_value

In [27]:
```
order_reviews[order_reviews.duplicated()].head()
```

Out[27]:

review_id	order_id	review_score	review_comment_title	review_comment_message	review_creation_date

In [28]:
```
product_category_name_translation[product_category_name_translation.duplicated()].head()
```

Out[28]:

product_category_name	product_category_name_english

In [29]:
```
geolocation[geolocation.duplicated()].head()
```

Out[29]:

	geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state
15	1046	-23.546081	-46.644820	sao paulo	SP
44	1046	-23.546081	-46.644820	sao paulo	SP
65	1046	-23.546081	-46.644820	sao paulo	SP
66	1009	-23.546935	-46.636588	sao paulo	SP
67	1046	-23.546081	-46.644820	sao paulo	SP

There are several duplicated rows in the geolocation table, so lets get rid of them.

In [30]:
```
geolocation.drop_duplicates(inplace=True)
```

Check again

In [31]:
```
geolocation[geolocation.duplicated()].head()
```

Out[31]:

geolocation_zip_code_prefix	geolocation_lat	geolocation_lng	geolocation_city	geolocation_state

Null Values

In [32]:
```
customers.isnull().sum()
```

Out[32]:
```
customer_id                0
customer_unique_id         0
customer_zip_code_prefix   0
customer_city              0
customer_state             0
dtype: int64
```

In [33]:
```
products.isnull().sum()
```

Out[33]:
```
product_id                    0
product_category_name       610
product_name_lenght         610
product_description_lenght  610
product_photos_qty          610
product_weight_g              2
product_length_cm             2
product_height_cm             2
product_width_cm              2
dtype: int64
```

Some products are not assigned to any product categorie, and they don't have description

In [34]:
```
orders.isnull().sum()
```

Out[34]:
```
order_id                           0
customer_id                        0
order_status                       0
order_purchase_timestamp           0
order_approved_at                160
order_delivered_carrier_date    1783
order_delivered_customer_date   2965
order_estimated_delivery_date      0
dtype: int64
```

Some products are missing information regading the handling of the order. maybe the data is missing because the orders were regected.

In [35]:
```
sellers.isnull().sum()
```

Out[35]:
```
seller_id                0
seller_zip_code_prefix   0
seller_city              0
seller_state             0
dtype: int64
```

In [36]:
```
order_items.isnull().sum()
```

Out[36]:
```
order_id               0
order_item_id          0
product_id             0
seller_id              0
shipping_limit_date    0
price                  0
freight_value          0
dtype: int64
```

In [37]:
```
order_payments.isnull().sum()
```

Out[37]:
```
order_id                0
payment_sequential      0
payment_type            0
payment_installments    0
payment_value           0
dtype: int64
```

In [38]:
```
order_reviews.isnull().sum()
```

Out[38]:
```
review_id                      0
order_id                       0
review_score                   0
review_comment_title       87656
review_comment_message     58247
review_creation_date           0
review_answer_timestamp        0
dtype: int64
```

Some reviews are missing a title or a messege. this is understandable. sometime, reviewers just rate the order.

In [39]:
```
product_category_name_translation.isnull().sum()
```

Out[39]:
```
product_category_name            0
product_category_name_english    0
dtype: int64
```

In [40]:
```
geolocation.isnull().sum()
```

Out[40]:
```
geolocation_zip_code_prefix    0
geolocation_lat                0
geolocation_lng                0
geolocation_city               0
geolocation_state              0
dtype: int64
```

This dataset contain NaN values, but they might not create problem. so no clean up of NaN values

Address categorial and continuos and date data types

examine if the data is stored in the perfered formats

In [41]:
```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 5 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   customer_id               99441 non-null  object
 1   customer_unique_id        99441 non-null  object
 2   customer_zip_code_prefix  99441 non-null  int64 
 3   customer_city             99441 non-null  object
 4   customer_state            99441 non-null  object
dtypes: int64(1), object(4)
memory usage: 3.8+ MB
```

In [42]:
```
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32951 entries, 0 to 32950
Data columns (total 9 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   product_id                  32951 non-null  object 
 1   product_category_name       32341 non-null  object 
 2   product_name_lenght         32341 non-null  float64
 3   product_description_lenght  32341 non-null  float64
 4   product_photos_qty          32341 non-null  float64
 5   product_weight_g            32949 non-null  float64
 6   product_length_cm           32949 non-null  float64
 7   product_height_cm           32949 non-null  float64
 8   product_width_cm            32949 non-null  float64
dtypes: float64(7), object(2)
memory usage: 2.3+ MB
```

In [43]:
```
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   order_id                       99441 non-null  object
 1   customer_id                    99441 non-null  object
 2   order_status                   99441 non-null  object
 3   order_purchase_timestamp       99441 non-null  object
 4   order_approved_at              99281 non-null  object
 5   order_delivered_carrier_date   97658 non-null  object
 6   order_delivered_customer_date  96476 non-null  object
 7   order_estimated_delivery_date  99441 non-null  object
dtypes: object(8)
memory usage: 6.1+ MB
```

In [44]:
```
sellers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3095 entries, 0 to 3094
Data columns (total 4 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   seller_id              3095 non-null   object
 1   seller_zip_code_prefix  3095 non-null   int64 
 2   seller_city            3095 non-null   object
 3   seller_state           3095 non-null   object
dtypes: int64(1), object(3)
memory usage: 96.8+ KB
```

In [45]:
```
order_items.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 112650 entries, 0 to 112649
Data columns (total 7 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   order_id             112650 non-null  object 
 1   order_item_id        112650 non-null  int64  
 2   product_id           112650 non-null  object 
 3   seller_id            112650 non-null  object 
 4   shipping_limit_date  112650 non-null  object 
 5   price                112650 non-null  float64
 6   freight_value        112650 non-null  float64
dtypes: float64(2), int64(1), object(4)
memory usage: 6.0+ MB
```

In [46]:
```
order_payments.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103886 entries, 0 to 103885
Data columns (total 5 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   order_id              103886 non-null  object 
 1   payment_sequential    103886 non-null  int64  
 2   payment_type          103886 non-null  object 
 3   payment_installments  103886 non-null  int64  
 4   payment_value         103886 non-null  float64
dtypes: float64(1), int64(2), object(2)
memory usage: 4.0+ MB
```

In [47]:
```
order_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99224 entries, 0 to 99223
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   review_id                99224 non-null  object
 1   order_id                 99224 non-null  object
 2   review_score             99224 non-null  int64 
 3   review_comment_title     11568 non-null  object
 4   review_comment_message   40977 non-null  object
 5   review_creation_date     99224 non-null  object
 6   review_answer_timestamp  99224 non-null  object
dtypes: int64(1), object(6)
memory usage: 5.3+ MB
```

In [48]:
```
product_category_name_translation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 71 entries, 0 to 70
Data columns (total 2 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   product_category_name          71 non-null     object
 1   product_category_name_english  71 non-null     object
dtypes: object(2)
memory usage: 1.2+ KB
```

In [49]:
```
geolocation.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 738332 entries, 0 to 1000161
Data columns (total 5 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   geolocation_zip_code_prefix  738332 non-null  int64  
 1   geolocation_lat              738332 non-null  float64
 2   geolocation_lng              738332 non-null  float64
 3   geolocation_city             738332 non-null  object 
 4   geolocation_state            738332 non-null  object 
dtypes: float64(2), int64(1), object(2)
memory usage: 33.8+ MB
```

There are several datetime data that are stored as objects and need to be converted to datetime

Datetime type

convert datetime data that is stored as objects to datetime

In [50]:
```
orders['order_purchase_timestamp'] = pd.to_datetime(orders['order_purchase_timestamp'])
orders['order_approved_at'] = pd.to_datetime(orders['order_approved_at'])
orders['order_delivered_carrier_date'] = pd.to_datetime(orders['order_delivered_carrier_
orders['order_delivered_customer_date'] = pd.to_datetime(orders['order_delivered_custome
orders['order_estimated_delivery_date'] = pd.to_datetime(orders['order_estimated_deliver
orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99441 entries, 0 to 99440
Data columns (total 8 columns):
 #   Column                         Non-Null Count  Dtype         
---  ------                         --------------  -----         
 0   order_id                       99441 non-null  object        
 1   customer_id                    99441 non-null  object        
 2   order_status                   99441 non-null  object        
 3   order_purchase_timestamp       99441 non-null  datetime64[ns]
 4   order_approved_at              99281 non-null  datetime64[ns]
 5   order_delivered_carrier_date   97658 non-null  datetime64[ns]
 6   order_delivered_customer_date  96476 non-null  datetime64[ns]
 7   order_estimated_delivery_date  99441 non-null  datetime64[ns]
dtypes: datetime64[ns](5), object(3)
memory usage: 6.1+ MB
```

In [51]:
```
order_reviews['review_creation_date'] = pd.to_datetime(order_reviews['review_creation_da
order_reviews['review_answer_timestamp'] = pd.to_datetime(order_reviews['review_answer_t
order_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99224 entries, 0 to 99223
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype         
---  ------                   --------------  -----         
 0   review_id                99224 non-null  object        
 1   order_id                 99224 non-null  object        
 2   review_score             99224 non-null  int64         
 3   review_comment_title     11568 non-null  object        
 4   review_comment_message   40977 non-null  object        
 5   review_creation_date     99224 non-null  datetime64[ns]
 6   review_answer_timestamp  99224 non-null  datetime64[ns]
dtypes: datetime64[ns](2), int64(1), object(4)
memory usage: 5.3+ MB
```

Create tables in the MSSQL database to store the datasets

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..customers') IS NOT NULL
DROP TABLE dbo.customers

CREATE TABLE customers
(
customer_id varchar(max),
customer_unique_id varchar(max),
customer_zip_code_prefix int,
customer_city varchar(max),
customer_state varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
customers.to_sql(name="customers", schema="dbo", con=engine, if_exists="replace", index=
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..products') IS NOT NULL
DROP TABLE dbo.products

CREATE TABLE products
(
product_id varchar(max),
product_category_name varchar(max),
product_name_lenght int,
product_description_lenght NUMERIC(2),
product_photos_qty NUMERIC(2),
product_weight_g NUMERIC(2),
product_length_cm NUMERIC(2),
product_height_cm NUMERIC(2),
product_width_cm NUMERIC(2)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
products.to_sql(name="products", schema="dbo", con=engine, if_exists="replace", index=Fa
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..orders') IS NOT NULL
DROP TABLE dbo.orders

CREATE TABLE orders
(
order_id varchar(max),
customer_id varchar(max),
order_status varchar(max),
order_purchase_timestamp varchar(max),
order_approved_at varchar(max),
order_delivered_carrier_date varchar(max),
order_delivered_customer_date varchar(max),
order_estimated_delivery_date varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
orders.to_sql(name="orders", schema="dbo", con=engine, if_exists="replace", index=False)
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..sellers') IS NOT NULL
DROP TABLE dbo.sellers

CREATE TABLE sellers
(
seller_id varchar(max),
seller_zip_code_prefix int,
seller_city varchar(max),
seller_state varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
sellers.to_sql(name="sellers", schema="dbo", con=engine, if_exists="replace", index=Fals
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..order_items') IS NOT NULL
DROP TABLE dbo.order_items

CREATE TABLE order_items
(
order_id varchar(max),
order_item_id int,
product_id varchar(max),
seller_id varchar(max),
shipping_limit_date varchar(max),
price NUMERIC(2),
freight_value NUMERIC(2)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
order_items.to_sql(name="order_items", schema="dbo", con=engine, if_exists="replace", in
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..order_payments') IS NOT NULL
DROP TABLE dbo.order_payments

CREATE TABLE order_payments
(
order_id varchar(max),
payment_sequential int,
payment_type varchar(max),
payment_installments int,
payment_value NUMERIC(2)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
order_payments.to_sql(name="order_payments", schema="dbo", con=engine, if_exists="replac
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..order_reviews') IS NOT NULL
DROP TABLE dbo.order_reviews

CREATE TABLE order_reviews
(
review_id varchar(max),
order_id varchar(max),
review_score int,
review_comment_title varchar(max),
review_comment_message varchar(max),
review_creation_date varchar(max),
review_answer_timestamp varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
order_reviews.to_sql(name="order_reviews", schema="dbo", con=engine, if_exists="replace"
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..product_category_name_translation') IS NOT NULL
DROP TABLE dbo.product_category_name_translation

CREATE TABLE product_category_name_translation
(
product_category_name varchar(max),
product_category_name_english varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
product_category_name_translation.to_sql(name="product_category_name_translation", schem
```

Create the table in the MSSQL database

In [ ]:
```
%%sql

IF OBJECT_ID('Olist_Store..geolocation') IS NOT NULL
DROP TABLE dbo.geolocation

CREATE TABLE geolocation
(
geolocation_zip_code_prefix int,
geolocation_lat NUMERIC(2),
geolocation_lng NUMERIC(2),
geolocation_city varchar(max),
geolocation_state varchar(max)
);
```

Append the data to the table in the MSSQL database

In [ ]:
```
geolocation.to_sql(name="geolocation", schema="dbo", con=engine, if_exists="replace", in
```

start the Exploratory data analysis

orders data

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the distribution of order status

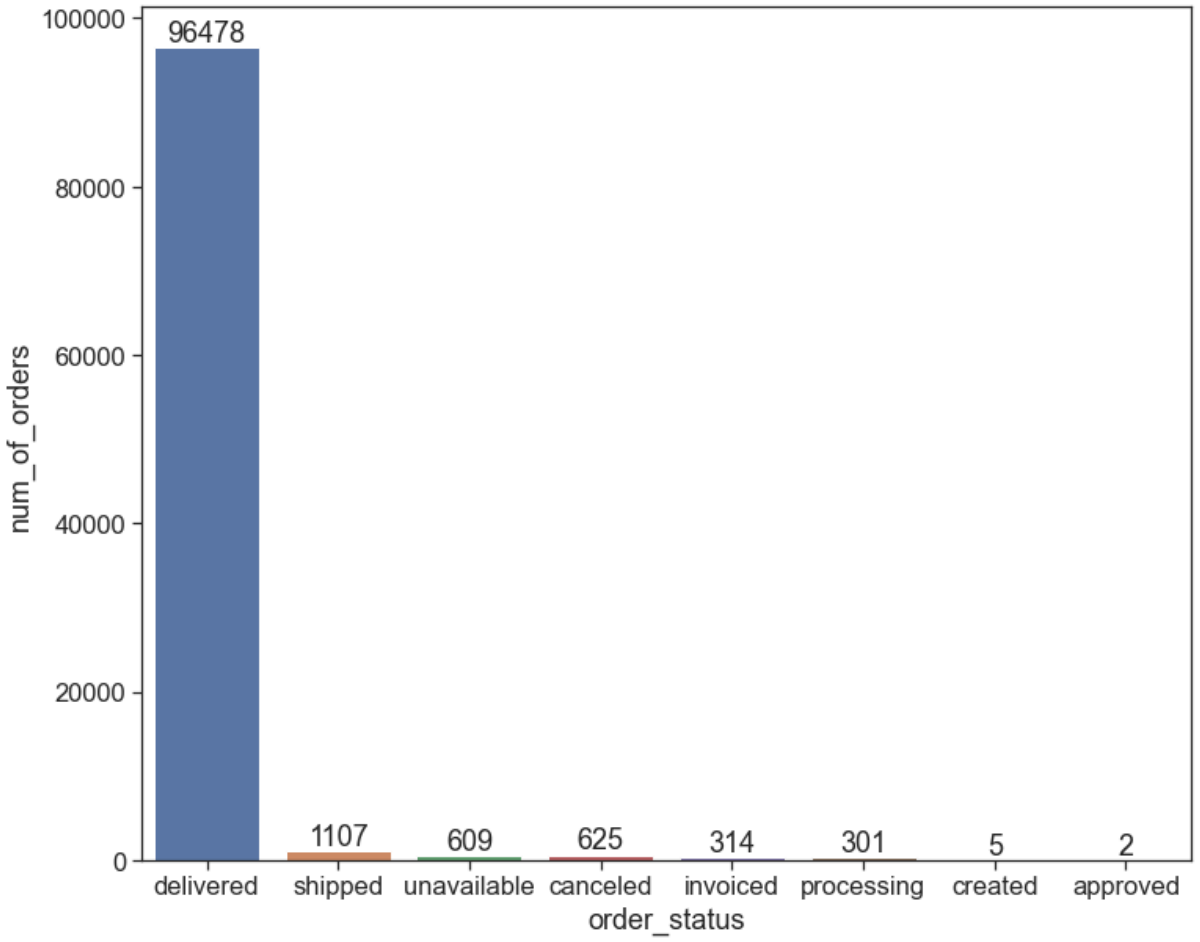
In [52]:
```
order_status_query = """

WITH CTE AS
(
SELECT DISTINCT order_status,
       COUNT(*) AS 'num_of_orders'
FROM orders
GROUP BY order_status
)
SELECT order_status, num_of_orders,
       ROUND(CAST(num_of_orders AS FLOAT) / SUM(num_of_orders) OVER (ORDER BY (SELECT NU
FROM CTE
ORDER BY 'percent' DESC

"""

order_status = pd.read_sql_query(order_status_query, connection)
order_status
```

Out[52]:

	order_status	num_of_orders	percent
0	delivered	96478	97.0
1	shipped	1107	1.1
2	unavailable	609	0.6
3	canceled	625	0.6
4	invoiced	314	0.3
5	processing	301	0.3
6	created	5	0.0
7	approved	2	0.0

draw a bar plot of the order status distribution

In [53]:
```
plt.rc('figure', figsize=(12, 10))
ax = sns.barplot(x='order_status', y='num_of_orders', data=order_status)
for container in ax.containers:
    ax.bar_label(container)
```



There are several order statues, and their logical order is:

created, approved, invoiced, processing, shipped, and delivered.

97% of the orders are delivered. Currently, 1.1% of orders are being shipped, and 0.3 are in processing. There are also, order status of canceled and unavailable. only 0.6% of the orders were canceled and only 0.6% of the orders could not be completed due to unavailable in the products.
Olist Store has 97% rate of delivering its orders.

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the num. of orders by month

In [54]:
```
orders_by_day_query = """

SELECT COUNT(*) AS num_of_orders,
       DATEADD(MONTH, DATEDIFF(MONTH, 0, order_purchase_timestamp), 0) AS 'date'
FROM orders
GROUP BY DATEADD(MONTH, DATEDIFF(MONTH, 0, order_purchase_timestamp), 0)
ORDER BY date

"""

orders_by_day = pd.read_sql_query(orders_by_day_query, connection)
orders_by_day
```

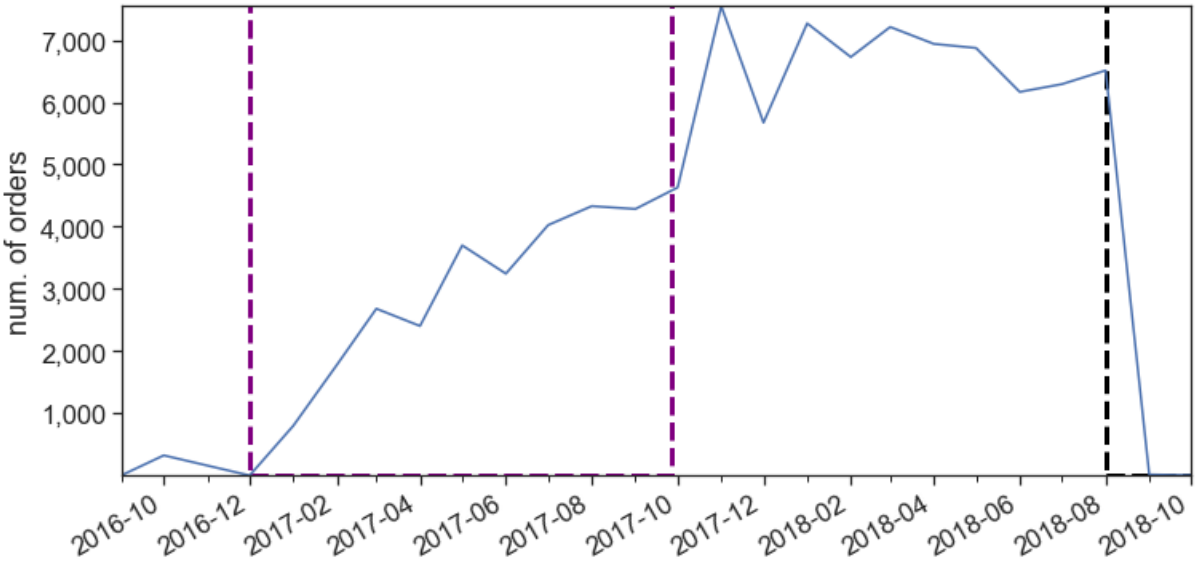
Out[54]:

	num_of_orders	date
0	4	2016-09-01
1	324	2016-10-01
2	1	2016-12-01
3	800	2017-01-01
4	1780	2017-02-01
5	2682	2017-03-01
6	2404	2017-04-01
7	3700	2017-05-01
8	3245	2017-06-01
9	4026	2017-07-01
10	4331	2017-08-01
11	4285	2017-09-01
12	4631	2017-10-01
13	7544	2017-11-01
14	5673	2017-12-01
15	7269	2018-01-01
16	6728	2018-02-01
17	7211	2018-03-01
18	6939	2018-04-01
19	6873	2018-05-01
20	6167	2018-06-01
21	6292	2018-07-01
22	6512	2018-08-01
23	16	2018-09-01
24	4	2018-10-01

draw a timelime (line chart) of the num of orders by month

In [55]:
```
fig, ax = plt.subplots(figsize=(12, 6))

monthly_locator = mdates.MonthLocator()
ax.xaxis.set_minor_locator(monthly_locator)

ax.plot(orders_by_day['date'], orders_by_day['num_of_orders'])

plt.ylabel('num. of orders')

ax.get_yaxis().set_major_formatter(
matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

# Ensure ticks fall once every other month (interval=2)
ax.xaxis.set_major_locator(mdates.MonthLocator(interval=2))

ax.margins(x=0, y=0)
fig.autofmt_xdate()

add_rectangle_datetime_x_axis(
    ax = ax,
    date_list =  orders_by_day['date'],
    date_start_index = 2,
    date_delta = 30*10,
    y_max_value = 8000,
    color = "purple"
    )

add_rectangle_datetime_x_axis(
    ax = ax,
    date_list =  orders_by_day['date'],
    date_start_index = 22,
    date_delta = 30*3,
    y_max_value = 8000,
    color = "black"
    )
```

Out[55]:
```
<matplotlib.patches.Rectangle at 0x1aa37d146a0>
```



Information from the num. of orders line chart

1. From October 2016 to December 2016 the store is probebly in the lunching phase, with less the 500 orders per month.
2. From December 2016 to October 2017 the store is perfoming very well with thoussands of orders per month reaching 4,000 orders per month.
3. November 2017 is the month with the highest number of orders (black firday?)

4. In 2018 the store peeks to over 7,000 orders per month.
5. the store (E-commerce) has a growing trend along the time. people buy more things online than before.

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the num. of orders by weekday

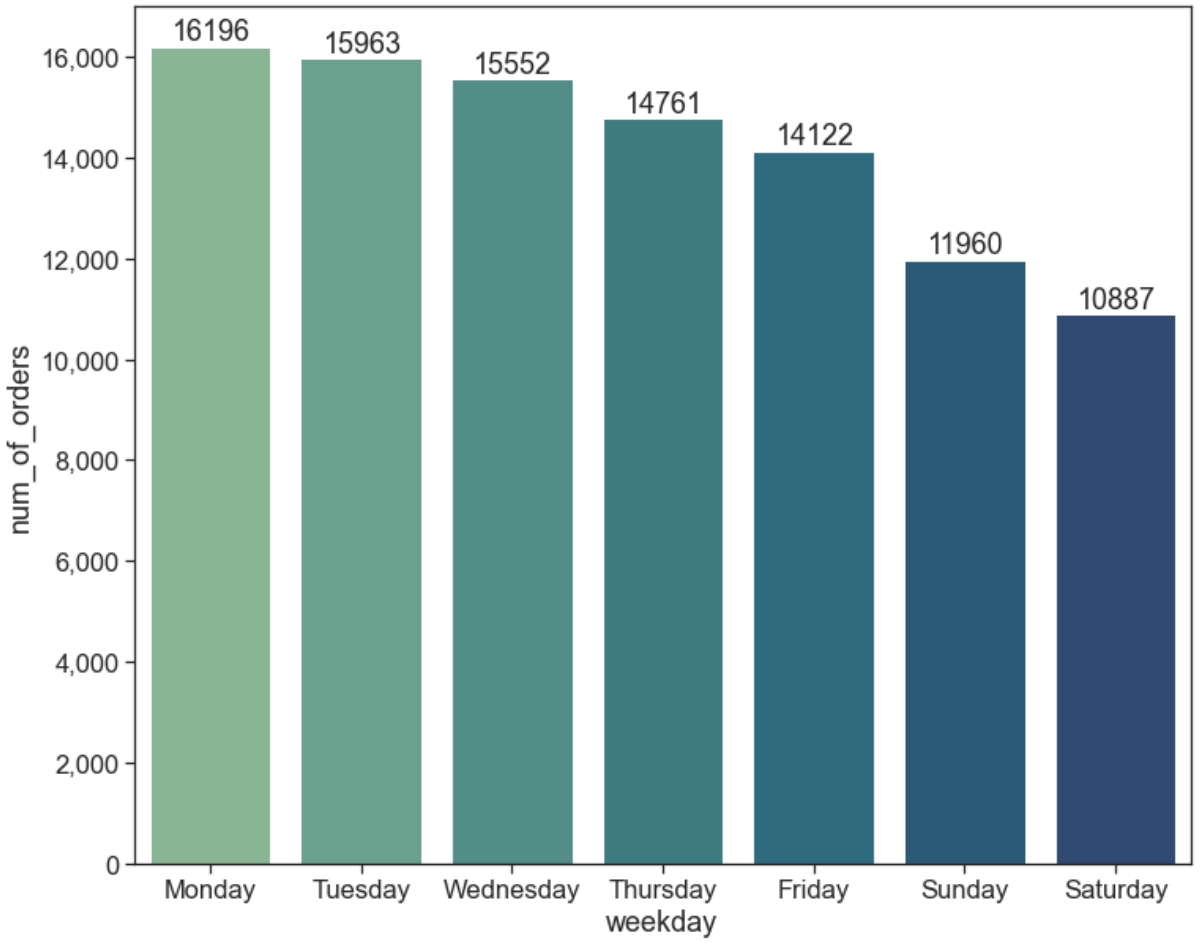
In [56]:
```
orders_by_weekday_query = """

SELECT COUNT(*) AS num_of_orders,
       DATENAME(weekday, order_purchase_timestamp) AS 'weekday'
FROM orders
GROUP BY DATENAME(weekday, order_purchase_timestamp)
ORDER BY num_of_orders DESC

"""

orders_by_weekday = pd.read_sql_query(orders_by_weekday_query, connection)
orders_by_weekday
```

Out[56]:

	num_of_orders	weekday
0	16196	Monday
1	15963	Tuesday
2	15552	Wednesday
3	14761	Thursday
4	14122	Friday
5	11960	Sunday
6	10887	Saturday

Draw a bar chart of the num of orders by weekday

In [57]:
```
plt.rc('figure', figsize=(12, 10))
ax = sns.barplot(x='weekday', y='num_of_orders', data=orders_by_weekday,  palette="crest

ax.get_yaxis().set_major_formatter(
matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

for container in ax.containers:
    ax.bar_label(container)
```



Information from the num. of orders by weekday bar chart

Monday, Tuesday,and Wednesday are the most prefered days for brazilian's customers

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the num. of orders by the part of the day (e.g. morning, evening, etc.)

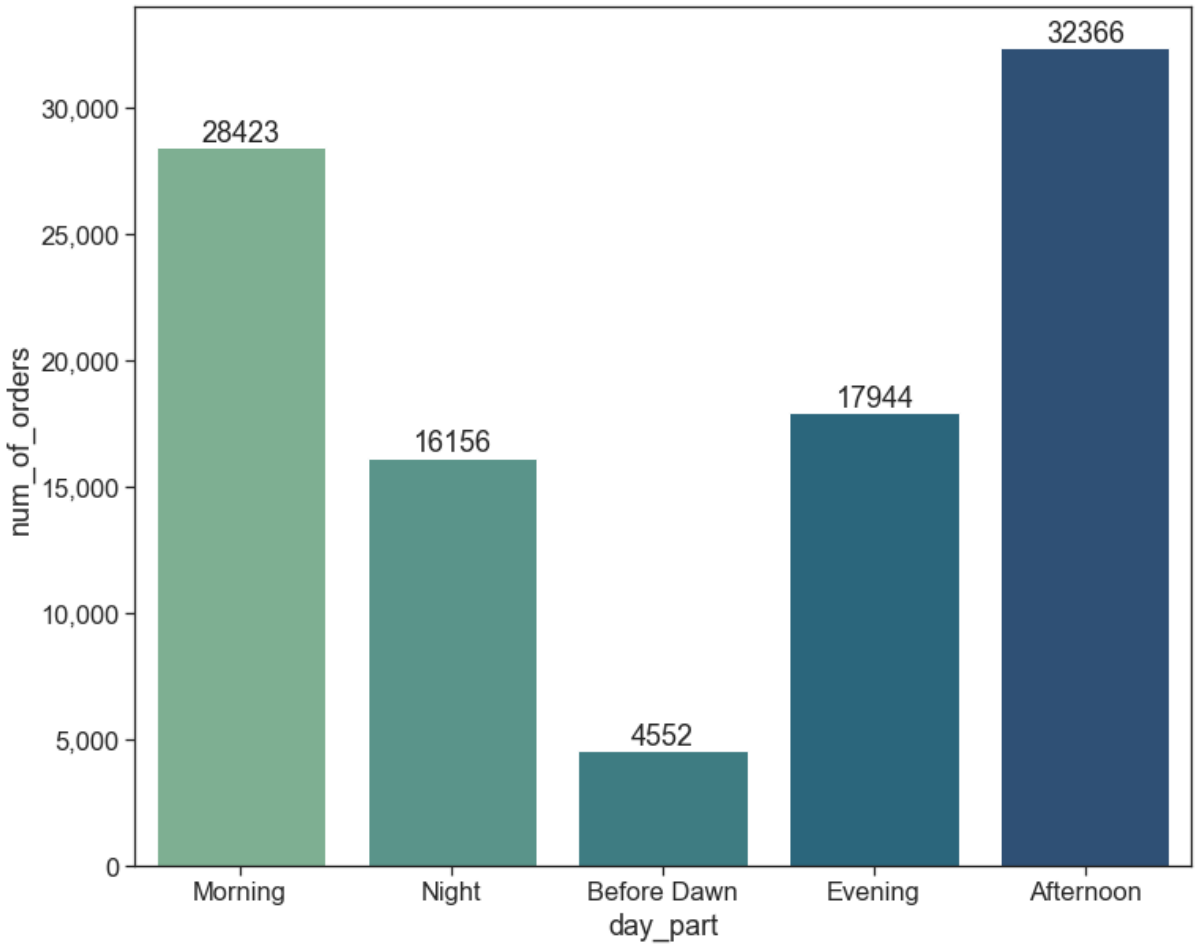
In [58]:
```
orders_by_day_part_query = """

WITH CTE AS
(
SELECT order_id,
       CASE WHEN DATEPART(HOUR, order_purchase_timestamp) BETWEEN 0 AND 4 THEN 'Before D
       WHEN DATEPART(HOUR, order_purchase_timestamp) BETWEEN 5 AND 12 THEN 'Morning'
       WHEN DATEPART(HOUR, order_purchase_timestamp) BETWEEN 13 AND 17 THEN 'Afternoon'
       WHEN DATEPART(HOUR, order_purchase_timestamp) BETWEEN 16 AND 20 THEN 'Evening'
       WHEN DATEPART(HOUR, order_purchase_timestamp) > 20 THEN 'Night'
END AS 'day_part'
FROM orders
)
SELECT COUNT(*) AS num_of_orders, day_part
FROM CTE
GROUP BY day_part

"""

orders_by_day_part = pd.read_sql_query(orders_by_day_part_query, connection)
orders_by_day_part
```

Out[58]:

	num_of_orders	day_part
0	28423	Morning
1	16156	Night
2	4552	Before Dawn
3	17944	Evening
4	32366	Afternoon

Draw a bar chart of the num. of orders distribution by the part of the day

In [59]:
```
plt.rc('figure', figsize=(12, 10))
ax = sns.barplot(x='day_part', y='num_of_orders', data=orders_by_day_part,  palette="cre

ax.get_yaxis().set_major_formatter(
matplotlib.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))

for container in ax.containers:
    ax.bar_label(container)
```



customers tend to buy more at afternoons and in the mornings

Create a dataframe in jupyter using an MSSQL query from the MSSQL database of the num. of orders by the states

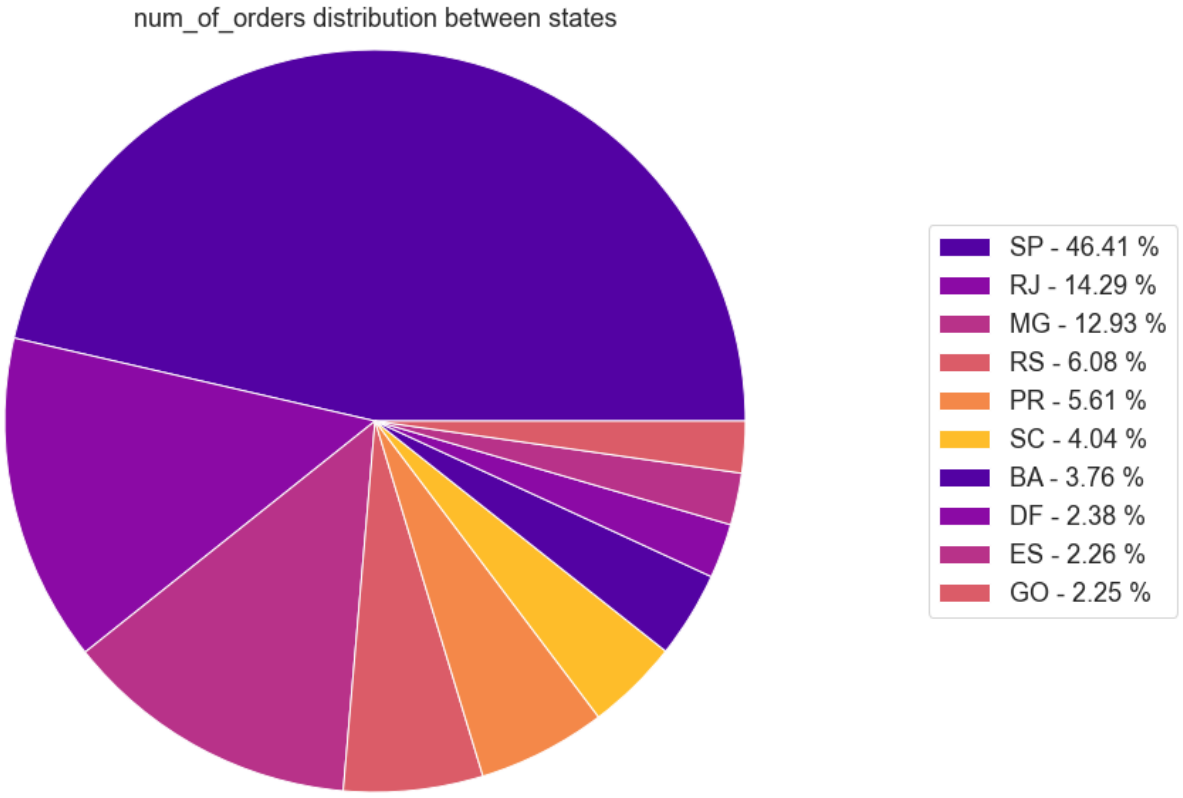
In [60]:
```
orders_state_query = """

SELECT TOP 10 COUNT(orders.order_id) AS 'num_of_orders',
       customers.customer_state
FROM orders JOIN customers
ON orders.customer_id = customers.customer_id
GROUP BY customer_state
ORDER BY num_of_orders DESC;

"""

orders_state = pd.read_sql_query(orders_state_query, connection)
orders_state
```

Out[60]:

	num_of_orders	customer_state
0	41746	SP
1	12852	RJ
2	11635	MG
3	5466	RS
4	5045	PR
5	3637	SC
6	3380	BA
7	2140	DF
8	2033	ES
9	2020	GO

draw a pie chart of the num. of orders distribution by states

In [61]:
```
pie_chart_function(df = orders_state,
                   categorial_column = 'customer_state',
                   values_column = 'num_of_orders',
                   title = 'num_of_orders distribution between states',
                   palette = 'plasma')
```



note:
A more in-depth analyses might divide the states into regions, to provide geographic information on the orders, but for this asignment, there is nit enough time, to understand all of the stats name

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the top 10 states (based on num. of orders) by month

In [62]:
```
top_ten_orders_state_query = """

SELECT COUNT(orders.order_id) AS 'num_of_orders',
       customers.customer_state,
       DATEADD(MONTH, DATEDIFF(MONTH, 0, order_purchase_timestamp), 0) AS 'date'
FROM orders JOIN customers
ON orders.customer_id = customers.customer_id
WHERE customers.customer_state IN
(
SELECT TOP 10 customers.customer_state
FROM orders JOIN customers
ON orders.customer_id = customers.customer_id
GROUP BY customer_state
ORDER BY COUNT(orders.order_id) DESC
)
GROUP BY customer_state, DATEADD(MONTH, DATEDIFF(MONTH, 0, order_purchase_timestamp), 0)
ORDER BY num_of_orders DESC;

"""

orders_state_date = pd.read_sql_query(top_ten_orders_state_query, connection)
orders_state_date
```

Out[62]:

	num_of_orders	customer_state	date
0	3253	SP	2018-08-01
1	3207	SP	2018-05-01
2	3059	SP	2018-04-01
3	3052	SP	2018-01-01
4	3037	SP	2018-03-01
...	...	...	...
214	2	SP	2018-10-01
215	1	PR	2016-12-01
216	1	RJ	2018-10-01
217	1	SC	2018-09-01
218	1	RS	2016-09-01

219 rows × 3 columns

pivot the data to a pivot table.
turn the date column to the index column - that is needed for the next visualization

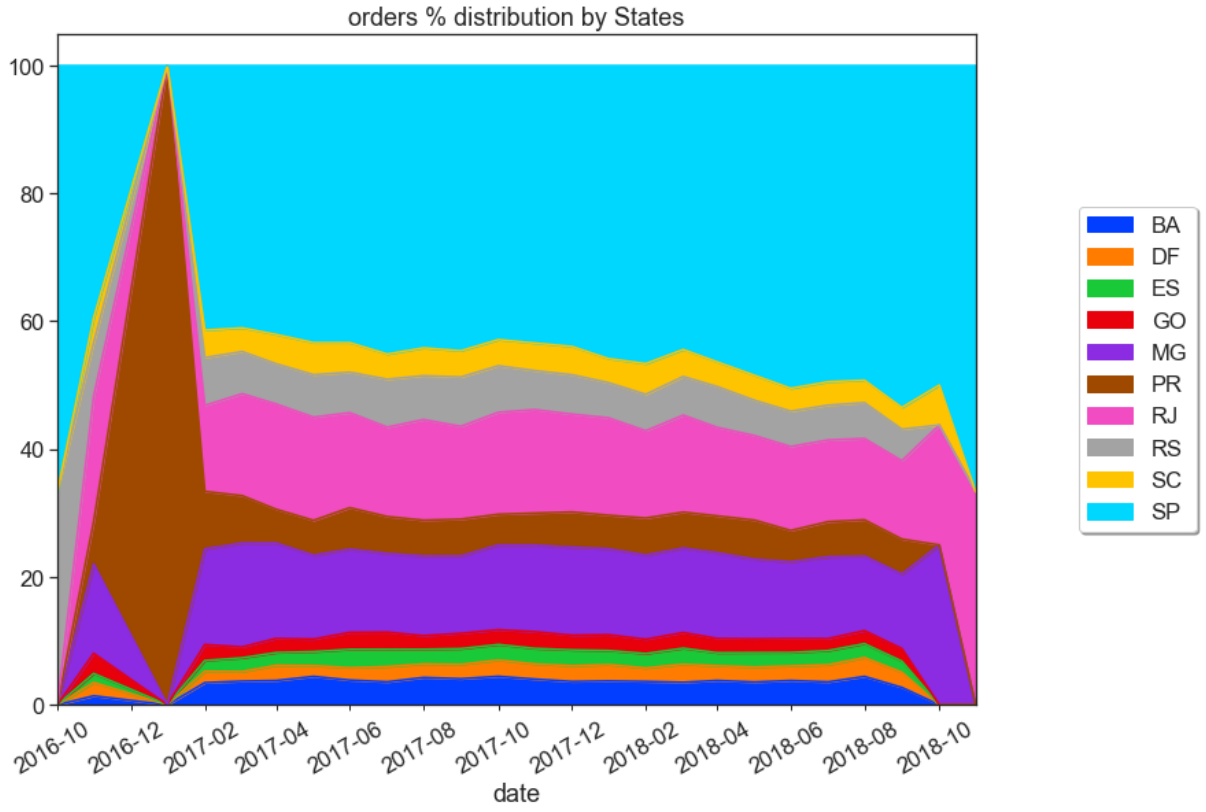
In [63]:
```
orders_state_date_pivot = orders_state_date.pivot_table(
    values = 'num_of_orders', index=['date'], columns = 'customer_state').reset_index()

orders_state_date_pivot.set_index(orders_state_date_pivot['date'], inplace=True)
orders_state_date_pivot.drop(['date'], axis = 1, inplace=True)

orders_state_date_pivot
```

Out[63]:

customer_state	BA	DF	ES	GO	MG	PR	RJ	RS	SC	SP
date										
2016-09-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	2.0
2016-10-01	4.0	6.0	4.0	9.0	40.0	19.0	56.0	24.0	11.0	113.0
2016-12-01	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	NaN	NaN
2017-01-01	25.0	13.0	12.0	18.0	108.0	65.0	97.0	54.0	31.0	299.0
2017-02-01	59.0	24.0	34.0	27.0	259.0	118.0	254.0	105.0	59.0	654.0
2017-03-01	91.0	57.0	48.0	53.0	358.0	127.0	395.0	151.0	110.0	1010.0
2017-04-01	93.0	35.0	46.0	41.0	275.0	114.0	338.0	139.0	105.0	908.0
2017-05-01	127.0	64.0	94.0	87.0	428.0	213.0	488.0	208.0	152.0	1425.0
2017-06-01	106.0	70.0	80.0	79.0	363.0	170.0	412.0	221.0	116.0	1331.0
2017-07-01	155.0	77.0	83.0	77.0	453.0	203.0	571.0	249.0	158.0	1604.0
2017-08-01	158.0	87.0	95.0	93.0	469.0	223.0	562.0	299.0	159.0	1729.0
2017-09-01	170.0	97.0	93.0	88.0	507.0	183.0	609.0	278.0	156.0	1638.0
2017-10-01	166.0	98.0	100.0	108.0	560.0	206.0	668.0	252.0	178.0	1793.0
2017-11-01	250.0	168.0	170.0	157.0	943.0	378.0	1048.0	422.0	303.0	3012.0
2017-12-01	192.0	131.0	113.0	127.0	691.0	270.0	783.0	283.0	193.0	2357.0
2018-01-01	239.0	138.0	147.0	146.0	863.0	378.0	893.0	373.0	314.0	3052.0
2018-02-01	214.0	172.0	152.0	149.0	804.0	342.0	922.0	368.0	257.0	2703.0
2018-03-01	249.0	150.0	134.0	146.0	879.0	377.0	907.0	418.0	252.0	3037.0
2018-04-01	225.0	148.0	142.0	136.0	786.0	386.0	834.0	349.0	246.0	3059.0
2018-05-01	241.0	144.0	134.0	139.0	762.0	311.0	833.0	351.0	227.0	3207.0
2018-06-01	201.0	150.0	124.0	105.0	717.0	308.0	716.0	305.0	205.0	2773.0
2018-07-01	250.0	166.0	123.0	115.0	658.0	320.0	717.0	316.0	198.0	2777.0
2018-08-01	165.0	145.0	105.0	120.0	708.0	333.0	745.0	300.0	206.0	3253.0
2018-09-01	NaN	NaN	NaN	NaN	4.0	NaN	3.0	NaN	1.0	8.0
2018-10-01	NaN	NaN	NaN	NaN	NaN	NaN	1.0	NaN	NaN	2.0

Draw an area chart of the orders distrbution by states (in percents)

In [64]:
```
pivot_dates = orders_state_date.pivot_table(
    values = 'num_of_orders', index=['date'], columns = 'customer_state').reset_index()[

ax = stacked_area_plot_percents_month(
    pivot_df = orders_state_date_pivot,
    title = 'orders % distribution by States')
```



Information from the area chart

Remebering that until 2017 there were less then 500 orders permonth, most of the orders on until the end of 2016 were from PR(Puerto Rico?). from 2017 until 2018 the orders distribution between states is stable, with SP (São Paulo?) as the state with the highest amount of buing customers.
SP (São Paulo?) num of orders also increases in time and its cut from the entire orders increases.

query the total num. of products and product categories

In [65]:
```
%%sql
SELECT COUNT(DISTINCT product_id) AS 'num_products',
       COUNT(DISTINCT product_category_name) AS 'num_product_categories'
FROM products;
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[65]:

num_products	num_product_categories
32951	73

there are 32,951 products and 73 product categories

query the top 10 largest product categories

In [66]:
```
%%sql
WITH CTE_1 AS
(
SELECT DISTINCT product_category_name,
       COUNT(DISTINCT product_id) AS 'num_products'
FROM products
GROUP BY product_category_name
),
CTE_2
AS (
SELECT product_trans.product_category_name_english, CTE_1.num_products,
       SUM(CTE_1.num_products) OVER (ORDER BY num_products DESC) AS 'cumm_sum',
       ROUND( CAST(CTE_1.num_products AS FLOAT) / SUM(CTE_1.num_products) OVER () * 100
FROM CTE_1 JOIN product_category_name_translation product_trans
ON  CTE_1.product_category_name = product_trans.product_category_name
)
SELECT TOP 10 *
FROM CTE_2;
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[66]:

product_category_name_english	num_products	cumm_sum	percent
bed_bath_table	3029	3029	9.37
sports_leisure	2867	5896	8.87
furniture_decor	2657	8553	8.22
health_beauty	2444	10997	7.56
housewares	2335	13332	7.22
auto	1900	15232	5.88
computers_accessories	1639	16871	5.07
toys	1411	18282	4.36
watches_gifts	1329	19611	4.11
telephony	1134	20745	3.51

1. Each product is assigned to only 1 category.
2. The 10 largest product categories are:
    A. bed_bath_table
    B. sports_leisure
    C. furniture_decor
    D. health_beauty
    E. housewares
    F. auto
    G. computers_accessories
    H. toys
    I. watches_gifts
    J. telephony

query the products that are not in any category

In [67]:
```
%%sql
SELECT product_category_name,
       COUNT(*) AS 'num_products'
FROM products
WHERE product_category_name IS NULL
GROUP BY product_category_name;
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[67]:

product_category_name	num_products
None	610

There are 610 products that are not in any category

create a MSSQL view to simplify a query
first, delete the view if it already exists

In [68]:
```
%%sql
IF EXISTS (SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'top_ten_categorie
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[68]: []

create a MSSQL view to simplify a query

In [69]:
```
%%sql
CREATE VIEW top_ten_categories AS
SELECT TOP 10 product_trans.product_category_name_english
FROM orders JOIN order_items
ON orders.order_id = order_items.order_id JOIN products
ON order_items.product_id = products.product_id JOIN product_category_name_translation p
ON product_trans.product_category_name = products.product_category_name
GROUP BY product_trans.product_category_name_english
ORDER BY SUM(order_items.order_item_id) DESC;
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[69]: []

Create a dataframe in jupyter using an MSSQL query from the MSSQL database
of the top 10 product categories (based on the num. of products ordered) by month
using the MSSQL view that was created before

In [70]:
```
top_ten_product_categories_query = """

SELECT product_trans.product_category_name_english,
       DATEADD(MONTH, DATEDIFF(MONTH, 0, order_purchase_timestamp), 0) AS 'date',
       SUM(order_items.order_item_id) AS 'quantity'
FROM orders JOIN order_items
ON orders.order_id = order_items.order_id JOIN products
ON order_items.product_id = products.product_id JOIN product_category_name_translation p
ON product_trans.product_category_name = products.product_category_name
WHERE product_trans.product_category_name_english IN (SELECT * FROM top_ten_categories)
GROUP BY product_trans.product_category_name_english, DATEADD(MONTH, DATEDIFF(MONTH, 0,

"""

top_ten_product_categories = pd.read_sql_query(top_ten_product_categories_query, connect
top_ten_product_categories
```

Out[70]:

	product_category_name_english	date	quantity
0	telephony	2018-04-01	395
1	sports_leisure	2018-04-01	693
2	health_beauty	2018-05-01	888
3	health_beauty	2017-04-01	202
4	housewares	2017-03-01	249
...	...	...	...
208	bed_bath_table	2017-09-01	639
209	furniture_decor	2017-11-01	1280
210	watches_gifts	2018-08-01	447
211	auto	2017-03-01	101
212	garden_tools	2018-06-01	257

213 rows × 3 columns

pivot the data to a pivot table.
turn the date column to the index column - that is needed for the next visualization

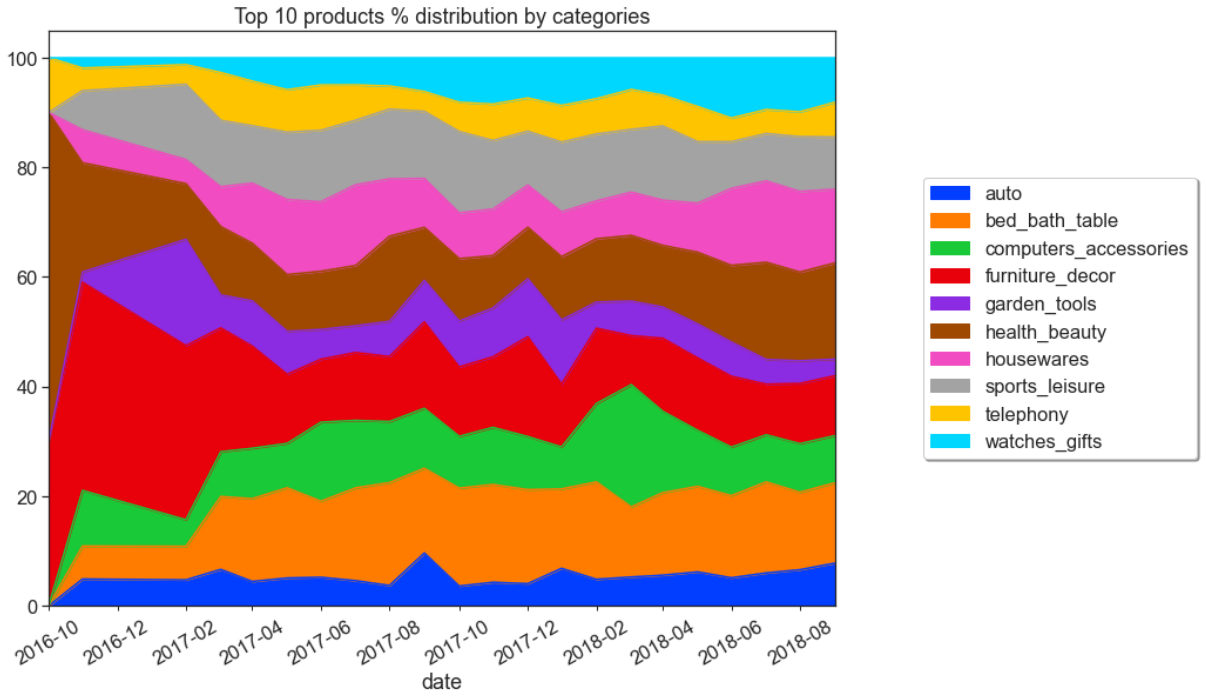
In [71]:
```
top_ten_product_categories_pivot = top_ten_product_categories.pivot_table(
    values = 'quantity', index=['date'], columns = 'product_category_name_english').rese

top_ten_product_categories_pivot.set_index(top_ten_product_categories_pivot['date'], inp
top_ten_product_categories_pivot.drop(['date'], axis = 1, inplace=True)

top_ten_product_categories_pivot
```

Out[71]:

product_category_name_english date	auto	bed_bath_table	computers_accessories	furniture_decor	garden_tools
2016-09-01	NaN	NaN	NaN	3.0	NaN
2016-10-01	13.0	16.0	27.0	101.0	5.0
2017-01-01	41.0	53.0	42.0	275.0	168.0
2017-02-01	96.0	192.0	118.0	326.0	87.0
2017-03-01	101.0	343.0	209.0	423.0	188.0
2017-04-01	99.0	321.0	158.0	246.0	153.0
2017-05-01	156.0	416.0	432.0	344.0	163.0
2017-06-01	121.0	447.0	325.0	327.0	129.0
2017-07-01	134.0	676.0	402.0	427.0	232.0
2017-08-01	376.0	603.0	427.0	615.0	296.0
2017-09-01	129.0	639.0	338.0	453.0	300.0
2017-10-01	182.0	756.0	442.0	545.0	378.0
2017-11-01	284.0	1206.0	680.0	1280.0	743.0
2017-12-01	307.0	650.0	344.0	515.0	526.0
2018-01-01	299.0	1097.0	884.0	848.0	295.0
2018-02-01	335.0	814.0	1422.0	569.0	402.0
2018-03-01	367.0	989.0	976.0	874.0	375.0
2018-04-01	382.0	963.0	632.0	815.0	383.0
2018-05-01	325.0	952.0	563.0	821.0	398.0
2018-06-01	342.0	946.0	490.0	526.0	257.0
2018-07-01	364.0	781.0	490.0	607.0	229.0
2018-08-01	428.0	805.0	473.0	600.0	167.0

Draw an area chart of the top 10 products distrbution by categories (in percents)

In [72]:
```
pivot_dates = top_ten_product_categories.pivot_table(
    values = 'quantity', index=['date'], columns = 'product_category_name_english').rese

ax = stacked_area_plot_percents_month(
    pivot_df = top_ten_product_categories_pivot,
    title = 'Top 10 products % distribution by categories')
```



Among the top 10 products categories:

not-regarding 2016-2017, and regarding 2017-2018

1. Increase of orders over time:
    A. watches_gifts
    B. health_beauty
    C. housewares
2. Decrease of orders over time:
    A. telephony
    B. sports_leisure
    C. garden_tools
3. no-change of orders over time:
    A. furniture_decor
    B. bed_bath_table
    C. auto

during March 2018 there was a peak of orders of computers_accessories, that has decreased sshortly after.

9 out 10 of the most sold product categories are also the largest categories. the 1 out of ten is toys, that is not in the top 10 seling product categors, but is in the top 7 largest categories.

query the total number of unique customers and returning customers

In [73]:
```
%%sql
SELECT COUNT(DISTINCT customer_unique_id) AS 'num_unique_customers',
       COUNT(DISTINCT customer_id) AS 'num_customers',
       COUNT(DISTINCT customer_id) - COUNT(DISTINCT customer_unique_id) AS 'num_returnin
       ROUND(
       CAST( COUNT(DISTINCT customer_id) - COUNT(DISTINCT customer_unique_id) AS FLOAT)
       / COUNT(DISTINCT customer_unique_id) * 100
       ,1) AS 'percent_returning_customers'
FROM customers;
```

```
 * mssql+pyodbc://nir:***@mssql
Done.
```

Out[73]:

num_unique_customers	num_customers	num_returning_customers	percent_returning_customers
96096	99441	3345	3.5

There are 96,096 unique customers. among them, 3,345 are returning customers, 3.5%

In [74]:
```
connection.close()
```