nb_ecommerce

October 15, 2021

0.0.1 Imports

```
[ ]: import math
     import datetime

     import matplotlib.pyplot as plt
     import pandas as pd
     import numpy as np
```

0.0.2 Constants

```
[ ]: PATH_FILE_ORDER = './file/olist_orders_dataset.csv'
     PATH_FILE_ORDER_ITEM = './file/olist_order_items_dataset.csv'
     PATH_FILE_ORDER_REVIEW = './file/olist_order_reviews_dataset.csv'
     PATH_FILE_ORDER_PAYMENT = './file/olist_order_payments_dataset.csv'

     PATH_FILE_PROD = './file/olist_products_dataset.csv'
     PATH_FILE_SELLER = './file/olist_sellers_dataset.csv'
     PATH_FILE_CUSTOMER = './file/olist_customers_dataset.csv'
     PATH_FILE_GEOLOCATION = './file/olist_geolocation_dataset.csv'
     PATH_FILE_PROD_CATEGORY_TRANSLATE = './file/product_category_name_translation.
      ↪csv'
```

1 Reviews

1.0.1 Constants

```
[ ]: # Original columns
     COL_REV_ID = 'review_id'
     COL_REV_MSG = 'review_comment_message'
     COL_REV_ORDER = 'order_id'
     COL_REV_SCORE = 'review_score'
     COL_REV_TITLE = 'review_comment_title'
     COL_REV_ANSWER = 'review_answer_timestamp'
     COL_REV_CREATION = 'review_creation_date'

     # Custom Columns
```

```
COL_REV_CUS_MSG_LEN = 'Message Length'
```

1.0.2 Build Dataframe

```
[ ]: # Import file
reviewsDF = pd.read_csv(PATH_FILE_ORDER_REVIEW)
reviewsDFClean = reviewsDF[reviewsDF[COL_REV_MSG].notnull()]

# Handle NaN values
reviewsDF.loc[reviewsDF[COL_REV_MSG].isnull(), COL_REV_MSG] = ''
reviewsDF.loc[reviewsDF[COL_REV_TITLE].isnull(), COL_REV_TITLE] = ''

reviewsDFClean.loc[reviewsDFClean[COL_REV_TITLE].isnull(), COL_REV_TITLE] = ''

# Compute review lengths
reviewsDF[COL_REV_CUS_MSG_LEN] = reviewsDF[COL_REV_MSG].apply(lambda msg:␣
 ↪len(msg))
reviewsDFClean[COL_REV_CUS_MSG_LEN] = reviewsDFClean[COL_REV_MSG].apply(lambda␣
 ↪msg: len(msg))

# Format
sort_review = [COL_REV_SCORE, COL_REV_CUS_MSG_LEN, COL_REV_CREATION,␣
 ↪COL_REV_MSG, COL_REV_TITLE]
# columns_review = [COL_REV_SCORE, COL_REV_CUS_MSG_LEN, COL_REV_MSG,␣
 ↪COL_REV_TITLE]
reviewsDF = reviewsDF.sort_values(by=sort_review, ascending=False)
reviewsDFClean = reviewsDFClean.sort_values(by=sort_review, ascending=False)
```

```
/home/hjcosta/.local/lib/python3.8/site-packages/pandas/core/indexing.py:1817:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value, pi)
/tmp/ipykernel_22252/527500927.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reviewsDFClean[COL_REV_CUS_MSG_LEN] = reviewsDFClean[COL_REV_MSG].apply(lambda
msg: len(msg))
```

1.0.3 Statistics

[ ]:
```
reviewsCount = reviewsDF.shape[0]
noMsgReviewsCount = reviewsDF.loc[reviewsDF[COL_REV_CUS_MSG_LEN] == 0].shape[0]
noMsgReviewsRatio = round(noMsgReviewsCount / reviewsCount * 100, 2)

print(f'Reviews: {reviewsCount}')
print(f'Empty message reviews: {noMsgReviewsCount} / {reviewsCount}␣
↪({noMsgReviewsRatio}%)')

scoreValues = sorted(reviewsDF[COL_REV_SCORE].unique())
for score in scoreValues:

    scoredDF = reviewsDF.loc[reviewsDF[COL_REV_SCORE] == score]

    scoredCount = scoredDF.shape[0]
    scoredRatio = round(scoredCount / reviewsCount * 100, 2)
    noMsgScoredCount = scoredDF.loc[reviewsDF[COL_REV_CUS_MSG_LEN] == 0].
↪shape[0]
    noMsgScoredRation = round(noMsgScoredCount / scoredCount * 100, 2)

    print(f'{score} Score: {scoredCount} / {reviewsCount} ({scoredRatio}%)')
    print(f'\tNo message: {noMsgScoredCount} / {scoredCount}␣
↪({noMsgScoredRation}%)')
```

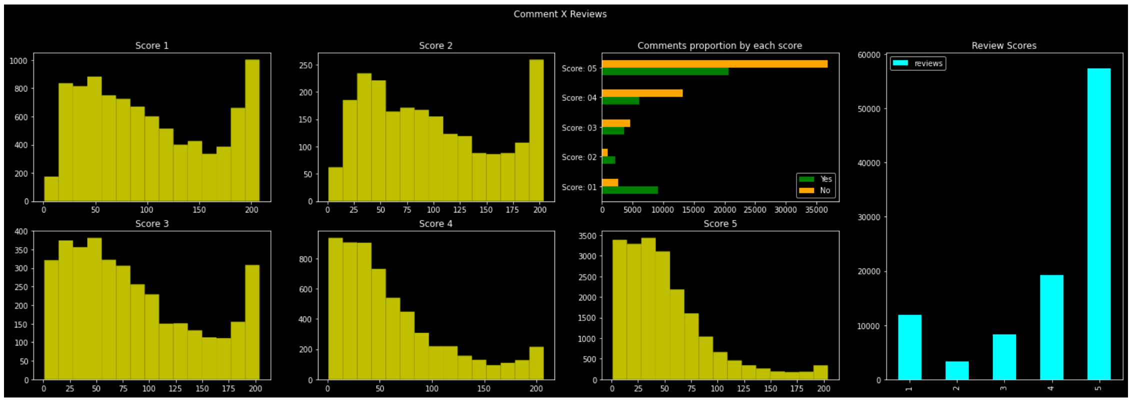
```
Reviews: 100000
Empty message reviews: 58247 / 100000 (58.25%)
1 Score: 11858 / 100000 (11.86%)
        No message: 2679 / 11858 (22.59%)
2 Score: 3235 / 100000 (3.23%)
        No message: 1006 / 3235 (31.1%)
3 Score: 8287 / 100000 (8.29%)
        No message: 4622 / 8287 (55.77%)
4 Score: 19200 / 100000 (19.2%)
        No message: 13166 / 19200 (68.57%)
5 Score: 57420 / 100000 (57.42%)
        No message: 36774 / 57420 (64.04%)
```

1.0.4 Plots

[ ]:
```
# Build grid
figure = plt.figure(figsize=(26, 8))

a11 = plt.subplot2grid((2, 4), (0, 0), fig=figure)
a12 = plt.subplot2grid((2, 4), (0, 1), fig=figure)
a13 = plt.subplot2grid((2, 4), (0, 2), fig=figure)

a14 = plt.subplot2grid((2, 4), (0, 3), fig=figure, rowspan=2)
```

```
a21 = plt.subplot2grid((2, 4), (1, 0), fig=figure)
a22 = plt.subplot2grid((2, 4), (1, 1), fig=figure)
a23 = plt.subplot2grid((2, 4), (1, 2), fig=figure)

# Bars graph: Total of review per each score.
bars = pd.DataFrame({
        'reviews': [
            reviewsDF[reviewsDF[COL_REV_SCORE] == 1].shape[0],
            reviewsDF[reviewsDF[COL_REV_SCORE] == 2].shape[0],
            reviewsDF[reviewsDF[COL_REV_SCORE] == 3].shape[0],
            reviewsDF[reviewsDF[COL_REV_SCORE] == 4].shape[0],
            reviewsDF[reviewsDF[COL_REV_SCORE] == 5].shape[0],
        ]
    },
    index=[1, 2, 3, 4, 5],
)

bars.plot.bar(ax=a14, title='Review Scores', color='cyan')

# Bars graph: Show proportion of reviews with or without comments per each␣
 ↪score level.
yes = []
no = []
scores = []

for i in range(1, 6):
    _yes = reviewsDF[(reviewsDF[COL_REV_SCORE] == i) &␣
 ↪(reviewsDF[COL_REV_CUS_MSG_LEN] > 0)].shape[0]
    _no = reviewsDF[(reviewsDF[COL_REV_SCORE] == i) &␣
 ↪(reviewsDF[COL_REV_CUS_MSG_LEN] == 0)].shape[0]
    total = _yes + _no

    scores.append('Score: 0' + str(i))
    yes.append(_yes)
    no.append(_no)

barh = pd.DataFrame({ 'Yes': yes, 'No': no }, index=scores)
barh.plot.barh(
    ax=a13,
    title='Comments proportion by each score',
    color={ 'Yes': 'green', 'No': 'orange', 'AVG': 'c'},
)

# Histograms: Show length of commentaries per each review score level
figPositionMap = {
    1: a11, 2: a12,
    3: a21, 4: a22, 5: a23,
}

for i in range(1, 6):
    a = figPositionMap.get(i)
    a.hist(reviewsDFClean[reviewsDFClean[COL_REV_SCORE] ==␣
↪i][COL_REV_CUS_MSG_LEN].values, bins=15, facecolor='y', snap=False)
    a.set_title('Score ' + str(i))

figure.suptitle('Comment X Reviews')
figure.show()
```

```
/tmp/ipykernel_22252/2605248469.py:62: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  figure.show()
```



2 Orders

2.0.1 Constants

```
[ ]: # Original columns: Orders
COL_ORD_ID = 'order_id'
COL_ORD_STATUS = 'order_status'
COL_ORD_CUSTOMER = 'customer_id'

COL_ORD_DATE_BUY = 'order_purchase_timestamp'
COL_ORD_DATE_APPROV = 'order_approved_at'
COL_ORD_DATE_CARRIER = 'order_delivered_carrier_date'
COL_ORD_DATE_DELIVER = 'order_delivered_customer_date'
COL_ORD_DATE_DELIVER_EST = 'order_estimated_delivery_date'

# Original columns: Order Items
```

```
COL_ORD_ITEM_ID = 'order_item_id'
COL_ORD_ITEM_PROD = 'product_id'
COL_ORD_ITEM_PRICE = 'price'
COL_ORD_ITEM_ORDER = 'order_id'
COL_ORD_ITEM_SELLER = 'seller_id'
COL_ORD_ITEM_FREIGHT = 'freight_value'
COL_ORD_ITEM_DATE_SHIP_LIMIT = 'shipping_limit_date'

# Custom columns
COL_ORD_CUS_TIME_DELAY = 'Delivery Delay'
COL_ORD_CUS_TIME_DELIVER = 'Time to Deliver'
COL_ORD_CUS_PRICE = 'Order Price'
COL_ORD_CUS_FREIGHT = 'Order Freight'

# Status
STATUS_ORD_APPROV = 'approved'
STATUS_ORD_CANCEL = 'canceled'
STATUS_ORD_CREATED = 'created'
STATUS_ORD_DELIVERED = 'delivered'
STATUS_ORD_INVOICED = 'invoiced'
STATUS_ORD_PROCESSING = 'processing'
STATUS_ORD_SHIPPED = 'shipped'
STATUS_ORD_UNAVAILABLE = 'unavailable'

# Rename
sortOrder = [COL_ORD_STATUS, COL_ORD_CUS_TIME_DELAY, COL_ORD_CUS_TIME_DELIVER,␣
↪COL_ORD_CUS_FREIGHT, COL_ORD_CUS_PRICE]

columnsOrder = [
    COL_ORD_STATUS,
    COL_ORD_DATE_BUY, COL_ORD_DATE_DELIVER_EST, COL_ORD_DATE_DELIVER,
    COL_ORD_CUS_TIME_DELIVER, COL_ORD_CUS_TIME_DELAY,
    COL_ORD_CUS_PRICE, COL_ORD_CUS_FREIGHT
]
```

2.0.2 Utils

[ ]:
```
def getDaysInterval(mainDate: datetime.date, relativeDate: datetime.date =␣
↪None, isRelativeToToday = True):
    '''
        TODO: 2021-10-14 - ADD Description
    '''

    upperDate = datetime.datetime.today() if isRelativeToToday else relativeDate
    return math.floor((upperDate - mainDate) / np.timedelta64(1,'D'))
```

```python
def getDaysDelay(maxDate: datetime.date, deliveryDate: np.datetime64 = None):
    '''
        TODO: 2021-10-14 - ADD Description
    '''

    upperDate = deliveryDate or datetime.datetime.today()
    isDelayed = upperDate > maxDate
    return getDaysInterval(deliveryDate, upperDate, False) if isDelayed else 0

def setDeliveryDays(df: pd.DataFrame) -> pd.Series:
    '''
        TODO: 2021-10-14 - ADD Description
    '''

    return np.vectorize(getDaysInterval)(
        df[COL_ORD_DATE_DELIVER_EST],
        df[COL_ORD_DATE_DELIVER],
        df[COL_ORD_STATUS] != STATUS_ORD_DELIVERED,
    )


def setDelayDays(df: pd.DataFrame) -> pd.Series:
    '''
        TODO: 2021-10-14 - ADD Description
    '''

    return np.vectorize(getDaysDelay)(
        df[COL_ORD_DATE_DELIVER_EST],
        df[COL_ORD_DATE_DELIVER],
    )

def getOrderPrice(source: pd.Series, orderID: str) -> float:
    '''
        TODO: 2021-10-15 - ADD Description
    '''

    try:
        return source.loc[orderID]
    except KeyError:
        return 0
```

2.0.3 Build Dataframe

```
[ ]: # Import files
dateColumns = [COL_ORD_DATE_BUY, COL_ORD_DATE_DELIVER, COL_ORD_DATE_DELIVER_EST]

ordersDF = pd.read_csv(PATH_FILE_ORDER, parse_dates=dateColumns)
orderItemDF = pd.read_csv(PATH_FILE_ORDER_ITEM)

# Clean data: Step 01 (remove inconsistent & unavailable data)
ordersDF = ordersDF[
  (ordersDF[COL_ORD_STATUS] != STATUS_ORD_UNAVAILABLE)
  & ~((ordersDF[COL_ORD_STATUS] == STATUS_ORD_DELIVERED) &␣
 ↪(ordersDF[COL_ORD_DATE_DELIVER].isnull())) # Avoid failure on time interval␣
 ↪calculations
]

# Add calculated interval fields
ordersDF[COL_ORD_CUS_TIME_DELAY] = setDelayDays(ordersDF)
ordersDF[COL_ORD_CUS_TIME_DELIVER] = setDeliveryDays(ordersDF)

# Clean data: Step 02 (keep only orders unfinished or finished with consistent␣
 ↪data)
endStatuses = [STATUS_ORD_CANCEL, STATUS_ORD_DELIVERED, STATUS_ORD_UNAVAILABLE]
midStatuses = [STATUS_ORD_CREATED, STATUS_ORD_APPROV, STATUS_ORD_INVOICED,␣
 ↪STATUS_ORD_PROCESSING, STATUS_ORD_SHIPPED]

ordersDF = ordersDF[
    (ordersDF[COL_ORD_CUS_TIME_DELAY] >= 0) |␣
 ↪(ordersDF[COL_ORD_CUS_TIME_DELIVER] >= 0)
    | ~ordersDF[COL_ORD_STATUS].isin(endStatuses)
]

# Add order price & freight price
# productMeanPrice = orderItemDF[ [COL_ORD_ITEM_PROD, COL_ORD_ITEM_PRICE] ].
 ↪groupby(COL_ORD_ITEM_PROD).mean().sort_values(by=COL_ORD_ITEM_PROD)
orderFreight = orderItemDF[ [COL_ORD_ITEM_ORDER, COL_ORD_ITEM_FREIGHT] ].
 ↪groupby(COL_ORD_ITEM_ORDER).sum().sort_values(by=COL_ORD_ITEM_ORDER)
orderPrice = orderItemDF[ [COL_ORD_ITEM_ORDER, COL_ORD_ITEM_PRICE] ].
 ↪groupby(COL_ORD_ITEM_ORDER).sum().sort_values(by=COL_ORD_ITEM_ORDER)

ordersDF[COL_ORD_CUS_FREIGHT] = ordersDF[COL_ORD_ID].apply(lambda id:␣
 ↪getOrderPrice(orderFreight[COL_ORD_ITEM_FREIGHT], id))
ordersDF[COL_ORD_CUS_PRICE] = ordersDF[COL_ORD_ID].apply(lambda id:␣
 ↪getOrderPrice(orderPrice[COL_ORD_ITEM_PRICE], id))

ordersDF = ordersDF.sort_values(by=sortOrder, ascending=True,␣
 ↪na_position='first')
```

```
ordersDF
```

```
[ ]:                              order_id                       customer_id  \
     88457  132f1e724165a07f6362532bfb97486e  b2191912d8ad6eac2e4dc3b6e1459515
     44897  a2e4c44360b4a57bdff22f3a4630c173  8886130db0ea6e9e70ba0b03d7c0d286
     68373  b059ee4de278302d550a3035c4cdb740  856336203359aa6a61bf3826f7d84c49
     60938  10a045cdf6a5650c21e9cfeb60384c16  a4b417188addbc05b26b72d5e44837a1
     37003  869997fbe01f39d184956b5c6bccfdbe  55c9dad94ec1a2ba57998bdb376c230a
     …                                   …                                 …
     47113  cda873529ca7ab71f677d5ec11a40304  76c74aaff2f3f7355f46d9818ad092b8
     19523  063b573b88fc80e516aba87df524f809  285195a5b585842e25bd1ef9015a8413
     34523  45973912e490866800c0aea8f63099c8  912f108a7026f25f99240a5c4c60e2c3
     22948  3f913d30288c117e41ffe5cc74743dc9  7aae6b74d7e0a2a11051bf8a16e16021
     4541   2e7a8482f6fb09756ca50c10d7bfc047  08c5351a6aca1c1589a38f244edeee9d

           order_status order_purchase_timestamp    order_approved_at  \
     88457     approved      2017-04-25 01:25:34  2017-04-30 20:32:41
     44897     approved      2017-02-06 20:18:17  2017-02-06 20:30:19
     68373     canceled      2018-10-16 20:16:02                  NaN
     60938     canceled      2018-10-17 17:30:18                  NaN
     37003     canceled      2018-09-26 08:40:15                  NaN
     …              …                        …                    …
     47113      shipped      2016-10-05 16:57:30  2016-10-06 15:52:49
     19523      shipped      2016-10-07 19:17:00  2016-10-07 19:29:20
     34523      shipped      2016-10-07 22:45:28  2016-10-07 22:58:37
     22948      shipped      2016-10-05 14:36:55  2016-10-06 15:53:06
     4541       shipped      2016-09-04 21:15:19  2016-10-07 13:18:03

           order_delivered_carrier_date order_delivered_customer_date  \
     88457                          NaN                           NaT
     44897                          NaN                           NaT
     68373                          NaN                           NaT
     60938                          NaN                           NaT
     37003                          NaN                           NaT
     …                                …                             …
     47113          2016-11-14 11:14:39                           NaT
     19523          2016-10-30 10:23:36                           NaT
     34523          2016-10-26 13:18:16                           NaT
     22948          2016-10-15 12:24:42                           NaT
     4541           2016-10-18 13:14:51                           NaT

           order_estimated_delivery_date  Delivery Delay  Time to Deliver  \
     88457                    2017-05-22               0             1607
     44897                    2017-03-01               0             1689
     68373                    2018-11-12               0             1068
     60938                    2018-10-30               0             1081
     37003                    2018-10-25               0             1086
```

```
…                       …              …              …
47113                   2016-12-01             0             1779
19523                   2016-12-01             0             1779
34523                   2016-12-01             0             1779
22948                   2016-11-29             0             1781
4541                    2016-10-20             0             1821

       Order Freight  Order Price
88457           9.56       169.90
44897          21.92        39.70
68373           0.00         0.00
60938           0.00         0.00
37003           0.00         0.00
…               …          …
47113          16.00        59.90
19523          17.63        69.90
34523          43.50       357.80
22948          18.65        79.90
4541           63.34        72.89

[98824 rows x 12 columns]
```