In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
!python -m spacy download pt
!pip install tensorflow_addons
```

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import spacy
import regex as re
from datetime import datetime
from tqdm import tqdm
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, Standa
rdScaler
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
import warnings
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow_addons.metrics import F1Score
from tensorflow.keras.preprocessing.sequence import pad_sequences
from xgboost import XGBClassifier
```

In [ ]:
```
from google.colab import files
from datetime import datetime
api_token = files.upload()
```

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!pip install --upgrade --force-reinstall --no-deps kaggle

!kaggle datasets download -d olistbr/brazilian-ecommerce
!unzip '/content/brazilian-ecommerce.zip'
```

In [ ]:

```
customers_dataset = pd.read_csv('/content/olist_customers_dataset.csv')
geolocation_dataset = pd.read_csv('/content/olist_geolocation_dataset.csv')
order_items_dataset = pd.read_csv('/content/olist_order_items_dataset.csv')
order_payments_dataset = pd.read_csv('/content/olist_order_payments_dataset.csv')
order_reviews_dataset = pd.read_csv('/content/olist_order_reviews_dataset.csv')
orders_dataset = pd.read_csv('/content/olist_orders_dataset.csv')
products_dataset = pd.read_csv('/content/olist_products_dataset.csv')
sellers_dataset = pd.read_csv('/content/olist_sellers_dataset.csv')
product_category_name_translation = pd.read_csv('/content/product_category_name_translation.csv')

order_items_products = pd.merge(order_items_dataset,products_dataset,on='product_id')
order_items_products_sellers = pd.merge(order_items_products,sellers_dataset,on='seller_id')
two_order_items_products_sellers = pd.merge(order_items_products_sellers,orders_dataset,on='order_id')
two_order_items_products_sellers_customer = pd.merge(two_order_items_products_sellers,customers_dataset,on='customer_id')
two_order_items_products_sellers_customer_reviews = pd.merge(two_order_items_products_sellers_customer,order_reviews_dataset,on='order_id')
final_dataframe = pd.merge(two_order_items_products_sellers_customer_reviews,order_payments_dataset,on='order_id')

mapping = dict(zip(product_category_name_translation['product_category_name'].tolist(),product_category_name_translation['product_category_nam
```

```
e_english'].tolist()))
final_dataframe['product_category_name'] = final_dataframe['product_cat
egory_name'].map(mapping)

final_dataframe = final_dataframe.drop_duplicates(subset=['order_id','o
rder_purchase_timestamp','product_id','customer_unique_id','review_comm
ent_message'])
final_dataframe.drop(['order_id','product_id','seller_id','customer_uni
que_id'], axis=1, inplace=True)
final_dataframe.dropna(subset=['shipping_limit_date','order_purchase_ti
mestamp','order_delivered_carrier_date','order_delivered_customer_date'
,'order_estimated_delivery_date'], inplace=True)
intermediate_time = final_dataframe['order_delivered_customer_date'].ap
ply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S").date()) - final
_dataframe['order_purchase_timestamp'].apply(lambda x: datetime.strptim
e(x, "%Y-%m-%d %H:%M:%S").date())
final_dataframe['purchase-delivery difference'] = intermediate_time.app
ly(lambda x:x.days)
intermediate_time = final_dataframe['order_estimated_delivery_date'].ap
ply(lambda x: datetime.strptime(x, "%Y-%m-%d %H:%M:%S").date()) - final
_dataframe['order_delivered_customer_date'].apply(lambda x: datetime.st
rptime(x, "%Y-%m-%d %H:%M:%S").date())
final_dataframe['estimated-actual delivery difference'] = intermediate_
time.apply(lambda x:x.days)

final_dataframe['product_category_name'].fillna(value=final_dataframe[
'product_category_name'].mode()[0], inplace=True)
final_dataframe['product_name_lenght'].fillna(value=final_dataframe['pr
oduct_name_lenght'].mode()[0], inplace=True)
final_dataframe['product_description_lenght'].fillna(value=final_datafr
ame['product_description_lenght'].median(), inplace=True)
final_dataframe['product_photos_qty'].fillna(value=final_dataframe['pro
duct_photos_qty'].mode()[0], inplace=True)
final_dataframe['product_weight_g'].fillna(value=final_dataframe['produ
ct_weight_g'].mode()[0], inplace=True)
final_dataframe['product_length_cm'].fillna(value=final_dataframe['prod
uct_length_cm'].mode()[0], inplace=True)
final_dataframe['product_height_cm'].fillna(value=final_dataframe['prod
uct_height_cm'].mode()[0], inplace=True)
final_dataframe['product_width_cm'].fillna(value=final_dataframe['produ
```

```
ct_width_cm'].mode()[0], inplace=True)
final_dataframe['review_comment_message'].fillna(value='indisponível',
inplace=True)

final_dataframe['review_score'] = final_dataframe['review_score'].apply
(lambda x: 1 if x > 3 else 0)
final_dataframe['price_category'] = final_dataframe['price'].apply(lamb
da x:'expensive' if x>=139 else ('affordable' if x>=40 and x<139 else
'cheap'))
final_dataframe = final_dataframe[final_dataframe['order_status'] != 'c
anceled']
final_dataframe['purchase_delivery_diff_per_price'] = final_dataframe[
'purchase-delivery difference']/final_dataframe['price']
final_dataframe.drop(['shipping_limit_date','order_purchase_timestamp',
'order_approved_at','order_delivered_carrier_date','order_delivered_cus
tomer_date','order_estimated_delivery_date','customer_id'], axis=1, inp
lace=True)

labels = final_dataframe['review_score']
final_dataframe.drop('review_score', axis=1, inplace=True)
final_dataframe['review_availability'] = final_dataframe['review_commen
t_message'].apply(lambda x: 1 if x != 'indisponível' else 0)

X_train, X_test, y_train, y_test = train_test_split(final_dataframe, la
bels, stratify=labels, test_size=0.2, random_state=0)
print('Train data:', X_train.shape, y_train.shape)
print('Test data:', X_test.shape, y_test.shape)
```

```
Train data: (80348, 32) (80348,)
Test data: (20087, 32) (20087,)
```

In [ ]:

```
#References:
#https://stackoverflow.com/a/47218282
#https://stackoverflow.com/a/52057778
#https://stackoverflow.com/a/11332580
#https://stackoverflow.com/a/9532388

sp = spacy.load('pt')
all_stopwords = sp.Defaults.stop_words
```

```
def process_texts(texts):
    processed_text = []
    links = '(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)'
    dates = '([0-2][0-9]|(3)[0-1])(\/|\.)(((0)[0-9])|((1)[0-2]))(\/|\.)
\d{2,4}'
    for text in texts:
        text = re.sub('[\n\r]', ' ', text)
        text = re.sub(links, ' URL ', text)
        text = re.sub(dates, ' ', text)
        text = re.sub('[ \t]+$', '', text)
        text = re.sub('\W', ' ', text)
        text = re.sub('[0-9]+', ' numero ', text)
        text = re.sub('\s+', ' ', text)
        text = ' '.join(e for e in text.split() if e.lower() not in all
_stopwords)
        processed_text.append(text.lower().strip())
    return processed_text

def train_response(frame):
  f1 = frame[frame.iloc[:,1] == 0]
  f2 = frame[frame.iloc[:,1] == 1]
  global dict_frame, dict_f1, dict_f2
  dict_frame = dict(frame.iloc[:,0].value_counts())
  dict_f1 = dict(f1.iloc[:,0].value_counts())
  dict_f2 = dict(f2.iloc[:,0].value_counts())
  state_0, state_1 = [],[],
  for i in range(len(frame)):
    if frame.iloc[:,1][i] == 0:
      state_0.append(dict_f1.get(frame.iloc[:,0][i],0)/dict_frame[frame
.iloc[:,0][i]])
      state_1.append(float(1-state_0[-1]))
    else:
      state_1.append(dict_f2.get(frame.iloc[:,0][i],0)/dict_frame[frame
.iloc[:,0][i]])
      state_0.append(float(1-state_1[-1]))
  df3 = pd.DataFrame({'State_0':state_0, 'State_1':state_1})
  return df3.to_numpy()

def test_response(test):
```

```
  t_state_0, t_state_1 = [],[]
  for i in range(len(test)):
    if dict_frame.get(test[i]):
      t_state_0.append(dict_f1.get(test[i],0)/dict_frame.get(test[i]))
      t_state_1.append(dict_f2.get(test[i],0)/dict_frame.get(test[i]))
    else:
      t_state_0.append(0.5)
      t_state_1.append(0.5)
  df4 = pd.DataFrame({'State_0':t_state_0, 'State_1':t_state_1})
  return df4.to_numpy()
```

In [ ]:

```
strn = StandardScaler()
strn.fit(X_train[['price','freight_value','product_photos_qty','product
_weight_g', 'product_length_cm',
       'product_height_cm', 'product_width_cm', 'payment_value','purcha
se-delivery difference','estimated-actual delivery difference','purchas
e_delivery_diff_per_price']])
X_train_strn = strn.transform(X_train[['price','freight_value','product
_photos_qty','product_weight_g', 'product_length_cm',
       'product_height_cm', 'product_width_cm', 'payment_value','purcha
se-delivery difference','estimated-actual delivery difference','purchas
e_delivery_diff_per_price']])
X_test_strn = strn.transform(X_test[['price','freight_value','product_p
hotos_qty','product_weight_g', 'product_length_cm',
       'product_height_cm', 'product_width_cm', 'payment_value','purcha
se-delivery difference','estimated-actual delivery difference','purchas
e_delivery_diff_per_price']])

X_train_resp_prod_cat = train_response(pd.concat([X_train['product_cate
gory_name'], y_train], axis=1).reset_index(drop=True))
X_test_resp_prod_cat = test_response(X_test['product_category_name'].va
lues)

ohe_order_item = OneHotEncoder()
ohe_order_item.fit(X_train['order_item_id'].values.reshape(-1,1))
X_train_order_item = ohe_order_item.transform(X_train['order_item_id'].
values.reshape(-1,1)).toarray()
X_test_order_item = ohe_order_item.transform(X_test['order_item_id'].va
lues.reshape(-1,1)).toarray()
```

```
X_train_resp_payment_seq = train_response(pd.concat([X_train['payment_s
equential'], y_train], axis=1).reset_index(drop=True))
X_test_resp_payment_seq = test_response(X_test['payment_sequential'].va
lues)

ohe_payment_type = OneHotEncoder()
ohe_payment_type.fit(X_train['payment_type'].values.reshape(-1,1))
X_train_payment_type = ohe_payment_type.transform(X_train['payment_typ
e'].values.reshape(-1,1)).toarray()
X_test_payment_type = ohe_payment_type.transform(X_test['payment_type']
.values.reshape(-1,1)).toarray()

enc_price = OrdinalEncoder()
enc_price.fit(X_train['price_category'].values.reshape(-1,1))
enc_price.categories_ = [np.array([ 'cheap', 'affordable', 'expensive'
], dtype=object)]
X_train_cat_price = enc_price.transform(X_train['price_category'].value
s.reshape(-1,1))
X_test_cat_price = enc_price.transform(X_test['price_category'].values.
reshape(-1,1))

X_train_comment_preprocess = process_texts(X_train['review_comment_mess
age'])
X_test_comment_preprocess = process_texts(X_test['review_comment_messag
e'])
X_train['embedded_review_comment_message'] = pickle.load(open('/conten
t/drive/MyDrive/Olist/final_models/X_train_embedded_review_comment_mess
age.pkl','rb'))
X_test['embedded_review_comment_message'] = pickle.load(open('/content/
drive/MyDrive/Olist/final_models/X_test_embedded_review_comment_messag
e.pkl','rb'))


tok = Tokenizer()
tok.fit_on_texts(X_train_comment_preprocess)
X_train_text_input = pad_sequences(tok.texts_to_sequences(X_train_comme
nt_preprocess), padding='post')
X_test_text_input = pad_sequences(tok.texts_to_sequences(X_test_comment
_preprocess), padding='post')
```

```
X_train_final = np.concatenate((X_train_strn,X_train_resp_prod_cat,X_train_order_item,
       X_train_resp_payment_seq,X_train_payment_type,X_train_cat_price,X_train['review_availability'].values.reshape(-1,1),
       np.vstack(X_train['embedded_review_comment_message'].values)), axis=1)

X_test_final = np.concatenate((X_test_strn,X_test_resp_prod_cat, X_test_order_item,
       X_test_resp_payment_seq,X_test_payment_type,X_test_cat_price,X_test['review_availability'].values.reshape(-1,1),
       np.vstack(X_test['embedded_review_comment_message'].values)), axis=1)
```

In [ ]:
```
X_final_truncated = pickle.load(open( "/content/drive/MyDrive/Olist/final_models/X_final_truncated.pkl","rb"))
X_train_final_truncated = X_final_truncated[:X_train_final.shape[0],:]
X_test_final_truncated = X_final_truncated[X_train_final.shape[0]:,:]
X_train_final_new, X_cv_final, y_train_new, y_cv = train_test_split(X_train_final_truncated, y_train, stratify=y_train, test_size=0.2, random_state=45)

X_train_encode = pickle.load(open('/content/drive/MyDrive/Olist/final_models/X_train_encode.pkl','rb'))
X_test_encode = pickle.load(open('/content/drive/MyDrive/Olist/final_models/X_test_encode.pkl','rb'))
```

In [ ]:
```
def final(x,y,algo,reduction=None):

  if reduction=='hard_svd':
    if algo=='knn':
      knn_truncate = pickle.load(open('/content/drive/MyDrive/Olist/final_models/knn_clf_truncated.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,knn_truncate.predict(x),'macro'))
    if algo=='log_reg':
      log_reg_truncate = pickle.load(open('/content/drive/MyDrive/Olist/final_models/log_reg_truncated.pkl','rb'))
```

```
      return 'F1 score: {}'.format(f1_score(y,log_reg_truncate.predict(
x),'macro'))
    if algo=='rf':
      rf_truncate = pickle.load(open('/content/drive/MyDrive/Olist/fina
l_models/rf_truncated.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,rf_truncate.predict(x),'m
acro'))
    if algo=='xgb':
      xgb_truncate = pickle.load(open('/content/drive/MyDrive/Olist/fin
al_models/xgb_truncated.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,xgb_truncate.predict(x),
'macro'))
    if algo=='mlp':
      mlp_truncate = load_model('/content/drive/MyDrive/Olist/final_mod
els/mlp_truncated.h5')
      return f1_score(y_test, (mlp_truncate.predict(x)>0.5).astype(int
),'macro')

  elif reduction=='autoencoders':
    if algo=='knn':
      knn_encode = pickle.load(open('/content/drive/MyDrive/Olist/final
_models/knn_clf_encode.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,knn_encode.predict(x),'ma
cro'))
    if algo=='log_reg':
      log_reg_encode = pickle.load(open('/content/drive/MyDrive/Olist/f
inal_models/log_reg_encode.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,log_reg_encode.predict(x
),'macro'))
    if algo=='rf':
      rf_encode = pickle.load(open('/content/drive/MyDrive/Olist/final_
models/rf_encode.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,rf_encode.predict(x),'mac
ro'))
    if algo=='xgb':
      xgb_encode = pickle.load(open('/content/drive/MyDrive/Olist/final
_models/xgb_encode.pkl','rb'))
      return 'F1 score: {}'.format(f1_score(y,xgb_encode.predict(x),'ma
cro'))
    if algo=='mlp':
```

```
        mlp_encode = load_model('/content/drive/MyDrive/Olist/final_models/mlp_encode.h5')
        return 'F1 score: {}'.format(f1_score(y_test, (mlp_encode.predict(x)>0.5).astype(int),'macro'))

  else:
    if algo=='rnn':
      rnn_data = load_model('/content/drive/MyDrive/Olist/final_models/model_rnn.h5')
      return 'F1 score: {}'.format(f1_score(y_test, (rnn_data.predict([x[0],x[1][:,:-300]])>0.5).astype(int),'macro'))
    if algo=='cnn_rnn':
      cnn_rnn_data = load_model('/content/drive/MyDrive/Olist/final_models/model_cnn_rnn.h5')
      return 'F1 score: {}'.format(f1_score(y_test, (cnn_rnn_data.predict([x[0],x[1][:,:-300]])>0.5).astype(int),'macro'))
```

In [ ]:
```
%%time
final(X_test_final_truncated,y_test,'xgb','autoencoders')
```

```
CPU times: user 218 ms, sys: 63.5 ms, total: 282 ms
Wall time: 1.23 s
```

Out[ ]:
```
'F1 score: 0.8738261430213325'
```

In [ ]:
```
%%time
final(X_test_final_truncated,y_test,'mlp','hard_svd')
```

```
CPU times: user 884 ms, sys: 89 ms, total: 973 ms
Wall time: 1.65 s
```

Out[ ]:
```
0.9091929649494824
```

In [ ]:
```
%%time
final([X_test_text_input,X_test_final],y_test,'rnn')
```

```
CPU times: user 10.3 s, sys: 7.32 s, total: 17.6 s
Wall time: 1min 16s
```

Out[ ]: 'F1 score: 0.9182254491754861'

In [ ]:
```
%%time
final([X_test_text_input,X_test_final],y_test,'cnn_rnn')
```

```
CPU times: user 5.75 s, sys: 1.6 s, total: 7.35 s
Wall time: 33.9 s
```

Out[ ]: 'F1 score: 0.902708001470408'