Final Functions

In [2]:
```
import numpy as np
import pandas as pd
from unidecode import unidecode
import pickle
import joblib

from math import radians
from sklearn.metrics.pairwise import haversine_distances
import warnings
warnings.filterwarnings("ignore")
from scipy.sparse import hstack
from sklearn.metrics import confusion_matrix,f1_score
```

Function which gives predicted review_score

In [3]:
```python
def function1(x):

    """This function takes single query point as input
        and
       preprocess,featurize itand finally gives the predicted target score"""

    ##### Convert to datetime type ####
    x["order_purchase_timestamp"] = pd.to_datetime(x["order_purchase_timestamp"])
    x["order_approved_at"] = pd.to_datetime(x["order_approved_at"])
    x["order_delivered_carrier_date"] = pd.to_datetime(x["order_delivered_carrier_date"])
    x["order_delivered_customer_date"] = pd.to_datetime(x["order_delivered_customer_date"])
    x["order_estimated_delivery_date"] = pd.to_datetime(x["order_estimated_delivery_date"])
    x["shipping_limit_date"] = pd.to_datetime(x["shipping_limit_date"])


    ################################ BASIC FEATURES ########################################
    #### Time based features #####
    #Time of estimated delivery
    x["estimated_time"] = round((x["order_estimated_delivery_date"]-x["order_purchase_timestamp"]).total_s
    #Time taken for delivery
    x["actual_time"]    = round((x["order_delivered_customer_date"]-x["order_purchase_timestamp"]).total_s
    #Difference between actual delivery time and estimated delivery time
    x["diff_actual_estimated"]   = round((x["order_delivered_customer_date"]-x["order_estimated_delivery_d
    # difference between purchase time and approved time
    x["diff_purchased_approved"] = round((x["order_approved_at"]-x["order_purchase_timestamp"]).total_seco
    # difference between purchase time and courrier delivery time
    x["diff_purchased_courrier"] = round((x["order_delivered_carrier_date"]-x["order_purchase_timestamp"])

    # some more features from timestamp(days, weekday, month, hour)
    x["delivery_day"]   = x["order_delivered_customer_date"].weekday()
    x["delivery_date"]  = x["order_delivered_customer_date"].day
    x["delivery_month"] = x["order_delivered_customer_date"].month
    x["delivery_hour"]  = x["order_delivered_customer_date"].hour

    x["purchased_day"]   = x["order_purchase_timestamp"].weekday()
    x["purchased_date"]  = x["order_purchase_timestamp"].day
    x["purchased_month"] = x["order_purchase_timestamp"].month
    x["purchased_hour"]  = x["order_purchase_timestamp"].hour



    ######## Distance based features ###############
    ### Distance between customer and seller ####
    cust_loc    = np.array([radians(x.lat_customer),radians(x.lng_customer)])
    seller_loc = np.array([radians(x.lat_seller),radians(x.lng_seller)])

    dist = haversine_distances([cust_loc, seller_loc])*6371
    x["distance"] = dist[0,1]

    ### Speed
    x["speed"] = x["distance"]/x["actual_time"]

    ### Binary features like same city or not, same state or not ###
    ### same state
    x["same_state"] = 1 if (x.customer_state == x.seller_state) else 0

    ### same city
    x["customer_city"] = unidecode(x["customer_city"].lower())
    x["seller_city"] = unidecode(x["seller_city"].lower())

    x["same_city"]   = 1 if (x.customer_city == x.seller_city) else 0

    ### late_shipping

    x["late_shipping"] = 1 if (x.shipping_limit_date < x.order_delivered_carrier_date) else 0

    ### high_freight
    x["high_freight"]  = 1 if (x.price < x.freight_value) else 0

    ### size of the product
    x["size"] = x["product_length_cm"]*x["product_height_cm"]*x["product_width_cm"]

    ########################## ADVANCED FEATURES ######################################
    ################# customer_seller similarity based on order_item_id ###############
    order_seller = pd.read_pickle("order_seller_table.pkl")
    total_order_id = pd.read_pickle("total_order_id.pkl")
```

```
    total_seller_id = pd.read_pickle("total_seller_id.pkl")
    user_order = pd.read_pickle("user_order_table.pkl")
    user_total = pd.read_pickle("user_total.pkl")
    order_total = pd.read_pickle("order_total.pkl")

    x["seller_share"] = order_seller.loc[(x["order_item_id"],x["seller_id"])]/total_order_id[x["order_item
    x["bs_share"]     = order_seller.loc[(x["order_item_id"],x["seller_id"])]/total_seller_id[x["seller_ic

    x["cust_share"]   = user_order.loc[(x["order_item_id"],x["customer_unique_id"])]/order_total[x["order_
    x["bu_share"]     = user_order.loc[(x["order_item_id"],x["customer_unique_id"])]/user_total[x["custome

    ### similarity
    x["similarity"]   = np.dot([x["seller_share"],x["bs_share"]] , [x["cust_share"],x["bu_share"]])

    ######################## customer_seller similarity based on category_name ###############
    cat_seller = pd.read_pickle("cat_seller_table.pkl")
    total_cat_order_id = pd.read_pickle("total_cat_order_id.pkl")
    total_cat_seller_id = pd.read_pickle("total_cat_seller_id.pkl")
    user_cat = pd.read_pickle("user_cat_table.pkl")
    user_cat_total = pd.read_pickle("user_cat_total.pkl")
    order_cat_total = pd.read_pickle("order_cat_total.pkl")

    x["seller_category_share"] = cat_seller.loc[(x["product_category_name"],x["seller_id"])]/total_cat_orc
    x["cat_seller_share"]      = cat_seller.loc[(x["product_category_name"],x["seller_id"])]/total_cat_sel

    x["cust_category_share"]   = user_cat.loc[(x["product_category_name"],x["customer_unique_id"])]/order_
    x["cat_cust_share"]        = user_cat.loc[(x["product_category_name"],x["customer_unique_id"])]/user_c

    ### similarity
    x["similarity_using_cat"]  = np.dot([x["seller_category_share"],x["cat_seller_share"]],[x["cust_catego
    ############################################################################################
    ########### Total customers for each seller and total seller for each customer #############
    dict_seller = pd.read_pickle('dict_seller.pkl')
    dict_customer = pd.read_pickle('dict_customer.pkl')
    dict_seller_order = pd.read_pickle("dict_seller_order.pkl")

    x["num_of_customers_for_seller"] = dict_seller[x["seller_id"]]
    x["num_of_sellers_for_cust"]     = dict_customer[x["customer_unique_id"]]
    x["total_order_for_seller"]      = dict_seller_order[x["seller_id"]]

################################################################################################
    label = joblib.load("seller_id_encode.pkl")
    x["seller_id_label"] = label.transform([x["seller_id"]])

    label = joblib.load("product_id_encode.pkl")
    x["product_id_label"] = label.transform([x["product_id"]])
    #################################################
    ############## countvectorizers ################
    ### payment_type
    vec = joblib.load("count_vect_payment_1.pkl")
    x_te_pay_type = vec.transform([x["payment_type"]])
    #### order_item_id
    x["order_item_id"] = x["order_item_id"].astype(str)
    vec = joblib.load("count_vect_item_1.pkl")
    x_te_id = vec.transform([x["order_item_id"]])
    ### product_category_name
    vec = joblib.load("count_vect_cat_1.pkl")
    x_te_cat = vec.transform([x["product_category_name"]])
    ####################################################
    ############### standardization ###################
    num = x[["payment_sequential","payment_installments","payment_value","seller_id_label","product_id_lat
              "bs_share","cust_share",
          "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
           "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
              "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
              "num_of_sellers_for_cust","total_order_for_seller",
           "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approvec
           "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

    norm = joblib.load("std_num_1.pkl")
    num = np.array(num).reshape(1,-1)
    x_te_num = norm.transform(num)
    #################################################################
    ######## concatenate all features to create query point #########
    query_point = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x.same_state,
                              x.same_city,x.late_shipping,x.high_freight)).toarray()
```

```
    ##################################################################
    model = joblib.load("binary_model.pkl")
    if model.predict(query_point) == 1:
        prediction = 5

    else:
        ################ CUSTOM ENSEMBLE ##########################
        label = joblib.load("seller_encode_2.pkl")
        x["seller_id_enc"] = label.transform([x["seller_id"]])

        label = joblib.load("product_encode_2.pkl")
        x["product_id_enc"] = label.transform([x["product_id"]])
        ####
        ######## countvectorizers ###########
        ### payment_type
        vec = joblib.load("countvec_pay_2.pkl")
        x_te_pay_type = vec.transform([x["payment_type"]])

        #### order_item_id
        x["order_item_id"] = x["order_item_id"].astype(str)
        vec = joblib.load("countvec_item_2.pkl")
        x_te_id = vec.transform([x["order_item_id"]])

        ### product_category_name
        vec = joblib.load("countvec_cat_2.pkl")
        x_te_cat = vec.transform([x["product_category_name"]])

        ####################################################
        ############### standardization ####################
        num = x[["payment_sequential","payment_installments","payment_value","seller_id_enc","product_id_e
                 "bs_share","cust_share",
                 "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_de
                 "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","de
                 "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_
                 "num_of_sellers_for_cust","total_order_for_seller",
                 "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_ap
                 "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

        norm = joblib.load("std_num_2.pkl")
        num = np.array(num).reshape(1,-1)
        x_te_num = norm.transform(num)
        #################################################################
        ######## concatenate all features to create query point #########
        query_point = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x.same_state,
                              x.same_city,x.late_shipping,x.high_freight)).toarray()

        models = joblib.load("base_models.pkl")
        predicts = []
        for model in models:
            predicts.append(model.predict(query_point))
        predicts = np.array(predicts).reshape(1,-1)

        meta_clf = joblib.load("meta_clf.pkl")
        prediction = meta_clf.predict(predicts)

################################################################################################

    return prediction
```

Function which calculates Macro F1 scores for given set of test data

In [2]:

```
def function2(x,y):

    """This function takes dataset as input
        and
       preprocess,featurize itand finally gives the predicted target score and calculates the macro F1 sco

    ##### Convert to datetime type ####
    x["order_purchase_timestamp"] = pd.to_datetime(x["order_purchase_timestamp"])
    x["order_approved_at"] = pd.to_datetime(x["order_approved_at"])
    x["order_delivered_carrier_date"] = pd.to_datetime(x["order_delivered_carrier_date"])
    x["order_delivered_customer_date"] = pd.to_datetime(x["order_delivered_customer_date"])
    x["order_estimated_delivery_date"] = pd.to_datetime(x["order_estimated_delivery_date"])
    x["shipping_limit_date"] = pd.to_datetime(x["shipping_limit_date"])


    ################################ BASIC FEATURES ########################################
    #### Time based features #####
    #Time of estimated delivery
    x["estimated_time"] = (x["order_estimated_delivery_date"]-x["order_purchase_timestamp"]).apply(
                                                                                        lambda x: x.total_sec
    #Time taken for delivery
    x["actual_time"]    = (x["order_delivered_customer_date"]-x["order_purchase_timestamp"]).apply(
                                                                                        lambda x: x.total_sec
    #Difference between actual delivery time and estimated delivery time
    x["diff_actual_estimated"]   = (x["order_delivered_customer_date"]-x["order_estimated_delivery_date"])
                                                                                        lambda x: x.total_sec
    # difference between purchase time and approved time
    x["diff_purchased_approved"] = (x["order_approved_at"]-x["order_purchase_timestamp"]).apply(
                                                                                        lambda x: x.total_sec
    # difference between purchase time and courrier delivery time
    x["diff_purchased_courrier"] = (x["order_delivered_carrier_date"]-x["order_purchase_timestamp"]).apply
                                                                                        lambda x: x.total_sec

    # some more features from timestamp(days, weekday, month, hour)
    x["delivery_day"]   = x["order_delivered_customer_date"].apply(lambda x: x.weekday())
    x["delivery_date"]  = x["order_delivered_customer_date"].apply(lambda x: x.day)
    x["delivery_month"] = x["order_delivered_customer_date"].apply(lambda x: x.month)
    x["delivery_hour"]  = x["order_delivered_customer_date"].apply(lambda x: x.hour)

    x["purchased_day"]   = x["order_purchase_timestamp"].apply(lambda x: x.weekday())
    x["purchased_date"]  = x["order_purchase_timestamp"].apply(lambda x: x.day)
    x["purchased_month"] = x["order_purchase_timestamp"].apply(lambda x: x.month)
    x["purchased_hour"]  = x["order_purchase_timestamp"].apply(lambda x: x.hour)



    ######## Distance based features ###############
    ### Distance between customer and seller ####
    X = []  # list to store customer latitude and longitude
    Y = []  # list to store seller latitude and longitude

    for i in range(len(x)):
        X.append([radians(x.lat_customer[i]),radians(x.lng_customer[i])])
        Y.append([radians(x.lat_seller[i]),radians(x.lng_seller[i])])

    #converting to numpy array
    cust_loc = np.array(X)
    seller_loc = np.array(Y)

    distance=[]
    for i in range(len(x)):
        #calculating distance and multiplying by radius of earth(6371) to get distance in km
        dist = haversine_distances([cust_loc[i], seller_loc[i]])*6371
        distance.append(dist[0,1])

    x["distance"] = distance

    ### Speed
    x["speed"] = x["distance"]/x["actual_time"]

    ### Binary features like same city or not, same state or not ###
    ### same state
    same = []
    for i in range(len(x)):
        if x.customer_state[i] == x.seller_state[i]:
            same.append(1)
```

```
        else:
            same.append(0)

    x["same_state"] = same

    ### same city
    x['customer_city'] = x.apply(lambda row: unidecode(row['customer_city'].lower()), axis=1)
    x['seller_city']   = x.apply(lambda row: unidecode(row['seller_city'].lower()), axis=1)

    same = []
    for i in range(len(x)):
        if x.customer_city[i] == x.seller_city[i]:
            same.append(1)
        else:
            same.append(0)

    x["same_city"] = same

    ### late_shipping
    late = []
    for i in range(len(x)):
        if x.shipping_limit_date[i] < x.order_delivered_carrier_date[i]:
            late.append(1)
        else:
            late.append(0)

    x["late_shipping"] = late

    ### high_freight
    high = []
    for i in range(len(x)):
        if x.price[i] < x.freight_value[i]:
            high.append(1)
        else:
            high.append(0)

    x["high_freight"] = high

    ### size of the product
    x["size"] = x["product_length_cm"]*x["product_height_cm"]*x["product_width_cm"]

    ########################## ADVANCED FEATURES ######################################
    ################# customer_seller similarity based on order_item_id ###############
    order_seller = pd.read_pickle("order_seller_table.pkl")
    total_order_id = pd.read_pickle("total_order_id.pkl")
    total_seller_id = pd.read_pickle("total_seller_id.pkl")
    user_order = pd.read_pickle("user_order_table.pkl")
    user_total = pd.read_pickle("user_total.pkl")
    order_total = pd.read_pickle("order_total.pkl")

    seller_share = []
    bs_share = []
    for i in range(len(x)):
        seller_share.append((order_seller.loc[(x["order_item_id"][i],x["seller_id"][i])]/total_order_id[x[
        bs_share.append((order_seller.loc[(x["order_item_id"][i],x["seller_id"][i])]/total_seller_id[x["se

    x["seller_share"] = seller_share
    x["bs_share"] = bs_share

    cust_share = []
    bu_share = []
    for i in range(len(x)):

        cust_share.append((user_order.loc[(x["order_item_id"][i],x["customer_unique_id"][i])]/order_total[

        bu_share.append((user_order.loc[(x["order_item_id"][i],x["customer_unique_id"][i])]/user_total[x["

    x["cust_share"] = cust_share
    x["bu_share"] = bu_share

    ### similarity
    similarity = []
    for i in range(len(x)):
         similarity.append((np.dot([x["seller_share"][i],x["bs_share"][i]] , [x["cust_share"][i],x["bu_sha

    x["similarity"] = similarity
```

```
    ######################## customer_seller similarity based on category_name ###############
    cat_seller = pd.read_pickle("cat_seller_table.pkl")
    total_cat_order_id = pd.read_pickle("total_cat_order_id.pkl")
    total_cat_seller_id = pd.read_pickle("total_cat_seller_id.pkl")
    user_cat = pd.read_pickle("user_cat_table.pkl")
    user_cat_total = pd.read_pickle("user_cat_total.pkl")
    order_cat_total = pd.read_pickle("order_cat_total.pkl")

    seller_share = []
    bs_share = []
    for i in range(len(x)):
        seller_share.append((cat_seller.loc[(x["product_category_name"][i],x["seller_id"][i])]/total_cat_o
        bs_share.append((cat_seller.loc[(x["product_category_name"][i],x["seller_id"][i])]/total_cat_selle

    x["seller_category_share"] = seller_share
    x["cat_seller_share"] = bs_share


    cust_share = []
    bu_share = []
    for i in range(len(x)):
        cust_share.append((user_cat.loc[(x["product_category_name"][i],x["customer_unique_id"][i])]/order_
        bu_share.append((user_cat.loc[(x["product_category_name"][i],x["customer_unique_id"][i])]/user_cat

    x["cust_category_share"] = cust_share
    x["cat_cust_share"] = bu_share

    ### similarity
    similarity = []
    for i in range(len(x)):
         similarity.append((np.dot([x["seller_category_share"][i],x["cat_seller_share"][i]] , [x["cust_cat

    x["similarity_using_cat"] = similarity
    ###########################################################################################
    ########### Total customers for each seller and total seller for each customer #############
    dict_seller = pd.read_pickle('dict_seller.pkl')
    dict_customer = pd.read_pickle('dict_customer.pkl')
    dict_seller_order = pd.read_pickle("dict_seller_order.pkl")

    num_customers = []
    for i in range(len(x)):
        num = dict_seller[x["seller_id"][i]]
        num_customers.append(num)
    x["num_of_customers_for_seller"] = num_customers

    num_sellers = []
    for i in range(len(x)):
        num = dict_customer[x["customer_unique_id"][i]]
        num_sellers.append(num)
    x["num_of_sellers_for_cust"] = num_sellers


    num_orders = []
    for i in range(len(x)):
        num = dict_seller_order[x["seller_id"][i]]
        num_orders.append(num)

    x["total_order_for_seller"] = num_orders

################################################################################################
    label = joblib.load("seller_id_encode.pkl")
    x["seller_id_label"] = label.transform(x["seller_id"])

    label = joblib.load("product_id_encode.pkl")
    x["product_id_label"] = label.transform(x["product_id"])
    ##################################################
    ############## countvectorizers ################
    ### payment_type
    vec = joblib.load("count_vect_payment_1.pkl")
    x_te_pay_type = vec.transform(x["payment_type"].values)
    #### order_item_id
    x["order_item_id"] = x["order_item_id"].astype(str)
    vec = joblib.load("count_vect_item_1.pkl")
    x_te_id = vec.transform(x["order_item_id"].values)
    ### product_category_name
    vec = joblib.load("count_vect_cat_1.pkl")
    x_te_cat = vec.transform(x["product_category_name"].values)
```

```
####################################################
############### standardization ###################
num = x[["payment_sequential","payment_installments","payment_value","seller_id_label","product_id_lab
          "bs_share","cust_share",
      "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_descripti
       "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","delivery
          "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_of_
          "num_of_sellers_for_cust","total_order_for_seller",
       "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_approved
       "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

norm = joblib.load("std_num_1.pkl")
x_te_num = norm.transform(num.values)
######### binary features ###########
x_te_same_state = x.same_state.values.reshape(-1,1)
x_te_same_city = x.same_city.values.reshape(-1,1)
x_te_late_shipping = x.late_shipping.values.reshape(-1,1)
x_te_high_freight = x.high_freight.values.reshape(-1,1)

#################################################################
######## concatenate all features to create query point #########
test = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x_te_same_state,
                          x_te_same_city,x_te_late_shipping,x_te_high_freight)).toarray()

##################################################################
# array with zeros to store predicted target values
predicted_targets = np.zeros(shape=(y.shape))

ind_1234 = [] #list of indices of points which are predicted to be 1,2,3,4
model = joblib.load("binary_model.pkl")
prediction = model.predict(test)
for i,pred in enumerate(prediction):
    if pred == 1:
        predicted_targets[i]=5
    else:
        ind_1234.append(i)


x2 = x.loc[ind_1234]  # set which deosnot consists review_score 5

    ################ custom ensemble ##########################
label = joblib.load("seller_encode_2.pkl")
x2["seller_id_enc"] = label.transform(x2["seller_id"])

label = joblib.load("product_encode_2.pkl")
x2["product_id_enc"] = label.transform(x2["product_id"])
####
############## countvectorizers ################
### payment_type
vec = joblib.load("countvec_pay_2.pkl")
x_te_pay_type = vec.transform(x2["payment_type"].values)

#### order_item_id
x2["order_item_id"] = x2["order_item_id"].astype(str)
vec = joblib.load("countvec_item_2.pkl")
x_te_id = vec.transform(x2["order_item_id"].values)

### product_category_name
vec = joblib.load("countvec_cat_2.pkl")
x_te_cat = vec.transform(x2["product_category_name"].values)

    ####################################################
    ############### standardization ###################
num = x2[["payment_sequential","payment_installments","payment_value","seller_id_enc","product_id_enc"
             "bs_share","cust_share",
             "lat_customer","lng_customer","lat_seller","lng_seller","product_name_lenght","product_de
             "product_photos_qty","product_weight_g","size","price","delivery_day","delivery_date","de
             "delivery_hour","purchased_day","purchased_date","purchased_month","purchased_hour","num_
             "num_of_sellers_for_cust","total_order_for_seller",
             "freight_value","estimated_time","actual_time","diff_actual_estimated","diff_purchased_ap
             "diff_purchased_courrier","distance","speed","similarity","similarity_using_cat"]]

norm = joblib.load("std_num_2.pkl")
x_te_num = norm.transform(num)
####### binary features #######
x_te_same_state = x2.same_state.values.reshape(-1,1)
x_te_same_city = x2.same_city.values.reshape(-1,1)
```

```python
    x_te_late_shipping = x2.late_shipping.values.reshape(-1,1)
    x_te_high_freight = x2.high_freight.values.reshape(-1,1)


            ##################################################################
    ######## concatenate all features to create query point #########
    test2 = hstack((x_te_pay_type,x_te_id,x_te_cat,x_te_num,x_te_same_state,
                              x_te_same_city,x_te_late_shipping,x_te_high_freight)).toarray()

    models = joblib.load("base_models.pkl")
    predicts = []
    for model in models:
        predicts.append(model.predict(test2))

    predicts = np.array(predicts).reshape(-1,150)

    meta_clf = joblib.load("meta_clf.pkl")

    prediction = meta_clf.predict(predicts)

    for i,j in enumerate(ind_1234):
        predicted_targets[j] = prediction[i]

############################################################################################################
################ Calculation of Macro F1 score ############################################################

    macro_f1 = f1_score(y,predicted_targets,average="macro",labels=[1,2,3,4,5])

    return macro_f1
```

In [ ]: