Appendices

A Machine Learning model code

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
import pickle


def clean_data():
    dataf = pd.read_csv("../data1.csv", sep=';')
    # standardize and data cleaning
    dataf['POST'] = dataf['req_type'].str.contains('POST')
    dataf['POST'] = dataf['POST'].astype('int')
    dataf['GET'] = dataf['req_type'].str.contains('GET')
    dataf['GET'] = dataf['GET'].astype('int')
    dataf = dataf.drop(columns=['url', 'req_type'])
    return dataf

# split the training and test data into different data sets
def split_train_test(df):
    split = int(len(df)*2.0/3)
    training = df[:split]
    test = df[split:]
    training_col= ['is_json', 'pl_isprebid', 'pl_appid', 'pl_domain', 'pl_imp', 'pl_
                   'pl_source', 'pl_arr_data_bidder', 'pl_arr_data_ad_unit_code', 'pl
                   'pld_arr_data_media_type', 'pl_schain', 'pl_sdk', 'pl_tags', 'pl_
                   'pl_wfv', 'pl_adv', 'pl_cdl', 'pl_cet', 'pl_ct', 'pl_crt', 'pl_csd
                   'pl_wrapper', 'pl_session', 'pl_auctions', 'pl_referrerUri', 'pl_
                   'pl_resources', 'pl_referrer', 'pl_documentWriteIntervention', 'pl
                   'pl_timingsv2', 'pl_st', 'pl_results', 'pl_m', 'pl_fp', 'pl_timing
                   'pl_csr', 'pl_req', 'pl_hs', 'pl_ccg', 'pl_rev', 'pl_s', 'pl_hsi'
                   'pl_o', 'pl_rf', 'pl_top', 'pl_u', 'pl_v', 'pl_provider', 'pl_ttd_
                   'pl_dl', 'pl_ul', 'pl_de', 'pl_dt', 'pl_sd', 'pl_sr', 'pl_vp', 'pl
                   'pl_tid', 'pl_slc', 'pl_cd5', 'pl_z', 'pl_cmpId', 'pl_configuration
                   'pl_sessionId', 'pl_displayType', 'pl_bundle', 'pl_cw', 'pl_lsw',
                   'pl_release_ver', 'pl_page', 'pl_tab_id', 'pl_per_page', 'pl_accou
                   'pl_metadata', 'pl_sensor_data', 'pl_operationName', 'pl_extension
                   'pl_tv', 'pl_site_id', 'pl_browser', 'pl_sfv', 'pl_ecs', 'pl_ists
                   'pl_idt', 'pl_oid', 'pl_ucis', 'pl_nach', 'pl_scr_x', 'pl_scr_y',
                   'pl_analytics', 'pl_experienceCloud', 'pl_execute', 'pl_prefetch'
                   'pl_payload', 'pl_trackingId', 'pl_distinctId', 'pl_campaign', 'pl
                   'pl_rec_value', 'pl_user_id', 'pl_res', 'pl_ref', 'pl_rst', 'pl_c
                   'pl_expires', 'pl_samesite', 'pl_originalValue', 'pl_currentURI',
                   'pl_sts', 'pl_slts', 'pl_inc', 'pl_pvid', 'pl_localefilter', 'pl_f
                   'pl_excludepages', 'pl_includepath', 'pl_browser_fingerprint_id',
                   'pl_vertical', 'pl_token', 'pl_session_id', 'pl_aip', 'pl_action'
    x_train =training[training_col]
    x_test = test[training_col]
    y_train = training['invasive']
    y_test = test['invasive']
```

```
    return x_train, x_test, y_train, y_test

def save_pickle(model, name):
    with open(name,'wb') as files:
        pickle.dump(model, files)

def logistic_regression(x_train, x_test, y_train, y_test):
    LogReg = LogisticRegression()
    LogReg.fit(x_train.values, y_train.values)
    score = LogReg.score(x_test.values, y_test.values)
    print("Logistic Regression Accuracy:", score)
    save_pickle(LogReg, 'logreg.pickle')
    return LogReg


def classify_decision_tree(x_train, x_test, y_train, y_test):
    dt = DecisionTreeClassifier()
    dt.fit(x_train.values, y_train.values)
    score = dt.score(x_test.values, y_test.values)
    print("Decision Tree Accuracy: ", score)
    save_pickle(dt, "decisionTree.pickle")
    return dt

def classify_svm(x_train, x_test, y_train, y_test):
    sv = SVC()
    sv.fit(x_train.values, y_train.values)
    score = sv.score(x_test.values, y_test.values)
    print("Support Vector Machine Accuracy:", score)
    save_pickle(sv, "svm.pickle")
    return sv

def plot_cm(actual, predicted):
    conf_m = metrics.confusion_matrix(actual, predicted)
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =  conf_m, display_

    accuracy = metrics.accuracy_score(actual, predicted)
    print("accuracy ", accuracy)
    precision = metrics.precision_score(actual, predicted)
    print("precision ", precision)
    sensitivity = metrics.recall_score(actual, predicted)
    print("sensitivity ", sensitivity)
    specificity = metrics.recall_score(actual, predicted, pos_label=0)
    print("Specificity ", specificity)
    f1score = metrics.f1_score(actual, predicted)
    print("f1score ", f1score)
    cm_display.plot()
    plt.show()
    return conf_m

def print_conf_matrix(cm):
    print("TP = ", cm[0][0])
    print("TN = ", cm[1][1])
    print("FP = ", cm[0][1])
    print("FN = ", cm[1][0])

if __name__ == '__main__':
```

```
df = clean_data()
x_train, x_test, y_train, y_test = split_train_test(df)
logReg = logistic_regression( x_train, x_test, y_train, y_test)
dt = classify_decision_tree( x_train, x_test, y_train, y_test)
svm = classify_svm( x_train, x_test, y_train, y_test)

lr_pred = logReg.predict(x_test.values)
lr_cm = plot_cm(y_test.values, lr_pred)

print("Logistic regression Confusion Matrix")
print_conf_matrix(lr_cm)
dt_pred = dt.predict(x_test.values)
dt_cm =plot_cm(y_test.values, dt_pred)
print("Decision tree Confusion Matrix")
print_conf_matrix(dt_cm)
svm_pred = svm.predict(x_test.values)
svm_cm = plot_cm(y_test.values, svm_pred)
print("Support Vector Machine Confusion Matrix")
print_conf_matrix(svm_cm)
```

B API service

```
from flask import Flask, request, jsonify
import numpy as np
import pickle as p

app = Flask(__name__)

def transformData(is_json, pl_isprebid, pl_appid, pl_domain, pl_imp, pl_site, pl_dev
                  pl_arr_data_bidder, pl_arr_data_ad_unit_code, pl_arr_data
                  pl_arr_data_media_type, pl_schain, pl_sdk, pl_tags, pl_r
                  pl_wfv, pl_adv, pl_cdl, pl_cet, pl_ct, pl_crt, pl_csd, p
                  pl_session, pl_auctions, pl_referrerUri, pl_referrerHostn
                  pl_documentWriteIntervention, pl_errorCount, pl_si, pl_p
                  pl_m, pl_fp, pl_timing, pl_av, pl_user, pl_a, pl_dyn, pl_
                  pl_hsi, pl_partner, pl_gdpr, pl_nbPage, pl_o, pl_rf, pl_t
                  pl_fmt, pl_wmode, pl_t, pl_dl, pl_ul, pl_de, pl_dt, pl_sc
                  pl_cid, pl_tid, pl_slc, pl_cd5, pl_z, pl_cmpId, pl_config
                  pl_displayType, pl_bundle, pl_cw, pl_lsw, pl_event, pl_ty
                  pl_tab_id, pl_per_page, pl_accountId, pl_projectId, pl_e
                  pl_operationName, pl_extensions, pl_ippd, pl_sv, pl_tv, p
                  pl_ists, pl_fas, pl_biw, pl_bih, pl_idt, pl_oid, pl_ucis
                  pl_window, pl_address, pl_analytics, pl_experienceCloud,
                  pl_isdeviceaccessgranted, pl_payload, pl_trackingId, pl_
                  pl_is_vpv, pl_session_only, pl_rec_value, pl_user_id, pl_
                  pl_storeType, pl_storeKey, pl_expires, pl_samesite, pl_o
                  pl_plid, pl_url, pl_sid, pl_sts, pl_slts, pl_inc, pl_pvid
                  pl_excludepages, pl_includepath, pl_browser_fingerprint_i
                  pl_token, pl_session_id, pl_aip, pl_action, pl_mkto_trk,
    prediction_array = np.array([is_json, pl_isprebid, pl_appid, pl_domain, pl_imp, p
                  pl_arr_data_bidder, pl_arr_data_ad_unit_code, pl_arr_data
                  pl_arr_data_media_type, pl_schain, pl_sdk, pl_tags, pl_r
                  pl_wfv, pl_adv, pl_cdl, pl_cet, pl_ct, pl_crt, pl_csd, p
                  pl_session, pl_auctions, pl_referrerUri, pl_referrerHostn
```

```
                                pl_documentWriteIntervention, pl_errorCount, pl_si, pl_p
                                pl_m, pl_fp, pl_timing, pl_av, pl_user, pl_a, pl_dyn, pl_
                                pl_hsi, pl_partner, pl_gdpr, pl_nbPage, pl_o, pl_rf, pl_t
                                pl_fmt, pl_wmode, pl_t, pl_dl, pl_ul, pl_de, pl_dt, pl_sc
                                pl_cid, pl_tid, pl_slc, pl_cd5, pl_z, pl_cmpId, pl_config
                                pl_displayType, pl_bundle, pl_cw, pl_lsw, pl_event, pl_ty
                                pl_tab_id, pl_per_page, pl_accountId, pl_projectId, pl_e
                                pl_operationName, pl_extensions, pl_ippd, pl_sv, pl_tv, p
                                pl_ists, pl_fas, pl_biw, pl_bih, pl_idt, pl_oid, pl_ucis
                                pl_window, pl_address, pl_analytics, pl_experienceCloud,
                                pl_isdeviceaccessgranted, pl_payload, pl_trackingId, pl_
                                pl_is_vpv, pl_session_only, pl_rec_value, pl_user_id, pl_
                                pl_storeType, pl_storeKey, pl_expires, pl_samesite, pl_o
                                pl_plid, pl_url, pl_sid, pl_sts, pl_slts, pl_inc, pl_pvid
                                pl_excludepages, pl_includepath, pl_browser_fingerprint_i
                                pl_token, pl_session_id, pl_aip, pl_action, pl_mkto_trk,
    return prediction_array

def get_array_from_body(y):
    is_json = y["is_json"]
    pl_isprebid = y["pl_isprebid"]
    pl_appid = y["pl_appid"]
    pl_domain = y["pl_domain"]
    pl_imp = y["pl_imp"]
    pl_site = y["pl_site"]
    pl_device = y["pl_device"]
    pl_regs = y["pl_regs"]
    pl_source = y["pl_source"]
    pl_arr_data_bidder = y["pl_arr_data_bidder"]
    pl_arr_data_ad_unit_code = y["pl_arr_data_ad_unit_code"]
    pl_arr_data_devicetype = y["pl_arr_data_devicetype"]
    pl_arr_data_env = y["pl_arr_data_env"]
    pl_arr_data_media_type = y["pl_arr_data_media_type"]
    pl_schain = y["pl_schain"]
    pl_sdk = y["pl_sdk"]
    pl_tags = y["pl_tags"]
    pl_referrer_detection = y["pl_referrer_detection"]
    pl_ext = y["pl_ext"]
    pl_fv = y["pl_fv"]
    pl_wfv = y["pl_wfv"]
    pl_adv = y["pl_adv"]
    pl_cdl = y["pl_cdl"]
    pl_cet = y["pl_cet"]
    pl_ct = y["pl_ct"]
    pl_crt = y["pl_crt"]
    pl_csd = y["pl_csd"]
    pl_trigger = y["pl_trigger"]
    pl_integration = y["pl_integration"]
    pl_wrapper = y["pl_wrapper"]
    pl_session = y["pl_session"]
    pl_auctions = y["pl_auctions"]
    pl_referrerUri = y["pl_referrerUri"]
    pl_referrerHostname = y["pl_referrerHostname"]
    pl_memory = y["pl_memory"]
    pl_resources = y["pl_resources"]
    pl_referrer = y["pl_referrer"]
```

```
pl_documentWriteIntervention = y["pl_documentWriteIntervention"]
pl_errorCount = y["pl_errorCount"]
pl_si = y["pl_si"]
pl_pageloadid = y["pl_pageloadid"]
pl_timingsv2 = y["pl_timingsv2"]
pl_st = y["pl_st"]
pl_results = y["pl_results"]
pl_m = y["pl_m"]
pl_fp = y["pl_fp"]
pl_timing = y["pl_timing"]
pl_av = y["pl_av"]
pl_user = y["pl_user"]
pl_a = y["pl_a"]
pl_dyn = y["pl_dyn"]
pl_csr = y["pl_csr"]
pl_req = y["pl_req"]
pl_hs = y["pl_hs"]
pl_ccg = y["pl_ccg"]
pl_rev = y["pl_rev"]
pl_s = y["pl_s"]
pl_hsi = y["pl_hsi"]
pl_partner = y["pl_partner"]
pl_gdpr = y["pl_gdpr"]
pl_nbPage = y["pl_nbPage"]
pl_o = y["pl_o"]
pl_rf = y["pl_rf"]
pl_top = y["pl_top"]
pl_u = y["pl_u"]
pl_v = y["pl_v"]
pl_provider = y["pl_provider"]
pl_ttd_pid = y["pl_ttd_pid"]
pl_fmt = y["pl_fmt"]
pl_wmode = y["pl_wmode"]
pl_t = y["pl_t"]
pl_dl = y["pl_dl"]
pl_ul = y["pl_ul"]
pl_de = y["pl_de"]
pl_dt = y["pl_dt"]
pl_sd = y["pl_sd"]
pl_sr = y["pl_sr"]
pl_vp = y["pl_vp"]
pl_je = y["pl_je"]
pl_jid = y["pl_jid"]
pl_gjid = y["pl_gjid"]
pl_cid = y["pl_cid"]
pl_tid = y["pl_tid"]
pl_slc = y["pl_slc"]
pl_cd5 = y["pl_cd5"]
pl_z = y["pl_z"]
pl_cmpId = y["pl_cmpId"]
pl_configurationHashCode = y["pl_configurationHashCode"]
pl_operationType = y["pl_operationType"]
pl_sessionId = y["pl_sessionId"]
pl_displayType = y["pl_displayType"]
pl_bundle = y["pl_bundle"]
pl_cw = y["pl_cw"]
```

```
pl_lsw = y["pl_lsw"]
pl_event = y["pl_event"]
pl_type = y["pl_type"]
pl_page_type = y["pl_page_type"]
pl_release_ver = y["pl_release_ver"]
pl_page = y["pl_page"]
pl_tab_id = y["pl_tab_id"]
pl_per_page = y["pl_per_page"]
pl_accountId = y["pl_accountId"]
pl_projectId = y["pl_projectId"]
pl_errorClass = y["pl_errorClass"]
pl_metadata = y["pl_metadata"]
pl_sensor_data = y["pl_sensor_data"]
pl_operationName = y["pl_operationName"]
pl_extensions = y["pl_extensions"]
pl_ippd = y["pl_ippd"]
pl_sv = y["pl_sv"]
pl_tv = y["pl_tv"]
pl_site_id = y["pl_site_id"]
pl_browser = y["pl_browser"]
pl_sfv = y["pl_sfv"]
pl_ecs = y["pl_ecs"]
pl_ists = y["pl_ists"]
pl_fas = y["pl_fas"]
pl_biw = y["pl_biw"]
pl_bih = y["pl_bih"]
pl_idt = y["pl_idt"]
pl_oid = y["pl_oid"]
pl_ucis = y["pl_ucis"]
pl_nach = y["pl_nach"]
pl_scr_x = y["pl_scr_x"]
pl_scr_y = y["pl_scr_y"]
pl_screen = y["pl_screen"]
pl_window = y["pl_window"]
pl_address = y["pl_address"]
pl_analytics = y["pl_analytics"]
pl_experienceCloud = y["pl_experienceCloud"]
pl_execute = y["pl_execute"]
pl_prefetch = y["pl_prefetch"]
pl_tagId = y["pl_tagId"]
pl_isdeviceaccessgranted = y["pl_isdeviceaccessgranted"]
pl_payload = y["pl_payload"]
pl_trackingId = y["pl_trackingId"]
pl_distinctId = y["pl_distinctId"]
pl_campaign = y["pl_campaign"]
pl_r_value = y["pl_r_value"]
pl_is_vpv = y["pl_is_vpv"]
pl_session_only = y["pl_session_only"]
pl_rec_value = y["pl_rec_value"]
pl_user_id = y["pl_user_id"]
pl_res = y["pl_res"]
pl_ref = y["pl_ref"]
pl_rst = y["pl_rst"]
pl_configId = y["pl_configId"]
pl_storeType = y["pl_storeType"]
pl_storeKey = y["pl_storeKey"]
```

```
    pl_expires = y["pl_expires"]
    pl_samesite = y["pl_samesite"]
    pl_originalValue = y["pl_originalValue"]
    pl_currentURI = y["pl_currentURI"]
    pl_rand = y["pl_rand"]
    pl_plid = y["pl_plid"]
    pl_url = y["pl_url"]
    pl_sid = y["pl_sid"]
    pl_sts = y["pl_sts"]
    pl_slts = y["pl_slts"]
    pl_inc = y["pl_inc"]
    pl_pvid = y["pl_pvid"]
    pl_localefilter = y["pl_localefilter"]
    pl_from = y["pl_from"]
    pl_returnedfields = y["pl_returnedfields"]
    pl_excludepages = y["pl_excludepages"]
    pl_includepath = y["pl_includepath"]
    pl_browser_fingerprint_id = y["pl_browser_fingerprint_id"]
    pl_query = y["pl_query"]
    pl_variables = y["pl_variables"]
    pl_code = y["pl_code"]
    pl_vertical = y["pl_vertical"]
    pl_token = y["pl_token"]
    pl_session_id = y["pl_session_id"]
    pl_aip = y["pl_aip"]
    pl_action = y["pl_action"]
    pl_mkto_trk = y["pl_mkto_trk"]
    POST = y["POST"]
    GET = y["GET"]
    result = transformData( is_json, pl_isprebid, pl_appid, pl_domain, pl_imp, pl_sit
                            pl_arr_data_bidder, pl_arr_data_ad_unit_code, pl_arr_data
                            pl_arr_data_media_type, pl_schain, pl_sdk, pl_tags, pl_r
                            pl_wfv, pl_adv, pl_cdl, pl_cet, pl_ct, pl_crt, pl_csd, p
                            pl_session, pl_auctions, pl_referrerUri, pl_referrerHostn
                            pl_documentWriteIntervention, pl_errorCount, pl_si, pl_p
                            pl_m, pl_fp, pl_timing, pl_av, pl_user, pl_a, pl_dyn, pl_
                            pl_hsi, pl_partner, pl_gdpr, pl_nbPage, pl_o, pl_rf, pl_t
                            pl_fmt, pl_wmode, pl_t, pl_dl, pl_ul, pl_de, pl_dt, pl_sc
                            pl_cid, pl_tid, pl_slc, pl_cd5, pl_z, pl_cmpId, pl_config
                            pl_displayType, pl_bundle, pl_cw, pl_lsw, pl_event, pl_ty
                            pl_tab_id, pl_per_page, pl_accountId, pl_projectId, pl_e
                            pl_operationName, pl_extensions, pl_ippd, pl_sv, pl_tv, p
                            pl_ists, pl_fas, pl_biw, pl_bih, pl_idt, pl_oid, pl_ucis
                            pl_window, pl_address, pl_analytics, pl_experienceCloud,
                            pl_isdeviceaccessgranted, pl_payload, pl_trackingId, pl_
                            pl_is_vpv, pl_session_only, pl_rec_value, pl_user_id, pl_
                            pl_storeType, pl_storeKey, pl_expires, pl_samesite, pl_o
                            pl_plid, pl_url, pl_sid, pl_sts, pl_slts, pl_inc, pl_pvid
                            pl_excludepages, pl_includepath, pl_browser_fingerprint_i
                            pl_token, pl_session_id, pl_aip, pl_action, pl_mkto_trk,
    return result

@app.route('/api/predict/lr', methods=['POST'])
def predict_lr():
    data = request.get_json()
    pred_array = get_array_from_body(data)
```

```
    resultlr =  lrmodel.predict_proba(pred_array)

    if resultlr[0][1] > 0.7:
        return jsonify(1)
    else:
        return jsonify(0)


@app.route('/api/predict/dt', methods=['POST'])
def predict_dt():
    data = request.get_json()
    pred_array = get_array_from_body(data)

    resultdt = dtmodel.predict(pred_array)

    return resultdt


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    modelfiledt = './decisionTree.pickle'
    modelfilesvm = './svm.pickle'
    modelfilelr = './logreg.pickle'
    dtmodel = p.load(open(modelfiledt, 'rb'))
    svmmodel = p.load(open(modelfilesvm, 'rb'))
    lrmodel = p.load(open(modelfilelr, 'rb'))
    app.run(debug=True,host='0.0.0.0')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

C Request collection websites

- Facebook
- InspiredTaste
- ModernProper
- skyscanner
- ultimateguitar
- tasteofhome
- airfrance
- antena3
- ele.ro
- experian
- foodnetwork
- insiderintelligence
- investec
- logitech
- medium
- momondo

- pf2easy
- tinuiti
- wizzair