Configuration Manual

MSc Research Project
Data Analytics

Rutuja Dinesh Mehta
Student ID: x20129751

School of Computing
National College of Ireland

Supervisor: Dr Giovani Estrada

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rutuja Dinesh Mehta
Student ID:	x20129751
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr Giovani Estrada
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	XXX
Page Count:	6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rutuja Dinesh Mehta
x20129751

1 Introduction

The research seeks to discover the consumers who make the profit and forecast the income generated by them in order to optimize their marketing tactics and have a recognizable stance in the industry. Gradient Boosting Regressor, Light Boost Gradient Model (LGBM) Regressor, XGBoost Regressor, Random Forest Regressor and the Stacked Regressor fed to the Meta Regressor have been trained after tuning the hyper-parameter for the forecast in the Python environment. This Configuration guidebook offers a step-by-step manual for project development, setup, implementation, and deployment for the research.

2 System Specification And Requirements

Information about the hardware and software configurations utilized and required for the research is included in this section.

2.1 Hardware Specification

The hardware setup needed for the experiment is shown in Table 1. Local machines were used for the research at hand while several investigations were conducted.

Hardware	DELL Vostro 3500
Processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2419 Mhz, 4 Core(s), 8 Logical Processor(s)
RAM	8BG
System Type	64-bit Operating System, x64-based PC
Operating System	Windows 10

Table 1: Hardware Specification

2.2 Software Specifications

On the Windows 10 operating system, the necessary programs and libraries were installed. The Jupyter Notebook 6.3.0 version was used to implement the application. The entire study was done in a Python 3.8.8 environment. The libraries are shown in the table below, together with the usage and version that was taken into account.

Libraries	Usage	Version
Category_encoders	For encoding categorical variables	2.5.0
Dask	For parallel computing	2022.6.1
JSON	A manifest file of library package	0.9.5
lightgbm	An opensource gradient boosting library	3.3.2
Matplotlib	For visualizations	3.3.4
Numpy	Used for working with arrays	1.20.1
Pandas	For data manipulation and analysis	1.2.4
Plotly	For visualizations	5.8.2
Seaborn	For visualizations	0.11.1
Scikitlearn	For Machine Learning and Statistical Modelling	1.1.1
tqdm	For progress meters or progress bars	4.59.0

Table 2: Software Specification

3 Implementation

This section outlines the procedures we used to develop models for identifying the valuable customers and predicting the profit generated by them.

3.1 Data Selection

The dataset was taken from the Kaggle dataset source. The "train.csv" and "test.csv" datasets each of size 1.5GB used in our study are shown in Figure 3. The link to access this data : https://www.kaggle.com/competitions/ga-customer-revenue-prediction/data

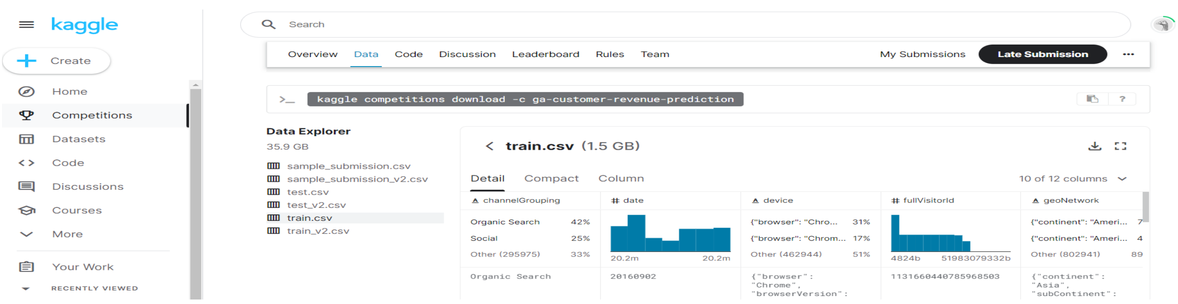

Figure 1: Data Selection

3.2 Loading the Dataset

Both the datasets were loaded in the Jupyter Notebook. The datasets had the class Dask.

```
In [2]: df = dd.read_csv('train.csv')

In [3]: csv_path = "train.csv"

In [4]: df.info()
        <class 'dask.dataframe.core.DataFrame'>
        Columns: 12 entries, channelGrouping to visitStartTime
        dtypes: object(7), int64(4), uint64(1)
```

Figure 2: Loading Dataset

3.3 Data Transformation

The datasets had four columns of the JSON format which were flattened into the normal dataframe using the json_normalize function. This resulted into 55 columns in total.

```
In [6]: def load_df(csv_path, nrows=None):
            JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource'] # we are definig a list of json column names

            df = dd.read_csv(csv_path,
                             converters={
                                 column: json.loads for column in JSON_COLUMNS}, # It will create JSON object for every json column
                             dtype={'fullVisitorId': 'str'})  # we are considering 'fullvisitor id as string'
                             #nrows=nrows)

            for column in tqdm(JSON_COLUMNS):
                column_as_df = json_normalize(df[column]) # json_normalize will return a flatten dataframe of json columns

                column_as_df.columns = ["{0}.{1}".format(column, subcolumn) for subcolumn in column_as_df.columns] # we are taking
                                                                                                                    # column names
                df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True) # we are dropping json column and
                                                                                                     # and merging data frame with parsed

            print("Loaded {0}. Shape: {1}".format(os.path.basename(csv_path), df.shape))

            return df

In [7]: ddf=load_df(csv_path)
          0%|                                                                      | 0/4 [00:00<?, ?it/s]<ipython
        -input-6-87b72699676f>:11: FutureWarning: pandas.io.json.json_normalize is deprecated, use pandas.json_normalize instead
          column_as_df = json_normalize(df[column]) # json_normalize will return a flatten dataframe of json columns
        100%|██████████████████████████████████████████████████████████████████████| 4/4 [05:20<00:00, 80.21s/it]
        Loaded train.csv. Shape: (Delayed('int-c8aa3bcd-cf67-4e45-bd05-0c1b778223df'), 55)
```

Figure 3: Flattening the JSON columns

3.4 Exploratory Data Analysis

Examining the data before making any assumptions is the main goal of exploratory data analysis (EDA). The target variable, revenue-generating customers, device category, geo network category, traffic source category, totals category, and date were all used in the EDA. The "re" module was used to check if the pattern exists amongst the features

```
In [15]: import re
         def columns_extract(category):
             cat_cols = list()
             for i in train_df.columns:
                 a = re.findall(r'^'+category+'.*',i)
                 if a:
                     cat_cols.append(a[0])
                 else:
                     continue
             return cat_cols

In [16]: def category_plots(col):
             a = train_df.loc[:,[col, 'totals.transactionRevenue']]
             a['totals.transactionRevenue'] = a['totals.transactionRevenue'].replace(0.0, np.nan)
             #a['totals.transactionRevenue'] = a['totals.transactionRevenue'].apply(np.expm1)
             cnt_srs = a.groupby(col)['totals.transactionRevenue'].agg(['size','count','mean'])
```

Figure 4: Exploratory Data Analysis

3.5 Data Cleaning and Pre-processing

Figure 7 shows the check for the null values for the categorical variables. The same procedure is carried out for the continuous variable. The treatment for null values is depicted in figure 8 and figure 9.

```
In [54]: total_test = train_df[factor_cols].isnull().sum().sort_values(ascending=False)
         percent = (train_df[factor_cols].isnull().sum()/train_df[factor_cols].isnull().count()).sort_values(ascending=False)*100
         data_to_be_removed = pd.concat([total_test, percent], axis=1,join='outer', keys=['Missing Value Count', 'Percentage of Missing Va
         data_to_be_removed.index.name ='Features'

         sns.barplot(y = data_to_be_removed.index[:15],x = data_to_be_removed['Percentage of Missing Values'][:15],color='blue').set_title
         missing_cols = list(data_to_be_removed[data_to_be_removed['Percentage of Missing Values']>50.0].iloc[:,1].index)
```

Figure 5: Checking the null values

```
In [57]: train_df['totals.bounces'] = train_df['totals.bounces'].fillna(0)
         test_df['totals.bounces'] = test_df['totals.bounces'].fillna(0)

         train_df['totals.newVisits'] = train_df['totals.newVisits'].fillna(0)
         test_df['totals.newVisits'] = test_df['totals.newVisits'].fillna(0)

         train_df['trafficSource.adwordsClickInfo.page'] = train_df['trafficSource.adwordsClickInfo.page'].fillna(0)
         test_df['trafficSource.adwordsClickInfo.page'] = test_df['trafficSource.adwordsClickInfo.page'].fillna(0)

In [58]: for col in ['trafficSource.keyword','trafficSource.isTrueDirect','trafficSource.referralPath','trafficSource.adwordsClickInfo.slo
             train_df[col].fillna('unknown', inplace=True)
             test_df[col].fillna('unknown', inplace=True)
```

Figure 6: Missing values treatment

```
In [71]: encoder=ce.TargetEncoder(cols=categorical_columns, handle_missing='median')
         encoder.fit( X = train_data[categorical_columns],y = train_data['totals.transactionRevenue'] )
         train_data[categorical_columns] = encoder.transform( train_data[categorical_columns], train_data['totals.transactionRevenue'] )

In [72]: filehandler = open("encoder.obj","wb")
         pickle.dump(encoder,filehandler)
         filehandler.close()
```

Figure 7: Target Encoder

3.6 Grouping the train and test data

Each entry in the provided dataset corresponds to a customer visit. Since this data covers the entire year, a single customer may have many entries since they frequented the business on different occasions.Here, we combined all training data such that each visitor has a single row in the dataset, with regard to Visitor ID.

```
In [17]: grouped_train_df = train_data.groupby('fullVisitorId').agg({ 'totals.pageviews':[('total_pageviews_max',lambda x : x.dropna().max
                                                                      ('total_pageviews_min',lambda x : x.dropna().min()),
                                                                      ('total_pageviews_mean',lambda x : x.dropna().mean()),
                                                                      ('total_pageviews_mode',lambda x : x.value_counts().index[0])]

                                             'channelGrouping': [('channelGrouping_max',lambda x : x.dropna().max()),
                                                                 ('channelGrouping_min',lambda x : x.dropna().min()),
                                                                 ('channelGrouping_mode',lambda x : x.value_counts().index[0])],

                                             'visitNumber': [('visitNumber_max',lambda x : x.dropna().max()),
                                                             ('visitNumber_mean',lambda x : x.dropna().mean()),
                                                             ('visitNumber_min',lambda x : x.dropna().min())],

                                             'device.browser':[('device_browser_max',lambda x : x.dropna().max()),
                                                               ('device_browser_min',lambda x : x.dropna().min()),
                                                               ('device_browser_mode',lambda x : x.value_counts().index[0])],

                                            'device.operatingSystem':[('device_operatingSystem_max',lambda x : x.dropna().max()),
                                                                      ('device_operatingSystem_min',lambda x : x.dropna().min()),
                                                                      ('device_operatingSystem_mode',lambda x : x.value_counts().index[0]
```

Figure 8: Data Grouping

3.7 Hyper-parameter Tuning

Before the models were trained, the hyper-parameters were tuned using the random search cross-validation method, and the best-suited parameters were obtained. Hyper-parameter tuning was carried out for Gradient Boosting Regressor, Light Boost Gradient Model (LGBM), XGBoost Regressor, and Random Forest Regressor using the scikit-learn library.

```
In [31]: #from sklearn.model_selection import RandomizedSearchCV
         random_search = RandomizedSearchCV(model_lgbm,GridParams,cv=3, scoring = "neg_root_mean_squared_error")
         %time random_search.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])

         Wall time: 6min 54s
```

Figure 9: Hyper-parameter Tuning

3.8 Modelling

Figure 10 depicts the training of the Light Boost Gradient Model (LGBM) Regressor along with the evaluation after the parameters were tuned.

```
In [13]: lgbm = lgb.LGBMRegressor(subsample= 0.7, reg_lambda =1, reg_alpha= 0.5 ,objective='regression', num_leaves=100, n_estimators = 35

         lgbm.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])

Out[13]: LGBMRegressor(colsample_bytree=0.8, metric='rmse', min_child_samples=1,
                       n_estimators=350, num_leaves=100, objective='regression',
                       reg_alpha=0.5, reg_lambda=1, subsample=0.7)
```

RMSE on Train Data

```
In [316]: np.sqrt(np.sum(np.square(lgbm.predict(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1))-grouped_train_df['log_Rever

Out[316]: 0.5979622712262007
```

Figure 10: Light Boost Gradient Model

The training of the Gradient Boosting Regressor and assessment following parameter tuning are shown in Figure 11.

```
In [8]: gbdt = GradientBoostingRegressor(subsample= 0.7,  n_estimators = 400,learning_rate= 0.1, min_samples_split = 2, min_samples_leaf=

        %time gbdt.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])

        Wall time: 34min 4s

Out[8]: GradientBoostingRegressor(max_depth=5, min_samples_leaf=3, n_estimators=400,
                                  subsample=0.7)
```

RMSE on Train Data

```
In [9]: %time np.sqrt(np.sum(np.square(gbdt.predict(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1))-grouped_train_df['log

        Wall time: 8.56 s

Out[9]: 1.195505263568269
```

Figure 11: Gradient Boosting Regressor

Figure 12 shows the training of the XGBoost Regressor as well as the assessment following the fine-tuning of the parameters.

Figure 13 shows the training of the Random Forest Regressor as well as the assessment after the parameters were adjusted.

```
In [10]: xgbt = XGBRegressor(subsample= 0.8, reg_lambda =0.5, reg_alpha= 1 ,objective='reg:squarederror', n_estimators = 200,learning_rate

         xgbt.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])

Out[10]: XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                      colsample_bynode=1, colsample_bytree=0.6, gamma=0, gpu_id=-1,
                      importance_type='gain', interaction_constraints='',
                      learning_rate=0.1, max_delta_step=0, max_depth=6,
                      min_child_weight=1, missing=nan, monotone_constraints='()',
                      n_estimators=200, n_jobs=4, num_parallel_tree=1, random_state=0,
                      reg_alpha=1, reg_lambda=0.5, scale_pos_weight=1, subsample=0.8,
                      tree_method='exact', validate_parameters=1, verbosity=None)
```

RMSE on Train Data

```
In [11]: np.sqrt(np.sum(np.square(xgbt.predict(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1))-grouped_train_df['log_Rever

Out[11]: 1.2216523001581814
```

Figure 12: XGBoost Regressor

```
In [7]: rfr = RandomForestRegressor( n_estimators = 100, min_samples_split = 6, min_samples_leaf = 2, max_depth = None, n_jobs = -1)

        %time rfr.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])

        Wall time: 4min 41s

Out[7]: RandomForestRegressor(min_samples_leaf=2, min_samples_split=6, n_jobs=-1)
```

Figure 13: Random Forest Regressor

The training of the Stacked Regressor fed to Meta Regressor (LGBM Regressor in this case) has been depicted in figure 14.

```
In [35]: from mlxtend.regressor import StackingRegressor

In [38]: lgbm = lgb.LGBMRegressor(subsample= 0.7, reg_lambda =1, reg_alpha= 0.5 ,objective='regression', num_leaves=100, n_estimators = 35
         gbdt = GradientBoostingRegressor(subsample= 0.7,  n_estimators = 400,learning_rate= 0.1, min_samples_split = 2, min_samples_leaf=
         xgbt = XGBRegressor(subsample= 0.8, reg_lambda =0.5, reg_alpha= 1 ,objective='reg:squarederror', n_estimators = 200,learning_rate
         rfr = RandomForestRegressor( n_estimators = 100, min_samples_split = 6, min_samples_leaf = 2, max_depth = None, n_jobs = -1)
         meta_lgbm = lgb.LGBMRegressor()

In [39]: regressors = [lgbm, gbdt, xgbt, rfr]
         stregr = StackingRegressor(regressors=regressors, meta_regressor=meta_lgbm )#, use_features_in_secondary=True

In [40]: %time stregr.fit(grouped_train_df.drop(['log_Revenue','fullVisitorId'], axis=1), grouped_train_df['log_Revenue'])
         Wall time: 40min 58s
```

Figure 14: Stacked Regressor fed to Meta Regressor

References

Nurbakova, D. and Saumet, T. (2020). Deal Closure Prediction based on User's Browsing Behaviour of Sales Content.
Available at : https://ieeexplore.ieee.org/document/9346295

Ohrimuk, E.S., Razmochaeva, N.V., Mikhailov, Y.I. and Bezrukov, A.A. (2020). Study of Supervised Algorithms for Solve the Forecasting Retail Dynamics Problem.
Available at : https://ieeexplore.ieee.org/document/9039112

Parikh, Y. and Abdelfattah, E. (2020). Clustering Algorithms and RFM Analysis Performed on Retail Transactions. 2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON).
Available at : https://ieeexplore.ieee.org/document/9298123