Importing Libraries

In [1]:

```
import pandas as pd
import json
from pandas.io.json import json_normalize
import pickle as pkl
import numpy as np
import matplotlib.pyplot as plt
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objs as go
from plotly import tools
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from wordcloud import WordCloud
from sklearn.preprocessing import OneHotEncoder
import math

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
import lightgbm as lgb
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from prettytable import PrettyTable
```

Pipeline:

Supporting functions

1. To load DataFrame and convert json and make sub columns

In [10]:

```
#https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields/notebook.
json_columns = ['device', 'geoNetwork','totals', 'trafficSource']  #these 4 columns are of
def load_dataframe(filename):
    df = pd.read_csv(filename, converters={column: json.loads for column in json_columns},
                     dtype={'fullVisitorId': 'str'}) #loading json columns and specifying t

    for column in json_columns:
        column_as_df = json_normalize(df[column])  #normalizing json columns
        column_as_df.columns = [f"{column}_{subcolumn}" for subcolumn in column_as_df.colum
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    return df
```

2. Adding date features

In [14]:

```
#https://docs.python.org/3/library/datetime.html
#adding datetime column in data
def add_date_features(df):
    df['date'] = df['date'].astype(str)
    df["date"] = df["date"].apply(lambda x : x[:4] + "-" + x[4:6] + "-" + x[6:])
    df["date"] = pd.to_datetime(df["date"])

    df["month"]   = df['date'].dt.month  #getting month
    df["day"]     = df['date'].dt.day   #getting day
    df["weekday"] = df['date'].dt.weekday   #getting date
    return df
```

3. To normalize the column values

In [17]:

```
def normalize_numerical_columns(dataframe, isTrainDataset = True):
    dataframe["totals_hits"] = dataframe["totals_hits"].astype(float)
    #dataframe["totals.hits"] = (dataframe["totals.hits"] - min(dataframe["totals.hits"]))

    dataframe["totals_pageviews"] = dataframe["totals_pageviews"].astype(float)
    #dataframe["totals.pageviews"] = (dataframe["totals.pageviews"] - min(dataframe["totals

    dataframe["totals_bounces"] = dataframe["totals_bounces"].astype(float)
    dataframe["totals_newVisits"] = dataframe["totals_newVisits"].astype(float)


    if isTrainDataset:
        dataframe["totals_transactionRevenue"] = dataframe["totals_transactionRevenue"].fil
    return dataframe
```

You just need to paas the raw data (.csv file) in th get prediction class and prediction results will get saved in your local disk

1. Final_fun_1 will take .csv file as input and return the predictions, and saves the predicted values and its respected id in .csv file

In [18]:

```
def final_fun_1(file,model,save_in):
    filename = file


    test_data = load_dataframe(filename)

    test_data = add_date_features(test_data)

    test_data["totals_hits"] = test_data["totals_hits"].astype(float)
    test_data["totals_pageviews"] = test_data["totals_pageviews"].astype(float)
    test_data["totals_bounces"] = test_data["totals_bounces"].astype(float)
    test_data["totals_newVisits"] = test_data["totals_newVisits"].astype(float)



    constant_columns = [column for column in test_data.columns if test_data[column].nunique

    #dropping constant columns
    test_data = test_data.drop(columns=constant_columns)

    #Sorting by date to perform time based slicing
    test_data = test_data.sort_values(by='date',ascending=True)

    submission = pd.DataFrame()
    submission["fullVisitorId"] = test_data["fullVisitorId"]
    submission['predictedLogRevenue'] = np.nan
    ## non relevant columns
    non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitS

    ## Droping non relevant columns
    test = test_data.drop(columns=non_relevant)

    categorical_features_test = (test.select_dtypes(include=[np.object]))
    categorical_columns = [column for column in test.columns if not column.startswith('tota
    categorical_columns = [column for column in categorical_features_test if column not in

    for column in categorical_columns:

        le = LabelEncoder()

        test_values = list(test[column].values.astype(str))

        le.fit(test_values)

        test[column] = le.transform(test_values)

    test['device_isMobile'] = le.transform(test_values)



    test = normalize_numerical_columns(test,isTrainDataset=False)

    test = test.fillna(0).astype('float32')

    test['mean_pageViews_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['to
    test['mean_hits_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['totals_

    test = test.drop('channelGrouping',axis=1)
```

```
    # load the model from disk
    loaded_model = pkl.load(open(model, 'rb'))

    test.columns = loaded_model.feature_names_
    submission['predictedLogRevenue'] = np.log(loaded_model.predict(test.values))
    submission.to_csv(save_in)
```

In [19]:

```
final_fun_1('test.csv','final_model.pkl','predictions.csv')
```

```
[21:24:18] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/src/obje
ctive/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:sq
uarederror.

C:\Users\vansh\Anaconda3\envs\nenv\lib\site-packages\ipykernel_launcher.py:6
5: RuntimeWarning:

invalid value encountered in log
```

2. final_fun_2 will take .csv file as input and make prediction, compares the predicted result with ground truth and given "root mean square error" and plots feature importance

In [22]:

```
def final_fun_2(file,model):
    filename = file


    test_data = load_dataframe(filename)

    test_data = add_date_features(test_data)

    test_data["totals_hits"] = test_data["totals_hits"].astype(float)
    test_data["totals_pageviews"] = test_data["totals_pageviews"].astype(float)
    test_data["totals_bounces"] = test_data["totals_bounces"].astype(float)
    test_data["totals_newVisits"] = test_data["totals_newVisits"].astype(float)



    constant_columns = [column for column in test_data.columns if test_data[column].nunique

    #dropping constant columns
    test_data = test_data.drop(columns=constant_columns)

    #Sorting by date to perform time based slicing
    test_data = test_data.sort_values(by='date',ascending=True)


    ## non relevant columns
    non_relevant = ["visitNumber", "date", "fullVisitorId", "sessionId", "visitId", "visitS

    ## Droping non relevant columns
    test = test_data.drop(columns=non_relevant)

    categorical_features_test = (test.select_dtypes(include=[np.object]))
    categorical_columns = [column for column in test.columns if not column.startswith('tota
    categorical_columns = [column for column in categorical_features_test if column not in

    for column in categorical_columns:

        le = LabelEncoder()

        test_values = list(test[column].values.astype(str))

        le.fit(test_values)

        test[column] = le.transform(test_values)

    test['device_isMobile'] = le.transform(test_values)



    test = normalize_numerical_columns(test,isTrainDataset=False)

    test = test.fillna(0).astype('float32')

    test['mean_pageViews_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['to
    test['mean_hits_per_networkDomain'] = test.groupby('geoNetwork_networkDomain')['totals_

    test = test.drop('channelGrouping',axis=1)


    # load the model from disk
```

```
    loaded_model = pkl.load(open(model, 'rb'))

    test.columns = loaded_model.feature_names_

    predictions = np.log(loaded_model.predict(test.values))

    mse =mean_squared_error(truth_value, predictions)
    rmse = math.sqrt(mse)
    print("Root Mean Squared Error:", rmse)

    print("-"*70)

    importances = loaded_model.feature_importances_

    indices = np.argsort(importances)[::-1]

    # Rearrange feature names so they match the sorted feature importances
    names = [test.columns[i] for i in indices]

    # Create plot
    plt.figure()

    # Create plot title
    plt.title("Feature Importance")

    # Add bars
    plt.bar(range(30), importances[indices])

    # Add feature names as x-axis labels
    plt.xticks(range(30),names, rotation=90)

    # Show plot
    plt.show()
```

In [ ]:

```
final_fun_2('test.csv','final_model.pkl')
```

Note: Since in test.csv we are not given "ground truth"

---Done