In [ ]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import lightgbm as lgb
from lightgbm.sklearn import LGBMRegressor
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import warnings
warnings.filterwarnings('ignore')
from pandas.io.json import json_normalize
from sklearn import metrics
import json
import feather
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('fivethirtyeight')
from sklearn.model_selection import KFold,StratifiedKFold

from sklearn import preprocessing
import seaborn as sns
from sklearn.model_selection import train_test_split
import os
print(os.listdir("../input"))
```

In [ ]:
```
#Developed a function to flatten columns having JSON objects.
#Added new columns having key values of JSON Objects as columns
import pandas as pd
import os
from pandas.io.json import json_normalize
import json
def load_df(csv_path='../input/ga-customer-revenue-prediction/train.csv'
, nrows=None):
    COLUMNS_HAVING_JSON = ['device', 'geoNetwork', 'totals', 'trafficSou
rce']
    df_json_columns = pd.read_csv(csv_path,
                      converters={column: json.loads for column in COLUMN
S_HAVING_JSON},
                      dtype={'fullVisitorId': 'str'},
                      nrows=nrows)
    for column in COLUMNS_HAVING_JSON:
        df_column = json_normalize(df_json_columns[column])
        df_column.columns = [f"{column}_{key}" for key in df_column.colu
mns]
        df_json_columns = df_json_columns.drop(column, axis=1)
        df_json_columns = df_json_columns.merge(df_column, right_index=T
rue, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df_json_column
s.shape}")
    return df_json_columns
```

In [ ]:
```
training_df = load_df("../input/ga-customer-revenue-prediction/train.cs
v")
test_df = load_df("../input/ga-customer-revenue-prediction/test.csv")
```

In [ ]:
```
training_df.to_feather('train_data.feather')
test_df.to_feather('test_data.feather')
print("Loaded data in feather file")
```

In [ ]:
```
df = pd.read_feather('train_data.feather')
test_df = pd.read_feather('test_data.feather')
print("Loaded from feather file")
```

In [ ]:
```
#Removed columns having constant and null values in test and training da
ta
constant_valued_columns = [c for c in df.columns if df[c].nunique(dropna
=False)==1]
non_relevant = ["visitNumber", "date", "fullVisitorId",
                "sessionId", "visitId", "visitStartTime"]
df = df.drop(constant_valued_columns, axis=1)
test_df = test_df.drop(constant_valued_columns, axis=1)
## non relevant columns
df=df.drop(['trafficSource_campaignCode'],axis=1)
```

In [ ]:
```
def convert_totals_columns_to_float(df,isTrain=True):
    numeric_cols_to_transform = ['totals_hits'
                ,'totals_pageviews'
                ,'totals_newVisits'
                ,'totals_bounces']
    if isTrain == True:
        df[numeric_cols_to_transform] = df[numeric_cols_to_transform].fi
llna(0)
        df['totals_transactionRevenue'] = df['totals_transactionRevenue'
].fillna(0)
    else:
        df[numeric_cols_to_transform] = df[numeric_cols_to_transform].fi
llna(0)
    for col in numeric_cols_to_transform:
        df[col] = df[col].astype('float32')
    if isTrain == True:
         df['totals_transactionRevenue'] = df['totals_transactionRevenu
e'].astype('float32')
    return df
```

In [ ]:
```
df['totals_transactionRevenue'] = df['totals_transactionRevenue'].astype
('float32')
df=df.drop(['trafficSource_campaignCode'],axis=1)
```

In [ ]:
```
#Plotting mean transaction revenue by continent
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_continent')['totals_transactionRevenue'
].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]
df_plot.plot(kind='bar',colors=my_colors,grid='false')
plt.title('Mean revenue by continent')
plt.ylabel('Mean Revenue')
```

In [ ]:
```
#Plotting mean transaction revenue by continent
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_subContinent')['totals_transactionReven
ue'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]
df_plot.plot(kind='bar',grid='false')
plt.title('Mean revenue by subContinent');
plt.ylabel('Mean Revenue')
```

In [ ]:
```
#Plotting total count transaction revenue, non-zero count of transaction
 revenue
#and mean of transaction revenue Vs Continent
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_continent')['totals_transactionRevenue'
].agg(['size','count'])
df_plot.columns = ["Size","Count"]
df_plot_count = df_plot.groupby('geoNetwork_continent')['Count'].mean().
sort_index()
df_plot_count.plot(kind='bar',grid='false')
plt.title('Total Non-Zero Count of Transaction_Revenue Vs Continent');
plt.ylabel('Count')
df.dtypes
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_subContinent')['totals_transactionReven
ue'].count()
df_plot.plot(kind='bar',grid='false')
plt.title('Total Non-Zero Count of Transaction Revenue Vs SubContinent')
plt.ylabel('Count')
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_continent')['totals_transactionRevenue'
].size()
df_plot.plot(kind='bar',grid='false')
plt.title('Size of Transaction Revenue Vs Continent')
plt.ylabel('Count')
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_subContinent')['totals_transactionReven
ue'].size()
df_plot.plot(kind='bar',grid='false')
plt.title('Total Count of Transaction Revenue Vs SubContinent')
plt.ylabel('Count')
```

In [ ]:
```
#Plotting total visits per continent
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_continent')['fullVisitorId'].count()
df_plot.plot(kind='bar',grid='false')
plt.title('Visits Per Continent')
plt.ylabel('Visits')
```

In [ ]:
```
#Plotting total visits per sub-continent
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('geoNetwork_subContinent')['fullVisitorId'].count()
df_plot.plot(kind='bar',grid='false')
plt.title('Visits Per subContinent')
plt.ylabel('Visits')
```

In [ ]:
```
#Plotting total count transaction revenue, non-zero count of
#transaction revenue and mean of transaction revenue Vs operatingSystem
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('device_operatingSystem')['totals_transactionRevenu
e']
.agg(['size','count','mean'])
df_plot.head()
df_plot.columns = ["Size", "Count", "Mean"]

df_plot_count = df_plot.groupby('device_operatingSystem')['Size'].mean()
df_plot_count.plot(kind='bar',grid='false')
plt.title('Total Count of Transaction Revenue Vs Operating System')
plt.ylabel('Count')
```

In [ ]:
```
df_plot_non_revenue_count = df_plot.groupby('device_operatingSystem')['C
ount'].mean()
df_plot_non_revenue_count.plot(kind='bar',grid='false')
plt.title('Total Non-Zero of Transaction Revenue Count Vs Operating Syst
em')
plt.ylabel('Count')
```

In [ ]:
```
df_plot_mean = df_plot.groupby('device_operatingSystem')['Mean'].mean()
df_plot_mean.plot(kind='bar',grid='false')
plt.title('Mean of Transaction Revenue Count Vs Operating System')
plt.ylabel('Count')
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('device_deviceCategory')['totals_transactionRevenu
e'].agg(['size','count'])
df_plot.head()
df_plot.columns = ["Size", "Count"]

df_plot_count = df_plot.groupby('device_deviceCategory')['Size'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]

df_plot_count.plot(kind='bar',grid='false',colors=my_colors)
plt.title('Total Count of Transaction Revenue Vs Device Category')
plt.ylabel('Count')
```

In [ ]:
```
df_plot_non_zero_count = df_plot.groupby('device_deviceCategory')['Coun
t'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]

df_plot_non_zero_count.plot(kind='bar',grid='false',colors=my_colors)
plt.title('Total Non-Zero Count of Transaction Revenue Vs Device Categor
y')
plt.ylabel('Count')
```

In [ ]:
```
#Plotting total count transaction revenue, non-zero count of
#transaction revenue and mean of transaction revenue Vs Device Browser

import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('device_browser')['totals_transactionRevenue'].agg
(['size','count','mean'])
df_plot.head()
df_plot.columns = ["Size", "Count","Mean"]
```

In [ ]:
```
df_plot_count = df_plot.groupby('device_browser')['Size'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]
df_plot = df_plot.sort_values(["Size"], ascending=False)
df_plot_count.plot(kind='bar',grid='false',figsize=(20,5))
plt.title('Total Count of Transaction Revenue Vs Device Browser')
plt.ylabel('Count')
```

In [ ]:
```
df_plot_count = df_plot.groupby('device_browser')['Count'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]

df_plot_count.plot(kind='bar',grid='false',figsize=(20,5))
plt.title('Total Non Transaction Revenue Vs Device Browser')
plt.ylabel('Count')
```

In [ ]:
```
df_plot_count = df_plot.groupby('device_browser')['Mean'].mean()
my_colors = [(x/10.0, x/20.0, 0.75) for x in range(len(df_plot))]

df_plot_count.plot(kind='bar',grid='false',figsize=(20,5))
plt.title('Mean of Transaction Revenue Vs Device Browser')
plt.ylabel('Mean')
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('totals_hits')['totals_transactionRevenue'].agg(['s
ize','count'])
df_plot.head()
df_plot.columns = ["Size", "Count"]
df_plot = df_plot.sort_values(["Size"], ascending=False)
df_plot1 = df_plot['Size'].head(60)
df_plot1.plot(kind='barh',grid='false',figsize=(9, 9),yticks=range(0,60
))
plt.title('Hits Vs Total Count of Transaction Revenue')
plt.ylabel('Hits')
plt.xlabel('Total Count')
```

In [ ]:
```
df_plot2 = df_plot['Count'].head(60)
df_plot2.plot(kind='barh',grid='false',figsize=(11, 11),yticks=range(0,6
0))
plt.title('Hits Vs Non-Zero Revenue Transactions')
plt.ylabel('Hits')
plt.xlabel('Total Count')
```

In [ ]:
```
import matplotlib.pyplot as plt
x = [{i:np.random.randint(1,5)} for i in range(10)]
df_plot = df.groupby('totals_pageviews')['totals_transactionRevenue'].ag
g(['size','count','mean'])
df_plot.head()
df_plot.columns = ["Size", "Count","Mean"]
df_plot = df_plot.sort_values(["Size"], ascending=False)
df_plot_pv_count = df_plot['Size'].head(60)
df_plot_pv_count.plot(kind='barh',grid='false',figsize=(9, 9))
plt.title('PageViews Vs Total Count of Transactions')
plt.ylabel('PageViews')
plt.xlabel('Count')
```

In [ ]:
```
df_plot_pv_count = df_plot['Count'].head(60)
df_plot_pv_count.plot(kind='barh',grid='false',figsize=(9, 9))
plt.title('PageViews Total Vs Non-Zero Revenue Transactions')
plt.ylabel('PageViews')
plt.xlabel('Count')
```

In [ ]:
```
#Converting object values to categories
from sklearn.preprocessing import LabelEncoder

categorical_columns = [c for c in df.columns if not c.startswith("total")]
categorical_columns = [c for c in categorical_columns if c not in constant_valued_columns + non_relevant]
for c in categorical_columns:

    le = LabelEncoder()
    train_vals = list(df[c].values.astype(str))
    test_vals = list(test_df[c].values.astype(str))

    le.fit(train_vals + test_vals)

    df[c] = le.transform(train_vals)
    test_df[c] = le.transform(test_vals)
```

In [ ]:
```
df = convert_totals_columns_to_float(df)
test_df = convert_totals_columns_to_float(test_df,False)
```

In [ ]:
```
def normalize_numerical_columns(df, isTrain = True):
    df["totals_hits"] = df["totals_hits"].astype(float)
    df["totals_hits"] = (df["totals_hits"] - min(df["totals_hits"])) /
    (max(df["totals_hits"]) - min(df["totals_hits"]))

    df["totals_pageviews"] = df["totals_pageviews"].astype(float)
    df["totals_pageviews"] = (df["totals_pageviews"] - min(df["totals_pageviews"])) /
    (max(df["totals_pageviews"]) - min(df["totals_pageviews"]))

    if isTrain:
        df["totals_transactionRevenue"] = df["totals_transactionRevenue"].fillna(0.0)
    return df
```

In [ ]:
```
df = normalize_numerical_columns(df)
test_df = normalize_numerical_columns(test_df, isTrain = False)
```

In [ ]:
```
features = [c for c in df.columns if c not in constant_valued_columns + non_relevant]
features.remove("totals_transactionRevenue")
df_new["totals_transactionRevenue"] = np.log1p(df["totals_transactionRevenue"].astype(float))
```

In [ ]:
```
#To Compute buying probability
df_count_nonzero_tr = df.groupby('fullVisitorId')['totals_transactionRev
enue'].agg(['count'])
df_count_nonzero_tr.columns = ["Count"]
totalNonZeroTransactionCount = df_count_nonzero_tr["Count"].sum()
df_count_nonzero_tr['buying_probability'] = df_count_nonzero_tr["Count"]
/totalNonZeroTransactionCount
df_count_nonzero_tr=df_count_nonzero_tr.sort_values(by=['buying_probabil
ity'],ascending=False)
print(df_count_nonzero_tr.head(10))

probSum = df_count_nonzero_tr['buying_probability'].sum()
print(probSum)
```

In [ ]:
```
train_x, valid_x, train_y, valid_y = train_test_split(df[features],
                                                      df_new["totals_tra
nsactionRevenue"],
                                                      test_size=0.20, ra
ndom_state=42)
```

In [ ]:
```
import lightgbm as lgb

lgb_params = {"objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 200,
        "min_child_samples" : 100,
        "learning_rate" : 0.05,
        "boosting_type" : "gbdt",
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.5,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1}

lgb_train = lgb.Dataset(train_x, label=train_y)
lgb_val = lgb.Dataset(valid_x, label=valid_y)
model = lgb.train(lgb_params, lgb_train, 700, valid_sets=[lgb_val], earl
y_stopping_rounds=150)
```

In [ ]:
```
unimportant_features = ["visitNumber", "date", "fullVisitorId", "session
Id", "visitId", "visitStartTime"]
features = [c for c in df.columns if c not in constant_valued_columns +
unimportant_features]
print (features)
```

In [ ]:
```
features = [c for c in df.columns if c not in constant_valued_columns +
unimportant_features]
features.remove("totals_transactionRevenue")
preds = model.predict(test_df[features], num_iteration=model.best_iterat
ion)
test_df["PredictedLogRevenue"] = np.expm1(preds)
result = test_df.groupby("fullVisitorId").agg({"PredictedLogRevenue" :
"sum"}).reset_index()
result["PredictedLogRevenue"] = np.log1p(result["PredictedLogRevenue"])
result["PredictedLogRevenue"] =  result["PredictedLogRevenue"].apply(lam
bda x : 0.0 if x < 0 else x)
result["PredictedLogRevenue"] = result["PredictedLogRevenue"].fillna(0)
result.to_csv("baseline.csv", index=False)
print(result.isnull().values.any())
result.head()
```

In [ ]:
```
def calculate_rmse(df_new):

    temp_df = pd.DataFrame()
    temp_df["totals_transactionRevenue"] = np.log1p(df_new["totals_trans
actionRevenue"].astype(float))
    train_x, valid_x, train_y, valid_y = train_test_split(df_new[feature
s],
                                                          temp_df["total
s_transactionRevenue"],
                                                          test_size=0.2,
                                                          random_state=2
0)


    lgb_params = {"objective" : "regression", "metric" : "rmse",
                  "num_leaves" : 50, "learning_rate" : 0.02,
                  "bagging_fraction" : 0.75, "feature_fraction" : 0.8,
"bagging_frequency" : 9}


    lgb_train = lgb.Dataset(train_x, label=train_y)
    lgb_val = lgb.Dataset(valid_x, label=valid_y)
    model = lgb.train(lgb_params, lgb_train, 400, valid_sets=[lgb_val],
                      early_stopping_rounds=100, verbose_eval=200)

    prediction = model.predict(valid_x, num_iteration=model.best_iterati
on)
    rmse = np.sqrt(metrics.mean_squared_error(prediction, valid_y))
    return rmse
```

In [ ]:
```
#To compute p-value
testing_cols = ['trafficSource_campaign','totals_pageviews','trafficSour
ce_adContent']
for cols in testing_cols:
    df_new = df.copy()
    count = 0
    i=1
    while i < 500: #No of iterations
        df_new[cols] = np.random.permutation(df_new[cols])
        if calculate_rmse(df_new)<bestValue:
            count=count+1
    print(count)
    print("For ",cols," number of times RMSE is lesser than base is : ",
count)
```