CENTER FOR DEVELOPMENT OF ADVANCED COMPUTING

Google Analytics Customer Revenue Prediction

PG-DBDA March 2024

Submitted by:

Mayur Mate.
Ameya Oka.
Ashwin Deshmukh.
Gourav Patil.
Prasad Khokle.
Shubam Mulay.

Project Guide: -

Vishal Khetmalis

Index

1. PROBLEM STATEMENT.

Problem Statement: Google Analytics Customer Revenue Prediction

Objective:

The objective of this project is to develop an automated system that predicts future revenue per customer for the Google Merchandise Store using historical customer data and transaction records. The prediction model will be instrumental in identifying high-value customers and optimizing marketing budget allocation, thereby increasing overall revenue and enhancing customer targeting strategies.

Background:

The Google Merchandise Store, an online retailer selling Google-branded products, collects extensive data on customer interactions, transactions, and demographics via Google Analytics. However, effectively utilizing this data to predict future revenue and identify high-value customers has been a challenge. Accurate predictions will allow the store to focus marketing efforts on segments that are likely to generate the most revenue, improving return on investment (ROI) and customer retention.

Key Goals:

1. **Predict Future Revenue:** Use historical data to forecast the revenue each customer is likely to generate in future periods.
2. **Identify High-Value Customers:** customers based on predicted revenue to prioritize high-value customers for targeted marketing.
3. **Optimize Marketing Budget:** Allocate marketing resources more efficiently by focusing on customer segments with the highest predicted return.

Approach:

1. **Data Collection:** Utilize Google Analytics data, including customer demographics, purchase history, session data, and transaction details.
2. **Data Preparation:** Clean and pre-process the data, handling missing values, outliers, and feature engineering to create relevant input features for the model.
3. **Model Development:** Build a predictive model using machine learning algorithms (e.g., linear regression, decision trees, random forests) to estimate future customer revenue.
4. **Model Evaluation:** Validate the model's performance using appropriate metrics such as RMSE and perform cross-validation to ensure robustness.
5. **Implementation:** Integrate the predictive model into an automated system that continuously updates predictions as new data becomes available.
6. **Business Insights:** Provide actionable insights based on the model's predictions, such as identifying which customers to target with marketing campaigns or special offers.

2. Introduction

- Google has published a Merchandise customer dataset (2016-2019), which includes online/referral customer information as well as the number of transactions gathered per customer.

- We have created a predictive model utilizing the Gstore Customer Dataset to anticipate total revenue per customer, allowing us to make better use of marketing budget.

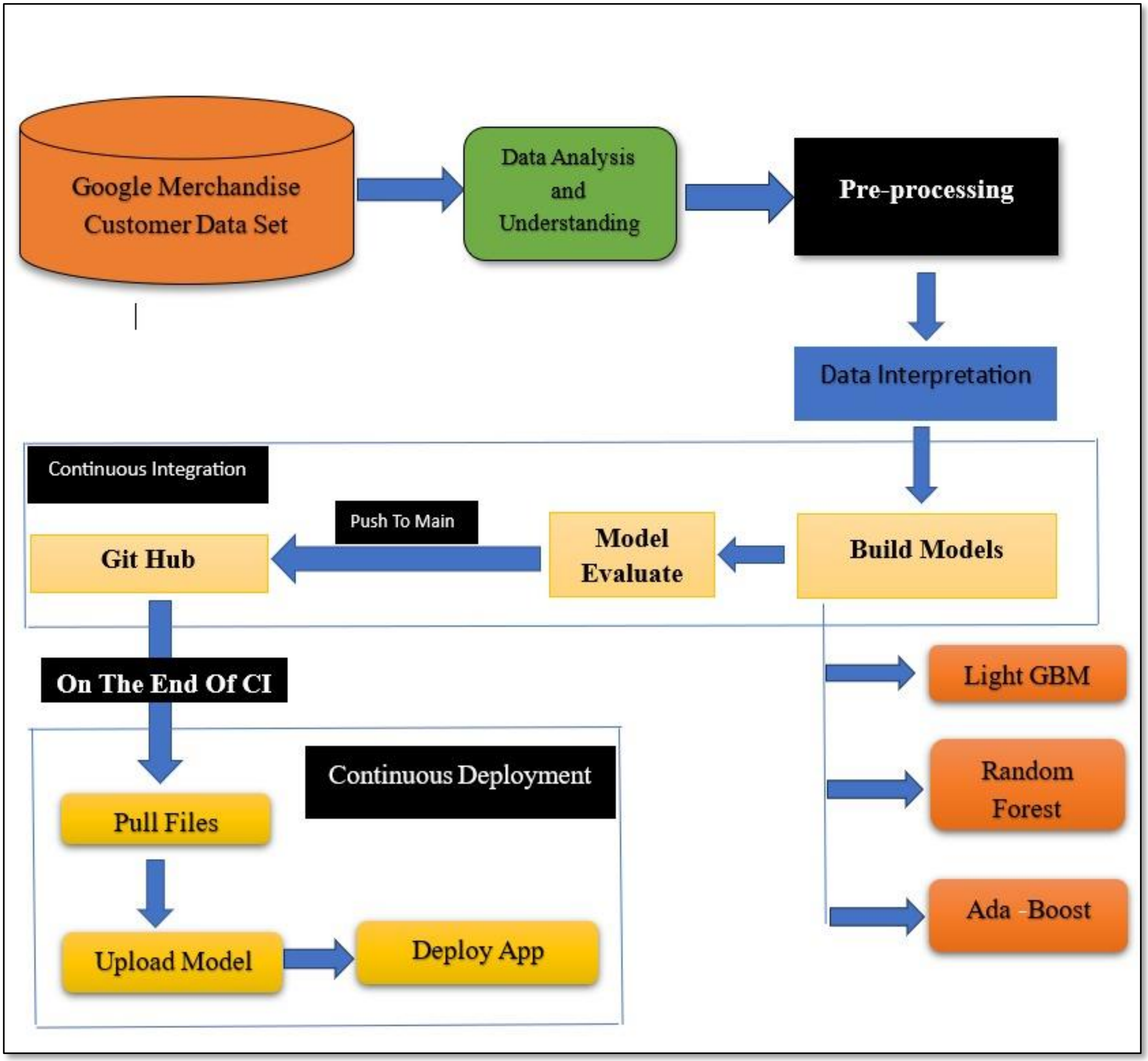
- In this study, we have also interpreted the most influential factors on Total Revenue projection using several models.

- The 80/20 rule has proven true for many businesses–only a small percentage of customers produce most of the revenue. As such, marketing teams are challenged to make appropriate investments in promotional strategies.

- RStudio, the developer of free and open tools for R and enterprise-ready products for teams to scale and share work, has partnered with Google Cloud and Kaggle to demonstrate the business impact that thorough data analysis can have.

- In this competition, you're challenged to analyse a Google Merchandise Store (also known as GStore, where Google swag is sold) customer dataset to predict revenue per customer. Hopefully, the outcome will be more actionable operational changes and a better use of marketing budgets for those companies who choose to use data analysis on top of GA data.

3.Project Overview



4.Data Understanding

The Gstore customer dataset is available on the below Kaggle website htts://www.kagle.com/c/ta-customer-revenueprediction

- Total 903,653 Observations.
- Total 12 attributes (some are JSON collection set).
- The dataset ranges from 2016 to 2017.
- 2 Dates, 4 numeric identifiers and 7 categorical attributes.
- The 4 categorical attributes ("'device', 'geoNetwork', 'totals', 'trafficSource) are in JSON set and can be divided into multiple attributes for analysis.
- Totals JSON attribute has Transaction Revenue values which will be the Response/Predictive attribute.
- All the description of 12 attributes is on the next slide.

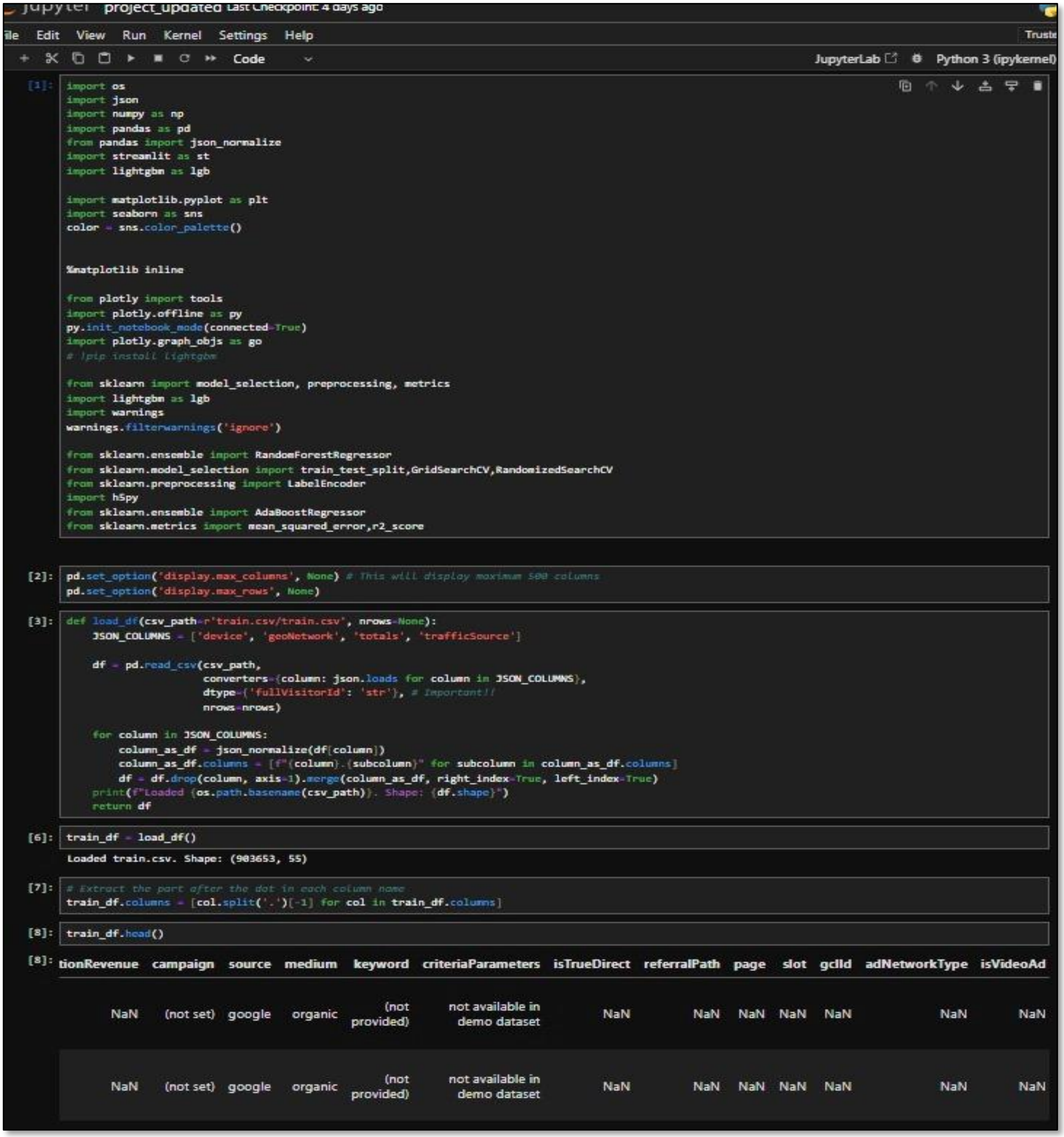
Columns :-

- **channel Grouping :-** T he channel via which the user came to the Store.
- **Date :-**The date on which the user visited the Store.
- **Device :-**The specifications for the device used to access the Store.
- **fullVisitorld** :- A unique identifier for each user of the Google Merchandise Store.
- **geaNetwork** :-This section contains information about the geography of the user.
- **Sessionid** :-A unique identifier for this visit to the store.
- **social Engagement Type** :-Engagement type, either "Socially Engaged" or "Not Socially Engaged".
- **Totals** :-This section contains aggregate values across the session.
- **traffic Source** :-This section contains information about the Traffic Source from which the session originated.
- **Visited** :-An identifier for this session. This is part of the value usually stored as thumb cookie. This is only unique to the fullVisitorld and visited. user. For a completely unique ID, you should use a combination of Full visitor and visit id
- **visit Number** :-The session number for this user. If this is the first session, then this is set to 1.
- **visitStartTime** :- The timestamp (expressed as POSIX time).

5.Data Preprocessing

1. Filtered only those records which have Transaction Revenue in the Total column. This reduced to 11516 observations from 90K.
2. Convert date column from character to Date class
3. Convert visitStartTime to POSIXct
4. Parse the JSON columns (device, geoNetwork, totals, traffic Source) into several columns and drop the old JSON columns
5. Replace various unknown values ['unknown.unknown', '(not set)', 'not available in demo
6. dataset', '(not provided)', '(none)', '<NA>'] with 'NA'
7. Drop the newly parsed JSON columns that have only 1 unique value (1 cardinality)
8. Convert all the newly JSON columns (hits, pageviews, new Visits, transaction Revenue) from character to numeric
9. The transaction Revenue column is multiplied by 106 and so Divide transaction Revenue by 1,00,000

5.1.Using Local Notebook :-



```
import os
import json
import numpy as np
import pandas as pd
from pandas import json_normalize
import streamlit as st
import lightgbm as lgb

import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()


%matplotlib inline

from plotly import tools
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
# !pip install lightgbm

from sklearn import model_selection, preprocessing, metrics
import lightgbm as lgb
import warnings
warnings.filterwarnings('ignore')

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
import h5py
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error,r2_score
```

```
pd.set_option('display.max_columns', None) # This will display maximum 500 columns
pd.set_option('display.max_rows', None)
```

```
def load_df(csv_path=r'train.csv/train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df
```

```
train_df = load_df()
```

Loaded train.csv. Shape: (903653, 55)

```
# Extract the part after the dot in each column name
train_df.columns = [col.split('.')[-1] for col in train_df.columns]
```

```
train_df.head()
```

tionRevenue	campaign	source	medium	keyword	criteriaParameters	isTrueDirect	referralPath	page	slot	gclId	adNetworkType	isVideoAd
NaN	(not set)	google	organic	(not provided)	not available in demo dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN
NaN	(not set)	google	organic	(not provided)	not available in demo dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5.2.Using Azure Data Bricks:-

```
%pip install azure-storage-blob

from azure.storage.blob import BlobServiceClient


from azure.storage.blob import BlobServiceClient
from io import BytesIO
from zipfile import ZipFile

# Define the storage account and container details
storage_account_name = "dbstoragescglksekujqmi"
container_name = "revpreddata"
storage_account_key = "hK8fFKx/KpPyo0urUIllPJTffn/HG707Qssd+AJGPaKPcwKugkEdjJ5tBGXGCUw3lL4/OIhScPFG+AStx97H0A=="

# Set up the BlobServiceClient
blob_service_client = BlobServiceClient(
    account_url=f"https://{storage_account_name}.blob.core.windows.net",
    credential=storage_account_key
)

# Get the container client
container_client = blob_service_client.get_container_client(container_name)

# Read a specific blob into a Pandas DataFrame
blob_name = "train.csv.zip"  # Replace with your blob name
blob_client = container_client.get_blob_client(blob_name)
blob_data = blob_client.download_blob().readall()

# Load the zipped CSV file into a Pandas DataFrame
with ZipFile(BytesIO(blob_data)) as z:
    with z.open(z.namelist()[0]) as f:
        df = pd.read_csv(f)

# Save the DataFrame as a CSV file in Databricks
dbutils.fs.mkdirs("/mnt/revpreddata")
df.to_csv("/dbfs/mnt/revpreddata/train.csv", index=False)
```

```
from pandas import json_normalize
import numpy as np
import json
import os

import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()


%matplotlib inline

from plotly import tools
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
# !pip install lightgbm

from sklearn import model_selection, preprocessing, metrics
import lightgbm as lgb
import warnings
warnings.filterwarnings('ignore')

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.preprocessing import LabelEncoder
import h5py
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error,r2_score
```

```python
def load_df(csv_path=r'/dbfs/mnt/revpreddata/train.csv', nrows=None):
    JSON_COLUMNS = ['device', 'geoNetwork', 'totals', 'trafficSource']

    df = pd.read_csv(csv_path,
                     converters={column: json.loads for column in JSON_COLUMNS},
                     dtype={'fullVisitorId': 'str'}, # Important!!
                     nrows=nrows)

    for column in JSON_COLUMNS:
        column_as_df = json_normalize(df[column])
        column_as_df.columns = [f"{column}.{subcolumn}" for subcolumn in column_as_df.columns]
        df = df.drop(column, axis=1).merge(column_as_df, right_index=True, left_index=True)
    print(f"Loaded {os.path.basename(csv_path)}. Shape: {df.shape}")
    return df
```

```python
train_df = load_df()
```

```
Loaded train.csv. Shape: (903653, 55)
```

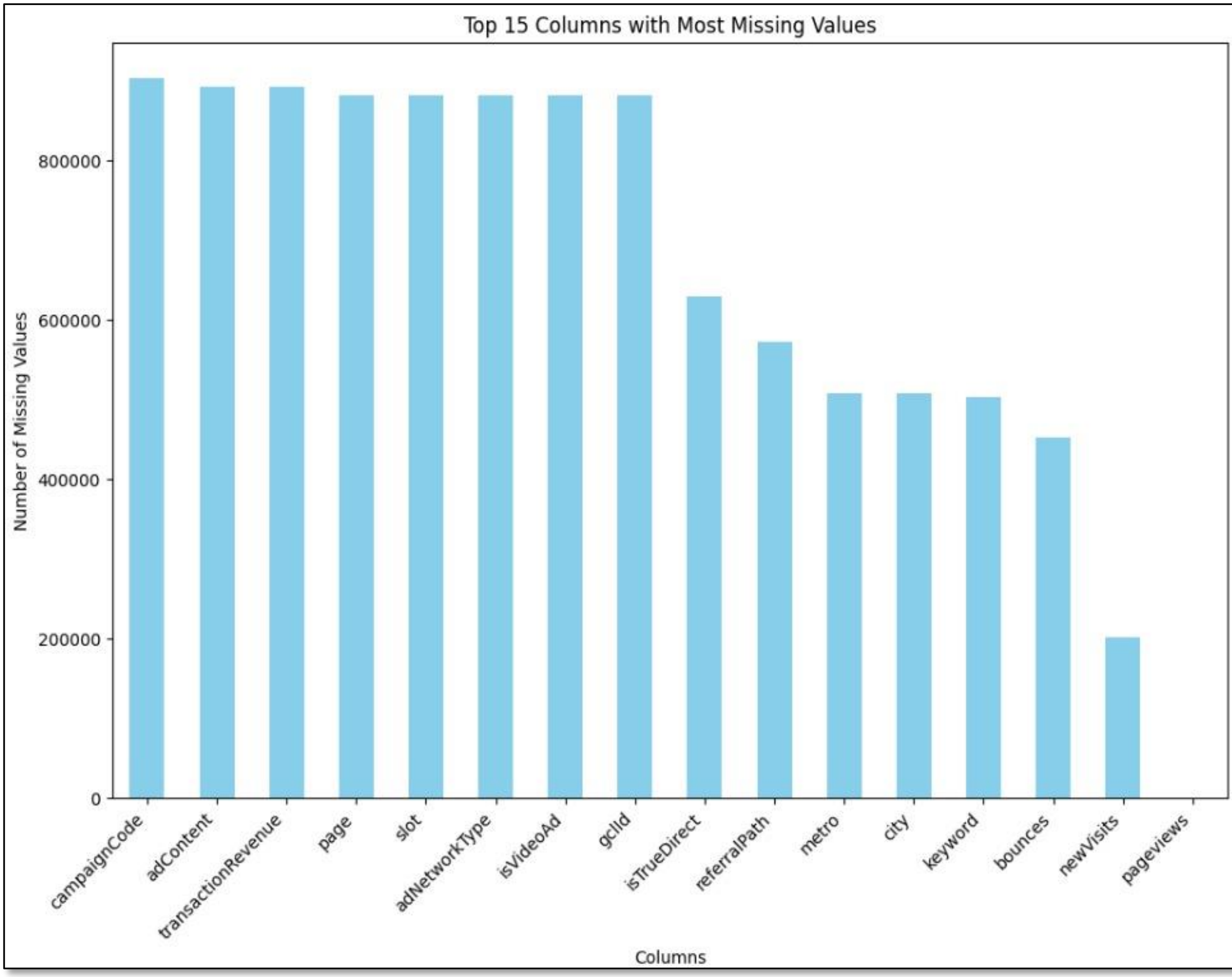
```python
train_df.columns = [col.split('.')[-1] for col in train_df.columns]
```

```python
train_df.head()
```

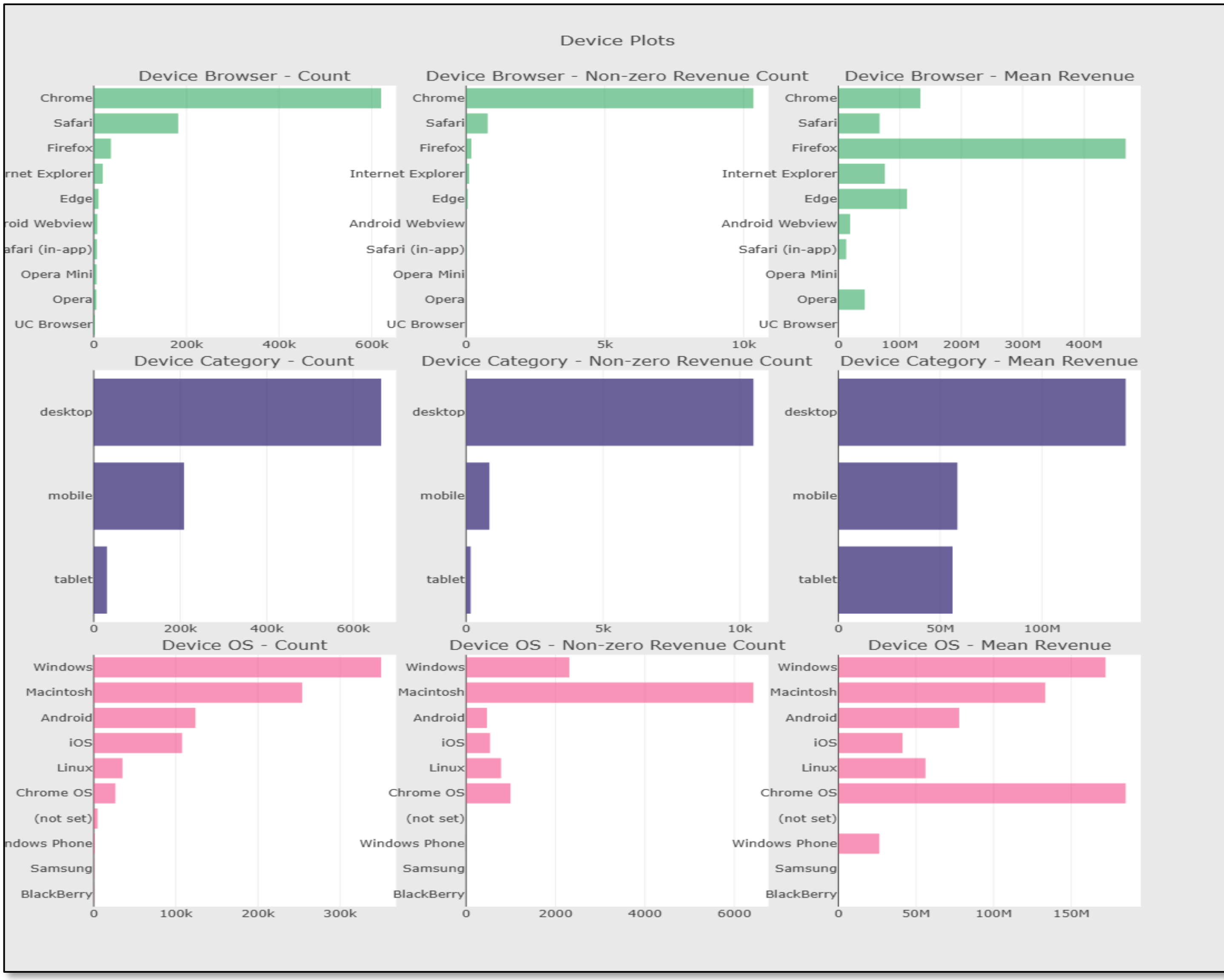
annelGrouping	date	fullVisitorId	sessionId	socialEngagementType	visitId	visitNumber	visitStartTim
Organic Search	20160902	1131660440785968503	1131660440785968503_1472830385	Not Socially Engaged	1472830385	1	147283038
Organic Search	20160902	377306020877927890	377306020877927890_1472880147	Not Socially Engaged	1472880147	1	147288014

6.Exploratory Data Analysis

6.1.Missing Values by Features



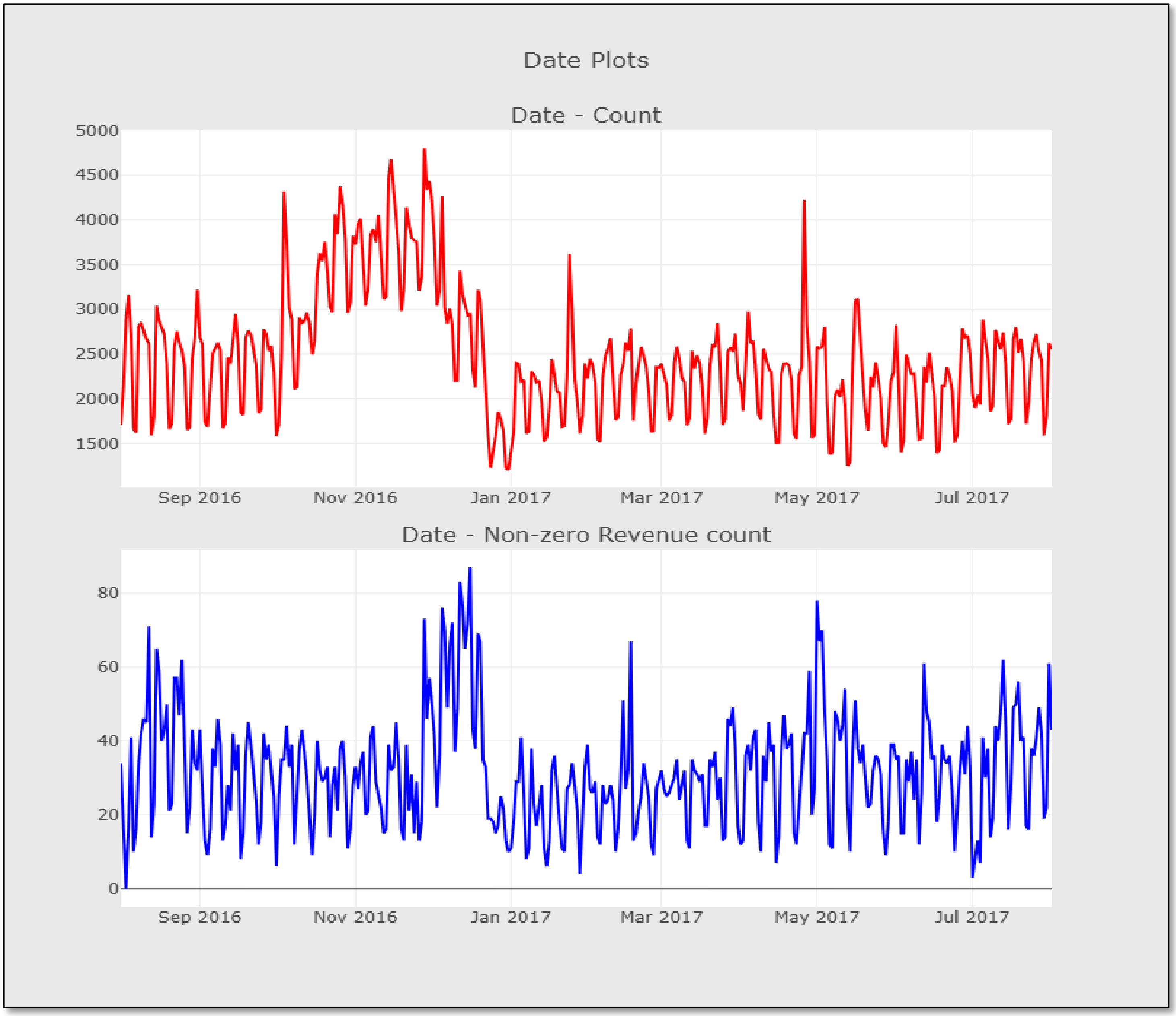
6.2.Device information



Inferences:

- Device browser distribution looks similar on both the count and count of non-zero revenue plots On the device category front, desktop seem to have higher percentage of non-zero revenue counts compared to mobile devices.
- In device operating system, though the number of counts is more from windows, the number of counts where revenue is not zero is more for Macintosh.
- Chrome OS also has higher percentage of non-zero revenue counts
- On the mobile OS side, iOS has more percentage of non-zero revenue counts compared to Android

6.3.Date Plots



Inferences:

- We have data from 1 Aug, 2016 to 31 July, 2017 in our training dataset.
- In Nov 2016, though there is an increase in the count of visitors, there is no increase in non-zero revenue counts during that time period (relative to the mean).

6.4.Geographic Information.

Inferences:

- On the continent plot, we can see that America has both higher number of counts as well as highest number of counts where the revenue is non-zero
- Though Asia and Europe have high number of counts, the number of non-zero revenue counts from these continents are comparatively low.
- We can infer the first two points from the sub-continents plot too.
- If the network domain is "unknown. Unknown" rather than "(not set)", then the number of counts with non-zero revenue tend to be lower

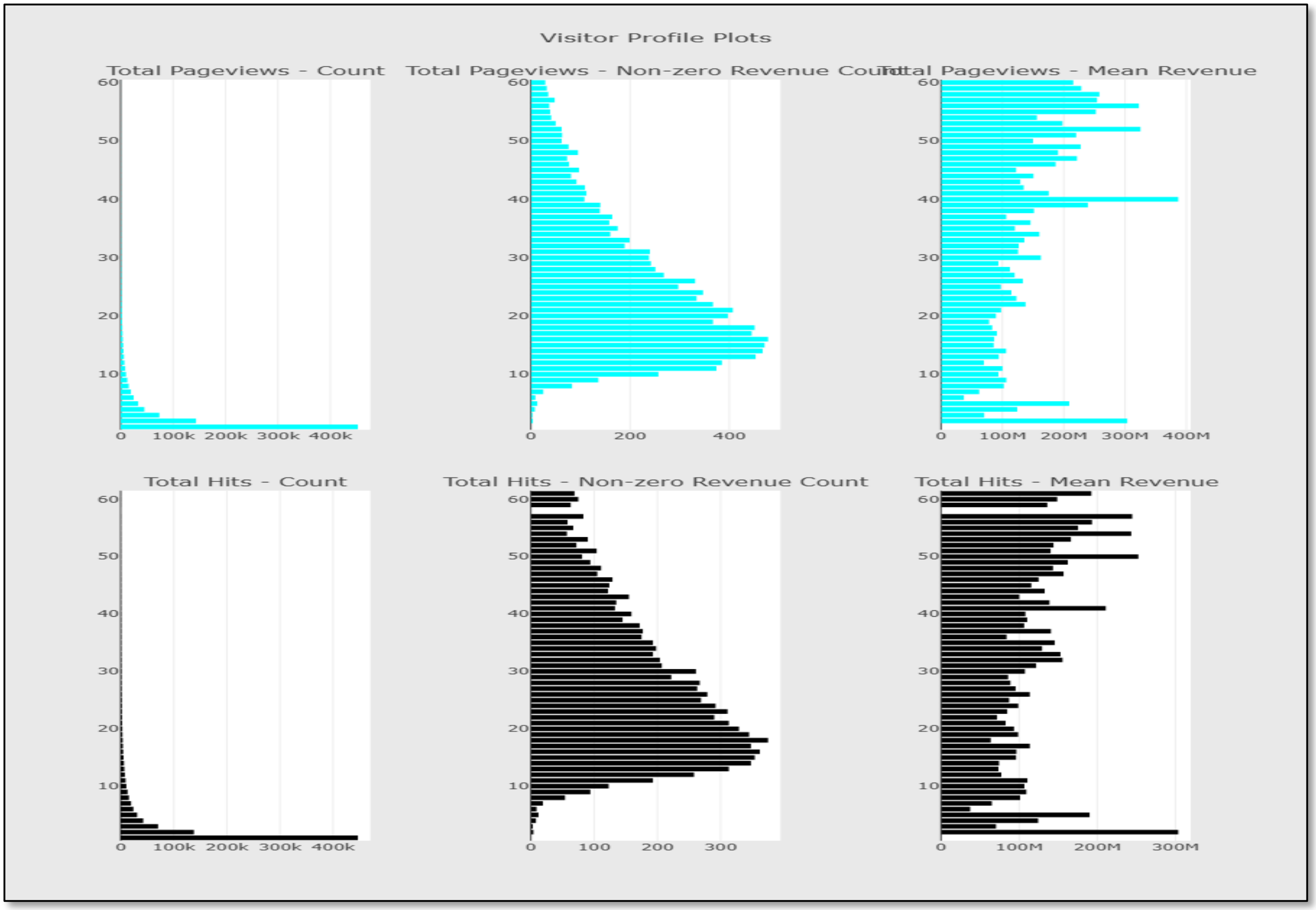
6.5.Traffic Source.

Inferences:

- In the traffic source plot, though YouTube has high number of counts in the dataset, the number of non-zero revenue counts are very less.
- Google plex has a high ratio of non-zero revenue count to total count in the traffic source plot.
- On the traffic source medium, "referral" has a greater number of non-zero revenue count compared to "organic" medium.

6.6.Visitor Profile Plots.

Now let us look at the visitor profile variables like number of pageviews by the visitor, number of hits by the visitor and see how they look.

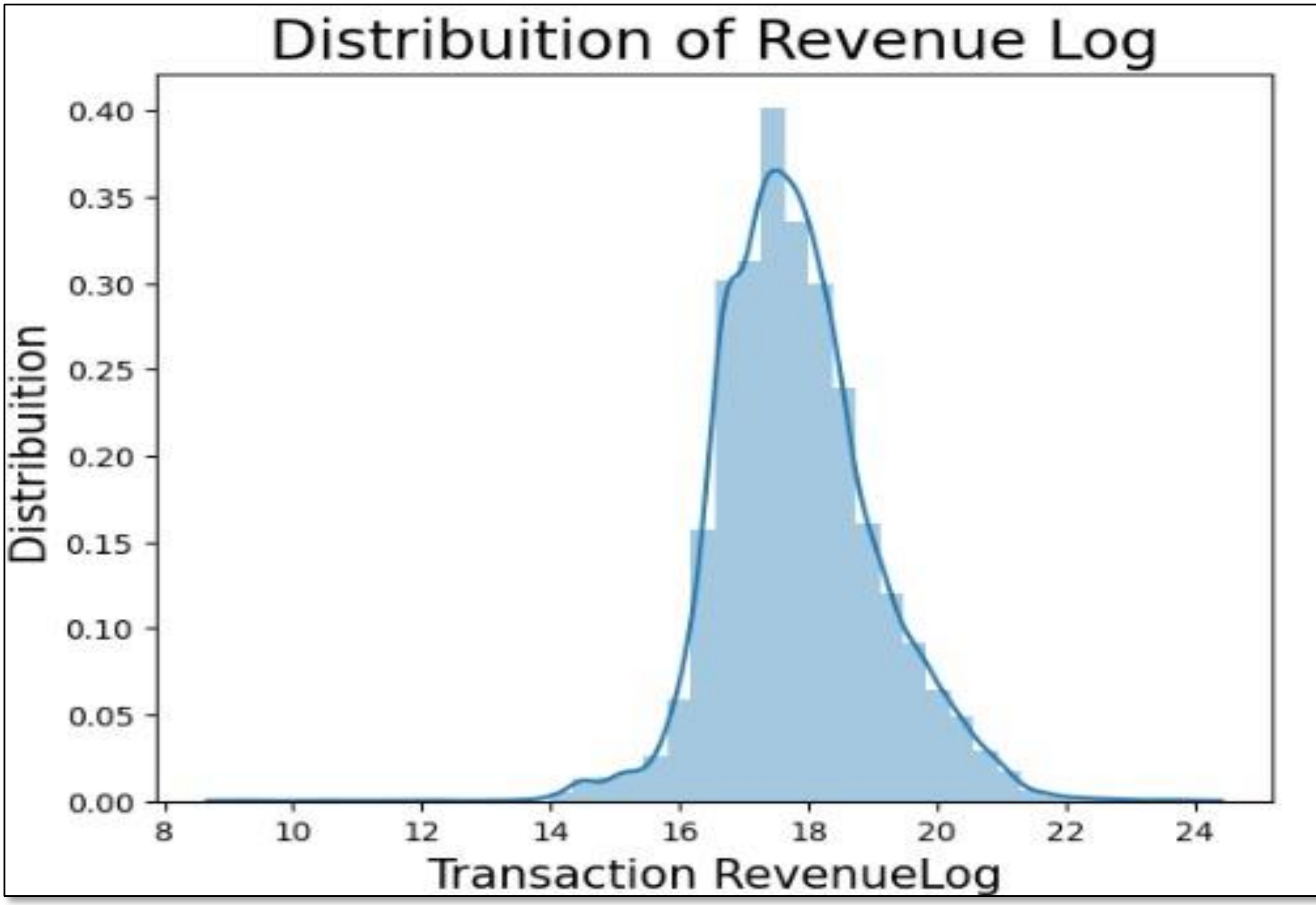


Inferences:

- Both these variables look very predictive
- Count plot shows decreasing nature i.e. we have a very high total count for less number of hits and page views per visitor transaction and the overall count decreases when the number of hits per visitor transaction increases.
- On the other hand, we can clearly see that when the number of hits / pageviews per visitor transaction increases, we see that there is a high number of non-zero revenue counts.

6.7.Distribution of Revenue.

We transform revenue column into log revenue



Inference:-

- The revenue from a single visit ranged from \$0.01 to \$23129.50
- If look at distribution of the log of transaction revenue from individual visits then the mean of the natural log of transactions revenue is normally distributed and appears to be approx. 4

7.Feature Engineering.

Feature Enginerring

```
[48]: const_cols = [c for c in train_df.columns if train_df[c].nunique(dropna=False)==1 ]
      const_cols
```

```
[48]: ['socialEngagementType',
       'browserVersion',
       'browserSize',
       'operatingSystemVersion',
       'mobileDeviceBranding',
       'mobileDeviceModel',
       'mobileInputSelector',
       'mobileDeviceInfo',
       'mobileDeviceMarketingName',
       'flashVersion',
       'language',
       'screenColors',
       'screenResolution',
       'cityId',
       'latitude',
       'longitude',
       'networkLocation',
       'visits',
       'criteriaParameters']
```

```
[49]: cols_to_drop = const_cols + ['sessionId']

      train_df = train_df.drop(cols_to_drop + ["campaignCode"], axis=1)
```

```
[50]: train_df.head()
```

[50]:

	channelGrouping	date	fullVisitorId	visitId	visitNumber	visitStartTime	browser	operatingSystem	isMobile	deviceCategory
0	Organic Search	2016-09-02	1131660440785968503	1472830385	1	1472830385	Chrome	Windows	False	desktop
1	Organic Search	2016-09-02	377306020877927890	1472880147	1	1472880147	Firefox	Macintosh	False	desktop
2	Organic Search	2016-09-02	3895546263509774583	1472865386	1	1472865386	Chrome	Windows	False	desktop
3	Organic Search	2016-09-02	4763447161404445595	1472881213	1	1472881213	UC Browser	Linux	False	desktop
4	Organic Search	2016-09-02	27294437909732085	1472822600	2	1472822600	Chrome	Android	True	mobile

```
train_df['pageviews'].fillna(train_df['pageviews'].median(),inplace=True)
train_df['bounces'].fillna(train_df['bounces'].median(),inplace=True)
train_df['newVisits'].fillna(train_df['newVisits'].median(),inplace=True)
```

```
train_df.isna().sum()
```

```
channelGrouping      0
date                 0
fullVisitorId        0
visitId              0
visitNumber          0
visitStartTime       0
browser              0
operatingSystem      0
isMobile             0
deviceCategory       0
continent            0
subContinent         0
country              0
hits                 0
pageviews            0
bounces              0
newVisits            0
transactionRevenue   0
campaign             0
medium               0
dtype: int64
```

Inference:-

- Apart from the target columns “const_cols” are constant variables in our data so we need to remove these variables while building model.
- We impute missing values for pageviews, bounces and new Visits by using median.

7.1.Label Encoding.

```
]: # Impute 0 for missing target values
train_df["transactionRevenue"].fillna(0, inplace=True)
train_y = train_df["transactionRevenue"].values
train_id = train_df["fullVisitorId"].values
# test_id = test_df["fullVisitorId"].values


# label encode the categorical variables and convert the numerical variables to float
cat_cols = ["channelGrouping", "browser",
            "deviceCategory", "operatingSystem",
             "continent",
            "country",
            "subContinent",
             "campaign",
             "medium",
             ]
for col in cat_cols:
    print(col)
    lbl = preprocessing.LabelEncoder()
    lbl.fit(list(train_df[col].values.astype('str')))
    train_df[col] = lbl.transform(list(train_df[col].values.astype('str')))
    # test_df[col] = lbl.transform(list(test_df[col].values.astype('str')))


num_cols = ["hits", "pageviews", "visitNumber", "visitStartTime", 'bounces',  'newVisits']
for col in num_cols:
    train_df[col] = train_df[col].astype(float)
    # test_df[col] = test_df[col].astype(float)
channelGrouping
browser
deviceCategory
operatingSystem
continent
country
subContinent
campaign
medium
```

```
[60]: ### To convert Data type of columns into float for revenue and also apply log because revenue is skewd data and
      # high range values
      train_df['isMobile']=train_df['isMobile'].astype(int)
      # train_df['transactionRevenue']=np.log1p(train_df['transactionRevenue'].astype(float))

[61]: train_df.info()
      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 903653 entries, 0 to 903652
      Data columns (total 17 columns):
       #   Column              Non-Null Count   Dtype
      ---  ------              --------------   -----
       0   channelGrouping     903653 non-null  int32
       1   visitNumber         903653 non-null  float64
       2   visitStartTime      903653 non-null  float64
       3   browser             903653 non-null  int32
       4   operatingSystem     903653 non-null  int32
       5   isMobile            903653 non-null  int32
       6   deviceCategory      903653 non-null  int32
       7   continent           903653 non-null  int32
       8   subContinent        903653 non-null  int32
       9   country             903653 non-null  int32
       10  hits                903653 non-null  float64
       11  pageviews           903653 non-null  float64
       12  bounces             903653 non-null  float64
       13  newVisits           903653 non-null  float64
       14  transactionRevenue  903653 non-null  float64
       15  campaign            903653 non-null  int32
       16  medium              903653 non-null  int32
      dtypes: float64(7), int32(10)
      memory usage: 82.7 MB
```

Inference: -

- These columns are categorical columns so we convert it into numerical so that we use label encoding.
- The data types of some columns are categorical so we convert into numeric.

8.Model Building.

Train Test splitting

```
64]: from sklearn.model_selection import train_test_split
```

```
65]: ## Split Data into train and test
x=train_df.drop(['transactionRevenue','visitStartTime'],axis=1)
y=train_df['transactionRevenue']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.70,random_state=2)
```

1.Light gbm model building

```
66]: ## Model building
# custom function to run light gbm model
def run_lgb(x_train,y_train,x_test,y_test):
    params = {
        "objective" : "regression",
        "metric" : "rmse",
        "num_leaves" : 30,
        "min_child_samples" : 100,
        "learning_rate" : 0.1,
        "bagging_fraction" : 0.7,
        "feature_fraction" : 0.5,
        "bagging_frequency" : 5,
        "bagging_seed" : 2018,
        "verbosity" : -1
    }

    lgtrain = lgb.Dataset(x_train, label=y_train)
    lgval = lgb.Dataset(x_test, label=y_test)
    model = lgb.train(params, lgtrain, 1000, valid_sets=[lgval])
    pred_test_y = model.predict(x_test, num_iteration=model.best_iteration)
    return pred_test_y,model

# Training the model #
pred_test, model = run_lgb(x_train, y_train, x_test, y_test)
```

```
67]: ## Train data evaluation
from sklearn.metrics import mean_squared_error
y_train_pred=model.predict(x_train)
mse=mean_squared_error(y_train,y_train_pred)
rmse=np.sqrt(mse)
print(rmse)

1.539109390948733
```

```
68]: ## Test Data Evaluation
from sklearn.metrics import mean_squared_error,r2_score
mse=mean_squared_error(y_test,pred_test)
rmse=np.sqrt(mse)
r2=r2_score(y_test,pred_test)
print(rmse)

1.6856155812395748
```

Inference: -

- Light gbm model gives us **rmse=1.6856** on test data

2.Random Forest

Hyperparameter Tunning

87]:
```python
rf_reg = RandomForestRegressor(random_state=4) # random_state >> bootstrapping

hyp = {
    "n_estimators" : np.arange(10,100), # 230
    "max_depth" : np.arange(2,10),
    "min_samples_split" : np.arange(2,20),
    "min_samples_leaf" : np.arange(2,10),
    "max_features" :['auto','log2'],
    "bootstrap": [True],
    "oob_score" : [False],
}

rscv_rf_model = RandomizedSearchCV(rf_reg,hyp, cv= 3)
rscv_rf_model.fit(x_train, x_train)
```

87]:
```
RandomizedSearchCV
estimator: RandomForestRegressor
RandomForestRegressor
```

88]:
```python
rscv_rf_model.best_estimator_
```

88]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=8, max_features='log2', min_samples_leaf=6,
                      min_samples_split=3, n_estimators=33, random_state=4)
```

90]:
```python
rf_reg=RandomForestRegressor(max_depth=9, max_features='log2', min_samples_leaf=3,
                      min_samples_split=5, n_estimators=53, random_state=4)
rf_reg.fit(x_train, y_train)
```

90]:
```
RandomForestRegressor
RandomForestRegressor(max_depth=9, max_features='log2', min_samples_leaf=3,
                      min_samples_split=5, n_estimators=53, random_state=4)
```

91]:
```python
## Test data
y_pred=rf_reg.predict(x_test)
mse=mean_squared_error(y_test,y_pred)
rmse=np.sqrt(mse)
print(f'The rmse is {rmse}')
```

```
The rmse is 1.6780351335249215
```

Inference:-

- By using hyperparameter tunning we find best parameters for random forest model and we build model on that parameter
- Random forest gives us **rmse= 1.67** which is better than **Light gbm** model.

3.Ada-Boost

AdaBosst

```
99]: from sklearn.ensemble import AdaBoostRegressor
     from sklearn.metrics import mean_squared_error,r2_score

00]: ada =AdaBoostRegressor()

     ada.fit(x_train, y_train)

00]: ▾ AdaBoostRegressor
     AdaBoostRegressor()

01]: y_pred=ada.predict(x_test)
     mse=mean_squared_error(y_test,y_pred)
     rmse=np.sqrt(mse)
     print(f'The rmse is {rmse}')

     The rmse is 2.1765363959348583
```

Inference: -

- The rmse of **Ada-Boost** is **2.17** which is worse than **Random Forest** and **Light gbm**
- So, we cannot go with this model so we preferred **Random Forest**

9.Feature Importance.

```python
# Get feature importances
importances = rf_reg.feature_importances_

# Create a DataFrame for better visualization
feature_importances = pd.DataFrame({'Feature': x_train.columns, 'Importance': importances})

# Sort the DataFrame by importance
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
print(feature_importances)
```

```
          Feature  Importance
1       pageviews    0.460104
0            hits    0.265918
2     visitNumber    0.080971
4         country    0.068163
8       continent    0.034464
3 operatingSystem    0.031105
7          medium    0.030135
5         browser    0.014638
6    subContinent    0.014501
```

Inference: -

- By using Random Forest model, we get these 9 best features.

App Creation Using Flask

```python
from flask import Flask, request, render_template
import pandas as pd
import numpy as np
import pickle

app = Flask(__name__)

# Define lists of options
countries = ['Turkey', 'Australia', 'Spain', 'Indonesia', 'United Kingdom', 'Italy', 'Pakistan', 'Austria', 'Netherlands', 'India', 'France', 'Brazil',
'China', 'Singapore', 'Argentina', 'Poland', 'Germany', 'Canada', 'Thailand', 'Hungary', 'Malaysia', 'Denmark', 'Taiwan', 'Russia', 'Nigeria', 'Belgium',
'South Korea', 'Chile', 'Ireland', 'Philippines', 'Greece', 'Mexico', 'Montenegro', 'United States', 'Bangladesh', 'Japan', 'Slovenia', 'Czechia', 'Sweden',
'United Arab Emirates', 'Switzerland', 'Portugal', 'Peru', 'Hong Kong', 'Vietnam', 'Sri Lanka', 'Serbia', 'Norway', 'Romania', 'Kenya', 'Ukraine', 'Israel',
'Slovakia', 'Lithuania', 'Puerto Rico', 'Bosnia & Herzegovina', 'Croatia', 'South Africa', 'Paraguay', 'Others', 'Colombia', 'Uruguay', 'Algeria', 'Finland',
'Guatemala', 'Egypt', 'Malta', 'Bulgaria', 'New Zealand', 'Kuwait', 'Uzbekistan', 'Saudi Arabia', 'Cyprus', 'Estonia', 'Côte d'Ivoire', 'Morocco', 'Tunisia',
'Venezuela', 'Dominican Republic', 'Senegal', 'Costa Rica', 'Kazakhstan', 'Macedonia (FYROM)', 'Oman', 'Laos', 'Ethiopia', 'Panama', 'Belarus', 'Myanmar
(Burma)', 'Moldova', 'Bahrain', 'Mongolia', 'Ghana', 'Albania', 'Kosovo', 'Georgia', 'Tanzania', 'Bolivia', 'Cambodia', 'Iraq', 'Jordan', 'Lebanon', 'Ecuador',
'Jamaica', 'Trinidad & Tobago', 'Libya', 'El Salvador', 'Azerbaijan', 'Nicaragua', 'Palestine', 'Réunion', 'Iceland', 'Armenia', 'Uganda', 'Qatar', 'Cameroon',
'Latvia', 'Congo - Kinshasa', 'Kyrgyzstan', 'Honduras', 'Nepal', 'Luxembourg', 'Sudan', 'Yemen', 'Macau']
browsers = ['Chrome', 'Safari', 'Firefox', 'Internet Explorer', 'Edge', 'Android Webview', 'Safari (in-app)', 'Opera Mini', 'Opera', 'UC Browser', 'YaBrowser',
'Coc Coc', 'Amazon Silk', 'Android Browser', 'Mozilla Compatible Agent', 'MRCHROME', 'Maxthon', 'BlackBerry', 'Nintendo Browser']
subcontinents = ['Western Asia', 'Australasia', 'Southern Europe', 'Southeast Asia', 'Northern Europe', 'Southern Asia', 'Western Europe', 'South America',
'Eastern Asia', 'Eastern Europe', 'Northern America', 'Western Africa', 'Central America', 'Eastern Africa', '(not set)', 'Caribbean', 'Southern Africa',
'Northern Africa', 'Central Asia', 'Middle Africa', 'Melanesia', 'Micronesian Region', 'Polynesia']
operating_systems = ['Windows', 'Macintosh', 'Linux', 'Android', 'iOS', 'Chrome OS', 'BlackBerry', '(not set)', 'Samsung', 'Windows Phone', 'Xbox', 'Nintendo
Wii', 'Firefox OS', 'Nintendo WiiU', 'FreeBSD', 'Nokia', 'NTT DoCoMo', 'Nintendo 3DS', 'SunOS', 'OpenBSD']
mediums = ['organic', 'referral', 'cpc', 'affiliate', 'cpm']
continents = ['Asia', 'Oceania', 'Europe', 'Americas', 'Africa']

# Load the model and encoders
def load_model_and_encoders():
    # Load the RandomForestRegressor model
    with open('rf_model.pkl', 'rb') as model_file:
        model = pickle.load(model_file)

    # Load the encoders
    with open('label_encoders.pkl', 'rb') as encoders_file:
        encoders = pickle.load(encoders_file)

    # Load feature names
    with open('feature_names.pkl', 'rb') as feature_file:
        feature_names = pickle.load(feature_file)

    return model, encoders, feature_names

model, encoders, feature_names = load_model_and_encoders()

# Route for the homepage
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        # Extract data from form
        hits = int(request.form.get('hits', 0))
        pageviews = int(request.form.get('pageviews', 0))
        visitNumber = int(request.form.get('visitNumber', 0))
```

```python
        operatingSystem = request.form.get('operatingSystem')
        medium = request.form.get('medium')

        # Prepare input data
        input_data = pd.DataFrame({
            'hits': [hits],
            'pageviews': [pageviews],
            'visitNumber': [visitNumber],
            'country': [country],
            'continent': [continent],
            'browser': [browser],
            'subContinent': [subContinent],
            'operatingSystem': [operatingSystem],
            'medium': [medium]
        })

        input_data.replace('', np.nan, inplace=True)

        for feature in encoders:
            if feature in input_data.columns:
                encoder = encoders[feature]
                if 'Unknown' not in encoder.classes_:
                    encoder.classes_ = np.append(encoder.classes_, 'Unknown')
                input_data[feature] = input_data[feature].fillna('Unknown')
                input_data[feature] = input_data[feature].apply(lambda x: x if x in encoder.classes_ else 'Unknown')
                input_data[feature] = encoder.transform(input_data[feature])

        # Reorder columns to match feature names used during model training
        input_data = input_data.reindex(columns=feature_names, fill_value=0)

        try:
            # Predict using the model
            log_prediction = model.predict(input_data)
            predicted_revenue = np.exp(log_prediction[0])  # Using np.exp to revert the log transformation
            predicted_revenue = 1000000*(max(0, predicted_revenue))  # Convert to millions and ensure no negative values

            return render_template('index.html', predicted_revenue=f"{predicted_revenue:.2f} ",
                                   countries=countries, continents=continents, browsers=browsers,
                                   subcontinents=subcontinents, operating_systems=operating_systems,
                                   mediums=mediums)
        except Exception as e:
            return render_template('index.html', error=str(e),
                                   countries=countries, continents=continents, browsers=browsers,
                                   subcontinents=subcontinents, operating_systems=operating_systems,
                                   mediums=mediums)

    return render_template('index.html',
                           countries=countries, continents=continents, browsers=browsers,
                           subcontinents=subcontinents, operating_systems=operating_systems,
                           mediums=mediums)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Inference: -

- After model building, we create app by using flask.

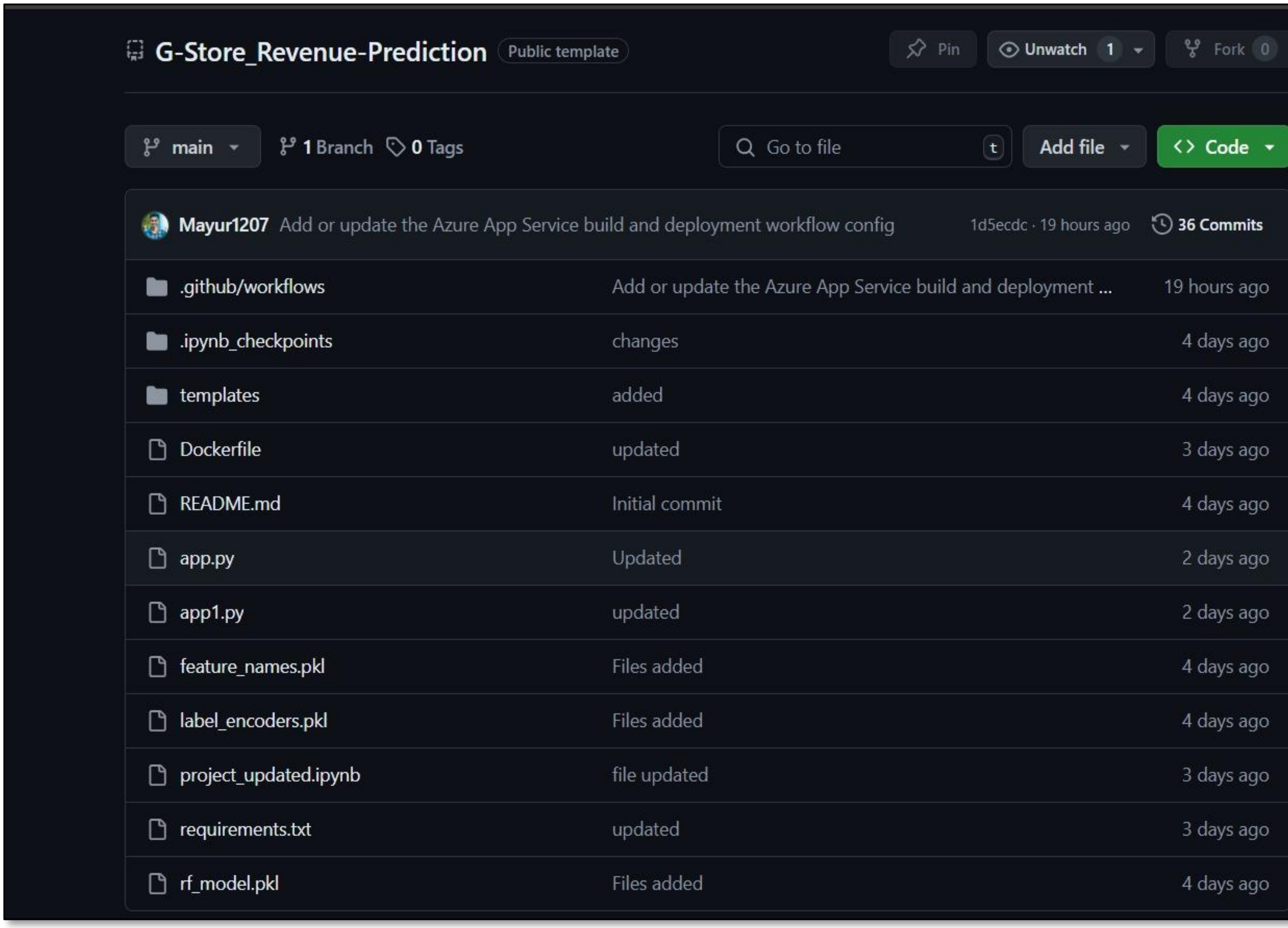
11.Git Hub Actions.

1. **To clone repository:**
    i. git clone [repository-URL]

2. **Add the File to Your Repository:**
    i. git add.

3. **Commit the Changes:**
    i. Commit the changes with a descriptive message.
    ii. git commit -m "Add description of the changes"

4. **Push the Changes to GitHub:**
    i. Push the committed changes to the remote repository on GitHub
    ii. git push origin main


G-Store_Revenue-Prediction
Public template
Pin
Unwatch 1
Fork 0
main
1 Branch
0 Tags
Go to file
t
Add file
Code
Mayur1207 Add or update the Azure App Service build and deployment workflow config
1d5ecdc · 19 hours ago
36 Commits
.github/workflows
Add or update the Azure App Service build and deployment ...
19 hours ago
.ipynb_checkpoints
changes
4 days ago
templates
added
4 days ago
Dockerfile
updated
3 days ago
README.md
Initial commit
4 days ago
app.py
Updated
2 days ago
app1.py
updated
2 days ago
feature_names.pkl
Files added
4 days ago
label_encoders.pkl
Files added
4 days ago
project_updated.ipynb
file updated
3 days ago
requirements.txt
updated
3 days ago
rf_model.pkl
Files added
4 days ago

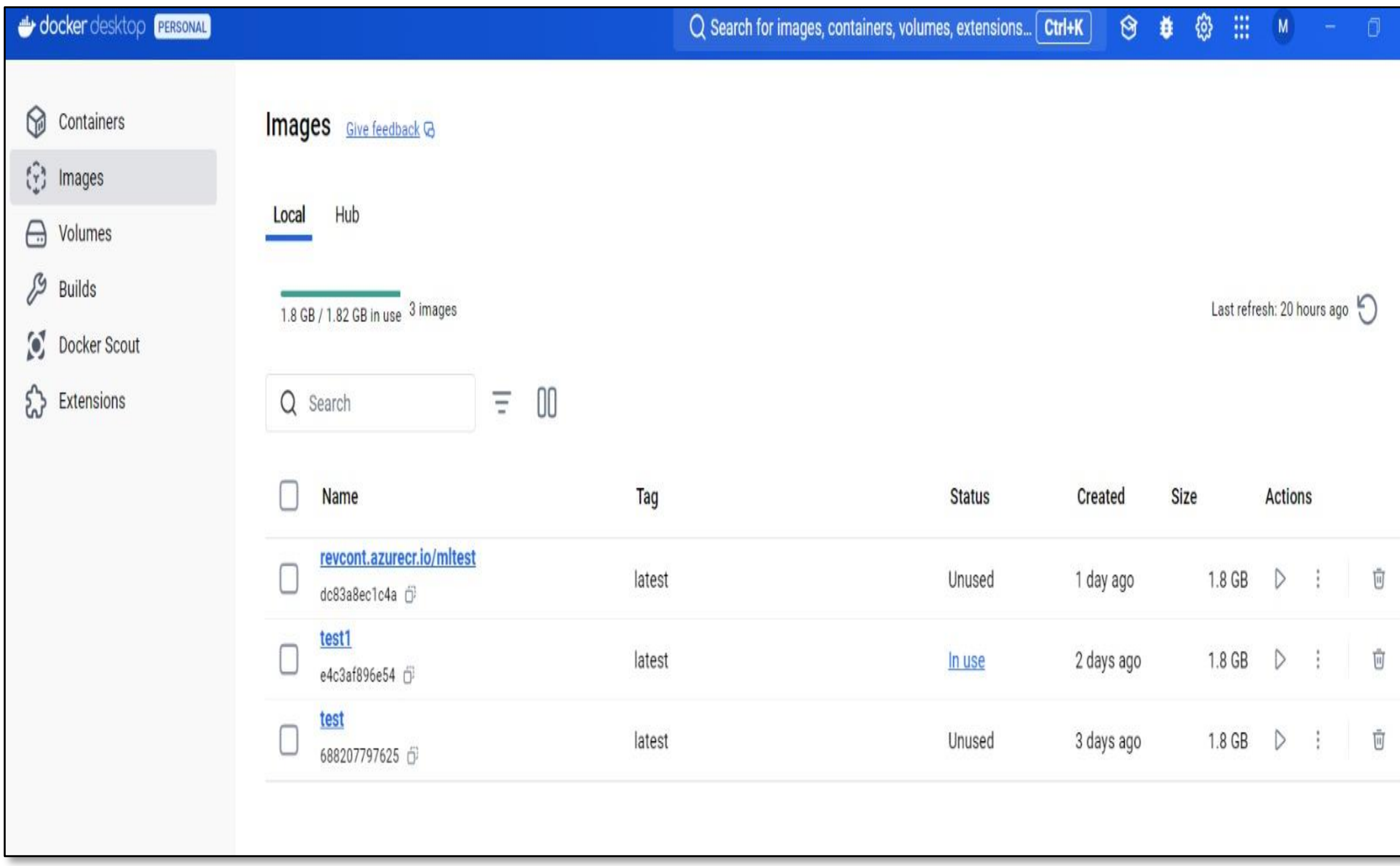
12.Docker Image Creation.

- **To build Docker Image**

  docker build -t “Image name”.

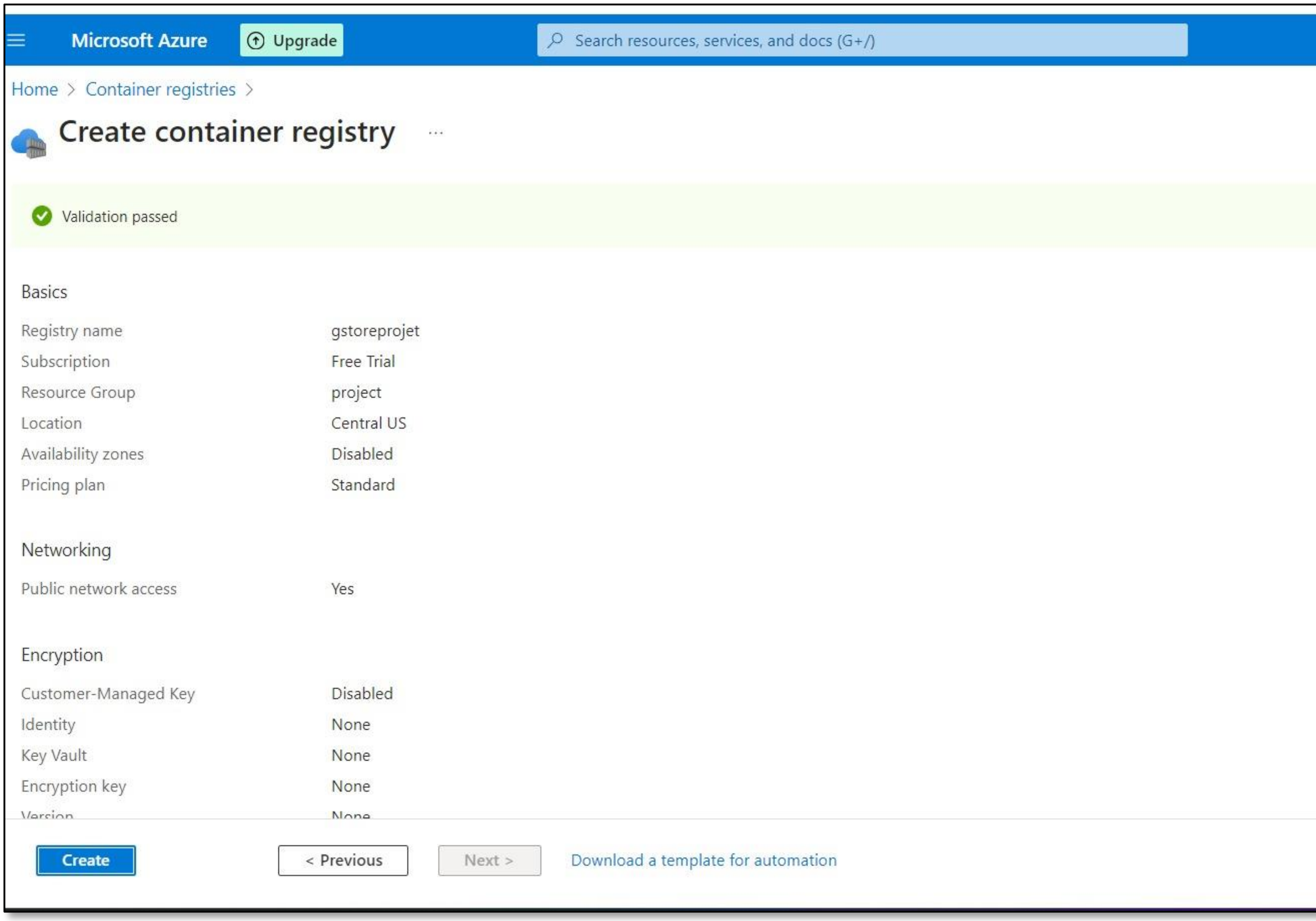
- **To run Docker Image Use**

  docker run -p 8000:8000 “Image name”

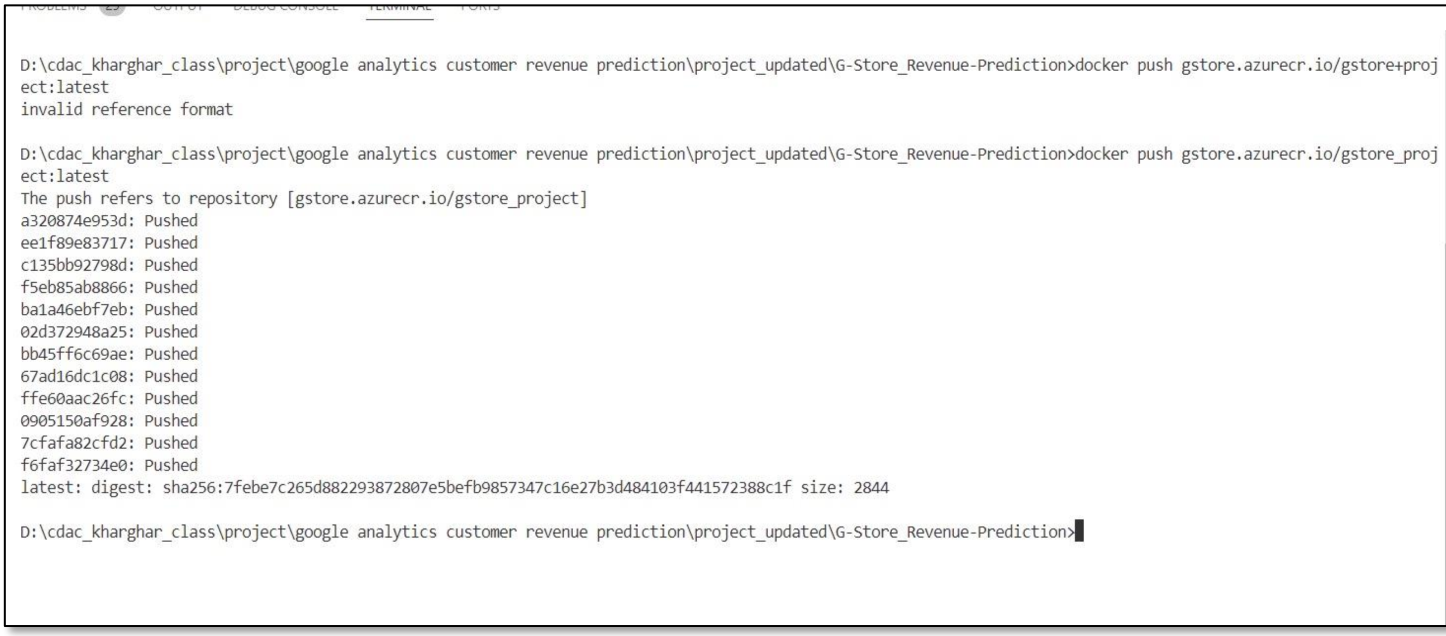


13.Azure Deployment.

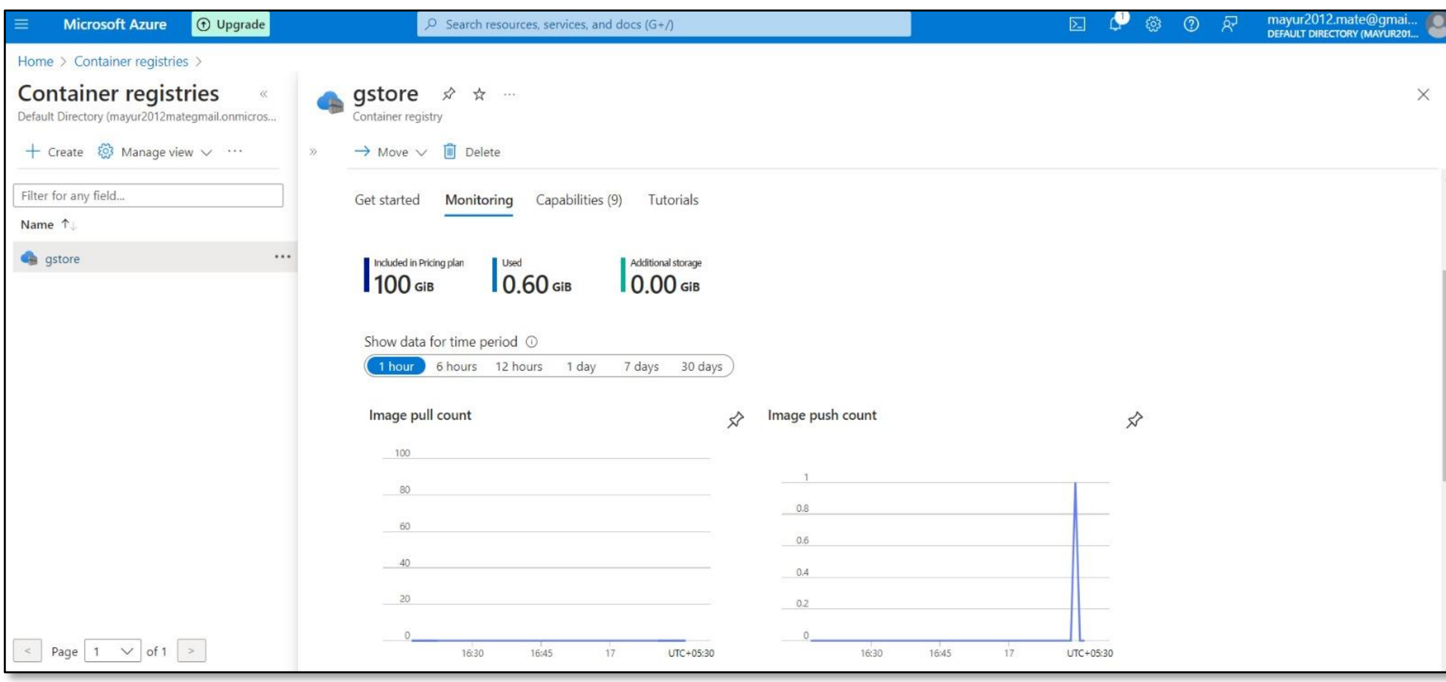
1.Create Container Registry :-



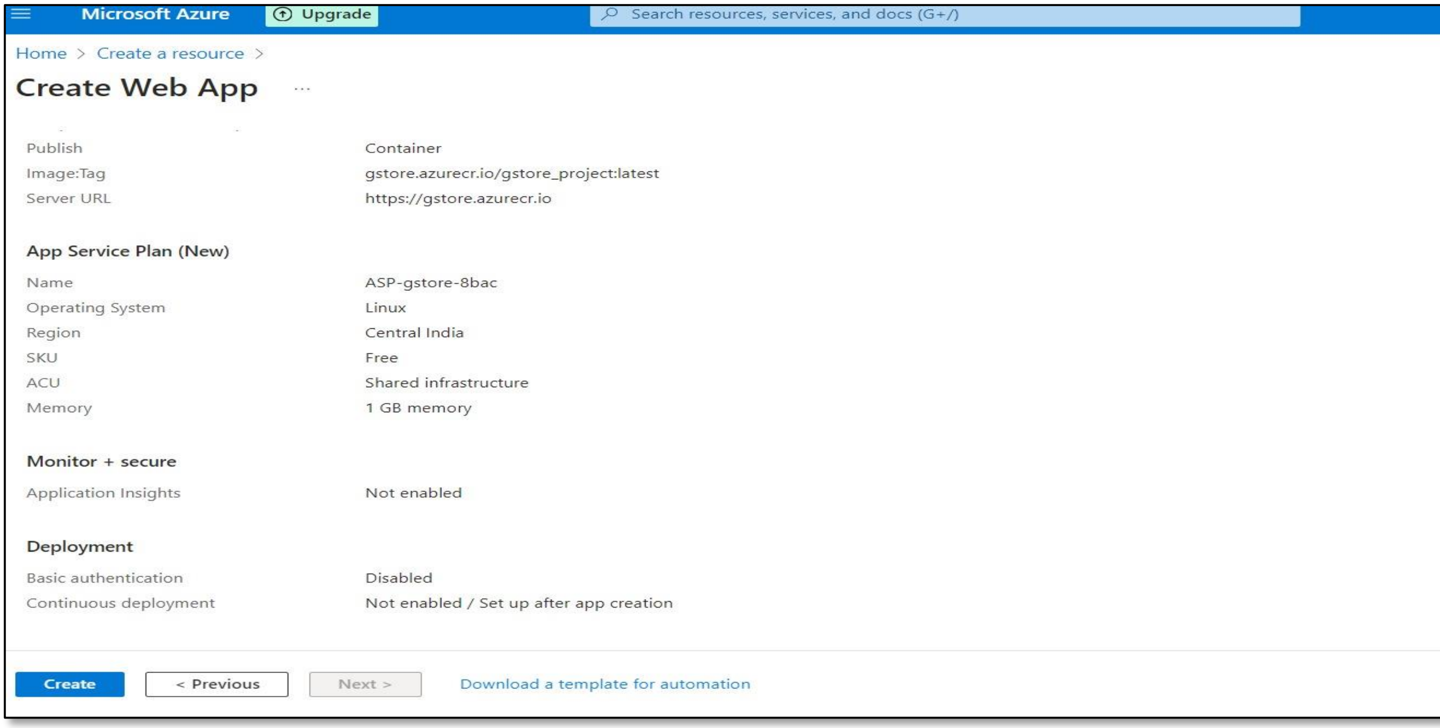
2.Upload Docker Image to Azure :-

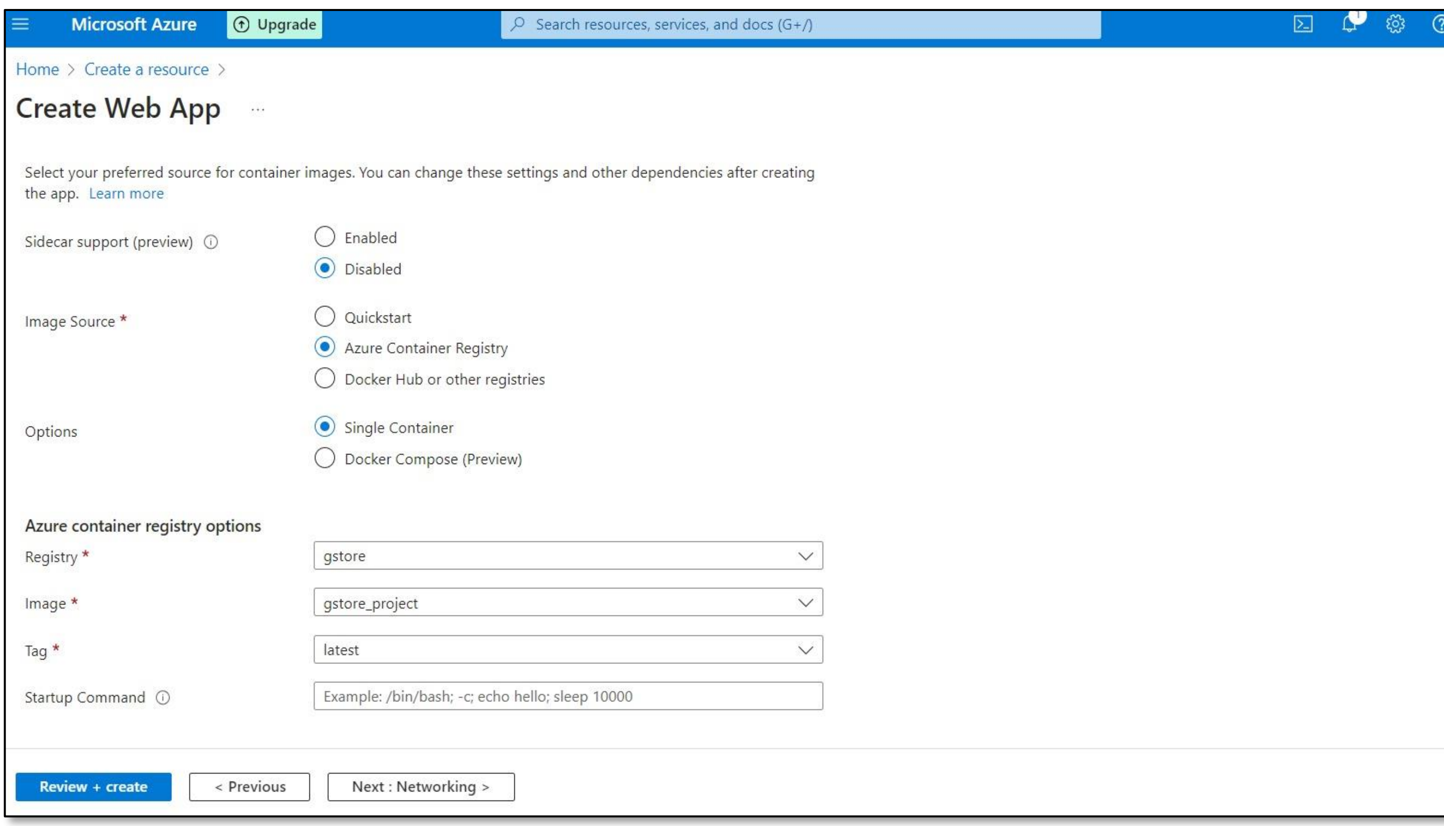


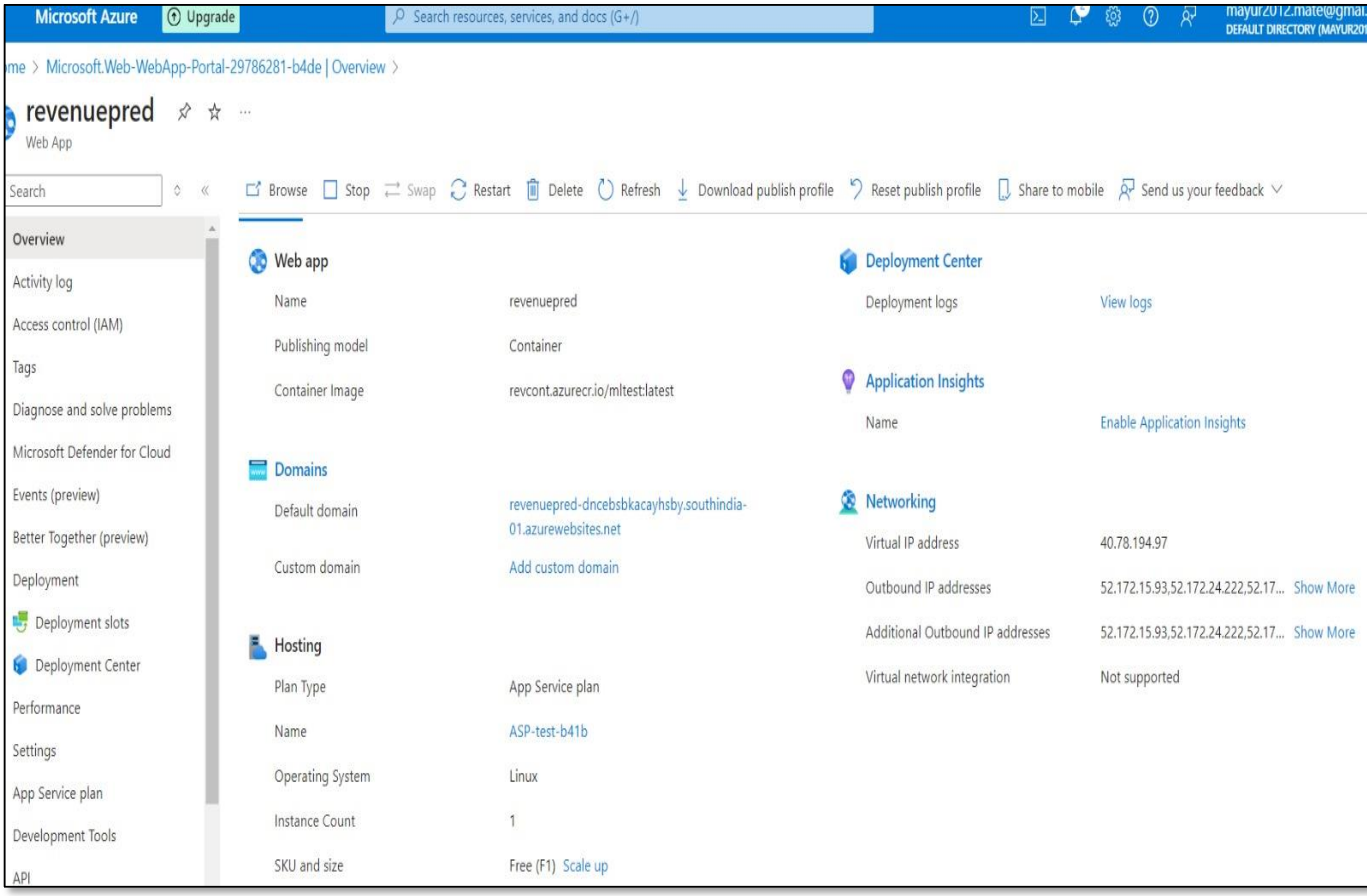
```
D:\cdac_kharghar_class\project\google analytics customer revenue prediction\project_updated\G-Store_Revenue-Prediction>docker push gstore.azurecr.io/gstore+proj
ect:latest
invalid reference format

D:\cdac_kharghar_class\project\google analytics customer revenue prediction\project_updated\G-Store_Revenue-Prediction>docker push gstore.azurecr.io/gstore_proj
ect:latest
The push refers to repository [gstore.azurecr.io/gstore_project]
a320874e953d: Pushed
ee1f89e83717: Pushed
c135bb92798d: Pushed
f5eb85ab8866: Pushed
ba1a46ebf7eb: Pushed
02d372948a25: Pushed
bb45ff6c69ae: Pushed
67ad16dc1c08: Pushed
ffe60aac26fc: Pushed
0905150af928: Pushed
7cfafa82cfd2: Pushed
f6faf32734e0: Pushed
latest: digest: sha256:7febe7c265d882293872807e5befb9857347c16e27b3d484103f441572388c1f size: 2844

D:\cdac_kharghar_class\project\google analytics customer revenue prediction\project_updated\G-Store_Revenue-Prediction>
```



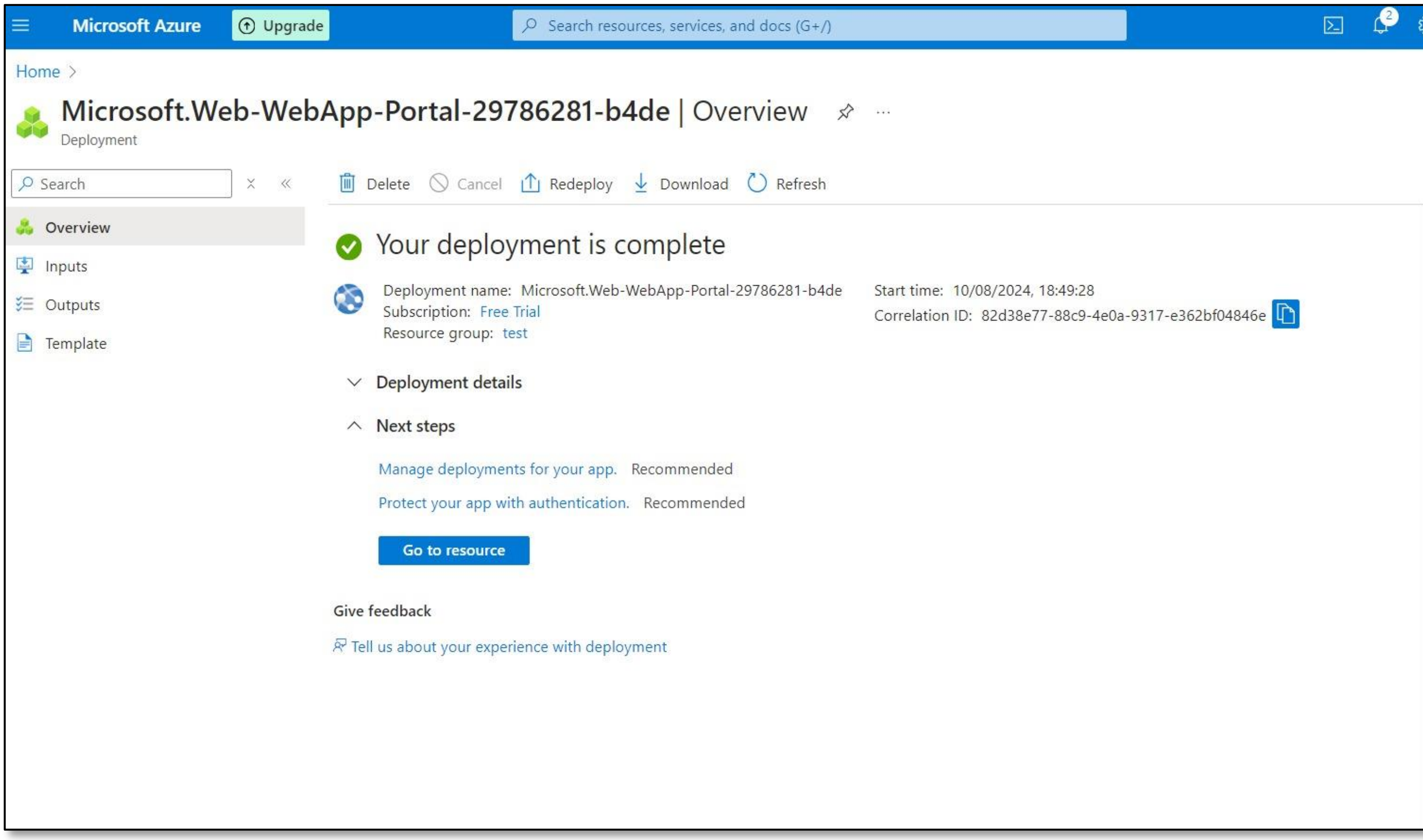
3.Create Web App :-

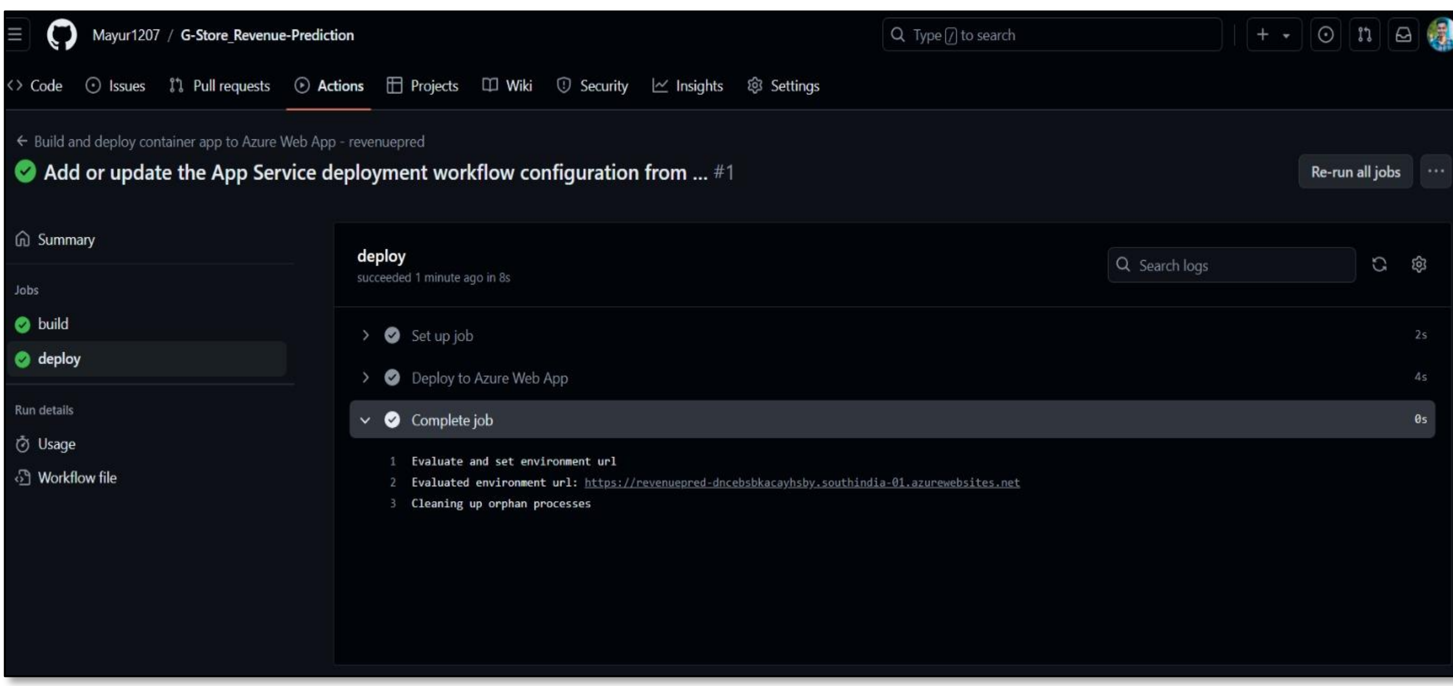


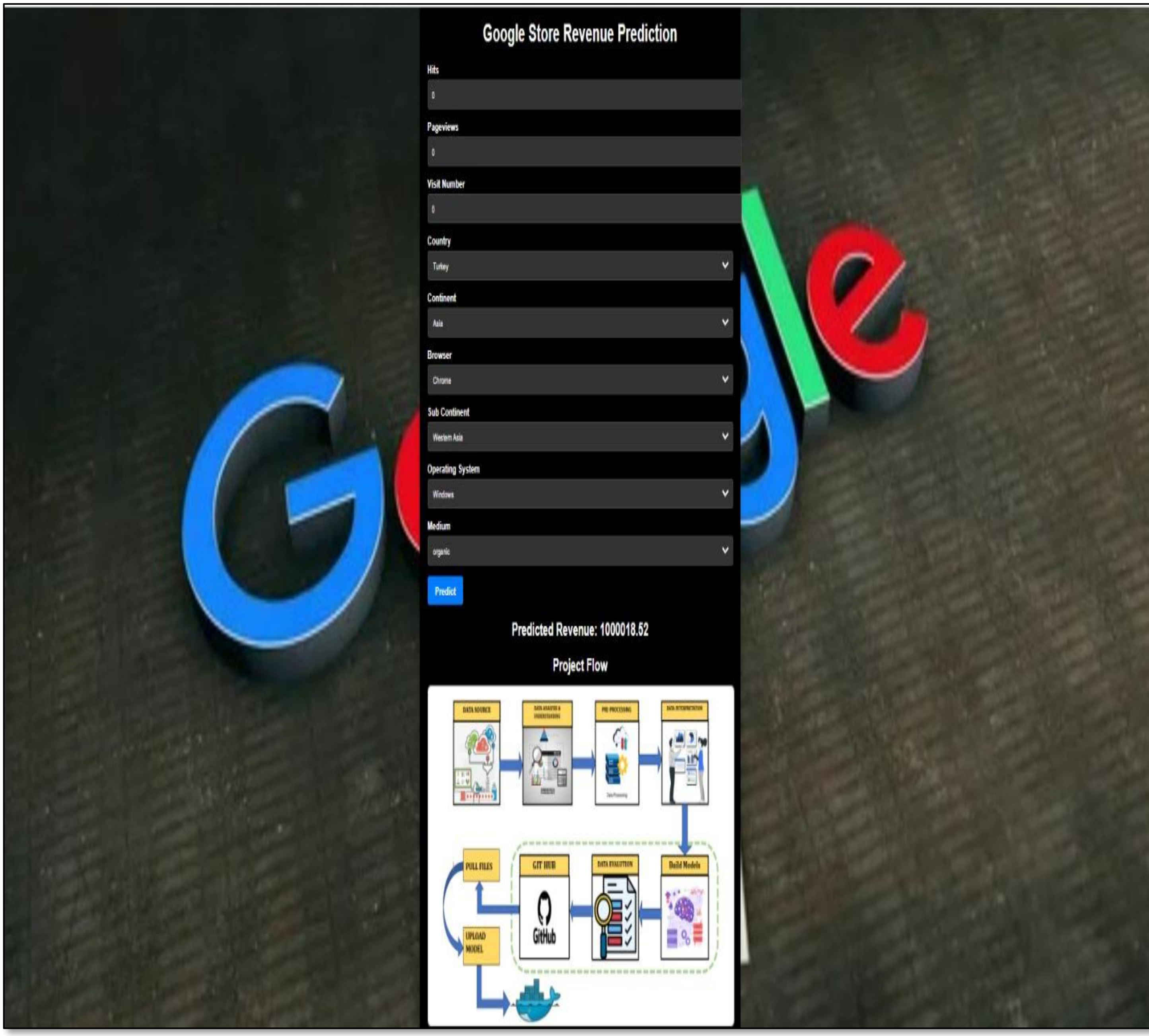



Microsoft Azure
Upgrade
Search resources, services, and docs (G+/)
mayur2012.mate@gmai...
DEFAULT DIRECTORY (MAYUR201...
me > Microsoft.Web-WebApp-Portal-29786281-b4de | Overview >
revenuepred
Web App
Search
Browse
Stop
Swap
Restart
Delete
Refresh
Download publish profile
Reset publish profile
Share to mobile
Send us your feedback
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Microsoft Defender for Cloud
Events (preview)
Better Together (preview)
Deployment
Deployment slots
Deployment Center
Performance
Settings
App Service plan
Development Tools
API
Web app
Name
revenuepred
Publishing model
Container
Container Image
revcont.azurecr.io/mltest:latest
Domains
Default domain
revenuepred-dncebsbkacayhsby.southindia-01.azurewebsites.net
Custom domain
Add custom domain
Hosting
Plan Type
App Service plan
Name
ASP-test-b41b
Operating System
Linux
Instance Count
1
SKU and size
Free (F1) Scale up
Deployment Center
Deployment logs
View logs
Application Insights
Name
Enable Application Insights
Networking
Virtual IP address
40.78.194.97
Outbound IP addresses
52.172.15.93,52.172.24.222,52.17... Show More
Additional Outbound IP addresses
52.172.15.93,52.172.24.222,52.17... Show More
Virtual network integration
Not supported

4.Web Deployment

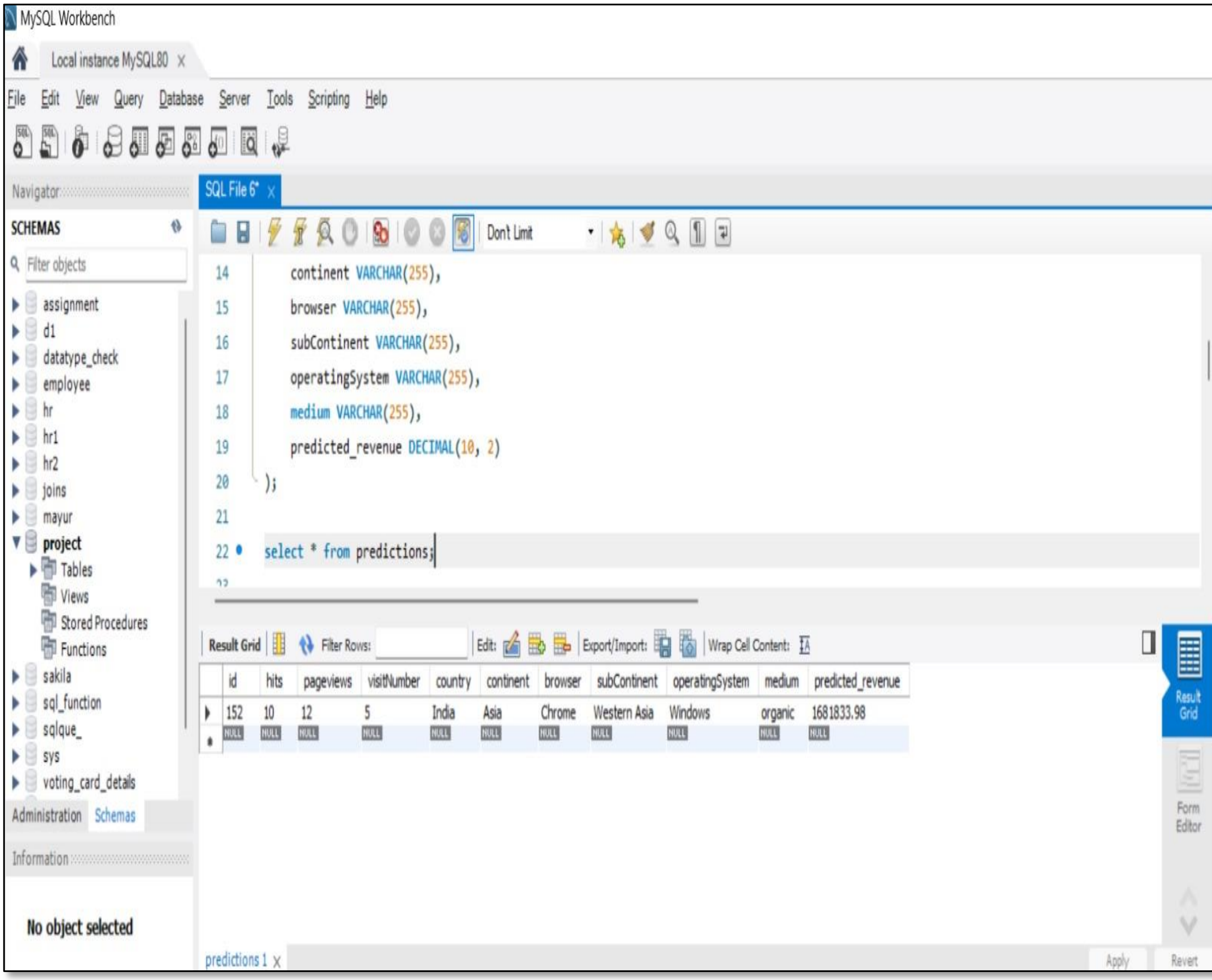





Google Store Revenue Prediction
Hits
0
Pageviews
0
Visit Number
0
Country
Turkey
Continent
Asia
Browser
Chrome
Sub Continent
Western Asia
Operating System
Windows
Medium
organic
Predict
Predicted Revenue: 1000018.52
Project Flow
DATA SOURCE
PRE PROCESSING
PULL FILES
GIT HUB
Build Models
UPLOAD MODEL
GitHub

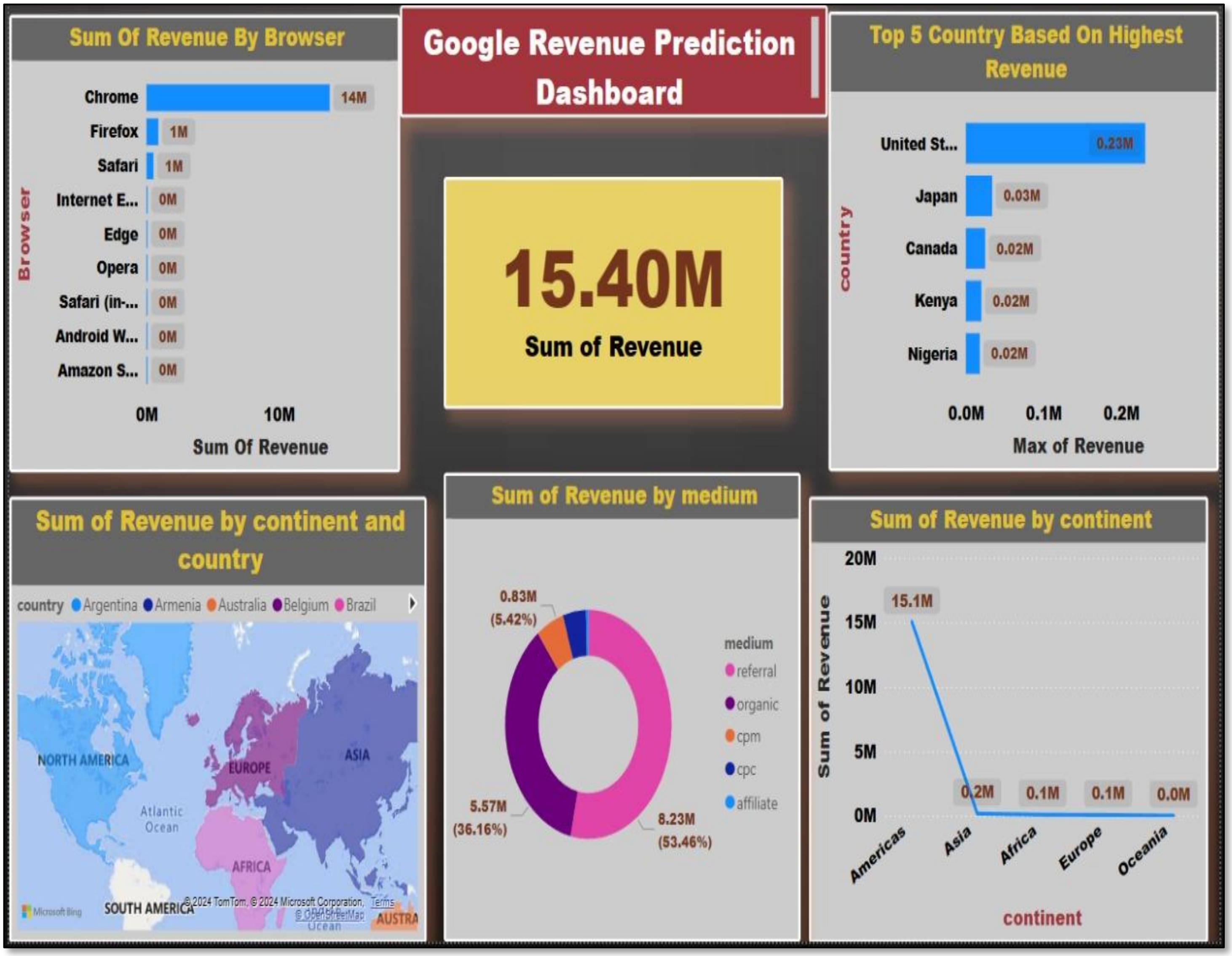
14.MySQL Database Connection.

We save prediction in MySQL database by connecting to WebApp



15.Dashboard Creation.

- We Created Dashboard using Power BI for predicted value by connecting Power Bi to MySQL Database and plot Different Types of graphs for Visualization.



16.Technology used.

- **Python**
- **Machine Learning**
- **Git Hub**
- **Docker**
- **Azure**
- **MySQL**
- **Power BI**

17.Conclusion.

1.Model Development:

1. **Feature Engineering:**
    a. We meticulously selected and encoded relevant features from the dataset, including country, browser, medium and other categorical variables.
2. **Model Training:**
    a. We used **Random Forest** model due to **low RSME** as compared to Light GBM and Ada-Boost model.
3. **Evaluation:**
    a. The model was evaluated based on performance metrics such as **RMSE** and reliability in predictions.

2.Deployment Strategy:

1. **Containerization:**
    a. To ensure consistency across different environments, we created a Docker image encapsulating the trained model and the necessary dependencies. This approach simplifies deployment and scaling.
2. **Cloud Deployment**:
    a. The Docker image was deployed on Azure, leveraging its infrastructure to manage and scale the application efficiently.
3. **CI/CD Pipeline:**
    a. We set up a CI/CD pipeline using GitHub Actions, automating the build, test, and deployment processes. This pipeline ensures continuous integration of code changes and smooth deployment of updates.

3.Future Scope:

1. **Real-Time Predictions:**
2. **Stream Processing:** Implement real-time data processing and prediction capabilities using technologies like Apache Kafka or Azure Stream Analytics to provide immediate insights and actions.

Model Monitoring and Retraining:

- **Performance Monitoring:** Develop systems to continuously monitor model performance and detect any drift or degradation over time.
- **Automated Retraining:** Set up automated pipelines for model retraining and evaluation based on new data to keep the model current and accurate.

References

Dataset Link:

https://www.kaggle.com/competitions/ga-customer-revenue-prediction/code?competitionId=10038&sortBy=voteCount&excludeNonAccessedDatasources=true

Models:

1.Light gbm

https://lightgbm.readthedocs.io/en/latest/Python-Intro.html

2.Random Forest

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

3.Ada-Boost

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

4.Docker:

https://docs.docker.com/reference/

5.Azure:

https://learn.microsoft.com/en-us/azure/?product=popular