Click Fraud Detection Project Documentation

Submitted By-

Team 2 BFS Track

Rahul Tomar – Team Lead

Dipanshu Upreti – Python Track

Vaishnavi Subramanya Desai – Python Track

Girish Kumar Reddy Tokala – Python Track

Ragini Gaurav - UI Track

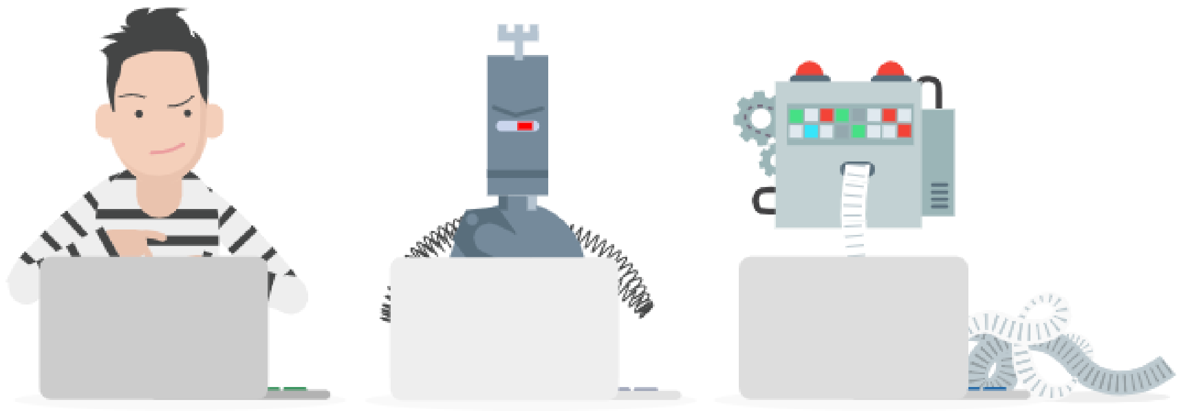
Table of Contents

CLICK FRAUD DETECTION PROJECT

Introduction

Click fraud is a type of fraud that occurs on the Internet in pay-per-click (PPC) online advertising. In this type of advertising, the owners of websites that post the ads are paid an amount of money determined by how many visitors to the sites click on the ads. Fraud occurs when a person, automated script, or computer program imitates a legitimate user of a web browser, clicking on such an ad without having actual interest in the target of the ad's link.

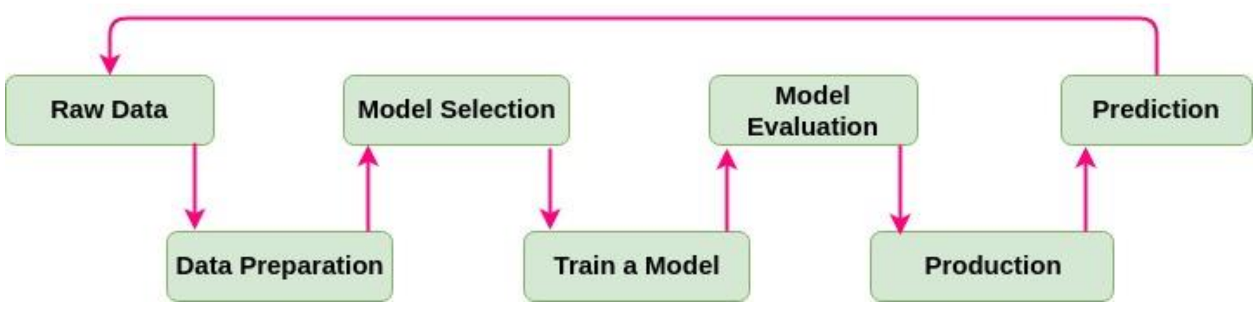
Fraud risk is everywhere, but for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad at a large scale.



Although Google Adsense has checked in place it is still advisable to monitor your traffic details and analyzing them from time to time to monitor the data.

That's where our product come in the picture. It serves as a way to predict the spamming IP and Devices and find the rate at which Channels are generating clicks to find and isolate wasteful channels and increase spending in the better ones.

Model Development



Dataset

Dataset used by us was picked from Kaggle TalkingData AdTracking Fraud Detection Challenge(https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data)

Each row of the training data contains a click record, with the following features.

- IP: IP address of click.
- app: app id for marketing.
- device: device **type** id of user mobile phone (e.g., iPhone 6 plus, iPhone 7, Huawei mate 7, etc.)
- os: os version id of user mobile phone
- channel: channel id of mobile ad publisher
- click_time: timestamp of click (UTC)
- attributed_time: if user download the app for after clicking an ad, this is the time of the app download
- is_attributed: the target that is to be predicted, indicating the app was downloaded

Note that IP, app, device, os, and channel are encoded.

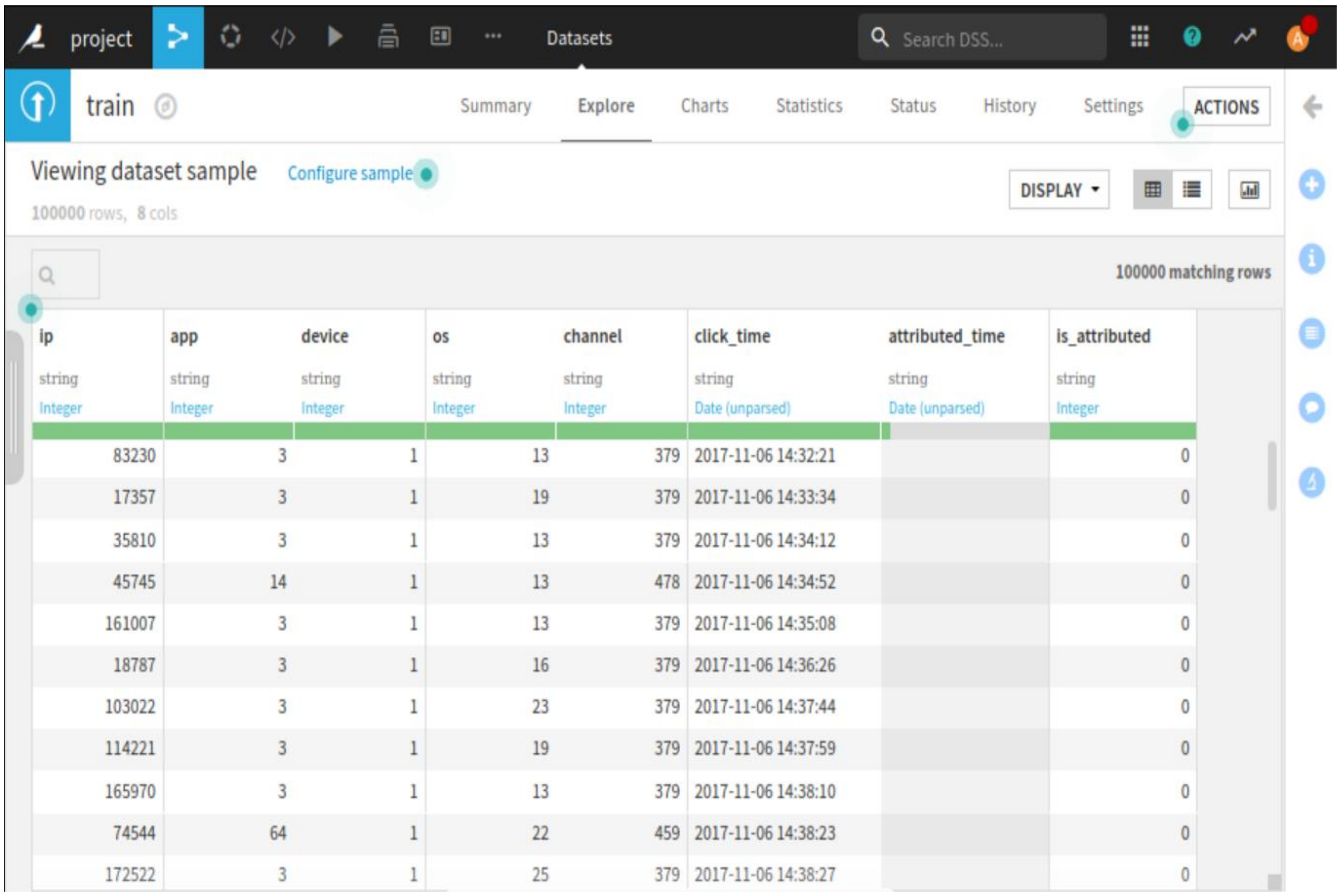
The test data is similar, with the following differences:

- click_id: a reference for making predictions
- is_attributed: not included

Our task was to build a machine learning model using **Dataiku DSS**(Data Science Software) which is a collaborative data science software platform for teams of data scientists, data analysts, and engineers to explore, prototype, build and deliver their data products more efficiently.

Loading dataset in Dataiku

This is how our dataset looked like



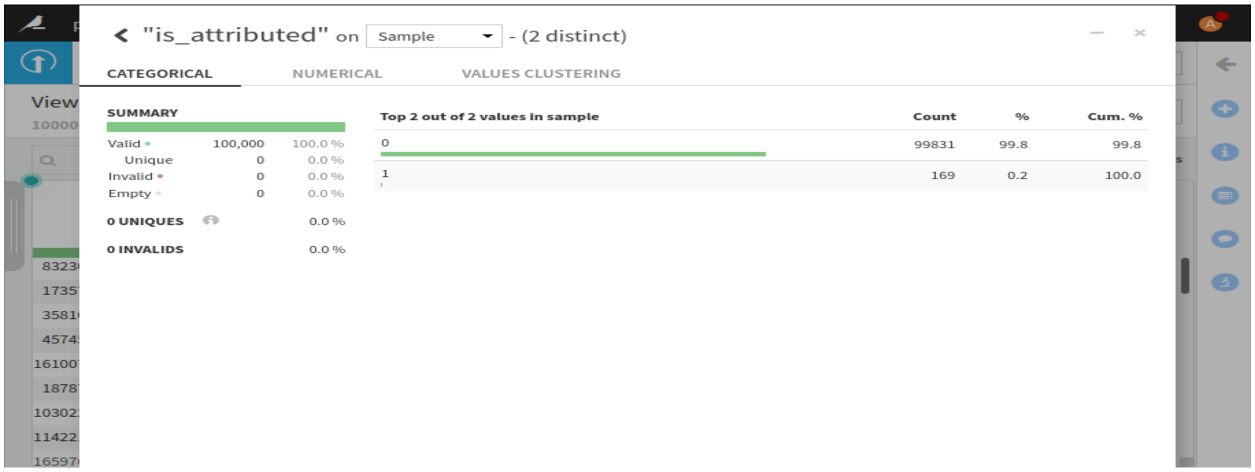
ip	app	device	os	channel	click_time	attributed_time	is_attributed
string	string	string	string	string	string	string	string
Integer	Integer	Integer	Integer	Integer	Date (unparsed)	Date (unparsed)	Integer
83230	3	1	13	379	2017-11-06 14:32:21		0
17357	3	1	19	379	2017-11-06 14:33:34		0
35810	3	1	13	379	2017-11-06 14:34:12		0
45745	14	1	13	478	2017-11-06 14:34:52		0
161007	3	1	13	379	2017-11-06 14:35:08		0
18787	3	1	16	379	2017-11-06 14:36:26		0
103022	3	1	23	379	2017-11-06 14:37:44		0
114221	3	1	19	379	2017-11-06 14:37:59		0
165970	3	1	13	379	2017-11-06 14:38:10		0
74544	64	1	22	459	2017-11-06 14:38:23		0
172522	3	1	25	379	2017-11-06 14:38:27		0

The actual train dataset was very large around **7GB** in size containing 180 million entries. With our systems, it was not feasible to work in such big dataset.

So, we worked only with 100k entries considering our system capabilities. Although the model is completely scalable and can be trained with a larger dataset in future.

Basic Exploratory Data Analysis

It can be seen how skewed our dataset was from a first look. It was very highly unbalanced and biased towards the 0 value that corresponds to no attribution.



"is_attributed" on Sample - (2 distinct)

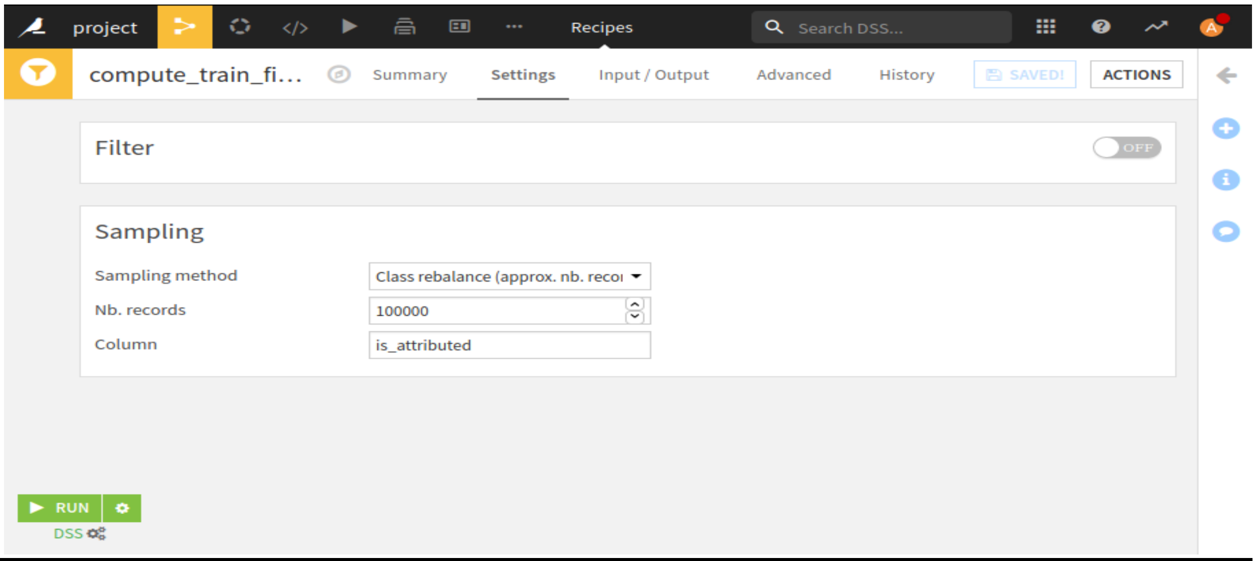
CATEGORICAL NUMERICAL VALUES CLUSTERING

SUMMARY

Valid	100,000	100.0 %
Unique	0	0.0 %
Invalid	0	0.0 %
Empty	0	0.0 %
0 UNIQUES		0.0 %
0 INVALIDS		0.0 %

Top 2 out of 2 values in sample	Count	%	Cum. %
0	99831	99.8	99.8
1	169	0.2	100.0

Our task was to balance the dataset for creating a more accurate model which we achieved using Dataiku sampling recipe.

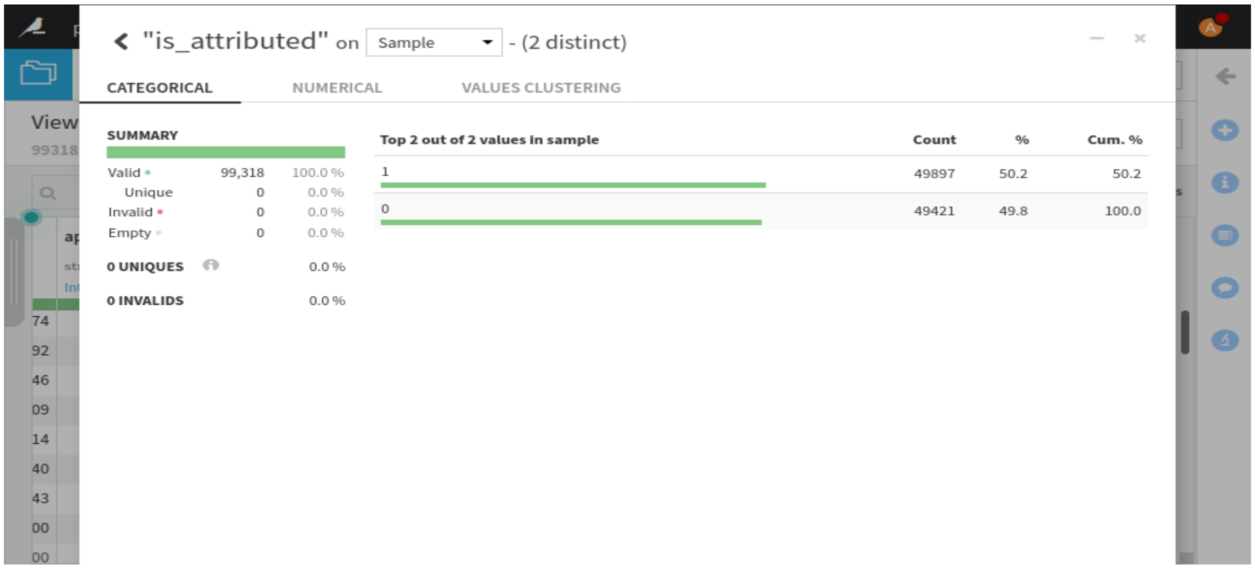


Filter OFF

Sampling

Sampling method	Class rebalance (approx. nb. recoı
Nb. records	100000
Column	is_attributed

We used class rebalance sampling method for balancing the is_attributed column and for filtering out 100k such entries from the dataset.


"is_attributed" on Sample - (2 distinct)

CATEGORICAL · NUMERICAL · VALUES CLUSTERING

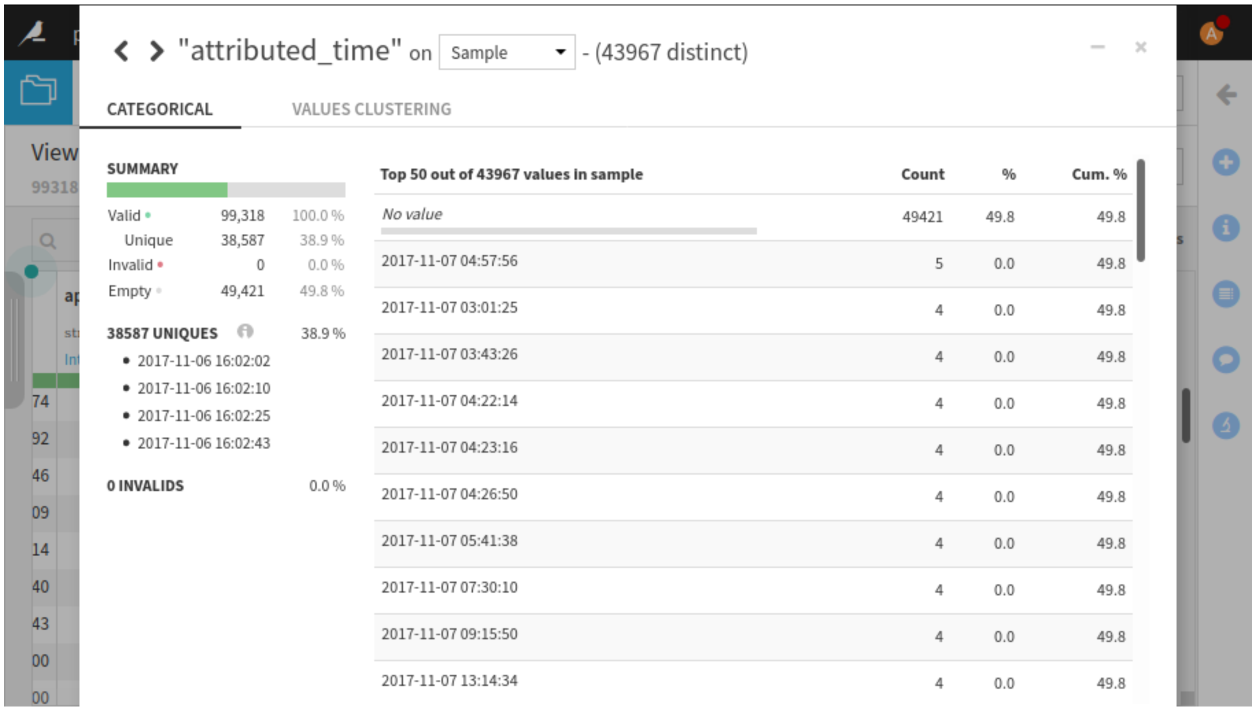
SUMMARY

Valid	99,318	100.0 %
Unique	0	0.0 %
Invalid	0	0.0 %
Empty	0	0.0 %
0 UNIQUES		0.0 %
0 INVALIDS		0.0 %

Top 2 out of 2 values in sample	Count	%	Cum. %
1	49897	50.2	50.2
0	49421	49.8	100.0

After performing sampling we got a new balanced dataset with almost equal values of 0 and 1 in is_attributed column.

Analysis of attributed_time showed that it contains mostly null or no values and hence was not helpful for prediction.


"attributed_time" on Sample - (43967 distinct)

CATEGORICAL · VALUES CLUSTERING

SUMMARY

Valid	99,318	100.0 %
Unique	38,587	38.9 %
Invalid	0	0.0 %
Empty	49,421	49.8 %
38587 UNIQUES		38.9 %
0 INVALIDS		0.0 %

- 2017-11-06 16:02:02
- 2017-11-06 16:02:10
- 2017-11-06 16:02:25
- 2017-11-06 16:02:43

Top 50 out of 43967 values in sample	Count	%	Cum. %
No value	49421	49.8	49.8
2017-11-07 04:57:56	5	0.0	49.8
2017-11-07 03:01:25	4	0.0	49.8
2017-11-07 03:43:26	4	0.0	49.8
2017-11-07 04:22:14	4	0.0	49.8
2017-11-07 04:23:16	4	0.0	49.8
2017-11-07 04:26:50	4	0.0	49.8
2017-11-07 05:41:38	4	0.0	49.8
2017-11-07 07:30:10	4	0.0	49.8
2017-11-07 09:15:50	4	0.0	49.8
2017-11-07 13:14:34	4	0.0	49.8

So, this feature was dropped and was not included for model creation.

Feature Engineering

Dataset had some features through which certain properties could be extracted. It had click_time column through which date components were selected.

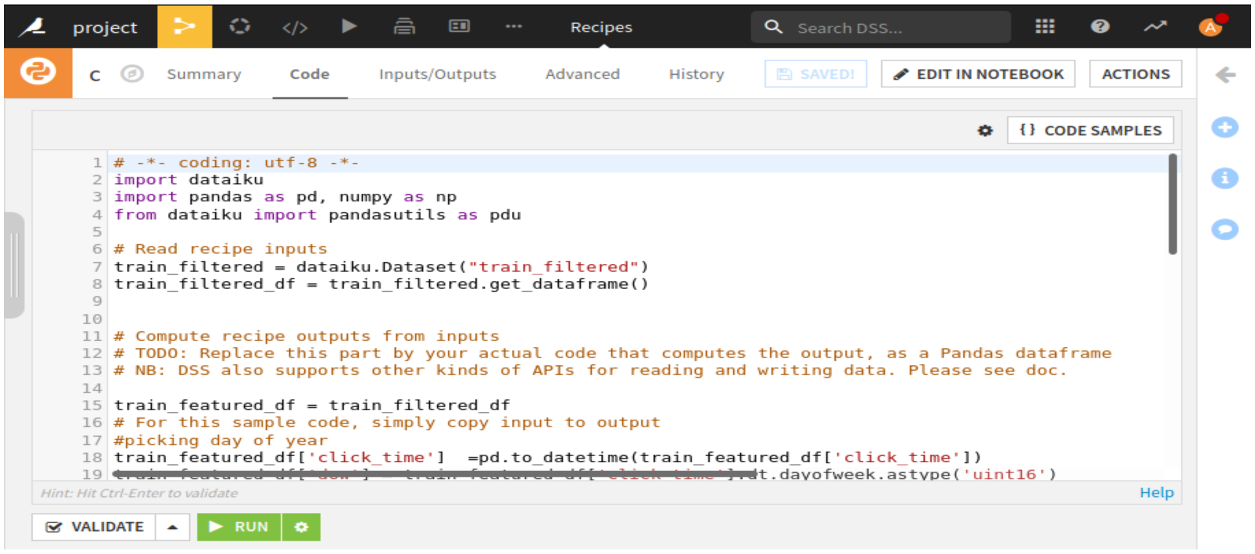
<u>Date Components selected were</u>:-

hour,doy(day of year),dow(day of week)

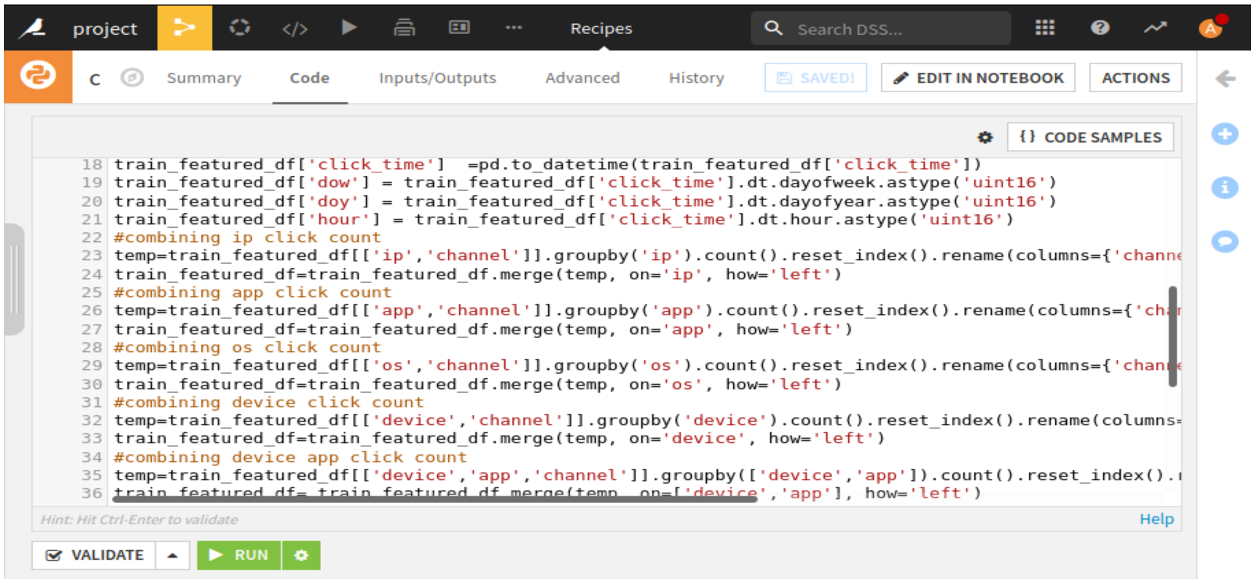
<u>Groupby Aggregations done are by click count of</u>:-

ip, os, device, app , ip and app, os and app, device and app

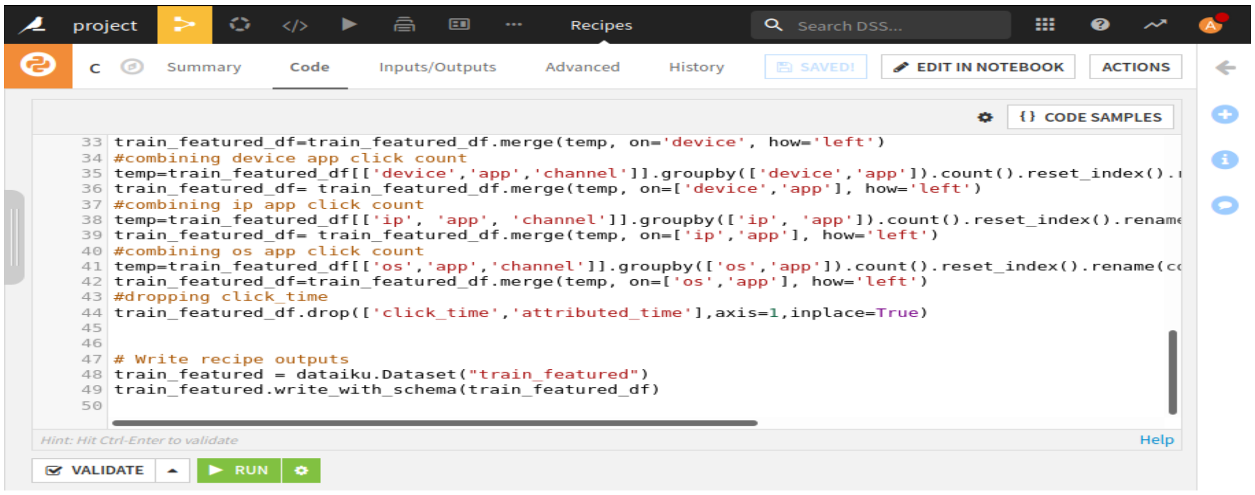
The python code recipe used was:-



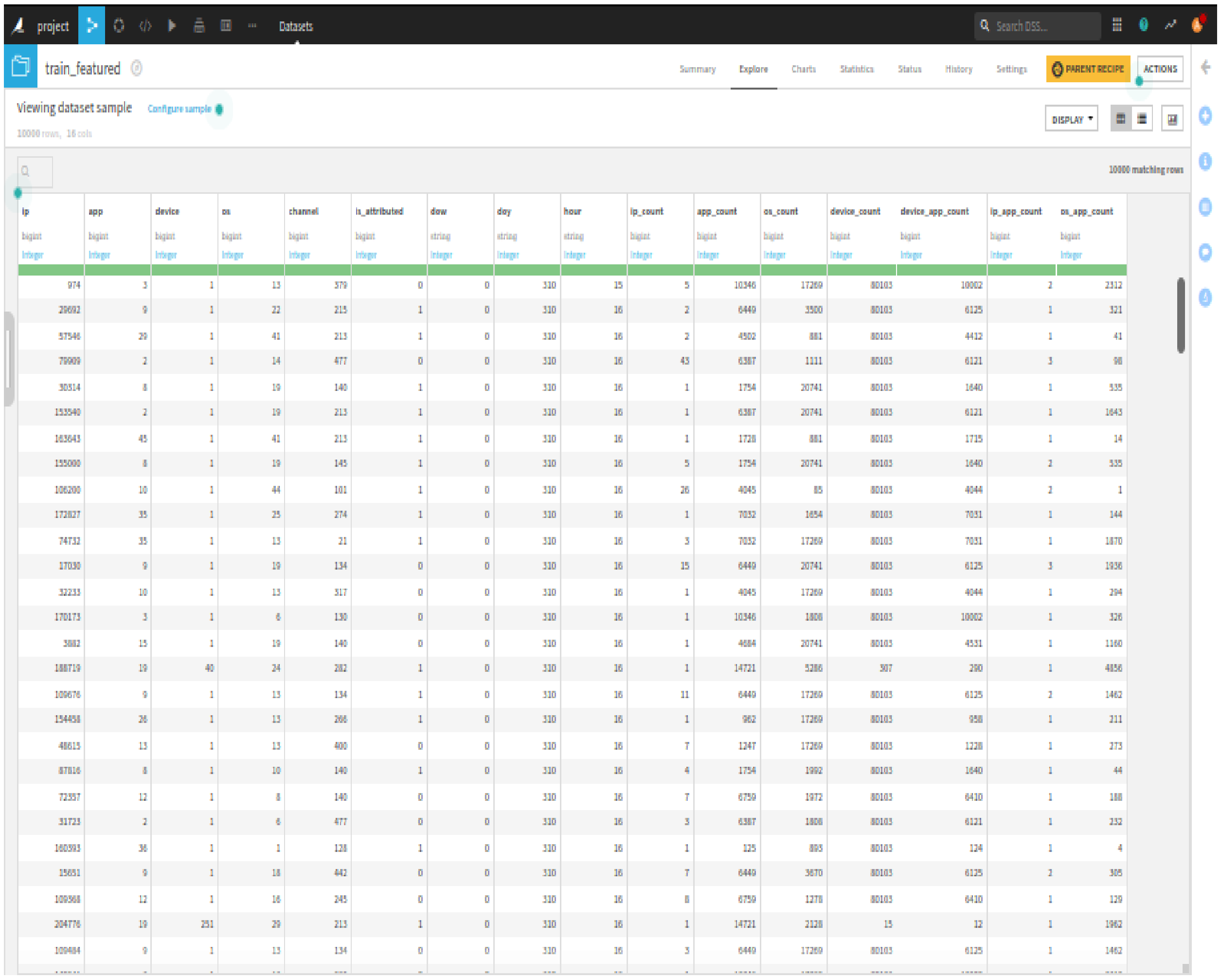
```
# -*- coding: utf-8 -*-
import dataiku
import pandas as pd, numpy as np
from dataiku import pandasutils as pdu

# Read recipe inputs
train_filtered = dataiku.Dataset("train_filtered")
train_filtered_df = train_filtered.get_dataframe()


# Compute recipe outputs from inputs
# TODO: Replace this part by your actual code that computes the output, as a Pandas dataframe
# NB: DSS also supports other kinds of APIs for reading and writing data. Please see doc.

train_featured_df = train_filtered_df
# For this sample code, simply copy input to output
#picking day of year
train_featured_df['click_time']  =pd.to_datetime(train_featured_df['click_time'])
train_featured_df['dow'] = train_featured_df['click_time'].dt.dayofweek.astype('uint16')
```



```
train_featured_df['click_time']  =pd.to_datetime(train_featured_df['click_time'])
train_featured_df['dow'] = train_featured_df['click_time'].dt.dayofweek.astype('uint16')
train_featured_df['doy'] = train_featured_df['click_time'].dt.dayofyear.astype('uint16')
train_featured_df['hour'] = train_featured_df['click_time'].dt.hour.astype('uint16')
#combining ip click count
temp=train_featured_df[['ip','channel']].groupby('ip').count().reset_index().rename(columns={'channe
train_featured_df=train_featured_df.merge(temp, on='ip', how='left')
#combining app click count
temp=train_featured_df[['app','channel']].groupby('app').count().reset_index().rename(columns={'ch
train_featured_df=train_featured_df.merge(temp, on='app', how='left')
#combining os click count
temp=train_featured_df[['os','channel']].groupby('os').count().reset_index().rename(columns={'chan
train_featured_df=train_featured_df.merge(temp, on='os', how='left')
#combining device click count
temp=train_featured_df[['device','channel']].groupby('device').count().reset_index().rename(columns=
train_featured_df=train_featured_df.merge(temp, on='device', how='left')
#combining device app click count
temp=train_featured_df[['device','app','channel']].groupby(['device','app']).count().reset_index().
train_featured_df= train_featured_df.merge(temp, on=['device','app'], how='left')
```


```python
train_featured_df=train_featured_df.merge(temp, on='device', how='left')
#combining device app click count
temp=train_featured_df[['device','app','channel']].groupby(['device','app']).count().reset_index().
train_featured_df= train_featured_df.merge(temp, on=['device','app'], how='left')
#combining ip app click count
temp=train_featured_df[['ip', 'app', 'channel']].groupby(['ip', 'app']).count().reset_index().rename
train_featured_df= train_featured_df.merge(temp, on=['ip','app'], how='left')
#combining os app click count
temp=train_featured_df[['os','app','channel']].groupby(['os','app']).count().reset_index().rename(c
train_featured_df=train_featured_df.merge(temp, on=['os','app'], how='left')
#dropping click_time
train_featured_df.drop(['click_time','attributed_time'],axis=1,inplace=True)


# Write recipe outputs
train_featured = dataiku.Dataset("train_featured")
train_featured.write_with_schema(train_featured_df)

```

After running this code the dataset looked like:-


Viewing dataset sample

10000 rows, 16 cols

10000 matching rows

ip	app	device	os	channel	is_attributed	dow	day	hour	ip_count	app_count	os_count	device_count	device_app_count	ip_app_count	os_app_count
bigint	bigint	bigint	bigint	bigint	bigint	string	string	string	bigint	bigint	bigint	bigint	bigint	bigint	bigint
Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer	Integer
974	3	1	13	379	0	0	310	15	5	10346	17269	80103	10002	2	2312
29692	9	1	22	215	1	0	310	16	2	6449	3500	80103	6125	1	321
57546	29	1	41	213	1	0	310	16	2	4502	881	80103	4412	1	41
79909	2	1	14	477	0	0	310	16	43	6387	1111	80103	6121	3	98
30314	8	1	19	140	1	0	310	16	1	1754	20741	80103	1640	1	535
153540	2	1	19	213	1	0	310	16	1	6387	20741	80103	6121	1	1643
163643	45	1	41	213	1	0	310	16	1	1728	881	80103	1715	1	14
155000	8	1	19	145	1	0	310	16	5	1754	20741	80103	1640	2	535
106200	10	1	44	101	1	0	310	16	26	4045	85	80103	4044	2	1
172827	35	1	25	274	1	0	310	16	1	7032	1654	80103	7031	1	144
74732	35	1	13	21	1	0	310	16	3	7032	17269	80103	7031	1	1870
17030	9	1	19	134	0	0	310	16	15	6449	20741	80103	6125	3	1936
32233	10	1	13	317	0	0	310	16	1	4045	17269	80103	4044	1	294
170173	3	1	6	130	0	0	310	16	1	10346	1808	80103	10002	1	326
3882	15	1	19	140	0	0	310	16	1	4684	20741	80103	4531	1	1160
188719	19	40	24	282	1	0	310	16	1	14721	5286	307	290	1	4856
109676	9	1	13	134	1	0	310	16	11	6449	17269	80103	6125	2	1462
154458	26	1	13	266	1	0	310	16	1	962	17269	80103	958	1	211
48615	13	1	13	400	0	0	310	16	7	1247	17269	80103	1228	1	273
87816	8	1	10	140	1	0	310	16	4	1754	1992	80103	1640	1	44
72357	12	1	8	140	0	0	310	16	7	6759	1972	80103	6410	1	188
31723	2	1	6	477	0	0	310	16	3	6387	1808	80103	6121	1	232
160593	36	1	1	128	1	0	310	16	1	125	893	80103	124	1	4
15651	9	1	18	442	0	0	310	16	7	6449	3870	80103	6125	2	305
109368	12	1	16	245	0	0	310	16	8	6759	1278	80103	6410	1	129
204776	19	251	29	213	1	0	310	16	1	14721	2128	15	12	1	1962
109484	9	1	13	134	0	0	310	16	3	6449	17269	80103	6125	1	1462

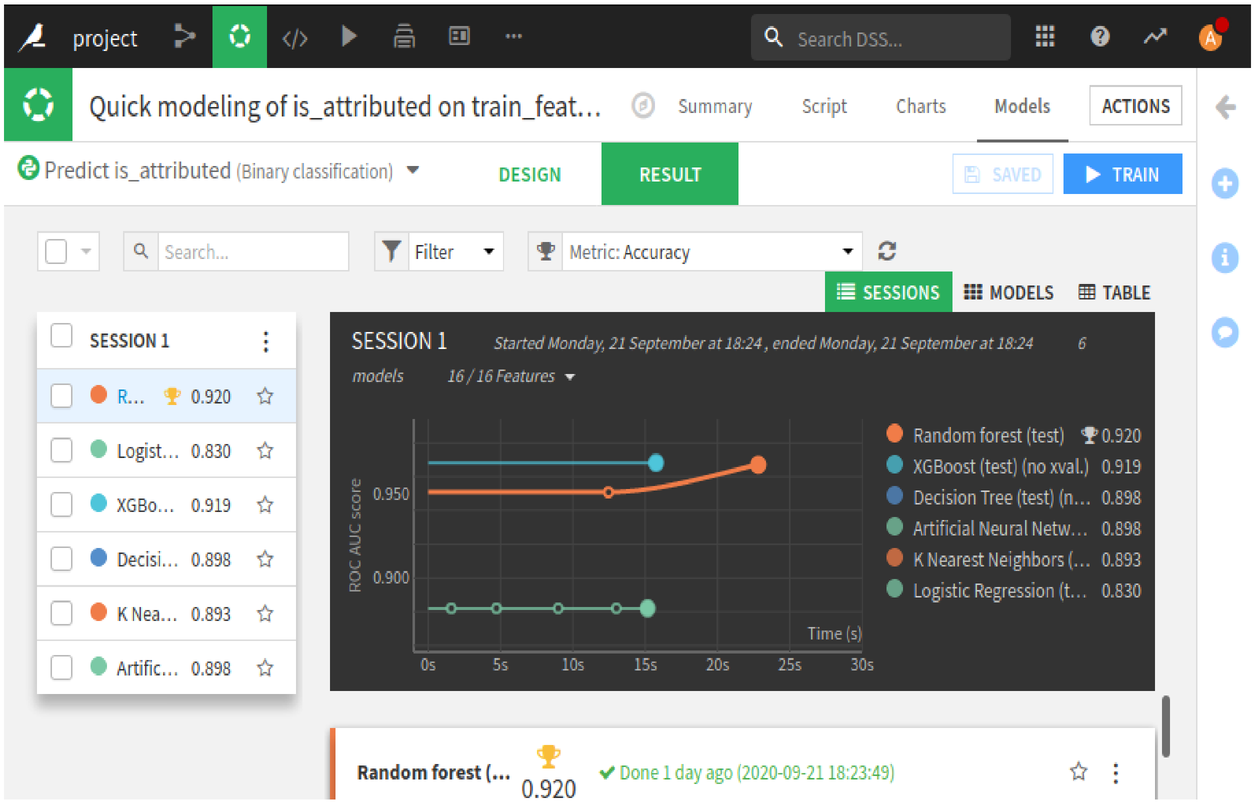
This dataset was now ready to be served to ML models.

Choosing Model

For building model, we took the advantage of Dataiku DSS auto ML capabilities. We chose 6 different classifiers for building our models which are:-

- Random Forest Classifier
- Logistic Regression
- XGBoost Classifier
- Decision Tree
- K Nearest Neighbor
- Artificial Neural Network

The training looked like:-



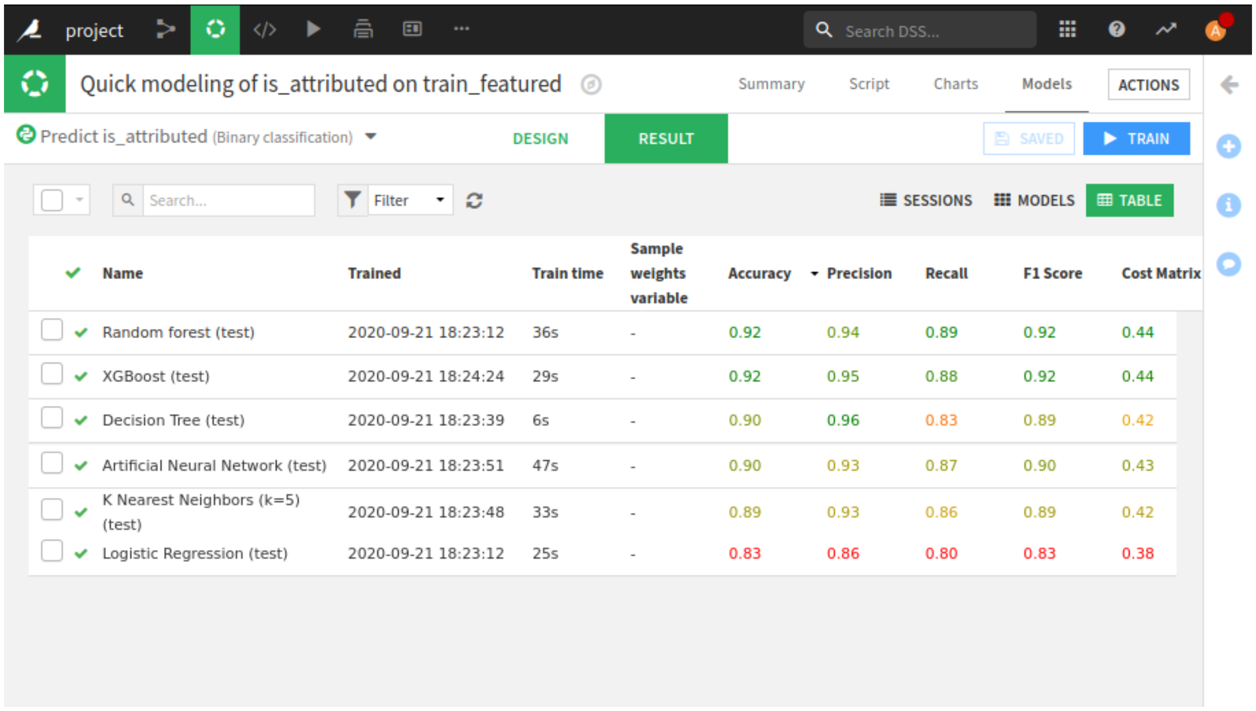
SESSION 1 Started Monday, 21 September at 18:24, ended Monday, 21 September at 18:24 6 models 16 / 16 Features

R...	0.920
Logist...	0.830
XGBo...	0.919
Decisi...	0.898
K Nea...	0.893
Artific...	0.898

● Random forest (test)	0.920
● XGBoost (test) (no xval.)	0.919
● Decision Tree (test) (n...	0.898
● Artificial Neural Netw...	0.898
● K Nearest Neighbors (...	0.893
● Logistic Regression (t...	0.830

Random forest (... 0.920 ✓ Done 1 day ago (2020-09-21 18:23:49)

We got the best results for Random Forest Classifier, detailed metrics of all the classifiers as:-

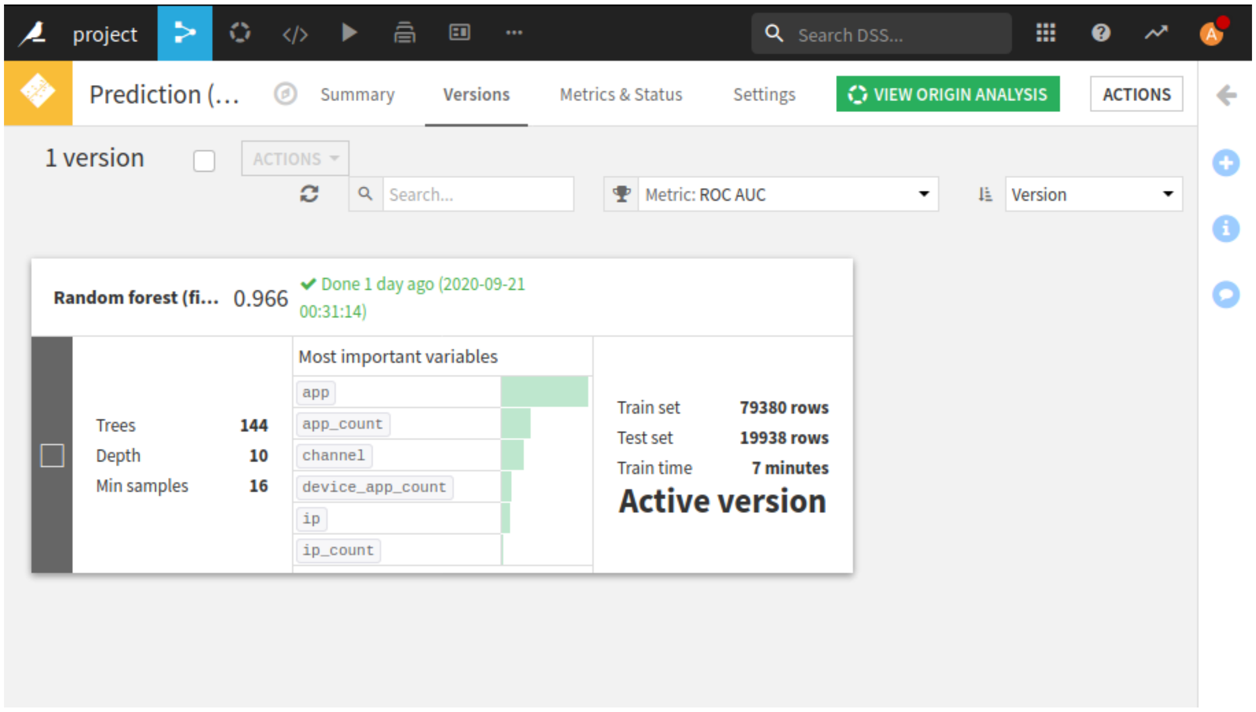


Quick modeling of is_attributed on train_featured

Predict is_attributed (Binary classification)

	Name	Trained	Train time	Sample weights variable	Accuracy	Precision	Recall	F1 Score	Cost Matrix
✓	Random forest (test)	2020-09-21 18:23:12	36s	-	0.92	0.94	0.89	0.92	0.44
✓	XGBoost (test)	2020-09-21 18:24:24	29s	-	0.92	0.95	0.88	0.92	0.44
✓	Decision Tree (test)	2020-09-21 18:23:39	6s	-	0.90	0.96	0.83	0.89	0.42
✓	Artificial Neural Network (test)	2020-09-21 18:23:51	47s	-	0.90	0.93	0.87	0.90	0.43
✓	K Nearest Neighbors (k=5) (test)	2020-09-21 18:23:48	33s	-	0.89	0.93	0.86	0.89	0.42
✓	Logistic Regression (test)	2020-09-21 18:23:12	25s	-	0.83	0.86	0.80	0.83	0.38

So, we chose Random Forest Classifier for our final model.



1 version

Random forest (fi... 0.966 ✓ Done 1 day ago (2020-09-21 00:31:14)

Trees	144
Depth	10
Min samples	16

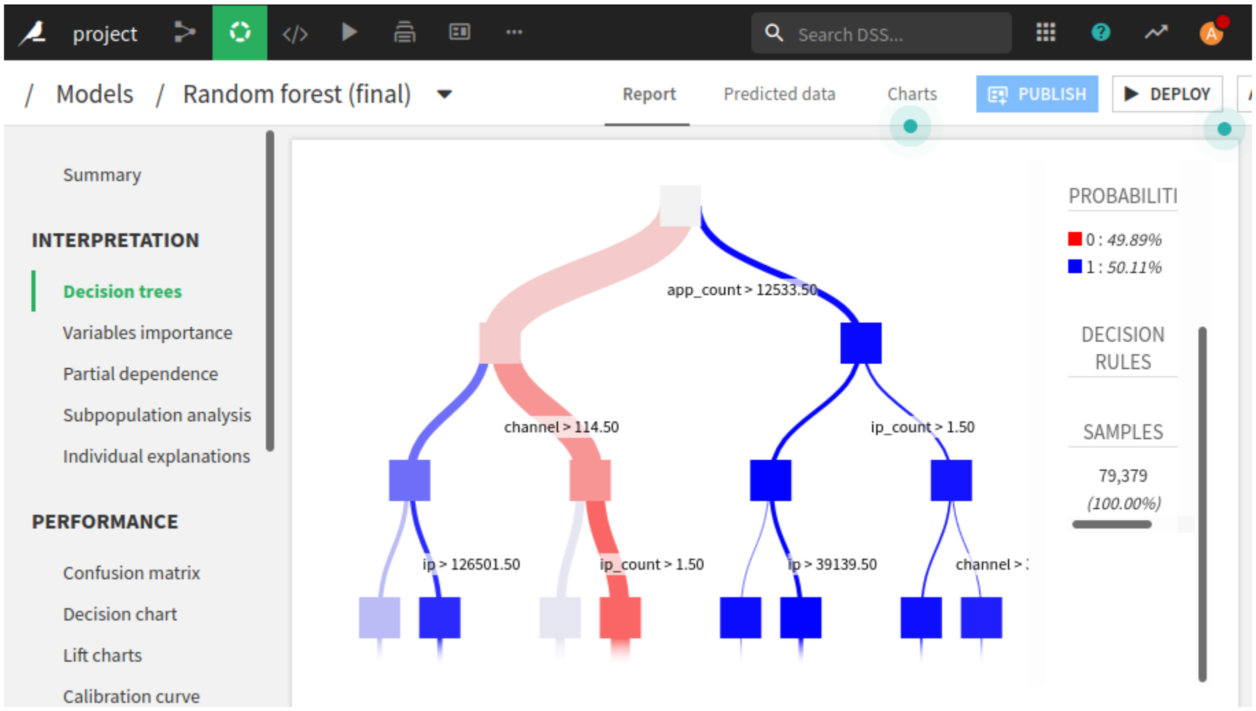
Most important variables

app
app_count
channel
device_app_count
ip
ip_count

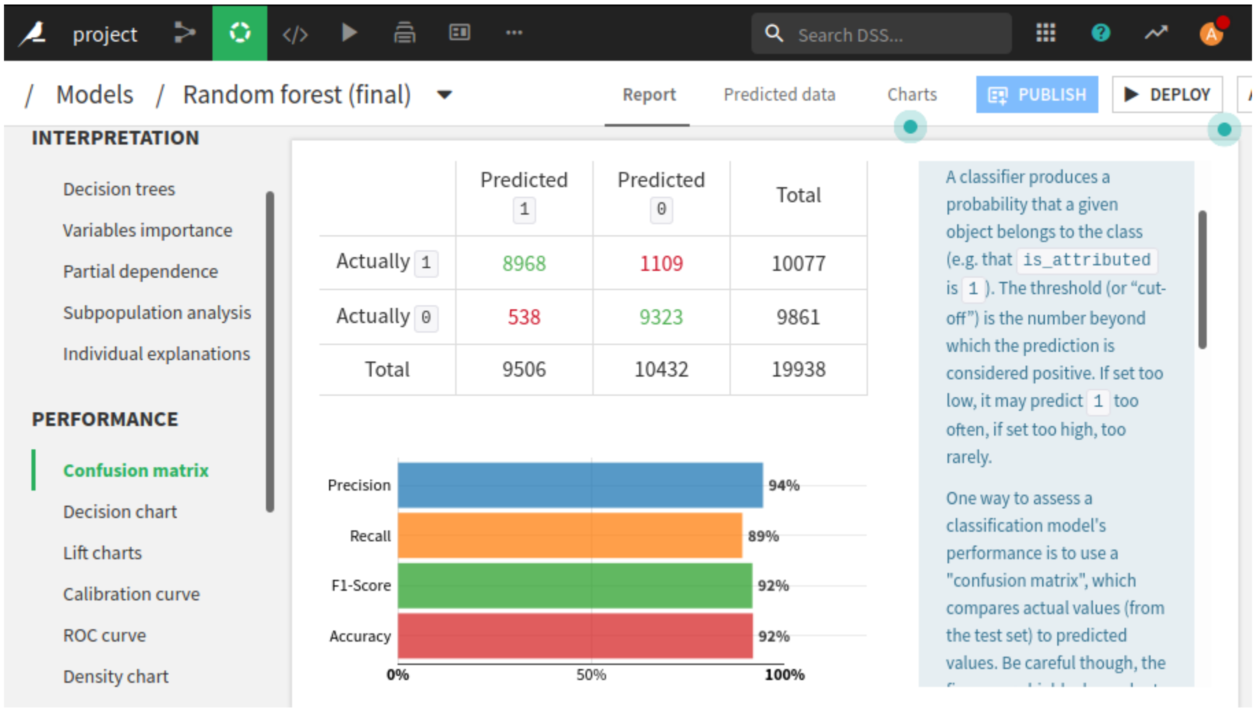
Train set	79380 rows
Test set	19938 rows
Train time	7 minutes

Active version

Random Forest Model Decision Tree:-



Confusion Matrix and Detailed Metrics:-



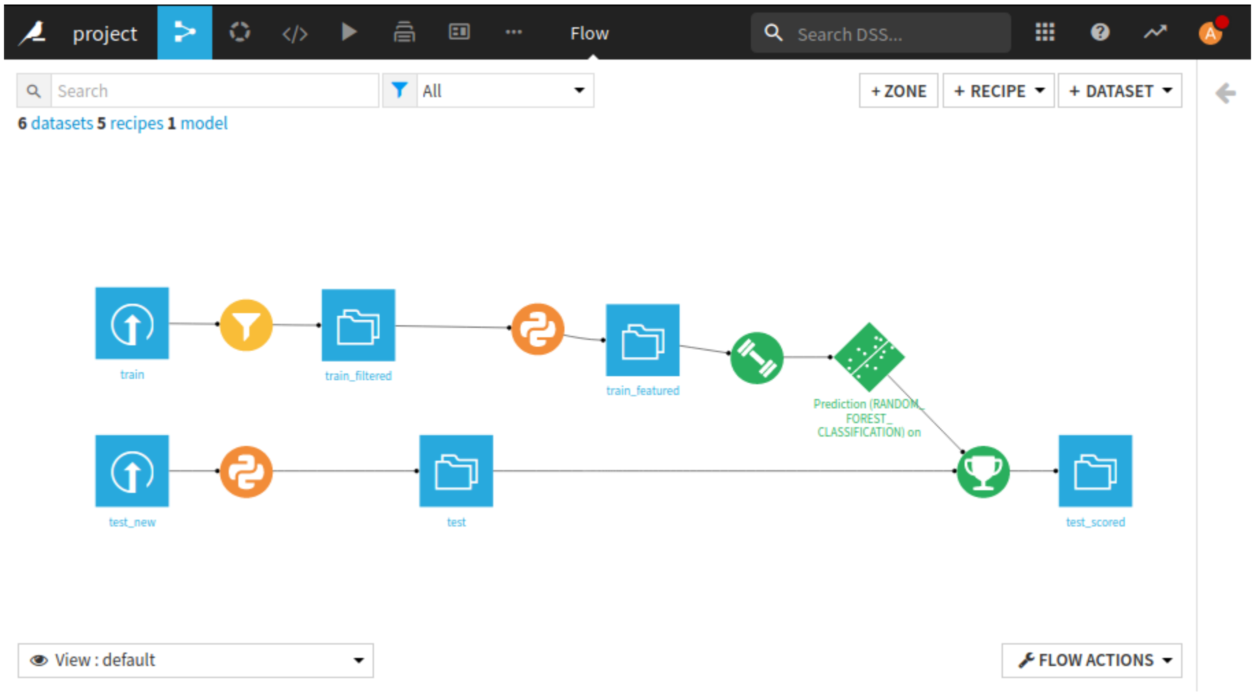
	Predicted 1	Predicted 0	Total
Actually 1	8968	1109	10077
Actually 0	538	9323	9861
Total	9506	10432	19938

A classifier produces a probability that a given object belongs to the class (e.g. that `is_attributed` is `1`). The threshold (or "cut-off") is the number beyond which the prediction is considered positive. If set too low, it may predict `1` too often, if set too high, too rarely.

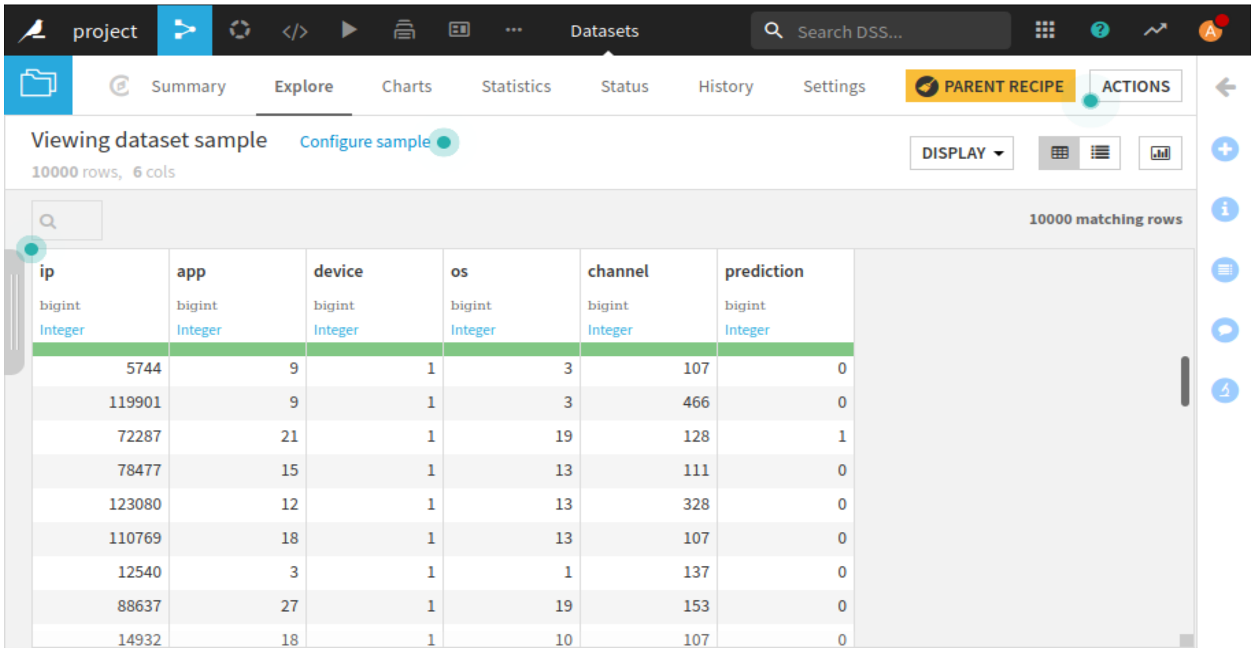
One way to assess a classification model's performance is to use a "confusion matrix", which compares actual values (from the test set) to predicted values. Be careful though, the

Final Flow

Our final flow after test predictions looked like this:-



Test Dataset predictions:-



Viewing dataset sample Configure sample

10000 rows, 6 cols

10000 matching rows

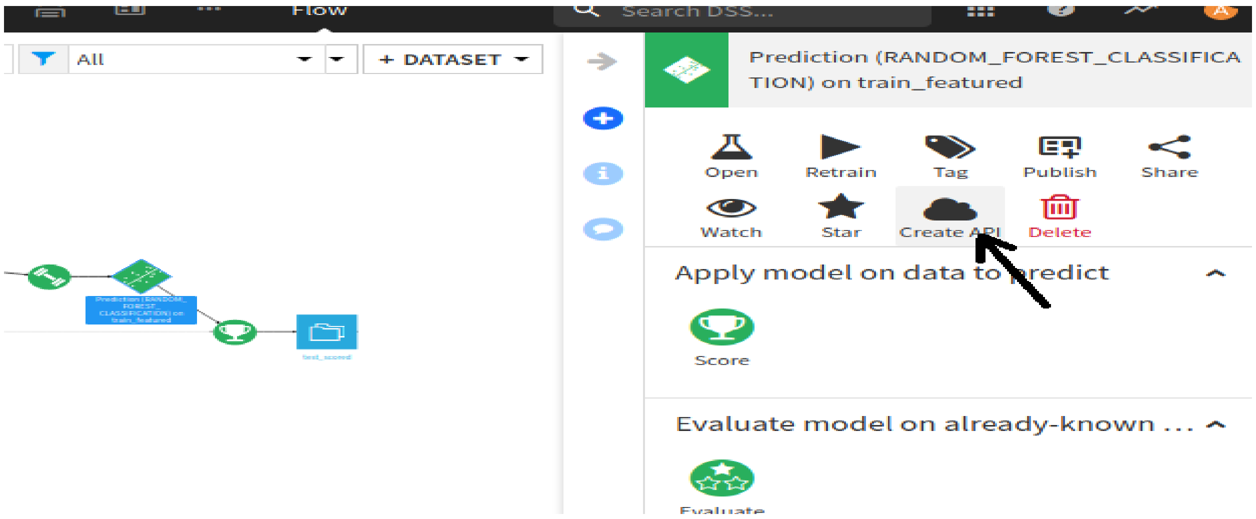
ip bigint Integer	app bigint Integer	device bigint Integer	os bigint Integer	channel bigint Integer	prediction bigint Integer
5744	9	1	3	107	0
119901	9	1	3	466	0
72287	21	1	19	128	1
78477	15	1	13	111	0
123080	12	1	13	328	0
110769	18	1	13	107	0
12540	3	1	1	137	0
88637	27	1	19	153	0
14932	18	1	10	107	0

How to use the model as an API?

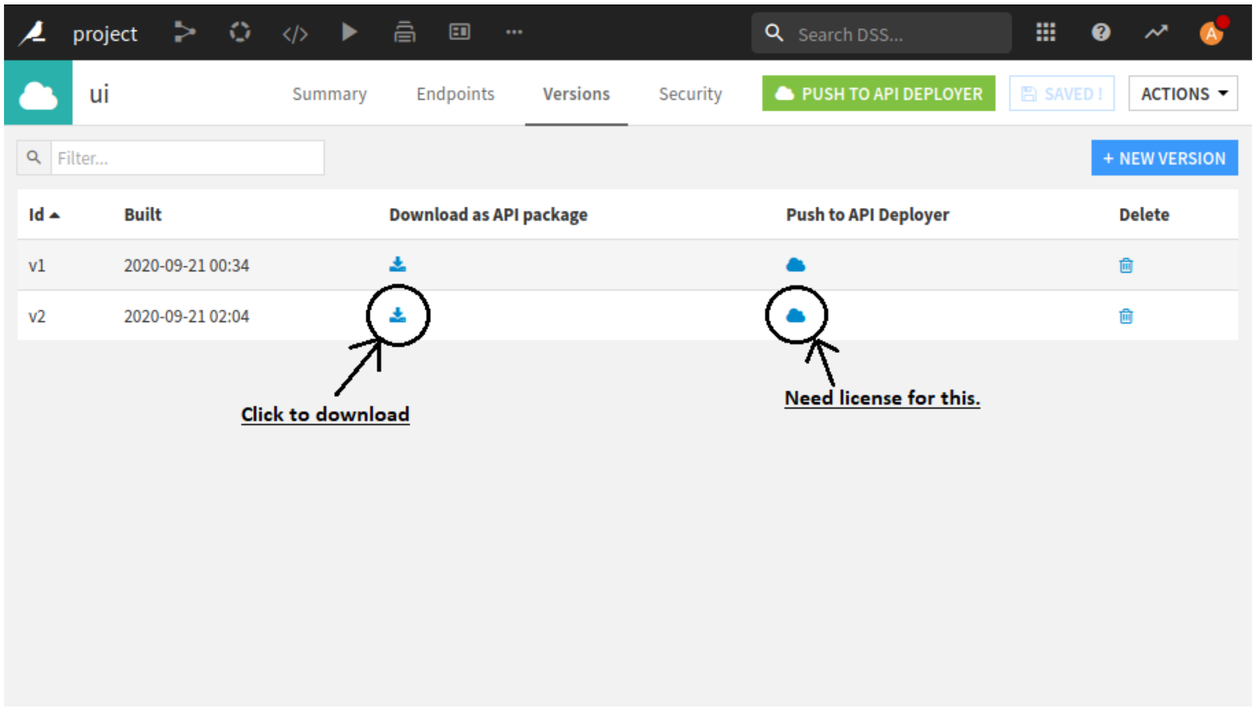
For using Dataiku API capabilities we must have DSS License. So, we had to figure out a way through which we could use this model for predictions outside Dataiku DSS.

Steps involved were:-

1. Click the create API option.

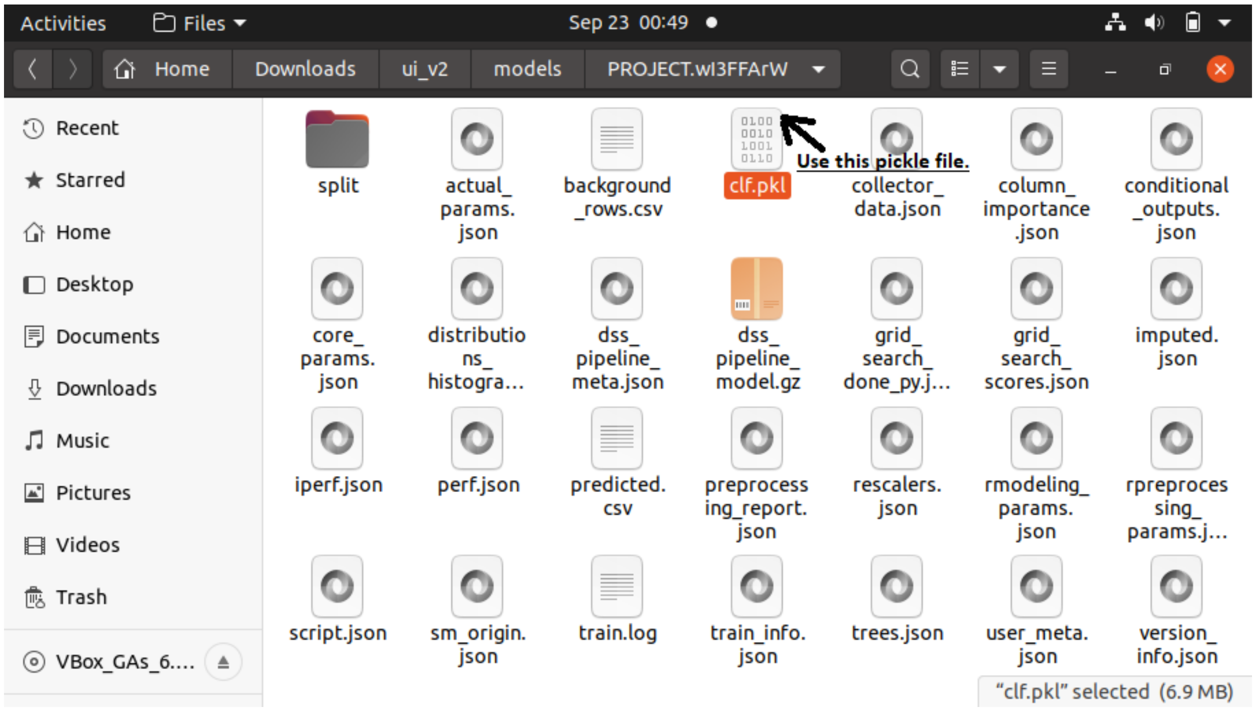

Flow
Search DSS...
All
+ DATASET
Prediction (RANDOM_FOREST_CLASSIFICATION) on train_featured
Open
Retrain
Tag
Publish
Share
Watch
Star
Create API
Delete
Apply model on data to predict
Score
Evaluate model on already-known ...
Evaluate

2. Download it as an API package; we could not use API deployer because of license limitations.



Id	Built	Download as API package	Push to API Deployer	Delete
v1	2020-09-21 00:34			
v2	2020-09-21 02:04			

3. After downloading it, we found a pickle file. This file can be used to make predictions for the test dataset outside DSS.



Flask API Documentation

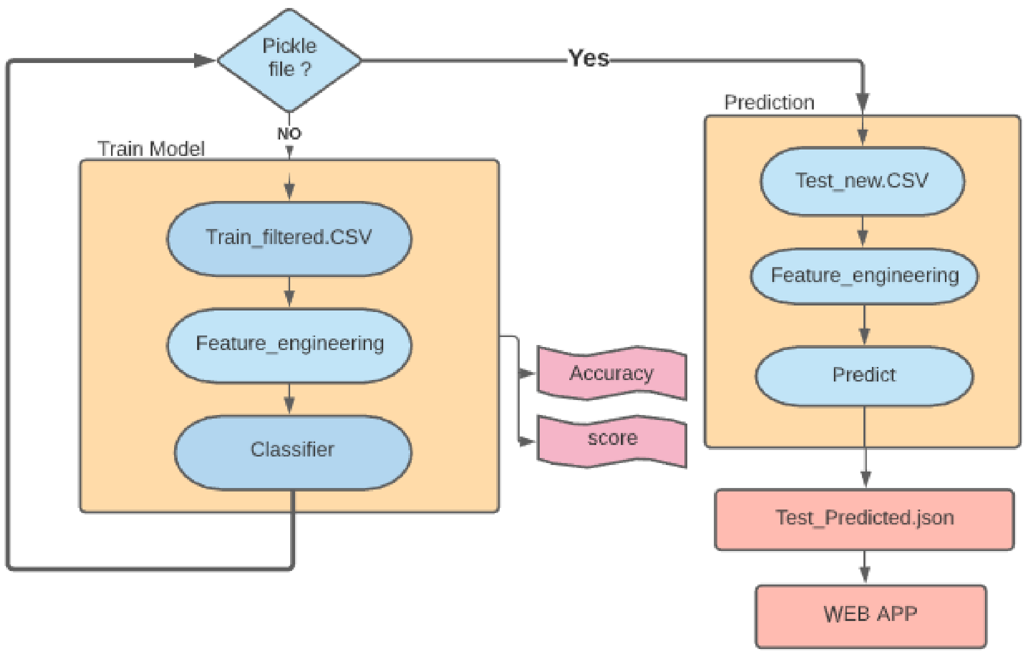
Introduction

Flask is a micro web framework which has a minimal footprint compared to others like Django and allows us to select which components and modules we want to integrate with it. Flask is highly reliable and performant.

The Flask Restful was chosen on the suggestion of Mentor when our team was unable to get an API from Dataiku

Project Layout

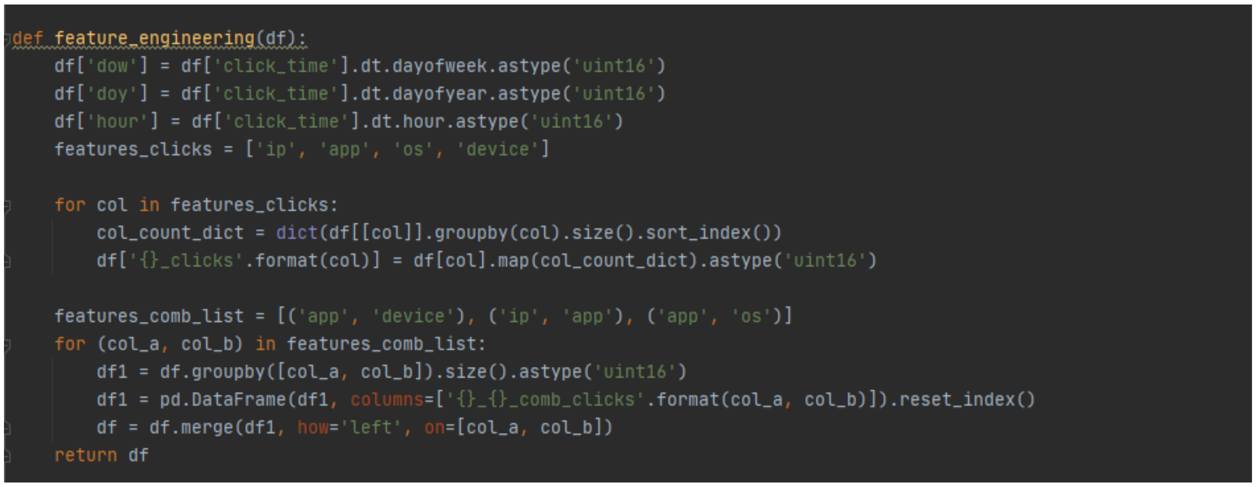
Project layout looks like the following folder structure.



API module will host our application code, from models-

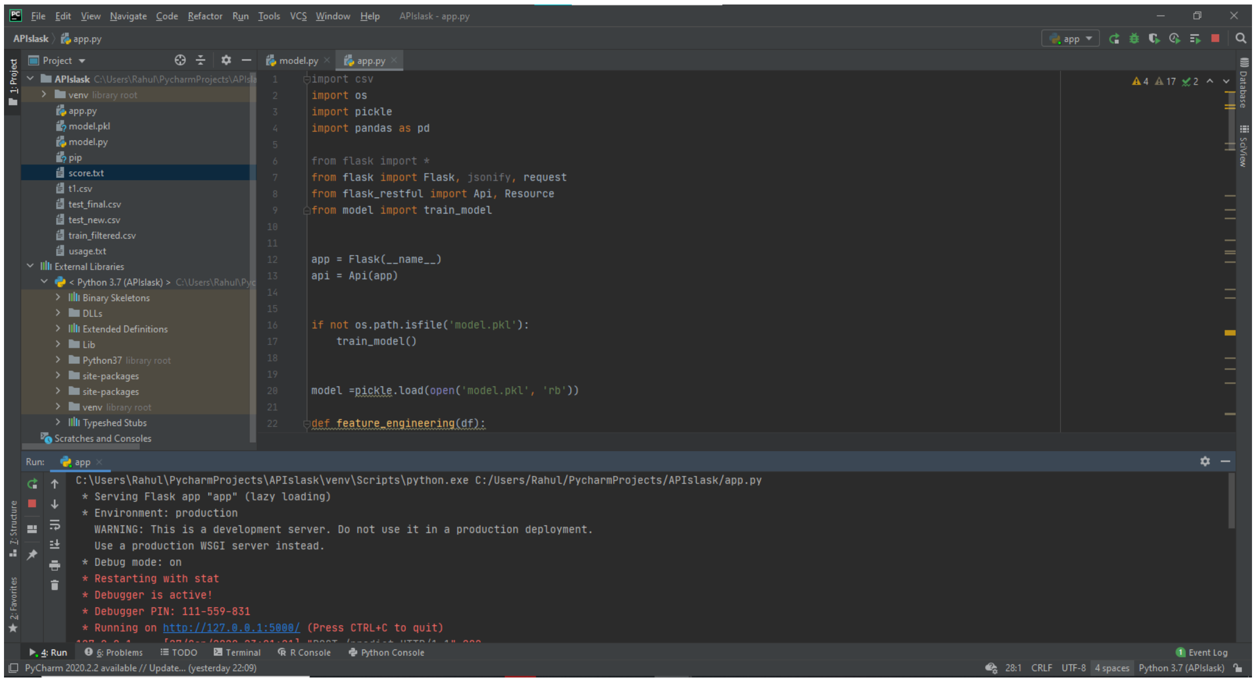
- App.py serves as the main file and hosts the server whereas models.py serves as the backend for the training.

- App.py has 3 classes each serving separate calls to provide a different response to calling application based on request
- Train_filtered.csv is the dataset used for training the module
- Test_new.csv is created from the received post request
- Usage and Scores file contains the data about the system resources



```python
def feature_engineering(df):
    df['dow'] = df['click_time'].dt.dayofweek.astype('uint16')
    df['doy'] = df['click_time'].dt.dayofyear.astype('uint16')
    df['hour'] = df['click_time'].dt.hour.astype('uint16')
    features_clicks = ['ip', 'app', 'os', 'device']

    for col in features_clicks:
        col_count_dict = dict(df[[col]].groupby(col).size().sort_index())
        df['{}_clicks'.format(col)] = df[col].map(col_count_dict).astype('uint16')

    features_comb_list = [('app', 'device'), ('ip', 'app'), ('app', 'os')]
    for (col_a, col_b) in features_comb_list:
        df1 = df.groupby([col_a, col_b]).size().astype('uint16')
        df1 = pd.DataFrame(df1, columns=['{}_{}_comb_clicks'.format(col_a, col_b)]).reset_index()
        df = df.merge(df1, how='left', on=[col_a, col_b])
    return df
```

Fig - Feature Engineering based on findings from Dataiku to get best results for prediction

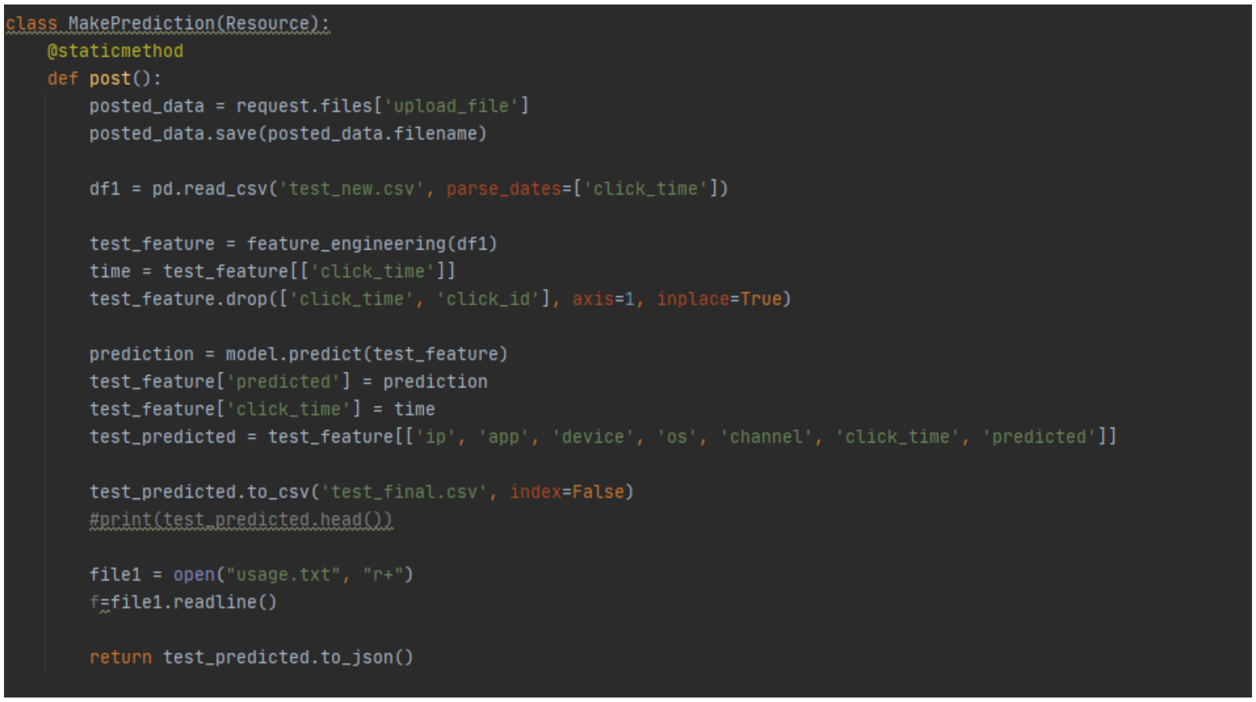


File Edit View Navigate Code Refactor Run Tools VCS Window Help APIslask - app.py
APIslask app.py
Project
model.py app.py
APIslask C:\Users\Rahul\PycharmProjects\APIsla
venv library root
app.py
model.pkl
model.py
pip
score.txt
t1.csv
test_final.csv
test_new.csv
train_filtered.csv
usage.txt
External Libraries
< Python 3.7 (APIslask) > C:\Users\Rahul\Pyc
Binary Skeletons
DLLs
Extended Definitions
Lib
Python37 library root
site-packages
site-packages
venv library root
Typeshed Stubs
Scratches and Consoles
Run: app
4: Run 6: Problems TODO Terminal R Console Python Console Event Log
PyCharm 2020.2.2 available // Update... (yesterday 22:09)
28:1 CRLF UTF-8 4 spaces Python 3.7 (APIslask)

```python
import csv
import os
import pickle
import pandas as pd

from flask import *
from flask import Flask, jsonify, request
from flask_restful import Api, Resource
from model import train_model


app = Flask(__name__)
api = Api(app)


if not os.path.isfile('model.pkl'):
    train_model()


model =pickle.load(open('model.pkl', 'rb'))

def feature_engineering(df):
```

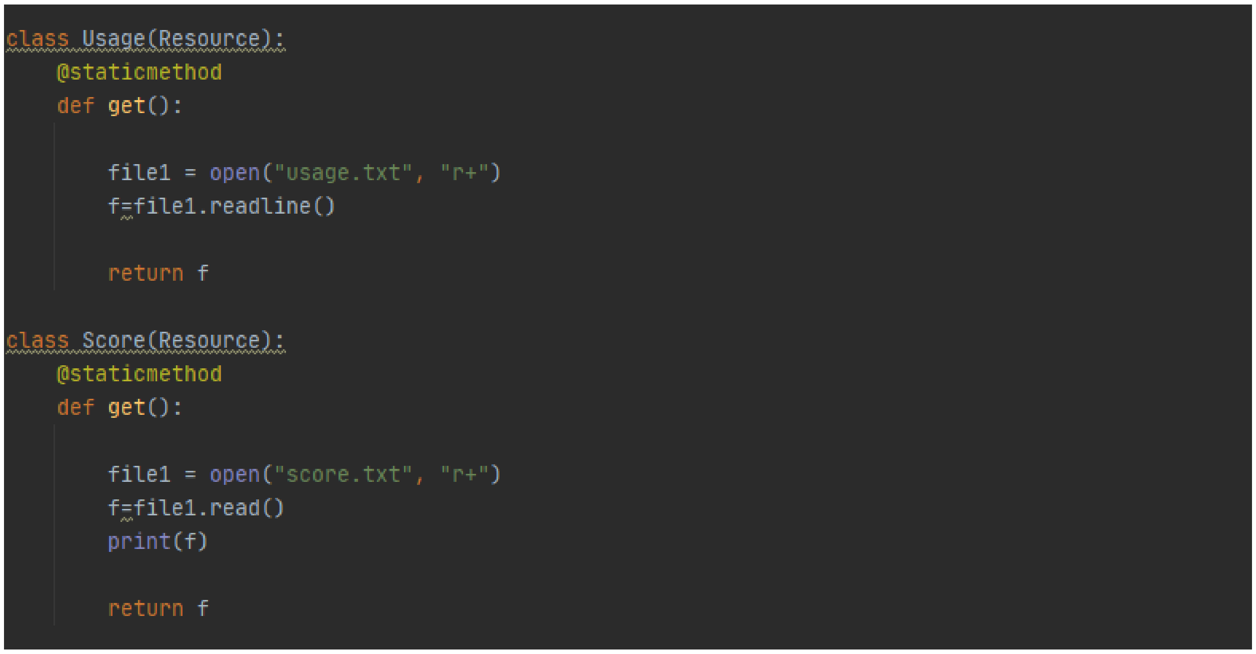
```
C:\Users\Rahul\PycharmProjects\APIslask\venv\Scripts\python.exe C:/Users/Rahul/PycharmProjects/APIslask/app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 111-559-831
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig – Api.py window while the server is running



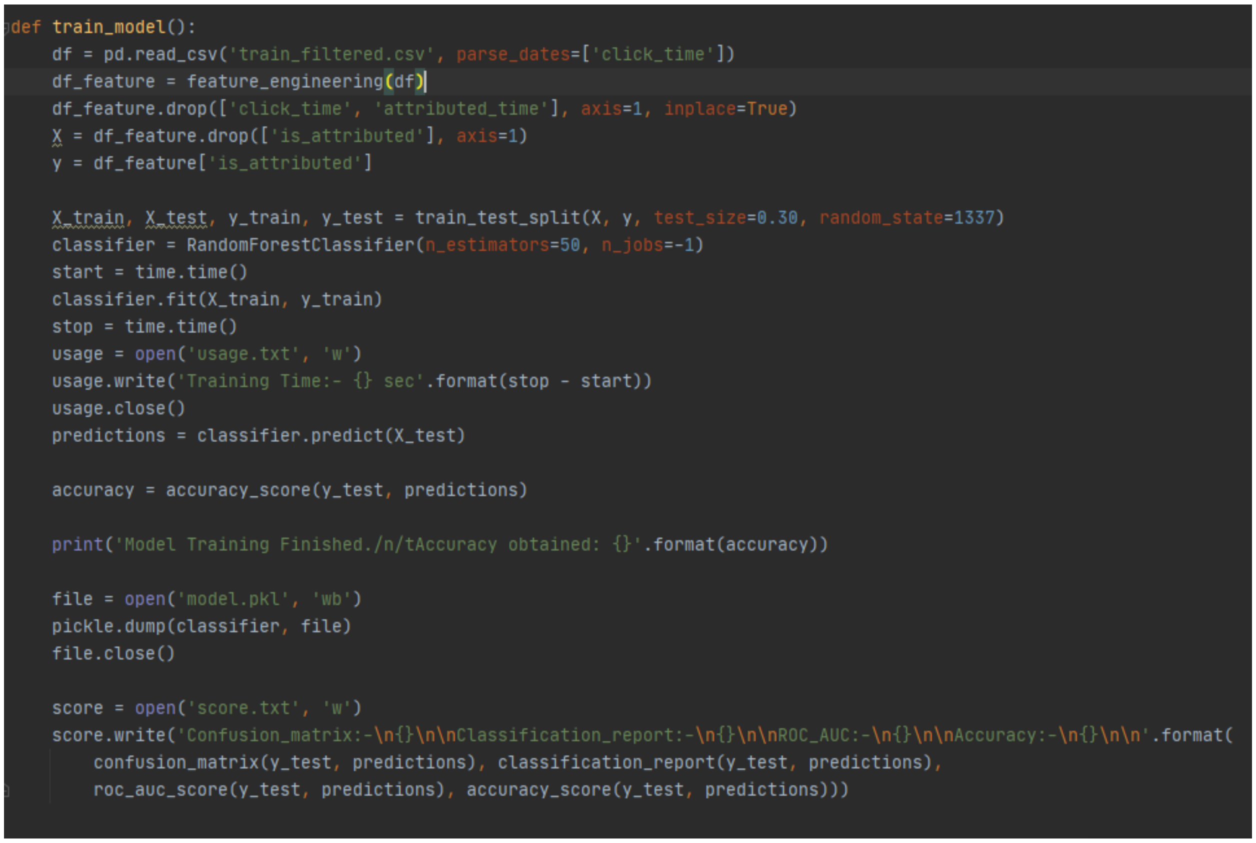
```python
class MakePrediction(Resource):
    @staticmethod
    def post():
        posted_data = request.files['upload_file']
        posted_data.save(posted_data.filename)

        df1 = pd.read_csv('test_new.csv', parse_dates=['click_time'])

        test_feature = feature_engineering(df1)
        time = test_feature[['click_time']]
        test_feature.drop(['click_time', 'click_id'], axis=1, inplace=True)

        prediction = model.predict(test_feature)
        test_feature['predicted'] = prediction
        test_feature['click_time'] = time
        test_predicted = test_feature[['ip', 'app', 'device', 'os', 'channel', 'click_time', 'predicted']]

        test_predicted.to_csv('test_final.csv', index=False)
        #print(test_predicted.head())

        file1 = open("usage.txt", "r+")
        f=file1.readline()

        return test_predicted.to_json()
```

Fig – Api.py – Make Prediction class which takes the incoming file from post request convert it to CSV and perform predictions and then return file as a json file



```python
class Usage(Resource):
    @staticmethod
    def get():

        file1 = open("usage.txt", "r+")
        f=file1.readline()

        return f

class Score(Resource):
    @staticmethod
    def get():

        file1 = open("score.txt", "r+")
        f=file1.read()
        print(f)

        return f
```

Fig – Api.py – Usage and Score class which return test files created during training n response to GET request

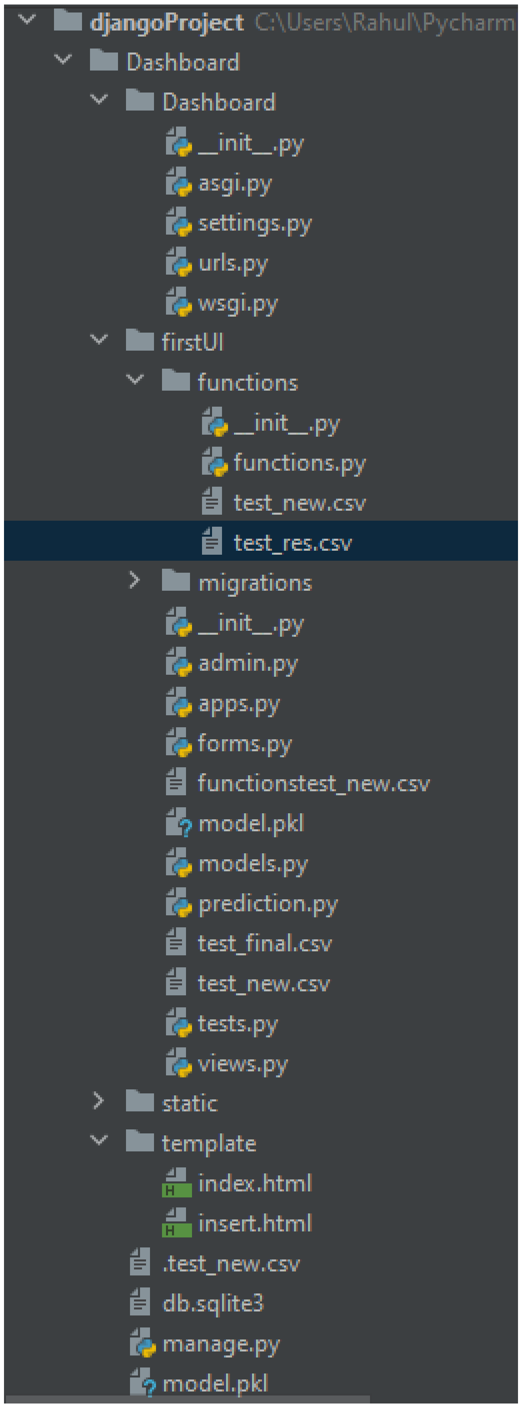


```
def train_model():
    df = pd.read_csv('train_filtered.csv', parse_dates=['click_time'])
    df_feature = feature_engineering(df)
    df_feature.drop(['click_time', 'attributed_time'], axis=1, inplace=True)
    X = df_feature.drop(['is_attributed'], axis=1)
    y = df_feature['is_attributed']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1337)
    classifier = RandomForestClassifier(n_estimators=50, n_jobs=-1)
    start = time.time()
    classifier.fit(X_train, y_train)
    stop = time.time()
    usage = open('usage.txt', 'w')
    usage.write('Training Time:- {} sec'.format(stop - start))
    usage.close()
    predictions = classifier.predict(X_test)

    accuracy = accuracy_score(y_test, predictions)

    print('Model Training Finished./n/tAccuracy obtained: {}'.format(accuracy))

    file = open('model.pkl', 'wb')
    pickle.dump(classifier, file)
    file.close()

    score = open('score.txt', 'w')
    score.write('Confusion_matrix:-\n{}\n\nClassification_report:-\n{}\n\nROC_AUC:-\n{}\n\nAccuracy:-\n{}\n\n'.format(
        confusion_matrix(y_test, predictions), classification_report(y_test, predictions),
        roc_auc_score(y_test, predictions), accuracy_score(y_test, predictions)))
```

Fig – Model.py train_model function which trains based on the random forest algorithm and creates a pickle file which would be used in the prediction

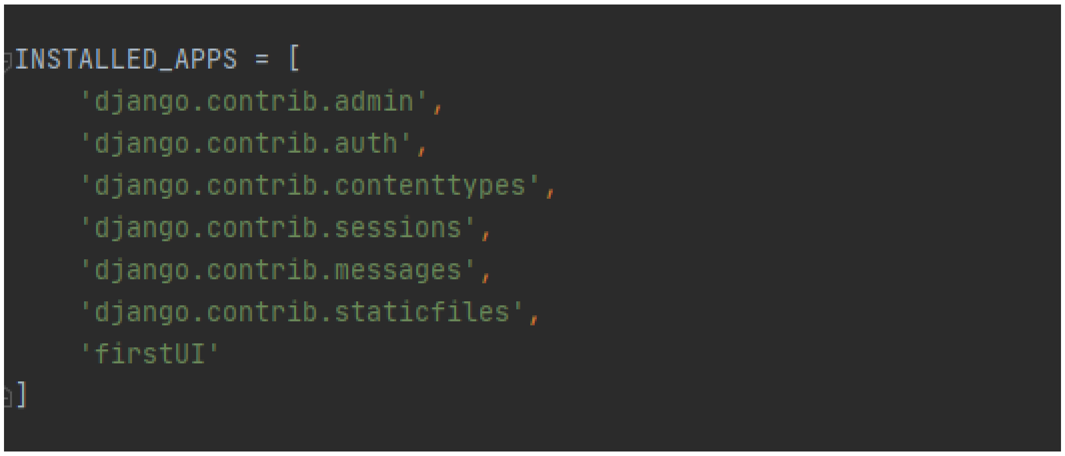
Dashboard

Django based web app to upload a CSV file and give visual analytics on the given file. The team was asked to build a Django based web app to give meaningful results to the user along with usage of system resource



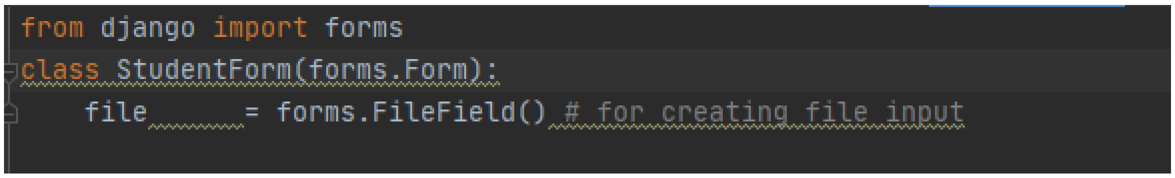
```
djangoProject C:\Users\Rahul\Pycharm
  Dashboard
    Dashboard
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
    firstUI
      functions
        __init__.py
        functions.py
        test_new.csv
        test_res.csv
      migrations
      __init__.py
      admin.py
      apps.py
      forms.py
      functionstest_new.csv
      model.pkl
      models.py
      prediction.py
      test_final.csv
      test_new.csv
      tests.py
      views.py
    static
    template
      index.html
      insert.html
    .test_new.csv
    db.sqlite3
    manage.py
    model.pkl
```

Fig – Directory structure of Django dashboard



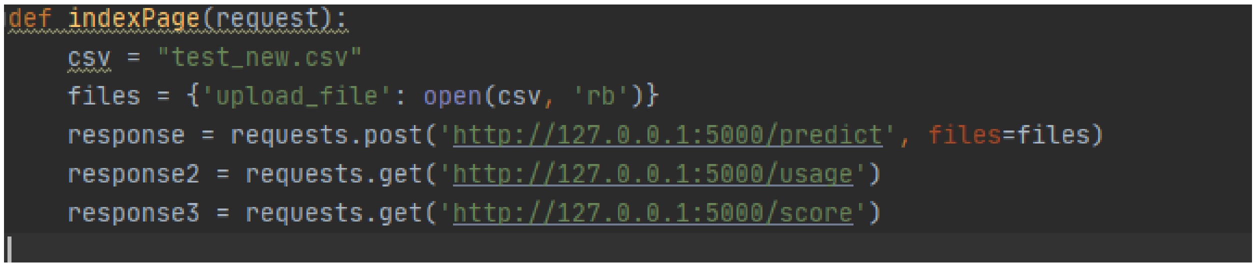
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'firstUI'
]
```

Fig – Adding the name of the UI in installed apps in settings.py



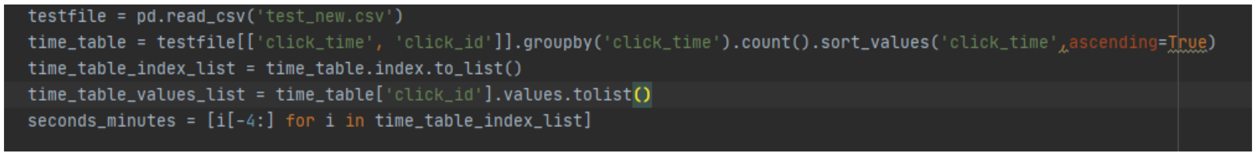
```
from django import forms
class StudentForm(forms.Form):
    file      = forms.FileField() # for creating file input
```

Fig – Forms.py



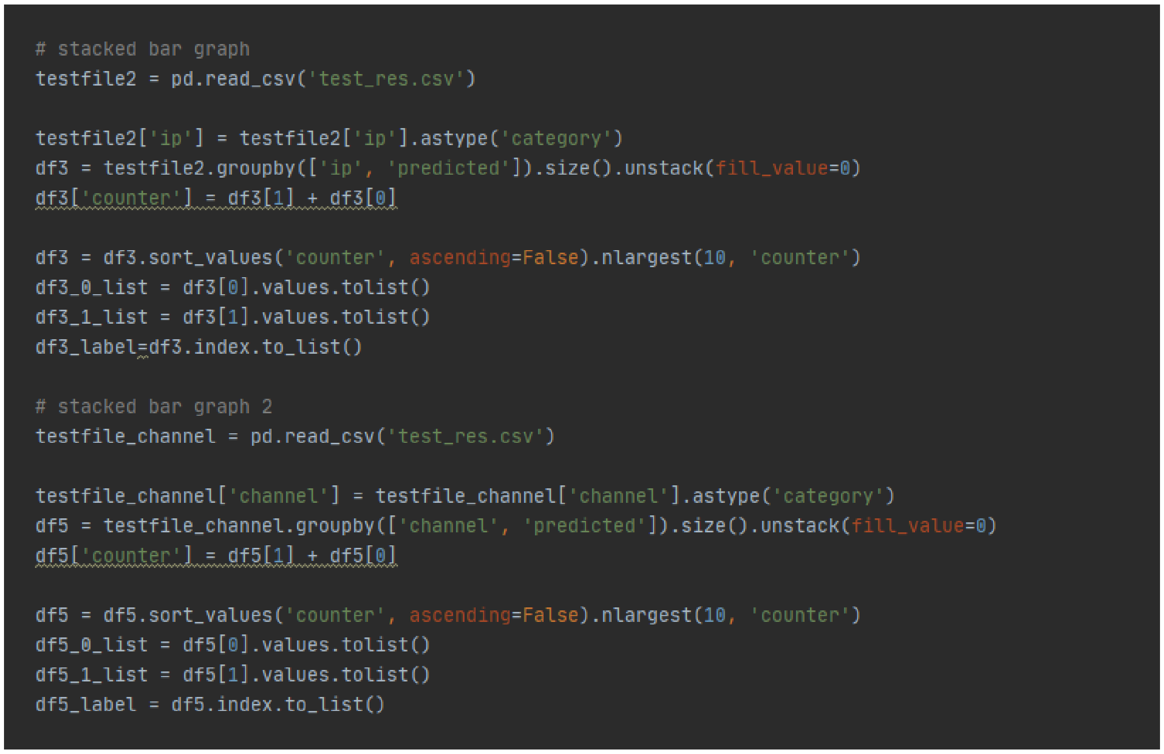
```
def indexPage(request):
    csv = "test_new.csv"
    files = {'upload_file': open(csv, 'rb')}
    response = requests.post('http://127.0.0.1:5000/predict', files=files)
    response2 = requests.get('http://127.0.0.1:5000/usage')
    response3 = requests.get('http://127.0.0.1:5000/score')
```

Fig – Views.py - Making POST and GET requests to the API serve and saving the response



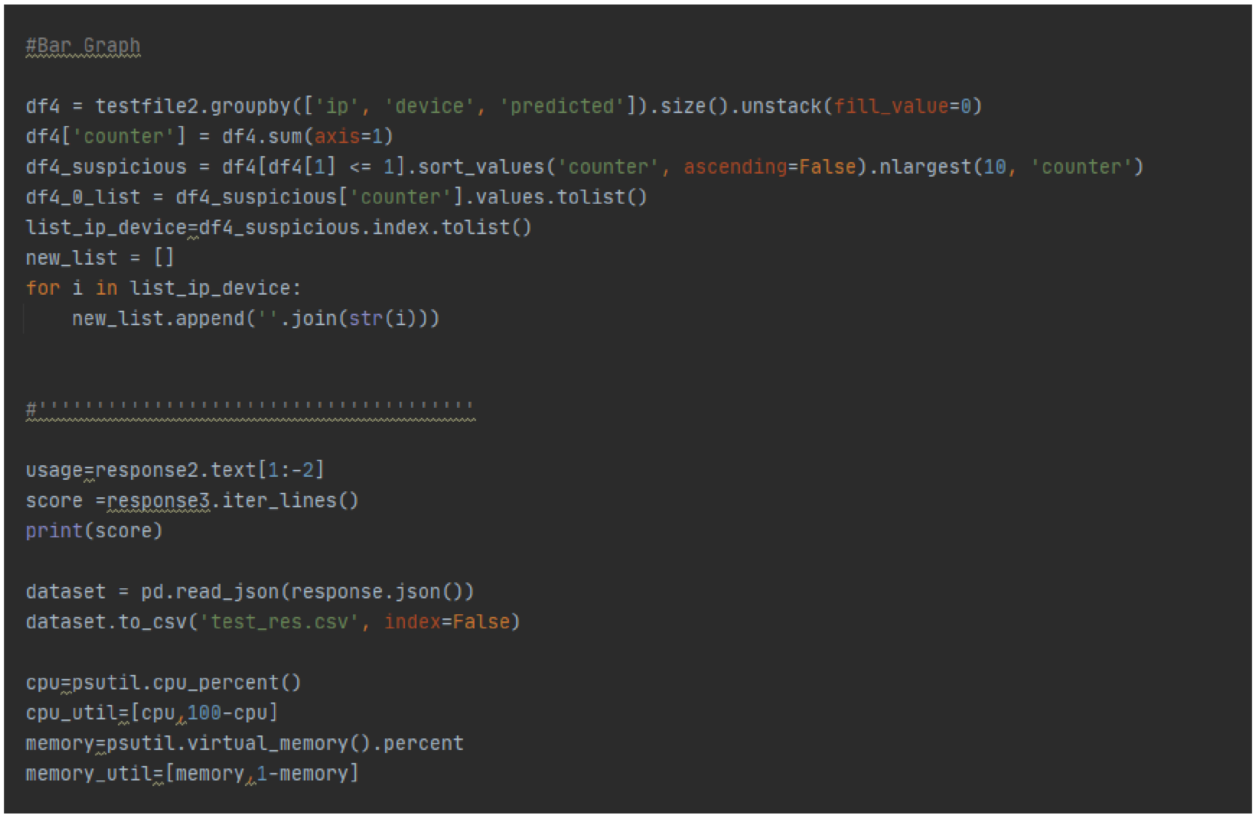
```
testfile = pd.read_csv('test_new.csv')
time_table = testfile[['click_time', 'click_id']].groupby('click_time').count().sort_values('click_time',ascending=True)
time_table_index_list = time_table.index.to_list()
time_table_values_list = time_table['click_id'].values.tolist()
seconds_minutes = [i[-4:] for i in time_table_index_list]
```

Fig – Reading the test file from insert.html and performing operations on it



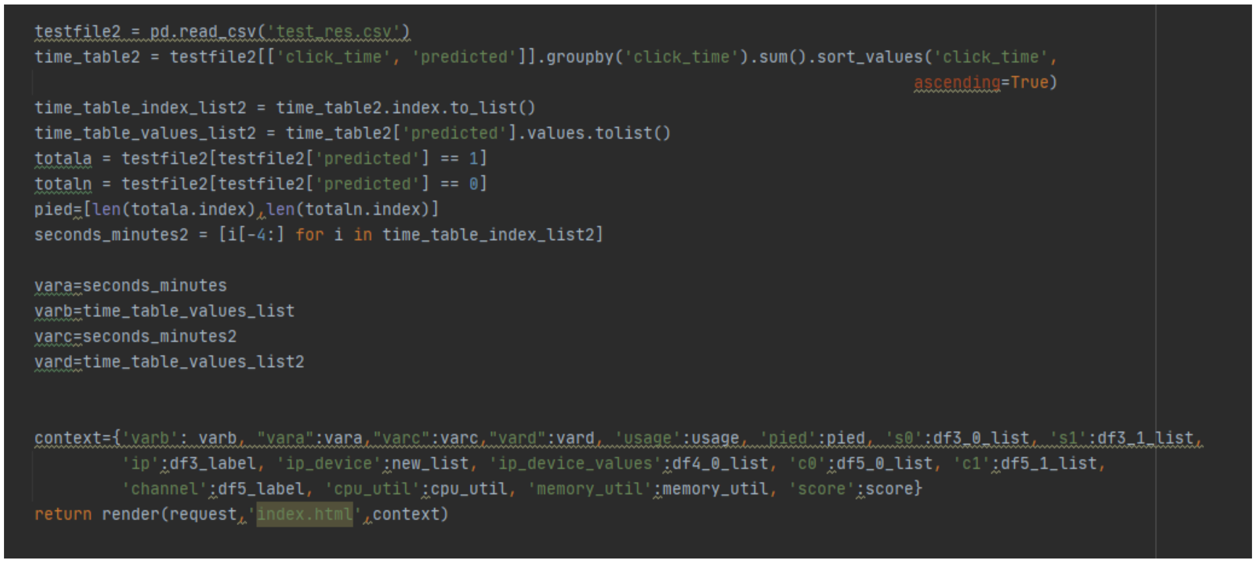
```
# stacked bar graph
testfile2 = pd.read_csv('test_res.csv')

testfile2['ip'] = testfile2['ip'].astype('category')
df3 = testfile2.groupby(['ip', 'predicted']).size().unstack(fill_value=0)
df3['counter'] = df3[1] + df3[0]

df3 = df3.sort_values('counter', ascending=False).nlargest(10, 'counter')
df3_0_list = df3[0].values.tolist()
df3_1_list = df3[1].values.tolist()
df3_label=df3.index.to_list()

# stacked bar graph 2
testfile_channel = pd.read_csv('test_res.csv')

testfile_channel['channel'] = testfile_channel['channel'].astype('category')
df5 = testfile_channel.groupby(['channel', 'predicted']).size().unstack(fill_value=0)
df5['counter'] = df5[1] + df5[0]

df5 = df5.sort_values('counter', ascending=False).nlargest(10, 'counter')
df5_0_list = df5[0].values.tolist()
df5_1_list = df5[1].values.tolist()
df5_label = df5.index.to_list()
```

Fig – Creating graphs from predicted files



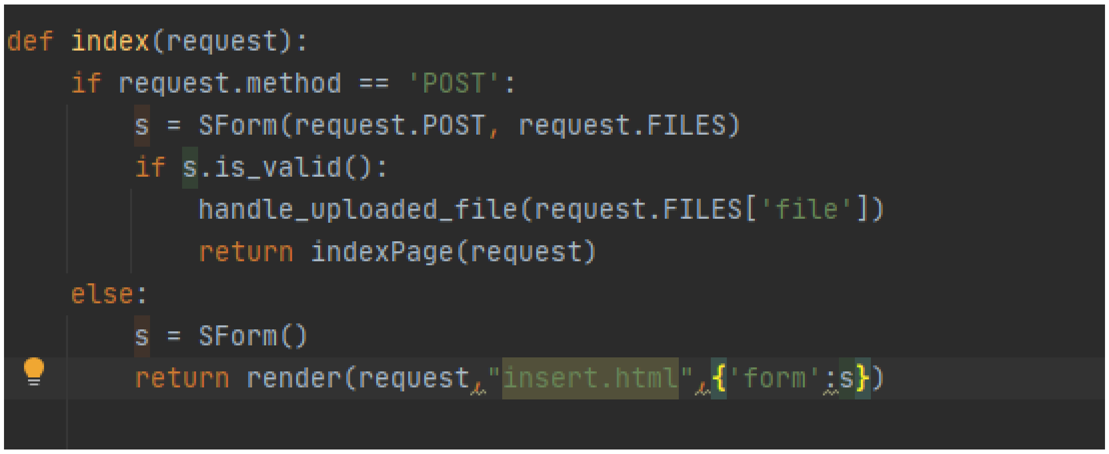
```
#Bar Graph

df4 = testfile2.groupby(['ip', 'device', 'predicted']).size().unstack(fill_value=0)
df4['counter'] = df4.sum(axis=1)
df4_suspicious = df4[df4[1] <= 1].sort_values('counter', ascending=False).nlargest(10, 'counter')
df4_0_list = df4_suspicious['counter'].values.tolist()
list_ip_device=df4_suspicious.index.tolist()
new_list = []
for i in list_ip_device:
    new_list.append(''.join(str(i)))


#''''''''''''''''''''''''''''''''''''''

usage=response2.text[1:-2]
score =response3.iter_lines()
print(score)

dataset = pd.read_json(response.json())
dataset.to_csv('test_res.csv', index=False)

cpu=psutil.cpu_percent()
cpu_util=[cpu,100-cpu]
memory=psutil.virtual_memory().percent
memory_util=[memory,1-memory]
```

Fig – Saving the values of resource utilization in variables



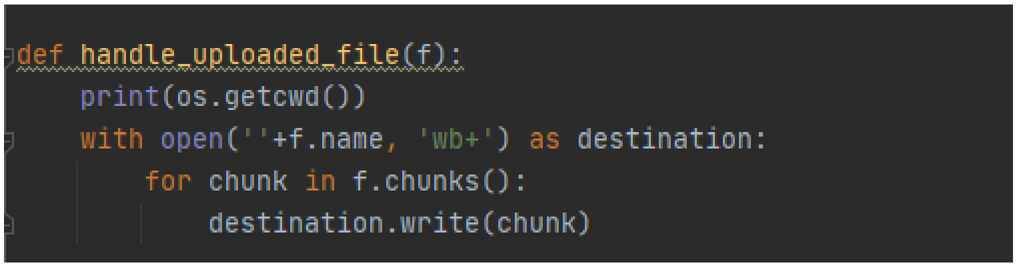
```python
testfile2 = pd.read_csv('test_res.csv')
time_table2 = testfile2[['click_time', 'predicted']].groupby('click_time').sum().sort_values('click_time',
                                                                                         ascending=True)
time_table_index_list2 = time_table2.index.to_list()
time_table_values_list2 = time_table2['predicted'].values.tolist()
totala = testfile2[testfile2['predicted'] == 1]
totaln = testfile2[testfile2['predicted'] == 0]
pied=[len(totala.index),len(totaln.index)]
seconds_minutes2 = [i[-4:] for i in time_table_index_list2]

vara=seconds_minutes
varb=time_table_values_list
varc=seconds_minutes2
vard=time_table_values_list2


context={'varb': varb, "vara":vara,"varc":varc,"vard":vard, 'usage':usage, 'pied':pied, 's0':df3_0_list, 's1':df3_1_list,
         'ip':df3_label, 'ip_device':new_list, 'ip_device_values':df4_0_list, 'c0':df5_0_list, 'c1':df5_1_list,
         'channel':df5_label, 'cpu_util':cpu_util, 'memory_util':memory_util, 'score':score}
return render(request,'index.html',context)
```

Fig – Passing various variables as context ad rendering the page



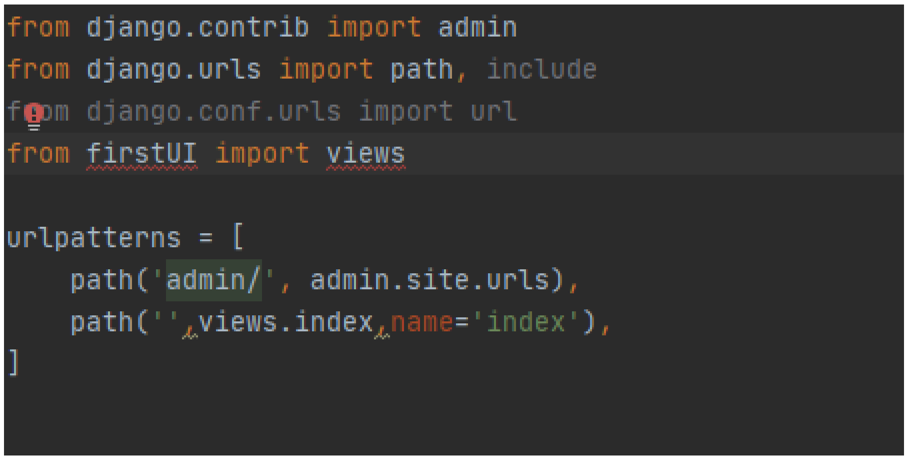
```python
def index(request):
    if request.method == 'POST':
        s = SForm(request.POST, request.FILES)
        if s.is_valid():
            handle_uploaded_file(request.FILES['file'])
            return indexPage(request)
    else:
        s = SForm()
    return render(request,"insert.html",{'form':s})
```

Fig – Create a form for the first page which accepts CSV file as input



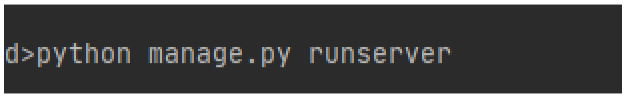
```python
def handle_uploaded_file(f):
    print(os.getcwd())
    with open(''+f.name, 'wb+') as destination:
        for chunk in f.chunks():
            destination.write(chunk)
```

Fig –functions.py – function to save the file as a CSV file on the server



```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls import url
from firstUI import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.index,name='index'),
]
```

Fig –url.py – urlpatterns save the name of files to be called



```
d>python manage.py runserver
```

Fig – terminal command to run the server

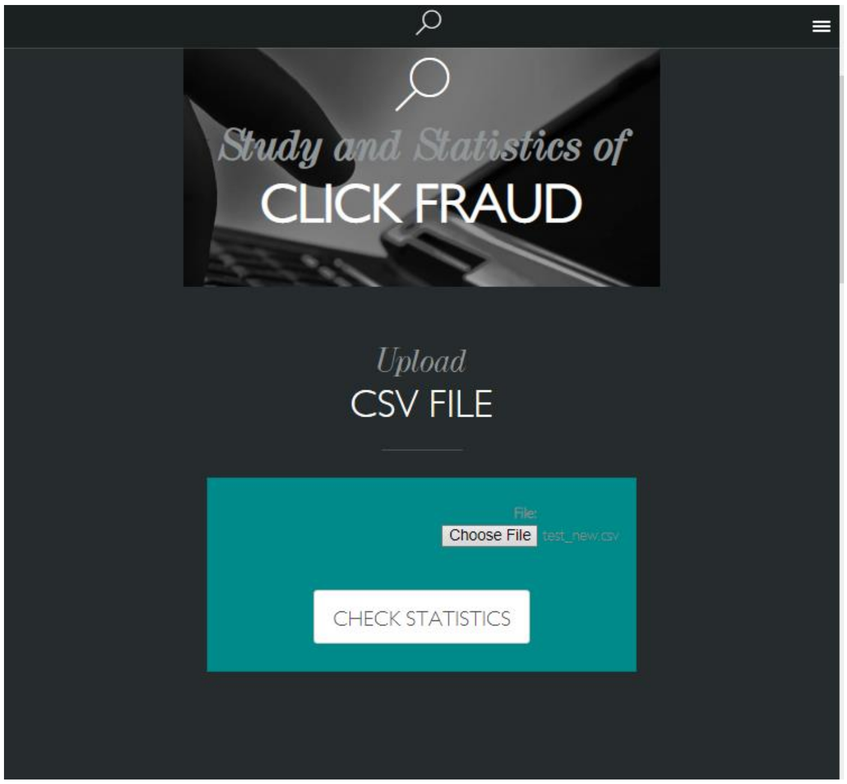
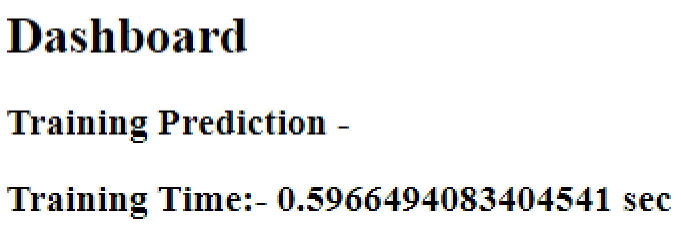


Fig – First page on opening the URL – it takes the CSV file as input

Data Visualization

The next page shows the training time for the model



Dashboard

Training Prediction -

Training Time:- 0.5966494083404541 sec

Fig –This part shows the training time required for the model

CPU Utilization

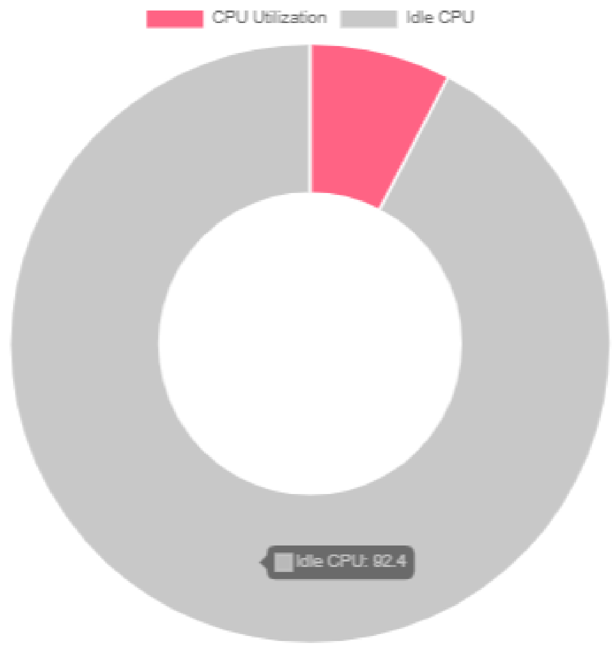


Fig. - This doughnut-shaped graph shows the CPU Utilization while running the server in percentage

Memory Utilization

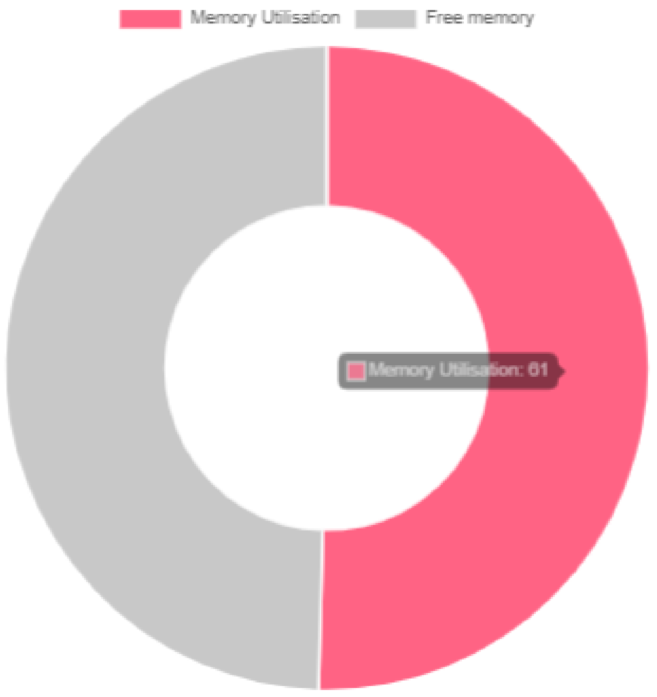


Fig - This doughnut shows the Memory Utilization while running the server in percentage

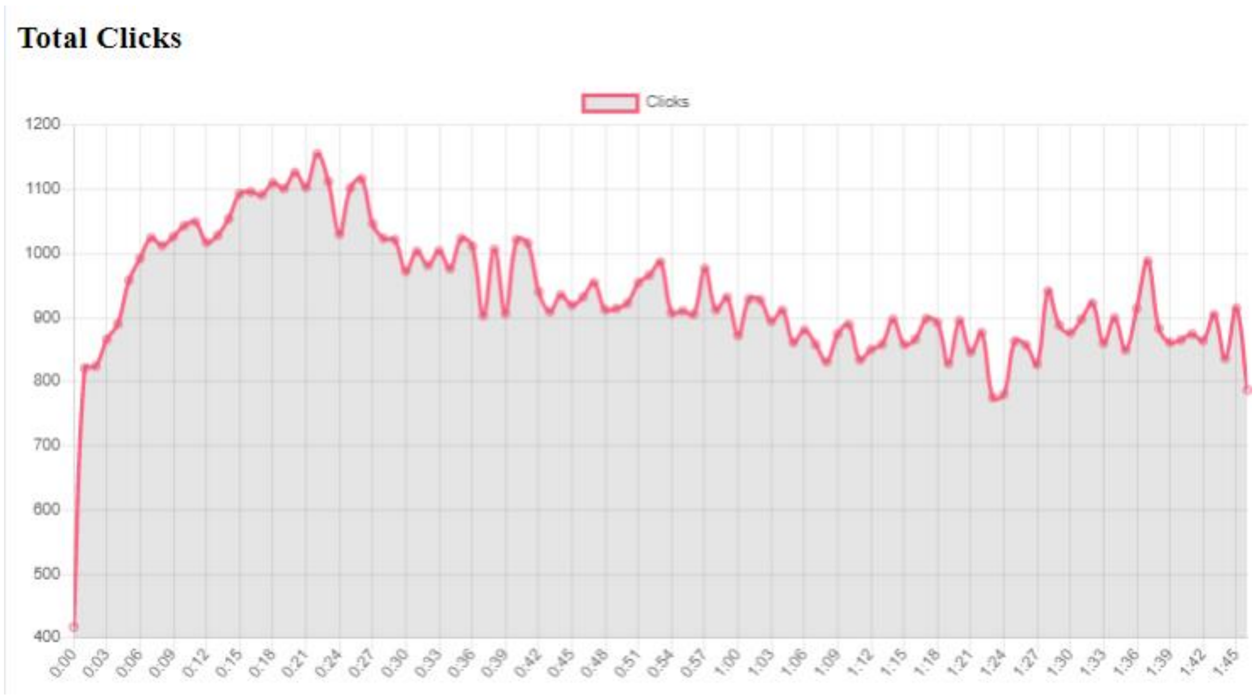


Fig – Graph showing the number of clicks with time on the x-axis

Predicted Attributed Clicks

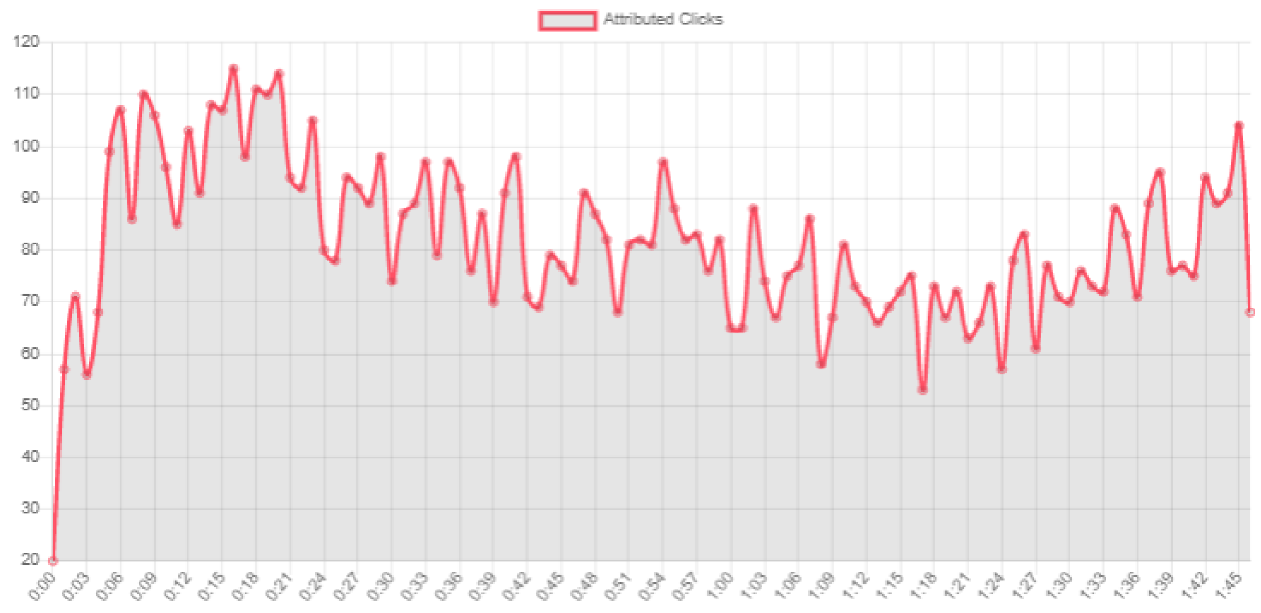


Fig - Number of predicted Attributions(app download)

Atrubuted vs Non-attributed Clicks

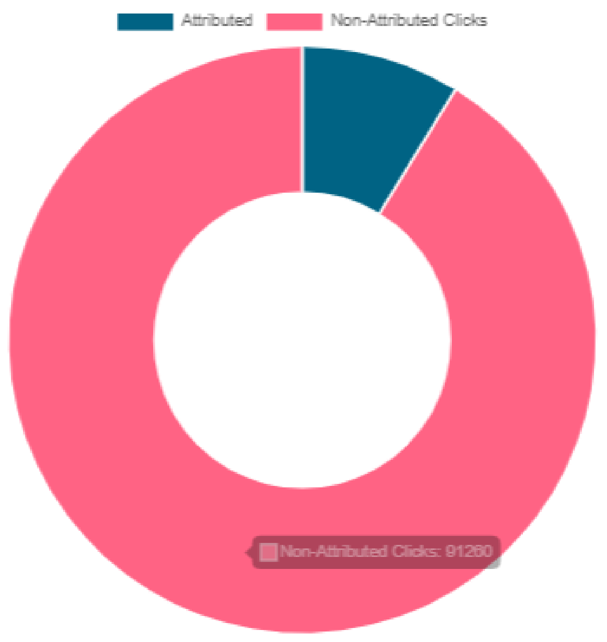


Fig - Attributed to non attributed clicks

IPs with most clicks

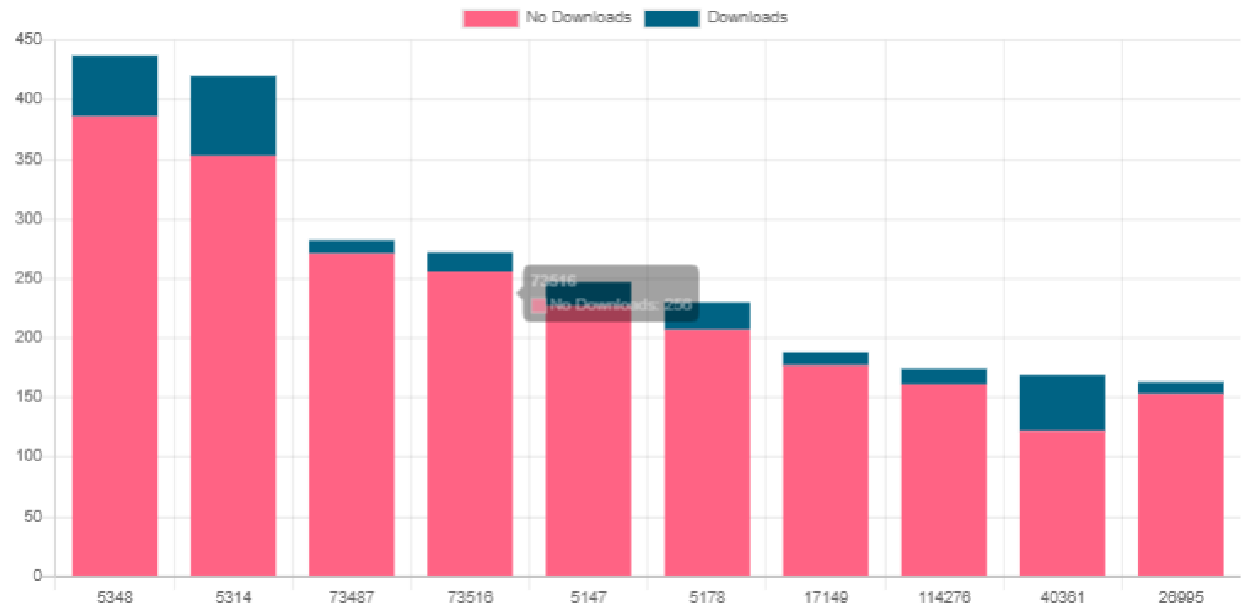


Fig - IPs with the most number of clicks

Channel with most clicks

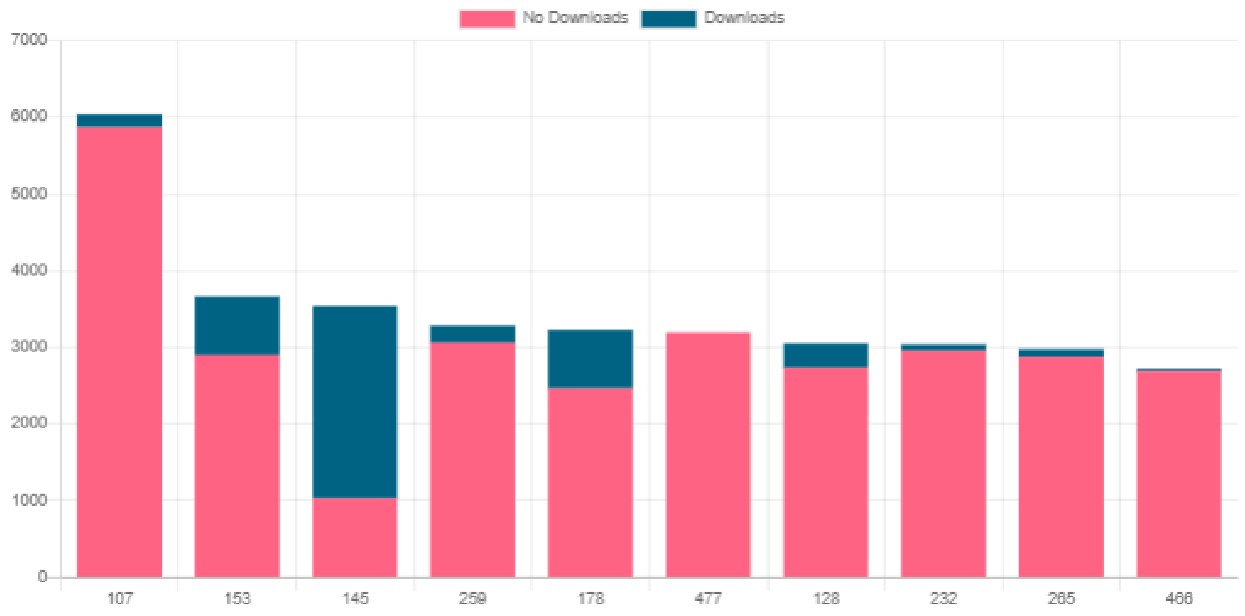


Fig - Channels with the most number of clicks

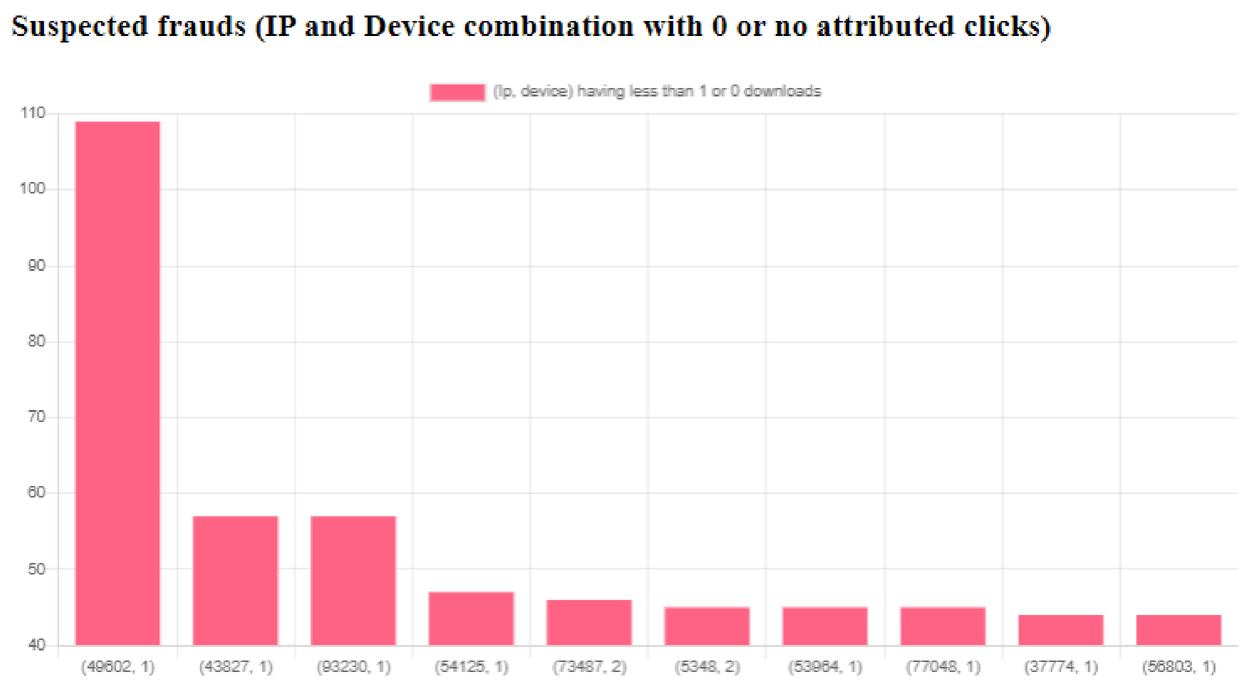


Fig - Suspected Frauds with 0 or 1 attributions

Practical Implications

- Businesses can measure the success rate per channel and invest as per the requirements on each channel
- Businesses can reduce the spending on the channels which are no giving sufficient returns
- Certain IPs can be monitored for suspicious activities and added security measures can be implemented
- Resource utilization will give an idea of resources being consumed by servers