

# Digital Marketing

Step Up Your Data Potential

## Overview

- Manage and analyze your marketing data from different platforms.
- Find your emails based on their sent status, campaign, and type to easily create and edit your email content.
- Visualize public marketing event data and analyze the convert rate.
- Create customized dashboards for your own purpose.



*Meetup*



## Software as a service (SaaS)

- ❖ Low cost
- ❖ Secure
- ❖ Data ownership
- ❖ Customizable
- ❖ Open source
- ❖ Easy to use

## Use of NEW Technologies

**Apache Airflow** - Gather emails, event registrations and other information from different sources.

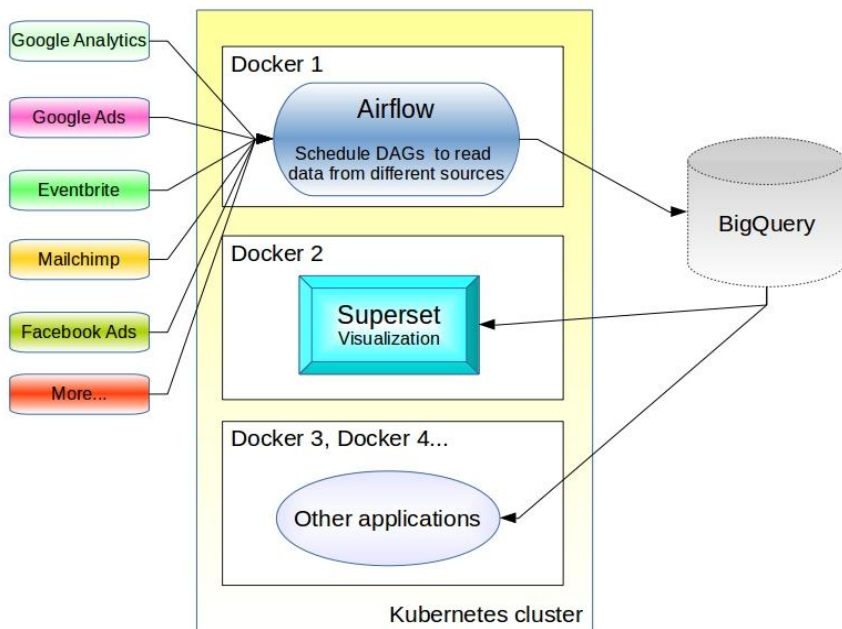
**Apache Superset** - create dashboard to display data.

**Docker** - Host application in containers

**Kubernetes** - Manage Docker images, load balancing, security



# Project Flow Chart



# Project Demo



## Components in the Demo

**Google Cloud Platform (GCP)** - Project monitoring, Google Cloud Shell, Kubernetes Engine, Container Registry, BigQuery, Cloud SQL, and etc.

**Social platform API usage example** - Eventbrite

**Docker** - Dockerfile, Supervisor multi process management

**Airflow** - DAG

# ~\$40

Monthly GCP Cost starting from

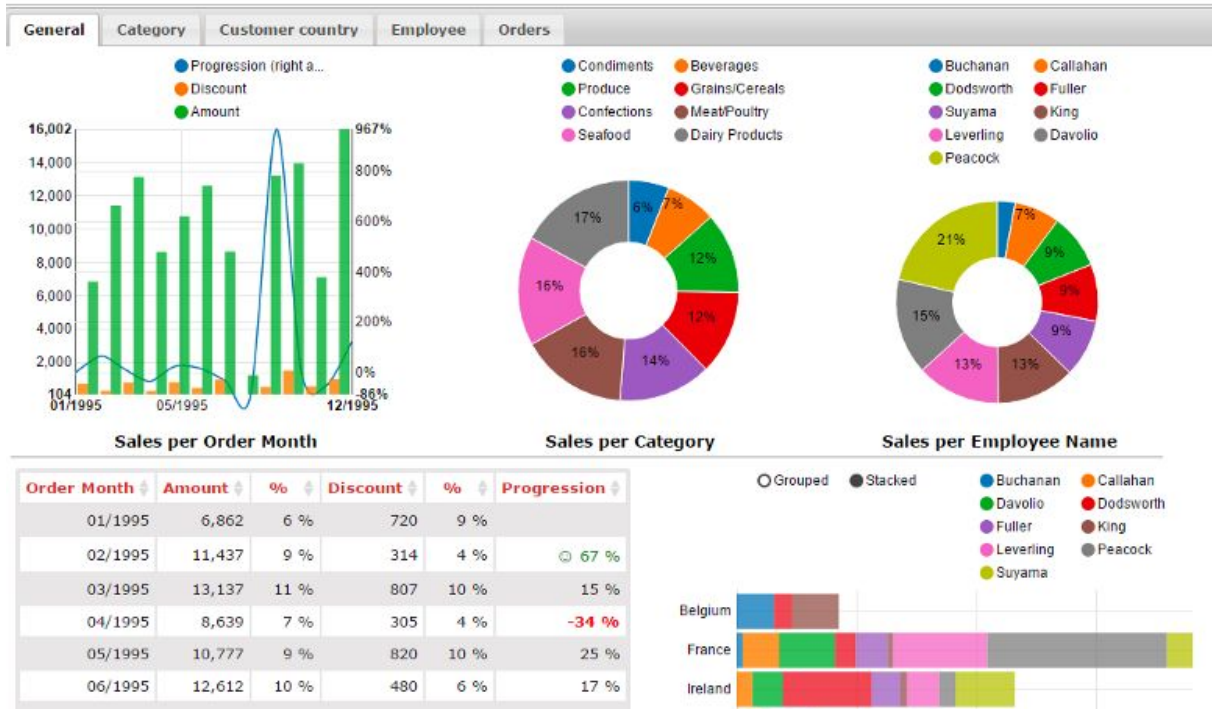


## Summary and Future Plan

1. Dash-Plotly application. Graphs are more flexible comparing to Superset.
2. Current project -> data acquisition and analysis automation  
Future -> Automating actions (e.g. send promotion emails based on the collected data statistic automatically.)
3. Collaborate with financial team
4. Machine learning

# Digital Marketing

## *Step Up Your Data Potential*



## OVERVIEW

The digital marketing project gives you the ability to manage and analyze your marketing data from different platforms such as Google Analytic, Gmail, Eventbrite, and Google Ad. You can find your emails based on their sent status, campaign, and type to easily create and edit your email content. You can visualize the summary of public marketing event data and analyze the conversion rate. You can also create customized dashboards using the acquired data for your purpose.

## WHAT WE PROVIDE

We help you **SET UP** and **TEACH** how to use different digital marketing products.

We **COLLECT** ALL data from your social media platforms.

We **BUILD** business insights customizable dashboards.

We provide continuous **SUPPORT**.

We give you ALL our **CODES**.



## FEATURE SUMMARY

The prospective scope of this project is closed to Software as a service (SaaS). According to Wikipedia, software as a service is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. It is sometimes referred to as "on-demand software", and was formerly referred to as "software plus services" by Microsoft. We feature in the following aspects:

1. Low cost: Monthly GCP service cost starting from \$40.
2. Customizable: We select useful data when doing data ingestion, choose storage methods, customize dashboard layout, do mini machine learning projects on data, migrate marketing data with other data, etc.
3. Secure: Using Kubernetes clustering, all authentication keys will not be exposed to the public.
4. Data ownership: You own all the data that we retrieve using APIs from various social platforms. If you don't want to do data update, we are able to provide you program that query all historical data once to do analyst using the traditional method like Excel.
5. Open source: All programs, services, and applications are open source product without cost. If you have a strong technical team, you can maintain the services once it is deployed without our support.
6. Easy to use: Superset is an easy to learn software that non-technical person can use to build customized dashboard.
7. **Business insight support:** We do projects for customers from different industries. We have good recommendations about how you can use your data to build meaningful visualizations and perform useful statistics or machine learning analysis.

**We provide many use cases in the project components section.**

8. Big data: All the components that this project uses are scalable. For example, Kubernetes can do scaling and load balancing automatically.
9. **Training:** We provide training to non-technical people about how to use digital marketing technologies from Google, Facebook, LinkedIn, etc.



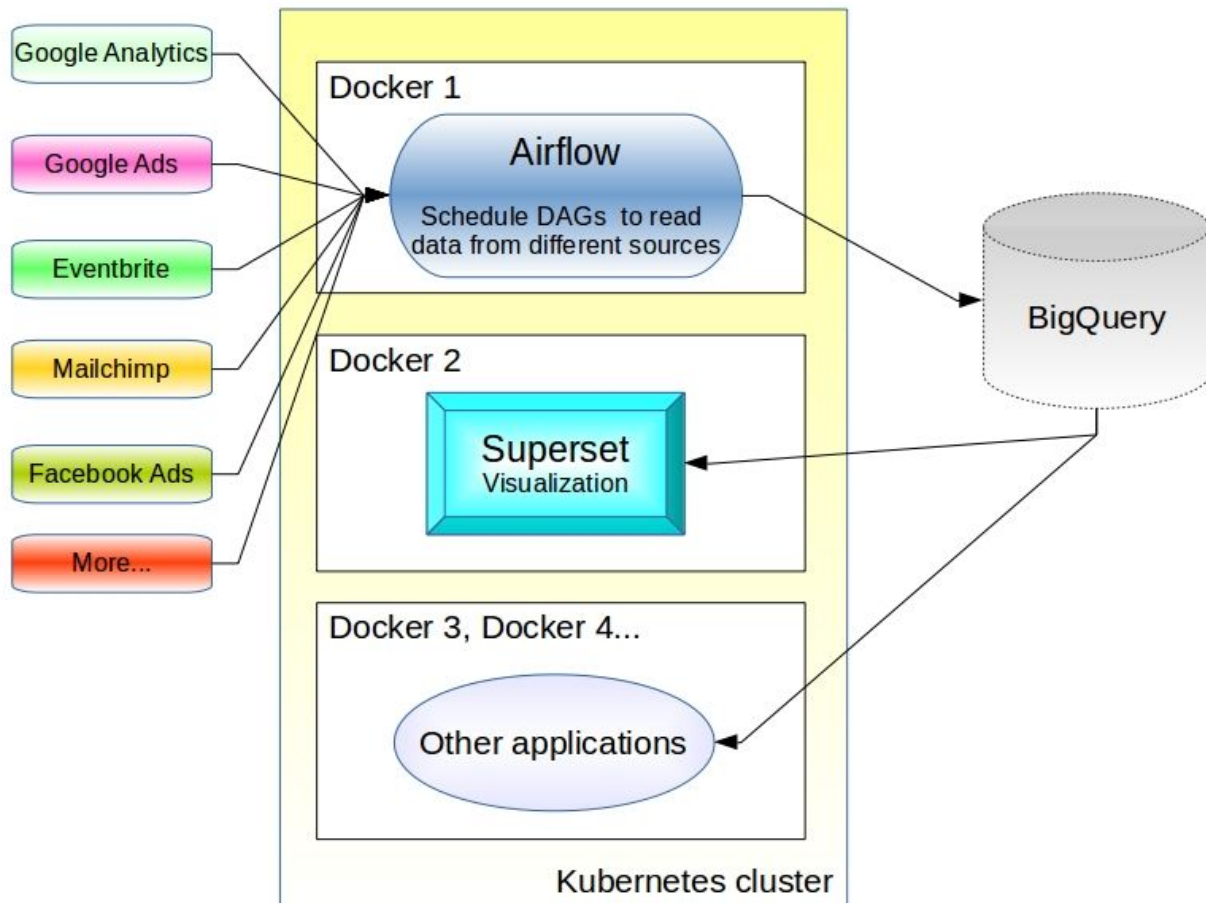
## PROCEDURES

1. **Collect data from different sources:** Build a Docker container that hosts Apache Airflow with various DAGs that gather emails, event registrations and other information from different sources and store them into Google BigQuery.
2. **Visualization using Apache Superset:** Build a Docker container that hosts Apache Superset. Connect Superset to BigQuery then create dashboards to display data.
3. **Host Docker applications on Google Cloud:** Create a Google Compute instance with Kubernetes that host multiple Dockers that serve the entire project. The advantage of Kubernetes includes auto scaling, application isolation, also good security.
4. **Extensions:** Consider creating more Docker containers to set up applications like machine learning model based on email data, Dash-Plotly application, etc.

## COST

1. Script to retrieve historical data ~ \$
2. Google Cloud Platform (GCP) service cost ~\$
  - a. Kubernetes Cluster
  - b. BigQuery
  - c. Cloud SQL
  - d. Geocoder
3. First deployment cost ~\$
4. Monthly maintenance fee ~\$
5. Training ~\$

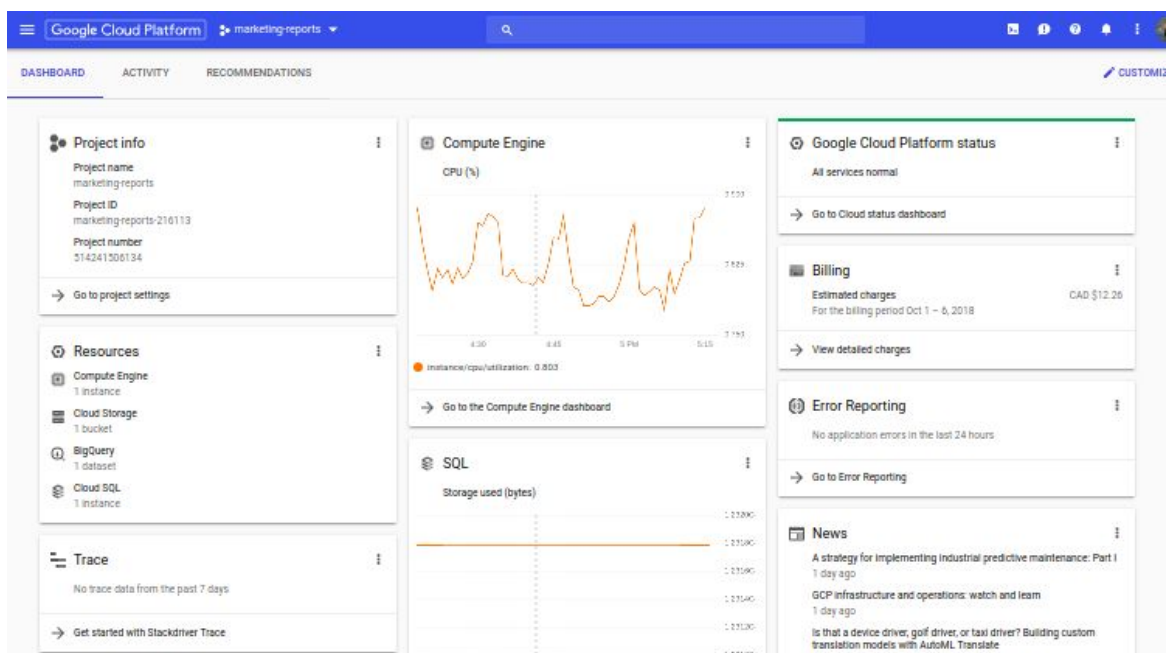
## COMPONENT FLOWCHART



# PROJECT COMPONENTS

## 1. Google Cloud Platform (Project monitoring)

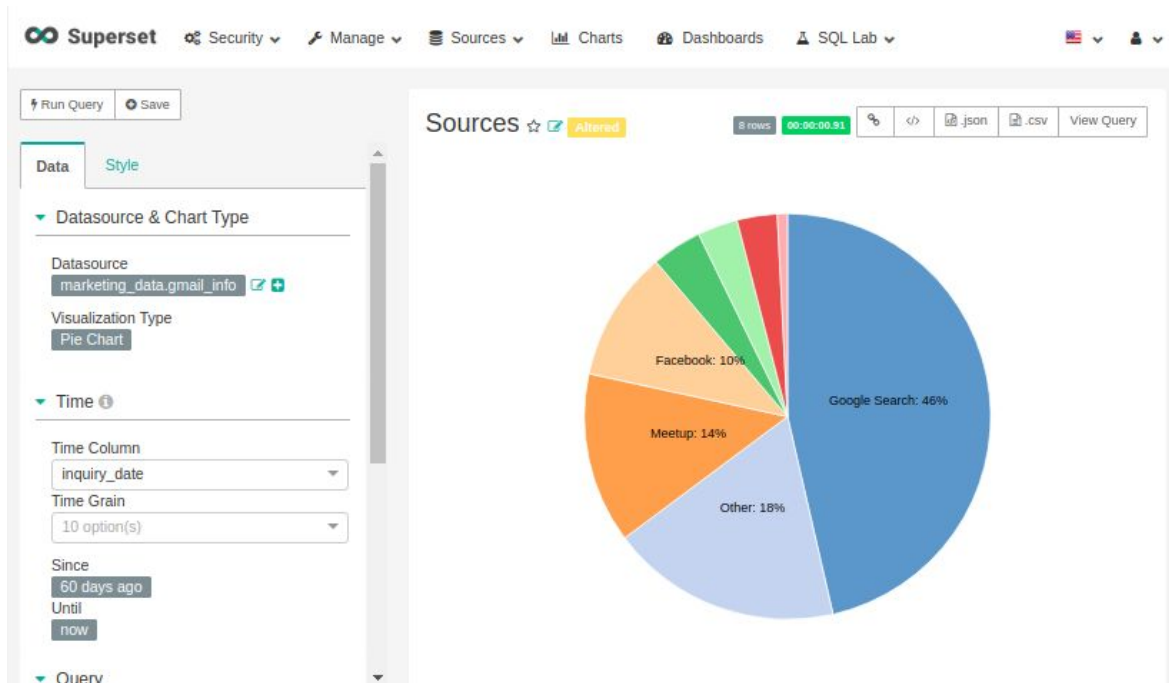
[Google Cloud Platform](#), offered by Google, is a suite of cloud computing services that run on the same infrastructure that Google uses internally for its end-user products. Alongside a set of management tools, it provides a series of modular cloud services including computing, data storage, data analytics, and machine learning.



## 2. Apache Superset (Frontend)

[Superset](#) is an easy to use data visualization tools that have fantastic templates. Non- technical people can quickly learn and create customizable dashboards based on business purposes. It supports various database connections and has security modules.





Superset recently supports BigQuery connections. To containerize Superset, we refer to this [Github example](#).

The most important Dockerfile command is to add:

```
COPY login_info.json /home/superset/
ENV GOOGLE_APPLICATION_CREDENTIALS login_info.json
```

Where login\_info.json is the BigQuery service account key.

For Kubernetes deployment, do the followings:

```
# Initialize the Superset Docker container inside the pod - set up username
and password
kubectl exec -it superset-c8ddc97d5-4xwtm superset-init
# Choose LoadBalancer expose type to obtain a public IP
kubectl expose deployment superset --type=LoadBalancer --port 80
--target-port 8088
```

To set up Superset-BigQuery connection, create tables ... follow Superset official documentation.

### 3. BigQuery(Backend)

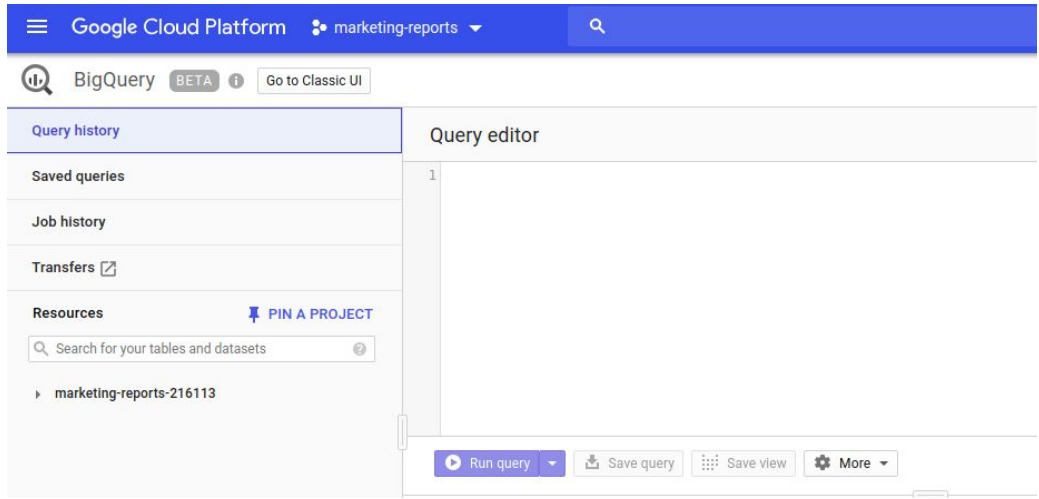
Follow Google official instructions to create datasets (equivalent to a database) in [BigQuery](#). Generate Keys



Google BigQuery

with different permissions to read or write data.

For the backend, we can also use Cloud SQL, MySQL, Redshift, MongoDB, PostgreSQL, etc. We will adjust according to customers' needs.



#### 4. Apache Airflow (Automation)

[Apache Airflow](#) (or simply Airflow) is a platform to programmatically author, schedule, and monitor workflows.



When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative.

Build a Docker image that host airflow. Write different DAGs that can gather data from different sources using those credential. (One source per DAG.) Create DAGs that delete obsolete data.

To build the image, we mainly refer to this [Github example](#), using supervisor.

Mainly we follow documents in this Github repository. We write one DAG to get data from one source and store data into multiple tables. Table schemas are already predetermined by inspecting the data. In the Python code, we need to parse the full query response and store them into different columns in the table. DAG properties such as retry times, failure email, and run frequency can be specified in the .py script.

DAGs will be all stored in AIRFLOW\_HOME/dags.

Since all the authentication files, passwords, tokens are in this Docker container, we cannot expose it to the public. Kubernetes provide a ClusterIP deployment method that will secure the Airflow Docker container, as follows:

```
# use ClusterIP expose method, only visible inside the cluster
kubectl expose deployment airflow --type=ClusterIP --port 8080 --target-port 8080
# forward traffic to localhost then use web preview in Google Cloud shell to see the web UI for Airflow.
kubectl port-forward airflow-666977cbcd-7qp2n 8080:8080
```

DAG script sample and some explanation:

```
# Main DAG function
def read_to_db():
    # create google analytics object
    analytics = initialize_analyticsreporting()
    # get report
    report = get_report(analytics, '2018-08-31', '2018-09-01')
    # parse response
    data = report['reports'][0]['data']['rows']
    # establish mysql connection
    mysql_hook = MySQLHook(mysql_conn_id='dashboard_test')
    # Insert into database
    row_insert = """INSERT INTO ga_visit (_date, _session, pageviews,
bounds, avg_session_dur)\
VALUES (STR_TO_DATE(%, '%Y%m%d'), %, %, %, %);"""
    for v in data:
        _date = v['dimensions'][0]
        _session = v['metrics'][0]['values'][0]
        pageviews = v['metrics'][0]['values'][1]
        bounds = v['metrics'][0]['values'][2]
        avg_session_dur = v['metrics'][0]['values'][3]
        mysql_hook.run(row_insert, parameters=\
(_date, _session, pageviews, bounds, avg_session_dur))
# Following are defaults which can be overridden later on
default_args = {
    'owner': 'bobo',
    'depends_on_past': False,
    'start_date': datetime(2018, 9, 1),
    'email': ['xxxxxxx@gmail.com'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 0,
    'retry_delay': timedelta(minutes=1),
}
```

```
dag = DAG('readGA', default_args=default_args)
read_to_db_task = PythonOperator(task_id='read_to_db',
                                provide_context=False,
                                python_callable=read_to_db, dag=dag)
```

## 5. Kubernetes (Host)

Create a Google Compute Engine instance with [Kubernetes](#) that host multiple Docker containers that serve this project. One application one Docker container.

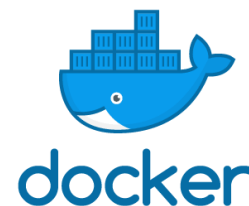


For Docker images, we build locally and push it to Google Container Registry then deploy to Kubernetes cluster.

```
docker build -t boboairflow .
docker images
docker tag boboairflow gcr.io/${PROJECT_ID}/boboairflow
# push it to the cloud
docker tag boboairflow gcr.io/${PROJECT_ID}/boboairflow
docker push gcr.io/${PROJECT_ID}/boboairflow
# deploy image
kubectl run airflow --image=gcr.io/${PROJECT_ID}/boboairflow:latest --port
8080
# expose deployment
kubectl expose deployment airflow --type=ClusterIP --port 8080 --target-port
8080
```

## 6. Docker

[Docker](#) is a computer program that performs operating-system-level virtualization, also known as "containerization". It is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud. Sample DockerFile for Apache Airflow:



```
FROM python:3.6.3
# supervisord setup
RUN apt-get update && apt-get install -y supervisor
COPY supervisord.conf /etc/supervisor/conf.d/supervisord.conf
# Airflow setup
```

```

ENV AIRFLOW_HOME=/app/airflow
# this is a fucking BUG... have to force these ENV variables
ENV SLUGIFY_USES_TEXT_UNIDECODER=yes
ENV AIRFLOW_GPL_UNIDECODER=yes
ENV GPL_UNIDECODER=yes
# No example DAGS
ENV AIRFLOW__CORE__LOAD_EXAMPLES=False
RUN pip install --upgrade google-cloud-bigquery apache-airflow
google-api-python-client pymysql\
    mysqlclient eventbrite meetup-api
COPY ./dags $AIRFLOW_HOME/dags
COPY cloud_sql_proxy $AIRFLOW_HOME
COPY cloudsql_writer.json $AIRFLOW_HOME
RUN airflow initdb
EXPOSE 8080
CMD ["/usr/bin/supervisord"]

```

We set up environment variables, copy files, install dependencies and run commands to build docker.

## 7. Supervisor

[Supervisor](#) is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

For example, to successfully use Apache Airflow, we need to start two processes, airflow scheduler and web server, supervisord is one of the best options to handle this requirement:

```

[supervisord]
nodaemon=true

[program:scheduler]
command=airflow scheduler
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true

[program:server]
command=airflow webserver -p 8080
stdout_logfile=/var/log/supervisor/%(program_name)s.log
stderr_logfile=/var/log/supervisor/%(program_name)s.log
autorestart=true

```



## 8. Emails

**Business Insight:** There are lots of information in our emails. We can write codes to filter out useful emails, to parse email contents, to perform natural language processing (NLP), and to send emails. Here are some use cases:

- Filter phone numbers, email addresses in emails to get contacts of potential clients.
- Send batch emails to promote products.
- Filter out keywords in inquiry emails for targeted promotions.
- Sentiment analysis on feedback emails.



Use [IMAP \(Internet Message Access Protocol\)](#) to query emails. Sample codes refer to this [blog post](#). For query format, refer to rfc3501.

```
daysback = 7 # 7 days
notsince = 0 # since now.
since = (date.today() - timedelta(daysback)).strftime("%d-%b-%Y")
before = (date.today() - timedelta(notsince)).strftime("%d-%b-%Y")
since_sql = (date.today() - timedelta(daysback)).strftime('%Y-%m-%d')
before_sql = (date.today() - timedelta(notsince)).strftime('%Y-%m-%d')
SEARCH = '(SINCE {si} BEFORE {bf} SUBJECT "Courses")'.format(si=since,
bf=before)
FEILDS = '(BODY.PEEK[HEADER.FIELDS (DATE FROM SUBJECT)] BODY.PEEK[TEXT])'
# Search and fetch emails!
email_ls = gmail.load_parse_query(search_query=SEARCH,\
                                  fetch_query=FEILDS,\
                                  folder=''[Gmail]/All Mail''')
```

This code will filter out the email contents and headers that have subject “Courses” from seven days ago till now. Using PEEK, the emails are not marked as seen, so that the users that using this email account won’t miss any unread emails.

## 9. Eventbrite/Meetup

**Business Insight:** A company might host hundreds of events in the past to promote their service or products. For example, a job training institute might use Eventbrite to create workshops such as Python training. We can use [Eventbrite SDK](#) or [Meetup API](#) to get the following information: Event name, event attendance count, attendee emails, attendee employment status, event locations, event time, event sales (if paid), etc. Some use cases:

- Check attendance percentage of subsequent events or repetitive promotion events to determine the values of re-hosting.
- Gather frequent attendees by emails; these people might be potential clients.
- Compare attendance of the same event in different locations to determine where to host future events.
- Rehosting popular workshops.
- Compare the effectiveness of different event hosting platforms (Eventbrite and Meetup)
- Customer churn analysis
- Get Events from competitors to compare.
- Relate event data to ads data to analyze the effectiveness of advertisement.



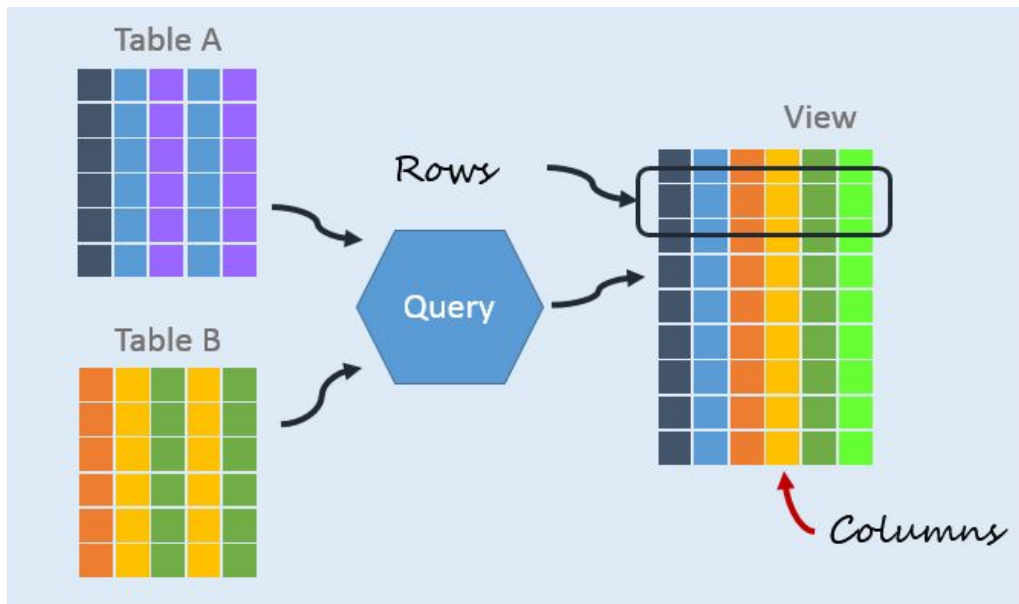
Since the raw data cannot be directly used by Superset, we use views to do data aggregation and simple arithmetic operation, sample:

```
CREATE VIEW eb_whole AS
SELECT attendee_c/event_capacity as attendee_percent, event_start_time,
event_name, event_id FROM
(SELECT count(*) AS attendee_c, event_id_a FROM eb_attendees GROUP BY
event_id_a) h LEFT JOIN eb_events
ON h.event_id_a=eb_events.event_id;
```

This query calculate the event attendance ratio.

The advantage of using view:

- View is a logical table that doesn't store data.
- Read-only.
- Automatic update when base table is updated.



## 10. Google Ads/Facebook Ads

**Business Insight:** Google Ads is an online advertising platform developed by Google, where advertisers pay to display brief advertisements, service offerings, product listings, video content and generate mobile application installs within the Google ad network to web users.

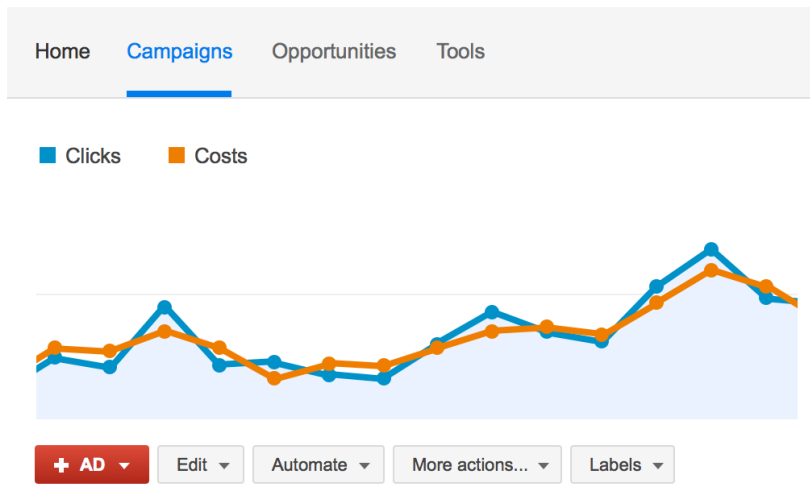


The [AdWords API](#) allows apps to interact directly with the Google Ads platform, vastly increasing the efficiency of managing large or complex Google Ads accounts and campaigns.

Facebook Ads is an advertising service provided by Facebook. With micro-targeting features, Facebook Ads allows you to reach your exact target audience based on demographics, location, interests, and even behaviors.

Facebook provides official [Marketing API](#) SDKs. This is the easiest way to develop on Marketing API. The SDKs are open-source, and provide support for authentication, core object models, tests, and stability on top of the HTTP-based API.

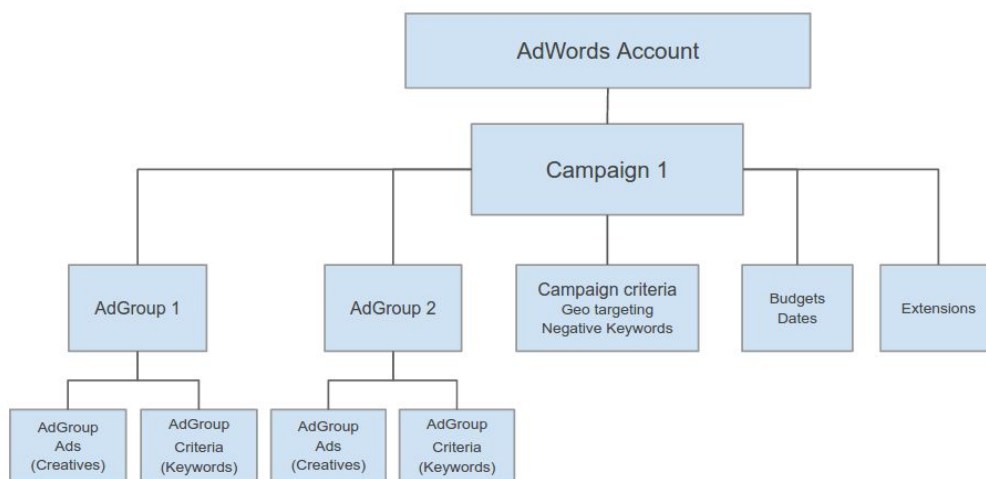




Use cases:

- Campaign Ad management
- Ad traffic estimation and click through behavior, combined with Google Analytics data
- Pick best strategies to do advertisement to reduce cost.

Google Ads campaign data hierarchy:



Some useful information about [Facebook Campaign Insight](#):

Field	Description
actions	The total number of actions people took that are attributed to your ads. Actions may include engagement, clicks or conversions.

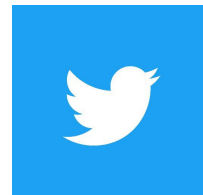
canvas_avg_view_percent	The average percentage of the Instant Experience that people saw. An Instant Experience is a screen that opens after someone interacts with your ad on a mobile device. It may include a series of interactive or multimedia components, including video, images product catalog and more.
canvas_avg_view_time	The average total time, in seconds, that people spent viewing an Instant Experience. An Instant Experience is a screen that opens after someone interacts with your ad on a mobile device. It may include a series of interactive or multimedia components, including video, images product catalog and more.
clicks	The number of clicks on your ads.
cost_per_estimated_ad_recallers	The average cost for each estimated ad recall lift. This metric is only available for assets in the Brand awareness, Post engagement and Video views Objectives. This metric is estimated and in development.
inline_link_click_ctr	The percentage of time people saw your ads and performed an inline link click.

## 11. Twitter

**Business Insight:** We can use [Tweepy API](#) to collect relevant tweets to analyze.

Use cases include:

- Collect Retweets, hashtags that related to the company to do sentiment analysis.
- Analyze retweet timestamp to find out the best time and best topic to tweet.
- Schedule tweet posting time



Sentiment analysis pipeline major steps:

- a. Load tweets
- b. Tokenize text to extract individual words or sentences
- c. Stem words / Lemmatize words
- d. Remove stopwords
- e. Train a model

```
# Remove non-alphanumeric characters
text = [re.sub(r'\W+', ' ', x) for x in text]
```

```

# tokenize
text_tokens = [nltk.word_tokenize(x) for x in text]
stemmer = PorterStemmer()
text_tokens = [[stemmer.stem(w) for w in x] for x in text_tokens]

# create an instance of TfidfVectorizer; get it to remove stopwords
tf = TfidfVectorizer(stop_words = 'english', use_idf = False, norm = None,
binary = True)

# convert text to features
text_tokens = tf.fit_transform([' '.join(x) for x in text_tokens])
# get feature names
feature_names = tf.get_feature_names()
# Use LabelEncoder to convert text labels ("positive") to numbers (1).
le = LabelEncoder()
sent = le.fit_transform(sent)

# Split the data into a training and test sets, train the model and evaluate
its performance.
X_train, X_test, y_train, y_test = train_test_split(text_tokens, sent,
random_state = 0)
clrf = RandomForestClassifier(n_estimators = 10)
clrf.fit(X_train, y_train)
y_predrf = clrf.predict(X_test)
f1rf = f1_score(y_test, y_predrf, average = "macro")
accuracyrf = accuracy_score(y_test, y_predrf)
print("Random forest f1 score:", f1rf, "accuracy:", accuracyrf)

# Use the output of the random forest to check importances of various words
in predicting classes.
from sklearn.preprocessing import scale
from collections import OrderedDict
def class_feature_importance(X, y, class_labels, feature_names,
feature_importances, max_top = 20):
    X = scale(X)
    result = {}
    # extract feature importances by class
    for c in set(y):
        result[class_labels[c]] = dict(zip(feature_names, np.mean(X[y == c,
:], axis = 0) * feature_importances))
    # sort by importance and keep the most important
    for cname in result.keys():
        result[cname] = sorted(result[cname].items(), key = lambda x: x[1],
reverse = True)[:max_top]
    return result

```

## 12. LinkedIn

**Business Insight:** The [LinkedIn API](#), also known as the [REST API](#) is the heart of all programmatic interactions with LinkedIn. All other methods of interacting, such as the JavaScript as well as the Mobile SDKs, are simply wrappers around the REST API to provide an added level of convenience for developers.



Some use cases include:

- Get a company's profile
- Get a company's updates
- Get comments for a specific company update
- Get a company's followers, by segment
- Add comment as a company
- Get historical follower statistics about a company
- Get historical status update statistics about a company

## 13. Google News API

**Business Insight:** Get and search live headlines, articles, images, and other article metadata from Google News with [Google News JSON API](#).



One use case is to search news topics related to the company and competitors, Perform topic modeling to extract popular topics. Then marketing can use these topics to write new posts.

Topic Modeling with LDA and NMF steps:

- a. Stem, lemmatize and remove punctuation and other non-alphanumeric characters from news contents.
- b. Create term-document matrices from the documents, e.g., tf-idf and binary (bag-of-words).
- c. Unsupervised machine learning for topic modeling: Latent Dirichlet Allocation (LDA) and Non-negative Matrix Factorization (NMF). They both achieve the same result (discover topics in the documents), but while LDA

uses a probabilistic approach, NMF uses linear algebra. NMF also may produce more meaningful topics on smaller datasets.

```
lm = WordNetLemmatizer()
ps = PorterStemmer()
def lemmatize(word):
    """
    Lemmatizes a word
    """
    # get part of speech (needed for the lemmatizer)
    pos = nltk.pos_tag([word])[0][1]
    print (pos)
    # convert into wordnet POS
    if pos.startswith("V"):
        pos_wn = wordnet.VERB
    elif pos.startswith("R"):
        pos_wn = wordnet.ADV
    elif pos.startswith("J"):
        pos_wn = wordnet.ADJ
    else:
        pos_wn = wordnet.NOUN
    # lemmatize
    return lm.lemmatize(word, pos = pos_wn)

def clean_string(string, lemmas = True):
    """
    Converts the string to lowercase, lemmatizes and removes
    non-alphanumerics
    """
    if pd.isnull(string): return ""
    # remove non-alphanumeric characters
    string = re.sub(r'^A-Za-z]+', ' ', string)
    # to lowercase and stem / lemmatize
    if lemmas:
        string = [lemmatize(x) for x in string.lower().split()]
    else:
        string = [ps.stem(x) for x in string.lower().split()]
    return " ".join(string)
max_features = 10000

# tf-idf
tf = TfidfVectorizer(max_df = 0.95, min_df = 2, max_features =
max_features, stop_words = 'english')
features_tfidf = tf.fit_transform(title_cleaned)
feature_names_tfidf = tf.get_feature_names()

# bag-of-words
```



```

bow = CountVectorizer(max_df = 0.95, min_df = 2, max_features =
max_features, stop_words = 'english')
features_bow = bow.fit_transform(title_cleaned)
feature_names_bow = bow.get_feature_names()

# specify the number of topics in the documents
topics_count = 5

# train an NMF model
nmf = NMF(n_components = topics_count, random_state = 0, alpha = 0.1,
l1_ratio = 0.5, init = "nndsvd")
nmf.fit(features_bow)

# train an LDA model
lda = LatentDirichletAllocation(n_components = topics_count, max_iter = 5,
random_state = 0, learning_method = "online")
lda.fit(features_tfidf)

def show_topics(model, feature_names, top_words = 10):
    """
    Displays the top words from a model
    """
    print("Model: %s" % model.__class__.__name__)
    for i, topic in enumerate(model.components_):
        print("Topic %i\n%r\n" % (i, ", ".join([feature_names[x] for x in
topic.argsort()[: -top_words - 1: -1]])))

```

## 14. MailChimp

**Business Insight:** [MailChimp](#) is a marketing automation platform and an email marketing service, mostly for sending out marketing emails. Mailchimp provides RESTful API to do tasks such as:



- Get response statistics of email recipients of company campaigns.
- Get collections of Google Ads Status.
- Send batch emails at a scheduled time.
- Get subscribed or unsubscribed member list to compare with the email address list got from Eventbrite API, Gmail, etc.

## 15. Instagram

**Business Insight:** Instagram is a photo and video-sharing social networking service. The [Instagram API](#) Platform can be used to build non-automated, authentic, high-quality apps and services that:



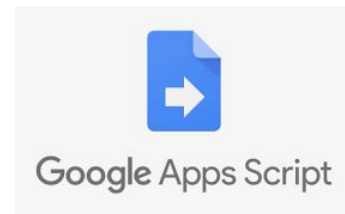
- Help individuals share their contents with 3rd party apps.
- Help brands and advertisers understand, manage their audience and media rights.
- Help broadcasters and publishers discover content, get digital rights to media, and share media with proper attribution.

Some use cases:

- Get user comments on event photos.
- Get information about tagged objects.
- Get a list of recent media objects from a given location with location coordinate.

## 16. Google App Script

**Business Insight:** [Google Apps Script](#) is a scripting language for light-weight application development in the G Suite platform. It is based on JavaScript 1.6 with some portions of 1.7 and 1.8 and provides subset of ECMAScript 5 API. App Script makes it easy to create and publish add-ons in an online store for Google Sheets, Docs, Slides, and Forms.

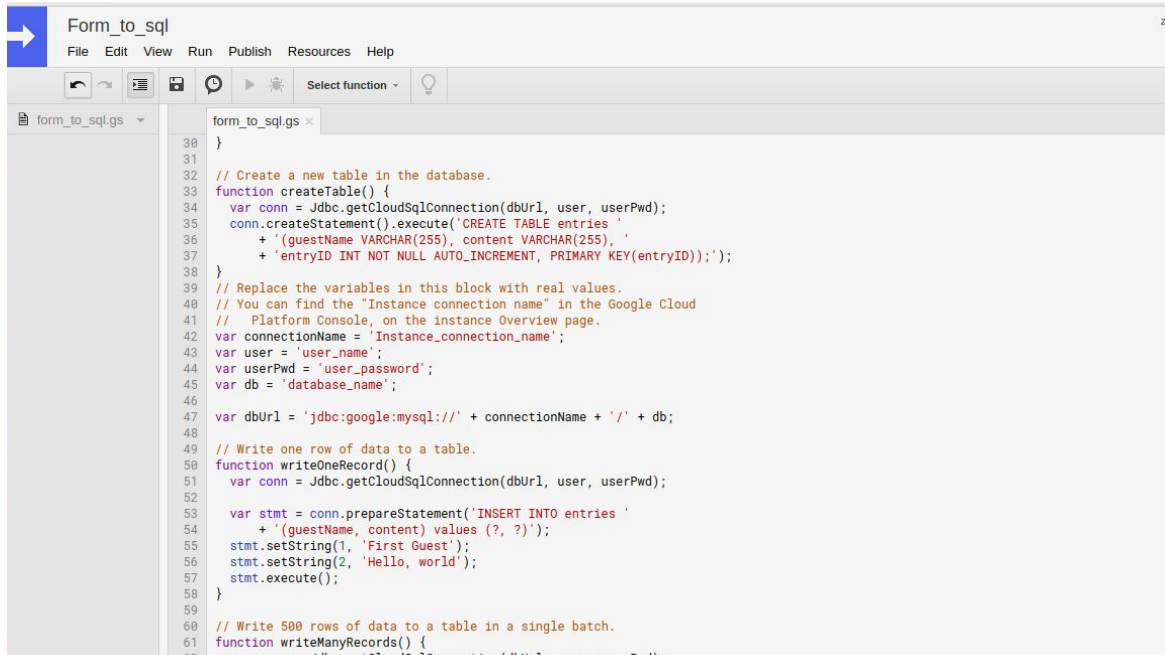


Some use cases:

- Ingest Google Form survey data to Google Cloud SQL (**GO PAPERLESS!**)
- Invite people to join events and automatically add to Google Calendar.

- List upcoming events in Google Calendar.
- Merge a template email with content.

The following is the Google App Script editor:



```

Form_to_sql
File Edit View Run Publish Resources Help

form_to_sql.gs
30 }
31
32 // Create a new table in the database.
33 function createTable() {
34   var conn = Jdbc.getCloudSqlConnection(dbUrl, user, userPwd);
35   conn.createStatement().execute('CREATE TABLE entries '
36     + '(guestName VARCHAR(255), content VARCHAR(255), '
37     + 'entryID INT NOT NULL AUTO_INCREMENT, PRIMARY KEY(entryID));');
38 }
39 // Replace the variables in this block with real values.
40 // You can find the "Instance connection name" in the Google Cloud
41 // Platform Console, on the instance Overview page.
42 var connectionName = 'Instance_connection_name';
43 var user = 'user_name';
44 var userPwd = 'user_password';
45 var db = 'database_name';
46
47 var dbUrl = 'jdbc:google:mysql://' + connectionName + '/' + db;
48
49 // Write one row of data to a table.
50 function writeOneRecord() {
51   var conn = Jdbc.getCloudSqlConnection(dbUrl, user, userPwd);
52
53   var stmt = conn.prepareStatement('INSERT INTO entries '
54     + '(guestName, content) values (?, ?)');
55   stmt.setString(1, 'First Guest');
56   stmt.setString(2, 'Hello, world');
57   stmt.execute();
58 }
59
60 // Write 500 rows of data to a table in a single batch.
61 function writeManyRecords() {

```

## 17. Google Analytics

**Business Insight:** [Google Analytics](#) is a freemium web analytics service offered by Google that tracks and reports website traffic. The [Google Analytics Reporting API v4](#) is the newest programmatic method to access report data in Google Analytics. With the Google Analytics Reporting API, you can:

- Build custom dashboards to display Google Analytics data.
- Automate complex reporting tasks to save time.
- Integrate your Google Analytics data with other business applications.



Some specific use cases:

- Hook up google analytics to the company website and get website bouncing rate, average session duration trend to find out the popularity of the websites.
- Get information about where visitors are rerouted in, which includes:

Google search, external links, etc.

- Get visitor geographic data and demographic data. (i.e., If the company is based in Toronto, get the number of visitors from Toronto comparing to other regions)

[Google Analytics Available Dimensions and Metrics](#), Examples:

Dimension	Metrics
ga:userType	ga:users
ga:sessionCount	ga:newUsers
ga:daysSinceLastSession	
ga:userBucket	

Dimension	Metrics
	ga:sessions
	ga:bounces
	ga:bounceRate
	ga:sessionDuration
	ga:avgSessionDuration

Dimension	Metrics
ga:referralPath	ga:organicSearches
ga:fullReferrer	
ga:campaign	
ga:source	
ga:medium	
ga:sourceMedium	
ga:keyword	

Dimension	Metrics
ga:continent	
ga:subContinent	
ga:country	
ga:region	
ga:metro	
ga:city	
ga:latitude	
ga:longitude	

.....

Usage example:

```
def get_report(analytics,s_dt,e_dt):
    # Use the Analytics Service Object to query the Analytics Reporting API
    return analytics.reports().batchGet(
        body={
            'reportRequests': [
                {
                    'viewId': VIEW_ID,
                    'dateRanges': [
                        {
                            'startDate': s_dt,
                            'endDate': e_dt
                        }
                    ],
                    'dimensions': [
                        {
                            'name': 'ga:date'
                        }
                    ],
                    'metrics': [
                        {
                            'expression': 'ga:sessions'
```

```
    },  
    {  
      'expression': 'ga:pageviews'  
    }  
  ]  
}  
].execute()
```

## 18. Web Scraping

We also provide web scraping service, mainly using Python, some packages:



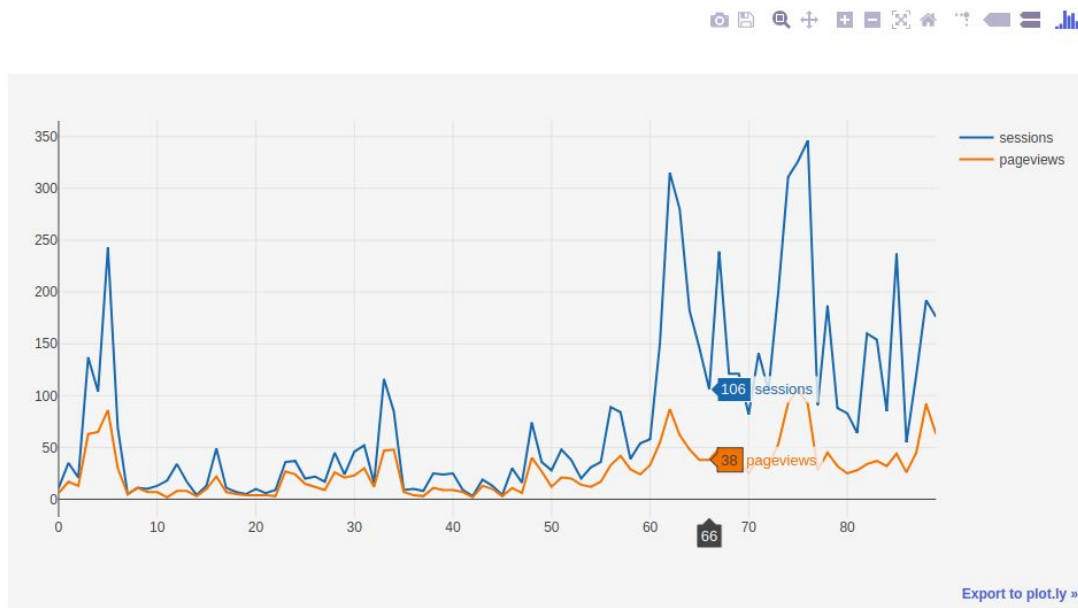
- BeautifulSoup, Request, Scrapy, lxml for basic/smart scraping
- Selenium, automate web browsers
- Redis Queue (rq) for distributed scraping
- Goose, article extractor

## 19. More...

There are more social media sites that provide API or SDK support. As long as it is legal and doable, we can provide technical support.

## FUTURE DEVELOPMENT

1. Consider building Docker container to host Dash-Plotly application. Graphs are more flexible comparing to Superset.



2. Current project only include data acquisition and analysis automation, we can also consider automating actions (e.g. send promotion emails based on the collected data statistic automatically)
3. Collaborate with financial team. (Marketing + Finance -> Collaborate decisions)
4. Set up machine learning model such as NLP analysis based on the collected data.

Use case:

- a. Sentiment Analysis on user feedbacks, filter out feedbacks that having positive scores and display as user comments on the company website.
- b. Topic classification using Latent Dirichlet allocation on relevant tweets about the company to find out better marketing strategies.
- c. Binary classification to determine customer churn rate.

# The Problem

## Problem statement

AutoZone 0097  
1924 CANDLER RD  
DECATUR, GA  
(404) 286-3991  
#433429 AS260Y  
4.29 P  
Prestone  
ing Fluid, 126.49 P  
#059018  
194LL Sylvania  
Long Life Bulbs, 2 PK  
SUBTOTAL  
TOTAL TAX @ 8.000%  
TOTAL  
XXXXXXXXXXXX6097 DEBIT  
APPROVAL #  
10.78  
0.86  
11.64  
Data Source: CHIP  
App Name/Label: US DEBIT  
AID: A0000000980840  
PIN Online Verified  
REG #11 CSR #68 RECEIPT  
#122711  
STR, TRANS #953910  
STORE #0097  
DATE 05/19/2018 11:05  
# OF ITEMS SOLD 2  
009795 39100519 18  
Take a survey for a  
chance to win \$5000

## Extract useful information accurately from scanned receipt OCR records

Once the phone camera scans the receipt using the Apple/Android APP, detail information about the receipt will be returned in JSON format using Google OCR.

These JSON records have to be parsed and processed **SMARTLY** to get useful information.





## Current status

Currently, rulebase method to return merchant name only has 29% recall score.

Recall score =

Merchant name correct

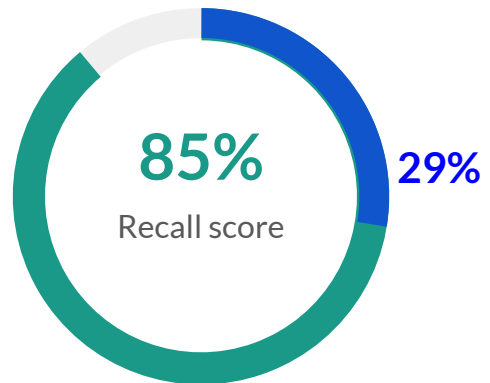
/ All return merchant name



## Goal

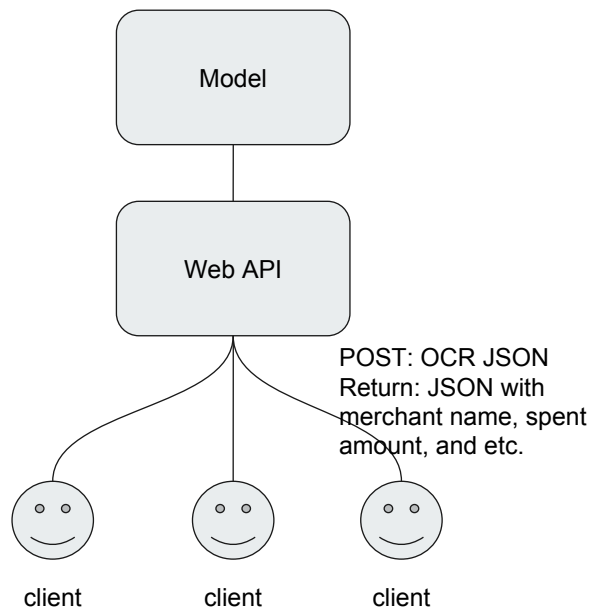
## Our goal - Model performance

Increase the recall score of  
returning correct merchant  
name from 29% to 85%



## Our goal - Model deployment

Build web server API that  
returns the model output in  
JSON response.



# Minestones

## Data exploration

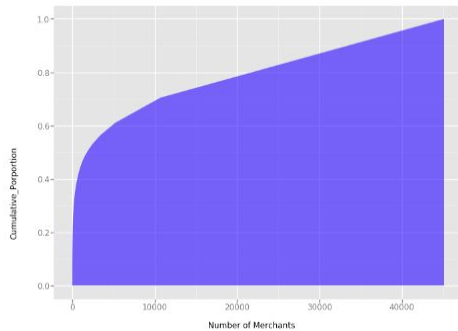
Google OCR

AutoZone 0097  
1924 CANDLER RD  
DECATUR, GA  
(404) 286-3991  
#433429 AS260Y  
4.29 P  
Prestone  
ing Fluid, 126.49 P  
#059018  
194LL Sylvania  
Long Life Bulbs, 2 PK  
SUBTOTAL  
TOTAL TAX @ 8.000%  
TOTAL  
XXXXXXXXXXXX6097 DEBIT  
APPROVAL #  
10.78  
0.86  
11.64  
#122711  
STR. TRANS #953910  
STORE #0097  
DATE 05/19/2018 11:05  
# OF ITEMS SOLD 2  
009795 39100519 18  
Take a survey for a  
chance to win \$5000

```
{
  "textAnnotations": [
    {
      "locale": "en",
      "boundingPoly": {
        "vertices": [
          { "x": 44, "y": 122 },
          { "x": 761, "y": 122 },
          { "x": 761, "y": 1257 },
          { "x": 44, "y": 1257 }
        ]
      },
      "description": "C RABBY JOE'S DONNTOWN\n276 DUNDAS ST\nLONDON, ON N6B1T6\n5196454880\nSALE\nServer #: 000200\nMID: 5800203\nnTID: 006\nBatch #: 417\n04/29/18\nAPPR CODE: 04239\nnVISA\nnREF#: 00000032\n210141\nChip\n7932\nnAMOUNT\nnTIP\nnTOTAL\nn$ 27.09\nn$4.06\nn$31.15\nnAPPROVED\n",
      "boundingPoly": {
        "vertices": [
          { "x": 153, "y": 122 },
          { "x": 315, "y": 129 },
          { "x": 313, "y": 173 },
          { "x": 151, "y": 166 }
        ]
      },
      "description": "C RABBY",
      "boundingPoly": {
        "vertices": [
          { "x": 345, "y": 133 },
          { "x": 445, "y": 137 },
          { "x": 443, "y": 170 },
          { "x": 344, "y": 166 }
        ]
      },
      "description": "JOE'S",
      "boundingPoly": {
        "vertices": [
          { "x": 470, "y": 142 },
          { "x": 673, "y": 151 },
          { "x": 671, "y": 190 },
          { "x": 468, "y": 181 }
        ]
      },
      "description": ""
    }
  ]
}
```

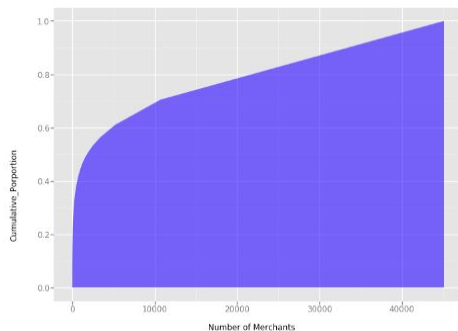
To be  
processed by  
our model

## Data exploration



- 500 merchants will cover 50% of all the receipts !
- Now what's next?

## Data exploration



- Construct a clean frequent merchant name list
- Fuzzy match the receipt words
- 50% accuracy WITHOUT machine learning model already !

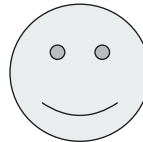
# Fuzzy match

Word in OCR JSON:

Mcdonald

Word in merchant name list:

Mcdonald's



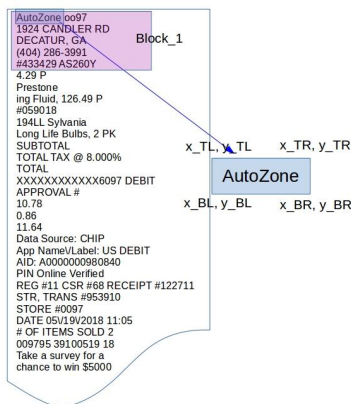
Pineapple

Mcdonald's



Some merchant names have two words or more -> Ngram

# Feature Engineering



## Features Parsed Directly from OCR File

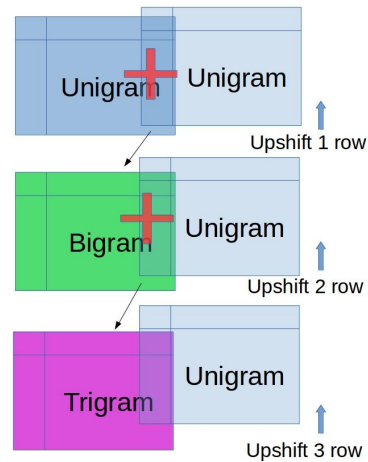
- word\_num
- x\_TL, y\_TL
- x\_TR, y\_TR
- x\_BL, y\_BR
- x\_BR, y\_BL
- block\_num
- block\_x\_TL, block\_y\_TL

## Additional Features Engineered

- Height, width
- Contains\_digit
- Contains\_alpha
- Has\_2decimals

## Generate Ngram word

- Why?  
Some merchant names have two words or more  
Want to keep features
- How?  
Shift Dataframe  
Combine  
Build features



## Train machine learning model

- Model: Random Forest, XGBoost, Logistic Regression
- Oversampling: SMOTE
- Other techniques: Gridsearch, K-fold validation, Confusion matrix, AUC, Feature importance



## Calculate recall score on receipt level

- Do training on all the receipt data
  - Some receipts have merchant name shows up more than one time
- BUT
- Only ONE merchant name will be returned for ONE receipt
  - We only consider the first one

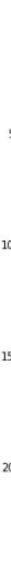


## Calculate recall score on receipt level

- $Y_{predict}$
  - Use ***pred\_proba*** to return the prediction probability of each Ngram words
  - Group words by receipt ID
  - For each receipt, ***y\_predict=1*** only for the word that having the max probability
- $Y_{true}$
  - only label (***y\_true=1***) for the first merchant name that having max height.

- 1

- 



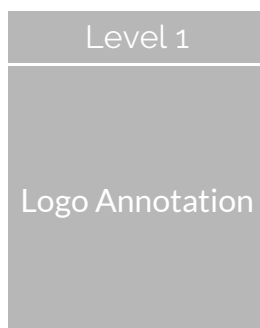




## Three Levels of Merchant Name Extraction



## Three Levels of Merchant Name Extraction



- Google OCR will catch the merchant logo.
- Approximately 20% of the OCR outputs have this information
- Use this merchant name directly if it exists



## Three Levels of Merchant Name Extraction

### Level 2

#### Fuzzy matching

- We can assume 99% of the time the matching word is the correct merchant name
- About 50% receipts can be matched
- Overlap with the ones having logo



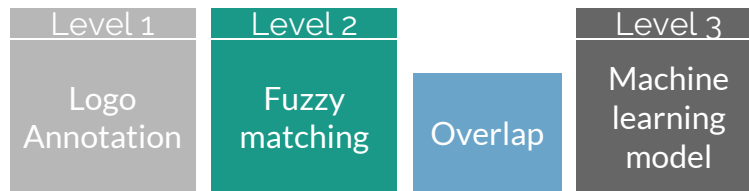
## Three Levels of Merchant Name Extraction

### Level 3

#### Machine learning model

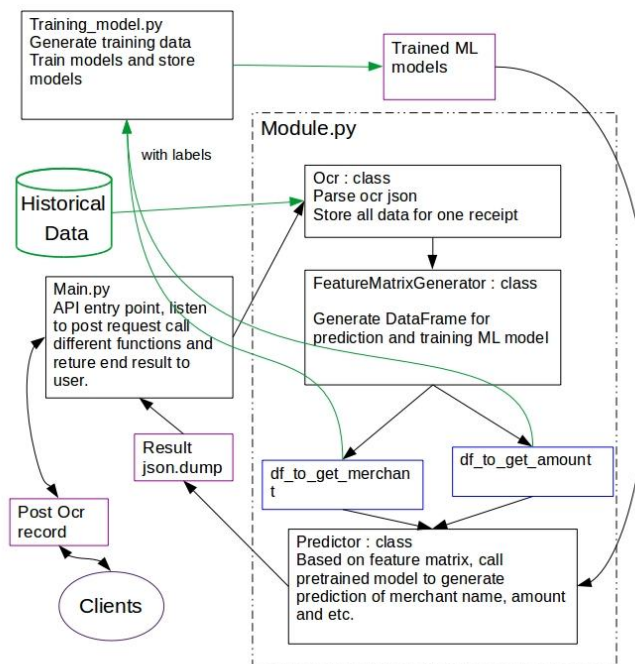
- the ngram word that having the highest probability being merchant name within a receipt will be returned
- From our model evaluation and tuning process, the best recall score is 0.6

# Three Levels of Merchant Name Extraction



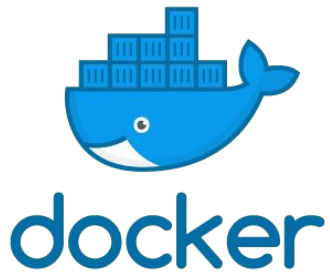
Accuracy: 20% + 50% - 15% + 0.6 \* 45% = 82%

**API**



---

## Deployment



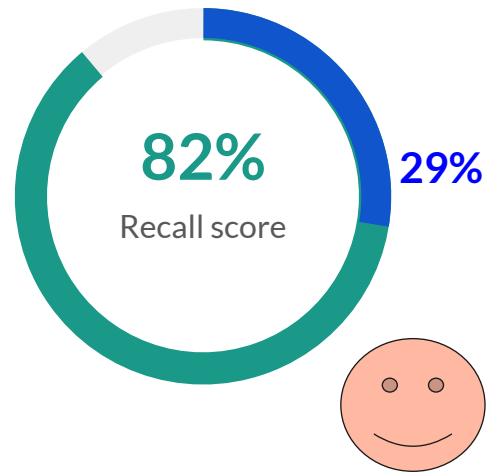
```
FROM tiangolo/uwsgi-nginx-flask:python3.6
EXPOSE 80
COPY ./app /app
RUN pip install -r requirements.txt
```

---

## Summary and next step

## Achievement

Model accuracy



## Achievement

Model deployment

Ready to  
Deploy!



## What's Next?

1. Wrap the google OCR API in the server side.
2. Balance server load.
3. Optimize cost.
4. Schedule to retrain machine learning model.
5. Update frequent merchant list automatically.

## Questions?





# Receipt Information Extraction

---

## Overview

This project is using machine learning techniques to facilitate the receipt information extraction process when users using their mobile phone to scan their receipts. Once the phone camera scans the receipt using the Apple/Android APP, detail information about the receipt will be returned in JSON format after sending the picture to Google OCR. Currently, naive rule-based method is being used to determine the merchant name, spent amount, receipt dates and other information from the JSON records. Unfortunately, the accuracy of the rule-based method is very poor. The goal of this project is to build better models to extract the correct information from different receipts using these OCR records.

## Goals

1. Build a promising machine learning models/non-machine learning models to extract correct information from the receipts.
2. Build web server API that returns the model output in JSON response.
3. Construct the whole pipeline that can be easily deployed and scaled up.

## Specifications

In this project, we mainly focus on: the accuracy of the machine learning models and model deployment spec.

### Machine Learning Model

Since of all the words in a receipt, only a small amount is useful, the data for machine learning model training is a highly **IMBALANCED** dataset. Accuracy score is useless to evaluate the performance of our model since even if predicting everything on the receipt is useless, the accuracy score is still more than 0.9. Instead, we need to use the recall score, which is calculated as:

$$Recall = \frac{True\ Positive}{All\ Positive} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Expected model recall score for merchant names, totally spent amount, tax location and receipt date: **0.85**. If recall score for merchant name prediction reaches 0.85, then 85% of the merchant names that the model prediction is the correct merchant name.

For different information in the receipt, we use different features to trained different models, some of the key features for different targets:

**Merchant name:** block\_number, text\_height, fuzzy match, ...

**Total spent amount:** is\_numeric, is\_biggest\_number, dis\_to\_word\_total, ...

**Tax:** is\_numeric, is\_smallest\_number, dis\_to\_word\_gst, ...

**Location:** block\_number, text\_height, fuzzy match, ...

**Receipt date:** is\_date, regular expression, ...

In the Minestones section, we only describe the merchant name extraction as an example.



## Web API Spec

Client POST the JSON OCR output, the server returns the model output in the response, as a part of a JSON response object.

- **Endpoint:** POST /v1/score\_receipt
- **Authentication:** Client
- **Form Values:** receipt\_ocr: Required. The OCR output returned by the Google OCR API, of the receipt that needs processing.
- **API Actions:**
  1. Validate Client authentication
  2. Apply the model to the OCR output
  3. Return model scoring results
- **Response Values:**

```
{
  'return_code': {
    0: 'success'
  },
  'data': {
    merchant: foo,
    address: bar,
    amount: 10.00,
    etc.
  }
}
```

## Milestones

### I. Data Exploration

Receipt OCR records in 3 months, about 20GB, 300 thousands receipts.

User response from the APP, manual merchant name corrections of the receipts.

Contents below is how the OCR record of a receipt looks like in JSON format:

```
{
  "textAnnotations": [
    {
      "locale": "en",
      "boundingPoly": {
        "vertices": [
          { "x": 44, "y": 122 },
          { "x": 761, "y": 122 },
          { "x": 761, "y": 1257 },
          { "x": 44, "y": 1257 }
        ]
      },
      "description": "CRABBY JOE'S DONNTOWN\n276 DUNDAS ST\nLONDON, ON N6B1T6\n5196454880\nSALE\nServer #: 000200\nMID: 5800203\nTID: 006\nBatch #: 417\n04/29/18\nAPPR CODE: 04239\nVISA\nREF#: 00000032\n210141\nChip\n7932\nAMOUNT\nTIP\nTOTAL\n$27.09\n$4.06\n$31.15\nAPPROVED\n",
      "boundingPoly": {
        "vertices": [
          { "x": 153, "y": 122 },
          { "x": 315, "y": 129 },
          { "x": 313, "y": 173 },
          { "x": 151, "y": 166 }
        ]
      },
      "description": "CRABBY"
    },
    {
      "boundingPoly": {
        "vertices": [
          { "x": 345, "y": 133 },
          { "x": 445, "y": 137 },
          { "x": 443, "y": 170 },
          { "x": 344, "y": 166 }
        ]
      },
      "description": "JOE'S"
    },
    {
      "boundingPoly": {
        "vertices": [
          { "x": 470, "y": 142 },
          { "x": 673, "y": 151 },
          { "x": 671, "y": 190 },
          { "x": 468, "y": 181 }
        ]
      },
      "description": "DONNTOWN"
    }
  ]
}
```

```

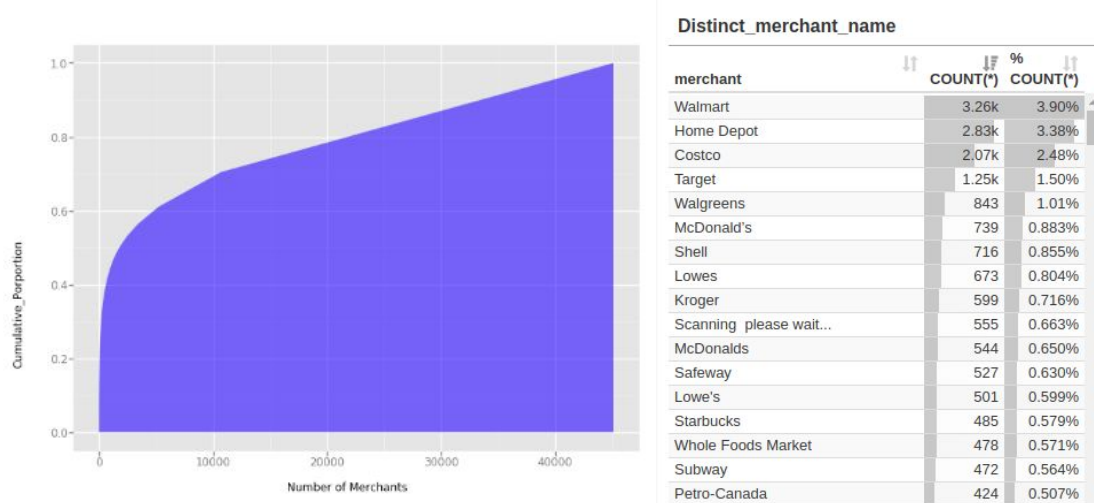
ly":{"vertices":[{"x":263,"y":176}, {"x":323,"y":181}, {"x":320,"y":218}, {"x":260,"y":213}], "desc
ription":"276"}, {"boundingPoly":{"vertices":[{"x":352,"y":181}, {"x":498,"y":194}, {"x":496,"y":22
6}, {"x":349,"y":213}], "description":"DUNDAS"}, {"boundingPoly":{"vertices":[{"x":526,"y":198}, {"
x":569,"y":202}, {"x":566,"y":233}, {"x":523,"y":229}], "description":"ST"}, {"boundingPoly":{"vert
ices":[{"x":220,"y":227}, {"x":380,"y":234}, {"x":378,"y":274}, {"x":218,"y":267}], "description":"
LONDON,"}, {"boundingPoly":{"vertices":[{"x":409,"y":237}, {"x":458,"y":239}, {"x":457,"y":272}, {"x
":408,"y":270}], "description":"ON"}, {"boundingPoly":{"vertices":[{"x":486,"y":245}, {"x":603,"y"
":250}, {"x":601,"y":286}, {"x":484,"y":281}], "description":"N6B1T6"}, {"boundingPoly":{"vertices":
[{"x":322,"y":278}, {"x":513,"y":295}, {"x":510,"y":331}, {"x":319,"y":314}], "description":"519645
4880"}, {"boundingPoly":{"vertices":[{"x":341,"y":401}, {"x":491,"y":401}, {"x":491,"y":447}]. . . . .
...

```

User response is stored in .csv files with the following information:

receipt\_id, receipt\_date, amount, tax, merchant, location, currency, date\_created.....

For merchant name, we did a histogram of merchant names that the user provide corrections (Even lots of the “user corrections” is not correct). As we can see from the graph, 500 merchants will cover half of the receipts.



This gives us an idea:

We can construct a clean frequent merchant name list to fuzzy match the receipt words to build correct label to train machine learning model & return correct machine name to the client without using machine learning model that is less accurate.

## II. Fuzzy Matching Merchant Name

[Fuzzy string matching](#) is the technique of finding strings that match a pattern approximately (rather than exactly).

E.g. the scanning returns

### Mcdonald

We construct a promising clean merchant list that has the name

### McDonald's

The fuzzy matching score of these two words will be very high. We will consider the receipt having the word Mcdonald is a receipt from McDonald's.

Since some merchant names have two words or more, we need to make ngram using each single word in the receipt. Details refer to section IV below.

[FuzzyWuzzy](#), a Python package, uses Levenshtein Distance to calculate the differences between sequences in a simple-to-use package. Specifically, we use the token sort ratio. Token sort ratio\_score is a measure of the strings similarity as an int in the range [0, 100]. The threshold to determine matching or not is a tuning parameter, currently we set it as 90 for all cases.

## III. Feature Engineering

By parsing JSON, we are able to get base features of words in a receipt, such as:

'x\_TL': top level horizontal coordinate in pixel

'y\_TL': top level vertical coordinate in pixel

'x\_TR','y\_TR', 'x\_BR', 'y\_BR', 'x\_BL', 'y\_BL', 'block\_num', 'block\_x\_TL', 'block\_y\_TL', 'block\_x\_TR', 'block\_y\_TR', 'block\_x\_BR', 'block\_y\_BR', 'block\_x\_BL', 'block\_y\_BL',.....

This information is stored in a master Dataframe that to be further processed to generate different feature matrices to predict different information in a receipt.

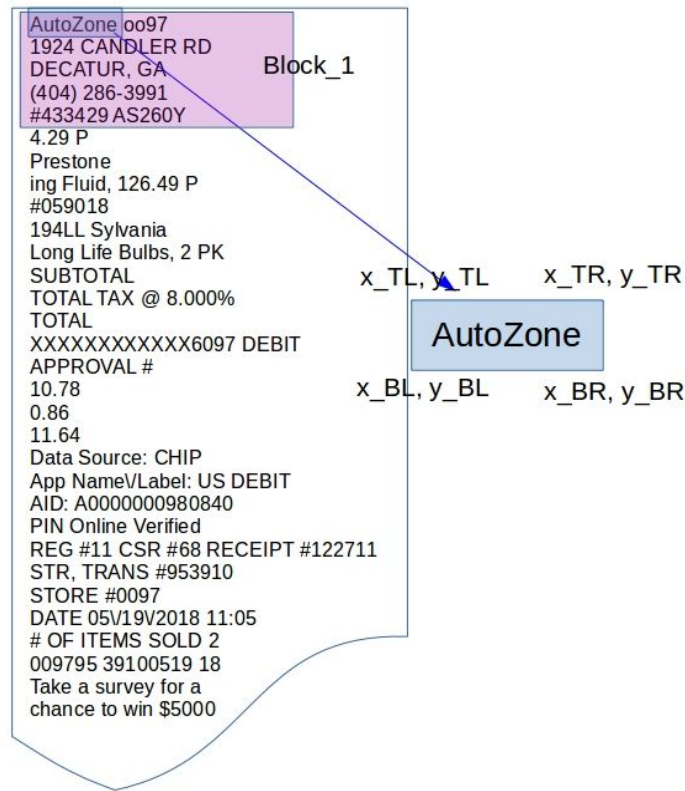
Specifically for merchant name prediction, we can built the features like:

'width': width of the word in pixel

'height': height of the word in pixel

'block\_width', 'block\_height', 'has\_digit', 'is\_alpha', 'word\_len', 'has\_cap', 'all\_cap'...

Block information is generated by Google OCR, later we found out this is a strong predictor to predict merchant name.

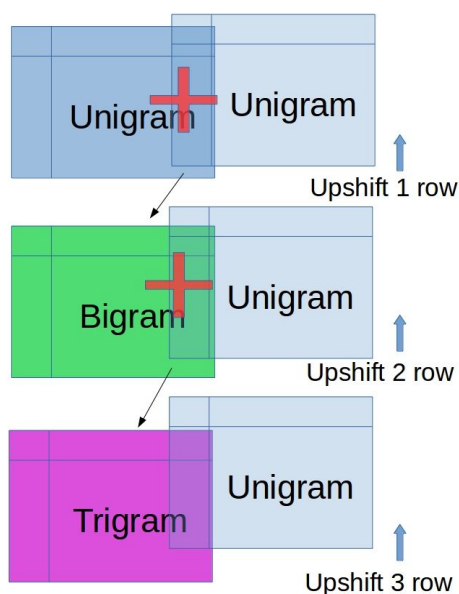


#### IV. Generate Ngram word

Since some merchant names have two words or more, we need to make ngram using the single word in the receipt before fuzzy matching.

After parsing the JSON, we obtain a Dataframe that having each word with its coordinate, block and other information as one row. When building Ngram word list, we don't want to lose other information, so we make Ngram words by shifting the Dataframe of unigram up by one row and combine with the original Dataframe.

The figure below demonstrate the process. The coordinates and block information of an Ngram word need to be specified carefully according to the properties of its candidates. Details will not be shown here.



## V. Machine Model Training, Tuning and Selection

**Techniques:** Gridsearch, K-fold validation, Confusion matrix, AUC, Feature importance, and more.

**Model tried:** Random forest, Gradient boosting, XGBoost, Logistic regression

**Sampling:** Because the dataset is IMBALANCED, oversampling is required, [SMOTE](#) can be used.

**Calculate recall score in receipt level:**

We do training on all the receipt data, but in reality, only **ONE** merchant name will be returned for **ONE** receipt. So the recall score has to be calculated on receipt level.

After training the machine learning model, use **pred\_proba** to return the prediction probability of each Ngram words in the test set. Then group words by receipt ID. For each receipt, **y\_predict=1** only for the word that has the max probability.

Since some receipts have merchant name shows up more than one time, we only label (**y\_true=1**) for the first merchant name that having max height.

```

classification_report - testing - threshold: 0.50
              precision    recall  f1-score   support

    0         1.00      1.00      1.00   1491158
    1         0.47      0.64      0.55     2551

 avg / total         1.00      1.00      1.00   1493709

```

```

classification_report - testing - threshold: 0.40
              precision    recall  f1-score   support

    0         1.00      1.00      1.00   1491158
    1         0.42      0.69      0.52     2551

 avg / total         1.00      1.00      1.00   1493709

```

```

classification_report - testing - threshold: 0.30
              precision    recall  f1-score   support

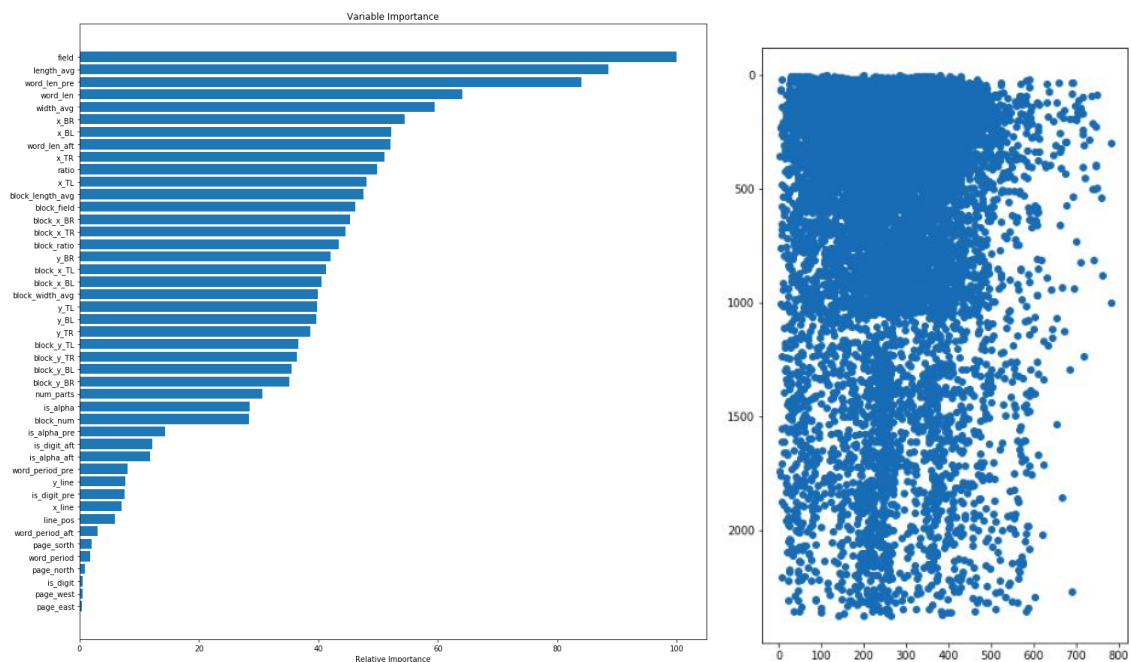
    0         1.00      1.00      1.00   1491158
    1         0.36      0.73      0.48     2551

 avg / total         1.00      1.00      1.00   1493709

```

### Feature importance (left figure below):

Some algorithms can return feature importance (e.g. random forest).



### Merchant location statistic (right figure above):

We found out most of the merchant names are located on the top left of the receipts.

## VI. Three Levels of Merchant Name Extraction to Improve Accuracy

We use three levels mechanism to increase the accuracy of returning true merchant name of a receipt.

1. **Logo Annotation:** Google OCR will catch the merchant logo on a receipt and generate the correct merchant name. Approximately 20% of the OCR outputs have this information. We use this merchant name directly if it exists.
2. **Fuzzy matching:** Fuzzy matching is not only good to do labeling for machine learning model training, but also can be used to return correct merchant name. If there is a high matching score between a word in the receipt and the frequent merchant name, 99% of the time the matching word is the correct merchant name. This serves as level two in our scheme. About 50% receipts can be matched, these overlap with the ones having a logo.
3. **Machine learning model:** If there is no logo detected nor fuzzy matching, we will use the machine learning model to return the merchant name, the ngram word that having the highest probability being merchant name within a receipt. From our model evaluation and tuning process, the best recall score is

~60%

using random forest model.

4. **Overall score:** By using three-level scheme, the accuracy is  
 $20\% \text{ (Logo Annotation)} + 50\% \text{ (Fuzzy matching)} - 15\% \text{ (Overlapping)} + 0.6 * 45\% =$

~82%

## VII. Build Client API

The Client API has three main components: Flask API entry, module and machine learning model off-line training program. The flowchart below demonstrate the connections among all the components.

API pipeline flow:

```
@app.route("/getresult", methods = ['POST'])
def getResult():
    # decode the post request stream to get str
```

```

ocr_json = request.data.decode('utf-16')
# load the decoded string to dictionary
ocr_dict = json.loads(ocr_json)
# print this message for debug only
if isinstance(ocr_dict, dict):
    return_code = 1
else:
    return_code = 0
# Parse the OCR info
this_ocr = Ocr()
this_ocr.read_ocr_dict(ocr_dict)
# Generate DataFrame to predict merchant name
this_feature_generator=FeatureGenerator(this_ocr)
merchant_df = this_feature_generator.get_merchant_df(with_label =
True)
print(merchant_df)
# Generate DataFrame to predict amount...
# Start to predict
predictions = Predictor()
predictions.predict_merchant(this_feature_generator, merchant_df, '')
# predict amount...
#build the response
response_dict = {"return_code": return_code,\
    "data": {"merchant": predictions.merchant, "amount":
predictions.amount, "tax": predictions.tax}}
return json.dumps(response_dict)

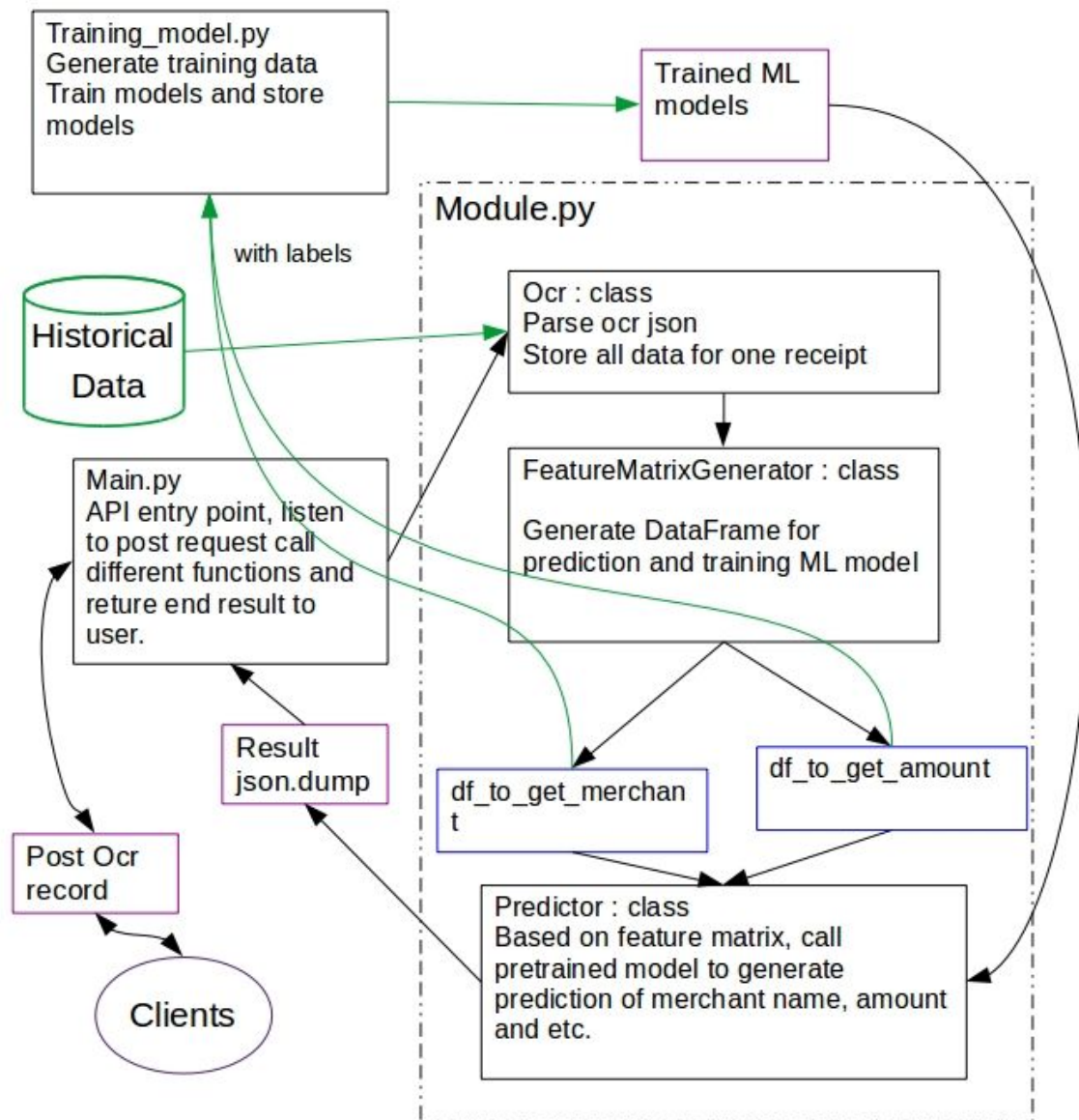
```

**Flask API entry:** Web server wrapper only.

**Modules:** Contains class Ocr that parses the OCR JSON receipt record, FeatureGenerator that generates different features table to extract different information from a receipt, and Predictor that returns the prediction of receipt information that we need.

**Model training and evaluation notebook:** A Jupyter notebook that utilizes classes in Modules to build training and testing dataset to train machine learning model offline. Notebook environment provides us a better way to perform visualization when doing the model evaluation.





## VIII. Deployment and Version Control

**Docker:** This API will be hosted using a Docker container that can be shifted anywhere. Dockerfiles to build the container is as follow:

```
FROM tiangolo/uwsgi-nginx-flask:python3.6
EXPOSE 80
COPY ./app /app
RUN pip install -r requirements.txt
```

Note: requirements.txt is the necessary Python package list.

**BitBucket:** Bitbucket is a web-based version control repository hosting service owned by Atlassian, for source code and development projects that use either Mercurial or Git revision control systems. In this project we use Git. BitBucket also offers deployment tools which are easy to use.



#### Deployment tracking with Bitbucket

Enable deployment tracking to view the status of your deployments and key information about every commit you deploy to your test, staging and production environments.

Deployments is powered by Pipelines, which you can configure below. Your current plan includes 50 free build minutes per month.

#### Choose a language template

Deployments is configured via the bitbucket-pipelines.yml file.

To run your deployment pipeline, choose a language template, edit it, then commit the file.



**AWS Elastic Beanstalk (EB):** This will run a containerized model serving application, reachable by the client via API calls. EB is great as it automatically performs load balances and scales up or down. More info: [Deploying a Flask Application to AWS Elastic Beanstalk](#)

**Version Control Procedure:** Master branch for production, dev branch for development, each contributor create his/her own branch based on dev branch. When merge to dev branch, code review and unit test will be performed.

## Summary and Future Works

In summary, the merchant name extraction model can reach about 80% accuracy, total spent amount can achieve 85% accuracy. This project is ready to deploy and use in a real production environment. Next steps include but not limited to:

1. Wrap the Google OCR API in the original App server to get rid of the POST request.
2. Balance the server load.
3. Optimize cost.
4. Schedule to retrain machine learning model.
5. Update frequent merchant list automatically.
6. Write more unit test code to make code review easier.
7. Continue support to resolve issues.