**Assignment 2**

**Database as a Service**

**Team 1**

**Vishal Satam**

**Manasi Dalvi**

**DOCKER & Execution Instructions**

The required files have been packaged into 2 Docker images and are present on Docker Hub with the image names as vishalsatam1988/wrangleanduploadassignment2 and vishalsatam1988/createdbanduseapi. The manual execution steps are given below.

**Note :** The docker container runs in the **UTC timezone**.

If you encounter any Memory Error issues or segmentation fault. Please increase the RAM of your docker virtual machine to minimum 4 GB.

**Docker image - vishalsatam1988/wrangleanduploadassignment2**

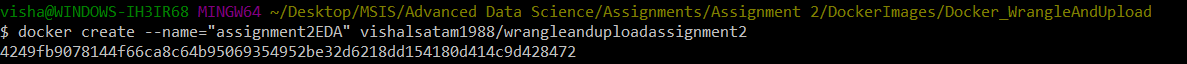
1. Step 1 : Pull the image

The docker image is present on the docker hub and is available to pull using the following command

docker pull vishalsatam1988/wrangleanduploadassignment2

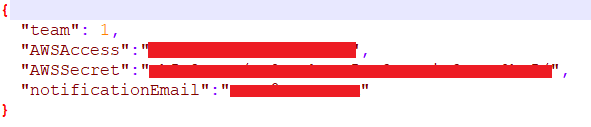
1. Step 2 : Create the container

docker create --name="assignment2EDA" vishalsatam1988/wrangleanduploadassignment2



1. Copy your config file. (The name of the file has to be **config.json**)

Sample contents of config.json file



docker cp <local file path> <containername>:/src/assignment2/config/

docker cp <path>/config.json assignment2EDA:/src/assignment2/config/



1. Start the container

docker start <containername>

docker start -i assignment2EDA

1. Commit the container if you want to see logs otherwise invoke the following command to check the jupyter notebooks.

Password to open jupyter notebook : keras

Command to commit - docker commit assignment2EDA vishalsatam1988/wrangleanduploadassignment2

docker run -it -d --name “assignment2EDAjupyter” -p 8888:8888 vishalsatam1988/wrangleanduploadassignment2 /bin/bash -c 'jupyter notebook --no-browser --allow-root --ip=\* --NotebookApp.password="$PASSWD" "$@"'

**Docker image - vishalsatam1988/createdbanduseapi**

1. Step 1 : Pull the image

The docker image is present on the docker hub and is available to pull using the following command

docker pull vishalsatam1988/createdbanduseapi

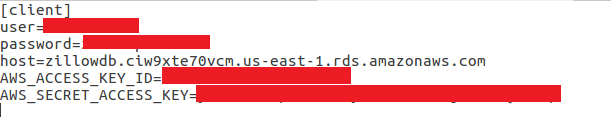
1. Step 2 : Create the container

docker create --name="assignment2RDS" vishalsatam1988/createdbanduseapi



1. Copy your config file. (The name of the file has to be **config.txt**)

Sample contents of config.txt file



docker cp <local file path> <containername>:/src/assignment2/config/

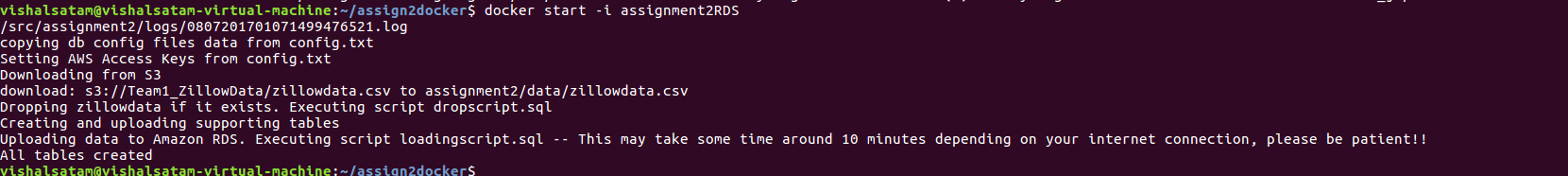
docker cp <path>/config.txt assignment2RDS:/src/assignment2/config/



1. Start the container

docker start <containername>

docker start -i assignment2RDS



1. Commit the container if you want to see logs otherwise invoke the following command to check the jupyter notebooks.

Command to commit - docker commit assignment2RDS vishalsatam1988/createdbanduseapi

docker run -it -d --name “assignment2RDS” -p 8888:8888 vishalsatam1988/createdbanduseapi /bin/bash -c 'jupyter notebook --no-browser --allow-root --ip=\* --NotebookApp.password="$PASSWD" "$@"'

Password to open jupyter notebook : keras

**Overview :**

Zillow has uploaded their properties\_2016.csv dataset for a competition and we are using this data to create a Data as a Service application.

The project involves 4 phases.

1. Exploratory Data Analysis on the raw data.
2. Data Wrangling to create clean data
3. Upload data to a Cloud Database (Amazon RDS)
4. Create a REST application provide an API to access this data.

Zillow data consisting of list of real estate properties in three counties (Los Angeles, Orange and Ventura, California) data for the year 2016.

Additional description files are provided with their corresponding ID’s in the main file. We have created separate tables for the same. The list of tables :

**zillowdata** - Main table containing the clean data created from the properties file downloaded from kaggle

**airconditiontype** - Table containing descriptions of Air Condition Types

**heatingsystemtypeid** - Table containing descriptions of Heating System Types

**propertydescid** - Table containing Property Type Descriptions

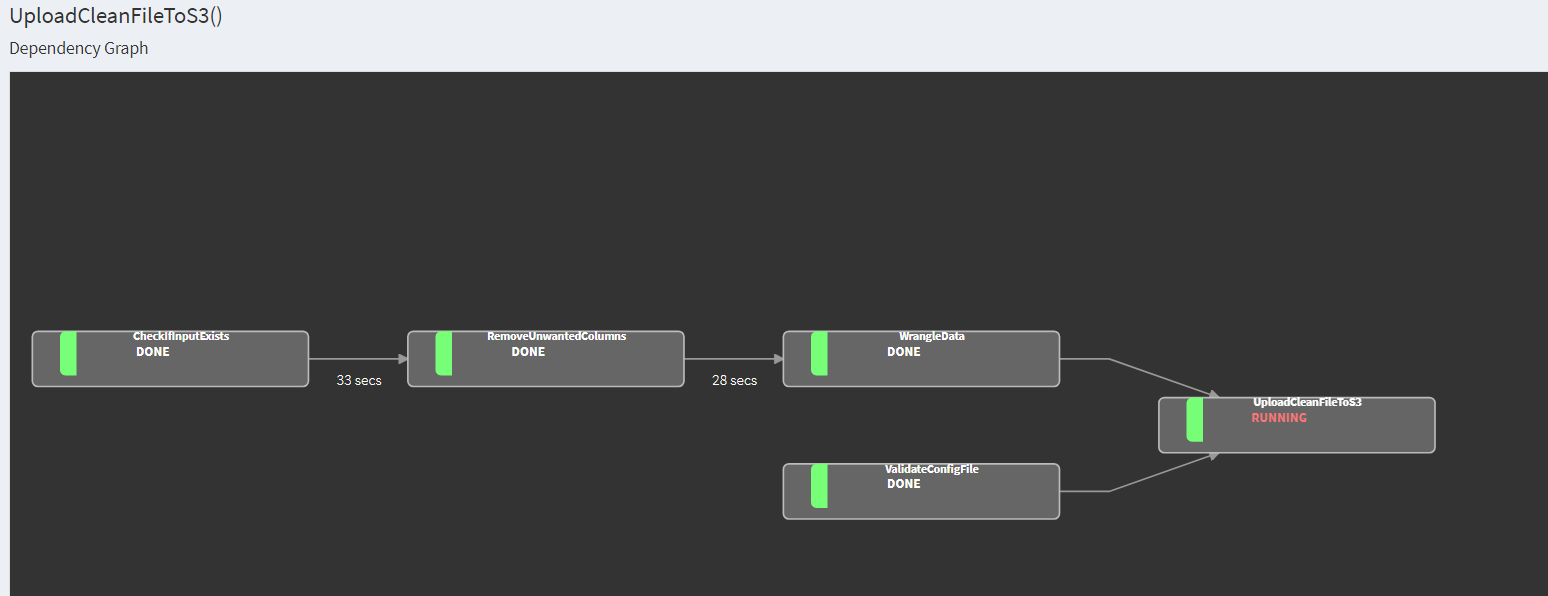
**Exploratory Data Analysis**

The exploratory data analysis has been covered in detail in the Jupyter notebook ZillowDataEDA.ipynb which is submitted on github as well as included in the Docker image.

We can see that the dataset contains a lot of missing values Few columns are repeated and can be used in place of the other. Mixed Datatypes exist in the data, we have to clean these out.

**Data Wrangling**

The Data Wrangling is performed using a Data pipeline which has been developed using Luigi as shown in the task dependency graph below.



The pipeline executes in 5 tasks in a sequential manner.

1. **CheckIfInputExists** – This task checks if the raw data file exists in the given path. This is to ensure that we don’t run into path errors for the next tasks.
2. **RemoveUnwantedColumns** – This task is responsible for reducing the dataset for processing by the next task. We had seen that this dataset contains many missing values. This task removes all the columns for which 90% or more of the data is missing.
3. **WrangleData –** This is the heart of the wrangling pipeline. This is the main task that performs the data wrangling. The following processing is carried out on the reduced dataset received from the RemoveUnwantedColumns task

* Replace fireplaceflag by 0 and 1 instead of True and False to make this column numeric. We are also replacing the fireplacecnt to 1 wherever the fireplaceflg was True in order to keep the maintain integrity. After this we remove the fireplaceflag column.

This is done because we can infer if a fireplace exists or not from the fireplacecnt value.

Missing fireplacecnt are kept as Null because almost 80% data is missing. Replacing with anything will affect the integrity of the data.

* Latitude and Longitude columns have been multiplied by 106. We divide the values present in this column by 106 to correct these values. Rows for which these values are missing will be removed because we cannot use these values.
* We group the fields (bathroomcnt, bedroomcnt, roomcnt, heatingorsystemtypeid, buildingqualitytypeid) by the propertylandusetypeid and fips and replace the missing values for the corresponding groups with the means of their groups.

What this means is that for every missing value, we take the mean value of this column based on the county (fips) and the property type in that county (residential / commercial, etc) and then replace the mean value.

We are doing this because, let’s say for instance we have bathroomnt. **A commercial building in a county will not have the bathroomcnt as that of a residential property.** This is why, we decided to take the average based on the property type and the county that the parceled belongs to.

* If there are still missing values for the above columns, we replace them with 0
* Even though the regionidzip contain obfuscated values, we still remove the outliers. As mentioned in the jupyter notebook, there are values that are greater than 99999. These will have to be replaced by some value. Since there are only 421 outlier values, we will replace them by the most common zip which is 96987.

1. **ValidateConfigFile** – This step validates the config file to fetch the AWS credentials and Team which is used by the next task.
2. **UploadCleanFieToS3** – This task collects the cleaned data and uploads it to the S3 bucket.

**Upload Dataset to Cloud to create the Database:**

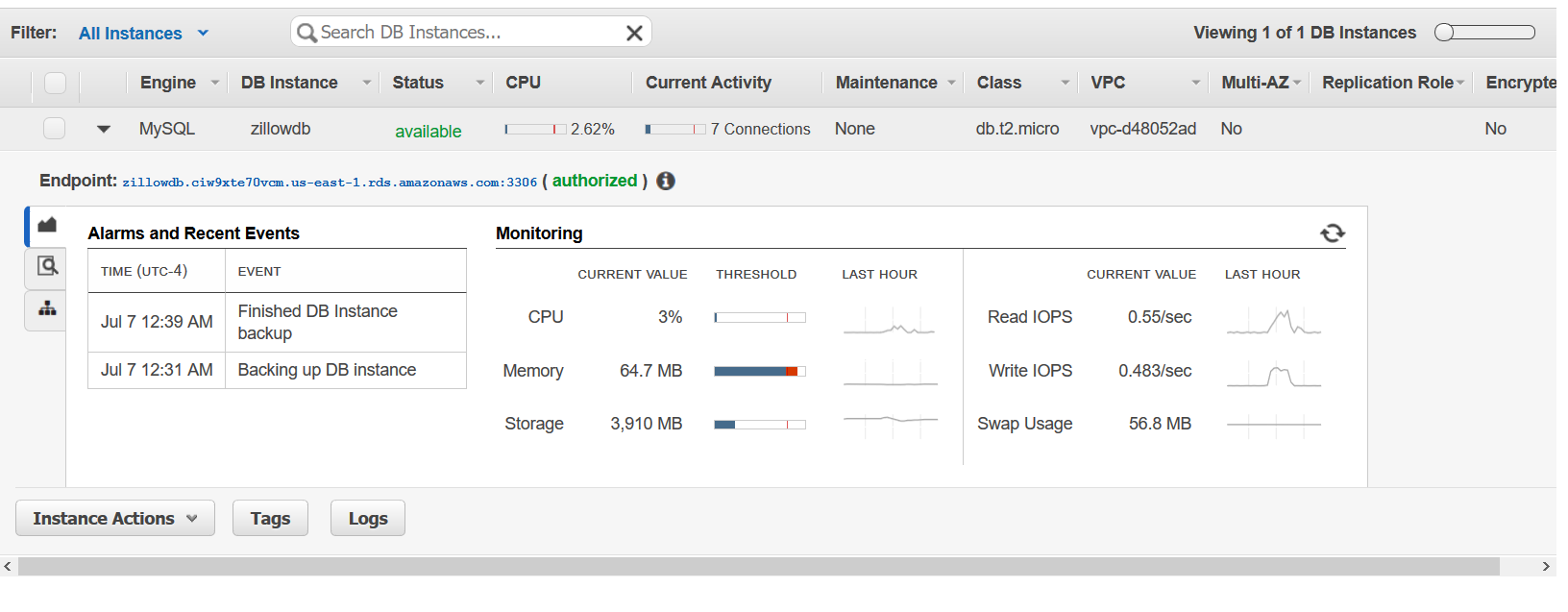
This step uploads our data file which is present on the S3 bucket to the cloud database.

We have been assigned to work on Amazon RDS. A MySQL instance has been created on Amazon RDS and the scripts to create and load the dataset are provided as a docker image.

**Creating the database on Amazon RDS:**

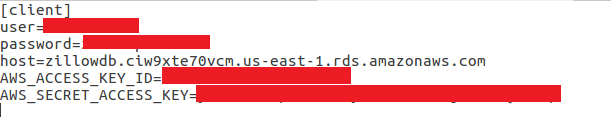
The database has been created on Amazon RDS as a MySQL instance and is operating on the host URL :

**zillowdb.ciw9xte70vcm.us-east-1.rds.amazonaws.com:3306**

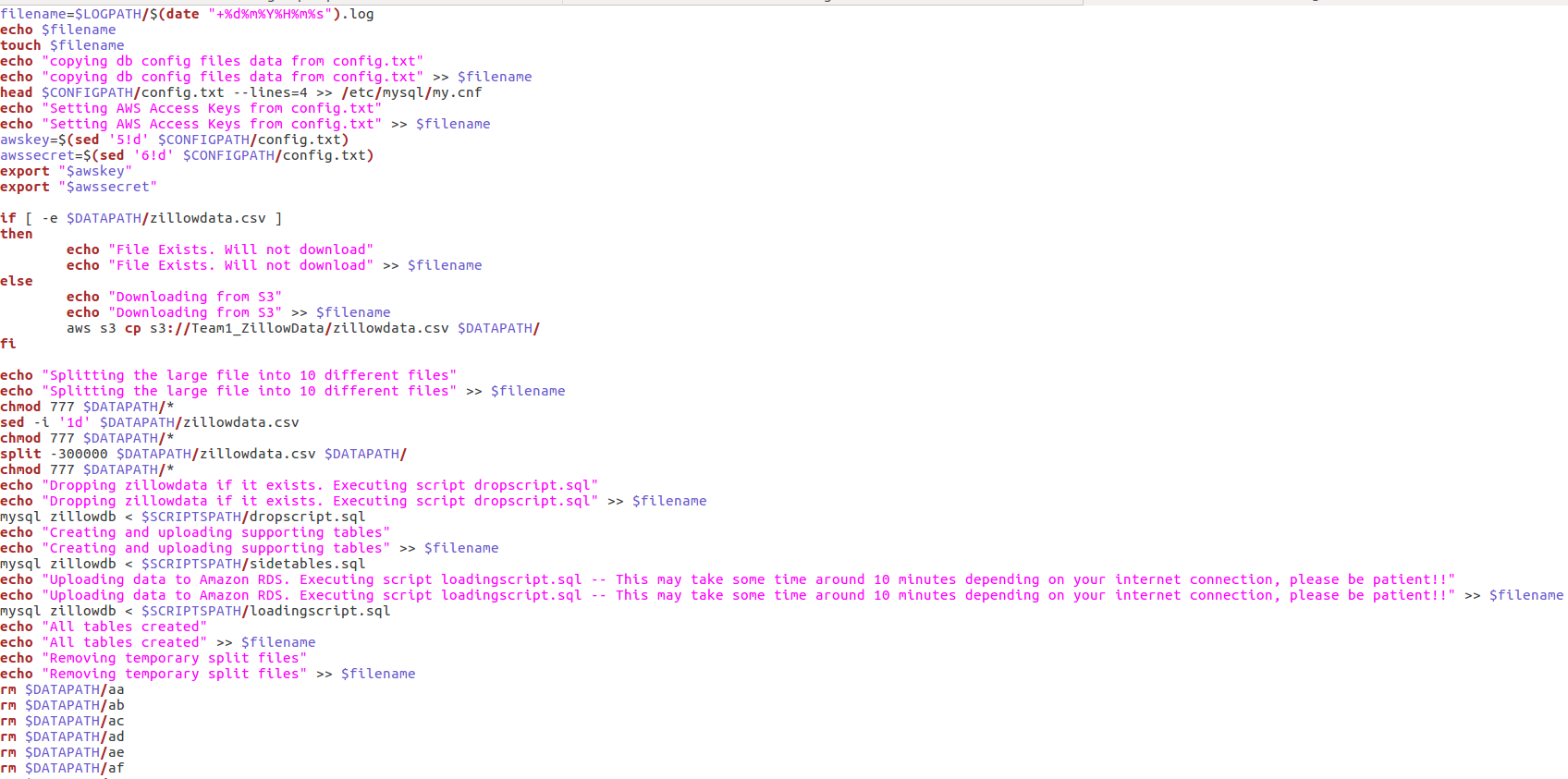


The docker image **vishalsatam1988/createdbanduseapi** contains command line scripts to download the dataset file from s3 using Amazon CLI.

A config.txt file exists in the docker image which provides the credentials to access the database and the amazon s3 bucket. Database credentials have been removed from the docker image intentionally but we can copy our config.txt file into the docker image as explained in the execution steps above.

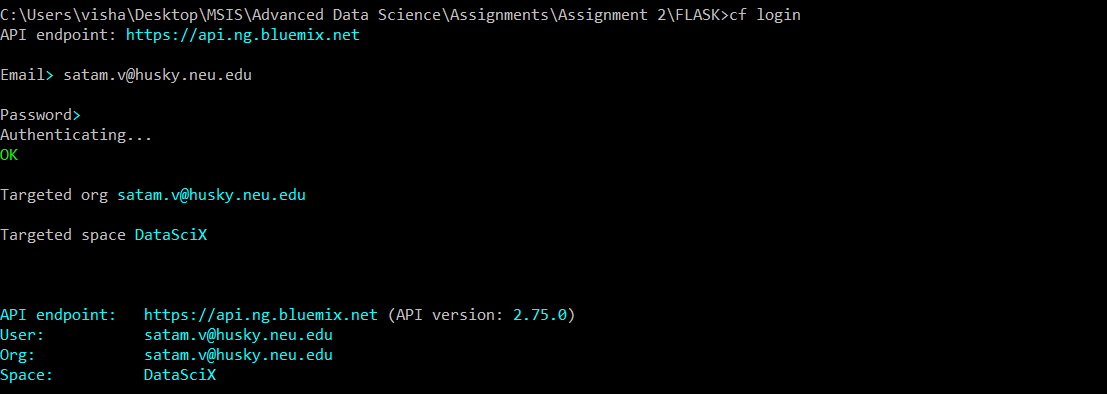


The following shell script file reads the config file and sets environment variables in the docker image which are required to execute the sql files successfully.

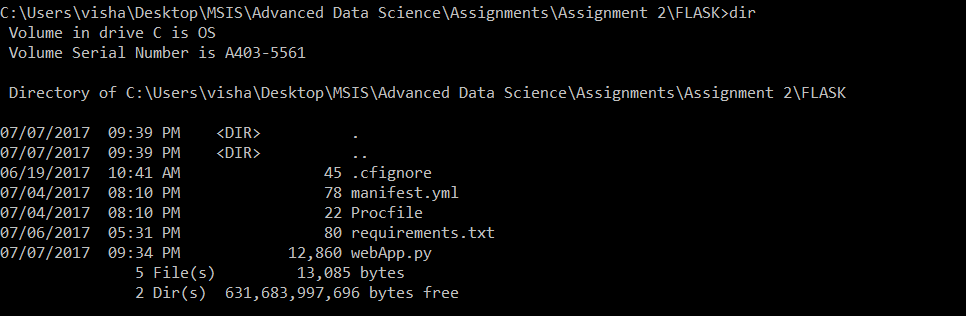


**Deploying FLASK Rest API on IBM Bluemix using Cloudfoundry CLI**

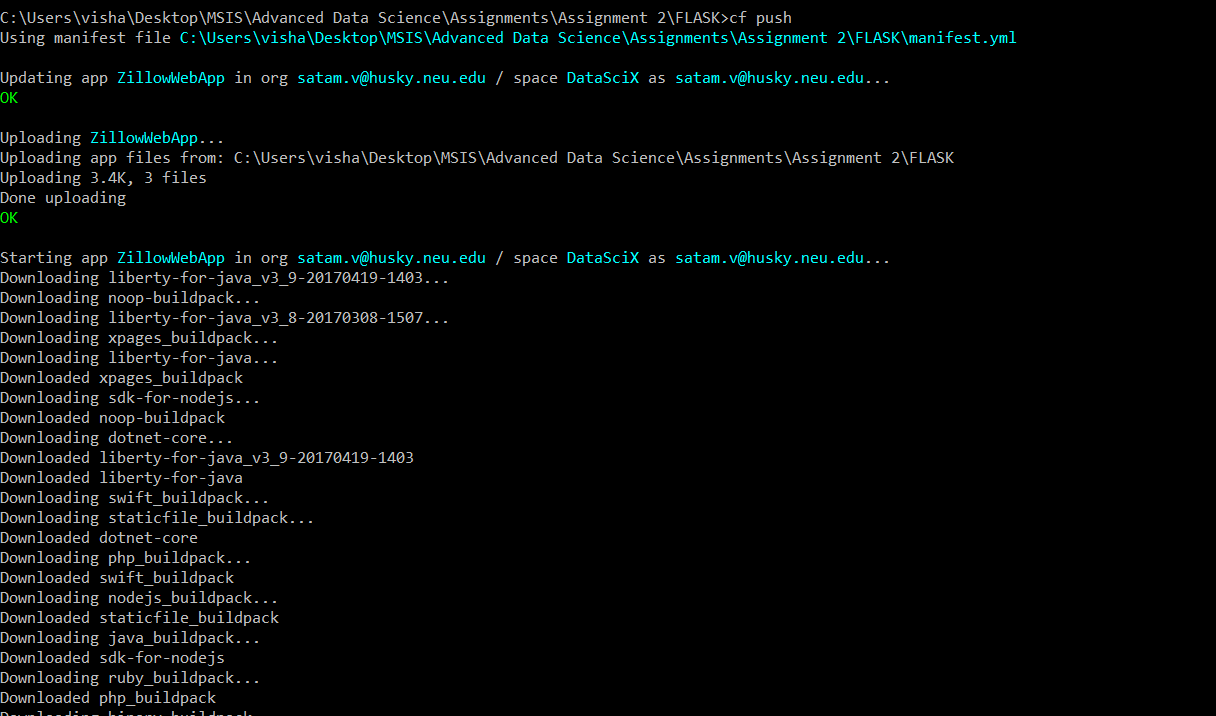
Logging in

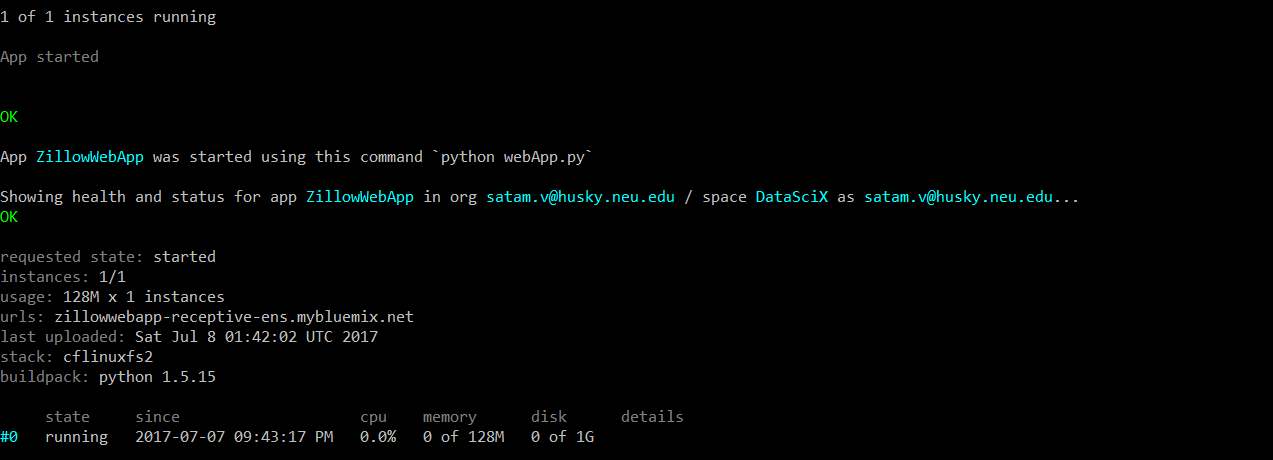


Directory Structure



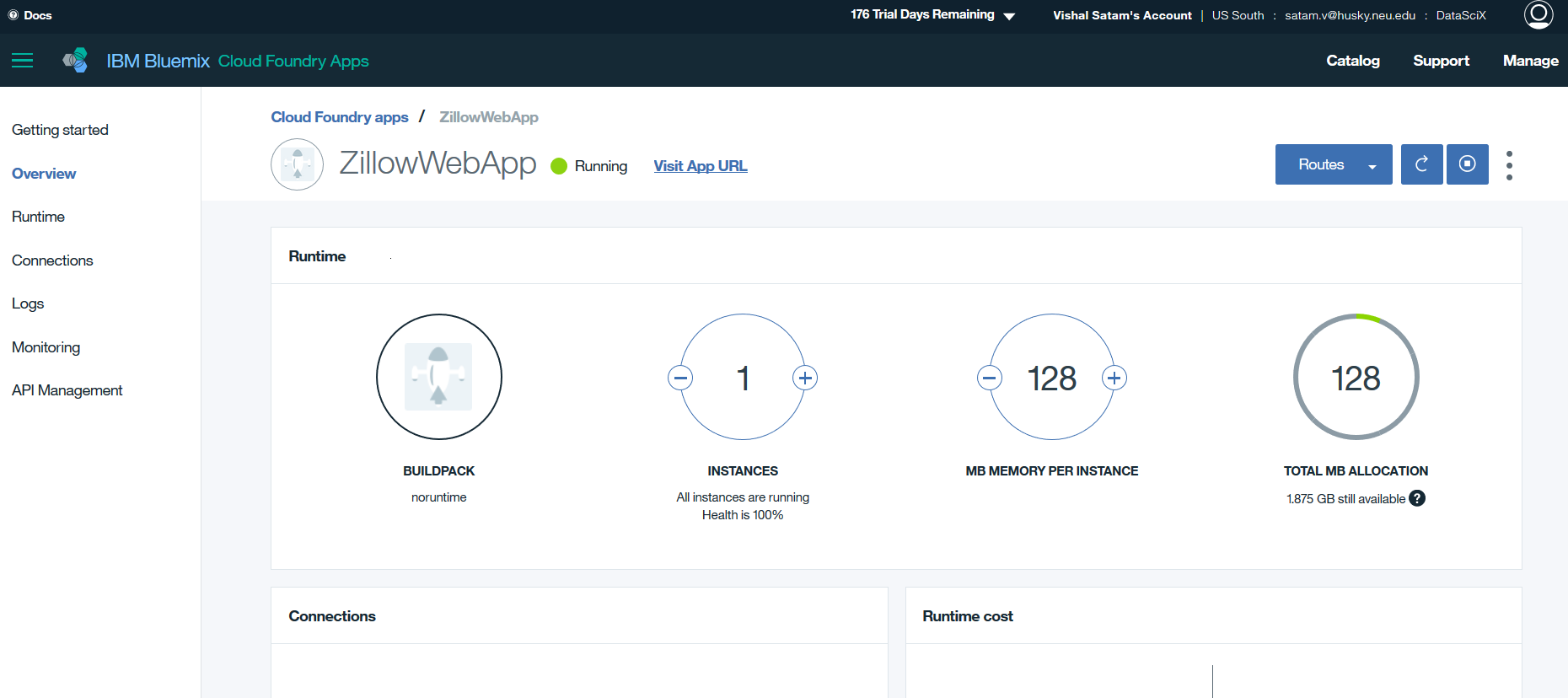
Pushing app to bluemix





**The flask application is deployed and running on the URL :**

**https://zillowwebapp-receptive-ens.mybluemix.net**



**API Details**

**Search Service API**

**Search parameter**:

<http://zillowwebapp-receptive-ens.mybluemix.net/search?zipcode=98698>

We can query the api using the search url to perform a search based on the following parameters.

**API URL**

Parameters For the API (Atleast one parameter is required)

* zipcode (int)
* parcelid (int)
* bathroom (int/float)
* totalarea (float)
* bedroom (int/float)
* yearbuilt (int)
* pool (int)
* heating (int)
* storeys (int)
* propertytype (int)
* aircondition (int)
* start (int) greater than 0 - This is the offset and indicates the index for the next set of results.

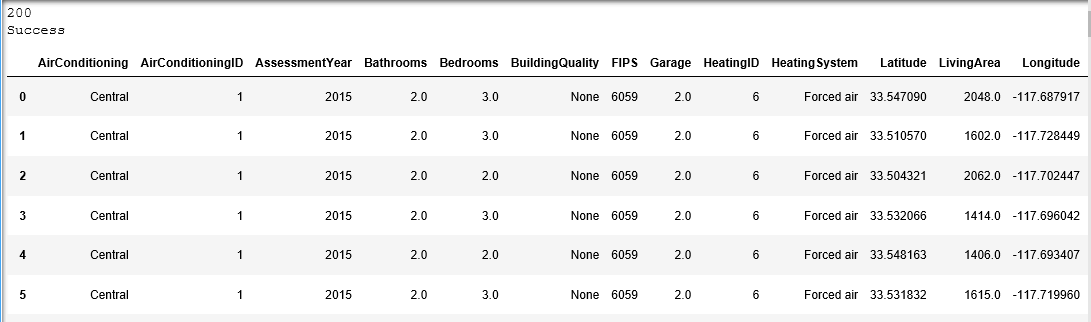
Values are returned in json and can be used directly in a dataframe.

**Example 1: query using zipcode, bathroomcount, heating:**

The list of search parameters is provided as ‘params’ dictionary and can be modified. At least one parameter is mandatory. Since the dataset size is huge, this search API only returns 100 records at a time. You can request for the next set of 100 records using the start parameter which acts as the offset in the database query.

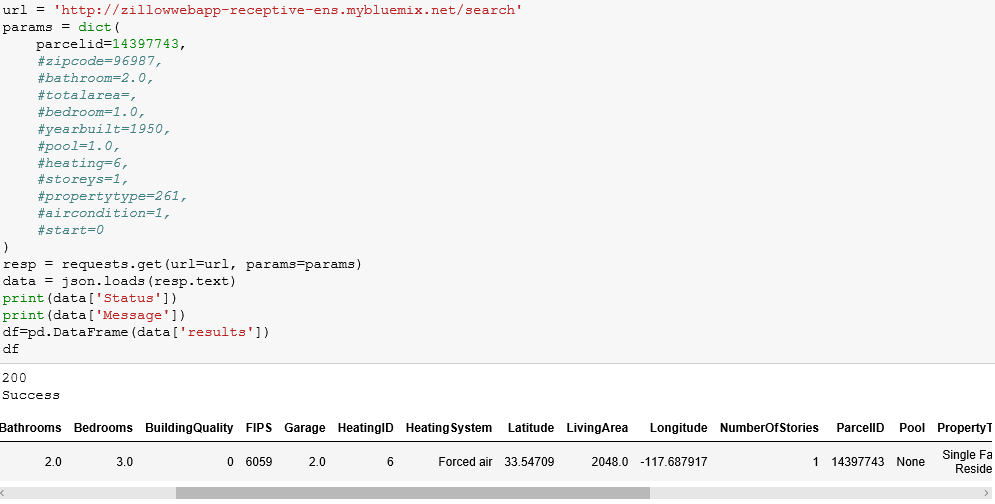


The result (show below) of the above search query is returned directly in a dataframe and can be used for further analysis.



**Example 2**: **Query on parceled (property id):**

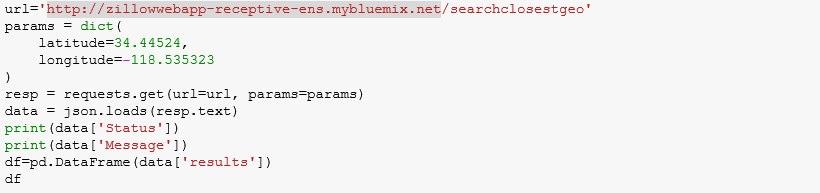
Return a single row corresponding to the inputted parceled.



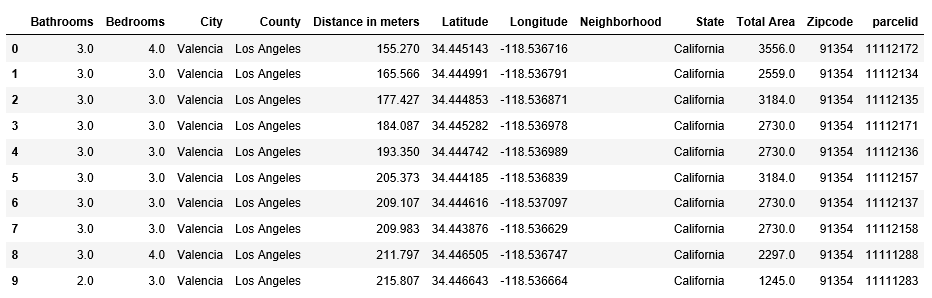
**Geospatial search for ten closest homes**

To display 10 closest properties from a given location requires latitude and longitude of the location. The required parameters are given as a dictionary and can be modified for searches.

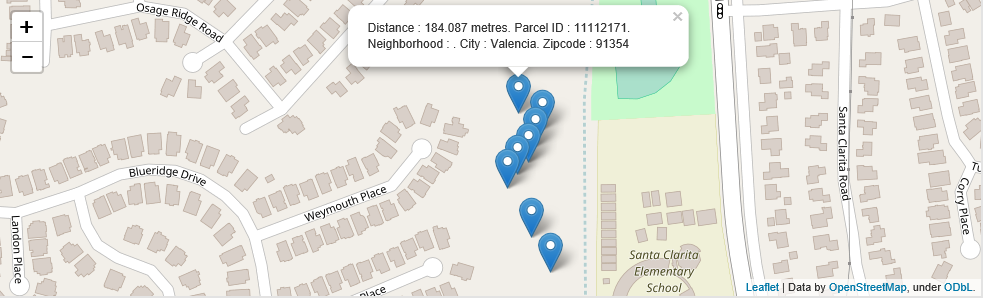
Following image shows geo search for latitude = 34.44524 and longitude -118.535323



The following image shows the ten closest properties for the above given location sorted by distance from the entered location.



The result can be seen on an interactive leaflet map. Python Folium and Vincent/Vega libraries are used for visualizations.



Map plotting the ten properties closest from a given location

**Analytics Services**

The columns AirconditiontypeID, PropertyLandDescriptionTypeID,HeatingSystemTypeID have corresponding description columns which are uploaded as additional tables.

**Types and Description Api** :

Provides search results for the mentioned features with its description from the corresponding tables.

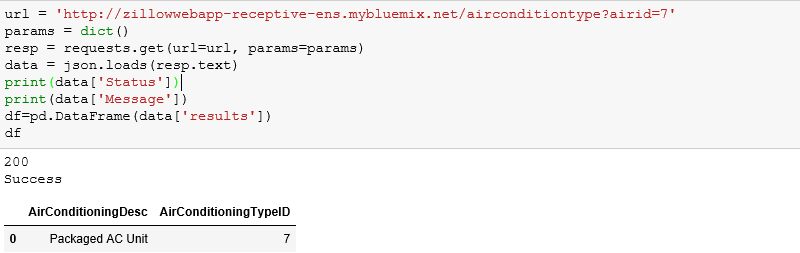
It can be queried with a single ID parameter for specific search or can return the entire set of values for all available IDs.

**Example: Air-condition system search**

Search by ID returning a single result:

Search parameters : <http://zillowwebapp-receptive-ens.mybluemix.net/airconditiontype?airid=7>

We can query for a single result for specific type searches based on ID.

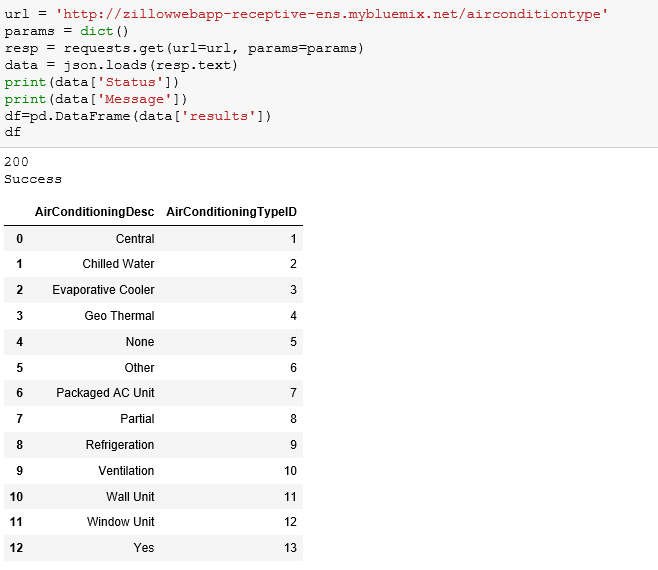


Query for single value

**Search by all values:**

Search parameters: <http://zillowwebapp-receptive-ens.mybluemix.net/airconditiontype>

The api returns all air-condition id’s with its respective description.



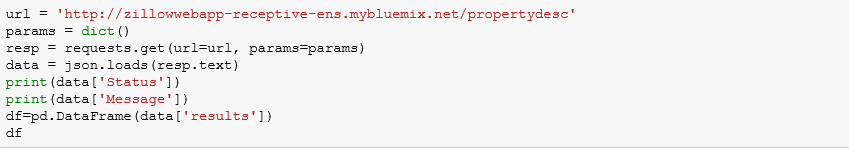
Query for searching all types or air-condition systems and result

**Example: Property type description** **search:**

**Search parameter:** <http://zillowwebapp-receptive-ens.mybluemix.net/propertydesc>

We can query for types of properties available and get the description and its associated ID

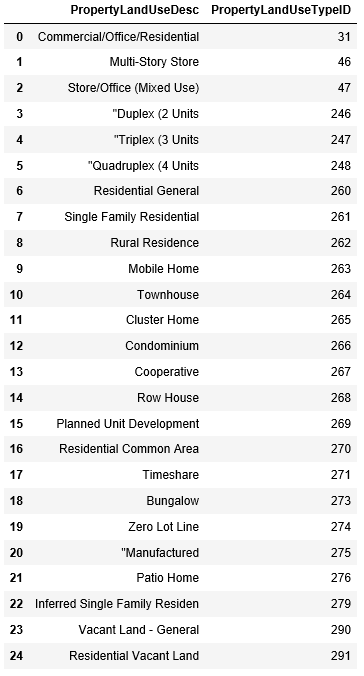
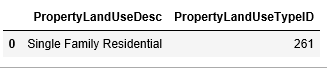
Following query returns all values as no specific id is provided.



For specific searches provide ID parameter as follows:



**The results:**

Without Id parameter Single result with id parameter

**Example: Heating System Description**

**Search parameter:** <http://zillowwebapp-receptive-ens.mybluemix.net/heatingsystemtype>

We can query for the type of heating systems installed by providing the type id or accessing all the types of heating systems available.

For all types:



For specific type:



Results:

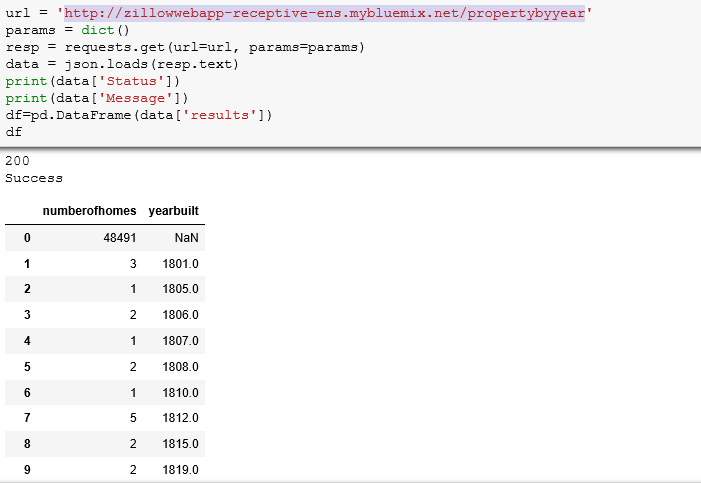
All available types For specific type ID

**Properties built by year**

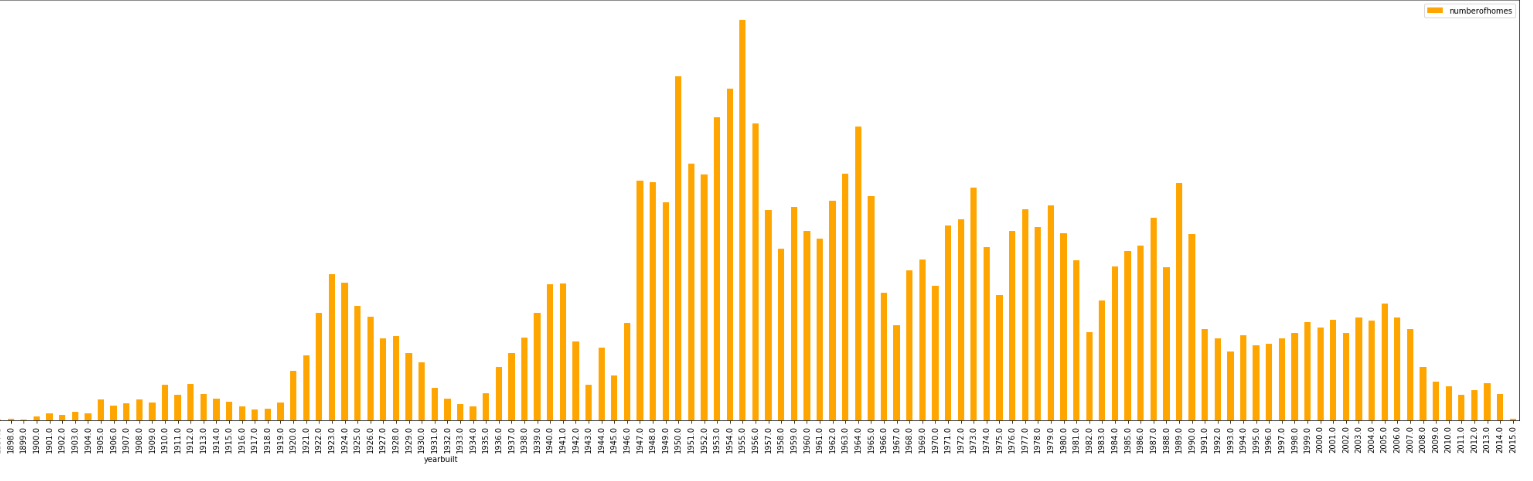
**Search parameters**:

<http://zillowwebapp-receptive-ens.mybluemix.net/propertybyyear>

Returns the count of all properties grouped by the year they were built in. To specifically search for properties built, the Search API can be used.



Provides a distribution of the properties by the year they were built in. The search api mentioned above provides search by year facility. The distribution shows most constructions are between the years 1950 to 1965, with maximum in 1955.



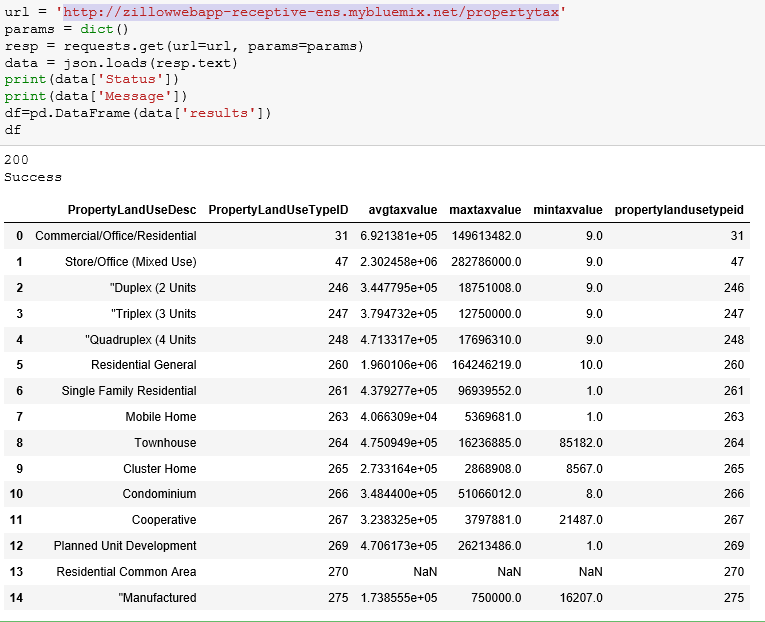
Property distribution by year built

**Tax value for type of properties**

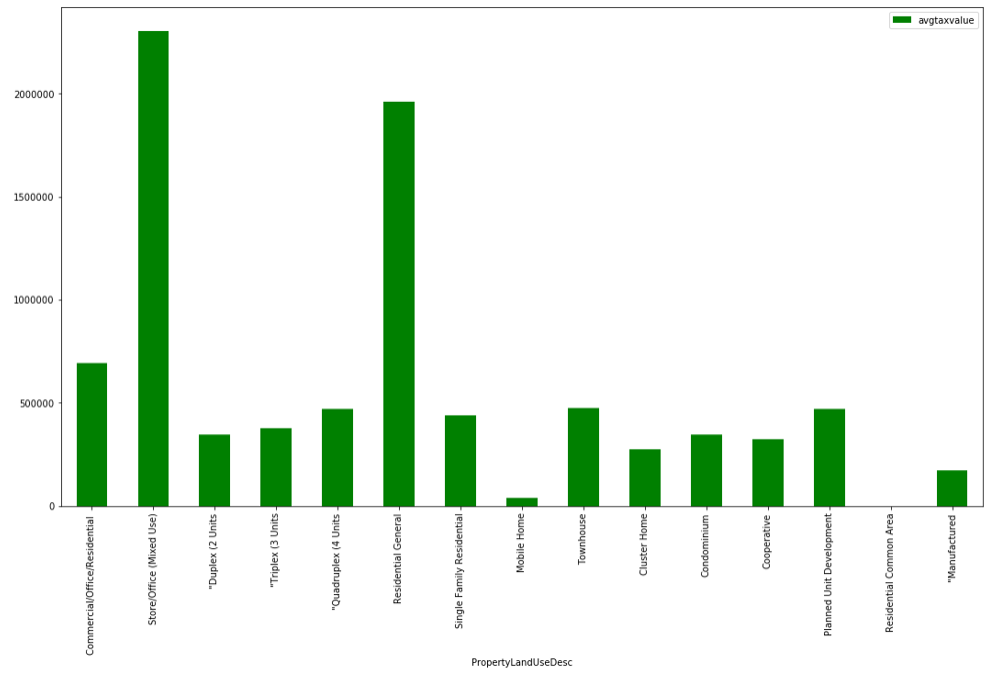
**Search parameters**:

<http://zillowwebapp-receptive-ens.mybluemix.net/propertytax>

Provides the tax value associated by the type of property available. It returns the minimum tax, maximum tax and average tax amount based on the property type.



Query with results for tax value for every property type



Tax amount by type of property