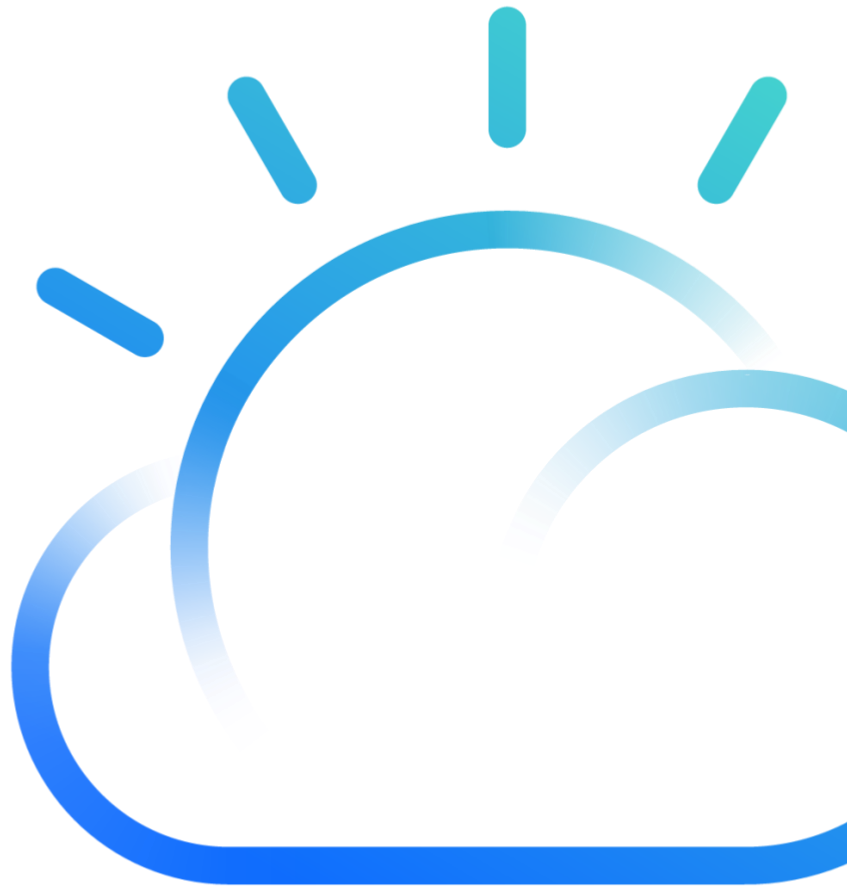


# Analyze San Francisco Reported Police Incidents

*Workshop Guide*



---

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 2019.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

## Contents

<b>SECTION 1.</b>	<b>PREFACE .....</b>	<b>4</b>
	OVERVIEW.....	4
	OBJECTIVES .....	6
	TOOLS.....	6
	FLOW.....	7
	PREREQUISITES .....	7
<b>SECTION 2.</b>	<b>WATCH THE VIDEO.....</b>	<b>8</b>
<b>SECTION 3.</b>	<b>STEPS.....</b>	<b>9</b>
	RUN THE NOTEBOOK .....	13
	ANALYZE THE RESULTS .....	14
	<b>QUESTION: WHERE IN MISSION DO MOST INCIDENTS HAPPEN?</b> .....	16
	<b>QUESTION: WHAT TIME OF DAY DO MOST INCIDENTS OCCUR?</b> .....	16
	WHAT HAVE WE LEARNED .....	17
<b>APPENDIX A: BUILDING THE PIXIEAPP DASHBOARD.....</b>		<b>18</b>
	WHAT YOU'LL NEED.....	18
	FAQ ABOUT THE CODE BELOW .....	18
	CREATE THE MAP OF INCIDENTS .....	20
	GENERATE THE PD_OPTIONS.....	20
	INITIALIZE THE PD_OPTIONS.....	21
	CREATE THE GEOJSON CUSTOM LAYERS .....	22
	CREATE THE CHECKBOXES FROM THE LAYERS.....	22
	SAVE AND SHARE.....	23
	<b>A. HOW TO SAVE YOUR WORK:</b> .....	23
	<b>B. HOW TO SHARE YOUR WORK:</b> .....	23


## Section 1. Preface

### Overview

Let's think back to the methodology of Data Science.


PROCESS DETAILS - BUSINESS UNDERSTANDING

**Every project starts with business understanding.**



Bob

**Business Sponsor**




Sally

**Data Journalist**

**The business sponsors**

who need the analytic solution play the most critical role in this stage by defining the problem, project objectives and solution requirements from a business perspective.

This first stage lays the **foundation for a successful resolution of the business problem.**



Maria

**Data Scientist**

**Analytic approach**

Once the business problem has been clearly stated, the **data scientist works to** define the analytic approach to solving the problem.

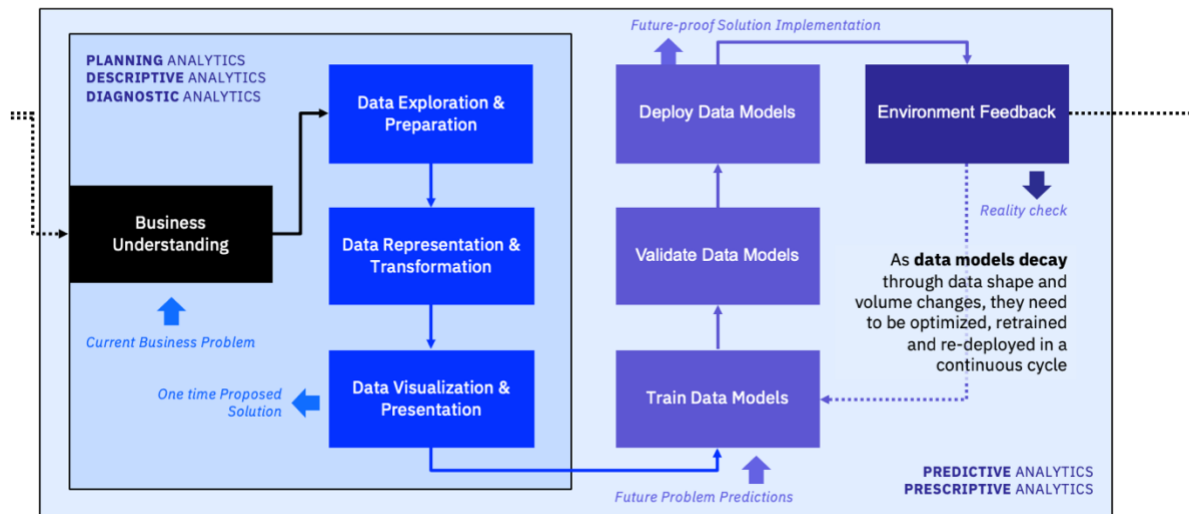
**Expressing the problem in the context of statistical and machine-learning techniques**, so the organization can identify the most suitable ones for the desired outcome.

The journey begins with the Business sponsor, who has noted a rise or at least a greater awareness amongst dwellers and visitors to the city of San Francisco pertaining to the following questions:

- Do incidents in one police district result in more arrests than other police districts?
- How does the number of incidents change over the course of the week?

Maria, the Data Scientist has commissioned Tom and Sally to bring their expertise forth in finding hidden insights from the data that is already available with the City of San Francisco.

Most of the Jupyter Notebook activities below is performed by the Data Engineer, Tom. Later, when Tom begins to visualizing the curated CSV file listing the incidents, Sally the Visualization expert, takes over towards the end in explaining the various findings as revealed by Pixiedust.



Sally uses Pixiedust, with Tom's help in manipulating the dependent and the independent variables, to further visualize another hypothesis. Often disproving a hypothesis is as much of a success story as is proving it.

In analyzing the geographical data, we can see a couple of clusters where incidents occur more frequently in The Mission - the southeastern corner looks particularly crowded. Some useful questions to ask at this point are:

- Do certain streets effect on the number of incidents?
- Are there more incidents in areas at the borders of police districts?
- Do police calming devices reduce the number of incidents?

We can test these hypotheses in two ways:

- 1) Download datasets for crime data and police calming in San Francisco and simply use the `display` API to visualize crime zones and areas with police calming devices separately.
- 2) Build a **Pixie App**, which encapsulates everything we have discussed thus far into an interactive way to explore multiple views of the data.

Only basic HTML and JavaScript are needed to write a Pixie App, so you don't have to learn any new languages or frameworks. In particular, a Pixie App will allow us to overlay mapping layers, and therefore give us a clearer view into the problem we are investigating.

## Objectives

Visualize and analyze San Francisco crime incidents using a Jupyter Notebook, PixieDust, and PixieApps

In this workshop we will use PixieDust running on IBM Watson Studio to analyze traffic calming and crime data from the City of San Francisco. Watson Studio is an interactive, collaborative, cloud-based environment where data scientists, developers, and others interested in data science can use tools (e.g., RStudio, Jupyter Notebooks, Spark, etc.) to collaborate, share, and gather insight from their data.

When the reader has completed this Code Pattern, they will understand how to:

- Use Jupyter Notebooks to load, visualize, and analyze data
- Run Notebooks in IBM Watson Studio
- Leverage PixieDust as a python notebook helper
- Build a dashboard using PixieApps
- Fetch data from City of San Francisco Open Data
- Create an interactive map with Mapbox GL

The intended audience for this Code Pattern is application developers and other stakeholders who wish to utilize the power of Data Science quickly and effectively.

## Tools



Jupyter Notebook



Python (PixieApps)

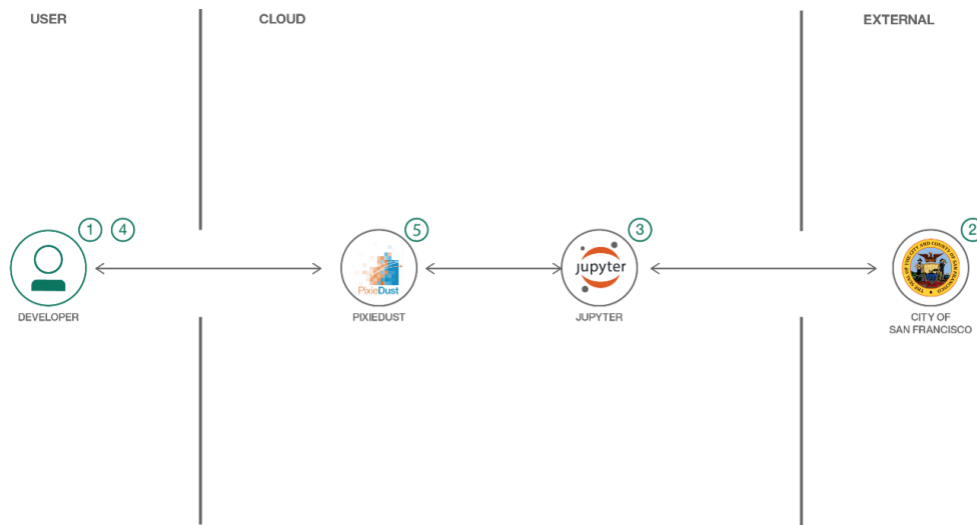


PixieDust



JavaScript (Mapbox GL)

## Flow



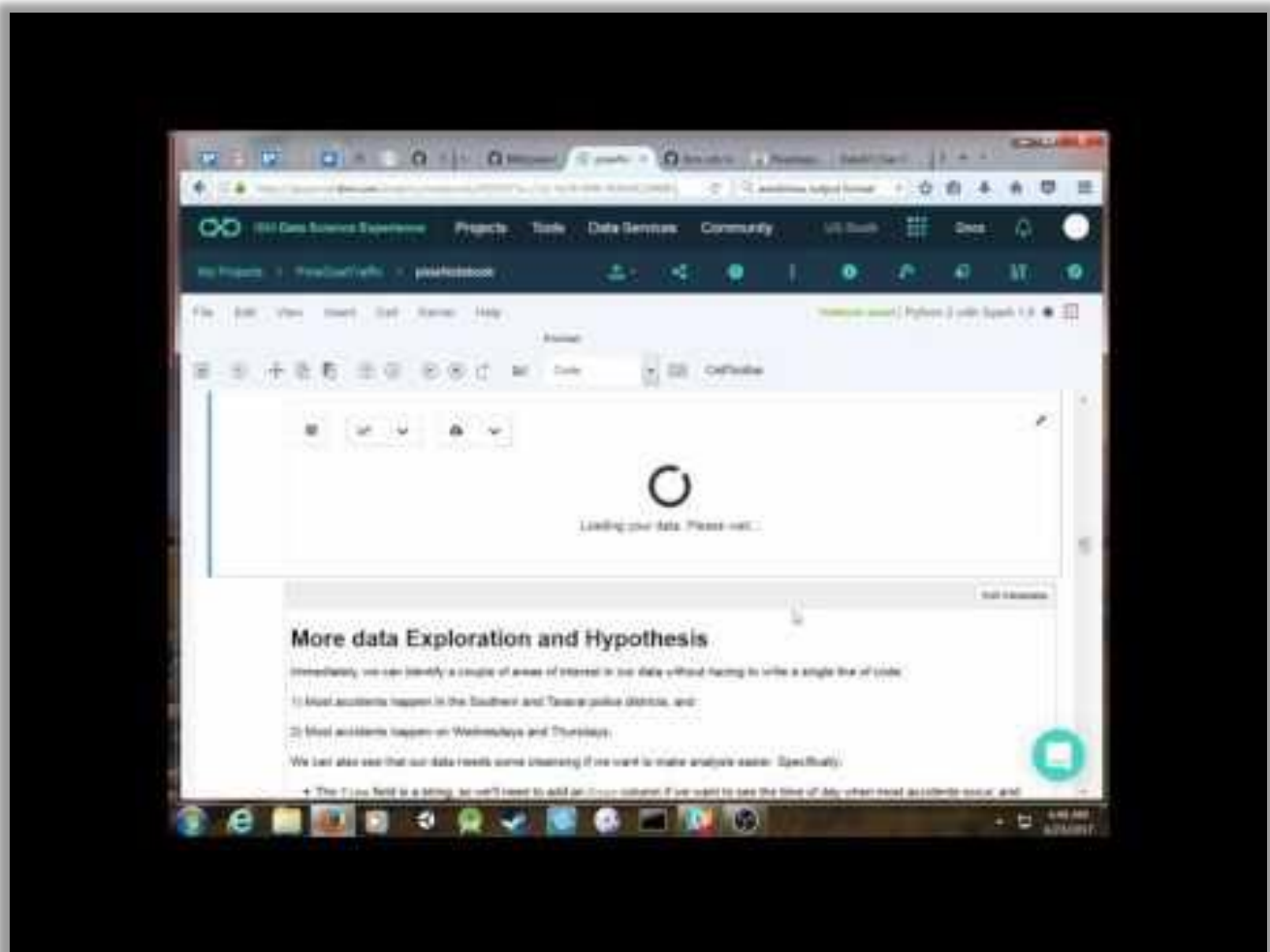
1. Load the provided notebook into the Watson Studio platform.
2. [DataSF Open Data](#) crime info is loaded into the Jupyter Notebook.
3. The notebook analyzes the crime info.
4. You can interactively change charts and graphs.
5. A PixieApp dashboard is created and can be interacted with.

## Prerequisites

- You will need a token from [Mapbox Token](#). For the purposes of this exercises, use this token:
- pk.eyJ1IjoiYXBpc2NoZG8iLCJhIjoiY2o2cXkxMjUxMDMyaTJ3bGEyYjFsZ3Y4cSJ9.mL2PT0XH2vrNiDozb7gO0w

## Section 2. Watch the Video

The video below will give you a sound idea of the steps you need to take to complete the data transformation and visualization.

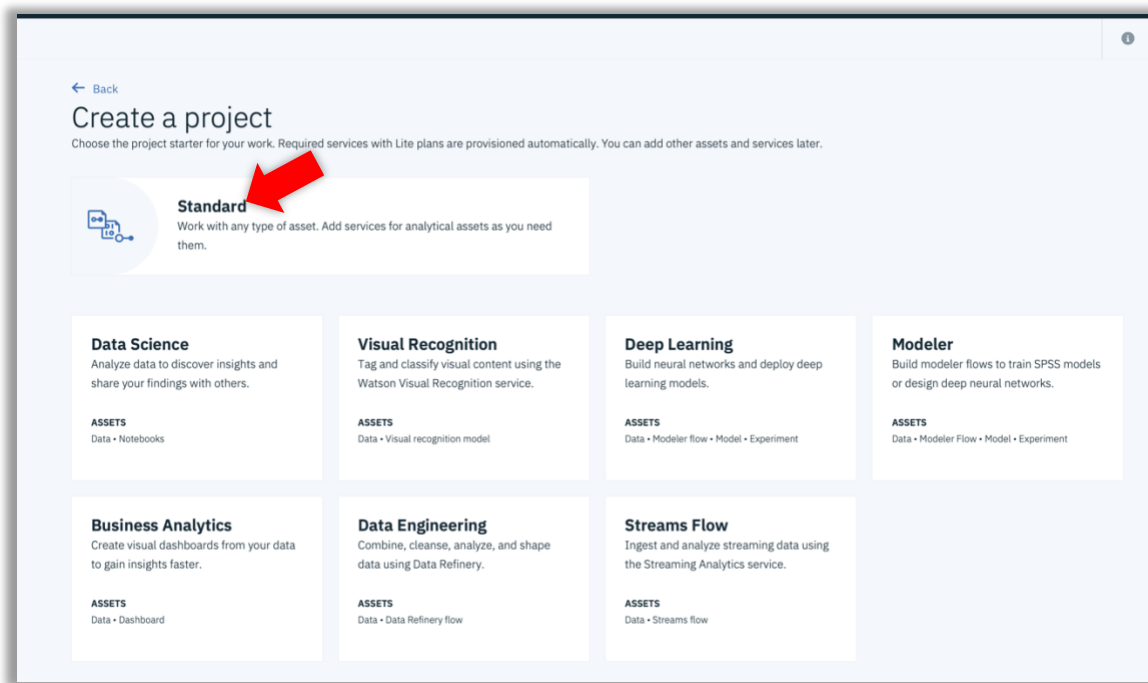




## Section 3. Steps

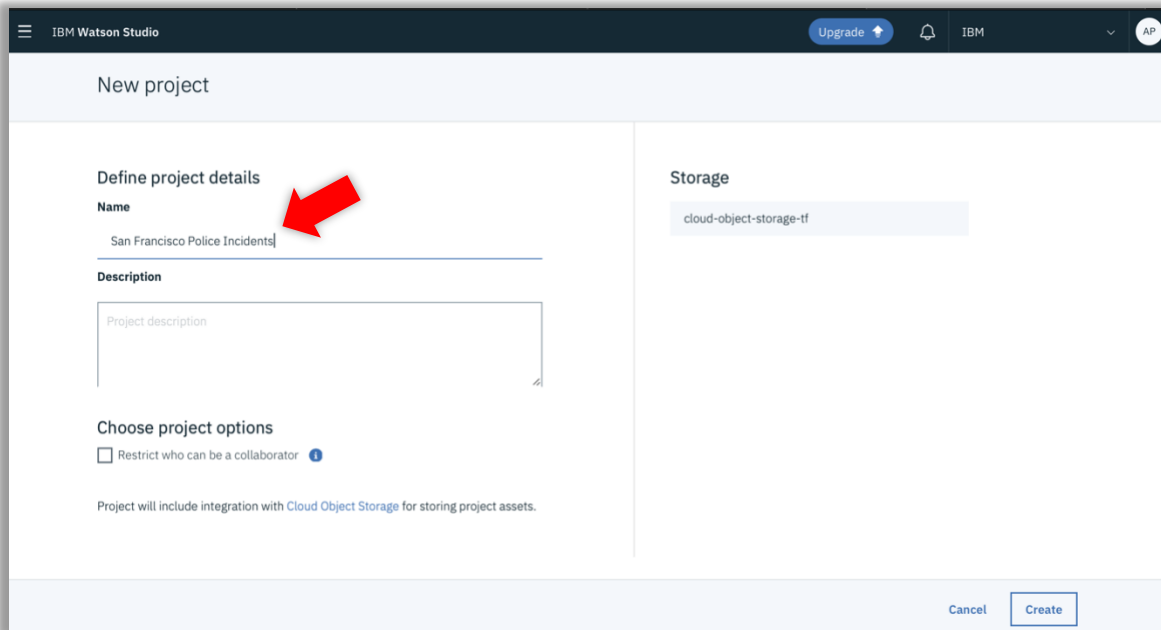
By now you have already registered with IBM Cloud. Let's begin our journey:

1. Login into IBM Cloud: <https://cloud.ibm.com>
2. Click the **Catalog** tab.
3. Search for the **Watson Studio** service and click that tile.
4. Click **Create**.
5. Click the **Get Started** button.
6. Click **New**.
7. Select the **Standard** tile.



8. Click **Create Project**.

9. Specify a name; for example: San Francisco Police Incidents.



IBM Watson Studio

Upgrade

IBM

AP

### New project

#### Define project details

**Name**

San Francisco Police Incidents

**Description**

Project description

#### Choose project options

☐ Restrict who can be a collaborator

Project will include integration with [Cloud Object Storage](#) for storing project assets.

**Storage**

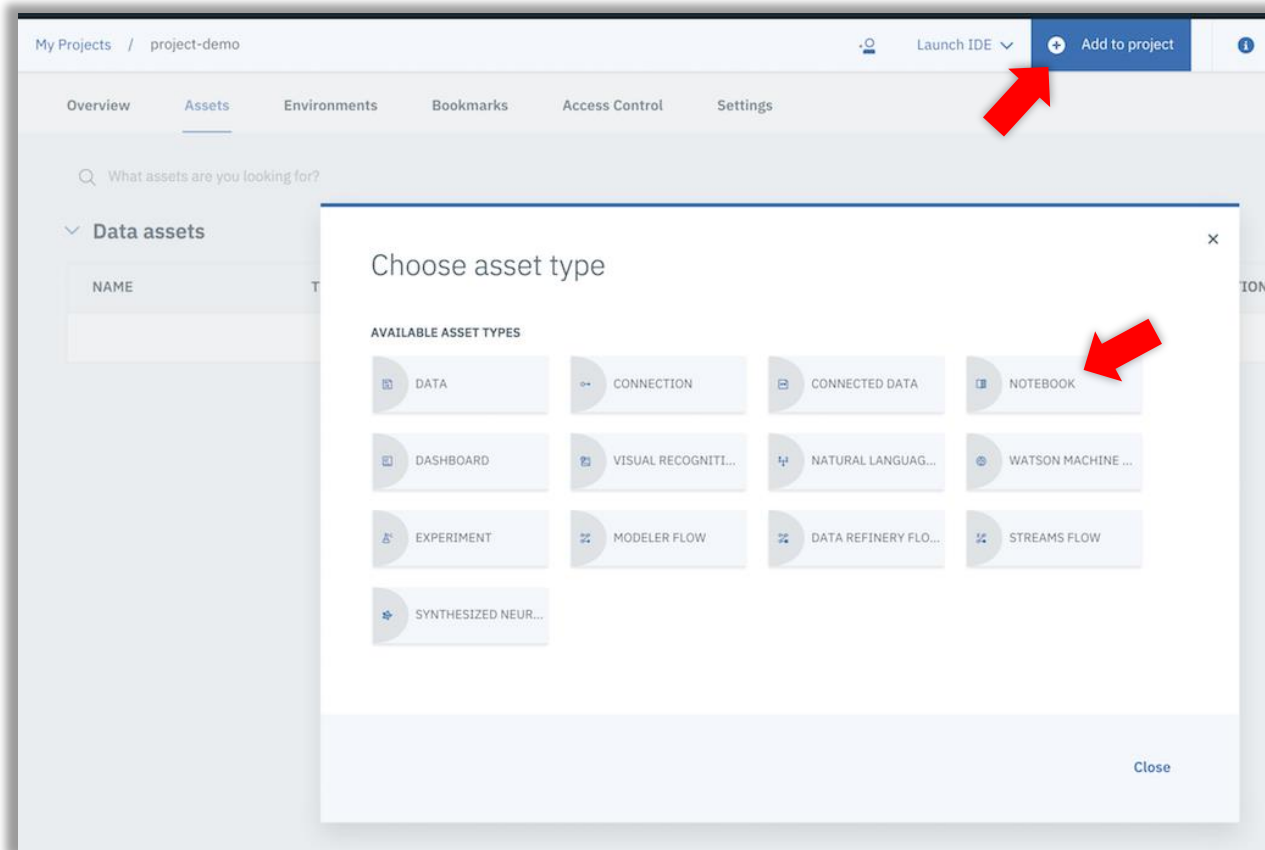
cloud-object-storage-tf

Cancel Create

10. Click **Create**

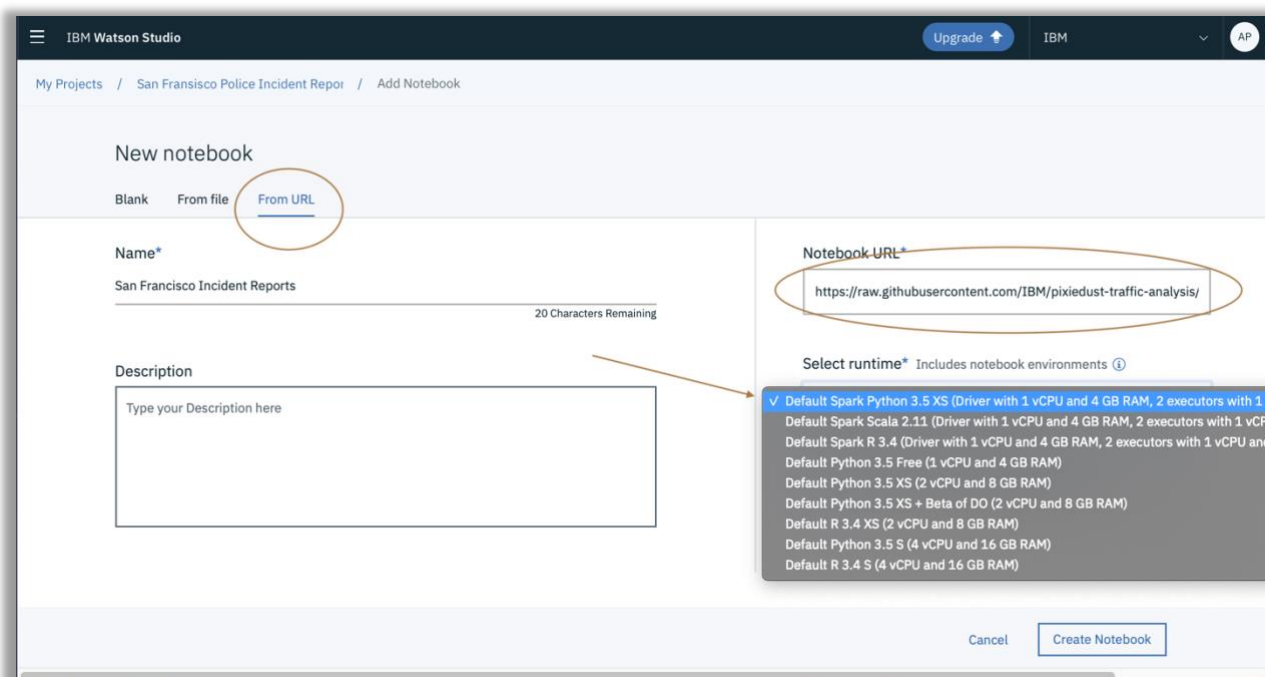
11. Click **Add to project**.

12. Click the **NOTEBOOK** tile.



13. Specify a name.

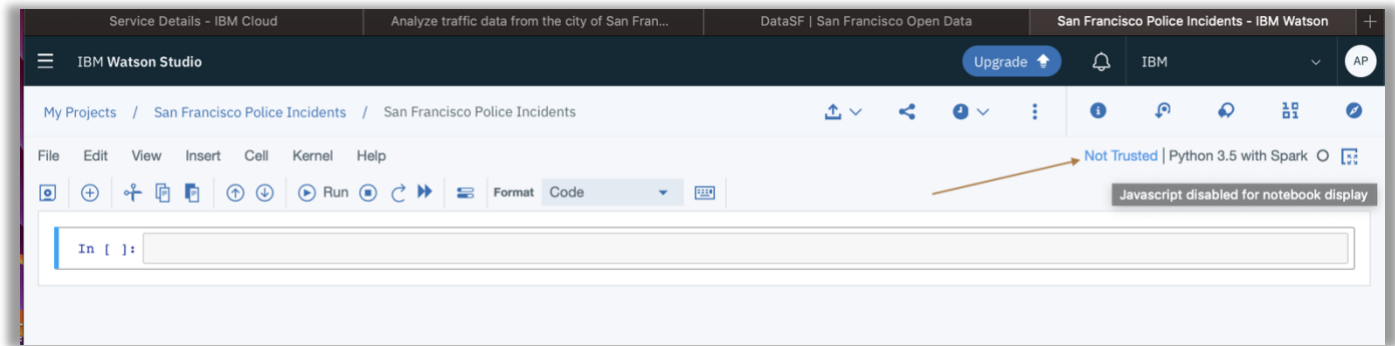
14. From the **Select runtime** drop down list, select the first option.



15. Copy and paste the following URL:

`https://github.com/apischdo/Bluemix-workshop-assets/blob/master/Analyze%20traffic%20data%20from%20the%20city%20of%20San%20Francisc.ipynb`

16. Change the Kernel to **Trusted** (top right corner of the notebook).



## Run the notebook

When a notebook is executed, what is actually happening is that each code cell in the notebook is executed, in order, from top to bottom.

Each code cell is selectable and is preceded by a tag in the left margin. The tag format is In [x]:. Depending on the state of the notebook, the x can be:

- A blank, this indicates that the cell has never been executed.
- A number, this number represents the relative order this code step was executed.
- A \*, this indicates that the cell is currently executing.

There are several ways to execute the code cells in your notebook:

- One cell at a time.
  - Select the cell, and then press the Play button in the toolbar.
- Batch mode, in sequential order.
  - From the Cell menu bar, there are several options available. For example, you can Run All cells in your notebook, or you can Run All Below, that will start executing from the first cell under the currently selected cell, and then continue executing all cells that follow.
- At a scheduled time.
  - Press the Schedule button located in the top right section of your notebook panel. Here you can schedule your notebook to be executed once at some future time, or repeatedly at your specified interval.

## Analyze the Results

After running each cell of the notebook, the results will display. When we use PixieDust display() to create an interactive dataset, we are able to change the visualization using tables, graphs, and charts.

After running cell #3 display(incidents), you can see by clicking the Options button that you are able to manipulate the keys and values for the fields used in the chart:

Pixiedust: Bar Chart Options

Chart Title:

Fields: [Show only numeric columns](#) **Keys:** [?](#)

Search Filter Fields

Field	Type
Address	string
Category	string
Date	string
DayOfWeek	string
Descript	string
Hour	numeric
IncidntNum	numeric
Location	string

**Keys:**

- DayOfWeek

**Values:**

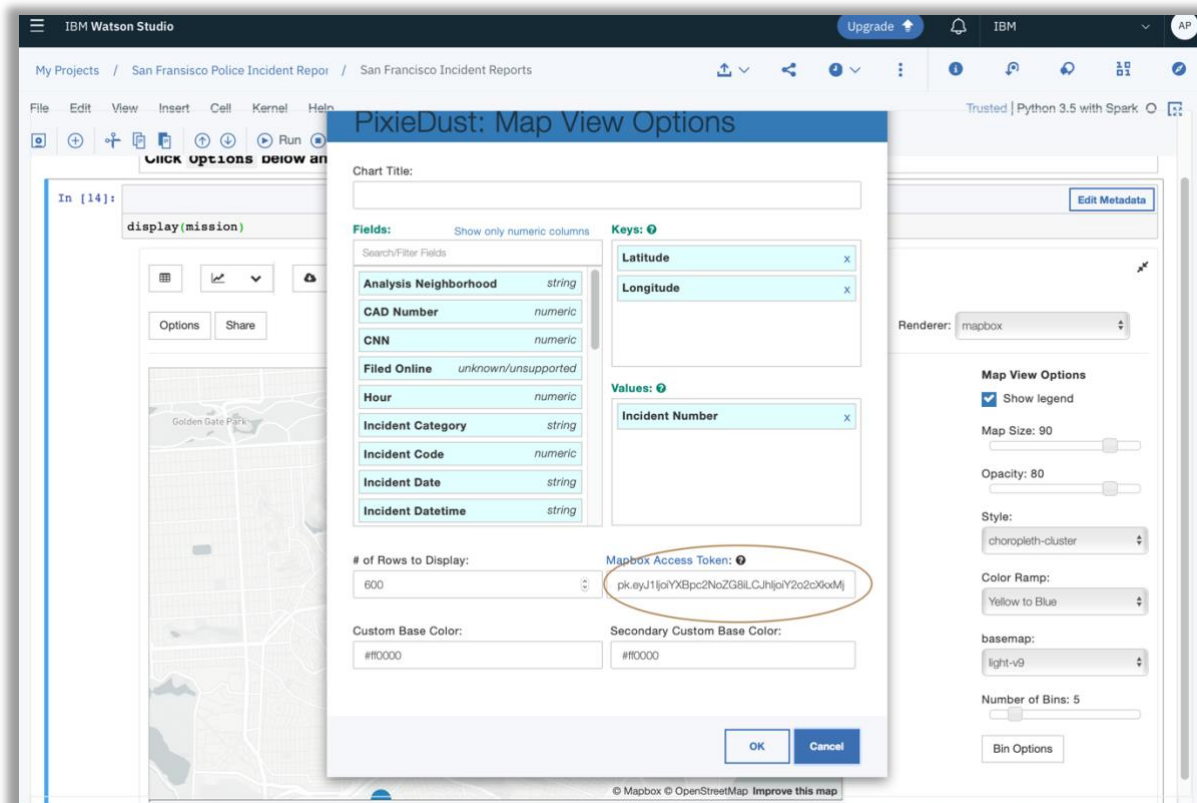
- IncidntNum

Aggregation: COUNT # of Rows to Display: 1000

OK CANCEL

Following the instructions, you use **DaysOfWeek** and **IncidntNum**, but you can change the keys and value to see how the chart will look with different inputs.

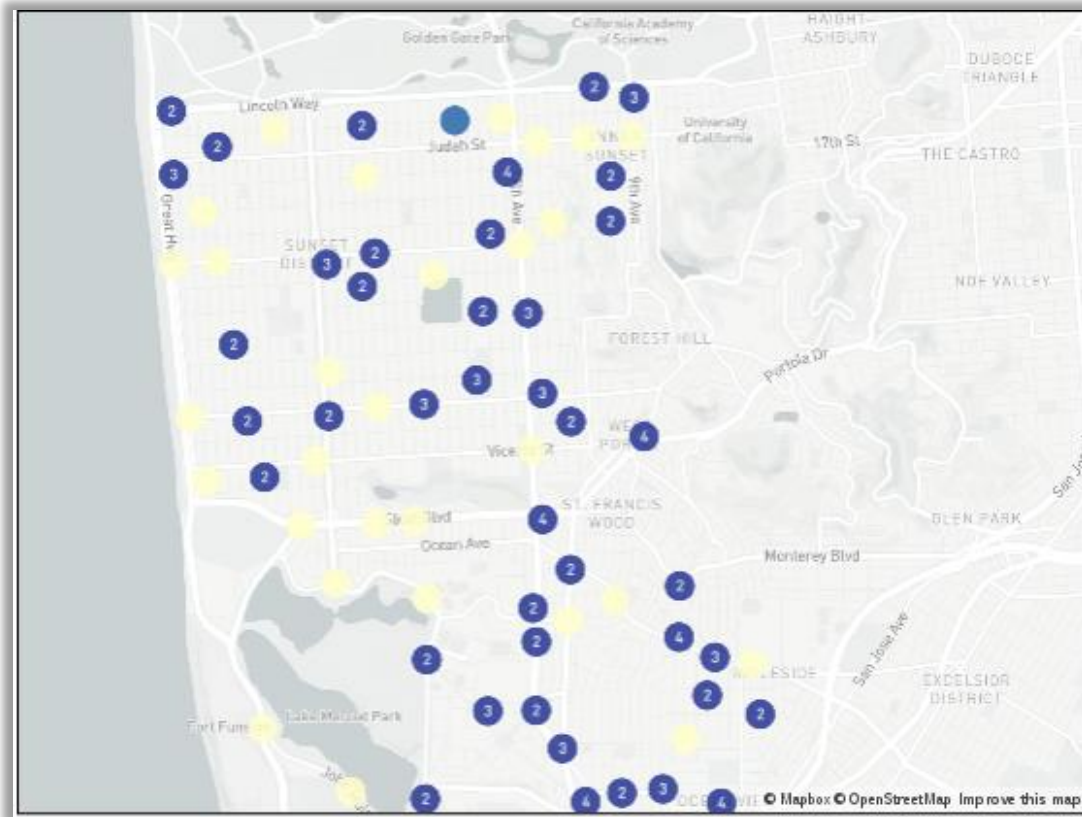
Let's use Spark SQL to isolate data to the Taraval district plus the Mapbox token is used in the Options section of the Pixiedust UI:



The following code, already in your Notebook cell, generates an interactive map of the Taraval district:

```
accidents.registerTempTable("accidents")
taraval = sqlContext.sql("SELECT * FROM accidents WHERE PdDistrict='TARAVAL'")
```

This analysis reveals the following discoveries:



- 1) Incidents in the Mission and Southern police districts are much more likely to result in arrest than all other districts, and
- 2) The number of incidents declines slightly during the middle of the week, through Thursday, but increases again through Saturday.

**Question: Where in Mission do most incidents happen?**

(Map - Options: Keys = [X,Y], Values = IncidentNum, Aggregation = Count, Renderer: mapbox, kind: chloropleth-cluster)

**Question: What time of day do most incidents occur?**

(Line Chart - Options: Keys = Hour, Values = IncidentNum, Aggregation = Count)



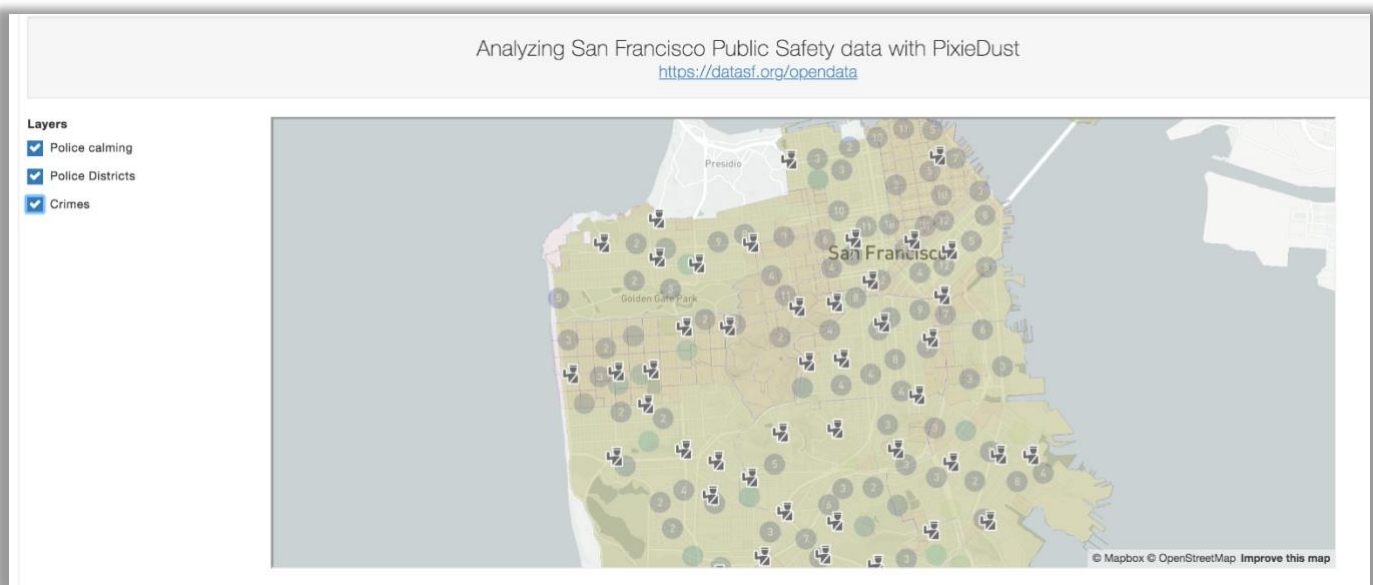
## What have we learned

Most of the results from looking at the incident times are unsurprising:

- Number of incidents drop sharply very early morning (people probably sleeping),
- Steady increase in number of incidents until noon,
- Fairly high numbers from 3:00 PM until 8:00 PM,
- Surprisingly, incidents decline after 8:00PM.

The interesting thing here is the fact that the peaks are at noon and from 3:00PM until 8:00 PM, as one might expect the later evening times to be more problematic.

With PixieApps, you can create a dashboard with map layers that is used to visualize various datasets (i.e. Traffic Calming, Police Districts, and Crimes):



Your challenge would be to import another data set and manipulate the code (mostly column headings) to depict new discoveries on your new dataset

## Appendix A: Building the PixieApp Dashboard

### What you'll need

- Mapbox token: A Mapbox token is included in the notebook, for now. To get your own visit [Mapbox](#)
- Mapbox layers Documentation: [circle](#), [fill](#), [symbols](#)
- Mapbox Maki Icons: <https://www.mapbox.com/maki-icons>
- Browse the data on [San Francisco Open Data](#) to get the GeoJSON url
- Some understanding of [Jinja2 template](#)

### FAQ about the code below

- How do I get the pixiedust options in `self.mapJSONOptions`?
  1. Call `display()` on a new cell
  2. Graphically select the options for your chart
  3. Select "View"/"Cell Toolbar"/"Edit Metadata" menu
  4. Click on the "Edit Metadata" button and copy the pixiedust metadata

- What's the `self.setLayers` call for?

This is a method from the MapboxBase class used to specify the custom layer definitions array. The fields are:

1. name: Layer name
  2. url: geojson url to download the data from
  3. type: (optional) style type e.g Symbol. If not defined, then default value will be inferred from geojson geometry
  4. paint: (optional) paint style, see appropriate documentation e.g. [circle](#)
  5. layout: (optional) layout style, see appropriate documentation e.g. [fill](#)
- How do I find new layer data to add?

Just go to [San Francisco Open Data](#), browse the data and click on the export button. You should see a geojson link among others (warning: not all datasets have a geojson link, if you don't find it, then move on to another one)

- What does the `mainScreen` method do?

This is a PixieApp View associated with the default route. See [PixieApp documentation](#) for more information.

- What's the `{{...}}` notation in the mainScreen markup for?

This is a Jinja2 template notation to call server side Python code. For more info see [Jinja2 template](#).

Let's take a look at the code below and observe what takes place:

```

from pixiedust.display.app import *
from pixiedust.apps.mapboxBase import MapboxBase

@PixieApp
class SFDashboard(MapboxBase):
    def setup(self):
        self.mapJSONOptions = {
            "mapboxtoken": "pk.eyJ1IjoiYXBpc2NoZG8iLCJhIjoiY2o2cXkxMjUxMDMyaTJ3bGEyaFJsZ3Y4cSJ9.mL2PT0XH2vrNiDozb7g00w",
            "chartsize": "90",
            "aggregation": "SUM",
            "rowCount": "500",
            "handlerId": "mapView",
            "rendererId": "mapbox",
            "valueFields": "IncidntNum",
            "keyFields": "X,Y",
            "basemap": "light-v9"
        }
        self.setLayers([
            {
                "name": "Police calming",
                "url": "https://data.sfgov.org/api/geospatial/ddye-rism?method=export&format=GeoJSON",
                "type": "symbol",
                "layout": {
                    "icon-image": "police-15",
                    "icon-size": 1.5
                }
            },
            {
                "name": "Police Districts",
                "url": "https://data.sfgov.org/api/geospatial/wkhw-cjsf?method=export&format=GeoJSON"
            },
            {
                "name": "Crimes",
                "url": "https://data.sfgov.org/api/geospatial/ms8q-rzqw?method=export&format=GeoJSON",
                "paint": {
                    "fill-color": "rgba(255,182,193,0.5)"
                }
            }
        ])

    def formatOptions(self, options):
        return ','.join(["{}={}".format(key, value) for (key, value) in iteritems(options)])

    @route()
    def mainScreen(self):
        return """
<div class="well">
  <center><span style="font-size:x-large">Analyzing San Francisco Public Safety data with PixieDust</span></center>
  <center><span style="font-size:large"><a href="https://datasf.org/opendata" target="new">https://datasf.org/opendata</a></span></center>
</div>
<div class="row">
  <div class="form-group col-sm-2" style="padding-right:10px;">
    <div><strong>Layers</strong></div>
    {% for layer in this.layers %}
    <div class="rendererOpt checkbox checkbox-primary">
      <input type="checkbox" pd_refresh="map{{prefix}}" pd_script="self.toggleLayer({{loop.index0}})"
    >
      <label>{{layer["name"]}}</label>
    </div>
    {%endfor%}
  </div>
  <div class="form-group col-sm-10">
    <div id="map{{prefix}}" pd_entity pd_options="{{this.formatOptions(this.mapJSONOptions)}}"/>
  </div>
</div>
"""

SFDashboard().run(incidents,runInDialog="false")

```

## Create the Map of Incidents

```
<div id="map{{prefix}}" pd_entity pd_options="{{this.formatOptions(this.mapJSONOptions)}}"/>
```

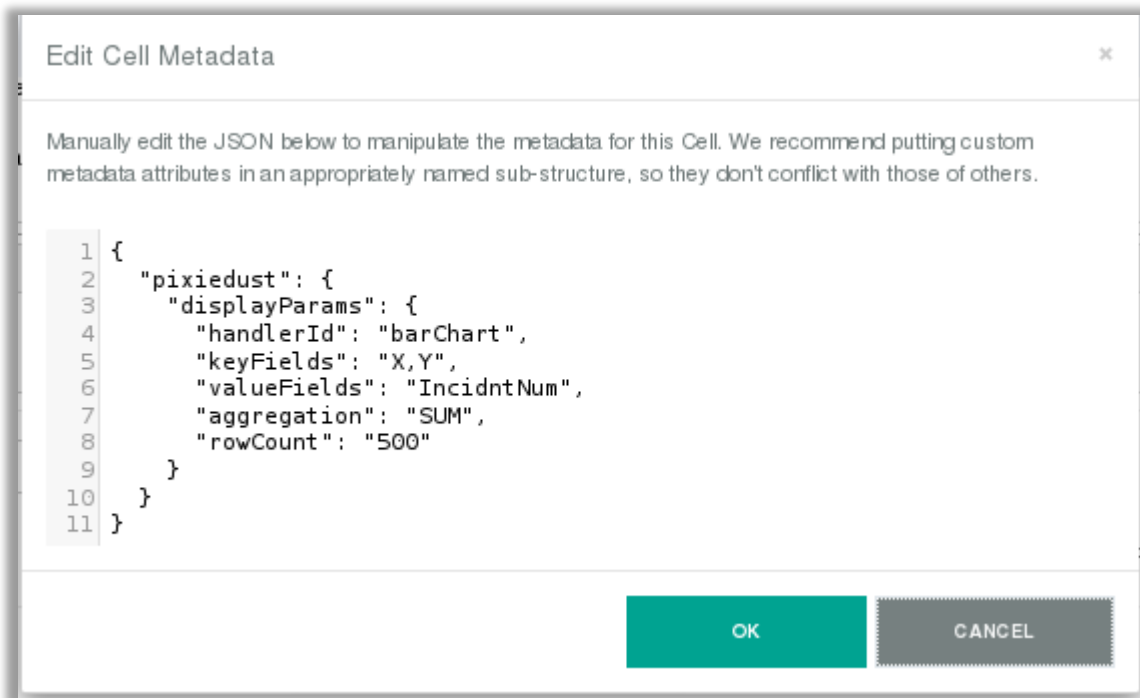
pd\_entity: Tell PixieDust which dataset to work on.

pd\_options: Contains the PixieDust options for the map.

## Generate the pd\_options

The best way to generate the pd\_options for a PixieDust visualization is to:

1. Call display() on a new cell
2. Graphically select the options for your chart
3. Select View/Cell Toobar/Edit metadata menu
4. Click on the Edit Metadata button and copy the PixieDust metadata



To conform to the pd\_options notation, we need to transform the PixieDust JSON metadata into an attribute string with the following format: "key1=value1;key2=value2;..."

To make it easier, we use a simple Python transform function:

```
def formatOptions(self, options):
    return ';'.join(["{}={}".format(k,v) for (k, v) in iteritems(options)])
```

The formatOptions is then invoked using JinJa2 notation from within the html:

```
pd_options = "{{this.formatOptions(this.mapJSONOptions)}}"
```

## Initialize the pd\_options

Note: setup is a special method that will be called automatically when the PixieApp is initialized.

A Mapbox token is included in the notebook, for now. To get your own visit [Mapbox](#)

```
def setup(self):
    self.mapJSONOptions = {
        "mapboxtoken":
"pk.eyJ1IjoicmFqcnpbmndoIiwiaYSI6ImNqM2s4ZDg4djAwcGYyd3BwaGxwaDV3bWoiLCJ0d5Rklkdu5MeGAnXu1GMNYw",
        "chartsize": "90",
        "aggregation": "SUM",
        "rowCount": "500",
        "handlerId": "mapView",
        "rendererId": "mapbox",
        "valueFields": "IncidentNum",
        "keyFields": "X,Y",
        "basemap": "light-v9"
    }
```

## Create the GeoJSON Custom Layers

```

from pixiedust.display.app import *
from pixiedust.apps.mapboxBase import MapboxBase

@PixieApp
class SFDashboard(MapboxBase):
    def setup(self):
        self.mapJSONOptions = {
            "mapboxtoken":
"pk.eyJ1IjoiYXBpc2NoZG8iLCJhIjoIY2o2cXkxMjUxMDMyaTJ3bGEyaJFsZ3Y4cSJ9.mL2PT0XH2vrNiDozb7g00w",
            "chartsSize": "90",
            "aggregation": "SUM",
            "rowCount": "500",
            "handlerId": "mapView",
            "rendererId": "mapbox",
            "valueFields": "IncidntNum",
            "keyFields": "X,Y",
            "basemap": "light-v9"
        }
        self.setLayers([
            {
                "name": "Police calming",
                "url": "https://data.sfgov.org/api/geospatial/ddye-rism?method=export&format=GeoJSON",
                "type": "symbol",
                "layout": {
                    "icon-image": "police-15",
                    "icon-size": 1.5
                }
            },
            {
                "name": "Police Districts",
                "url": "https://data.sfgov.org/api/geospatial/wkhw-cjsf?method=export&format=GeoJSON"
            },
            {
                "name": "Crimes",
                "url": "https://data.sfgov.org/api/geospatial/ms8q-rzqw?method=export&format=GeoJSON",
                "paint": {
                    "fill-color": "rgba(255,182,193,0.5)"
                }
            }
        ])
...<snip>...

```

## Create the Checkboxes from the Layers

```

...<snip>...

{% for layer in this.layers %}
<div class="rendererOpt checkbox checkbox-primary">
  <input type="checkbox" pd_refresh="map{{prefix}}"
pd_script="self.toggleLayer({{loop.index0}})">
  <label>{{layer["name"]}}</label>
</div>
{%endfor%}

...<snip>...

```

The user can now select layers and the map will dynamically add or remove them.

## Save and Share

### A. How to save your work:

Under the File menu, there are several ways to save your notebook:

- Save will simply save the current state of your notebook, without any version information.
- Save Version will save your current state of your notebook with a version tag that contains a date and time stamp. Up to 10 versions of your notebook can be saved, each one retrievable by selecting the Revert To Version menu item.

### B. How to share your work:

You can share your notebook by selecting the “Share” button located in the top right section of your notebook panel. The end result of this action will be a URL link that will display a “read-only” version of your notebook. You have several options to specify exactly what you want shared from your notebook:

- Only text and output: will remove all code cells from the notebook view.
- All content excluding sensitive code cells: will remove any code cells that contain a *sensitive* tag. For example, # @hidden\_cell is used to protect your dashDB credentials from being shared.
- All content, including code: displays the notebook as is.
- A variety of download as options are also available in the menu.



---

© Copyright IBM Corporation 2019.

The information contained in these materials is provided for informational purposes only and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).



Please Recycle

---