

# Auto Insurance Fraud Analyzed in Jupyter Notebooks

## Overview

Combatting fraud and performing investigative action demands an end-to-end data science solution. It empowers an organization to scale analysis with ready access to public clouds, private clouds and on-premises. The platform also speeds modeling, training and deployment time and simplifies collaboration with data scientists, risk analysts, investigators, and other subject matter experts while adhering to strong governance and security posture. Further, in order to respond to new types of fraud, waste and abuse while minimizing false negatives and accelerating response, the platform needs to continuously accommodate real-time data, monitor and detect fraudulent activities and adapt as the patterns change and spot anomalies.

The global Fraud Detection and Prevention (FDP) market size is expected to grow from USD 20 billion to 63.5 billion by 2023, according to various analyst reports (i.e. ["Fraud Detection and Prevention Market by solution"](#)). Predictive analytics segment is projected to be the largest contributor to the FDP market during the forecast period.

Predictive analytics solutions help enterprises identify the possibilities of fraud incidents by analyzing the current data. The solutions are used to identify potential threats, payment frauds, frauds in insurance processes, and credit/debit card frauds. Organizations are trying to impart these solutions for predicting fraud or suspicious activity and their pattern to help drastically reduce losses due to frauds.

A global fraud report from Experian says that 72% of businesses cite fraud as a growing concern.


Digital transformation has created data issues that make it difficult to detect fraud. Silos of data residing in your lines of business, departments, and geos and varying analytical techniques across channels and transaction systems have opened you up to increased risk exposures and attacks from fraudsters.

It's time to track behavior and exposure so you can prevent fraud before it happens. In this lab you will perform all of the steps that you undertook with a UI-driven Watson Studio activities, except that you will perform these tasks in Jupyter Notebooks. The code snippets are provided, so need for coding acumen, yet you are welcome to experiment with the code and realize other insights and feature engineering task that may reveal other behavior indicative of fraudulent auto insurance claims.

The app below represents relevant data to the data scientist. The focal point of this lab is about fraud prediction by assigning a probability value to certain behavior that may predict fraudulent behavior.




Watson Studio provides tools to build the data assets used by this app: refine data and build, train, deploy the fraud probability model.



### Driver Profile

Ivan Stiegelmeyer



738-534-0226

istiegelmeyer@gmxx.com

**Drivers License:** IL  
**Expires:** 2018-05-05  
**Prior claims:** 5

[Details...](#)

### Policy Details

2008 Nissan Sentra

**Member Since:** 2016-04-05  
**Expiration:** 2017-04-05  
**Initial Odometer:** 179568.0

☐ Low Mileage Use

[Details...](#)

### Incident Summary

Cause: Driver error


**Loss Event Time:** 2017-03-28 00:00  
**Claim Filed:** 2017-04-01 00:00  
**Claim Amount:** \$2196.50

[Details...](#)

**Odometer Reading:** 192322.9  
☐ Police Report Filed

### AI Fraud Detection

Fraud Likely




**Probability:** 98%  
**Contributing Factors:**

- High number of prior claims: 37%
- Near policy expiration: 28%
- No police report: 12%

[Details...](#)

### Weather Data


Conditions at time of incident:



**Rain**

Temperature: 40 degrees  
Hourly precipitation: 1 inches

### Map Data



**Estimated Time to Complete:** 1 Hour

## Objectives

The following constitute hypothesis formulated by subject matter experts (SMEs) in the field of auto insurance:

- Loss event claimed within 15 days of policy expiration
- Expired drivers' license
- Expensive vehicle damages
- Frequent changes of residence
- High mileage at loss event for a policyholder with a low mileage discount
- High number of previous claims
- No police report

You will build models that:

- Find the data that shows the fraud indicators
- Prepare a training data set with the fraud indicators
- Train a fraud prediction model
- Deploy the model for use in the fraud triaging app

The learning goals of this notebook are:

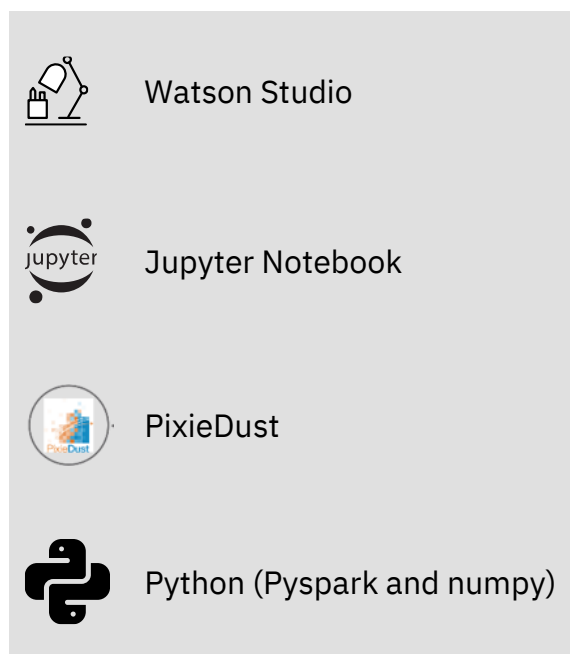
- Load the auto insurance CSV file into the Object Storage Service linked to your Watson Studio project or import the Jupyter Notebook into your project. In this lab you will build the notebook from scratch.
- Create a machine learning model
- Train and evaluate a model
- Persist a model in a Watson Machine Learning repository

There are five Milestones you must complete:

- Create the Notebook
- About Running Jupyter notebooks
- Importing libraries and Understanding the Raw data
- Understanding the Data
- Feature Engineering



## Tools

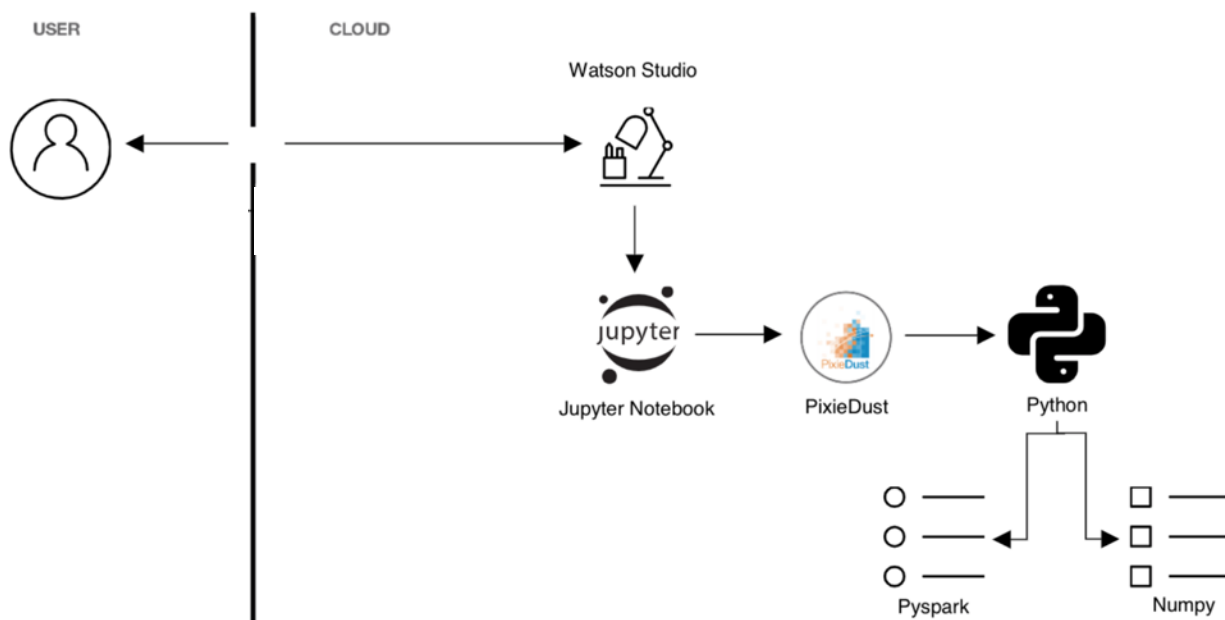


**Jupyter Notebook** is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

**PixieDust** is an open source Python helper library that works as an add-on to Jupyter notebooks to improve the user experience of working with data. It also fills a gap for users who have no access to configuration files when a notebook is hosted on the cloud.

**Python** is an interpreted, high-level, general-purpose programming language. Pyspark and Numpy are Python libraries.

## Flow



1. The User will create a Watson Studio Service.
2. From Watson Studio connect to Jupyter Notebook.
3. Utilize PixieDust, an open-source visualization program.

4. Manipulate Python code using Python Libraries, such as Pyspark, sklearn and Numpy.

---

## Milestone 1: Create the Notebook

### Milestone Overview

This lab requires you to complete five Milestones:

1. **Create the Notebook**
2. About Running Jupyter notebooks
3. Importing libraries and Understanding the Raw data
4. Understanding the Data
5. Feature Engineering

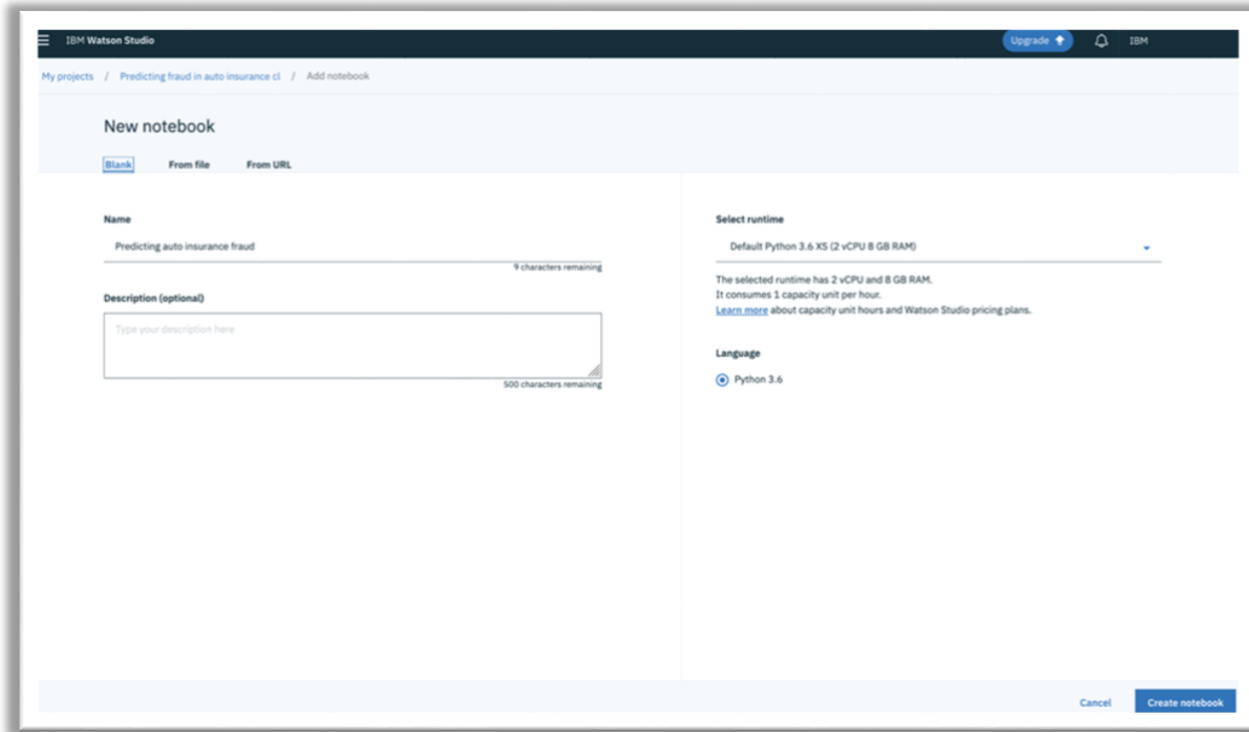
By now you have already registered with IBM Cloud and created your Watson Studio service. Let's begin our journey.

### Steps

1. Access the Watson Studio service from the IBM Cloud Dashboard
2. Click **Create a project** or, you can use the same project space per previous exercises.
3. Select the **Create an empty project** tile.
4. Specify a name; for example: **Predicting fraud in auto insurance claims**.
5. Click **Create**.

If this is your first-time visiting Watson Studio, you may need to add a **Cloud Object Storage (COS)** if the cloud-object-storage does not appear and the Create button is greyed out. Click the link under Choose project options and after you add the COS, click **Refresh** so you can view the COS instance. This is a one-time event when you first provision the Watson Studio service.

1. Click **Add to project** and select **Notebook**.
2. Select **Blank**
3. Give it a meaningful name. For example: Predicting auto insurance fraud
4. Click **Create notebook**.



5. Ensure that the kernel depicts Trusted by clicking Not Trusted and then Trust. All the notebook the few seconds it needs to save that setting.





---

## Milestone 2: About Running Jupyter notebooks

### Milestone Overview

This lab requires you to complete five Milestones:

1. Create the Notebook
2. **About Running Jupyter notebooks**
3. Importing libraries and Understanding the Raw data
4. Understanding the Data
5. Feature Engineering

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.

After running each cell of the notebook, the results will display.



## Milestone 3: Importing libraries and Understanding the Raw data

### Milestone Overview

This lab requires you to complete five Milestones:

1. Create the Notebook
2. About Running Jupyter notebooks
3. **Importing libraries and Understanding the Raw data**
4. Understanding the Data
5. Feature Engineering

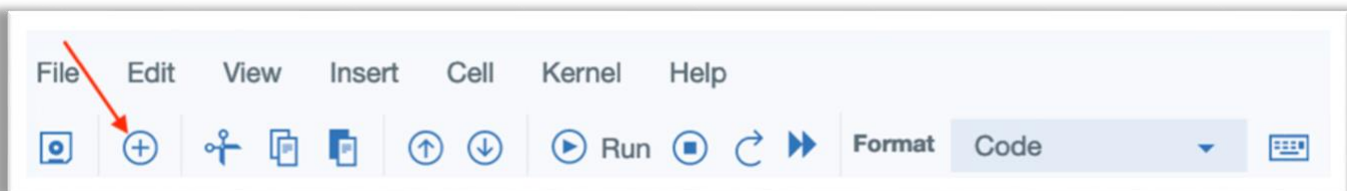
You are about to copy paste the following commands in the notebook cells. Ensure to wait until the star to the left of the cell has turned into a number, then move to the next cell using the + sign (just under the edit menu item)

### Steps

1. Copy and paste the following commands (all of it in the first cell). One of the first things that you do in notebooks, is to install and import libraries that will do various operations for you.

```
!pip install scikit-learn  
!pip install --upgrade pixiedust
```

2. Click Run from the top menu bar. Allow for the cell to run and install scikit and pixiedust. Only after the [\*] in a cell has become a whole number, then proceed to the next cell.
3. Click the + sign from the menu bar to create another cell below.



- Copy and paste the following command in the newly created cell.

```
!pip install brunel
!pip install ibm_watson_machine_learning
import pixiedust
import sklearn
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.model_selection import train_test_split
from scipy.io import arff
import brunel
from ibm_watson_machine_learning import APIClient
```

Allow for the cell to run completely. This import takes upwards of two minutes. Only after the cell displays [2], then create another cell underneath.

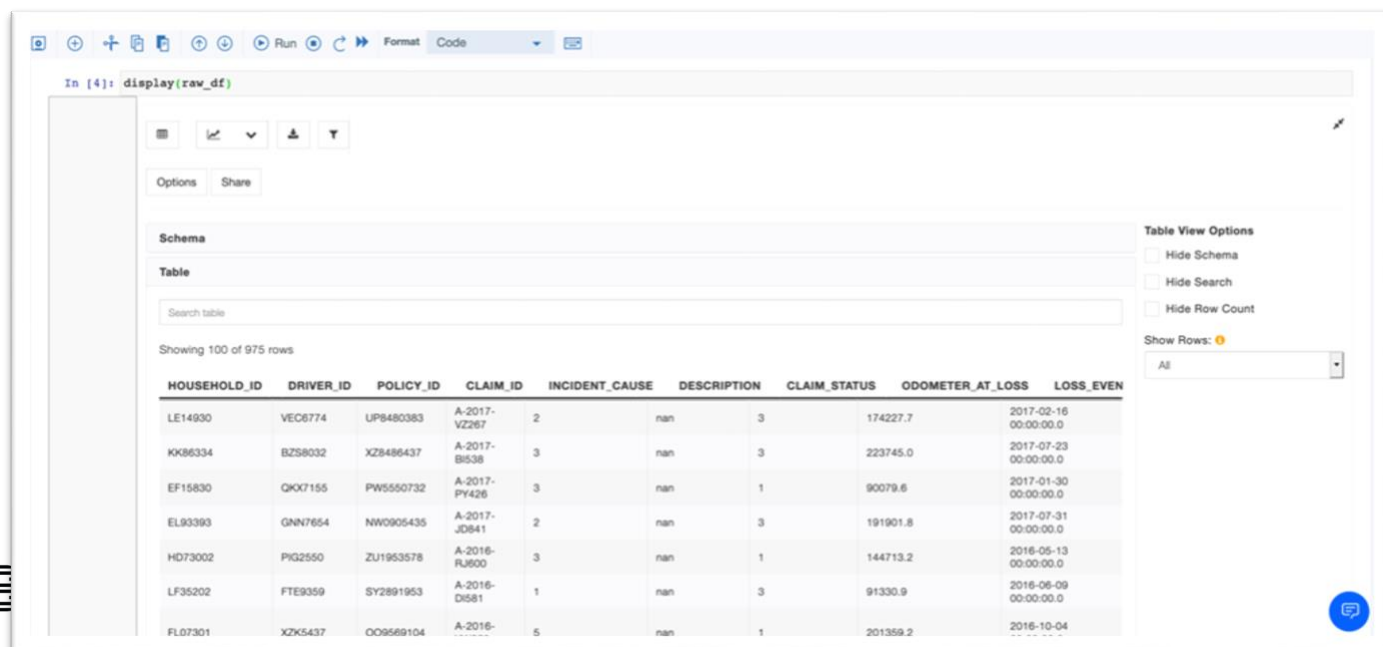
- Enter the following command in the new cell:

```
raw_df=pixiedust.sampleData('https://raw.githubusercontent.com/apischdo/skillsacademy/master/Denormalized%20claims%20data.csv')
```

- Enter the Display command to view the data set using Pixiedust

```
display(raw_df)
```

- Take a moment and review the columns and values within the table



In [4]: display(raw\_df)

Options Share

Schema

Table

Search table

Showing 100 of 975 rows

HOUSEHOLD_ID	DRIVER_ID	POLICY_ID	CLAIM_ID	INCIDENT_CAUSE	DESCRIPTION	CLAIM_STATUS	ODOMETER_AT_LOSS	LOSS_EVEN
LE14930	VEC6774	UP8480383	A-2017-VZ267	2	nan	3	174227.7	2017-02-16 00:00:00.0
KK86334	BZ58032	XZ8486437	A-2017-BIS38	3	nan	3	223745.0	2017-07-23 00:00:00.0
EF15830	QKX7155	PW5550732	A-2017-PY426	3	nan	1	90079.6	2017-01-30 00:00:00.0
EL93393	GNN7654	NW0905435	A-2017-JD841	2	nan	3	191901.8	2017-07-31 00:00:00.0
HD73002	PIG2550	ZU1953578	A-2016-RU600	3	nan	1	144713.2	2016-05-13 00:00:00.0
LF35202	FTE9359	SY2891953	A-2016-DIS81	1	nan	3	91330.9	2016-06-09 00:00:00.0
FL07301	XZK5437	OO9569104	A-2016-	5	nan	1	201359.2	2016-10-04

Table View Options

- ☐ Hide Schema
- ☐ Hide Search
- ☐ Hide Row Count

Show Rows: 10

All



## Milestone 4: Understanding the Data

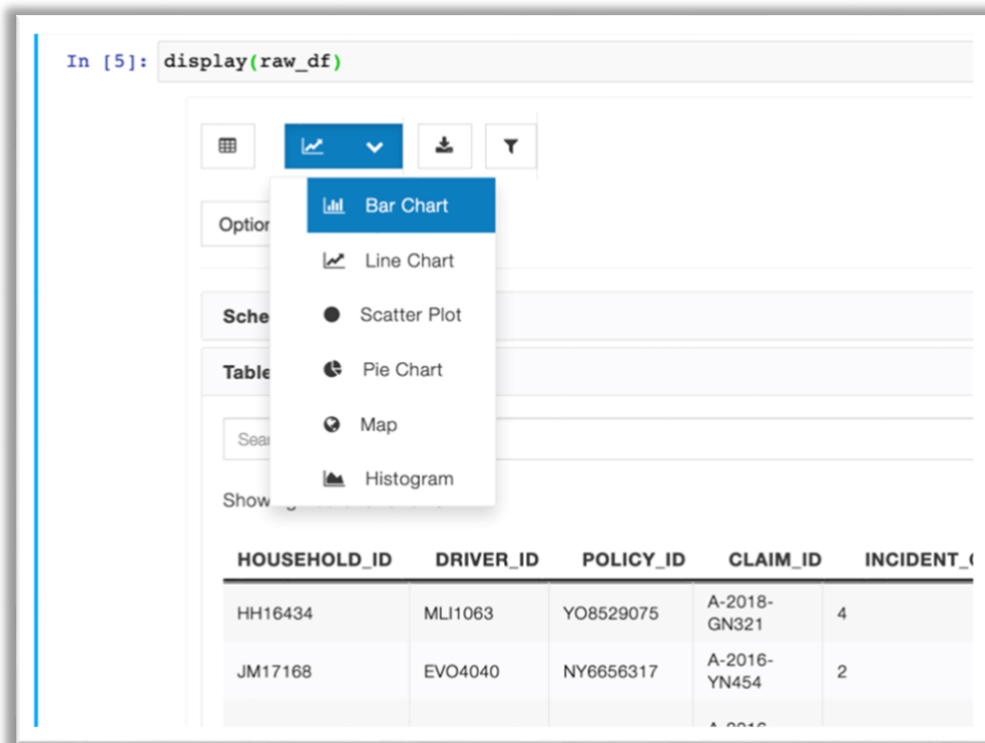
### Milestone Overview

This lab requires you to complete five Milestones:

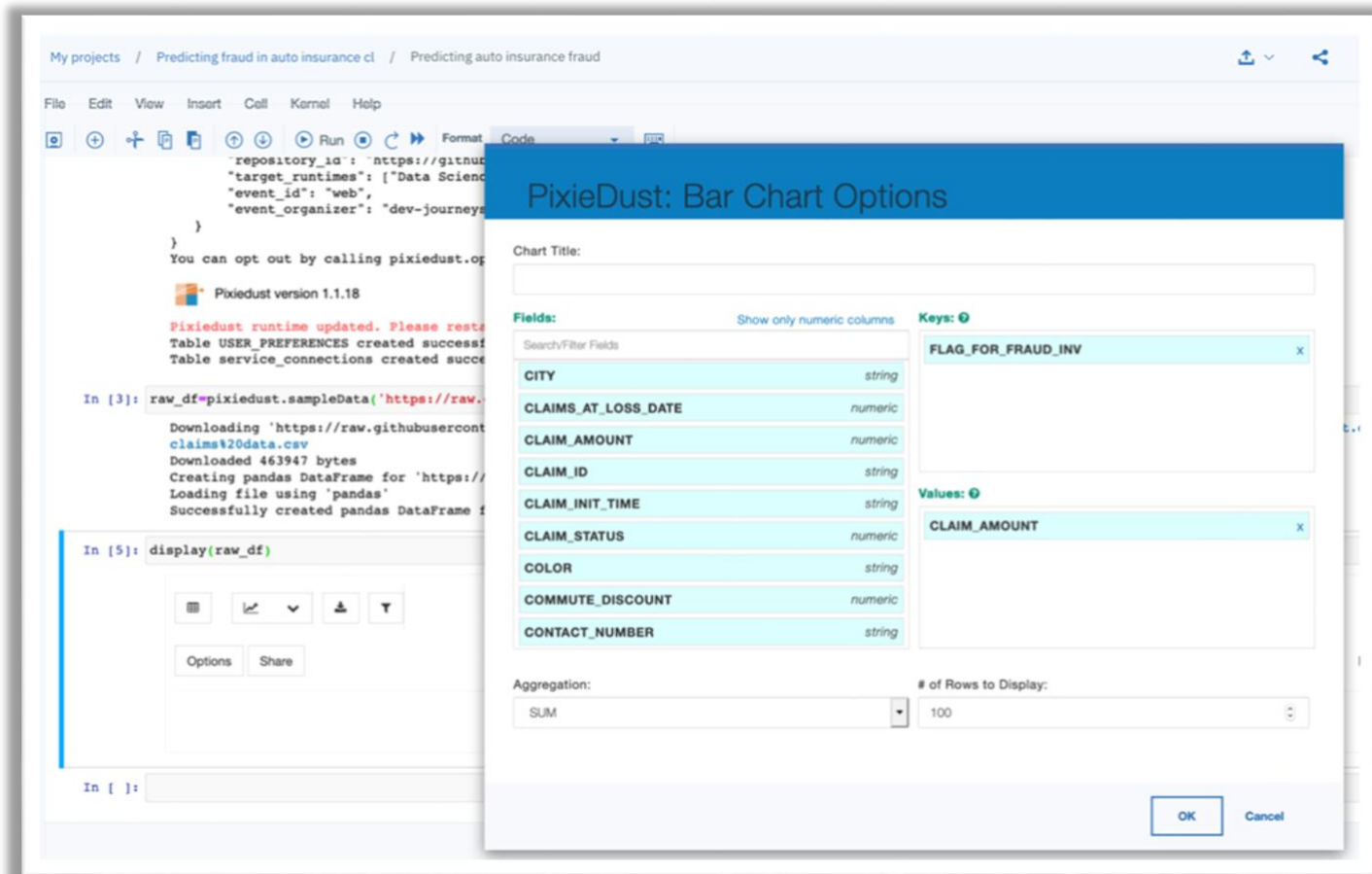
1. Create the Notebook
2. About Running Jupyter notebooks
3. Importing libraries and Understanding the Raw data
4. **Understanding the Data**
5. Feature Engineering

### Steps

1. From the drop-down menu, select **Bar Chart**.



- Find and drag the **FLAG\_FOR\_FRAUD\_INV** to the X axis and the **CLAIM\_AMOUNT** to the Y axis and click OK.



**Figure 4-2 Bar Chart Options**

- Observe the rendering.
- Perform the visualization tasks with the following dependent and independent variables:

X-axis (keys)	Y-axis (values)
FLAG_FOR_FRAUD_INV	CLAIMS_AT_LOSS_DATE
FLAG_FOR_FRAUD_INV	ODOMETER_AT_LOSS
FLAG_FOR_FRAUD_INV	POLICE_REPORT

- Let's view the presence of police reports on the map.

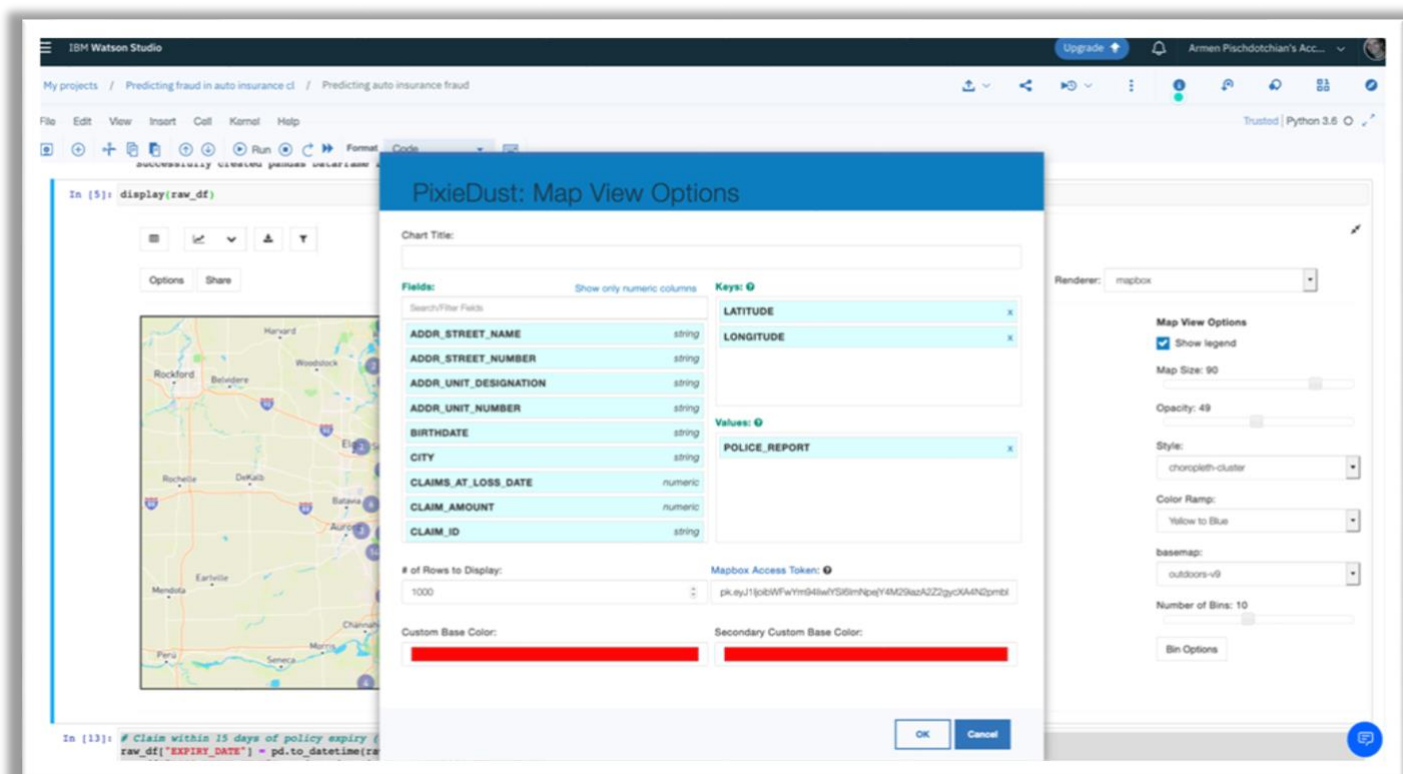


Return to the drop-down where you selected Bar Chart and this time select **Map**.

7. Click **Options** and drag the **LATTITUDE** and **LONGITUDE** to Keys section and the **POLICE\_REPORT** to the Values section.
8. If the Mapbox API key does not appear, use this key (or you may need to create your own API by following the links provided).

pk.eyJ1IjoieXBpc2NoZG8iLCJhIjoieY2o2cXkxMjUxMDMyaTJ3bGEyYjFsZ3Y4cSJ9.mL2PT0XH2vrNiDozb7gO0w

Change some of the setting to the right. You can use the settings that you see in the screen capture or your own preferences.



Notice that the closer the incident was to downtown, the greater the police reports. Perhaps you can think of ways to close the gap between plenty of accidents, yet sparse police reports in rural areas.

What other variables would you peg against the locations of incidents?

---

## Milestone 5: Feature Engineering

### Milestone Overview

This lab requires you to complete five Milestones:

1. Create the Notebook
2. About Running Jupyter notebooks
3. Importing libraries and Understanding the Raw data
4. Understanding the Data
5. **Feature Engineering**

Feature engineering goes beyond just mere calculations of values in a column; it involves creating new columns that hold 'engineered' values that begin to tell the tale of potential fraudulent behavior.

We are going to look at the following hypothesis:

- Claim within 15 days of policy expiry (date of loss - insurance\_policy.expiry)
- Expired drivers' license (if date of loss > insurance\_driver.drivers\_license\_expiry)
- Days living at current address (date of loss - insurance\_driver.date\_at\_current\_address)
- Conflict on whether a policyholder with a low mileage discount experienced a loss with high mileage at the point of loss.

Let's begin our journey with the following commands that you will enter into each cell as depicted below.

### Steps

1. Enter (copy/paste) the following command in a new cell:

```
# Claim within 15 days of policy expiry (date of loss - insurance_policy.expiry)
raw_df["EXPIRY_DATE"] = pd.to_datetime(raw_df["EXPIRY_DATE"])
raw_df["LOSS_EVENT_TIME"] = pd.to_datetime(raw_df["LOSS_EVENT_TIME"])

raw_df["DAYS_FROM_LOSS"] = raw_df["LOSS_EVENT_TIME"] - raw_df["EXPIRY_DATE"]
raw_df["DAYS_FROM_LOSS"] = abs(raw_df.DAYS_FROM_LOSS.dt.days)

raw_df.loc[raw_df['DAYS_FROM_LOSS'] >= 15, 'SUSPICIOUS_CLAIM_TIME'] = 1
raw_df.loc[raw_df['DAYS_FROM_LOSS'] < 15, 'SUSPICIOUS_CLAIM_TIME'] = 0
```





2. Enter (copy/paste) the following command in a new cell:

```
raw_df["SUSPICIOUS_CLAIM_TIME"].value_counts()
```

3. Enter (copy/paste) the following command in a new cell:

```
# Expired drivers license (if date of loss > insurance_driver.drivers_license_expiry)
raw_df["DRIVERS_LICENSE_EXPIRY"] = pd.to_datetime(raw_df["DRIVERS_LICENSE_EXPIRY"])

raw_df["DAYS_FROM_L_EXPIRY"] = raw_df["DRIVERS_LICENSE_EXPIRY"] - raw_df["LOSS_EVENT_TIME"]
raw_df["DAYS_FROM_L_EXPIRY"] = raw_df.DAYS_FROM_L_EXPIRY.dt.days

raw_df.loc[raw_df['DAYS_FROM_L_EXPIRY'] >= 0, 'EXPIRED_LICENSE'] = 0
raw_df.loc[raw_df['DAYS_FROM_L_EXPIRY'] < 0, 'EXPIRED_LICENSE'] = 1
```

4. Enter (copy/paste) the following command in a new cell:

```
# Days living at current address (date of loss - insurance_driver.date_at_current_address)

raw_df["DATE_AT_CURRENT_ADDRESS"] = pd.to_datetime(raw_df["DATE_AT_CURRENT_ADDRESS"])
raw_df["DAYS_AT_ADDRESS"] = raw_df["LOSS_EVENT_TIME"] - raw_df["DATE_AT_CURRENT_ADDRESS"]
raw_df["DAYS_AT_ADDRESS"] = abs(raw_df.DAYS_AT_ADDRESS.dt.days)

raw_df.loc[raw_df['DAYS_AT_ADDRESS'] >= 15, 'SUSPICIOUS_LIVING'] = 1
raw_df.loc[raw_df['DAYS_AT_ADDRESS'] < 15, 'SUSPICIOUS_LIVING'] = 0
```

5. Enter (copy/paste) the following command in a new cell:

```
raw_df["SUSPICIOUS_LIVING"].value_counts()
```

6. Enter (copy/paste) the following command in a new cell:

```
#7500/year
raw_df["START_DATE"] = pd.to_datetime(raw_df["START_DATE"])
#find number of days between policy creation and accident
raw_df["LENGTH_OF_POLICY"]=(raw_df["LOSS_EVENT_TIME"] - raw_df["START_DATE"]).dt.days

#convert to years
raw_df["LENGTH_OF_POLICY"]=raw_df["LENGTH_OF_POLICY"]/365

#divide Odometer at loss by years
raw_df["MILES/YEAR"] = raw_df["ODOMETER_AT_LOSS"]/raw_df["LENGTH_OF_POLICY"]
raw_df["MILES/YEAR"].value_counts()
```

7. Enter (copy/paste) the following command in a new cell:

```
# Conflict on whether a policyholder with a low mileage discount experienced a loss with high mileage at the point of loss
raw_df.loc[raw_df["MILES/YEAR"] < 7500, 'LOW_MILEAGE_AT_LOSS'] = 1
raw_df.loc[raw_df["MILES/YEAR"] >= 7500, 'LOW_MILEAGE_AT_LOSS'] = 0
```

8. Enter (copy/paste) the following command in a new cell:

```
raw_df.loc[raw_df["LOW_MILEAGE_USE"]==raw_df["LOW_MILEAGE_AT_LOSS"], 'SUSPICIOUS_MILEAGE'] = 0
raw_df.loc[raw_df["LOW_MILEAGE_USE"]!=raw_df["LOW_MILEAGE_AT_LOSS"], 'SUSPICIOUS_MILEAGE'] = 1
```

9. Enter (copy/paste) the following command in a new cell:

```
raw_df.loc[raw_df["CLAIM_AMOUNT"] < 3000, 'EXCESSIVE_CLAIM_AMOUNT'] = 0
raw_df.loc[raw_df["CLAIM_AMOUNT"] >= 3000, 'EXCESSIVE_CLAIM_AMOUNT'] = 1
```

10. Enter (copy/paste) the following command in a new cell:

```
# dataframes for certain features
features = ['FLAG_FOR_FRAUD_INV',
            'SUSPICIOUS_MILEAGE',
            'EXPIRED_LICENSE',
            'SUSPICIOUS_CLAIM_TIME',
            'SUSPICIOUS_LIVING',
            'EXCESSIVE_CLAIM_AMOUNT']
```

11. Enter (copy/paste) the following command in a new cell:

```
df_model = raw_df[features]
```

12. Enter (copy/paste) the following command in a new cell:

```
#ensure all relevant features are integers
df_model["SUSPICIOUS_LIVING"] = df_model["SUSPICIOUS_LIVING"].astype(int)
df_model["EXPIRED_LICENSE"] = df_model["EXPIRED_LICENSE"].astype(int)
df_model["SUSPICIOUS_CLAIM_TIME"] = df_model["SUSPICIOUS_CLAIM_TIME"].astype(int)
df_model["SUSPICIOUS_MILEAGE"] = df_model["SUSPICIOUS_MILEAGE"].astype(int)
df_model["EXCESSIVE_CLAIM_AMOUNT"] = df_model["EXCESSIVE_CLAIM_AMOUNT"].astype(int)
```

13. Enter (copy/paste) the following command in a new cell:

```
raw_df.groupby("FLAG_FOR_FRAUD_INV", as_index=False).mean()
```

14. Enter (copy/paste) the following command in a new cell:

```
#split data into x and y variables
xVar =
df_model[["EXPIRED_LICENSE", "SUSPICIOUS_CLAIM_TIME", "SUSPICIOUS_LIVING", "SUSPICIOUS_MILEAGE", "EXCESSIVE_CLAIM_AMOUNT"]]

yVar = df_model["FLAG_FOR_FRAUD_INV"]
```



15. Enter (copy/paste) the following command in a new cell:

```
xVar.head()
```

16. Enter (copy/paste) the following command in a new cell:

```
#split into a test/train set
X_train, X_test, y_train, y_test = train_test_split(xVar, yVar, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

17. Enter (copy/paste) the following command in a new cell:

```
#train model
clf = RandomForestClassifier(n_jobs=2, random_state=0)

clf.fit(X_train, y_train)
```

18. Enter (copy/paste) the following command in a new cell:

```
#create confusion matrix to gut check model
preds = clf.predict(X_test)
pd.crosstab(y_test, preds, rownames=['Actual Result'], colnames=['Predicted Result'])
```

Let's take a moment and understand the significance of the confusion matrix below (your results may vary).

Predicted Result		0	1
Actual Result	0	108	12
	1	5	70