# Exploratory Data Analysis Using Python and BigQuery

## Learning Objectives

1. Analyze a Pandas Dataframe
2. Create Seaborn plots for Exploratory Data Analysis in Python
3. Write a SQL query to pick up specific fields from a BigQuery dataset
4. Exploratory Analysis in BigQuery

## Introduction

This lab is an introduction to linear regression using Python and Scikit-Learn. This lab serves as a foundation for more complex algorithms and machine learning models that you will encounter in the course. We will train a linear regression model to predict housing price.

Each learning objective will correspond to a **#TODO** in the student lab notebook -- try to complete that notebook first before reviewing this solution notebook.

### Import Libraries

In [ ]:
```
# Run the chown command to change the ownership
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

In [ ]:
```
# Install the Google Cloud BigQuery library
!pip install --user google-cloud-bigquery==1.25.0
```

Please ignore any incompatibility warnings and errors.

**Restart** the kernel before proceeding further (On the Notebook menu - Kernel - Restart Kernel).

In [2]:
```
# You can use any Python source file as a module by executing an import statement in some other Python source file.
# The import statement combines two operations; it searches for the named module, then it binds the results of that search
# to a name in the local scope.
import os
import pandas as pd
import numpy as np
# Import matplotlib to visualize the model
import matplotlib.pyplot as plt
# Seaborn is a Python data visualization library based on matplotlib
import seaborn as sns
%matplotlib inline
```

### Load the Dataset

Here, we create a directory called usahousing. This directory will hold the dataset that we copy from Google Cloud Storage.

In [3]:
```
# Create a directory to hold the dataset
if not os.path.isdir("../data/explore"):
    os.makedirs("../data/explore")
```

Next, we copy the Usahousing dataset from Google Cloud Storage.

In [4]:
```
# Copy the file using `gsutil cp` from Google Cloud Storage in the required directory
!gsutil cp gs://cloud-training-demos/feat_eng/housing/housing_pre-proc.csv ../data/explore
```

```
Copying gs://cloud-training-demos/feat_eng/housing/housing_pre-proc.csv...
/ [1 files][  1.4 MiB/  1.4 MiB]
Operation completed over 1 objects/1.4 MiB.
```

Then we use the "ls" command to list files in the directory. This ensures that the dataset was copied.

In [5]:
```
# `ls` shows the working directory's contents.
# The `l` flag list the all files with permissions and details
!ls -l ../data/explore
```

```
total 1404
-rw-r--r-- 1 jupyter jupyter 1435069 Jun 24 21:24 housing_pre-proc.csv
```

Next, we read the dataset into a Pandas dataframe.

In [6]:
```
# TODO 1
# Read a comma-separated values (csv) file into a DataFrame using the read_csv() function
```

```
df_USAhousing = pd.read_csv('../data/explore/housing_pre-proc.csv')
```

## Inspect the Data

In [7]:

```
# Get the first five rows using the head() method

df_USAhousing.head()
```

Out[7]:

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | households | median_income | median_house_value | ocean_proximity |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41.0 | 880.0 | 129.0 | 322.0 | 126.0 | 8.3252 | 452600.0 | NEAR BAY |
| **1** | -122.22 | 37.86 | 21.0 | 7099.0 | 1106.0 | 2401.0 | 1138.0 | 8.3014 | 358500.0 | NEAR BAY |
| **2** | -122.24 | 37.85 | 52.0 | 1467.0 | 190.0 | 496.0 | 177.0 | 7.2574 | 352100.0 | NEAR BAY |
| **3** | -122.25 | 37.85 | 52.0 | 1274.0 | 235.0 | 558.0 | 219.0 | 5.6431 | 341300.0 | NEAR BAY |
| **4** | -122.25 | 37.85 | 52.0 | 1627.0 | 280.0 | 565.0 | 259.0 | 3.8462 | 342200.0 | NEAR BAY |

Let's check for any null values.

In [8]:

```
# `isnull()` finds a null value in a column and `sum()` counts it
df_USAhousing.isnull().sum()
```

Out[8]:

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

In [9]:

```
# Get some basic statistical details using describe() method
df_stats = df_USAhousing.describe()
# Transpose index and columns of the dataframe
df_stats = df_stats.transpose()
df_stats
```

Out[9]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **longitude** | 20433.0 | -119.570689 | 2.003578 | -124.3500 | -121.8000 | -118.4900 | -118.010 | -114.3100 |
| **latitude** | 20433.0 | 35.633221 | 2.136348 | 32.5400 | 33.9300 | 34.2600 | 37.720 | 41.9500 |
| **housing_median_age** | 20433.0 | 28.633094 | 12.591805 | 1.0000 | 18.0000 | 29.0000 | 37.000 | 52.0000 |
| **total_rooms** | 20433.0 | 2636.504233 | 2185.269567 | 2.0000 | 1450.0000 | 2127.0000 | 3143.000 | 39320.0000 |
| **total_bedrooms** | 20433.0 | 537.870553 | 421.385070 | 1.0000 | 296.0000 | 435.0000 | 647.000 | 6445.0000 |
| **population** | 20433.0 | 1424.946949 | 1133.208490 | 3.0000 | 787.0000 | 1166.0000 | 1722.000 | 35682.0000 |
| **households** | 20433.0 | 499.433465 | 382.299226 | 1.0000 | 280.0000 | 409.0000 | 604.000 | 6082.0000 |
| **median_income** | 20433.0 | 3.871162 | 1.899291 | 0.4999 | 2.5637 | 3.5365 | 4.744 | 15.0001 |
| **median_house_value** | 20433.0 | 206864.413155 | 115435.667099 | 14999.0000 | 119500.0000 | 179700.0000 | 264700.000 | 500001.0000 |

In [10]:

```
# Get a concise summary of a DataFrame
df_USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20433 entries, 0 to 20432
Data columns (total 10 columns):
longitude             20433 non-null float64
latitude              20433 non-null float64
housing_median_age    20433 non-null float64
total_rooms           20433 non-null float64
total_bedrooms        20433 non-null float64
population            20433 non-null float64
households            20433 non-null float64
median_income         20433 non-null float64
median_house_value    20433 non-null float64
ocean_proximity       20433 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Let's take a peek at the first and last five rows of the data for all columns.

In [11]:

```
print ("Rows     : " ,df_USAhousing.shape[0])
print ("Columns  : " ,df_USAhousing.shape[1])
print ("\nFeatures : \n" ,df_USAhousing.columns.tolist())
print ("\nMissing values :  ", df_USAhousing.isnull().sum().values.sum())
print ("\nUnique values :  \n",df_USAhousing
       .nunique())
```

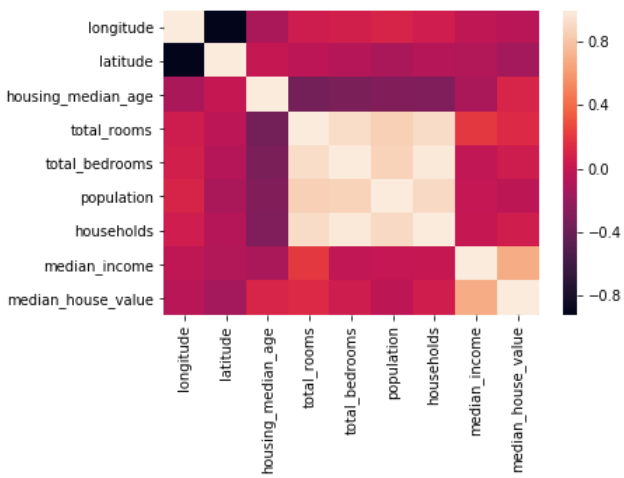
```
Rows     :  20433
Columns  :  10

Features :
 ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'medi
an_house_value', 'ocean_proximity']

Missing values :   0

Unique values :
 longitude               844
latitude               861
housing_median_age      52
total_rooms           5911
total_bedrooms        1923
population            3879
households            1809
median_income        12825
median_house_value    3833
ocean_proximity          5
dtype: int64
```

## Explore the Data

Let's create some simple plots to check out the data!

In [12]:
```
# `heatmap` plots a rectangular data in a color-encoded matrix and
# `corr` finds the pairwise correlation of all columns in the dataframe
sns.heatmap(df_USAhousing.corr())
```

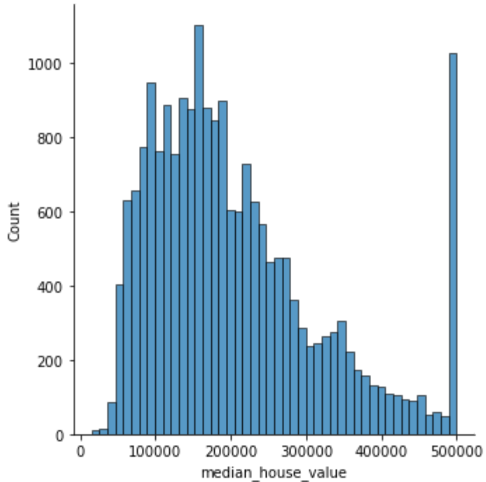
Out[12]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f25ba7c1dd8>`




Create a distplot showing "median_house_value".

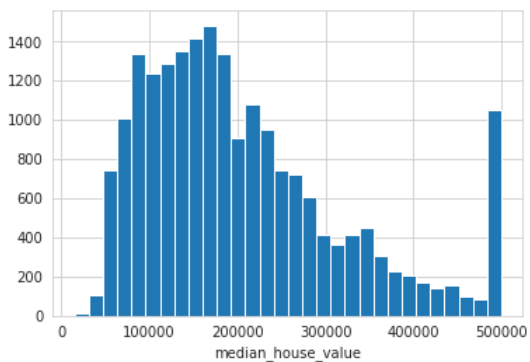
In [13]:
```
# TODO 2a
# Plot a univariate distribution of observations using seaborn `distplot()` function
sns.displot(df_USAhousing['median_house_value'])
```

Out[13]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f25ba6d15f8>`

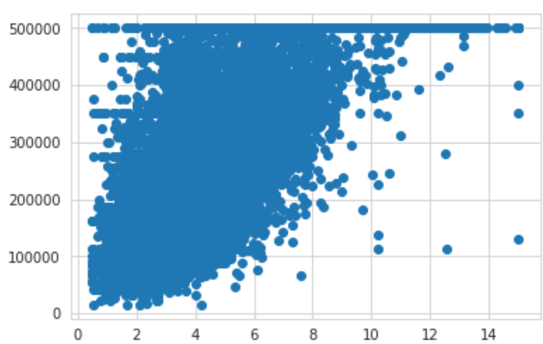



In [14]:
```
# Set the aesthetic style of the plots
sns.set_style('whitegrid')
# Plot a histogram using `hist()` function
df_USAhousing['median_house_value'].hist(bins=30)
plt.xlabel('median_house_value')
```

Out[14]: `Text(0.5, 0, 'median_house_value')`



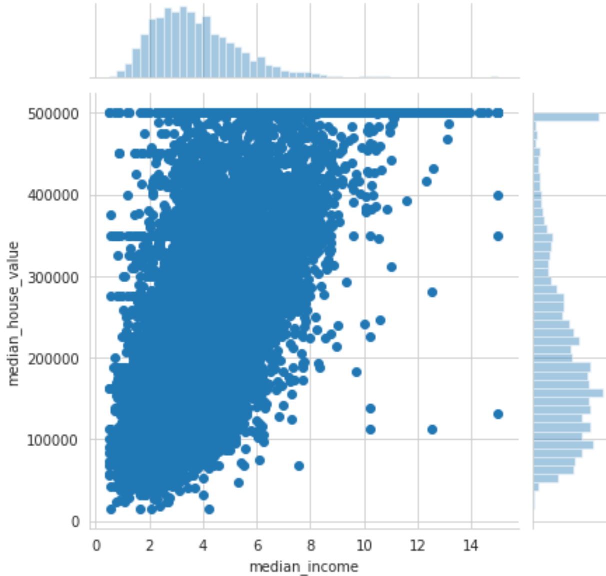

In [15]:
```
x = df_USAhousing['median_income']
y = df_USAhousing['median_house_value']

# Scatter plot of y vs x using scatter() and `show()` display all open figures
plt.scatter(x, y)
plt.show()
```



Create a jointplot showing "median_income" versus "median_house_value".

In [16]:
```
# TODO 2b
# `joinplot()` draws a plot of two variables with bivariate and univariate graphs.
sns.jointplot(x='median_income',y='median_house_value',data=df_USAhousing)
```

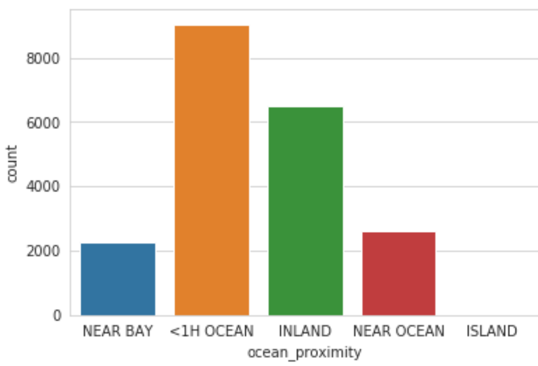
Out[16]: `<seaborn.axisgrid.JointGrid at 0x7f25ba74a630>`




In [17]:

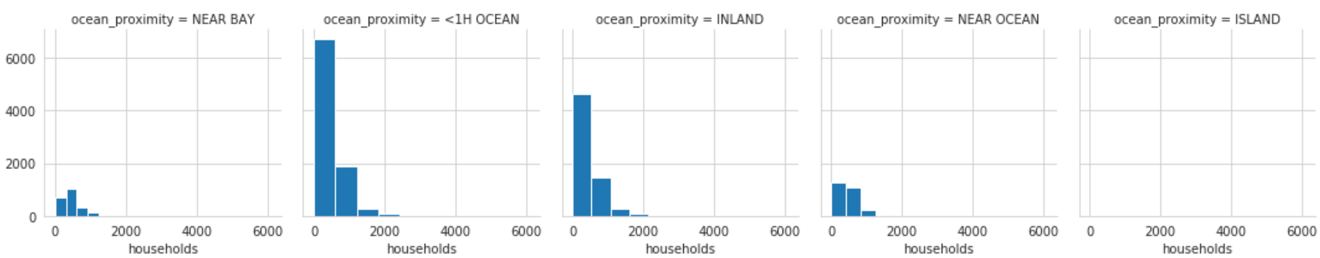
```
# `countplot()` shows the counts of observations in each categorical bin using bars
sns.countplot(x = 'ocean_proximity', data=df_USAhousing)
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f25ba4ef400>



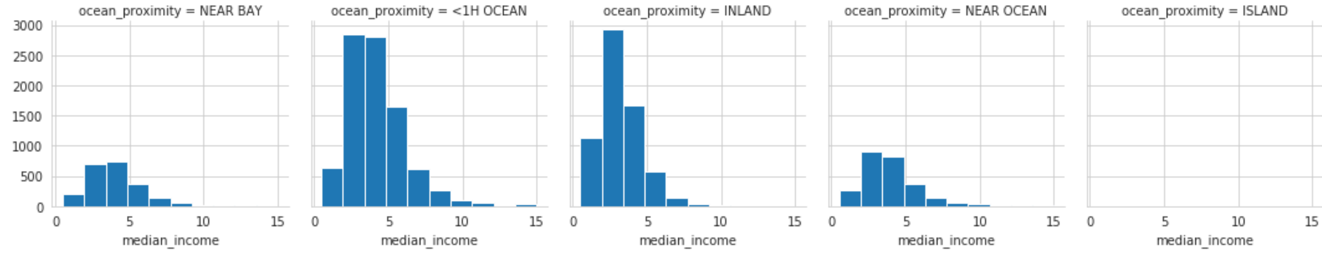

In [18]:

```
# takes numeric only?
# plt.figure(figsize=(20,20))
# Draw a multi-plot on every facet using `FacetGrid()`
g = sns.FacetGrid(df_USAhousing, col="ocean_proximity")
# Pass a function and the name of one or more columns in the dataframe
g.map(plt.hist, "households");
```



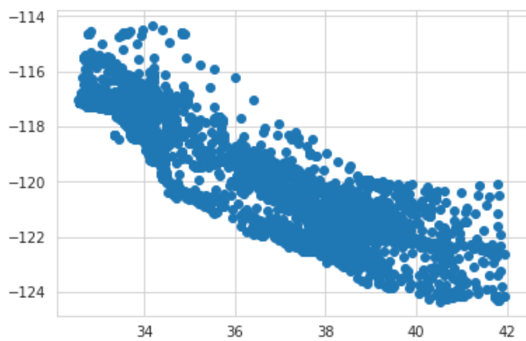

In [60]:

```
# takes numeric only?
# plt.figure(figsize=(20,20))
# Draw a multi-plot on every facet using `FacetGrid()`
g = sns.FacetGrid(df_USAhousing, col="ocean_proximity")
# Pass a function and the name of one or more columns in the dataframe
g.map(plt.hist, "median_income");
```




You can see below that this is the state of California!

In [20]:
```
x = df_USAhousing['latitude']
y = df_USAhousing['longitude']

# Scatter plot of y vs x and display all open figures
plt.scatter(x, y)
plt.show()
```



# Explore and create ML datasets

In this notebook, we will explore data corresponding to taxi rides in New York City to build a Machine Learning model in support of a fare-estimation tool. The idea is to suggest a likely fare to taxi riders so that they are not surprised, and so that they can protest if the charge is much higher than expected.

## Learning Objectives

- Access and explore a public BigQuery dataset on NYC Taxi Cab rides
- Visualize your dataset using the Seaborn library

First, **restart the Kernel**. Now, let's start with the Python imports that we need.

In [1]:
```
# Import the python libraries
from google.cloud import bigquery
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

### Extract sample data from BigQuery

The dataset that we will use is a BigQuery public dataset. Click on the link, and look at the column names. Switch to the Details tab to verify that the number of records is one billion, and then switch to the Preview tab to look at a few rows.

Let's write a SQL query to pick up interesting fields from the dataset. It's a good idea to get the timestamp in a predictable format.

In [2]:
```
%%bigquery
# SQL query to get a fields from dataset which prints the 10 records
SELECT
    FORMAT_TIMESTAMP(
        "%Y-%m-%d %H:%M:%S %Z", pickup_datetime) AS pickup_datetime,
    pickup_longitude, pickup_latitude, dropoff_longitude,
    dropoff_latitude, passenger_count, trip_distance, tolls_amount,
    fare_amount, total_amount
# TODO 3
FROM
    `nyc-tlc.yellow.trips`
LIMIT 10
```

Out[2]:

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount | total_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2010-03-23 16:04:00 UTC | -73.937463 | 40.758173 | -73.937462 | 40.758175 | 1 | 0.00 | 0.0 | 0.0 | |
| **1** | 2010-03-31 19:34:22 UTC | -73.848701 | 40.738563 | -73.980793 | 40.764666 | 1 | 83.60 | 0.0 | 0.0 | |
| **2** | 2010-02-18 23:22:37 UTC | -74.045752 | 40.720509 | -74.045752 | 40.720509 | 3 | 0.00 | 0.0 | 0.0 | |
| **3** | 2015-01-10 15:20:08 UTC | -73.972122 | 40.760014 | -73.971947 | 40.760063 | 1 | 0.05 | 0.0 | 0.0 | |
| **4** | 2010-03-26 21:44:51 UTC | -73.989504 | 40.730073 | -73.915033 | 40.766109 | 1 | 6.00 | 0.0 | 0.0 | |
| **5** | 2010-02-26 21:15:13 UTC | -73.790439 | 40.644783 | -74.036504 | 40.722646 | 2 | 218.00 | 0.0 | 0.0 | |
| **6** | 2010-02-14 23:41:37 UTC | -73.862951 | 40.734516 | -73.982297 | 40.761860 | 1 | 80.00 | 0.0 | 0.0 | |
| **7** | 2010-03-27 14:46:11 UTC | -73.980504 | 40.748292 | -73.980181 | 40.748143 | 1 | 0.00 | 0.0 | 0.0 | |
| **8** | 2010-02-04 01:00:03 UTC | -73.903466 | 40.747651 | -73.903396 | 40.747749 | 1 | 0.30 | 0.0 | 0.0 | |
| **9** | 2013-08-01 01:23:36 UTC | -73.980993 | 40.764660 | -73.981463 | 40.764557 | 1 | 2.80 | 0.0 | 0.0 | |

Let's increase the number of records so that we can do some neat graphs. There is no guarantee about the order in which records are returned, and so no guarantee about which records get returned if we simply increase the LIMIT. To properly sample the dataset, let's use the HASH of the pickup time and return 1 in 100,000 records -- because there are 1 billion records in the data, we should get back approximately 10,000 records if we do this.

We will also store the BigQuery result in a Pandas dataframe named "trips"

In [3]:

```
%%bigquery trips
# SQL query to save the results in the trips dataframe
SELECT
    FORMAT_TIMESTAMP(
        "%Y-%m-%d %H:%M:%S %Z", pickup_datetime) AS pickup_datetime,
    pickup_longitude, pickup_latitude,
    dropoff_longitude, dropoff_latitude,
    passenger_count,
    trip_distance,
    tolls_amount,
    fare_amount,
    total_amount
FROM
    `nyc-tlc.yellow.trips`
WHERE
    ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 100000)) = 1
```

Out[3]:

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2011-06-04 02:52:10 UTC | -73.984681 | 40.769893 | -74.007312 | 40.705326 | 1 | 5.30 | 0.00 | 15.3 |
| **1** | 2014-10-06 15:16:00 UTC | -73.980130 | 40.760910 | -73.861730 | 40.768330 | 2 | 11.47 | 5.33 | 36.5 |
| **2** | 2014-12-08 21:50:00 UTC | -73.870867 | 40.773782 | -74.003297 | 40.708215 | 2 | 11.81 | 0.00 | 33.5 |
| **3** | 2010-05-26 16:15:03 UTC | -74.002922 | 40.714474 | -73.978505 | 40.758280 | 1 | 6.10 | 0.00 | 20.9 |
| **4** | 2012-05-05 22:46:05 UTC | -74.009790 | 40.712483 | -73.959293 | 40.768908 | 1 | 5.20 | 0.00 | 16.9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10784** | 2010-02-25 20:14:00 UTC | -73.991358 | 40.749645 | -73.971768 | 40.792898 | 5 | 3.84 | 0.00 | 12.9 |
| **10785** | 2012-06-06 12:34:32 UTC | -73.946107 | 40.778459 | -73.975723 | 40.760311 | 1 | 2.80 | 0.00 | 12.9 |
| **10786** | 2010-05-06 18:16:06 UTC | -73.989262 | 40.741876 | -73.951123 | 40.782685 | 1 | 4.00 | 0.00 | 12.9 |
| **10787** | 2012-08-10 12:17:02 UTC | -74.015546 | 40.707420 | -73.990367 | 40.729593 | 1 | 2.90 | 0.00 | 12.9 |

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount |
|---|---|---|---|---|---|---|---|---|---|
| **10788** | 2011-08-11 23:35:42 UTC | -73.978100 | 40.725390 | -73.992485 | 40.698344 | 1 | 4.00 | 0.00 | 12.9 |

10789 rows × 10 columns

In [4]:

```
print(len(trips))
```

```
10789
```

In [5]:

```
# We can slice Pandas dataframes as if they were arrays
trips[:10]
```

Out[5]:

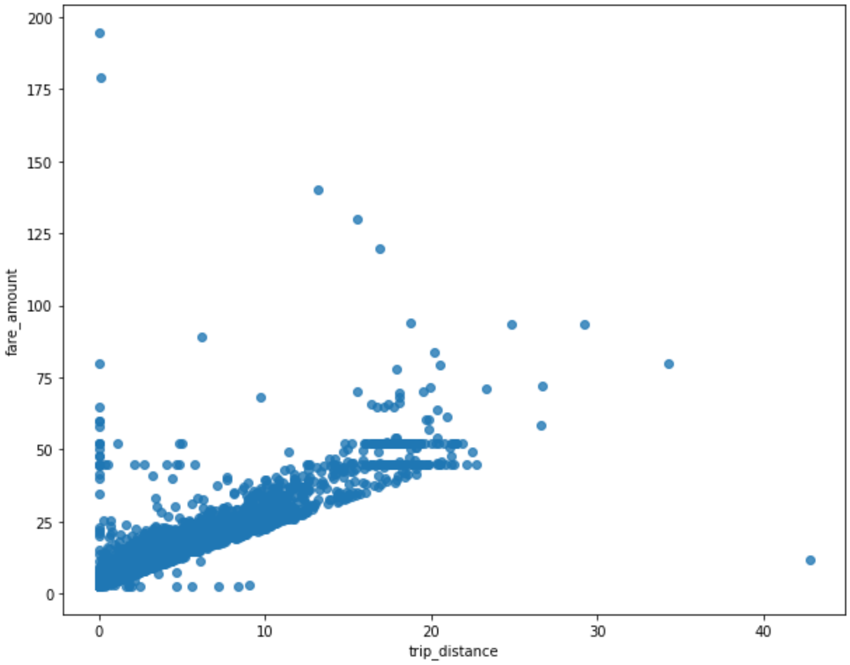
| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount | total_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2011-06-04 02:52:10 UTC | -73.984681 | 40.769893 | -74.007312 | 40.705326 | 1 | 5.30 | 0.00 | 15.3 | |
| **1** | 2014-10-06 15:16:00 UTC | -73.980130 | 40.760910 | -73.861730 | 40.768330 | 2 | 11.47 | 5.33 | 36.5 | |
| **2** | 2014-12-08 21:50:00 UTC | -73.870867 | 40.773782 | -74.003297 | 40.708215 | 2 | 11.81 | 0.00 | 33.5 | |
| **3** | 2010-05-26 16:15:03 UTC | -74.002922 | 40.714474 | -73.978505 | 40.758280 | 1 | 6.10 | 0.00 | 20.9 | |
| **4** | 2012-05-05 22:46:05 UTC | -74.009790 | 40.712483 | -73.959293 | 40.768908 | 1 | 5.20 | 0.00 | 16.9 | |
| **5** | 2010-12-21 13:08:00 UTC | -73.982422 | 40.739847 | -73.981658 | 40.768732 | 2 | 2.64 | 0.00 | 14.9 | |
| **6** | 2011-12-03 10:28:00 UTC | -73.998822 | 40.680933 | -73.968960 | 40.757878 | 1 | 8.28 | 0.00 | 20.9 | |
| **7** | 2014-05-20 23:09:00 UTC | -73.995203 | 40.727307 | -73.948775 | 40.813487 | 1 | 10.31 | 0.00 | 33.5 | |
| **8** | 2010-10-09 23:50:28 UTC | -73.956644 | 40.771152 | -74.005279 | 40.740280 | 1 | 10.10 | 0.00 | 25.7 | |
| **9** | 2012-07-05 14:18:00 UTC | -73.916517 | 40.743212 | -73.956785 | 40.780882 | 1 | 4.74 | 0.00 | 15.7 | |

## Exploring data

Let's explore this dataset and clean it up as necessary. We'll use the Python Seaborn package to visualize graphs and Pandas to do the slicing and filtering.

In [6]:

```
# TODO 4
# Use Seaborn `regplot()` function to plot the data and a linear regression model fit.
ax = sns.regplot(
    x="trip_distance", y="fare_amount",
    fit_reg=False, ci=None, truncate=True, data=trips)
ax.figure.set_size_inches(10, 8)
```




Hmm ... do you see something wrong with the data that needs addressing?

It appears that we have a lot of invalid data that is being coded as zero distance and some fare amounts that are definitely illegitimate. Let's remove them from our analysis. We can do this by modifying the BigQuery query to keep only trips longer than zero miles and fare amounts that are at least the minimum cab fare ($2.50).

Note the extra WHERE clauses.

In [7]:

```
%%bigquery trips
# SQL query with where clause to save the results in the trips dataframe
SELECT
    FORMAT_TIMESTAMP(
        "%Y-%m-%d %H:%M:%S %Z", pickup_datetime) AS pickup_datetime,
    pickup_longitude, pickup_latitude,
    dropoff_longitude, dropoff_latitude,
    passenger_count,
    trip_distance,
    tolls_amount,
    fare_amount,
    total_amount
FROM
    `nyc-tlc.yellow.trips`
WHERE
    ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)), 100000)) = 1
# TODO 4a
    AND trip_distance > 0
    AND fare_amount >= 2.5
```

Out[7]:

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2014-10-06 15:16:00 UTC | -73.980130 | 40.760910 | -73.861730 | 40.768330 | 2 | 11.47 | 5.33 | 36.5 |
| **1** | 2014-12-08 21:50:00 UTC | -73.870867 | 40.773782 | -74.003297 | 40.708215 | 2 | 11.81 | 0.00 | 33.5 |
| **2** | 2010-05-26 16:15:03 UTC | -74.002922 | 40.714474 | -73.978505 | 40.758280 | 1 | 6.10 | 0.00 | 20.9 |
| **3** | 2012-05-05 22:46:05 UTC | -74.009790 | 40.712483 | -73.959293 | 40.768908 | 1 | 5.20 | 0.00 | 16.9 |
| **4** | 2010-12-21 13:08:00 UTC | -73.982422 | 40.739847 | -73.981658 | 40.768732 | 2 | 2.64 | 0.00 | 14.9 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10711** | 2010-02-25 20:14:00 UTC | -73.991358 | 40.749645 | -73.971768 | 40.792898 | 5 | 3.84 | 0.00 | 12.9 |
| **10712** | 2012-06-06 12:34:32 UTC | -73.946107 | 40.778459 | -73.975723 | 40.760311 | 1 | 2.80 | 0.00 | 12.9 |
| **10713** | 2011-10-27 07:55:01 UTC | -73.991334 | 40.749870 | -73.954420 | 40.764275 | 1 | 3.20 | 0.00 | 12.9 |

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount |
|---|---|---|---|---|---|---|---|---|---|
| **10714** | 2010-05-06 18:16:06 UTC | -73.989262 | 40.741876 | -73.951123 | 40.782685 | 1 | 4.00 | 0.00 | 12.9 |
| **10715** | 2009-09-25 03:47:00 UTC | -73.991533 | 40.735107 | -73.964660 | 40.687678 | 1 | 4.69 | 0.00 | 12.9 |

10716 rows × 10 columns

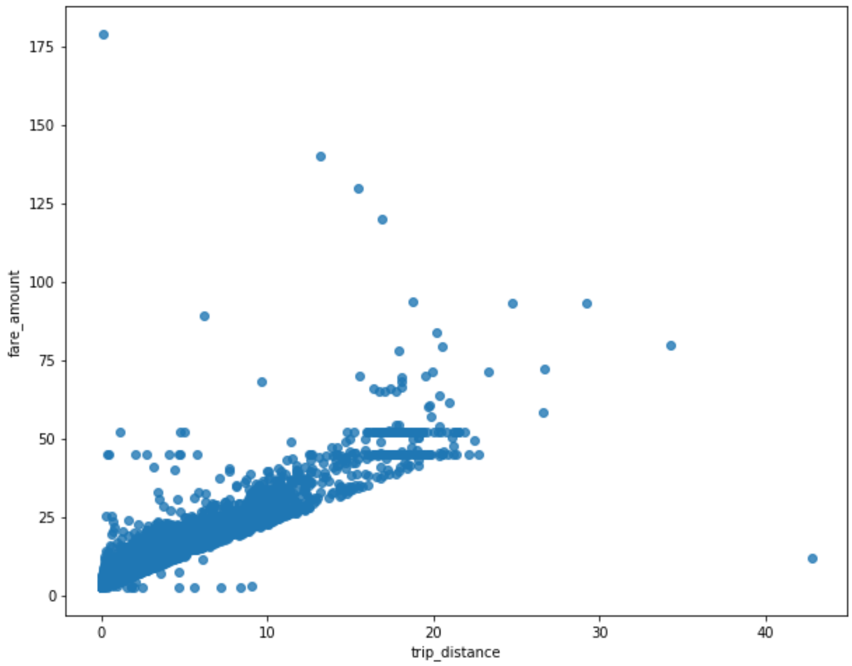
In [8]:

```
print(len(trips))
```

```
10716
```

In [9]:

```
# Use Seaborn `regplot()` function to plot the data and a linear regression model fit.
ax = sns.regplot(
    x="trip_distance", y="fare_amount",
    fit_reg=False, ci=None, truncate=True, data=trips)
ax.figure.set_size_inches(10, 8)
```




What's up with the streaks around 45 dollars and 50 dollars? Those are fixed-amount rides from JFK and La Guardia airports into anywhere in Manhattan, i.e. to be expected. Let's list the data to make sure the values look reasonable.

Let's also examine whether the toll amount is captured in the total amount.

In [10]:

```
tollrides = trips[trips["tolls_amount"] > 0]
tollrides[tollrides["pickup_datetime"] == "2012-02-27 09:19:10 UTC"]
```

Out[10]:

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount | tota |
|---|---|---|---|---|---|---|---|---|---|---|
| **16** | 2012-02-27 09:19:10 UTC | -73.874431 | 40.774011 | -73.983967 | 40.744082 | 1 | 11.6 | 4.8 | 27.7 | |

In [11]:

```
notollrides = trips[trips["tolls_amount"] == 0]
notollrides[notollrides["pickup_datetime"] == "2012-02-27 09:19:10 UTC"]
```

Out[11]:

| | pickup_datetime | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount |
|---|---|---|---|---|---|---|---|---|---|
| **52** | 2012-02-27 09:19:10 UTC | -73.972311 | 40.753067 | -73.957389 | 40.817824 | 1 | 5.6 | 0.0 | 16.9 |
| **7799** | 2012-02-27 09:19:10 UTC | -73.987582 | 40.725468 | -74.016628 | 40.715534 | 1 | 2.8 | 0.0 | 12.1 |
| **10537** | 2012-02-27 09:19:10 UTC | -74.015483 | 40.715279 | -73.998045 | 40.756273 | 1 | 3.3 | 0.0 | 10.9 |

Looking at a few samples above, it should be clear that the total amount reflects fare amount, toll and tip somewhat arbitrarily -- this is because when customers pay cash, the tip is not known. So, we'll use the sum of fare_amount + tolls_amount as what needs to be predicted. Tips are discretionary and do not have to be included in our fare estimation tool.

Let's also look at the distribution of values within the columns.

In [12]:

```
# Print the distribution of values within the columns using `describe()`
trips.describe()
```

Out[12]:

| | pickup_longitude | pickup_latitude | dropoff_longitude | dropoff_latitude | passenger_count | trip_distance | tolls_amount | fare_amount | total_amount |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 | 10716.000000 |
| **mean** | -72.602192 | 40.002372 | -72.594838 | 40.002052 | 1.650056 | 2.856395 | 0.226428 | 11.109446 | 13.217078 |
| **std** | 9.982373 | 5.474670 | 10.004324 | 5.474648 | 1.283577 | 3.322024 | 1.135934 | 9.137710 | 10.953156 |
| **min** | -74.258183 | 0.000000 | -74.260472 | 0.000000 | 0.000000 | 0.010000 | 0.000000 | 2.500000 | 2.500000 |
| **25%** | -73.992153 | 40.735936 | -73.991566 | 40.734310 | 1.000000 | 1.040000 | 0.000000 | 6.000000 | 7.300000 |
| **50%** | -73.981851 | 40.753264 | -73.980373 | 40.752956 | 1.000000 | 1.770000 | 0.000000 | 8.500000 | 10.000000 |
| **75%** | -73.967400 | 40.767340 | -73.964142 | 40.767510 | 2.000000 | 3.160000 | 0.000000 | 12.500000 | 14.600000 |
| **max** | 0.000000 | 41.366138 | 0.000000 | 41.366138 | 6.000000 | 42.800000 | 16.000000 | 179.000000 | 179.000000 |