Deep Learning Walkthrough - 04

Code in github.com/google-aai/sc17

Cassie Kozyrkov
Chief Decision Scientist, Google Cloud
GitHub: kozyrkov; Twitter: @quaesita

Google Cloud

Step 4 | Explore some data

Avoid danger and unleash your creativity

Chief actors:

Analyst

Domain expert

Danger! Pitfall alert

Don’t look at validation and testing instances.

Design your fixing and processing code using inspiration from you training data, then apply those scripts to all your datasets.

Garbage in, garbage out

Your ML model is only as good as the data that goes into it

Google Cloud

Garbage in, garbage out

You need to check for:

- Bad instances
  - missing data
  - values out of range
  - bugs / data entry errors

Why?

- Diagnose bad features
- Fix / clean instances

Wine example

Instance ID	Label	Grape	Vintage	...
Wine1	yes	syrah	2012	...
Wine2	yes	pinotage	2010	...
Wine3	no	merlot	15	...
Wine4	no	riesling	2015	...
Wine5	no	moscato	2014	...
...	...	...	...	...

Columns: features (a.k.a. variables, predictors, attributes)
Rows: instances (a.k.a. examples, observations, records)
Correct answers: labels (a.k.a. targets, ground truth)

Wine example

Instance ID	Label	Grape	Vintage	...
Wine1	yes	syrah	2012	...
Wine2	yes	pinotage	2010	...
Wine3	no	merlot	2015	...
Wine4	no	riesling	2015	...
Wine5	no	moscato	2014	...
...	...	...	...	...

Columns: features (a.k.a. variables, predictors, attributes)
Rows: instances (a.k.a. examples, observations, records)
Correct answers: labels (a.k.a. targets, ground truth)

Don’t forget to clean it

Part of getting high quality features is the data wrangling and cleaning process

Garbage in, garbage out

You need to check for:

- Bad features
  - no variability
  - bad distribution
  - bugs

Why?

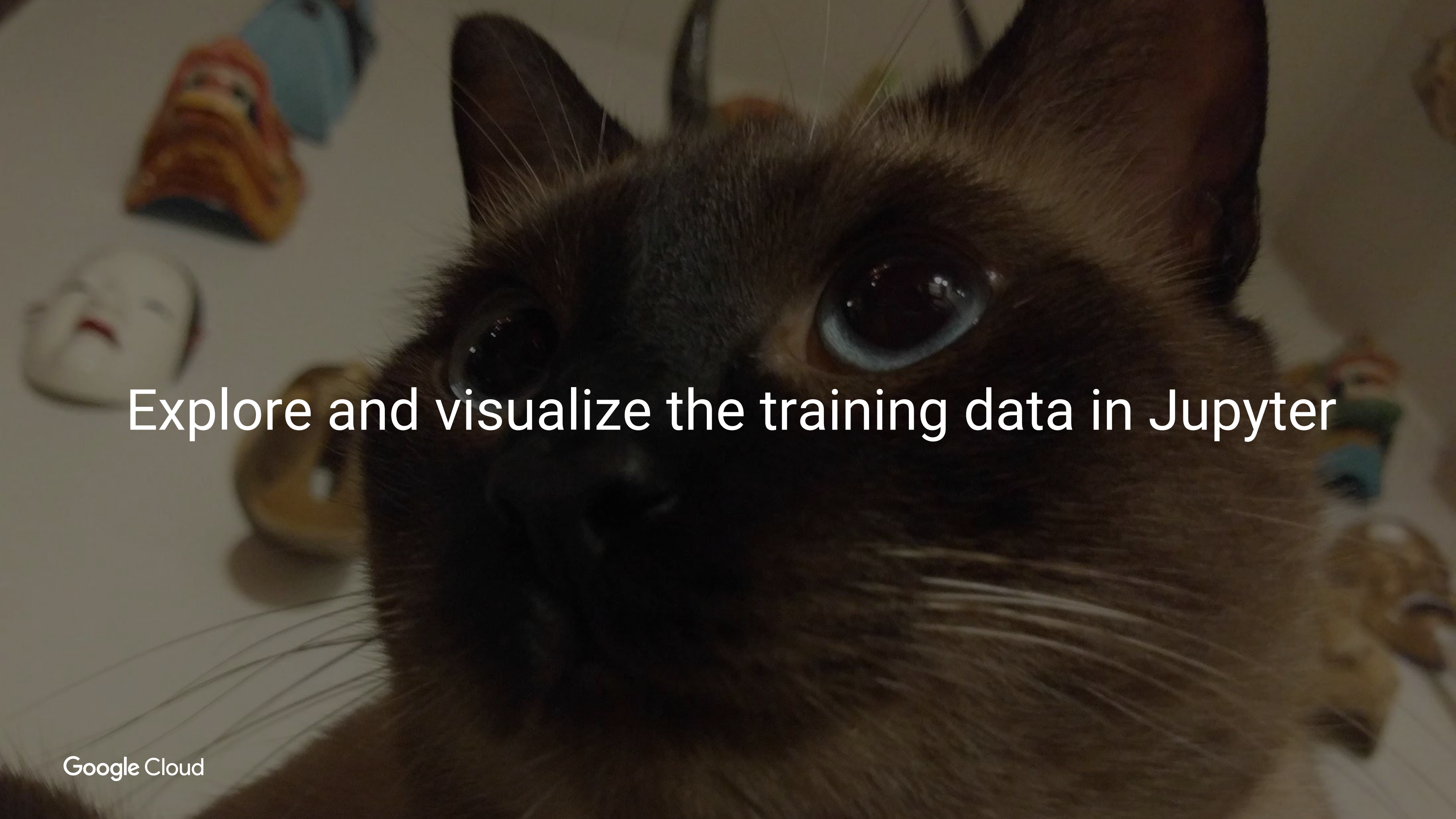
- Overfitting risk
- Uh-oh...

What if it gets fixed later? Uh-oh!

Instance ID	**Label**	**Set**	**Grape**	**Vintage**	**...**
Wine156	yes	train	merlot	0	...
Wine8	yes	train	blend	0	...
Wine786	no	train	cabernet	0	...
Wine912	yes	train	blend	0	...
Wine91	no	train	pinot noir	0	...
...	...		...	...	...

Recipe has ... + 10 * Year + ...

Explore and visualize the training data in Jupyter



Secure | https://ssh.cloud.google.com/projects/super-188716/zones/us-east1-c/instances/super-188716-compute-instance?authus...

```
(env) user super-188716-compute-instance:~$ mkdir -p ~/data/training_small
```

```
mkdir -p ~/data/training_small

gsutil -m cp gs://$BUCKET/catimages/training_images/000*.png ~/data/training_small/

gsutil -m cp gs://$BUCKET/catimages/training_images/001*.png ~/data/training_small/

mkdir -p ~/data/debugging_small

gsutil -m cp gs://$BUCKET/catimages/training_images/002*.png ~/data/debugging_small

echo "done!"
```

Make folders and copy over a small subset of training data over for examination

```
Copying gs://super-188716-bucket/catimages/training_images/002969_038_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002970_038_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002971_038_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002972_038_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002973_038_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002974_038_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002975_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002976_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002977_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002978_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002979_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002980_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002982_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002983_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002985_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002984_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002986_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002987_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002988_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002989_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002990_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002991_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002992_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002993_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002994_039_1.png...
Copying gs://super-188716-bucket/catimages/training_images/002995_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002996_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002997_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002998_039_0.png...
Copying gs://super-188716-bucket/catimages/training_images/002999_039_0.png...
- [977/977 files][ 22.6 MiB/ 22.6 MiB] 100% Done   1.8 MiB/s ETA 00:00:00
Operation completed over 977 objects/22.6 MiB.
(env) user @super-188716-compute-instance:~$ echo "done!"
done!
(env) user @super-188716-compute-instance:~$
```

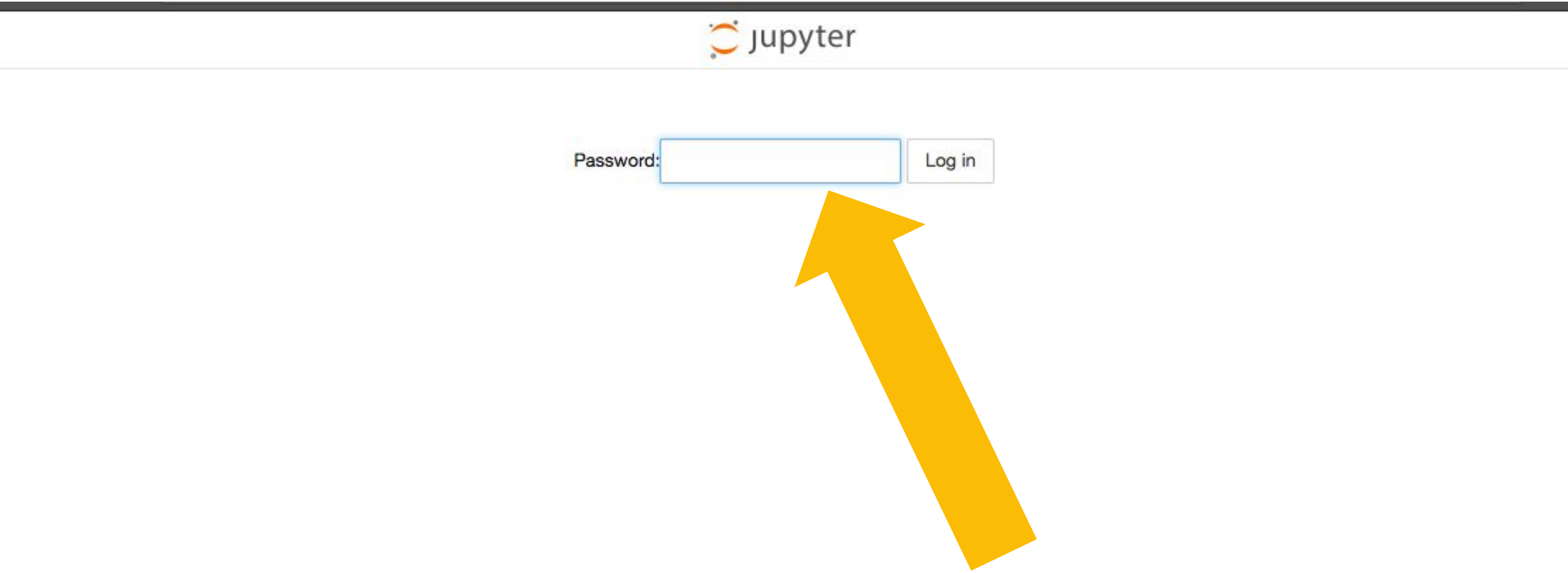
Secure | https://ssh.cloud.google.com/projects/super-188716/zones/us-east1-c/instances/super-188716-compute-instance?authus...

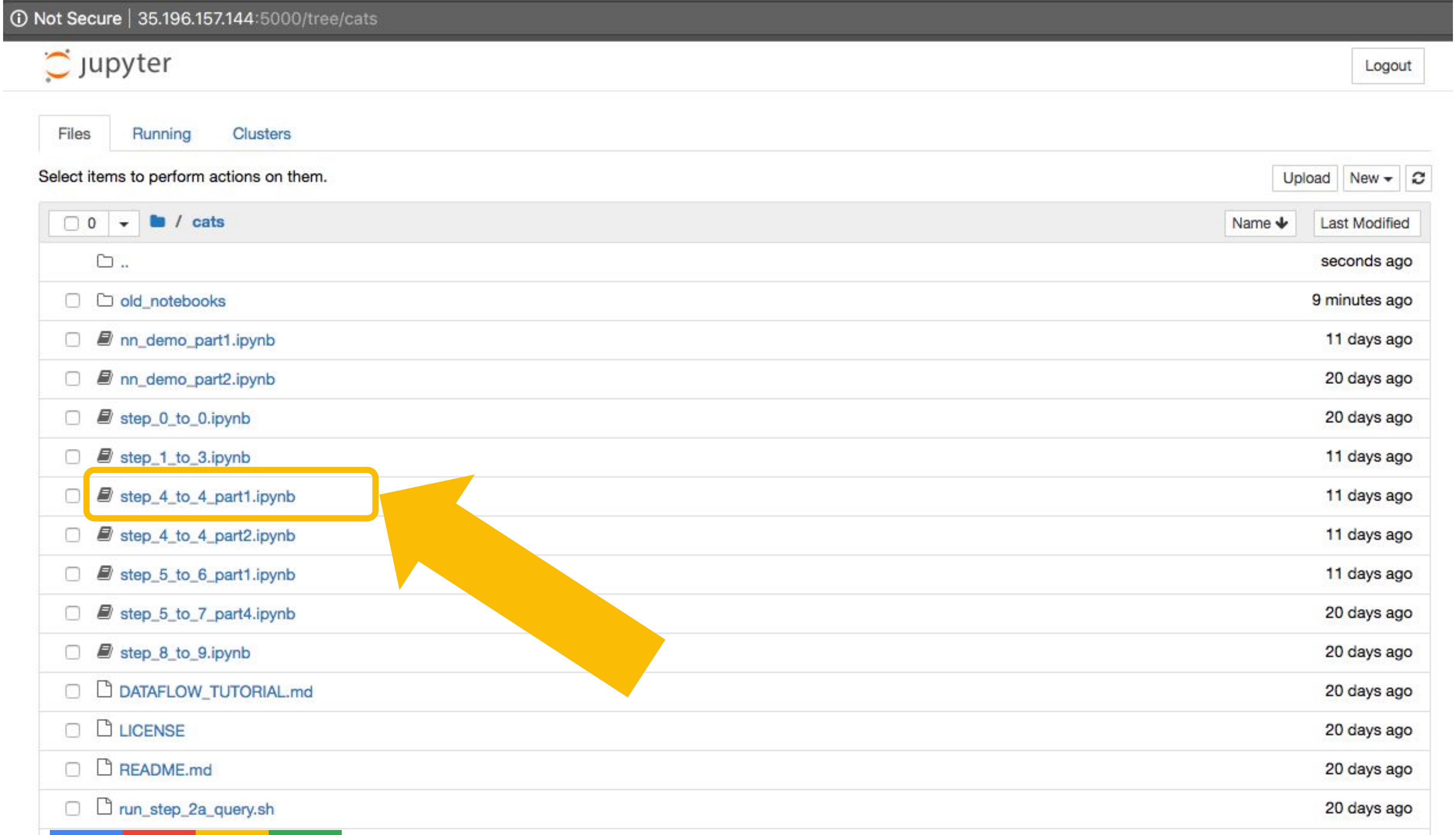
```
(env) user @super-188716-compute-instance:~$ jupyter notebook
```

jupyter notebook

Launch Jupyter notebook

Google Cloud


jupyter
Password:
Log in


Not Secure | 35.196.157.144:5000/tree/cats
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ cats
Name
Last Modified
..
seconds ago
old_notebooks
9 minutes ago
nn_demo_part1.ipynb
11 days ago
nn_demo_part2.ipynb
20 days ago
step_0_to_0.ipynb
20 days ago
step_1_to_3.ipynb
11 days ago
step_4_to_4_part1.ipynb
11 days ago
step_4_to_4_part2.ipynb
11 days ago
step_5_to_6_part1.ipynb
11 days ago
step_5_to_7_part4.ipynb
20 days ago
step_8_to_9.ipynb
20 days ago
DATAFLOW_TUTORIAL.md
20 days ago
LICENSE
20 days ago
README.md
20 days ago
run_step_2a_query.sh
20 days ago

jupyter step_4_to_4_part1 Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 2

Run Code

Exploring the Training Set

Author(s): kozyr@google.com, bfoo@google.com

In this notebook, we gather exploratory data from our training set to do feature engineering and model tuning. Before running this notebook, make sure that:

- You have already run steps 2 and 3 to collect and split your data into training, validation, and test.
- Your training data is in a Google storage folder such as gs://[your-bucket]/[dataprep-dir]/training_images/

In the spirit of learning to walk before learning to run, we'll write this notebook in a more basic style than you'll see in a professional setting.

Setup

TODO for you: In screen terminal 1 (Go to the VM shell and type `Ctrl+a 1`), create a folder to store your training and debugging images, and then copy a small sample of training images from cloud storage:

```
mkdir -p ~/data/training_small
gsutil -m cp gs://$BUCKET/catimages/training_images/000*.png ~/data/training_small/
gsutil -m cp gs://$BUCKET/catimages/training_images/001*.png ~/data/training_small/
mkdir -p ~/data/debugging_small
gsutil -m cp gs://$BUCKET/catimages/training_images/002*.png ~/data/debugging_small
echo "done!"
```

Note that we only take the images starting with those IDs to limit the total number we'll copy over to under 3 thousand images.

In [1]:
```python
# Enter your username:
YOUR_GMAIL_ACCOUNT = '******' # Whatever is before @gmail.com in your email address
```

In [2]:
```python
# Libraries for this section:
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import pandas as pd
import cv2
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Grab the filenames:
TRAINING_DIR = os.path.join('../..', YOUR_GMAIL_ACCOUNT, 'data/training_small/')
files = os.listdir(TRAINING_DIR)  # Grab all the files in the VM images directory
print(files[0:5])  # Let's see some filenames
```

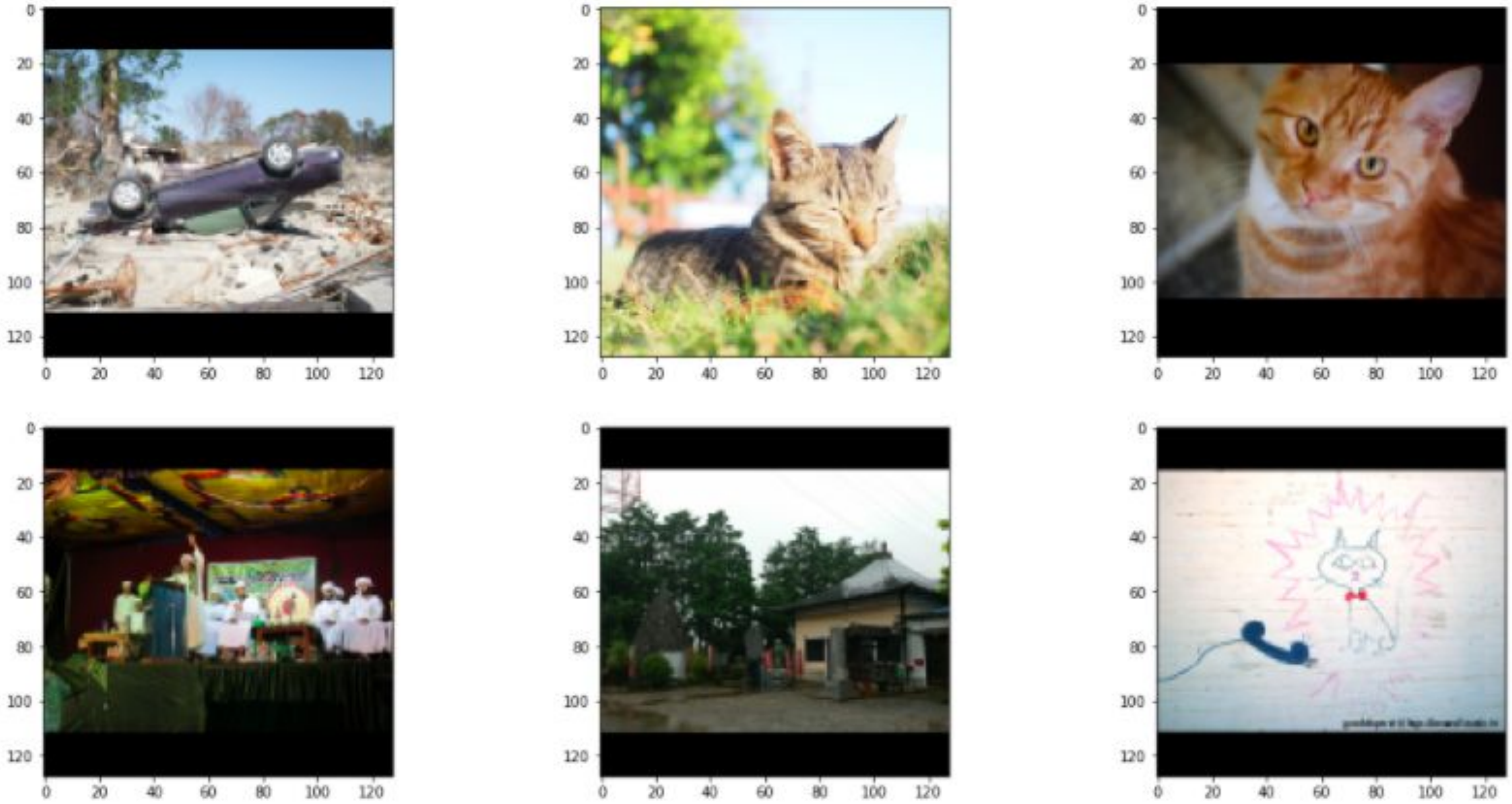
```
['001639_021_0.png', '001297_016_1.png', '001012_013_1.png', '001812_024_0.png', '000320_004_0.png']
```

Eyes on the data!

In [4]:
```
def show_pictures(filelist, dir, img_rows=2, img_cols=3, figsize=(20, 10)):
  """Display the first few images.

  Args:
    filelist: list of filenames to pull from
    dir: directory where the files are stored
    img_rows: number of rows of images to display
    img_cols: number of columns of images to display
    figsize: sizing for inline plots

  Returns:
    None
  """
  plt.close('all')
  fig = plt.figure(figsize=figsize)

  for i in range(img_rows * img_cols):
    a=fig.add_subplot(img_rows, img_cols,i+1)
    img = mpimg.imread(os.path.join(dir, filelist[i]))
    plt.imshow(img)
  plt.show()
```

```python
    a=fig.add_subplot(img_rows, img_cols,i+1)
    img = mpimg.imread(os.path.join(dir, filelist[i]))
    plt.imshow(img)
  plt.show()
```

In [5]:
```python
show_pictures(files, TRAINING_DIR)
```



Check out the colors at rapidtables.com/web/color/RGB_Color, but don't forget to flip order of the channels to BGR.

In [6]:
```
# What does the actual image matrix look like?  There are three channels:
img = cv2.imread(os.path.join(TRAINING_DIR, files[0]))
print('\n***Colors in the middle of the first image***\n')
print('Blue channel:')
print(img[63:67,63:67,0])
print('Green channel:')
print(img[63:67,63:67,1])
print('Red channel:')
print(img[63:67,63:67,2])
```

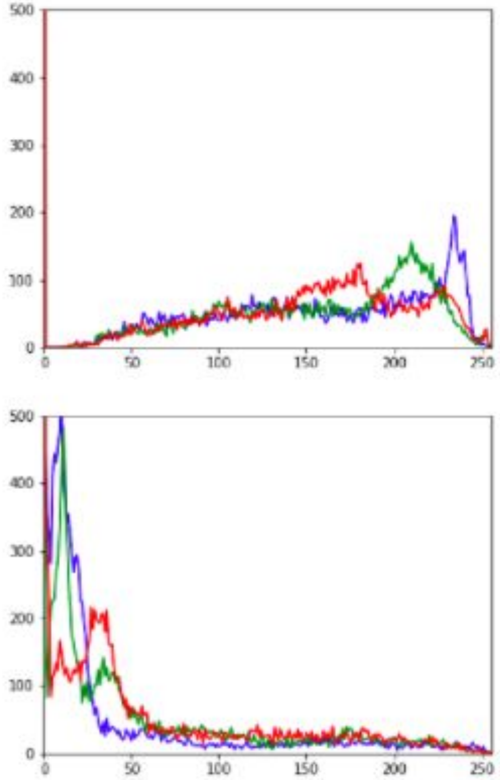
```
***Colors in the middle of the first image***

Blue channel:
[[131 132 135 133]
 [130 131 132 131]
 [132 130 130 124]
 [120 116 107 106]]
Green channel:
[[101 101 105 102]
 [100 100 102 101]
 [103 101 101  96]
 [ 94  90  84  86]]
Red channel:
[[105 106 109 107]
 [103 106 106 103]
 [106 105 103  97]
 [ 94  92  85  88]]
```

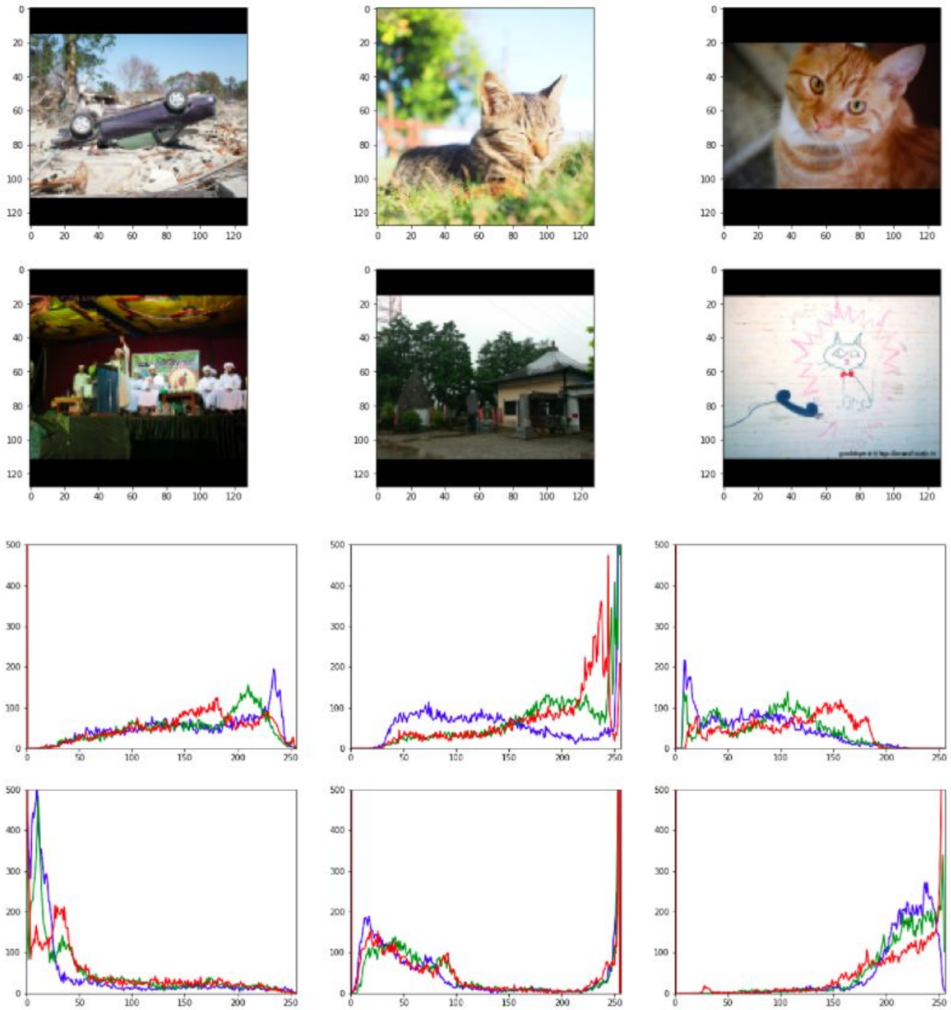
In [7]:
```
def show_bgr(filelist, dir, img_rows=2, img_cols=3, figsize=(20, 10)):
  """Make histograms of the pixel color matrices of first few images.

  Args:
    filelist: list of filenames to pull from
    dir: directory where the files are stored
    img_rows: number of rows of images to display
    img_cols: number of columns of images to display
    figsize: sizing for inline plots

  Returns:
    None
  """
  plt.close('all')
  fig = plt.figure(figsize=figsize)
  color = ('b','g','r')

  for i in range(img_rows * img_cols):
    a=fig.add_subplot(img_rows, img_cols, i + 1)
    img = cv2.imread(os.path.join(TRAINING_DIR, files[i]))
    for c,col in enumerate(color):
      histr = cv2.calcHist([img],[c],None,[256],[0,256])
      plt.plot(histr,color = col)
      plt.xlim([0,256])
      plt.ylim([0,500])
  plt.show()
```

In [8]:
```
show_bgr(files, TRAINING_DIR)
```

```
        histr = cv2.calcHist([img],[c],None,[256],[0,256])
        plt.plot(histr,color = col)
        plt.xlim([0,256])
        plt.ylim([0,500])
    plt.show()
```

In [8]:
```
show_bgr(files, TRAINING_DIR)
```





Do some sanity checks

For example:

- Do we have blank images?
- Do we have images with very few colors?

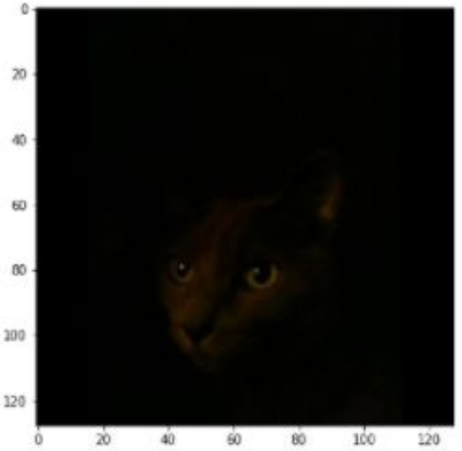
In [9]:
```
# Pull in blue channel for each image, reshape to vector, count unique values:
unique_colors = []
landscape = []
for f in files:
  img = np.array(cv2.imread(os.path.join(TRAINING_DIR, f)))[:,:,0]
  # Determine if landscape is more likely than portrait by comparing
    #amount of zero channel in 3rd row vs 3rd col:
  landscape_likely = (np.count_nonzero(img[:,2]) > np.count_nonzero(img[2,:])) * 1
  # Count number of unique blue values:
  col_count = len(set(img.ravel()))
  # Append to array:
  unique_colors.append(col_count)
  landscape.append(landscape_likely)

unique_colors = pd.DataFrame({'files': files, 'unique_colors': unique_colors,
                              'landscape': landscape})
unique_colors = unique_colors.sort_values(by=['unique_colors'])
print(unique_colors[0:10])
```

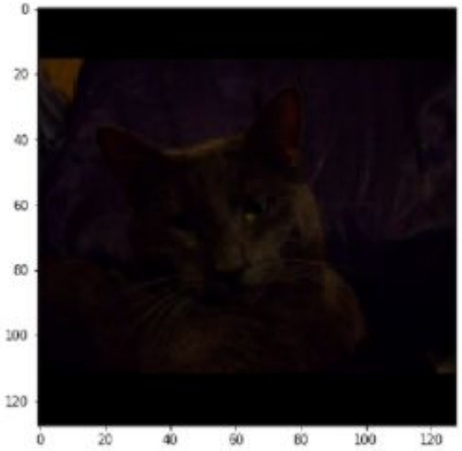
```
                files  landscape  unique_colors
1418  000038_000_1.png          0             25
181   000874_011_1.png          1             33
505   001009_013_0.png          1             33
42    001595_021_0.png          0             33
268   001187_015_1.png          1             58
```

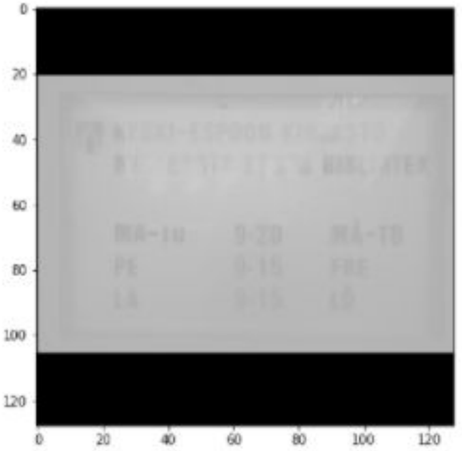
```
unique_colors = unique_colors.sort_values(by=['unique_colors'])
print(unique_colors[0:10])
```

```
              files  landscape  unique_colors
1418  000038_000_1.png          0             25
181   000874_011_1.png          1             33
505   001009_013_0.png          1             33
42    001595_021_0.png          0             33
268   001187_015_1.png          1             58
1616  000019_000_0.png          1             78
478   001902_025_0.png          0             84
366   001738_023_1.png          1             84
542   001084_014_0.png          1             85
146   000256_003_0.png          0             85
```

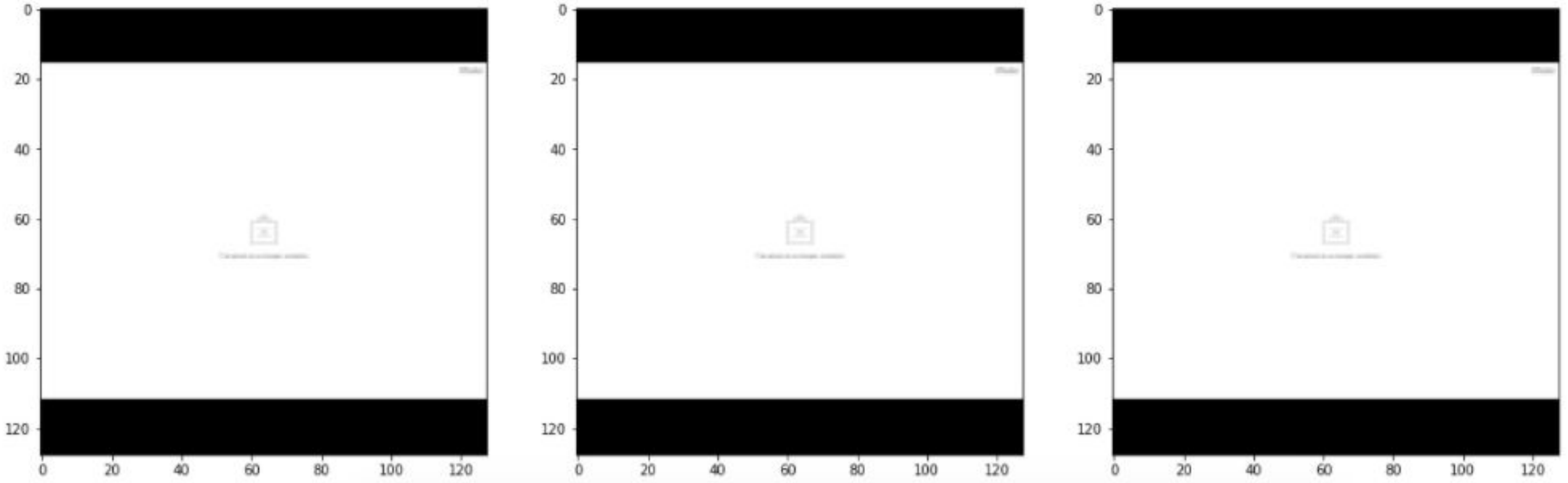
In [10]:
```
# Plot the pictures with the lowest diversity of unique color values:
suspicious = unique_colors['files'].tolist()
show_pictures(suspicious, TRAINING_DIR, 1)
```

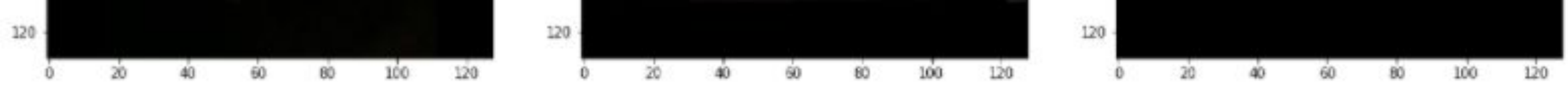






In [9]:
```
# Plot the pictures with the lowest diversity of unique color values:
suspicious = unique_colors['files'].tolist()
show_pictures(suspicious, TRAINING_DIR, 1)
```





Get labels

Extract labels from the filename and create a pretty dataframe for analysis.

In [11]:
```
def get_label(str):
  """
  Split out the label from the filename of the image, where we stored it.
  Args:
    str: filename string.
  Returns:
    label: an integer 1 or 0
  """
  split_filename = str.split('_')
  label = int(split_filename[-1].split('.')[0])
  return(label)

# Example:
get_label('12550_0.1574_1.png')
```

Out[11]: 1

Create DataFrame

In [12]:
```
df = unique_colors[:]
df['label'] = df['files'].apply(lambda x: get_label(x))
df['landscape_likely'] = df['landscape']
df = df.drop(['landscape', 'unique_colors'], axis=1)
df[:10]
```

Out[12]:

	files	label	landscape_likely
1418	000038_000_1.png	1	0
181	000874_011_1.png	1	1
505	001009_013_0.png	0	1
42	001595_021_0.png	0	0
268	001187_015_1.png	1	1
1616	000019_000_0.png	0	1
478	001902_025_0.png	0	0
366	001738_023_1.png	1	1
542	001084_014_0.png	0	1
146	000256_003_0.png	0	0

Basic Feature Engineering

Below, we show an example of a very simple set of features that can be derived from an image. This function simply pulls the mean, standard deviation, min, and max of pixel values in one image band (red, green, or blue)

In [14]:
```
def general_img_features(band):
    """
    Define a set of features that we can look at for each color band
    Args:
      band: array which is one of blue, green, or red
    Returns:
      features: unique colors, nonzero count, mean, standard deviation,
                min, and max of the channel's pixel values
    """
    return [len(set(band.ravel())), np.count_nonzero(band),
            np.mean(band), np.std(band),
            band.min(), band.max()]

def concat_all_band_features(file, dir):
    """
    Extract features from a single image.
     Args:
           file - single image filename
           dir - directory where the files are stored
    Returns:
           features - descriptive statistics for pixels
    """
    img = cv2.imread(os.path.join(dir, file))
    features = []
    blue = np.float32(img[:,:,0])
    green = np.float32(img[:,:,1])
```

In [14]:
```
def general_img_features(band):
  """
  Define a set of features that we can look at for each color band
  Args:
    band: array which is one of blue, green, or red
  Returns:
    features: unique colors, nonzero count, mean, standard deviation,
              min, and max of the channel's pixel values
  """
  return [len(set(band.ravel())), np.count_nonzero(band),
          np.mean(band), np.std(band),
          band.min(), band.max()]

def concat_all_band_features(file, dir):
  """
  Extract features from a single image.
   Args:
         file - single image filename
         dir - directory where the files are stored
  Returns:
         features - descriptive statistics for pixels
  """
  img = cv2.imread(os.path.join(dir, file))
  features = []
  blue = np.float32(img[:,:,0])
  green = np.float32(img[:,:,1])
  red = np.float32(img[:,:,2])
  features.extend(general_img_features(blue)) # indices 0-4
  features.extend(general_img_features(green)) # indices 5-9
  features.extend(general_img_features(red)) # indices 10-14
  return features
```

In [15]:
```
# Let's see an example:
print(files[0] + '\n')
example = concat_all_band_features(files[0], TRAINING_DIR)
print(example)
```

```
001639_021_0.png

[245, 12288, 119.79291, 87.197845, 0.0, 253.0, 243, 12288, 119.24084, 84.254837, 0.0, 254.0,
243, 12288, 116.98547, 82.173241, 0.0, 253.0]
```

In [16]:
```
# Apply it to our dataframe:
feature_names = ['blue_unique', 'blue_nonzero', 'blue_mean', 'blue_sd', 'blue_min', 'blue_max',
                 'green_unique', 'green_nonzero', 'green_mean', 'green_sd', 'green_min', 'green_
                 'red_unique', 'red_nonzero', 'red_mean', 'red_sd', 'red_min', 'red_max']

# Compute a series holding all band features as lists
band_features_series = df['files'].apply(lambda x: concat_all_band_features(x, TRAINING_DIR))

# Loop through lists and distribute them across new columns in the dataframe
for i in range(len(feature_names)):
  df[feature_names[i]] = band_features_series.apply(lambda x: x[i])
df[:10]
```

Out[16]:

	files	label	landscape_likely	blue_unique	blue_nonzero	blue_mean	blue_sd	blue_min	blue_max	green_
1418	000038_000_1.png	1	0	25	5644	0.553894	1.359564	0.0	74.0	
181	000874_011_1.png	1	1	33	12288	7.442017	6.383089	0.0	33.0	
505	001009_013_0.png	0	1	33	10880	123.887939	88.271904	0.0	204.0	
42	001595_021_0.png	0	0	33	6126	0.719482	1.800417	0.0	53.0	

```
In [17]: # Are these features good for finding cats?
         # Let's look at some basic correlations.
         df.corr().round(2)
```

Out[17]:

	label	landscape_likely	blue_unique	blue_nonzero	blue_mean	blue_sd	blue_min	blue_max	green_unique	g
label	1.00	0.07	-0.01	0.10	-0.10	-0.11	0.02	-0.08	0.02	
landscape_likely	0.07	1.00	0.03	-0.31	-0.14	0.04	-0.25	0.00	-0.01	
blue_unique	-0.01	0.03	1.00	0.02	0.21	0.52	-0.13	0.87	0.87	
blue_nonzero	0.10	-0.31	0.02	1.00	0.39	-0.08	0.40	0.02	0.05	
blue_mean	-0.10	-0.14	0.21	0.39	1.00	0.70	0.19	0.27	0.10	
blue_sd	-0.11	0.04	0.52	-0.08	0.70	1.00	-0.12	0.50	0.37	
blue_min	0.02	-0.25	-0.13	0.40	0.19	-0.12	1.00	-0.03	-0.04	
blue_max	-0.08	0.00	0.87	0.02	0.27	0.50	-0.03	1.00	0.76	
green_unique	0.02	-0.01	0.87	0.05	0.10	0.37	-0.04	0.76	1.00	
green_nonzero	0.10	-0.32	0.01	0.98	0.38	-0.09	0.39	0.02	0.04	
green_mean	-0.08	-0.16	0.15	0.43	0.91	0.59	0.19	0.21	0.10	
green_sd	-0.07	0.03	0.41	-0.10	0.62	0.89	-0.08	0.41	0.40	
green_min	0.01	-0.24	-0.07	0.39	0.19	-0.11	0.78	-0.00	-0.07	
green_max	-0.08	-0.02	0.76	0.02	0.22	0.41	-0.01	0.88	0.84	
red_unique	0.01	-0.02	0.70	0.04	-0.01	0.22	-0.03	0.60	0.88	
red_nonzero	0.10	-0.33	0.01	0.99	0.37	-0.09	0.40	0.01	0.04	
red_mean	-0.01	-0.18	0.09	0.46	0.78	0.45	0.20	0.14	0.11	

blue_sd	-0.11	0.04	0.52	-0.08	0.70	1.00	-0.12	0.50	0.37
blue_min	0.02	-0.25	-0.13	0.40	0.19	-0.12	1.00	-0.03	-0.04
blue_max	-0.08	0.00	0.87	0.02	0.27	0.50	-0.03	1.00	0.76
green_unique	0.02	-0.01	0.87	0.05	0.10	0.37	-0.04	0.76	1.00
green_nonzero	0.10	-0.32	0.01	0.98	0.38	-0.09	0.39	0.02	0.04
green_mean	-0.08	-0.16	0.15	0.43	0.91	0.59	0.19	0.21	0.10
green_sd	-0.07	0.03	0.41	-0.10	0.62	0.89	-0.08	0.41	0.40
green_min	0.01	-0.24	-0.07	0.39	0.19	-0.11	0.78	-0.00	-0.07
green_max	-0.08	-0.02	0.76	0.02	0.22	0.41	-0.01	0.88	0.84
red_unique	0.01	-0.02	0.70	0.04	-0.01	0.22	-0.03	0.60	0.88
red_nonzero	0.10	-0.33	0.01	0.99	0.37	-0.09	0.40	0.01	0.04
red_mean	-0.01	-0.18	0.09	0.46	0.78	0.45	0.20	0.14	0.11
red_sd	-0.01	0.04	0.31	-0.14	0.43	0.68	-0.10	0.31	0.36
red_min	0.03	-0.27	-0.08	0.44	0.19	-0.09	0.66	-0.04	-0.03
red_max	-0.08	-0.05	0.60	0.03	0.15	0.30	0.01	0.69	0.75

These coarse features look pretty bad individually. Most of this is due to features capturing absolute pixel values. But photo lighting could vary significantly between different image shots. What we end up with is a lot of noise.

Are there some better feature detectors we can consider? Why yes, there are! Several common features involve finding corners in pictures, and looking for pixel gradients (differences in pixel values between neighboring pixels in different directions).

Harris Corner Detector

The following snippet runs code to visualize harris corner detection for a few sample images. Configuring the threshold determines how strong of a signal we need to determine if a pixel corresponds to a corner (high pixel gradients in all directions).

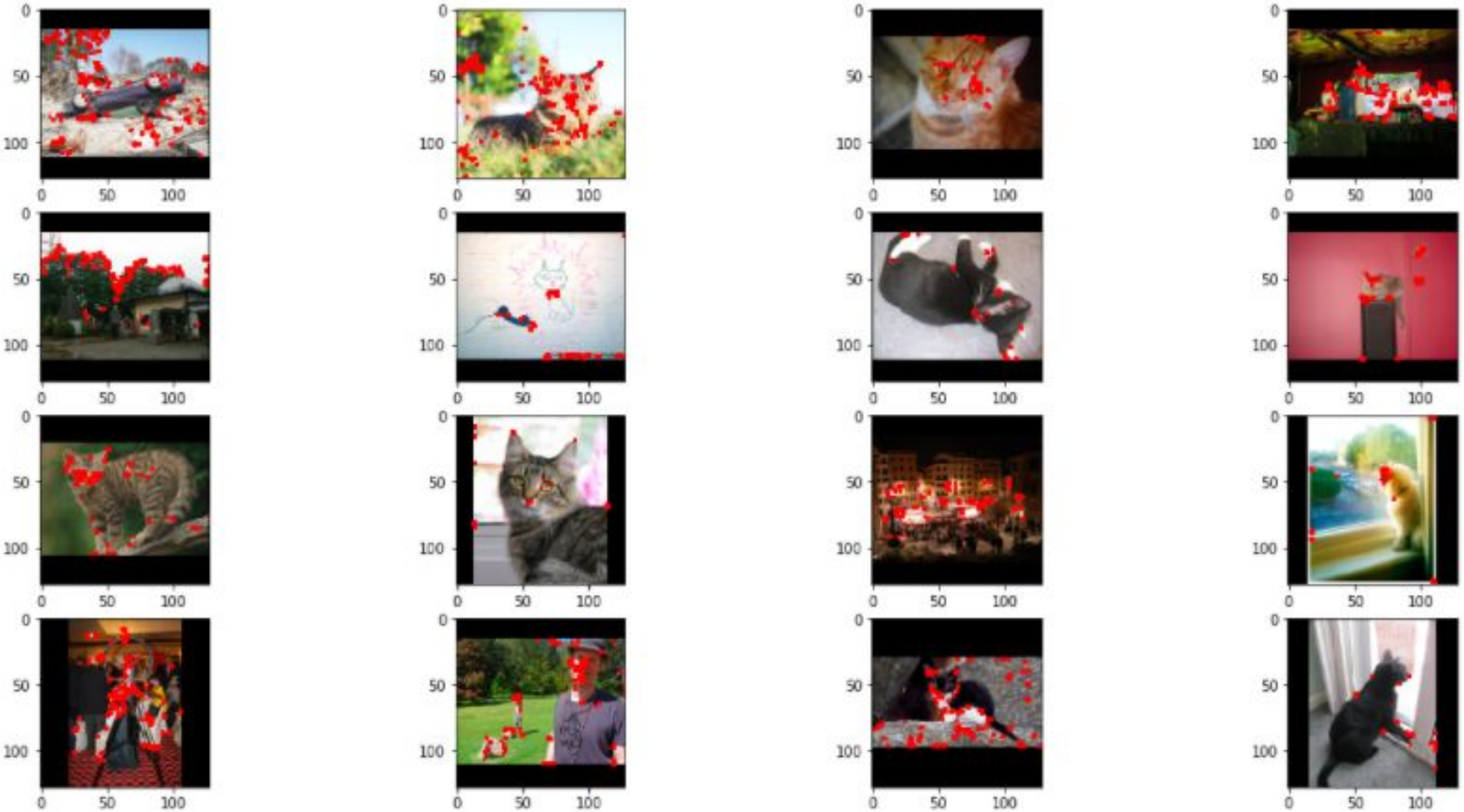
Note that because a Harris corner detector returns another image map with values corresponding to the likelihood of a corner at that pixel, it can also be fed into general_img_features() to extract additional features. What do you notice about corners on cat images?

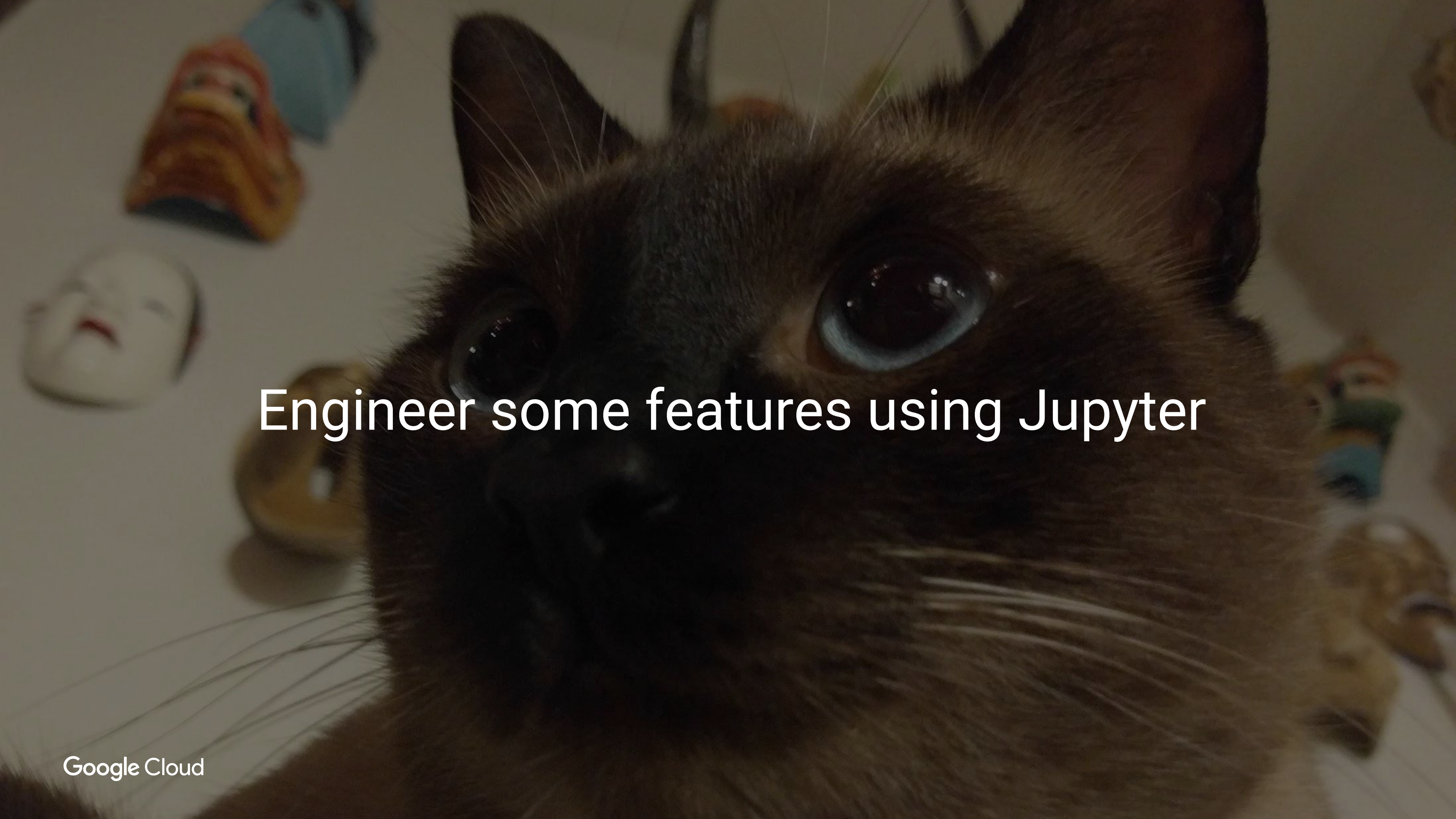
In [18]:

```
THRESHOLD = 0.05

def show_harris(filelist, dir, band=0, img_rows=4, img_cols=4, figsize=(20, 10)):
  """
  Display Harris corner detection for the first few images.
  Args:
    filelist: list of filenames to pull from
    dir: directory where the files are stored
    band: 0 = 'blue', 1 = 'green', 2 = 'red'
    img_rows: number of rows of images to display
    img_cols: number of columns of images to display
    figsize: sizing for inline plots
  Returns:
    None
  """
  plt.close('all')
  fig = plt.figure(figsize=figsize)

  def plot_bands(src, band_img):
    a=fig.add_subplot(img_rows, img_cols, i + 1)
    dst = cv2.cornerHarris(band_img, 2, 3, 0.04)
```

```python
def show_harris(filelist, dir, band=0, img_rows=4, img_cols=4, figsize=(20, 10)):
  """
  Display Harris corner detection for the first few images.
  Args:
    filelist: list of filenames to pull from
    dir: directory where the files are stored
    band: 0 = 'blue', 1 = 'green', 2 = 'red'
    img_rows: number of rows of images to display
    img_cols: number of columns of images to display
    figsize: sizing for inline plots
  Returns:
    None
  """
  plt.close('all')
  fig = plt.figure(figsize=figsize)

  def plot_bands(src, band_img):
    a=fig.add_subplot(img_rows, img_cols, i + 1)
    dst = cv2.cornerHarris(band_img, 2, 3, 0.04)
    dst = cv2.dilate(dst,None) # dilation makes the marks a little bigger

    # Threshold for an optimal value, it may vary depending on the image.
    new_img = src.copy()
    new_img[dst > THRESHOLD * dst.max()]=[0, 0, 255]
    # Note: openCV reverses the red-green-blue channels compared to matplotlib,
    # so we have to flip the image before showing it
    imgplot = plt.imshow(cv2.cvtColor(new_img, cv2.COLOR_BGR2RGB))

  for i in range(img_rows * img_cols):
    img = cv2.imread(os.path.join(dir, filelist[i]))
    plot_bands(img, img[:,:,band])

  plt.show()
```

```
In [19]: show_harris(files, TRAINING_DIR)
```





Engineer some features using Jupyter

Google Cloud

jupyter step_4_to_4_part2 Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 2

Run Code

Feature Engineering

Author(s): bfoo@google.com, kozyr@google.com

In this notebook, we gather exploratory data from our training set to do feature engineering and model tuning. Before running this notebook, make sure that:

- You have already run steps 2 and 3 to collect and split your data into training, validation, and test.
- Your entire training dataset is in a Cloud Storage Bucket such as gs://[your-bucket]/[dataprep-dir]/training_images/
- You have a small subset of the training data available on your VM already (from the exploration we did in the previous notebook):

```
mkdir -p ~/data/training_small
gsutil -m cp gs://$BUCKET/catimages/training_images/000*.png ~/data/training_small/
gsutil -m cp gs://$BUCKET/catimages/training_images/001*.png ~/data/training_small/
mkdir -p ~/data/debugging_small
gsutil -m cp gs://$BUCKET/catimages/training_images/002*.png ~/data/debugging_small
echo "done!"
```

Note that we only take the images starting with those IDs to limit the number we'll copy over to only a few thousand images.

Setup

In [1]:
```
# Enter your username:
YOUR_GMAIL_ACCOUNT = '******' # Whatever is before @gmail.com in your email address
```

In [2]:
```
# Libraries for this section:
import os
import cv2
import pickle
import numpy as np
from sklearn import preprocessing
```

In [3]:
```
# Directories:
PREPROC_DIR = os.path.join('../..', YOUR_GMAIL_ACCOUNT, 'data/')
TRAIN_DIR = os.path.join('../..', YOUR_GMAIL_ACCOUNT, 'data/training_small/')  # Where the training dataset lives.
DEBUG_DIR = os.path.join('../..', YOUR_GMAIL_ACCOUNT, 'data/debugging_small/')  # Where the debugging dataset lives.
```

Feature Engineering Functions

Basic features and concatenation

In [4]:
```
def general_img_features(band):
    """
    Define a set of features that we can look at for each color band
    Args:
      band: array which is one of blue, green, or red
    Returns:
      features: unique colors, nonzero count, mean, standard deviation,
                min, and max of the channel's pixel values
    """
    return [len(set(band.ravel())), np.count_nonzero(band),
            np.mean(band), np.std(band),
            band.min(), band.max()]

def concat_all_band_features(file, dir):
    """
    Extract features from a single image.
     Args:
          file - single image filename
          dir - directory where the files are stored
    Returns:
          features - descriptive statistics for pixels
    """
    img = cv2.imread(os.path.join(dir, file))
    features = []
    blue = np.float32(img[:,:,0])
    green = np.float32(img[:,:,1])
```

```
In [4]: def general_img_features(band):
          """
          Define a set of features that we can look at for each color band
          Args:
            band: array which is one of blue, green, or red
          Returns:
            features: unique colors, nonzero count, mean, standard deviation,
                      min, and max of the channel's pixel values
          """
          return [len(set(band.ravel())), np.count_nonzero(band),
                  np.mean(band), np.std(band),
                  band.min(), band.max()]

        def concat_all_band_features(file, dir):
          """
          Extract features from a single image.
           Args:
                 file - single image filename
                 dir - directory where the files are stored
          Returns:
                 features - descriptive statistics for pixels
          """
          img = cv2.imread(os.path.join(dir, file))
          features = []
          blue = np.float32(img[:,:,0])
          green = np.float32(img[:,:,1])
          red = np.float32(img[:,:,2])
          features.extend(general_img_features(blue)) # indices 0-4
          features.extend(general_img_features(green)) # indices 5-9
          features.extend(general_img_features(red)) # indices 10-14
          return features
```

Harris Corner Detector Histograms

We'll create features based on the histogram of the number of corners detected in every small square in the picture. The threshold indicates how "sharp" that corner must be to be detected.

In [5]:
```
def harris_density(harris_img, square_size, threshold):
  """Apply Harris Corner Detection to image and get count of corners.

  Args:
    harris_img: image already processed by Harris Corner Detector (in cv2 package).
    square_size: number of pixels per side of the window in which we detect corners.
    threshold: indicates how "sharp" that corner must be to be detected.

  Returns:
    bins - counts in each bin of histogram.
  """
  max_val = harris_img.max()
  shape = harris_img.shape
  bins = [0] * (square_size * square_size + 1)
  for row in xrange(0, shape[0], square_size):
      for col in xrange(0, shape[1], square_size):
          bin_val = sum(sum(harris_img[row: row + square_size,
                                       col: col + square_size] > threshold * max_val))
          bins[int(bin_val)] += 1
  return bins
```

Building Feature Vectors

We've defined some functions and checked their outputs. Here is a sample feature vector constructor from pulling out summary features from grayscale, red, green, and blue channels along with harris corner detector output thresholding.

In [6]:
```
def get_features(img_path):
  """Engineer the features and output feature vectors.

  Args:
    img_path: filepath to image file

  Returns:
    features: np array of features
  """
  img = cv2.imread(img_path)
  # Get the channels
  gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
  blue = np.float32(img[:, :, 0])
  green = np.float32(img[:, :, 1])
  red = np.float32(img[:, :, 2])

  # Run general summarization on each
  features = general_img_features(gray)
  features.extend(general_img_features(blue))
  features.extend(general_img_features(green))
  features.extend(general_img_features(red))
```

```python
# Run general summarization on each
features = general_img_features(gray)
features.extend(general_img_features(blue))
features.extend(general_img_features(green))
features.extend(general_img_features(red))

# Get Harris corner detection output
gray = cv2.cornerHarris(gray, 2, 3, 0.04)
blue = cv2.cornerHarris(blue, 2, 3, 0.04)
green = cv2.cornerHarris(green, 2, 3, 0.04)
red = cv2.cornerHarris(red, 2, 3, 0.04)

# Get general stats on each Harris detector results
features.extend(general_img_features(gray))
features.extend(general_img_features(blue))
features.extend(general_img_features(green))
features.extend(general_img_features(red))

# Get density bins on Harris detector results
features.extend(harris_density(gray, 4, 0.05))

return features
```

In [7]:

```
def get_features_and_labels(dir):
  """Get preprocessed features and labels.

  Args:
    dir: directory containing image files

  Returns:
    features: np array of features
    labels: 1-d np array of binary labels
  """
  i = 0
  features = None
  labels = []
  print('\nImages processed (out of {:d})...'.format(len(os.listdir(dir))))
  for filename in os.listdir(dir):
    feature_row = np.array([get_features(os.path.join(dir, filename))])
    if features is not None:
      features = np.append(features, feature_row, axis=0)
    else:
      features = feature_row
    split_filename = filename.split('_')
    label = int(split_filename[-1].split('.')[0])
    labels = np.append(labels, label)
    i += 1
    if i % 100 == 0:
      print(features.shape[0])
  print(features.shape[0])
  return features, labels
```

In [8]:
```
# Use a limited set of images, this is computationally expensive:
training_features, training_labels = get_features_and_labels(TRAIN_DIR)
debugging_features, debugging_labels = get_features_and_labels(DEBUG_DIR)

print('\nDone!')
```

```
Images processed (out of 1960)...
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
1960
```

Standardize and save

If we don't want the magnitude of a feature column to have an undue influence on the results, we should standardize our features. **Standardization** is a process where the mean is subtracted from feature values, and the result is divided by the standard deviation.

In [9]:
```
# Standardize features:
standardizer = preprocessing.StandardScaler().fit(training_features)
training_std = standardizer.transform(training_features)
debugging_std = standardizer.transform(debugging_features)

# Save features as pkl:
pickle.dump(training_std, open(os.path.join(PREPROC_DIR, 'training_std.pkl'), 'w'))
pickle.dump(debugging_std, open(os.path.join(PREPROC_DIR, 'debugging_std.pkl'), 'w'))
pickle.dump(training_labels, open(os.path.join(PREPROC_DIR, 'training_labels.pkl'), 'w'))
pickle.dump(debugging_labels, open(os.path.join(PREPROC_DIR, 'debugging_labels.pkl'), 'w'))

print ('\nFeaturing engineering is complete!')
```

```
Featuring engineering is complete!
```

Key message

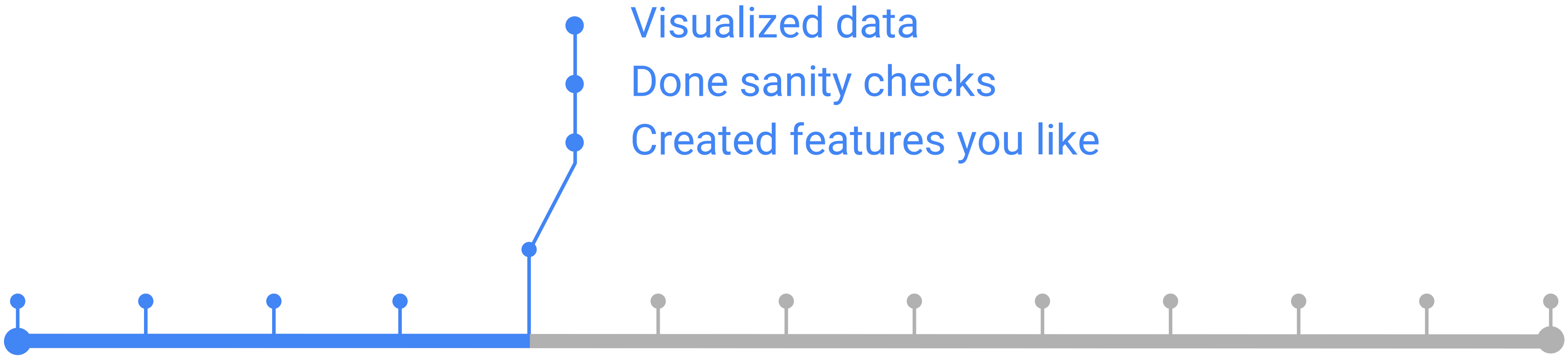
!

Your ML system is only as good as the data that went into it

Getting data involves lots of engineering effort

Getting the right data is an art that involves domain knowledge, analytics, and data exploration

Step 4 is finished | You’ve used the training instances and:



Chief actors:

Analyst

Domain expert

Google Cloud

Step 4 is finished | You've used the training instances and:

Plots

No blank images, enough color variety, etc.

Data preprocessing

Chief actors:

Analyst

Domain expert