Udemy - 2 - Exploratory Data Analysis

https://www.udemy.com/course/aws-machine-learning/learn/lecture/16397700#overview

Python in Data Science and Machine Learning

Python is becoming ubiquitous

- But the test will not test your Python knowledge.

```
In [1]: %matplotlib inline
        import numpy as np
        import pandas as pd

        df = pd.read_csv("PastHires.csv")
        df.head()
```

Out[1]:

	Years Experience	Employed?	Previous employers	Level of Education	Top-tier school	Interned	Hired
0	10	Y	4	BS	N	N	Y
1	0	N	0	BS	Y	Y	Y
2	7	N	6	BS	N	N	N
3	2	Y	1	MS	Y	N	Y
4	20	N	2	PhD	Y	N	N

Pandas

- A Python library for slicing and dicing your data
  - Data Frames
  - Series
  - Interoperates with numpy

Pandas to extract first 5 rows and 2 columns:

```
In [12]: df[['Years Experience', 'Hired']][:5]
```

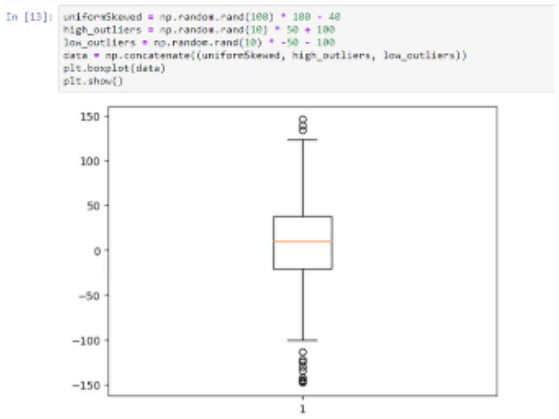
Out[12]:

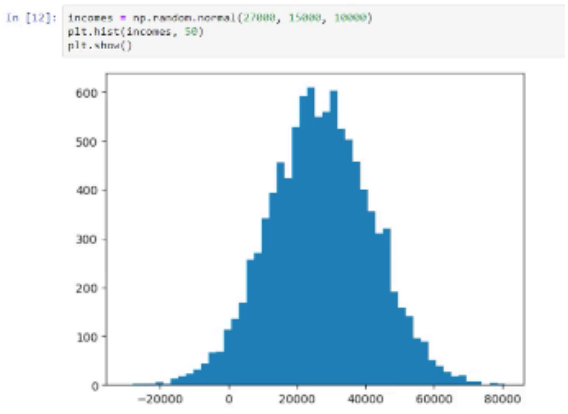
	Years Experience	Hired
0	10	Y
1	0	Y
2	7	N
3	2	Y
4	20	N

Sum up how many unique values within a column exist: value_counts()

```
In [14]: degree_counts = df['Level of Education'].value_counts()
         degree_counts

Out[14]: BS     7
         PhD    4
         MS     2
         Name: Level of Education, dtype: int64
```

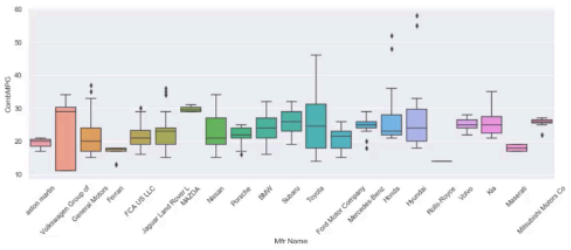
Matplotlib

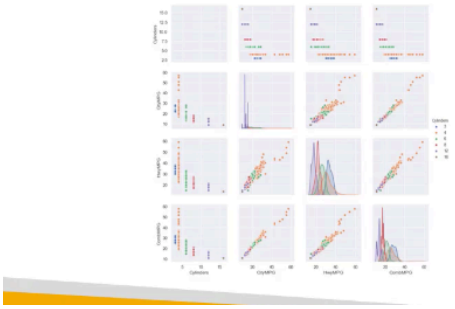


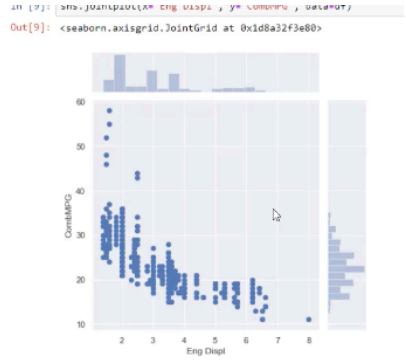


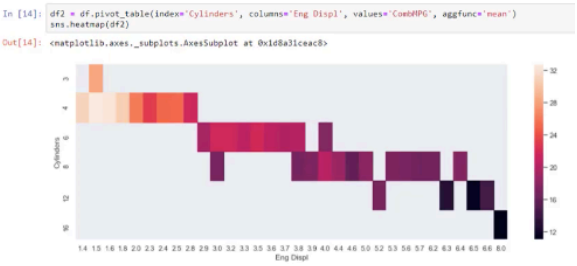
Seaborn is Matplotlib on steroids

Seaborn









scikit_learn

- Python library for machine learning models

Ensemble learning: using a random forest

We'll use a random forest of 10 decision trees to predict employment of specific candidate profiles:

In [7]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=10)
clf = clf.fit(X, y)

#Predict employment of an employed 10-year veteran
print (clf.predict([[10, 1, 4, 0, 0, 0]]))
#...and an unemployed 10-year veteran
print (clf.predict([[10, 0, 4, 0, 0, 0]]))
```

```
[1]
[0]
```

Jupiter Notebook example

Let's begin: prepare your data

Start by importing the mammographic_masses.data.txt file into a Pandas dataframe (hint: use read_csv) and take a look at it.

In [1]:
```
import pandas as pd

masses_data = pd.read_csv('mammographic_masses.data.txt')
masses_data.head()
```

Out[1]:

	5	67	3	5.1	3.1	1
0	4	43	1	1	?	1
1	5	58	4	5	3	1
2	4	28	1	1	3	0
3	5	74	1	5	?	1
4	4	65	1	?	3	0

Make sure you use the optional parmaters in read_csv to convert missing data (indicated by a ?) into NaN, and to add the appropriate column names (BI_RADS, age, shape, margin, density, and severity):

In [2]:
```
masses_data = pd.read_csv('mammographic_masses.data.txt', na_values=['?'], names = ['BI-RADS', 'age', 'shape', 'margin', 'densit
masses_data.head()
```

Out[2]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
1	4.0	43.0	1.0	1.0	NaN	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
4	5.0	74.0	1.0	5.0	NaN	1

=> we replaced the columns and the “?” Values

Evaluate whether the data needs cleaning; your model is only as good as the data it's given. Hint: use describe() on the dataframe.

In [12]:
```
masses_data.describe()
```

Out[12]:

	BI-RADS	age	shape	margin	density	severity
count	959.000000	956.000000	930.000000	913.000000	885.000000	961.000000
mean	4.348279	55.487448	2.721505	2.796276	2.910734	0.463059
std	1.783031	14.480131	1.242792	1.566546	0.380444	0.498893
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	45.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000
75%	5.000000	66.000000	4.000000	4.000000	3.000000	1.000000
max	55.000000	96.000000	4.000000	5.000000	4.000000	1.000000

Extracting rows that contain a null value in one of the columns

There are quite a few missing values in the data set. Before we just drop every row that's missing data, let's make sure we don't bias our data in doing so. Does there appear to be any sort of correlation to what sort of data has missing fields? If there were, we'd have to try and go back and fill that data in.

In [4]:
```
masses_data.loc[(masses_data['age'].isnull()) |
                (masses_data['shape'].isnull()) |
                (masses_data['margin'].isnull()) |
                (masses_data['density'].isnull())]
```

Out[4]:

	BI-RADS	age	shape	margin	density	severity
1	4.0	43.0	1.0	1.0	NaN	1
4	5.0	74.0	1.0	5.0	NaN	1
5	4.0	65.0	1.0	NaN	3.0	0
6	4.0	70.0	NaN	NaN	3.0	0
7	5.0	42.0	1.0	NaN	3.0	0
9	5.0	60.0	NaN	5.0	1.0	1
12	4.0	64.0	1.0	NaN	3.0	0
19	4.0	40.0	1.0	NaN	NaN	0

=> don't really see any obvious patter. Missing data seems evenly distributed.
We can drop them with **dropna()**

778	4.0	60.0	NaN	4.0	3.0	0
819	4.0	35.0	3.0	NaN	2.0	0
824	6.0	40.0	NaN	3.0	4.0	1
884	5.0	NaN	4.0	4.0	3.0	1
923	5.0	NaN	4.0	3.0	3.0	1

130 rows × 6 columns

If the missing data seems randomly distributed, go ahead and drop rows with missing data. Hint: use dropna().

In [14]:
```
masses_data.dropna(inplace=True)
masses_data.describe()
```

Out[14]:

	BI-RADS	age	shape	margin	density	severity
count	830.000000	830.000000	830.000000	830.000000	830.000000	830.000000
mean	4.393976	55.781928	2.781928	2.813253	2.915663	0.489542
std	1.888371	14.671782	1.242361	1.567175	0.350936	0.500092
min	0.000000	18.000000	1.000000	1.000000	1.000000	0.000000
25%	4.000000	46.000000	2.000000	1.000000	3.000000	0.000000
50%	4.000000	57.000000	3.000000	3.000000	3.000000	0.000000
75%	5.000000	66.000000	4.000000	4.000000	3.000000	1.000000
max	55.000000	96.000000	4.000000	5.000000	4.000000	1.000000

Next you'll need to convert the Pandas dataframes into numpy arrays that can be used by scikit_learn. Create an array that extracts only the feature data we want to work with (age, shape, margin, and density) and another array that contains the classes (severity). You'll also need an array of the feature name labels.

In [15]:
```
all_features = masses_data[['age', 'shape',
                            'margin', 'density']].values


all_classes = masses_data['severity'].values

feature_names = ['age', 'shape', 'margin', 'density']

all_features
```

Out[15]:
```
array([[67.,  3.,  5.,  3.],
       [58.,  4.,  5.,  3.],
       [28.,  1.,  1.,  3.],
       ...,
       [64.,  4.,  5.,  3.],
       [66.,  4.,  5.,  3.],
       [62.,  3.,  3.,  3.]])
```

Age may have a bigger weight than other features because the numbers are higher
=> need to normalize
We can use scikitlearn **StandardScaler()**

Some of our models require the input data to be normalized, so go ahead and normalize the attribute data. Hint: use preprocessing.StandardScaler().

In [16]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler()
all_features_scaled = scaler.fit_transform(all_features)
all_features_scaled
```

Out[16]:
```
array([[ 0.7650629 ,  0.17563638,  1.39618483,  0.24046607],
       [ 0.15127063,  0.98104077,  1.39618483,  0.24046607],
       [-1.89470363, -1.43517241, -1.157718  ,  0.24046607],
       ...,
       [ 0.56046548,  0.98104077,  1.39618483,  0.24046607],
       [ 0.69686376,  0.98104077,  1.39618483,  0.24046607],
       [ 0.42406719,  0.17563638,  0.11923341,  0.24046607]])
```

Now set up an actual MLP model using Keras:

In [8]:
```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

def create_model():
    model = Sequential()
    #4 feature inputs going into an 6-unit layer (more does not seem to help - in fact you can go down to 4)
    model.add(Dense(6, input_dim=4, kernel_initializer='normal', activation='relu'))
    # "Deep learning" turns out to be unnecessary - this additional hidden layer doesn't help either.
    #model.add(Dense(4, kernel_initializer='normal', activation='relu'))
    # Output layer with a binary classification (benign or malignant)
    model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
    # Compile model; rmsprop seemed to work best
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

In [9]:
```
from sklearn.model_selection import cross_val_score
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

# Wrap our Keras model in an estimator compatible with scikit_learn
estimator = KerasClassifier(build_fn=create_model, epochs=100, verbose=0)
# Now we can use scikit_learn's cross_val_score to evaluate this model identically to the others
cv_scores = cross_val_score(estimator, all_features_scaled, all_classes, cv=10)
cv_scores.mean()
```

```
WARNING:tensorflow:From E:\Anaconda3\lib\site-packages\tensorflow\python\keras\initializers.py:143: calling Rando
__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
WARNING:tensorflow:From E:\Anaconda3\lib\site-packages\tensorflow\python\ops\nn_impl.py:180: add_dispatch_support
per (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Out[9]: 0.8036144554615021

Types of Data

Many Flavors of Data

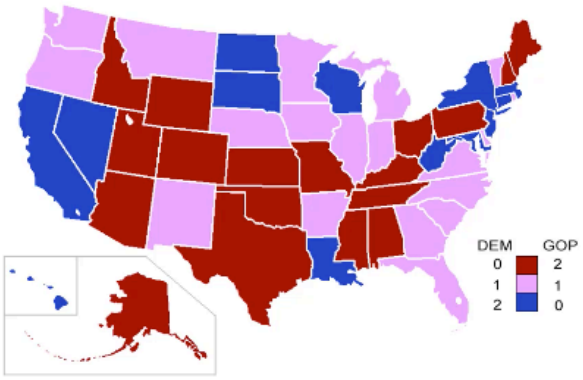
Numerical

- Represents some sort of quantitative measurement
  - Heights of people, page load times, stock prices, etc.
- Discrete Data
  - Integer based; often counts of some event.
    - How many purchases did a customer make in a year?
    - How many times did I flip "heads"?
- Continuous Data
  - Has an infinite number of possible values
    - How much time did it take for a user to check out?
    - How much rain fell on a given day?

written a note here

Categorical

- Qualitative data that has no inherent mathematical meaning
  - Gender, Yes/no (binary data), Race, State of Residence, Product Category, Political Party, etc.
- You can assign numbers to categories in order to represent them more compactly, but the numbers don't have mathematical meaning

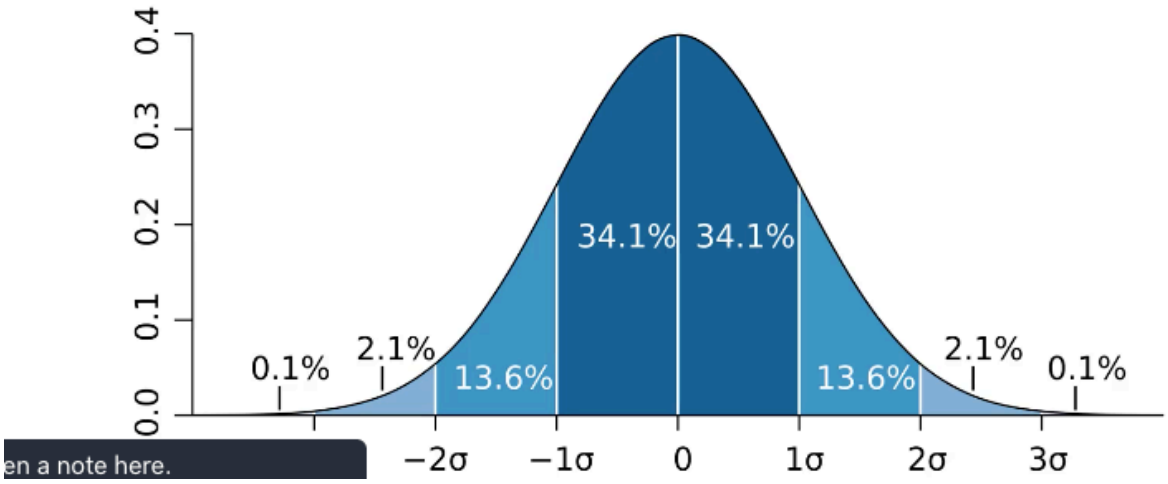


Ordinal

- A mixture of numerical and categorical
- Categorical data that has mathematical meaning
- Example: movie ratings on a 1-5 scale.
  - Ratings must be 1, 2, 3, 4, or 5
  - But these values have mathematical meaning; 1 means it's a worse movie

Data Distributions

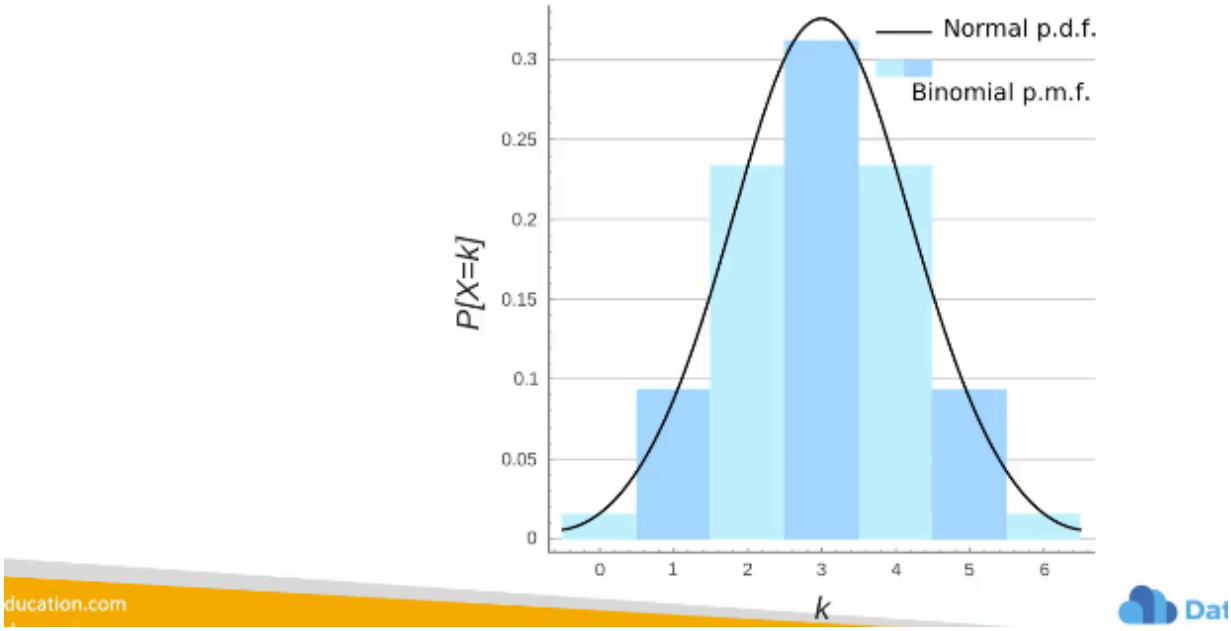
A "normal distribution"



en a note here.

For **discrete** data, use a **Probability Mass function** => histogram

Probability Mass Function



Example - dealing with discrete data
Like selling a certain number of house per day

Poisson Distribution

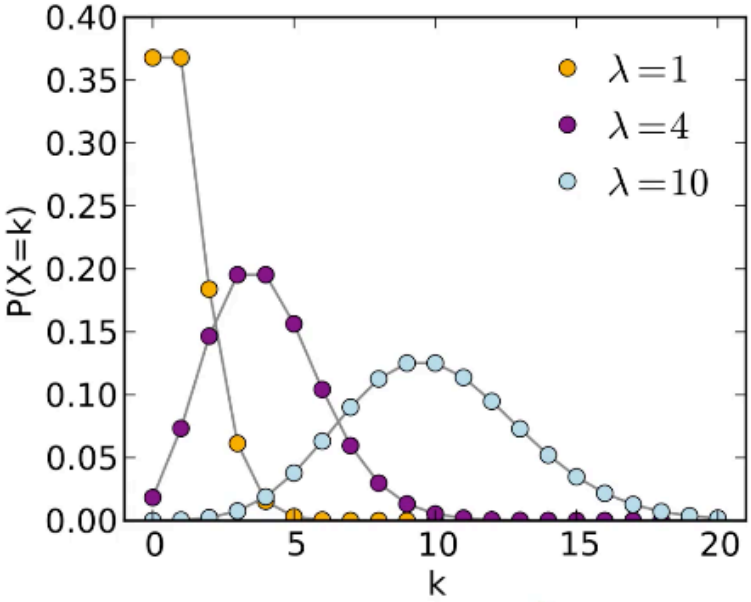
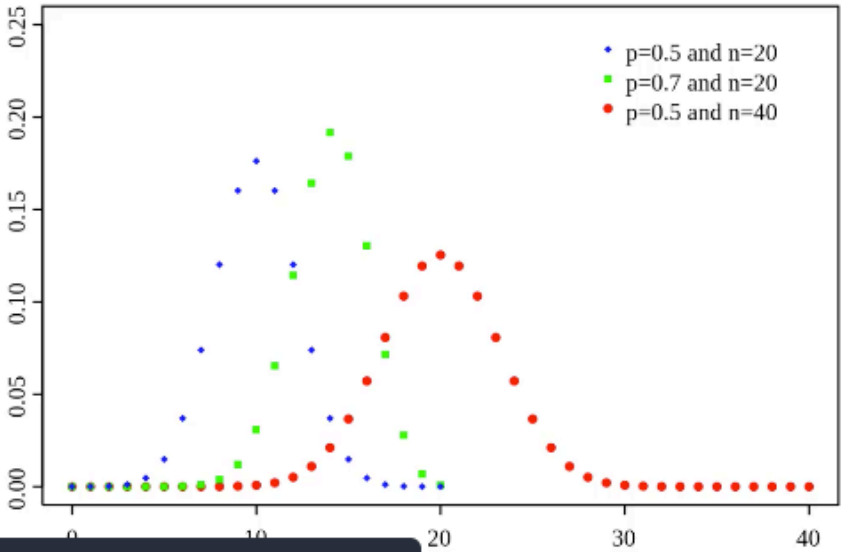


Image: Skbkekas, CC BY 3.0

0 or 1 outcome with many consecutive tries

Binomial Distribution

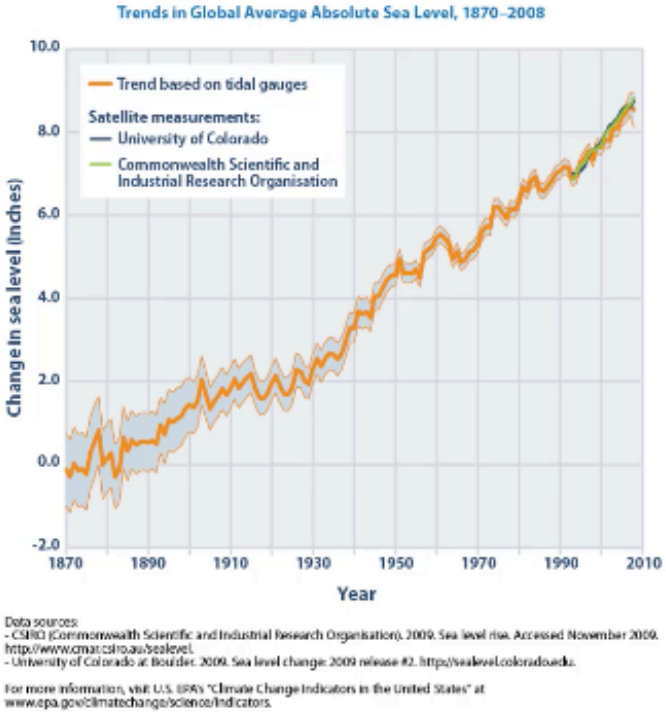


Bernoulli Distribution

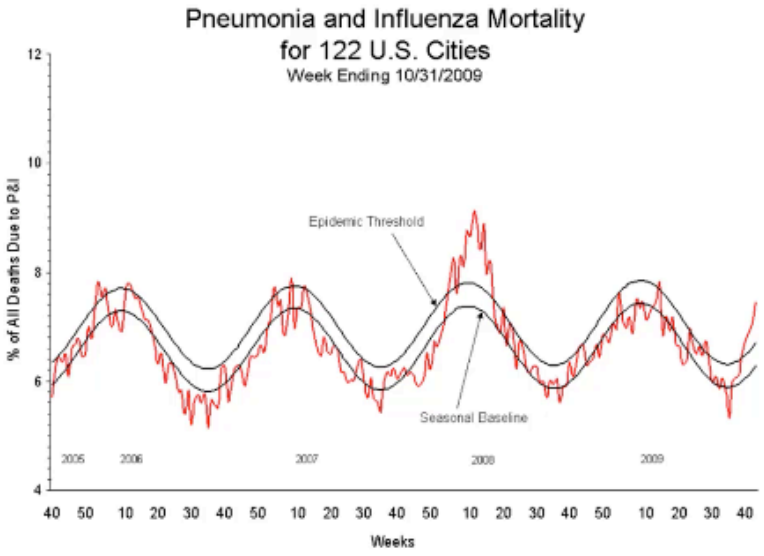
- Special case of binomial distribution
- Has a single trial (n=1)
- Can think of a binomial distribution as the sum of Bernoulli distributions

Time series Analysis

Trends

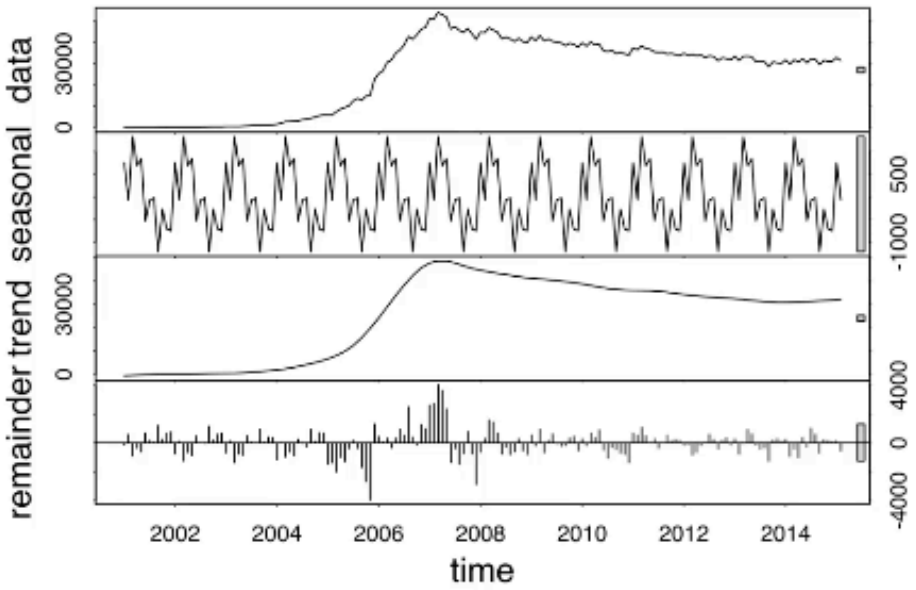


Seasonality



Wikipedia data.
If removing seasonality, we can see the trend

You can have both



Noise

- Some variations are just random in nature
- Seasonality + Trends + Noise = time series
  - Additive model
  - Seasonal variation is constant
- Or sometimes, seasonality * trends * noise – trends sometimes amplify seasonality and noise.
  - Multiplicative model
  - Seasonal variation increases as the trend increases

Amazon Athena

Key points
- **serverless**
- **can query with SQL on any type of data** (structured/unstructured: JSON, parquet, ORC,...)
- **cost effective if using columnar format: ORC, parquet** (better perf and save 30-60%)

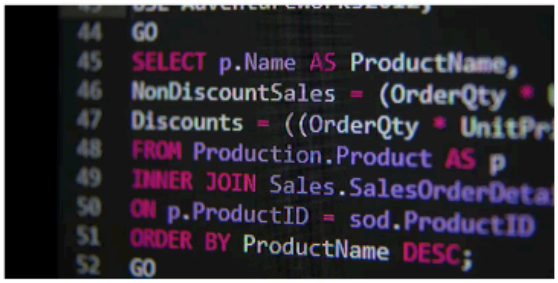
What is Athena?

- Interactive query service for S3 (SQL)
  - No need to load data, it stays in S3
- Presto under the hood
- Serverless!
- Supports many data formats
  - CSV (human readable)
  - JSON (human readable)
  - ORC (columnar, splittable)
  - Parquet (columnar, splittable)
  - Avro (splittable)
- Unstructured, semi-structured, or structured

Amazon Athena

Some examples

- Ad-hoc queries of web logs
- Querying staging data before loading to Redshift
- Analyze CloudTrail / CloudFront / VPC / ELB etc logs in S3
- Integration with Jupyter, Zeppelin, RStudio notebooks
- Integration with QuickSight
- Integration via ODBC / JDBC with other visualization tools



Athena + Glue

Glue crawler tacts structure from S3. Athena ca query on it. Quicksight can be used to visualize.

Athena cost model

- Pay-as-you-go
  - $5 per TB scanned
  - Successful or cancelled queries count, failed queries do not.
  - No charge for DDL (CREATE/ALTER/DROP etc.)
- Save LOTS of money by using columnar formats
  - ORC, Parquet
  - Save 30-90%, and get better performance
- Glue and S3 have their own charges

Athena Security

- Access control
  - IAM, ACLs, S3 bucket policies
  - AmazonAthenaFullAccess / AWSQuicksightAthenaAccess
- Encrypt results at rest in S3 staging directory
  - Server-side encryption with S3-managed key (SSE-S3)
  - Server-side encryption with KMS key (SSE-KMS)
  - Client-side encryption with KMS key (CSE-KMS)
- Cross-account access in S3 bucket policy possible
- Transport Layer Security (TLS) encrypts in-transit (between Athena and S3)

Athena anti-patterns

- Highly formatted reports / visualization
  - That's what QuickSight is for
- ETL
  - Use Glue instead

Amazon Quicksight

What is QuickSight?

- Fast, easy, cloud-powered business analytics service
- Allows all employees in an organization to:
  - Build visualizations
  - Perform ad-hoc analysis
  - Quickly get business insights from data
  - Anytime, on any device (browsers, mobile)
- Serverless

QuickSight Data Sources

- Redshift
- Aurora / RDS
- Athena
- EC2-hosted databases
- Files (S3 or on-premises)
  - Excel
  - CSV, TSV
  - Common or extended log format
- Data preparation allows limited ETL

SPICE

- Data sets are imported into SPICE
  - Super-fast, Parallel, In-memory Calculation Engine
  - Uses columnar storage, in-memory, machine code generation
  - Accelerates interactive queries on large datasets
- Each user gets 10GB of SPICE
- Highly available / durable
- Scales to hundreds of thousands of users

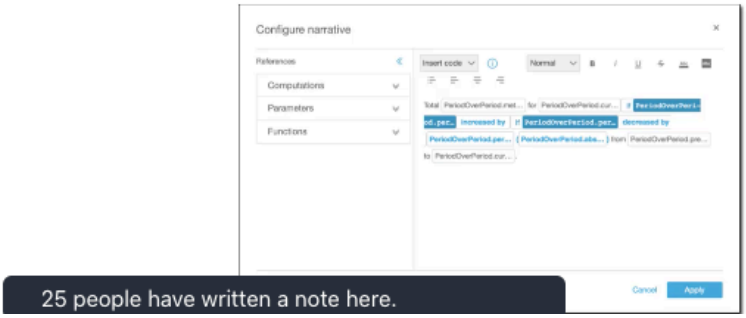
SPICE - mechanism by which Quisight is fast

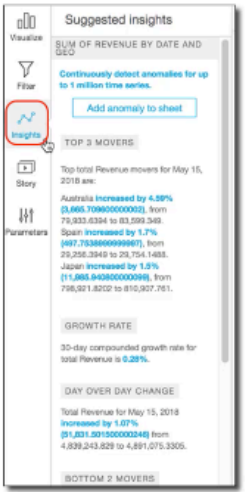
QuickSight Use Cases

- Interactive ad-hoc exploration / visualization of data
- Dashboards and KPI's
- Stories
  - Guided tours through specific views of an analysis
  - Convey key points, thought process, evolution of an analysis
- Analyze / visualize data from:
  - Logs in S3
  - On-premise databases
  - AWS (RDS, Redshift, Athena, S3)
  - SaaS applications, such as Salesforce
  - Any JDBC/ODBC data source

Machine Learning Insights

- Anomaly detection
- Forecasting
- Auto-narratives





Using RCF for anomaly detection
Also doing forecasting (with RCF)

QuickSight Anti-Patterns

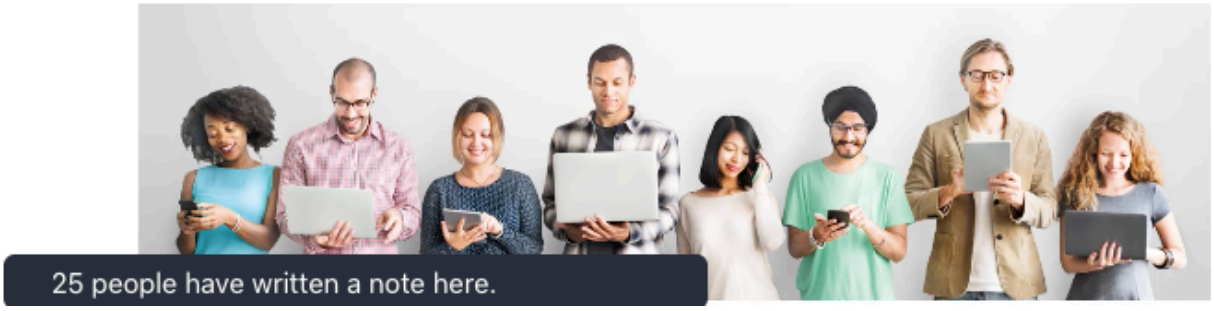
- Highly formatted canned reports
  - QuickSight is for ad-hoc queries, analysis, and visualization
- ETL
  - Use Glue instead, although QuickSight can do some transformations

QuickSight Security

- Multi-factor authentication on your account
- VPC connectivity
  - Add QuickSight's IP address range to your database security groups
- Row-level security
- Private VPC access
  - Elastic Network Interface, AWS Direct Connect

QuickSight User Management

- Users defined via IAM, or email signup
- Active Directory integration with QuickSight Enterprise Edition



QuickSight Pricing

- Annual subscription
  - Standard: $9 / user /month
  - Enterprise: $18 / user / month
- Extra SPICE capacity (beyond 10GB)
  - $0.25 (standard) $0.38 (enterprise) / GB / month
- Month to month
  - Standard: $12 / GB / month
  - Enterprise: $24 / GB / month
- Enterprise edition
  - Encryption at rest
  - Microsoft Active Directory integration

Quicksight Types of Visualization and When to use Them

QuickSight Dashboards

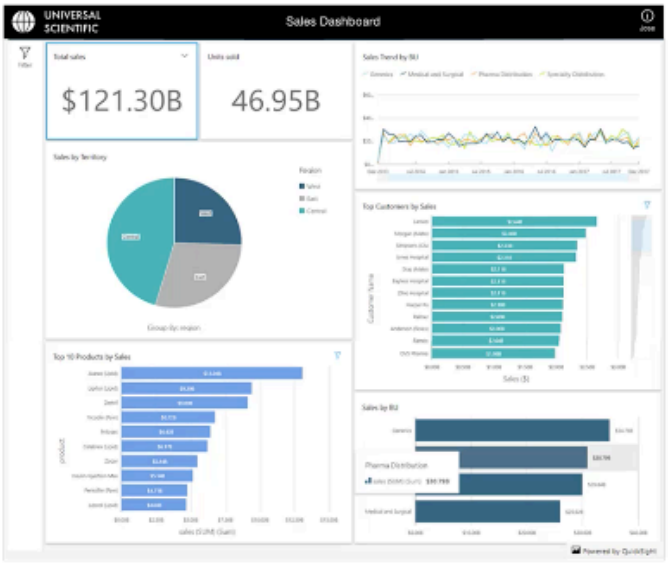
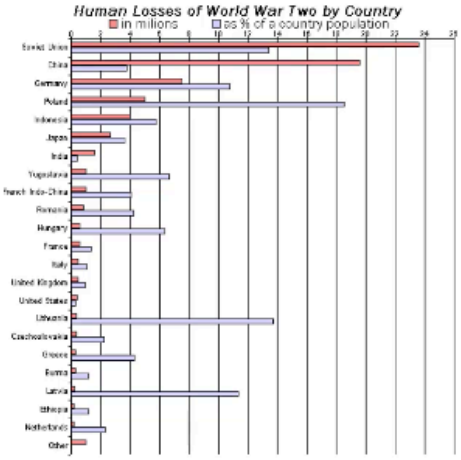


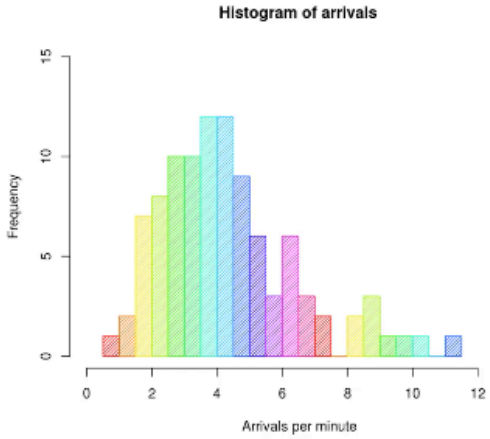
Image: AWS Big Data Blog

QuickSight Visual Types

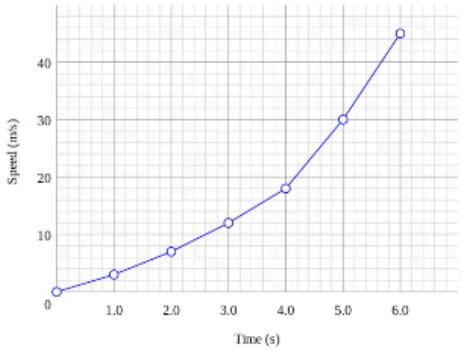
- AutoGraph
- Bar Charts
  - For comparison and distribution (histograms)
- Line graphs
  - For changes over time
- Scatter plots, heat maps
  - For correlation
- Pie graphs, tree maps
  - For aggregation
- Pivot tables
  - For tabular data
- Stories

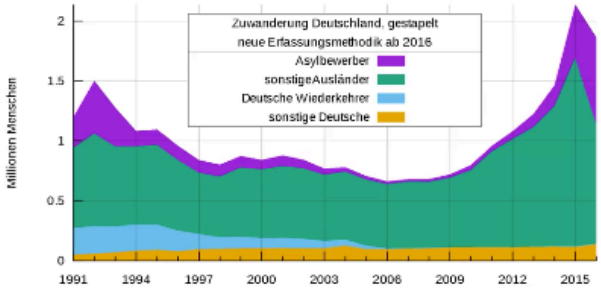
Bar Charts: Comparison, Distribution



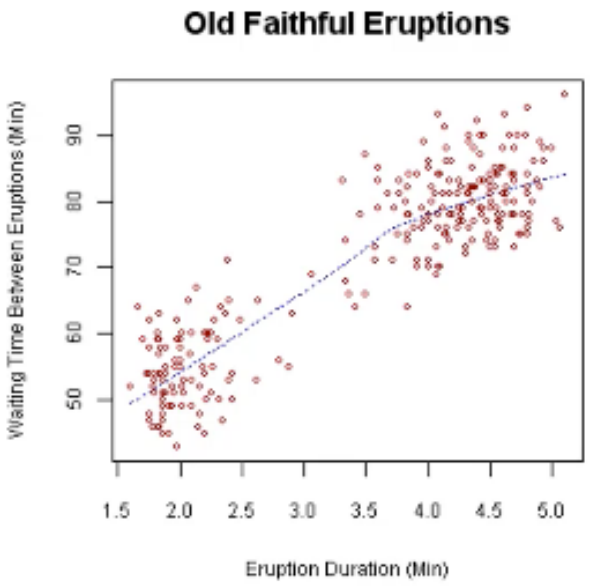


Line Charts: Changes over Time

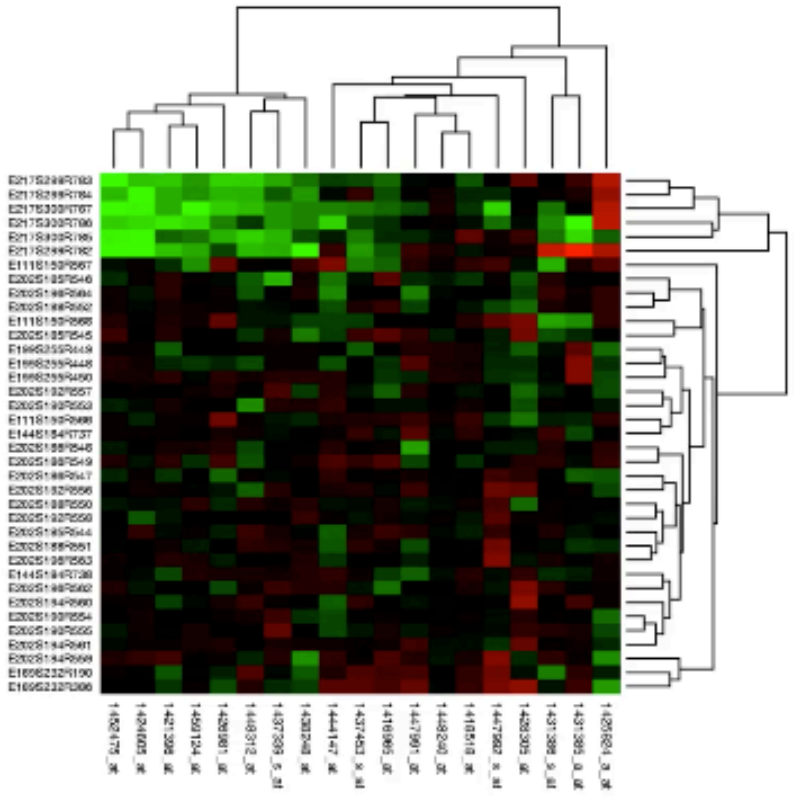




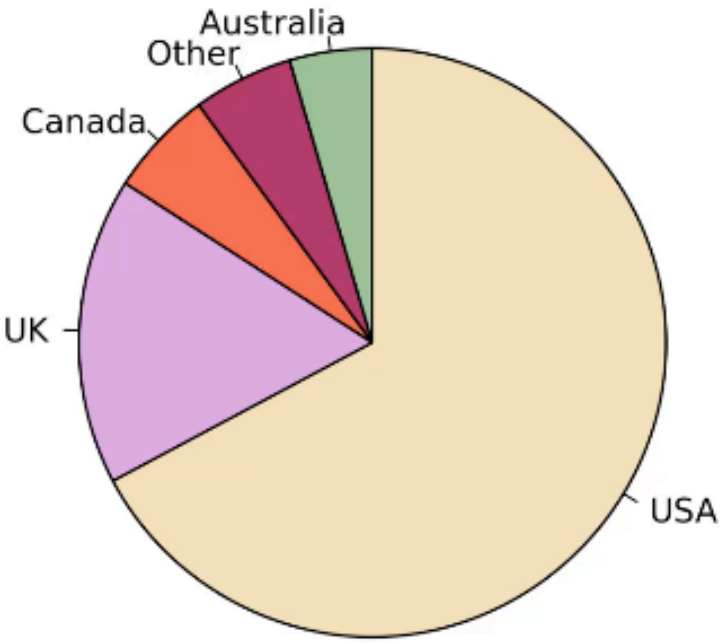
Scatter Plots: Correlation



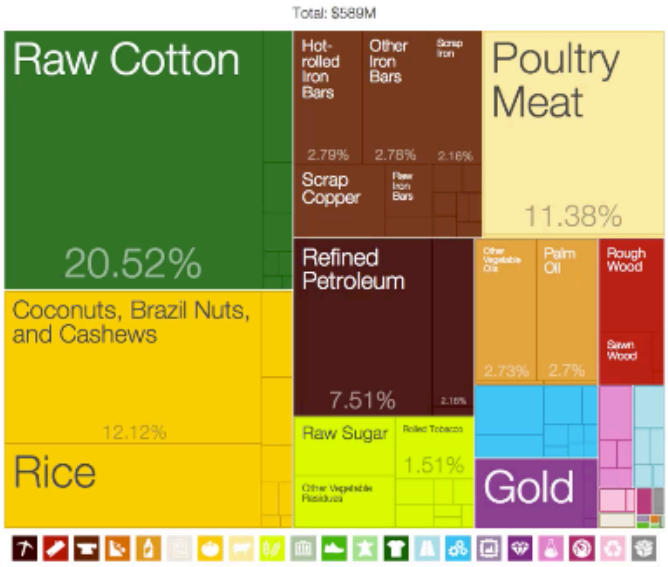
Heat Maps: Correlation



Pie Charts: Aggregation



Tree Maps: Heirarchical Aggregation



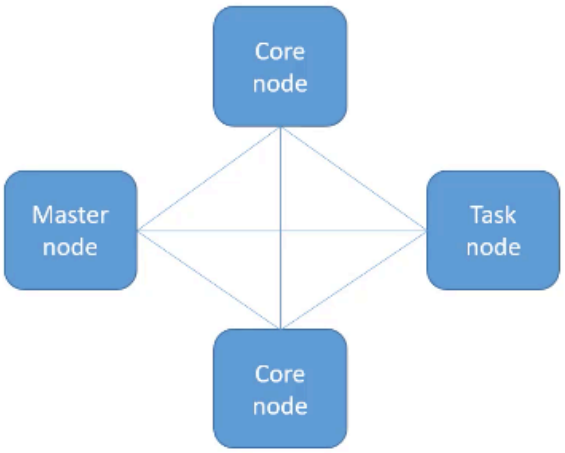
EMR (Elastic MapReduce) and Hadopp

What is EMR?

- Elastic MapReduce
- Managed Hadoop framework on EC2 instances
- Includes Spark, HBase, Presto, Flink, Hive & more
- EMR Notebooks
- Several integration points with AWS

Amazon EMR

An EMR Cluster



- **Master node**: manages the cluster
  - Single EC2 instance
- **Core node**: Hosts HDFS data and runs tasks
  - Can be scaled up & down, but with some risk
- **Task node**: Runs tasks, does not host data
  - No risk of data loss when removing
  - Good use of **spot instances**

EMR Usage

- Transient vs Long-Running Clusters
  - Can spin up task nodes using Spot instances for temporary capacity
  - Can use reserved instances on long-running clusters to save $
- Connect directly to master to run jobs
- Submit ordered steps via the console

EMR / AWS Integration

- Amazon EC2 for the instances that comprise the nodes in the cluster
- Amazon VPC to configure the virtual network in which you launch your instances
- Amazon S3 to store input and output data
- Amazon CloudWatch to monitor cluster performance and configure alarms
- AWS IAM to configure permissions
- AWS CloudTrail to audit requests made to the service
- AWS Data Pipeline to schedule and start your clusters

EMR Storage

- HDFS
- EMRFS: access S3 as if it were HDFS
  - EMRFS Consistent View – Optional for S3 consistency
  - Uses DynamoDB to track consistency
- Local file system
- EBS for HDFS

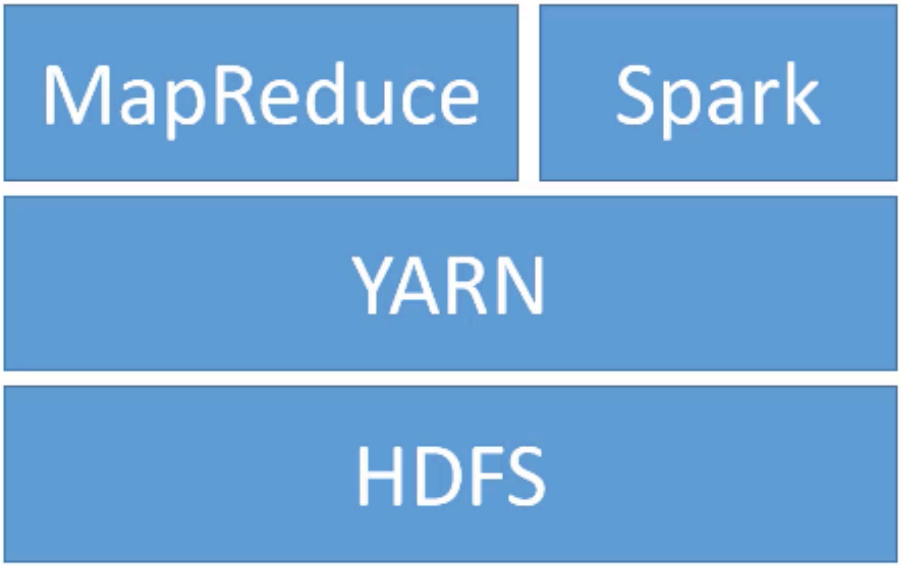
HDFS is ephemeral, attached to the cluster. It is fast (local on the nodes) but when cluster goes downs, the data is lost.
Alternatively, EMRFS uses S3, is still pretty fast but provides data consistency

EMR promises

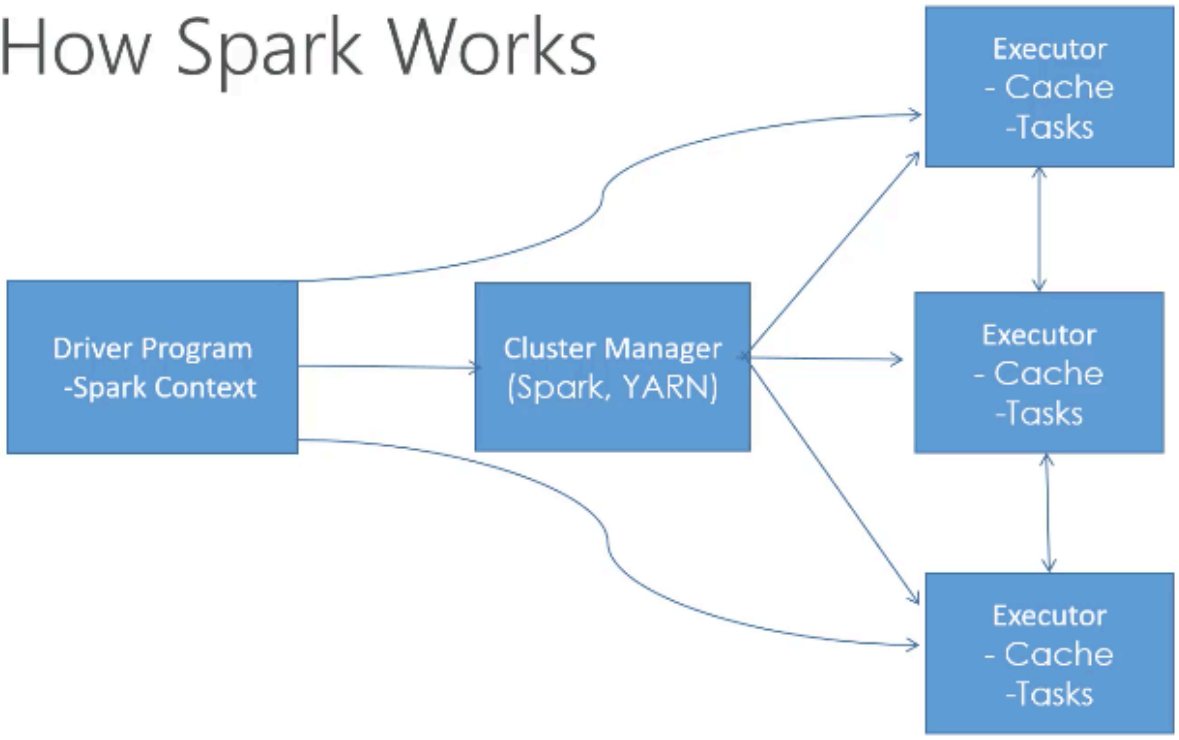
- EMR charges by the hour
  - Plus EC2 charges
- Provisions new nodes if a core node fails
- Can add and remove tasks nodes on the fly
- Can resize a running cluster's core nodes

Apache Spark on EMR

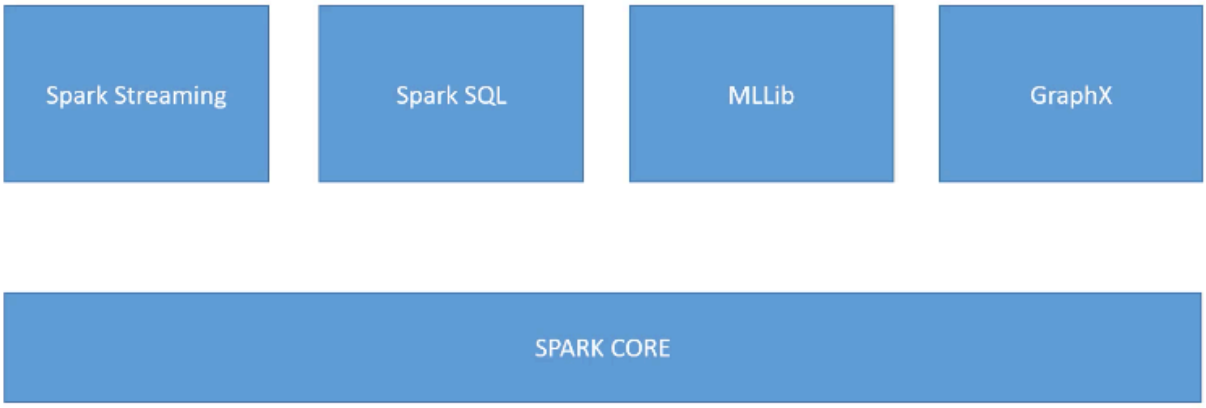
Apache Spark



How Spark Works



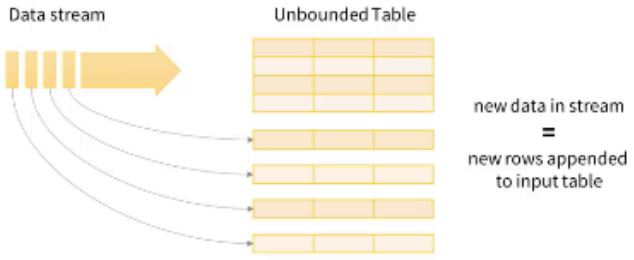
Spark Components



Spark MLLib

- Classification: logistic regression, naïve Bayes
- Regression
- Decision trees
- Recommendation engine (ALS)
- Clustering (K-Means)
- LDA (topic modeling)
- ML workflow utilities (pipelines, feature transformation, persistence)
- SVD, PCA, statistics

Spark Structured Streaming



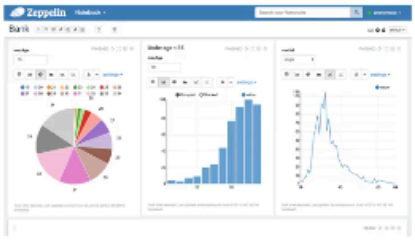
Data stream as an unbounded Input Table

```
val inputDF = spark.readStream.json("s3://logs")
inputDF.groupBy($"action", window($"time", "1 hour")).count()
          .writeStream.format("jdbc").start("jdbc:mysql//...")
```

Spark Streaming + Kinesis

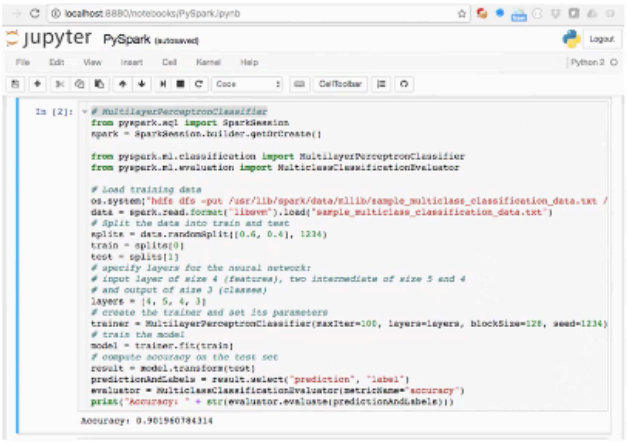
Zeppelin + Spark

- Can run Spark code interactively (like you can in the Spark shell)
  - This speeds up your development cycle
  - And allows easy experimentation and exploration of your big data
- Can execute SQL queries directly against SparkSQL
- Query results may be visualized in charts and graphs
- Makes Spark feel more like a data science tool!



EMR Notebook

- Similar concept to Zeppelin, with more AWS integration
- Notebooks backed up to S3
- Provision clusters from the notebook!
- Hosted inside a VPC
- Accessed only via AWS console



EMR Security

- IAM policies
- Kerberos
- SSH
- IAM roles

EMR: Choosing Instance Types

- Master node:
  - m4.large if < 50 nodes, m4.xlarge if > 50 nodes
- Core & task nodes:
  - m4.large is usually good
  - If cluster waits a lot on external dependencies (i.e. a web crawler), t2.medium
  - Improved performance: m4.xlarge
  - Computation-intensive applications: high CPU instances
  - Database, memory-caching applications: high memory instances
  - Network / CPU-intensive (NLP, ML) – cluster computer instances
- Spot instances
  - Good choice for task nodes
  - Only use on core & master if you're testing or very cost-sensitive; you're risking partial data loss

Feature Engineering and the Curse of Dimensionality

What is feature engineering?

- Applying your knowledge of the data – and the model you're using - to create better features to train your model with.
  - Which features should I use?
  - Do I need to transform these features in some way?
  - How do I handle missing data?
  - Should I create new features from the existing ones?
- You can't just throw in raw data and expect good results
- This is the art of machine learning; where expertise is applied
- "Applied machine learning is basically feature engineering" – Andrew Ng

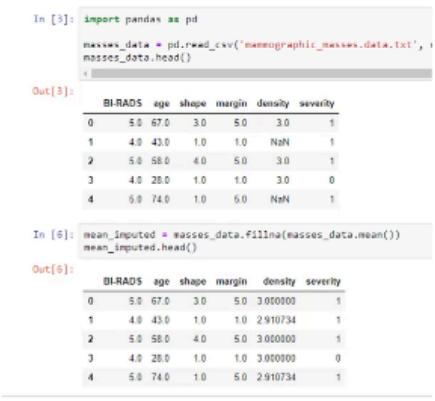
The Curse of Dimensionality

- Too many features can be a problem – leads to sparse data
- Every feature is a new dimension
- Much of feature engineering is selecting the features most relevant to the problem at hand
  - This often is where domain knowledge comes into play
- Unsupervised dimensionality reduction techniques can also be employed to distill many features into fewer features
  - PCA
  - K-Means

Imputing Missing data

Imputing Missing Data: Mean Replacement

- Replace missing values with the mean value from the rest of the column (columns, not rows! A column represents a single feature; it only makes sense to take the mean from other samples of the same feature.)
- Fast & easy, won't affect mean or sample size of overall data set
- Median may be a better choice than mean when outliers are present
- But it's generally pretty terrible.
  - Only works on column level, misses correlations between features
  - Can't use on categorical features (imputing with most frequent value can work in this case, though)
  - Not very accurate



```
In [3]: import pandas as pd

        masses_data = pd.read_csv('mammographic_masses.data.txt', n
        masses_data.head()
```

Out[3]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
1	4.0	43.0	1.0	1.0	NaN	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
4	5.0	74.0	1.0	5.0	NaN	1

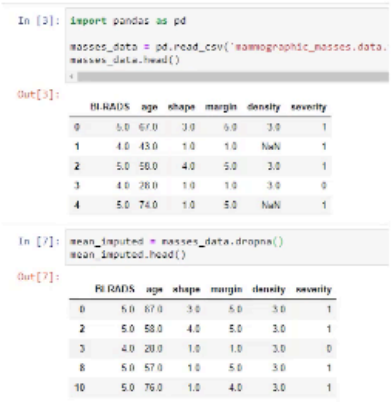
```
In [6]: mean_imputed = masses_data.fillna(masses_data.mean())
        mean_imputed.head()
```

Out[6]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.000000	1
1	4.0	43.0	1.0	1.0	2.910734	1
2	5.0	58.0	4.0	5.0	3.000000	1
3	4.0	28.0	1.0	1.0	3.000000	0
4	5.0	74.0	1.0	5.0	2.910734	1

Imputing Missing Data: Dropping

- If not many rows contain missing data…
  - …and dropping those rows doesn't bias your data…
  - …and you don't have a lot of time…
  - …maybe it's a reasonable thing to do.
- But, it's never going to be the right answer for the "best" approach.
- Almost anything is better. Can you substitute another similar field perhaps? (i.e., review summary vs. full text)



```
In [3]: import pandas as pd

        masses_data = pd.read_csv('mammographic_masses.data.
        masses_data.head()
```

Out[3]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
1	4.0	43.0	1.0	1.0	NaN	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
4	5.0	74.0	1.0	5.0	NaN	1

```
In [7]: mean_imputed = masses_data.dropna()
        mean_imputed.head()
```

Out[7]:

	BI-RADS	age	shape	margin	density	severity
0	5.0	67.0	3.0	5.0	3.0	1
2	5.0	58.0	4.0	5.0	3.0	1
3	4.0	28.0	1.0	1.0	3.0	0
8	5.0	57.0	1.0	5.0	3.0	1
10	5.0	76.0	1.0	4.0	3.0	1

What we REALLY want to do for dealing with missing values:

Imputing Missing Data: Machine Learning

- KNN: Find K "nearest" (most similar) rows and average their values
  - Assumes numerical data, not categorical
  - There are ways to handle categorical data (Hamming distance), but categorical data is probably better served by...
- Deep Learning
  - Build a machine learning model to impute data for your machine learning model!
  - Works well for categorical data. Really well. But it's complicated.
- Regression
  - Find linear or non-linear relationships between the missing feature and other features
  - Most advanced technique: MICE (Multiple Imputation by Chained Equations)

Best way for dealing with missing data:

Imputing Missing Data: Just Get More Data

- What's better than imputing data? Getting more real data!
- Sometimes you just have to try harder or collect more data

Dealing with Unbalanced Data

What is unbalanced data?

- Large discrepancy between "positive" and "negative" cases
  - i.e., fraud detection. Fraud is rare, and most rows will be not-fraud
  - Don't let the terminology confuse you; "positive" doesn't mean "good"
    - It means the thing you're testing for is what happened.
    - If your machine learning model is made to detect fraud, then fraud is the positive case.
- Mainly a problem with neural networks

Oversampling

- Duplicate samples from the minority class
- Can be done at random

Undersampling

- Instead of creating more positive samples, remove negative ones
- Throwing data away is usually not the right answer
  - Unless you are specifically trying to avoid "big data" scaling issues

Something better than Under or Over Sampling: SMOTE

SMOTE

- Synthetic Minority Over-sampling TEchnique
- Artificially generate new samples of the minority class using nearest neighbors
  - Run K-nearest-neighbors of each sample of the minority class
  - Create a new sample from the KNN result (mean of the neighbors)
- Both generates new samples and undersamples majority class
- Generally better than just oversampling

Adjusting thresholds

- When making predictions about a classification (fraud / not fraud), you have some sort of threshold of probability at which point you'll flag something as the positive case (fraud)
- If you have too many false positives, one way to fix that is to simply increase that threshold.
  - Guaranteed to reduce false positives
  - But, could result in more false negatives

Handling Outliers:

Data Points that lie **more than one standard deviation from the mean can be considered unusual**
Extreme data points will usually be measured based **on how many signs they are away from the mean**
A good way to deal with "anomalies" and "outliers" is to use **RCF**

Variance measures how "spread-out" the data is.

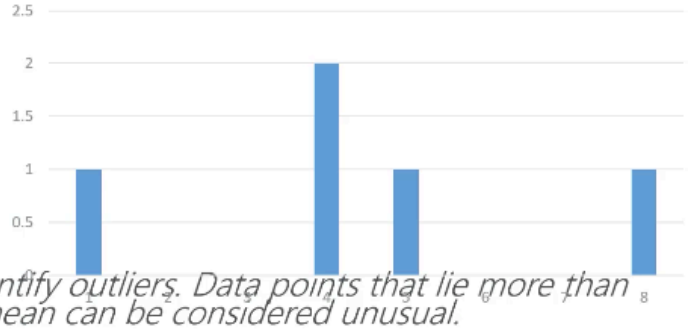
- Variance (σ^2) is simply the **average of the squared differences from the mean**
- Example: What is the variance of the data set (1, 4, 5, 4, 8)?
  - First find the mean: (1+4+5+4+8)/5 = 4.4
  - Now find the differences from the mean: (-3.4, -0.4, 0.6, -0.4, 3.6)
  - Find the squared differences: (11.56, 0.16, 0.36, 0.16, 12.96)
  - Find the average of the squared differences:
    - σ^2 = (11.56 + 0.16 + 0.36 + 0.16 + 12.96) / 5 = 5.04

Standard Deviation σ is just the square root of the variance.

$\sigma^2 = 5.04$

$\sigma = \sqrt{5.04} = 2.24$

So the standard deviation of (1, 4, 5, 4, 8) is 2.24.



This is usually used as a way to identify outliers. Data points that lie more than one standard deviation from the mean can be considered unusual.

You can talk about how extreme a data point is by talking about "how many sigmas" away from the mean it is.

Dealing with Outliers

- Sometimes it's appropriate to remove outliers from your training data
- Do this responsibly! Understand why you are doing this.
- For example: in collaborative filtering, a single user who rates thousands of movies could have a big effect on everyone else's ratings. That may not be desirable.
- Another example: in web log data, outliers may represent bots or other agents that should be discarded.
- But if someone really wants the mean income of US citizens for example, don't toss out billionaires just because you want to.

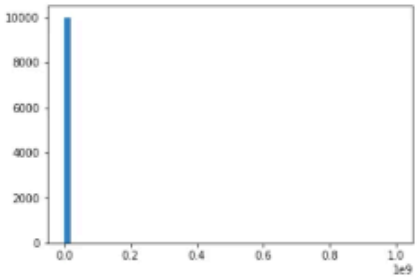
Dealing with Outliers

- Our old friend standard deviation provides a principled way to classify outliers.
- Find data points more than some multiple of a standard deviation in your training data.
- What multiple? You just have to use common sense.
- Remember AWS's Random Cut Forest algorithm creeps into many of its services – it is made for outlier detection
  - Found within QuickSight, Kinesis Analytics, SageMaker, and more

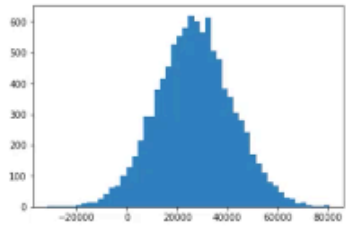
Rejecting outliers to see properly the data distort:

Example: Income Inequality

```
%matplotlib inline
import numpy as np

incomes = np.random.normal(27000, 15000, 10000)
incomes = np.append(incomes, [1000000000])

import matplotlib.pyplot as plt
plt.hist(incomes, 50)
plt.show()
```




```
def reject_outliers(data):
    u = np.median(data)
    s = np.std(data)
    filtered = [e for e in data if (u - 2 * s < e < u + 2 * s)]
    return filtered

filtered = reject_outliers(incomes)

plt.hist(filtered, 50)
plt.show()
```



That looks better. And, our mean is more, well, meangingful now as well.

```
np.mean(filtered)
```

```
26894.643431053548
```

Binning, Transforming, Encoding, Scaling and Shufling

Binning

- Bucket observations together based on ranges of values.
- Example: estimated ages of people
  - Put all 20-somethings in one classification, 30-somethings in another, etc.
- Quantile binning categorizes data by their place in the data distribution
  - Ensures even sizes of bins
- Transforms numeric data to ordinal data
- Especially useful when there is uncertainty in the measurements

Quantile Binning -> same number of values per bin

Transforming

- Applying some function to a feature to make it better suited for training
- Feature data with an exponential trend may benefit from a logarithmic transform
- Example: YouTube recommendations
  - A numeric feature x is also represented by x^2 and $\sqrt{x}$
  - This allows learning of super and sub-linear functions
- (ref: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf)

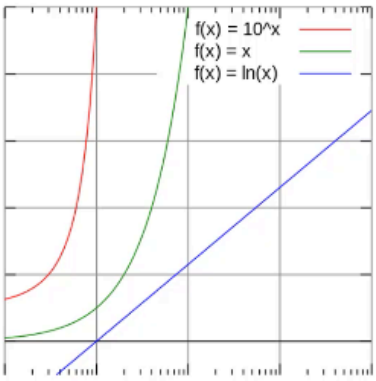
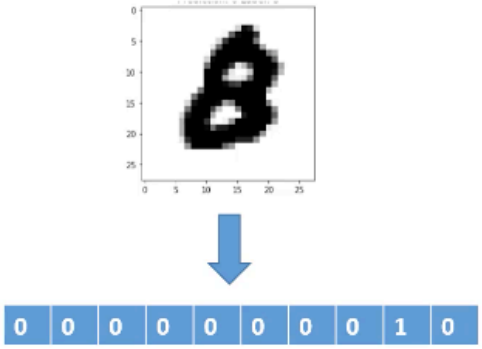


Image: By Autopilot - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10733854

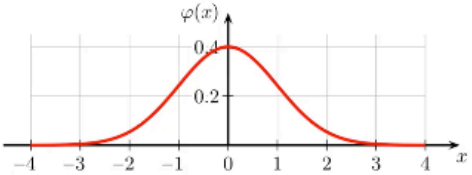
Encoding

- Transforming data into some new representation required by the model
- One-hot encoding
  - Create "buckets" for every category
  - The bucket for your category has a 1, all others have a 0
  - Very common in deep learning, where categories are represented by individual output "neurons"



Scaling / Normalization

- Some models prefer feature data to be normally distributed around 0 (most neural nets)
- Most models require feature data to at least be scaled to comparable values
  - Otherwise features with larger magnitudes will have more weight than they should
  - Example: modeling age and income as features – incomes will be much higher values than ages
- Scikit_learn has a preprocessor module that helps (MinMaxScaler, etc)
- Remember to scale your results back up



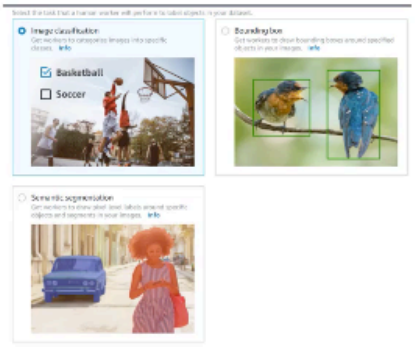
Shuffling

- Many algorithms benefit from shuffling their training data
- Otherwise they may learn from residual signals in the training data resulting from the order in which they were collected

SageMaker GroundTruth
Leveraging human to get labels on data.
If missing data that can be inferred by human, GroundTruth can help set tasks to get the info from human beings and manage a “fleet of human beings”

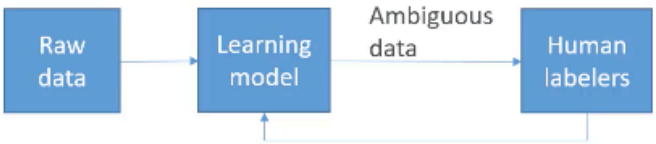
What is Ground Truth?

- Sometimes you don’t have training data at all, and it needs to be generated by humans first.
- Example: training an image classification model. Somebody needs to tag a bunch of images with what they are images of before training a neural network
- Ground Truth manages humans who will label your data for training purposes



But it’s more than that

- Ground Truth creates its own model as images are labeled by people
- As this model learns, only images the model isn’t sure about are sent to human labelers
- This can reduce the cost of labeling jobs by 70%



Who are these human labelers?

- Mechanical Turk
- Your own internal team
- Professional labeling companies

Other ways to generate training labels

- Rekognition
  - AWS service for image recognition
  - Automatically classify images
- Comprehend
  - AWS service for text analysis and topic modeling
  - Automatically classify text by topics, sentiment
- Any pre-trained model or unsupervised technique that may be helpful

Quiz

Question 1:

Which kind of graph is best suited for visualizing outliers in your training data?

- ○ Pair Plot
- ◉ Box and Whisker plot
- ○ Tree map

Question 2:

What sort of data distribution would be relevant to flipping heads or tails on a coin?

- ◉ Binomial distribution
- ○ Poisson distribution
- ○ Normal distribution

Binomial distributions are used for binary classifications of discrete events, such as flipping a coin.

Poisson is also for discrete events, but not necessarily binary ones.

The **binomial distribution** with parameters *n* and *p* is the discrete probability distribution of the number of successes in a sequence of *n* independent experiments, each asking a yes–no question, and each with its own Boolean-valued outcome: success/yes/true/one (with probability *p*) or failure/no/false/zero (with probability *q* = 1 − *p*). A single success/failure experiment is also called a Bernoulli trial or Bernoulli experiment, and a sequence of outcomes is called a Bernoulli process; for a single trial, i.e., *n* = 1, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

Question 3:

What is a serverless, fully-managed solution for querying unstructured data in S3?

- ○ Amazon Redshift
- ○ Amazon RDS
- ● Amazon Athena

Athena, when paired with a Glue crawler, can issue SQL queries directly on S3 data in a fully-managed setting.

Question 4:

Which of the following is NOT a feature of Amazon Quicksight's Machine Learning Insights?

- ● Visualization of precision and recall
- ○ Anomaly detection
- ○ Forecasting
- ○ Auto-narratives

Question 5:

Which imputation technique for missing data would produce the best results?

- ○ Mean or median replacement
- ○ Dropping
- ● MICE