Aftermarket Concert Tickets

The All-American Regex

September 2018

Introduction

Concert Ticket Industry

SIZE

$5.5Bn

US Revenue, 2017*

STRUCTURE

Primary

ticketmaster®

axs

TICKETFLY

Secondary

StubHub

SeatGeek

VIVIDSEATS

ISSUES

Tickets sell out instantly on primary market

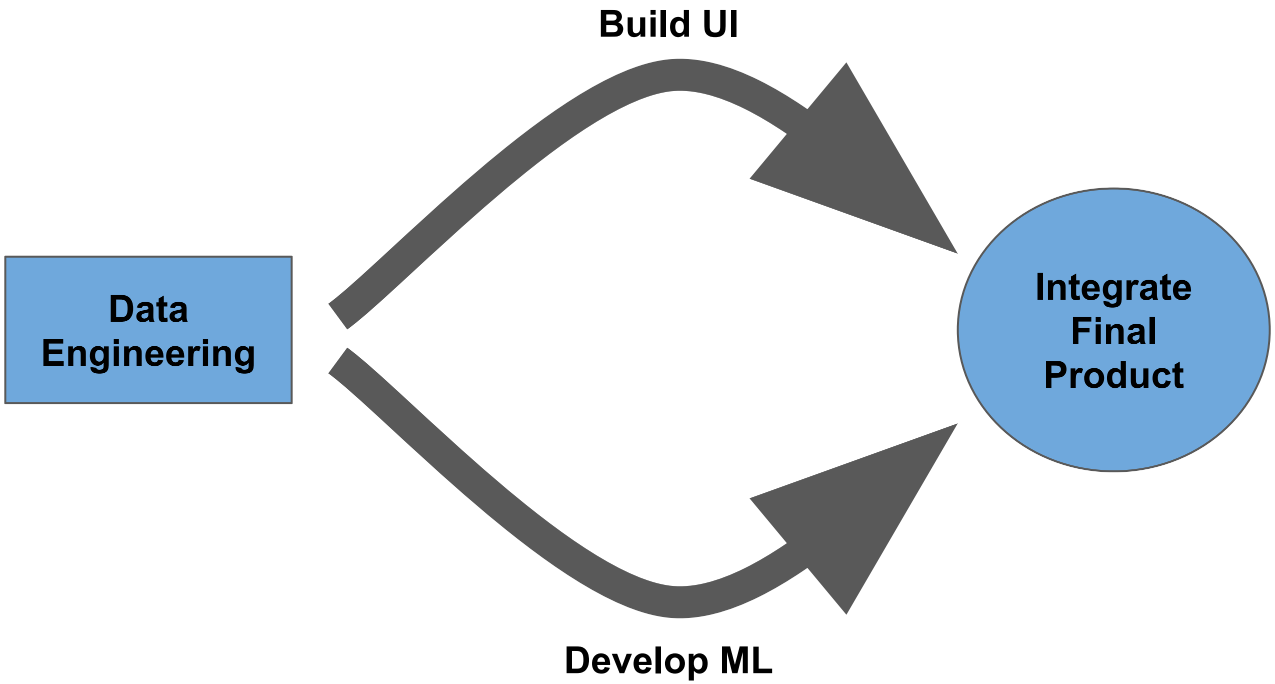
Hard to navigate aftermarket options

Artists want fans to be able to purchase at reasonable prices

Stop scalpers from reaping profits

**Statista*

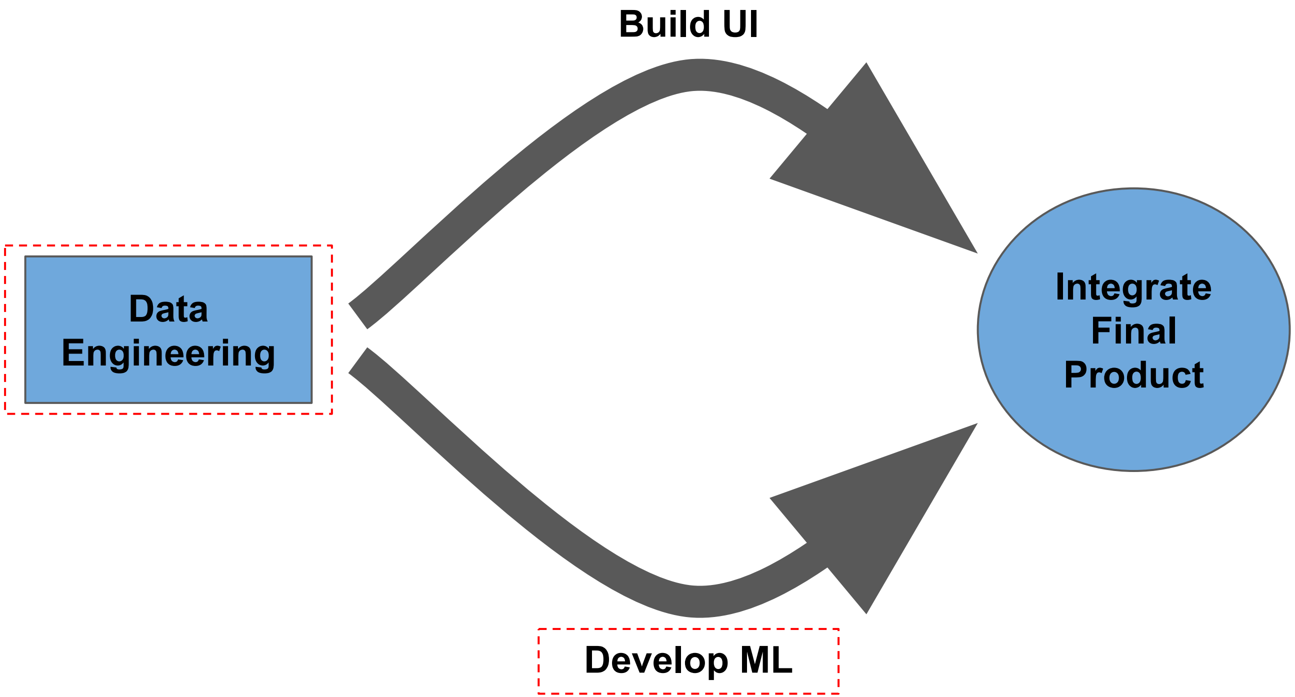
Project Goal: Consumer Ticket Application



Product Features

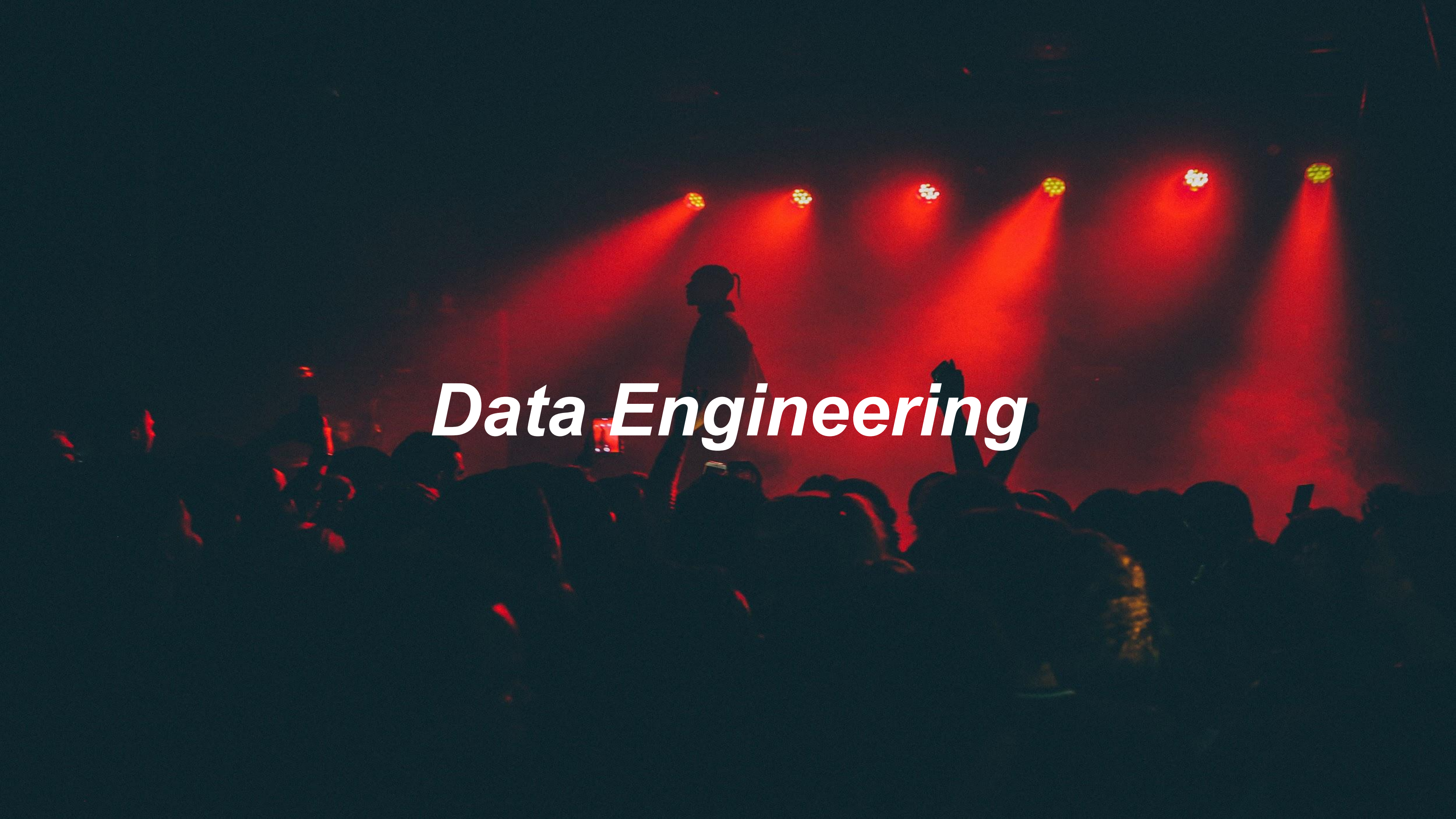
- Browse available concert tickets
- Compare availability on aftermarket sources
- Get recommended action (buy or wait)

Project Goal: Consumer Ticket Application



Product Features

- Browse available concert tickets
- Compare availability on aftermarket sources
- Get recommended action (buy or wait)



Data Engineering

Data Sources

	ticketmaster®	StubHub	SeatGeek	Spotify®
Upcoming Event Details	✔	✔	✔	Artist information such as genres, followers, images, and popularity
Individual Ticket Listings		✔		
Daily API Limit	5k requests	~15k requests	Unknown	None (50 artists per request)


ticketmaster
StubHub
SeatGeek
Spotify
amazon
S3
amazon
REDSHIFT

End-to-End Process

API

Wrote scripts to:

- Make API calls to obtain jsons
- Extract fields of interest from jsons
- Compile into pandas dataframes and then send CSV to S3

S3

Simple Cloud Storage Service

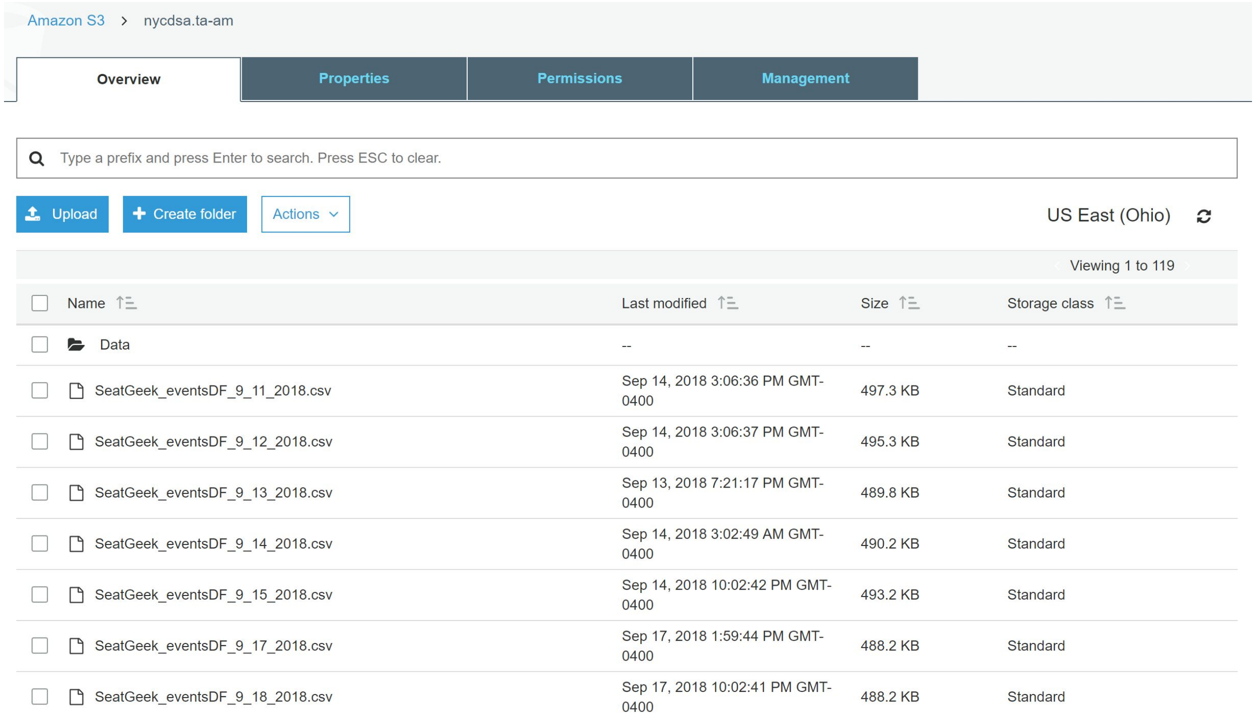
- Stored raw data collected from APIs as CSVs

Redshift

Fully Managed Data Warehouse

- Upload S3 data to cluster
- Fast query performance with SQL-based tools

S3 - Simple Storage Service



At end of each Python script:

Push CSVs to S3 cloud

S3 stores raw data -- > Redshift clusters will be shut down

Redshift - Relational Database Warehouse

- ❖ Python scripts to create and update tables in Redshift SQL database
- ❖ Schema to organize the tables:
  - ➢ Ticketmaster, Stubhub, Seatgeek

NAME ▲	TYPE ▲	CATALOG ▲	SCHEMA ▲
artist_details	TABLE	dev	ticketmaster
event_details	TABLE	dev	ticketmaster
presales_details	TABLE	dev	ticketmaster
price_areas	TABLE	dev	ticketmaster
prices	TABLE	dev	ticketmaster

NAME ▲	TYPE ▲	CATALOG ▲	SCHEMA ▲
events_df	TABLE	dev	stubhub
events_perf	TABLE	dev	stubhub
events_scores	TABLE	dev	stubhub
events_ticket_summary	TABLE	dev	stubhub
tickets_deliv_method	TABLE	dev	stubhub
tickets_deliv_type	TABLE	dev	stubhub
tickets_df	TABLE	dev	stubhub
tickets_listing_attr	TABLE	dev	stubhub
tickets_splits	TABLE	dev	stubhub
venues_df	TABLE	dev	stubhub

NAME ▲	TYPE ▲	CATALOG ▲	SCHEMA ▲
events_df	TABLE	dev	seatgeek
prices_df	TABLE	dev	seatgeek
venues_df	TABLE	dev	seatgeek

Redshift - Relational Database Warehouse

- ❖ Events, performers, venues, only need to be updated if new events are added to API

```
# fill only if event_id is unique
engine.execute(text("""INSERT INTO stubhub.events_df
    SELECT w.*
    FROM working.events_df AS w
    WHERE
        NOT EXISTS (SELECT 1
            FROM stubhub.events_df AS s
            WHERE s.event_id = w.event_id);""").execution_options(autocommit=True))
```

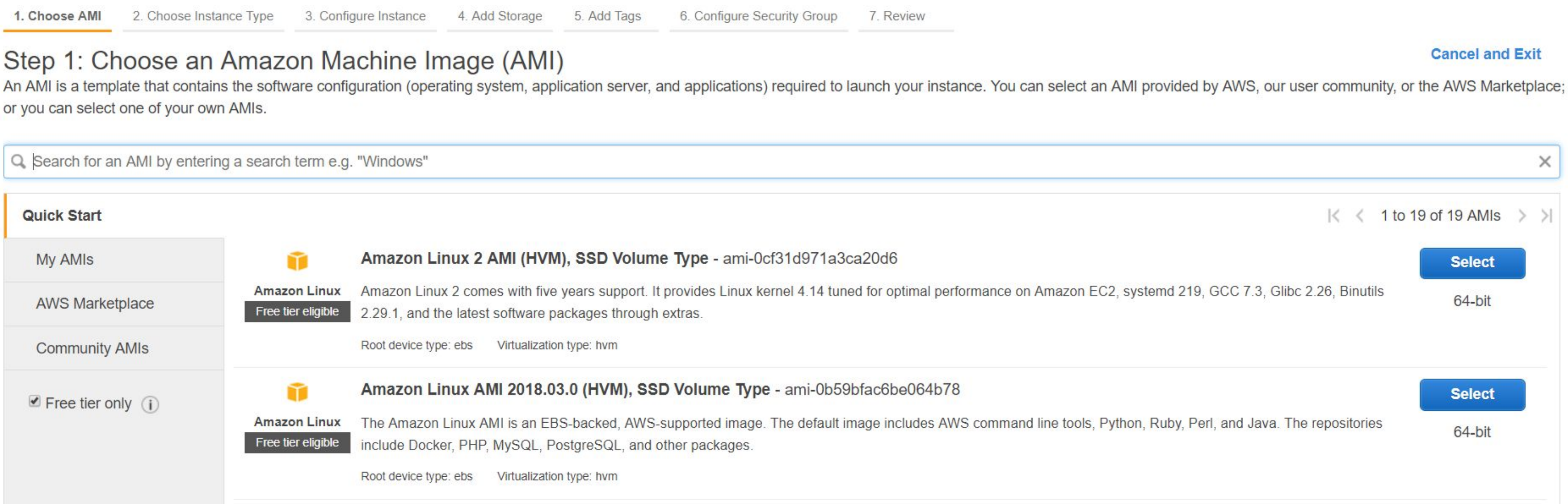
- ❖ Ticket listings to be appended to table each day
  - ➢ Need to track if ticket is listed or sold

```
# fill into original table
engine.execute(text("""ALTER TABLE stubhub.events_ticket_summary APPEND FROM
working.events_ticket_summary;""").execution_options(autocommit=True))
```


ticketmaster
StubHub
SeatGeek
Spotify
amazon
S3
amazon
REDSHIFT

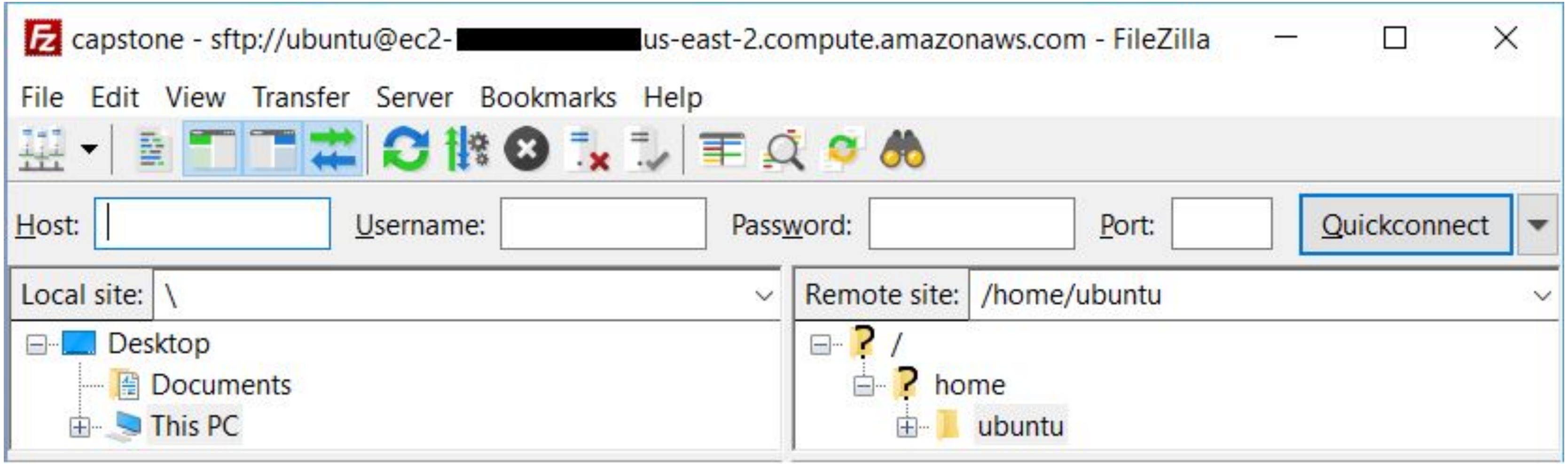
Setting Up Our Virtual Machine with EC2

- Amazon Elastic Compute Cloud (EC2) is a cloud-computing platform where users can rent virtual machines (VM)
- We wanted to run our Python scripts and process the data in the cloud



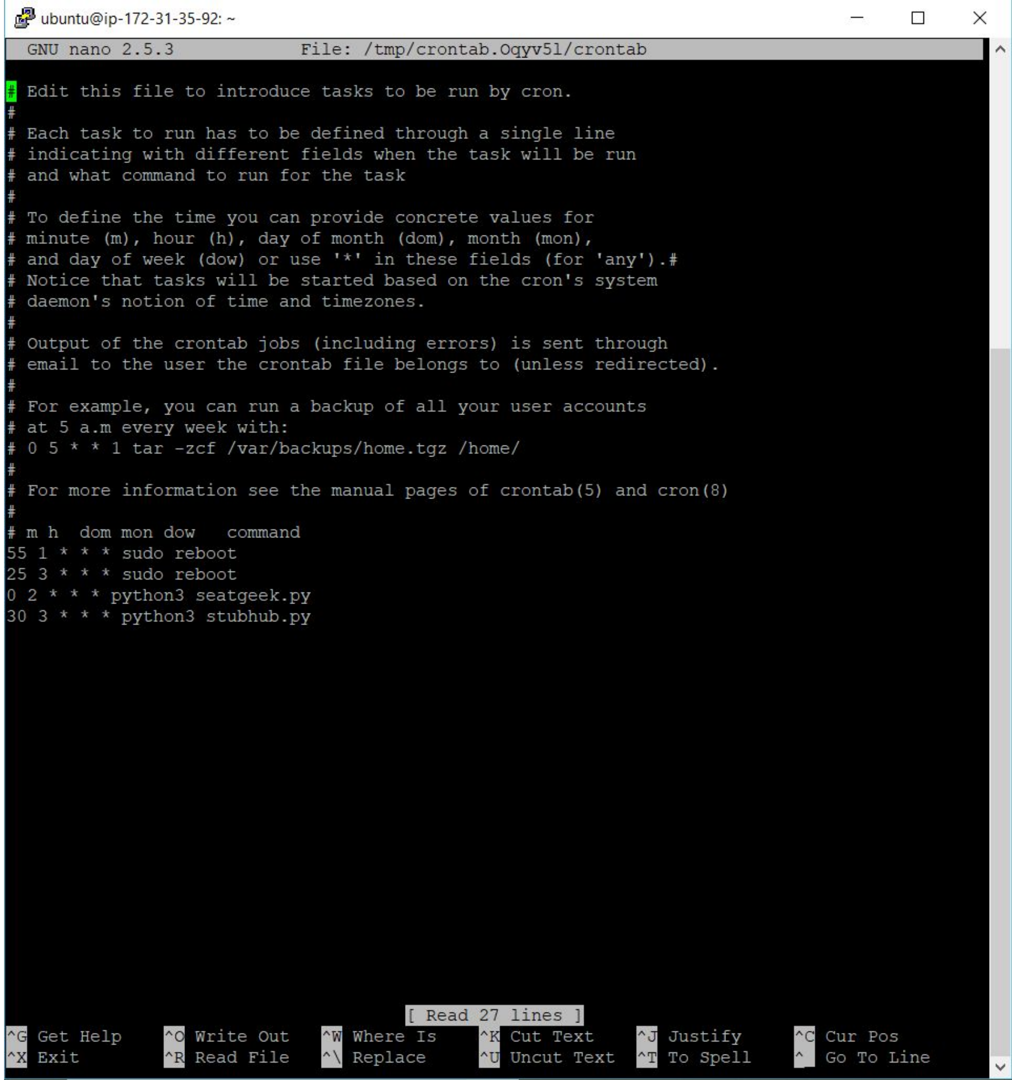
Interacting With Our Virtual Machine

- We can access the terminal of our VM using an SSH client (PuTTY)
- Similarly, we can use an FTP client (FileZilla) to access its directories



Scheduling

Since our VM's operating system was Ubuntu, we took advantage of *cron* to schedule our scripts to run in the middle of the night



ubuntu@ip-172-31-35-92: ~

```
GNU nano 2.5.3            File: /tmp/crontab.Oqyv5l/crontab

# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
55 1 * * * sudo reboot
25 3 * * * sudo reboot
0 2 * * * python3 seatgeek.py
30 3 * * * python3 stubhub.py

                         [ Read 27 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

Predictive Modeling

Data Overview

Data Used for Modeling	
Source	StubHub only
Dates Collected	9/8/2018 - 9/12/2018 5 days observed → 3 days training data + 1 day holdout data + 1 day lost as reference
Cities	New York, Boston, Chicago, Washington D.C., San Francisco

1.2M
Observations

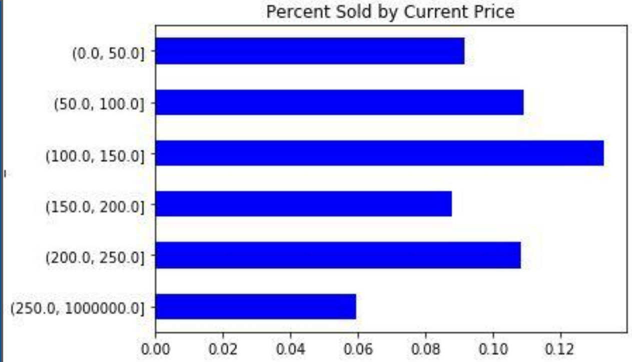
4.9K
Concerts

2.5K
Performers

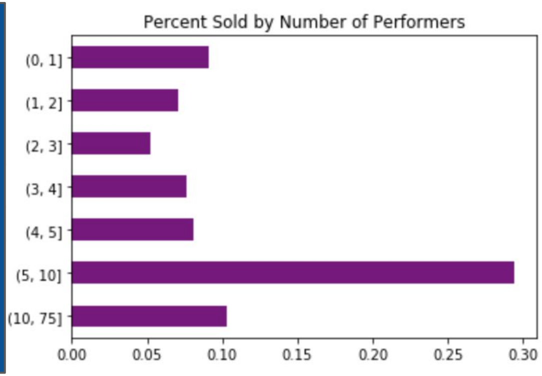
345
Venues

Feature Exploration

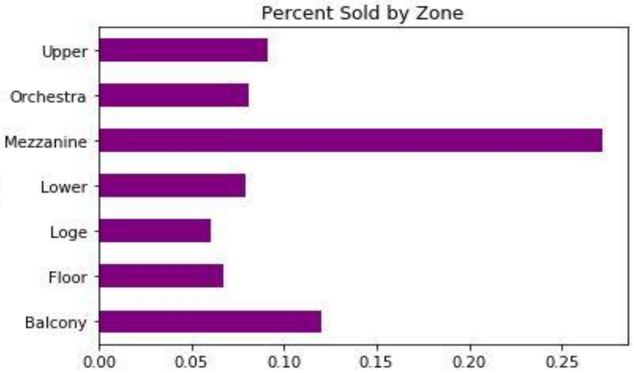
Price



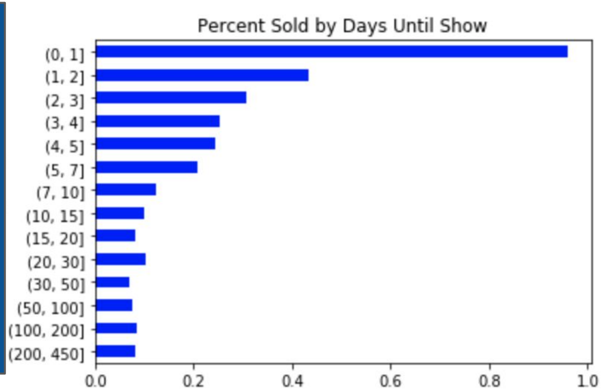
Event Attributes



Ticket Attributes



Urgency



Model Evaluation

Default Cut-Off (p = 0.5)

$$\begin{bmatrix} 30819 & 173 \\ 3614 & 172 \end{bmatrix}$$

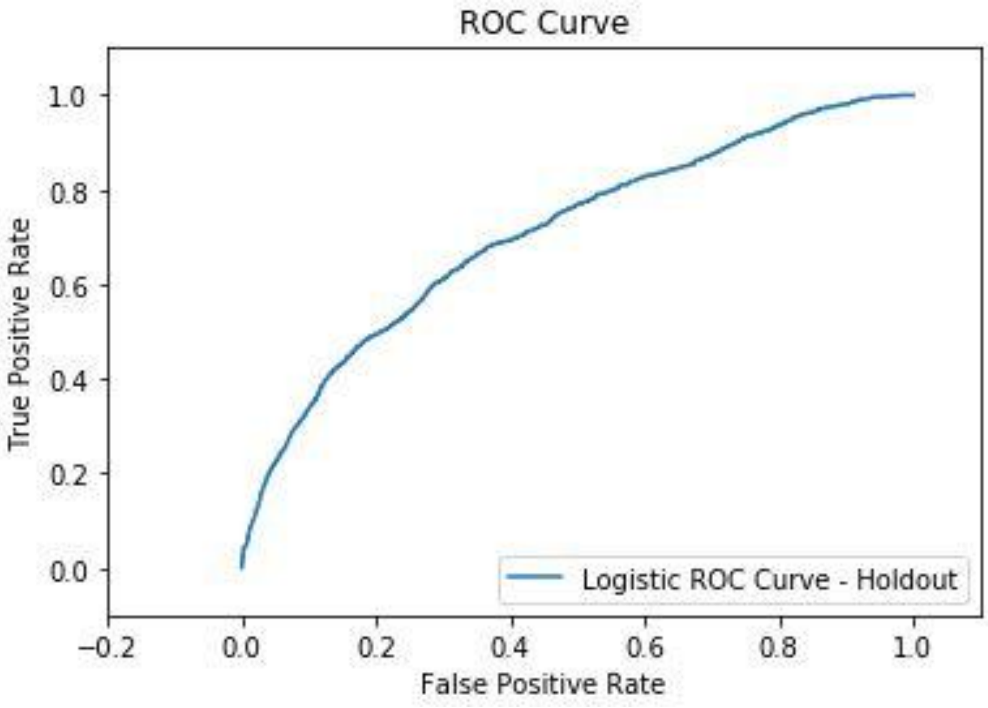
- Great accuracy! (.89)
- However, places observations almost entirely in the majority class
  - Not terribly useful

Bayesian Decision Boundary (p = 0.1)

$$\begin{bmatrix} 21468 & 9524 \\ 1443 & 2343 \end{bmatrix}$$

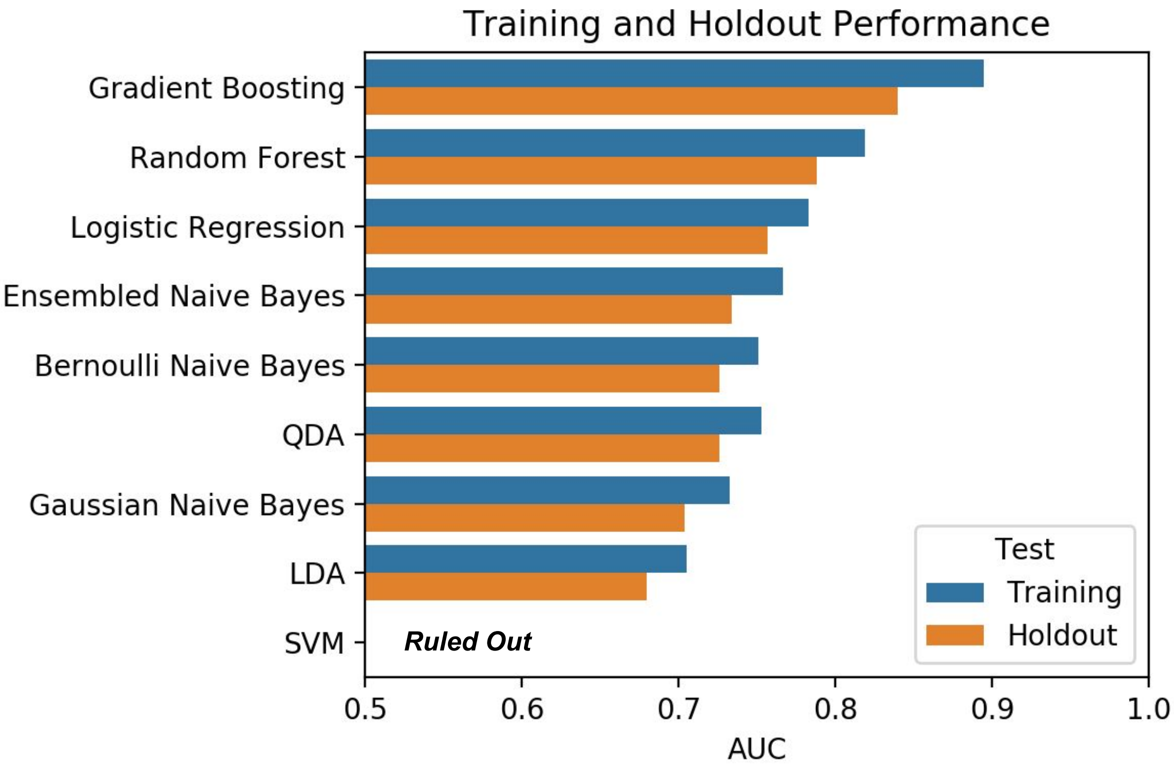
- Substantial decrease in accuracy (.69)
  - Much better classification to minority class

AUC Metric



- We will analyze the area under the Receiver Operating Characteristic (AUC)
- The ROC curve plots TPR against FPR as cut-off threshold varies
- Seek to maximize AUC

Model Comparison: AUC



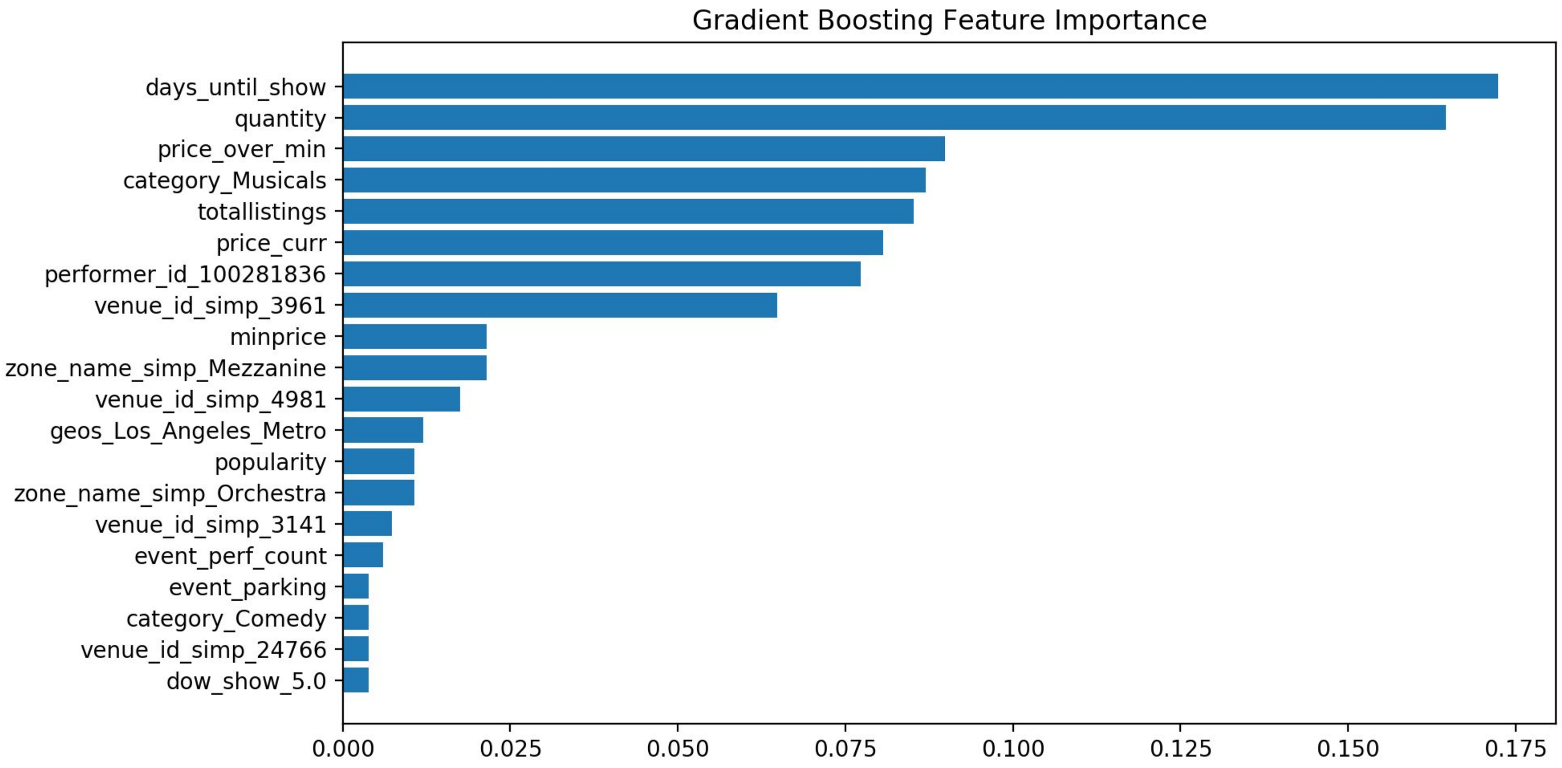
Tree-based models:

- Highest AUC in both training and holdout
- Prone to overfitting
- Long training times

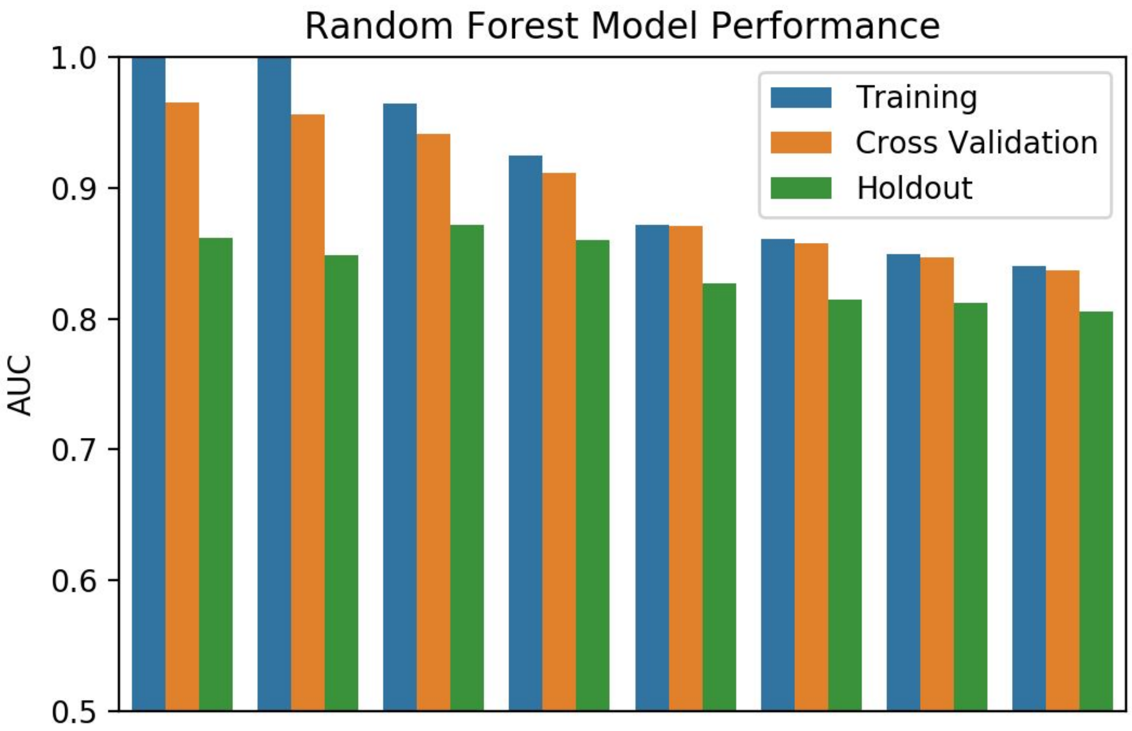
Probabilistic models:

- Reasonable AUC results
- Still some overfitting
- Much faster training

Gradient Boosting

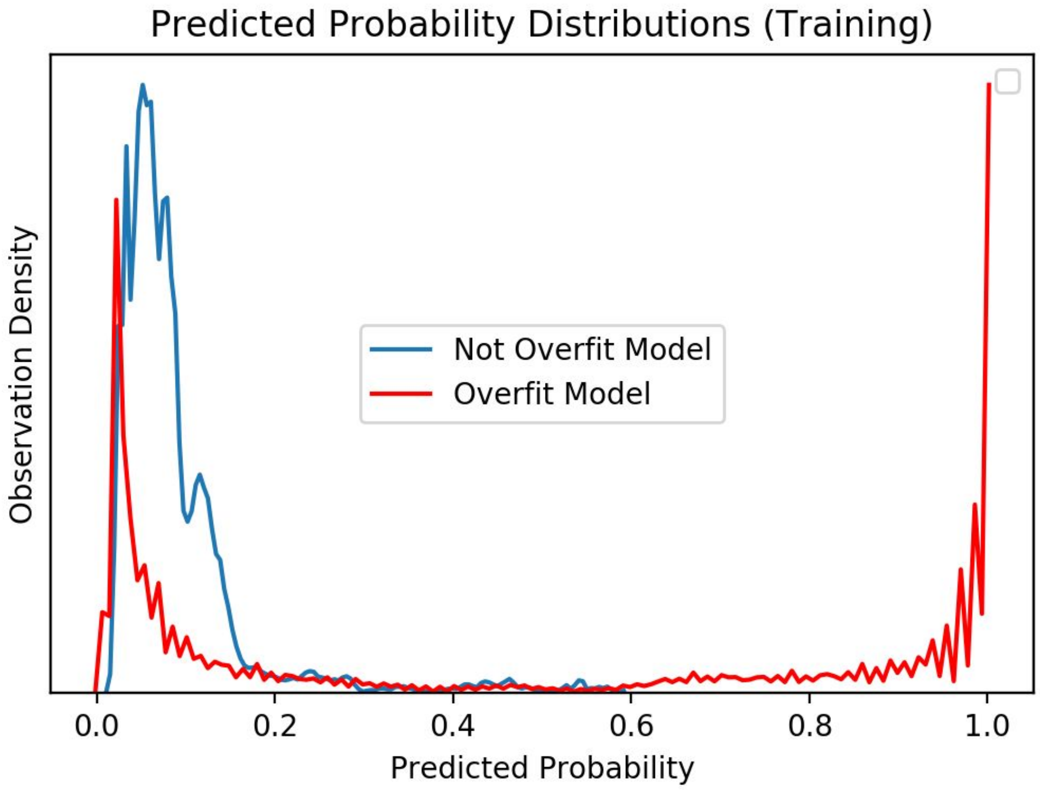


Random Forest



- Cross-validation scores were not always indicative of holdout performance
- Best-performing models on the holdout set (according to AUC) were still substantially overfit on the training set

Random Forest



- Overfit models tended to make predictions at either extreme of the probability spectrum
- A smoother distribution of predicted probabilities gives us more control down the line when we set threshold values to determine predicted classes

Naive Bayes with Mixed Features

Build Separate Models

Bernoulli NB	**497** features **0.75** training AUC **0.73** holdout AUC

Gaussian NB	**15** features **0.73** training AUC **0.70** holdout AUC

Estimate Probabilities

ID	Predicted Probabilities		Actual Class
	Bernoulli	Gaussian	
0	0.03	0.05	0
1	0.12	0.14	1
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Ensemble

Logistic Regression	**0.77** training AUC **0.73** holdout AUC

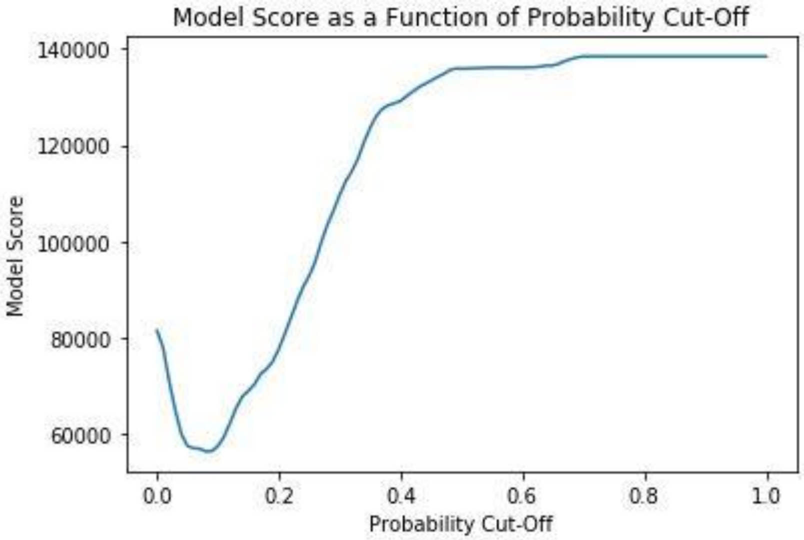
Loss Score Matrix: Use Case

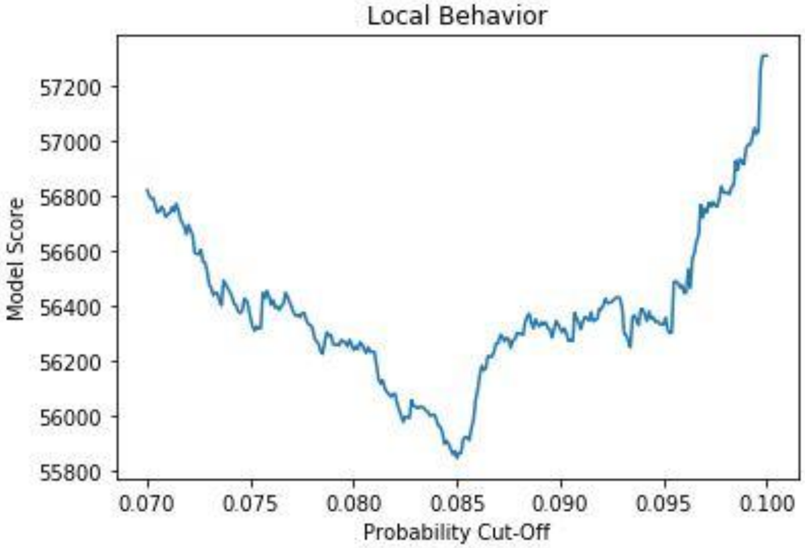
- User views a ticket listing. They are interested in buying the ticket. Our product predicts that the ticket will still be available tomorrow. User displays no sense of urgency in purchasing the ticket as a result. User returns tomorrow to find that the ticket listing is no longer available.
    - Frustrating user experience
- Need to institute harsh penalty on false negatives
    - 1 = listing unavailable following day
    - 0 = listing still available following day
- Custom Loss Matrix

$$LM = \begin{bmatrix} 0 & 1 \\ 15 & 0 \end{bmatrix}$$

- Model Score = Sum(Loss Matrix * Confusion Matrix)

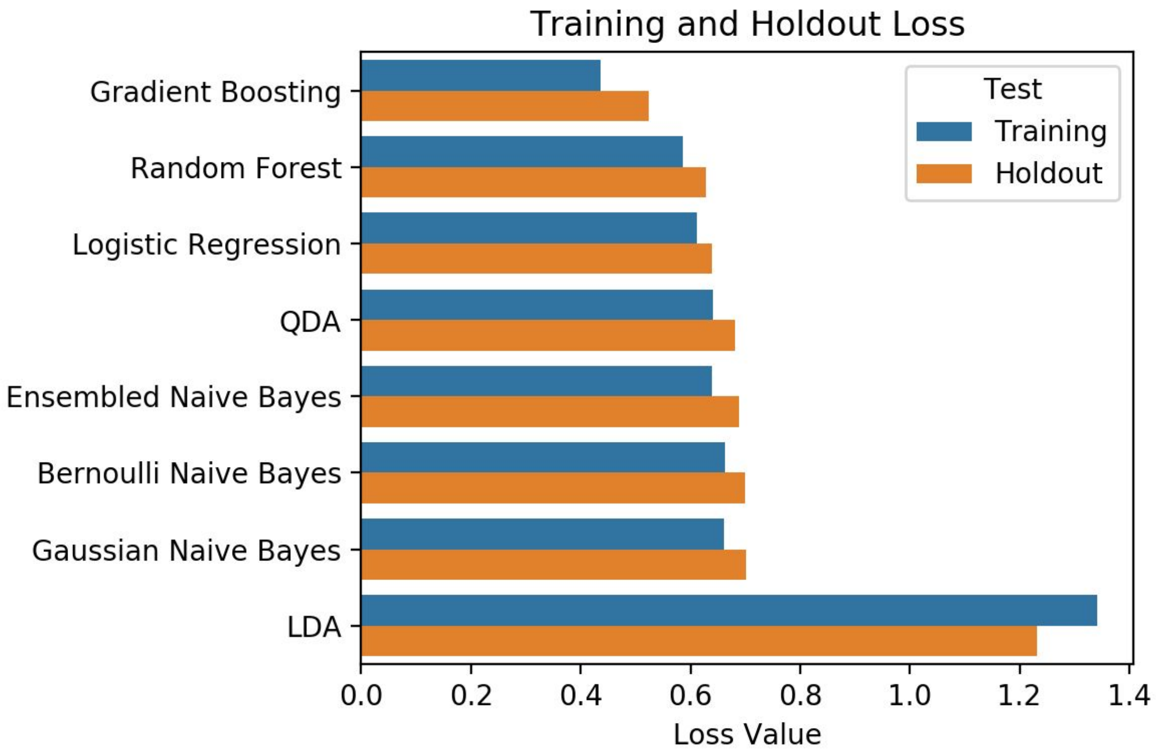
Model Score Function





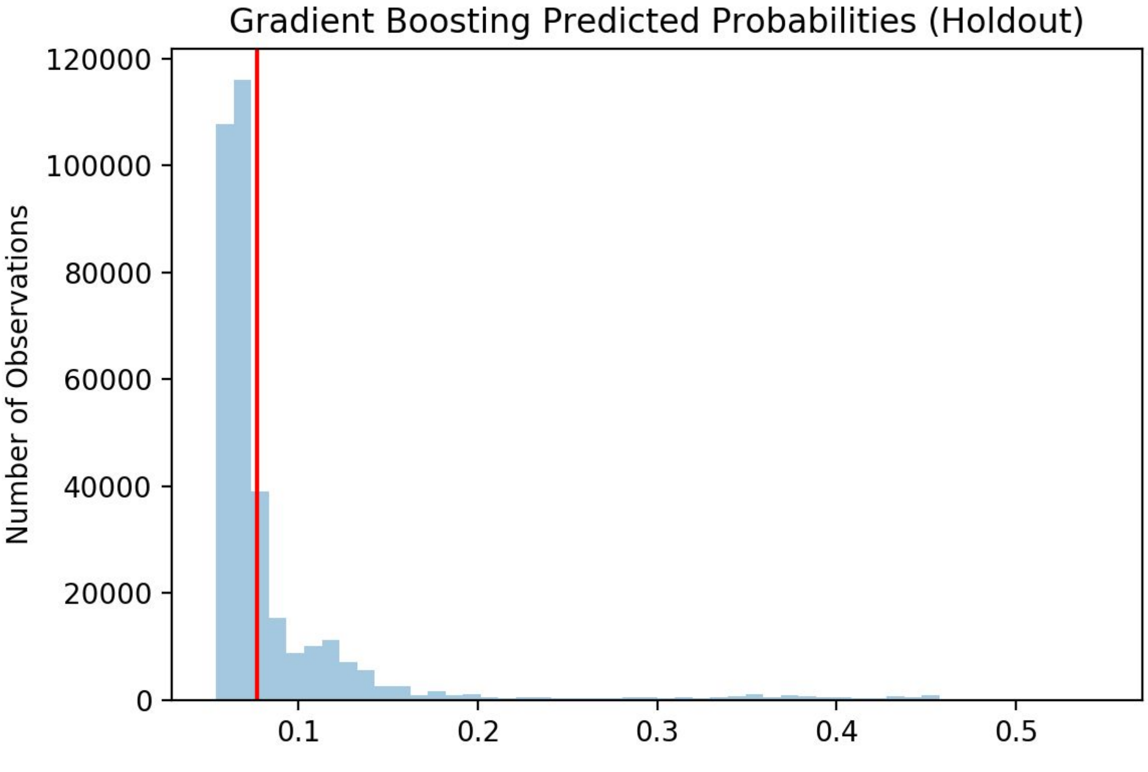
- Implemented recursive binary search to find optimal probability cut-off given custom loss matrix
  - Best model minimizes model score
  - Can run into issues due to local behavior

Model Results: Custom Loss Matrix



- We found similar results using our custom loss matrix -- gradient boosting was the most effective, but also displayed some overfitting
- Model tuning and prediction speed may also be a factor that could push us towards Naive Bayes, although predictions would likely be batched overnight rather than real-time

Gradient Boosting: Results



Predicted

Actual		**Not Sold**	**Sold**
	Not Sold	67%	24%
	Sold	2%	6%

Conclusion and Next Steps

Conclusions

- Imbalanced classification problems can be tricky (domain-specific tradeoff)
- Gear feature engineering towards model choice

Next Steps

- Add additional days of data and cross-validate using days as folds
- Re-tune individual models using loss-function minimization rather than ROC
- Explore features that account for availability and price of similar tickets
- Include time-lagged variables to account for time series effect
- Incorporate TicketMaster, SeatGeek, and Spotify data using fuzzy matching
- Tier price predictions (High Risk, Moderate Risk, Low Risk)

Thank You!