Big Data Solution for Gaming eCommerce Platform

Team 3:

Han Jeon, Jim Fang,

Tam Nguyen, WooJong Choi

Content

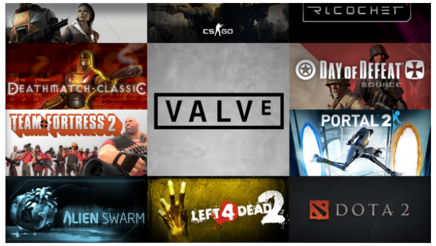
- Background & Business Use Case
- Data Pipeline
- Interesting Findings
- EDA & Feature Engineering
- Modeling: Regression & Classification
- Clustering, Recommendation & Graph Analysis
- Conclusion

Background & Objectives

Steam Background

Video game digital distribution service by Valve

- Valve: Steam parent company
- Valve has enjoyed enormous success as a game developer - Half-Life, Counter-Strike: Global Offensive, and Dota 2.



2003

First launched
Sep 2003

Growth

2017

2019

Largest digital distribution platform for PC gaming: ~75% of the market space (digital sales)

Users purchasing games through Steam totaled ~ US$4.3 billion (~ 18% of global PC game sales)

Over 34,000 games, over 95 million monthly active users

Project Objectives

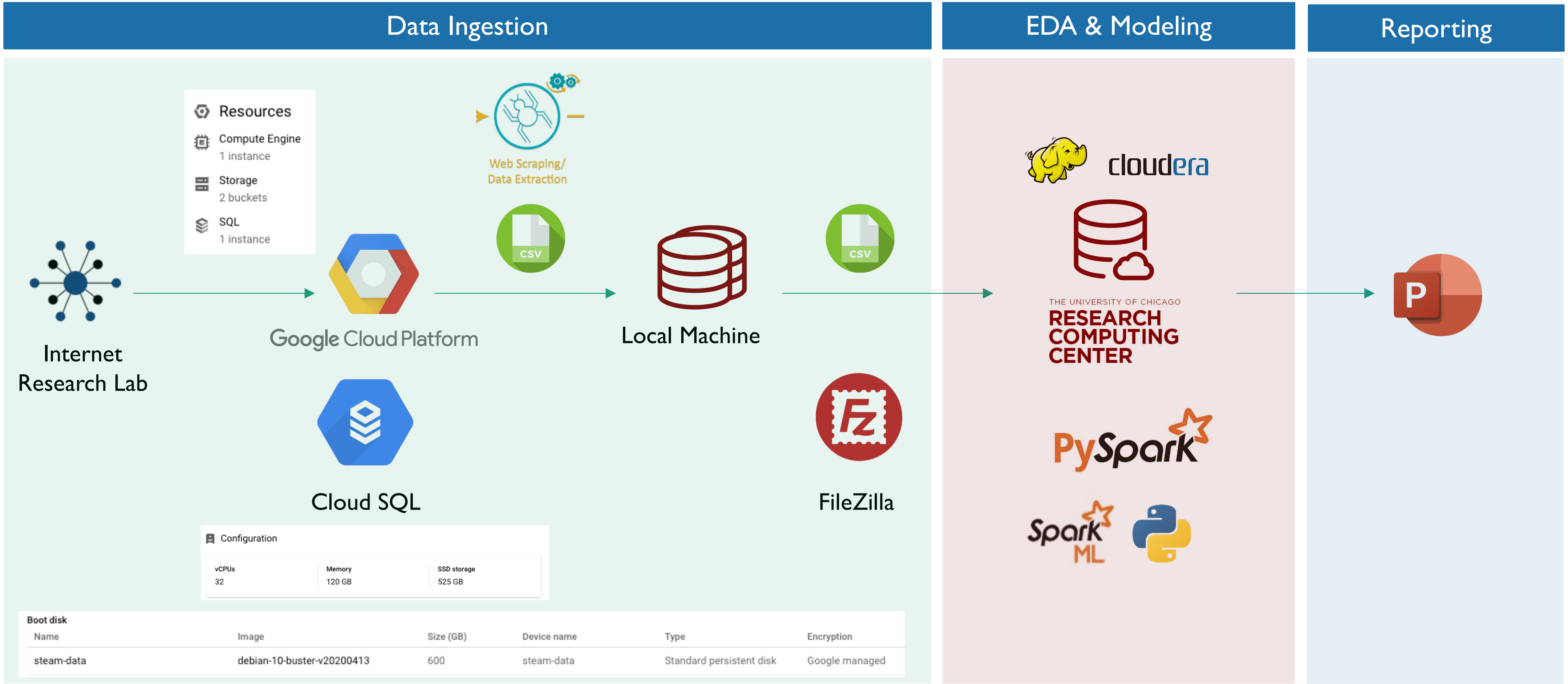
Understand gamer behavior and habits

- Play time, money spent on games
- Game genres, friends & groups
- and more.

Predict user playtime / build recommender using tree based regressors/classifiers

Group players with similar attributes, and make recommendations using ALS/Graph Algorithms

Data Pipeline



Dataset & Schema

Cover **10 years periods:**
2003 – 2013

.sql format | Size: **160Gb**

Total tables: **10**

Subset
Total : 24 GB

- Games2_df: **5GB**
- Player_summary: **5.5 GB**
- App_id_info: **1.4 MB**
- Groups : **3.8 GB**
- Friends : **9.2 GB**
- Game developer : 0.4 MB
- Game genre : 0.6 MB
- Game publisher : 0.4MB
- Achievement: **5.3MB**
- Steam Web Scraping: **20.6MB**

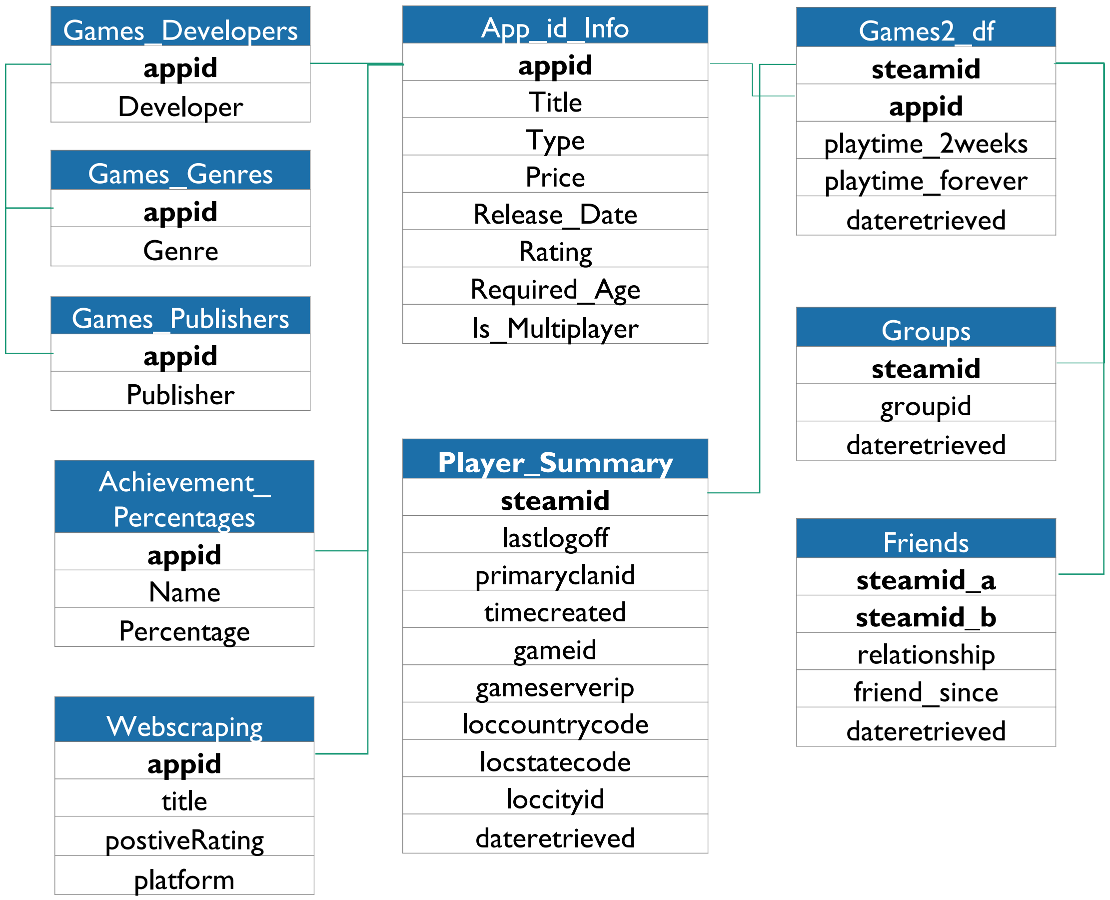
HOW DID WE SUBSET?

Except for games2_df and player_summary, the rest of data: fully downloaded

Games2_df: subset 100M observations

Player_summary: 10M unique users

Data Schema



The schema does not represent cardinality

EDA & Feature Engineering & Challenges

Join tables

App_id_Info
appid
Title
Type
Price
Release_Date
Rating
Required_Age
Is_Multiplayer

Games2_df
steamid
appid
playtime_2weeks
playtime_forever
dateretrieved

Games_Genres
appid
Genre

CHALLENGE:

- Data explodes to **7.6 B rows** if using normal join

SOLUTION:

- Column wise approach: make sure the # rows is the same (aggregate, one hot encoding)

Split columns

Player_Summary
steamid
lastlogoff
primaryclanid
timecreated
gameid
gameserverip
loccountrycode
locstatecode
loccityid
dateretrieved

CHALLENGE:

- Columns are not parsed correctly when export from Cloud SQL to CSV

SOLUTION:

- Based on the distribution of number of commas within each column, identify columns to split dataset into subsets with similar patterns
- Define logic based on the split dataset

Other feature engineering

Games_Developers
appid
Developer

Games_Publishers
appid
Publisher

Games_Genres
appid
Genre

Groups
steamid
groupid
deteretrieved

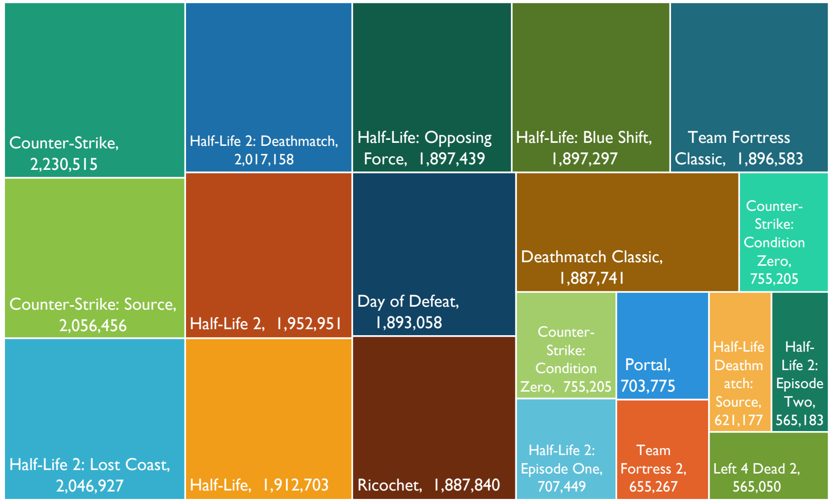
Friends
steamid_a
steamid_b
relationship
friend_since
dateretrieved

- Join tables
- Add new features, i.e. years of friendship, game release year
- Aggregation: total # groups, total # friends, count of hours played, count of games by genre, etc.
- Remove or impute missing values

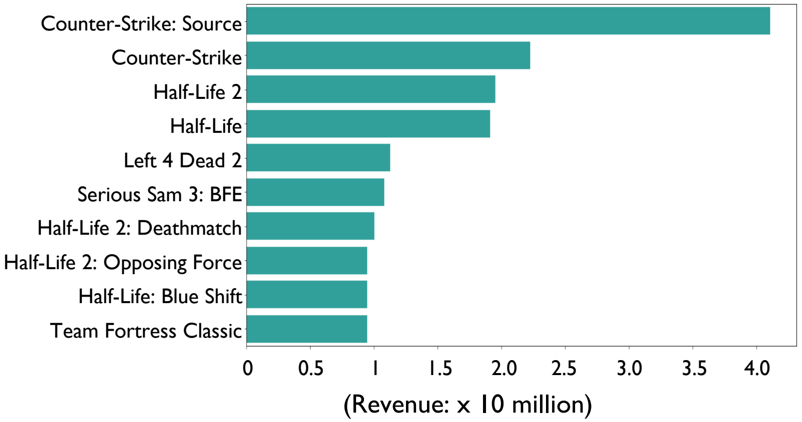
Total features: 39

Interesting Findings (1/2)

Top 20 games

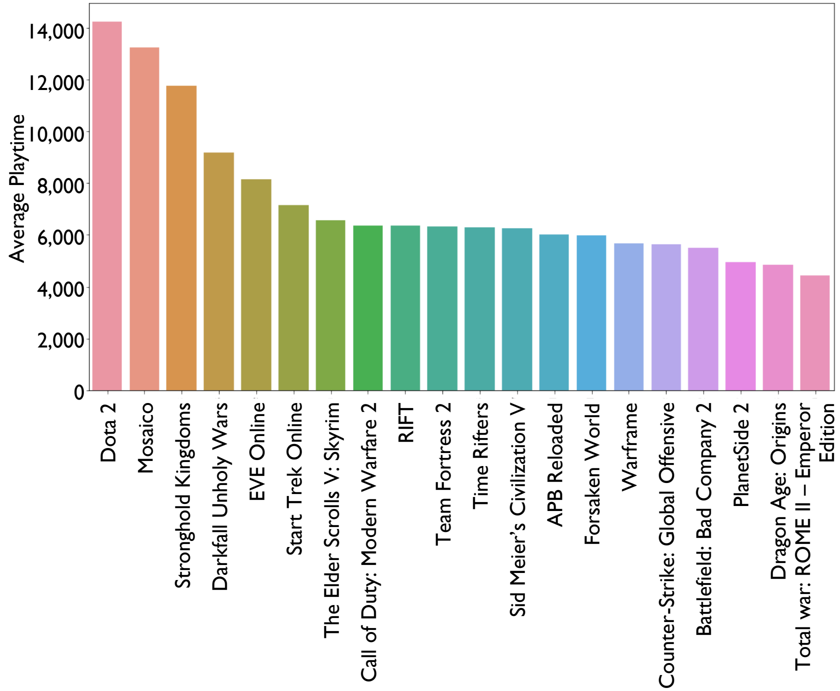


Revenue by game (x10 mil)



Over 10 years period: 2003 -2013

Most addicted game (play time)



AVG # friends/ player

9

AVG # groups/ player

1.6

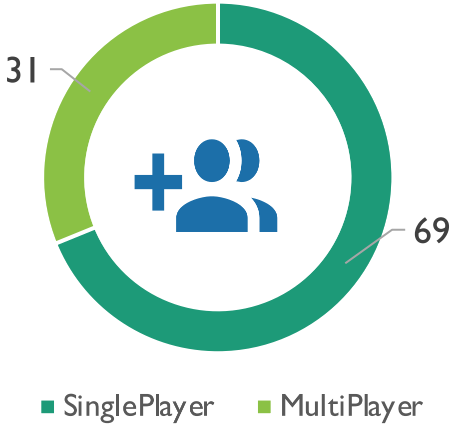
Base (N = ~3.2 million users)

games by Developer & Publisher

gamesDeveloper	count
Ubisoft - San Francisco	850
Dovetail Games	233
Feral Interactive (Mac)	227
SmiteWorks USA, LLC	185
Avalanche	158
Feral Interactive (Linux)	148
Relic Entertainment	130
KOEI TECMO GAMES CO., LTD.	120
Paradox Development Studio	112
Stainless Games	111

gamesPublisher	count
Ubisoft	376
SEGA	339
Dovetail Games - Trains	279
Paradox Interactive	247
Disney Interactive	227
Feral Interactive (Mac)	221
Activision	218
Degica	190
Nordic Games	164
Square Enix	148

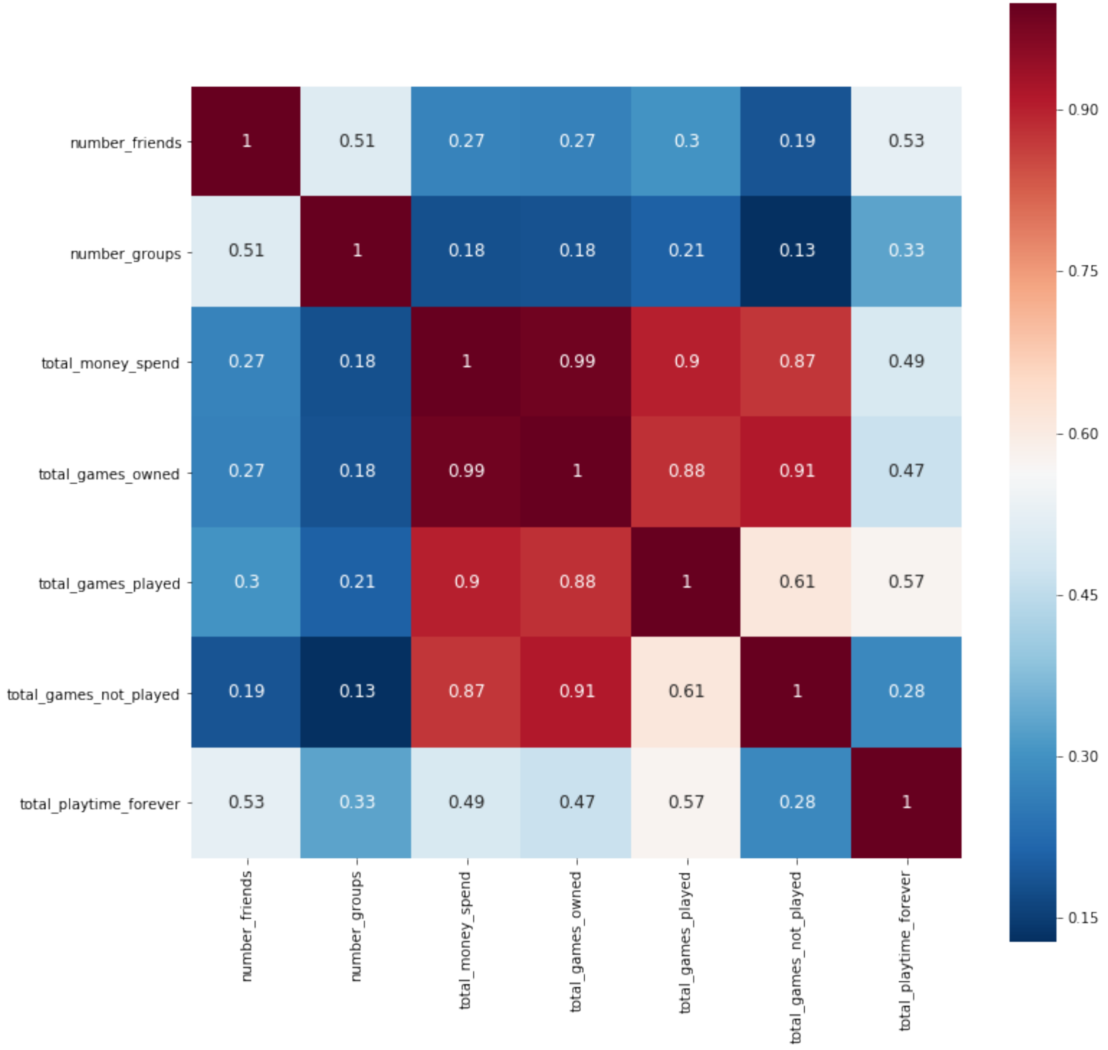
Game types



Top genres

	total
Action	1537.0
Indie	1523.0
Adventure	958.0
Strategy	834.0
Casual	737.0
RPG	582.0
Simulation	477.0
Racing	152.0
Free to Play	146.0
Sports	102.0
Early Access	93.0

Interesting Findings (2/2)



- Surprisingly, number of friends is not highly correlated with number of groups
- High number of games a player has does not necessarily translate to more playtime

Run on a subset data of 600k observations – keep distribution the same as original 100M observations

Regression & Classification

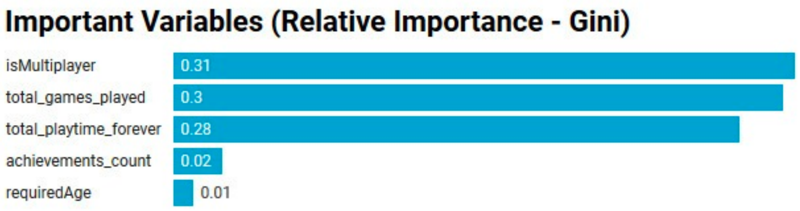
Regression | Playtime_forever

Linear Regression

```
grid = (ParamGridBuilder()
        .addGrid(lr.maxIter, [5, 10]) \
        .addGrid(lr.regParam, [0.01, 0.1]) \
        .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
        .build())
```

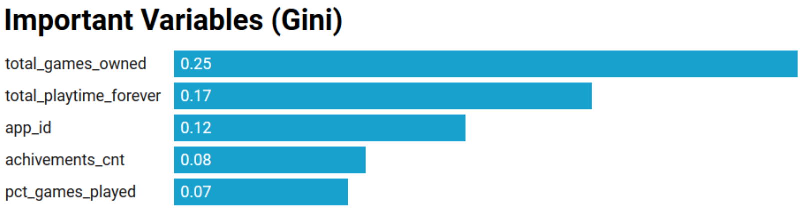
Decision tree

```
paramgrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [4,6,8,12])
             .addGrid(dt.maxBins, [250,500])
             .addGrid(dt.maxIter, [10])
             .addGrid(dt.criterion=['gini', 'entropy'])
             .build())
```



Gradient Boosting

```
paramGrid = (ParamGridBuilder()
             .addGrid(gbt.maxDepth, [4,5,6])
             .addGrid(gbt.maxBins, [250,500])
             .addGrid(gbt.maxIter, [10,20])
             .build())
```



Model	Train RMSE	Test RMSE	Test R^2 (%)
Linear Regression	6221.4	6209.1	5.2
Decision Trees	5980.7	6035.2	12.33
Gradient Boosting	5689.3	5762.4	20.8

- Difficulty in finding a relationship between the feature engineered variables and the overall playtime forever

Classification | Game Played or Not

Logistic

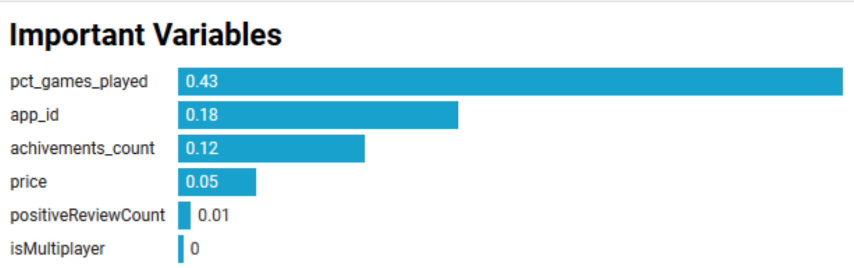
```
paramGrid = ParamGridBuilder()\
    .addGrid(lr.aggregationDepth,[2])\
    .addGrid(lr.elasticNetParam,[0.0, 0.5, 1.0])
    .addGrid(lr.maxIter,[10])\
    .addGrid(lr.regParam,[0.01, 2.0]) \
    .build()
```

Test Conf Matrix	Actual Yes	Actual No
Predicted Yes	6,434,587	1,909,805
Predicted No	1,904,835	3,316,757

Random Forest

```
paramGrid = {"max_depth": [3, None],
             "max_features": [10,20],
             "min_samples_split": [1,10],
             "min_samples_leaf": [1,10],
             "bootstrap": [True,False],
             "criterion": ["gini"],
             "n_estimators": [10,20,30]}
```

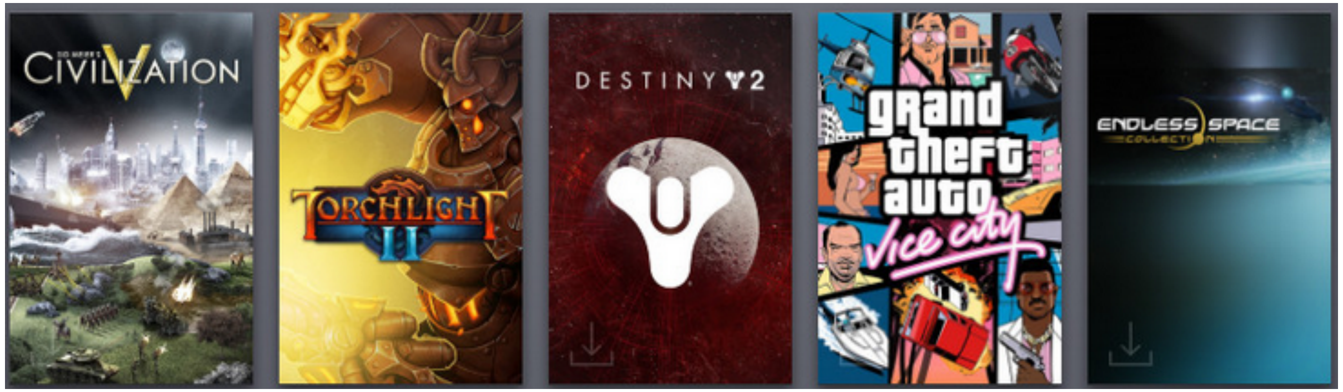
Test Conf Matrix	Actual Yes	Actual No
Predicted Yes	7,588,084	2,693,208
Predicted No	745,087	2,533,714



Model	Train Recall / Precision	Test Recall / Precision	Train F1	Test F1
Logistic Regression	79% / 76%	77% / 77%	0.775	0.771
Random Forest	91% / 74%	91% / 74%	0.819	0.815

Classification | Recommendation Workflow

Steam_ID: 224435345636



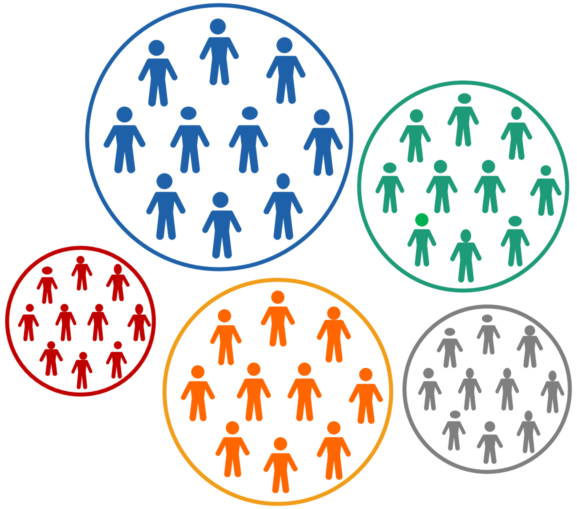
Predicted Probability: **0.97** **0.91** **0.87** 0.85 **0.84**

Start with a User ID

Score using ML pipeline on games not seen, based on customer play history and preferences

Generate Top 5 Steam Games Recommendation on Landing Page, sorted on predicted probability

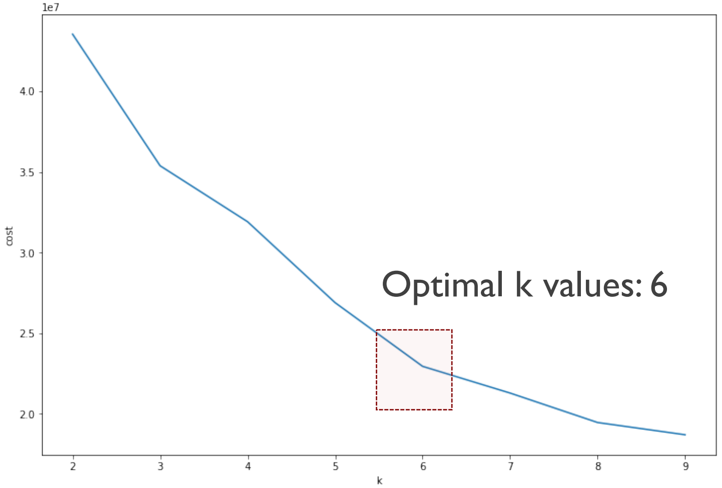
Clustering

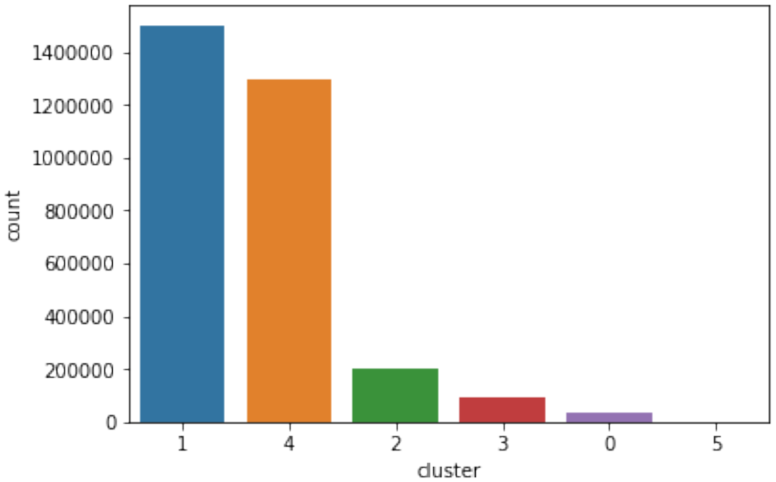


Clustering

Base (N = ~3.2 million users)

K – Means Clustering





2. Convert datetype to float (KMeans - Readable format)

In [13]:
```
for col in df1.columns:
    if col in feature_col:
        df1 = df1.withColumn(col,df1[col].cast('float'))
```

3. Vector Assemble

In [14]:
```
vecAssembler = VectorAssembler(inputCols=feature_col, outputCol="features", handleInvalid="keep")
df_kmeans = vecAssembler.transform(df1).select("steam_id", "features")
```

4. Scale data ¶

In [15]:
```
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True, withMean=False)
```

In [16]:
```
# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(df_kmeans)

# Normalize each feature to have unit standard deviation.
df_kmeans = scalerModel.transform(df_kmeans)
```

5. K.Means - select optimal k value

> Based on the graph, 6 should be the optimal number of clusters

In [18]:
```
%%time
cost = np.zeros(10)
for k in range(2,10):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("scaled_features").setPredictionCol("cluster")
    model = kmeans.fit(df_kmeans.sample(False,0.1, seed=911))
    cost[k] = model.computeCost(df_kmeans)
```

```
CPU times: user 118 ms, sys: 42.1 ms, total: 160 ms
Wall time: 1min 11s
```

In [19]:
```
fig, ax = plt.subplots(1,1, figsize =(12,8))
ax.plot(range(2,10),cost[2:15])
ax.set_xlabel('k')
ax.set_ylabel('cost')
```

Cluster profile

Base (N = ~3.2 million users)

The Ordinary (~ 48%)

Makes up half the total population and are regular, normal, Average Joe type of gamers

Money: ~ $ 200
Game owned: ~ 20 (does not play 50%)
Total Playtime: ~ 350 hrs
Game type preference: Multiplayer
Friends: 8
Group: 1.3
active

The Bandwagoner (~ 41%)

More likely to only play games that are popular, plays for a while and leaves Steam. Need based customers who are not hardcore gamers

Money: ~ $ 60
Game owned: ~ 8 (does not play 87%)
Total Playtime: ~ 13 hrs
Game type preference: Multi player
Friends: ~2
Group: ~0
in-active

The Old-Fashioned (~ 6.5%)

More likely to be older gamers who prefer older games

Money: ~ $ 1800
Game owned: ~ 130 (does not play 40%)
Total Playtime: ~ 1,600 hrs
Game type preference: Single Player
Friends: 20
Group: 4
active

The Savvy (~3%)

a.k.a smart shoppers. Takes full advantage of what's available on Steam. Doesn't own many games but plays a lot and is the most well connected

Money: ~ $ 500
Game owned: ~ 35 (does not play 40%)
Total Playtime: ~ 2,300 hrs
Game type preference: Multiplayer
Friends: 92
Group: 20
active

The Early Adopters (~1%)

Loyal, second most hardcore and may be relatively younger gamers compared to The Old-fashioned gamers, also more likely to play newer games

Money: ~ $ 5,000
Game owned: ~ 350 (does not play 50%)
Total Playtime: ~ 2,000 hrs
Game type preference: Single player
Friends: ~37
Group: ~8
active

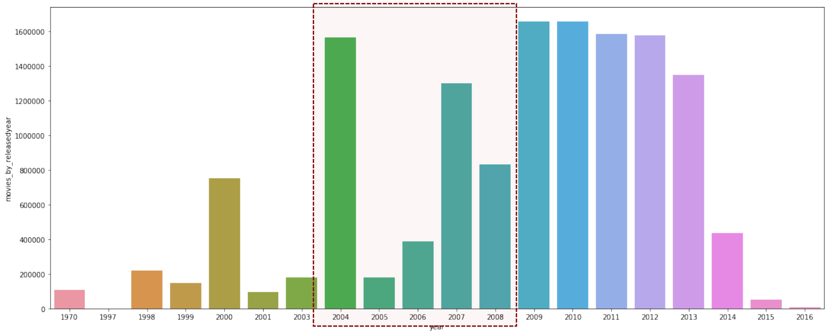
The VVIP (0.05%)

The most loyal, the most hardcore gamers, a.k.a the collectors. Games are a big part of their lives. Every game platform provider's dream.

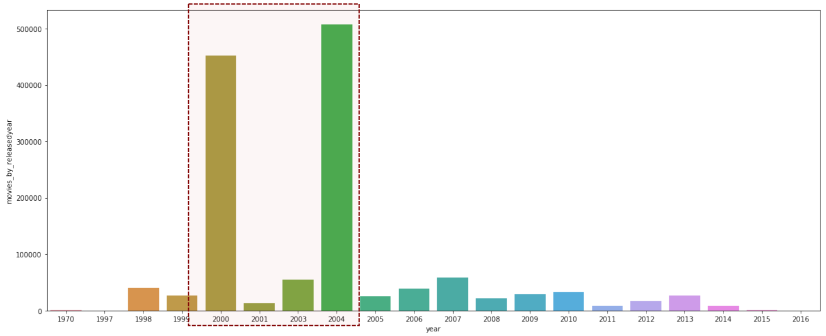
Money: ~ $ 15,000
Game owned: ~ 1,300 (does not play 73%)
Total Playtime: ~ 2,600 hrs
Game type preference: Single player
Friends: ~50
Group: ~17
active

Cluster – Released year game played

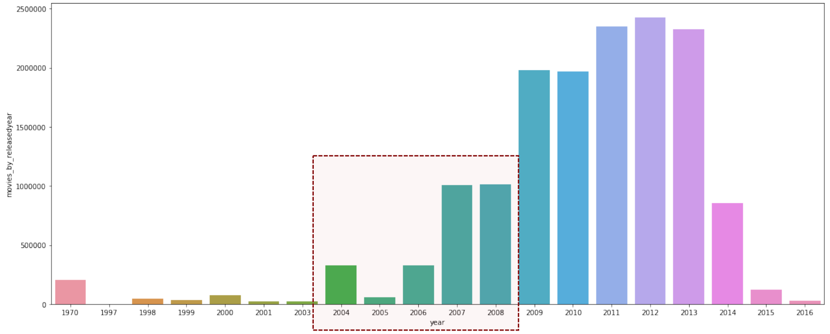
The Ordinary (~ 48%)



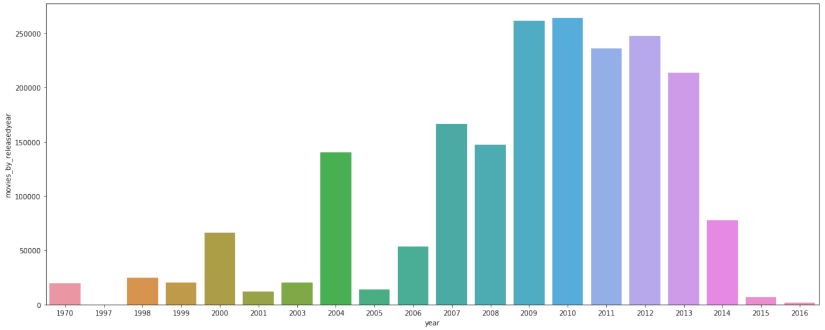
The Bandwagoner (~ 41%)



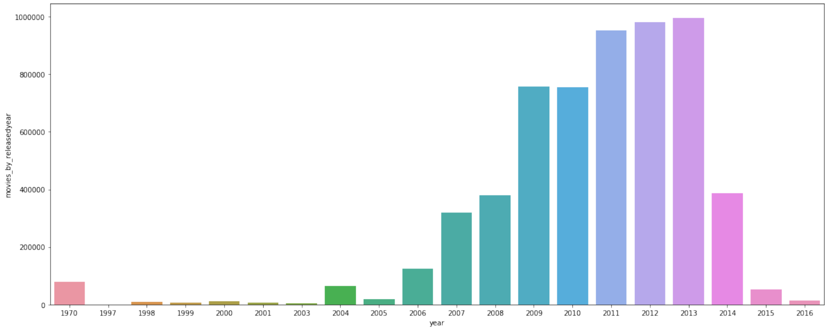
The Old-Fashioned (~ 6.5%)



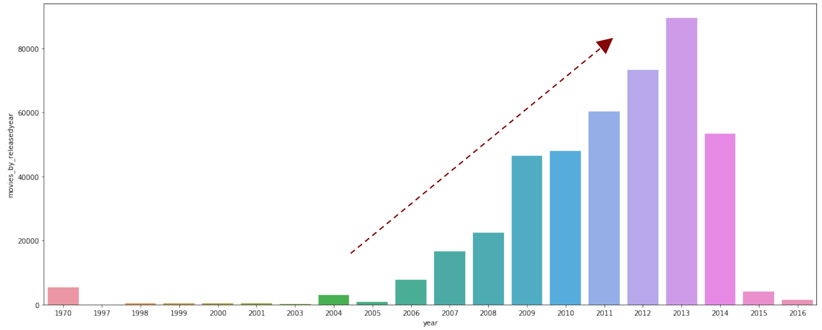
The Savvy (~3%)



The Early Adopters (~1%)



The VVIP (0.05%)



Top games by clusters

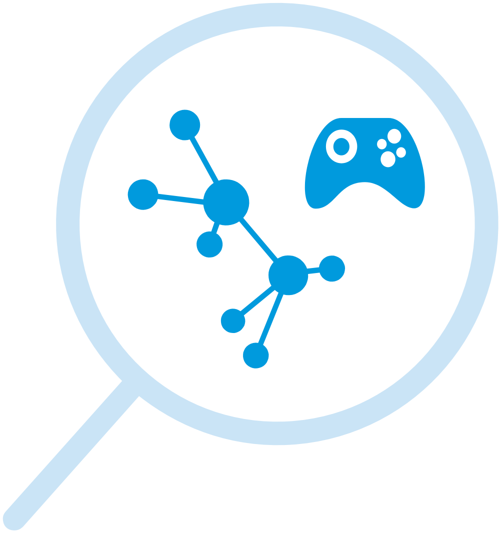
Common games

Valve

Different games

cluster	title	year	genre	price	multi	positive review%
The Ordinary	Sid Meier's Civilization V	2010	Strategy	29.99	Yes	20
	FTL: Faster Than Light	2012	Strategy	9.99		96
	Terraria	2011	Action	9.99	Yes	97
	Just Cause 2	2010	Action	14.99		90
The Bandwagoner	-	-	-	-	-	-
The Old-Fashioned	Day of Defeat: Source	2010	Action	9.99	Yes	88
The Savvy	-	-	-	-	-	-
The Early-Adopter	Dota 2	2013	Action	0	Yes	84
	Left 4 Dead	2008	Action	19.99	Yes	76
The VVIP	Trine 2: Complete Story	2013	Action	19.99	Yes	95
	Trine Enchanted Edition	2009	Action	14.99	Yes	95
	The Binding of Isaac	2011	Action	4.99		95
	Mark of the Ninja	2012	Action	14.99		96
	Batman: Arkham Asylum Game of the Year Edition	2010	Action	19.99		95
	Serious Sam 3: BFE	2011	Action	99.99	Yes	88
	Batman: Arkham City - Game of the Year Edition	2012	Action	19.99		94
	Space Pirates and Zombies	2011	Action	9.99		92

Recommendation Engine & Graph Analysis



Recommendation engine

Alternating Least Squares - ALS

Assumption

- If players like the game, they are likely to spend more time to play that game. Therefore, we chose total_playtime (hours) as the target feature for ALS model
- A random subset of the cluster (50 - 60% of the original size) could still represent the distribution of that cluster.
- Observations with total_playtime is equal to 0 are removed from the data before modeling, since:
    - we want to reduce the size of data and
    - don't want to bias the model with any games that no gamer played

Matrix Factorization

Total_Playtime Matrix (User × Game)

User \ Game	W	X	Y	Z
A		4.5	2.0	
B	4.0		3.5	
C		5.0		2.0
D		3.5	4.0	1.0

=

User Matrix

A	*1.2*	*0.8*
B	*1.4*	*0.9*
C	*1.5*	*1.0*
D	*1.2*	*0.8*

X

Game Matrix

W	X	Y	Z
1.5	*1.2*	*1.0*	*0.8*
1.7	*0.6*	*1.1*	*0.4*

RMSE - Cluster

Cluster	RMSE
The Ordinary	137.553989
The Bandwagoner	80.459183
The Old-Fashioned	100.762091
The Savvy	505.611293
The Early Adopter	97.831911
The VVIP	98.369387

- Train | test split 80%, 20%
- Using 10 iterations

Run ALS on the subset of The Ordinary and The Old-Fashioned (~ 6 -7 M observations)

Top 10 recommendation games per cluster

The Ordinary (~ 48%)

Game	year
Day of Defeat	2003
Deathmatch Classic	2001
Team Fortress Classic	1999
Counter-Strike	2000
Half-Life: Opposing Force	1999
Ricochet	2000
Darkfall Unholy Wars	2013
Half-Life	1998
SiN Episodes: Emergence	2006
Peggle Extreme	2007

The Bandwagoner (~ 41%)

Game	year
Team Fortress Classic	1999
Day of Defeat	2003
Deathmatch Classic	2001
Counter-Strike	2000
Half-Life: Opposing Force	1999
Ricochet	2000
Half-Life	1998
SiN Episodes: Emergence	2006
Peggle Extreme	2007
RIP - Trilogy	2007

The Old-Fashioned (~ 6.5%)

Game	year
Team Fortress Classic	1999
Day of Defeat	2003
Counter-Strike	2000
Deathmatch Classic	2001
Half-Life: Opposing Force	1999
Ricochet	2000
Darkfall Unholy Wars	2013
Half-Life	1998
Peggle Extreme	2007
SiN Episodes: Emergence	2006

The Savvy (~3%)

Game	year
Forsaken World	2011
Darkfall Unholy Wars	2013
Stronghold Kingdoms	2012
Making History II: The War of the World	2010
EVE Online	2010
Zen of Sudoku	2006
War of the Immortals	2012
Mabinogi	2012
Vindictus	2012
Civilization IV: Beyond the Sword	2007

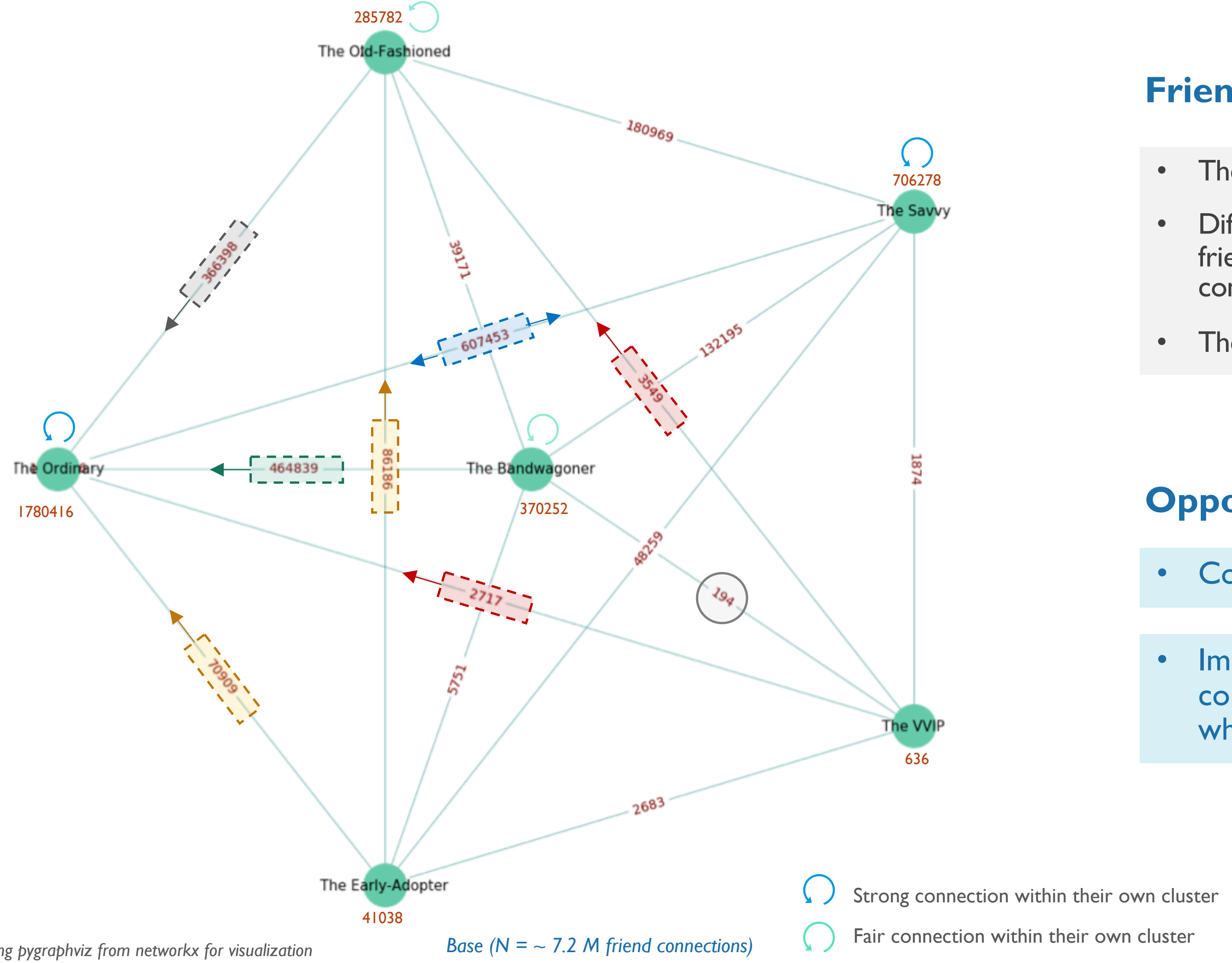
The Early Adopters (~1%)

Game	year
Stronghold Kingdoms	2012
Forsaken World	2011
Darkfall Unholy Wars	2013
Making History II: The War of the World	2010
EVE Online	2010
EVE Online	2012
Zen of Sudoku	2006
War of the Immortals	2012
Vindictus	2012
Maya LT (with Stingray)	2014

The VVIP (0.05%)

Game	year
Mosaico	2013
Fallout: New Vegas	2010
RIFT	2011
APB Reloaded	2011
You Need a Budget 4	2012
Dota 2	2013
Neverwinter	2013
FINAL FANTASY XIV: A Realm Reborn	2014
Team Fortress 2	2007
Call of Duty: Modern Warfare 2	2009

Graph | Friend connection between Clusters



Using pygraphviz from networkx for visualization

Base (N = ~ 7.2 M friend connections)

Friend connection between Clusters

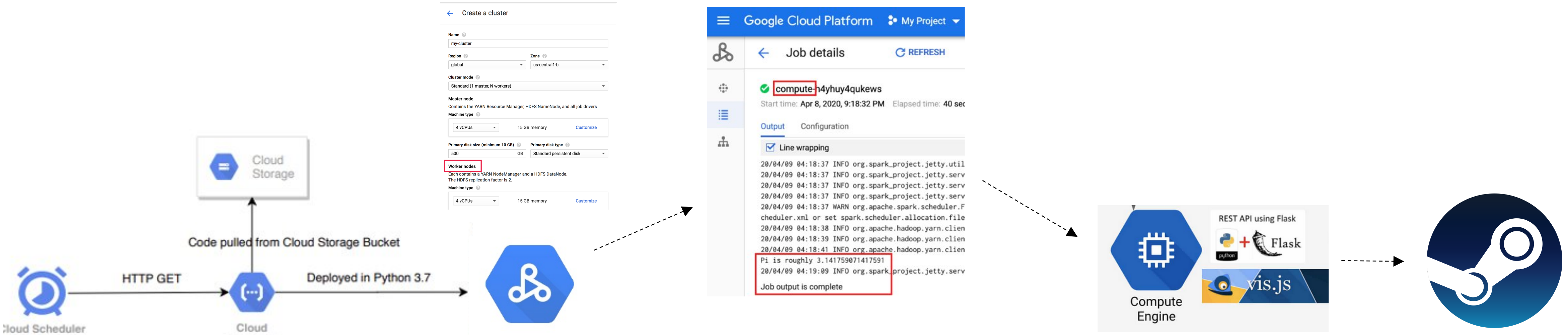
- The Ordinary and the Savvy mainly make friends within their clusters.
- Different from other clusters, the majority of VVIP and Early Adopter friends are outside of their clusters. The Old-Fashioned are their top connection
- Then Bandwagoner and the VVIP don't seem to get along well

Opportunity

- Convert the Early Adopter to the VVIP
- Improve recommendation engine and up-sell by identifying the common games of the VVIP, Early-Adopter and Old-Fashioned, who are in the same friend circle

Final Thoughts and Conclusions

Future Considerations: Pipeline & Scheduling



Run Scheduled Job at 6am everyday

Output Result to Cloud Storage and Visualize Recommendations in Flask

Validate Results and Connect to Steam platform by tapping into Compute Engine API

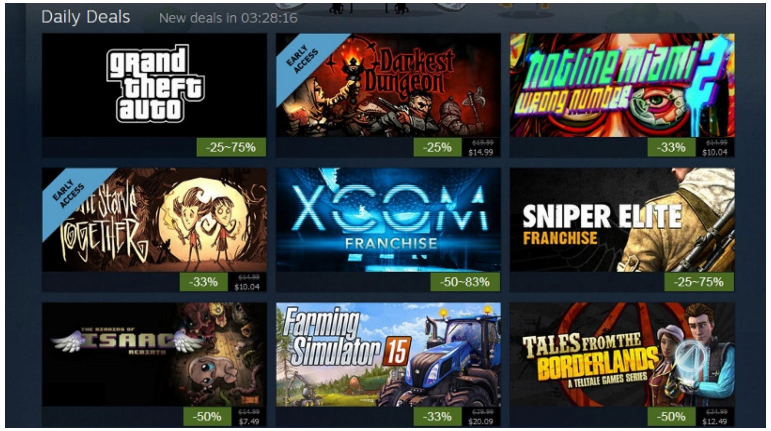
Size: 17gb storage per day with snapshot build (not incremental, history not needed)

Runtime: 15 min to score 10mil users on RandomForest Model + 10 mins to cluster user and run ALS recommender system, run as **batch process** once per day to output recommendation to Steam platform for each user = <u>25min total runtime per day</u> (on RCC)

Future Challenges:

- Scale from 10mil to 95mil users
- Scale to over >24GB YARN Memory (hard limit set by GCP) with 1 Master + 4 Worker nodes with highmem machines to decrease runtimes

Challenges | Lessons Learned | Next Steps



Spark H2O

Amazon EMR

Google Cloud Platform

Cloud Dataproc

Challenges

- Processing Big Data on Cloud is cost prohibitive
- Not enough computing capacity on RCC
- Improperly parsed data due to with multiple delimiters

Lessons learned

- Utilize compression techniques
- Clusters behave very differently. VVIP will eventually buy every game sold on the platform, while bandwagoners need to be marketed towards more strategically. Email campaigns and Daily Deals / Flash Sales should be employed with these groups in mind to maximize revenue
- We can utilize social networks in-platform to recommend games to users that are popular within their friend group or offer discounts/bundles on 2-player/4-player games (Savvy - Ordinary cluster relationship)

Next steps

- Migrate data to cloud platform such as AWS EMR or GCP Dataproc to scale/productionalize model
- Build pipeline to automate the entire process, add scheduling and build recommender UI
- Refactor code to use H2O Sparkling Water library for hyperparameter tuning of classification models
- Enhance current scraper to bring in specific user reviews for each game to incorporate as feature word embeddings to enhance classification, transactional data to see actual purchase prices to recommend prices during flash sales

Questions?


Thank you!