Technical Report:

# Overview:

## Pre-Work

* We are started gather all Kaggle repos we could find that had game related data. We ended up with 9 csv’s having various amounts of data in varying degrees of completeness. We eliminated one after the other until we settled on one with three files found in the Kaggle repo:

<https://www.kaggle.com/sidtwr/videogames-sales-dataset>

Three files containing video game data.

* PS4\_GamesSales.csv
* Video\_Games\_Sales\_as\_at\_22\_Dec\_2016.csv
* XboxOne\_GameSales.csv

The columns/data broke down as follows

RangeIndex: 16719 entries, 0 to 16718

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Name 16717 non-null object

1 Platform 16719 non-null object

2 Year\_of\_Release 16450 non-null float64

3 Genre 16717 non-null object

4 Publisher 16665 non-null object

5 NA\_Sales 16719 non-null float64

6 EU\_Sales 16719 non-null float64

7 JP\_Sales 16719 non-null float64

8 Other\_Sales 16719 non-null float64

9 Global\_Sales 16719 non-null float64

10 Critic\_Score 8137 non-null float64

11 Critic\_Count 8137 non-null float64

12 User\_Score 7590 non-null float64

13 User\_Count 7590 non-null float64

14 Developer 10096 non-null object

15 Rating 9950 non-null object

dtypes: float64(10), object(6)

We decided to use columns that contained the most complete data and ended up dropping columns 10-15 from video game sales since the other files do not contain matching columns.

## Extract:

* + Download the following datasets and save in the project folder:
    - PS4\_GamesSales.csv
    - Video\_Games\_Sales\_as\_at\_22\_Dec\_2016.csv
    - XboxOne\_GameSales.csv
  + We setup the notebook with the following dependencies needed for data manipulation and visualization
    - # !pip install plotly
    - import pandas as pd
    - import matplotlib.pyplot as plt
    - import numpy as np
    - import scipy.stats as st
    - import random
    - import seaborn as sns
    - import plotly.graph\_objects as go
    - from plotly.offline import init\_notebook\_mode, iplot
  + We used the “pd.read\_csv” function to get the csv’s into the notebook for further analysis.
  + Import the 3 data sets using pandas read\_csv
    - Encoding for the csv files = "ISO-8859-1" for the PS4 and Xbox csv’s

## Transform:

* + We started with add, remove and rename columns as necessary to get all of the columns for the 3 files to be the same
    - Add Platform column to PS4 and Xbox dataframes, so platform is set to PS4 and XOne, respectively
    - Create ID\_key column for each dataset, whose value is a combination of the Name and Platform columns
      * This is done to ensure that when appending data, we are not adding duplicates
    - Final column names and order for each file should be:
      * ID\_key
      * Name
      * Platform
      * Year\_of\_Release
      * Genre Publisher
      * NA\_Sales
      * EU\_Sales
      * JP\_Sales
      * Other\_Sales
      * Global\_Sales
  + We appended these three data sources to great one single data source of game sales data
  + Drop missing values from the dataset, and drop any duplicates from the dataset
  + Rename columns to match the ERD of our SQL database
  + Set data to 3rd normal form by creating separate deduplicated tables for Platform, Genre, Publisher, Year of Release and Sales columns
    - Platform, Genre, Publisher, and Year of Release table only includes the column itself, and the index for the table is set to an ID for the value
    - Sales data includes NA, EU, JP, Other and Global Sales columns
      * game\_id is the index for the Sales table
  + We need to re-consolidate indices from each of the above tables into a single table by merging them to create the games\_main table
    - Merge these tables using a left join onto the overall games dataset (FVGData), then dropped unnecessary columns
    - Final columns:
      * game\_name
      * platform\_id
      * genre\_id
      * publisher\_id
      * rel\_year\_id
    - Set the index to game\_id

## Load

* When you get to the section labeled “Load” you will have to leave the notebook to setup the DB Env.
  + First, we needed to make sure we create a database called “ETL\_db” in PGAdmin.
    - Make sure you have an account with the appropriate credentials to access db, create tables and update information in that DB.
  + Right-click on the “ETL\_DB” and select the “Query Tool”
  + Copy/paste the contents of the “ETLProject.sql” into the query tool and run to create the tables and appropriate joins between them.
  + We thencreate a connection from the notebook to the newly created “ETL\_db”
  + Then we can push data from the sales, release\_year, genre, publisher, platform and games\_main dataframes to the SQL DB using the “.to\_sql” command.
    - games\_main needs to be done after the others because it has dependencies on them
    - pay attention to the index Boolean to avoid “null indices” errors
  + Check that the data has made it to the appropriate table by performing a from the “pd .read\_sql\_query” from the DB

## Visualization and Front End

### Visualization:

* + Use matplotlib, iplot and seaborn for the visualization
  + We visualize the following subsets of data:
    - Global sales by Platform
    - Top 25 Global sales by Publisher
    - Regional sales by Platform
  + The following libraries are used for visualization
    - import seaborn as sns
    - import plotly.graph\_objects as gofrom plotly.offline
    - import init\_notebook\_mode, iplotimport matplotlib.pyplot as plt
  + Read the table created previously named “Final\_VGData.csv” which houses our total dataset
  + Use .groupby and .sum to create the subsets of data that will be charted
  + Use plotly to create graphs for the above subsets

### Front End Development:

* + The front-end data dashboard for this project was made using several different bootstrap elements including cards, jumbotron, grids, and a fixed footer and navbar with dropdown buttons. The entire site is responsive to the screen size of the user thanks to bootstrap.
  + The index.html file consists of all images displayed on cards, a jumbotron to give a summary of the project, and a navbar and footer.
  + Data.html is a responsive table from bootstrap which allows the user to scroll over specific info based on their screen size. This was done with the following code:
    - * string = FVGData\_clean\_df.to\_html(index=False, classes ="table")

with open('table.html', 'w', encoding='utf-8') as html\_table:

html\_table.write(string)

* + This code was necessary because trying to export to an html file directly from the dataframe resulted in errors due to character type which were not being encoded properly.
  + Comp.html is a card grid with 4 cards displayed in a 2x2 grid structure. This is done for even spacing and aesthetic, as well as on small screens the images will all appear in a vertical line.
  + There are 6 pages which are found in the plots drop down, these are the eu.html, jp.html, na.html, row.html, total\_sales.html, and top25.html. These pages all have a card, jumbotron, and navbar/footer. The card is broken up into 2 columns, one contains the image and the other contains the analysis body text.
  + All 8 .html files have the same basic skeleton as far as navbar, footer, and jumbotron, the footer is absolute on some pages and fixed on others, this allows the bar to fit the page better on vertically long or short pages. All values can be changed to absolute and the page will be virtually unchanged, fixed on all files will make the data, comp, and index files look improper.
  + We exported images of our analysis in the jupyter notebooks, saved them as .png files and linked to their image location in the folder architecture to have the images displayed properly.