The automated detection of fraudulent peer-to-peer transactions in massively multiplayer online economies

Student Number: 1902055

[1] [2]

Abstract—**Massively Multiplayer Online games are a hugely popular and successful genre within the gaming industry. These games allow players to trade items between eachother, but some players choose to buy and sell in-game items for real-world money, known as Real Money Trading. This leads to people preferring to buy from illicit sources rather than through the game itself, resulting in a loss of revenue for the developers. Additionally, this practice enables people to make money through methods such as botting, account theft, or cheating.**

Index Terms—**Anomaly Detection, Massively Multiplayer Online, Real Money Trading**

I. Introduction

REAL-MONEY Trading (RMT) is the practice of buying and selling virtual items and currency within massively multiplayer online (MMO) video games. This practice often violates the games' Terms of Service or Code of Conduct[3][32], and as a result, players who engage in RMT are often at risk of being banned from the game. RMT has negative effects on the game's economy and community, and it is therefore discouraged by game developers and players alike. The solution to this issue is anomaly detection algorithms. These are useful in serveral facets of computer science and this paper will discuss the use of these algorithms and compare different anomaly detection algorithms which utilise Standard Deviation. This paper is working to address the issues of RMT in video games.

II. Literature Review

A. Anomaly Detection for RMT

Anomaly detection is a technique used to help address the issue of RMT in MMOs[33][1]. By using computers to flag in-game transactions automatically for human review, it is possible to identify and prevent RMT activity in a way where humans don't have to analyse the all the data manually by hand. Preventing RMT helps to protect the game's economy and maintain a fair and balanced playing experience for all players. However, it is important to note that the effectiveness of this approach will depend on the specific details of the game and the methods used for detecting anomalies.

[1]Source code for all algorithms is available on GitHub: https://github.com/cdgamedev/dissertation

[2]Source LaTeXis available on Falmouth GitHub: https://github.falmouth.ac.uk/Games-Academy-Student-Work-22-23/1902055-comp3xx-dissertation

Fujita et al. propose a method for addressing the issue of RMT in MMO games, which involves identifying suspects, verifying their involvement in RMT activity, and banning their accounts[14]. They also classify RMT players into three categories:

- *Sellers* are those who sell the virtual property to players for real-world money.
- *Earners* acquire virtual property (currency and items) from non-player characters (NPCs) and real players.
- *Collectors* convey virtual property from earners to sellers.

Fujita et al. manually classified a set of players and then used an algorithm proposed by Newman and Girvan[15] to extract communities and further identify players. They ranked players based on the number of times they traded currency, the number of trades they made in total, and the total volume of currency traded.

Fujita et al. argue that RMT is harmful to both the game's economy and its players, as it is often associated with other illicit activities such as cheating, botting, and account theft. RMT can also drive away legitimate players who become frustrated with the problems it causes, and it can discourage new players from joining the game[27]. Overall, Fujita et al. believe that RMT is a serious issue that needs to be addressed to protect the integrity and health of MMO games. Han et al. and Sifa et al. back up Fujita's claims adding that "cheating in MMOs often reduces the shelf life of the game" by causing people to abandon it[17][27].

B. Types of Anomaly

Anomalous data can typically be categorized in three different ways[2][6]. These categories are:

- **Point Anomaly** where a data point is unusually out of range.
- **Contextual Anomaly** (or collective anomaly[31]) where sometimes, a data point which seems anomalous is actually within range depending on a varying factors.
- **Collective Anomaly** is where multiple data points are out of range, but when considered individually, are not out of range.

Understanding these categories can help researchers identify and address potential issues in their data. By detecting anomalies, it may be possible to uncover hidden patterns or trends and improve the accuracy and reliability of data-driven systems and models.

C. Machine Learning

Due to the nature of anomaly detection is often done with Machine Learning (ML) algorithms[23][20][38]. These algorithms are split into two categories, supervised and unsupervised[23]. Supervised methods ”require a labeled training set containing both normal and anomalous samples”, whereas, unsupervised methods don’t require training data as they assume a fraction of data points are anomalous[23].

Nassif et al. state that they ”recommend that researchers conduct more research on ML studies of anomaly detection to gain more evidence on ML model performance and efficiency” following their own review of prior research. They mention that unsupervised datasets have a greater number of research papers than supervised datasets. They also identify 29 different machine learning models for detecting anomalies[21].

D. Nearest-neighbor Based Algorithms

Local Outlier Factor (LOF) works by getting each object in the dataset to ”indicate its [own] degree of outlier-ness”[7].

k-Nearest Neighbor (kNN) is a supervised learning algorithm based on Nearest Neighbor and it’s discussed frequently within anomaly detection and ties in with lazy learning algorithms. kNN functions by finding the K number of the nearest points and assigns the data point to a label that is most suited, this can be used for anomaly detection if the assignment is different to how the data is already labelled[10].

Lazy learning algorithms work by[35]:

- Storing all training data, and deferring processing until queries are given the required replies.
- Answering queries by combining the training data.
- After replying, the answer and any results are discarded.

Many improvements to the base kNN algorithm have been developed by other researchers. Muti-label kNN (ML-kNN) exists to provide lazy learning to problems such as text categorization or bioinformatics. This approach works by [37].

E. Graph-based Anomaly Detection

Graph-based anomaly detection algorithms ”[detect] patterns (substructures) within graphs” where a ”substructure is a connected subgraph in the overall graph”[22]. Noble et al. utilized their anomaly detection, Subdue, for intrusion detection, utilizing data which ”contained 41 features describing the connection” and labelling them as ”one of 37 different attack types” or ”normal”. This is a form of unsupverised learning[22]. Graph-based anomaly detection works well with large datasets and is often used for collective anomalies[12].

Davis et al. discuss the use of Yet Another Graph-based Anomaly Detection Algoritm (YAGADA). They find that in other methods of anomaly detection, single time events are detected easily, but anomalous patterns of data which are anomalous in context such as ”an airport technician who regularly hangs around in the baggage handling area, or a clerk who is spending an unusually long time on their own in the cash room” would not normally be detected. They conclude that YAGADA works well for static graphs and suggest using it in ”forensic analysis of graph transaction databases”. They also conclude that LOF[7] is more suitable to numeric anomaly detection.

F. Autoencoders

Misra et al. focus on an autoencoder based model for detecting fraudulent transactions within the financial domain, specifically credit cards[20]. They propose a two stage method where ”a lower dimension of features are extracted from the input” before ”a model decides whether the transaction is fraud or not”. The first stage utilizes an autoencoder and the final stage utilizes a classification algorithm. ”Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion”[4]. They state that having too many features can cause classification algorithms to ”run poorly” and that the ”data becomes very expensive [when] time complexity is concerned” and is resolved by reducing the number of features. Misra defined features or attributes as parts of a whole data point and that autoencoders can extract these features nicely on any dataset. For credit card fraud, some of these attributes are, time/amount/mode/location of transaction, a user’s account number, a user’s age.

Deep Autoencoding Gaussian Mixture Model (DAGMM) works by preserving ”information of an input sample in a low-dimensional space” and then performs a ”Gaussian Mixture Model over the learned low-dimensional space” before utilizing ”a sub-network called estimation network that takes the low-dimensional input from the compression network and outputs mixture membership prediction for each sample”[38].

G. Standard Deviation

Yang et al. discuss the use of standard deviation (SD) within anomaly detection[36]. However, they note that simple datasets can cause false positives[24] and researchers misuse SD methods frequently[28]. The main issue with SD algorithms is that outliers influence the standard deviation and averages for the dataset.

To solve these issues Yang set out to develop a modified SD algorithm which could be relied on to give more accurate results. Two-stage thresholding (2T)[36] works similarly to Clever Standard Deviation (Clever SD)[9] by utilizing recursion. 2T works by recursively removing outliers one at a time and is the most accurate method of SD algorithms based on Yang’s findings.

H. Anomaly Detection for Other Datasets

Bergman and Hoshen talk about anomaly detection for general data and classification of anomalies using AI. Examples, of where this is used, are for fraudulent credit transactions and detecting cyber attacks amongst others[5]. They state that ”classification-based methods have dominated supervised anomaly detection”, these are methods of anomaly detection which utilise a classifier trained by an ML model. They further discuss the use of the following semi-supervised methods; one-class classification and geometric-transformation classification. They make a comparison between SVMs[26], LOF[7] and DAGMM[38].

Similarly, Misra and Sadineni investigate the use of anomaly detection within Credit Card transactions[20][25].

III. Research Questions

The questions this paper sets out to answer are:

- Which anomaly detection algorithm is the most performant for peer-to-peer transactions within MMOs?
- Which algorithm will detect the most anomalous data points?

By answering these, game developers working on MMO titles can easily identify and choose a method that suits the needs for their game. This will help reduce the revenue loss caused by RMT which could be millions[11].

IV. Hypotheses

A. Which anomaly detection algorithm is the most performant for peer-to-peer transactions within MMOs?

Hypothesis: Following research conducted, for data specifically from Lost Ark, a simple method which doesn't require training data is most likely to be best. The 2T algorithm[36] could be beneficial for a dataset which has a small number of features as the data from Lost Ark requires only 2 features, a more complex algorithm likely isn't required.
Data Source: The average compute time of each algorithm compared.

B. Which algorithm will detect the most anomalous data points?

Hypothesis: Due to the fact that the dataset used cannot be labelled, it will be difficult to quantify if the algorithms function to detect all fraudulent transactions in the dataset. Instead by looking at the quantity, rather than quality, we can assess if an algorithm successfully identifies more or less anomalous transactions than another algorithm. I believe that CleverSD will find the most anomalous transactions due to it processing the data recursively.
Data Source: The number of anomalous transactions found by each algorithm.

V. Computing Artefact

A. Game Background

For this research, the primary focus will be on Smilegate and Amazon Games' Fantasy MMORPG; Lost Ark[29] with a specific focus on its Gem system. Gems are collected by players during gameplay and can be sold to others using the in-game marketplace. Each Gem has different attributes, such as, Level, Name, Tier, Gem Effect, Sale Price and Sale Date.

- *Level* is a number between 1 and 10 which impact the effect of the Gem. Level n gems are created by merging 3 Level $n-1$ gems. A Level 10 gem should always cost more than a Level 1 gem as it takes 3^{10} Level 1 gems to make a single Level 10 gem.
- *Name* is a tag given to the gem and doesn't effect the gem. This means that name shouldn't affect the price.
- *Tier* is a number between 1 and 3 for all items in Lost Ark, however, gems only exist at the start of Tier 2 meaning all gems will either be tagged with Tier 2 or Tier 3. Tier 3 gems get unlocked at Item Level 1302 and offer higher effects than their Tier 2 counterparts. This means that a Level 1 Tier 2 gem should cost less than a Level 1 Tier 3 gem, however, a Level 10 Tier 2 gem is likely to cost more than a Level 1 Tier 3 gem.
- *Gem Effect* is the overall effect of the gem. This will either, increase the damage or decrease the cooldown of an ability in the game. Various gem effects will be more favourable based on the specific character build a player chooses. Favourable gem choices depend on the games meta and players often utilize a service like Maxroll[19] to get the best build for their characters class. Gem Effect can have an impact on the price of a gem, however, this data is near impossible to get without direct access to the internal marketplace database. However, gems can also be rerolled for silver which means that there shouldn't be a huge price disparity between two Level 1 or two Level 10 gems of the same tier.

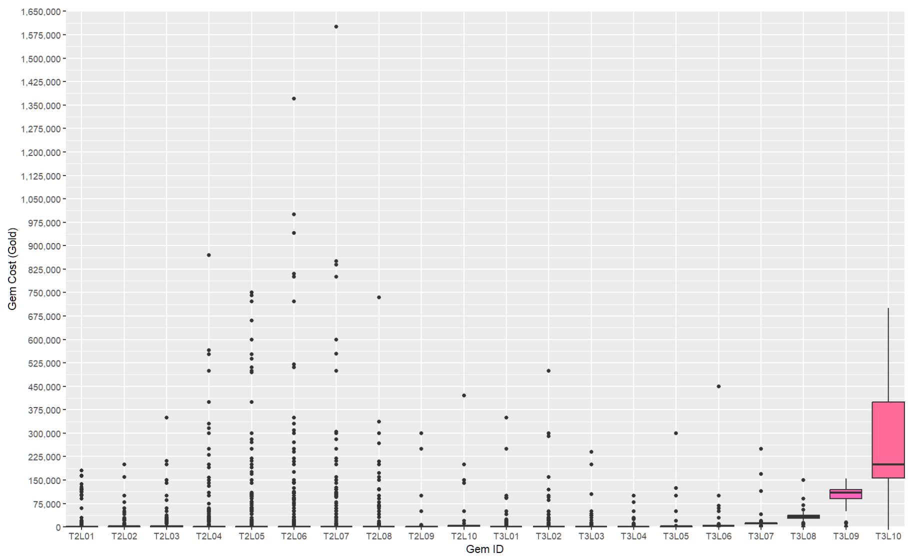


Fig. 1. Box plot showing the price variation of each gem ID. R shown in Figure 10

- *Sale Price* is the price which a gem was sold for.
- *Sale Date* is the date which a gem sold on.

For this research, an anomaly detection algorithm would utilize 4 factors; Level, Tier, Sale Price and Sale Date. This should be enough for a basis to detect prices which are too high at any given time. It also allows us to investigate fluctuations in price of gems over time which could be due to various factors in the game, such as events which inflate the number of gems within the game through increased drop rates of gems, gem giveaways or a change in the number of players which leads to less supply/demand.

Whilst looking at gem data, the average price per gem can be seen in Figure 1. This graph also shows the upper and lower bounds for gem sale price.

B. Detection Algorithms

For this paper, 4 different algorithms were written which can detect anomalies. These revolve around checking each data point and if the deviance of that data point is within a specific range. All results from these algorithms, use a constant threshold which scales to the remaining dataset. This initial threshold has not been tailored per algorithm.

1) Two Stage Thresholding Algorithm: Figure 13 showcases the 2T algorithm written for use within this paper. This algorithm is recursive and follows the general function outlined by Yang[36] in their original algorithm. This algorithm works similarly to the Mean Standard Deviation with its main change being that it runs recursively.

2) Clever SD Algorithm: Figure 12 showcases the Clever SD algorithm written for use within this paper. This algorithm is recursive and follows the general function outlined by Buzzi[9] in their original algorithm. This algorithm removes a single anomaly per function call until all anomalies have been removed.

3) Mean Standard Deviation Algorithm: Figure 14 showcases a function for standard deviaion around the mean. This is a common approach and checks if all datapoints are in range. If they are outside of the range, they are added to the anomalies array and the anomalies array is returned after the function is finished.

4) Median Standard Deviation Algorithm: Figure 15 showcases a function for standard deviaion around the median. This is a common approach and checks if all datapoints are in range. If they are outside of the range, they are added to the anomalies array and the anomalies array is returned after the function is finished.

VI. Data Collection Methodology

A. Internal Data

The best method of gathering data, is to contact the developers directly. The developer's internal policy could impact the ability to get the raw data from them directly. They may log a lot more data than is publicly accessible via the marketplace, for example, the internal data could contain account identifying information.

B. Public Data

In Korea, a public resource exist which allows users to view trasactional data across the entire auction house[30]. However, this website is inaccessible without name and age verification, due Korea's Game Industry Promotion Act[18].

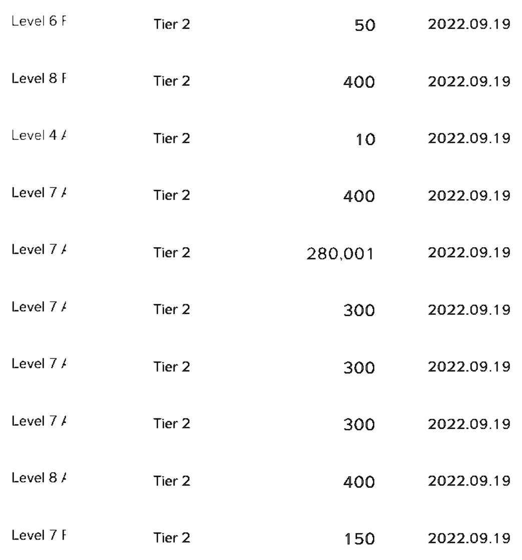
C. In-game Data

Another method for data collection surrounds screenshotting pages from the Sale History of Gems from Lost Ark's Marketplace, as seen in Figure 3. Then, the screenshot is cropped to remove the Gem Name, Starting Bid and Quality fields and converted to a greyscale image along with other image manipulation techniques to ensure clear text, shown in Figure 2 (the algorithm which produces this processed image is shown in Figure 16).

Optical character recognition (OCR) is then used on the image and checked automatically for any errors (see Figure 17). The data can then be randomly sampled and manually reviewed to ensure the accuracy of the data following this process. Due to the nature in which this algorithm works, unreadable data will be logged in the saved file as a ”-” alongside its page and entry numbers, making the issue much quicker to rectify.

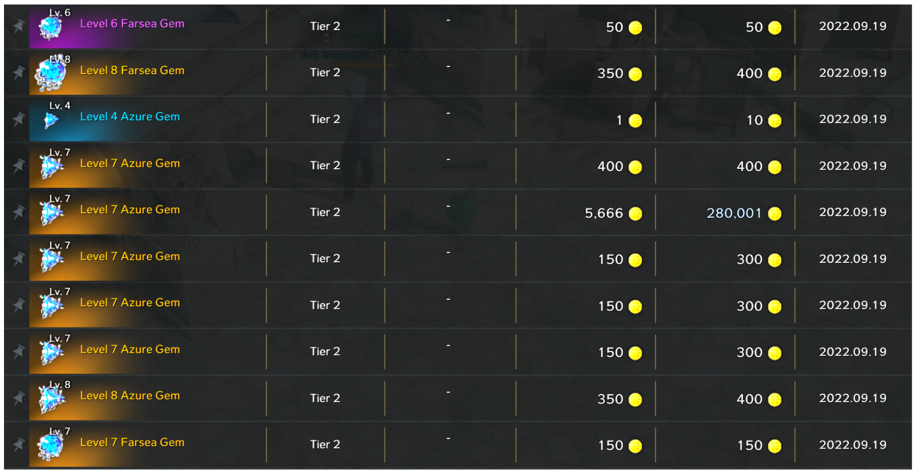
Removing the name is done as this has no bearing on the price of a gem. Removing the Starting Bid field is done because the Starting Bid is optional when adding a gem to the marketplace, it is also not indicative of RMT. Instead, for the RMT transaction to occur correctly, Gems have a Buy Now Price set to a specific value. Removing the Quality field is done since gems don't have a quality value; this is always ”-”.

A consideration which could also affect the price of Gems is the ”Gem Effect”. This is a specific ability that Gem does to impact a character's Damage or Cooldown Time on a specific move. However, the Gem Effect is optionally rerolled within



Level 6 F	Tier 2	50	2022.09.19
Level 8 F	Tier 2	400	2022.09.19
Level 4 /	Tier 2	10	2022.09.19
Level 7 /	Tier 2	400	2022.09.19
Level 7 /	Tier 2	280,001	2022.09.19
Level 7 /	Tier 2	300	2022.09.19
Level 7 /	Tier 2	300	2022.09.19
Level 7 /	Tier 2	300	2022.09.19
Level 8 /	Tier 2	400	2022.09.19
Level 7 F	Tier 2	150	2022.09.19

Fig. 2. Auction house screenshot after image processing.



Lv. 6 Level 6 Farsea Gem	Tier 2	-	50 ●	50 ●	2022.09.19
Lv. 8 Level 8 Farsea Gem	Tier 2	-	350 ●	400 ●	2022.09.19
Lv. 4 Level 4 Azure Gem	Tier 2	-	1 ●	10 ●	2022.09.19
Lv. 7 Level 7 Azure Gem	Tier 2	-	400 ●	400 ●	2022.09.19
Lv. 7 Level 7 Azure Gem	Tier 2	-	5,666 ●	280,001 ●	2022.09.19
Lv. 7 Level 7 Azure Gem	Tier 2	-	150 ●	300 ●	2022.09.19
Lv. 7 Level 7 Azure Gem	Tier 2	-	150 ●	300 ●	2022.09.19
Lv. 7 Level 7 Azure Gem	Tier 2	-	150 ●	300 ●	2022.09.19
Lv. 8 Level 8 Azure Gem	Tier 2	-	350 ●	400 ●	2022.09.19
Lv. 7 Level 7 Farsea Gem	Tier 2	-	150 ●	150 ●	2022.09.19

Fig. 3. Auction house screenshot before image processing.

the game player. Rerolling requires the use of Silver, a much easier resource to gather, so this doesn't have a huge impact on the price of the gem.

Using this data, humans can easily recognise when a sale price for a specific level/tier gem is too high compared to other gems being sold at around the same time.

Overall, after collecting data for the period 25th August 2022 until 29th December 2022*, a total of 14,928 gems were sold on the EU West market within Lost Ark and tracked for this paper (see Figure 4).

VII. Validation and Verification

Without having a labelled dataset, it will be difficult to accurately verify anomalous transactions using an algorithm. However, by utilising different algorithms for the testing of data, it is possible to check overlapping anomalies between different algorithms. This will allow for the verification that a data point is anomalous.

The algorithms can also be validated by running them multiple times. As computation is likely to be nearly instantaneous running on modern machines, running it hundereds of times, the time to run the algorithms can be calculated more accurately. To ensure accuracy and authenticity

A. *Verifying Algorithms by Comparing Results of Different Algorithms*

Comparing the results is a way in which the data can be validated. By assigning each gem with a unique value based on its location within the database we can check each gem for its occurrence as an anomaly across multiple algorithms. An example in Python would look like 11

This method, however, suffers from a problem where anomalies which aren't detected by any algorithm won't be verified at all, leading to a reduced accuracy rate.

VIII. Considerations

A. *Legal*

Smilegate, the developers of Lost Ark, didn't explicitly give permission for their dataset to use be used in this paper, however, it doesn't breach their Code of Concuct[3]. This dataset is also available

B. *Ethical*

The data processed in the paper is gathered directly from Lost Ark's public marketplace where Smilegate likely already follow data anomyization practices[16]. The data gathered doesn't contain identifiable information and therefore conforms to both General Data Protection Regulation (GDPR) and the Nuremberg Code[34]. As this papers research is carried out at Falmouth University, it also follows the Research Policy[13].

There is also a concern that as this is to do with a form of financial data, that people may rely on RMT in order to make money in thier respective counties. This could pose an issue as the plan is not to ruin someone's livlihood. On the other hand, RMT is not allowed by the game and could be comparable to turning towards crime during hardship. However, preventing this action is beneficial for companies as then they can sell their currency to consumers directly and actually make a return on their investment into the game.

C. *Professional*

Professionally, it's important that this paper follows the BCS Code of Conduct[8]. This includes working for the public interest by outlining solutions to current problems rather than ways to make current problems worse.

IX. Results and Analysis

The data shown has a GemID. This GemID coincides with two metrics each gem has, Tier and Level. Tier is represented via a T2 or T3 at the beginning of the ID and Level is represeted by L01-L10 at the end of the ID. This means that a Tier 2 Level 7 gem would be represented as T2L07 and a Tier 3 Level 10 gem would be T3L10.

The data shown in Figure 4 showcases the total size of the data for each gem type sold. There is a stark drop off in the sample size for T2L08-T2L10 gems. This will be due to Tier

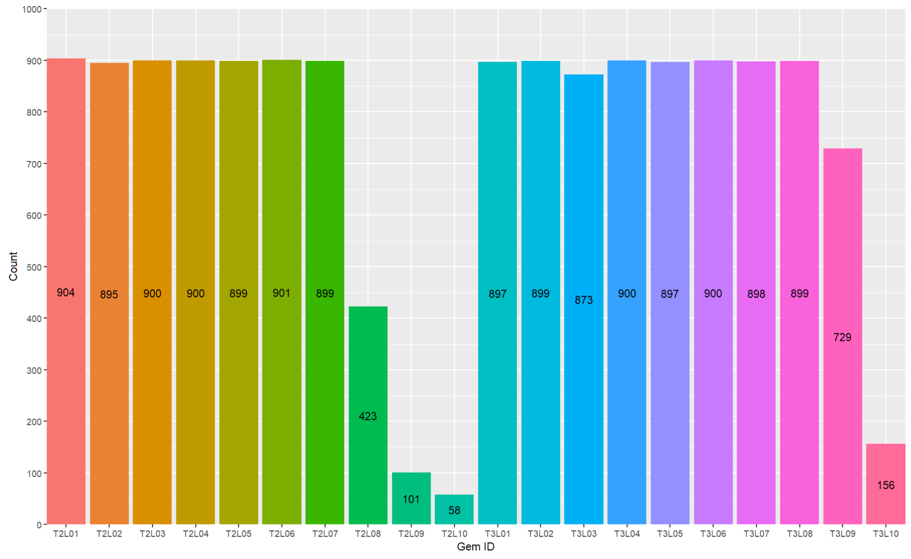


Fig. 4. Bar chart showing the number of gems recorded for each gem ID. R shown in Figure 7

2 being replaced by Tier 3 through game progression. In my personal experience, this was around the time I started needing T2L08 gems, explaining the lack of these types of gems. There is also a drop off related to T3L09 and T3L10 gems which could be attributed to their difficulty to gather as they don't drop naturally and must be created by merging 3x T3L08 and 3x T3L09 gems respectively.

Figure 5 showcases the proportion of anomalies found for each Gem ID. This data shows that T2L01 gems are likely used more often for fraudulent transactions. Due to thier low value in-game however, players typically spend 1 Gold each on them. Therefore, it is likely that the low cost gems were detected as anomalies by the 2T and CleverSD algorithms, this in and of itself could highlight how these gems being sold at vast range of prices actually increases their detection for these algorithms. The graph also shows a huge proportion of T2L05 being detected by the CleverSD and 2T algorithms. This could be attributed to fraudulent transactions. The Median SD algorithm used detected anomalies a comparitably large proportion of Level 10 gems; T2L10 and T3L10. For T3L10, this could be attributed to the high range in price for these gems as shown in Figure 1. As for T2L10, this could be attributed to an insufficient data size when compared with other gems, as shown in Figure 4.

Figure 6 shows that overall, CleverSD is extremely slow when compared to the other three algorithms. This is unperformant, especially when considering the data in Figure 5 where CleverSD and 2T both found the same amount of anomalies for each Gem ID.

Overall, these graphs help highlight that CleverSD and 2T both itentified a nearly identical number of anomalous data points, suggesting that the two algorithms work very

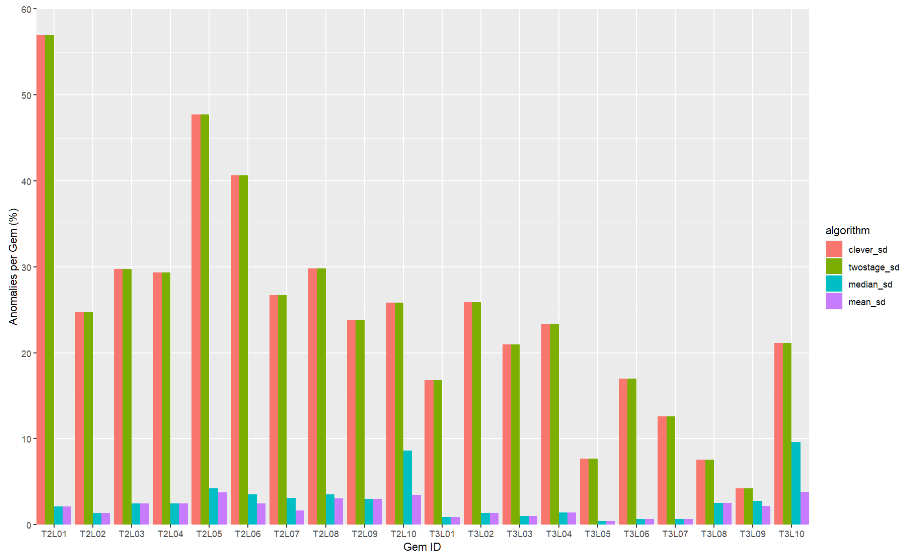


Fig. 5. Bar chart showing the percentage of anomalies found for each gem depending upon the algorithm. R shown in Figure 9

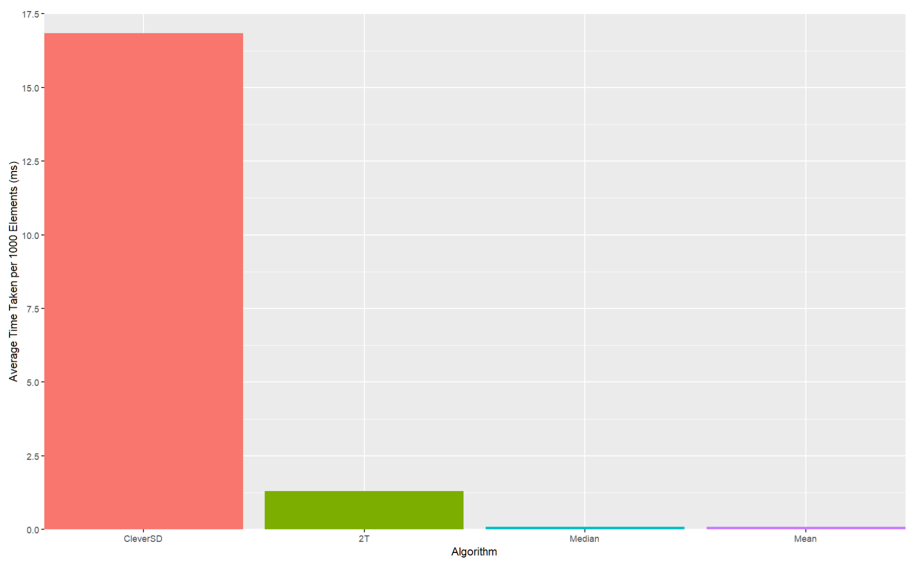


Fig. 6. Bar chart showing the average computation time per 1000 gems for each algorithm (lower is better). R shown in Figure 8

similarly. However, this doesn't guarentee accuracy and could be down to over-reporting of anomalous findings, especially when compared with SD Mean and SD Median. On average, SD Median found more anomalous data points than SD Mean, which could highlight how the data is skewed especially when looking at the sale prices shown in Figure 1. Overall, its difficult to assess accuracy due to the lack of a labelled dataset.

X. Disccussion

Looking at the data generated, it is clear that the hypotheses were along the right lines, but didn't paint the full picture. For the first research question, "Which anomaly detection algorithm is the most performant for peer-to-peer transactions within MMOs?" its clear than CleverSD was a huge way off in its computational time, performing the slowest by a substantial margin. The Mean SD was the quickest, beating out the Median SD algorithms, due to the lack of recursion.

For Hypothesis 2, "Which algorithm will detect the most anomalous data points?", the data shows that CleverSD, alongside 2T, identified the most anomalous datapoints, however, this is due to the recursive nature of the algorithms when compared with Mean and Median SD. The lack of tuning the algorithms or the small datasets used could also factor into the detection.

References

[1] Muhammad Aurangzeb Ahmad et al. "Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGs". In: *2009 International Conference on Computational Science and Engineering*. Vol. 4. Aug. 2009, pp. 340–345. DOI: 10.1109/CSE.2009.307.

[2] Mohiuddin Ahmed, Abdun Naser Mahmood, and Md. Rafiqul Islam. "A survey of anomaly detection techniques in financial domain". In: 55 (2016), pp. 278–288. ISSN: 0167-739X. DOI: 10.1016/j.future.2015.01.001.

[3] Amazon Games. *Amazon Games Code of Conduct*. URL. Accessed on November 11th 2022. URL: https://web.archive.org/web/20221110052745/https://www.amazon.com/gp/help/customer/display.html?nodeId=GK4QHHHAC82SQTS8.

[4] Pierre Baldi. "Autoencoders, Unsupervised Learning and Deep Architectures". In: *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*. UTLW'11. Washington, USA: JMLR.org, 2011, pp. 37–50.

[5] Liron Bergman and Yedid Hoshen. "Classification-Based Anomaly Detection for General Data". In: (May 2020). arXiv: 2005.02359 [cs.LG].

[6] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. "Network anomaly detection: methods, systems and tools". In: *Ieee communications surveys & tutorials* 16.1 (2013), pp. 303–336.

[7] Markus M. Breunig et al. *LOF: identifying density-based local outliers. identifying density-based local outliers*. May 2000. DOI: 10.1145/342009.335388.

[8] British Computer Society. *Research Integrity And Ethics Policy For Taught Courses*. 2022. URL: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf (visited on 12/08/2022).

[9] Guido Buzzi-Ferraris and Flavio Manenti. "Outlier detection in large data sets". In: *Computers & chemical engineering* 35.2 (2011), pp. 388–390.

[10] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[11] Julian Dibbell. "The Life of the Chinese Gold Farmer". In: *The New York Times Magazine* (June 2007).

[12] Hilmi E Egilmez and Antonio Ortega. "Spectral anomaly detection using graph-based filtering for wireless sensor networks". In: *2014 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2014, pp. 1085–1089.

[13] Falmouth University. *Research Integrity And Ethics Policy For Taught Courses*. 2022. URL: https://www.falmouth.ac.uk/sites/default/files/media/downloads/research_integrity_and_ethics_policy_for_taught_courses.pdf (visited on 12/08/2022).

[14] Atsushi Fujita, Hiroshi Itsuki, and Hitoshi Matsubara. "Detecting Real Money Traders in MMORPG by Using Trading Network". In: *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011, October 10-14, 2011, Stanford, California, USA*. Ed. by Vadim Bulitko and Mark O. Riedl. The AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE11/paper/view/4057.

[15] M. Girvan and M. E. J. Newman. "Community structure in social and biological networks". English. In: *Proceedings of the National Academy of Sciences of the United States of America* 99.12 (2002), pp. 7821–7826. ISSN: 0027-8424. DOI: 10.1073/pnas.122653799. URL: www.pnas.org/content/vol99/issue12/#APPLIED_MATHEMATICS.

[16] Nils Gruschka et al. "Privacy issues and data protection in big data: a case study analysis under GDPR". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 5027–5033.

[17] Mee Lan Han, Byung Il Kwak, and Huy Kang Kim. "Cheating and Detection Method in Massively Multiplayer Online Role-Playing Game: Systematic Literature Review". In: *IEEE Access* 10 (2022), pp. 49050–49063. DOI: 10.1109/ACCESS.2022.3172110.

[18] Korea Legislation Research Institute. *GAME INDUSTRY PROMOTION ACT. Act No. 17396*. 2020. URL: https://elaw.klri.re.kr/eng_service/lawView.do?hseq=54546&lang=ENG (visited on 12/08/2022).

[19] Maxroll GmbH. *Character Build Guides*. URL. Accessed on December 5th 2022. URL: https://web.archive.org/web/20221205003955/https://maxroll.gg/lost-ark/category/build-guides.

[20] Sumit Misra et al. "An Autoencoder Based Model for Detecting Fraudulent Credit Card Transaction". In: 167 (2020), pp. 254–262. ISSN: 1877-0509. DOI: 10.1016/j.procs.2020.03.219.

[21] Ali Bou Nassif et al. "Machine Learning for Anomaly Detection: A Systematic Review". In: *IEEE Access* 9 (2021), pp. 78658–78700. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3083060.

[22] Caleb C. Noble and Diane J. Cook. "Graph-Based Anomaly Detection". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: Association for Computing Machinery, 2003, pp. 631–636. ISBN: 1581137370. DOI: 10.1145/956750.956831. URL: https://doi.org/10.1145/956750.956831.

[23] Salima Omar, Asri Ngadi, and Hamid H. Jebur. "Machine Learning Techniques for Anomaly Detection: An Overview". In: 79 (), pp. 33–41. ISSN: 0975-8887. DOI: 10.5120/13715-1478.

[24] Thomas V Pollet and Leander Van Der Meij. "To remove or not to remove: the impact of outlier handling on significance testing in testosterone data". In: *Adaptive Human Behavior and Physiology* 3.1 (2017), pp. 43–60.

[25] Praveen Kumar Sadineni. *Detection of Fraudulent Transactions in Credit Card using Machine Learning Algorithms*. 2020. DOI: 10.1109/i-smac49090.2020.9243545.

[26] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. Ed. by Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller. The MIT Press, 1999, pp. 582–588. URL: http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.

[27] Rafet Sifa et al. "Archetypal Analysis Based Anomaly Detection for Improved Storytelling in Multiplayer Online Battle Arena Games". In: *2021 Australasian Computer Science Week Multiconference*. ACSW '21. Dunedin, New Zealand: Association for Computing Machinery, 2021. ISBN: 9781450389563. DOI: 10.1145/3437378.3442690. URL: https://doi.org/10.1145/3437378.3442690.

[28] Joseph P Simmons, Leif D Nelson, and Uri Simonsohn. "False-positive psychology: undisclosed flexibility in data collection and analysis allows presenting anything as significant." In: (2016).

[29] Amazon Games Smilegate Amazon Game Studios. *Lost Ark - Free to Play MMO Action RPG*. Digital Game. 2019. URL: https://www.playlostark.com/en-us.

[30] Smilegate Stove Smilegate RPG. *Lost Ark - Stove Website*. Website. 2019. URL: https://lostark.game.onstove.com/Markets.

[31] Xiuyao Song et al. "Conditional anomaly detection". In: *IEEE Transactions on knowledge and Data Engineering* 19.5 (2007), pp. 631–645.

[32] Square Enix. *Square Enix Final Fantasy XI Online Real Money Trading (RMT)*. URL. Accessed on November 11th 2022. URL: https://web.archive.org/web/20221110062944/https://support.na.square-enix.com/faqarticle.php?id=20&kid=12802.

[33] Jianrong Tao et al. "MVAN: Multi-view Attention Networks for Real Money Trading Detection in Online Games". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, July 2019, pp. 2536–2546. ISBN: 9781450362016. DOI: 10.1145/3292500.3330687. URL: https://doi.org/10.1145/3292500.3330687.

[34] "The Nuremberg Code (1947)". In: *BMJ* 313.7070 (1996). Ed. by, p. 1448. DOI: 10.1136/bmj.313.7070.1448. eprint: https://www.bmj.com/content/313/7070/1448.1. URL: https://www.bmj.com/content/313/7070/1448.1.

[35] Dietrich Wettschereck, David W Aha, and Takao Mohri. "A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms". In: *Artificial Intelligence Review* 11.1 (1997), pp. 273–314.

[36] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. "Outlier Detection: How to Threshold Outlier Scores?" In: *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing*. AIIPCC '19. Sanya, China: Association for Computing Machinery, 2019. ISBN: 9781450376334. DOI: 10.1145/3371425.3371427. URL: https://doi.org/10.1145/3371425.3371427.

[37] Min-Ling Zhang and Zhi-Hua Zhou. "A k-nearest neighbor based algorithm for multi-label classification". In: *2005 IEEE international conference on granular computing*. Vol. 2. IEEE. 2005, pp. 718–721.

[38] Bo Zong et al. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL: https : / / openreview . net / forum ? id = BJJLHbb0-.

APPENDIX A
ACKNOWLEDGEMENTS

I would like to thank my supervisor John Speakman for support during this project. I would also like to thank Michael Scott for his work in creating this module, and his work towards improving the University as a whole. I'd also like to thank all my lecturers at Falmouth University who have supported my learning over the last 3 years, as well as their openness to change based on what the student body is after.

APPENDIX A
REFLECTIVE ADDENDUM

I am happy overall with the progress which I have made on this dissertation, however, it hasn't been without its issues and looking back, I would have done a lot differently if I was to start this project fresh.

A. Change the dataset

One of the major issues I have faced has been regarding the dataset used. When I started this project, I had been playing Lost Ark with friends for the prior couple of months and wanted to solve a problem that I felt had plagued the game. I did contact the developers, hoping I could access this data as part of my dissertation but I never recieved a response. Instead of going ahead with the data I gathered, it would have been a great idea to use an open and pre-labelled dataset from another game; these overall are difficult to come by and I wouldn't have the same connection and understanding of a different dataset.

B. Look at different algorithms

Another thing I would have liked to look into would have been other methods of anomaly detection. Originally this was the plan, but to help manage scope and not overwork myself for this project - and other committments I had - I chose to focus on a small subsection of algorithms, which focused solely around Standard Deviation. I would have liked to have Machine Learning or Autoencoders but this felt out of reach during the middle of the project.

C. Overall dislike of academic writing

One of my biggest issues with school in general, is academic writing. I don't enjoy writing essays - and I'm sure not many people do - but that has been a big blocker for me personally. It led to me developing my algorithms quite early, learning about OCR and detection algorithms without putting a huge amount of thought into this paper as a whole. It also led to me focusing on my other, more practical, modules or work experience through internships. The literature review was the only part I particularly found enjoyable as I found the academic reading to be interesting. This was due to me learning new things, something I enjoy doing.

D. Conclusion

Overall, I believe that the project has gone well, I feel happy and content with what I have produced. Whilst it could have been better, it also could have gone much worse. With a little more thought into time management and planning, I could have produced something better.

APPENDIX B
STATISTICS ADDENDUM

```r
library(ggplot2)

metrics <- read.csv("metrics.csv")

sample <- subset(metrics, algorithm == "mean_sd")

p <-
  ggplot(sample, aes(x = gemID, y = dataSize, fill =
     factor(gemID))) +
  geom_bar(stat="identity") +
  geom_text(aes(label = dataSize), nudge_y = sample$
    dataSize * -0.5) +
  labs(x = "Gem ID", y = "Count") +
  coord_cartesian(expand = FALSE, xlim = c(NA, NA),
    ylim = c(0, 1000)) +
  scale_y_continuous(breaks=seq(0, 1000, 100)) +
  theme(legend.position = "none", plot.title =
    element_text(hjust = 0.5))

print(p)
```

Fig. 7. R Code which displays the gem datasize graph shown in Figure 4.

```r
library(ggplot2)

metrics <- read.csv("time-metrics.csv")
metrics$algorithm <- factor(metrics$algorithm,
    levels = c("CleverSD", "2T", "Median", "Mean"))

```

```
p <-
  ggplot(metrics, aes(x = algorithm, y = (time /
    dataSize) * 1000)) +
  geom_bar(aes(fill = algorithm), position="dodge",
    stat="identity") +
  labs(x = "Algorithm", y = "Average Time Taken per
    1000 Elements (ms)", color = "Algorithm") +
  coord_cartesian(expand = FALSE, xlim = c(NA, NA),
    ylim = c(0, 17.5)) +
  scale_y_continuous(breaks=seq(0, 17.5, 2.5)) +
  theme(legend.position = "none", plot.title =
    element_text(hjust = 0.5))

print(p)
```

Fig. 8. R Code which displays the average time taken per 100 elements graph shown in Figure 6.

```
library(ggplot2)

metrics <- read.csv("metrics.csv")
metrics$algorithm <- factor(metrics$algorithm,
    levels = c("clever_sd", "twostage_sd", "median_
    sd", "mean_sd"))

p <-
  ggplot(metrics, aes(x = gemID, y = (anomalies /
    dataSize) * 100, group = algorithm)) +
  scale_color_manual(values = c("red", "orange", "
    green", "blue")) +
  geom_bar(aes(fill = algorithm), position="dodge",
    stat="identity") +
  labs(x = "Gem ID", y = "Anomalies per Gem (%)",
    color = "Algorithm") +
  coord_cartesian(expand = FALSE, xlim = c(NA, NA),
    ylim = c(0, 60)) +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks=seq(0, 60, 10))

print(p)
```

Fig. 9. R Code which displays the graph showing number of anomalies generated from each algorithm in Figure 5.

```
library(ggplot2)
library(scales)

data <- read.csv('./Gem Data/all-gems.csv')

p <-
  ggplot(data, aes(x = GemID, y = Cost, fill =
    factor(GemID))) +
  labs(x = "Gem ID", y = "Gem Cost (Gold)") +
  geom_boxplot(aes(x = GemID, y = Cost)) +
  coord_cartesian(expand = FALSE, xlim = c(NA, NA),
    ylim = c(0, 1650000)) +
  scale_y_continuous(breaks=seq(0, 1650000, 75000),
    labels = comma) +
  theme(legend.position = "none", plot.title =
    element_text(hjust = 0.5))

print(p)
```

Fig. 10. R Code which displays the graph showing number of anomalies generated from each algorithm in Figure 1.

Appendix C
Testing Addendum

Appendix D
Data Collection Code Samples

```
# create a dictionary for anomalies
a = {}

# function to calculate results
def count_result(p):
    try:
        # if point already exists in anomalies
    increment it
        a[p] = a[p] + 1
    except:
        # create point in anomalies
        a[p] = 1

# function to check all results in the data
def check_results(d):
    # for all indexes in the data
    for i in d:
        # count the result of the index
        count_result(i)

# load the dataset
d = load_dataset('data.txt')

# check the results of the data
check_results(d)

# print the anomalies
print(a)
```

Fig. 11. Comparison Algorithm.

```
# CleverSD algorithm
def clever_sd(data, threshold, anomalies=None):
    # get the gem costs
    costs = get_costs(data)
    # calculate the anomaly threshold based on the
    costs
    anomaly_threshold = calculate_threshold(costs,
    threshold)
    # calculate the mean of the data
    average = np.mean(costs)
    # set a base index of the largest value
    largest_index = -1
    # set a base deviance of the largest value
    largest_deviance = -1
    # run through the data
    for i in range(len(data)):
        # get the gem
        gem = data[i]
        # calculate the deviance
        deviance = abs(gem.cost - average)
        # if the deviance is above the threshold AND
     above the largest deviance
        if deviance > anomaly_threshold and deviance
     > largest_deviance:
            # set the largest index
            largest_index = i
            # set the largest deviance
            largest_deviance = deviance
    # if the anomalies array doesn't exist, create
     it
    if (anomalies == None):
        anomalies = []
    # if the largest index is -1, return all the
    anomalies
    if (largest_index == -1):
        return anomalies
    # get the gem
    gem = data[largest_index]
    # add the gem to the anomalies
    anomalies.append(gem)
    # remove the gem from the dataset
    data.remove(gem)
    # recursively call the function
    return clever_sd(data, threshold, anomalies)
```

Fig. 12. CleverSD Algorithm.

Appendix E
Anomaly Detection Algorithms

```python
# 2T algorithm
def twostage_sd(data, threshold, anomalies=None):
    # get the gem costs
    costs = get_costs(data)
    # calculate the anomaly threshold based on the costs
    anomaly_threshold = calculate_threshold(costs, threshold)
    # calculate the mean of the data
    average = np.mean(costs)
    # if the anomalies array doesn't exist, create it
    if (anomalies == None):
        anomalies = []
    # set a default value for anomalies found
    anomaly_found = False
    # iterate through the data and detect anomalies
    for i in range(len(data)):
        # if i is greater than the data length, break from the for loop
        if (len(data) <= i):
            break
        # get the gem
        gem = data[i]
        # calculate the deviance
        deviance = abs(gem.cost - average)
        # if the deviance is above the threshold
        if deviance > anomaly_threshold:
            # log that an anomaly was found
            anomaly_found = True
            # add the gem to the anomalies
            anomalies.append(gem)
            # remove the gem from the dataset
            data.remove(gem)
    # if an anomaly has been found, recursively call the function
    if anomaly_found:
        return twostage_sd(data, threshold, anomalies)
    # return the array of anomalies
    return anomalies
```

Fig. 13. 2T Algorithm.

```python
# standard deviation around the mean algorithm
def mean_anomaly_detection(data, threshold):
    # get the gem costs
    costs = get_costs(data)
    # calculate the mean and standard deviation of the data
    average = np.mean(costs)
    # calculate the anomaly threshold based on the costs
    anomaly_threshold = calculate_threshold(costs, threshold)
    # initialize a list to store the anomalies
    anomalies = []
    # iterate through the data and add out of range data to anomalies
    for i in range(len(data)):
        gem = data[i]
        deviance = abs(gem.cost - average)
        if deviance > anomaly_threshold:
            anomalies.append(gem)
    # return the anomlies found
    return anomalies
```

Fig. 14. Standard Deviation (Mean) Algorithm.

```python
# standard deviation around the median algorithm
def median_anomaly_detection(data, threshold):
    # get the gem costs
    costs = get_costs(data)
    # calculate the median and standard deviation of the data
    average = np.median(costs)
    # calculate the anomaly threshold based on the costs
    anomaly_threshold = calculate_threshold(costs, threshold)
    # initialize a list to store the anomalies
    anomalies = []
    # iterate through the data and add out of range data to anomalies
    for i in range(len(data)):
        gem = data[i]
        deviance = abs(gem.cost - average)
        if deviance > anomaly_threshold:
            anomalies.append(gem)
    # return the anomlies found
    return anomalies
```

Fig. 15. Standard Deviation (Median) Algorithm.

```python
import os
from PIL import Image, ImageEnhance, ImageOps, ImageFilter
import numpy as np
# input and output of the images
INPUT_LOCATION = "original-screenshots"
OUTPUT_LOCATION = "output-screenshots"
# the crop locations for the images (x pos, width)
IMAGE_CROPS = [(1270, 30), (570, 500), (90, 210)]
# output width of the images (retain original height)
IMAGE_OUTPUT_WIDTH = 800

# crop a column out of an image
def image_crop_column(img, crop):
    # get the starting pos
    crop_x, crop_width = crop
    # convert the image to an array
    img_arr = np.array(img)
    # move the data from the crop_width to the current x value
    img_arr[:, crop_x:img.width-crop_width] = img_arr[:, crop_x+crop_width:img.width]
    # convert the array back to an image and return the image
    crop = Image.fromarray(img_arr)
    return crop

# resize the image
def image_resize(img):
    # crop the image to be size [WIDTH, HEIGHT] and return
    resize = img.crop((0, 0, IMAGE_OUTPUT_WIDTH, img.height))
    return resize

# apply effects to the image
def apply_effect(img):
    enhancer = ImageEnhance.Brightness(img)
    output = enhancer.enhance(1)
    enhancer = ImageEnhance.Contrast(output)
    output = enhancer.enhance(0.8)
    output = ImageOps.posterize(output, 1)
    output = output.filter(ImageFilter.EDGE_ENHANCE_MORE)
    enhancer = ImageEnhance.Sharpness(output)
    output = enhancer.enhance(1)
    output = ImageOps.invert(output)
    return output

# for all files in the directory
for file in os.listdir(INPUT_LOCATION):
    # escape clause if they aren't pngs, continue to next cycle
    if not file.endswith(".png"):
        continue
    # open the image and convert it to greyscale
    img = Image.open(r"{0}\\{1}".format(INPUT_LOCATION, file)).convert('L')
    # crop out the columns specified
    for i in IMAGE_CROPS:
        img = image_crop_column(img, i)
    # resize the image after cropping
    img = image_resize(img)
    # apply the desired effects to the image
    img = apply_effect(img)
    # save the image
    img.save(r"{0}\\{1}".format(OUTPUT_LOCATION, file))
    print(file)

print("\n\n!! COMPLETED !!")
```

Fig. 16. Image Formatter Algorithm.

```python
import pytesseract
import re
# important consts for program to know
IMAGE_COUNT = 165
DATASET_NAME = "Dataset 1 - 08.09.2022.csv"
INPUT_DATA_LOCATION = "output-screenshots"
# tesseract specific configuration
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"
custom_oem_psm_config = r'''
-c tessedit_char_whitelist="01234567890TierLvl,. "
-c preserve_interword_spaces=1x1
--oem 3 --psm 6'''
# ensure user knows they are about to overwrite all data from previously
i = input("Continuing will clear existing data. Would you like to continue? [Y/N] ")
if (i == "Y"):
    # clear all data
    output = open(DATASET_NAME, "w+")
    output.write("Page,EntryNo,Level,Tier,Cost,Date\n")
    output.close()
else:
    # exit program
    log = "Input not recognised. Exiting program."
    if (i == "N"):
        log = "Exiting program."

    print(log)
    exit()
# gem class to store info about gems
class Gem():
    # when creating a new class
    def __init__(self, page = "-", entry_no = "-", level = "-", tier= "-", cost="-", date="-"):
        self.page = page
        self.entry_no = entry_no
        self.level = level
        self.tier = tier
        self.cost = cost
        self.date = date
    # convert gem stucture to a string
    def __str__(self) -> str:
        return r"{0},{1},{2},{3},{4},{5}".format(self.id, self.page, self.entry_no, self.level, self.tier, self.cost, self.date)
# get the data in rows
def get_data_rows(ocr):
    # the minimum characters for the row to be counted
    min_row_characters = 10
    # split with regex of new line
    rows = re.split(r'\n+', ocr)
    # create a new array for cleaner rows
    filtered_rows = []
    # only get rows which are within the threshold
    for row in rows:
        #print(len(row))
        if (len(row) > min_row_characters):
            filtered_rows.append(row)
    # return the cleaned rows
    return filtered_rows
# function to get the gem data
# passes through the entry number
# passes through the row to get the data from
def get_gem_data(number, row):
    # get the row data by splitting entries with >=2 space characters
    row = re.split(r"\s{2,}", row)
    # set the filters
    filter_level  = r'Level'
    filter_tier = r'er \d'
    filter_date = r'\d\d\d\d.\d\d.\d\d'
    filter_cost = '\d{1,3}[\,.]{1}\d{1,3}|\d{1,3}'
    # create a new gem using the page number and index number
    gem = Gem(page, number)
    # for all the cells in the row
    for cell in row:
        cell = cell.replace(',', '')
        # check through cell with each filter and process accordingly
        if re.search(filter_level, cell):
            gem.level = handle_level(cell)
        elif re.search(filter_tier, cell):
            gem.tier = cell
        elif re.search(filter_date, cell):
            gem.date = cell
        elif re.search(filter_cost, cell):
            gem.cost = handle_cost(cell)
        else:
            print(r"Data issue: {0}".format(cell))
    return str(gem)
# format the level data
def handle_level(level):
    number = re.search('\d+', level).group(0)
    if (int(number) > 10):
        number = number[0]
    level = "Level " + number
    return level
# format the cost data
def handle_cost(cost):
    cost = re.sub('\D', '', cost)
    return cost
# parsing string information
def parse_string_data(ocr):
    # get the row data from the OCR
    data_rows = get_data_rows(ocr)
    # create a new list to store gems
    gem_list = []
    # for each row in the data, get the gem data
    for i in range(len(data_rows)):
        row = data_rows[i]
        gem_list.append(get_gem_data(i + 1, row))
    # create output and add the gem data to it
    output = open(DATASET_NAME, "a")
    output.write('\n'.join(gem_list))
    output.write('\n')
    output.close()
    return "[SUCCESS] {0}\n".format(page)
```

```python
# for each image
for page in range(1, IMAGE_COUNT + 1):
    page_id = r"page_{0}.png".format(page)
    # get the ocr data from tesseract
    ocr = pytesseract.image_to_string(r"{0}\{1}".
     format(INPUT_DATA_LOCATION, page_id), config=
     custom_oem_psm_config)
    # generate the text data
    text_data = parse_string_data(ocr)
    print(text_data)
```

Fig. 17. Image Processing Algorithm.