ML Insider Trading Project: Fall 2024

Atharva Beesan, Raksha Govind, Sneha Jaiswal, Michelle Liang, Soham Samal

Final Report

Introduction

Insider trading, where individuals with confidential company information trade stocks for personal gain, poses a major threat to market integrity. With machine learning and big data, we can detect illegal activities by identifying anomalies in trading patterns. By highlighting these anomalies, we can flag suspicious transactions for further investigation. Previous research applied techniques like K-means clustering and statistically validated networks. Esen et al. (2019) analyzed 1M+ transactions, including 60K insiders, over 7 years to detect outliers. The study found that outlier transactions earned larger abnormal returns using traditional event study methods. Mazzarisi et al. (2024) used two methods to detect insider trading in Italian markets: an event-based approach and Statistically Validated Networks to identify investor groups with trading discontinuities around market-sensitive events. We will use the S&P500 Insider Trading dataset from Kaggle, covering insider trading activity from 2016-2017, including transaction details such as date, insider name/position, company name/ticker, transaction type, shares traded, price, total value, and ownership.

Problem

Insider trading is a significant issue in financial markets, potentially distorting market fairness and discouraging investor confidence. Detecting and mitigating insider trading can be a complicated task due to the secret nature of these trades and the subtle ways in which these traders are able to manipulate stock prices. Manual audits and reactive regulatory actions are commonly used to detect insider trading. Our primary motivation is to use a data-driven approach to come up with a more efficient way to catch these crimes.

Methods

Data Pre-Processing Methods

For insider trading detection, proper data preprocessing is crucial since stock prices, financial ratios, and stock ownership operate on different scales. One method we used is normalization, scaling factors like trading volume and volatility consistently across the dataset. We applied StandardScaler() from scikit-learn to normalize features with a mean of 0 and variance of 1.

Time-Series Transformation is also needed, as insider trading requires analyzing trades before and after events. Pandas .shift() helps convert data to a time-series format. We created 14-day windows of stock data for 7 days before and after the trade. Key features like average price, trading volume and volatility are computed separately for pre and post trade periods for each of the windows. This helped us quantify changes between trades. Each window is labeled as “insider” or “non-insider” which creates structured data ready for ML classification.

Since insider trading events are rare, we resampled using imbalanced-learns Synthetic Minority Oversampling Technique (SMOTE). By using SMOTE, we generated synthetic samples for insider trades, the minority class. This helped us address class imbalance because instead of duplicating existing data, SMOTE considers the nearest neighbors to interpolate new data point. This balanced our dataset and prevented overfitting in our model.

Selected Algorithms/Models

We have selected three key supervised algorithms: Random Forest, XGBoost, and Long Short-Term memory (LSTM) networks. We chose our first model, Random Forest, because it is helpful against overfitting, handles high-dimensional data well and provides feature importance scores that help us understand how impactful each feature is. This worked by creating an ensemble of decision trees, each of which make predictions and then aggregating all of these predictions into a final classification. The data was split into training and test sets and the model was trained on features like pre and post trade price changes. By using 100 trees in the forest we improved accuracy by averaging multiple outcomes. We then evaluated the model’s performance with various metrics.

Our second algorithm, XGBoost, handles uneven data and finds patterns using multiple decision trees, combining models to enhance prediction accuracy. It is a gradient boosting algorithm, so it is able to handle issues like missing values or non-linear relationships making it ideal for insider trading detection. XGBoost processes our generated features including price movements (pre/post trade average), volatility

measures, and volume indicators sequentially. Each tree learned to correct errors from previous trees. We implemented XGBoost with optimized parameters as we have 18 features in our feature space. We trained it on the labeled dataset where insider trades were marked as 1, so the model could effectively learn patterns and capture any subtle changes in market behavior.

Our final model was a Long Short-Term Memory model. We chose LSTM because of its ability to capture long-term dependencies. LSTM is a recurrent neural network, meaning that it is good at processing sequential data, making it valuable in our case because it would be able to remember any patterns in our features over extended periods. We implemented LSTM by first structuring our features into sequences, allowing the model to learn from the temporal relationships between our pre-trade and post-trade metric. It processed our feature set including percentage price changes and average changes to learn any patterns that would identify if there is insider trading activity.

Results & Discussion

We had used 3 key metrics to analyze our models performance and areas of improvement. The first was f1-score, which is the harmonic mean of precision and recall. This allows us to have a more balanced measure of accuracy, especially for imbalanced datasets. In our case, the number of insider trades is minute compared to the number of normal trades, and thus, f1-score is perfect for measurement here. The second is the cross-validation scores computed with the cross_val_score function assess model stability and generalization across different splits of the dataset. The average accuracy across folds gives a more robust measure than a single train-test split. Finally, the feature importance from the RandomForestClassifier model gives insight into which features the model deems most influential for making predictions. This can help interpret the model and understand which stock features correlate most strongly with insider trading.

Random Forest

The Random Forest models average accuracy rate was 0.9182 when trained with an equal 30% of insider/non-insider trades of our dataset and tested with 70%, which helped reduce bias and improve generalization in model training. The model achieved high precision, recall, f1-score, and support scores, with an accuracy of ~92% post-training, correctly predicting 44 non-insider trades from 50 non-insider windows. While random forests can often be seen as a “black box”, the feature importance histogram shows that the three most significant features in predicting a trade are post trade averages, pre-trade averages, and average volume within the window. Through using random forest in this implementation, our model was able to decrease overfitting, thus improving prediction accuracy, however could be improved on with a larger training dataset.

```
199
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        48
           1       1.00      0.94      0.97        52

    accuracy                           0.97       100
   macro avg       0.97      0.97      0.97       100
weighted avg       0.97      0.97      0.97       100

[[48  0]
 [ 3 49]]
```

```python
from sklearn.model_selection import cross_val_score

# Cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
print(f"Cross-validation scores: {scores}")
print(f"Average accuracy: {scores.mean():.4f}")
```

```
Cross-validation scores: [0.975      0.975      1.         1.         0.64102564]
Average accuracy: 0.9182
```

Figure 1- Precision, Recall, & F1-Scores for Random Forest Implementation

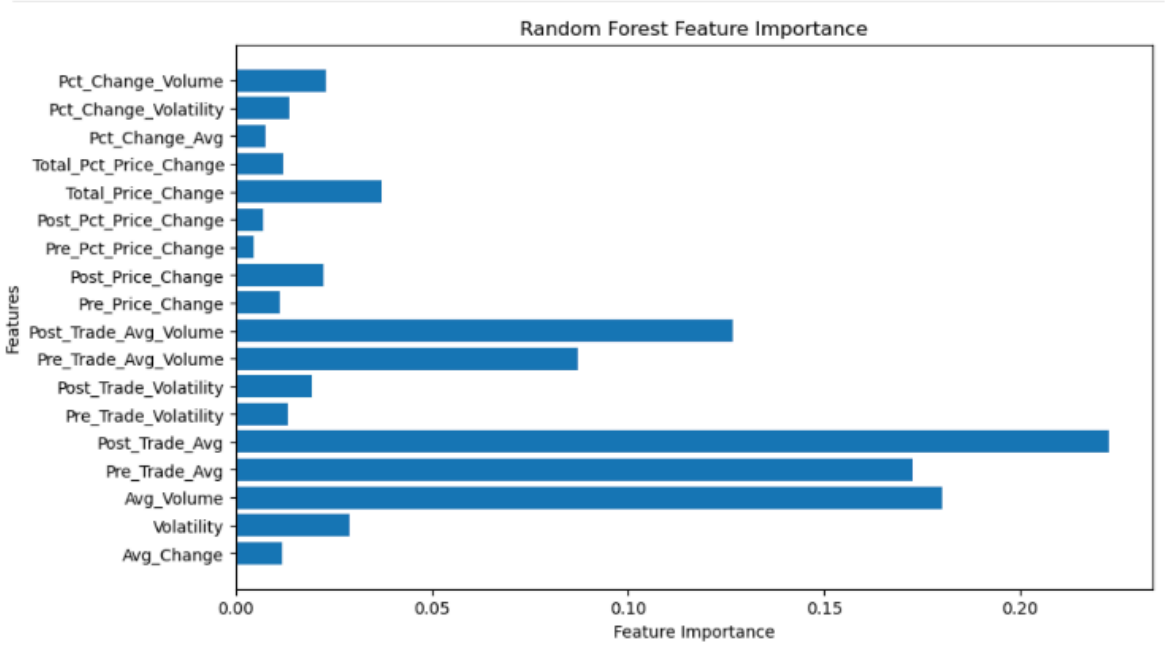


Figure 2- Feature Importance for Random Forest Implementation

XGBoost

Using XGBoost, the model demonstrated the strongest performance of all three models in detecting insider trading in financial markets. The average accuracy rate of the model was 95%, significantly higher than the baseline, with balanced class precision and recall scores. The classification report revealed that all negative predictions were correct, 92% of positive predictions were correct, 89% of non-insider samples were identified correctly, and all insider trades were identified correctly. XGBoost particularly excels in handling class 0 better than Random Forest. This can be observed by the confusion matrix below:

```
XGBoost Performance:
              precision    recall  f1-score   support

           0       1.00      0.89      0.94        19
           1       0.92      1.00      0.96        22

    accuracy                           0.95        41
   macro avg       0.96      0.95      0.95        41
weighted avg       0.96      0.95      0.95        41

[[17  2]
 [ 0 22]]
```

Figure 3- Precision, Recall, & F1-Scores for XGBoost Implementation

The confusion matrix indicates that the model misclassified 2 non-insider trades but correctly classified all insider trades, suggesting high sensitivity to the minority class. The SMOTE preprocessing step ensured better model generalization by balancing the class distribution, thereby reducing bias.The feature importance comparison shows that the most significant features in predicting insider trades are: pre-trade averages (highest contribution), post-trade averages, and average volume within the window.

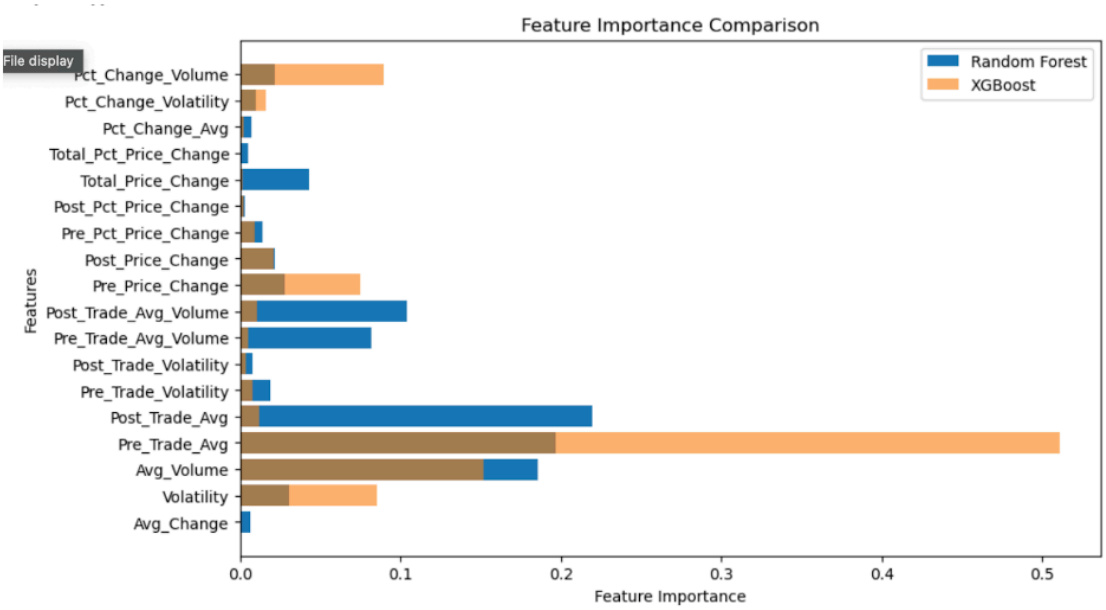


Figure 4- Feature Importance for XGBoost Implementation

These features highlight that both the patterns preceding and following a trade, as well as the overall volume, play a crucial role in distinguishing insider from non-insider activity. The feature importance histogram also illustrates that XGBoost prioritizes specific features more distinctly than the Random Forest model, indicating a nuanced understanding of the relationships in the data.

LSTM

Using LSTM as the model architecture resulted in suboptimal performance, on average <15% less accurate than XGBoost and Random Forest. The average accuracy of the model was 80%, across precision, recall, and f1-score metrics. Among all negative predictions, 78% of false predictions were correct and 83% of non-insider-trading cases were detected correctly. Across positive predictions, 82% of true predictions were correct and 77% of insider-trading cases were detected correctly.

LSTM Classification Report:

	precision	recall	f1-score	support
0	0.78	0.83	0.81	30
1	0.82	0.77	0.79	30
accuracy			0.80	60
macro avg	0.80	0.80	0.80	60
weighted avg	0.80	0.80	0.80	60

Figure 5- Precision, Recall, & F1-Scores for LSTM Implementation

The LSTM model architecture is slightly better at predicting class 0 (positive predictions) over class 1 (negative predictions), as there is a higher average recall for class 0 metrics. In general, LSTM has a generally equal distribution performance between both classes, however it remains to have the least model accuracy across all methods.

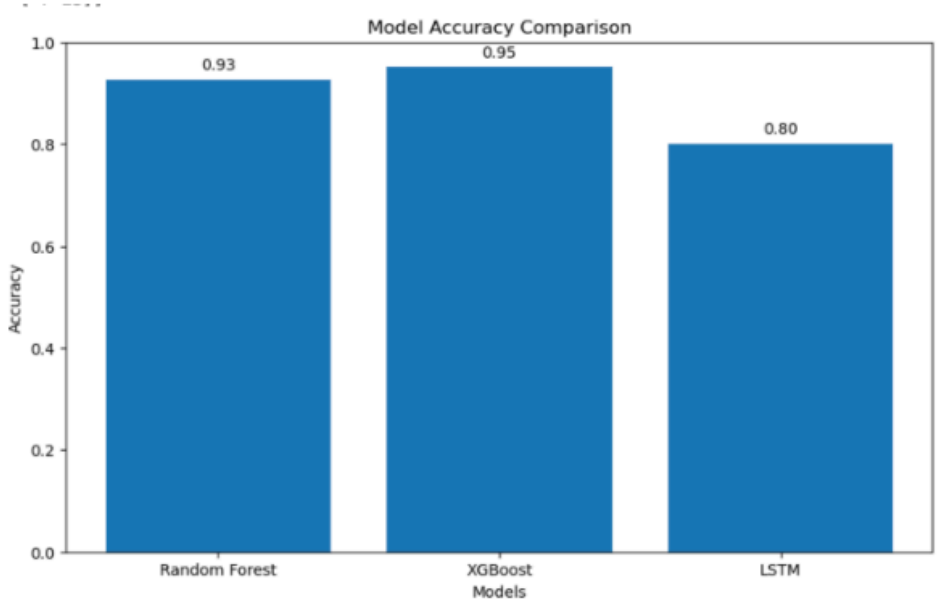


Figure 6- Model Accuracy Comparisons for Insider Trading Methods

Next Steps

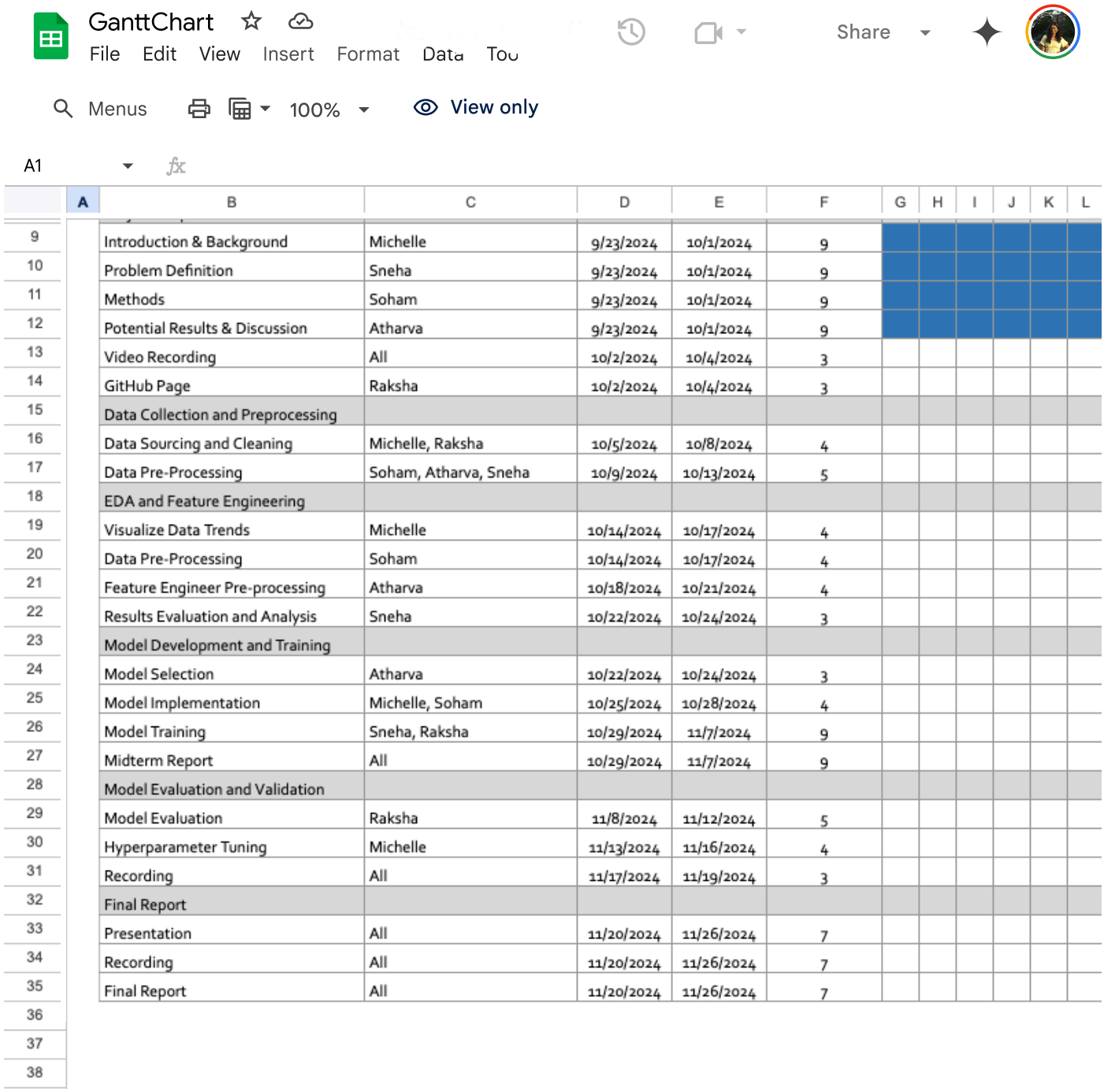
Moving forward, we aim to enhance our models by refining our feature selection for Random Forest and feature engineering for LSTM. This will hopefully enhance the relevance and interpretability to insider trading patterns. We also could hopefully increase the dataset size to help mitigate overfitting and improve robustness. We also hope to incorporate ensemble methods by combining the models. If we combined RandomForest and XGBoost could potentially improve accuracy by combining our models similar strengths. We could also incorporate LSTM into the ensemble learning. Even though LSTM underperformed in our research, we could use its advantage of strong temporal pattern analysis to create a more comprehensive prediction. Finally, we could use Bayesian optimization to fine tune parameters such as number of estimators, maximum tree depth, learning rate (for XGBoost), and more. If we incorporated Bayesian optimization, we could expect XGBoost to further improve its accuracy/prediction. Random Forest might address the slight drop in recall, and LSTM could benefit significantly by improving sequential modeling capabilities.

References

Esen, M. F., Bilgic, E., & Basdas, U. (2019). How to detect illegal corporate insider trading? A data mining approach for detecting suspicious insider transactions. Intelligent Systems in Accounting, Finance and Management, 26(2), 60/70. https://doi.org/10.1002/isaf.1446

Mazzarisi, P., Camacho-Collados, I., & Esposito, G. (2024). Detecting insider trading using statistically validated networks. SSRN. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4294752

Gantt Chart



	A	B	C	D	E	F	G	H	I	J	K	L
9		Introduction & Background	Michelle	9/23/2024	10/1/2024	9						
10		Problem Definition	Sneha	9/23/2024	10/1/2024	9						
11		Methods	Soham	9/23/2024	10/1/2024	9						
12		Potential Results & Discussion	Atharva	9/23/2024	10/1/2024	9						
13		Video Recording	All	10/2/2024	10/4/2024	3						
14		GitHub Page	Raksha	10/2/2024	10/4/2024	3						
15		Data Collection and Preprocessing										
16		Data Sourcing and Cleaning	Michelle, Raksha	10/5/2024	10/8/2024	4						
17		Data Pre-Processing	Soham, Atharva, Sneha	10/9/2024	10/13/2024	5						
18		EDA and Feature Engineering										
19		Visualize Data Trends	Michelle	10/14/2024	10/17/2024	4						
20		Data Pre-Processing	Soham	10/14/2024	10/17/2024	4						
21		Feature Engineer Pre-processing	Atharva	10/18/2024	10/21/2024	4						
22		Results Evaluation and Analysis	Sneha	10/22/2024	10/24/2024	3						
23		Model Development and Training										
24		Model Selection	Atharva	10/22/2024	10/24/2024	3						
25		Model Implementation	Michelle, Soham	10/25/2024	10/28/2024	4						
26		Model Training	Sneha, Raksha	10/29/2024	11/7/2024	9						
27		Midterm Report	All	10/29/2024	11/7/2024	9						
28		Model Evaluation and Validation										
29		Model Evaluation	Raksha	11/8/2024	11/12/2024	5						
30		Hyperparameter Tuning	Michelle	11/13/2024	11/16/2024	4						
31		Recording	All	11/17/2024	11/19/2024	3						
32		Final Report										
33		Presentation	All	11/20/2024	11/26/2024	7						
34		Recording	All	11/20/2024	11/26/2024	7						
35		Final Report	All	11/20/2024	11/26/2024	7						
36												
37												
38												

Contribution Table

Team Member	Contribution
Atharva Beesen	Data Preprocessing
Raksha Govind	Analysis of Algorithm Model, Github Pages
Sneha Jaiswal	Data Preprocessing and ML Algorithms Methods
Michelle Liang	Next Steps
Soham Samal	Algorithm Implementation, Quantititative Metrics