



COMP SCI 7015

Software Engineering & Project

Sprint Retrospective 3

Rail Break Prediction AI

a1882259

Nilangi Maheesha Sithumini Edirisinghe

Group IF_PG1

List of Group Members:

a1873818 Zewei You, a1898921 Yong Kheng Beh, a1882259 Nilangi Edirisinghe, a1879038 Bhagya Indeewari Senanayake Senanayake Mudiyanse, a1878048 Wenjing Shen, a1879321 Xinyue Ma, a1877644 Jiemin Zhanga, 1878387 Qiaoqiao Chen

1. Snapshots (Group)

1.1 Product Backlog and Task Board

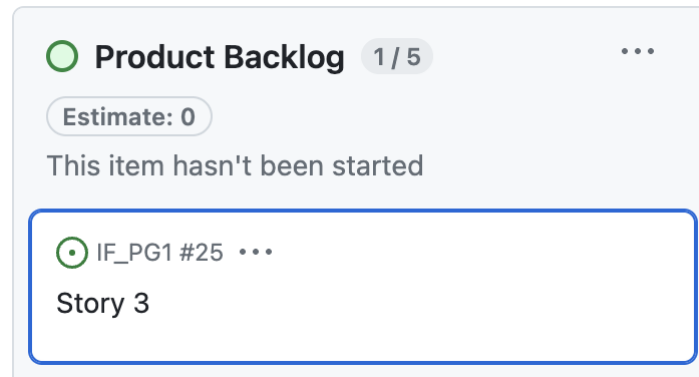


Figure 1. The Screenshot of the Product Backlog

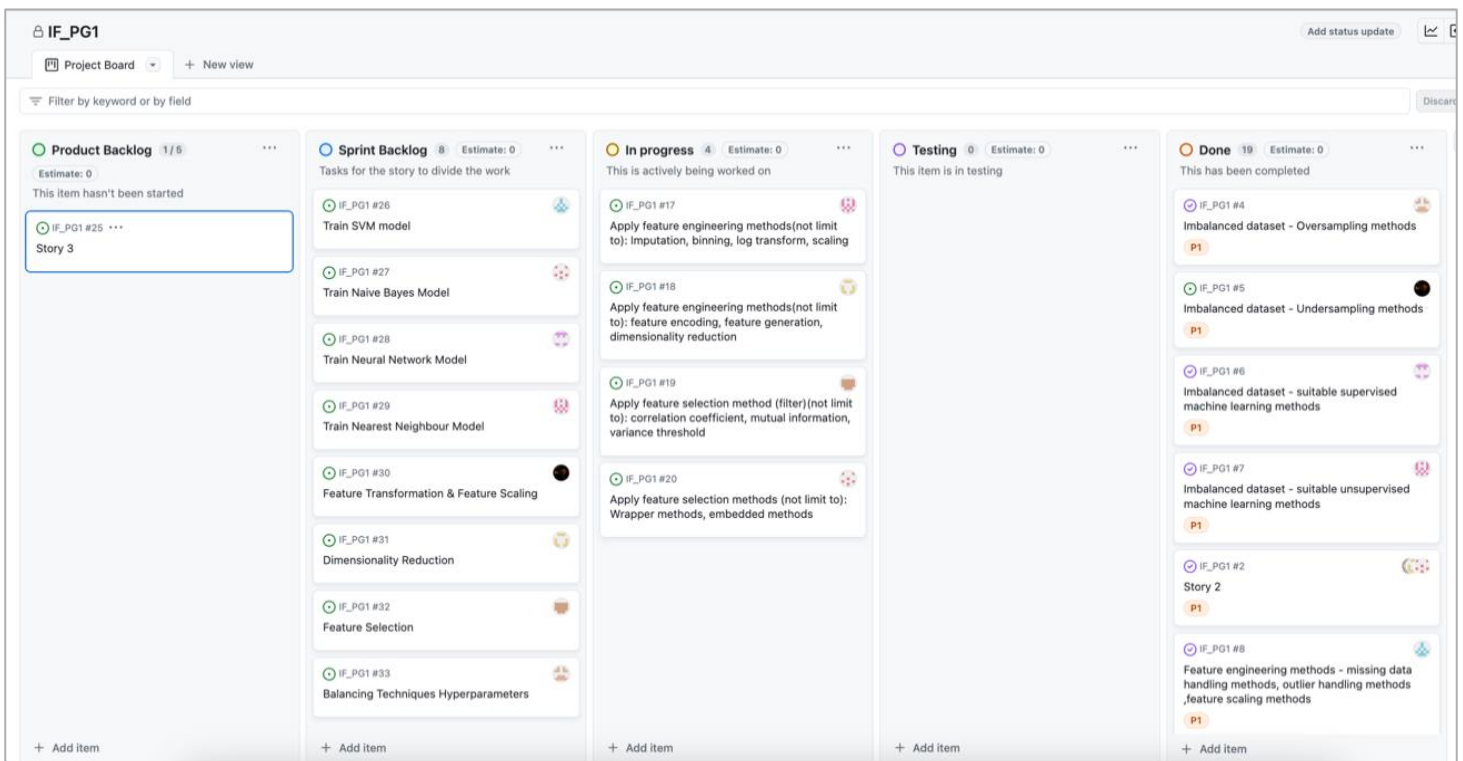


Figure 2. The Screenshot of the Task Board in First Week of Sprint 3 (From Snapshot 3.1)

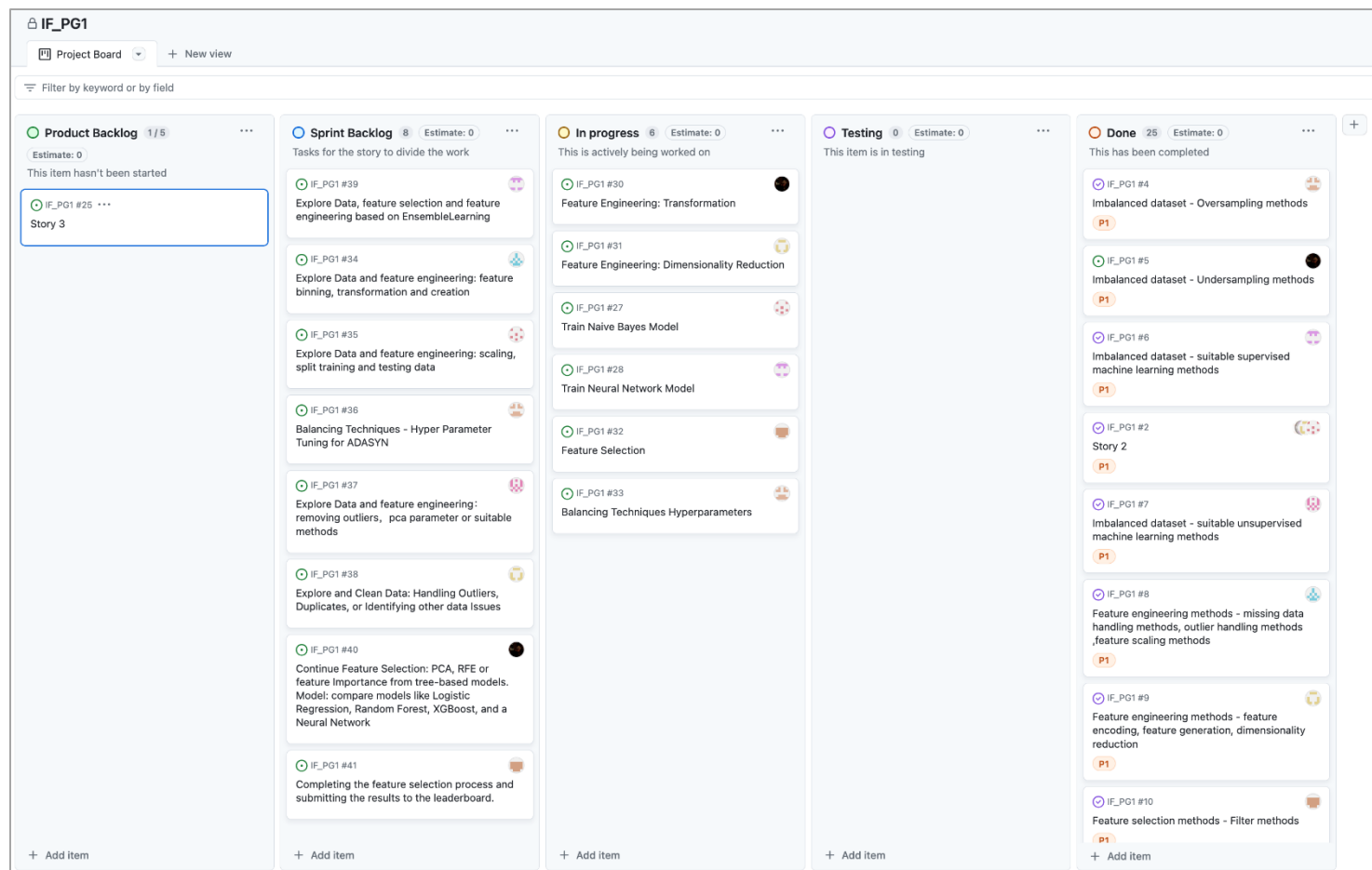


Figure 3. The Screenshot of the Task Board in Second Week of Sprint 3 (From Snapshot 3.2)

1.2 Sprint Backlog and User Stories

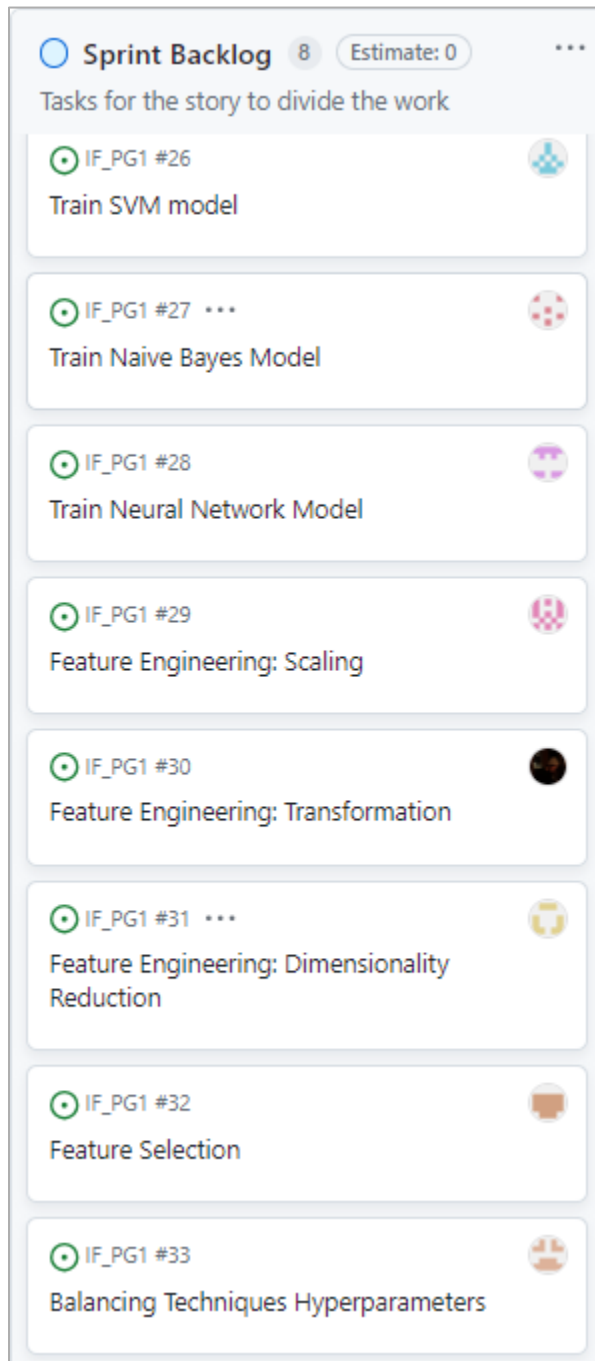


Figure 4. The Screenshot of the Sprint Backlog for First Week of Sprint 3.

(From snapshot 3.1)

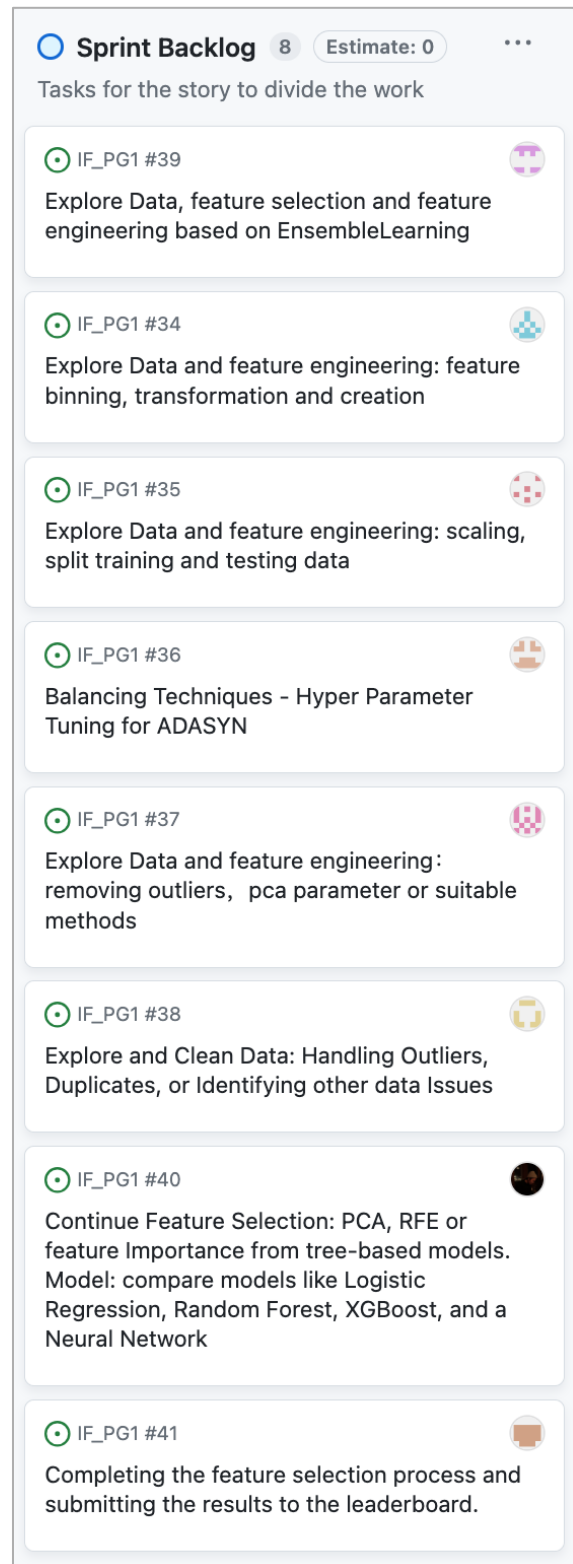


Figure 5. The Screenshot of the Sprint Backlog for Second Week of Sprint 3.

(From snapshot 3.2)

1.3 User Story 3

As a software engineer, I want to tryout:

- 1) different models while fine-tuning their hyperparameters.**
- 2) additional techniques for feature engineering, feature selection, and methods for addressing imbalanced datasets, so that I can improve the test set predictions to achieve an F1 score exceeding 55%.**

Acceptance criteria :

- 1) Fine-tune hyperparameters of:**
 - at least 3 different models
 - feature selection, feature engineering, and imbalanced dataset handling techniques, if they have parameters to tune
- 2) Implement and try out at least 2 more techniques each for:**
 - feature engineering methods
 - feature selection methods
 - ML techniques to approach a problem with an imbalanced dataset
- 3) Report what combination of techniques and what model worked so far**
- 4) Achieve a minimum InsightFactory leaderboard score of 55%**

1.4 Definition of Done (DOD)

Code Standards

- 1) Readability and Maintainability:
 - Code must be written with clarity.
 - All functions and classes should have clear, concise docstrings explaining their purpose, parameters, and return values.
- 2) Testing:
 - The entire pipeline must be able to run successfully.
 - The result of the prediction must be reflected on the leaderboard to signify successful submission.

Documentation

- 1) Reproducibility:
 - Scripts and notebooks must be structured for easy reruns by team members.
- 2) Data Pipeline Documentation:
 - Document the entire pipeline, including data cleaning, feature engineering/selection, and model decisions.
 - Ensure that all assumptions, processes, and justifications are recorded for future reference.
- 3) Leaderboard Score Tracking:
 - Record leaderboard scores during experimentation to guide model improvements.
 - Include dates and versions of models associated with specific scores.

Data Management

- 1) Data Cleaning:
 - Define clear steps for data cleaning (e.g., handling missing values, outliers) and log changes for traceability.
 - Validate data after cleaning to ensure no unexpected data loss or errors.
- 2) Feature Engineering and Selection:
 - Document feature engineering and selection processes, with justification for decisions. Regularly evaluate feature importance.
- 3) Balanced Data Handling:
 - Clearly document the rebalancing techniques used, such as SMOTE, under sampling, or ADASYN, along with the rationale for choosing each method.
 - Measure and log the performance of models before and after rebalancing to ensure the effectiveness of the technique.

Machine Learning Model

- 1) Model Selection and Validation:
 - All selected models must be validated using appropriate techniques, including cross-validation and/or out-of-sample testing.
 - Performance metrics must be calculated for each model and compared against baseline models (e.g., logistic regression).
- 2) Hyperparameter Tuning:
 - Document the hyperparameter tuning process, including which parameters were tuned, the ranges tested, and the final selected values.
- 3) Model Interpretability:
 - If possible, include model interpretability steps (e.g., feature importance, SHAP values) to provide insights into how the model makes predictions.

Review and Sign-Off

- 1) Peer Review:
 - All code and documentation must undergo peer review before being marked as complete.
 - Reviewers must verify that all DoD criteria have been met.
- 2) Sign-Off:
 - The team must collectively sign off on each completed user story, ensuring all acceptance criteria are met and all necessary documentation is provided.

1.5 Summary of Changes

- **For Week 1 of Sprint 3 (Snapshot 3.1)**

In the second sprint, the team initially experimented with various methods and submitted a basic model. As the third sprint progressed, the team gained a deeper understanding of the importance of data cleaning and feature engineering, and the tasks became more specific, aiming to identify the optimal combination of techniques to achieve an F1 score exceeding 55%. Based on this, we adjusted the DoD, adding Leaderboard score tracking to guide continuous model improvement. Additionally, we enhanced the requirements for Data Cleaning, Feature Engineering and Selection. These changes ensure that the DoD more effectively supports the team's efforts in model optimization and performance enhancement.

- **For Week 2 of Sprint 3 (Snapshot 3.2)**

During the first week, we allocated tasks based on different machine learning techniques, feature selection, feature engineering, and balancing methods. However, after noticing limited progress despite trying various approaches, we decided to adjust our strategy. For the second week, each team member independently created and selected tasks they believed would best contribute to our overall goal of achieving an F1 score of above 55%. This approach provided more flexibility and ownership over the work. Aside from this adjustment, no major changes were made for Sprint 3.2.

I attended the sprint planning meetings on Monday, the 2nd of September with my group to do the initial planning for sprint 3. Then, after the first week of the sprint, I attended the sprint meeting on Monday, the 9th of September 2024 and presented my progress to the team. After that, we discussed what should be improved for the next week of the sprint and due to the break, we had our sprint review meeting on Monday, the 30th of September and I presented the results of my work.

2. What went well in the sprint?

During sprint 3, our team was tasked with fine-tuning hyperparameters for at least three models while also exploring methods for feature engineering, feature selection, and handling imbalanced datasets. We divided tasks efficiently, with some members focusing on techniques like PCA, RFE, and tree-based feature importance for feature selection, while others handled hyperparameter tuning for models such as Logistic Regression, Random Forest, XGBoost, and a Neural Network. Additionally, we performed data cleaning and applied feature engineering techniques such as binning, scaling, and transformation.

I was responsible for feature selection using filtering methods, including Mutual Information, Biserial Correlation, and Variance Threshold. This helped us identify the most important features, improving model performance and increasing the F1 score of the validation set to 0.8. Although our goal was a 55% F1 score, we achieved 50%, ranking third on the leaderboard with a score of 0.5079 through collaborative effort.

The combination of feature selection, data processing, and tuning helped us build more accurate models and optimize the software. Our ability to balance feature selection, hyperparameter tuning, and data processing not only improved our models but also strengthened our software development practices, leading to more accurate and reliable predictive models for future sprints.

3. What Could be Improved?

During sprint 3, while many group members and I successfully achieved a high F1 score in the validation set, this improvement was not reflected in the leaderboard score. This discrepancy suggests a need for a deeper understanding of what specific improvements are required to reach our goal of 55%.

One challenge I faced was the instruction from the group leader to focus solely on feature selection and not on hyperparameter tuning. This limited my ability to contribute more effectively to improving the leaderboard score. The iterative feature selection process I was responsible for took around 14 hours per run, and I had to analyze each feature for each model and filtering method through multiple runs and trial and error. This left no time for optimizing hyperparameters, which could have made a more significant impact on our overall performance. Therefore, one area that could be significantly improved is the team's approach to task distribution and balance.

Additionally, the group's overall collaboration could be improved. We often worked individually, and the group's communication style was not always constructive, which made it harder to seek assistance or feedback.

By fostering better communication and encouraging a more collaborative, supportive environment, we could collectively enhance our models and ensure a stronger and more efficient software development process. My experience serves as an example of how clearer direction and better team collaboration could lead to more impactful contributions and improved performance.

4. What will the Group commit to Improve in the Next Sprint?

In the next sprint, our team will commit to improving collaboration, task distribution, and integrating explainable AI techniques to enhance our performance. We will ensure that tasks are more evenly distributed among the team. This balanced approach will optimize model performance and help us achieve better results. Additionally, we will implement regular check-ins to improve communication, making it easier to seek assistance and provide feedback. This will create a more supportive and collaborative environment, ensuring all members are aligned and productive.

We also plan to explore explainable AI techniques to make our models more interpretable. Understanding model behavior will help us make more informed decisions, improving both the accuracy and transparency of our software. By focusing on these improvements, we will strengthen our software development process, making it more efficient and ensuring that we deliver better-performing, reliable models in future sprints.

5. Comment on My Progress This Sprint

In Sprint 3, I was responsible for implementing feature selection techniques using three filtering methods: Mutual Information, Biserial Correlation, and Variance Threshold, applied across various machine learning models. My primary goal was to identify features that significantly impacted the models' F1 scores. I worked with five models: K-Nearest Neighbors (KNN), XGBoost, Random Forest, Logistic Regression, and a Deep Neural Network (DNN). The task involved using selected features to optimize model performance by identifying features that increased or decreased the F1 score. The complete process, results, and analysis of the features and their importance across models are presented below.

5.1 Methodology

The feature selection techniques were applied through a Python script, which included filtering methods such as Mutual Information, Biserial Correlation, and Variance Threshold. Below is an overview of the different parts of the code and the methodology used. The code has been attached to the appendix.

1. **Library Imports:** The code imports essential libraries like NumPy, Matplotlib, and XGBoost for model building and visualization, along with Scikit-learn for feature selection and model evaluation.

2. **Feature Selection:** Three filtering methods were used to rank features based on relevance:

- **Mutual Information:** Assesses the dependency between each feature and the target variable.
- **Biserial Correlation:** Measures the correlation between continuous features and the binary target.
- **Variance Threshold:** Eliminates low-variance features that contribute little to the model's performance.

In the previous sprint the above techniques were used to rank features. For this sprint, I used these rankings to order the input features for each model, with the highest-scoring feature inputted first. Each model was evaluated using these three ordered sequences to assess how feature

prioritization impacted performance. This approach allowed us to identify the most influential features and those that negatively affected the F1 score, guiding feature selection for optimal model performance.

3. **Model Training:** The selected features were evaluated using five models:

- K-Nearest Neighbors (KNN)
- XGBoost
- Random Forest
- Logistic Regression
- Deep Neural Network (DNN)

The process involved iterative training of the models with increasing subsets of features and tracking their F1 scores to determine which features contributed most to model improvement (Increasing Intersections) or model deterioration (Decreasing Intersections).

4. **Handling Imbalanced Datasets:** SMOTE (Synthetic Minority Over-sampling Technique) was applied to address the class imbalance issue, ensuring that the models received balanced data for training.

5. **Model Performance Evaluation:** The F1 score was used as the primary metric to evaluate the models' performance. I plotted the F1 score against the number of features and analyzed how the inclusion of more features influenced the models' predictive capabilities.

6. **Results Visualization:** The F1 scores for each model were plotted against the number of features, and key insights were derived from these graphs. For instance, in the Random Forest model, the best F1 score (0.78) was achieved using 25 features, as identified by the Mutual Information method.

Through my contribution, I helped the team identify the most important and least important features for the predictive maintenance model. This process was essential to improve model performance, as it allowed us to optimize the feature sets for each model. For example, in the KNN model, I identified 'BrakeCylinder', 'Tonnage', and 'IntrainForce' as consistently important features. Conversely, features such as 'VACC_L' and 'Track_Offset' were consistently shown to decrease model performance and were marked for exclusion.

5.2 Results and Analysis

From the feature selection process, the following insights were identified:

- **Most Important Features (Increasing Intersections):** These features consistently improved F1 scores across models. IntrainForce and Tonnage were identified as consistently important across multiple models.
- **Secondly Important Features (Increasing Unions):** These features improved F1 scores in certain models but did not show universal importance across all models.

- **Decreasing Features (Decreasing Unions/ To be used only if necessary):** These features showed a decrease in F1 scores in certain models but were not consistently detrimental across all models. Features such as SND, Twist14m, BodyRockRr, LP2, and Curvature are examples of features that negatively affected performance in some models and should be used cautiously.
- **Least Important Features (Decreasing Intersections):** These features consistently led to lower F1 scores and should be avoided or deprioritized in future training.

The table below outlines the features identified for each model, showing which ones increased the F1 score (Increasing Intersections and Unions) and which ones decreased it (Decreasing Intersections and Unions).

Model	Increasing Intersections (Most Important Features)	Increasing Unions (Secondly Important Features)	Decreasing Unions (Only to be used if needed)	Decreasing Intersections (Avoid Features)
KNN	BrakeCylinder, Twist14m, Tonnage, LP3, Speed, LP1, Acc1_RMS, Rail_Pro_R, IntrainForce	SND, Twist14m, Tonnage, BodyRockRr, LP2, Curvature, SND_R, VACC, BrakeCylinder, SND_L, BodyRockFrt, Speed, Rail_Pro_L, VACC_R, Acc1_RMS, IntrainForce, Twist2m, Acc4, Acc3_RMS, BounceRr, Acc4_RMS, Rail_Pro_R, Acc2_RMS, Acc3, LP3, LP1, Acc1, Acc2	SND, Acc4, Acc3_RMS, BodyRockRr, LP4, LP2, BounceRr, Curvature, Acc4_RMS, SND_R, VACC, Acc2_RMS, Acc3, VACC_L, SND_L, BodyRockFrt, Rail_Pro_L, BounceFrt, Track_Offset, VACC_R, Acc1, Twist2m, Acc2	VACC_L, LP4, Track_Offset
XGBoost	BrakeCylinder, Acc3, Tonnage, Twist14m, BodyRockFrt, Speed,	SND, Twist14m, Tonnage, BodyRockRr, LP4, LP2, Curvature,	SND, Acc4, Acc3_RMS, BodyRockRr, LP2, BounceRr, Curvature,	Acc1_RMS, SND_L

	Rail_Pro_L, BounceFrt, Rail_Pro_R, Acc1, IntrainForce	SND_R, VACC, BrakeCylinder, VACC_L, BodyRockFrt, Speed, BounceFrt, Rail_Pro_L, VACC_R, IntrainForce, Twist2m, Acc4, Acc3_RMS, BounceRr, Acc4_RMS, Rail_Pro_R, Acc2_RMS, Acc3, LP3, Track_Offset, LP1, Acc1, Acc2	Acc4_RMS, SND_R, VACC, Acc2_RMS, VACC_L, LP3, SND_L, Track_Offset, LP1, VACC_R, Acc1_RMS, Acc2, Twist2m, LP4	
Random Forest	BrakeCylinder, Twist14m, Tonnage, SND_L, Speed, Track_Offset, Curvature, Acc4_RMS, Acc1_RMS, IntrainForce	SND, Twist14m, Tonnage, BodyRockRr, LP4, LP2, Curvature, SND_R, BrakeCylinder, VACC_L, SND_L, BodyRockFrt, Speed, Rail_Pro_L, BounceFrt, VACC_R, Acc1_RMS, IntrainForce, Twist2m, Acc3_RMS, BounceRr, Acc4_RMS, Rail_Pro_R, Acc2_RMS, Acc3, LP3, Track_Offset, LP1, Acc1, Acc2	SND, Acc4, Acc3_RMS, BodyRockRr, LP2, BounceRr, Rail_Pro_R, SND_R, VACC, Acc2_RMS, Acc3, VACC_L, LP3, BodyRockFrt, BounceFrt, Rail_Pro_L, LP1, VACC_R, Acc2, Acc1, Twist2m, LP4	VACC, Acc4
Logistic Regression	BrakeCylinder, Tonnage, SND_L, Track_Offset,	SND, Twist14m, Tonnage, BodyRockRr,	SND, Twist14m, BodyRockRr, LP2,	Acc4, Rail_Pro_L, Acc4_RMS,

	Rail_Pro_R, IntrainForce	LP2, Curvature, SND_R, VACC, BrakeCylinder, VACC_L, SND_L, BodyRockFrt, Speed, BounceFrt, Rail_Pro_R, VACC_R, IntrainForce, Twist2m, Acc3_RMS, BounceRr, Acc2_RMS, Acc3, LP3, Track_Offset, LP1, Acc2, LP4	Curvature, SND_R, VACC, VACC_L, BodyRockFrt, BounceFrt, Rail_Pro_L, Speed, VACC_R, Acc1_RMS, Twist2m, Acc4, Acc3_RMS, BounceRr, Acc4_RMS, Acc2_RMS, Acc3, LP3, LP1, Acc2, Acc1, LP4	Acc1_RMS, Acc1
DNN	Twist14m, Acc3, Tonnage, Speed, LP1, VACC_R, IntrainForce	SND, Twist14m, Tonnage, BodyRockRr, LP2, Curvature, SND_R, VACC, BrakeCylinder, VACC_L, SND_L, BodyRockFrt, Speed, Rail_Pro_L, BounceFrt, VACC_R, Acc1_RMS, IntrainForce, Twist2m, Acc4, Acc3_RMS, BounceRr, Acc4_RMS, Rail_Pro_R, Acc2_RMS, Acc3, LP3, Track_Offset, LP1, Acc2, Acc1, LP4	SND, Acc4, Acc3_RMS, BodyRockRr, LP4, LP2, BounceRr, Curvature, Acc4_RMS, Rail_Pro_R, SND_R, VACC, Acc2_RMS, BrakeCylinder, VACC_L, LP3, SND_L, BodyRockFrt, BounceFrt, Rail_Pro_L, Track_Offset, Acc1_RMS, Acc1, Twist2m, Acc2	None

Table 1: Final Results of Feature Selection for Each Model

5.2 Graphical Results of Feature Selection

1. Deep Neural Network (DNN) Model

The DNN model's performance improved as more features were added. The highest F1 score was achieved with 28 features using Biserial Correlation, 32 features with Mutual Information, and 26 features with Variance Threshold. After 25 features, performance gains diminished.

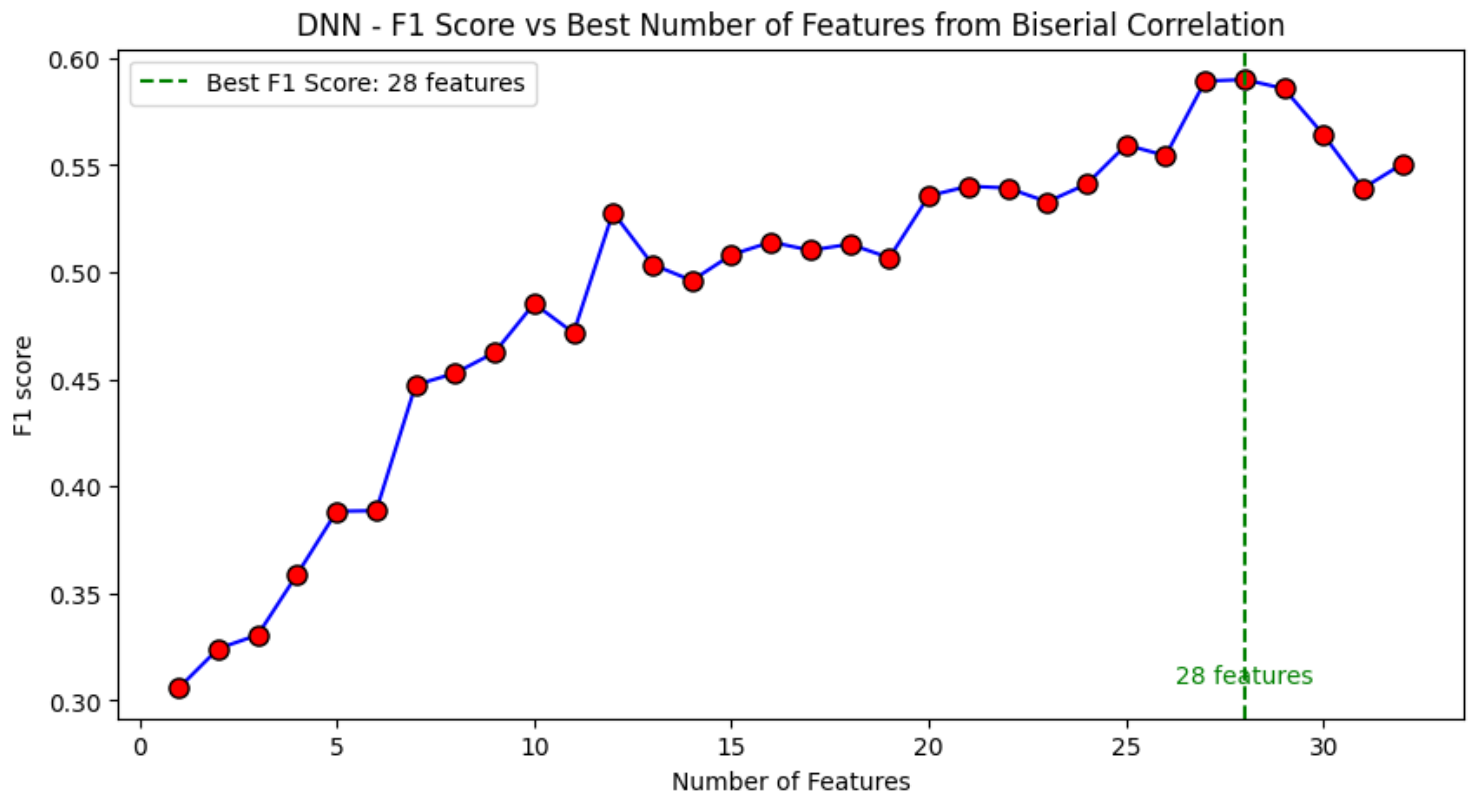


Figure 6: DNN - F1 Score vs. Number of Features (Biserial Correlation)

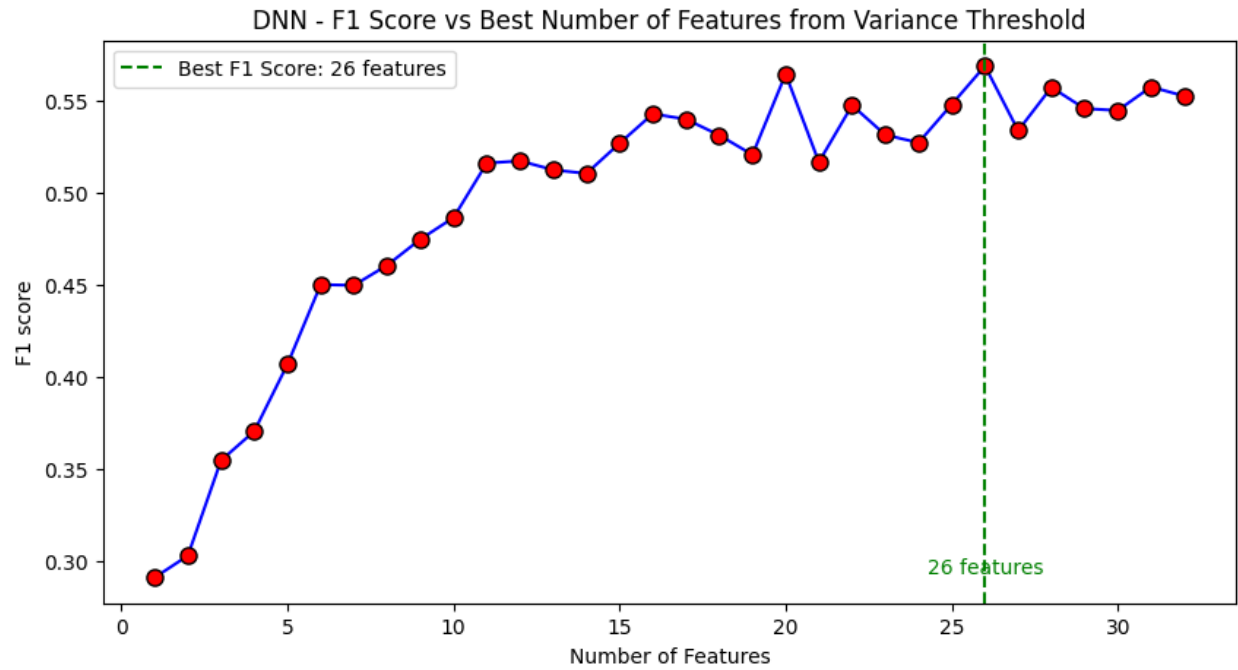


Figure 7: DNN - F1 Score vs. Number of Features (Variance Threshold)

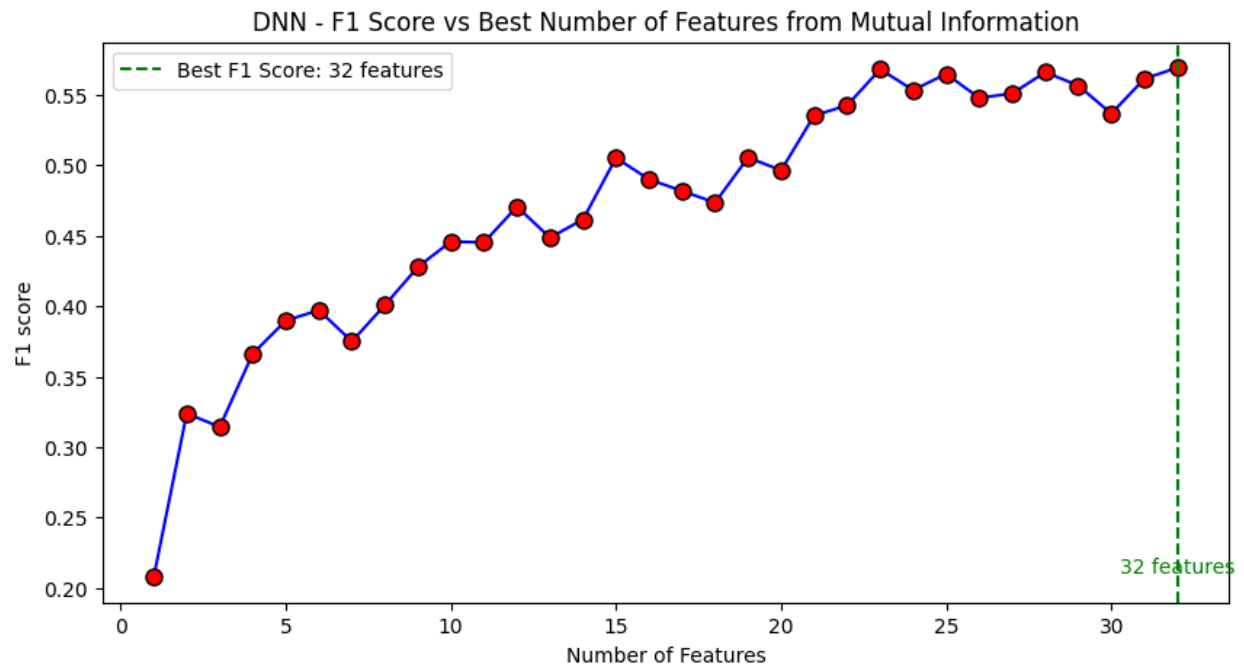


Figure 8: F1 Score vs. Number of Features (Mutual Information)

2. K-Nearest Neighbors (KNN) Model

The KNN model exhibited optimal performance with Biserial Correlation and Mutual Information at 29 features. However, for the Variance Threshold method, the best F1 score was achieved with only 20 features. This demonstrates that the optimal number of features for KNN varies significantly based on the feature selection method used.

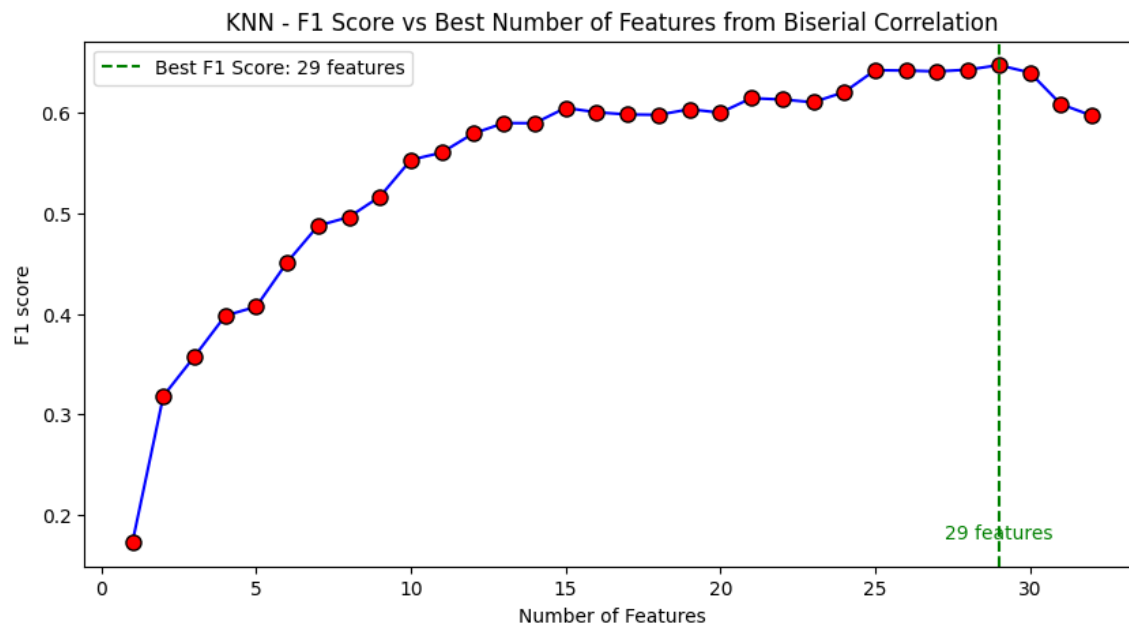


Figure 9: KNN - F1 Score vs Best Number of Features from Biserial Correlation

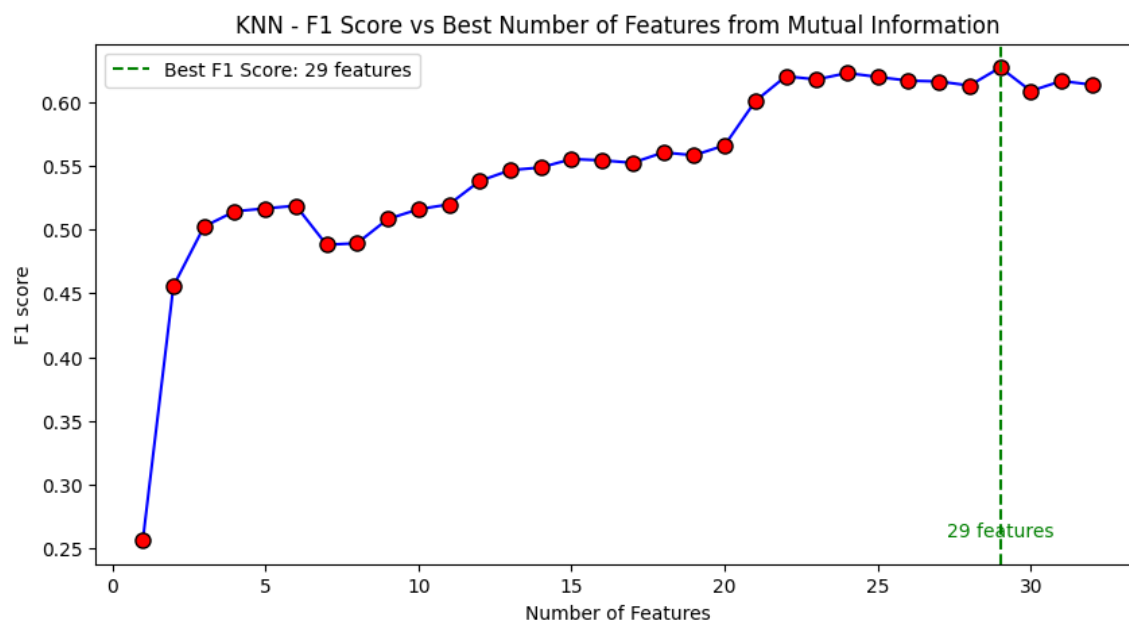


Figure 10: KNN - F1 Score vs Best Number of Features from Mutual Information

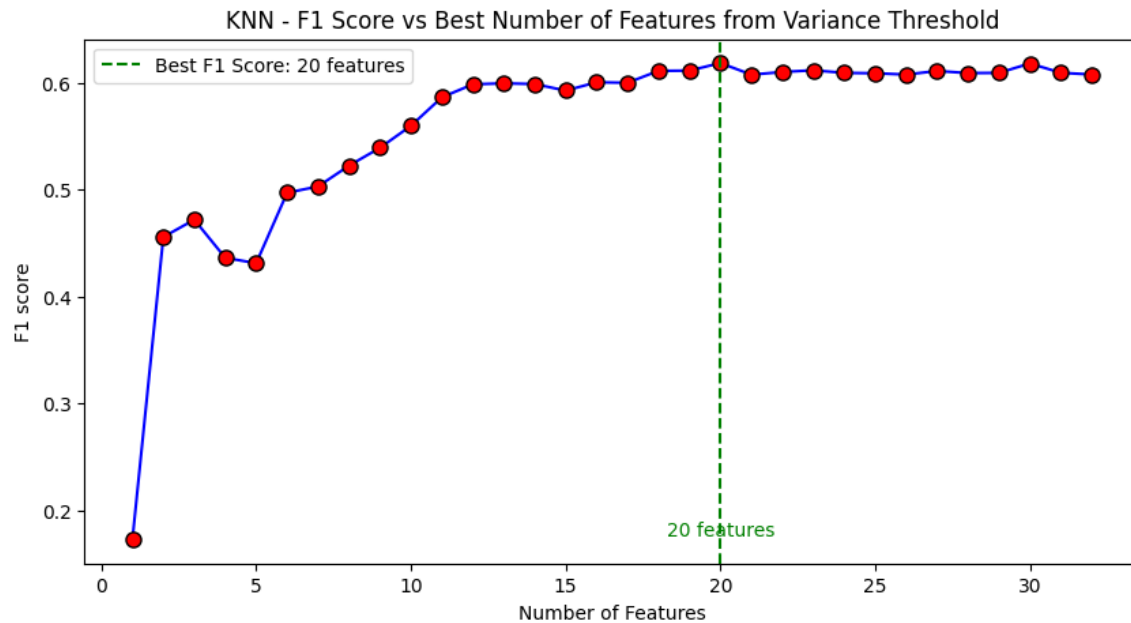


Figure 11: KNN - F1 Score vs Best Number of Features from Variance Threshold

3. Logistic Regression Model

For Logistic Regression, the optimal number of features varied more widely between methods. Biserial Correlation gave the best performance with 13 features, while Mutual Information required 22 features for the best F1 score, and the Variance Threshold method performed best with 32 features.

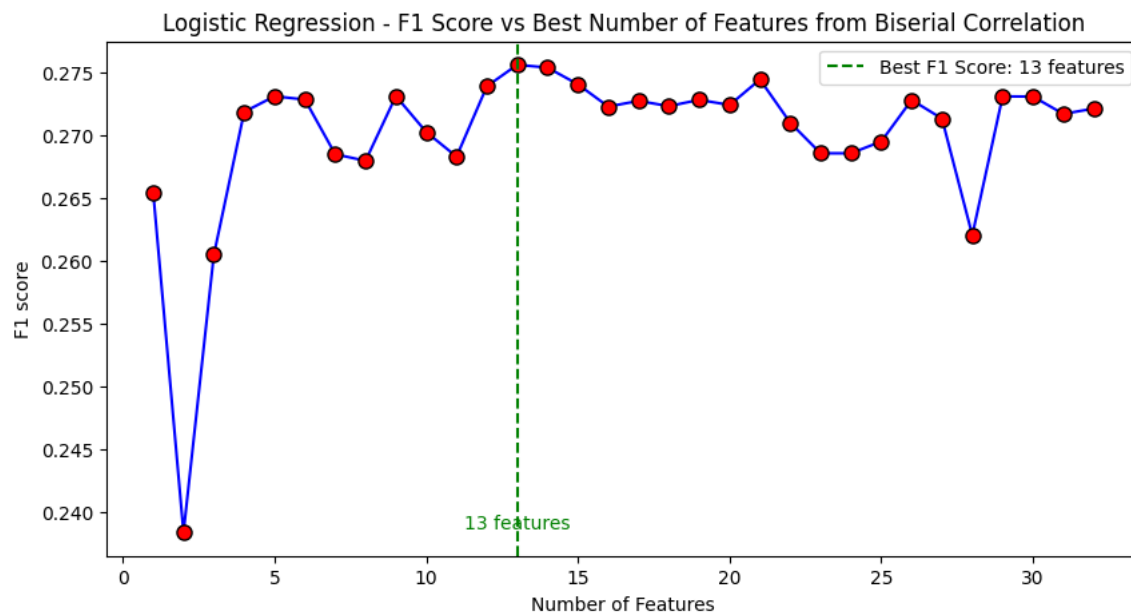


Figure 12: Logistic Regression - F1 Score vs Best Number of Features from Biserial Correlation

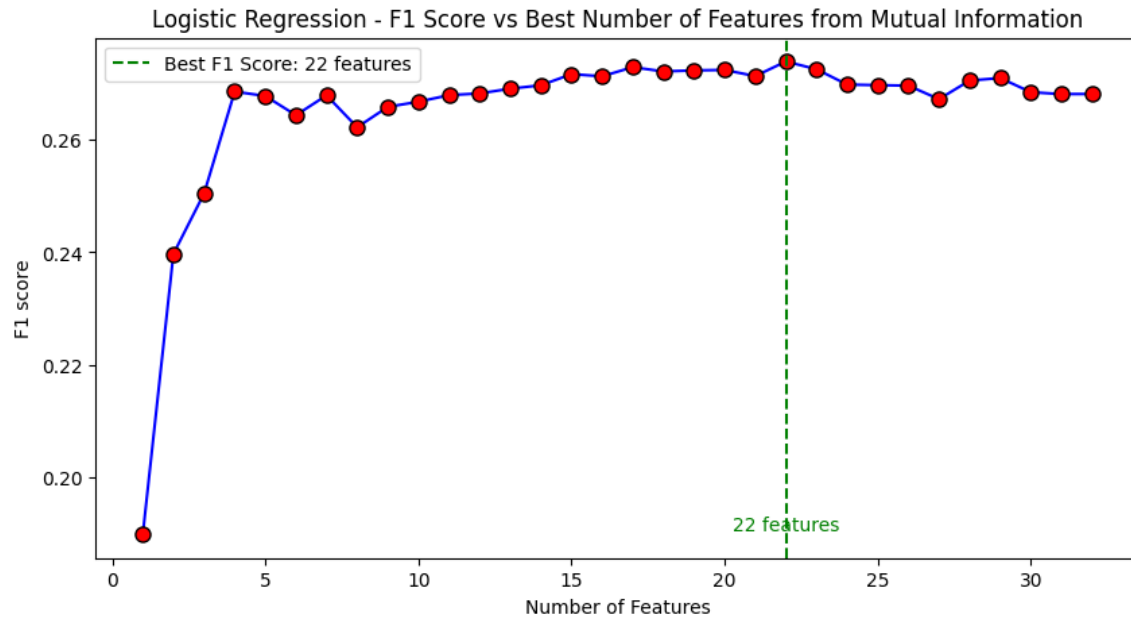


Figure 13: Logistic Regression - F1 Score vs Best Number of Features from Mutual Information

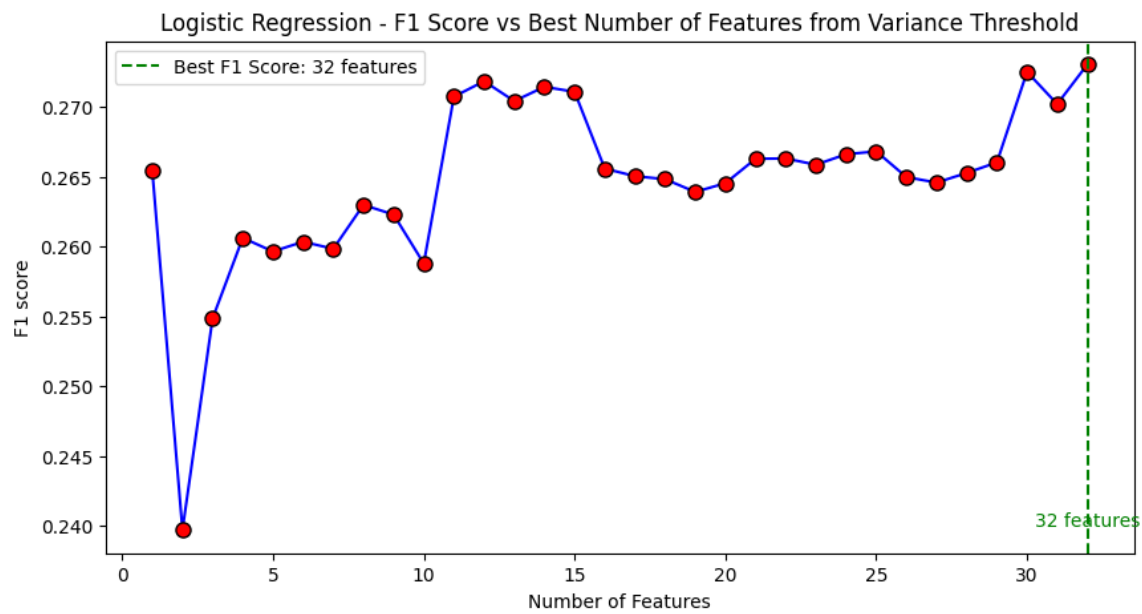


Figure 14: Logistic Regression - F1 Score vs Best Number of Features from Variance Threshold

4. Random Forest Model

The Random Forest model showed robust performance across all feature selection methods. Biserial Correlation indicated an optimal number of 30 features, Mutual Information suggested 25 features, and Variance Threshold resulted in the best performance with 32 features.

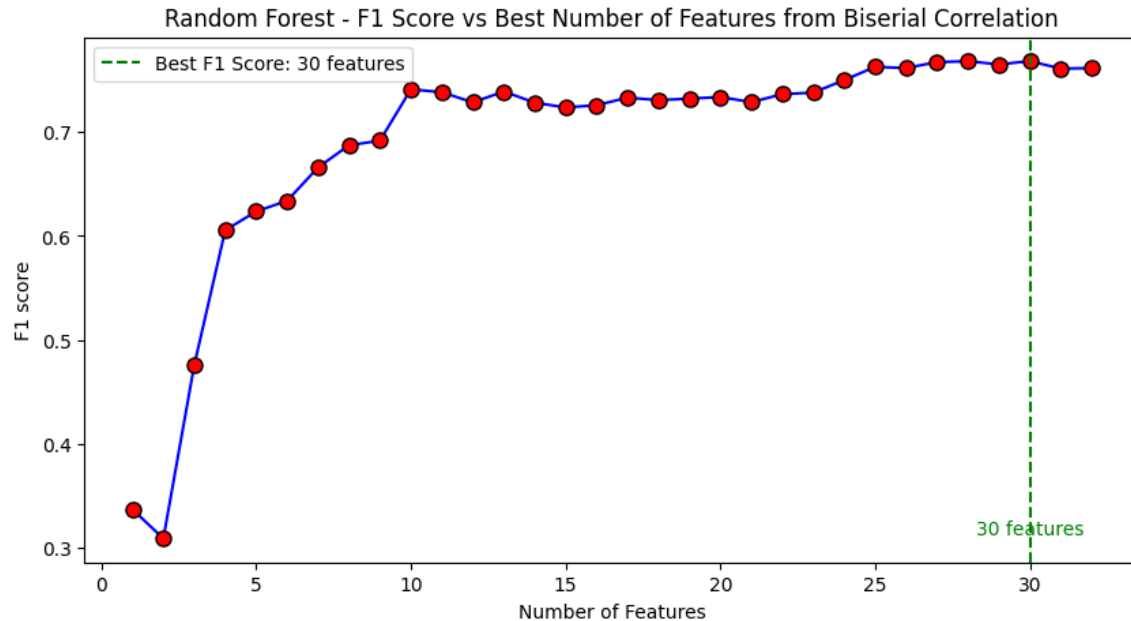


Figure 15: Random Forest - F1 Score vs Best Number of Features from Biserial Correlation.

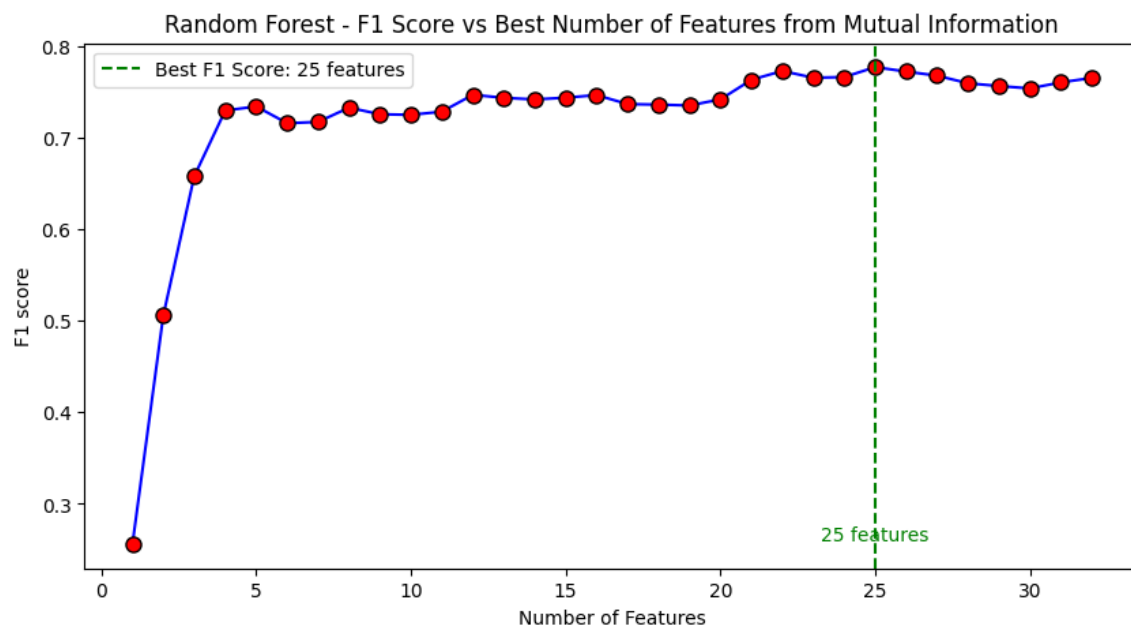


Figure 16: Random Forest - F1 Score vs Best Number of Features from Mutual Information.

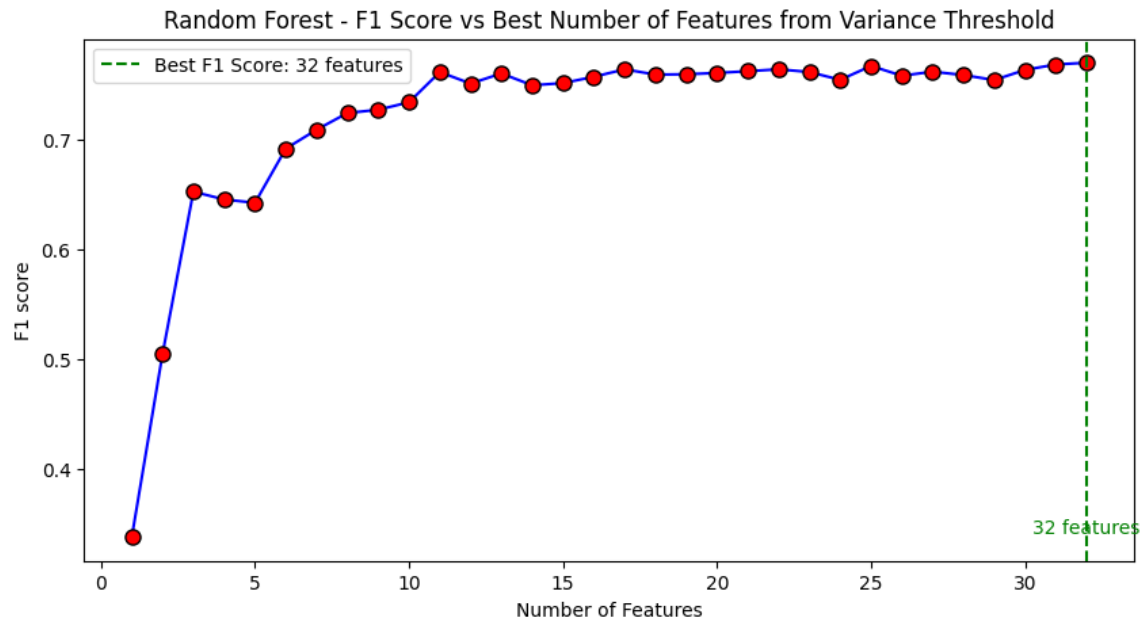


Figure 17: Random Forest - F1 Score vs Best Number of Features from Variance Threshold.

5. XG Boost Model

XG Boost achieved the best F1 score using Biserial Correlation with 26 features, while Mutual Information and Variance Threshold required 32 features to reach optimal performance.

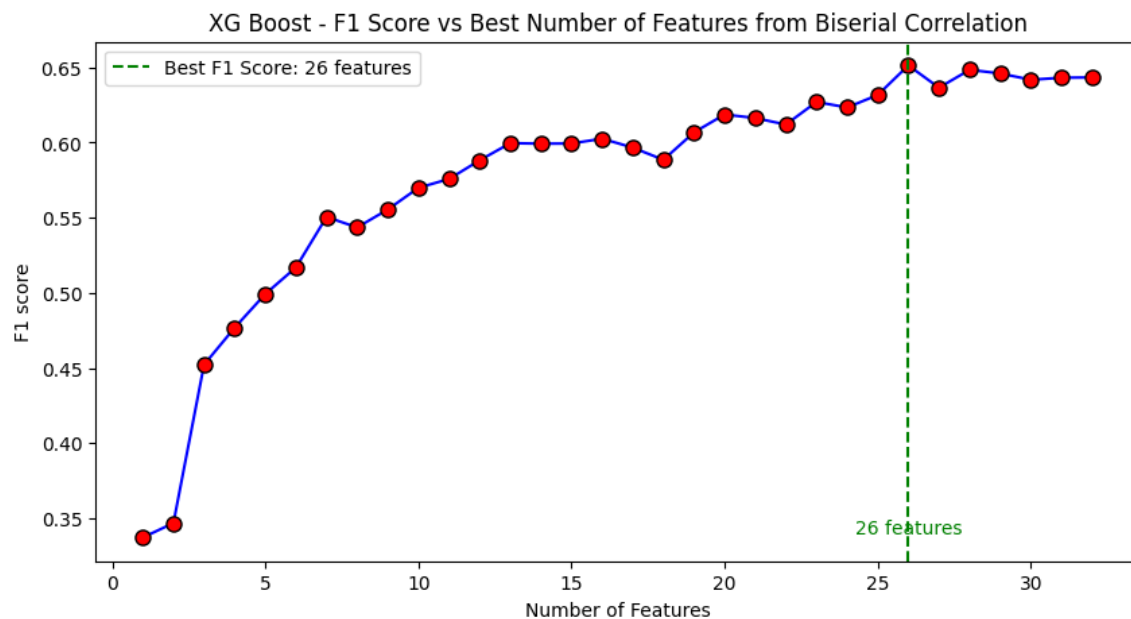


Figure 18: XG Boost - F1 Score vs Best Number of Features from Biserial Correlation.

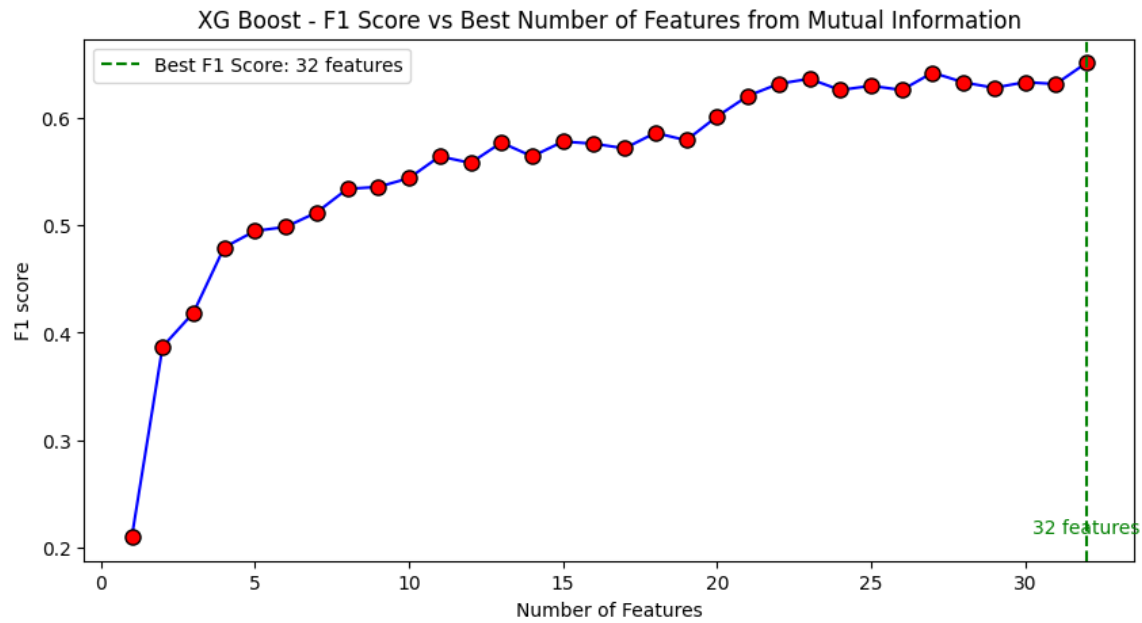


Figure 19: XG Boost - F1 Score vs Best Number of Features from Mutual Information.

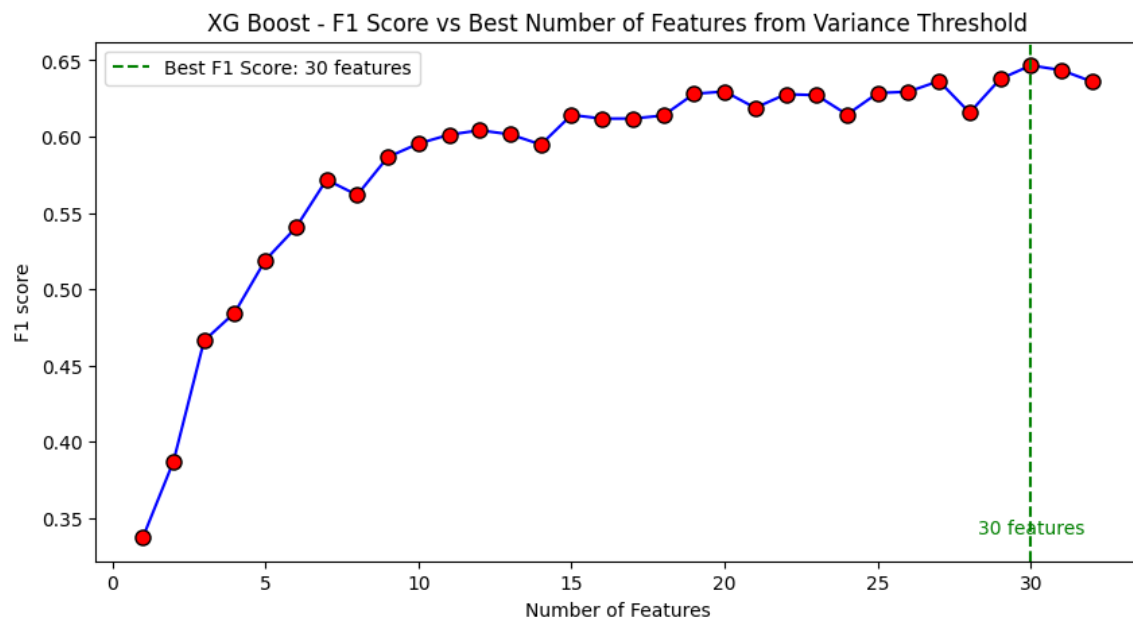


Figure 20: XG Boost - F1 Score vs Best Number of Features from Variance Threshold.

5.3 Overall Results of Feature Selection

The feature selection process provided valuable insights into the features that consistently improved or hindered model performance across all models (KNN, Logistic Regression, Random Forest, and XG Boost) and all feature selection methods (Biserial Correlation, Mutual Information, and Variance Threshold). The results can be categorized as follows:

- **Increasing Intersections (Most Important Features):** These features consistently improved F1 scores across all models and all feature selection methods. *IntrainForce* and *Tonnage* were identified as the most important features, playing a crucial role in improving predictive performance across the board.
- **Increasing Unions (Secondly Important Features):** These features showed improvements in F1 scores for certain models or methods but were not universally important across all cases. Features such as *SND*, *Twist14m*, *BodyRockRr*, *LP4*, and *Curvature* contributed positively to specific models and methods but did not consistently enhance performance in all situations.
- **Decreasing Unions (To be used only if necessary):** These features generally had a negative impact on F1 scores in some models or methods, though not across the entire set. Features like *SND*, *Twist14m*, *BodyRockRr*, and *LP2* were identified as potentially detrimental in certain contexts and should only be considered if more important features are insufficient.
- **Decreasing Intersections (Least Important Features - Avoid):** While no features were identified as universally harmful across all models and methods, certain features, such as *LP4* and *Track_Offset*, showed negative effects in specific instances and should be deprioritized wherever possible.

In this sprint, I contributed to the group by sharing all my findings from the feature selection process, which were derived from the comprehensive analysis of all models and feature selection methods. These insights were made available to the entire team, allowing everyone to apply them and improve F1 scores and our leaderboard ranking, in alignment with the sprint's user story.

5.4 KNN Model Submission to Insight Factory Leaderboard

For this sprint, I submitted a K-Nearest Neighbors (KNN) model to the Insight Factory leaderboard. Initially, without feature selection, the model achieved an F1 score of around 0.6. After applying feature selection and running the model with only the selected features, the F1 score improved significantly to 0.832. This demonstrates the positive impact of feature selection in enhancing model performance. However, despite this improvement in the F1 score, the leaderboard score remained unchanged. This suggests that the leaderboard scoring may consider additional factors beyond the F1 score. Nonetheless, the selected features are valuable findings and can be utilized by the group to improve other models, potentially leading to better leaderboard rankings in future submissions.

```
3 days ago (6s)

## import your training data
df = spark.sql("""select * from feature_selection_2.training_table_1""")
pandas_df = df.toPandas()
pandas_df = pandas_df.drop(columns=['p_key', 'ICWVehicle'])
X = pandas_df.drop(columns=['target'])
X = X[[
    # 'Acc4', 'Twist14m', 'Tonnage', 'Speed', 'Rail_Pro_L', 'Acc4_RMS', 'Rail_Pro_R'
    'BrakeCylinder', 'Twist14m', 'Tonnage', 'LP3', 'Speed', 'LP1', 'Acc1_RMS', 'Rail_Pro_R', 'IntrainForce'
]]
y = pandas_df['target']

(2) Spark Jobs
df: pyspark.sql.connect.dataframe.DataFrame = [p_key: string, Twist14m: double ... 33 more fields]
```

Figure 21: Screenshot of how the feature have been selected for KNN model

```
# Train-Test Split with Stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Apply MICE imputation
imputer = IterativeImputer()
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Standardize the Features (standardize only after resampling)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_imputed)
X_test_scaled = scaler.transform(X_test_imputed)

# Hybrid Sampling using SMOTEENN (resample the standardized training set)
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled, y_train)

# KNN Model
KNN_model = KNeighborsClassifier(n_neighbors=5)
KNN_model.fit(X_train_resampled, y_train_resampled)
```

Figure 22: Screenshot of the code for KNN model training which was submitted to leaderboard.

```
✓ 12.01 seconds

# 8. Make Predictions on Test Set
pipe = Pipeline([
    ('imputer', imputer),
    ('scaler', scaler),
    ('model', KNN_model)
])
y_pred = pipe.predict(X_test_scaled)

# Evaluate the model using F1-score
f1 = f1_score(y_test, y_pred, average='weighted') # 'weighted' accounts for label imbalance
classification_rep = classification_report(y_test, y_pred)


print(f"F1-score on Test Set: {f1}")
print(f"Classification Report:\n(classification_rep)")

/databricks/python/lib/python3.10/site-packages/sklearn/base.py:450: UserWarning: X does not have valid
warnings.warn(
/databricks/python/lib/python3.10/site-packages/sklearn/neighbors/_classification.py:237: FutureWarning:
et `keepdims` to True or False to avoid this warning.
mode, _ = stats.mode(y[neigh_ind, k], axis=1)
F1-score on Test Set: 0.8225482511964106
Classification Report:
      precision    recall  f1-score   support

      0       0.89       0.94       0.91       12261
      1       0.16       0.09       0.12        1579

 accuracy       0.84       0.84       0.84       13840
 macro avg       0.53       0.52       0.52       13840
weighted avg       0.81       0.84       0.82       13840
```

Figure 23: Screenshot of the KNN model results from validation set – F1 score increased to 0.82 after feature selection






RANKING			
1	nSum	0.5378 	0.2259
2	Best group	0.5129 	0.0000
3	IF_PG1	0.5079 	0.0360

Figure 24: Screenshot of the leaderboard results after KNN model submission.

The leaderboard results indicated no change in ranking despite an improved F1 score of 0.832. This suggests that further investigation is needed into the metrics used by the leaderboard to rank submissions. The feature selection process has clearly proven its value, and these selected features can be applied to enhance other models for potential improvements in both F1 score and leaderboard position.

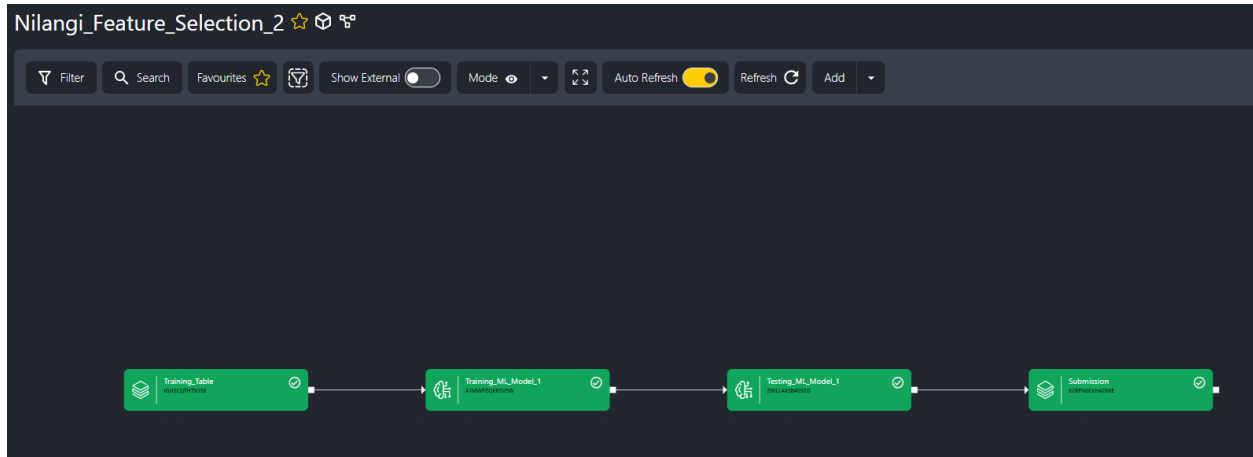


Figure 25: Screenshot of the pipeline created for KNN model submission.

5.4 GitHub Issues Covered:

I successfully completed the following issues assigned on GitHub during this sprint:

1. Feature Selection (#32): Applied feature selection techniques (Biserial Correlation, Mutual Information, Variance Threshold) across multiple models (KNN, XGBoost, Random Forest, Logistic Regression, and DNN), identifying the most impactful features to improve F1 scores.
2. Feature Selection Completion and Leaderboard Submission (#41): Finalized the feature selection process, which contained a detailed analysis including intersections and unions of features that increased and decreased f1 scores, and shared the results with the group.

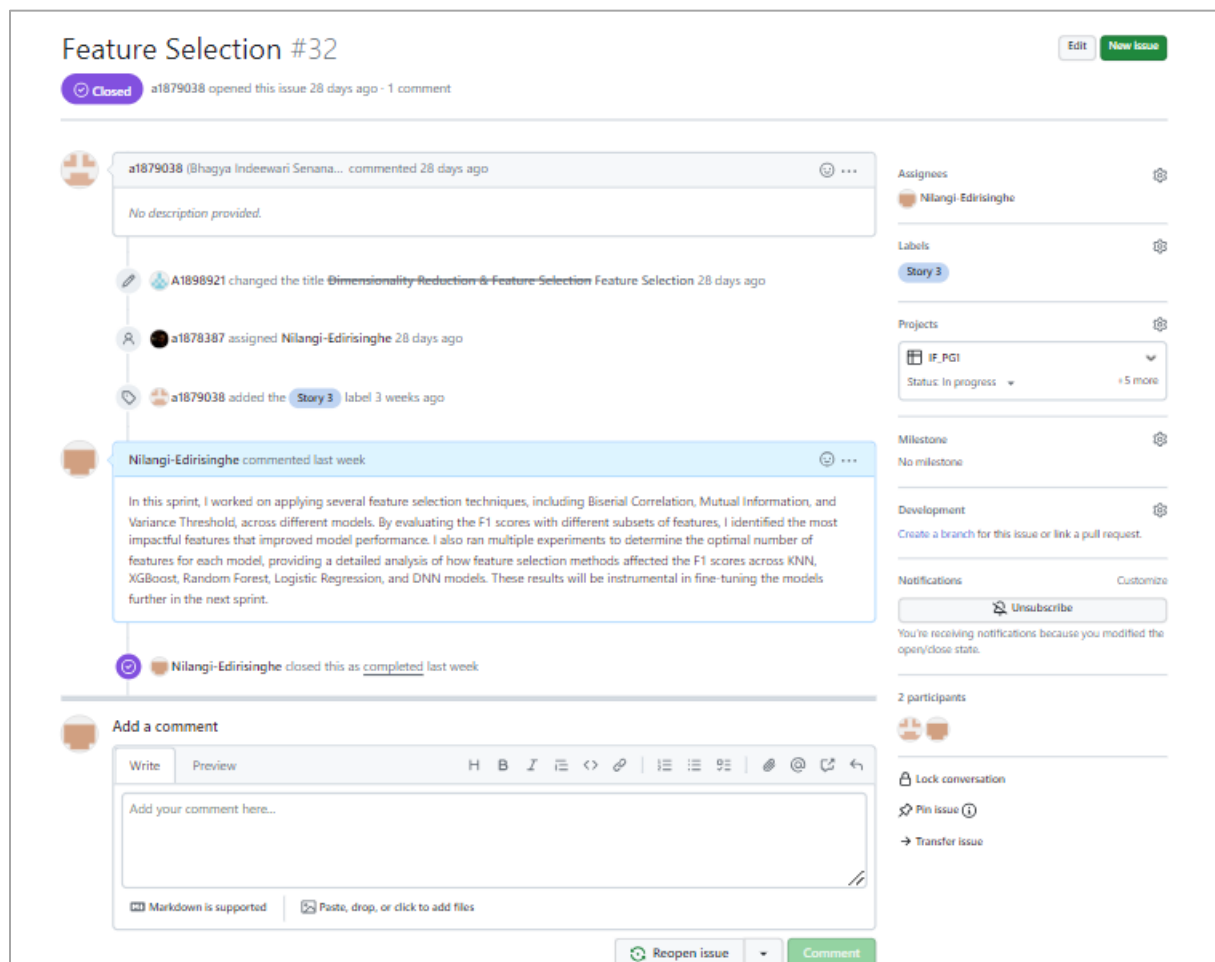


Figure 26: Screenshot of GitHub Issue #32 - Feature Selection.

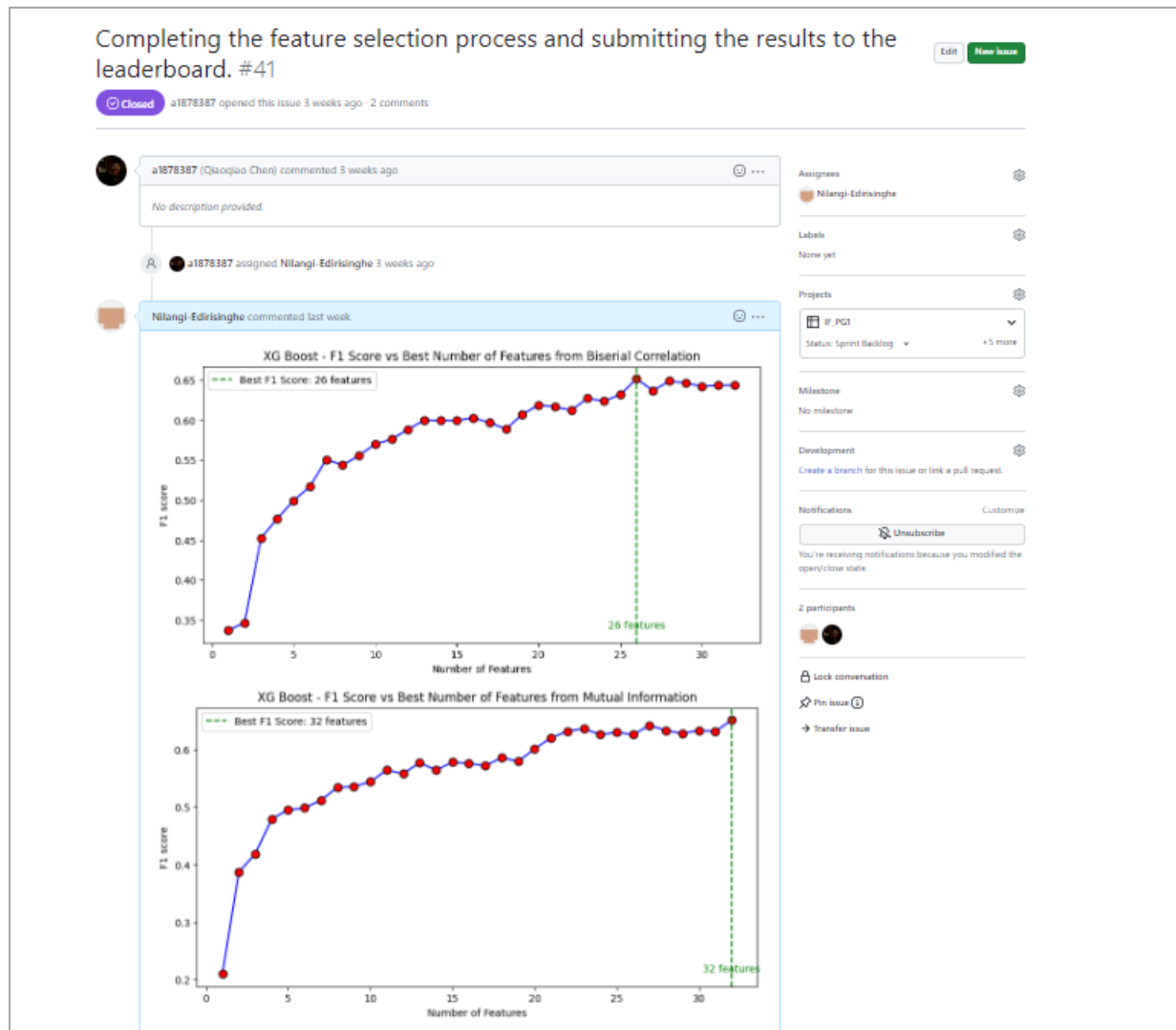


Figure 27: Screenshot of GitHub Issue #41 - Completing the Feature Selection Process and Submitting to the Leaderboard.

6. Requirements Changes

There have been no significant requirement changes since the last sprint. However, some areas of the project could potentially lead to requirement adjustments in the future. One such area is the correlation between the F1 score and the leaderboard ranking on the Insight Factory platform. During this sprint, we observed that while the F1 score improved after applying feature selection (from 0.6 to 0.832), the leaderboard score remained unchanged. This disconnect could prompt a change in the evaluation criteria or prioritization of performance metrics for future submissions.

Additionally, as the team delves deeper into feature engineering and model explainability, new requirements might emerge to ensure that our models are interpretable and provide actionable insights. This would affect project planning by necessitating more focus on model explainability techniques (such as SHAP or LIME) and could introduce new acceptance criteria for the project deliverables. Thus, future sprints may need to allocate time for additional exploratory analysis and implementation of interpretability methods.

7. Appendix

Code for Iterative Process Conducted for Feature Selection and Analysis:

```
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import warnings
import scipy.stats

# Suppress specific FutureWarnings
warnings.filterwarnings("ignore", category=FutureWarning, module="scipy.stats")

# Alternatively, if you're using sklearn, suppress warnings for the entire sklearn module
warnings.filterwarnings("ignore", category=FutureWarning, module="sklearn")

# If you want to suppress all warnings (not recommended for production)
warnings.filterwarnings("ignore")

def create_dnn_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid')) # Binary classification output
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

def plot_graphs(models, feature_dict):
    features = feature_dict["features"]
    method = feature_dict["method"]
    increasing_features_dict = {}
    decreasing_features_dict = {}
    for name, model in models:
        f1_scores = []
        num_features = []
        increasing_features = []
        decreasing_features = []
```

```

previous_f1 = 0
for i in range(1, len(features)+1):
    X,y = get_X_y(df_sel,feature_list=features[:i])
    # 2. Train-Test Split with Stratification
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)

    # 3. Apply MICE imputation
    imputer = IterativeImputer()
    X_train_imputed = imputer.fit_transform(X_train)
    X_test_imputed = imputer.transform(X_test)

    # 4. Standardize the Features (standardize only after resampling)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train_imputed)
    X_test_scaled = scaler.transform(X_test_imputed)

    # 5. Hybrid Sampling using SMOTEENN (resample the standardized training
set)
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train_scaled,
y_train)

    # Model
    if name == 'DNN':
        dnn_model = create_dnn_model(X_train_resampled.shape[1])
        dnn_model.fit(X_train_resampled, y_train_resampled, epochs=100,
batch_size=32, verbose=0)
        y_pred = (dnn_model.predict(X_test_scaled)>0.5).astype(int)
    else:
        model.fit(X_train_resampled, y_train_resampled)
        # Predict on the test set
        y_pred = model.predict(X_test_scaled)

    f1 = f1_score(y_test, y_pred)
    f1_scores.append(f1)
    num_features.append(i)

    if f1 > previous_f1:
        increasing_features.append(features[i-1])
    else:
        decreasing_features.append(features[i-1])

```

```

        previous_f1 = f1

    increasing_features_dict [f'{name}_{method}'] = increasing_features
    decreasing_features_dict [f'{name}_{method}'] = decreasing_features

    # Find the best F1 score and corresponding number of features
    max_f1_index = np.argmax(f1_scores)
    best_f1_score = f1_scores[max_f1_index]
    best_num_features = num_features[max_f1_index]

    print(f'{name} - Best F1 score: {best_f1_score} with {best_num_features}
features')

    # Plot the F1 scores
    plt.figure(figsize=(10,5))
    plt.plot(num_features, f1_scores, marker='o', linestyle='--',
markerfacecolor='r', markersize=8, color='b', markeredgecolor='black')
    plt.xlabel('Number of Features')
    plt.ylabel('F1 score')
    plt.title(f'{name} - F1 Score vs Best Number of Features from {method}')
    # plt.gca().invert_xaxis()

    # Add a vertical stem line at the best number of features
    plt.axvline(x=best_num_features, color='green', linestyle='--', label=f'Best
F1 Score: {best_num_features} features')

    # Add text annotation for the number of features at the bottom of the plot
    plt.text(best_num_features, min(f1_scores), f'{best_num_features} features',
            ha='center', va='bottom', color='green')

    # Show the plot with the stem line and annotation
    plt.legend()
    plt.show()
    print(f'Features where F1 Score Decreased in {name} model using
{method}:{decreasing_features}')
    print(f'Features where F1 Score Increased in {name} model using
{method}:{increasing_features}')

    return increasing_features_dict, decreasing_features_dict

def compute_interception_union_for_model(increasing_features_dict,
decreasing_features_dict, models):
    results = {}

```

```

    for model_name, _ in models:
        increasing_features_for_model = {k:v for k, v in
increasing_features_dict.items() if k.startswith(model_name)}
        decreasing_features_for_model = {k:v for k, v in
decreasing_features_dict.items() if k.startswith(model_name)}

        increasing_intersection = set.intersection(*[set(v) for v in
increasing_features_for_model.values()]) if increasing_features_for_model else set()
        decreasing_intersection = set.intersection(*[set(v) for v in
decreasing_features_for_model.values()]) if decreasing_features_for_model else set()

        increasing_union = set.union(*[set(v) for v in
increasing_features_for_model.values()]) if increasing_features_for_model else set()
        decreasing_union = set.union(*[set(v) for v in
decreasing_features_for_model.values()]) if decreasing_features_for_model else set()

        results[model_name] = {'increasing_intersection':increasing_intersection,
'increasing_union':increasing_union, 'decreasing_intersection':decreasing_intersection,
'decreasing_union':decreasing_union}
    return results

def compute_interception_union_overall(increasing_features_dict,
decreasing_features_dict):
    increasing_intersection = set.intersection(*[set(v) for v in
increasing_features_dict.values()]) if increasing_features_dict else set()
    decreasing_intersection = set.intersection(*[set(v) for v in
decreasing_features_dict.values()]) if decreasing_features_dict else set()

    increasing_union = set.union(*[set(v) for v in
increasing_features_dict.values()]) if increasing_features_dict else set()
    decreasing_union = set.union(*[set(v) for v in
decreasing_features_dict.values()]) if decreasing_features_dict else set()

    print("-----Overall Results of Feature Selection-----")
    print(f'Increasing Intersections (Most Important Features Set):
{increasing_intersection}')
    print(f'Increasing Unions (Secondly Important Features Set): {increasing_union}')
    print(f'Decreasing Unions (Only to be used if above is not sufficient):
{decreasing_union}')
    print(f'Decreasing Intersections (Least Important Features - Avoid these
features): {decreasing_intersection}')
    print('\n')

```



```

models = [
    ('KNN', KNeighborsClassifier(n_neighbors=5)),
    ('XG Boost', xgb.XGBClassifier(random_state=42)),
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(max_iter=1000, random_state=42)),
    ('DNN', 'dnn')
]

feature_dicts =[
    {'features':[
        "Speed", "Tonnage", "BrakeCylinder", "IntrainForce", "Acc4_RMS",
        "Acc3_RMS", "Acc2", "LP1", "Acc3", "SND_R", "Acc2_RMS", "SND_L",
        "LP3", "BodyRockRr", "BodyRockFrt", "SND", "BounceRr", "BounceFrt",
        "LP4", "Curvature", "Twist14m", "Rail_Pro_R", "Acc4", "Acc1_RMS",
        "VACC_R", "VACC_L", "Acc1", "VACC", "Twist2m",
        "Rail_Pro_L", "LP2", "Track_Offset"
    ], 'method':'Mutual Information'},
    {'features':[
        'Tonnage', 'Speed', 'IntrainForce', 'SND',
        'Twist2m', 'Twist14m', 'Acc4_RMS', 'SND_L', 'LP1', 'Rail_Pro_L',
        'BrakeCylinder', 'BodyRockFrt', 'Acc1_RMS', 'SND_R',
        'LP4', 'LP3', 'BodyRockRr', 'VACC', 'Acc1',
        'LP2', 'Track_Offset', 'VACC_R', 'Acc4', 'VACC_L',
        'BounceFrt', 'Acc3', 'BounceRr', 'Acc2_RMS', 'Acc2', 'Rail_Pro_R',
        'Curvature', 'Acc3_RMS'
    ], 'method':'Variance Threshold'},
    {'features':[
        'Tonnage', 'SND', 'IntrainForce', 'BrakeCylinder', 'Curvature', 'Acc1',
        'Rail_Pro_L',
        'Acc1_RMS', 'Acc3', 'Speed', 'Acc4', 'Rail_Pro_R', 'BounceFrt', 'VACC',
        'BounceRr',
        'VACC_L', 'SND_L', 'VACC_R', 'LP3', 'Acc2_RMS', 'LP1', 'Acc4_RMS', 'BodyRockFrt',
        'Twist2m', 'Twist14m', 'LP2', 'Acc2', 'BodyRockRr', 'Acc3_RMS', 'SND_R', 'LP4',
        'Track_Offset'
    ], 'method':'Biserial Correlation'}
]

all_increasing_features_dict ={}
all_decreasing_features_dict ={}

for feature_dict in feature_dicts:
    increasing_features_dict, decreasing_features_dict = plot_graphs(models,
feature_dict)

```

```

    all_increasing_features_dict.update(increasing_features_dict)
    all_decreasing_features_dict.update(decreasing_features_dict)

results_for_each_model =
compute_interception_union_for_model(all_increasing_features_dict,
all_decreasing_features_dict, models)
print("-----Results for Each Model-----")
for model_name, result in results_for_each_model.items():
    print(f'Model: {model_name}')
    print(f'Increasing Intersections (Most Important Features Set):
{result["increasing_intersection"]}')
    print(f'Increasing Unions (Secondly Important Features Set):
{result["increasing_union"]}')
    print(f'Decreasing Unions (Only to be used if above is not sufficient):
{result["decreasing_union"]}')
    print(f'Decreasing Intersections (Least Important Features - Avoid these
features): {result["decreasing_intersection"]}')
    print('\n')

compute_interception_union_overall(all_increasing_features_dict,
all_decreasing_features_dict)

```