

Practical Machine Learning and Data Exploration

Project 2: Machine Learning and Feature Selection for GitHub Repository CI Prediction

Student Name: Ibrahim Mohammed Alsidrani

Student ID: 262000987

Dataset: GitHub Repository Metrics Dataset (5K+ Repos)

Kaggle Handle: lorenzoscaturchio/github-repo-metrics

Target Variable: has_ci

Contents

Project 2: Machine Learning and Feature Selection for GitHub Repository CI Prediction

Project Objectives

1. Importing Required Libraries
2. Downloading the Dataset from Kaggle
3. Loading the Dataset
4. Dataset Understanding
5. Initial Data Cleaning
6. Missing Values
7. Target Variable Analysis
8. Train-Test Split and Train-Only Missing Value Treatment
9. Preparing Features and Scaling
10. Model Evaluation
11. Machine Learning Models with GridSearchCV
12. Best Model Before Feature Selection
13. Feature Selection Method 1 - SelectKBest
14. Feature Selection Method 2 - RFE
15. Comparison of Selected Features
16. Machine Learning Using SelectKBest Features
17. Machine Learning Using RFE Features
18. Final Performance Comparison
19. Confusion Matrix for the Best Final Configuration
20. Final Verification
21. Conclusion

Practical Machine Learning and Data Exploration

Project 2: Machine Learning and Feature Selection for GitHub Repository CI Prediction

Student Name: Ibrahim Mohammed Alsidrani

Student ID: 262000987

Project Objectives

1. Download the GitHub Repository Metrics dataset from Kaggle.
2. Handle missing values appropriately to prepare the dataset for machine learning.
3. Apply machine learning classification models with GridSearchCV optimization and evaluate their performance.
4. Apply two feature selection methods to identify the most important features.
5. Apply machine learning models to the selected features and evaluate their performance.
6. Compare model performance before and after feature selection.

Dataset: GitHub Repository Metrics Dataset (5K+ Repos)

Kaggle Dataset Handle: `lorenzoscaturchio/github-repo-metrics`

Target Variable: `has_ci`

1. Importing Required Libraries

```
In [1]:

import os
import sys
import subprocess
from pathlib import Path
import warnings

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.base import clone
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report,
    ConfusionMatrixDisplay
)

warnings.filterwarnings("ignore")
pd.set_option("display.max_columns", None)

RANDOM_STATE = 42

print("Libraries imported successfully.")

Libraries imported successfully.
```

2. Downloading the Dataset from Kaggle

```
In [2]:

KAGGLE_DATASET = "lorenzoscaturchio/github-repo-metrics"
local_csv = Path("github_repos.csv")

if local_csv.exists():
    data_file = local_csv
    print("Using local dataset:", data_file)
else:
    try:
        import kagglehub
    except ImportError:
        subprocess.check_call([
            sys.executable, "-m", "pip", "install", "-q", "kagglehub"
        ])
        import kagglehub

    dataset_path = Path(
        kagglehub.dataset_download(KAGGLE_DATASET)
    )

    csv_files = list(dataset_path.rglob("*.csv"))
    if not csv_files:
        raise FileNotFoundError("No CSV file was found.")

    matching_files = [
        file for file in csv_files
        if file.name.lower() == "github_repos.csv"
    ]

    data_file = matching_files[0] if matching_files else csv_files[0]

    print("Dataset downloaded from Kaggle.")
    print("Dataset directory:", dataset_path)
    print("CSV file:", data_file)

Downloading to /root/.cache/kagglehub/datasets/lorenzoscaturchio/github-repo-metrics/12.archive...
100%|██████████| 499k/499k [00:00<00:00, 33.6MB/s]

Extracting files...
Dataset downloaded from Kaggle.
Dataset directory: /root/.cache/kagglehub/datasets/lorenzoscaturchio/github-repo-metrics/versions/12
CSV file: /root/.cache/kagglehub/datasets/lorenzoscaturchio/github-repo-metrics/versions/12/github_repos.csv
```

3. Loading the Dataset

```
In [3]:

df = pd.read_csv(data_file)

print("Dataset loaded successfully.")
print("Dataset shape:", df.shape)

Dataset loaded successfully.
Dataset shape: (5500, 29)
```

4. Dataset Understanding

```
In [4]:

df.head()
```

columns 1-9 of 29

	repo_name	language	description	stars	forks	watchers	open_issues	closed_issu es	open_pull_r equests
0	serve-ml	Ruby	server framework for building authenticatio	13	3	0	12	99	3

			n						
1	easy-link	TypeScript	Scalable pipeline for database management	32	5	2	2	19	0
2	go-graph-agent	Kotlin	lightweight dashboard built with Kotlin	2	0	0	0	10	0
3	turbo-core-forge	C++	engine framework for building database management	109	17	6	0	1	0
4	deep-graph-dash	Python	dashboard framework for building database mana...	21	10	0	13	31	4

columns 10-18 of 29

	merged_pull_requests	contributors	commits	releases	license	topics	created_date	last_commit_date	readme_length
0	36	5	200	2	GPL-3.0	api automation devops rails deep-learning	2024-09-14	2024-12-19	1282
1	10	6	1167	24	BSD-3	angular react web api	2022-12-18	2024-12-26	1902
2	4	2	10	0	NaN	mobile desktop	2024-12-13	2024-12-15	1251
3	0	4	195	10	MIT	systems deep-learning tool graphql serverless	2023-12-09	2024-12-22	155
4	15	9	2187	14	MIT	automation data-science api rag	2018-01-04	2024-11-20	3860

columns 19-27 of 29

	has_ci	test_coverage	has_code_of_conduct	has_contributing_guide	has_wiki	has_pages	has_discussions	default_branch	is_archived
0	0	NaN	0	0	0	0	1	main	0
1	0	NaN	0	0	0	0	0	master	0
2	0	NaN	0	0	1	0	0	develop	0
3	1	73.5	0	0	0	0	0	main	0
4	0	NaN	0	1	0	0	0	develop	0

columns 28-29 of 29

	is_fork	size_kb
0	1	497
1	1	4622
2	0	2148
3	0	812
4	1	5932

In [5]:

```
df.head().T
```

	0	1	2	3	4
repo_name	serve-ml	easy-link	go-graph-agent	turbo-core-forge	deep-graph-dash
language	Ruby	TypeScript	Kotlin	C++	Python
description	server framework for building authentication	Scalable pipeline for database management	lightweight dashboard built with Kotlin	engine framework for building database management	dashboard framework for building database mana...
stars	13	32	2	109	21
forks	3	5	0	17	10
watchers	0	2	0	6	0
open_issues	12	2	0	0	13
closed_issues	99	19	10	1	31
open_pull_requests	3	0	0	0	4
merged_pull_requests	36	10	4	0	15
contributors	5	6	2	4	9
commits	200	1167	10	195	2187
releases	2	24	0	10	14
license	GPL-3.0	BSD-3	NaN	MIT	MIT
topics	api automation devops rails deep-learning	angular react web api	mobile desktop	systems deep-learning tool graphql serverless	automation data-science api rag
created_date	2024-09-14	2022-12-18	2024-12-13	2023-12-09	2018-01-04
last_commit_date	2024-12-19	2024-12-26	2024-12-15	2024-12-22	2024-11-20
readme_length	1282	1902	1251	155	3860
has_ci	0	0	0	1	0
test_coverage	NaN	NaN	NaN	73.5	NaN
has_code_of_conduct	0	0	0	0	0
has_contributing_guide	0	0	0	0	1
has_wiki	0	0	1	0	0
has_pages	0	0	0	0	0
has_discussions	1	0	0	0	0
default_branch	main	master	develop	main	develop
is_archived	0	0	0	0	0
is_fork	1	1	0	0	1
size_kb	497	4622	2148	812	5932

In [6]:

```
print("Number of rows:", df.shape[0])
print("Number of columns:", df.shape[1])

print("\nColumn names:")
print(df.columns.tolist())

print("\nNumber of duplicated rows:", df.duplicated().sum())
```

Number of rows: 5500
Number of columns: 29

Column names:
['repo_name', 'language', 'description', 'stars', 'forks', 'watchers', 'open_issues', 'closed_issues', 'open_pull_requests', 'merged_pull_requests', 'contributors', 'commits', 'releases', 'license', 'topics', 'created_date', 'last_commit_date', 'readme_length', 'has_ci', 'test_coverage', 'has_code_of_conduct', 'has_contributing_guide', 'has_wiki', 'has_pages', 'has_discussions', 'default_branch', 'is_archived', 'is_fork', 'size_kb']

Number of duplicated rows: 0

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5500 entries, 0 to 5499
Data columns (total 29 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   repo_name             5500 non-null   object
 1   language              5500 non-null   object
 2   description            5500 non-null   object
 3   stars                 5500 non-null   int64
 4   forks                 5500 non-null   int64
 5   watchers              5500 non-null   int64
 6   open_issues           5500 non-null   int64
 7   closed_issues         5500 non-null   int64
 8   open_pull_requests    5500 non-null   int64
 9   merged_pull_requests  5500 non-null   int64
10   contributors          5500 non-null   int64
11   commits               5500 non-null   int64
12   releases              5500 non-null   int64
13   license               4760 non-null   object
14   topics                5500 non-null   object
15   created_date          5500 non-null   object
16   last_commit_date      5500 non-null   object
17   readme_length         5500 non-null   int64
18   has_ci                5500 non-null   int64
19   test_coverage         1702 non-null   float64
20   has_code_of_conduct   5500 non-null   int64
21   has_contributing_guide 5500 non-null   int64
22   has_wiki              5500 non-null   int64
23   has_pages             5500 non-null   int64
24   has_discussions       5500 non-null   int64
25   default_branch        5500 non-null   object
26   is_archived           5500 non-null   int64
27   is_fork               5500 non-null   int64
28   size_kb               5500 non-null   int64
dtypes: float64(1), int64(20), object(8)
memory usage: 1.2+ MB
```

5. Initial Data Cleaning

In [8]:

```
df.columns = df.columns.str.strip()

object_columns = df.select_dtypes(include="object").columns

for column in object_columns:
    df[column] = df[column].apply(
        lambda value: value.strip()
        if isinstance(value, str)
        else value
    )

df.replace(
    ["", "?", "nan", "NaN", "None", "null", "NULL"],
    np.nan,
    inplace=True
)

print("Initial data cleaning completed.")
```

Initial data cleaning completed.

6. Missing Values

In [9]:

```
missing_summary = pd.DataFrame({
    "Missing Values": df.isnull().sum(),
    "Missing Percentage (%)": df.isnull().mean() * 100
})

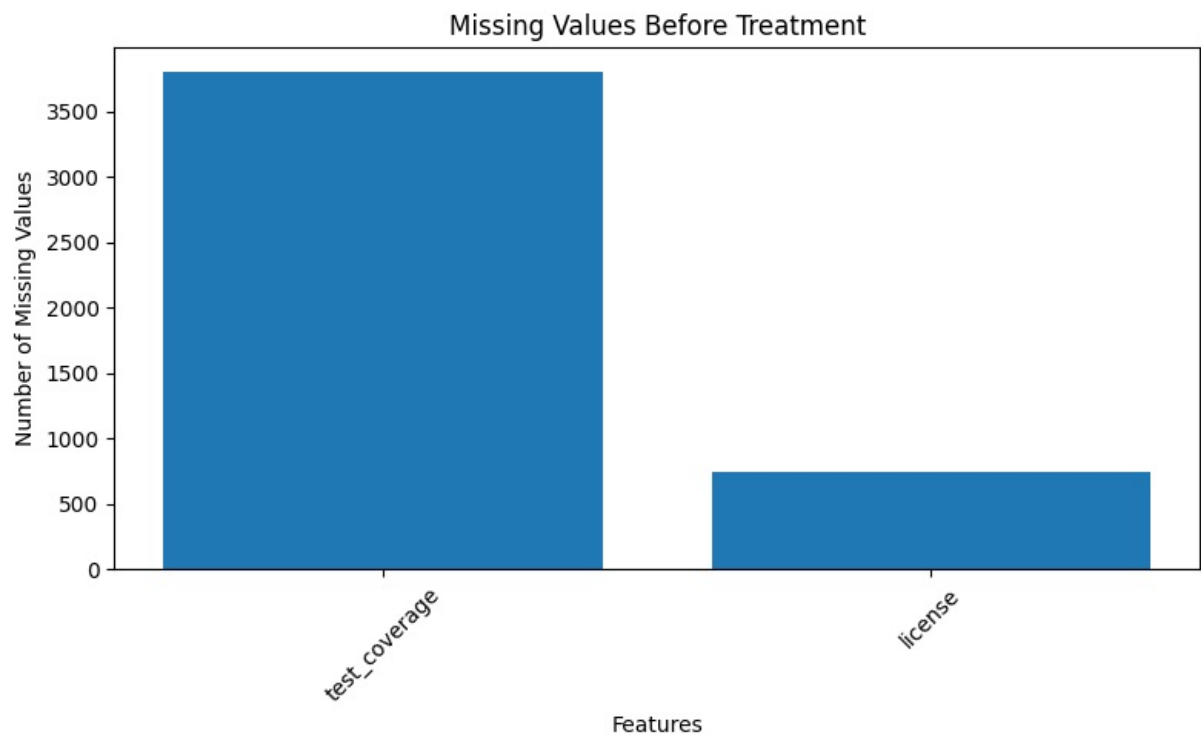
missing_summary = missing_summary[
    missing_summary["Missing Values"] > 0
].sort_values(
    by="Missing Values",
    ascending=False
)

missing_summary.round(2)
```

	Missing Values	Missing Percentage (%)
test_coverage	3798	69.05
license	740	13.45

In [10]:

```
if not missing_summary.empty:
    plt.figure(figsize=(8, 5))
    plt.bar(
        missing_summary.index,
        missing_summary["Missing Values"]
    )
    plt.title("Missing Values Before Treatment")
    plt.xlabel("Features")
    plt.ylabel("Number of Missing Values")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```



In [11]:

```
print("Missing values identified before the train-test split.")
```

Missing values identified before the train-test split.

In [12]:

```
missing_summary.round(2)
```

	Missing Values	Missing Percentage (%)
test_coverage	3798	69.05
license	740	13.45

7. Target Variable Analysis

Target: `has_ci` (`0` = No CI, `1` = CI)

In [13]:

```
TARGET = "has_ci"

if TARGET not in df.columns:
    raise KeyError(f"Target column '{TARGET}' was not found.")

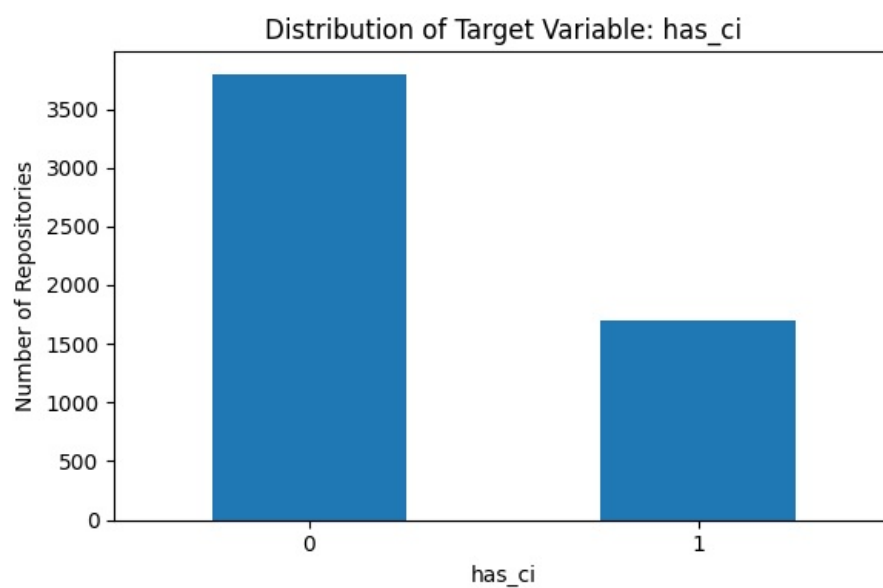
print(df[TARGET].value_counts().sort_index())
print()
print(
    df[TARGET]
    .value_counts(normalize=True)
    .sort_index()
    .mul(100)
    .round(2)
)
```

```
has_ci
0    3798
1    1702
Name: count, dtype: int64

has_ci
0    69.05
1    30.95
Name: proportion, dtype: float64
```

In [14]:

```
plt.figure(figsize=(6, 4))
df[TARGET].value_counts().sort_index().plot(kind="bar")
plt.title("Distribution of Target Variable: has_ci")
plt.xlabel("has_ci")
plt.ylabel("Number of Repositories")
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



8. Train-Test Split and Train-Only Missing Value Treatment

In [15]:

```
X_raw = df.drop(columns=[TARGET]).copy()
y = df[TARGET].astype(int).copy()

X_train_raw, X_test_raw, y_train, y_test = train_test_split(
    X_raw,
    y,
    test_size=0.20,
    random_state=RANDOM_STATE,
    stratify=y
)

numerical_columns = X_train_raw.select_dtypes(
    include=["number"]
).columns.tolist()

categorical_columns = X_train_raw.select_dtypes(
    include=["object", "category", "bool"]
).columns.tolist()

train_medians = X_train_raw[numerical_columns].median()

X_train_raw[numerical_columns] = (
    X_train_raw[numerical_columns]
    .fillna(train_medians)
)

X_test_raw[numerical_columns] = (
    X_test_raw[numerical_columns]
    .fillna(train_medians)
)

for column in categorical_columns:
    mode_values = X_train_raw[column].mode(dropna=True)
    if not mode_values.empty:
        train_mode = mode_values.iloc[0]
        X_train_raw[column] = X_train_raw[column].fillna(train_mode)
        X_test_raw[column] = X_test_raw[column].fillna(train_mode)

print("Training set shape:", X_train_raw.shape)
print("Testing set shape:", X_test_raw.shape)
print("Training missing values:", X_train_raw.isnull().sum().sum())
print("Testing missing values:", X_test_raw.isnull().sum().sum())
```

```
Training set shape: (4400, 28)
Testing set shape: (1100, 28)
Training missing values: 0
Testing missing values: 0
```

9. Preparing Features and Scaling

In [16]:

```
excluded_features = ["test_coverage"]

feature_columns = [
    column
    for column in X_train_raw.select_dtypes(include=["number"]).columns
    if column not in excluded_features
]

X_train = X_train_raw[feature_columns].copy()
X_test = X_test_raw[feature_columns].copy()

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Excluded from predictors:", excluded_features)
print("Number of input features:", len(feature_columns))
print("Training feature matrix:", X_train.shape)
print("Testing feature matrix:", X_test.shape)
print()
print("Training target distribution:")
print(y_train.value_counts(normalize=True).round(3))
print()
print("Testing target distribution:")
print(y_test.value_counts(normalize=True).round(3))
```

```
Excluded from predictors: ['test_coverage']
Number of input features: 19
Training feature matrix: (4400, 19)
Testing feature matrix: (1100, 19)
```

```
Training target distribution:
has_ci
0    0.69
1    0.31
Name: proportion, dtype: float64
```

```
Testing target distribution:
has_ci
0    0.691
1    0.309
Name: proportion, dtype: float64
```

10. Model Evaluation

In [17]:

```
def evaluate_model(model_name, y_true, y_pred):
    return {
        "Model": model_name,
        "Accuracy": accuracy_score(
            y_true, y_pred
        ),
        "Precision": precision_score(
            y_true, y_pred, zero_division=0
        ),
        "Recall": recall_score(
            y_true, y_pred, zero_division=0
        ),
        "F1 Score": f1_score(
            y_true, y_pred, zero_division=0
        )
    }

print("Evaluation function created successfully.")
```

Evaluation function created successfully.

11. Machine Learning Models with GridSearchCV

In [18]:

```
models_and_parameters = {
    "Logistic Regression": {
        "model": LogisticRegression(
            max_iter=2000,
            random_state=RANDOM_STATE
        ),
        "params": {
            "C": [0.1, 1, 10],
            "solver": ["liblinear"]
        }
    },

    "KNN": {
        "model": KNeighborsClassifier(),
        "params": {
            "n_neighbors": [5, 9, 15, 25],
            "weights": ["uniform", "distance"],
            "p": [2]
        }
    },

    "Random Forest": {
        "model": RandomForestClassifier(
            random_state=RANDOM_STATE,
            class_weight="balanced",
            n_jobs=-1
        ),
        "params": {
            "n_estimators": [100, 200],
            "max_depth": [None, 10, 20],
            "min_samples_split": [2, 5],
            "min_samples_leaf": [1]
        }
    }
}

print("Models and parameter grids prepared.")
```

Models and parameter grids prepared.

In [19]:

```
grid_results = []
best_estimators = {}

for model_name, configuration in models_and_parameters.items():

    print("=" * 70)
    print("Running GridSearchCV for:", model_name)

    grid_search = GridSearchCV(
        estimator=configuration["model"],
        param_grid=configuration["params"],
        scoring="f1",
        cv=5,
        n_jobs=-1,
        refit=True
    )

    grid_search.fit(
        X_train_scaled,
        y_train
    )

    best_model = grid_search.best_estimator_
    best_estimators[model_name] = best_model

    y_pred = best_model.predict(
        X_test_scaled
    )

    result = evaluate_model(
        model_name,
        y_test,
        y_pred
    )

    result["Best CV F1"] = grid_search.best_score_
    result["Best Parameters"] = str(
        grid_search.best_params_
    )

    grid_results.append(result)

    print("Best Parameters:")
    print(grid_search.best_params_)

    print(
        "Best Cross-Validation F1:",
        round(grid_search.best_score_, 4)
    )

    print("\nClassification Report:")
    print(
        classification_report(
            y_test,
            y_pred,
            zero_division=0
        )
    )
```

```

=====
Running GridSearchCV for: Logistic Regression
Best Parameters:
{'C': 0.1, 'solver': 'liblinear'}
Best Cross-Validation F1: 0.0202

Classification Report:
      precision    recall  f1-score   support

      0       0.69       1.00       0.82        760
      1       0.57       0.01       0.02        340

 accuracy         0.63
 macro avg       0.63       0.50       0.42        1100
weighted avg       0.66       0.69       0.57        1100

=====
Running GridSearchCV for: KNN
Best Parameters:
{'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
Best Cross-Validation F1: 0.224

Classification Report:
      precision    recall  f1-score   support

      0       0.70       0.85       0.77        760
      1       0.35       0.18       0.24        340

 accuracy         0.53
 macro avg       0.53       0.52       0.50        1100
weighted avg       0.59       0.64       0.61        1100

=====
Running GridSearchCV for: Random Forest
Best Parameters:
{'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
Best Cross-Validation F1: 0.2264

Classification Report:
      precision    recall  f1-score   support

      0       0.69       0.84       0.76        760
      1       0.32       0.17       0.22        340

 accuracy         0.51
 macro avg       0.51       0.50       0.49        1100
weighted avg       0.58       0.63       0.59        1100

```

In [20]:

```

grid_results_df = pd.DataFrame(
    grid_results
).sort_values(
    by="F1 Score",
    ascending=False
).reset_index(drop=True)

numeric_result_columns = [
    "Accuracy",
    "Precision",
    "Recall",
    "F1 Score",
    "Best CV F1"
]

grid_results_df[
    numeric_result_columns
] = grid_results_df[
    numeric_result_columns
].round(4)

grid_results_df

```

	Model	Accuracy	Precision	Recall	F1 Score	Best CV F1	Best Parameters
0	KNN	0.6445	0.3543	0.1824	0.2408	0.2240	{'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
1	Random Forest	0.6318	0.3204	0.1706	0.2226	0.2264	{'max_depth': 10, 'min_samples_leaf': 1, 'min_...
2	Logistic Regression	0.6918	0.5714	0.0118	0.0231	0.0202	{'C': 0.1, 'solver': 'liblinear'}

12. Best Model Before Feature Selection

In [21]:

```
best_all_features_row = (
    grid_results_df
    .sort_values(
        by="F1 Score",
        ascending=False
    )
    .iloc[0]
)

best_all_features_name = (
    best_all_features_row["Model"]
)

print(
    "Best model using all features:",
    best_all_features_name
)

print(
    "Test F1 Score:",
    best_all_features_row["F1 Score"]
)

print(
    "Best Parameters:",
    best_all_features_row["Best Parameters"]
)
```

```
Best model using all features: KNN
Test F1 Score: 0.2408
Best Parameters: {'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
```

13. Feature Selection Method 1 - SelectKBest

In [22]:

```
NUMBER_OF_FEATURES = min(
    8,
    X_train.shape[1]
)

select_k_best = SelectKBest(
    score_func=f_classif,
    k=NUMBER_OF_FEATURES
)

select_k_best.fit(
    X_train_scaled,
    y_train
)

kbest_mask = select_k_best.get_support()

kbest_features = X_train.columns[
    kbest_mask
].tolist()

kbest_scores = pd.DataFrame({
    "Feature": X_train.columns,
    "F Score": select_k_best.scores_
}).sort_values(
    by="F Score",
    ascending=False
).reset_index(drop=True)

print(
    f"Top {NUMBER_OF_FEATURES} features selected by SelectKBest:"
)
print(kbest_features)

kbest_scores.head(NUMBER_OF_FEATURES)
```

Top 8 features selected by SelectKBest:

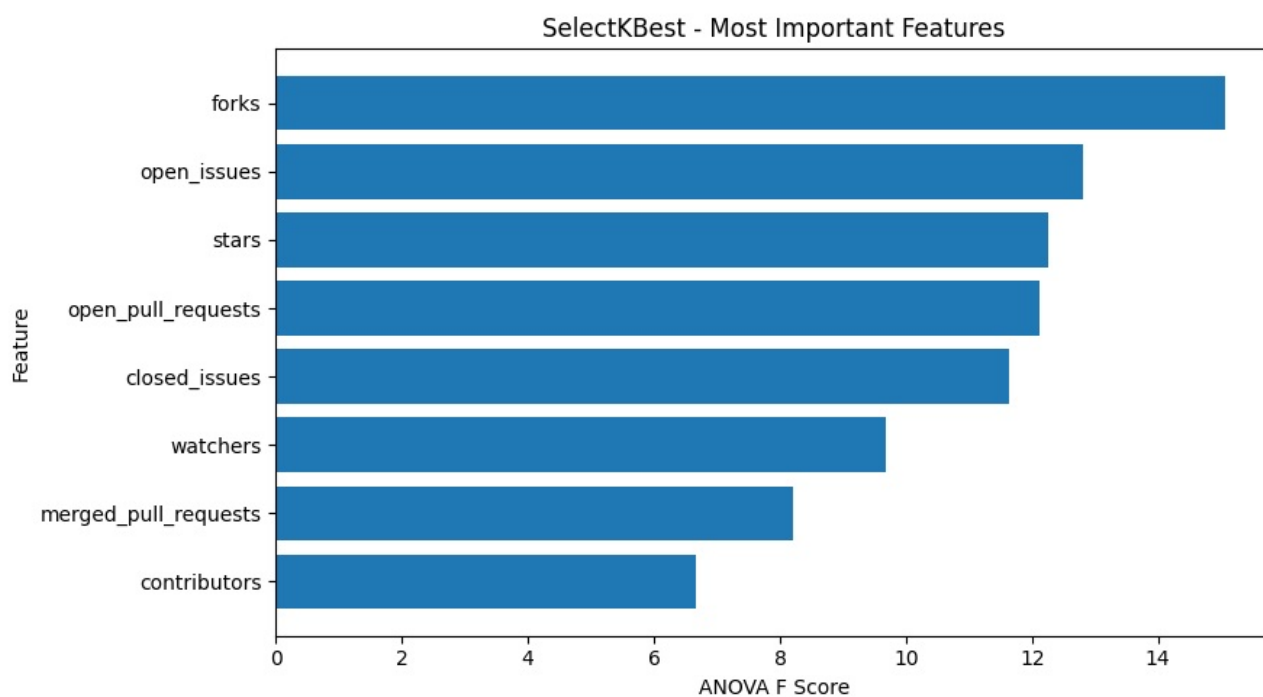
['stars', 'forks', 'watchers', 'open_issues', 'closed_issues', 'open_pull_requests', 'merged_pull_requests', 'contributors']

	Feature	F Score
0	forks	15.071092
1	open_issues	12.802770
2	stars	12.258361
3	open_pull_requests	12.118608
4	closed_issues	11.644840
5	watchers	9.688617
6	merged_pull_requests	8.219187
7	contributors	6.669861

In [23]:

```
top_kbest = kbest_scores.head(
    NUMBER_OF_FEATURES
).sort_values(
    by="F Score",
    ascending=True
)

plt.figure(figsize=(9, 5))
plt.barh(
    top_kbest["Feature"],
    top_kbest["F Score"]
)
plt.title("SelectKBest - Most Important Features")
plt.xlabel("ANOVA F Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



14. Feature Selection Method 2 - RFE

In [24]:

```
rfe_estimator = LogisticRegression(
    max_iter=2000,
    solver="liblinear",
    random_state=RANDOM_STATE
)

rfe = RFE(
    estimator=rfe_estimator,
    n_features_to_select=NUMBER_OF_FEATURES,
    step=1
)

rfe.fit(
    X_train_scaled,
    y_train
)

rfe_mask = rfe.get_support()

rfe_features = X_train.columns[
    rfe_mask
].tolist()

rfe_ranking = pd.DataFrame({
    "Feature": X_train.columns,
    "RFE Ranking": rfe.ranking_,
    "Selected": rfe.support_
}).sort_values(
    by=["RFE Ranking", "Feature"]
).reset_index(drop=True)

print(
    f"Top {NUMBER_OF_FEATURES} features selected by RFE:"
)
print(rfe_features)

rfe_ranking
```

Top 8 features selected by RFE:

['forks', 'watchers', 'closed_issues', 'open_pull_requests', 'merged_pull_requests', 'commits', 'readme_length', 'is_fork']

	Feature	RFE Ranking	Selected
0	closed_issues	1	True
1	commits	1	True
2	forks	1	True
3	is_fork	1	True
4	merged_pull_requests	1	True
5	open_pull_requests	1	True
6	readme_length	1	True
7	watchers	1	True
8	size_kb	2	False
9	has_wiki	3	False
10	stars	4	False
11	has_code_of_conduct	5	False
12	contributors	6	False
13	has_discussions	7	False
14	is_archived	8	False
15	has_contributing_guide	9	False
16	open_issues	10	False
17	has_pages	11	False
18	releases	12	False

15. Comparison of Selected Features

In [25]:

```
feature_selection_comparison = pd.DataFrame({
    "SelectKBest Features": pd.Series(kbest_features),
    "RFE Features": pd.Series(rfe_features)
})
```

feature_selection_comparison

	SelectKBest Features	RFE Features
0	stars	forks
1	forks	watchers
2	watchers	closed_issues
3	open_issues	open_pull_requests
4	closed_issues	merged_pull_requests
5	open_pull_requests	commits
6	merged_pull_requests	readme_length
7	contributors	is_fork

In [26]:

```
common_selected_features = sorted(
    set(kbest_features).intersection(
        set(rfe_features)
    )
)

print("Features selected by both methods:")
print(common_selected_features)
```

```
Features selected by both methods:
['closed_issues', 'forks', 'merged_pull_requests', 'open_pull_requests', 'watchers']
```

16. Machine Learning Using SelectKBest Features

In [27]:

```
X_train_kbest = select_k_best.transform(
    X_train_scaled
)
X_test_kbest = select_k_best.transform(
    X_test_scaled
)

kbest_model_results = []

for model_name, best_model in best_estimators.items():

    model = clone(best_model)

    model.fit(
        X_train_kbest,
        y_train
    )

    y_pred = model.predict(
        X_test_kbest
    )

    result = evaluate_model(
        model_name,
        y_test,
        y_pred
    )

    result["Feature Selection"] = "SelectKBest"
    result["Number of Features"] = NUMBER_OF_FEATURES

    kbest_model_results.append(result)

kbest_results_df = pd.DataFrame(
    kbest_model_results
).sort_values(
    by="F1 Score",
    ascending=False
).reset_index(drop=True)

kbest_results_df[
    ["Accuracy", "Precision", "Recall", "F1 Score"]
] = kbest_results_df[
    ["Accuracy", "Precision", "Recall", "F1 Score"]
].round(4)

kbest_results_df
```

	Model	Accuracy	Precision	Recall	F1 Score	Feature Selection	Number of Features
0	Random Forest	0.6027	0.3052	0.2235	0.2581	SelectKBest	8
1	KNN	0.6191	0.3128	0.1941	0.2396	SelectKBest	8
2	Logistic Regression	0.6918	0.5714	0.0118	0.0231	SelectKBest	8

17. Machine Learning Using RFE Features

In [28]:

```
X_train_rfe = rfe.transform(
    X_train_scaled
)
X_test_rfe = rfe.transform(
    X_test_scaled
)

rfe_model_results = []

for model_name, best_model in best_estimators.items():

    model = clone(best_model)

    model.fit(
        X_train_rfe,
        y_train
    )

    y_pred = model.predict(
        X_test_rfe
    )

    result = evaluate_model(
        model_name,
        y_test,
        y_pred
    )

    result["Feature Selection"] = "RFE"
    result["Number of Features"] = NUMBER_OF_FEATURES

    rfe_model_results.append(result)

rfe_results_df = pd.DataFrame(
    rfe_model_results
).sort_values(
    by="F1 Score",
    ascending=False
).reset_index(drop=True)

rfe_results_df[
    ["Accuracy", "Precision", "Recall", "F1 Score"]
] = rfe_results_df[
    ["Accuracy", "Precision", "Recall", "F1 Score"]
].round(4)

rfe_results_df
```

	Model	Accuracy	Precision	Recall	F1 Score	Feature Selection	Number of Features
0	Random Forest	0.6036	0.3095	0.2294	0.2635	RFE	8
1	KNN	0.6245	0.3166	0.1853	0.2338	RFE	8
2	Logistic Regression	0.6918	0.5714	0.0118	0.0231	RFE	8

18. Final Performance Comparison

In [29]:

```
all_features_results = grid_results_df[
    ["Model", "Accuracy", "Precision", "Recall", "F1 Score"]
].copy()

all_features_results["Feature Selection"] = "All Features"
all_features_results["Number of Features"] = X_train.shape[1]

comparison_df = pd.concat([
    all_features_results,
    kbest_results_df,
    rfe_results_df
], ignore_index=True)

comparison_df = comparison_df[
    [
        "Feature Selection",
        "Model",
        "Number of Features",
        "Accuracy",
        "Precision",
        "Recall",
        "F1 Score"
    ]
].sort_values(
    by="F1 Score",
    ascending=False
).reset_index(drop=True)

comparison_df
```

	Feature Selection	Model	Number of Features	Accuracy	Precision	Recall	F1 Score
0	RFE	Random Forest	8	0.6036	0.3095	0.2294	0.2635
1	SelectKBest	Random Forest	8	0.6027	0.3052	0.2235	0.2581
2	All Features	KNN	19	0.6445	0.3543	0.1824	0.2408
3	SelectKBest	KNN	8	0.6191	0.3128	0.1941	0.2396
4	RFE	KNN	8	0.6245	0.3166	0.1853	0.2338
5	All Features	Random Forest	19	0.6318	0.3204	0.1706	0.2226
6	All Features	Logistic Regression	19	0.6918	0.5714	0.0118	0.0231
7	SelectKBest	Logistic Regression	8	0.6918	0.5714	0.0118	0.0231
8	RFE	Logistic Regression	8	0.6918	0.5714	0.0118	0.0231

In [30]:

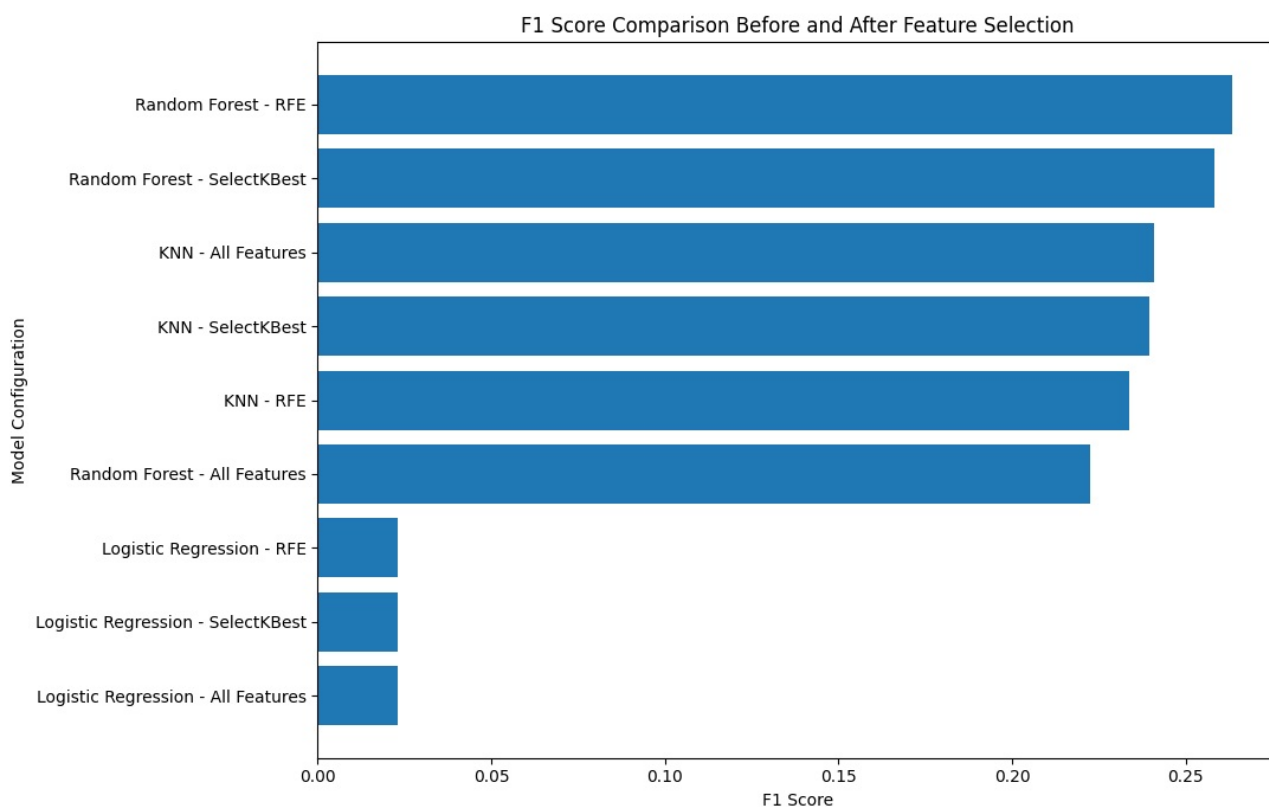
```
plot_df = comparison_df.copy()

plot_df["Configuration"] = (
    plot_df["Model"]
    + " - "
    + plot_df["Feature Selection"]
)

plot_df = plot_df.sort_values(
    by="F1 Score",
    ascending=True
)

plt.figure(figsize=(11, 7))
plt.barh(
    plot_df["Configuration"],
    plot_df["F1 Score"]
)

plt.title("F1 Score Comparison Before and After Feature Selection")
plt.xlabel("F1 Score")
plt.ylabel("Model Configuration")
plt.tight_layout()
plt.show()
```



19. Confusion Matrix for the Best Final Configuration

In [31]:

```
best_final_row = comparison_df.iloc[0]

best_final_model_name = best_final_row["Model"]
best_final_selection = best_final_row["Feature Selection"]

best_final_model = clone(
    best_estimators[best_final_model_name]
)

if best_final_selection == "All Features":
    best_X_train = X_train_scaled
    best_X_test = X_test_scaled

elif best_final_selection == "SelectKBest":
    best_X_train = X_train_kbest
    best_X_test = X_test_kbest

else:
    best_X_train = X_train_rfe
    best_X_test = X_test_rfe

best_final_model.fit(
    best_X_train,
    y_train
)

best_final_predictions = best_final_model.predict(
    best_X_test
)

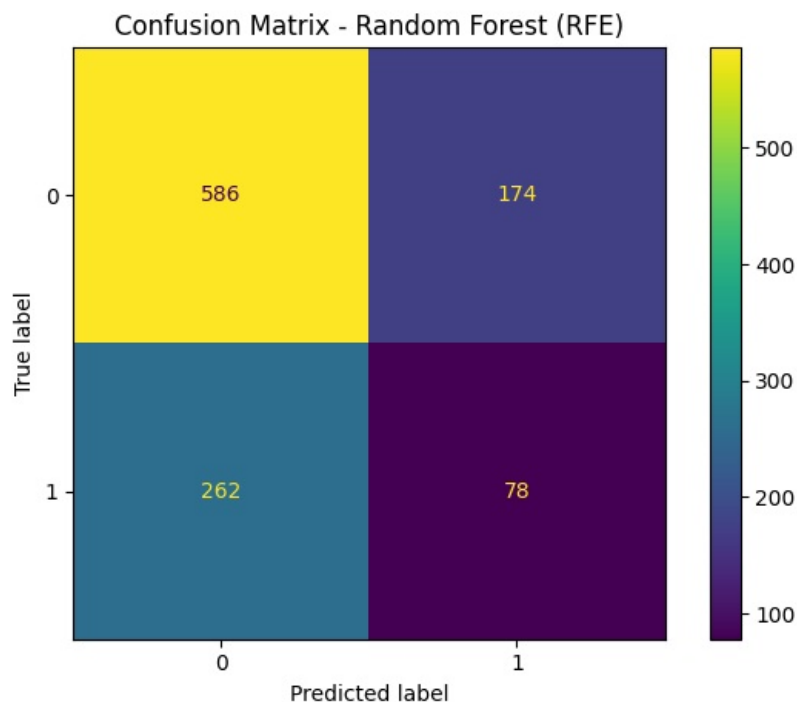
print("Best final model:", best_final_model_name)
print("Feature selection:", best_final_selection)
print(
    "Final F1 Score:",
    round(
        f1_score(
            y_test,
            best_final_predictions,
            zero_division=0
        ),
        4
    )
)

ConfusionMatrixDisplay.from_predictions(
    y_test,
    best_final_predictions
)

plt.title(
    f"Confusion Matrix - {best_final_model_name} ({best_final_selection})"
)

plt.tight_layout()
plt.show()
```

Best final model: Random Forest
Feature selection: RFE
Final F1 Score: 0.2635



20. Final Verification

In [32]:

```
print("FINAL PROJECT VERIFICATION")
print("=" * 60)

print("1. Dataset downloaded/loaded:", df.shape[0] > 0)
print("2. Training missing values:", X_train_raw.isnull().sum().sum())
print("3. Testing missing values:", X_test_raw.isnull().sum().sum())
print("4. GridSearchCV models evaluated:", len(grid_results_df))
print("5. Feature selection methods used: SelectKBest and RFE")
print("6. SelectKBest selected features:", len(kbest_features))
print("7. RFE selected features:", len(rfe_features))
print("8. Total evaluated configurations:", len(comparison_df))
print(
    "9. Best final configuration:",
    comparison_df.iloc[0][
        ["Feature Selection", "Model", "F1 Score"]
    ].to_dict()
)
```

```
FINAL PROJECT VERIFICATION
=====
1. Dataset downloaded/loaded: True
2. Training missing values: 0
3. Testing missing values: 0
4. GridSearchCV models evaluated: 3
5. Feature selection methods used: SelectKBest and RFE
6. SelectKBest selected features: 8
7. RFE selected features: 8
8. Total evaluated configurations: 9
9. Best final configuration: {'Feature Selection': 'RFE', 'Model': 'Random Forest', 'F1 Score': 0.2635}
```

21. Conclusion

The project completed the required workflow: Kaggle dataset loading, missing-value treatment using training-data statistics, GridSearchCV model optimization, two feature-selection methods, retraining with selected features, and final performance comparison.