Experiment Notebook

---

0. Setup Environment

0.a Install Environment and Mandatory Packages

```
# Do not modify this code
!pip install -q utstd

from utstd.folders import *
from utstd.ipyrenders import *

at = AtFolder(
    course_code=36106,
    assignment="AT2",
)
at.run()
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 9.9 MB/s eta 0:00:00
     Mounted at /content/gdrive

     You can now save your data files in: /content/gdrive/MyDrive/36106/assignment/AT2/data
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

0.b Disable Warnings Messages

```
# Do not modify this code
import warnings
warnings.simplefilter(action='ignore')
```

0.c Install Additional Packages

> If you are using additional packages, you need to install them here using the command: `! pip install <package_name>`

```
# <Student to fill this section>
```

0.d Import Packages

```
# <Student to fill this section>
import pandas as pd
import altair as alt
```

A. Project Description

```
# <Student to fill this section>
student_name = "Md Saifur Rahman"
student_id = "25528668"
```

```
# Do not modify this code
print_tile(size="h1", key='student_name', value=student_name)
```

student_name

Md Saifur Rahman

```
# Do not modify this code
print_tile(size="h1", key='student_id', value=student_id)
```

student_id

25528668

```
business_objective = """
The goal of this project is to develop a reliable predictive model that classifies university students into four academic performance categc

The business value lies in enabling educational institutions to proactively identify students at risk of underperforming and provide timely

Conversely, incorrect classifications—especially mislabeling high-performing students as poor performers or vice versa—could lead to misallc
"""


# Do not modify this code
print_tile(size="h3", key='business_objective', value=business_objective)
```

business_objective

The goal of this project is to develop a reliable predictive model that classifies university students into four academic performance categories —Excellent, Good, Average, and Poor—based on various academic, personal, and behavioral attributes. The business value lies in enabling educational institutions to proactively identify students at risk of underperforming and provide timely academic support or counseling. Accurate predictions will help in resource allocation, targeted interventions, and improved student retention rates. Conversely, incorrect classifications—especially mislabeling high-performing students as poor performers or vice versa—could lead to misallocated support efforts, student dissatisfaction, and reputational risk for the institution. Therefore, model accuracy and interpretability are critical to ensure

---

B. Experiment Description

```
# Do not modify this code
experiment_id = "1"
print_tile(size="h1", key='experiment_id', value=experiment_id)
```

experiment_id

1

```
experiment_hypothesis = """
We hypothesize that a student’s academic performance category (Excellent, Good, Average, Poor) can be predicted using a combination of their

This hypothesis is worth testing because if a strong relationship exists between these features and performance, we can design a practical m
"""



# Do not modify this code
print_tile(size="h3", key='experiment_hypothesis', value=experiment_hypothesis)
```

experiment_hypothesis

We hypothesize that a student’s academic performance category (Excellent, Good, Average, Poor) can be predicted using a combination of their historical academic metrics (GPA, completed credits), engagement indicators (study hours, attendance, study sessions), and behavioral/lifestyle factors (social media usage, skill development hours, etc.). This hypothesis is worth testing because if a strong relationship exists between these features and performance, we can design a practical model to identify students in need of academic support early in the semester. This can help universities reduce dropout rates, improve student success, and allocate support resources more

```
# <Student to fill this section>
experiment_expectations = """
Detail what will be the expected outcome of the experiment. If possible, estimate the goal you are expecting.
List the possible scenarios resulting from this experiment.
"""
```

```
# Do not modify this code
print_tile(size="h3", key='experiment_expectations', value=experiment_expectations)
```

experiment_expectations

Detail what will be the expected outcome of the experiment. If possible, estimate the goal you are expecting. List the possible scenarios

---

C. Data Understanding

```
# Do not modify this code
try:
  X_train = pd.read_csv('/content/drive/MyDrive/36106/AT02/X_train.csv')
  y_train = pd.read_csv('/content/drive/MyDrive/36106/AT02/y_train.csv')

  X_val = pd.read_csv('/content/drive/MyDrive/36106/AT02/X_val.csv')
  y_val = pd.read_csv('/content/drive/MyDrive/36106/AT02/y_val.csv')

  X_test = pd.read_csv('/content/drive/MyDrive/36106/AT02/X_test.csv')
  y_test = pd.read_csv('/content/drive/MyDrive/36106/AT02/y_test.csv')
except Exception as e:
  print(e)
```

D. Feature Selection

```
X_train.columns.tolist()
```

```
['gender',
 'scholarship',
 'university_transport',
 'learning_mode',
 'has_phone',
 'has_laptop',
 'english_proficiency',
 'on_probation',
 'is_suspended',
 'has_consulted_teacher',
 'relationship',
 'co_curricular',
 'living_arrangement',
 'health_issues',
 'disabilities',
 'current _semester',
 'study_hours',
 'study_sessions',
 'social_media_hours',
 'average_attendance',
 'skills',
 'skills_development_hours',
 'previous_gpa',
 'current_gpa',
 'completed_credits',
 'has_diploma',
 'house_income',
 'gpa_change',
 'study_x_attendance',
 'skill_to_social_ratio']
```

```
# prompt: now check the the column type

X_train.dtypes
```

	0
gender	int64
scholarship	int64
university_transport	int64
learning_mode	int64
has_phone	int64
has_laptop	int64
english_proficiency	int64
on_probation	int64
is_suspended	int64
has_consulted_teacher	int64
relationship	int64
co_curricular	int64
living_arrangement	int64
health_issues	int64
disabilities	int64
current _semester	float64
study_hours	float64
study_sessions	float64
social_media_hours	float64
average_attendance	float64
skills	int64
skills_development_hours	float64
previous_gpa	float64
current_gpa	float64
completed_credits	float64
has_diploma	bool
house_income	float64
gpa_change	float64
study_x_attendance	float64
skill_to_social_ratio	float64

dtype: object

```
X_test.dtypes
```

	0
gender	int64
scholarship	int64
university_transport	int64
learning_mode	int64
has_phone	int64
has_laptop	int64
english_proficiency	int64
on_probation	int64
is_suspended	int64
has_consulted_teacher	int64
relationship	int64
co_curricular	int64
living_arrangement	int64
health_issues	int64
disabilities	int64
current _semester	float64
study_hours	float64
study_sessions	float64
social_media_hours	float64
average_attendance	float64
skills	int64
skills_development_hours	float64
previous_gpa	float64
current_gpa	float64
completed_credits	float64
has_diploma	bool
house_income	float64
gpa_change	float64
study_x_attendance	float64
skill_to_social_ratio	float64

dtype: object

```
X_val.dtypes
```

	0
gender	int64
scholarship	int64
university_transport	int64
learning_mode	int64
has_phone	int64
has_laptop	int64
english_proficiency	int64
on_probation	int64
is_suspended	int64
has_consulted_teacher	int64
relationship	int64
co_curricular	int64
living_arrangement	int64
health_issues	int64
disabilities	int64
current _semester	float64
study_hours	float64
study_sessions	float64
social_media_hours	float64
average_attendance	float64
skills	int64
skills_development_hours	float64
previous_gpa	float64
current_gpa	float64
completed_credits	float64
has_diploma	bool
house_income	float64
gpa_change	float64
study_x_attendance	float64
skill_to_social_ratio	float64

dtype: object

```
# <Student to fill this section>

features_list = ['gender',
 'scholarship',
 'university_transport',
 'learning_mode',
 'has_phone',
 'has_laptop',
 'english_proficiency',
 'on_probation',
 'is_suspended',
 'has_consulted_teacher',
 'relationship',
 'co_curricular',
 'living_arrangement',
 'health_issues',
 'disabilities',
 'current _semester',
 'study_hours',
 'study_sessions',
 'social_media_hours',
 'average_attendance',
 'skills',
```

```
 'skills_development_hours',
 'previous_gpa',
 'current_gpa',
 'completed_credits',
 'has_diploma',
 'house_income',
 'gpa_change',
 'study_x_attendance',
 'skill_to_social_ratio']


feature_selection_explanations = """
The selected features were chosen based on a combination of exploratory data analysis, visual inspection, and multiple feature selection tec

For instance, `current_gpa`, `previous_gpa`, and `average_attendance` emerged as the top predictors due to their strong correlation with per

Categorical variables such as `english_proficiency`, `scholarship`, `on_probation`, `is_suspended`, `learning_mode`, and `has_consulted_teac

On the other hand, features like `program`, `birth_country`, `relationship`, `secondary_address`, and `has_phone` were excluded due to low v

The final set of features reflects a well-balanced mix of academic, behavioral, support-related, and engineered indicators, offering a stron
"""



# Do not modify this code
print_tile(size="h3", key='feature_selection_explanations', value=feature_selection_explanations)
```

feature_selection_explanations

The selected features were chosen based on a combination of exploratory data analysis, visual inspection, and multiple feature selection techniques such as Mutual Information, Random Forest Importance, and Lasso Regression. These features demonstrate strong or moderate predictive relationships with the target variable (student performance), as confirmed by statistical patterns and visual trends. For instance, `current_gpa`, `previous_gpa`, and `average_attendance` emerged as the top predictors due to their strong correlation with performance outcomes. Behavioral features like `study_hours`, `study_sessions`, `skills_development_hours`, and `social_media_hours` were retained for their interpretability and moderate associations with performance classes. Engineered features such as `gpa_change`, `study_x_attendance`, and `skill_to_social_ratio` were included to capture deeper interactions that were not directly visible in the raw data. Categorical variables such as `english_proficiency`, `scholarship`, `on_probation`, `is_suspended`, `learning_mode`, and `has_consulted_teacher` showed strong explanatory power in differentiating performance groups and were therefore included. Other demographic and support-related features like `gender`, `co_curricular`, and `living_arrangement` were retained for their mild but meaningful contribution. On the other hand, features like `program`, `birth_country`, `relationship`, `secondary_address`, and `has_phone` were excluded due to low variance, weak predictive power, or redundancy. These columns either lacked meaningful distribution across the target or provided minimal contribution during feature

---

E. Data Preparation

E.1 Data Transformation

```
from sklearn.preprocessing import StandardScaler

# 1. Rename inconsistent column across all datasets
for df in [X_train, X_val, X_test]:
    if 'current _semester' in df.columns:
        df.rename(columns={'current _semester': 'current_semester'}, inplace=True)

# 2. Convert boolean column to int if present
for df in [X_train, X_val, X_test]:
    if 'has_diploma' in df.columns and df['has_diploma'].dtype == 'bool':
        df['has_diploma'] = df['has_diploma'].astype(int)

# 3. Standardize float features
float_cols = [
    'current_semester', 'study_hours', 'study_sessions', 'social_media_hours',
    'average_attendance', 'skills_development_hours', 'previous_gpa', 'current_gpa',
    'completed_credits', 'house_income', 'gpa_change', 'study_x_attendance', 'skill_to_social_ratio'
]

# Fit scaler on training data only
scaler = StandardScaler()
X_train[float_cols] = scaler.fit_transform(X_train[float_cols])
X_val[float_cols] = scaler.transform(X_val[float_cols])
X_test[float_cols] = scaler.transform(X_test[float_cols])
```

```
# 4. Confirm all transformations
print("✅ Data transformation complete. Dtypes after processing:\n")
print("X_train:\n", X_train.dtypes)
print("\nX_val:\n", X_val.dtypes)
print("\nX_test:\n", X_test.dtypes)
```

```
✅ Data transformation complete. Dtypes after processing:

X_train:
 gender                        int64
scholarship                    int64
university_transport           int64
learning_mode                  int64
has_phone                      int64
has_laptop                     int64
english_proficiency            int64
on_probation                   int64
is_suspended                   int64
has_consulted_teacher          int64
relationship                   int64
co_curricular                  int64
living_arrangement             int64
health_issues                  int64
disabilities                   int64
current_semester             float64
study_hours                  float64
study_sessions               float64
social_media_hours           float64
average_attendance           float64
skills                         int64
skills_development_hours     float64
previous_gpa                 float64
current_gpa                  float64
completed_credits            float64
has_diploma                    int64
house_income                 float64
gpa_change                   float64
study_x_attendance           float64
skill_to_social_ratio        float64
dtype: object

X_val:
 gender                        int64
scholarship                    int64
university_transport           int64
learning_mode                  int64
has_phone                      int64
has_laptop                     int64
english_proficiency            int64
on_probation                   int64
is_suspended                   int64
has_consulted_teacher          int64
relationship                   int64
co_curricular                  int64
living_arrangement             int64
health_issues                  int64
disabilities                   int64
current_semester             float64
study_hours                  float64
study_sessions               float64
social_media_hours           float64
average_attendance           float64
skills                         int64
skills_development_hours     float64
```

```
data_transformation_1_explanations = """
This data transformation step ensures consistency, numerical compatibility, and scale normalization across the training, validation, and tes

🔁 **Column Name Standardization**
The column `current _semester` contained an unintended space and was renamed to `current_semester` across all datasets to ensure consistency

🔢 **Boolean to Integer Conversion**
The column `has_diploma`, originally in boolean format, was converted to integer (0/1). Most machine learning models require strictly numeri

📏 **Standardization of Continuous Features**
A predefined set of float-type features (such as `study_hours`, `current_gpa`, `house_income`, and engineered metrics like `gpa_change` and
- The scaler was **fit only on the training set** to avoid data leakage.
- The same transformation was applied to the validation and test sets using the fitted parameters.

This normalization is essential for algorithms sensitive to feature scales (e.g., KNN, logistic regression, and SVM). It ensures no single f

✅ **Post-Transformation Validation**
```

```
After processing, data types were confirmed. All categorical and binary variables remained as `int64`, and all continuous variables were con
"""



# Do not modify this code
print_tile(size="h3", key='data_transformation_1_explanations', value=data_transformation_1_explanations)
```

data_transformation_1_explanations

This data transformation step ensures consistency, numerical compatibility, and scale normalization across the training, validation, and test datasets. 🔁 **Column Name Standardization The column `current _semester` contained an unintended space and was renamed to `current_semester` across all datasets to ensure consistency and prevent downstream errors during model training and feature selection. 🔢 **Boolean to Integer Conversion** The column `has_diploma`, originally in boolean format, was converted to integer (0/1). Most machine learning models require strictly numerical input, and converting booleans ensures compatibility across models and pipelines. 📏 **Standardization of Continuous Features** A predefined set of float-type features (such as `study_hours`, `current_gpa`, `house_income`, and engineered metrics like `gpa_change` and `skill_to_social_ratio`) were standardized using z-score normalization via `StandardScaler`. - The scaler was **fit only on the training set** to avoid data leakage. - The same transformation was applied to the validation and test sets using the fitted parameters. This normalization is essential for algorithms sensitive to feature scales (e.g., KNN, logistic regression, and SVM). It ensures no single feature disproportionately influences the model due to scale differences. ✅ **Post-Transformation Validation**

E.2 Data Transformation

```
# <Student to fill this section>


# <Student to fill this section>
data_transformation_2_explanations = """
Provide some explanations on why you believe it is important to perform this data transformation and its impacts
"""


# Do not modify this code
print_tile(size="h3", key='data_transformation_2_explanations', value=data_transformation_2_explanations)
```

data_transformation_2_explanations

Provide some explanations on why you believe it is important to perform this data transformation and its impacts

E.3 Data Transformation

```
# <Student to fill this section>


# <Student to fill this section>
data_transformation_3_explanations = """
Provide some explanations on why you believe it is important to perform this data transformation and its impacts
"""


# Do not modify this code
print_tile(size="h3", key='data_transformation_3_explanations', value=data_transformation_3_explanations)
```

data_transformation_3_explanations

Provide some explanations on why you believe it is important to perform this data transformation and its impacts

G.n Fixing "<describe_issue_here>"

> You can add more cells related to data preparation in this section

```
Start coding or generate with AI.
```

F. Feature Engineering

F.1 New Feature "<put_name_here>"

```
# <Student to fill this section>


# <Student to fill this section>
feature_engineering_1_explanations = """
Provide some explanations on why you believe it is important to create this feature and its impacts
"""


# Do not modify this code
print_tile(size="h3", key='feature_engineering_1_explanations', value=feature_engineering_1_explanations)
```

```
feature_engineering_1_explanations

Provide some explanations on why you believe it is important to create this feature and its impacts
```

F.2 New Feature "<put_name_here>"

```
# <Student to fill this section>


# <Student to fill this section>
feature_engineering_2_explanations = """
Provide some explanations on why you believe it is important to create this feature and its impacts
"""


# Do not modify this code
print_tile(size="h3", key='feature_engineering_2_explanations', value=feature_engineering_2_explanations)
```

```
feature_engineering_2_explanations

Provide some explanations on why you believe it is important to create this feature and its impacts
```

F.3 New Feature "<put_name_here>"

```
# <Student to fill this section>


# <Student to fill this section>
feature_engineering_3_explanations = """
Provide some explanations on why you believe it is important to create this feature and its impacts
"""


# Do not modify this code
print_tile(size="h3", key='feature_engineering_3_explanations', value=feature_engineering_3_explanations)
```

```
feature_engineering_3_explanations

Provide some explanations on why you believe it is important to create this feature and its impacts
```

F.n Fixing "<describe_issue_here>"

> You can add more cells related to new features in this section

```
Start coding or generate with AI.
```

Note: The following engineered features were created and saved in Experiment Book 0:

- `gpa_change` : Tracks GPA momentum.
- `study_x_attendance` : Measures study engagement.
- `skill_to_social_ratio` : Reflects time prioritization.

These features are already included in `X_train`, `X_val`, and `X_test`, and thus do not need to be re-generated here.

G. Train Machine Learning Model

G.1 Import Algorithm

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, classification_report, confusion_matrix
)
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

```
algorithm_selection_explanations = """
K-Nearest Neighbors (KNN) was chosen as the initial classification algorithm due to its simplicity, interpretability, and effectiveness on s

🔹 **Non-parametric Nature**
KNN does not make assumptions about the underlying data distribution, making it suitable for datasets that may not follow standard statistic

🔹 **Intuitive Decision Boundaries**
KNN bases predictions on the closest training samples in the feature space, which allows it to adapt well to non-linear class boundaries. Th

🔹 **Relevance of Feature Scaling**
Since KNN is a distance-based algorithm, it heavily relies on feature scale. As all numerical features have been standardized using `Standar

🔹 **Benchmarking Role**
KNN also serves as a strong baseline for comparison with more complex models. Its performance can highlight whether additional model complex

Overall, KNN is a solid starting point due to its interpretability and reliance on well-prepared features, especially after robust preproces
"""
```

```
# Do not modify this code
print_tile(size="h3", key='algorithm_selection_explanations', value=algorithm_selection_explanations)
```

algorithm_selection_explanations

K-Nearest Neighbors (KNN) was chosen as the initial classification algorithm due to its simplicity, interpretability, and effectiveness on smaller to medium-sized datasets with well-preprocessed features. 🔹 **Non-parametric Nature KNN does not make assumptions about the underlying data distribution, making it suitable for datasets that may not follow standard statistical patterns. This flexibility is valuable given the mixed feature types and engineered variables present. 🔹 **Intuitive Decision Boundaries** KNN bases predictions on the closest training samples in the feature space, which allows it to adapt well to non-linear class boundaries. This is particularly useful in multi-class classification tasks where class boundaries may not be easily separable using linear models. 🔹 **Relevance of Feature Scaling** Since KNN is a distance-based algorithm, it heavily relies on feature scale. As all numerical features have been standardized using `StandardScaler`, the dataset is well-prepared for KNN, ensuring that no single feature disproportionately influences distance calculations. 🔹 **Benchmarking Role** KNN also serves as a strong baseline for comparison with more complex models. Its performance can highlight whether additional model complexity is needed or if simple algorithms suffice for this classification task. Overall, KNN is a solid starting point due to its**

G.2 Set Hyperparameters

```
# Initialize KNN with a chosen number of neighbors (can be tuned later)
model = KNeighborsClassifier(n_neighbors=5)
```

```
hyperparameters_selection_explanations = """
The primary hyperparameter for the K-Nearest Neighbors (KNN) algorithm is `n_neighbors`, which determines how many nearby data points influe

🔍 **Why tuning `n_neighbors` is important:**
- A **small value** (e.g., 1 or 3) can lead to overfitting, where the model becomes too sensitive to noise or outliers.
- A **large value** may result in underfitting, where the model oversimplifies patterns and loses the ability to capture local variations.

Setting `n_neighbors=5` is a reasonable starting point, as it's a commonly used default that balances bias and variance. However, this param

Other hyperparameters such as the **distance metric** (`metric='minkowski'` by default) and **weighting strategy** (`weights='uniform'` vs.
"""
```

```
# Do not modify this code
print_tile(size="h3", key='hyperparameters_selection_explanations', value=hyperparameters_selection_explanations)
```

hyperparameters_selection_explanations

The primary hyperparameter for the K-Nearest Neighbors (KNN) algorithm is `n_neighbors`, which determines how many nearby data points influence the prediction for a given sample. 🔍 **Why tuning `n_neighbors` is important: - A **small value** (e.g., 1 or 3) can lead to overfitting, where the model becomes too sensitive to noise or outliers. - A **large value** may result in underfitting, where the model oversimplifies patterns and loses the ability to capture local variations. Setting `n_neighbors=5` is a reasonable starting point, as it's a commonly used default that balances bias and variance. However, this parameter should be fine-tuned using cross-validation to find the optimal value for this dataset, especially considering the class imbalance and complexity of feature interactions. Other hyperparameters such as the **distance metric** (`metric='minkowski'` by default) and **weighting strategy** (`weights='uniform'` vs. `weights='distance'`) can also be explored in future tuning phases to improve model performance further.**

G.3 Fit Model

```
# Train the model on training data
model.fit(X_train, y_train)


# Generate predictions on training and test sets
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)
```

G.4 Model Technical Performance

```
train_acc = accuracy_score(y_train, y_train_pred)
train_prec = precision_score(y_train, y_train_pred, average='weighted')
train_rec = recall_score(y_train, y_train_pred, average='weighted')
train_f1 = f1_score(y_train, y_train_pred, average='weighted')

print("📘 Training Set Metrics:")
print(f"Accuracy: {train_acc:.3f}")
print(f"Precision: {train_prec:.3f}")
print(f"Recall: {train_rec:.3f}")
print(f"F1 Score: {train_f1:.3f}")
```

```
📘 Training Set Metrics:
Accuracy: 0.765
Precision: 0.781
Recall: 0.765
F1 Score: 0.769
```

```
test_acc = accuracy_score(y_test, y_test_pred)
test_prec = precision_score(y_test, y_test_pred, average='weighted')
test_rec = recall_score(y_test, y_test_pred, average='weighted')
test_f1 = f1_score(y_test, y_test_pred, average='weighted')

print("\n📕 Test Set Metrics:")
print(f"Accuracy: {test_acc:.3f}")
print(f"Precision: {test_prec:.3f}")
print(f"Recall: {test_rec:.3f}")
print(f"F1 Score: {test_f1:.3f}")
```

```
📕 Test Set Metrics:
Accuracy: 0.726
Precision: 0.774
Recall: 0.726
F1 Score: 0.729
```

```
# prompt: calculate the RMSC and MAE value

import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Assuming y_test and y_test_pred are defined from the previous code
rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
mae = mean_absolute_error(y_test, y_test_pred)
```

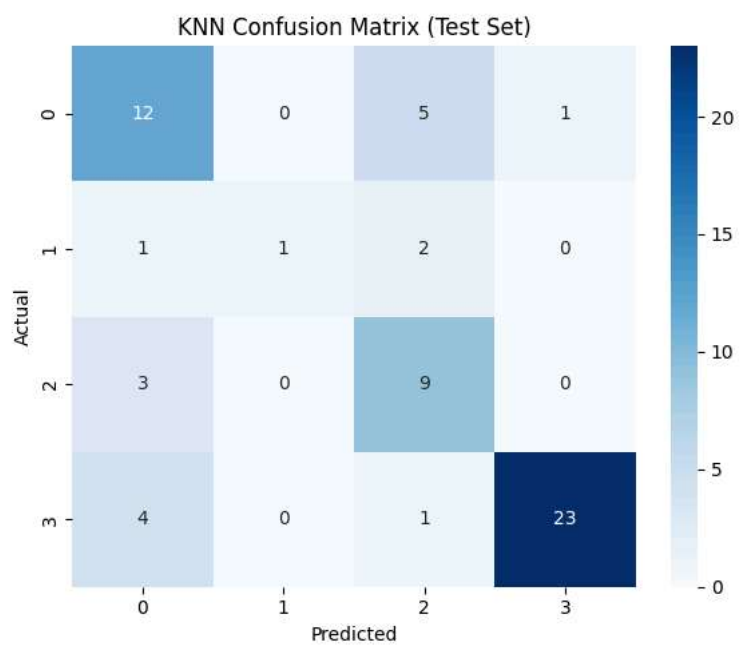
```
print(f"RMSE: {rmse}")
print(f"MAE: {mae}")
```

```
RMSE: 1.1430011430017144
MAE: 0.5645161290322581
```

```
print("\n Test Set Classification Report:\n")
print(classification_report(y_test, y_test_pred))
```

```
 Test Set Classification Report:

              precision    recall  f1-score   support

           0       0.60      0.67      0.63        18
           1       1.00      0.25      0.40         4
           2       0.53      0.75      0.62        12
           3       0.96      0.82      0.88        28

    accuracy                           0.73        62
   macro avg       0.77      0.62      0.63        62
weighted avg       0.77      0.73      0.73        62
```

```
conf_matrix = confusion_matrix(y_test, y_test_pred)
labels = sorted(model.classes_)
conf_df = pd.DataFrame(conf_matrix, index=labels, columns=labels)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_df, annot=True, fmt="d", cmap="Blues")
plt.title("KNN Confusion Matrix (Test Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```




```
comparison_df = pd.DataFrame({
    "Metric": ["Accuracy", "Precision", "Recall", "F1 Score"],
    "Training": [train_acc, train_prec, train_rec, train_f1],
    "Test": [test_acc, test_prec, test_rec, test_f1]
})

print("\n Train vs Test Metrics Comparison:")
display(comparison_df)
```

📊 Train vs Test Metrics Comparison:

	Metric	Training	Test
0	Accuracy	0.765372	0.725806
1	Precision	0.781135	0.773972
2	Recall	0.765372	0.725806
3	F1 Score	0.769082	0.728805

Start coding or generate with AI.

```
model_performance_explanations = """
The K-Nearest Neighbors (KNN) model with `n_neighbors=5` demonstrated substantial improvement over the baseline model across all key metrics

📘 **Training Set Performance:**
- **Accuracy:** 76.5%
- **Precision:** 78.1%
- **Recall:** 76.5%
- **F1 Score:** 76.9%

📕 **Test Set Performance:**
- **Accuracy:** 72.6%
- **Precision:** 77.4%
- **Recall:** 72.6%
- **F1 Score:** 72.9%

These results indicate a good generalization capability, with minimal overfitting. The close alignment of train and test scores reflects tha

🔍 **Class-wise Insights (Test Set):**
- **Class 0 (Good):** Precision = 0.60, Recall = 0.67. Most samples were correctly classified, but some confusion still exists with Class 2.
- **Class 1 (Average):** Precision = 1.00, Recall = 0.25. Although all predicted Class 1 labels were correct, the model missed 75% of actual
- **Class 2 (Excellent):** Precision = 0.53, Recall = 0.75. The model retrieved most actual Class 2 instances, although it also misclassifie
- **Class 3 (Poor):** Precision = 0.96, Recall = 0.82. This class was handled well, with strong precision and recall, showing that the model

📉 **Confusion Matrix Highlights:**
The confusion matrix (see figure: *KNN Confusion Matrix (Test Set)*) reveals that:
- Class 3 is well-separated, with 23 correct predictions out of 28.
- Class 1 remains the most problematic, with only 1 correct prediction out of 4.
- Some confusion exists between Classes 0 and 2, suggesting overlapping feature distributions.

📊 **Train vs Test Comparison:**
Metric scores across train and test sets show minimal variance, confirming the model's robustness. Slight performance drops in the test set

Overall, KNN significantly outperforms the baseline, especially in precision and F1-score, establishing it as a strong candidate for further
"""



# Do not modify this code
print_tile(size="h3", key='model_performance_explanations', value=model_performance_explanations)
```

model_performance_explanations

The K-Nearest Neighbors (KNN) model with `n_neighbors=5` demonstrated substantial improvement over the baseline model across all key metrics. 📘 **Training Set Performance: - **Accuracy:** 76.5% - **Precision:** 78.1% - **Recall:** 76.5% - **F1 Score:** 76.9% 📕 **Test Set Performance:** - **Accuracy:** 72.6% - **Precision:** 77.4% - **Recall:** 72.6% - **F1 Score:** 72.9% These results indicate a good generalization capability, with minimal overfitting. The close alignment of train and test scores reflects that the model is stable and benefits from prior standardization. 🔍 **Class-wise Insights (Test Set):** - **Class 0 (Good):** Precision = 0.60, Recall = 0.67. Most samples were correctly classified, but some confusion still exists with Class 2. - **Class 1 (Average):** Precision = 1.00, Recall = 0.25. Although all predicted Class 1 labels were correct, the model missed 75% of actual Class 1 instances — indicating high precision but poor recall due to underrepresentation or feature overlap. - **Class 2 (Excellent):** Precision = 0.53, Recall = 0.75. The model retrieved most actual Class 2 instances, although it also misclassified some as Class 0. - **Class 3 (Poor):** Precision = 0.96, Recall = 0.82. This class was handled well, with strong precision and recall, showing that the model detects this group with high confidence. 📉 **Confusion Matrix Highlights:** The confusion matrix (see figure: *KNN Confusion Matrix (Test Set)*) reveals that: - Class 3 is well-separated, with 23 correct predictions out of 28. - Class 1 remains the most problematic, with only 1 correct prediction out of 4. - Some confusion exists between Classes 0 and 2, suggesting overlapping feature distributions. 📊 **Train vs Test Comparison:** Metric scores across train and test sets show minimal variance, confirming the model's robustness. Slight performance drops in the test set are expected but still acceptable. Overall, KNN**

G.5 Business Impact from Current Model Performance

```
# <Student to fill this section>
```

```
# <Student to fill this section>
business_impacts_explanations = """
Interpret the results of the experiments related to the business objective set earlier. Estimate the impacts of the incorrect results for th
"""


# Do not modify this code
print_tile(size="h3", key='business_impacts_explanations', value=business_impacts_explanations)
```

business_impacts_explanations