KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

(Deemed to be University)

SCHOOL OF COMPUTER APPLICATIONS

AUTUMN SEMESTER 2025-26

MCA & MSc. 3rd Semester DA Lab Report

Course title: Data Analytics Lab (MC5191) Roll: 2470106 Name: Bijayeeni Halder

Serial No.	Assignment	Assignment Title
1	**Data Collection**	Collect any open-source dataset and complete the following. 1. Locate open-source data and provide its source, i.e., URL of the website. 2. Load the dataset into the pandas frame
2	**Report Generation**	Generate a Profile Report of the dataset using the "ydata-profiling" library and analyse the following: 1. Missing data 2. Variable description 3. Types of variables 4. Dimension of the data set, etc.
3	**Data Preprocessing**	Perform a basic data preprocessing, such as: 1. Delete all duplicate data from the dataset 2. Replace the missing values with a proper value/text 3. If variables are not in the correct data type, then apply proper type conversions. 4. Perform label encoding to convert categorical variables into numerical variables.
4	**Descriptive Statistics**	Perform the following data descriptive analysis, such as: 1. Skewness of the variables 2. Outliers of the variables
5	**Data Transformations**	Perform the following data transformations, such as: 1. Normalise all features with an appropriate method, among min-max, decimal scaling, z-score, etc. 2. Perform dimension reduction using an appropriate method among PCA, LDA, FDA, etc.
6	**Performance Analysis**	Perform the following for the predictive analysis of the target variable. 1. Apply four supervised/unsupervised machine learning algorithms appropriate for your dataset. 2. Compare their effectiveness using their suitable evaluation methods, such as the confusion matrix, RMSE, etc.

Predicting Crop Production across Indian States (1997–2020)

A Data Analytics Project

Bijayeeni Halder

November 4, 2025

Abstract

This report presents an end-to-end data analytics workflow for predicting crop production using a multi-state Indian agricultural dataset spanning 1997–2020. The analysis includes data import and profiling, exploratory data analysis (EDA), preprocessing (deduplication, label encoding, IQR-based outlier removal), model training and evaluation (Random Forest, XGBoost, AdaBoost, KNN), scaling and PCA experiments, and cross-validation. XGBoost emerged as the best model ($R^2 \approx 0.959$; RMSE ≈ 1405). The notebook, profiling report, and reproducible code are provided for transparency.

Contents

1 Introduction **5**

2 Background **5**

3 Dataset Description **5**

3.1 Data source 5

3.2 Data import and profiling 6

4 Exploratory Data Analysis (EDA) **6**

4.1 Missing values and duplicates 6

4.2 Basic statistics and distribution of Production 6

4.3 Categorical features and label encoding 6

4.4 Skewness 6

4.5 Correlation analysis 7

5 Preprocessing **7**

5.1 Deduplication and missing-value outcome 7

5.2 Label encoding 7

5.3 Outlier detection and removal 8

6 Modeling **9**

6.1 Train/Test split 9

6.2 Algorithms evaluated 9

6.3 Evaluation metrics 9

7 Results **9**

7.1 Interpretation 9

8 Scaling Experiments **10**

9 PCA and Dimensionality Reduction **10**

10 Cross-Validation **11**

11 Discussion **11**

12 Reproducibility and Code **11**

13 Conclusion **12**

A Selected Code Snippets **12**

A.1 Import, profiling and deduplication 12

A.2 Label encoding and skewness 13

A.3 Outlier removal (iterative IQR) 13

A.4 Model training and evaluation 13

A.5 Scaling experiment (XGBoost baseline) 14

A.6 PCA experiment 15

A.7 Cross-validation (XGBoost) 15

B How to Use This Template in Overleaf **15**

1 Introduction

Agriculture is a critical sector in India, and accurate crop production prediction helps in policy planning, resource allocation, and food security. Recent studies have demonstrated the value of machine learning models, particularly tree-based ensembles like Random Forest and XGBoost, in achieving high accuracy for crop yield prediction in various Indian regions [3, 2, 4]. This project uses a public dataset covering crop, season, state-level area, production, rainfall, and input usage (fertilizer, pesticide) from 1997–2020 to build predictive models for crop production. The analysis investigates data distributions, missing values, encoding strategies, outlier handling, and comparative model performance.

2 Background

Accurate crop production forecasting has long been a key objective in agricultural analytics, supporting national food security planning and resource allocation. Traditional statistical models capture temporal trends but often miss nonlinear, high-dimensional relationships between agronomic inputs, climate variables, and production. Ensemble and gradient-boosting methods have become standard tools for tabular agricultural prediction tasks.

Recent Indian studies report consistent gains from tree-based ensembles. Uppugunduri et al. [3] found Random Forest and XGBoost outperform linear methods for South Indian yield prediction. Singh et al. [4] demonstrated robust generalization of tree ensembles across heterogeneous agro-climatic regions. Kumar et al. [2] confirmed ensemble adaptability to region-specific rainfall and soil conditions. Dey et al. [1] further illustrate the value of climatic and soil covariates for crop recommendation systems. Building on these works, this study applies and compares multiple models on a nationwide dataset (1997–2020) and documents a reproducible preprocessing-to-evaluation pipeline.

3 Dataset Description

3.1 Data source

`https://www.kaggle.com/datasets/akshatgupta7/crop-yield-in-indian-states-dataset`

The dataset contains the following columns:

- **Crop**: crop name (categorical)
- **Crop_Year**: year (1997–2020)
- **Season**: cropping season (e.g., Kharif, Rabi, Whole Year)
- **State**: Indian state
- **Area**: cultivated area (hectares)
- **Production**: production (metric tons) — the primary target
- **Annual_Rainfall**: rainfall in mm
- **Fertilizer**: fertilizer used (kg)
- **Pesticide**: pesticide used (kg)
- **Yield**: calculated as Production / Area (metric tons per hectare)

3.2 Data import and profiling

The dataset was imported from Google Drive in Colab. A data profile was generated using `ydata_profiling` (saved as `profile_report.html`). Initial cleaning steps included deduplication (`df.drop_duplicates()`) and structural checks (`df.info()`, `df.isnull().sum()`). Import code snippet used in the notebook:

```
# Mount drive and load
from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/DA-LAB/crop_yield.csv')

# Profile
from ydata_profiling import ProfileReport
profile = ProfileReport(df, title="Crop Yield Dataset  Profile
    Report", explorative=True)
profile.to_file("profile_report.html")
```

4 Exploratory Data Analysis (EDA)

4.1 Missing values and duplicates

The dataset contained no missing values (all columns reported zero nulls via `df.isnull().sum()`) after loading. Duplicate rows were removed prior to analysis. Because there are no missing values, no imputation was required.

4.2 Basic statistics and distribution of Production

Summary statistics for `Production` revealed a strongly right-skewed distribution: mean > median > mode, with the mode at 0 (many zero production entries). Specifically, the median showed roughly 13,804 (noting the dataset's scale), indicating half the observations have production below that level. A histogram and KDE show heavy right tail (few large-production observations inflate the mean).

4.3 Categorical features and label encoding

Categorical columns (`Crop`, `Season`, `State`) were encoded with `LabelEncoder` for compact numeric representation suitable for tree-based models. The encoded DataFrame was used for correlation analysis and modeling.

4.4 Skewness

Skewness measured on encoded/numeric columns indicates strong right-skew in several variables (representative values observed in profiling):

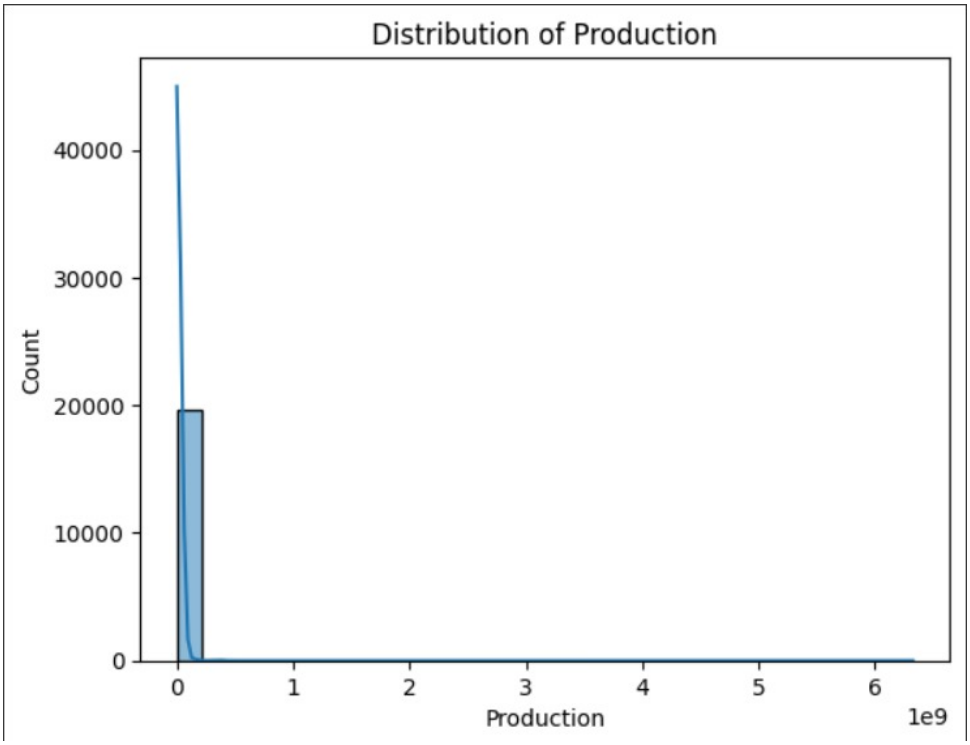

Figure 1: Distribution of Production (histogram + KDE).

- `Pesticide`: 25.63
- `Area`: 21.85
- `Production`: 19.29
- `Fertilizer`: 13.41
- `Yield`: 12.78

Tree-based methods are robust to these skews; however, linear models, KNN, and PCA-based approaches benefit from scaling or log transformation.

4.5 Correlation analysis

A correlation heatmap was computed on the label-encoded dataset to inspect pairwise relationships and to guide feature engineering efforts.

5 Preprocessing

5.1 Deduplication and missing-value outcome

Duplicates were dropped; no null values were present so no imputation was required.

5.2 Label encoding

Categorical variables were label-encoded using `sklearn.preprocessing.LabelEncoder` and stored in `df_encoded`.

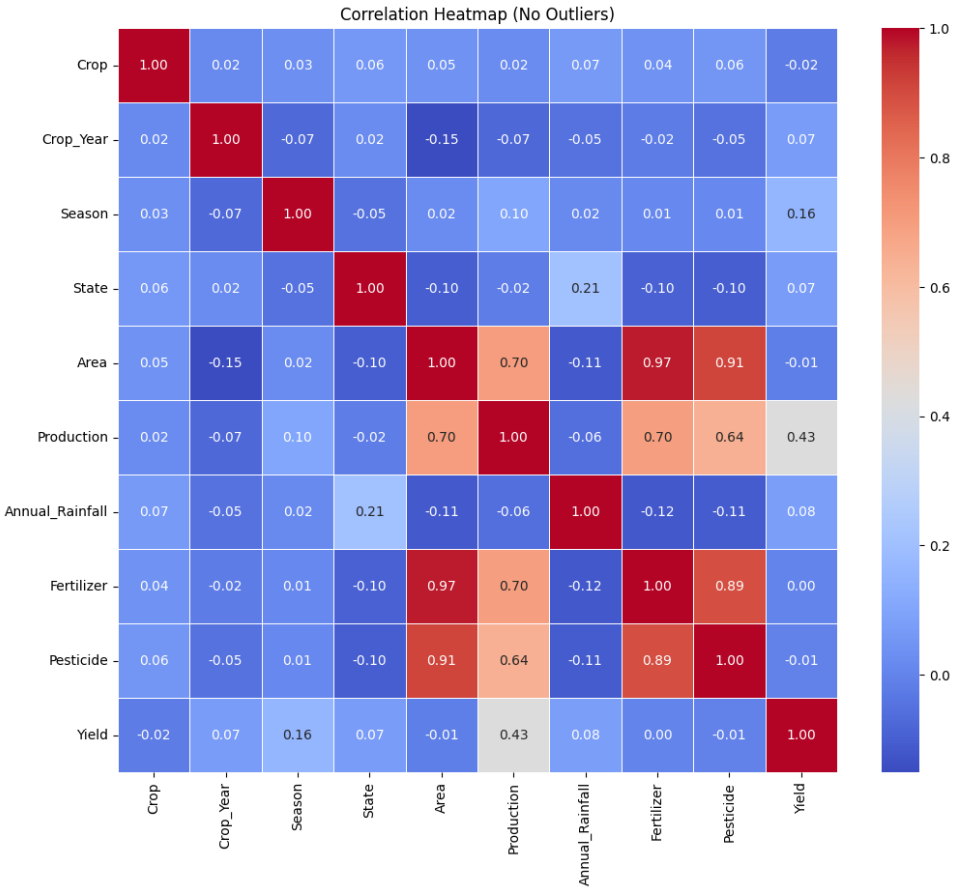

Figure 2: Correlation heatmap (label-encoded features).

5.3 Outlier detection and removal

An iterative IQR-based outlier filtering was applied: for each column the interquartile range (IQR) was computed and rows outside $[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$ were removed. This aggressive approach reduced variance from extreme values but also reduced sample size; use with caution and verify that genuine extreme observations are not inadvertently removed.

Code excerpt used for outlier removal:

```
df_no_outliers = df_encoded.copy()
for col in df_no_outliers:
    Q1 = df_no_outliers[col].quantile(0.25)
    Q3 = df_no_outliers[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df_no_outliers[(df_no_outliers[col] >=
        lower_bound) & (df_no_outliers[col] <= upper_bound)]
```

6 Modeling

6.1 Train/Test split

After preprocessing (encoding and outlier removal) features and target were separated:

```
X = df_no_outliers.drop('Production', axis=1)
y = df_no_outliers['Production']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)
```

6.2 Algorithms evaluated

The following regressors were trained and evaluated:

- Random Forest Regressor (`sklearn.ensemble.RandomForestRegressor`)
- XGBoost Regressor (`xgboost.XGBRegressor`)
- AdaBoost Regressor (`sklearn.ensemble.AdaBoostRegressor`)
- K-Nearest Neighbors Regressor (`sklearn.neighbors.KNeighborsRegressor`)

6.3 Evaluation metrics

Model performance was evaluated using:

- R^2 (coefficient of determination)
- RMSE (root mean squared error)
- MAE (mean absolute error)

7 Results

Representative test-set results after preprocessing and outlier removal are summarized below.

Model	R^2 Score	RMSE	MAE
Random Forest	0.96	1437.29	434.24
XGBoost	0.959	1405.23	479.03
AdaBoost	0.72	3658.51	3076.57
K-Nearest Neighbors	0.34	5641.49	2631.84

Table 1: Model performance on test set (after preprocessing and outlier removal).

7.1 Interpretation

- **Random Forest** and **XGBoost** perform best by a large margin, explaining roughly 96% of variance and yielding RMSE around 1.4k metric tons.
- **AdaBoost** is moderately effective ($R^2 \approx 0.72$), but higher RMSE indicates larger prediction errors.

- **KNN** performed poorly on raw features and without dimensionality reduction/scaling ($R^2 \approx 0.34$).

8 Scaling Experiments

Multiple scalers were compared with XGBoost as a reference model: MinMax, Standard, Robust, L2 Normalizer, QuantileTransformer (uniform and normal), PowerTransformer (Yeo-Johnson), and a log transform. Results show tree-based models (XGBoost) are largely insensitive to feature scaling; nearly all scalers produced very similar $R^2 \approx 0.959$ and RMSE $\approx$ 1405, with L2 Normalization performing worse. A log-transform helps some distance-based / linear methods but is not required for tree ensembles.

Code snippet for scaling experiment:

```
from sklearn.pipeline import Pipeline
scalers = {
    "Min-Max": MinMaxScaler(),
    "Z-Score (Standard)": StandardScaler(),
    "Robust Scaling": RobustScaler(),
    "L2 Normalization": Normalizer(norm='l2'),
    "Quantile Uniform": QuantileTransformer(output_distribution='
        uniform'),
    "Quantile Normal": QuantileTransformer(output_distribution='
        normal'),
    "Power Transformer": PowerTransformer(method='yeo-johnson'),
    "Log Transform": 'log'
}
# Loop and evaluate using XGBRegressor as shown in the notebook
```

9 PCA and Dimensionality Reduction

PCA was applied after standard scaling to retain 95% variance. PCA reduced dimensionality but slightly degraded performance for tree-based models:

- XGBoost (original features): $R^2 \approx 0.959$, RMSE $\approx$ 1405
- XGBoost (PCA, 95% variance): $R^2 \approx 0.934$, RMSE $\approx$ 1775

Tree-based models are able to exploit nonlinear interactions in the original feature space, so PCA (a linear compression) may remove discriminative information and reduce predictive performance.

Code excerpt:

```
from sklearn.decomposition import PCA
X_scaled = StandardScaler().fit_transform(X_full)
pca = PCA(n_components=0.95, svd_solver='full')
X_pca = pca.fit_transform(X_scaled)
```

10 Cross-Validation

A 5-fold cross-validation was performed for the XGBoost model:

`XGB 5-fold CV | RMSE: (noted around test RMSE), R²: (mean ~0.95+)`

(See the notebook for the exact printed values produced by `cross_val_score`; the test-split results reported above are consistent with the CV summary.)

11 Discussion

- **Best models:** XGBoost and Random Forest consistently gave top-tier performance. XGBoost slightly outperformed Random Forest on RMSE in this run.
- **Data characteristics:** Many zero-production entries and very heavy right skew require careful domain verification (are zeros true zeros or sentinel/missing encodings?). Outlier removal improved model stability but reduces sample size; consider winsorization or robust models if preserving extremes is important.
- **Scaling and PCA:** Tree ensembles are resilient to scaling; scaling benefits KNN and linear models. PCA reduced dimensionality but caused a modest drop in XGBoost performance, so avoid PCA if interpretability of original features is desired and tree models are used.
- **Next steps:** Hyperparameter tuning (grid search or Bayesian optimization) for XGBoost/Random Forest, richer feature engineering (temporal lags, rolling rainfall aggregates, interactions), and careful treatment/verification of zero-production rows will likely yield further gains. Consider stacking/ensembling and region-specific models.

12 Reproducibility and Code

The full Colab notebook with code, plots, and model checkpoints is available at:

Google Colab: https://colab.research.google.com/drive/1Bx774fthKqK04hh5Ali-ca4BX9AsGVpE?usp=sharing

Key steps to reproduce:

1. Mount Google Drive and load `crop_yield.csv`.
2. Generate profiling report with `ydata_profiling` and inspect.
3. Remove duplicate rows, label-encode categorical columns.
4. (Optional) Apply log-transform to skewed numeric features for linear/KNN models.
5. Apply IQR-based outlier removal (or alternative robust strategies).
6. Split data, train models, evaluate with RMSE and R^2, and perform cross-validation.

13 Conclusion

This project demonstrates an end-to-end pipeline for predicting crop production across Indian states (1997–2020). Tree-based ensemble methods, particularly XGBoost, produced the best predictive performance ($R^2 \approx 0.959$, RMSE ≈ 1405). PCA and dimensionality reduction did not improve tree-based model performance. Future work should focus on careful treatment of zero-production records, targeted feature engineering (climate aggregates, soil and NPK if available), and systematic hyperparameter optimization.

Acknowledgements

Thanks to publicly available agricultural datasets and the Python data science ecosystem (pandas, scikit-learn, xgboost, ydata-profiling, seaborn, matplotlib).

References

[1] Ahmed R Dey B, Ferdous J. Machine learning based recommendation of agricultural and horticultural crop farming in india under the regime of npk, soil ph and three climatic variables. *PMC*, 2024. https://pmc.ncbi.nlm.nih.gov/articles/PMC10844259/.

[2] A. Kumar et al. Machine learning based crop yield prediction model in rajasthan region of india. In *IEEE Conference 2024*, 2024. https://ieeexplore.ieee.org/document/10442746/.

[3] Uppugunduri Vijay Nikhil, Athiya M. Pandiyan, S. P. Raja, and Zoran Stamenkovic. Machine learning-based crop yield prediction in south india: Performance analysis of various models. *Computers*, 13(6), 2024. https://www.mdpi.com/2073-431X/13/6/137.

[4] R. Singh et al. Enhancing crop yield prediction in india: A comparative analysis of machine learning models. *IEEE Access*, 2023. https://ieeexplore.ieee.org/document/10442746/.

A Selected Code Snippets

A.1 Import, profiling and deduplication

```
# Mount drive and load
from google.colab import drive
drive.mount('/content/drive')
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/DA-LAB/crop_yield.csv')

# Profile
from ydata_profiling import ProfileReport
```

```
profile = ProfileReport(df, title="Crop Yield Dataset  Profile
    Report", explorative=True)
profile.to_file("profile_report.html")

# Remove duplicates
df = df.drop_duplicates()
```

A.2 Label encoding and skewness

```
from sklearn.preprocessing import LabelEncoder
df_encoded = df.copy()
categorical_cols = df.select_dtypes(include=['object']).columns
le = LabelEncoder()
for col in categorical_cols:
    df_encoded[col] = le.fit_transform(df_encoded[col])

# Skewness table
skew_table = df_encoded.skew().sort_values(ascending=False)
print(skew_table)
```

A.3 Outlier removal (iterative IQR)

```
df_no_outliers = df_encoded.copy()
for col in df_no_outliers:
    Q1 = df_no_outliers[col].quantile(0.25)
    Q3 = df_no_outliers[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df_no_outliers[(df_no_outliers[col] >=
        lower_bound) & (df_no_outliers[col] <= upper_bound)]
```

A.4 Model training and evaluation

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor,
    AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
import numpy as np

X = df_no_outliers.drop('Production', axis=1)
y = df_no_outliers['Production']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=42)

models = {
    "Random Forest": RandomForestRegressor(random_state=42),
    "XGBoost": XGBRegressor(random_state=42),
    "AdaBoost": AdaBoostRegressor(random_state=42),
    "K-Nearest Neighbors": KNeighborsRegressor()
}

results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    mae = mean_absolute_error(y_test, y_pred)
    results[name] = [r2, rmse, mae]
    print(f"{name}: R2={r2:.3f}, RMSE={rmse:.2f}, MAE={mae:.2f}")
```

A.5 Scaling experiment (XGBoost baseline)

```
from sklearn.preprocessing import MinMaxScaler, StandardScaler,
    RobustScaler, Normalizer, QuantileTransformer, PowerTransformer
from sklearn.pipeline import Pipeline

scalers = {
    "Min-Max": MinMaxScaler(),
    "Z-Score (Standard)": StandardScaler(),
    "Robust Scaling": RobustScaler(),
    "L2 Normalization": Normalizer(norm='l2'),
    "Quantile Uniform": QuantileTransformer(output_distribution='
        uniform'),
    "Quantile Normal": QuantileTransformer(output_distribution='
        normal'),
    "Power Transformer": PowerTransformer(method='yeo-johnson'),
    "Log Transform": 'log'
}

best_model = XGBRegressor(random_state=42)
scaling_results = {}
for name, scaler in scalers.items():
    if name == "Log Transform":
        X_train_log = np.log1p(X_train)
        X_test_log = np.log1p(X_test)
        best_model.fit(X_train_log, y_train)
```

```
        y_pred = best_model.predict(X_test_log)
    else:
        pipeline = Pipeline([('scaler', scaler), ('model',
            best_model)])
        pipeline.fit(X_train, y_train)
        y_pred = pipeline.predict(X_test)
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    scaling_results[name] = [r2, rmse]
```

A.6 PCA experiment

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

X_full = df_encoded.drop(columns=['Production'])
X_scaled = StandardScaler().fit_transform(X_full)
pca = PCA(n_components=0.95, svd_solver='full')
X_pca = pca.fit_transform(X_scaled)
print("PCA components:", pca.n_components_, "Explained variance:",
    pca.explained_variance_ratio_.sum())
```

A.7 Cross-validation (XGBoost)

```
from sklearn.model_selection import cross_val_score, KFold
cv = KFold(n_splits=5, shuffle=True, random_state=42)
cv_rmse = (-cross_val_score(models["XGBoost"], X, y, scoring='
    neg_root_mean_squared_error', cv=cv)).mean()
cv_r2 = (cross_val_score(models["XGBoost"], X, y, scoring='r2', cv=
    cv)).mean()
print(f"XGB 5-fold CV  RMSE: {cv_rmse:.2f}, R : {cv_r2:.3f}")
```

B How to Use This Template in Overleaf

1. Open Overleaf and create a new project ("Blank Project").
2. Copy-paste the contents of this file into the `main.tex` file.
3. Ensure the images referenced in `figures/` are present (or update the paths to your new plot files).
4. Compile to generate the report.