University for the Creative Arts

BERLIN SCHOOL OF BUSINESS & INNOVATION

Essay / Assignment Title:

Regression and classification analysis for informed decision-making

Programme title:

Predictive Analytics and Machine Learning using Python

Name: Sima Niaztalkhouncheh

Year:2025

CONTENTS

Statement of compliance with academic ethics and the avoidance of plagiarism

I honestly declare that this dissertation is entirely my own work and none of its part has been copied from printed or electronic sources, translated from foreign sources and reproduced from essays of other researchers or students. Wherever I have been based on ideas or other people texts I clearly declare it through the good use of references following academic ethics.

(In the case that is proved that part of the essay does not constitute an original work, but a copy of an already published essay or from another source, the student will be expelled permanently from the program).

Name and Surname (Capital letters):

SIMA NIAZTALKHOUNCHEH

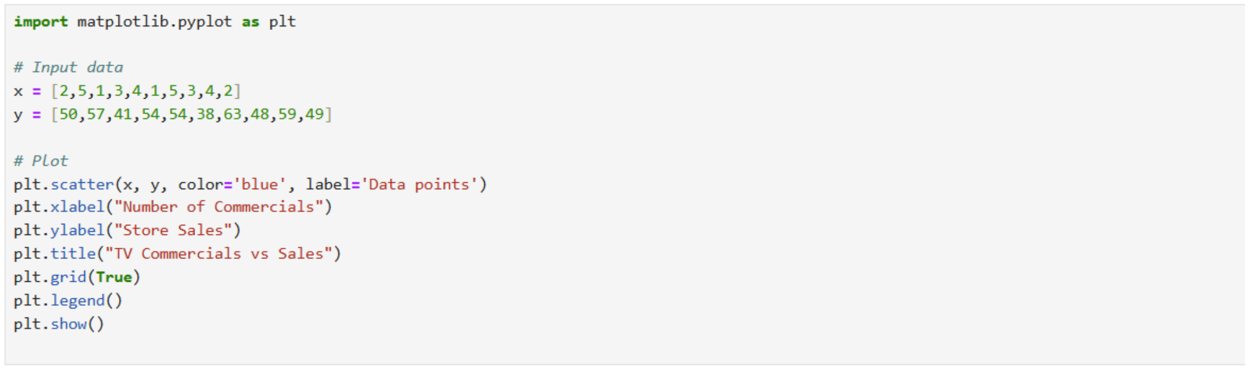
Date: 2025/06/06

Task 1 - Linear Relationship with TV Commercials

In this task, we considered how the weekend TV advertisements affect store sales the following week. By graphing the data, we could see a clear rising trend, so there is a positive linear relationship. We checked this twice using the Pearson correlation coefficient (around 0.9) and had good linearity. We then went ahead with linear regression to obtain a straight line fit, and the model was good. This means linear regression is adequate for this data.

We used linear regression to fit a line through the data, assuming a linear relationship between commercials and sales.
This method is suitable when the correlation is strong and the residuals are normally distributed (James et al., 2013).

Graphing and visual inspection:



```
import matplotlib.pyplot as plt

# Input data
x = [2,5,1,3,4,1,5,3,4,2]
y = [50,57,41,54,54,38,63,48,59,49]

# Plot
plt.scatter(x, y, color='blue', label='Data points')
plt.xlabel("Number of Commercials")
plt.ylabel("Store Sales")
plt.title("TV Commercials vs Sales")
plt.grid(True)
plt.legend()
plt.show()
```

Figure 1Graphing Code

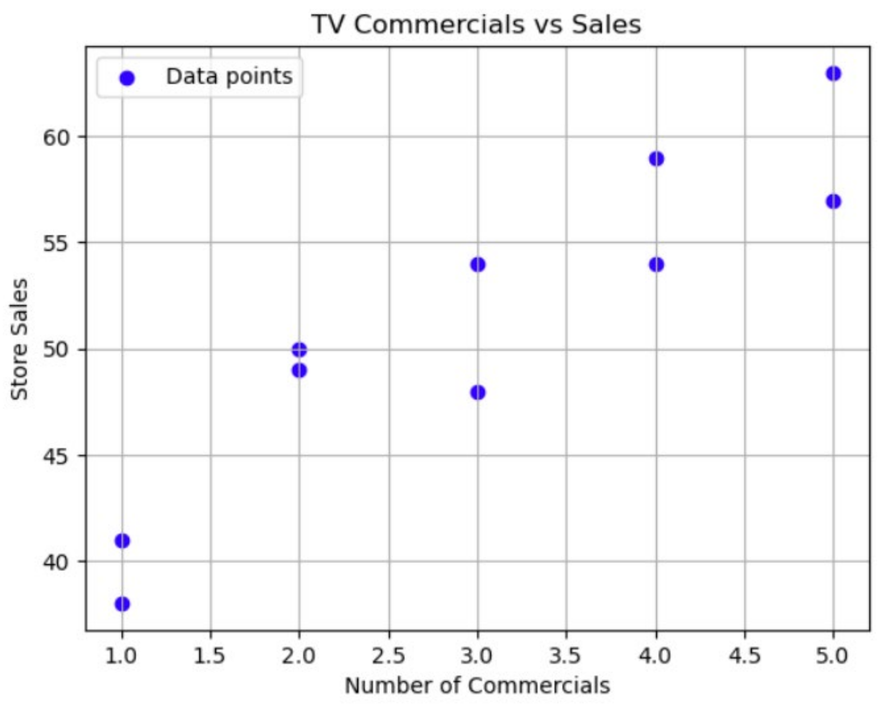
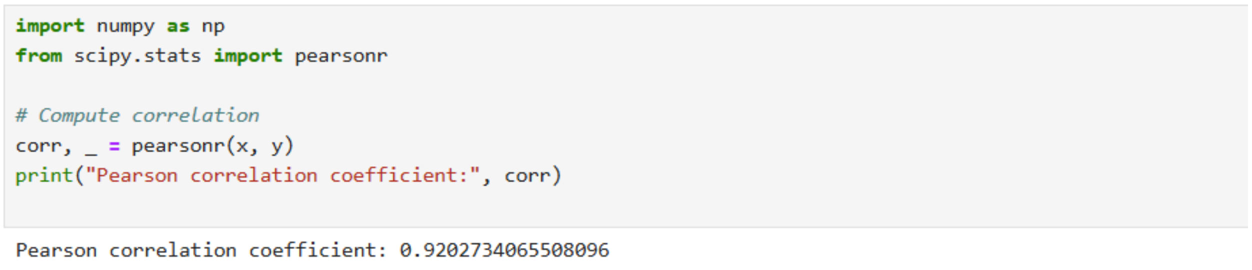


Figure 2The relationship between the number of television commercials on weekends and store sales

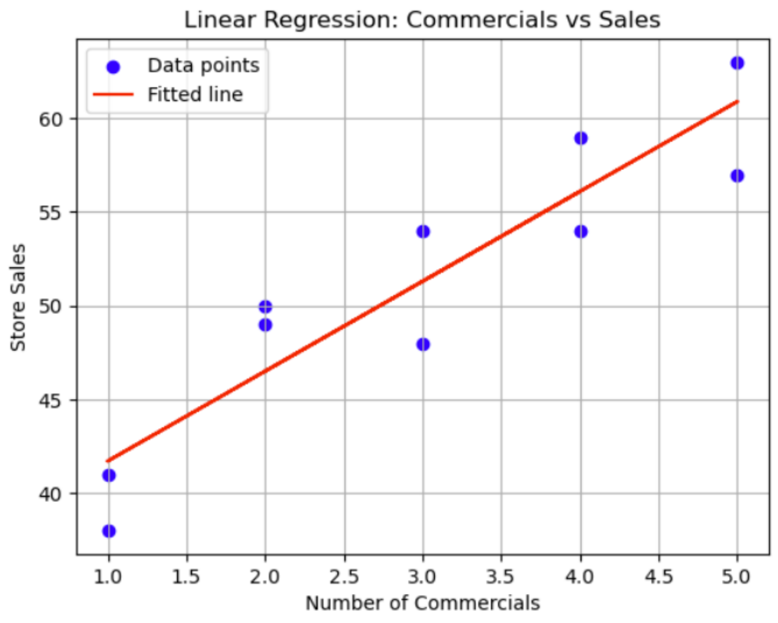
Visual analysis: There seems to be a nearly linear relationship between the number of ads and sales.



```
import numpy as np
from scipy.stats import pearsonr

# Compute correlation
corr, _ = pearsonr(x, y)
print("Pearson correlation coefficient:", corr)
```

```
Pearson correlation coefficient: 0.9202734065508096
```

Figure 3Pearson Correlation



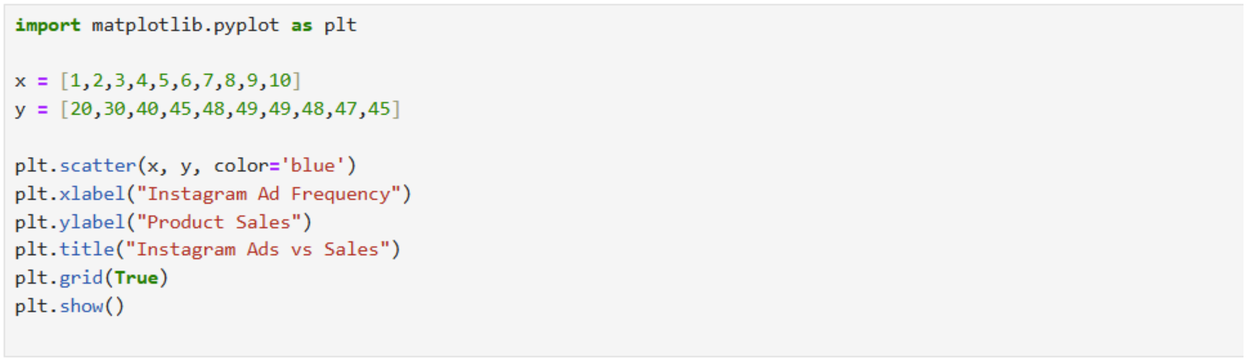
```
Intercept: 36.9
Slope: 4.799999999999999
```

Figure 4Linear Regression with Scikit-learn

Task 2 – Non-Linear Relationship with Instagram Ads

In this task, we considered the impact of the frequency of Instagram advertisements on sales of a product. Although the correlation coefficient was around 0.75 (very strong), the data exhibited a curved pattern: sales increased at first but then remained steady or decreased slightly. This suggested a non-linear relationship. The data did not fit well to a linear regression model, so we used polynomial regression (degree 2), which followed the shape of the data much better and yielded lower prediction error (James et al., 2013).

Graphing and visual inspection:



```
import matplotlib.pyplot as plt

x = [1,2,3,4,5,6,7,8,9,10]
y = [20,30,40,45,48,49,49,48,47,45]

plt.scatter(x, y, color='blue')
plt.xlabel("Instagram Ad Frequency")
plt.ylabel("Product Sales")
plt.title("Instagram Ads vs Sales")
plt.grid(True)
plt.show()
```

Figure 5Graphing Code

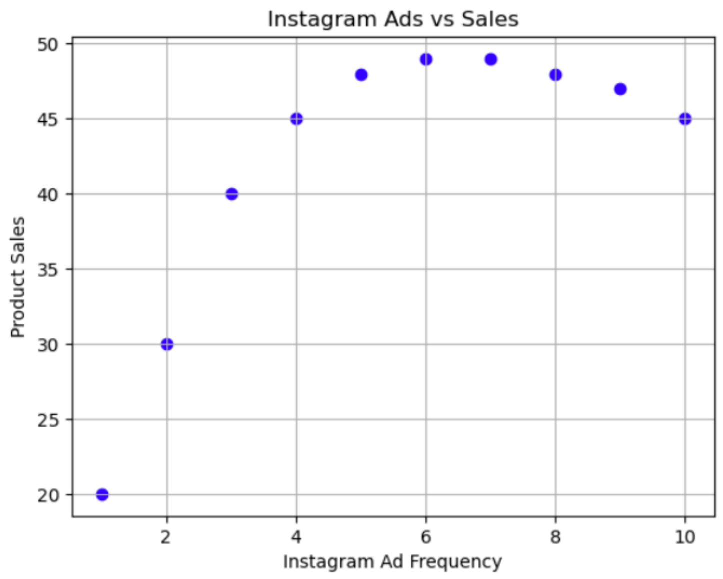
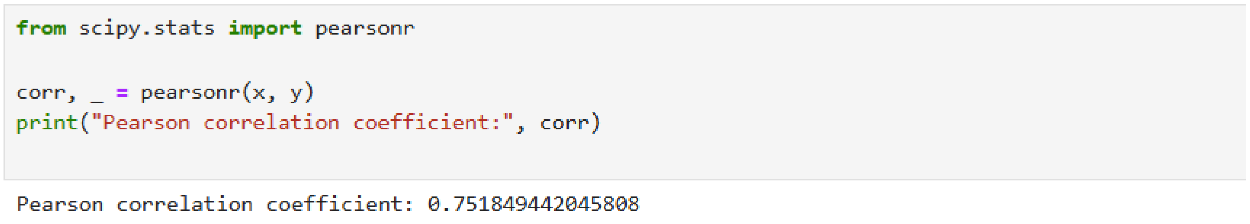


Figure 6The relationship between the number of times you advertise on Instagram and product sales

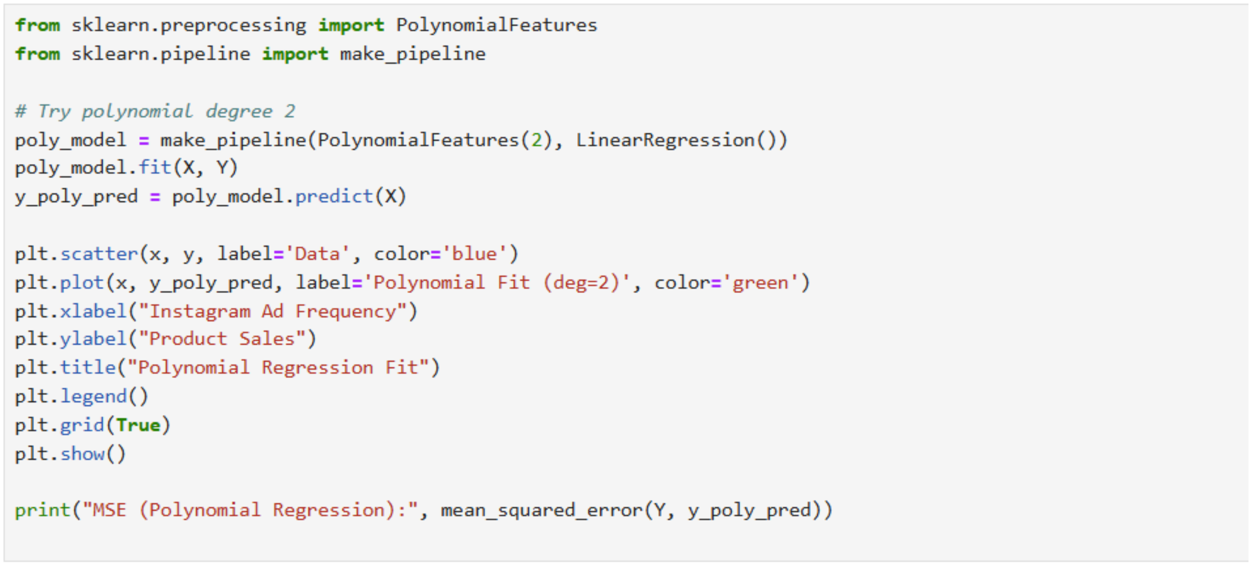
Visual analysis: Sales increase at first, but after a certain point they stabilize or even decrease.



```
from scipy.stats import pearsonr

corr, _ = pearsonr(x, y)
print("Pearson correlation coefficient:", corr)
```

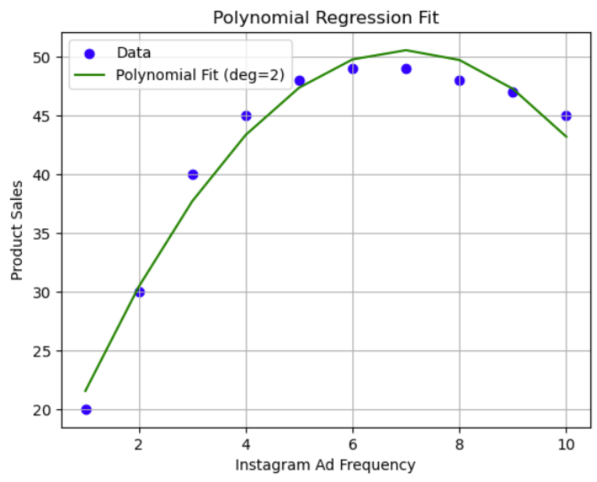
```
Pearson correlation coefficient: 0.751849442045808
```

Figure 7Correlation



```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Try polynomial degree 2
poly_model = make_pipeline(PolynomialFeatures(2), LinearRegression())
poly_model.fit(X, Y)
y_poly_pred = poly_model.predict(X)

plt.scatter(x, y, label='Data', color='blue')
plt.plot(x, y_poly_pred, label='Polynomial Fit (deg=2)', color='green')
plt.xlabel("Instagram Ad Frequency")
plt.ylabel("Product Sales")
plt.title("Polynomial Regression Fit")
plt.legend()
plt.grid(True)
plt.show()

print("MSE (Polynomial Regression):", mean_squared_error(Y, y_poly_pred))
```

Figure 8Polynomial Regression



```
MSE (Polynomial Regression): 2.035757575757582
```

Figure 9 nonlinear and the Polynomial Regression model

Task 3 – Fertilizer Prediction Using Machine Learning

1. Dataset Overview and Source

This dataset was selected because it offers a realistic and structured scenario for applying machine learning in agriculture. It contains both numerical and categorical features commonly found in real-world agricultural problems such as soil type, crop type, and nutrient levels in the soil. The task of predicting the appropriate fertilizer type is both practically valuable and technically challenging, especially given the multi-class nature of the target. Moreover, the dataset is large (750,000 rows), making it suitable for testing the scalability and performance of various machine learning models, from simple classifiers like SVM to more advanced ones like XGBoost.
Additionally, this dataset was part of a real Kaggle competition called **Playground Series - Season 5, Episode 6**, which provided an excellent context for experimentation, benchmarking, and comparison of models in a competitive setting.
The dataset used for this project was released as part of a Kaggle competition focused on fertilizer prediction (Kaggle, 2024).

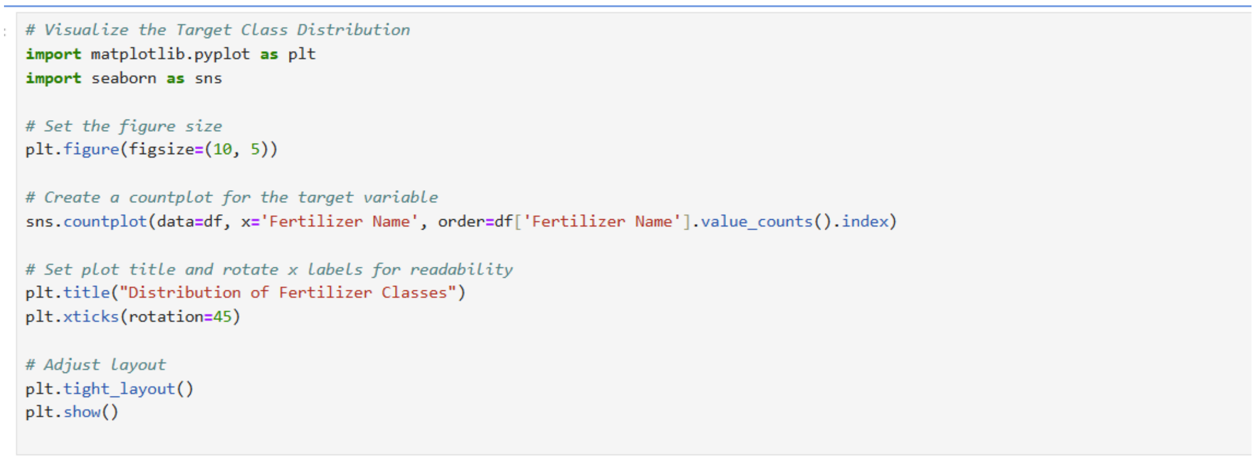
2. EDA

➢ **Loading and Initial Exploration of the Dataset**

- We loaded the train.csv file.
- The dataset contains **750,000 rows** and **10 columns**.
- There were **no missing values** in the dataset.
- The columns include:
  - **Numerical features:**
    Temparature, Humidity, Moisture, Nitrogen, Phosphorous, Potassium
  - **Categorical features:**
    Soil Type, Crop Type
  - **Target column:**
    Fertilizer Name contains 7 different fertilizer types : making this a **multi-class classification problem**

➢ Visualizing Target Class Distribution

- Using seaborn and countplot, we visualized the frequency of each fertilizer type in the Fertilizer Name column.
- This helped us check whether the target classes were balanced (they were).



```python
# Visualize the Target Class Distribution
import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size
plt.figure(figsize=(10, 5))

# Create a countplot for the target variable
sns.countplot(data=df, x='Fertilizer Name', order=df['Fertilizer Name'].value_counts().index)

# Set plot title and rotate x labels for readability
plt.title("Distribution of Fertilizer Classes")
plt.xticks(rotation=45)

# Adjust layout
plt.tight_layout()
plt.show()
```

Figure 10Visualize the Target Class Distribution

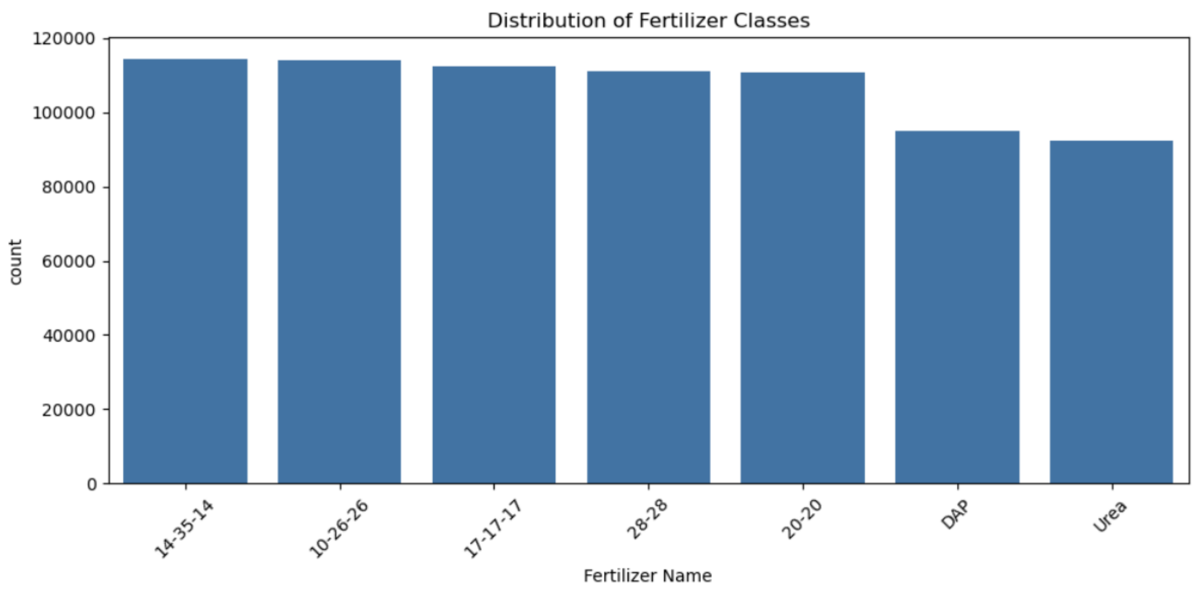
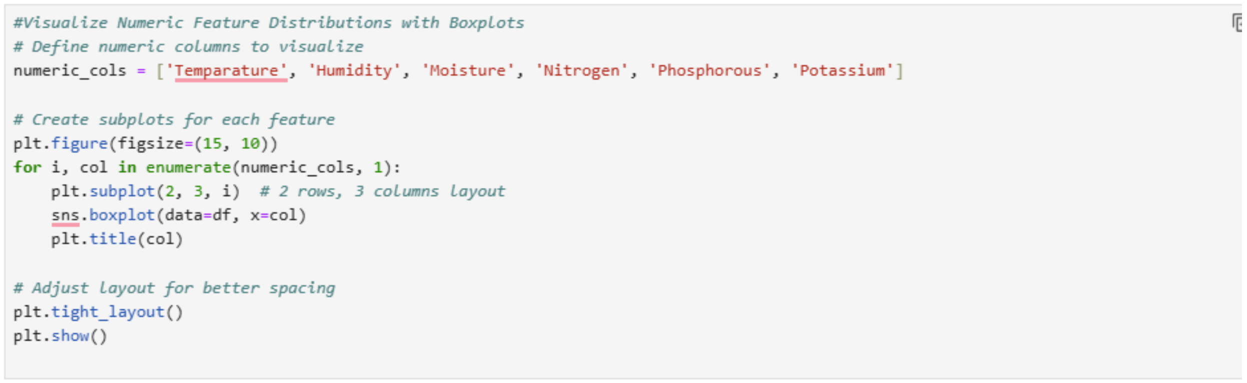


Figure 11Distribution of Fertilizer Classes

➢ **Exploring Distribution of Numerical Features**

- We used boxplot for each numerical feature to examine their ranges, distributions, and possible **outliers**.
- This was useful for later steps like **feature selection** and **feature scaling**.



```
#Visualize Numeric Feature Distributions with Boxplots
# Define numeric columns to visualize
numeric_cols = ['Temparature', 'Humidity', 'Moisture', 'Nitrogen', 'Phosphorous', 'Potassium']

# Create subplots for each feature
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(2, 3, i)  # 2 rows, 3 columns layout
    sns.boxplot(data=df, x=col)
    plt.title(col)

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```

Figure 12Visualize Numeric Feature Distributions with Boxplots

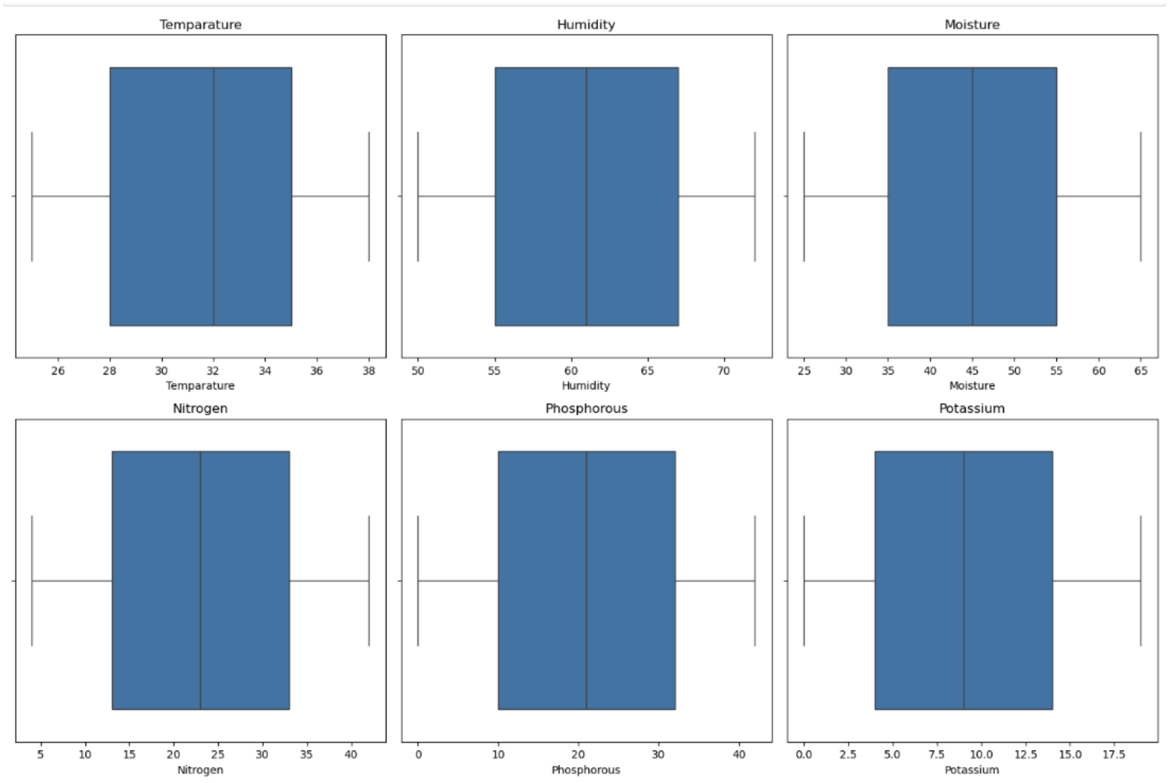
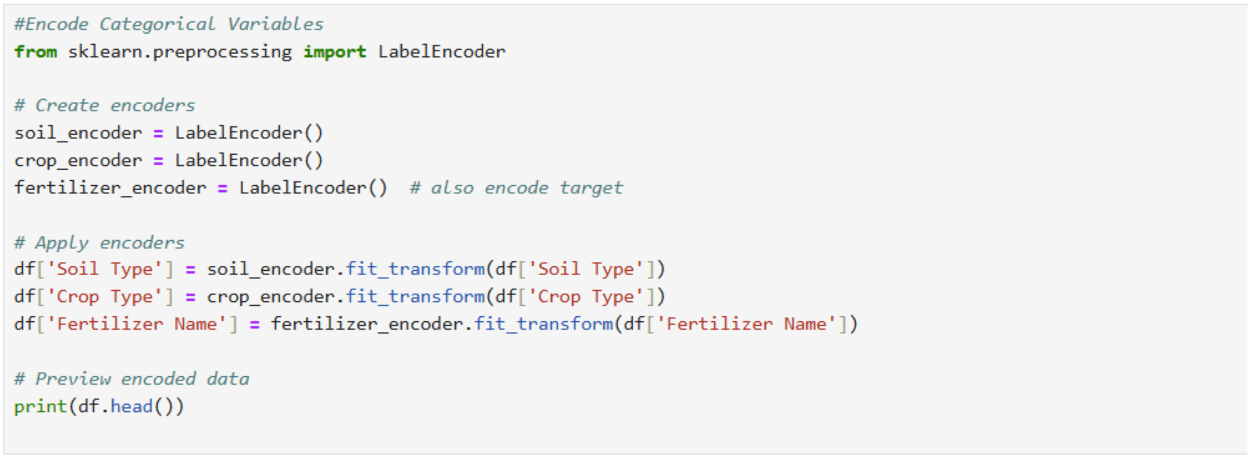


Figure 13Numeric Feature Distributions

➢ **Encoding Categorical Data**

- Since machine learning models like SVM can't handle text directly, we used Label Encoder to convert Soil Type and Crop Type into numeric values.
- We also encoded the target column Fertilizer Name numerically so it could be used with classification models like SVM and Decision Trees.



```
#Encode Categorical Variables
from sklearn.preprocessing import LabelEncoder

# Create encoders
soil_encoder = LabelEncoder()
crop_encoder = LabelEncoder()
fertilizer_encoder = LabelEncoder()  # also encode target

# Apply encoders
df['Soil Type'] = soil_encoder.fit_transform(df['Soil Type'])
df['Crop Type'] = crop_encoder.fit_transform(df['Crop Type'])
df['Fertilizer Name'] = fertilizer_encoder.fit_transform(df['Fertilizer Name'])

# Preview encoded data
print(df.head())
```

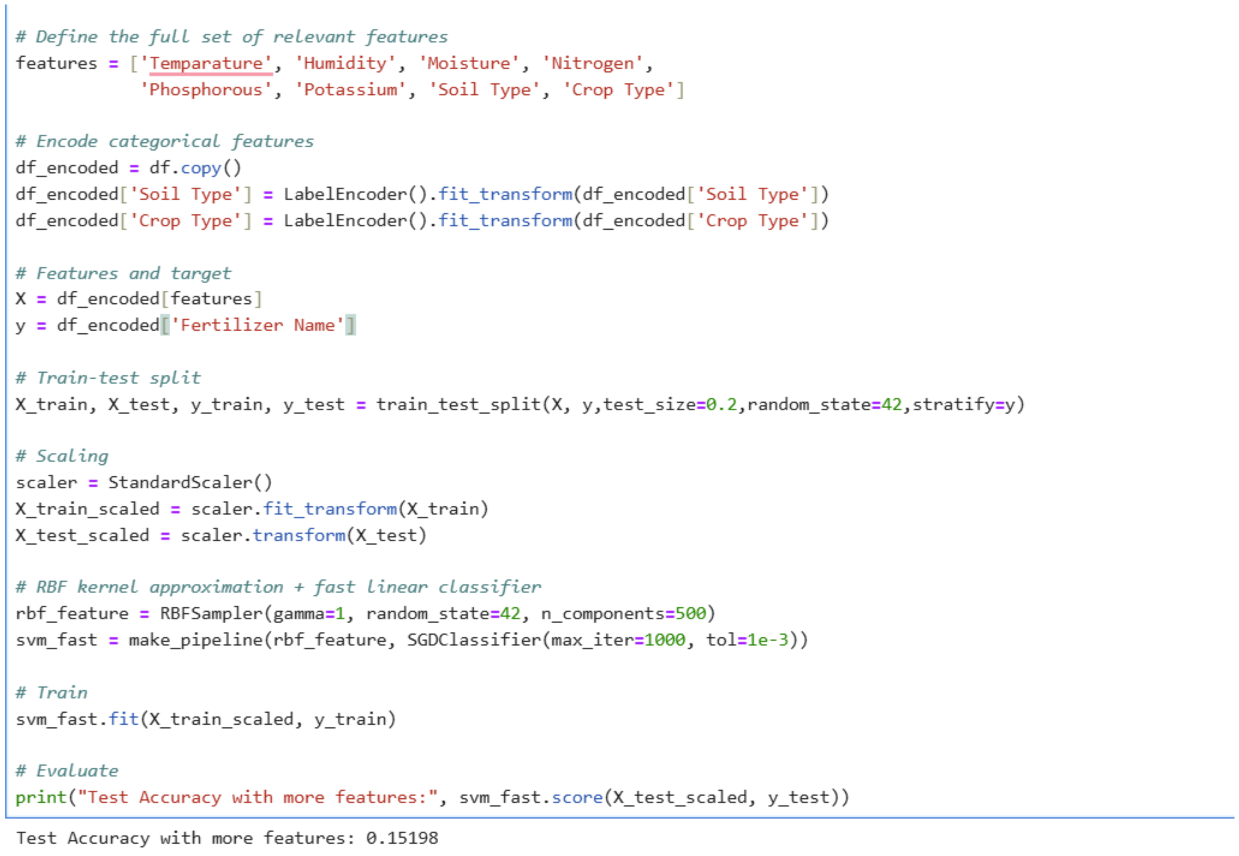
Figure 14Encode Categorical Variables

3. *Attempted SVM and RBF Approximation*

We initially attempted to train a Support Vector Machine (SVM) using only three features (Moisture, Soil Type, Crop Type) for binary classification (DAP vs Urea). SVM is known for performing well in low-dimensional problems with clear boundaries, and we chose it also because it allows 3D visualization of decision boundaries. However, due to the large dataset size (750,000 rows), the model took several hours to train and still did not converge. Additionally, the accuracy remained very low (around 15%), making it impractical for our use case.

We implemented both SVM and SGDClassifier with RBF kernel approximation using the Scikit-learn library (Pedregosa et al., 2011).

To address the speed issue, we tried approximating the RBF kernel using RBFSampler combined with a fast linear classifier (SGDClassifier). This reduced training time significantly and maintained the same kernel logic, but the accuracy did not improve and remained close to 15%. We concluded that both SVM and its fast approximation were ineffective for this dataset.



```
# Define the full set of relevant features
features = ['Temparature', 'Humidity', 'Moisture', 'Nitrogen',
            'Phosphorous', 'Potassium', 'Soil Type', 'Crop Type']

# Encode categorical features
df_encoded = df.copy()
df_encoded['Soil Type'] = LabelEncoder().fit_transform(df_encoded['Soil Type'])
df_encoded['Crop Type'] = LabelEncoder().fit_transform(df_encoded['Crop Type'])

# Features and target
X = df_encoded[features]
y = df_encoded['Fertilizer Name']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,random_state=42,stratify=y)

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

# RBF kernel approximation + fast linear classifier
rbf_feature = RBFSampler(gamma=1, random_state=42, n_components=500)
svm_fast = make_pipeline(rbf_feature, SGDClassifier(max_iter=1000, tol=1e-3))

# Train
svm_fast.fit(X_train_scaled, y_train)

# Evaluate
print("Test Accuracy with more features:", svm_fast.score(X_test_scaled, y_test))
```

```
Test Accuracy with more features: 0.15198
```

Figure 15SVM and RBF Approximation

4. 3D Visualization of Fertilizer Classes

Before fully committing to model training, we attempted to understand how well the fertilizer classes could be separated using only three features: Moisture, Soil Type, and Crop Type. These were selected as top features using SelectKBest. We visualized a small random sample of the dataset in 3D using these three features as axes and colored each point according to its fertilizer class.

This 3D scatter plot helped us intuitively observe class separation in feature space. While some patterns were visible, the classes were not clearly separable especially for a multi-class problem with 7 categories. Additionally, plotting decision boundaries in 3D for more than two classes is extremely complex and usually not done.

Nevertheless, this visualization provided an early insight:

A simple model using just these three features would likely struggle to distinguish all fertilizer types effectively.

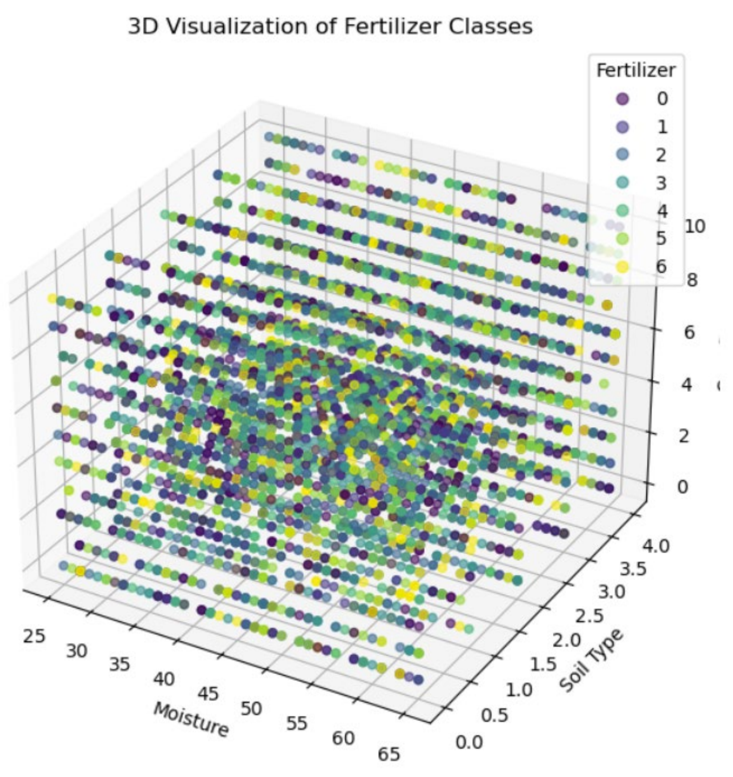


Figure 163D Visualization of Fertilizer Classes

5. Using XGBoost (Tree-Based Model)

We initially trained an XGBoost model on the full dataset with all 7 fertilizer classes using 8 core features. The result was disappointing:

XGBoost Test Accuracy: 17.09%

Issue	Possible Impact
Low feature quality	The selected features likely lacked strong predictive power.
Encoding problems	Using LabelEncoder for categorical features imposes false order.
Overlapping classes	Similar fertilizers may apply to similar crops/soils, confusing the model.
Lack of domain knowledge	The model has to guess complex relationships that humans understand through experience.

Table 1Key issues that likely affected performance

Strategy Shift: Simplify with Binary Classification

To improve model learnability and focus on structure, we simplified the task:

- Switched from 7-class prediction to binary classification (e.g. DAP vs Urea)
- Applied one-hot encoding for categorical variables
- Engineered new features such as nutrient ratios

This allowed us to evaluate whether the model can truly learn from data before scaling to all classes.

XGBoost was selected as the primary model due to its performance in tabular data and built-in feature importance tools (Chen and Guestrin, 2016).

6. XGBoost on Binary Classification (Urea vs DAP)

We simplified the problem by reducing it to only two fertilizer classes: DAP and Urea.

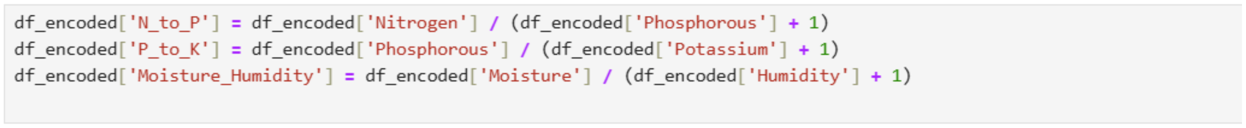
We trained XGBoost with:

- All features: Accuracy ~56%
- Engineered features (ratios) : Accuracy still ~56%
- Top 10 features via SelectKBest: Accuracy dropped to ~54%

Binary Classification Accuracy: 56.73%

While better than random guessing (50%), this accuracy showed that the model could only capture simple patterns.

We introduced ratio-based features such as N_to_P and P_to_K, which are commonly used in applied feature engineering strategies (Kuhn and Johnson, 2019).



```
df_encoded['N_to_P'] = df_encoded['Nitrogen'] / (df_encoded['Phosphorous'] + 1)
df_encoded['P_to_K'] = df_encoded['Phosphorous'] / (df_encoded['Potassium'] + 1)
df_encoded['Moisture_Humidity'] = df_encoded['Moisture'] / (df_encoded['Humidity'] + 1)
```

Figure 17To improve this, we applied feature engineering

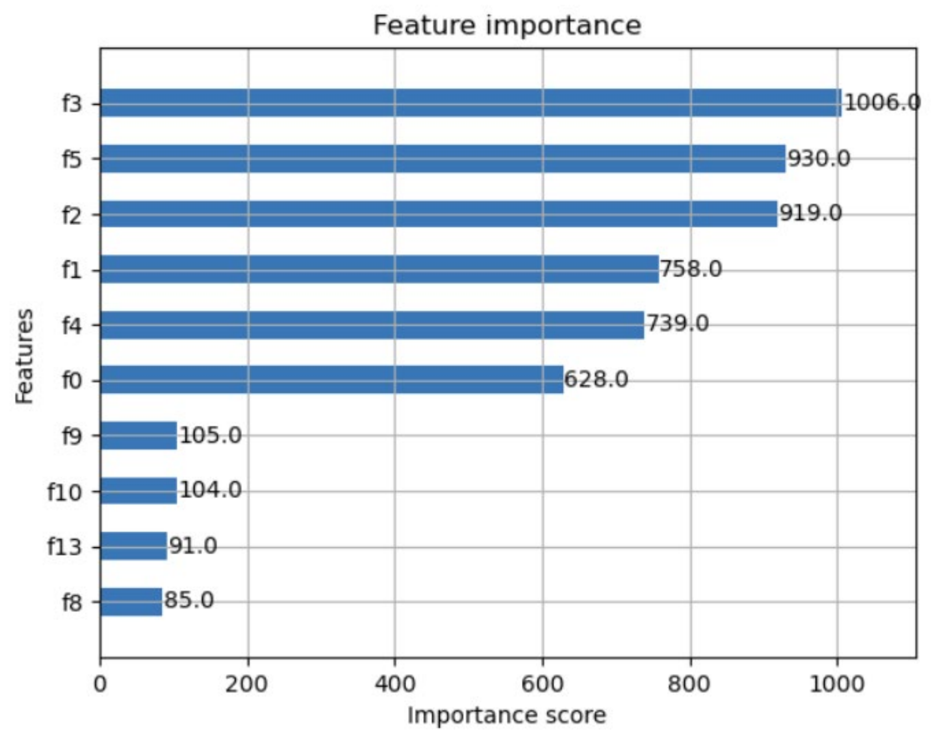
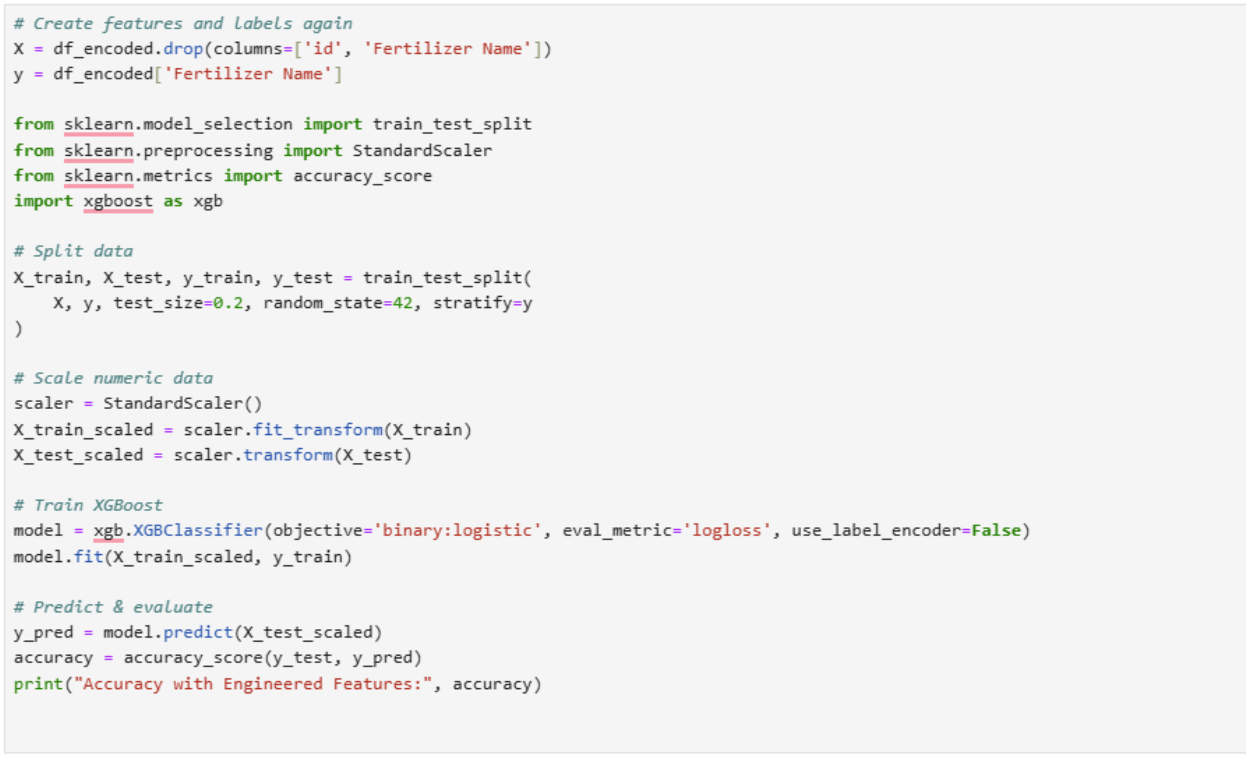


Figure 18explored feature importance using XGBoost's built-in plot

f0 = Temparature	**f1 = Humidity**	**f2 = Moisture**	**f3 = Nitrogen**	**f4 = Potassium**
f5 = Phosphorous	f6 = Soil Type_Black	f7 = Soil Type_Clayey	f8 = Soil Type_Loamy	f9 = Soil Type_Red
f10 = Soil Type_Sandy	f11 = Crop Type_Barley	f12 = Crop Type_Cotton	f13 = Crop Type_Ground Nuts	f14 = Crop Type_Maize
f15 = Crop Type_Millets	f16 = Crop Type_Oil seeds	f17 = Crop Type_Paddy	f18 = Crop Type_Pulses	f19 = Crop Type_Sugarcane
f20 = Crop Type_Tobacco	f21 = Crop Type_Wheat			

Table 2mapping of feature index to column name



```python
# Create features and labels again
X = df_encoded.drop(columns=['id', 'Fertilizer Name'])
y = df_encoded['Fertilizer Name']

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
import xgboost as xgb

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale numeric data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train XGBoost
model = xgb.XGBClassifier(objective='binary:logistic', eval_metric='logloss', use_label_encoder=False)
model.fit(X_train_scaled, y_train)

# Predict & evaluate
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with Engineered Features:", accuracy)
```

Figure 19Accuracy with Engineered Features

7. *Ensemble Modeling with VotingClassifier*

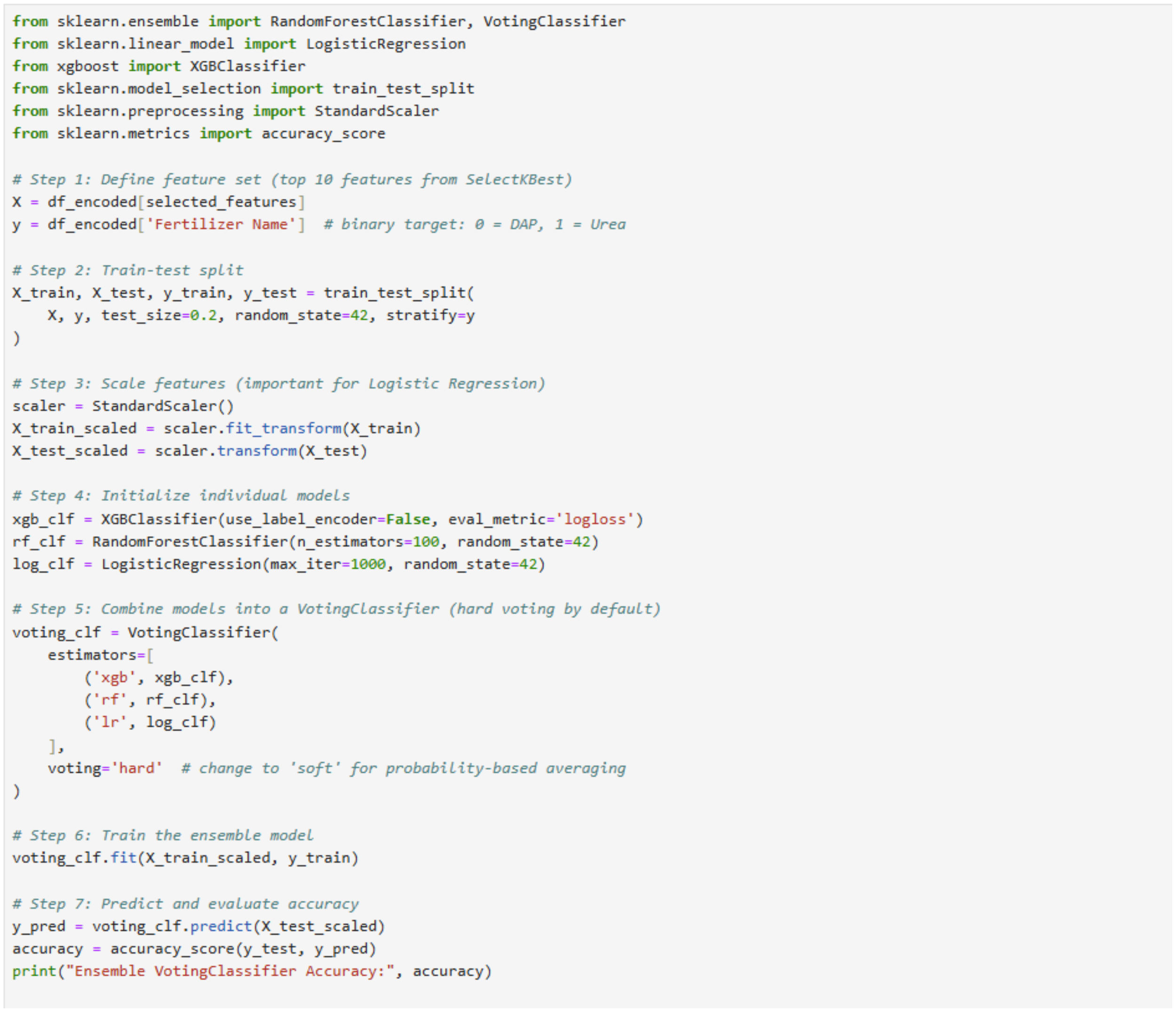
To explore whether combining multiple models would improve performance, we implemented an ensemble using VotingClassifier. The ensemble included three models:

- XGBoost (our strongest base model)
- RandomForestClassifier
- LogisticRegression

We tested both **hard voting** (majority class vote) and **soft voting** (average class probabilities):

- **Hard Voting Accuracy:** ~53.7%
- **Soft Voting Accuracy:** ~52.6%

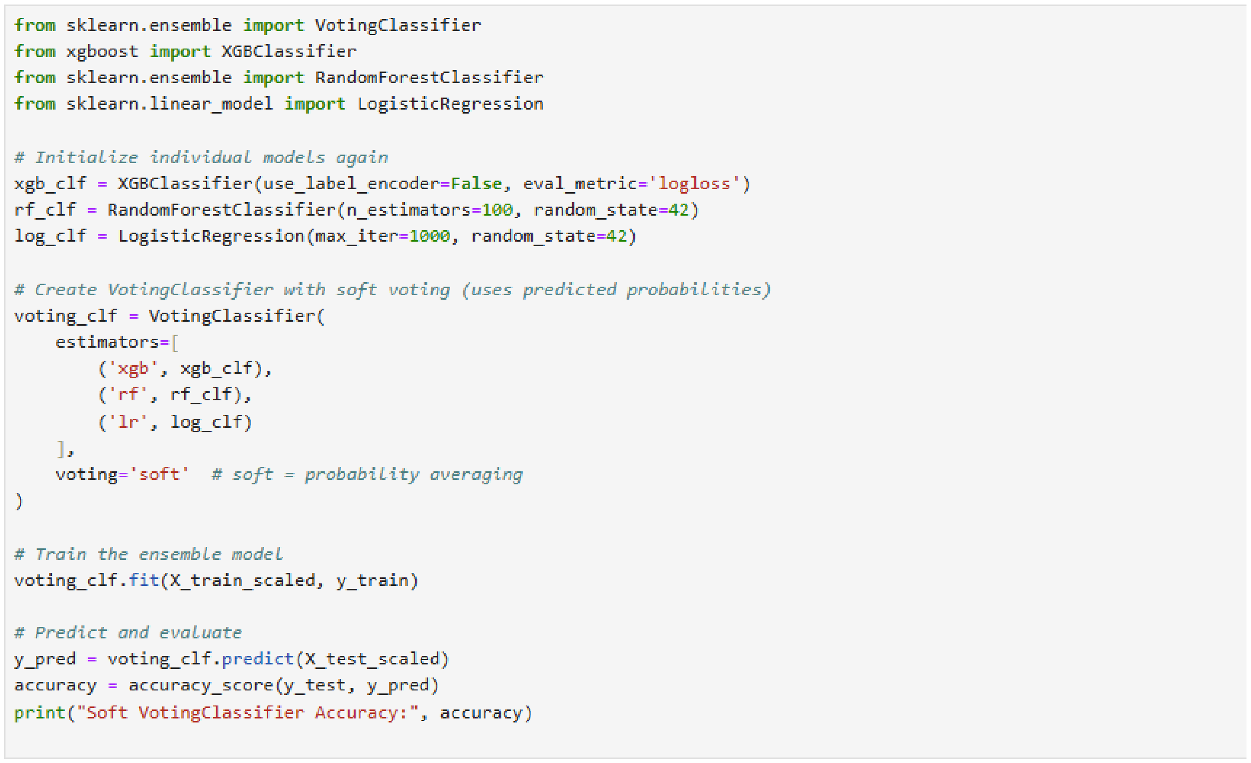
Surprisingly, the ensemble performed worse than using XGBoost alone. This is likely because RandomForest and LogisticRegression had significantly lower individual performance, and their contribution weakened the ensemble's overall accuracy. This experiment showed that simply combining models doesn't always lead to better results especially when weaker models are included without weighting or tuning.



```python
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Step 1: Define feature set (top 10 features from SelectKBest)
X = df_encoded[selected_features]
y = df_encoded['Fertilizer Name']  # binary target: 0 = DAP, 1 = Urea

# Step 2: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Step 3: Scale features (important for Logistic Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 4: Initialize individual models
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
log_clf = LogisticRegression(max_iter=1000, random_state=42)

# Step 5: Combine models into a VotingClassifier (hard voting by default)
voting_clf = VotingClassifier(
    estimators=[
        ('xgb', xgb_clf),
        ('rf', rf_clf),
        ('lr', log_clf)
    ],
    voting='hard'  # change to 'soft' for probability-based averaging
)

# Step 6: Train the ensemble model
voting_clf.fit(X_train_scaled, y_train)

# Step 7: Predict and evaluate accuracy
y_pred = voting_clf.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Ensemble VotingClassifier Accuracy:", accuracy)
```

Figure 20Ensemble Hard VotingClassifier

Ensemble Hard Voting Classifier Accuracy: 0.5374505823271717



```python
from sklearn.ensemble import VotingClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Initialize individual models again
xgb_clf = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
log_clf = LogisticRegression(max_iter=1000, random_state=42)

# Create VotingClassifier with soft voting (uses predicted probabilities)
voting_clf = VotingClassifier(
    estimators=[
        ('xgb', xgb_clf),
        ('rf', rf_clf),
        ('lr', log_clf)
    ],
    voting='soft'  # soft = probability averaging
)

# Train the ensemble model
voting_clf.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = voting_clf.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print("Soft VotingClassifier Accuracy:", accuracy)
```

Figure 21Ensemble Soft VotingClassifier

Soft Voting Classifier Accuracy: 0.5268992413719414

8. SHAP Analysis Explaining Model Decisions

To gain a deeper understanding of how our XGBoost model made predictions, we used **SHAP (SHapley Additive exPlanations)**. SHAP provides interpretable explanations by quantifying the contribution of each feature to the final model output for individual predictions.

SHAP was used to explain model decisions and feature impact through additive contribution values (Lundberg and Lee, 2017).

SHAP Summary Plot:

We first generated a SHAP summary plot using a sample from the test set. This plot ranks the most influential features based on their average absolute SHAP values.

- **Most important features:**
    - Moisture

- Phosphorous
- Nitrogen
  These features had the largest impact on the model's decisions across samples.

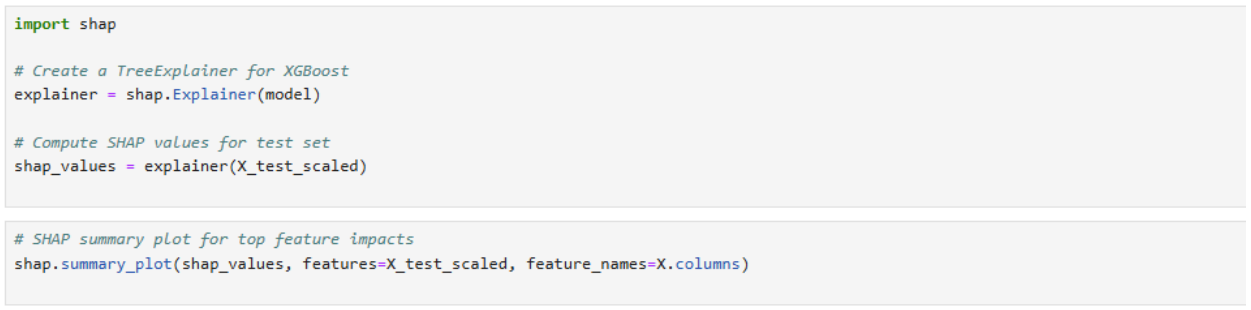
- **Less influential features:**
  - One-hot encoded variables like Soil Type and Crop Type had relatively low impact, suggesting they were not key discriminators in predicting the fertilizer type.

SHAP Force Plot (Single Sample):

To better interpret a specific prediction, we used a SHAP force plot for a **misclassified sample**. This visualization illustrated how each feature pushed the model's decision toward or away from the predicted class.

- **Red bars** indicated features that increased the likelihood of the predicted class.
- **Blue bars** indicated features that opposed it.
- The final prediction was the result of this balancing act.

This analysis revealed that even when the model was wrong, it followed a logical reasoning path based on the data it had seen further validating our feature engineering and helping us identify possible sources of confusion in the feature space.



```python
import shap

# Create a TreeExplainer for XGBoost
explainer = shap.Explainer(model)

# Compute SHAP values for test set
shap_values = explainer(X_test_scaled)
```

```python
# SHAP summary plot for top feature impacts
shap.summary_plot(shap_values, features=X_test_scaled, feature_names=X.columns)
```

Figure 22SHAP summary plot for top feature impacts

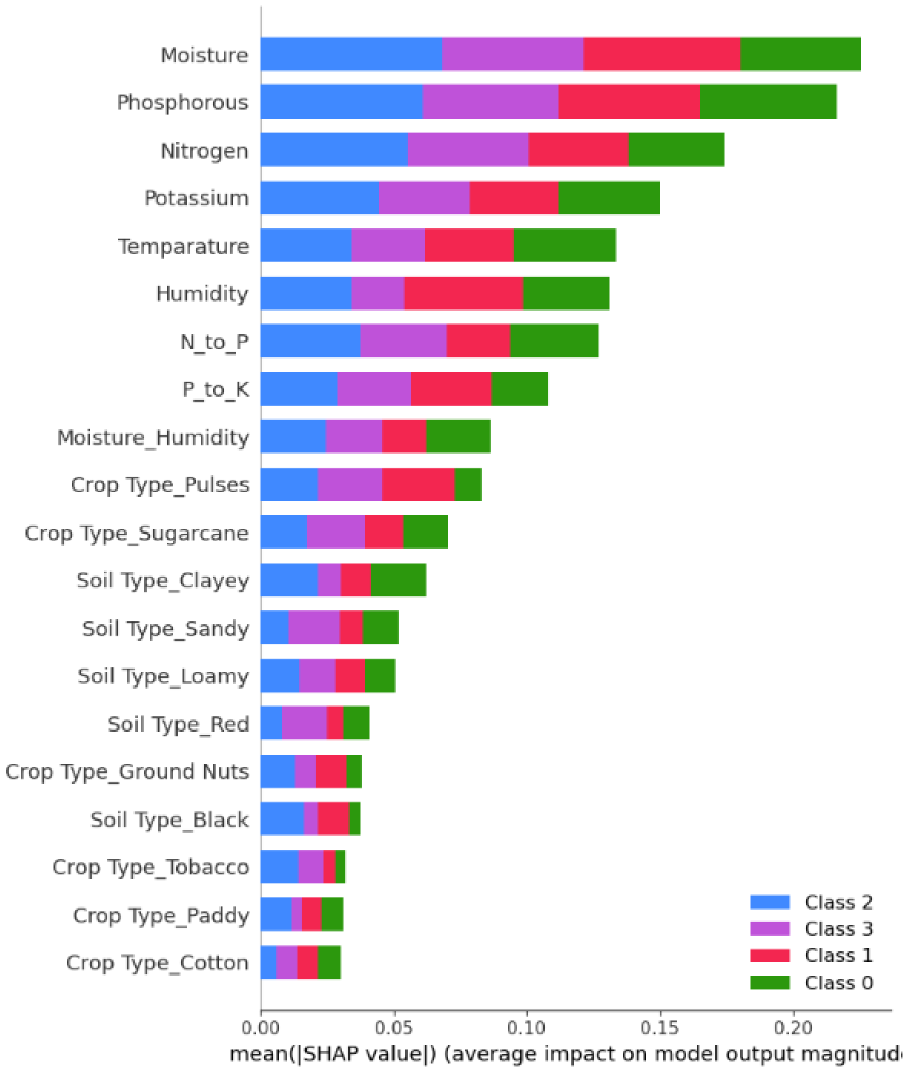
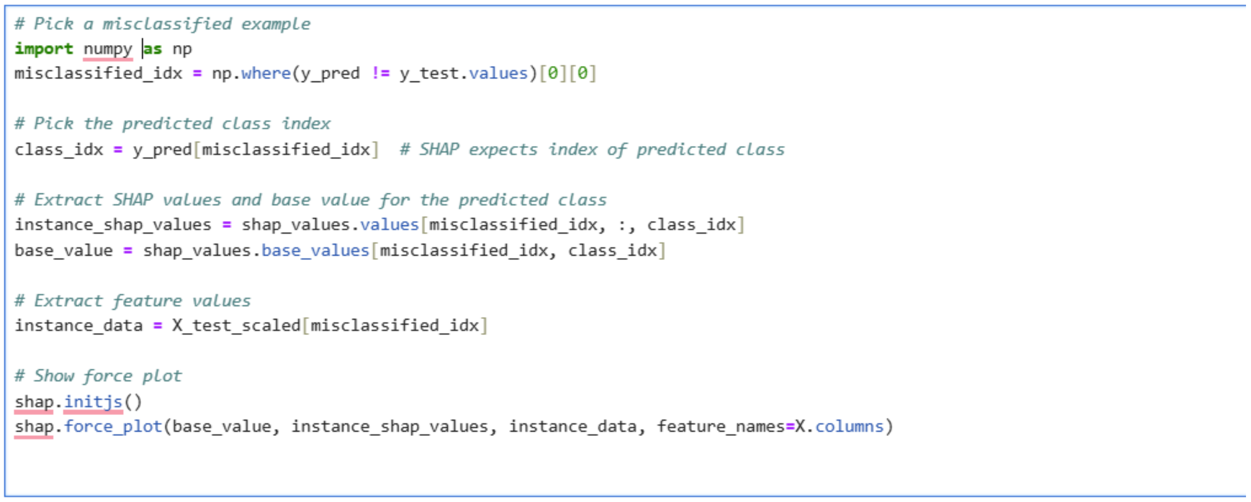


Figure 23SHAP Result



```python
# Pick a misclassified example
import numpy as np
misclassified_idx = np.where(y_pred != y_test.values)[0][0]

# Pick the predicted class index
class_idx = y_pred[misclassified_idx]  # SHAP expects index of predicted class

# Extract SHAP values and base value for the predicted class
instance_shap_values = shap_values.values[misclassified_idx, :, class_idx]
base_value = shap_values.base_values[misclassified_idx, class_idx]

# Extract feature values
instance_data = X_test_scaled[misclassified_idx]

# Show force plot
shap.initjs()
shap.force_plot(base_value, instance_shap_values, instance_data, feature_names=X.columns)
```

Figure 24Pick a misclassified example

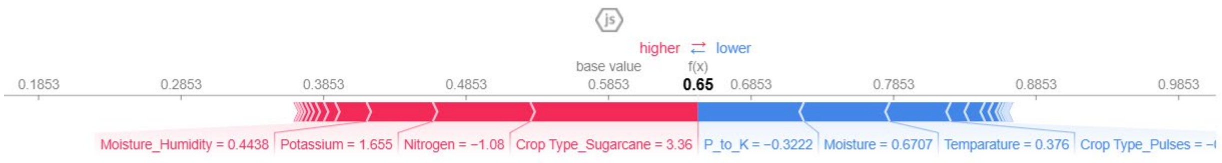


Figure 25misclassified example Result

CONCLUDING REMARKS

Throughout the project, we tested a variety of models and strategies to predict fertilizer types. SVM and kernel approximations proved too slow or ineffective for the dataset's scale. XGBoost outperformed all other models, especially in binary classification tasks. Feature selection and ensembling didn't significantly improve results. Multi-class classification remained a challenge due to similar classes and limited feature depth. SHAP analysis helped explain model behavior and reinforced the importance of numeric features like Moisture, Phosphorous, and Nitrogen. Ultimately, XGBoost with binary framing and basic feature engineering delivered the most reliable results.

Step	Model/Technique	Result
SVM (3 features)	Support Vector Machine	Low accuracy, very slow
SGDClassifier + RBF	Fast kernel approximation	Still poor accuracy (~15%)
XGBoost (Binary)	Full features	Best result (~56%)
Feature Selection	Top 10 features	Slight accuracy drop (54%)
Ensemble (Voting)	3 combined models	Worse than XGBoost
XGBoost (Multi-class)	All 7 fertilizers	Poor result (~17%)
XGBoost (4 classes)	Filtered classes	Slightly better (~32%)
SHAP	Explainability	Useful insights into feature behavior

Table 3Summary of Experiments and Results

personal reflection

I worked hard to solve this task and submitted a sample for the Kaggle competition. Although I couldn't make good predictions, I learned a great deal.

Future Work and Recommendations

Although this project explored a variety of machine learning approaches, several improvements could help boost performance and move toward a more accurate and real-world ready fertilizer prediction system:

1. **Feature Enrichment from Domain Knowledge**
   Incorporating expert knowledge from agronomy could dramatically improve prediction quality. For example:
   - Classifying fertilizers by nutrient type (e.g., nitrogen-rich vs. phosphorus-rich)
   - Mapping which fertilizers are recommended for specific soil-crop combinations
2. **Contextual Environmental Data**
   The current dataset lacks environmental factors such as:
   - Weather conditions (rainfall, temperature trends)
   - Soil pH, texture, organic content
   - Season and regional location
     Including such features could give the model more context to make accurate recommendations.
3. **Multi-Label or Ranking Models**
   Instead of forcing the model to choose **one** fertilizer, a ranking model could suggest **top 2 or 3 suitable options** with confidence scores. This better reflects real-world decision-making in agriculture.
4. **Improved Categorical Encoding**
   While one-hot encoding works, other methods like **target encoding**, **embedding layers** (in neural networks), or **entity embeddings** could better capture relationships among categories.
5. **Model Explainability in Deployment**
   Continue using SHAP in production to ensure transparency and trust in predictions. This is especially important for agricultural applications, where decisions impact cost, yield, and sustainability.
6. **Transfer Learning / Pretrained Models**
   Pretraining a model on a larger agronomic dataset (or simulation-based data) and fine-tuning on this one could improve generalization, especially for underrepresented fertilizer types.

BIBLIOGRAPHY

Chen, T. and Guestrin, C., 2016. *XGBoost: A scalable tree boosting system*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794). ACM.

https://doi.org/10.1145/2939672.2939785

doi = {10.1145/2939672.2939785}

James, G., Witten, D., Hastie, T. and Tibshirani, R., 2013. *An Introduction to Statistical Learning: With Applications in R*. Springer.

Kaggle, 2024. *Playground Series - Season 5, Episode 6*. [online] Available at: https://www.kaggle.com/competitions/playground-series-s5e6 [Accessed 7 Jun. 2025].

Kuhn, M. and Johnson, K., 2019. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press.

https://doi.org/10.1080/00031305.2020.1790217

Lundberg, S.M. and Lee, S.-I., 2017. *A Unified Approach to Interpreting Model Predictions*. In Advances in Neural Information Processing Systems (NeurIPS), 30.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011. *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, pp.2825–2830.

APPENDIX (if necessary)

Table of Figure

Table of Table