Project 4

Machine Learning Pipeline

Introduction to Data Science with Python

Computer Science Program

Kutaisi International University

Project Information

Total Weight:	6% of Final Grade
Task Breakdown:	3 Tasks × 2% each
Due Date:	End of Week 11 (Sunday, 23:59 GT)
Format:	Individual Assignment
Deliverable:	Jupyter Notebook (.ipynb) or Python Script (.py)

Contents

1 Project Overview

1.1 Introduction

Machine Learning is transforming how we solve complex problems by learning patterns from data. In this project, you will build a complete machine learning pipeline for regression—predicting house prices based on various features. You'll learn the full ML workflow: data preprocessing, feature engineering, model training, validation, and evaluation.

This project focuses on practical implementation using Scikit-learn, the industry-standard Python library for machine learning. You'll work with real-world messy data, make preprocessing decisions, train multiple models, and compare their performance using appropriate metrics.

Important: Data Files

You will be provided with a Python script (`generate_project4_data.py`) that generates a comprehensive house price dataset with 800 houses and 25+ features including location, size, condition, amenities, and neighborhood characteristics.
Download the data generator script from:

- LMS course materials folder
- Or instructor will provide via email/announcement

Run the script: `python generate_project4_data.py`
This will create: `house_prices.csv`

1.2 Learning Objectives

Upon completion of this project, you will be able to:

- **Data Preprocessing:** Handle missing values, encode categorical variables, scale features
- **Feature Engineering:** Create new features to improve model performance
- **Train-Test Split:** Properly divide data to prevent overfitting
- **Model Training:** Train regression models using Scikit-learn
- **Model Validation:** Use cross-validation to assess model performance
- **Performance Metrics:** Calculate and interpret R^2, MAE, MSE, RMSE
- **Model Comparison:** Compare multiple models and select the best one
- **Pipeline Creation:** Build end-to-end ML pipelines for reproducibility

1.3 Project Structure

Task	Focus Area	Weight
Task 1	Data Preprocessing & Feature Engineering	2%
Task 2	Model Training & Validation	2%
Task 3	Model Evaluation & Improvement	2%
Total		**6%**

Critical Deadline Information

Academic Integrity Policy:

- All work must be your own individual effort
- You may consult official documentation (Scikit-learn, Pandas) and course materials
- Copying code from online sources, AI tools, or other students will result in zero points
- If you reference any external resources for concepts, cite them in comments or markdown cells

2 Task Specifications

2.1 Task 1: Data Preprocessing & Feature Engineering (2%)

Objective

Prepare the raw data for machine learning by handling missing values, encoding categorical variables, creating new features, and splitting the dataset.

Requirements

Part A: Data Loading & Exploration (20%)

1. **Load and examine the dataset:**
   - Load `house_prices.csv` using Pandas
   - Display basic information: shape, columns, data types
   - Show statistical summary with `.describe()`
   - Identify missing values in each column
   - Check for duplicate rows
2. **Exploratory analysis:**
   - Display distribution of target variable (Price)
   - Identify numerical vs categorical features
   - Check for outliers in Price (values > 3 std from mean)
   - Examine correlation between features and target

Part B: Data Preprocessing (40%)

1. **Handle Missing Values:**
   - For `Years_Since_Renovation`: Fill with house age (if not renovated)
   - For `School_Rating`: Fill with median
   - For `Crime_Rate`: Fill with median
   - For `Energy_Rating`: Fill with mode (most common)
   - Document your strategy for each column
   - Verify no missing values remain
2. **Handle Outliers:**
   - Identify price outliers (>3 standard deviations)
   - Option 1: Remove outliers OR
   - Option 2: Cap at 3 standard deviations
   - Document which approach you chose and why
3. **Encode Categorical Variables:**
   - `Neighborhood`: Use One-Hot Encoding or Label Encoding
   - `Heating_Type`: Use One-Hot Encoding

- `Energy_Rating`: Map to numerical (A=7, B=6, ..., G=1) OR One-Hot
- Explain your encoding choices
- Use `pd.get_dummies()` or `sklearn.preprocessing.LabelEncoder/OneHotEncoder`

4. **Feature Scaling:**

- Separate features (X) and target (y)
- Identify features that need scaling
- Apply StandardScaler or MinMaxScaler to numerical features
- Do NOT scale the target variable
- Store the scaler for later use

Part C: Feature Engineering (20%)
Create at least 3 new features that might improve predictions:

1. **Suggested new features:**

- `Price_Per_SqFt`: Price divided by Square Footage
- `Total_Rooms`: Bedrooms + Bathrooms
- `Is_New`: 1 if house age < 5 years, else 0
- `Luxury_Score`: Has_Pool + Has_AC + (Garage_Spaces > 1)
- `Location_Score`: Combination of School_Rating and Crime_Rate
- Any other creative feature you think is relevant

2. Document why you created each feature

Part D: Train-Test Split (20%)

1. **Split the data:**

- Use `train_test_split` from sklearn
- 80% training, 20% testing
- Set `random_state=42` for reproducibility
- Verify split sizes

2. **Final preprocessing check:**

- Print shapes of X_train, X_test, y_train, y_test
- Verify no data leakage (train and test are separate)
- Ensure all features are numerical
- Confirm no missing values in train/test sets

Evaluation Criteria

- Correct data loading and exploration (15%)
- Appropriate missing value handling (20%)
- Proper categorical encoding (20%)
- Feature scaling implementation (15%)

- Creative and useful feature engineering (15%)
- Correct train-test split (10%)
- Code documentation (5%)

Essential Scikit-learn Imports

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder,
    OneHotEncoder
from sklearn.impute import SimpleImputer
import pandas as pd
import numpy as np
```

2.2 Task 2: Model Training & Validation (2%)

Objective

Train multiple regression models, validate them using cross-validation, and compare their performance.

Requirements

Part A: Baseline Model (25%)

1. **Train Linear Regression:**
   - Import `LinearRegression` from sklearn
   - Create and fit the model on training data
   - Make predictions on both train and test sets
   - Calculate R^2 score for both train and test
   - Calculate MAE and RMSE for test set
2. **Interpret baseline results:**
   - What is the R^2 score? (good if > 0.7)
   - Is there overfitting? (train R^2 » test R^2)
   - What is the average prediction error (MAE)?

Part B: Additional Models (40%)

Train and evaluate at least 3 more regression models:

1. **Ridge Regression:**
   - Import `Ridge` from sklearn.linear_model
   - Train with default parameters
   - Try different alpha values: [0.1, 1, 10, 100]
   - Select best alpha using cross-validation
   - Evaluate on test set
2. **Lasso Regression:**
   - Import `Lasso` from sklearn.linear_model
   - Train with default parameters
   - Try different alpha values
   - Check which features have zero coefficients (feature selection)
   - Evaluate on test set
3. **Decision Tree Regressor:**
   - Import `DecisionTreeRegressor` from sklearn.tree
   - Train with `max_depth=5` (prevent overfitting)
   - Evaluate on test set
   - Compare with unrestricted tree (no max_depth)

4. **Random Forest Regressor (Bonus):**
   - Import `RandomForestRegressor` from sklearn.ensemble
   - Train with `n_estimators=100`
   - Evaluate on test set
   - Extract feature importance

Part C: Cross-Validation (35%)
Validate models using k-fold cross-validation:

1. **Implement cross-validation:**
   - Use `cross_val_score` from sklearn.model_selection
   - Set `cv=5` (5-fold cross-validation)
   - Calculate mean and std of CV scores for each model
   - Use scoring='r2' or 'neg_mean_squared_error'

2. **Compare models:**
   - Create a comparison table or DataFrame showing:
     - Model name
     - Train R^2
     - Test R^2
     - CV Mean R^2
     - CV Std R^2
     - MAE
     - RMSE
   - Identify which model performs best on test data
   - Check for overfitting (large gap between train and test)

Evaluation Criteria

- Correct baseline model implementation (25%)
- Training of multiple models (30%)
- Proper cross-validation usage (25%)
- Comprehensive model comparison (15%)
- Code organization and documentation (5%)

Performance Metrics Formulas

- **R^2 Score:** $1 - \frac{\sum(y-\hat{y})^2}{\sum(y-\bar{y})^2}$ (1 is perfect, <0 is worse than mean)
- **MAE:** $\frac{1}{n}\sum|y-\hat{y}|$ (average absolute error in same units as target)
- **MSE:** $\frac{1}{n}\sum(y-\hat{y})^2$ (penalizes large errors more)
- **RMSE:** $\sqrt{MSE}$ (in same units as target, easier to interpret)

2.3 Task 3: Model Evaluation & Improvement (2%)

Objective

Perform comprehensive evaluation of the best model, analyze errors, and attempt to improve performance through hyperparameter tuning.

Requirements

Part A: Model Selection & Detailed Evaluation (30%)

1. **Select best model:**
   - Based on Task 2 results, select the best performing model
   - Retrain it on the full training set
   - Make final predictions on test set

2. **Residual Analysis:**
   - Calculate residuals: $residuals = y_{test} - \hat{y}_{test}$
   - Create a scatter plot: Predicted vs Actual prices
   - Create a residual plot: Predicted vs Residuals
   - Check if residuals are normally distributed (histogram)
   - Identify houses with largest prediction errors

3. **Error Analysis:**
   - Calculate percentage errors: $\frac{|y-\hat{y}|}{y} \times 100$
   - Find average percentage error
   - Identify which types of houses have highest errors
   - Analyze if errors are related to specific features

Part B: Hyperparameter Tuning (40%)

Improve model performance through systematic hyperparameter optimization:

1. **Grid Search (Option 1):**
   - Use `GridSearchCV` from sklearn.model_selection
   - Define parameter grid for your best model:
     - Ridge: `{'alpha': [0.01, 0.1, 1, 10, 100]}`
     - Decision Tree: `{'max_depth': [3,5,7,10], 'min_samples_split': [2,5,10]}`
     - Random Forest: `{'n_estimators': [50,100,200], 'max_depth': [5,10,None]}`
   - Set `cv=5` for cross-validation
   - Fit GridSearchCV on training data
   - Extract best parameters and best score

2. **Random Search (Option 2 - Bonus):**
   - Use `RandomizedSearchCV`
   - Define parameter distributions
   - Set `n_iter=20` for 20 random combinations

- Compare with GridSearch results

3. **Evaluate tuned model:**
   - Train best model with optimal parameters
   - Evaluate on test set
   - Compare performance with untuned model
   - Calculate improvement in R^2 and RMSE

Part C: Feature Importance Analysis (30%)
Understand which features drive predictions:

1. **Extract feature importance:**
   - For Linear/Ridge/Lasso: Use `model.coef_` to get coefficients
   - For Tree-based models: Use `model.feature_importances_`
   - Create a DataFrame with feature names and importance scores
   - Sort by importance (descending)

2. **Visualize importance:**
   - Create horizontal bar chart of top 10 features
   - Include clear labels and title
   - Interpret: Which features matter most for price?

3. **Insights:**
   - Which feature has strongest impact on price?
   - Are engineered features useful?
   - Any surprising findings?
   - What would you recommend to homeowners wanting to increase value?

Part D: Final Report (Bonus - up to +5%)

Write a brief analysis (200+ words) summarizing:

1. Best model and its performance metrics
2. Key features driving house prices
3. Model limitations and potential improvements
4. Business recommendations based on findings

Evaluation Criteria

- Comprehensive model evaluation (25%)
- Correct hyperparameter tuning (30%)
- Feature importance analysis (25%)
- Visualization quality (10%)

- Interpretation and insights (10%)

Excellence Indicators: To achieve >95%, demonstrate:

- Systematic comparison of multiple models
- Effective hyperparameter tuning with clear improvement
- Insightful residual and error analysis
- Clear interpretation of feature importance
- Professional visualizations
- Evidence-based recommendations

3 Submission Guidelines

3.1 Deliverable Format

Submit **ONE** file in your preferred format:

- Jupyter Notebook: `Project4_FirstName_LastName.ipynb`, OR
- Python Script: `Project4_FirstName_LastName.py`

You may use any IDE (PyCharm, VS Code, Jupyter, Spyder, etc.)

Required File Structure

For Jupyter Notebook (.ipynb):

1. **Title Cell (Markdown):**
   - Project title and number
   - Your full name and Student ID
   - Submission date
   - Honor code: "I certify this work is my own"
2. **For Each Task:**
   - Clear header with task number and title
   - Brief markdown explanation before code sections
   - Well-commented code cells
   - All cells executed with visible outputs
   - Interpretation of results after each analysis
3. **Include Generated Files:**
   - Original CSV file (from data generator)
   - Do NOT include the data generator script itself

For Python Script (.py):

1. **Header Comment Block:**

```
"""
Project 4: Machine Learning Pipeline
Name: [Your Full Name]
Student ID: [Your ID]
Date: [Submission Date]
Honor Code: I certify this work is my own
"""
```

2. **For Each Task:**
   - Clear section comments
   - Code for preprocessing, training, and evaluation
   - Print statements showing key results
   - Comments explaining decisions and interpretations

3.2 Code Quality Standards

Mandatory Standards:

- **Imports:** All at the top, organized by library
- **Reproducibility:** Set random_state=42 where applicable
- **Comments:** Explain preprocessing decisions and model choices
- **Validation:** Check data shapes and types after transformations
- **Error Handling:** Code should not crash with errors
- **Results:** Print key metrics clearly (formatted output)
- **Documentation:** Document why you chose specific approaches

3.3 Machine Learning Best Practices

- **Never fit on test data:** Only use test for final evaluation
- **Scale after splitting:** Fit scaler on train, transform both train/test
- **Use random_state:** Ensure reproducible results
- **Validate assumptions:** Check for overfitting, examine residuals
- **Compare multiple models:** Don't just use one model
- **Interpret results:** Don't just report numbers, explain what they mean
- **Document decisions:** Explain why you chose specific methods

3.4 Submission Methods

Choose **ONE** submission method:

Option 1: Direct LMS Upload (Recommended)

1. Ensure all code runs without errors
2. For Jupyter: verify all cells executed with outputs visible
3. Gather all files (notebook/script + CSV file)
4. Create ZIP file: `Project4_FirstName_LastName.zip`
5. Upload ZIP to LMS before deadline
6. Verify upload was successful

Option 2: GitHub Repository Submission

1. Create a **public** GitHub repository: `KIU-DS-Project4-FirstName-LastName`
2. Repository structure:

```
Project4/
 Project4_FirstName_LastName.ipynb (or .py)
 data/
    house_prices.csv
 README.md
 requirements.txt
```

3. `requirements.txt`:

```
pandas==2.0.3
numpy==1.24.3
scikit-learn==1.3.0
matplotlib==3.7.1
seaborn==0.12.2
```

4. Submit repository URL via LMS **before deadline**
5. **CRITICAL:** Do not commit after deadline

Critical Deadline Information

Deadline Enforcement:

- **Due Date:** End of Week 11 - Sunday, 23:59 Georgian Time
- **LMS Submissions:** Timestamp automatically recorded
- **GitHub Submissions:** Last commit timestamp verified
- **Any commits after deadline = Automatic 0 points**
- **Technical Issues:** Submit at least 2 hours early
- **No Extensions:** Start early!

3.5 Pre-Submission Checklist

Before submitting, verify:

- ☐ All three tasks completed
- ☐ Code runs from start to finish without errors
- ☐ All model results printed/displayed
- ☐ Train-test split properly implemented
- ☐ Multiple models trained and compared
- ☐ Cross-validation performed
- ☐ Hyperparameter tuning attempted
- ☐ Feature importance analyzed
- ☐ CSV file included
- ☐ Filename follows convention
- ☐ Header includes name, ID, honor code
- ☐ Random states set for reproducibility
- ☐ (If ZIP) all files included
- ☐ (If GitHub) README and requirements.txt present
- ☐ Submitting well before deadline

4 Grading Rubric

4.1 Grade Distribution

Component	Points	% of Final Grade
Task 1: Preprocessing & Feature Engineering	2.0	2%
Task 2: Model Training & Validation	2.0	2%
Task 3: Evaluation & Improvement	2.0	2%
Project Total	**6.0**	**6%**

Table 1: Project Point Distribution

4.2 Detailed Rubric

Criterion	Excellent (90-100%)	Good (70-89%)	Needs Work (<70%)
Preprocessing	Proper handling of all issues, well-documented	Minor issues, adequate documentation	Significant errors or poor choices
Model Training	Multiple models correctly trained	Some models work, minor errors	Major errors or incomplete
Validation	Proper CV and evaluation	CV present but limited	Missing or incorrect validation
Interpretation	Deep insights, quantified results	Basic interpretation	Superficial or missing analysis
Code Quality	Clean, organized, reproducible	Functional but could improve	Messy or hard to follow

Table 2: Quality Assessment Rubric

4.3 Deductions

Points will be deducted for:

- **-15%** Code doesn't run or produces errors
- **-10%** Missing CSV file
- **-15%** Data leakage (fitting on test data, scaling before split)
- **-10%** Missing train-test split
- **-10%** Training only one model (multiple required)
- **-10%** No cross-validation performed
- **-10%** Results not printed/displayed
- **-5%** No random_state set (not reproducible)
- **-100%** Late submission
- **-100%** Academic integrity violations

4.4 Bonus Opportunities (Up to +10%)

- **+5%:** Exceptional feature engineering with measurable improvement
- **+3%:** Advanced techniques (ensemble methods, stacking)
- **+2%:** Outstanding analysis and business insights

Note: Bonus points apply to this project only, maximum score capped at 6%

5 Resources & Support

5.1 Official Documentation

- **Scikit-learn Documentation:** https://scikit-learn.org/stable/
- **User Guide:** https://scikit-learn.org/stable/user_guide.html
- **API Reference:** https://scikit-learn.org/stable/modules/classes.html
- **Examples Gallery:** https://scikit-learn.org/stable/auto_examples/

5.2 Recommended Study Materials

- Course lecture slides (Weeks 8-11)
- Lab notebooks on ML
- Scikit-learn tutorials: https://scikit-learn.org/stable/tutorial/
- Introduction to Machine Learning with Python (course textbook)

5.3 Key Scikit-learn Modules

- `sklearn.preprocessing`: StandardScaler, LabelEncoder, OneHotEncoder
- `sklearn.model_selection`: train_test_split, cross_val_score, GridSearchCV
- `sklearn.linear_model`: LinearRegression, Ridge, Lasso
- `sklearn.tree`: DecisionTreeRegressor
- `sklearn.ensemble`: RandomForestRegressor
- `sklearn.metrics`: r2_score, mean_absolute_error, mean_squared_error

5.4 Getting Help

1. **Review Course Materials:** Lectures and labs on ML
2. **Check Documentation:** Scikit-learn docs are excellent
3. **Email Instructor:** Nika Gagua at `Nika.Gagua@kiu.edu.ge`
4. **Study Groups:** Discuss approaches (write your own code)

Success Strategies

- **Start Early:** ML projects take time to debug
- **Run Data Generator First:** Get familiar with the data
- **Work Incrementally:** Test each step before moving on
- **Check Shapes Often:** Verify data dimensions after each transformation
- **Start Simple:** Get linear regression working first
- **Compare Models:** Don't commit to one model too early
- **Validate Everything:** Use CV, check for overfitting
- **Interpret Results:** Understand what metrics mean
- **Document Decisions:** Explain your choices
- **Save Progress:** ML experiments can take time

5.5 Common Pitfalls to Avoid

- **Data Leakage:** Fitting scaler/encoder on full data before split
- **Not splitting data:** Training and testing on same data
- **Scaling target variable:** Don't scale y
- **Forgetting to inverse transform:** If you scale y, inverse predictions
- **Training on test data:** Never use test for anything except final evaluation
- **Not handling missing values:** Models will crash
- **Not encoding categoricals:** Models need numerical input
- **Ignoring overfitting:** Check train vs test performance
- **Not using random_state:** Results won't be reproducible
- **Starting too late:** ML requires experimentation

6 Frequently Asked Questions

6.1 General Questions

Q: How long should this project take?
A: Plan for 10-12 hours. Task 1: 3-4 hours, Task 2: 3-4 hours, Task 3: 4-5 hours. ML requires experimentation.

Q: Can I use other ML libraries like TensorFlow?
A: Focus on Scikit-learn as required. Other libraries are beyond scope.

Q: What's a good R^2 score?
A: For this dataset, aim for >0.75. Above 0.85 is excellent. Below 0.6 needs improvement.

Q: Where do I get the data generator script?
A: Download from LMS course materials or contact instructor.

6.2 Technical Questions

Q: Should I scale the target variable (Price)?
A: Generally no. Scale only features (X), not target (y).

Q: When do I split the data?
A: After loading and before any fitting (scaling, encoding with fit_transform).

Q: How do I handle categorical variables?
A: Use pd.get_dummies() or OneHotEncoder for nominal, LabelEncoder for ordinal.

Q: What if my model has terrible performance?
A: Check: data leakage, missing value handling, feature scaling, model choice. Debug systematically.

Q: How many models should I try?
A: Minimum 4: Linear Regression + 3 others. More is better for comparison.

Q: What's the difference between Ridge and Lasso?
A: Ridge: L2 regularization, shrinks coefficients. Lasso: L1 regularization, can zero out coefficients (feature selection).

6.3 Preprocessing Questions

Q: Should I remove outliers?
A: Your choice. Document your decision. Removing may improve metrics but lose real data.

Q: How do I fill missing values?
A: Use median for skewed numerical, mean for normal numerical, mode for categorical.

Q: What if a feature has 50+ unique categories?
A: One-hot encoding creates too many features. Try label encoding or group rare categories.

6.4 Evaluation Questions

Q: My train R^2 is 0.95 but test is 0.6. What's wrong?
A: Overfitting. Use regularization, reduce model complexity, or get more data.

Q: Which metric is most important?
A: For regression: R^2 shows variance explained, RMSE shows average error in original units. Report both.

Q: Can I use accuracy for regression?
A: No! Accuracy is for classification. Use R^2, MAE, MSE, RMSE for regression.

Best of Luck!

Machine Learning is powerful but requires careful implementation.
Focus on understanding the pipeline, validate your assumptions,
and don't forget: garbage in, garbage out—data quality matters!

May your models generalize well!

- Your Data Science Instructors