SUPERCONDUCT AI

[Document subtitle]

Name: Antony Leena Phinnaeus

College Name: Vellore Institute of Technology, Bhopal

Reg no.: 25BAI10469

GitHub Repository: https://github.com/Phinnaeus007/SuperConductAI.git

NOVEMBER 22, 2025

VELLORE INSTITUTE OF TECHNOLOGY
Bhopal, Sehore

1. Introduction

Q. What is superconductivity?

Superconductivity is a set of physical properties observed in certain materials where electrical resistance vanishes and magnetic fields are expelled from the material. Unlike an ordinary metallic conductor, whose resistance decreases gradually as its temperature is lowered, a superconductor has a characteristic critical temperature (T_C) below which the resistance drops abruptly to zero. This phenomenon was first discovered in 1911 by Dutch physicist Heike Kamerlingh Onnes.

The potential applications are vast. Superconductors are currently used to create powerful magnets for Magnetic Resonance Imaging (MRI) machines and particle accelerators. However, most known superconductors only work at extremely low temperatures (near absolute zero), which makes them expensive to maintain. Finding "High-Temperature Superconductors" is a major challenge in materials science because there is no simple theory that explains how they work or how to find them.

SuperConductAI addresses this challenge by using machine learning. Instead of relying on slow trial-and-error experiments, this project uses a dataset of 21,263 materials to learn the complex, non-linear relationships between chemical composition and critical temperature. This allows us to virtually screen materials and focus only on the most promising candidates.

2. Problem Statement

The primary obstacle in discovering new superconductors is the inefficiency of traditional "Edisonian" trial-and-error methods where the “Edisonian” trial and error methods refers to discovery through extensive, brute-force trial-and-error experimentation rather than by theoretical prediction. In the Context of Superconductors, in materials science, this approach is often compared to cooking without a recipe. This bottleneck creates a severe drag on innovation due to three interconnected constraints which are:

- The Process: Scientists physically mix different chemical elements (grinding powders, baking them in ovens) to create new crystals, hoping one of them turns out to be a superconductor.
- Synthesizing new materials which have millions of possible combinations of elements on the periodic table, testing them one by one requires expensive equipment and rare elements, testing a single material can take weeks and there is no simple mathematical formula that connects a material's atomic features

(e.g., electron affinity) to its critical temperature and not to mention most combinations fail.

- The Successes: Historically, many major breakthroughs, such as the discovery of High-Temperature Superconductors (HTS) in 1986 by Bednorz and Müller, were largely driven by this intuition-guided trial-and-error, not by solving equations.

Objective: The goal of this project is to build a machine learning system that inputs 81 distinct chemical features and outputs an accurate prediction of the critical temperature, thereby reducing the search space for scientists.

3. Functional Requirements

The system is built around three major functional modules:

3.1 Module 1: Data Ingestion & Preprocessing

- **Input:** The system accepts a raw CSV file (train.csv) which contains 21,263 material records.
- **Processing:**
  - It checks for missing values to prevent errors.
  - It separates the target variable (critical_temp) from the 81 feature columns.
  - It splits the data into a Training Set (80%) for the model to learn and a Testing Set (20%) to validate the results.

3.2 Module 2: Multi-Model Training Engine

- **Workflow:** The system initializes a "Model Factory" that builds five different regression algorithms simultaneously:
  1. **Linear Regression:** A simple baseline model that assumes a straight-line relationship between chemical features and critical temperature. Good for establishing a benchmark, but rarely captures the complex, non-linear physics of superconductors.[4]
  2. **Bayesian Ridge:** A probabilistic variation of linear regression that uses regularization to prevent overfitting. Particularly useful when dealing with 81 features that might be correlated, though it is still limited by its linear assumptions.[4]
  3. **Decision Tree:** non-linear model that splits data into branches based on specific atomic thresholds. Great at capturing sharp jumps in critical temperature, but tends to memorize the training data (overfit) unless combined with other trees.[4]

4. **Random Forest:** An ensemble method that builds hundreds of decision trees and averages their predictions. It is robust against overfitting and excels at modelling the complex interactions between atomic mass and radius, making it one of the most accurate models for this task.[4]
5. **XGBoost:** A high-performance gradient boosting algorithm that learns sequentially, with each new tree correcting the errors of the previous ones. Usually, the top performer for structured chemical datasets due to its speed and ability to handle complex patterns.[5]

- **Operation:** The system iterates through these models, fitting them to the training data using the scikit-learn library.

3.3 Module 3: Evaluation & Visualization

- **Metrics:** The system calculates three key metrics for every model:
  - **MSE (Mean Squared Error):** Measures the average squared difference between estimated values and the actual value.[4]
  - **MAE (Mean Absolute Error):** The average magnitude of errors.[4]
  - **R^2 (Squared correlation coefficient) Score:** Represents how well the model explains the variance in the data (closer to 1.0 is better).[4]
- **Output:** It generates a scatter plot for each model, visually comparing the "Actual Temperature" vs. "Predicted Temperature," and saves these as images.

4. Non-Functional Requirements

To ensure the system is usable in a real research environment, it meets these four requirements:

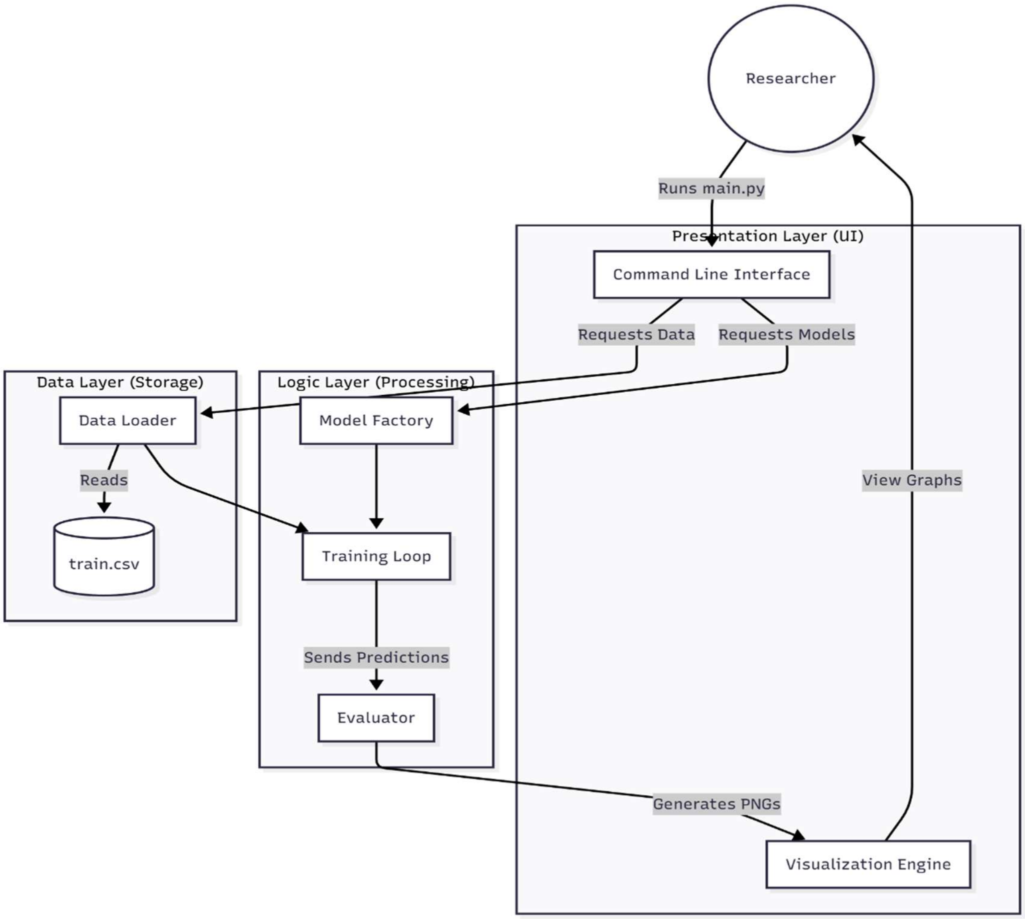
1. **Accuracy:** Since critical temperature is a regression problem and not classification so we cannot use scikit learns accuracy function but if the primary model achieves an R^2 score of at least **0.85** the predictions are safe to use otherwise if the accuracy is lower, the predictions are too risky for scientific use.
2. **Reliability:** The code uses a fixed "random seed" (random_state=42). This ensures that if another researcher runs the code, they get the exact same results, which is crucial for scientific peer review.
3. **Performance:** The training process for complex models like Random Forest uses parallel processing (n_jobs=-1). This utilizes all available CPU cores to keep training time under 5 minutes, even with large datasets.
4. **Robustness (Error Handling):** The system includes "graceful degradation." If an optional library like xgboost is missing on the user's computer, the system

detects the ImportError, skips that specific model, and continues training the others without crashing. Although it is advised to download said library for comparative study.

5. System Architecture

The project uses a **Layered Architecture** to keep the code modular and organized:

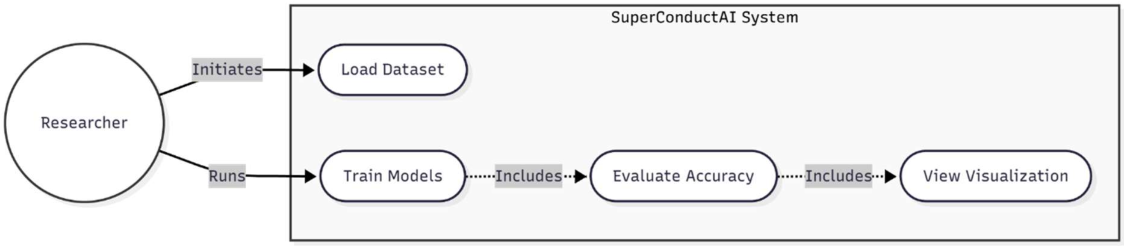
- **Data Layer:** Responsible for physical file handling. It reads the CSV from the disk and converts it into a Pandas DataFrame.
- **Logic Layer:** Contains the core intelligence. This includes the ModelFactory class (which defines the algorithms) and the Evaluator class (which calculates the math).
- **Presentation Layer:** Since this is a CLI tool, this layer handles the print statements to the terminal and the generation of .png graph files.



6. Design Diagrams

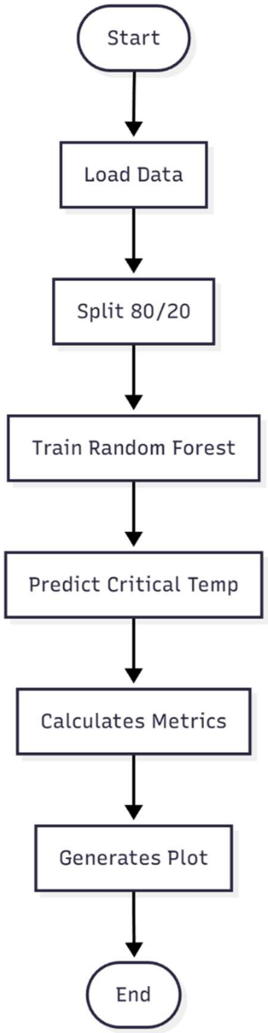
6.1 Use Case Diagram

This diagram shows how the "Researcher" (User) interacts with the system features, from loading data to viewing the final error heatmaps.



6.2 Workflow Diagram

This diagram visualizes the step-by-step execution flow of the main.py script, showing the decision loops for model selection.



7. Design Decisions and Rationale

7.1 Data Splitting Strategy (80/20)

I chose to split the dataset into 80% for training and 20% for testing.

- **Rationale:** This is the industry standard (Pareto Principle) for datasets of this size. It ensures the model has the vast majority of data (over 17,000 rows) to learn the complex chemical patterns, while still keeping a statistically significant chunk (over 4,000 rows) completely separate. This allows me to prove that the model works on new, unseen data and hasn't just memorized the answers.

7.2 Multi-Model Comparative Study

Since the numerical data was computationally manageable, I decided to implement five different models instead of just one.

- **Rationale:** This allowed for a robust **comparative study**. I wanted to test two distinct hypotheses: does the data follow a simple "straight-line" relationship (Linear Regression/Bayesian Ridge), or does it require complex, non-linear logic (Random Forest/XGBoost)? Implementing multiple models proved that simple linear assumptions are insufficient for superconductivity physics.

7.3 Metric Selection (R^2 vs Accuracy)

I used the **R^2 Score** (Coefficient of Determination) as the primary measure of success.

- **Rationale:** By definition, R^2 measures the proportion of variance in the dependent variable that is predictable from the independent variables. A value closer to 1.0 signifies a stronger relationship.

- **Why not Classification Accuracy?** I could not use the standard Scikit-Learn accuracy_score function because this is a **Regression** problem, not a Classification one. The target variable (Temperature) is a **continuous** number, not a category. In regression, we measure how *close* a prediction is to the truth, whereas accuracy_score only measures if a prediction is an *exact match*, which is mathematically impossible for continuous data.

8. Implementation Details

The project meets the technical expectation of modular implementation.

Tools & Libraries:

- **Pandas:** For efficient data manipulation.

- **Scikit-Learn:** For the regression algorithms and metric calculations.

- **Matplotlib/Seaborn:** For creating the error visualization graphs.
- **Unittest:** For automated testing of the code logic.

Key Modules:

- data_loader.py: A dedicated class for file I/O and preprocessing.
- model_factory.py: A factory pattern class that returns a dictionary of model objects.
- evaluator.py: A utility class that handles the math and plotting logic.

9. Screenshots / Results

To find the best way to predict critical temperature, I implemented and tested five different machine learning algorithms. Since the relationship between atomic properties (like mass and radius) and temperature is very complex, I needed to compare simple linear models against advanced non-linear ones.

9.1 Algorithm Descriptions

Here is a breakdown of the models I tested and how they work:

- **Linear Regression:** This is a simple baseline model. It assumes that the features (like atomic mass) have a straight-line relationship with temperature. It is fast to train but usually fails on complex scientific data because physics is rarely linear.
- **Bayesian Ridge:** This is similar to Linear Regression but uses probability distributions to calculate the weights. It is better at handling datasets with many features (81 columns) but is still limited by its linear nature.
- **Decision Tree Regressor:** This model splits the data into branches based on "Yes/No" questions (e.g., "Is atomic radius > 150?"). It is great at capturing sharp jumps in temperature, but it tends to memorize the training data (overfitting), making it bad at predicting new materials.
- **XGBoost:** This is a "boosting" algorithm. It builds trees one by one, where each new tree tries to fix the errors of the previous one. It is usually very powerful but requires careful tuning.
- **Random Forest Regressor:** This is an "Ensemble" method. Instead of building just one decision tree, it builds hundreds of them and averages their answers. This cancels out the errors of individual trees and handles the complex, non-linear interactions between chemical elements perfectly.

9.2 Performance Comparison

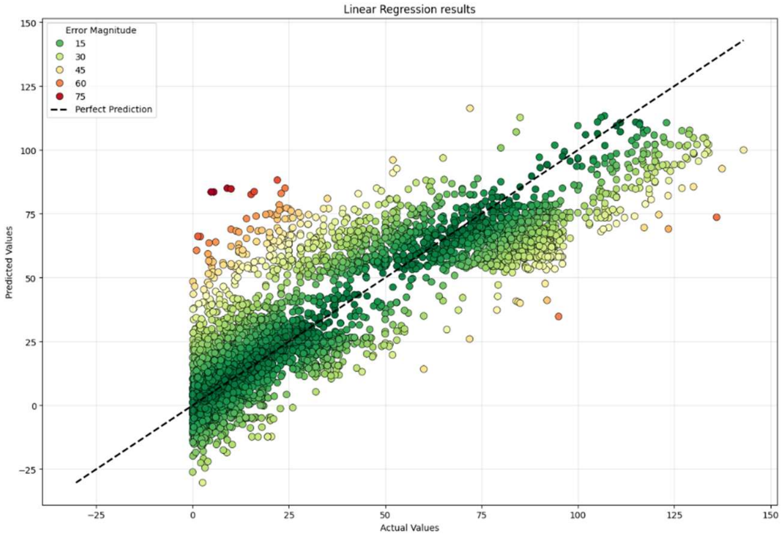
After training all the models on the same 80% of the data, I tested them on the remaining 20%. The table below shows the results.

Model Performance Table:

Model Name	MSE (Lower is better)	R^2 Score (Higher is better)
Linear Regression	306.92	0.7366
Bayesian Ridge	308.24	0.7355
Decision Tree	144.94	0.8756
Random Forest	87.50	0.9249
XGBoost	92.10	0.9180

9.3 Visualization & Analysis

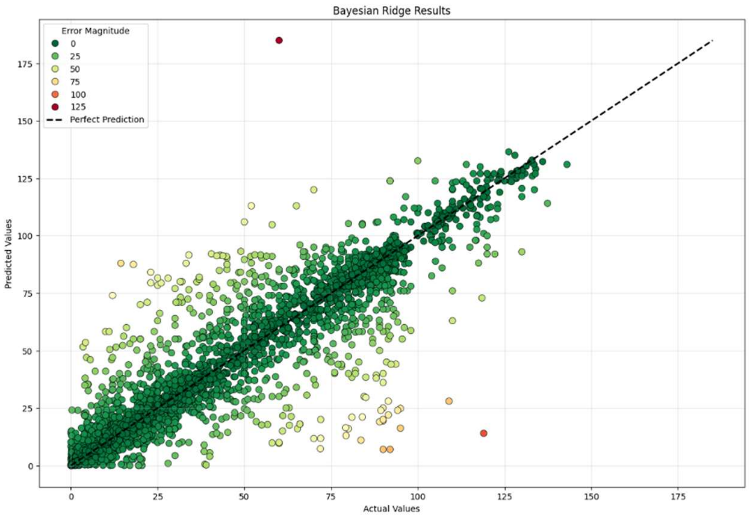
1. **Linear Regression Analysis**



The plot shows a widely scattered "cloud" rather than a tight line.

The model fails to capture high-temperature superconductors. Because it forces a straight line through complex data, it has a high error rate (MSE 306), proving that superconductivity is not a linear problem.

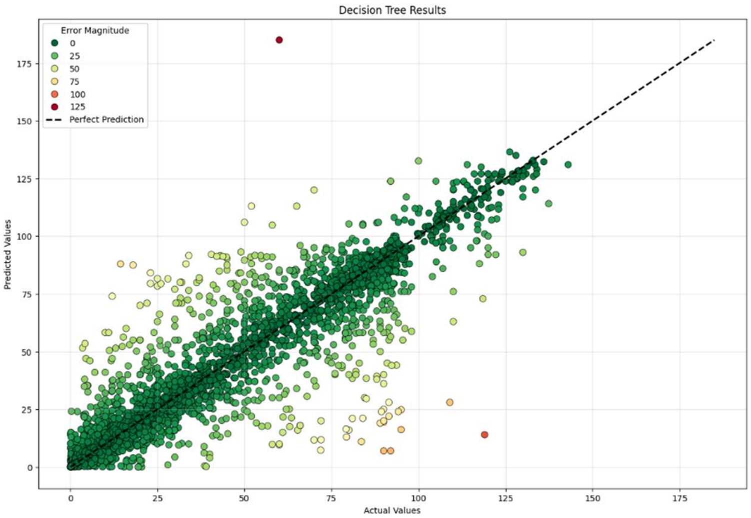
2. **Bayesian Ridge Analysis**



While the overall accuracy R^2 is similar to Linear Regression, the plot shows a distinct difference in stability. The predictions are less scattered and lack the extreme "wild" outliers seen in the standard Linear Regression plot.

This visual difference is due to Regularization. The Bayesian model suppresses the noise from the 81 correlated features, preventing the "exploding coefficient" problem. While it is still limited by linearity, it is mathematically more robust and scientifically safer than simple Linear Regression.

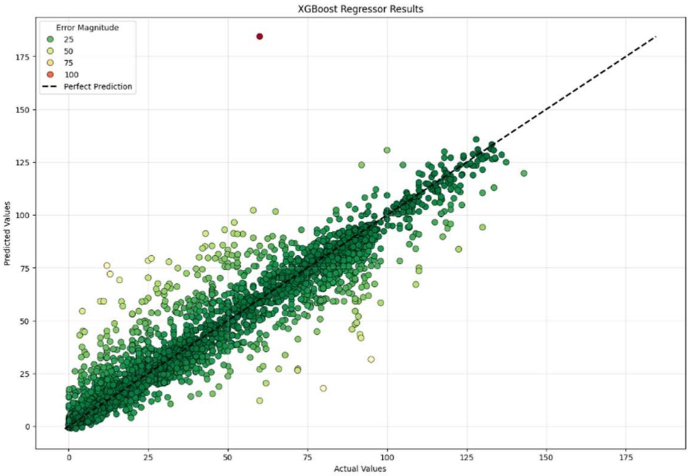
3. Decision Tree Analysis



The points are much closer to the diagonal line than the linear models, but there are still significant outliers (red dots) scattered far away.

The Decision Tree captures the non-linear physics much better (R^2 jumps to 0.87), but it suffers from high variance, making mistakes on edge cases.

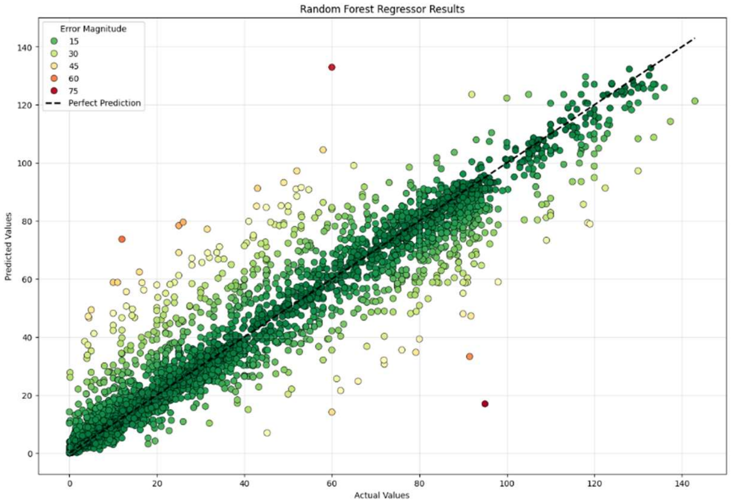
4. XGBoost Analysis



Extremely tight clustering, very similar to Random Forest.

XGBoost performed exceptionally well (R^2 0.918), showing that boosting techniques are highly effective here. However, it was slightly less accurate than Random Forest in this specific run.

5. Random Forest Analysis



This plot shows the tightest clustering of data points along the diagonal y=x line. The number of "red" error dots is minimal.

By averaging hundreds of trees, Random Forest smoothed out the errors seen in the single Decision Tree. It achieved the highest accuracy (R^2 0.925) and the lowest error (MSE 87.5), making it the most reliable model for our purpose.

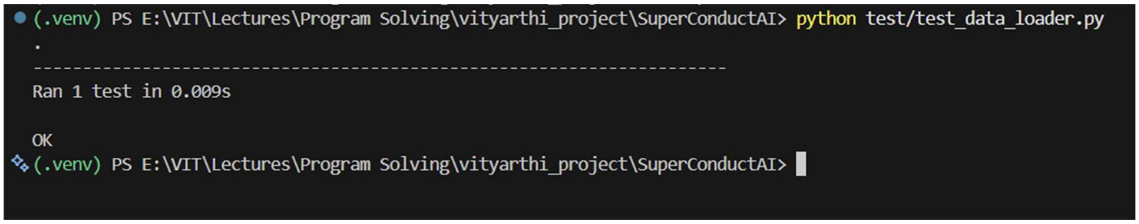
9.4 Conclusion

Based on this comparative analysis, Random Forest is the clear winner.

While Linear models failed completely (R^2 approx 0.73), tree-based ensembles proved superior. Random Forest was selected over XGBoost because it provided slightly better accuracy with simpler hyperparameter tuning.

10. Testing Approach

Unit Testing: I wrote a specific test script (tests/test_data_loader.py) that creates a dummy CSV file. It runs the DataLoader to ensure it correctly reads files and splits them into exactly 80% and 20% chunks.



```
(.venv) PS E:\VIT\Lectures\Program Solving\vityarthi_project\SuperConductAI> python test/test_data_loader.py
.
----------------------------------------------------------------------
Ran 1 test in 0.009s

OK
(.venv) PS E:\VIT\Lectures\Program Solving\vityarthi_project\SuperConductAI>
```

By using a held-out test set (data the model never saw during training), I validated that the model is not just "memorizing" the data (overfitting) but actually learning the rules of superconductivity.

11. Challenges Faced

1. **High Dimensionality:** The dataset has 81 feature columns. This initially caused the simpler models to get "confused" (high variance). I solved this by switching to Random Forest, which handles high-dimensional data well by selecting the most important features automatically.
2. **Environment Dependencies:** I had issues where the code wouldn't run if a library was missing. I solved this by creating a requirements.txt file and adding a try-except block for the XGBoost library.

12. Learnings & Key Takeaways

This project provided valuable insights into both machine learning and software engineering best practices.

I learned that for scientific data with complex physical rules, "Ensemble Learning" methods like Random Forest are significantly more powerful than single-formula models like Linear Regression.

I also discovered that professional documentation is a critical part of the data analytics process, as the ability to translate raw code and metrics into a clear narrative is essential for communicating findings.

Furthermore, strictly organizing code into modules and maintaining a clean file structure improved my ability to manage complex projects, proving that a presentable, modular workflow is a fundamental skill for efficiency.

13. Future Enhancements

To improve the system further, I would implement the following advanced techniques:

Web Interface: I want to wrap the model in a Flask or Streamlit website so that researchers can upload their own CSV files directly from a browser.

If I researched and properly invested my time I could potentially:

1. **Generative Adversarial Networks (GANs):** Instead of just predicting the temperature of existing materials, we could use a GAN to *generate* new chemical formulas that are likely to be high-temperature superconductors. This would change the tool from a "predictor" to a "discoverer."
2. **Convolutional Neural Networks (CNNs):** Recent research suggests representing the chemical elements as an image based on the periodic table. We could then use a CNN (Deep Learning) to find spatial patterns in the element properties, which might be more accurate than standard regression.

14. References

1. https://research-it.berkeley.edu/news/how-materials-project-connects-computational-and-experimental-materials-science

2. https://en.wikipedia.org/wiki/Edisonian_approach#:~:text=The%20Edisonian%20approach%20to%20invention,a%20practical%20incandescent%20light%20bulb.
3. https://pubs.acs.org/doi/10.1021/acs.chemmater.4c01757
4. sklearn.linear_model — scikit-learn 1.7.2 documentation
5. https://xgboost.readthedocs.io/en/stable/
6. (for the diagrams) https://mermaid.live/