1. Why should this code be written in C/C++ only? Why not Java, C# or Python?

Performance and Efficiency: C/C++ offers higher performance and efficiency, crucial for mathematical computations and machine learning models, especially when handling complex calculations like the Taylor Series.

Memory Management: C/C++ provides more control over memory management, which is beneficial for optimizing the performance of algorithms, particularly on devices with limited memory.

Build difference: There is an option to have two different builts with the Conditional Compilation: Some code might be included or excluded in different builds using preprocessor directives (like #ifdef DEBUG). This is not available in other languages. However it is doable.

1. Name two other possible common usages for this design pattern.
   1. Random Forest: Random Forest is a popular ensemble learning technique that combines multiple decision trees to make predictions. Each decision tree is considered a component of the ensemble, and the final prediction is obtained through a voting or averaging mechanism. Random Forests can handle both classification and regression tasks and are known for their ability to handle complex data and reduce overfitting.
   2. Gradient Boosting: Gradient Boosting is another ensemble technique that builds a strong predictive model by combining multiple weak models, typically decision trees. Each weak model is trained to correct the mistakes made by the previous models in the ensemble. Gradient Boosting algorithms, such as XGBoost or LightGBM, have achieved remarkable success in various machine learning competitions and real-world applications.

3. Did you use interface classes for all components in this assignment? If not, why?

I didn’t use interface for all components just with the view class that handles two different view outputs. interfaces might have been an overkill, especially for components with no need for interchangeable behavior.

4. What might cause a test build to behave differently from a release build?

1. Debugging Information: Test builds often include additional debugging information and are compiled with less optimization to facilitate debugging.
2. Optimization Levels: Release builds are usually optimized for performance, which can lead to different behavior, especially in timing or resource usage.
3. Conditional Compilation: Some code might be included or excluded in different builds using preprocessor directives (like #ifdef DEBUG).