Polymorphism is the ability of different objects responding to the same message ore method call in various ways. It's used in this case through the implementation of an abstract class 'SummaryStrategy' and its subclasses 'AverageSummary' and 'MinMaxSummary'. 'SummaryStrategy' declares and abstract method 'PrintSummary(List<int> numbers)', serves as a contract making sure that any class inheriting must provide its own implementation of 'PrintSummary(List<int> numbers)'. 'AverageSummary' calculates and prints the average number while 'MinMaxSummary' finds and prints the minimum and maximum values. In 'DataAnalyser' theres a reference to 'SummaryStrategy' which holds 'AverageSummary' and 'MinMaxSummary' which allows to use 'PrintSummary' Polymorphically.

The principle of abstraction focuses on hiding complexity, generalization as well as the essential characteristics of an object rather its specific instances. For an example a file handling system, without abstraction the user will need to worry about dealing with complex file handling such as open, read, write, error handling and close operation directly. With abstraction we can introduce a FileHandler interface or class with simple operations such as open, close, edit, save. The user doesn't need to know how the files are opened, edited, or saved internally, making the process simpler.

The issue with the original design in Task 1 had 'DataAnalyser' class directly implenmenting the logic for different summary types through a series of conditional statements. It becomes problematic with increment of summaries e.g., 50 summaries. It would have scalability issues due to each new summary type would require modifying the 'DataAnalyser' class leads to a bloated class. It becomes more complex and more prone to errors.