Note on Pydantic models

compiled by D.Gueorguiev, 6/24/25

One of the primary ways of defining schema in Pydantic is via models. Models are simply classes which inherit from BaseModel and define fields as annotated attributes.

One can think of models as similar to structs in languages like C, or as the requirements of a single endpoint in an API.

Models share many similarities with Python’s dataclasses, but have been designed with some subtle-yet-important differences that streamline certain workflows related to validation, serialization, and JSON schema generation.

Untrusted data can be passed to a model and , after parsing and validation, Pydantic guarantees that the fields of the resultant model instance will conform to the field types defined on the model.

**Note**: The term *validation* refers to the process of instantiating a model (or other type) that adheres to specified types and constraints. However, Pydantic’s primary focus does not align precisely with the dictionary definition of *validation*. Pydantic guarantees the types and constraints of the output, not the input data. This distinction becomes apparent when considering that Pydantic’s ValidationError is raised when data cannot be successfully parsed into a model instance.

While this distinction may initially seem subtle, it holds practical significance. In some cases *validation* goes beyond just model creation, and can include the copying and coercion of data. This can involve copying arguments passed to the constructor in order to perform coercion to the new type without mutating the original input data.

The primary goal of Pydantic is to assure that the resulting structure post-processing (referred to as *validation*) precisely conforms to the applied type hints.