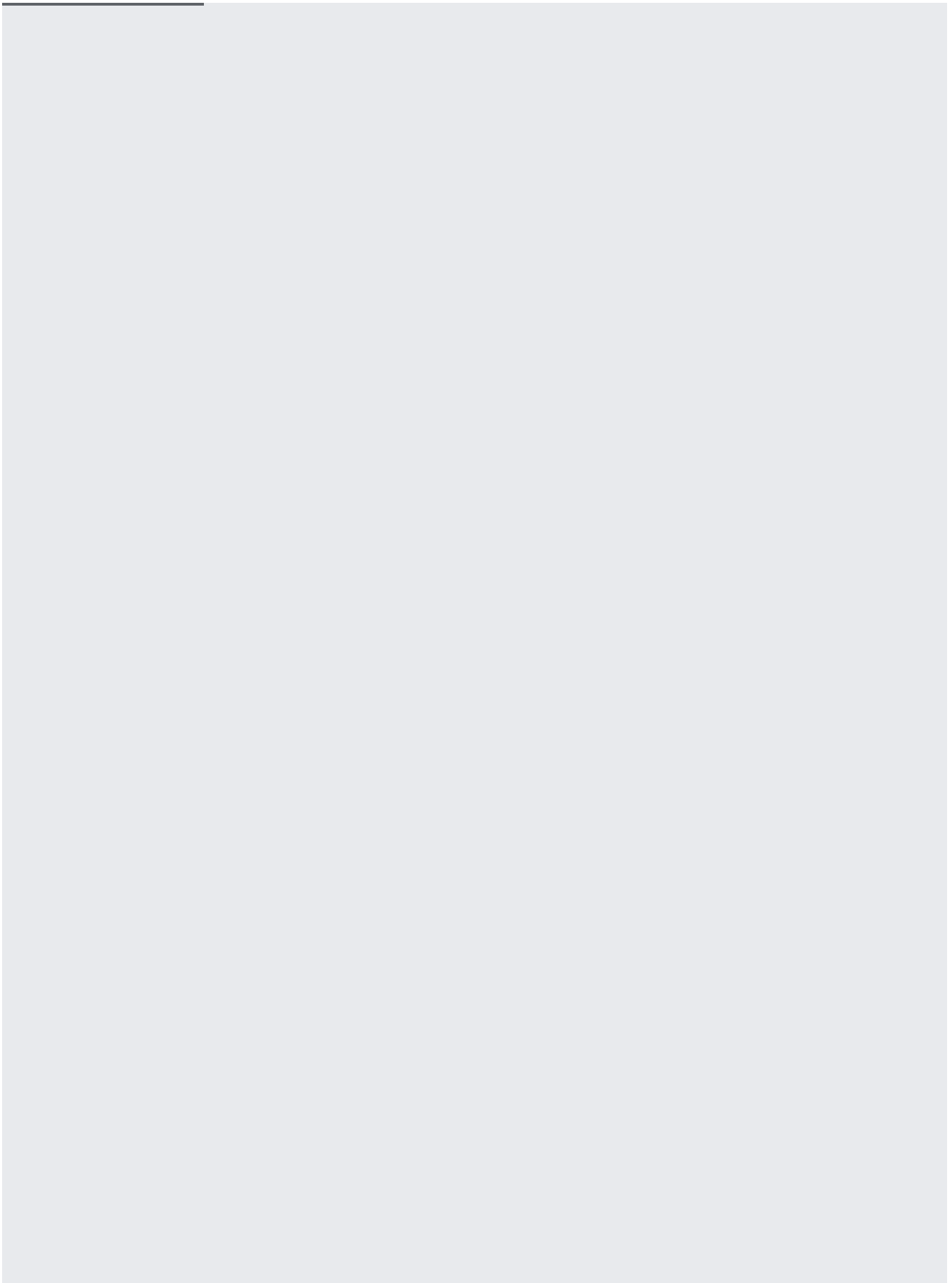


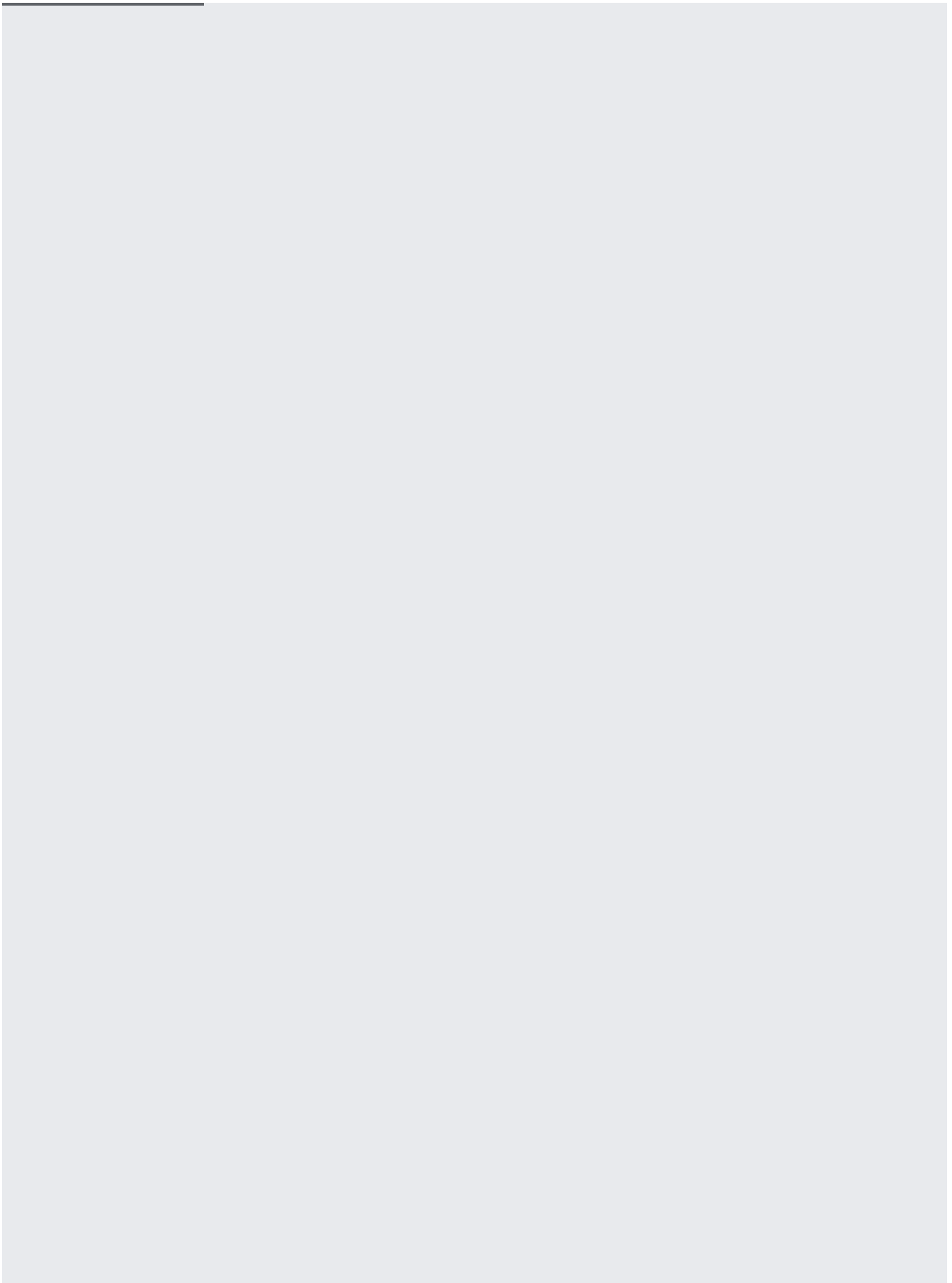
Cloud Spanner allows you to create STRUCT objects from data, as well as to use STRUCT objects as bound parameters when running a SQL query with one of the Cloud Spanner client libraries.

For more information about the STRUCT type in Cloud Spanner, see [Data types](/spanner/docs/data-types#struct-type) (/spanner/docs/data-types#struct-type).

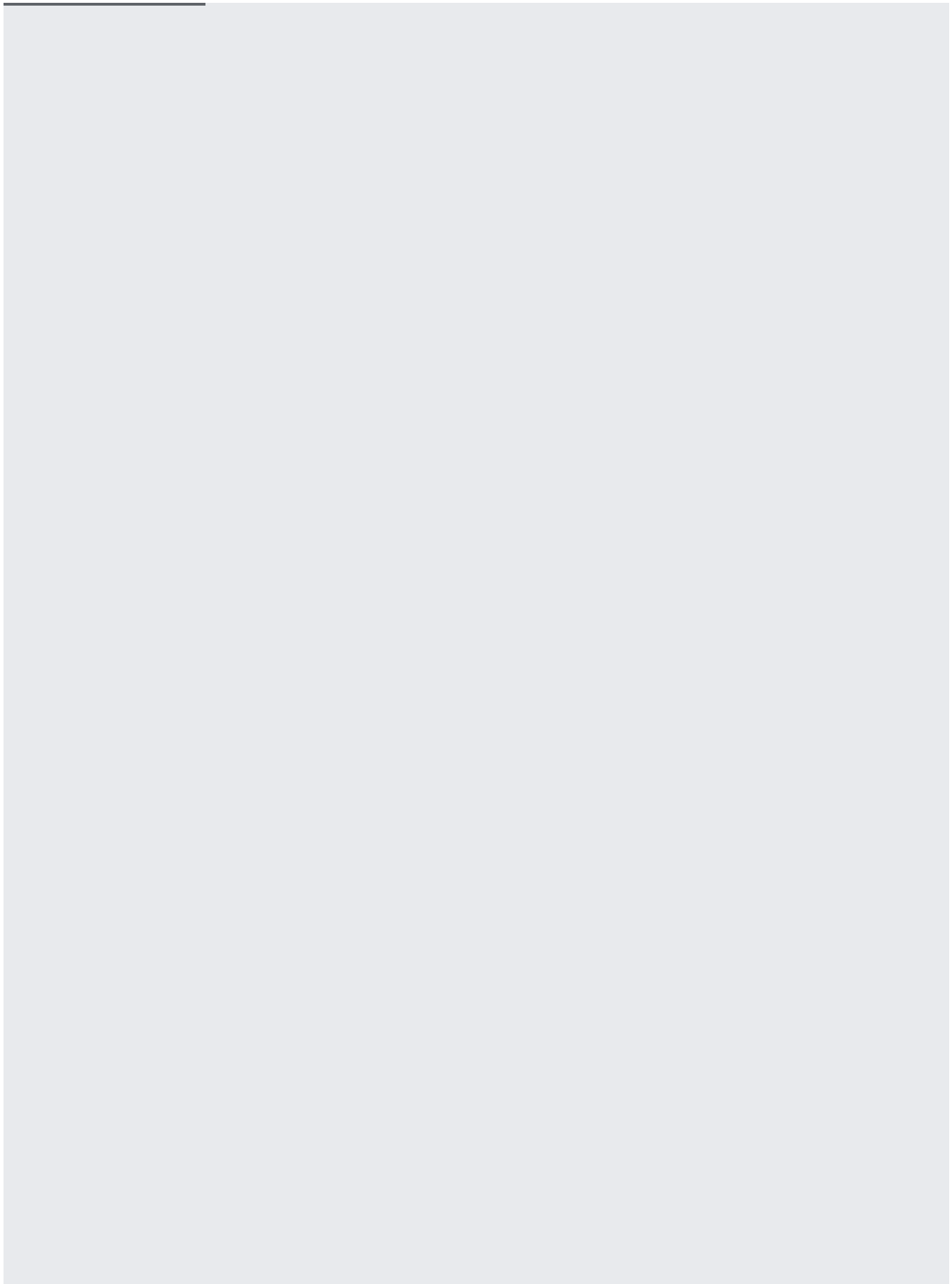
You can declare a STRUCT object in queries using the syntax described in [Declaring a STRUCT type](/spanner/docs/data-types#constructing-a-struct) (/spanner/docs/data-types#constructing-a-struct).

You can define a type of STRUCT object as a sequence of field names and their data types. You can then supply this type along with queries containing STRUCT-typed parameter bindings and Cloud Spanner will use it to check that the STRUCT parameter values in your query are valid.

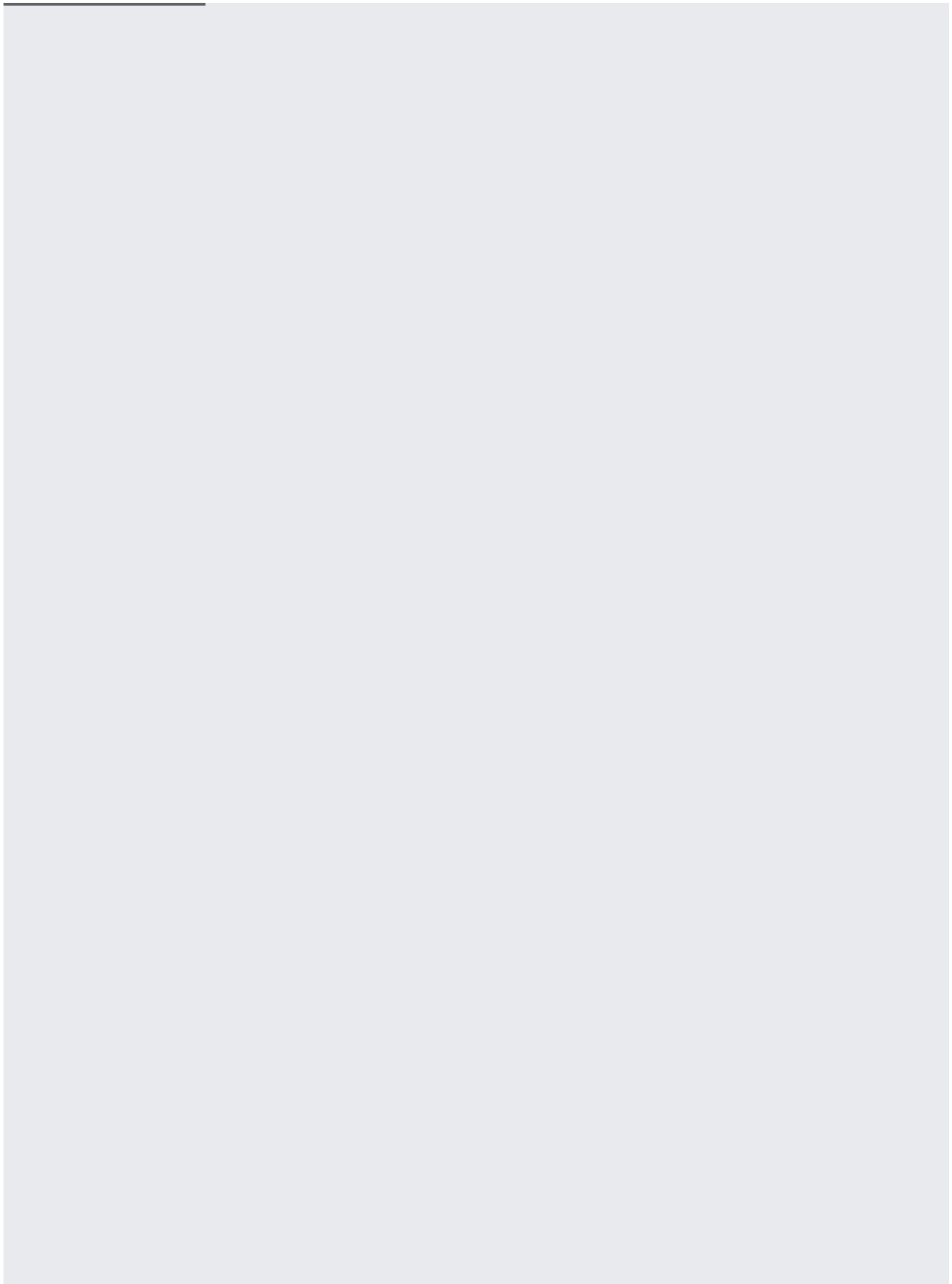


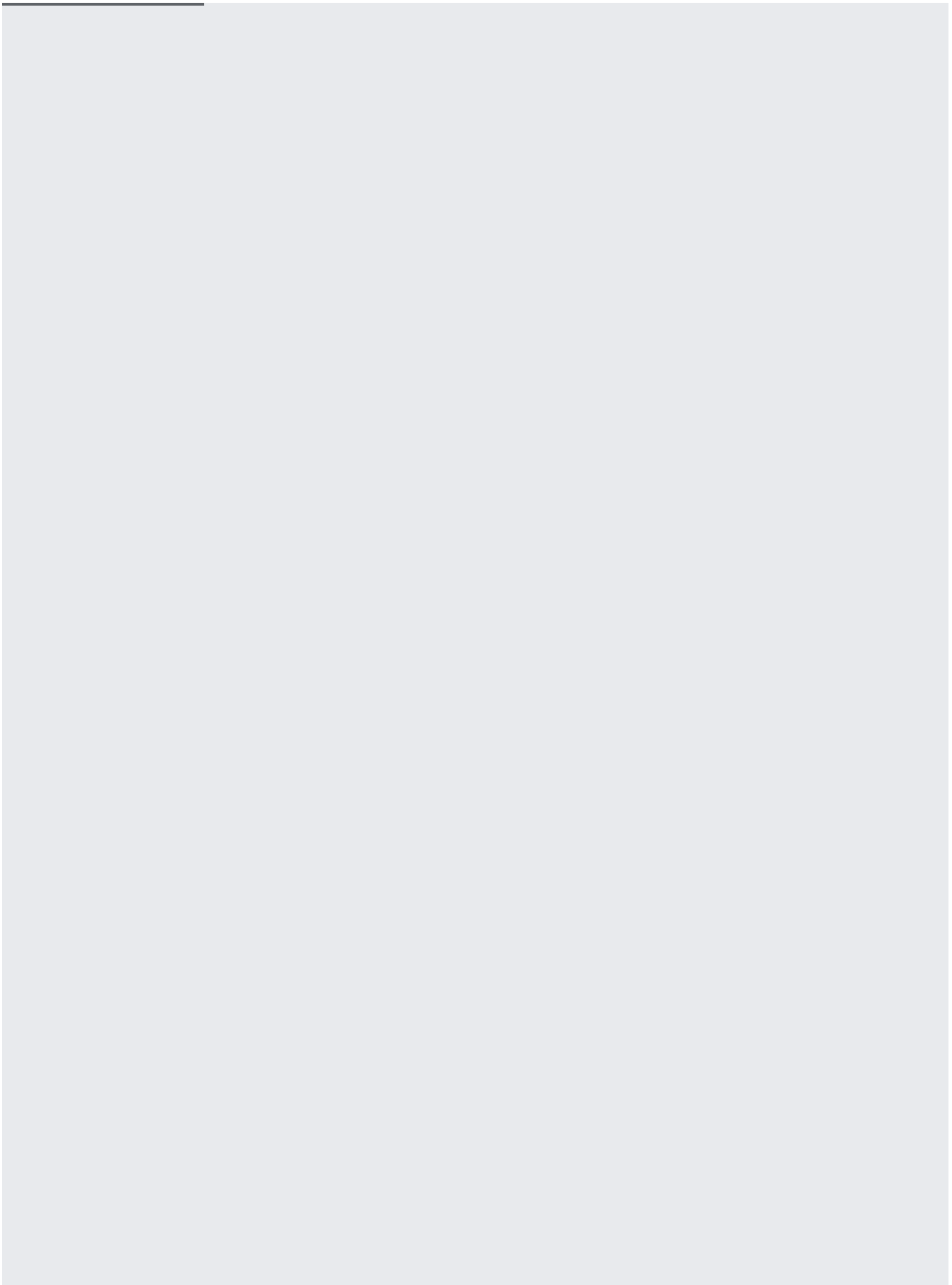


The following sample shows how to create STRUCT objects using the Cloud Spanner client libraries.



You can also use the client libraries to create an array of `STRUCT` objects, as seen in the following sample:



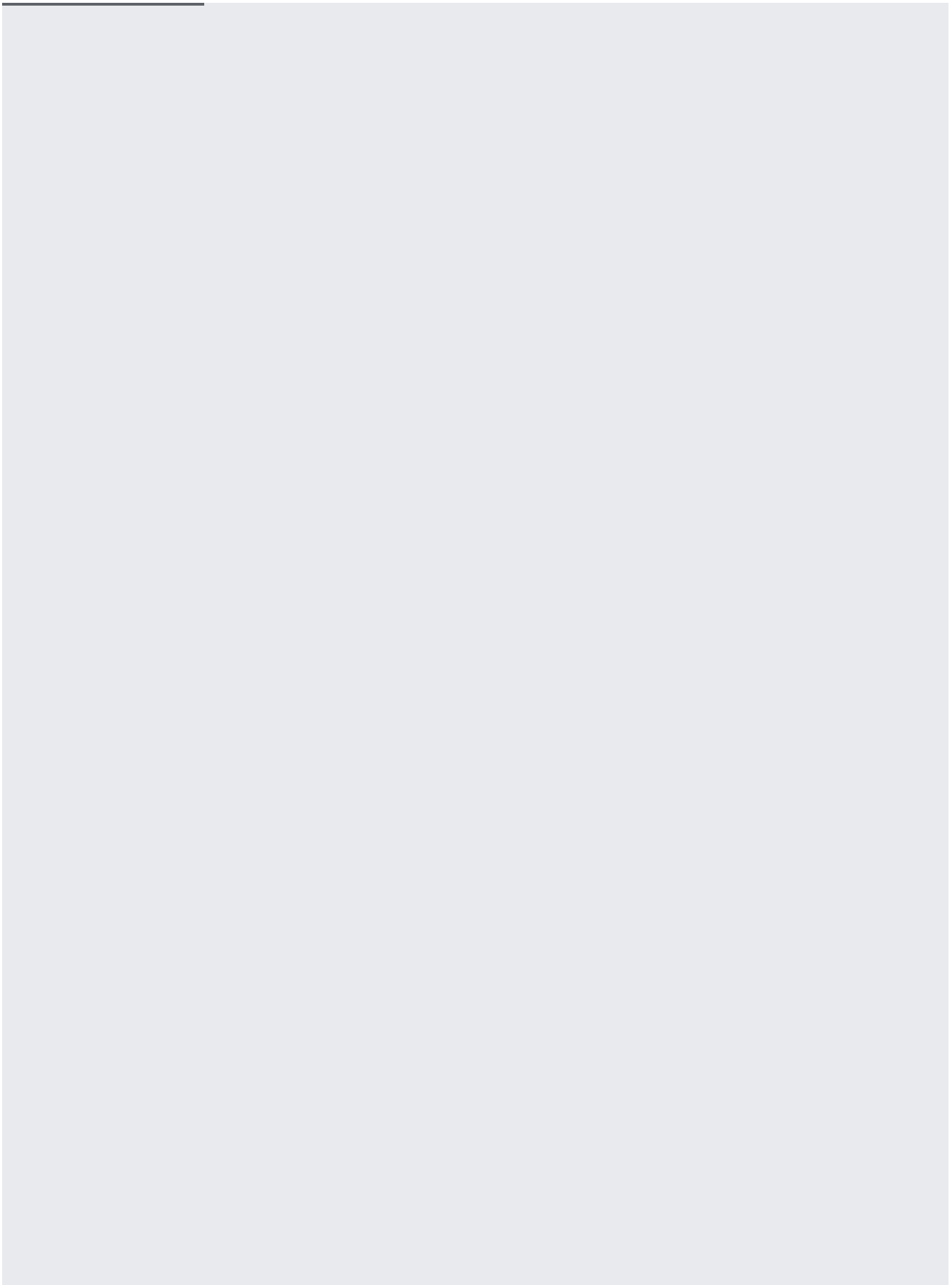


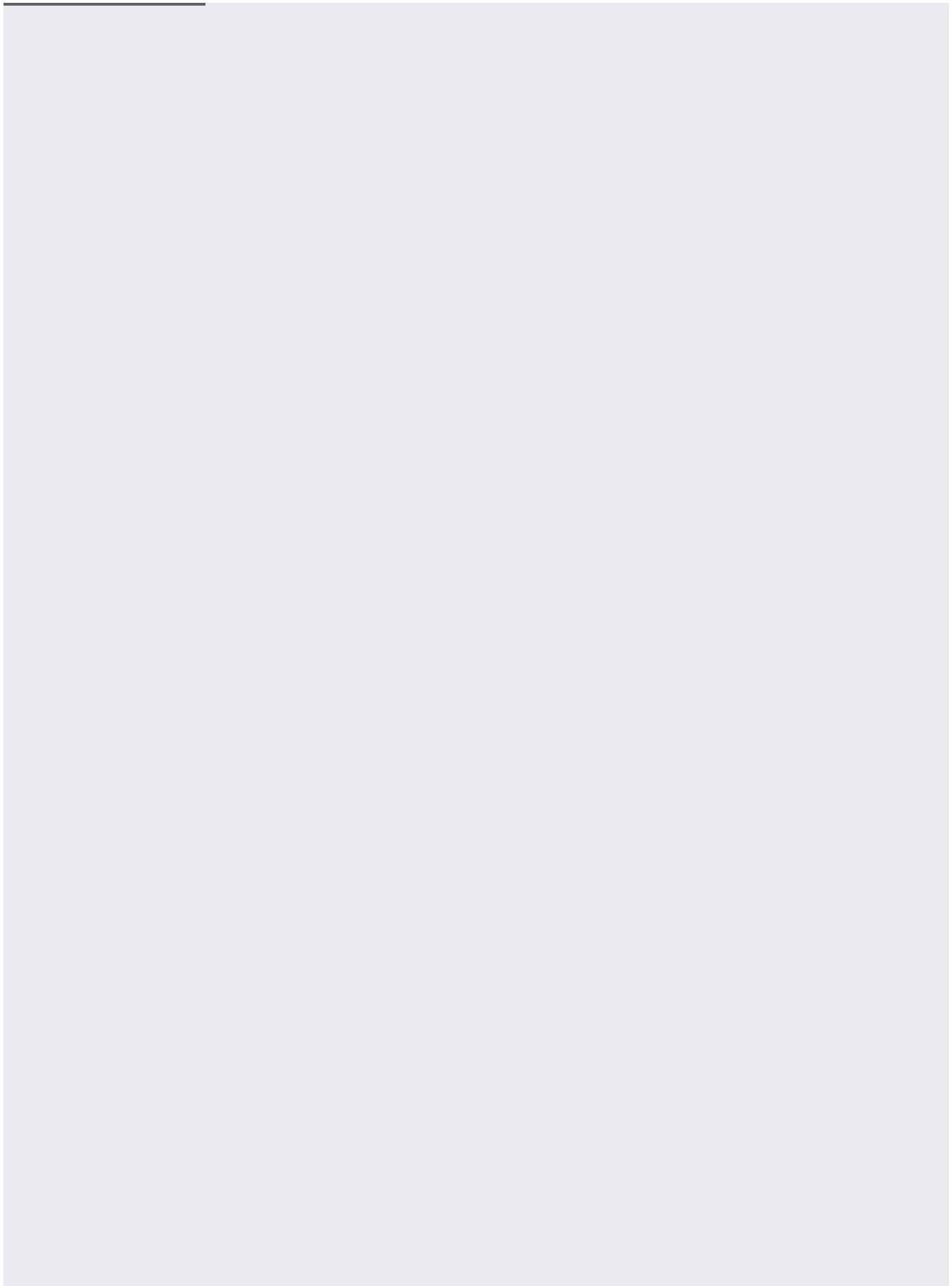


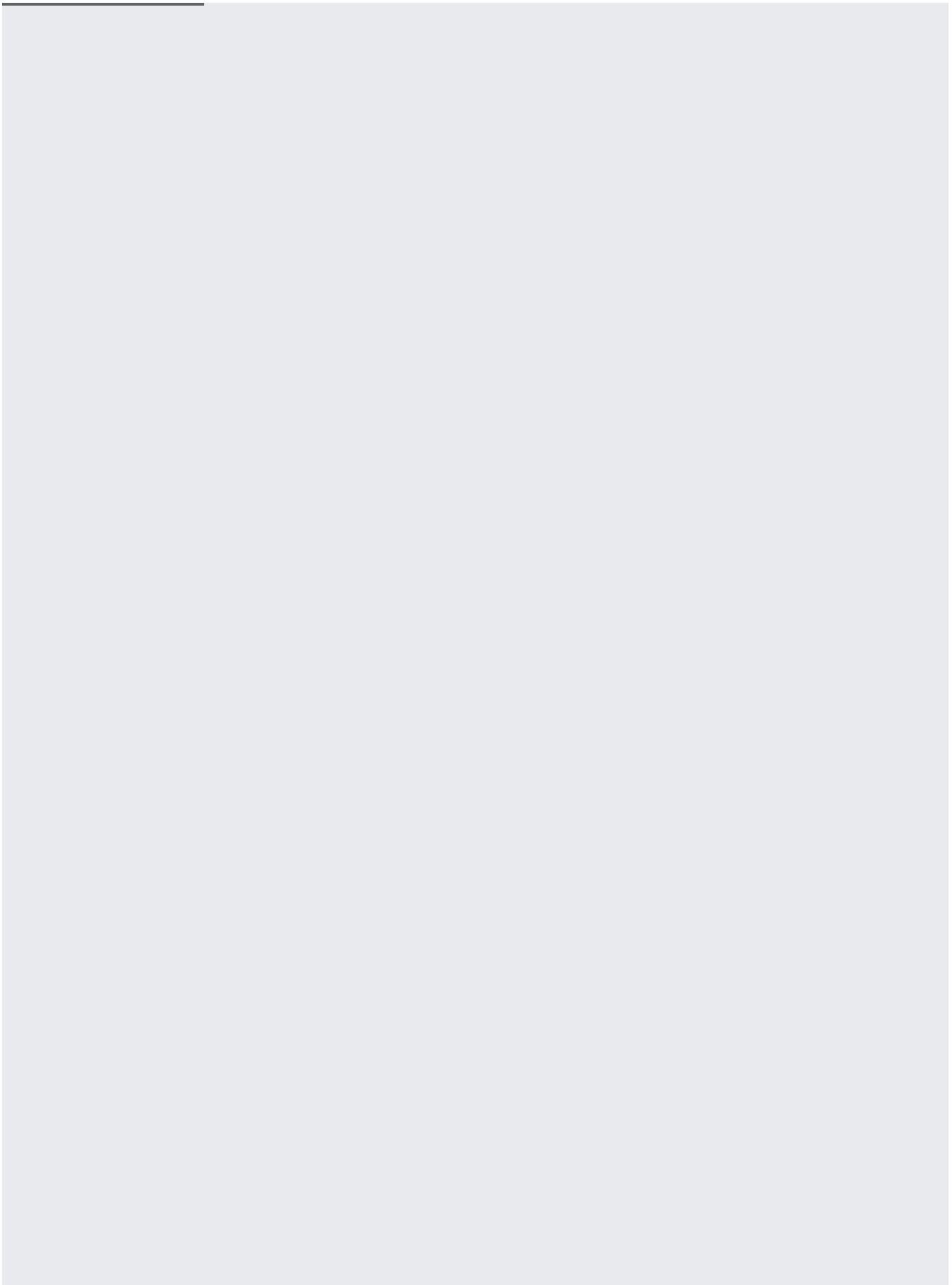
A Cloud Spanner SQL query can return an array of `STRUCT` objects as a column for certain queries. For more information, see [Using STRUCTS with SELECT](/spanner/docs/query-syntax#using-structs-with-select) (/spanner/docs/query-syntax#using-structs-with-select).

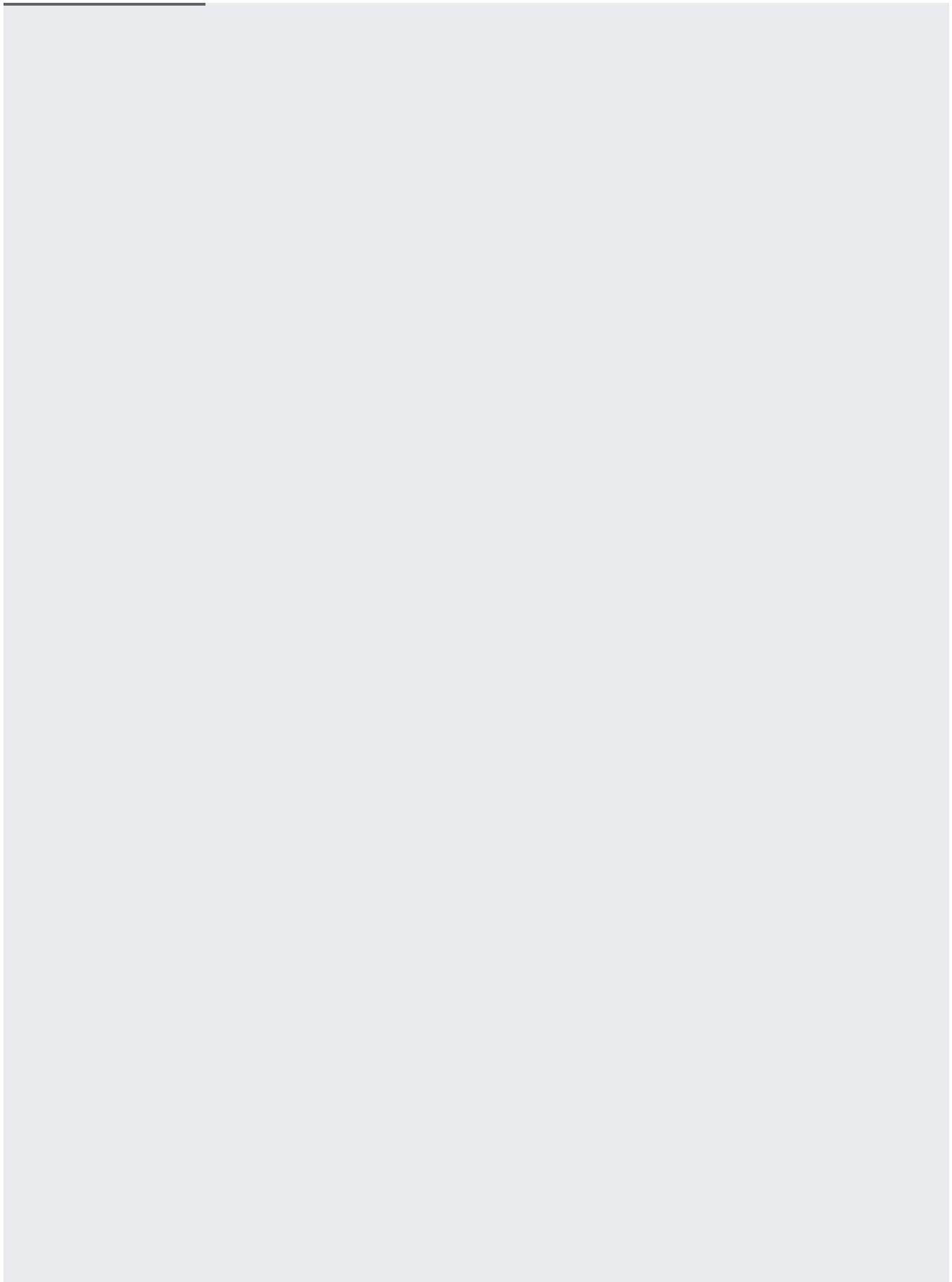
You can use `STRUCT` objects as bound parameters in a SQL query. For more information about parameters, see [Query parameters](/spanner/docs/lexical#query-parameters) (/spanner/docs/lexical#query-parameters).

The following sample shows how to bind values in a `STRUCT` object to parameters in a SQL query statement, execute the query, and output the results.

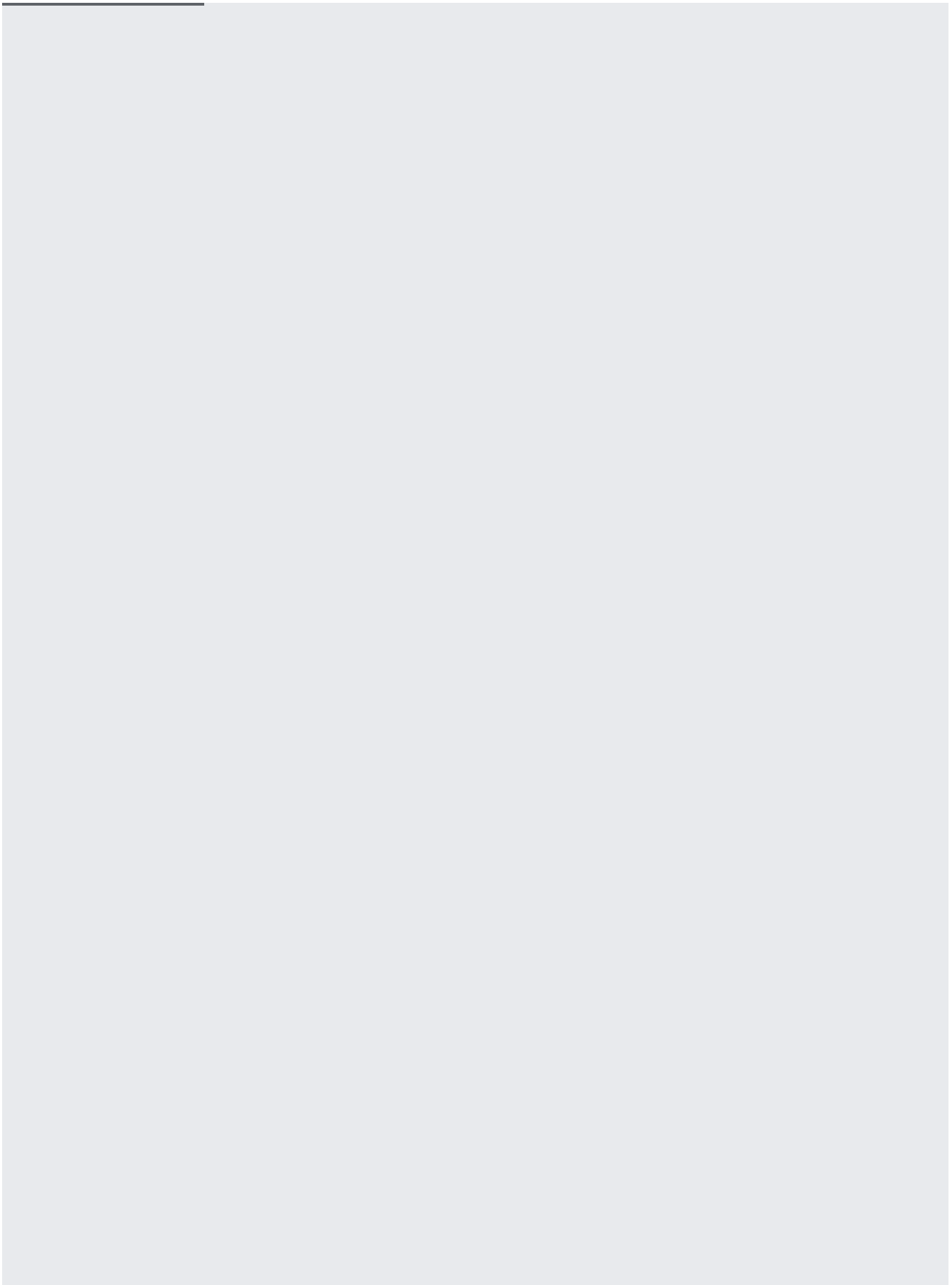


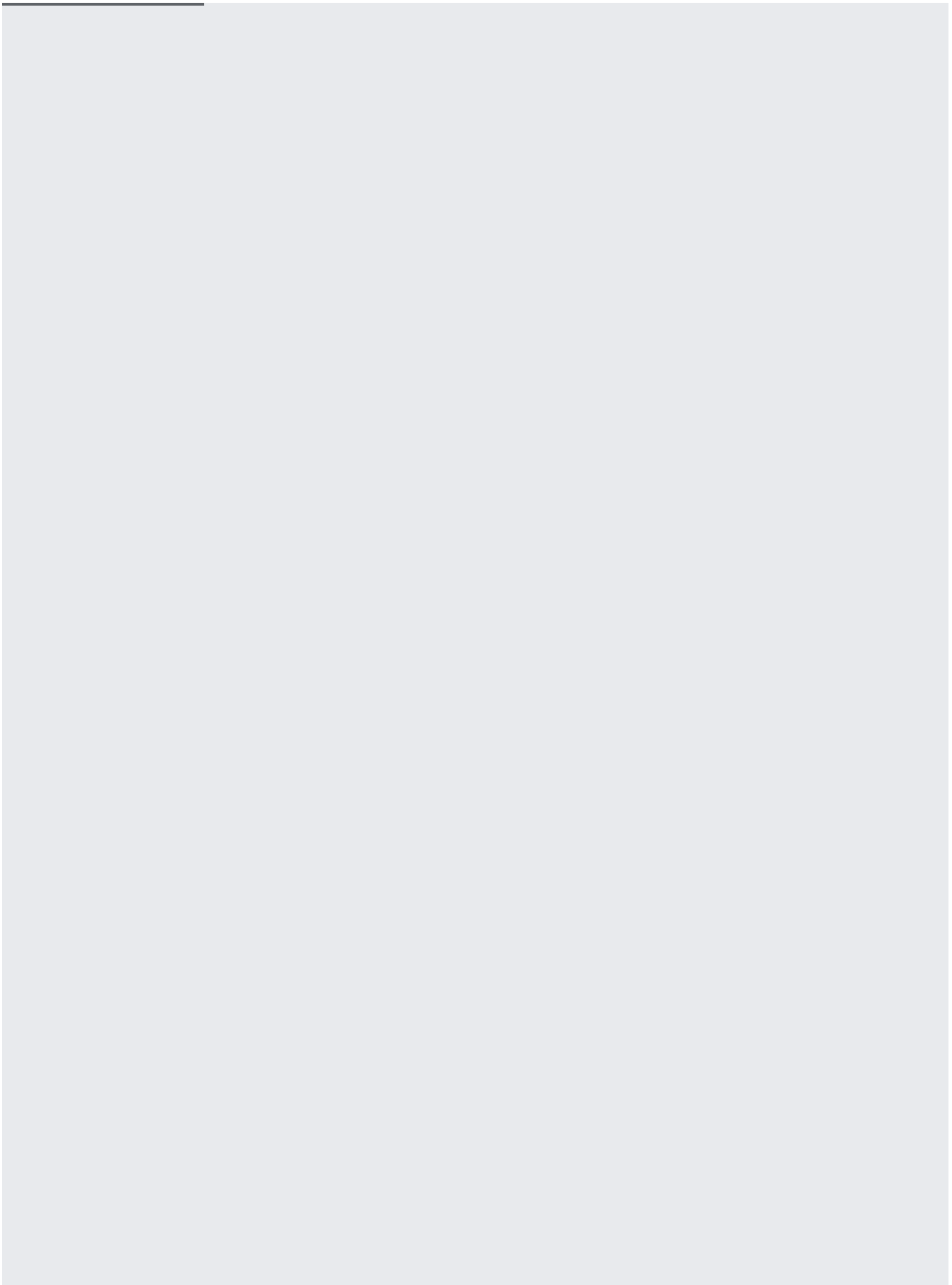




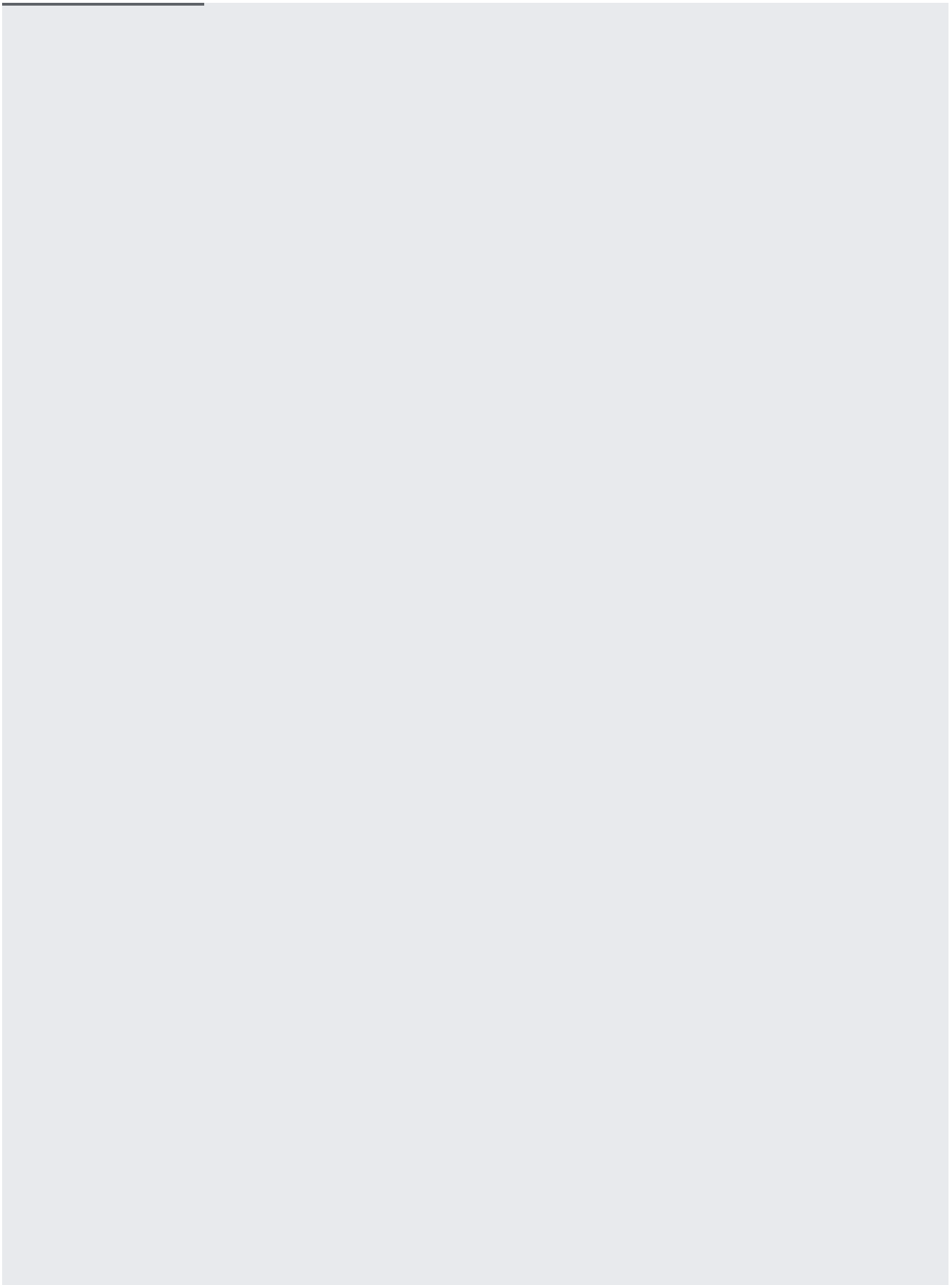


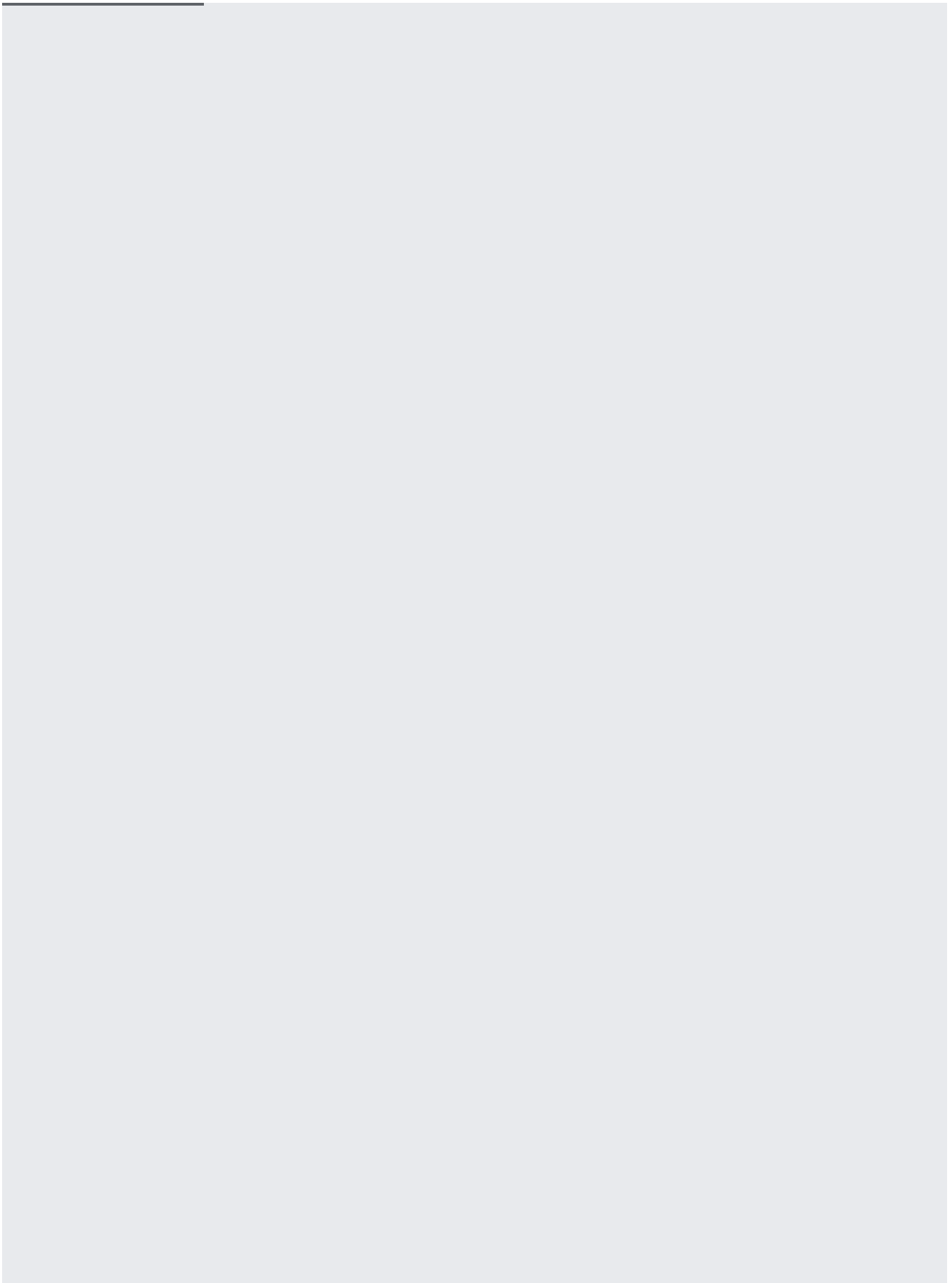
The following sample shows how to execute a query that uses an array of `STRUCT` objects. Use the `UNNEST` (</spanner/docs/query-execution-operators#array-unnest>) operator to flatten an array of `STRUCT` objects into rows:



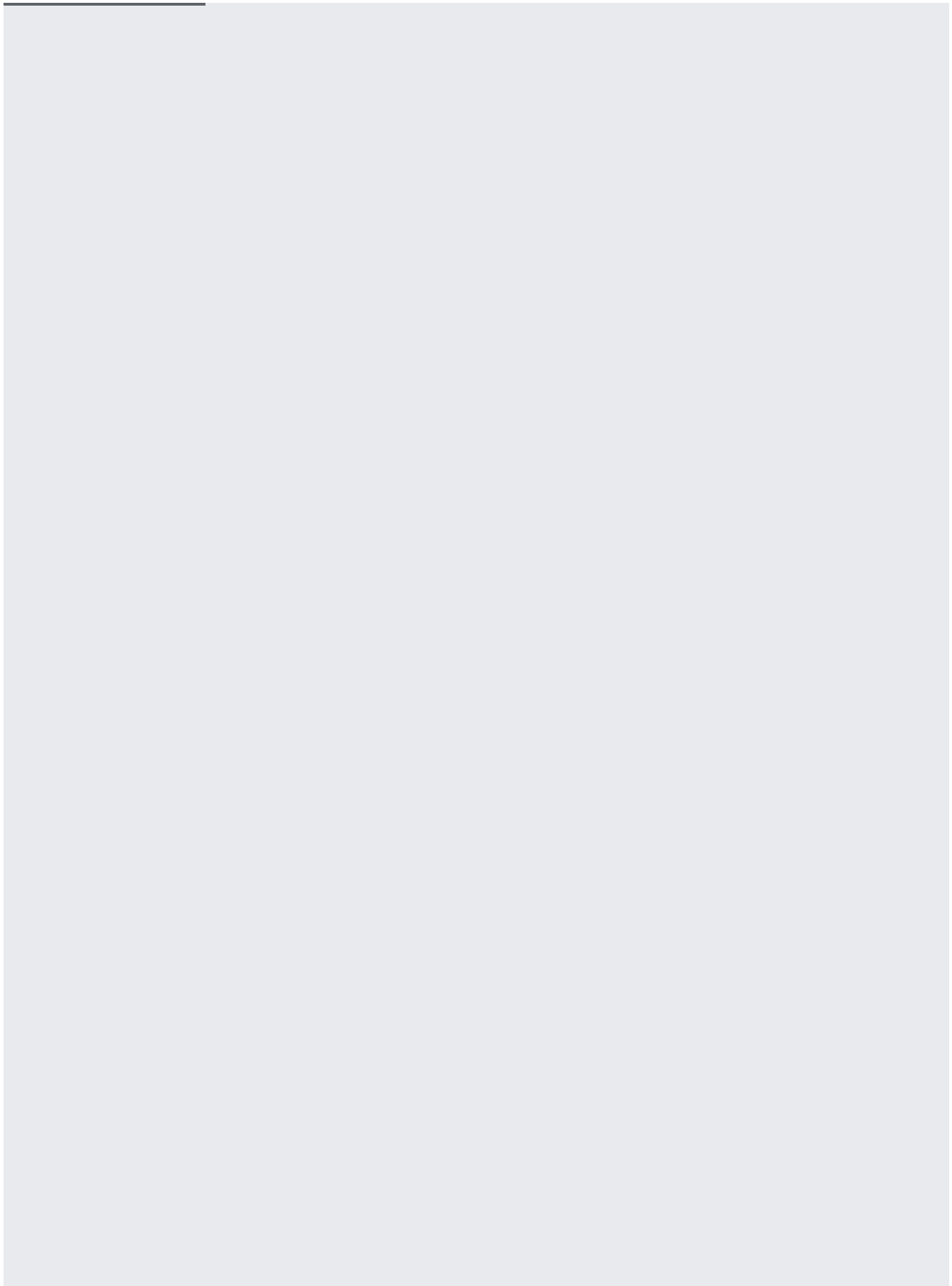


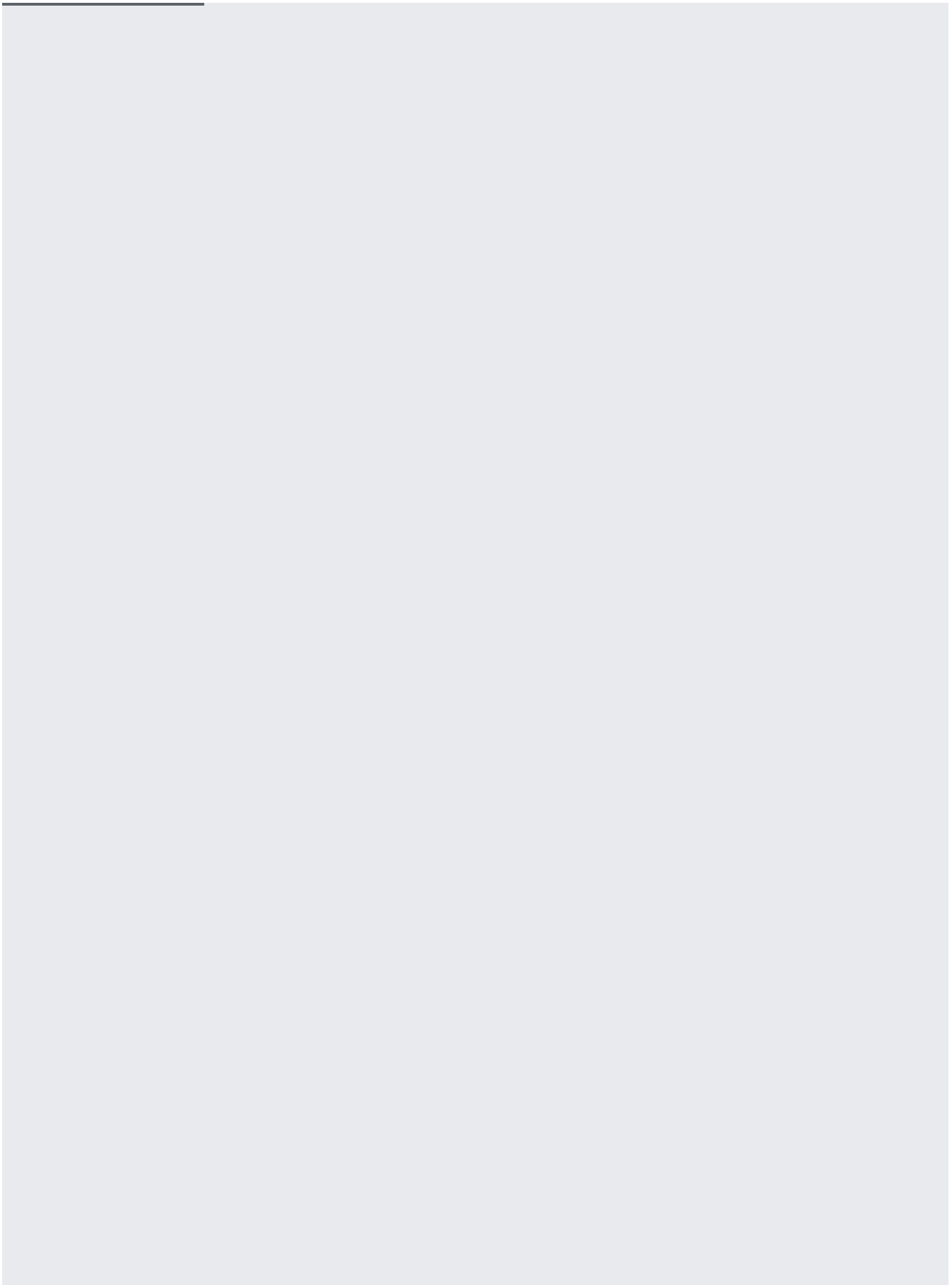


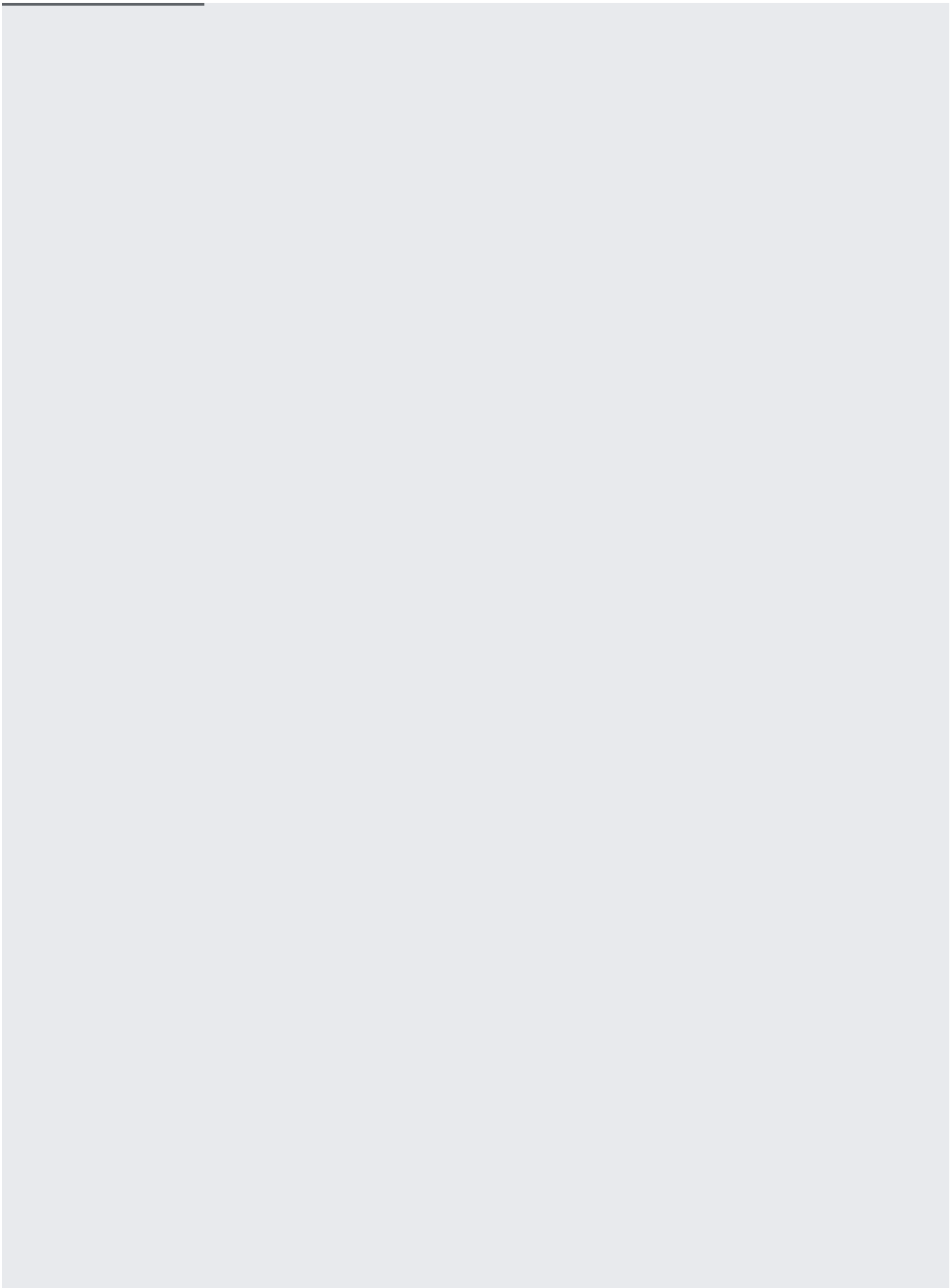


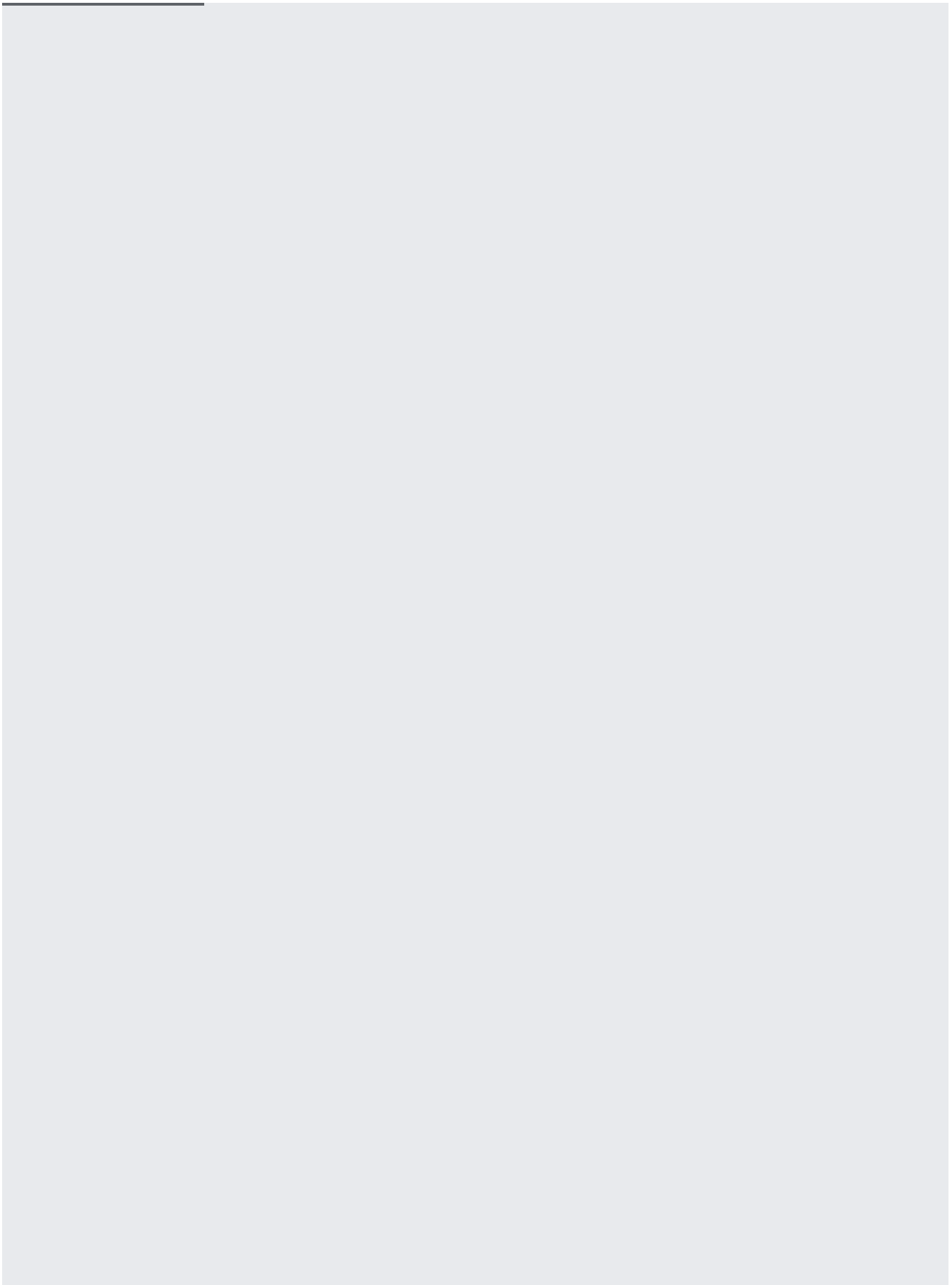


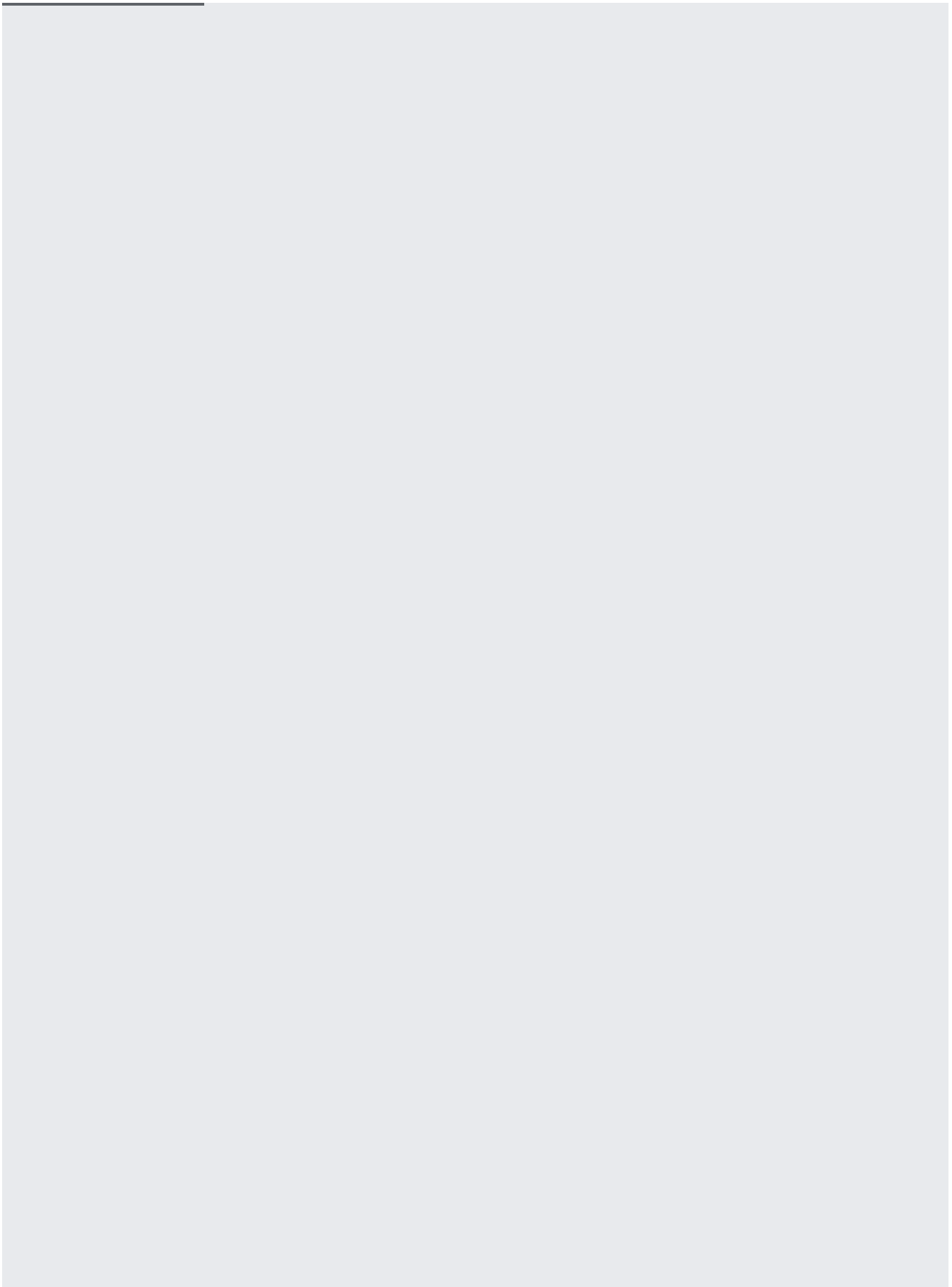
The following code example uses a `STRUCT` with bound parameters and Data Manipulation Language (DML) to update a single value in rows that match the `WHERE` clause condition. For rows where the `FirstName` is `Timothy` and the `LastName` is `Campbell`, the `LastName` is updated to `Grant`.





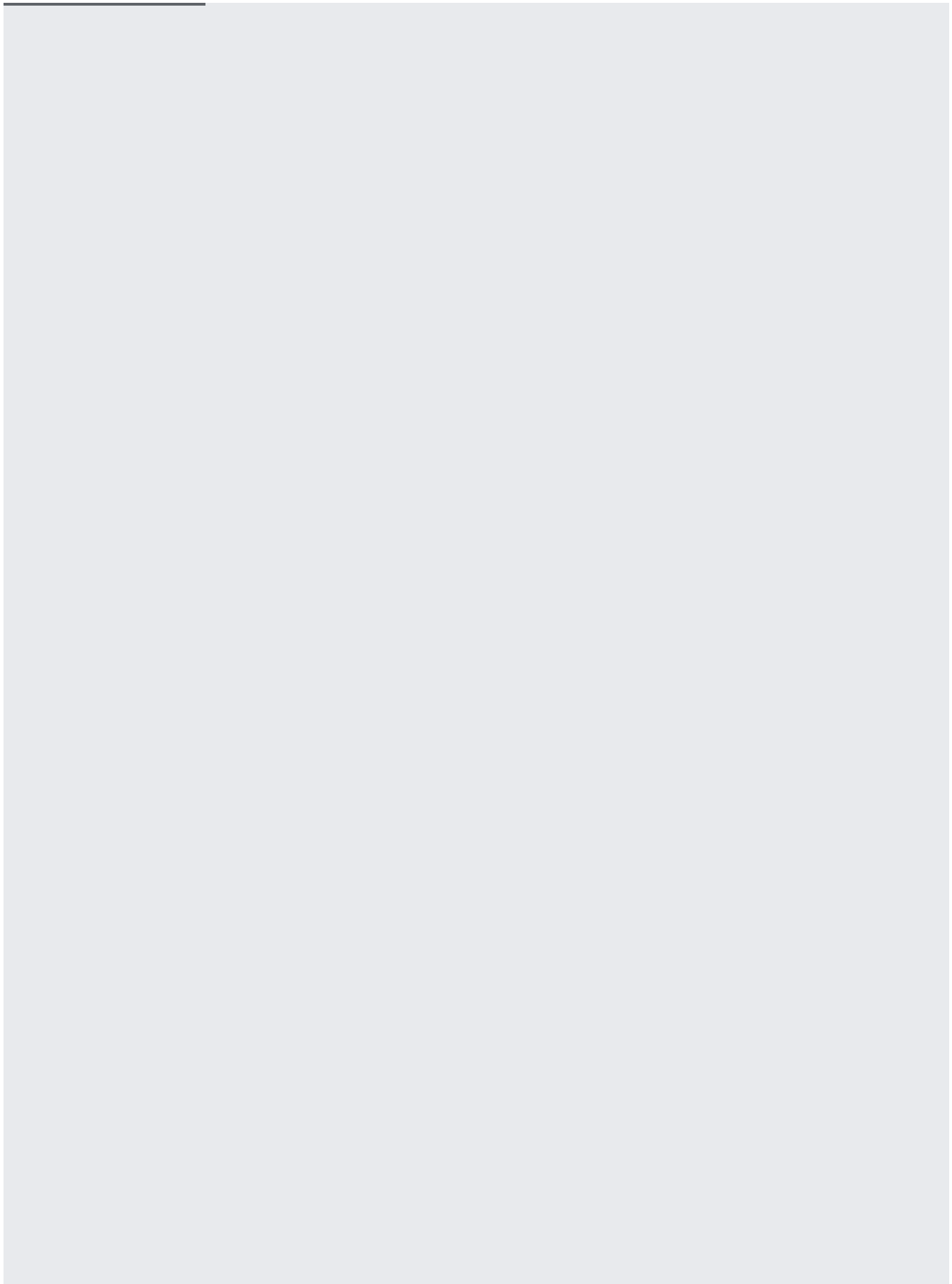


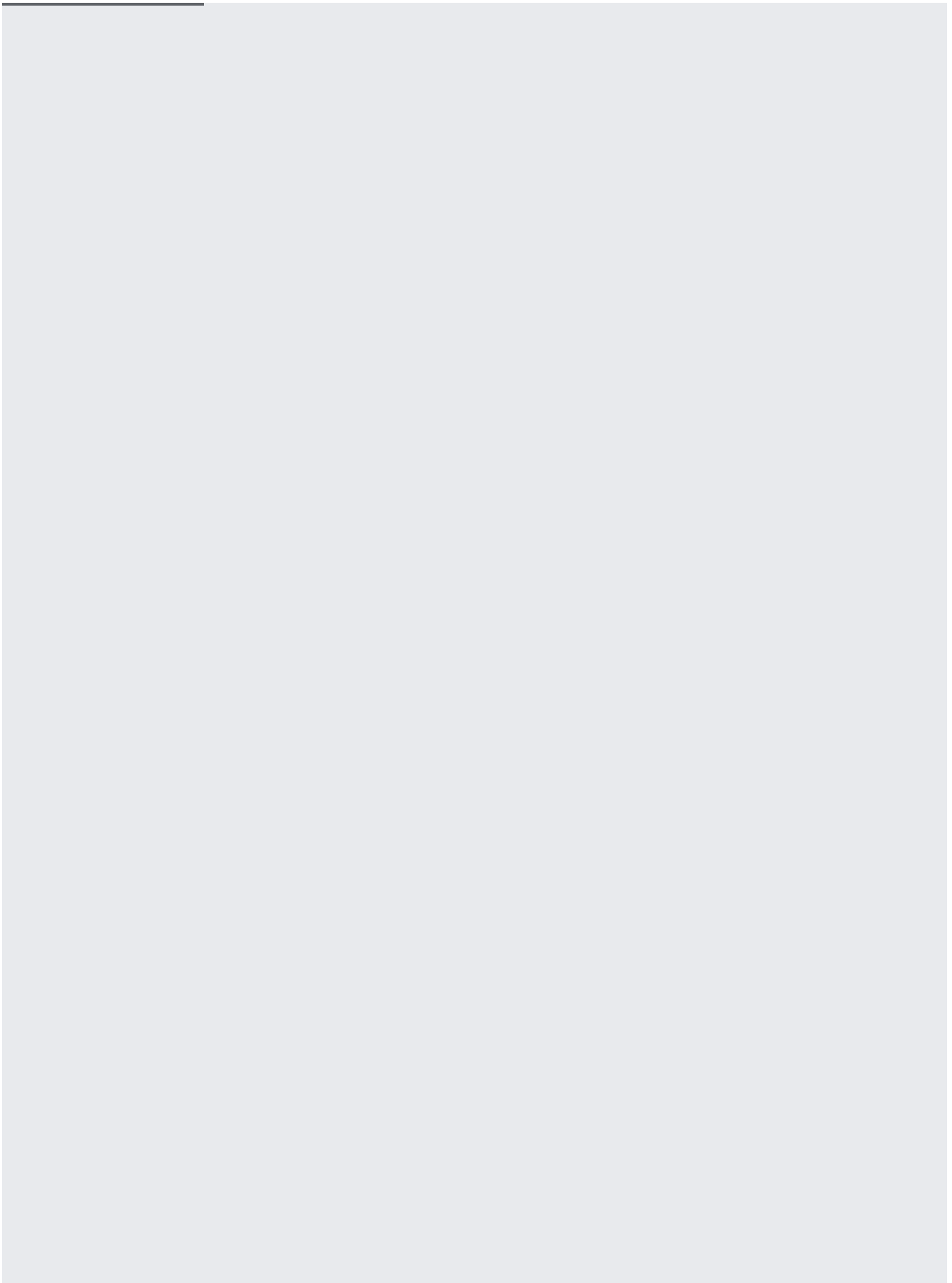


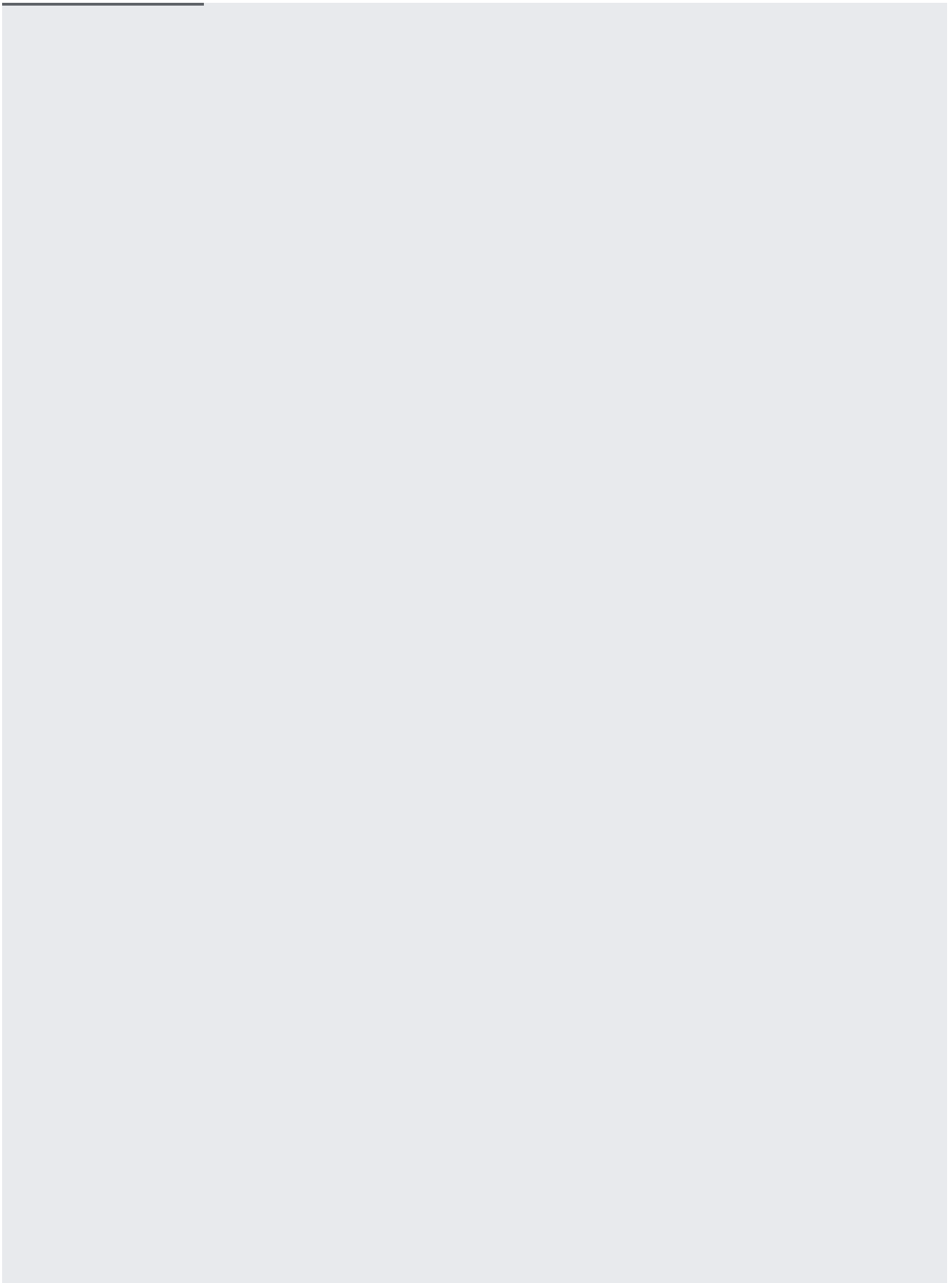


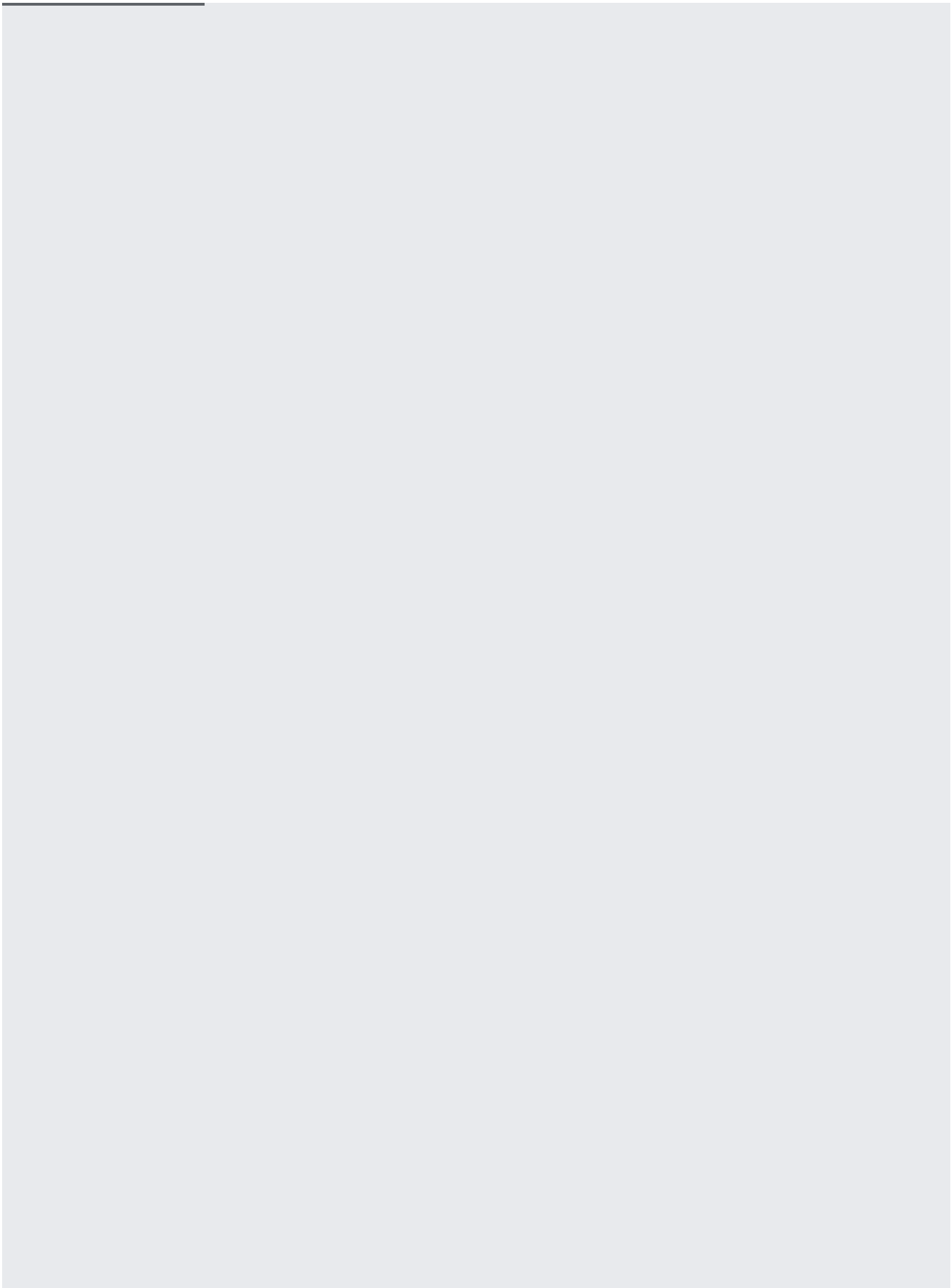


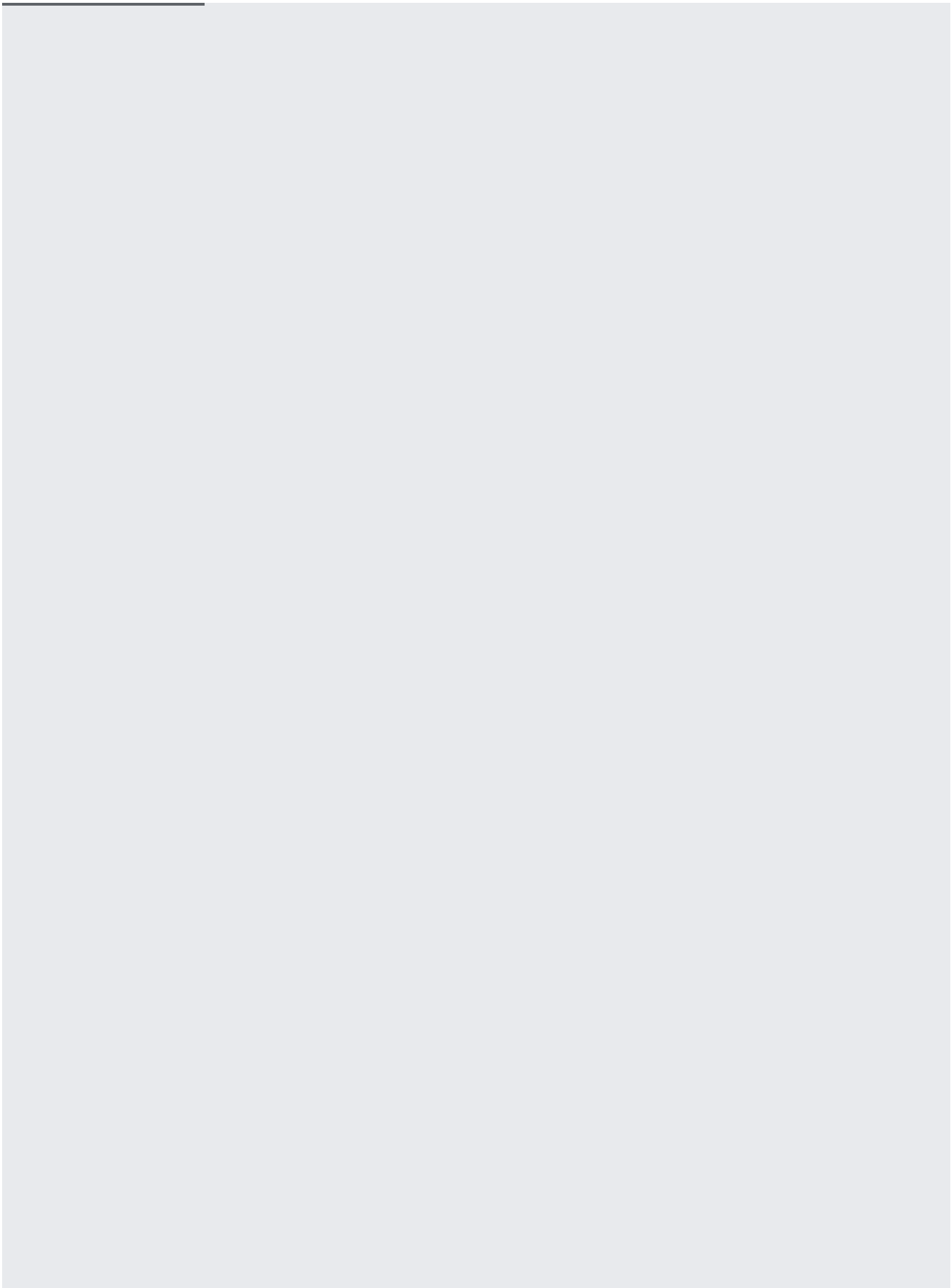
You can access fields inside a **STRUCT** object by name.

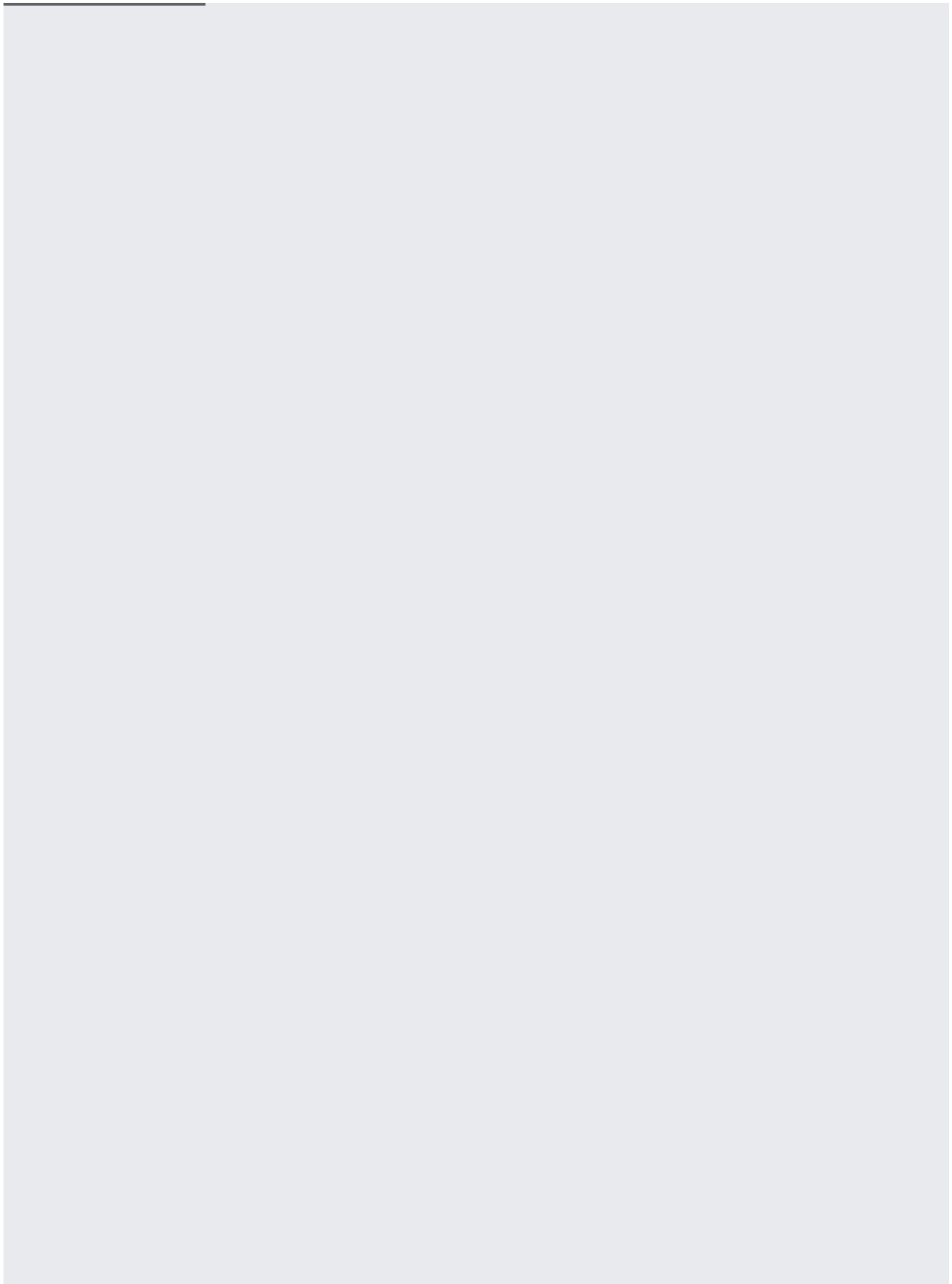


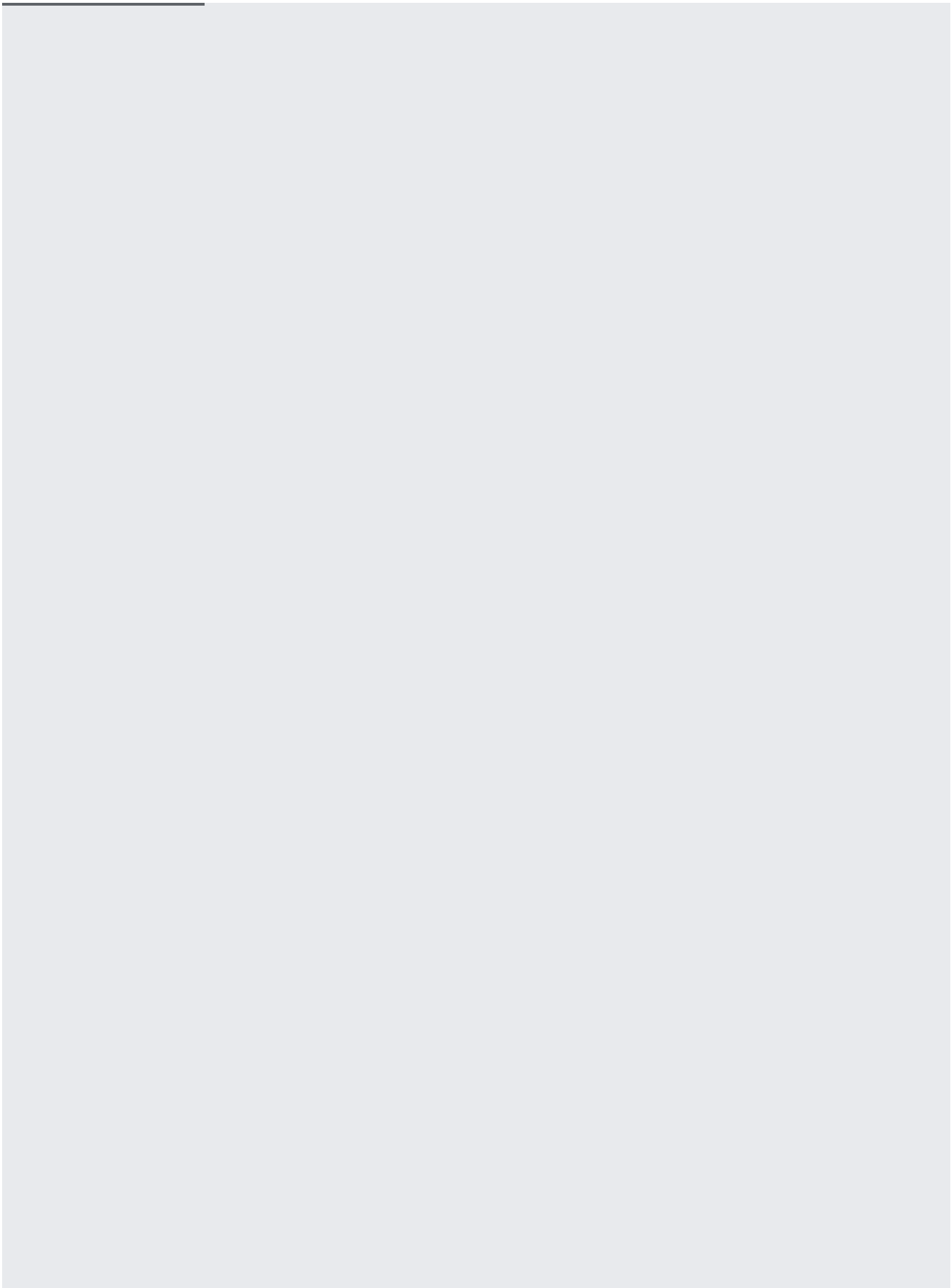




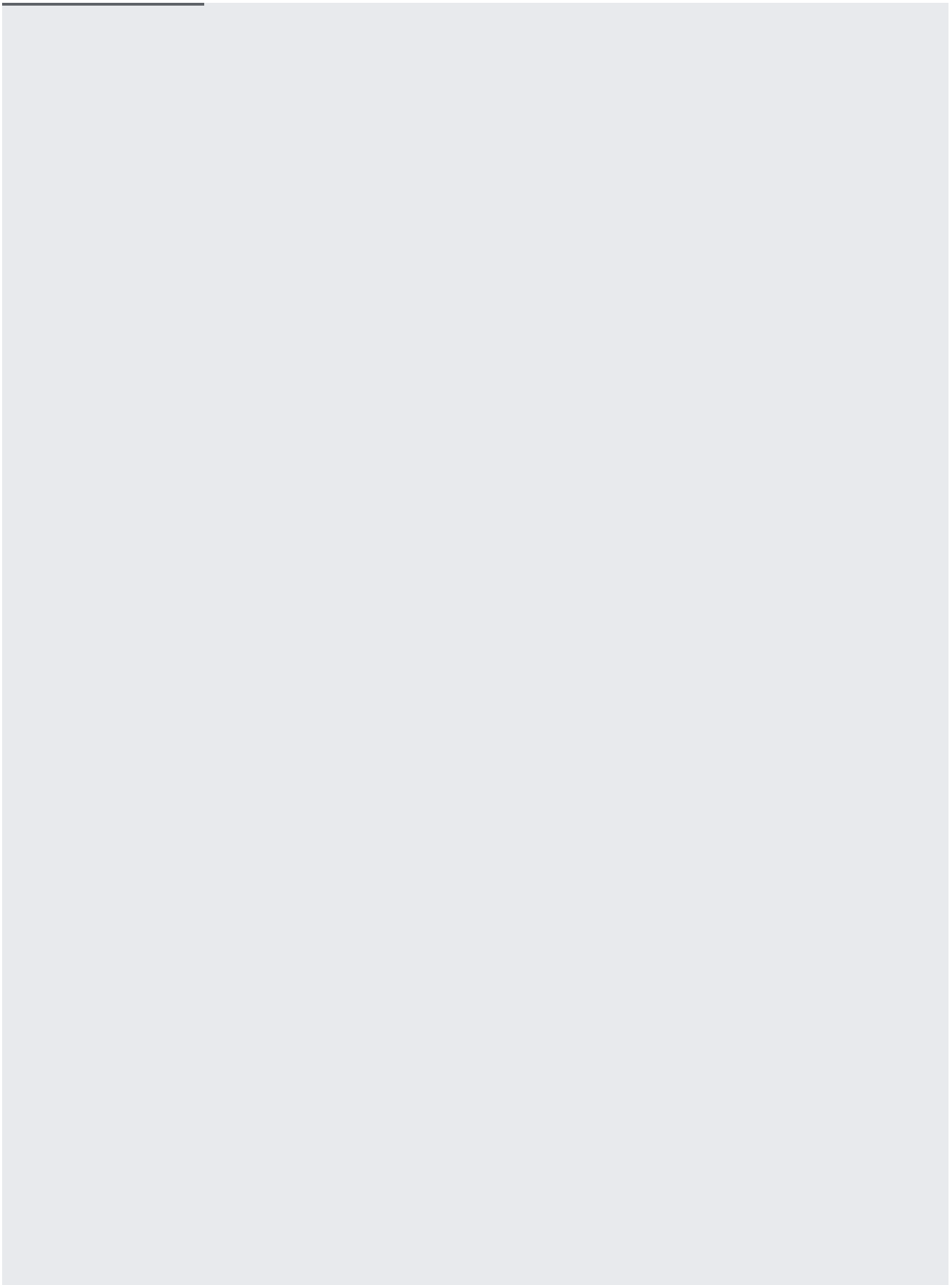


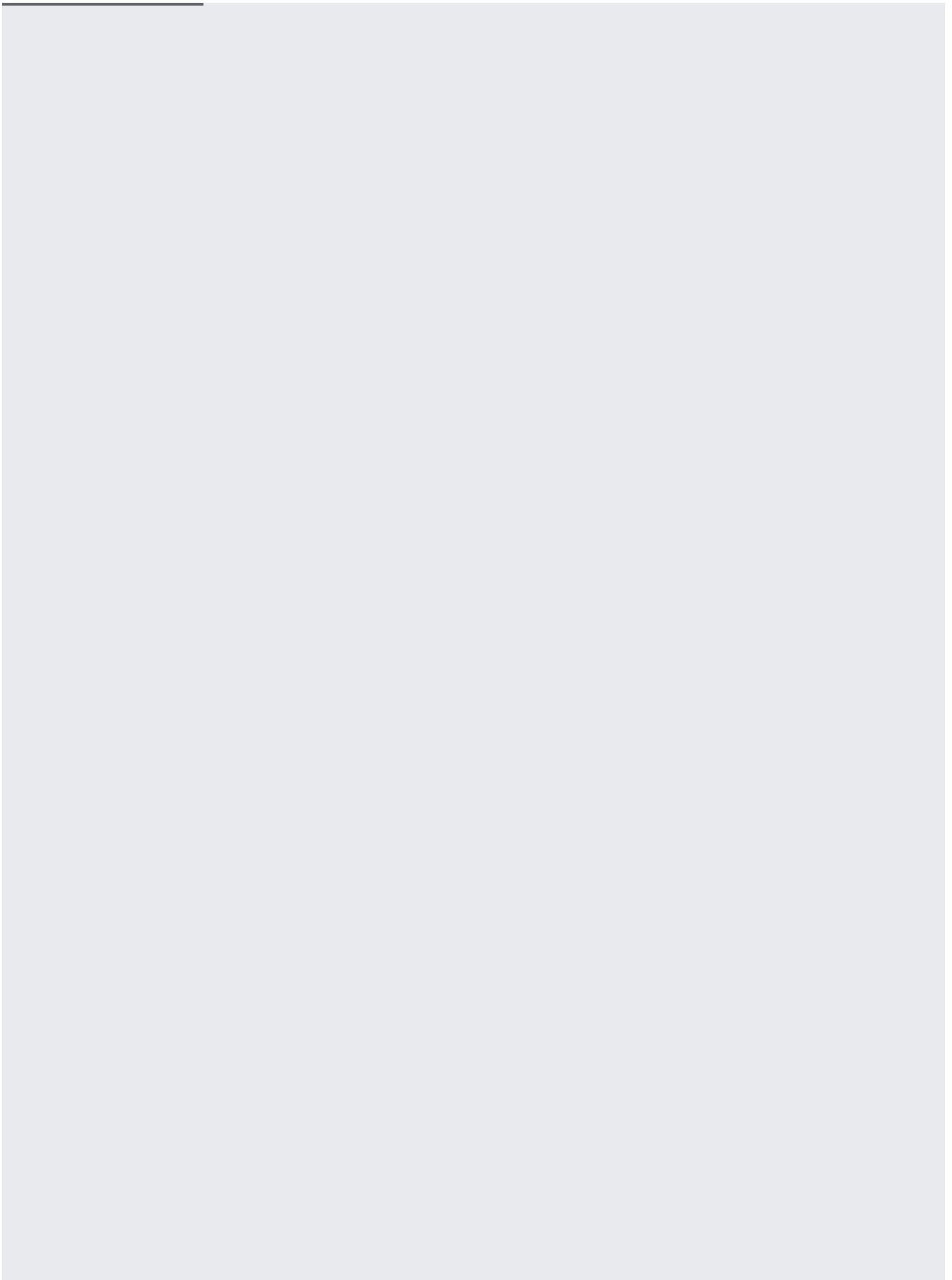


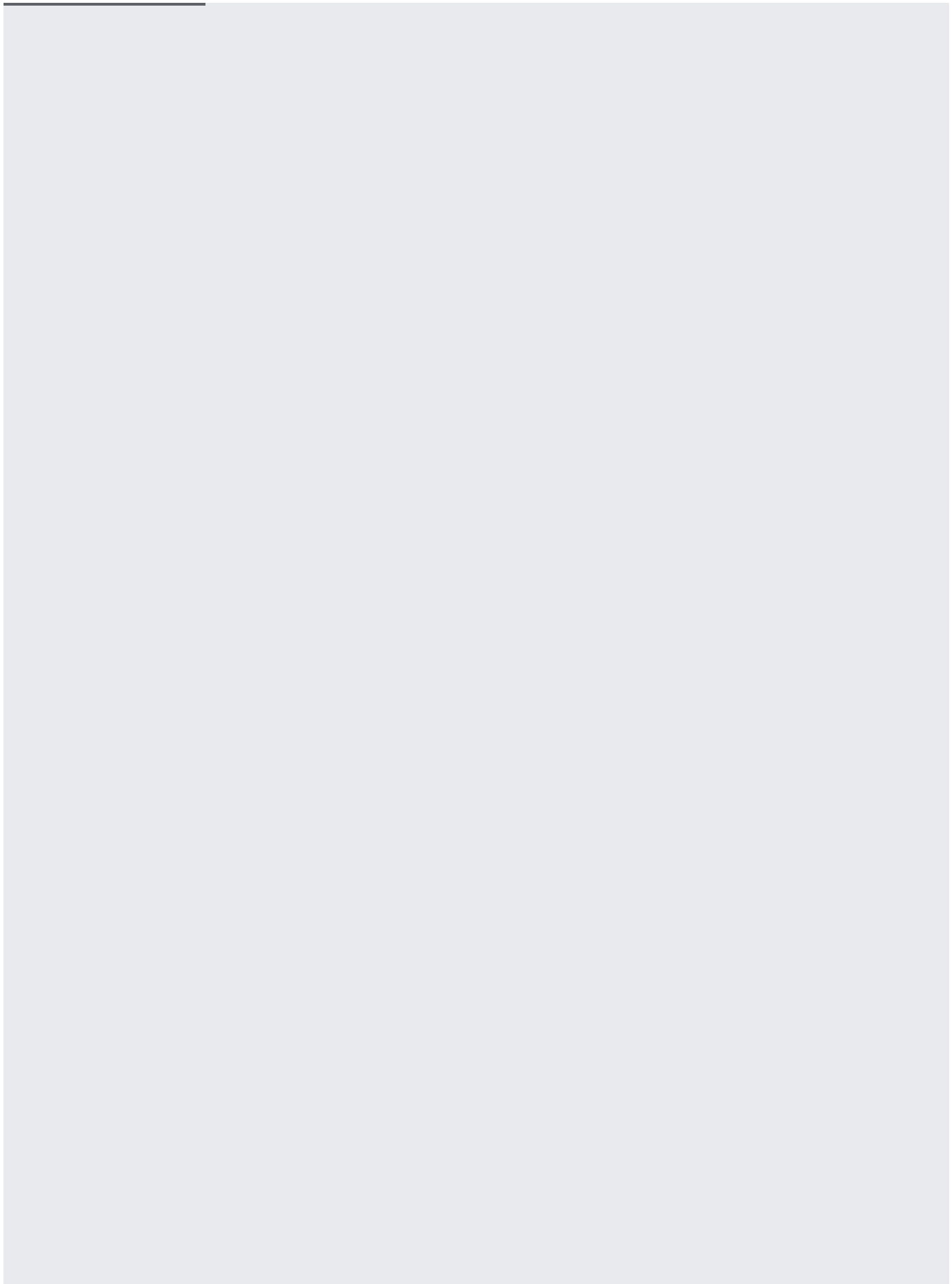


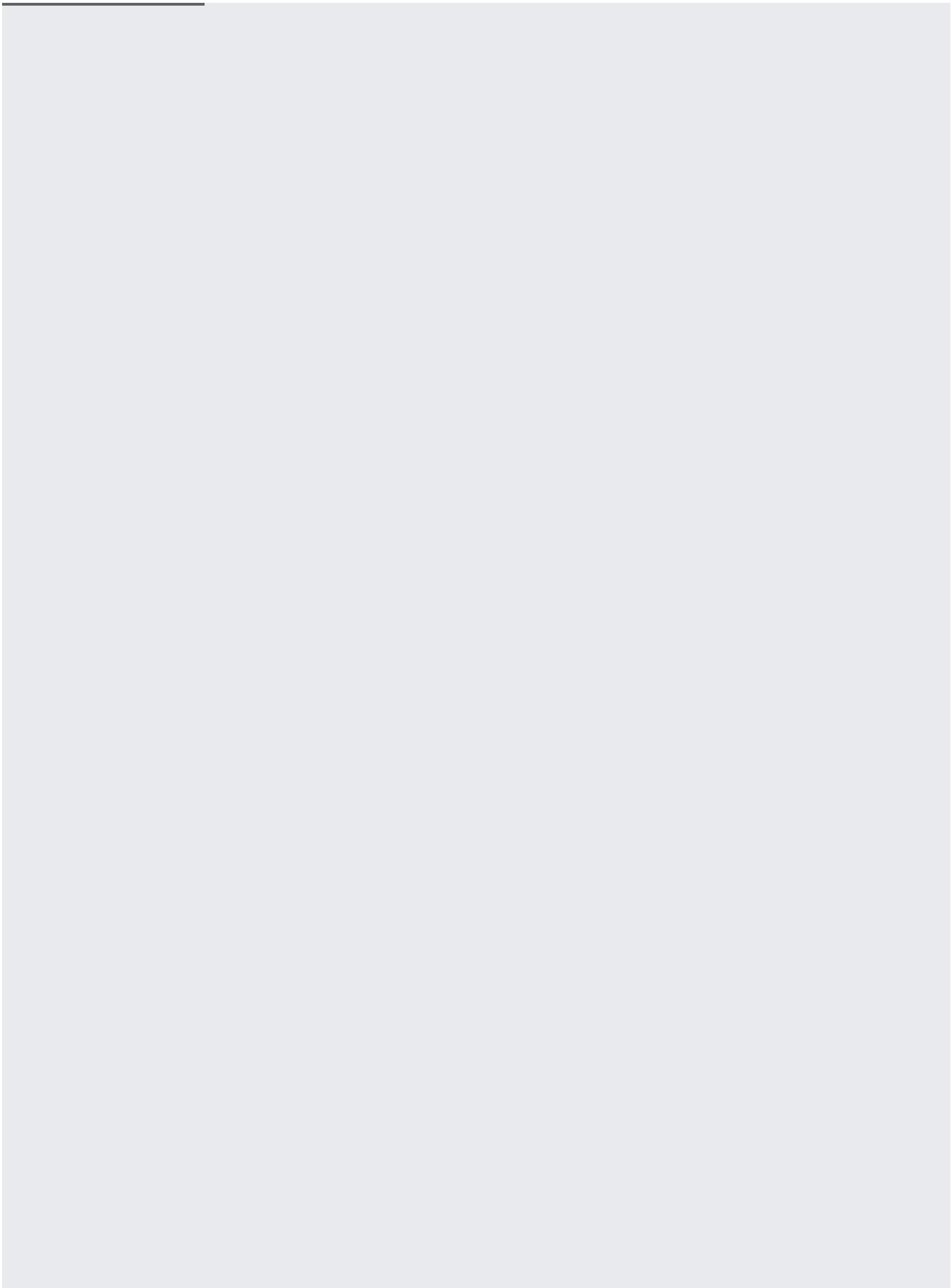


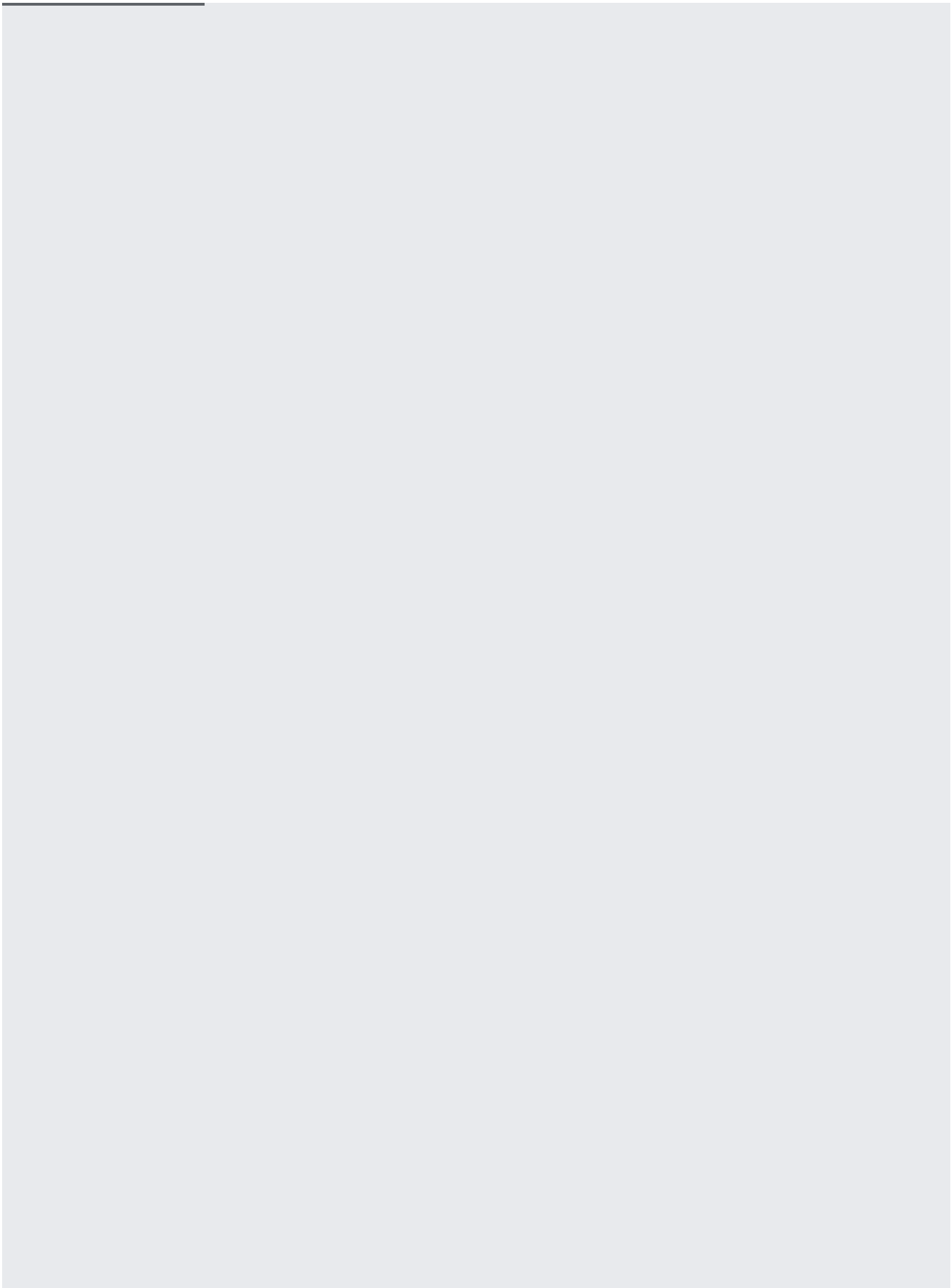


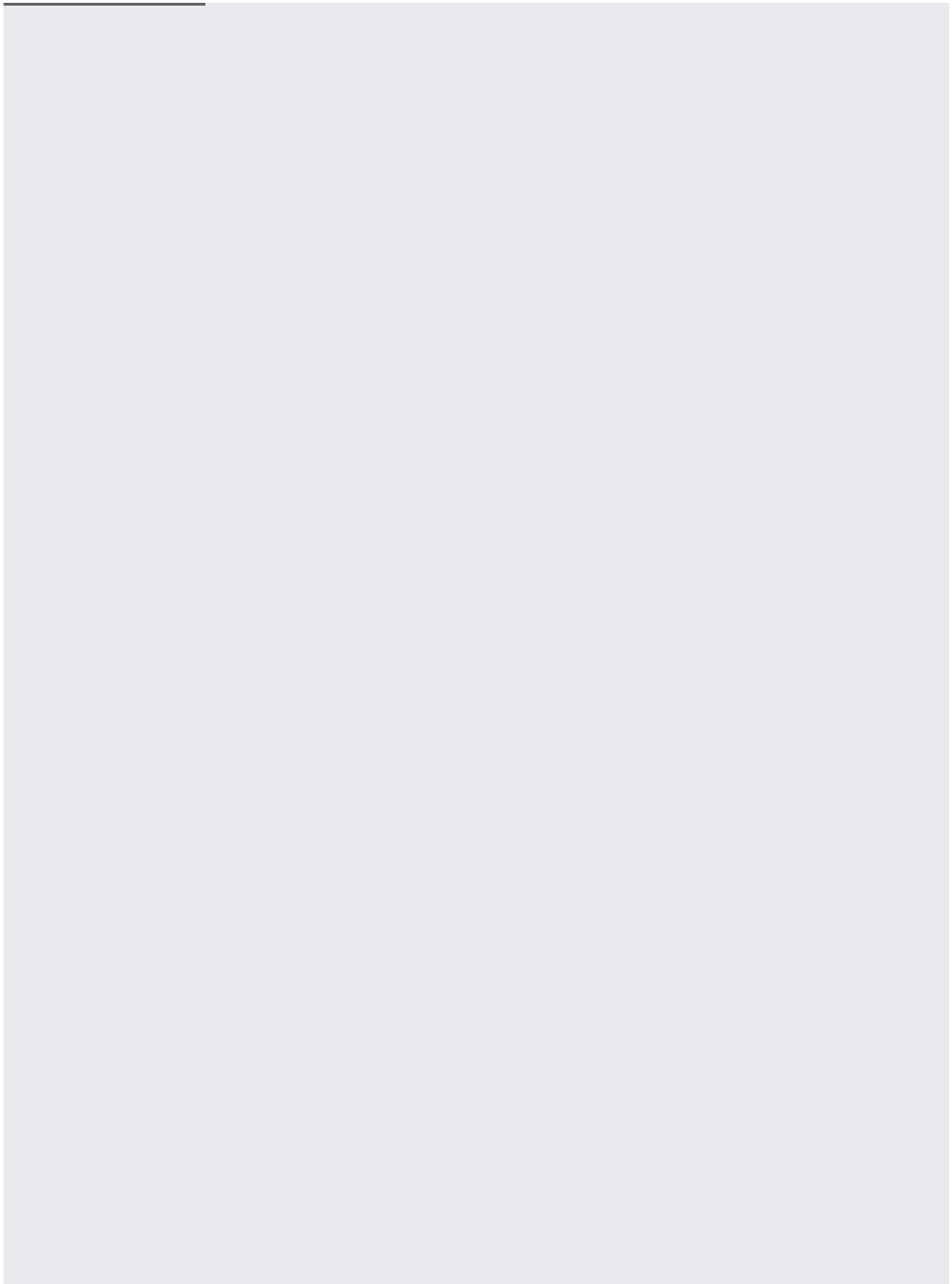


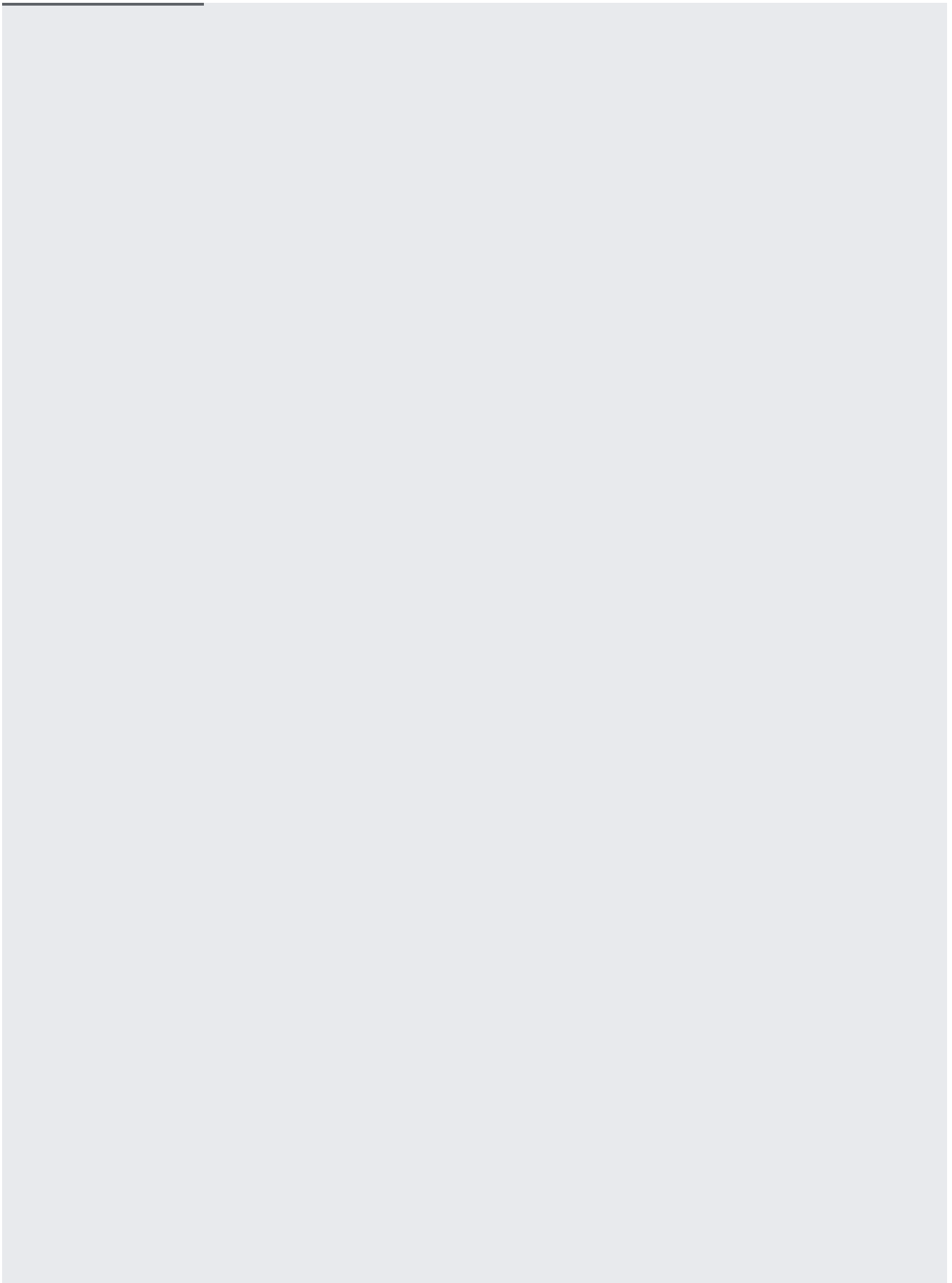


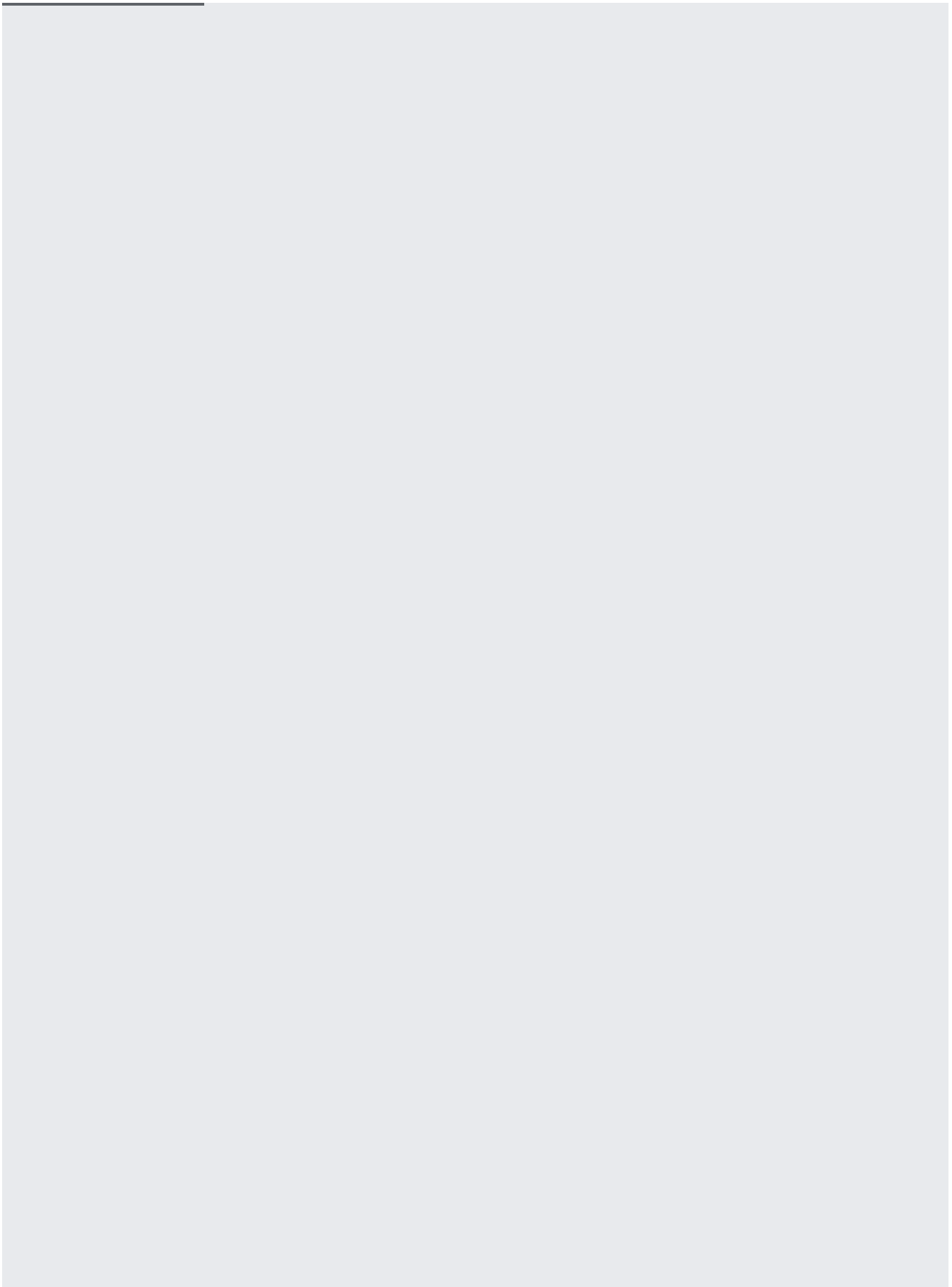




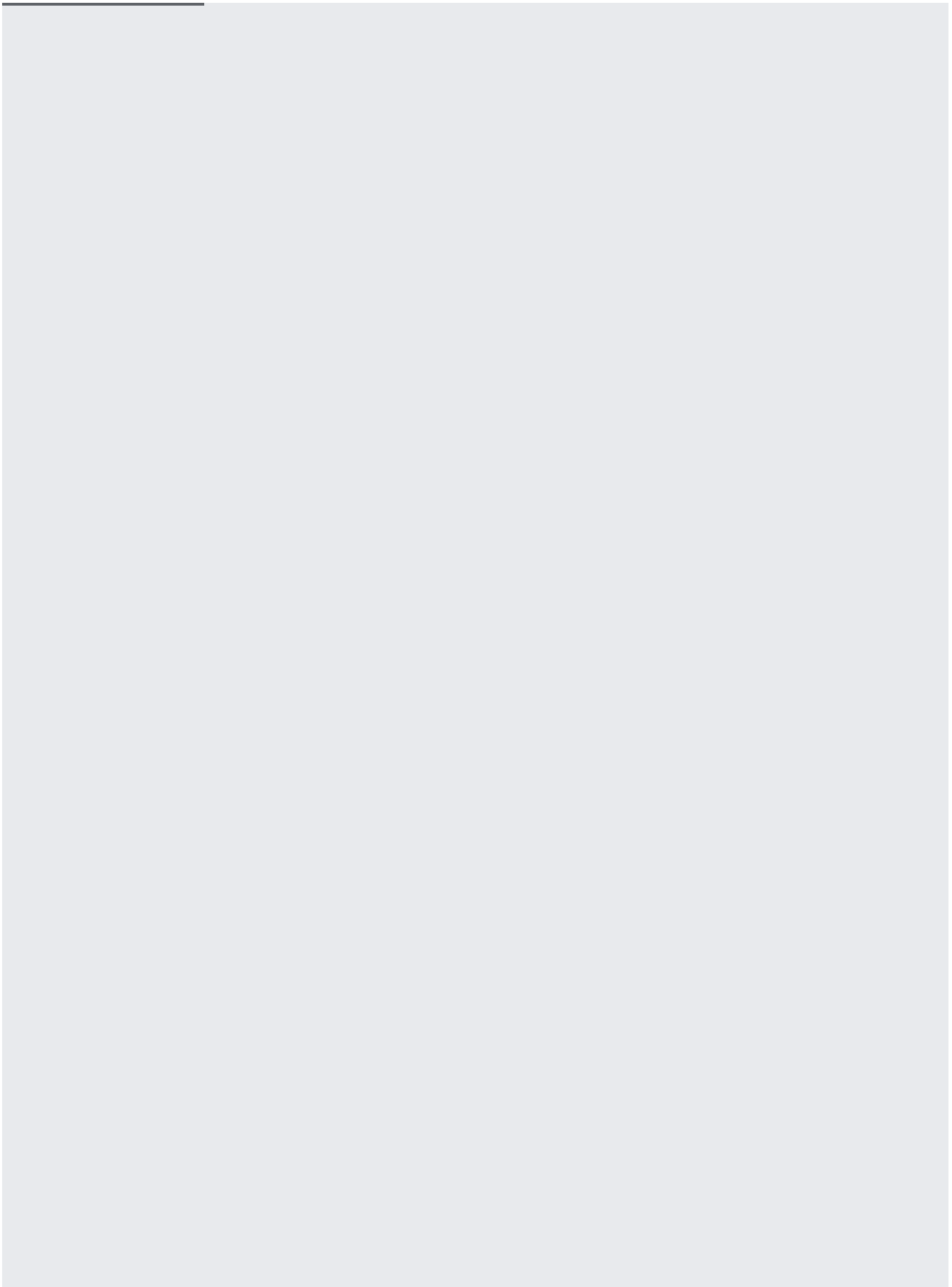


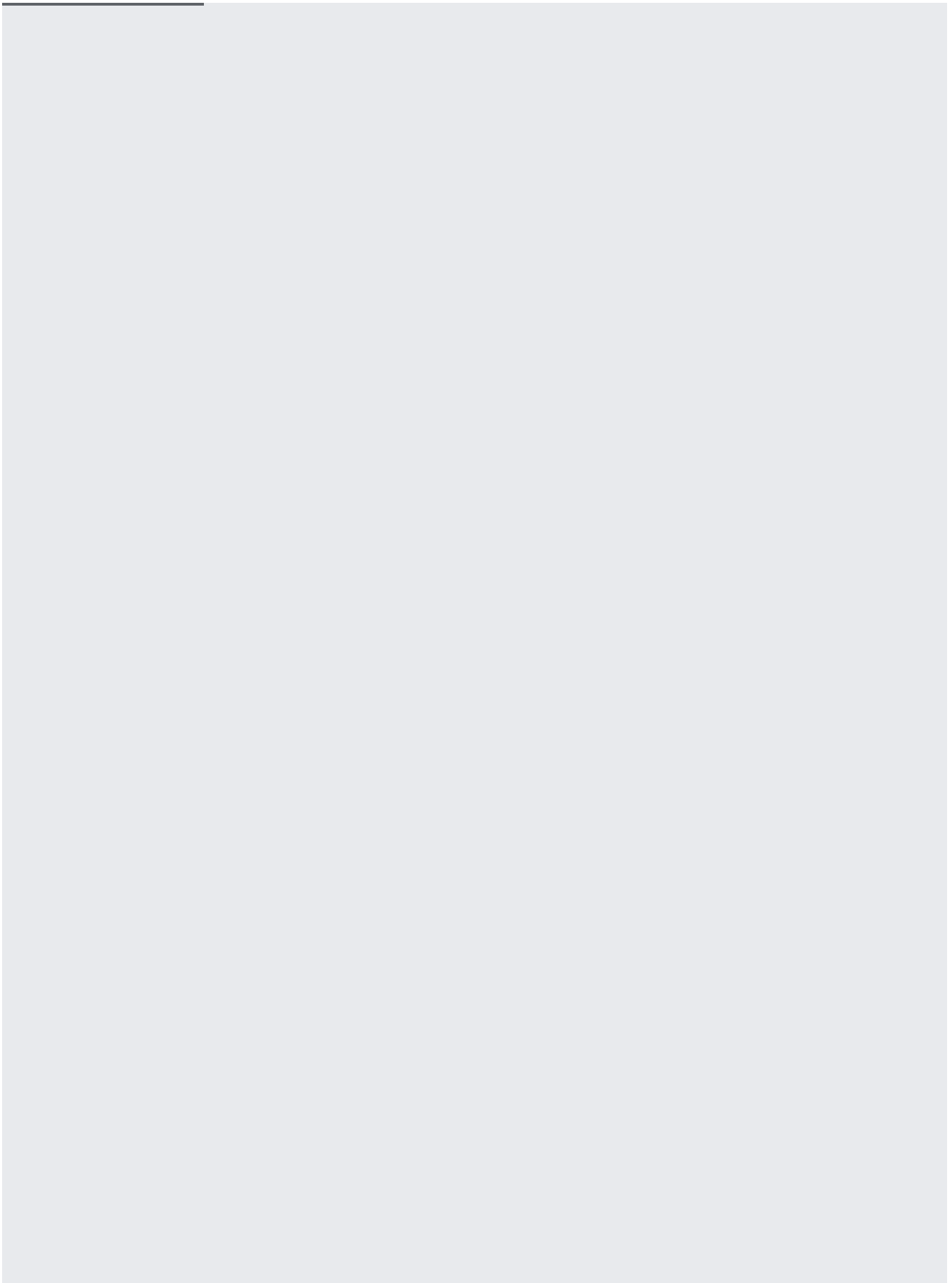


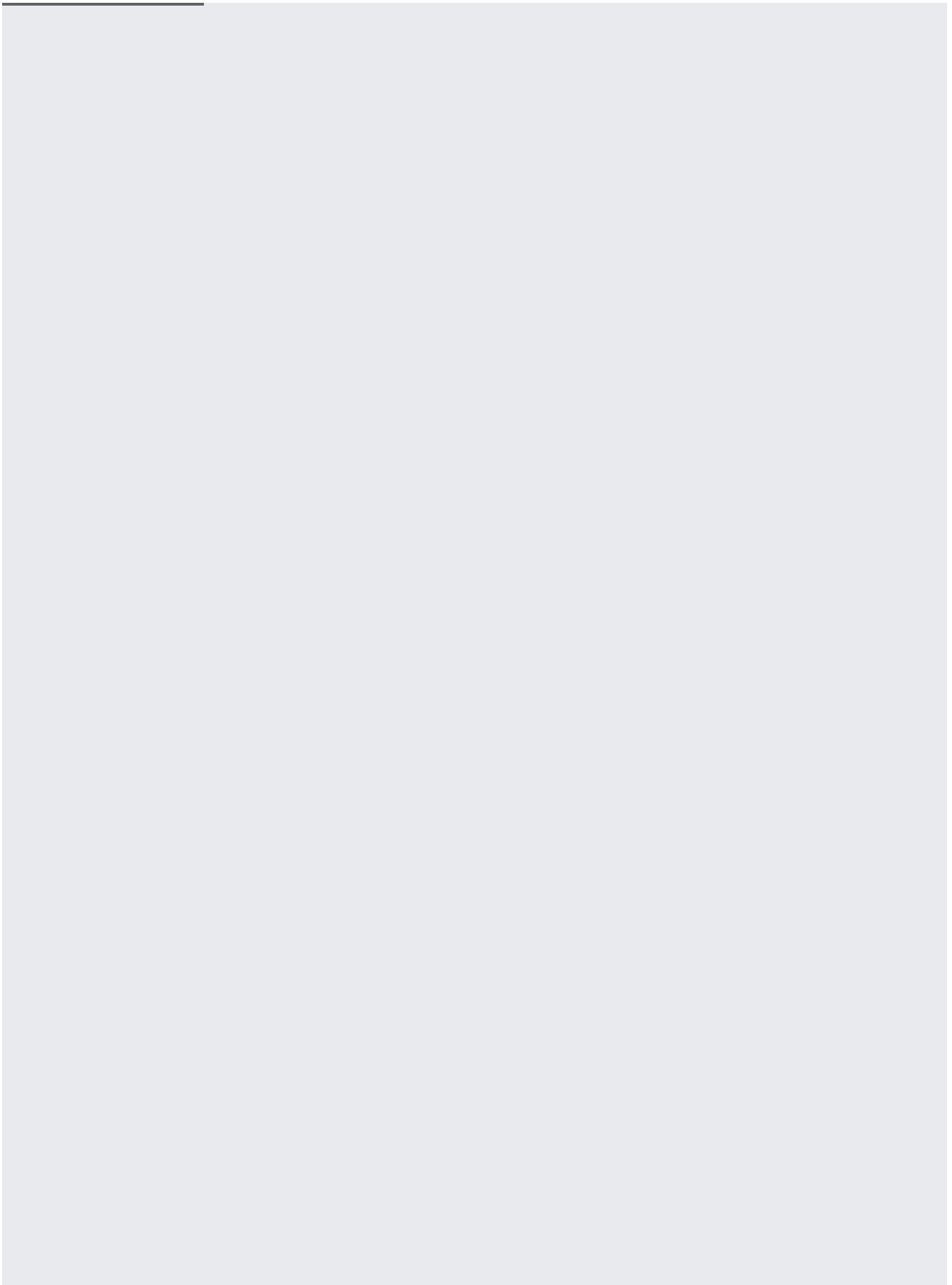


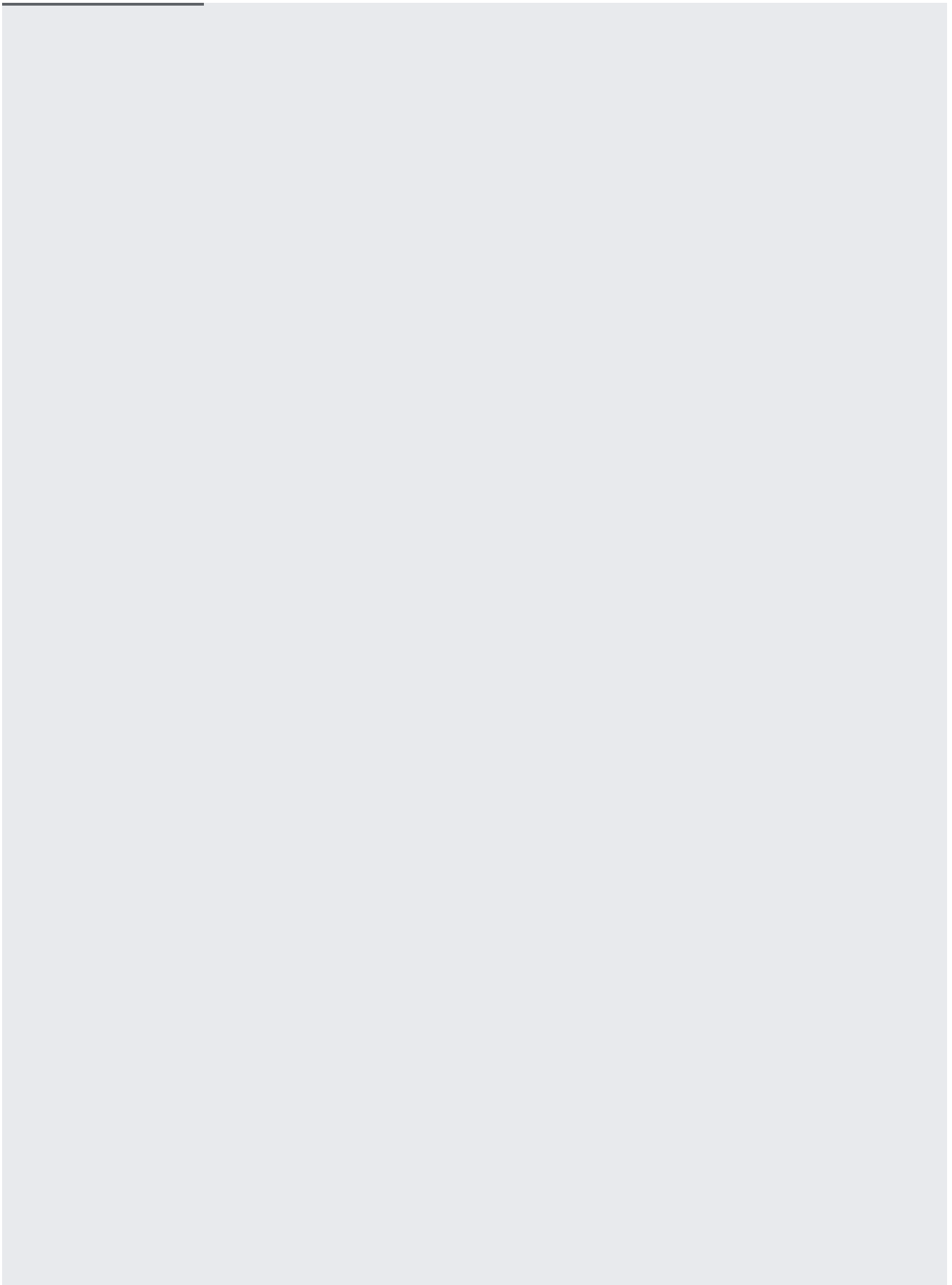


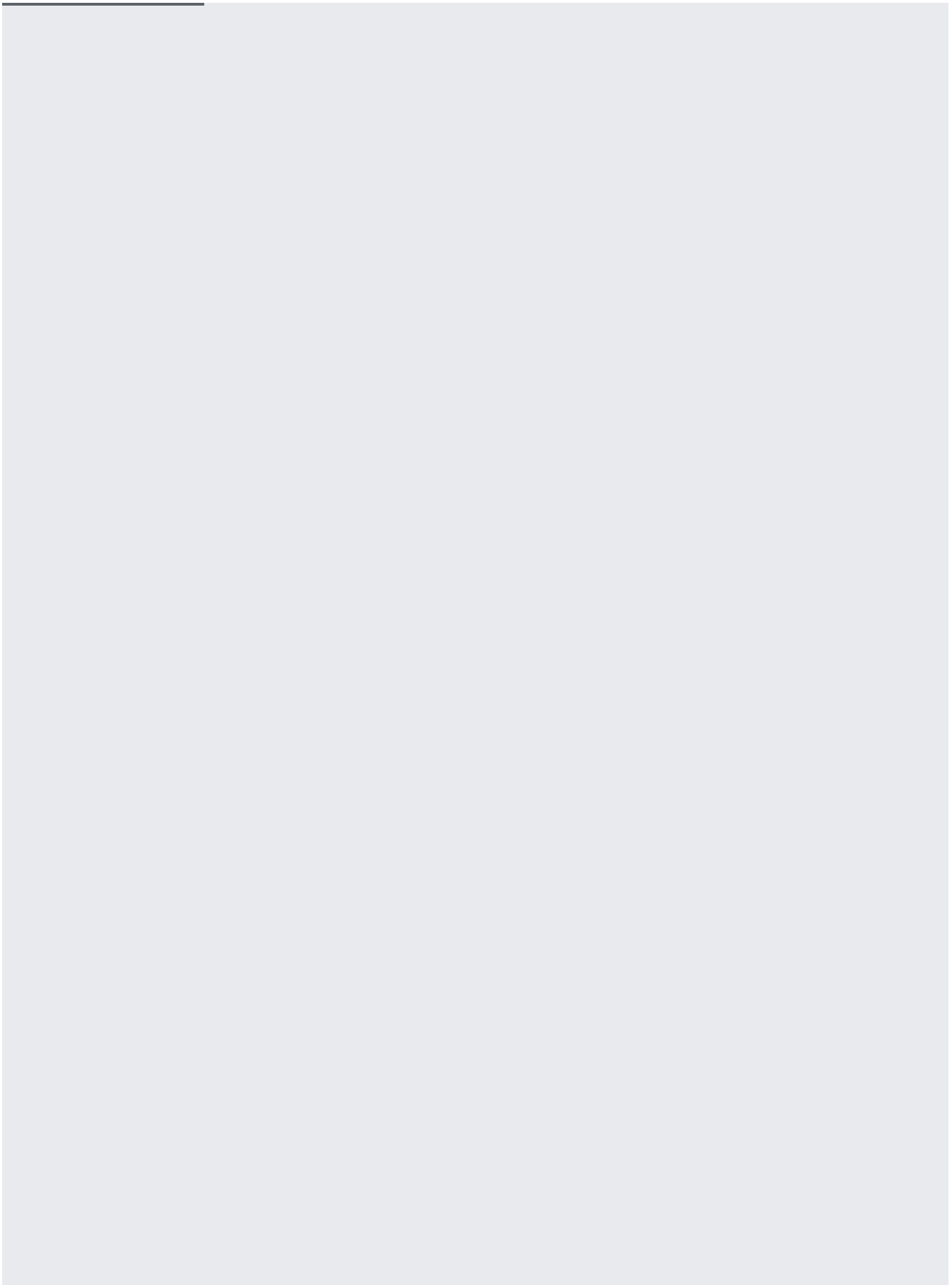


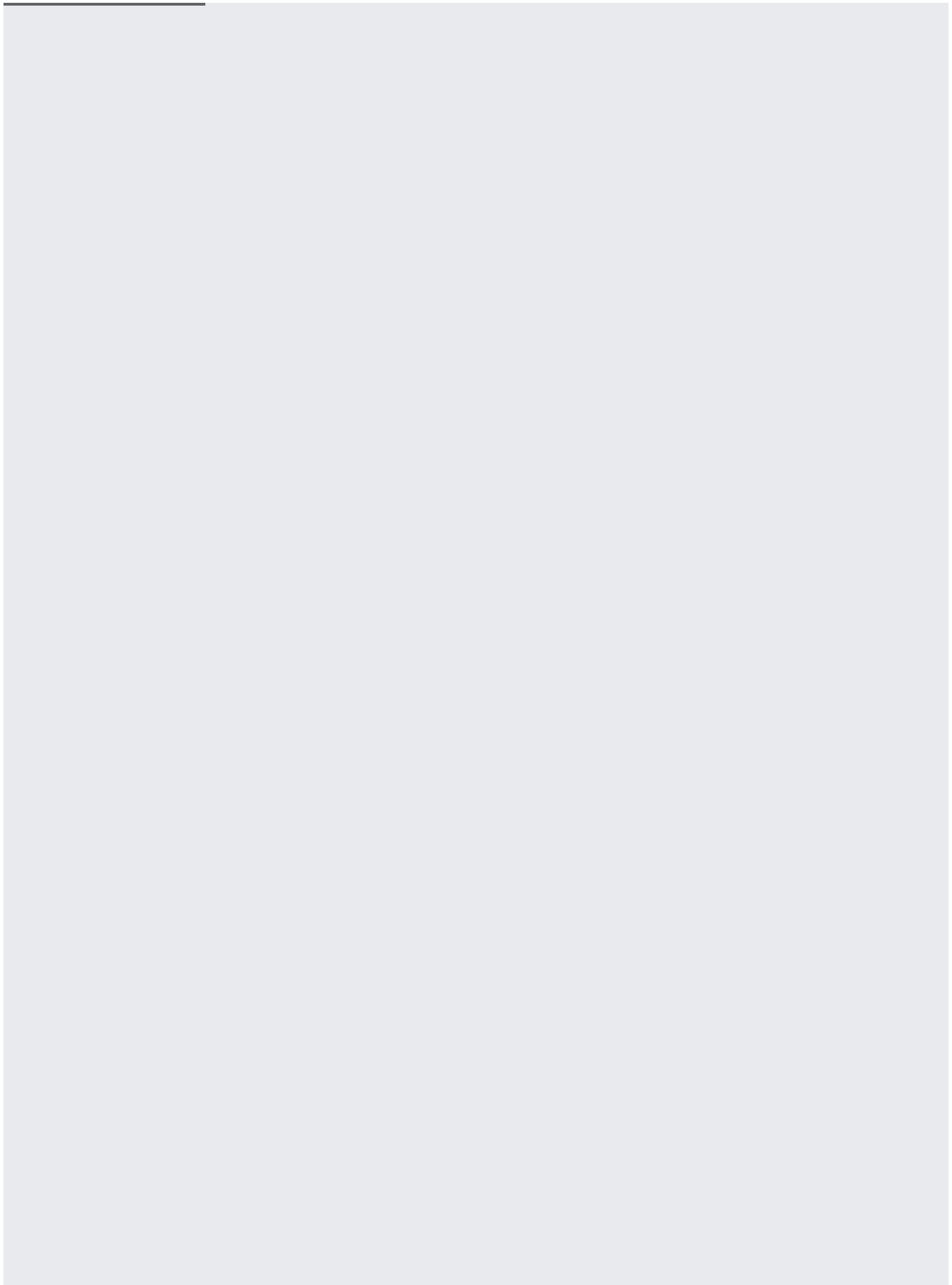


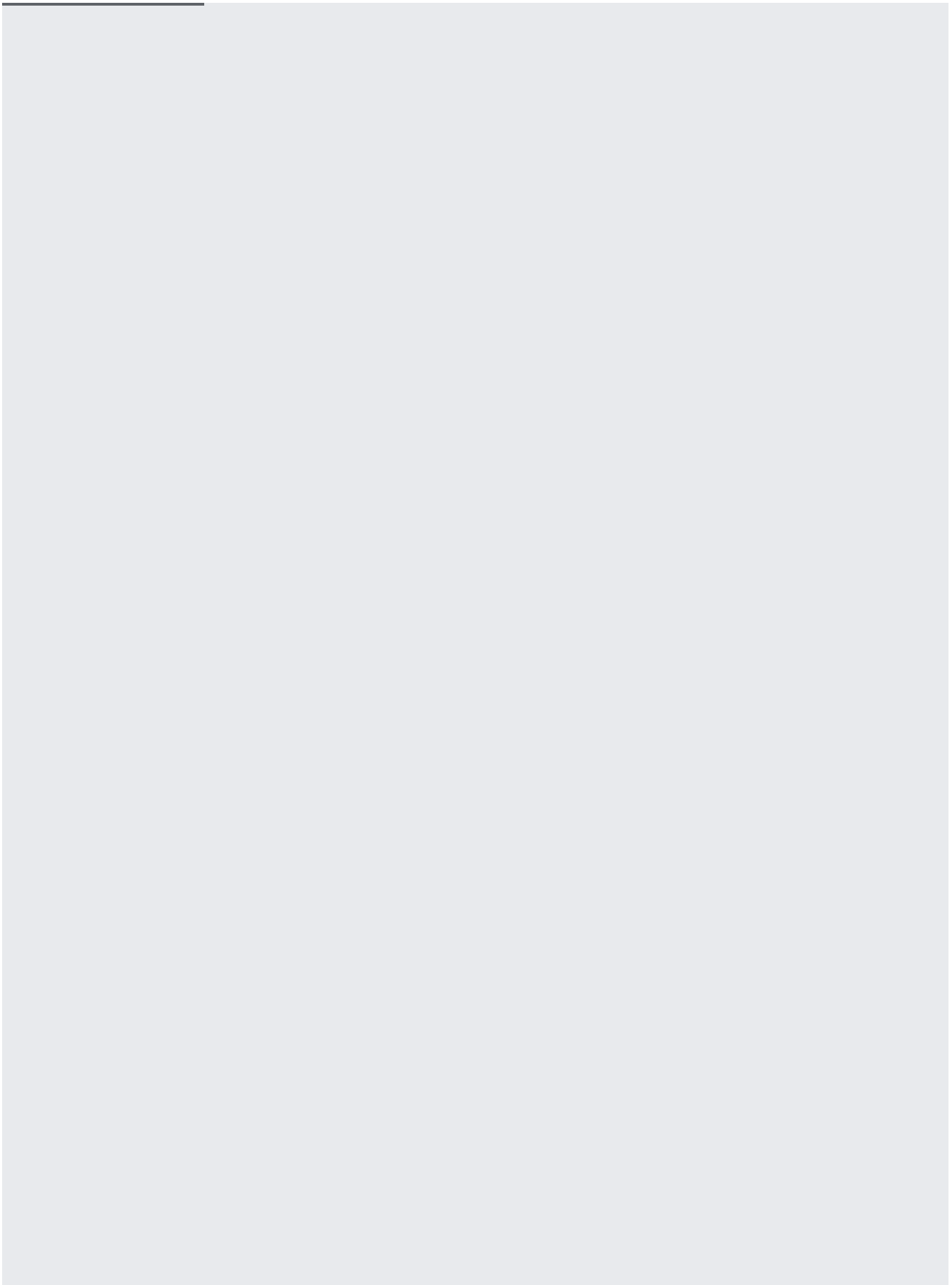


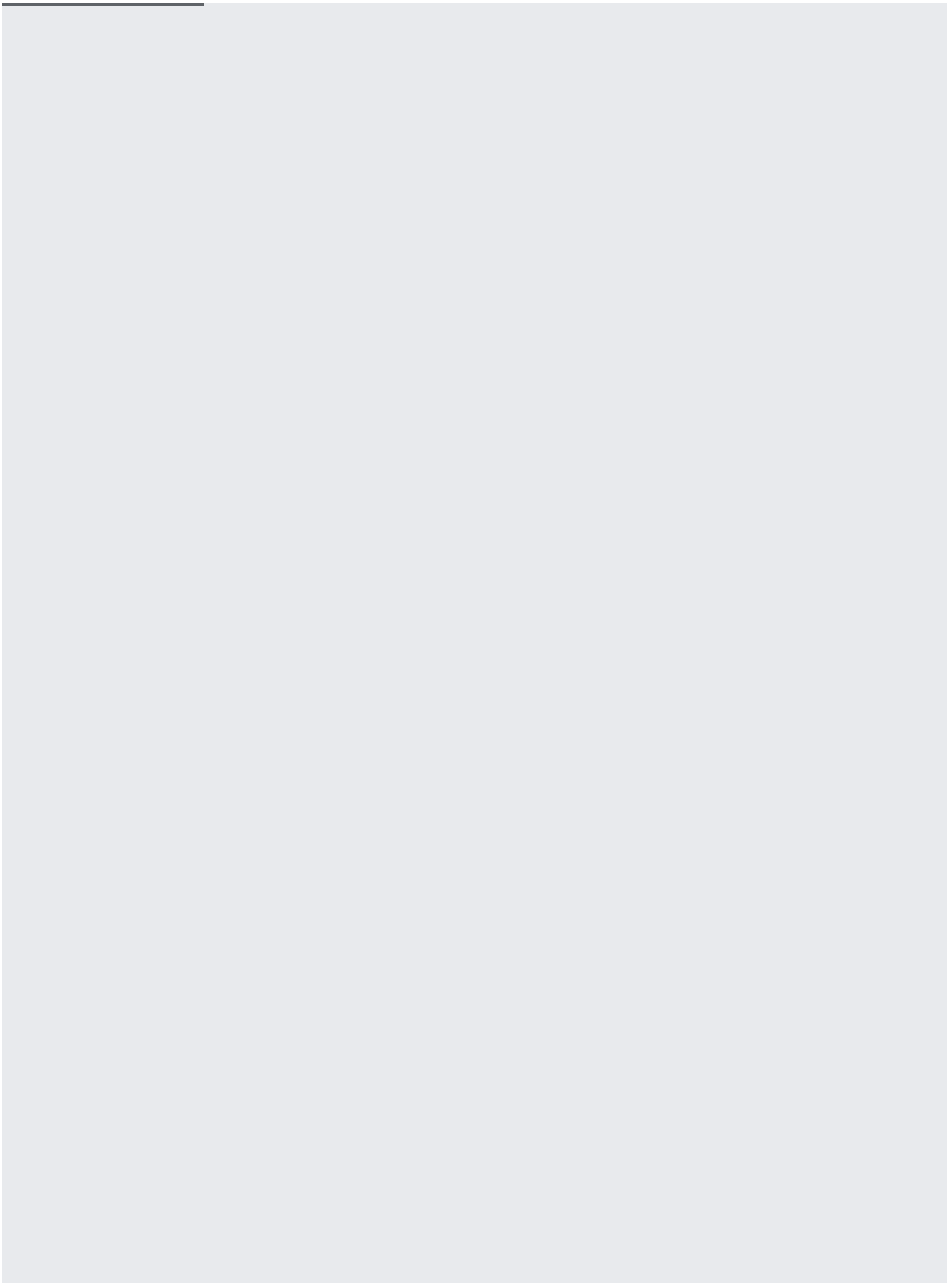




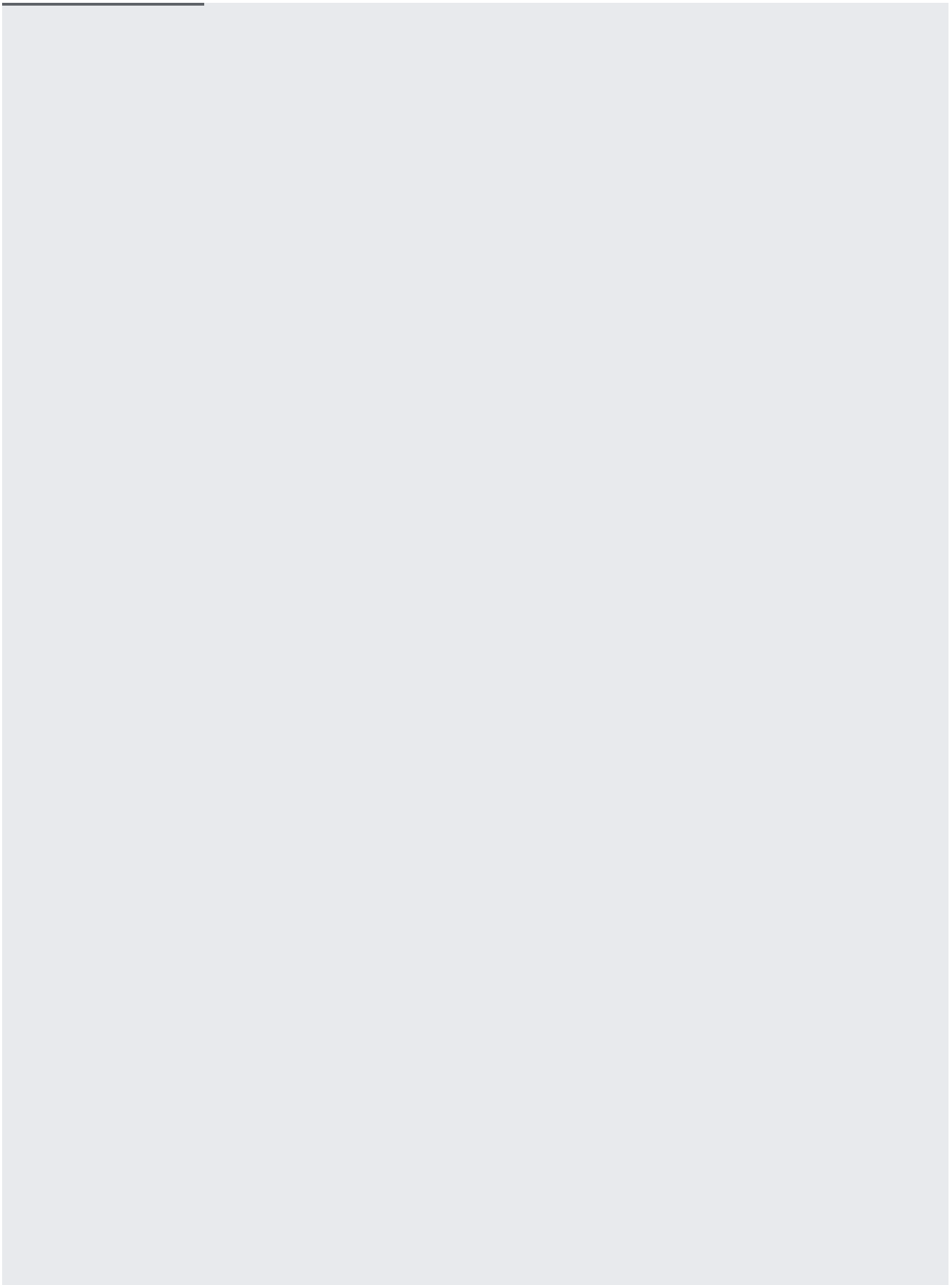












You can even have fields of `STRUCT` or `ARRAY<STRUCT>` type inside `STRUCT` values and access them similarly:

