Let’s Perform

# Model Validation

in Django Framework

# Introduction

In this particular post, we will going to see various options to perform validation on model fields or a complete model

# Model Validation

Basically, for model validation we mostly use these options in django :

1. clean method
2. clean_flelds method
3. Validators
4. Fields options

Let’s understand these options

# clean method

This method is used to perform validation on whole model fields.

```python
from django.core.exceptions import ValidationError
from django.db import models

class MyModel(models.Model):
    name = models.CharField(max_length=100)
    age = models.PositiveIntegerField()

    def clean(self):
        if self.age < 18:
            raise ValidationError("Age must be 18 or older.")
```

Here we have override the clean method and added the custom validation on age field. This should raise a ValidationError if validation fails, and same way we can write validation for other model fields.

# clean_fields method

This method is used to perform validation on all fields.

The optional exclude argument lets you provide

a set of field names to exclude from validation

```python
from django.core.exceptions import ValidationError
from django.db import models

class MyModel(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()

    def clean_fields(self, exclude=None):
        super().clean_fields(exclude=exclude)

        # Custom validation logic for the 'age' field
        if self.age < 0:
            raise ValidationError(
              {'age': 'Age must be a non-negative number.'})
```

Here we have override the clean_fields method and added the custom validation on age field. This should raise a ValidationError if validation fails.

Here in exclude argument we can pass the list of fields names to be exclude from validation

# Validators

Django allows you to define custom validators for model fields using the validators parameter. You can define a function that takes a value and raises a ValidationError if the value is invalid.

```python
from django.core.exceptions import ValidationError
from django.db import models

def validate_even(value):
    if value % 2 != 0:
        raise ValidationError("Value must be even.")

class Number(models.Model):
    even_number = models.IntegerField(validators=[validate_even])
```

Here in validators attribute, we can pass the list of validations

Here we define custom validator function, that takes a field value and raise a ValidationError, if validation fails

# Fields Options

Django provides several built-in field options for validation, such as :

1. null
2. blank
3. unique
4. max_length
5. min_length
6. choices

```python
from django.db import models

class MyModel(models.Model):
    name = models.CharField(max_length=100, null=True, blank=False)
    age = models.PositiveIntegerField()
    email = models.EmailField(unique=True)
```

# Conclusion

These are the some options that we could use to perform validation on our models and its fields.

Apart from that we could use full_clean, validate_unique, validate_constraints methods to add morevalidation on our model fields.

#pythonworld

For more django
related content

# Subscribe

To My Youtube
Channel

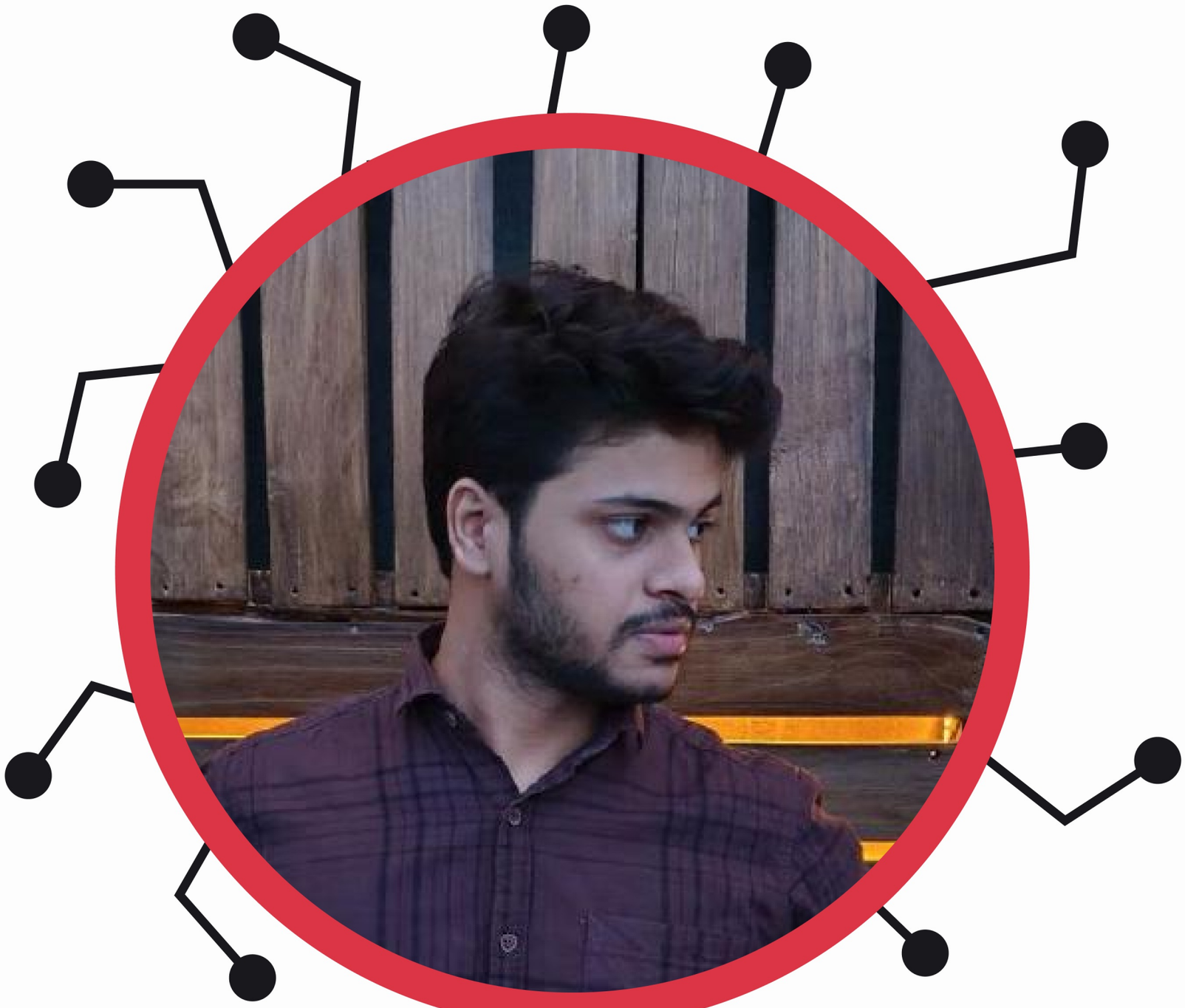
Link in bio

#pythonworld

# DID YOU FIND THIS

# HELPFUL

Let me know in the comment



Aashish Kumar

Software Engineer

Follow for more ❤️