

union

PanDera



O que é o Pandera

- Validação de tipos de dados
- Verificação de intervalos de valores e condições específicas
- Suporte para validações personalizadas através de funções definidas pelo usuário
- Teste de hipóteses
- Inferência de propriedades de um dataframe

O que é o Pandera

- Pandas
- FastAPI
- PySpark
- Polars
- Geopandas
- Modin



Base de dados a ser analisado

- [Dataset](#)
- Escolhida por ter múltiplos campos que possam ser feito diferentes validações padrões assim como aplicar algumas possíveis 'regras de negócio'

Id	age	sex	bmi	children	smoker	region	charges
1	19	female	27.900	0	yes	southwest	16884.92400
2	18	male	33.770	1	no	southeast	1725.55230
3	28	male	33.000	3	no	southeast	4449.46200
4	33	male	22.705	0	no	northwest	21984.47061
5	32	male	28.880	0	no	northwest	3866.85520
...
1334	50	male	30.970	3	no	northwest	10600.54830
1335	18	female	31.920	0	no	northeast	2205.98080
1336	18	female	36.850	0	no	southeast	1629.83350
1337	21	female	25.800	0	no	southwest	2007.94500
1338	61	female	29.070	0	yes	northwest	29141.36030

Verificação de DataFrames

- Várias maneiras de fazer verificações
- Usando DataFrameSchema é uma das mais comuns de longe

Tipo que o ~~GPT~~ Google entrega se tu só pedir para usar pandera

```
schema = pa.DataFrameSchema({
    "Id": Column(int, checks=Check.gt(0), nullable=False, unique=True),
    "age": Column(int, checks=Check.ge(0), nullable=False),
    "sex": Column(str, checks=Check.isin(["female", "male"])),
    "bmi": Column(float),
    "children": Column(int, checks=Check.ge(0)),
    "smoker": Column(str, checks=Check.isin(["yes", "no"])),
    "region": Column(str, checks=Check.isin(["southwest", "southeast", "northwest", "northeast"])),
    "charges": Column(float, checks=Check.ge(0))
})

validated_df = schema.validate(df)
print(validated_df)
```

✓ 0.0s

	Id	age	sex	bmi	children	smoker	region	charges
0	1	19	female	27.900	0	yes	southwest	16884.92400
1	2	18	male	33.770	1	no	southeast	1725.55230
2	3	28	male	33.000	3	no	southeast	4449.46200
3	4	33	male	22.705	0	no	northwest	21984.47061
4	5	32	male	28.880	0	no	northwest	3866.85520
...
1333	1334	50	male	30.970	3	no	northwest	10600.54830
1334	1335	18	female	31.920	0	no	northeast	2205.98080
1335	1336	18	female	36.850	0	no	southeast	1629.83350
1336	1337	21	female	25.800	0	no	southwest	2007.94500
1337	1338	61	female	29.070	0	yes	northwest	29141.36030

[1338 rows x 8 columns]

DataFrameModel

- Maneira de fazer validação mais orientado a POO
- Mais fácil iterar sobre utilizando heranças e propriedades de classes
- Melhor legibilidade na minha opinião
- Maior flexibilidade em inserir checks no código através de decorator

Possui uma versão similar que usa SchemaModel com leves diferenças

```
class MedCost(DataFrameModel):  
    Id: Series[int] = Field(gt=0, nullable=False, unique=True)  
    age: Series[int] = Field(ge=0, nullable=False)  
    sex: Series[str] = Field(isin=["female", "male"])  
    bmi: Series[float]  
    children: Series[int] = Field(ge=0)  
    smoker: Series[str] = Field(isin=["yes", "no"])  
    region: Series[str] = Field(isin=["southwest", "southeast", "northwest", "northeast"])  
    charges: Series[float] = Field(ge=0)
```

Passo a passo

Definição simples de comportamentos das colunas da classe

Verificação de valores assim como assegurar que colunas podem ou não ser nullas, ou únicas

```
class MedCost(DataFrameModel):  
    Id: Series[int]  
    age: Series[int]  
    sex: Series[str]  
    bmi: Series[float]  
    children: Series[int]  
    smoker: Series[str]  
    region: Series[str]  
    charges: Series[float]
```

0.0s

```
class MedCost(DataFrameModel):  
    Id: Series[int] = Field(gt=0, nullable=False, unique=True)  
    age: Series[int] = Field(ge=0, nullable=False)  
    sex: Series[str] = Field(isin=["female", "male"])  
    bmi: Series[float]  
    children: Series[int] = Field(ge=0)  
    smoker: Series[str] = Field(isin=["yes", "no"])  
    region: Series[str] = Field(isin=["southwest", "southeast", "northwest", "northeast"])  
    charges: Series[float] = Field(ge=0)
```

Aplicação de regra de negócio

- Precisamos fazer uma checagem sobre uma coluna ou sobre o dataframe como um todo, que é derivado da regra de negócio da aplicação

Neste caso queremos calcular se o custo da cobrança foi justa baseado em um cálculo utilizando idade e IMC(coluna bmi)

```
def validate_charges(csl, df: pd.DataFrame) -> pd.Series:  
    min_charges = (df['age'] + df['bmi']) * 15  
    max_charges = (df['age'] + df['bmi']) * 850  
    return df['charges'].between(min_charges, max_charges)
```

Agora falta integrar ele no DataFrameModel existente

Integração

```
class MedCost(DataFrameModel):
    Id: Series[int] = Field(gt=0, nullable=False, unique=True)
    age: Series[int] = Field(ge=0, nullable=False)
    sex: Series[str] = Field(isin=["female", "male"])
    bmi: Series[float]
    children: Series[int] = Field(ge=0)
    smoker: Series[str] = Field(isin=["yes", "no"])
    region: Series[str] = Field(isin=["southwest", "southeast", "northwest", "northeast"])
    charges: Series[float] = Field(ge=0)

    @pa.dataframe_check
    def validate_charges(csl, df: pd.DataFrame) -> pd.Series:
        min_charges = (df['age'] + df['bmi']) * 15
        max_charges = (df['age'] + df['bmi']) * 850
        return df['charges'].between(min_charges, max_charges)
```

```
class MedCost(DataFrameModel):
    Id: Series[int] = Field(gt=0, nullable=False, unique=True)
    age: Series[int] = Field(ge=0, nullable=False)
    sex: Series[str] = Field(isin=["female", "male"])
    bmi: Series[float]
    children: Series[int] = Field(ge=0)
    smoker: Series[str] = Field(isin=["yes", "no"])
    region: Series[str] = Field(isin=["southwest", "southeast", "northwest", "northeast"])
    charges: Series[float] = Field(ge=0)

    @pa.check("bmi", name="check_bmi")
    def check_bmi(cls, bmi : Series[float]) -> Series[bool]:
        return bmi < 100

    @pa.dataframe_check
    def validate_charges(csl, df: pd.DataFrame) -> pd.Series:
        min_charges = (df['age'] + df['bmi']) * 15
        max_charges = (df['age'] + df['bmi']) * 850
        return df['charges'].between(min_charges, max_charges)
```

Validação

```
MedCost.validate(df)
```

✓ 0.0s

	id	age	sex	bmi	children	smoker	region	charges
0	1	19	female	27.900	0	yes	southwest	16884.92400
1	2	18	male	33.770	1	no	southeast	1725.55230
2	3	28	male	33.000	3	no	southeast	4449.46200
3	4	33	male	22.705	0	no	northwest	21984.47061
4	5	32	male	28.880	0	no	northwest	3866.85520
...
1333	1334	50	male	30.970	3	no	northwest	10600.54830
1334	1335	18	female	31.920	0	no	northeast	2205.98080
1335	1336	18	female	36.850	0	no	southeast	1629.83350
1336	1337	21	female	25.800	0	no	southwest	2007.94500
1337	1338	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 8 columns

SchemaError: DataFrameSchema 'MedCost' failed element-wise validator number 0: <Check validate_charges> failure cases: 1, 2, 4, 10, 11, 12, 13, 15, 16, 18, 20, Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

Utilizando Decorator em uma função

```
@check_types
def select_males(df: DataFrame[MedCost]) -> DataFrame[MedCost]:
    males_df = df[df[MedCost.sex] == 'male']
    return males_df
```

Faz a verificação do dataframe juntamente da função selecionada, todo o processo de validação é embutido no decorator @check_types

- Através de decorator se acopla verificação a qualquer função
- Regras de validação podem ser tão simples ou complexas quanto for necessário adicionando checks customizáveis

Iterando através de herança

```
class FemaleMedCost(MedCost):
    charges: Series[float] = Field(ge=0)

    @pa.dataframe_check
    def validate_charges(csl, df: pd.DataFrame) -> pd.Series:
        max_charges = df["children"] * 1000 + 5000
        return df["charges"] <= max_charges
```

```
@check_types
def select_females(df: DataFrame[FemaleMedCost]) -> DataFrame[FemaleMedCost]:
    females_df = df[df[FemaleMedCost.sex] == 'female']
    return females_df
```

✓ 0.0s

Teste de hipótese

- Método padrão
- Suportado pela documentação do pandera
- Recomendado para quem gostou desta forma de anotação

```
# Sample DataFrame
df = pd.DataFrame({
    "height_in_feet": [6.5, 7, 6.1, 5.1, 4],
    "sex": ["M", "M", "F", "F", "F"]
})

# Define the schema with the hypothesis test
schema = DataFrameSchema({
    "height_in_feet": Column(
        float, [
            Hypothesis.two_sample_ttest(
                sample1="M",
                sample2="F",
                groupby="sex",
                relationship="greater_than",
                alpha=0.05,
                equal_var=True
            ),
        ]
    ),
    "sex": Column(str)
})

# Validate the DataFrame against the schema
try:
    schema.validate(df)
except pa.errors.SchemaError as exc:
    print(exc)
```

0.0s

- Podemos ter o mesmo comportamento através de checks

```
from scipy.stats import ttest_ind

class SmokerMedCost(MedCost):

    @pa.dataframe_check
    def validate_charges(cls, df: pd.DataFrame) -> pd.Series:
        return True

    @pa.dataframe_check(name="smoker_vs_non_smoker_charges")
    def validate_smoker_charges(cls, df: pd.DataFrame) -> Series[bool]:
        smokers = df[df["smoker"] == "yes"]["charges"]
        non_smokers = df[df["smoker"] == "no"]["charges"]

        t_stat, p_value = ttest_ind(smokers, non_smokers, equal_var=False, alternative="greater")

        # Reject the null hypothesis if p-value < 0.05, indicating smokers have higher charges
        return pd.Series([p_value < 0.05])
```

Inferência de Schema de um dataframe

```
import pandas as pd
from pandera import DataFrameModel, Field, check_types, infer_schema, Check
from pandera.typing import DataFrame, Series

df = pd.read_csv("medical_cost.csv")

schema_script = infer_schema(df).to_script()

with open("schema_med.py", "w") as file:
    file.write(schema_script)
```

- Sempre gera um schema a qual o df passa na validação
- Fortemente recomendado alterar o schema antes de usar em outro df
- Gera schemas que utilizam DataFrameSchema

Parsers

```
class MaleMedCost(MedCost):  
    @pa.parser("charges")  
    def add_tax(cls, series):  
        return series + 200  
  
@check_types  
def select_males(df: DataFrame[MedCost]) -> DataFrame[MaleMedCost]:  
    males_df = df[df["sex"] == 'male']  
    return males_df  
  
# Example usage  
select_males(df).head(5)
```

Antes

Depois

Id	age	sex	bmi	children	smoker	region	charges
2	18	male	33.770	1	no	southeast	1725.55230
3	28	male	33.000	3	no	southeast	4449.46200
4	33	male	22.705	0	no	northwest	21984.47061
5	32	male	28.880	0	no	northwest	3866.85520
9	37	male	29.830	2	no	northeast	6406.41070

Id	age	sex	bmi	children	smoker	region	charges
2	18	male	33.770	1	no	southeast	1925.55230
3	28	male	33.000	3	no	southeast	4649.46200
4	33	male	22.705	0	no	northwest	22184.47061
5	32	male	28.880	0	no	northwest	4066.85520
9	37	male	29.830	2	no	northeast	6606.41070

Material de Apoio

- [Repositório da apresentação](#)
- [Documentação](#)

Se precisar estamos aí!!