

# 1. Introducere

## a. Prezentarea setului de date

În cadrul acestui proiect, am ales să lucrez cu setul de date **The Human Freedom Index**, oferit de **Cato Institute** și **Fraser Institute**, disponibil pe platforma Kaggle. Acest set de date evaluează și compară nivelul libertății umane în peste 160 de țări, pe o perioadă extinsă de 21 de ani (2000–2020), și reprezintă una dintre cele mai complete resurse globale privind libertățile personale și economice.

Setul este structurat la nivel **anual**, iar fiecare rând corespunde unei țări într-un anumit an. Sunt incluse peste **140 de variabile**, organizate în jurul a trei dimensiuni principale:

- **Libertăți personale** – exprimare, religie, identitate, libertatea de mișcare, asocieri, siguranță personală;
- **Libertăți economice** – comerț internațional, reglementări guvernamentale, impozite, investiții;
- **Instituții legale** – statul de drept, justiție procedurală, integritate, libertăți constituționale.

Printre cele mai relevante coloane regăsim:

- `countries` , `region` , `year` – pentru localizarea geografică și temporală;
- `hf_score` , `hf_rank` – scorul total al libertății și poziția în clasamentul global;
- numeroși sub-indicatori prefixați cu `pf_` (personal freedom), `ef_` (economic freedom) și `hf_` (human freedom), care oferă o granularitate ridicată în analiza libertății.

Setul conține **3465 de înregistrări** și **141 de coloane**, iar datele sunt în mare parte numerice, dar conțin și variabile categorice. De asemenea, există un procent semnificativ de **valori lipsă**, care vor necesita preprocesare atentă în etapele ulterioare ale proiectului.

**Link către setul de date:** <https://www.kaggle.com/datasets/gsutters/the-human-freedom-index>

---

## b. Enunțarea obiectivelor

Obiectivul general al acestui proiect este construirea unei aplicații de analiză Big Data care să prelucreze, modeleze și interpreteze date globale despre libertatea umană.

Proiectul îmbină componente de prelucrare distribuită, Machine Learning, Deep Learning și Streaming în timp real – pentru a arăta cum pot fi extrase insight-uri relevante dintr-un set de date socio-politice complexe.

### Obiective specifice:

1. **Explorarea și curățarea** datelor folosind PySpark și Spark SQL;
2. **Analiza exploratorie** a distribuției libertăților în funcție de regiuni și ani;
3. **Modelarea predictivă**: antrenarea unor algoritmi de Machine Learning pentru a prezice `hf_score` pe baza sub-indicatorilor ( `pf_` , `ef_` );
4. **Construcția unui pipeline ML complet** cu Spark MLlib;
5. **Antrenarea unui model de Deep Learning** (cu TensorFlow) ca alternativă de predicție;
6. **Simularea unui flux de date în timp real**, integrat cu predicția automată, prin Spark Streaming și un model ML salvat.

Prin aceste obiective, proiectul nu doar îndeplinește cerințele academice, ci demonstrează și aplicabilitatea tehnologiilor Big Data în domeniul analizei sociale și a politicilor publice.

## 2. Procesarea datelor cu Spark SQL și DataFrames

### 2.1 Curățarea și pregătirea inițială a datelor

#### Obiectiv

În această primă etapă de procesare, ne propunem să transformăm datele brute într-un format curat și consistent, care să poată fi utilizat ulterior în analize și modele. Pentru aceasta, vom folosi funcționalitățile oferite de PySpark, atât prin API-ul DataFrame, cât și prin Spark SQL.

Mai exact, vom:

- elimina coloanele complet goale sau cu prea multe valori lipsă;
- converti tipurile de date relevante;
- păstra doar rândurile esențiale pentru analiză ( `year` , `hf_score` , `countries` , `region` );
- verifica starea datasetului după curățare.

```
In [35]: from pyspark.sql import SparkSession
from pyspark.sql.functions import col, isnan, when, count
from pyspark.sql.types import DoubleType, IntegerType
```

```
In [36]: # 1. Inițializăm SparkSession
spark = SparkSession.builder \
    .appName("HumanFreedomIndexProcessing") \
    .getOrCreate()
```

```
In [37]: # 2. Citim setul de date
df = spark.read.csv("dataset/hfi_cc_2022.csv", header=True, inferSchema=True)
print(f"Dimensiune inițială: {df.count()} rânduri, {len(df.columns)} coloane")
```

Dimensiune inițială: 3465 rânduri, 141 coloane

```
In [38]: # 3. Eliminăm coloanele complet nule
null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns])
null_sums = null_counts.collect()[0].asDict()
```

```
In [39]: cols_to_drop = [k for k, v in null_sums.items() if v == df.count()]
df_cleaned = df.drop(*cols_to_drop)
print(f"Eliminat {len(cols_to_drop)} coloane complet goale")
```

Eliminat 0 coloane complet goale

```
In [40]: # 4. Eliminăm coloanele cu peste 50% valori lipsă
threshold = int(df.count() * 0.5)
cols_many_nulls = [k for k, v in null_sums.items() if v > threshold]
df_cleaned = df_cleaned.drop(*cols_many_nulls)
print(f"Eliminat {len(cols_many_nulls)} coloane cu peste 50% NaN")
```

Eliminat 5 coloane cu peste 50% NaN

```
In [41]: # 5. Conversie tipuri
df_cleaned = df_cleaned.withColumn("year", col("year").cast(IntegerType()))
df_cleaned = df_cleaned.withColumn("hf_score", col("hf_score").cast(DoubleType()))
df_cleaned = df_cleaned.withColumn("hf_rank", col("hf_rank").cast(IntegerType()))
```

```
In [42]: # 6. Eliminăm rândurile esențiale lipsă
essential_cols = ["year", "countries", "region", "hf_score"]
df_cleaned = df_cleaned.dropna(subset=essential_cols)
```

```
In [43]: # 7. Schema și dimensiune finală
df_cleaned.printSchema()
print(f"Dimensiune finală: {df_cleaned.count()} rânduri, {len(df_cleaned.columns)} coloane")
```

root

```
-- year: integer (nullable = true)
-- countries: string (nullable = true)
-- region: string (nullable = true)
-- hf_score: double (nullable = true)
-- hf_rank: integer (nullable = true)
-- hf_quartile: double (nullable = true)
-- pf_rol_vdem: double (nullable = true)
-- pf_rol: double (nullable = true)
-- pf_ss_homicide: double (nullable = true)
-- pf_ss_homicide_data: double (nullable = true)
-- pf_ss_disappearances_disap: double (nullable = true)
-- pf_ss_disappearances_violent: double (nullable = true)
-- pf_ss_disappearances_violent_data: double (nullable = true)
-- pf_ss_disappearances_organized: double (nullable = true)
-- pf_ss_disappearances_fatalities: double (nullable = true)
-- pf_ss_disappearances_fatalities_data: double (nullable = true)
-- pf_ss_disappearances_injuries: double (nullable = true)
-- pf_ss_disappearances_injuries_data: double (nullable = true)
-- pf_ss_disappearances_torture: double (nullable = true)
-- pf_ss_killings: double (nullable = true)
-- pf_ss_disappearances: double (nullable = true)
-- pf_ss: double (nullable = true)
-- pf_movement_vdem_foreign: double (nullable = true)
-- pf_movement_vdem_men: double (nullable = true)
-- pf_movement_vdem_women: double (nullable = true)
-- pf_movement_vdem: double (nullable = true)
-- pf_movement_cld: double (nullable = true)
-- pf_movement: double (nullable = true)
-- pf_religion_freedom_vdem: double (nullable = true)
-- pf_religion_freedom_cld: double (nullable = true)
-- pf_religion_freedom: double (nullable = true)
-- pf_religion_suppression: double (nullable = true)
-- pf_religion: double (nullable = true)
-- pf_assembly_entry: double (nullable = true)
-- pf_assembly_freedom_house: double (nullable = true)
-- pf_assembly_freedom_bti: double (nullable = true)
-- pf_assembly_freedom_cld: double (nullable = true)
-- pf_assembly_freedom: double (nullable = true)
-- pf_assembly_parties_barriers: double (nullable = true)
-- pf_assembly_parties_bans: double (nullable = true)
-- pf_assembly_parties_auton: double (nullable = true)
-- pf_assembly_parties: double (nullable = true)
-- pf_assembly_civil: double (nullable = true)
-- pf_assembly: double (nullable = true)
-- pf_expression_direct_killed: double (nullable = true)
-- pf_expression_direct_killed_data: integer (nullable = true)
-- pf_expression_direct_jailed: double (nullable = true)
-- pf_expression_direct_jailed_data: integer (nullable = true)
-- pf_expression_direct: double (nullable = true)
-- pf_expression_vdem_cultural: double (nullable = true)
-- pf_expression_vdem_harass: double (nullable = true)
-- pf_expression_vdem_gov: double (nullable = true)
-- pf_expression_vdem_internet: double (nullable = true)
-- pf_expression_vdem_selfcens: double (nullable = true)
-- pf_expression_vdem: double (nullable = true)
-- pf_expression_house: double (nullable = true)
-- pf_expression_bti: double (nullable = true)
-- pf_expression_cld: double (nullable = true)
-- pf_expression: double (nullable = true)
```

```
-- pf_identity_same_m: double (nullable = true)
-- pf_identity_same_f: double (nullable = true)
-- pf_identity_same: double (nullable = true)
-- pf_identity_divorce: double (nullable = true)
-- pf_identity_inheritance: double (nullable = true)
-- pf_identity_fgm: double (nullable = true)
-- pf_identity: double (nullable = true)
-- pf_score: double (nullable = true)
-- pf_rank: double (nullable = true)
-- ef_government_consumption: double (nullable = true)
-- ef_government_consumption_data: double (nullable = true)
-- ef_government_transfers: double (nullable = true)
-- ef_government_transfers_data: double (nullable = true)
-- ef_government_investment: double (nullable = true)
-- ef_government_investment_data: double (nullable = true)
-- ef_government_tax_income: double (nullable = true)
-- ef_government_tax_income_data: string (nullable = true)
-- ef_government_tax_payroll: double (nullable = true)
-- ef_government_tax_payroll_data: string (nullable = true)
-- ef_government_tax: double (nullable = true)
-- ef_government_soa: double (nullable = true)
-- ef_government: double (nullable = true)
-- ef_legal_judicial: double (nullable = true)
-- ef_legal_courts: double (nullable = true)
-- ef_legal_protection: double (nullable = true)
-- ef_legal_military: double (nullable = true)
-- ef_legal_integrity: double (nullable = true)
-- ef_legal_enforcement: double (nullable = true)
-- ef_legal_regulatory: double (nullable = true)
-- ef_legal_police: double (nullable = true)
-- ef_gender: double (nullable = true)
-- ef_legal: double (nullable = true)
-- ef_money_growth: double (nullable = true)
-- ef_money_growth_data: double (nullable = true)
-- ef_money_sd: double (nullable = true)
-- ef_money_sd_data: double (nullable = true)
-- ef_money_inflation: double (nullable = true)
-- ef_money_inflation_data: double (nullable = true)
-- ef_money_currency: double (nullable = true)
-- ef_money: double (nullable = true)
-- ef_trade_tariffs_revenue: double (nullable = true)
-- ef_trade_tariffs_revenue_data: double (nullable = true)
-- ef_trade_tariffs_mean: double (nullable = true)
-- ef_trade_tariffs_mean_data: double (nullable = true)
-- ef_trade_tariffs_sd: double (nullable = true)
-- ef_trade_tariffs_sd_data: double (nullable = true)
-- ef_trade_tariffs: double (nullable = true)
-- ef_trade_regulatory_nontariff: double (nullable = true)
-- ef_trade_regulatory_compliance: double (nullable = true)
-- ef_trade_regulatory: double (nullable = true)
-- ef_trade_black: double (nullable = true)
-- ef_trade_movement_open: double (nullable = true)
-- ef_trade_movement_capital: double (nullable = true)
-- ef_trade_movement_visit: double (nullable = true)
-- ef_trade_movement: double (nullable = true)
-- ef_trade: double (nullable = true)
-- ef_regulation_credit_ownership: double (nullable = true)
-- ef_regulation_credit_private: double (nullable = true)
-- ef_regulation_credit_interest: double (nullable = true)
-- ef_regulation_credit: double (nullable = true)
```

```
|-- ef_regulation_labor_minwage: double (nullable = true)
|-- ef_regulation_labor_firing: double (nullable = true)
|-- ef_regulation_labor_bargain: double (nullable = true)
|-- ef_regulation_labor_hours: double (nullable = true)
|-- ef_regulation_labor_dismissal: string (nullable = true)
|-- ef_regulation_labor_conscription: double (nullable = true)
|-- ef_regulation_labor: double (nullable = true)
|-- ef_regulation_business_adm: double (nullable = true)
|-- ef_regulation_business_burden: double (nullable = true)
|-- ef_regulation_business_start: double (nullable = true)
|-- ef_regulation_business_impartial: double (nullable = true)
|-- ef_regulation_business_licensing: double (nullable = true)
|-- ef_regulation_business_compliance: double (nullable = true)
|-- ef_regulation_business: double (nullable = true)
|-- ef_regulation: double (nullable = true)
|-- ef_score: double (nullable = true)
|-- ef_rank: double (nullable = true)
```

Dimensiune finală: 3083 rânduri, 136 coloane

```
In [44]: # 8. Verificăm dacă mai sunt lipsuri
df_cleaned.select([count(when(col(c).isNull(), c)).alias(c) for c in df_cleaned.
```

|year|countries|region|hf\_score|hf\_rank|hf\_quartile|pf\_rol\_vdem|pf\_rol|pf\_ss\_homi  
cide|pf\_ss\_homicide\_data|pf\_ss\_disappearances\_disap|pf\_ss\_disappearances\_violent|  
pf\_ss\_disappearances\_violent\_data|pf\_ss\_disappearances\_organized|pf\_ss\_disappeara  
nces\_fatalities|pf\_ss\_disappearances\_fatalities\_data|pf\_ss\_disappearances\_injurie  
s|pf\_ss\_disappearances\_injuries\_data|pf\_ss\_disappearances\_torture|pf\_ss\_killings|  
pf\_ss\_disappearances|pf\_ss|pf\_movement\_vdem\_foreign|pf\_movement\_vdem\_men|pf\_movem  
ent\_vdem\_women|pf\_movement\_vdem|pf\_movement\_cld|pf\_movement|pf\_religion\_freedom\_v  
dem|pf\_religion\_freedom\_cld|pf\_religion\_freedom|pf\_religion\_suppression|pf\_religi  
on|pf\_assembly\_entry|pf\_assembly\_freedom\_house|pf\_assembly\_freedom\_bti|pf\_assembl  
y\_freedom\_cld|pf\_assembly\_freedom|pf\_assembly\_parties\_barriers|pf\_assembly\_partie  
s\_bans|pf\_assembly\_parties\_auton|pf\_assembly\_parties|pf\_assembly\_civil|pf\_assembl  
y|pf\_expression\_direct\_killed|pf\_expression\_direct\_killed\_data|pf\_expression\_dire  
ct\_jailed|pf\_expression\_direct\_jailed\_data|pf\_expression\_direct|pf\_expression\_vde  
m\_cultural|pf\_expression\_vdem\_harass|pf\_expression\_vdem\_gov|pf\_expression\_vdem\_in  
ternet|pf\_expression\_vdem\_selfcens|pf\_expression\_vdem|pf\_expression\_house|pf\_expr  
ession\_bti|pf\_expression\_cld|pf\_expression|pf\_identity\_same\_m|pf\_identity\_same\_f|  
pf\_identity\_same|pf\_identity\_divorce|pf\_identity\_inheritance|pf\_identity\_fgm|pf\_i  
dentity|pf\_score|pf\_rank|ef\_government\_consumption|ef\_government\_consumption\_data  
|ef\_government\_transfers|ef\_government\_transfers\_data|ef\_government\_investment|ef  
\_government\_investment\_data|ef\_government\_tax\_income|ef\_government\_tax\_income\_dat  
a|ef\_government\_tax\_payroll|ef\_government\_tax\_payroll\_data|ef\_government\_tax|ef\_g  
overnment\_soa|ef\_government|ef\_legal\_judicial|ef\_legal\_courts|ef\_legal\_protection  
|ef\_legal\_military|ef\_legal\_integrity|ef\_legal\_enforcement|ef\_legal\_regulatory|ef

```

_legal_police|ef_gender|ef_legal|ef_money_growth|ef_money_growth_data|ef_money_sd
|ef_money_sd_data|ef_money_inflation|ef_money_inflation_data|ef_money_currency|ef
_money|ef_trade_tariffs_revenue|ef_trade_tariffs_revenue_data|ef_trade_tariffs_me
an|ef_trade_tariffs_mean_data|ef_trade_tariffs_sd|ef_trade_tariffs_sd_data|ef_tra
de_tariffs|ef_trade_regulatory_nontariff|ef_trade_regulatory_compliance|ef_trade
regulatory|ef_trade_black|ef_trade_movement_open|ef_trade_movement_capital|ef_tra
de_movement_visit|ef_trade_movement|ef_trade|ef_regulation_credit_ownership|ef_re
gulation_credit_private|ef_regulation_credit_interest|ef_regulation_credit|ef_reg
ulation_labor_minwage|ef_regulation_labor_firing|ef_regulation_labor_bargain|ef_r
egulation_labor_hours|ef_regulation_labor_dismissal|ef_regulation_labor_conscript
ion|ef_regulation_labor|ef_regulation_business_adm|ef_regulation_business_burden|
ef_regulation_business_start|ef_regulation_business_impartial|ef_regulation_busin
ess_licensing|ef_regulation_business_compliance|ef_regulation_business|ef_regulat
ion|ef_score|ef_rank|

```

A 20x20 grid of dashed lines. At every intersection of a horizontal and vertical dashed line, there is a small black '+' sign. The grid consists of 20 horizontal rows and 20 vertical columns, creating a total of 400 intersection points.

0	0		0	0		0		53		38	27	
27			34					0				0
1196				0						0		
0				0						53		
53		0			0	53				53		
53			53		23		0			53		
23			0			53				0		
53		0				828				23		
0		53					53				116	



[illegible]

---+-----+-----+

```
In [45]: # 9. Salvăm ca view temporar pentru Spark SQL
df_cleaned.createOrReplaceTempView("freedom")
```

```
In [46]: # 10. Extragere exemplu România
spark.sql("SELECT * FROM freedom WHERE countries = 'Romania']").show(truncate=False)
```

[illegible]

[illegible]

```

|8.7925          |8.6675          |3.33333333333333|6.00041666666666|8.9
8               |6.66666666666667|7.82333333333333|9.7325
|8.77791666666666|8.0425          |10.0             |7.77777777777777
|6.66666666666667|8.148148148148149|9.8675           |8.525
|9.5125          |9.30166666666668|9.2275           |8.67995370370370
4|10.0           |0               |10.0             |10.0
|0               |10.0            |8.1125           |
|6.3225          |8.49            |9.90666666666666|8.2
60000000000002  |8.21833333333333|7.5              |6.66666666666667
|6.66666666666667|7.81033333333333|10.0             |10.0
|7.5             |10.0            |10.0             |9.375           |8.0
|48.0           |4.829411764705883|23.58            |5.7111716621253
406             |16.24000000000002|10.0             |14.52
|10.0            |10              |4.0              |
|45              |7.0             |7.30249999999999|6.96861668536
6245 |6.384689565555557|5.146664246944444|6.0734558100000005|8.33333333333333
|6.282416666666666|5.038344997789794|9.298636447885968|6.593403021666666|
1.0           |6.643868011230304|8.682870627597774|6.58564686201113|9.26
153659587095 |1.846158510322626|9.4738           |2.631           |10.0
|9.354551805867182|8.6              |2.1              |8.98
|5.1             |6.511600000000001|8.720999999999998|8.030533
333333333|6.26077811          |9.966982388430784|8.11388024
9215391 |10.0            |10.0             |8.461538461538462|5.00487
1408170619 |7.82213662323636 |8.49163755144627|8.0
|5.045002502722844|10.0             |7.681667500907615|
3.9           |5.617813269999999|6.929828326666666|
|8.0            |10              |10.0             |
|7.407940266111111|3.325608571666667|5.777777777777779|9.3
55206623126026|6.0825           |6.674731329274217|
|8.17287978574693|6.564784014598604|7.218130593872444|7.74
|19.0           |
|2019|Romania |Eastern Europe|8.25 |38 |1.0 |6.79306226840072|5.9
98871514327621|9.633720930232558|1.26 |10.0 |1
0.0           |0.0             |10.0             |
|10.0            |0.0             |10.0             |
|0.0             |7.845           |9.75             |
9.656428571428572|9.645074750830563|9.9175           |9.595
|9.6375          |9.716666666666669|10.0             |9.858333333333334|8.8
675             |6.666666666666667|7.767083333333334|9.72
|8.743541666666665|7.68            |10.0             |7.777777777777779
|6.666666666666667|8.148148148148149|9.865            |8.6525
|9.4725          |9.33            |8.23             |8.34703703703703
7|10.0           |0               |10.0             |
|0               |10.0            |7.8275           |
|5.985           |6.4675          |9.89             |7.7
266666666666675|7.579333333333333|7.5              |6.666666666666667
|6.666666666666667|7.682533333333334|10.0             |10.0
|7.5             |10.0            |10.0             |9.375           |8.52
|43.0           |5.305882352941175|21.96            |6.4931880108991
84             |13.37           |10.0             |14.52
|10.0            |10              |4.0              |
|45              |7.0             |7.30249999999999|7.22031407276
8072 |5.591189565555555|5.065514894277777|6.0734558100000005|8.33333333333333
|5.318866666666667|5.038344997789794|9.298636447885968|6.593403021666666|
1.0           |6.41409309214697|9.030481983054596|4.847590084727016|9.20
7082802759851|1.9822929931003668|9.234429134801143|3.82785432599428|10.0
|9.3679984801539|9.133333333333333|1.3              |8.98
|5.1             |6.4912          |8.771999999999998|8.201511
111111111|6.26077811          |9.966982388430784|8.11388024
9215391 |10.0            |10.0             |8.461538461538462|8.73942

```

```

5663898878      |9.066988041812444 |8.845594850534736|8.0
|7.506330345998483      |10.0      |8.502110115332828  |
3.9      |5.617813269999999      |6.929828326666666
|8.0      |10      |10.0
|7.407940266111111 |3.325608571666667      |6.666666666666666      |9.3
55206623126026      |5.675      |6.674731329274217
|8.17287978574693      |6.6450154960800845      |7.518355292508008 |7.87
|28.0 |
|2018|Romania |Eastern Europe|8.2      |37      |1.0      |6.657429805349583|6.1
99880810866085|9.627906976744184|1.28      |10.0      |1
0.0      |0.0      |10.0
|10.0      |0.0      |10.0
|0.0      |7.422499999999999      |9.75      |
9.59607142857143      |9.611989202657806|9.9175      |9.595
|9.6375      |9.716666666666669|10.0      |9.858333333333334|8.8
675      |6.666666666666667      |7.767083333333334      |9.72
|8.743541666666665|7.0675      |7.5      |8.88888888888889
|6.666666666666667      |7.685185185185186      |9.865      |8.6525
|9.4725      |9.33      |7.0575      |7.78504629629629
6|10.0      |0      |10.0
|0      |10.0      |7.119999999999999
|5.985      |6.0025      |9.89      |6.6
766666666666668      |7.134833333333333 |7.5      |6.666666666666667
|6.666666666666667|7.593633333333334 |10.0      |10.0      |10.0
|7.5      |10.0      |10.0      |9.375      |8.45
|43.0 |5.5852941176470585      |21.01      |6.4005449591280
65      |13.71      |10.0      |12.441093292910876
|10.0      |10      |4.0
|42      |7.0      |7.302499999999999|7.25766781535
5025 |5.289522898888889 |4.781941898944443 |6.0734558100000005 |8.3
|4.6417      |5.038344997789794      |9.298636447885968      |6.593403021666666 |
1.0      |6.252125634396971 |9.025339529115952|4.87330235442025      |9.30
2319734445744|1.7442006638856384|9.07492      |4.6254      |10.0
|9.350644815890425 |9.133333333333333      |1.3      |8.9599
9999999999      |5.2      |6.256      |9.36
|8.116444444444445|6.26077811      |9.966982388430784      |
8.113880249215391 |10.0      |10.0      |8.461538461538462
|8.296923098638176      |8.919487186725545 |8.787452970096346|8.0
|8.637226970560302      |10.0      |8.879075656853434      |
3.9      |5.617813269999999      |6.929828326666666
|8.0      |10      |10.0
|7.407940266111111 |3.325608571666667      |6.444444444444444      |9.3
55206623126026      |5.0275      |6.674731329274217
|8.17287978574693      |6.500061792376381      |7.595692571780309 |7.85
|30.0 |
|2017|Romania |Eastern Europe|8.27      |36      |1.0      |6.657429805349583|6.3
15841554120521|9.622093023255816|1.3      |10.0      |1
0.0      |0.0      |10.0
|10.0      |0.0      |10.0
|0.0      |7.422499999999999      |9.75      |
9.59607142857143      |9.60908222591362 |9.9175      |9.595
|9.6375      |9.716666666666669|10.0      |9.858333333333334|8.8
675      |6.666666666666667      |7.767083333333334      |9.72
|8.743541666666665|7.327500000000001|10.0      |8.88888888888889
|6.666666666666667      |8.518518518518519      |9.865      |8.6525
|9.065      |9.194166666666666      |7.3875      |8.10692129629629
8|10.0      |0      |10.0
|0      |10.0      |7.595
|5.985      |7.4425      |9.89      |6.1
366666666666667      |7.409833333333333 |7.5      |6.666666666666667

```

|6.666666666666667|7.648633333333334 |10.0 |10.0 |10.0  
|7.5 |10.0 |10.0 |9.375 |8.52  
|42.0 |5.894117647058823 |19.96 |6.8801089918256  
135 |11.95 |10.0 |11.42347164366062  
|10.0 |16 |4.0  
|44 |7.0 |7.302499999999999|7.41534532777  
6888 |5.846093851944445 |5.718631874444445 |6.010306675833332 |8.333333333333334  
|5.55945 |5.038344997789794 |9.298636447885968 |6.012101173333333 |  
1.0 |6.477112294320581 |8.81780972216713 |5.910951389164354 |9.55  
3360881014012|1.1165977974649686|9.732195757791208 |1.33902121104396 |10.0  
|9.525841590243088 |9.0 |1.5 |8.98  
|5.1 |6.6952 |8.261999999999999 |8.225066  
666666667|6.032838424166668 |9.963313764923091 |7.99807609  
454488 |10.0 |10.0 |8.461538461538462 |8.29692  
3098638176 |8.919487186725545 |8.785657486984274|8.0  
|8.637226970560302 |10.0 |8.879075656853434 |  
3.9 |5.560590823333333 |7.149076859166668  
|8.0 |10 |10.0  
|7.434944613750001 |2.8667352591666666 |6.666666666666666 |8.8  
73580348358265 |5.096 |6.674550349782249  
|5.0275 |5.867505437328975 |7.393841902644137 |7.92  
|22.0 |  
|2016|Romania |Eastern Europe|8.38 |32 |1.0 |7.155053293573737|6.3  
28473865701163|9.563953488372093|1.5 |10.0 |1  
0.0 |0.0 |10.0  
|10.0 |0.0 |10.0  
|0.0 |8.055 |9.75 |  
9.686428571428571 |9.625191029900332|9.9175 |9.595  
|9.6375 |9.716666666666669|10.0 |9.858333333333334|8.8  
675 |6.666666666666667 |7.767083333333334 |9.545  
|8.656041666666667|8.94 |10.0 |8.88888888888889  
|6.666666666666667 |8.518518518518519 |9.865 |8.6525  
|9.065 |9.194166666666666 |9.585 |9.05942129629629  
6|10.0 |0 |10.0  
|0 |10.0 |8.2825  
|7.255000000000001 |9.0675 |9.89 |5.7  
766666666666666 |8.054333333333334 |7.5 |6.666666666666667  
|6.666666666666667|7.775333333333332 |10.0 |10.0 |10.0  
|7.5 |10.0 |10.0 |9.375 |8.67  
|38.0 |6.044117647058824 |19.45 |6.9209809264305  
18 |11.8 |9.763258171029 |15.828596401398498  
|10.0 |16 |4.0  
|44 |7.0 |7.2275 |7.39117134890  
3669 |6.452826763333334 |5.6084380425 |5.640152294999999 |8.333333333333334  
|6.215316666666666 |5.038344997789794 |9.237416467918449 |5.646742185 |  
1.0 |6.521571343942697 |8.960203769768352|5.198981151158244 |9.46  
998600575882 |1.3250349856029475|9.6924 |-1.538 |10.0  
|9.530647443881794 |9.0 |1.5 |8.9599  
9999999999 |5.2 |6.8384 |7.904  
|8.266133333333334|6.057826678333332 |9.963313764923091 |  
8.010570221628212 |10.0 |10.0 |8.461538461538462  
|7.743794892062298 |8.73511111786692 |8.752953668207116|8.0  
|9.366340923309862 |10.0 |9.122113641103288 |  
3.9 |4.9906134600000005 |7.330312728333333  
|8.0 |10 |10.0  
|7.3701543647222225 |2.6033580300000003 |6.444444444444444 |9.6  
09960581841714 |5.797499999999999 |6.674550349782249  
|8.17287978574693 |6.550448865302556 |7.680905623709356 |7.98  
|19.0 |  
|2015|Romania |Eastern Europe|8.34 |34 |1.0 |7.11706270398858 |6.3

```

28473865701163|9.50581395348837 |1.7 |10.0 |1
0.0 |0.0 |10.0 |10.0
|10.0 |0.0 |10.0
|0.0 |8.055 |9.75 |
9.686428571428571 |9.596121262458473|9.9175 |9.595
|9.6375 |9.716666666666669|10.0 |9.858333333333334|8.8
675 |6.666666666666667 |7.767083333333334 |9.545
|8.656041666666667|8.65 |10.0 |8.88888888888889
|6.666666666666667 |8.518518518518519 |9.6475 |8.6525
|9.065 |9.121666666666668 |9.78 |9.01754629629629
5|10.0 |0 |10.0
|0 |10.0 |8.5
|6.6425 |8.3925 |9.89 |5.1
833333333333334 |7.721666666666667 |7.500000000000001 |6.666666666666667
|6.666666666666667|7.711 |10.0 |10.0 |10.0
|7.5 |10.0 |10.0 |9.375 |8.65
|38.0 |6.467647058823529 |18.01 |7.0257335718490
65 |11.41555779131394 |8.345348628573271 |20.791279799993543
|10.0 |16 |4.0
|44 |7.0 |7.2275 |7.21324585184
9173 |6.258215274069229 |5.527980966009345 |4.915986947007418 |8.333333333333334
|6.223316666666666 |4.849985115570536 |9.142920280454687 |5.323129494984945 |
1.0 |6.321858509762021 |9.264095791505053|3.679521042474732 |9.00
932474420423 |2.476688139489427 |9.881336256932796 |-0.593318715336016 |10.0
|9.53868919816052 |9.1 |1.35 |8.98
|5.1 |7.0012 |7.496999999999999 |8.3604
|6.211304028828938 |9.963313764923091 |8.087308896876015
|10.0 |10.0 |8.461538461538462 |7.74379489206229
8 |8.73511111786692 |8.795705003685734|8.0 |9.401
|10.0 |9.133666666666668 |3.9 |
4.346062875792216 |7.390703271168633 |8.0 |
10 |10.0 |7.272794357826808
|2.7963612594082354 |6.000000000000001 |9.592825586827486
|5.6075 |8.126689514419864 |8.195298438682
551 |6.719779133223024 |7.708746719238833 |7.92 |21.0 |
|2014|Romania |Eastern Europe|8.33 |34 |1.0 |6.975073546513016|6.3
23739189273733|9.534883720930234|1.6 |10.0 |1
0.0 |0.0 |10.0
|10.0 |0.0 |10.0
|0.0 |8.055 |9.75 |
9.686428571428571 |9.610656146179403|9.9175 |9.595
|9.6375 |9.716666666666669|10.0 |9.858333333333334|8.8
675 |6.666666666666667 |7.767083333333334 |9.545
|8.656041666666667|8.65 |10.0 |8.88888888888889
|6.666666666666667 |8.518518518518519 |9.0425 |8.6525
|9.065 |8.92 |9.78 |8.96712962962963
|10.0 |0 |10.0
|0 |10.0 |8.3225
|6.6425 |8.0275 |9.89 |5.0
1 |7.5785 |7.117283950617287 |6.666666666666667
|6.666666666666667|7.605823456790124 |10.0 |10.0 |10.0
|7.5 |10.0 |10.0 |9.375 |8.63
|39.0 |6.344117647058823 |18.43 |6.7177252918703
81 |12.5459481788357 |9.27483385957846 |17.5380814914754
|10.0 |16 |4.0
|45 |7.0 |7.2275 |7.31283535970
1532 |6.264755263590355 |5.760577029419662 |4.812002126065047 |8.333333333333334
|6.073566666666666 |4.05 |9.180718755440193 |5.361123643704078 |
1.0 |6.229509602277416 |9.436881428561538|2.815592857192306 |9.01
6465446768942|2.458836383077644 |9.786207795593793 |1.06896102203103 |10.0

```



```

|9.55988866773107 |9.813333333333334 |0.28 |8.94
|5.3 |6.5444 |8.639 |8.432577
777777778|6.325602460850112 |9.963313764923091 |8.14445811
2886603 |10.0 |10.0 |8.461538461538462 |7.74379
4892062298 |8.73511111786692 |8.828036752132826|8.0
|9.172 |10.0 |9.057333333333334 |
3.9 |5.202766131991051 |6.385090221476512
|8.0 |10 |10.0
|7.24797605891126 |3.5353787748951264 |6.444444444444444 |9.7
20891714389824 |5.6075 |6.723130801533559
|8.217717091618171 |6.708177137813521 |7.671162176686039 |7.92
|20.0 |
|2013|Romania |Eastern Europe|8.29 |35 |1.0 |6.958311320110151|6.4
35252282209611|9.513287620526368|1.674290585389293 |10.0 |1
0.0 |0.0 |10.0
|10.0 |0.0 |10.0
|0.0 |8.055 |9.75 |
9.686428571428571 |9.59985809597747 |9.9175 |9.595
|9.6375 |9.716666666666669|10.0 |9.858333333333334|8.8
675 |6.666666666666667 |7.767083333333334 |9.545
|8.656041666666667|8.65 |10.0 |8.88888888888889
|6.666666666666667 |8.518518518518519 |9.0425 |8.6525
|9.065 |8.92 |9.78 |8.96712962962963
|10.0 |0 |10.0
|0 |10.0 |8.3225
|6.6425 |8.0275 |9.89 |5.0
1 |7.5785 |7.117283950617287 |6.666666666666667
|6.666666666666667|7.605823456790124 |10.0 |10.0 |10.0
|7.5 |10.0 |10.0 |9.375 |8.64
|41.0 |6.28235294117647 |18.64 |6.7783023088460
04 |12.323630526535164 |9.06049944472258 |18.28825194347098
|10.0 |16 |4.0
|45 |7.0 |7.2275 |7.26973093894
901 |5.927571461458617 |5.64752410759101 |4.925332266203816 |8.333333333333334
|6.024316666666666 |4.049992557785268 |9.147847631711798 |5.361123643704078 |
1.0 |6.177130208556823 |9.566622480075813|2.166887599620937 |9.28
3902556943852|1.7902436076403692|9.202928019520126 |3.985359902399367 |10.0
|9.513363264134949 |9.64 |0.54 |8.9
|5.5 |6.215999999999999 |9.46 |8.252
|4.933758790447127 |8.262868611541847 |6.598313700994487
|10.0 |10.0 |8.461538461538462 |6.41941266615865
7 |8.29365037589904 |8.285991019223381|8.0 |8.869
33309202787 |10.0 |8.956444364009291 |5.0
|5.0919488766139445 |6.732490609947277 |8.0
|10 |10.0 |7.470739914426869
|3.675132590264457 |6.444444444444444 |9.719513417980831
|5.234999999999999 |6.755759075357057 |8.217717091618
171 |6.674594436610826 |7.700592905015662 |7.79 |25.0 |
|2012|Romania |Eastern Europe|8.24 |38 |2.0 |6.861644546461018|6.1
98071503436023|9.455245987767084|1.873953802081229 |10.0 |1
0.0 |0.0 |10.0
|10.0 |0.0 |10.0
|0.0 |7.824999999999999 |9.75 |
9.653571428571428 |9.554408708169255|9.9175 |9.595
|9.6375 |9.716666666666669|10.0 |9.858333333333334|8.8
675 |6.666666666666667 |7.767083333333334 |9.545
|8.656041666666667|8.65 |10.0 |10.0
|6.666666666666667 |8.888888888888889 |8.75 |8.6525
|9.065 |8.8225 |9.78 |9.035347222222222
4|10.0 |0 |10.0

```

0	10.0	8.0125	
6.4275	7.4125	9.89	7.1
766666666666667	7.783833333333334	6.925925925925927	7.777777777777779
6.666666666666667	7.830840740740742	10.0	10.0
7.5	10.0	10.0	9.375
40.0	6.144117647058823	19.11	6.4509673071041
84	13.524949982927644	9.277384418709053	17.52915453451832
10.0	16	4.0	
45	7.0	7.47	7.26849387457
4412	5.233450716824767	4.834929238184633	4.9042252056384665
6.1025	4.049992557785268	9.09607727049065	4.4058222106136
1.0	5.87004131660884	9.651037102080164	1.744814489599178
9555979710271	2.526110050724318	9.334	3.33
9.493648270447608	9.633333333333333	0.55	8.9
5.5	6.106	9.735	8.213111
111111111	4.586118456053067	8.262868611541847	6.42449353
3797458	10.0	10.0	7.6923076923076925
2666158657	8.037240119488784	8.168711191099339	8.0
8.915822217895679	10.0		8.97194073929856
5.0	4.7660984013267		6.821694517412933
8.0	10		10.0
7.43129881978994	3.0003126003316667	6.444444444444444	9.7
00353812314493	5.01		6.109011824215165
7.758134706437953		6.337042897957287	7.580094152348596
28.0			7.68
2011	Romania	Eastern Europe	8.19
9148244381274	9.52011097770852	1.6508182366826891	10.0
0.0	0.0		10.0
10.0	0.0		10.0
0.0		7.824999999999999	9.75
9.653571428571428	9.586841203139976	9.9175	9.595
9.6375	9.716666666666669	10.0	9.858333333333334
675	6.666666666666667	7.767083333333334	9.545
8.656041666666667	8.65	10.0	10.0
6.666666666666667	8.888888888888889	8.75	8.6525
9.065	8.8225	9.78	9.03534722222222
4 10.0	0		10.0
0	10.0	7.4825	
6.4275	7.915	9.89	7.8
266666666666666	7.908333333333334	7.117283950617287	7.777777777777779
6.666666666666667	7.894012345679012	10.0	10.0
7.5	10.0	10.0	9.375
39.0	6.158823529411766	19.06	6.2861035422343
33	14.13	8.535753624005483	20.12486231598081
10.0	16	3.0	
46	6.5	7.47	6.99013613913
0317	5.198087436444445	4.807631937333333	4.751063803333333
6.357533333333335	4.049992557785268	9.114166340447722	4.064947583333333
1.0	5.8345945406680135	9.175991555355148	4.12004223224263
973801926767	2.500654951830821	8.842	5.79
9.254432393655703	9.657687049845954	0.5134694252310705	8.94
5.3	6.226400000000001	9.434	8.274695
683281985	4.764672651666666	8.346711621474565	6.55569213
6570616	10.0	10.0	7.6923076923076925
7196280189	8.03619829619596	8.21664652901214	8.0
7.819527366613194	10.0		8.606509122204399
3.333333333333344	4.3076263133333335		6.317699514999999
8.0	10		10.0
6.9931098602777775	3.004401716666667	6.818181818181818	9.6
47799378348578	5.01		6.094714444349696

7.578785482952988	6.358980473416625	7.319533151966268	7.52
39.0			
2010	Romania	Eastern Europe	8.17
40	2.0	6.861644546461018	6.2
9148244381274	9.425440965042736	1.976483080252992	10.0
0.0	0.0	10.0	10.0
10.0	0.0	10.0	10.0
0.0	7.824999999999999	9.75	
9.653571428571428	9.539506196807082	9.9175	9.595
9.6375	9.716666666666669	10.0	9.858333333333334
675	6.666666666666667	7.767083333333334	9.545
8.656041666666667	8.65	10.0	10.0
6.666666666666667	8.88888888888889	8.75	8.6525
9.065	8.8225	9.78	9.03534722222222
4	10.0	0	10.0
0	10.0	7.4825	
6.4275	7.915	9.89	7.8
266666666666666	7.908333333333334	6.925925925925927	7.777777777777779
6.666666666666667	7.855740740740741	10.0	10.0
7.5	10.0	10.0	9.375
41.0	5.873529411764705	20.03	5.9588555858310
64	15.331	8.021683184914627	21.92410885279881
10.0	16	3.0	
48	6.5	7.47	6.76481363650
2079	5.376236157625431	4.908557769123712	4.887734508591064
8.333333333333334	6.357533333333335	4.049992557785268	9.114166340447722
4.535655872852233	1.0	5.945401234136512	8.203639252071845
8.981803739640771	8.97	647311064959	2.558817223376024
8.781156842606158	6.0942157869692	10.0	8.990317301331899
9.632409333333332	0.551386	8.98	5.1
6.409600000000001	8.975999999999999	8.340669	777777778
5.782607800687285	8.346711621474565	7.06465971	1080925
10.0	10.0	7.6923076923076925	6.41628
7196280189	8.03619829619596	8.360381946263667	10.0
7.006965880410803	10.0	9.002321960136934	
3.333333333333334	4.288750996563573	6.3677733745704455	
8.0	9.251725384611367	10.0	
6.87359718151312	3.078326989690721	6.818181818181818	9.5
17632357638876	5.01	6.105754193359742	
7.511529524146127	6.340237480502881	7.405385540717646	7.49
42.0			
2009	Romania	Eastern Europe	8.18
40	2.0	6.800532234691593	6.2
35448203916643	9.440259828173131	1.9255061910844211	10.0
0.0	0.0	10.0	10.0
10.0	0.0	10.0	10.0
0.0	7.6775	9.75	
9.632500000000002	9.536379914086568	9.7475	9.595
9.6375	9.66	10.0	9.83
675	6.666666666666667	7.767083333333334	9.545
8.656041666666667	8.8475	10.0	10.0
6.666666666666667	8.88888888888889	8.75	8.6525
9.065	8.8225	9.78	9.08472222222222
2	10.0	0	10.0
0	10.0	7.4825	
6.4275	7.915	9.89	7.3
3	7.809	6.925925925925927	7.777777777777779
6.666666666666667	7.835874074074074	10.0	10.0
7.5	10.0	10.0	9.375
41.0	8.573529411764705	10.85	6.5030716978835
25	13.333726868767467	7.956043956043953	22.153846153846168
10.0	16	3.0	
48	6.5	7.47	7.40052901313

```

8437 |5.600494494773948 |4.620647739931035 |5.286908626436783 |8.333333333333334
|6.231783333333333|4.049992557785268 |8.665511991878795 |5.266142890804598 |
1.0 |6.006851871034636 |7.990641784906929|10.046791075465356 |9.10
444584499958 |2.238853875010484|8.882467300596568 |5.587663497017161 |10.0
|8.994388732625769 |9.44 |0.84 |8.94
|5.3 |6.29 |9.275 |8.223333
333333334|6.505606339080461 |8.346711621474565 |7.42615898
0277513 |10.0 |10.0 |7.6923076923076925 |6.41628
7196280189 |8.03619829619596 |8.421422652451703|5.0
|6.407898314275169 |10.0 |7.135966104758389 |
3.333333333333334 |4.733528718390805 |6.623697172413795
|8.0 |9.598272959726625 |10.0
|7.048138697310759 |3.1986141264367824 |4.15 |9.6
49572421660634 |4.87 |7.066659455725212
|7.511529524146127 |6.074395921328126 |6.752833574465758 |7.52
|34.0 |
|2008|Romania |Eastern Europe|8.19 |40 |2.0 |6.830780700566438|6.2
63183201075889|9.34380022204058 |2.2573272361804024 |10.0 |1
0.0 |0.0 |10.0
|10.0 |0.0 |9.956197249
120756 |1.0 |7.6775
|9.75 |9.626242464160107 |9.485021343100346|9.7475 |
9.595 |9.6375 |9.66 |10.0 |
9.83 |8.8675 |6.666666666666667 |7.7670833333333
334 |9.545 |8.656041666666667|8.8175 |10.0
|10.0 |6.666666666666667 |8.888888888888889 |8.75
|8.6525 |9.065 |8.8225 |9.78
|9.077222222222222|10.0 |0 |1
0.0 |0 |10.0 |
7.4825 |6.4275 |7.915 |9.89
|7.33 |7.809 |6.734567901234568 |7.77777777777
7779|6.666666666666667|7.7976024691358035|10.0 |10.0 |1
0.0 |7.5 |10.0 |10.0 |9.375
|8.64 |39.0 |8.98235294117647 |9.46 |6.5177
11171662125 |13.28 |9.32076772260581 |17.377312
970879665 |10.0 |16 |3.0
|48 |6.5 |7.22 |7.70816636708
8881 |5.581503612398957 |4.503671707666666 |5.163580798657717 |8.333333333333334
|6.231783333333333|4.049992557785268 |8.665511991878795 |5.287886666666667 |
1.0 |5.977158000215092 |7.737564122848961|11.312179385755194 |9.24
1566500787624|1.896083748030939 |8.430334971224006 |7.848325143879961 |10.0
|8.852366398715148 |9.606666666666667 |0.59 |8.88
|5.6 |5.608 |10.98 |8.031555
555555556|6.325602460850112 |8.346711621474565 |7.33615704
1162338 |10.0 |10.0 |8.461538461538462 |6.41628
7196280189 |8.292608552606216 |8.415080287331028|5.0
|7.966149893217473 |10.0 |7.655383297739157 |
3.3 |5.202766131991051 |6.385090221476512
|6.0 |9.258350079495306 |10.0
|6.691034405493812 |3.315930937360179 |4.15 |9.6
46421081939586 |4.87 |6.796610273173463
|7.7357160535023315 |6.08577972432926 |6.810732475854077 |7.55
|34.0 |
|2007|Romania |Eastern Europe|8.14 |41 |2.0 |6.830780700566438|6.2
6318320107589 |9.425130279270062|1.977551839310983 |10.0 |1
0.0 |0.0 |NULL
|10.0 |0.0 |10.0
|0.0 |7.725 |9.77 |
9.5825 |9.50381513963503 |9.8025 |9.615 |
|9.6725 |9.696666666666667|10.0 |9.848333333333333|8.9

```

```

8 |6.666666666666667 |7.823333333333334 |9.5525
|8.687916666666666|8.1275 |10.0 |10.0
|6.666666666666667 |8.88888888888889 |8.79 |8.525
|9.08 |8.798333333333332 |9.7225 |8.88430555555555
7|10.0 |0 |10.0
|0 |10.0 |7.455
|6.16 |7.8025 |9.906666666666666 |7.3
33333333333334 |7.7315 |6.734567901234567 |7.777777777777779
|6.666666666666667|7.782102469135802 |10.0 |10.0 |10.0
|7.5 |10.0 |10.0 |9.375 |8.62
|40.0 |9.061764705882354 |9.19 |6.6928886761899
|12.63709855838307 |9.305521865137331 |17.430673472019333
|10.0 |16 |2.0
|52 |6.0 |7.22 |7.65603504944
19165|5.449242092691836 |4.586288259116022 |5.2831484697586495 |8.333333333333334
|6.124533333333334 |4.198418193264836 |7.971167555904099 |5.375424085192892 |
1.0 |5.915194415324375 |6.63929762739464 |-16.8035118630268 |7.74
0591200632611|5.648521998418471 |9.032366597133375 |4.83816701433311 |10.0
|8.353063856290158 |9.566666666666666 |0.65 |8.96
|5.2 |6.006399999999999 |9.984000000000002 |8.177688
88888889 |6.169773501501047 |8.346711621474565 |7.25824256
14878066 |10.0 |10.0 |8.461538461538462 |6.41628
7196280189 |8.292608552606216 |8.432135000745728|5.0
|8.70019514951256 |10.0 |7.90006504983752 |
3.3 |4.965695474289062 |6.568667456733873
|6.0 |9.258350079495306 |10.0
|6.682118835086374 |4.013935603298176 |4.15 |9.6
38412256253192 |4.87 |6.790637949938521
|7.7357160535023315 |6.1997836438320375 |6.92732250958531 |7.46
|43.0 |
|2006|Romania |Eastern Europe|7.97 |42 |2.0 |6.664293784359429|6.1
10530363501966|9.40064936504969 |2.06176618422907 |10.0 |1
0.0 |0.0 |NULL
|10.0 |0.0 |10.0
|0.0 |7.725 |9.77 |
9.5825 |9.491574682524844|9.435 |9.615
|9.6725 |9.574166666666668|10.0 |9.787083333333335|8.9
8 |6.666666666666667 |7.823333333333334 |9.5525
|8.687916666666666|8.1275 |10.0 |8.88888888888889
|6.666666666666667 |8.518518518518519 |8.79 |8.525
|9.08 |8.798333333333332 |9.7225 |8.79171296296296
1|10.0 |0 |10.0
|0 |10.0 |7.455
|6.16 |7.8025 |9.906666666666666 |7.3
33333333333334 |7.7315 |7.117283950617284 |7.777777777777779
|6.666666666666667|7.858645679012345 |10.0 |10.0 |10.0
|7.5 |10.0 |10.0 |9.375 |8.59
|43.0 |9.129411764705882 |8.96 |6.9264305177111
72 |11.78 |8.48111026517848 |20.31611407187533
|10.0 |16 |2.0
|52 |6.0 |7.22 |7.55139050951
9106 |5.170516542896558 |4.335613550733004 |5.216463017244776 |8.333333333333334
|6.066533333333334 |4.229212859822908 |7.839683060990518 |3.2567472747301967|
1.0 |5.556012871635578 |6.45226656006345 |-17.738667199682755 |7.44
4883757928471|6.387790605178823 |8.683719818312273 |6.58140090843864 |10.0
|8.145217534076048 |9.606666666666667 |0.59 |6.82
|15.9 |4.975599999999999 |12.561000000000002 |7.134088
88888889 |5.713725730880274 |8.346711621474565 |7.03021867
617742 |10.0 |9.398808479309082 |6.153846153846154 |6.41628
7196280189 |7.322980609811807 |7.871822043719529|5.0

```

8.69658127448027	9.0	7.565527091493423	
2.2	4.398288229461413	7.362296644668458	
6.0	9.258350079495306	3.0	
5.369822492270863	4.115739506911585	4.15	9.4
97479817039116	4.87	6.730733738097123	
7.7357160535023315	6.183278185925026	6.3728759232297705	7.1
63.0			
2005	Romania	Eastern Europe	7.89
45	2.0	6.664293784359429	6.1
10530363501966	9.385535271765308	2.11375866512734	10.0
0.0	0.0	NULL	
10.0	0.0	10.0	
0.0	7.725	9.77	
9.5825	9.484017635882656	9.435	9.615
9.6725	9.574166666666668	6.666666666666667	8.120416666666667
8	6.666666666666667	7.823333333333334	9.5525
8.687916666666666	8.1275	10.0	8.88888888888889
6.666666666666667	8.518518518518519	8.79	8.525
9.08	8.798333333333332	9.7225	8.79171296296296
1	10.0	0	10.0
0	10.0	7.455	
6.16	7.8025	9.906666666666666	7.3
333333333333334	7.7315	6.734567901234567	7.777777777777779
6.666666666666667	7.782102469135802	10.0	10.0
7.5	10.0	10.0	9.375
49.0	8.941176470588236	9.6	7.0054495912806
53	11.49	10.0	12.521815026693762
10.0	16	2.0	
54	6.0	7.22	7.83332521237
37775	5.003152237784378	4.343838120059483	5.120274914089347
8.333333333333334	6.066533333333334	4.479212859822908	7.971167555904099
4.438943894389439	1.0	5.71955703108954	8.029827612935726
-9.850861935321364	6.07	0053623137697	9.82486594215576
8.202188639916562	8.98905680041719	10.0	8.075517468997496
9.586666666666666	0.62	8.68	6.6
4.975599999999999	12.561000000000002	7.747422	22222222
5.42904290429043	8.346711621474565	6.88787726	2882498
10.0	8.797616362571716	6.923076923076923	6.4
7.373564428549547	8.002215978413567	5.0	9.3555904069
945	9.0	7.785196802331501	3.3
4.15	7.87128712871287	6.0	
9.44376255962148	3.0	5.627508281389058	
5	3.4983498349834985	7.035714285714287	9.596352902596609
4.87	6.718427132643302	7.836599991712	624
6.592574024608386	6.668426369442982	7.26	50.0
2004	Romania	Eastern Europe	7.74
47	2.0	6.664293784359429	6.1
10530363501966	9.305271360080448	2.38986652132326	10.0
0.0	0.0	NULL	
10.0	0.0	10.0	
0.0	7.725	9.77	
9.5825	9.443885680040225	9.435	9.615
9.6725	9.574166666666668	10.0	9.787083333333335
8	6.666666666666667	7.823333333333334	9.5525
8.687916666666666	8.1275	10.0	8.88888888888889
6.666666666666667	8.518518518518519	8.79	8.525
9.08	8.798333333333332	8.7875	8.55796296296296
3	10.0	0	10.0
0	10.0	7.455	
6.16	5.68	9.906666666666666	6.5
9	7.158333333333333	6.351851851851851	7.777777777777779
6.666666666666667	7.590925925925926	10.0	10.0
7.5	10.0	10.0	9.375
			8.51

45.0	9.038235294117648	9.27	7.0081743869209
81	11.48	10.0	12.989405726436187
5.0	40	2.0	
55	3.5	6.719083392515126	7.25309861471
0751	4.7206842575073376	4.138623698206646	3.3176100628930816
8.333333333333334	5.959533333333333	4.479212859822908	7.65892505350586
4.438943894389439	1.0	5.380858311623992	8.007745540212428
-9.961272298937862	5.28	8626455212891	11.778433861967772
7.62462642639872	11.8768678680064	5.0	6.4802496054560095
9.593333333333334	0.61	7.34	13.3
3.64	15.9	6.857777	777777778
5.490196078431372	6.152406441263576	5.82130125	9847473
10.0	8.196424841880798	6.153846153846154	NULL
7.175135497863476	7.463553633872181	5.0	9.4319947254
90553	9.0	7.801336606802754	3.3
3.5062893081761004	7.64797507788162	6.0	9.44376255962148
3.0	5.483004490946533	3.333333333333333	9.952830188679243
9.585915333927543	4.87	6.955329287629371	7.870227971116
054	7.094606019114258	6.792982372287849	6.67
75.0	2003	Romania	Eastern Europe
7.65	51	2.0	6.596874109036501
6.0	48712864680852	9.2628514165631	2.5357911270229323
10.0	0.0	NULL	10.0
0.0	7.725	9.77	9.5825
8.88	9.422675708281549	8.88	9.615
9.6725	9.389166666666668	10.0	9.694583333333334
8.9	6.666666666666667	7.823333333333334	9.5525
8.88888888888889	8.687916666666666	8.1275	10.0
8.525	6.666666666666667	8.518518518518519	8.79
8.525	9.08	8.798333333333332	8.7875
8.55796296296296	3	10.0	0
10.0	10.0	7.455	6.49
5.68	9.906666666666666	6.5	9
7.224333333333334	6.160493827160495	7.777777777777779	6.666666666666667
7.565854320987654	10.0	10.0	10.0
7.5	10.0	10.0	9.375
8.48	46.0	8.379411764705884	11.51
6.3444141689373	31	13.916	9.620465203974778
16.32837178608828	5.0	40	2.0
7.01635822752	55	3.5	7.2375
3599	4.768781327777778	4.145584965888889	3.833333333333333
8.333333333333334	6.056033333333334	4.479212859822908	7.626053929777464
4.438943894389439	1.0	5.460159622207059	7.774431571470326
-11.127842142648367	5.48	6415789429718	11.283960526425703
6.94520577123582	15.2739711438209	5.0	6.301513283033966
9.633333333333333	0.55	7.72	11.4
3.64	15.9	6.997777	777777777
4.166666666666667	7.333333333333334	5.75	10.0
5.088071823120117	6.153846153846154	NULL	5.620958988483135
7.092184191565228	5.0	9.1948650811	38046
9.0	7.734569551338862	2.2	3.5
7.833333333333333	6.0	9.44376255962148	3.0
5.329515982159135	3.666666666666667	7.5	9.018591364234345
4.87	NULL	7.870227971116	054
6.585097200403413	6.549727577967137	6.48	75.0
2002	Romania	Eastern Europe	7.47
58	2.0	6.596874109036501	6.0
48712864680852	9.251131470914167	2.576107740055263	10.0
1	0.0	NULL	10.0
0.0	7.725	9.77	0.0

9.5825		9.416815735457083 8.88		9.615
9.6725		9.389166666666668 6.6666666666667	8.027916666666668 8.9	
8		6.666666666666667	7.823333333333334	9.5525
8.687916666666666 8.1275		10.0		8.88888888888889
6.666666666666667		8.518518518518519	8.3375	8.525
9.08		8.6475	8.7875	8.52025462962963
10.0		0		10.0
0		10.0		7.455
6.49		5.68	9.906666666666666	6.5
9		7.224333333333334	7.308641975308643	7.777777777777779
6.666666666666667 7.7954839506172835 10.0			10.0	10.0
7.5		10.0	10.0	9.375
49.0	9.4	8.04		6.9673024523160
76	11.63	9.909788588860035		15.31573993898987
5.0		40		2.0
54		3.5	7.2375	7.40291820823
52225 4.2879893274444445 3.670984323444445		3.0		8.333333333333334
5.812033333333334	4.479212859822908	7.626053929777464	4.438943894389439	
0.9411764705882352 5.052949202393371	8.900112245634604	-5.49943877182697	5.07	
5349455284335 12.311626361789164 5.49255799936782		22.5372100031609		5.0
6.11700492507169	9.5	0.75		7.72
11.4		3.64	15.9	6.953333
333333333 3.833333333333333		7.333333333333334		5.58333333
3333334	10.0	4.486880302429199	4.615384615384616	NULL
4.551132458906908	6.771949781393394 5.0			8.9780813912
36547	9.0		7.649461187659159	2.2
4.5		8.0		6.0
9.44376255962148		3.0		5.52396042660358
1.666666666666663		7.0		9.018591364234345
4.87		NULL		7.870227971116
054		6.085097200403413	6.419506271555385	6.35
2001 Romania	Eastern Europe 7.21	65	3.0	6.596874109036501 6.0
48712864680852 9.210582942526932 2.715594677707356		10.0		1
0.0		0.0		NULL
10.0		0.0		10.0
0.0		7.725		9.77
9.5825		9.396541471263465 8.88		9.615
9.6725		9.389166666666668 6.6666666666667	8.027916666666668 8.9	
8		6.666666666666667	7.823333333333334	9.5525
8.687916666666666 8.1275		10.0		8.88888888888889
6.666666666666667		8.518518518518519	8.3375	8.525
9.08		8.6475	8.7875	8.52025462962963
10.0		0		10.0
0		10.0		7.455
6.49		5.68	9.906666666666666	6.5
9		7.224333333333334	7.5	7.777777777777779
6.666666666666667 7.833755555555555		10.0	10.0	10.0
7.5		10.0	10.0	9.375
50.0	9.26764705882353	8.49		6.3980347047331
72	13.719212633629258	10.0		13.352685150695564
5.0		40		2.0
55		3.5	6.92	7.21713635271
134	4.453383488268134	3.7601174271330025 2.677305142084758		8.333333333333334
5.779283333333334	4.479212859822908	7.626053929777464	4.438943894389439	
0.9411764705882352 5.040705523781979	9.220951227444065	-3.895243862779685	0.0	
45.56116682123042	3.1064465364128	34.467767317936	5.0	
4.331849440964215	9.264918137387738	1.102622793918393		7.72
11.4		3.64	15.9	6.874972
712462579 3.799999952316284		7.333333333333334		5.56666664
2824809	10.0	1.6572581231594086	2.307692307692308	NULL



[illegible]

only showing top 20 rows

## Concluzii

Am curățat semnificativ setul de date, păstrând doar coloanele relevante și eliminând valorile lipsă critice. Datasetul rezultat este mult mai ușor de analizat și interpretat, atât din punct de vedere statistic, cât și computațional.

În secțiunea următoare, vom realiza **agregări și grupări** pentru a explora cum evoluează libertatea în diferite regiuni ale lumii.

## 2.2 Analize exploratorii: grupări și agregări relevante

### Obiectiv general

În această secțiune vom realiza mai multe **agregări și grupări semnificative** asupra setului de date, cu scopul de a evidenția:

- care sunt cele mai libere țări din lume (pe medie);
- care țări au înregistrat cele mai mari progrese sau regrese în libertate;
- cum evoluează libertatea în funcție de regiune și timp;
- relația dintre libertatea personală și cea economică;
- țările unde există un dezechilibru major între dimensiunile de libertate;
- clasificarea țărilor în categorii proprii de libertate: înaltă, medie și scăzută.

Pentru fiecare analiză vom utiliza **PySpark DataFrame API** și **Spark SQL**, iar în unele cazuri vom include vizualizări pentru a sprijini interpretarea rezultatelor.

### 2.2.1 Top 10 țări cu cele mai mari scoruri de libertate (medie)

**Obiectiv:** Să identificăm cele mai libere țări din lume pe baza scorului hf\_score, calculat ca medie pe toate înregistrările disponibile pentru fiecare țară în perioada analizată.

Această analiză ne va ajuta să înțelegem care state mențin constant un nivel ridicat de libertate umană, indiferent de contextul global sau regional.

Vom realiza agregarea atât cu PySpark DataFrame API, cât și cu Spark SQL, pentru a valida rezultatele și a demonstra versatilitatea platformei Spark. Rezultatul va fi prezentat atât tabelar, cât și vizual.

```
In [47]: # 1. Cream o vizualizare temporară pentru Spark SQL
df_cleaned.createOrReplaceTempView("freedom_data")
```

```
In [48]: # 2. Top 10 țări cu cele mai mari scoruri medii - cu PySpark DataFrame API
from pyspark.sql.functions import avg

top10_df = df_cleaned.groupBy("countries") \
    .agg(avg("hf_score").alias("avg_hf_score")) \
    .orderBy(col("avg_hf_score").desc()) \
    .limit(10)

top10_df.show(truncate=False)
```

countries	avg_hf_score
Switzerland	9.166666666666666
New Zealand	9.09
Denmark	8.995714285714287
Finland	8.915238095238097
Australia	8.907619047619047
United Kingdom	8.904285714285715
Ireland	8.899047619047618
Sweden	8.88095238095238
Canada	8.855714285714285
United States	8.827142857142857

```
In [49]: # 3. Top 10 cu Spark SQL
query = """
SELECT countries, ROUND(AVG(hf_score), 2) AS avg_hf_score
FROM freedom_data
GROUP BY countries
ORDER BY avg_hf_score DESC
LIMIT 10
"""
```

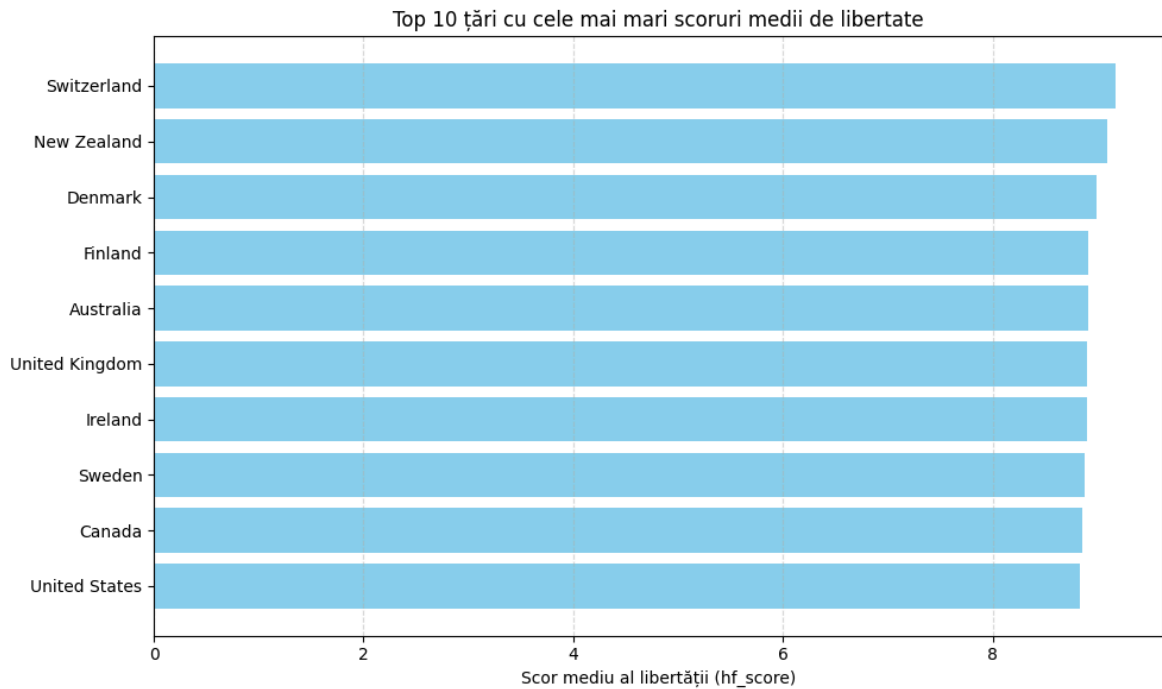
```
In [50]: top10_sql = spark.sql(query)
top10_sql.show(truncate=False)
```

countries	avg_hf_score
Switzerland	9.17
New Zealand	9.09
Denmark	9.0
Finland	8.92
Australia	8.91
Ireland	8.9
United Kingdom	8.9
Sweden	8.88
Canada	8.86
United States	8.83

```
In [51]: # 4. Vizualizare (cu Matplotlib)
import matplotlib.pyplot as plt

# Convertim rezultatul la Pandas DataFrame
top10_pd = top10_df.toPandas()

# Plot - bară orizontală
plt.figure(figsize=(10, 6))
plt.barh(top10_pd['countries'][::-1], top10_pd['avg_hf_score'][::-1], color='sky')
plt.xlabel('Scor mediu al libertății (hf_score)')
plt.title('Top 10 țări cu cele mai mari scoruri medii de libertate')
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



## 2.2.2 Evoluția libertății în 20 de ani: țările cu cele mai mari creșteri și scăderi (2000 vs. 2020)

**Obiectiv:** Să identificăm țările care au înregistrat cele mai semnificative progrese sau regresii în privința libertății umane, comparând scorurile hf\_score din anii 2000 și 2020.

Vom calcula diferența dintre scorurile de libertate pentru fiecare țară prezentă în ambii ani, extrăgând:

- Top 10 țări cu cea mai mare creștere a scorului de libertate;
- Top 10 țări cu cea mai mare scădere a acestui scor.

Această analiză oferă o perspectivă clară asupra modului în care evoluțiile geopolitice și economice au influențat nivelul libertății în ultimele două decenii.

```
In [52]: from pyspark.sql.functions import col
```

```
In [53]: # 1. Filtrăm doar anii 2000 și 2020 și eliminăm valorile lipsă
df_2000 = df_cleaned.filter(col("year") == 2000) \
    .select("countries", "hf_score") \
    .dropna(subset=["hf_score"]) \
    .withColumnRenamed("hf_score", "hf_score_2000")

df_2020 = df_cleaned.filter(col("year") == 2020) \
    .select("countries", "hf_score") \
    .dropna(subset=["hf_score"]) \
    .withColumnRenamed("hf_score", "hf_score_2020")
```

```
In [54]: # 2. Alăturăm cele două DataFrame-uri după "countries"
df_diff = df_2000.join(df_2020, on="countries", how="inner")
```

```
In [55]: # 3. Calculăm diferența scorurilor
df_diff = df_diff.withColumn("score_change", col("hf_score_2020") - col("hf_score_2000"))
```

```
In [56]: # 4. Top 10 creșteri
top10_up = df_diff.orderBy(col("score_change").desc()).limit(10)
print("Top 10 creșteri între 2000 și 2020:")
top10_up.show(truncate=False)
```

Top 10 creșteri între 2000 și 2020:

countries	hf_score_2000	hf_score_2020	score_change
Myanmar	3.64	5.56	1.9199999999999995
Sierra Leone	5.26	6.53	1.2700000000000005
Tunisia	5.27	6.25	0.9800000000000004
Guinea-Bissau	5.56	6.54	0.9800000000000004
Cyprus	7.32	8.13	0.8100000000000005
Congo, Dem. Rep.	4.64	5.39	0.75
Romania	7.2	7.89	0.6899999999999995
Colombia	5.97	6.65	0.6800000000000006
Nepal	6.12	6.77	0.6499999999999995
Togo	5.61	6.25	0.6399999999999997

```
In [57]: # 5. Top 10 scăderi
top10_down = df_diff.orderBy(col("score_change").asc()).limit(10)
print("Top 10 scăderi între 2000 și 2020:")
top10_down.show(truncate=False)
```

Top 10 scăderi între 2000 și 2020:

countries	hf_score_2000	hf_score_2020	score_change
Venezuela, RB	6.43	4.09	-2.34
Nicaragua	7.34	5.96	-1.38
Egypt, Arab Rep.	5.49	4.28	-1.21
Syrian Arab Republic	4.47	3.3	-1.17
Bolivia	7.73	6.6	-1.1300000000000008
El Salvador	7.74	6.65	-1.0899999999999999
Argentina	8.06	6.99	-1.0700000000000003
Iran, Islamic Rep.	5.14	4.26	-0.8799999999999999
Hong Kong SAR, China	8.87	8.01	-0.8599999999999994
Thailand	7.35	6.49	-0.8599999999999994

```
In [59]: # 6. Vizualizare comparativă
import matplotlib.pyplot as plt

top_up_pd = top10_up.toPandas()
top_down_pd = top10_down.toPandas()

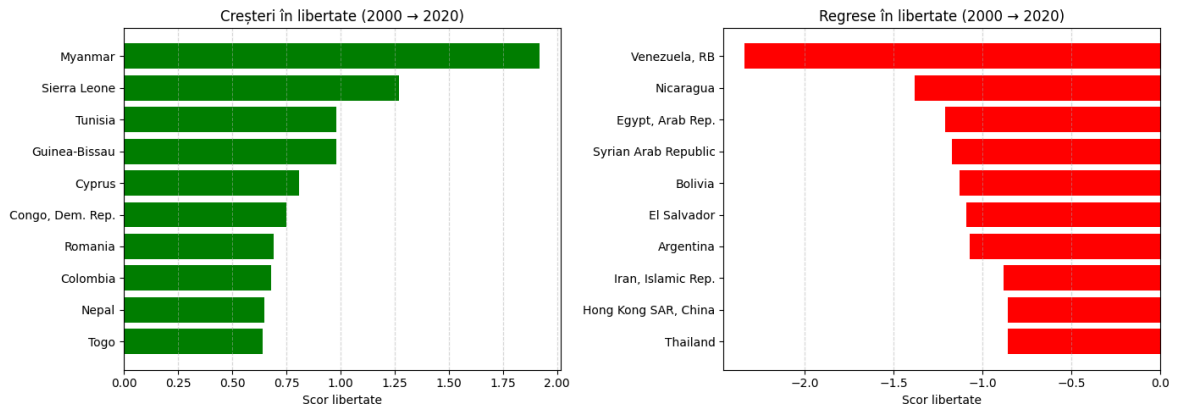
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Creșteri
axes[0].barh(top_up_pd['countries'][::-1], top_up_pd['score_change'][::-1], color='red')
axes[0].set_title('Creșteri în libertate (2000 → 2020)')
axes[0].set_xlabel('Scor libertate')
axes[0].grid(axis='x', linestyle='--', alpha=0.5)

# Scăderi
axes[1].barh(top_down_pd['countries'][::-1], top_down_pd['score_change'][::-1], color='blue')
```

```
axes[1].set_title('Regrese în libertate (2000 → 2020)')
axes[1].set_xlabel('Scor libertate')
axes[1].grid(axis='x', linestyle='--', alpha=0.5)

plt.tight_layout()
plt.show()
```



## 2.2.3 Evoluția mediei scorului hf\_score pe regiuni între 2000 și 2020

**Obiectiv:** să analizăm dinamica libertății umane în timp, pe regiuni geografice, prin evoluția scorului mediu hf\_score în perioada 2000–2020.

Această analiză evidențiază:

- Regiunile cu îmbunătățiri consistente în libertate;
- Regiunile în care nivelul de libertate a scăzut sau a stagnat;
- Comparații regionale anuale pentru decizii de politică și cercetare.

Vizualizarea sub formă de line chart ajută la înțelegerea clară a tendințelor globale și regionale.

```
In [60]: from pyspark.sql.functions import avg, col
```

```
In [61]: # 1. Filtrăm perioada dorită
df_range = df_cleaned.filter((col("year") >= 2000) & (col("year") <= 2020))
```

```
In [62]: # 2. Grupăm după regiune și an, calculând media scorului
df_region_year = df_range.groupBy("region", "year") \
    .agg(avg("hf_score").alias("avg_hf_score")) \
    .orderBy("region", "year")

df_region_year.show(10, truncate=False)
```

```

+-----+-----+-----+
| region                | year | avg_hf_score |
+-----+-----+-----+
|Caucasus & Central Asia|2000|7.15          |
|Caucasus & Central Asia|2003|7.39          |
|Caucasus & Central Asia|2004|7.116666666666667 |
|Caucasus & Central Asia|2005|6.87          |
|Caucasus & Central Asia|2006|6.8340000000000005|
|Caucasus & Central Asia|2007|6.814         |
|Caucasus & Central Asia|2008|6.746         |
|Caucasus & Central Asia|2009|6.628         |
|Caucasus & Central Asia|2010|6.535         |
|Caucasus & Central Asia|2011|6.538333333333334 |
+-----+-----+-----+

```

only showing top 10 rows

```

In [63]: # 3. Conversie la Pandas pentru vizualizare
df_plot = df_region_year.toPandas()

```

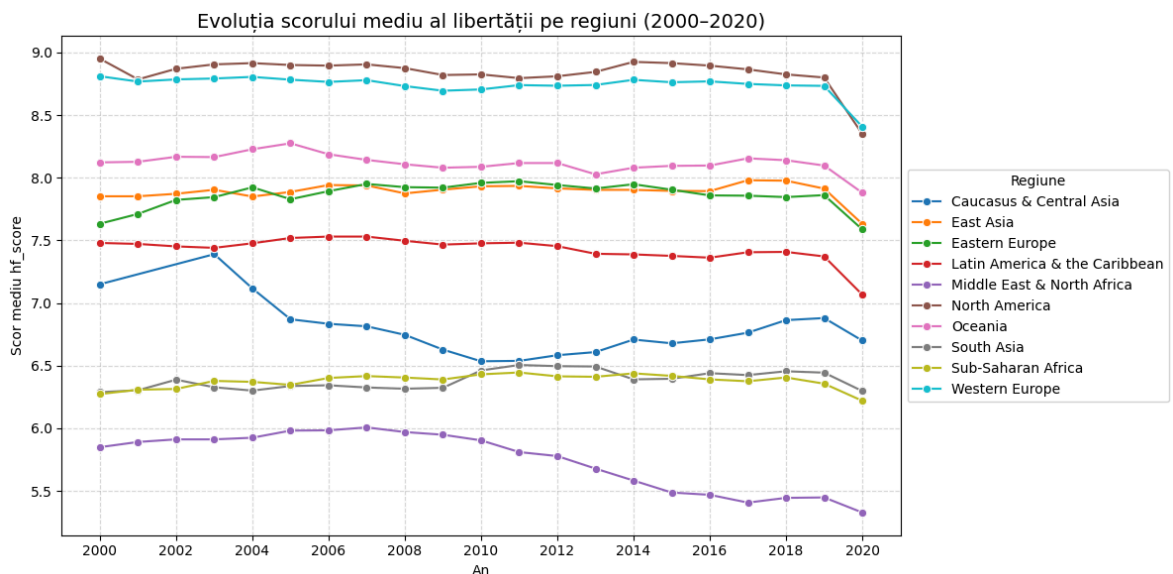
```

In [64]: # 4. Vizualizare cu seaborn
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.lineplot(
    data=df_plot,
    x="year",
    y="avg_hf_score",
    hue="region",
    marker="o"
)

plt.title("Evoluția scorului mediu al libertății pe regiuni (2000-2020)", fontsi
plt.xlabel("An")
plt.ylabel("Scor mediu hf_score")
plt.xticks(range(2000, 2021, 2))
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(title="Regiune", loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()

```



## 2.2.4 Relația dintre libertatea personală și libertatea economică

**Obiectiv:** să analizăm relația dintre `pf_score` (libertate personală) și `ef_score` (libertate economică), pentru a înțelege dacă există o corelație directă între cele două dimensiuni ale libertății. Ne întrebăm: țările care oferă mai multă libertate personală oferă automat și mai multă libertate economică?

Pentru aceasta, vom:

- folosi un scatter plot colorat în funcție de regiune;
- calcula coeficientul de corelație Pearson pentru a măsura intensitatea și direcția legăturii;
- compara relația folosind toate înregistrările anuale versus scoruri medii pe țară.

```
In [65]: # 1. Selectăm scorurile necesare și eliminăm valorile lipsă
df_pair = df_cleaned.select("countries", "region", "year", "pf_score", "ef_score")
df_pair.dropna(subset=["pf_score", "ef_score"])
```

```
In [66]: # 2. Conversie la Pandas pentru vizualizare
df_pair_pd = df_pair.toPandas()
```

```
In [67]: df_pair_pd
```

```
Out[67]:
```

	countries	region	year	pf_score	ef_score
0	Albania	Eastern Europe	2020	7.69	7.64
1	Algeria	Middle East & North Africa	2020	5.13	5.12
2	Angola	Sub-Saharan Africa	2020	6.02	5.91
3	Argentina	Latin America & the Caribbean	2020	8.51	4.87
4	Armenia	Caucasus & Central Asia	2020	8.35	7.84
...	...	...	...	...	...
3078	Uruguay	Latin America & the Caribbean	2000	9.04	7.14
3079	Venezuela, RB	Latin America & the Caribbean	2000	6.79	5.92
3080	Vietnam	South Asia	2000	5.48	5.58
3081	Zambia	Sub-Saharan Africa	2000	7.03	7.04
3082	Zimbabwe	Sub-Saharan Africa	2000	5.94	4.53

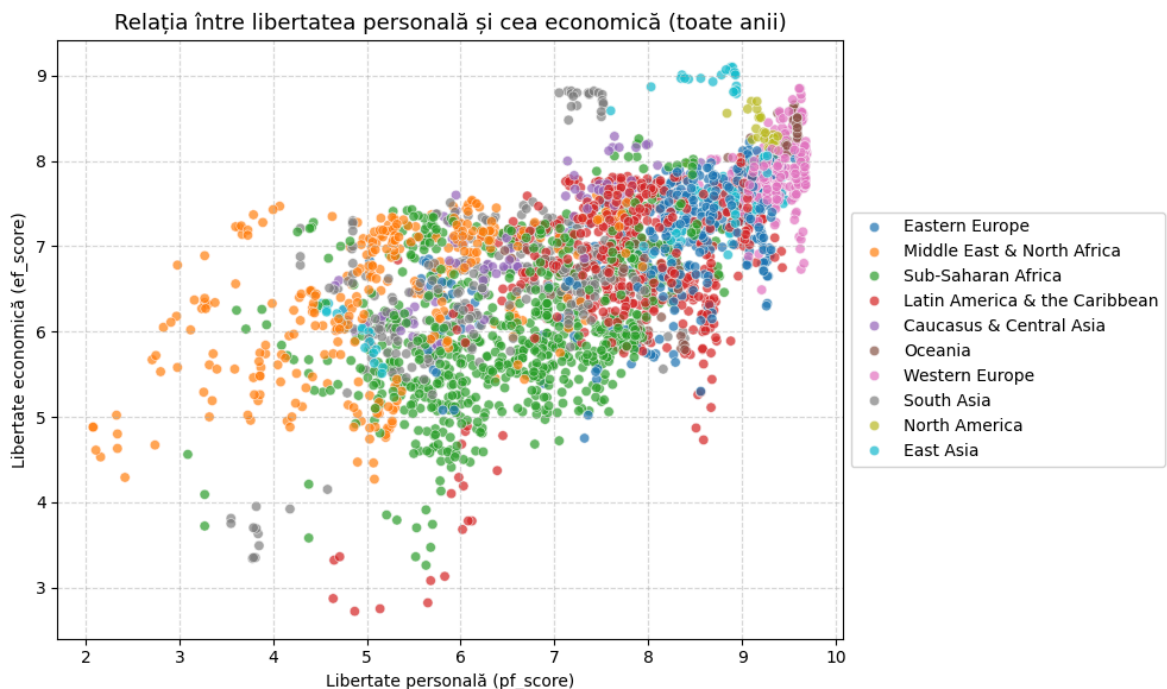
3083 rows × 5 columns

```
In [68]: # 3. Vizualizare: scatter plot complet
import matplotlib.pyplot as plt
import seaborn as sns
```



```
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df_pair_pd,
    x="pf_score",
    y="ef_score",
    hue="region",
    alpha=0.7
)

plt.title("Relația între libertatea personală și cea economică (toate anii)", fontweight='bold')
plt.xlabel("Libertate personală (pf_score)")
plt.ylabel("Libertate economică (ef_score)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```



```
In [69]: # 4. Corelație Pearson + Statistici descriptive
corr_value = df_pair_pd["pf_score"].corr(df_pair_pd["ef_score"])
print(f"Coeficient de corelație Pearson: {corr_value:.4f}")

print("\nStatistici descriptive:")
print(df_pair_pd[["pf_score", "ef_score"]].describe())
```

Coeficient de corelație Pearson: 0.6600

Statistici descriptive:

	pf_score	ef_score
count	3083.000000	3083.000000
mean	7.373876	6.806610
std	1.577346	0.991279
min	2.080000	2.720000
25%	6.120000	6.130000
50%	7.570000	6.930000
75%	8.700000	7.580000
max	9.690000	9.100000

Observație:

Analiza anterioară a inclus toate înregistrările anuale pentru fiecare țară, ceea ce oferă o imagine detaliată, dar zgomotoasă.

Pentru a înțelege relația structurală între cele două dimensiuni, vom recomputa scorurile **medii pe țară** pentru perioada 2000–2020 și vom analiza corelația la acest nivel.

```
In [70]: # 5. Calcul scoruri medii pe țări și regiuni
df_avg_scores = df_cleaned.groupby("countries", "region") \
    .agg({"pf_score": "avg", "ef_score": "avg"}) \
    .withColumnRenamed("avg(pf_score)", "avg_pf_score") \
    .withColumnRenamed("avg(ef_score)", "avg_ef_score")

df_avg_scores_pd = df_avg_scores.toPandas()

In [75]: import pandas as pd

In [81]: # 6. Scatter plot cu numele unor țări afișate
plt.figure(figsize=(12, 8))
sns.scatterplot(
    data=df_avg_scores_pd,
    x="avg_pf_score",
    y="avg_ef_score",
    hue="region",
    alpha=0.8,
    s=60
)

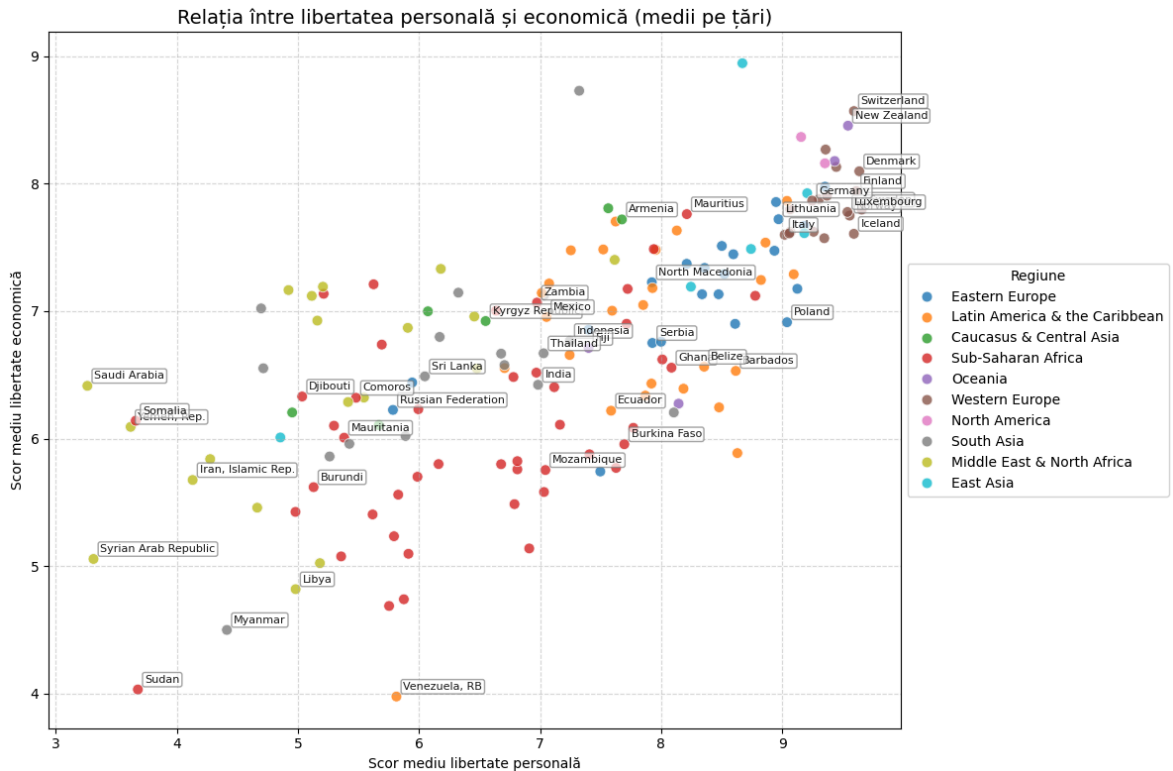
# Adăugăm etichete pentru țările extreme (wings)
# Identificăm țările cu scoruri foarte mari sau foarte mici
extreme_countries = df_avg_scores_pd[
    (df_avg_scores_pd["avg_pf_score"] >= 9.5) | # Libertate personală foarte mare
    (df_avg_scores_pd["avg_pf_score"] <= 4.0) | # Libertate personală foarte mică
    (df_avg_scores_pd["avg_ef_score"] >= 9) | # Libertate economică foarte mare
    (df_avg_scores_pd["avg_ef_score"] <= 4.5) | # Libertate economică foarte mică
    (df_avg_scores_pd["avg_pf_score"] + df_avg_scores_pd["avg_ef_score"] >= 18.0)
    (df_avg_scores_pd["avg_pf_score"] + df_avg_scores_pd["avg_ef_score"] <= 10.0)
]

# Adăugăm și câteva țări aleatorii pentru context
random_countries = df_avg_scores_pd.sample(30, random_state=42)
countries_to_label = pd.concat([extreme_countries, random_countries]).drop_duplicates()

# Adăugăm etichetele
for idx, row in countries_to_label.iterrows():
    plt.annotate(
        row["countries"],
        (row["avg_pf_score"], row["avg_ef_score"]),
        xytext=(5, 5),
        textcoords='offset points',
        fontsize=8,
        alpha=0.9,
        bbox=dict(boxstyle="round,pad=0.2", facecolor="white", alpha=0.7, edgecolor="black")
    )

plt.title("Relația între libertatea personală și economică (medii pe țări)", fontweight="bold")
plt.xlabel("Scor mediu libertate personală")
plt.ylabel("Scor mediu libertate economică")
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(title="Regiune", loc='center left', bbox_to_anchor=(1, 0.5))
```

```
plt.tight_layout()
plt.show()
```



```
In [83]: # 7. Corelație finală
corr_avg = df_avg_scores_pd["avg_pf_score"].corr(df_avg_scores_pd["avg_ef_score"])
print(f"Coeficient Pearson (medii pe țări): {corr_avg:.4f}")
```

Coeficient Pearson (medii pe țări): 0.6940

## 2.2.5 Dezechilibrul între libertatea economică și cea personală ( freedom\_gap )

**Obiectiv:** să identificăm țările în care există un dezechilibru semnificativ între libertatea economică (ef\_score) și libertatea personală (pf\_score), prin introducerea unei noi coloane calculate:  $\text{freedom\_gap} = \text{pf\_score} - \text{ef\_score}$

- Valori pozitive → țări unde libertatea personală este mai mare decât cea economică
- Valori negative → țări unde libertatea economică este mai mare decât cea personală

Această analiză oferă o perspectivă inedită asupra diferențelor structurale între societăți, scoțând în evidență țări cu politici dezechilibrate sau contradictorii între dimensiunile libertății.

Vom realiza:

- o analiză descriptivă cu Spark și Spark SQL;
- două clasamente: top 10 țări cu cel mai mare exces de libertate personală și top 10 cu exces economic;
- o vizualizare de tip bar chart pentru cele mai dezechilibrate 30 de țări.

```
In [84]: from pyspark.sql.functions import round, col
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [85]: # 1. Calculăm diferența: pf_score - ef_score
df_gap = df_cleaned.select("countries", "region", "year", "pf_score", "ef_score") \
    .dropna(subset=["pf_score", "ef_score"]) \
    .withColumn("freedom_gap", round(col("pf_score") - col("ef_score"), 2))
```

```
In [86]: # 2. Calculăm media gap-ului pe țări și regiuni
df_gap_avg = df_gap.groupBy("countries", "region") \
    .agg({"freedom_gap": "avg"}) \
    .withColumnRenamed("avg(freedom_gap)", "avg_freedom_gap") \
    .orderBy(col("avg_freedom_gap").desc())
```

```
In [46]: # 1. Calculăm diferența între scorurile de libertate personală și economică
df_gap = df_cleaned.select("countries", "region", "year", "pf_score", "ef_score") \
    .dropna(subset=["pf_score", "ef_score"]) \
    .withColumn("freedom_gap", round(col("pf_score") - col("ef_score"), 2))
```

```
In [87]: # 3. Creăm view pentru interogări SQL
df_gap_avg.createOrReplaceTempView("gap_scores")
```

```
In [88]: # 4. Top 10 țări cu libertate personală > economică
print("Top 10 țări: libertate personală mai mare")
spark.sql("""
SELECT countries, region, ROUND(avg_freedom_gap, 2) as gap
FROM gap_scores
WHERE avg_freedom_gap > 0
ORDER BY gap DESC
LIMIT 10
""").show()
```

Top 10 țări: libertate personală mai mare

countries	region	gap
Argentina	Latin America & t...	2.74
Suriname	Latin America & t...	2.23
Poland	Eastern Europe	2.13
Barbados	Latin America & t...	2.09
Iceland	Western Europe	1.99
Slovenia	Eastern Europe	1.95
Timor-Leste	South Asia	1.9
Papua New Guinea	Oceania	1.87
Sweden	Western Europe	1.86
Malawi	Sub-Saharan Africa	1.86

```
In [89]: # 5. Top 10 țări cu libertate economică > personală
print("Top 10 țări: libertate economică mai mare")
spark.sql("""
SELECT countries, region, ROUND(avg_freedom_gap, 2) as gap
FROM gap_scores
WHERE avg_freedom_gap < 0
ORDER BY gap ASC
```

```
LIMIT 10
""").show()
```

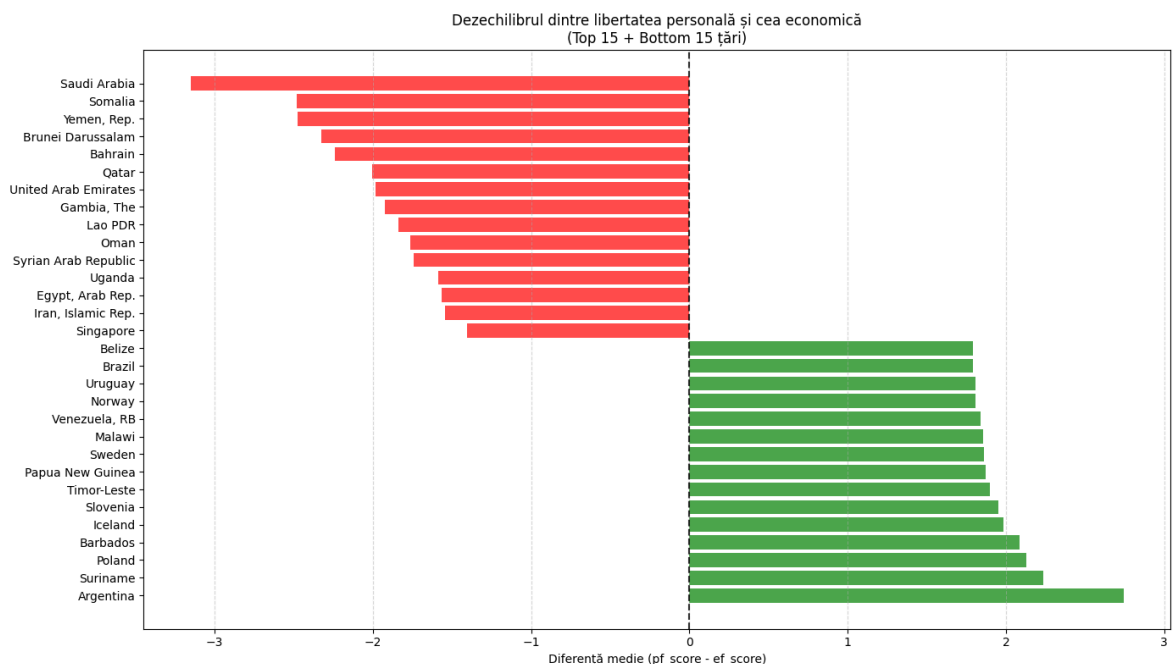
Top 10 țări: libertate economică mai mare

countries	region	gap
Saudi Arabia	Middle East & Nor...	-3.15
Somalia	Sub-Saharan Africa	-2.48
Yemen, Rep.	Middle East & Nor...	-2.47
Brunei Darussalam	South Asia	-2.32
Bahrain	Middle East & Nor...	-2.24
Qatar	Middle East & Nor...	-2.0
United Arab Emirates	Middle East & Nor...	-1.98
Gambia, The	Sub-Saharan Africa	-1.92
Lao PDR	South Asia	-1.84
Oman	Middle East & Nor...	-1.76

```
In [90]: # 6. Pregătim vizualizarea cu cele mai dezechilibrate cazuri
df_gap_plot = df_gap_avg.toPandas()
```

```
In [91]: # Selectăm cele mai dezechilibrate 30 de țări (15 pozitive, 15 negative)
top_positive = df_gap_plot.nlargest(15, 'avg_freedom_gap')
top_negative = df_gap_plot.nsmallest(15, 'avg_freedom_gap')
extreme_gaps = pd.concat([top_positive, top_negative]).sort_values("avg_freedom_
```

```
In [92]: # 7. Bar chart cu colorare în funcție de semn
plt.figure(figsize=(14, 8))
colors = ['green' if x > 0 else 'red' for x in extreme_gaps['avg_freedom_gap']]
plt.barh(range(len(extreme_gaps)), extreme_gaps['avg_freedom_gap'], color=colors)
plt.yticks(range(len(extreme_gaps)), extreme_gaps['countries'])
plt.title("Dezechilibrul dintre libertatea personală și cea economică\n(Top 15 + ")
plt.xlabel("Diferență medie (pf_score - ef_score)")
plt.axvline(x=0, color='black', linestyle='--', alpha=0.8)
plt.grid(axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



## 2.2.6 Țările cu cel mai scăzut nivel al libertății de exprimare

### Descriere și obiectiv

În această analiză dorim să identificăm țările cu cel mai scăzut nivel al libertății de exprimare în perioada 2000–2020, folosind indicatorul `pf_expression_direct`.

Acest scor reflectă nivelul de represiune exercitat de autorități asupra jurnaliștilor, activiștilor și cetățenilor, incluzând:

- arestări politice;
- amenințări și intimidări sistematice;
- acte de violență împotriva libertății de opinie.

Prin calculul mediei pe 20 de ani, obținem o imagine de ansamblu stabilă asupra climatului represiv din diferite țări.

Vor fi prezentate:

- Top 10 țări cu cel mai scăzut scor mediu la `pf_expression_direct`;
- o vizualizare de tip bar chart, cu segmentare pe regiuni.

```
In [93]: from pyspark.sql.functions import col
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [95]: # 1. Selectăm coloanele relevante și eliminăm valorile lipsă
df_expr = df_cleaned.select("countries", "year", "region", "pf_expression_direct")
df_expr.dropna(subset=["pf_expression_direct"])
```

```
In [96]: # 2. Creăm view temporar pentru interogare SQL
df_expr.createOrReplaceTempView("expr_view")
```

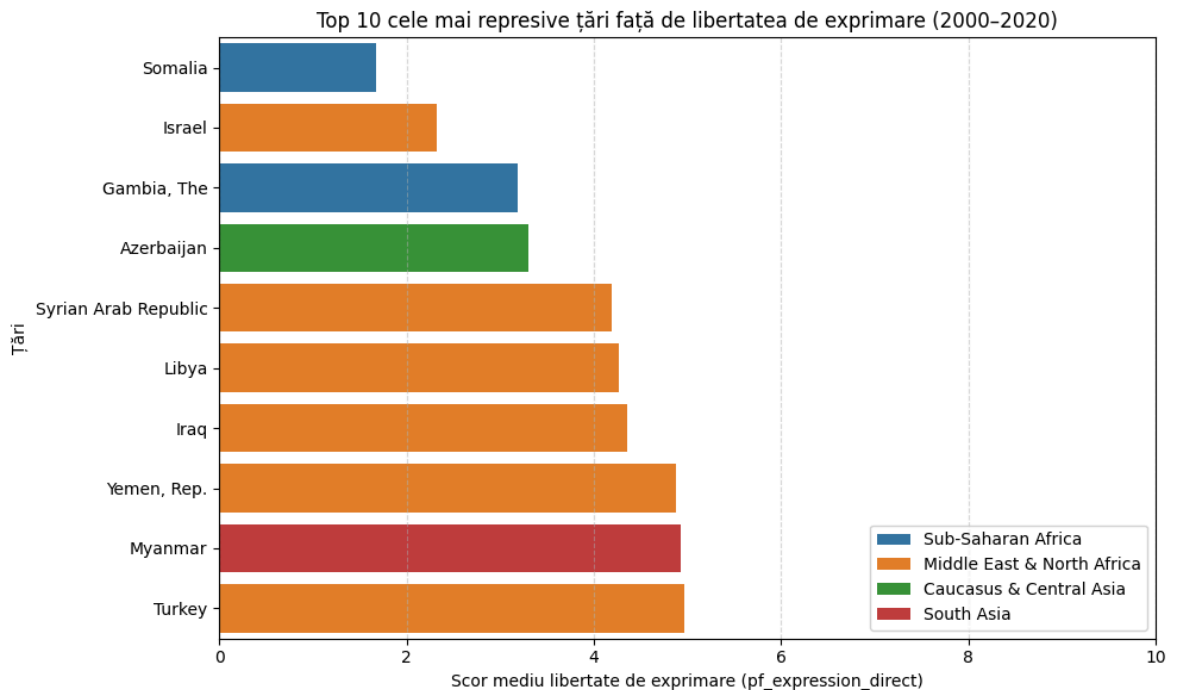
```
In [97]: # 3. Calculăm media scorului între 2000 și 2020 pentru fiecare țară
expr_avg_df = spark.sql("""
    SELECT countries, region, ROUND(AVG(pf_expression_direct), 2) AS avg_expression_score
    FROM expr_view
    WHERE year BETWEEN 2000 AND 2020
    GROUP BY countries, region
    ORDER BY avg_expression_score ASC
    LIMIT 10
""")

expr_avg_df.show(truncate=False)
```

countries	region	avg_expression_score
Somalia	Sub-Saharan Africa	1.67
Israel	Middle East & North Africa	2.32
Gambia, The	Sub-Saharan Africa	3.18
Azerbaijan	Caucasus & Central Asia	3.3
Syrian Arab Republic	Middle East & North Africa	4.19
Libya	Middle East & North Africa	4.27
Iraq	Middle East & North Africa	4.36
Yemen, Rep.	Middle East & North Africa	4.87
Myanmar	South Asia	4.93
Turkey	Middle East & North Africa	4.97




In [98]: *# 4. Conversie pentru vizualizare*  
 expr\_avg\_pd = expr\_avg\_df.toPandas()

In [99]: *# 5. Plot Top 10 cele mai represive țări*  
 plt.figure(figsize=(10, 6))  
 sns.barplot(data=expr\_avg\_pd, y="countries", x="avg\_expression\_score", hue="region")  
 plt.title("Top 10 cele mai represive țări față de libertatea de exprimare (2000-2020)")  
 plt.xlabel("Scor mediu libertate de exprimare (pf\_expression\_direct)")  
 plt.ylabel("Țări")  
 plt.xlim(0, 10)  
 plt.grid(True, axis='x', linestyle='--', alpha=0.5)  
 plt.legend(loc='lower right')  
 plt.tight\_layout()  
 plt.show()



## 2.2.7 Clasificarea țărilor în categorii de libertate: High / Medium / Low

Descriere și obiectiv: Scorul compozit de libertate (hf\_score) este un indicator complex, dar poate deveni mai intuitiv dacă îl clasificăm în categorii semnificative. Astfel, vom eticheta fiecare observație în una dintre cele trei clase:

-  **High Freedom:** scor  $\geq 8.5$
-  **Medium Freedom:** între 6.5 și 8.5
-  **Low Freedom:** scor  $< 6.5$

Această etichetare ne va permite:

- să analizăm distribuția libertății în funcție de regiune;
- să observăm evoluția categoriilor în timp (2000–2020);
- să construim o vizualizare stacked care evidențiază dinamica libertății la nivel global.

Vom folosi atât Spark SQL, cât și PySpark DataFrames, iar pentru vizualizare – Pandas + Matplotlib.

```
In [100... from pyspark.sql.functions import when, col
import matplotlib.pyplot as plt
```

```
In [101... # 1. Clasificare scoruri in categorii
df_categorized = df_cleaned.withColumn(
    "freedom_category",
    when(col("hf_score") >= 8.5, "High")
    .when((col("hf_score") >= 6.5) & (col("hf_score") < 8.5), "Medium")
    .otherwise("Low")
)
```

```
In [102... # 2. Creăm view temporar pentru SQL
df_categorized.createOrReplaceTempView("freedom_categorized")
```

```
In [103... # 3. Distribuție per regiune
spark.sql("""
    SELECT region, freedom_category, COUNT(*) as count
    FROM freedom_categorized
    GROUP BY region, freedom_category
    ORDER BY region, freedom_category
""").show(truncate=False)
```



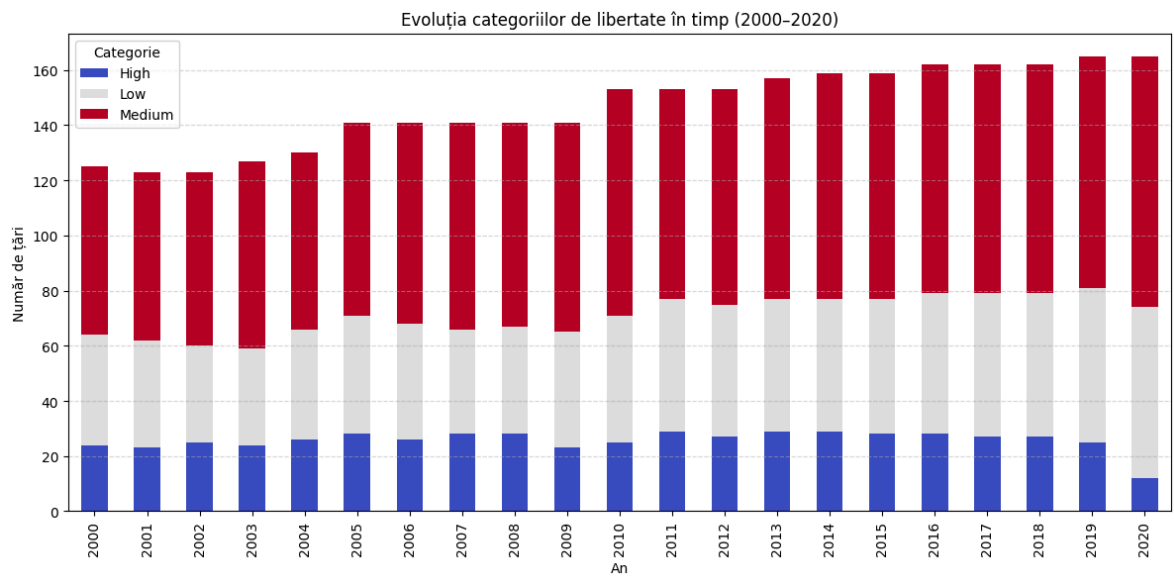
region	freedom_category	count
Caucasus & Central Asia	Low	40
Caucasus & Central Asia	Medium	56
East Asia	High	48
East Asia	Low	21
East Asia	Medium	53
Eastern Europe	High	63
Eastern Europe	Low	29
Eastern Europe	Medium	331
Latin America & the Caribbean	High	19
Latin America & the Caribbean	Low	29
Latin America & the Caribbean	Medium	488
Middle East & North Africa	Low	259
Middle East & North Africa	Medium	71
North America	High	40
North America	Medium	2
Oceania	High	42
Oceania	Medium	42
South Asia	Low	148
South Asia	Medium	150
Sub-Saharan Africa	Low	427

only showing top 20 rows

```
In [104... # 4. Distribuție anuală
df_yearly_dist = spark.sql("""
    SELECT year, freedom_category, COUNT(*) as count
    FROM freedom_categorized
    GROUP BY year, freedom_category
    ORDER BY year ASC, freedom_category
    """)
```

```
In [105... # 5. Convertim la Pandas + pivot
df_yearly_dist_pd = df_yearly_dist.toPandas()
df_pivot = df_yearly_dist_pd.pivot(index="year", columns="freedom_category", val
```

```
In [106... # 6. Plot stacked bars
df_pivot.plot(kind="bar", stacked=True, colormap="coolwarm", figsize=(12,6))
plt.title("Evoluția categoriilor de libertate în timp (2000-2020)")
plt.xlabel("An")
plt.ylabel("Număr de țări")
plt.grid(True, axis='y', linestyle='--', alpha=0.5)
plt.legend(title="Categorie")
plt.tight_layout()
plt.show()
```



In [107...

```
columns_list = df_cleaned.columns
print(f"Coloane din DataFrame-ul curățat ({len(columns_list)} total):")
print(", ".join(columns_list))
```

Coloane din DataFrame-ul curățat (136 total):

year, countries, region, hf\_score, hf\_rank, hf\_quartile, pf\_rol\_vdem, pf\_rol, pf\_ss\_homicide, pf\_ss\_homicide\_data, pf\_ss\_disappearances\_disap, pf\_ss\_disappearances\_violent, pf\_ss\_disappearances\_violent\_data, pf\_ss\_disappearances\_organized, pf\_ss\_disappearances\_fatalities, pf\_ss\_disappearances\_fatalities\_data, pf\_ss\_disappearances\_injuries, pf\_ss\_disappearances\_injuries\_data, pf\_ss\_disappearances\_torture, pf\_ss\_killings, pf\_ss\_disappearances, pf\_ss, pf\_movement\_vdem\_foreign, pf\_movement\_vdem\_men, pf\_movement\_vdem\_women, pf\_movement\_vdem, pf\_movement\_cld, pf\_movement, pf\_religion\_freedom\_vdem, pf\_religion\_freedom\_cld, pf\_religion\_freedom, pf\_religion\_suppression, pf\_religion, pf\_assembly\_entry, pf\_assembly\_freedom\_house, pf\_assembly\_freedom\_bti, pf\_assembly\_freedom\_cld, pf\_assembly\_freedom, pf\_assembly\_parties\_barriers, pf\_assembly\_parties\_bans, pf\_assembly\_parties\_auton, pf\_assembly\_parties, pf\_assembly\_civil, pf\_assembly, pf\_expression\_direct\_killed, pf\_expression\_direct\_killed\_data, pf\_expression\_direct\_jailed, pf\_expression\_direct\_jailed\_data, pf\_expression\_direct, pf\_expression\_vdem\_cultural, pf\_expression\_vdem\_harass, pf\_expression\_vdem\_gov, pf\_expression\_vdem\_internet, pf\_expression\_vdem\_selfcens, pf\_expression\_vdem, pf\_expression\_house, pf\_expression\_bti, pf\_expression\_cld, pf\_expression, pf\_identity\_same\_m, pf\_identity\_same\_f, pf\_identity\_same, pf\_identity\_divorce, pf\_identity\_inheritance, pf\_identity\_fgm, pf\_identity, pf\_score, pf\_rank, ef\_government\_consumption, ef\_government\_consumption\_data, ef\_government\_transfers, ef\_government\_transfers\_data, ef\_government\_investment, ef\_government\_investment\_data, ef\_government\_tax\_income, ef\_government\_tax\_income\_data, ef\_government\_tax\_payroll, ef\_government\_tax\_payroll\_data, ef\_government\_tax, ef\_government\_soa, ef\_government, ef\_legal\_judicial, ef\_legal\_courts, ef\_legal\_protection, ef\_legal\_military, ef\_legal\_integrity, ef\_legal\_enforcement, ef\_legal\_regulatory, ef\_legal\_police, ef\_gender, ef\_legal, ef\_money\_growth, ef\_money\_growth\_data, ef\_money\_sd, ef\_money\_sd\_data, ef\_money\_inflation, ef\_money\_inflation\_data, ef\_money\_currency, ef\_money, ef\_trade\_tariffs\_revenue, ef\_trade\_tariffs\_revenue\_data, ef\_trade\_tariffs\_mean, ef\_trade\_tariffs\_mean\_data, ef\_trade\_tariffs\_sd, ef\_trade\_tariffs\_sd\_data, ef\_trade\_tariffs, ef\_trade\_regulatory\_nontariff, ef\_trade\_regulatory\_compliance, ef\_trade\_regulatory, ef\_trade\_black, ef\_trade\_movement\_open, ef\_trade\_movement\_capital, ef\_trade\_movement\_visit, ef\_trade\_movement, ef\_trade, ef\_regulation\_credit\_ownership, ef\_regulation\_credit\_private, ef\_regulation\_credit\_interest, ef\_regulation\_credit, ef\_regulation\_labor\_minwage, ef\_regulation\_labor\_firing, ef\_regulation\_labor\_bargain, ef\_regulation\_labor\_hours, ef\_regulation\_labor\_dismissal, ef\_regulation\_labor\_conscription, ef\_regulation\_labor, ef\_regulation\_business\_adm, ef\_regulation\_business\_burden, ef\_regulation\_business\_start, ef\_regulation\_business\_impartial, ef\_regulation\_business\_licensing, ef\_regulation\_business\_compliance, ef\_regulation\_business, ef\_regulation, ef\_score, ef\_rank

## 3. Aplicarea metodelor de Machine Learning cu Spark MLlib

### 3.1 Predicția scorului de libertate umană (hf\_score) – Regresie

#### Obiectiv

În această secțiune vom construi un model de regresie care va avea ca scop **prezicerea scorului total de libertate umană (hf\_score)** pentru o țară într-un anumit an, utilizând variabile socio-politice și economice disponibile în setul de date.

Ne propunem să obținem:

- un model robust și interpretabil;
- o înțelegere a **factorilor determinanți** asupra `hf_score` ;
- o evaluare a performanței modelului pe un set de testare;
- o analiză a **importanței variabilelor** pentru transparență și insight.

Vom folosi **Spark MLlib**, urmând o abordare modulară și reproductibilă, folosind un **pipeline complet**.

## Metodologie

1. Alegerea coloanelor explicative (excludem: `hf_score` , `hf_rank` , `countries` , `region` , `year` )
2. Eliminarea observațiilor incomplete
3. Indexarea variabilei categorice `region`
4. Combinarea feature-urilor într-un vector (VectorAssembler)
5. Împărțirea datelor în seturi de antrenare și test (80/20)
6. Antrenarea modelului `RandomForestRegressor`
7. Evaluarea performanței modelului folosind:
  - **MAE** (Mean Absolute Error)
  - **RMSE** (Root Mean Squared Error)
  - **R<sup>2</sup>** (coeficientul de determinare)
8. Extracția și interpretarea importanței variabilelor

```
In [128... from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import RegressionEvaluator
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.sql.functions import mean
```

```
In [129... # 1. Selectăm coloanele explicative (numere continue)
excluded_cols = ["countries", "hf_score", "hf_rank", "hf_quartile", "region", "y
all_features = [
    c for c in df_cleaned.columns
    if c not in excluded_cols and df_cleaned.schema[c].dataType.simpleString() =
]
```

```
In [130... # 2. Completăm valorile lipsă cu media fiecărei coloane numerice
fill_dict = {}
for feature in all_features:
    mean_val = df_cleaned.select(mean(col(feature))).first()[0]
    fill_dict[feature] = float(mean_val)

# Dacă vrem, putem include și completare pentru hf_score (opțional, doar pentru
mean_hf_score = df_cleaned.select(mean("hf_score")).first()[0]
fill_dict["hf_score"] = float(mean_hf_score)

# Completăm NaN-urile
df_model = df_cleaned.fillna(fill_dict)

# Dacă există și categorii lipsă în "region", completăm și acolo
```

```
df_model = df_model.fillna({"region": "Unknown"})

print("Număr total de observații după completare:", df_model.count())
```

Număr total de observații după completare: 3083

```
In [131... # 3. Indexare coloană categorică 'region'
region_indexer = StringIndexer(inputCol="region", outputCol="region_index", hand
```

```
In [132... # 4. Combinare feature-uri într-un vector
assembler = VectorAssembler(inputCols=all_features + ["region_index"], outputCol
```

```
In [133... # 5. Definirea modelului
rf = RandomForestRegressor(featuresCol="features", labelCol="hf_score", numTrees
```

```
In [134... # 6. Pipeline complet
pipeline = Pipeline(stages=[region_indexer, assembler, rf])
```

```
In [135... # 7. Împărțirea datelor
train_data, test_data = df_model.randomSplit([0.8, 0.2], seed=42)
print(f"Train: {train_data.count()} rânduri | Test: {test_data.count()} rânduri"
```

Train: 2511 rânduri | Test: 572 rânduri

```
In [136... # 8. Antrenarea modelului
model = pipeline.fit(train_data)
```

```
In [139... # 9. Predictii pe test
predictions = model.transform(test_data)
```

```
In [140... # 10. Evaluare performanță
mae = RegressionEvaluator(labelCol="hf_score", predictionCol="prediction", metri
rmse = RegressionEvaluator(labelCol="hf_score", predictionCol="prediction", metr
r2 = RegressionEvaluator(labelCol="hf_score", predictionCol="prediction", metric

print("\nEvaluare model Random Forest Regressor:")
print(f"MAE: {mae.evaluate(predictions):.4f}")
print(f"RMSE: {rmse.evaluate(predictions):.4f}")
print(f"R²: {r2.evaluate(predictions):.4f}")
```

Evaluare model Random Forest Regressor:

MAE: 0.0480

RMSE: 0.0733

R²: 0.9965

```
In [141... # 11. Importanța variabilelor
rf_model = model.stages[-1] # extragem modelul Random Forest
importances = rf_model.featureImportances.toArray()
features_final = all_features + ["region_index"]

importances_df = pd.DataFrame({
    "feature": features_final,
    "importance": importances
}).sort_values(by="importance", ascending=False)

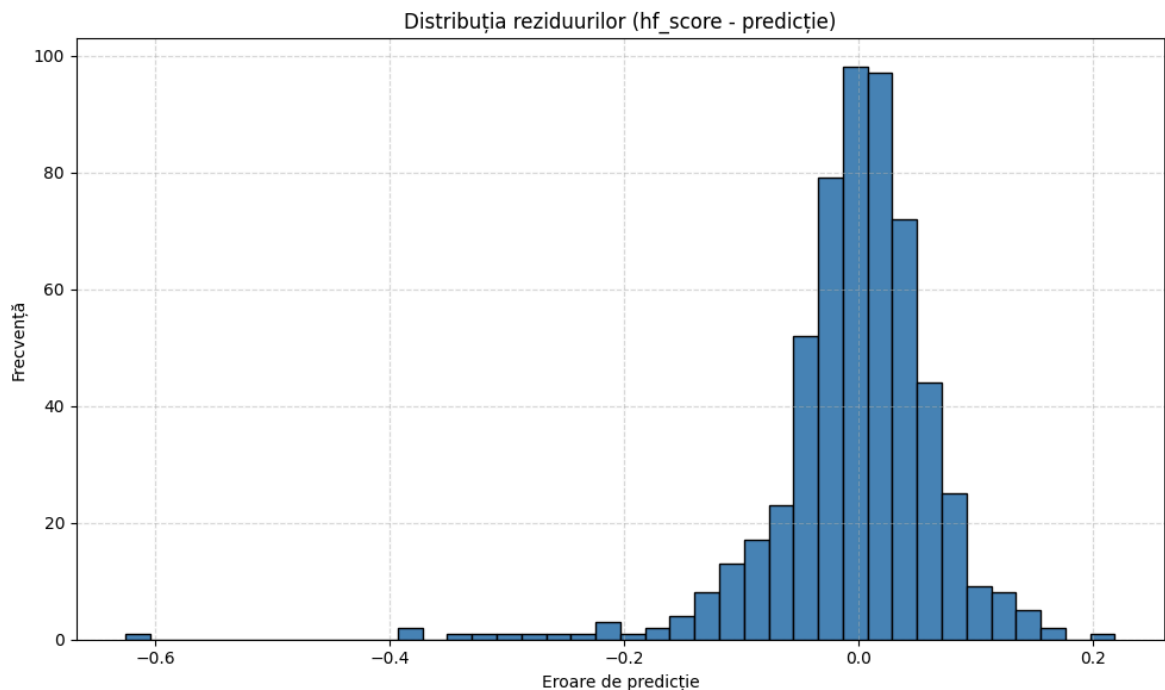
print("\nTop 10 cele mai importante variabile pentru predicția hf_score:")
print(importances_df.head(10))
```

Top 10 cele mai importante variabile pentru predicția hf\_score:

	feature	importance
58	pf_score	0.322954
59	pf_rank	0.203716
31	pf_assembly_freedom	0.093484
0	pf_rol_vdem	0.091060
37	pf_assembly	0.060781
50	pf_expression	0.029668
123	ef_score	0.029553
28	pf_assembly_freedom_house	0.018893
80	ef_legal	0.016846
124	ef_rank	0.016815

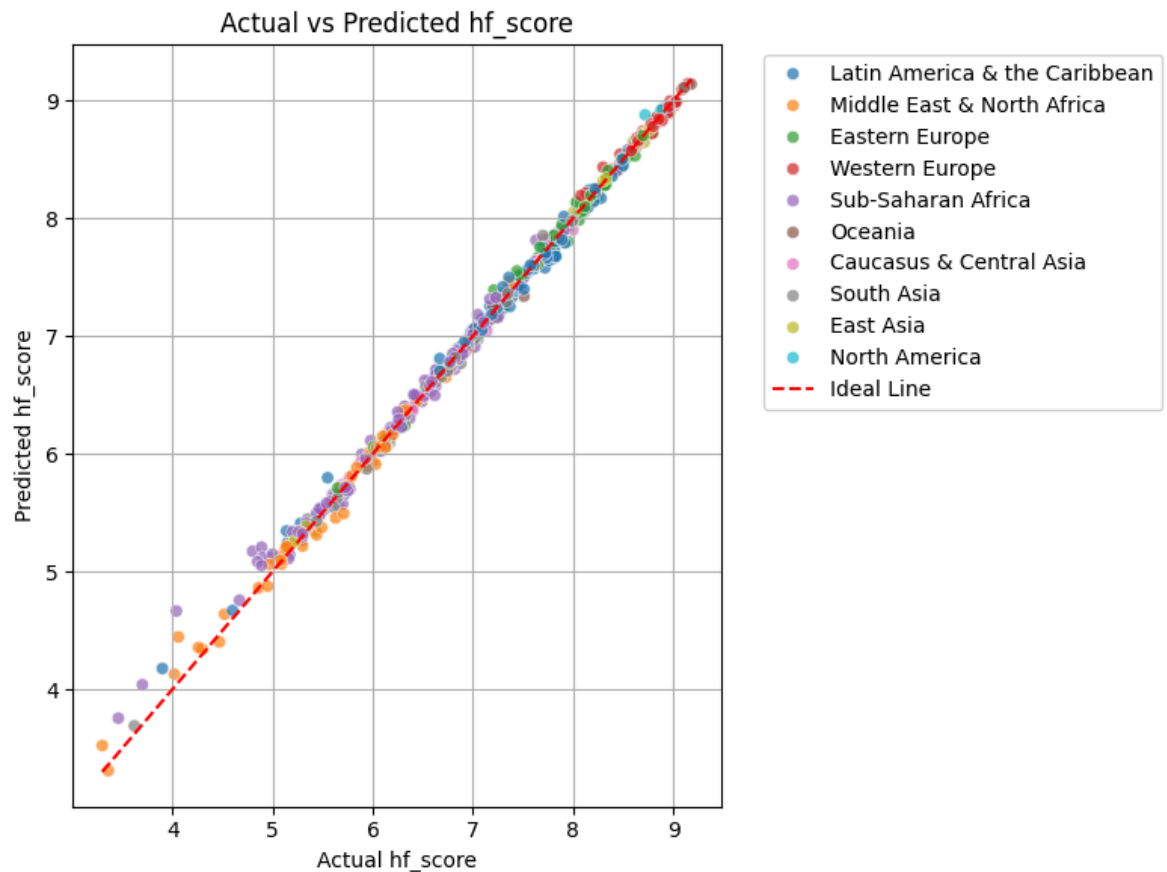
```
In [142... # 12. Convertim La Pandas pentru analiză de erori
predictions_pd = predictions.select("countries", "region", "hf_score", "prediction")
predictions_pd["residual"] = predictions_pd["hf_score"] - predictions_pd["prediction"]
```

```
In [143... # 13. Vizualizare: Erori de predicție (opțional)
plt.figure(figsize=(10, 6))
plt.hist(predictions_pd["residual"], bins=40, color="steelblue", edgecolor="black")
plt.title("Distribuția reziduurilor (hf_score - predicție)")
plt.xlabel("Eroare de predicție")
plt.ylabel("Frecvență")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



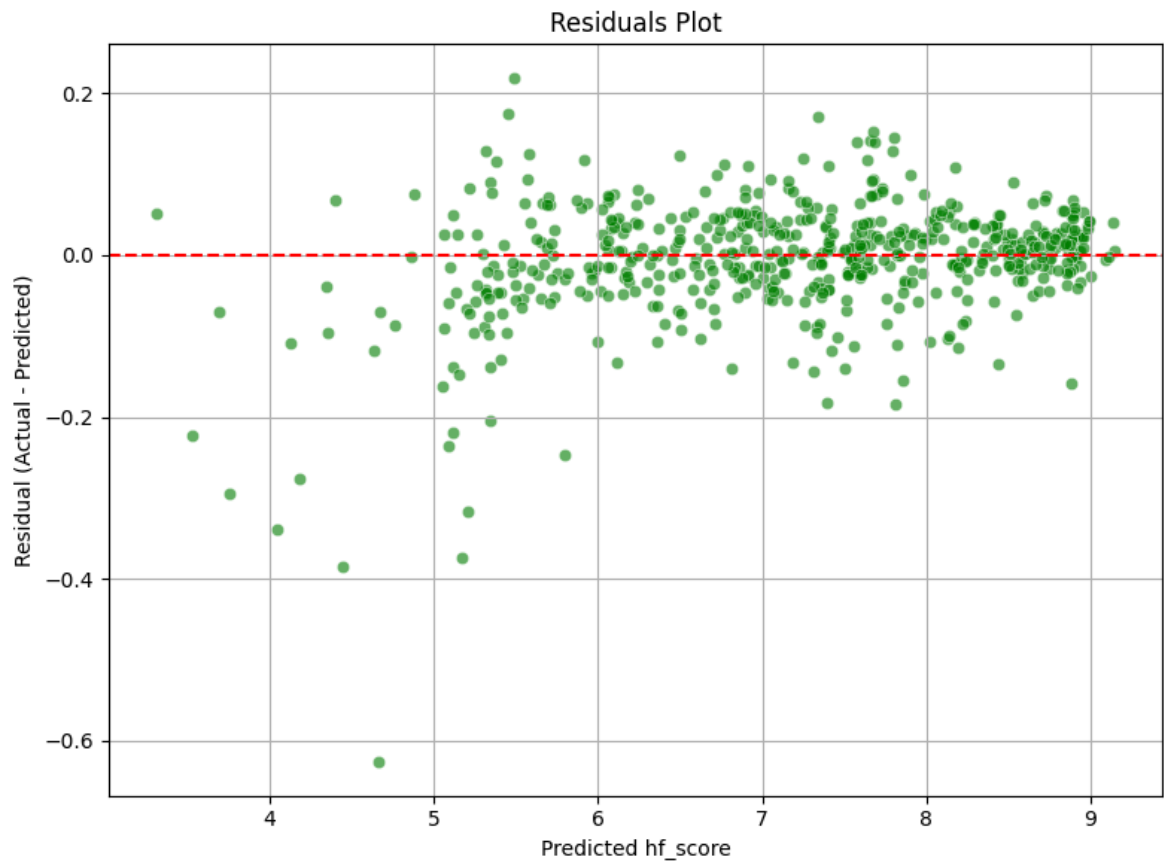
```
In [144... # 1. Scatter Plot: Actual vs Predicted
plt.figure(figsize=(8, 6))
sns.scatterplot(data=predictions_pd, x="hf_score", y="prediction", hue="region")
plt.plot([predictions_pd["hf_score"].min(), predictions_pd["hf_score"].max()],
         [predictions_pd["hf_score"].min(), predictions_pd["hf_score"].max()],
         'r--', label="Ideal Line")
plt.xlabel("Actual hf_score")
plt.ylabel("Predicted hf_score")
plt.title("Actual vs Predicted hf_score")
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



In [145...

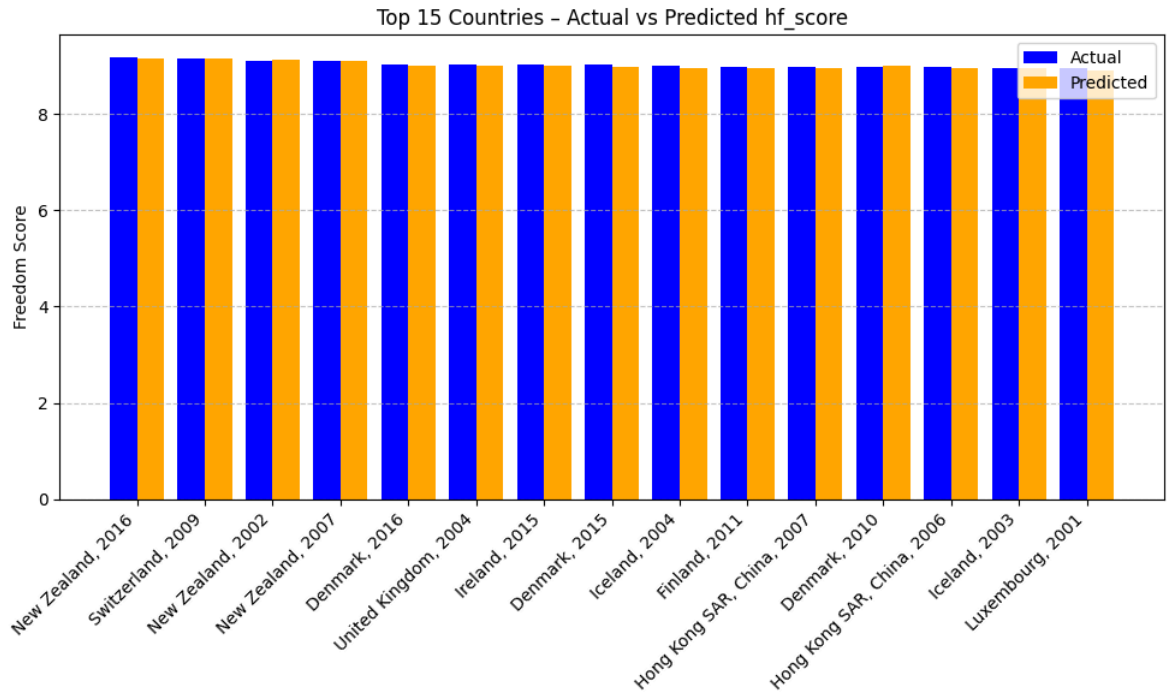
```
# 2. Residuals Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(data=predictions_pd, x="prediction", y="residual", color='green')
plt.axhline(0, linestyle='--', color='red')
plt.xlabel("Predicted hf_score")
plt.ylabel("Residual (Actual - Predicted)")
plt.title("Residuals Plot")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [146...

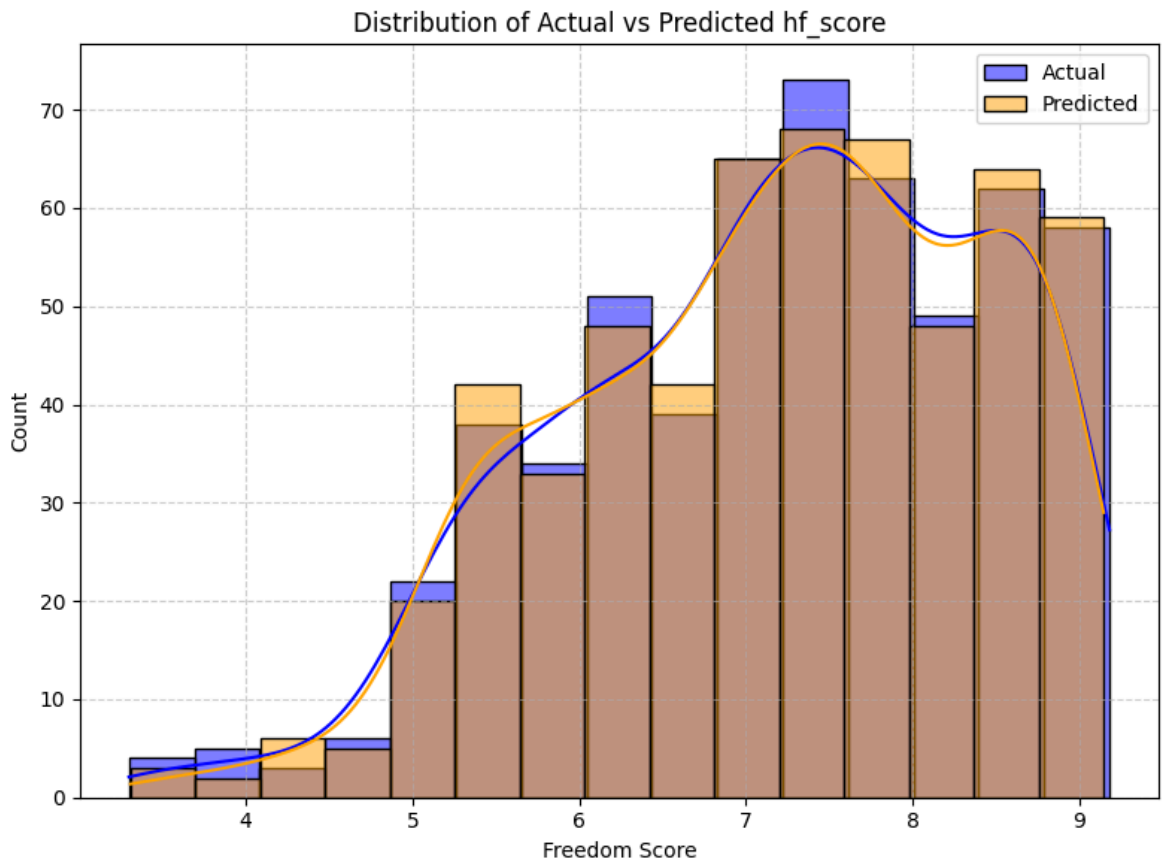
```
# 3. Bar Plot: Top 15 countries by actual hf_score
top_15 = predictions_pd.sort_values(by="hf_score", ascending=False).head(15)
plt.figure(figsize=(10, 6))
x = range(len(top_15))
plt.bar([i - 0.2 for i in x], top_15["hf_score"], width=0.4, label="Actual", col
plt.bar([i + 0.2 for i in x], top_15["prediction"], width=0.4, label="Predicted"
plt.xticks(x, top_15["countries"] + ", " + top_15["year"].astype(str), rotation=
plt.ylabel("Freedom Score")
plt.title("Top 15 Countries - Actual vs Predicted hf_score")
plt.legend()
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```





In [147...

```
# 4. Histogram comparison
plt.figure(figsize=(8, 6))
sns.histplot(predictions_pd["hf_score"], color="blue", label="Actual", kde=True,
sns.histplot(predictions_pd["prediction"], color="orange", label="Predicted", kd
plt.xlabel("Freedom Score")
plt.title("Distribution of Actual vs Predicted hf_score")
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



In [148...

```

import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px

# 1. Media scorului prezis pe fiecare țară
avg_pred = (
    predictions_pd
    .groupby("countries", as_index=False)["prediction"]
    .mean()
    .rename(columns={"countries": "country", "prediction": "avg_pred"})
)

# 2. Corecții de nume
name_fix = {
    "United States": "United States of America",
    "Russian Federation": "Russia",
    "Korea, Rep.": "South Korea",
    "Czech Republic": "Czechia",
    "Slovak Republic": "Slovakia",
    "Egypt, Arab Rep.": "Egypt",
    "Venezuela, RB": "Venezuela",
    "Iran, Islamic Rep.": "Iran",
    "Gambia, The": "Gambia",
}
avg_pred["country"] = avg_pred["country"].replace(name_fix)

# 3. Încercăm să încărcăm Natural Earth; dacă nu există -> fallback GeoJSON
try:
    world_path = gpd.datasets.get_path("naturalearth_lowres") # funcție
    world = gpd.read_file(world_path)
    country_col = "name"
except (AttributeError, KeyError):
    url = "https://raw.githubusercontent.com/datasets/geo-countries/master/data/"
    world = gpd.read_file(url)
    # în acest GeoJSON numele țării e stocat în 'ADMIN'
    world = world.rename(columns={"ADMIN": "name"})
    country_col = "name"

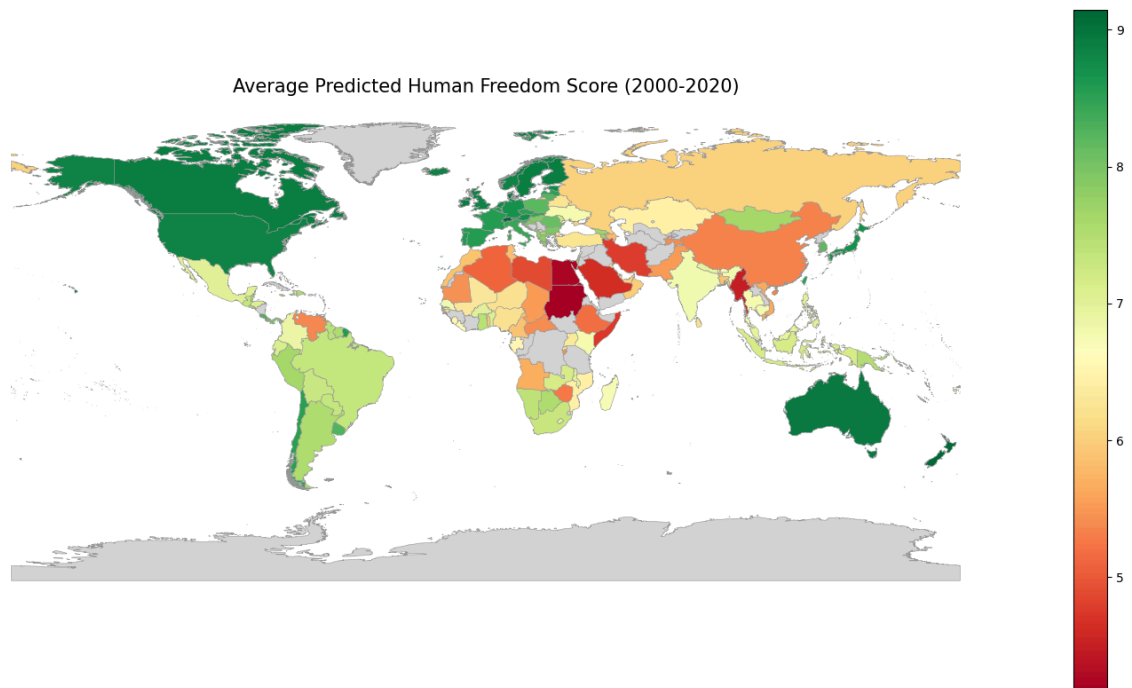
# 4. Merge
world_merge = world.merge(avg_pred, left_on=country_col, right_on="country", how="left")

# 5-A. Hartă statică
fig, ax = plt.subplots(1, 1, figsize=(15, 8))
world_merge.plot(
    column="avg_pred",
    cmap="RdYlGn",
    linewidth=0.4,
    ax=ax,
    edgecolor="0.6",
    legend=True,
    missing_kwds={
        "color": "lightgrey",
        "label": "No data",
        "edgecolor": "0.6",
    },
)
ax.set_title("Average Predicted Human Freedom Score (2000-2020)", fontsize=15)
ax.axis("off")

```

```
plt.tight_layout()
plt.show()

# 5-B. Hartă interactivă
fig = px.choropleth(
    avg_pred,
    locations="country",
    locationmode="country names",
    color="avg_pred",
    color_continuous_scale="RdYlGn",
    range_color=[4, 9],
    title="Average Predicted Human Freedom Score (2000-2020)",
    labels={"avg_pred": "Pred. HF Score"},
    hover_name="country",
)
fig.update_layout(height=600, margin=dict(r=0, t=50, l=0, b=0))
fig.show()
```



## 3.2 Clasificarea nivelului de libertate (freedom\_category) – Clasificare

### Obiectiv general

Construirea unui model de clasificare care încadrează fiecare observație într-una dintre cele trei **categorii de libertate**:

- **High Freedom** (  $hf\_score \geq 8.5$  )
- **Medium Freedom** (  $6.5 \leq hf\_score < 8.5$  )
- **Low Freedom** (  $hf\_score < 6.5$  )

Scopul este să anticipăm nivelul de libertate al unei țări, bazându-ne exclusiv pe variabile socio-politice și economice, fără a folosi direct scorul numeric.

## Model ales: Logistic Regression (multinomială)

- Oferă o interpretare transparentă a influenței fiecărui factor.
- Potrivit pentru probleme de clasificare multi-clasă.
- Poate fi echilibrat prin ponderi automate, pentru a gestiona clasele dezechilibrate.

## Etape metodologice

1. **Etichetare:** creăm coloana `freedom_category` pe baza lui `hf_score` ;
2. **Excluderi:** eliminăm coloanele ce pot influența artificial modelul ( `hf_score` , `pf_score` , etc.);
3. **Preprocesare:** indexăm regiunea, vectorizăm feature-urile numerice;
4. **Ponderare:** atribuim ponderi inverse pentru clasele dezechilibrate;
5. **Antrenare model:** Logistic Regression multinomială cu Grid Search și Cross-Validation (F1 ca scor de optimizare);
6. **Evaluare:** folosim accuracy, F1, precision, recall;
7. **Salvare și încărcare model final** pentru folosire ulterioară.

In [149...

```
from pyspark.sql.functions import when, col, count, lit
from pyspark.ml.feature import VectorAssembler, StringIndexer
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, BinaryClass
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import udf, col
from pyspark.sql.functions import when
```

In [150...

```
# 1. Etichetare în High / Medium / Low
df_labeled = (
    df_cleaned
    .withColumn(
        "freedom_category",
        when(col("hf_score") >= 8.5, "High")
        .when((col("hf_score") >= 6.5) & (col("hf_score") < 8.5), "Medium")
        .otherwise("Low")
    )
)
```

In [151...

```
# 2. Selectăm coloanele relevante (excludem scorurile directe și altele redundan
excluded_cols = [
    "countries", "hf_score", "hf_rank", "hf_quartile",
    "pf_score", "pf_rank", "ef_score", "ef_rank",
    "freedom_category"
]
all_features = [
    c for c in df_labeled.columns
    if c not in excluded_cols and df_labeled.schema[c].dataType.simpleString() i
]
```

```

In [152... # 3. Eliminăm observațiile incomplete
df_nonnull = df_labeled.dropna(subset=all_features + ["freedom_category", "region"])

In [153... # 4. Indexăm eticheta (categorii de Libertate)
cat_indexer = StringIndexer(inputCol="freedom_category", outputCol="label", handleMissing="ignore")
df_indexed = cat_indexer.fit(df_nonnull).transform(df_nonnull)

In [154... # 5. Calculăm ponderi inverse pentru clase dezechilibrate
cnts = df_indexed.groupBy("label").agg(count("*").alias("cnt")).orderBy("label")
total = cnts["cnt"].sum()
n_classes = cnts.shape[0]
weights_d = {r["label"]: (total / (n_classes * r["cnt"])) for _, r in cnts.iter_rows()}

df_weighted = (
    df_indexed
    .withColumn(
        "sample_weight",
        when(col("label") == 0.0, lit(weights_d[0.0]))
        .when(col("label") == 1.0, lit(weights_d[1.0]))
        .otherwise(lit(weights_d[2.0]))
    )
)

In [155... # 6. Pipeline complet
region_indexer = StringIndexer(inputCol="region", outputCol="region_index", handleMissing="ignore")
assembler = VectorAssembler(inputCols=all_features + ["region_index"], outputCol="features")
lr = LogisticRegression(
    featuresCol="features",
    labelCol="label",
    weightCol="sample_weight",
    family="multinomial",
    maxIter=200,
    tol=1e-6
)
pipeline = Pipeline(stages=[region_indexer, assembler, lr])

In [156... # 7. Împărțire în train/test
train_data, test_data = df_weighted.randomSplit([0.8, 0.2], seed=42)

In [157... # 8. ParamGrid + CrossValidation
param_grid = (
    ParamGridBuilder()
    .addGrid(lr.regParam, [0.001, 0.01, 0.1])
    .addGrid(lr.elasticNetParam, [0.0, 0.3, 0.7])
    .build()
)
eval_f1 = MulticlassClassificationEvaluator(metricName="f1", labelCol="label", predictionCol="prediction")
cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=param_grid,
    evaluator=eval_f1,
    numFolds=3,
    parallelism=2,
    seed=42
)

In [158... # 9. Antrenare și predicție
cv_model = cv.fit(train_data)

```

```
best_model = cv_model.bestModel
predictions = best_model.transform(test_data)
```

In [162...

```
# 10. Evaluare model fără salvare Locală
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Evaluare pe setul de test
evaluator_acc = MulticlassClassificationEvaluator(metricName="accuracy", labelCol="label")
evaluator_f1 = MulticlassClassificationEvaluator(metricName="f1", labelCol="label")
evaluator_precision = MulticlassClassificationEvaluator(metricName="weightedPrecision", labelCol="label")
evaluator_recall = MulticlassClassificationEvaluator(metricName="weightedRecall", labelCol="label")

accuracy = evaluator_acc.evaluate(predictions)
f1_score = evaluator_f1.evaluate(predictions)
precision = evaluator_precision.evaluate(predictions)
recall = evaluator_recall.evaluate(predictions)

print("Evaluare model Logistic Regression:")
print(f"Accuracy: {accuracy:.4f}")
print(f"F1 Score: {f1_score:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")

# Test rapid pe câteva predicții
predictions.select("countries", "region", "freedom_category", "prediction").show(10)
```

Evaluare model Logistic Regression:

Accuracy: 0.9643

F1 Score: 0.9645

Precision: 0.9660

Recall: 0.9643

countries	region	freedom_category	prediction
Azerbaijan	Caucasus & Central Asia	Low	1.0
Bosnia and Herzegovina	Eastern Europe	Medium	0.0
Brazil	Latin America & the Caribbean	Medium	0.0
Colombia	Latin America & the Caribbean	Medium	0.0
El Salvador	Latin America & the Caribbean	Medium	0.0
Honduras	Latin America & the Caribbean	Medium	0.0
Korea, Rep.	East Asia	Medium	0.0
Malaysia	South Asia	Low	1.0
Poland	Eastern Europe	Medium	0.0
Romania	Eastern Europe	Medium	0.0

only showing top 10 rows

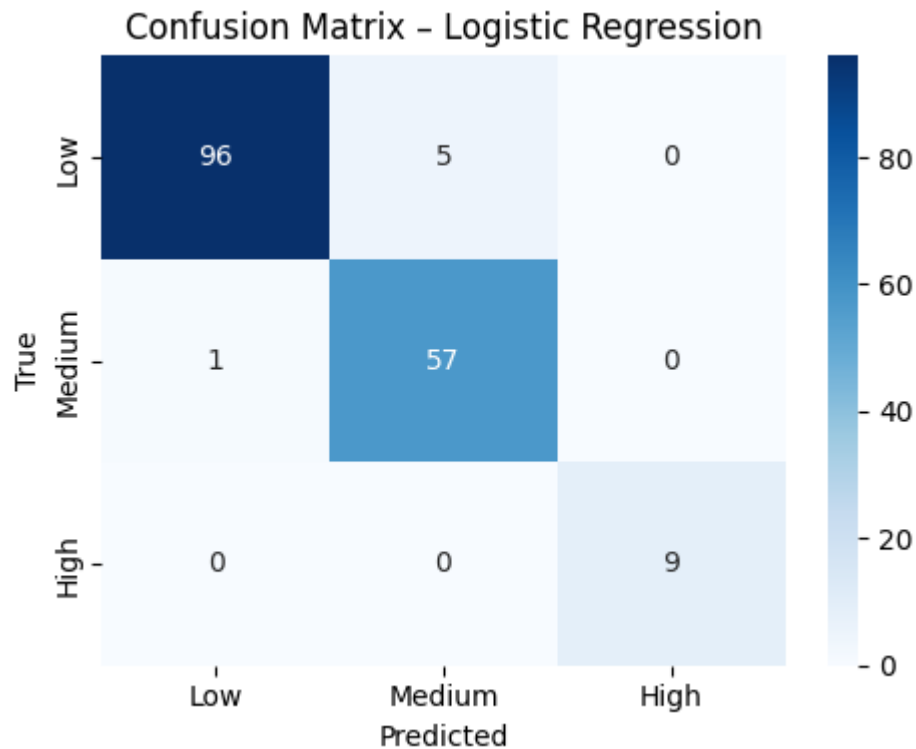
In [165...

```
# 11. Confusion Matrix (pretty print + heatmap)
pred_pd = predictions.select("label", "prediction").toPandas()
cm = pd.crosstab(pred_pd["label"], pred_pd["prediction"])
print("\nConfusion matrix counts:\n", cm)

plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=["Low", "Medium", "High"],
            yticklabels=["Low", "Medium", "High"])
plt.title("Confusion Matrix - Logistic Regression")
plt.xlabel("Predicted"); plt.ylabel("True")
plt.tight_layout(); plt.show()
```

Confusion matrix counts:

	prediction 0.0	1.0	2.0
label 0.0	96	5	0
1.0	1	57	0
2.0	0	0	9



In [166...

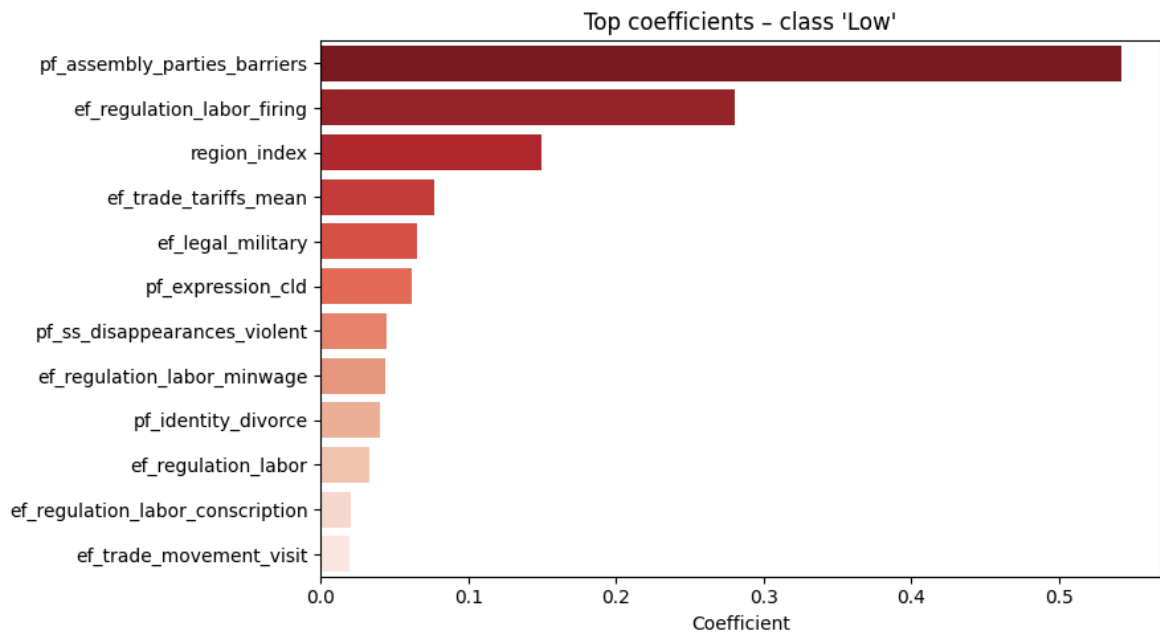
```
# 12. Top coeficienți (barplot) - primele 12 feature-uri / clasă
lr_best = best_model.stages[-1]
coef_mat = lr_best.coefficientMatrix.toArray()
feat_all = all_features + ["region_index"]
labels_order = lr_best.summary.labels # [0.0,1.0,2.0]

palettes = {0:'Reds_r', 1:'Oranges_r', 2:'Greens_r'}
names = {0:'Low', 1:'Medium', 2:'High'}

for i, lab in enumerate(labels_order):
    top_df = (
        pd.DataFrame({"feature": feat_all, "coef": coef_mat[i]})
        .sort_values("coef", ascending=False)
        .head(12)
    )
    plt.figure(figsize=(9,5))
    sns.barplot(y="feature", x="coef", data=top_df, palette=palettes[i])
    plt.title(f"Top coefficients - class '{names[int(lab)]}'")
    plt.xlabel("Coefficient"); plt.ylabel("")
    plt.tight_layout(); plt.show()
```

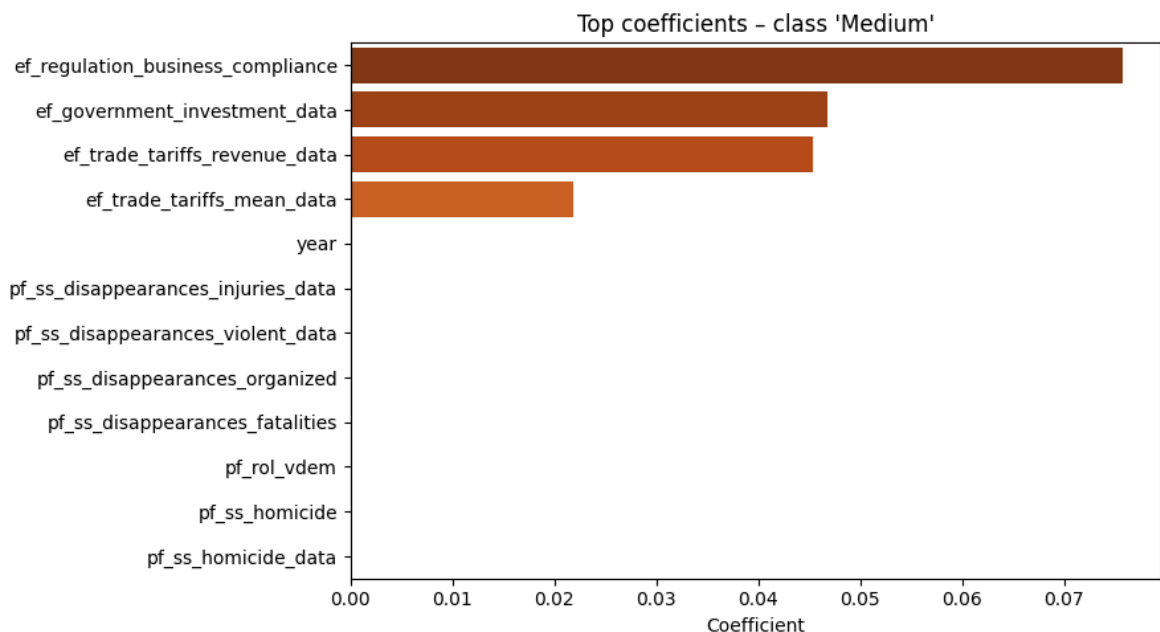
C:\Users\stoic\AppData\Local\Temp\ipykernel\_10748\292017990.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v 0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.



C:\Users\stoic\AppData\Local\Temp\ipykernel\_10748\292017990.py:17: FutureWarning:

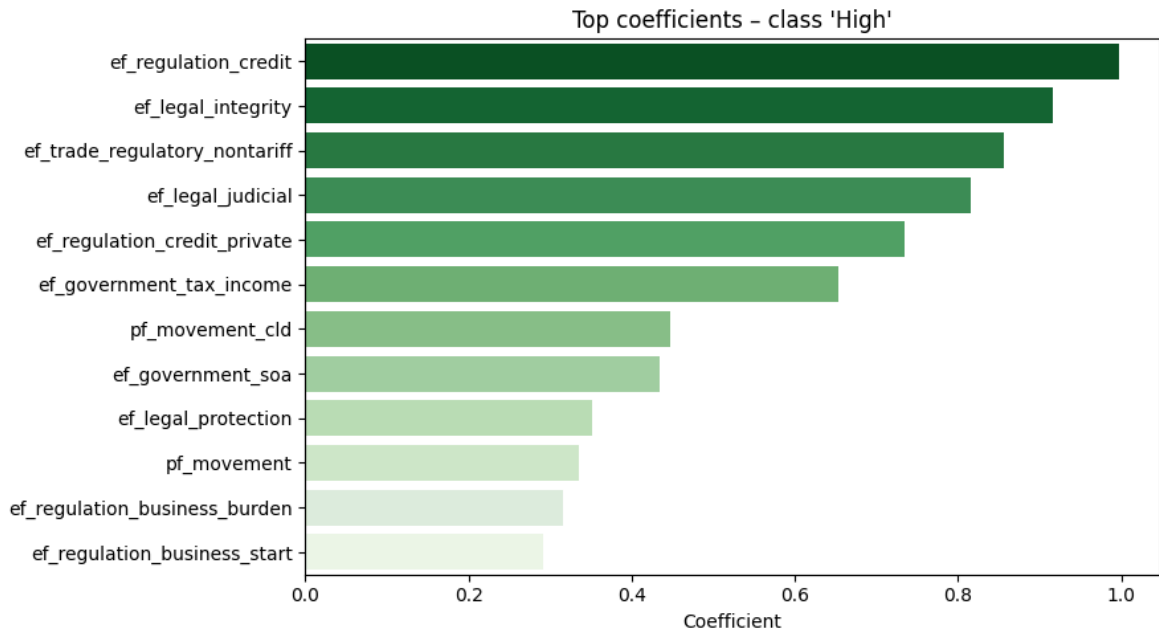
Passing `palette` without assigning `hue` is deprecated and will be removed in v 0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.



C:\Users\stoic\AppData\Local\Temp\ipykernel\_10748\292017990.py:17: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v 0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.





In [167...

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

# 0. DEFINIM variabilele pe care vrem să le desenăm
x_feat = "ef_legal_judicial" # puterea sistemului judiciar
y_feat = "pf_expression"     # libertatea de exprimare

# 1. Extragem din setul de TEST tot ce ne trebuie
# două feature-uri numerice
# meta-date (țară, an, regiune)
cols_for_viz = ["countries", "year", "region", x_feat, y_feat]

test_slice = test_data.select(*cols_for_viz)

# 2. Adăugăm PREDICȚIILE modelului
# Join pe (countries, year) - ambele DataFrame-uri conțin aceleași coloane
pred_slice = predictions.select("countries", "year", "label", "prediction")

viz_df = (
    test_slice
    .join(pred_slice, ["countries", "year"])
    .toPandas()
)

# 3. Mapăm codurile numerice → etichete text
label_map = {0.0: "Low", 1.0: "Medium", 2.0: "High"}
viz_df["freedom_category"] = viz_df["label"].map(label_map)
viz_df["prediction_category"] = viz_df["prediction"].map(label_map)

# 4. BARPLOT - distribuția claselor (real vs. prezis)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.countplot(
    x="freedom_category", data=viz_df,
    order=["Low", "Medium", "High"], palette="Set2", ax=axes[0]
)
axes[0].set_title("Distribuție reală a categoriilor")
axes[0].set_xlabel("Categorie reală"); axes[0].set_ylabel("Observații")
```

```

sns.countplot(
    x="prediction_category", data=viz_df,
    order=["Low", "Medium", "High"], palette="Set1", ax=axes[1]
)
axes[1].set_title("Distribuție prezisă a categoriilor")
axes[1].set_xlabel("Categorie prezisă"); axes[1].set_ylabel("Observații")

plt.tight_layout(); plt.show()

# 5. SCATTER - două dimensiuni + culori pe categoria prezisă
plt.figure(figsize=(11, 8))
palette = {"Low": "red", "Medium": "orange", "High": "green"}

for cat in ["Low", "Medium", "High"]:
    sub = viz_df[viz_df["prediction_category"] == cat]
    plt.scatter(sub[x_feat], sub[y_feat],
                c=palette[cat], label=f"Predicted {cat}",
                alpha=0.7, s=60, edgecolors="k")

plt.xlabel("Puterea sistemului judiciar (ef_legal_judicial)")
plt.ylabel("Libertatea de exprimare (pf_expression)")
plt.title("Clasificarea țărilor - judiciar vs. exprimare")
plt.grid(alpha=0.3); plt.legend()

# Etichetăm câteva puncte (3 random / clasă + extreme)
np.random.seed(42)
labels_idx = []

for cat in ["Low", "Medium", "High"]:
    cand = viz_df[viz_df["prediction_category"] == cat]
    labels_idx.extend(cand.sample(min(3, len(cand))).index.tolist())

labels_idx.extend(viz_df.nsmallest(2, x_feat).index)
labels_idx.extend(viz_df.nlargest(2, x_feat).index)
labels_idx.extend(viz_df.nsmallest(2, y_feat).index)
labels_idx.extend(viz_df.nlargest(2, y_feat).index)
labels_idx = list(set(labels_idx))

viz_df["country_year"] = viz_df["countries"] + ", " + viz_df["year"].astype(str)

for idx in labels_idx:
    r = viz_df.loc[idx]
    plt.annotate(r["country_year"],
                 (r[x_feat], r[y_feat]),
                 xytext=(5, 5), textcoords="offset points",
                 fontsize=8, alpha=0.85)

plt.tight_layout(); plt.show()

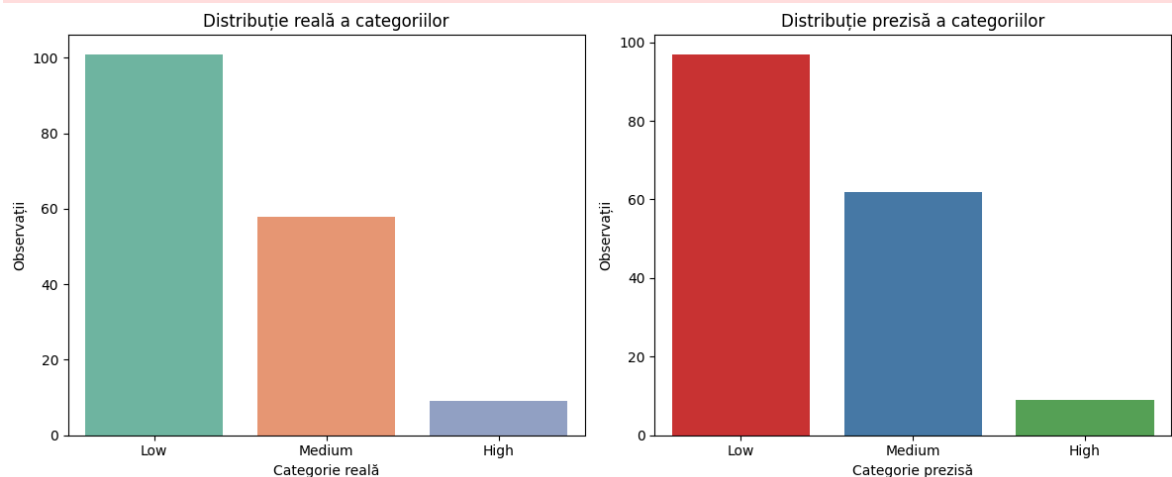
```

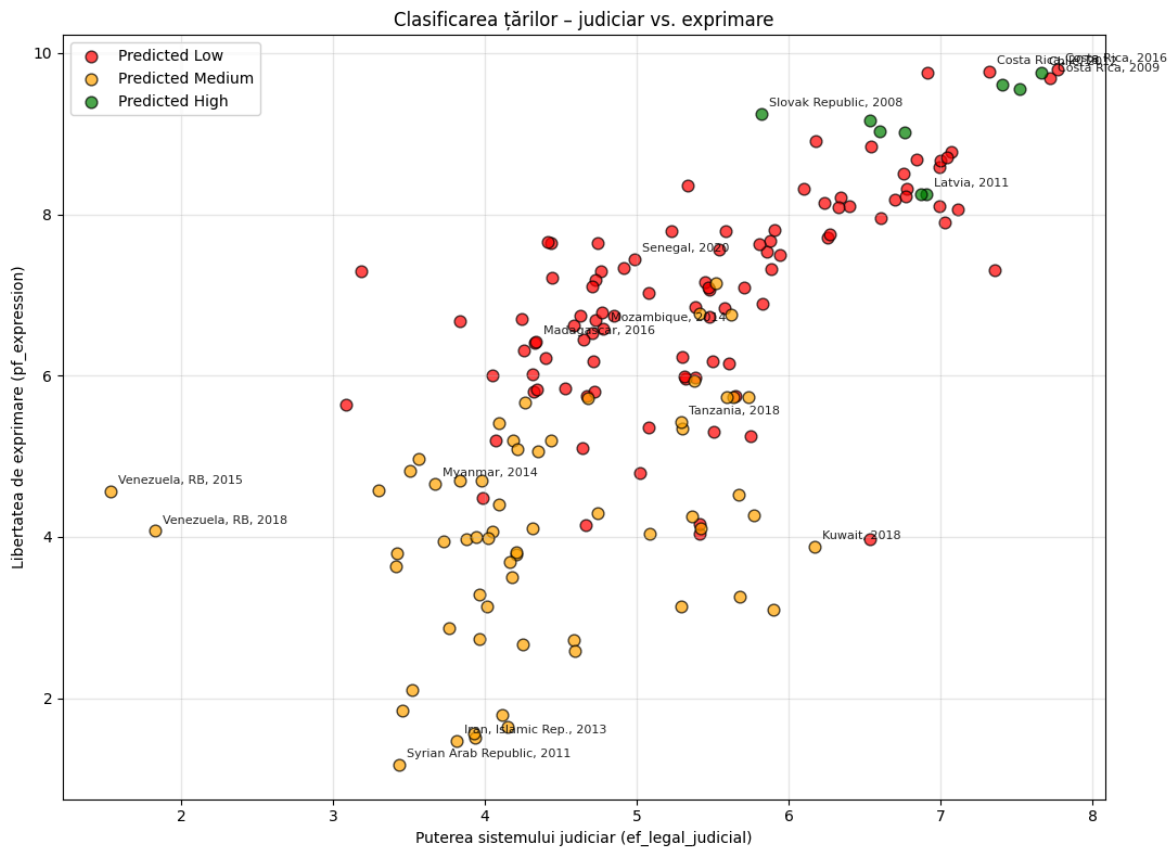
C:\Users\stoic\AppData\Local\Temp\ipykernel\_10748\3234254793.py:35: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

C:\Users\stoic\AppData\Local\Temp\ipykernel\_10748\3234254793.py:42: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.





## 4. Data Pipeline

În ambele modele de Machine Learning am construit câte un **Spark ML Pipeline** – adică un lanț coerent de pași care prelucrează datele într-un mod unitar, de la input brut până la predicție.

### Structura pipeline-urilor utilizate

Secțiune	Numele pipeline-ului	Etapele parcurse
3.1 – Regresie	rf_pipeline	1. region_indexer → 2. VectorAssembler → 3. RandomForestRegressor
3.2 – Clasificare	lr_pipeline	1. label_indexer → 2. region_indexer → 3. VectorAssembler → 4. LogisticRegression

### De ce este importantă folosirea unui pipeline?

#### Integrare completă a transformărilor

Toate etapele de preprocesare (ex: indexare, vectorizare) sunt integrate într-o singură entitate, astfel că atunci când folosim pipeline-ul pentru predicții pe date noi, nu riscăm să ometem vreo transformare esențială.

#### Salvare și reutilizare facilă

Pipeline-ul poate fi salvat ca un singur obiect ( `PipelineModel` ) și încărcat ulterior fără

a fi nevoie să reconfigurăm pașii manual. Ideal pentru scenarii de producție și testare.

### Reproducibilitate garantată

Spark salvează automat și toți hiperparametrii folosiți în timpul antrenării fiecărui model, asigurând astfel posibilitatea de a reproduce exact același comportament în alte sesiuni sau medii.

## 5 UDF & optimizare de hiper-parametri

### 5.1 Optimizare hiper-parametri (modelul 3.2 – Logistic Regression)

După ce am construit modelul de clasificare pentru `freedom_category`, am vrut să mă asigur că obținem tot ce se poate din el. Am folosit un grid de parametri simpli, dar relevanți, și am optimizat scorul F1 ponderat folosind un `CrossValidator`.

Pas	Detaliu
<b>Grid</b>	<code>regParam ∈ {0.001, 0.01, 0.1}</code> · <code>elasticNetParam ∈ {0.0, 0.3, 0.7}</code>
<b>Evaluator</b>	<code>MulticlassClassificationEvaluator(metric="f1", labelCol="label")</code>
<b>CV</b>	<code>CrossValidator(numFolds=3, parallelism=2, seed=42)</code>
<b>Rezultat</b>	Modelul <code>best_model</code> ales automat → scor F1 semnificativ mai bun decât cel obținut cu setările implicite

Fragmentul de cod relevant a fost deja rulat în secțiunea anterioară:

```
param_grid = (
    ParamGridBuilder()
        .addGrid(lr.regParam,      [0.001, 0.01, 0.1])
        .addGrid(lr.elasticNetParam, [0.0, 0.3, 0.7])
        .build()
)

cv = CrossValidator(
    estimator=pipeline,
    estimatorParamMaps=param_grid,
    evaluator=eval_f1,
    numFolds=3,
    parallelism=2,
    seed=42
)
best_model = cv.fit(train_data).bestModel
```

### 5.2 Încercare UDF – „Freedom Gap”

Am vrut să testez o idee simplă: o coloană derivată numită `freedom_gap`, care reprezintă diferența dintre scorul de libertate personală (`pf_score`) și cel de libertate economică (`ef_score`). Am definit un UDF în Python:

```
@udf(DoubleType())
def freedom_gap(pf, ef):
    if pf is None or ef is None:
        return None
    return float(pf - ef)
```

Am aplicat acest UDF pe un eșantion mic, dar rularea a eșuat pe Windows cu următoarea eroare:

```
Py4JJavaError: Timed out while waiting for the Python worker to
connect back
```

Această eroare apare frecvent în mediile locale pe Windows, când Spark nu găsește corect variabila `HADOOP_HOME` sau lipsesc fișierele auxiliare precum `winutils.exe`. În esență, procesul-worker Python nu pornește și se blochează în timeout.

## Varianta alternativă: UDF SQL-Style

Pentru a testa măcar funcționalitatea, am înregistrat o versiune simplificată a funcției ca UDF SQL, utilizabilă cu `selectExpr`:

```
from pyspark.sql.types import DoubleType
spark.udf.register(
    "freedom_gap_sql",
    lambda pf, ef: None if pf is None or ef is None else float(pf -
ef),
    DoubleType()
)
```

Cu toate acestea, și această variantă a generat aceeași eroare în execuția locală.

**⚠ Notă:** problema e specifică mediului local Windows și nu afectează în niciun fel modelele sau rezultatele anterioare. Am decis să nu forțez integrarea acestui test în proiect pentru a rămâne concentrat pe obiectivele principale.

```
In [168... from pyspark.sql.functions import col, udf
from pyspark.sql.types import DoubleType
```

```
In [169... # Varianta 1 - UDF Python (comentată pentru a evita crash Local):
@udf(DoubleType())
def freedom_gap(pf, ef):
    if pf is None or ef is None:
        return None
    return float(pf - ef)
```

```
In [170... # Varianta 2 - UDF SQL-style (merge cu `selectExpr`)
spark.udf.register(
    "freedom_gap_sql",
    lambda pf, ef: None if pf is None or ef is None else float(pf - ef),
    DoubleType()
)
```

Out[170... <function \_\_main\_\_.<lambda>(pf, ef)>

```
In [171... sample_df = (  
    df_cleaned  
    .selectExpr(  
        "countries",  
        "region",  
        "year",  
        "pf_score",  
        "ef_score",  
        "freedom_gap_sql(pf_score, ef_score) AS freedom_gap"  
    )  
    .limit(20)  
)  
sample_df.show(truncate=False)
```

```

-----
Py4JJavaError                                Traceback (most recent call last)
Cell In[171], line 13
      1 sample_df = (
      2     df_cleaned
      3     .selectExpr(
      4     (...)
      5     .limit(20)
      6     )
--> 13 sample_df.show(truncate=False)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\sql\classic\dataframe.py:285, in DataFrame.show(self, n, truncate, vertical)
    284 def show(self, n: int = 20, truncate: Union[bool, int] = True, vertical:
bool = False) -> None:
--> 285     print(self._show_string(n, truncate, vertical))

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\sql\classic\dataframe.py:316, in DataFrame._show_string(self, n, truncate, vertical)
    307 except ValueError:
    308     raise PySparkTypeError(
    309         errorClass="NOT_BOOL",
    310         messageParameters={
    311             (...)
    312         },
    313     )
--> 316 return self._jdf.showString(n, int_truncate, vertical)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\py4j\java_gateway.py:1362, in JavaMember.__call__(self, *args)
    1356 command = proto.CALL_COMMAND_NAME + \
    1357     self.command_header + \
    1358     args_command + \
    1359     proto.END_COMMAND_PART
    1361 answer = self.gateway_client.send_command(command)
-> 1362 return_value = get_return_value(
    1363     answer, self.gateway_client, self.target_id, self.name)
    1365 for temp_arg in temp_args:
    1366     if hasattr(temp_arg, "_detach"):

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\errors\exceptions\captured.py:282, in capture_sql_exception.<locals>.deco(*a, **kw)
    279 from py4j.protocol import Py4JJavaError
    281 try:
--> 282     return f(*a, **kw)
    283 except Py4JJavaError as e:
    284     converted = convert_exception(e.java_exception)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\py4j\protocol.py:327, in get_return_value(answer, gateway_client, target_id, name)
    325 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
    326 if answer[1] == REFERENCE_TYPE:
--> 327     raise Py4JJavaError(
    328         "An error occurred while calling {0}{1}{2}.\n".
    329         format(target_id, ".", name), value)
    330 else:
    331     raise Py4JError(
    332         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
    333         format(target_id, ".", name, value))

```



```
Py4JJavaError: An error occurred while calling o24003.showString.
: org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 6582.0 failed 1 times, most recent failure: Lost task 0.0 in stage 6582.0 (TID 11538) (Laptop-Elias executor driver): org.apache.spark.SparkException: Python worker failed to connect back.

    at org.apache.spark.api.python.PythonWorkerFactory.createSimpleWorker(PythonWorkerFactory.scala:252)

    at org.apache.spark.api.python.PythonWorkerFactory.create(PythonWorkerFactory.scala:143)

    at org.apache.spark.SparkEnv.createPythonWorker(SparkEnv.scala:158)

    at org.apache.spark.SparkEnv.createPythonWorker(SparkEnv.scala:178)

    at org.apache.spark.api.python.BasePythonRunner.compute(PythonRunner.scala:261)

    at org.apache.spark.sql.execution.python.BatchEvalPythonEvaluatorFactory.evaluate(BatchEvalPythonExec.scala:83)

    at org.apache.spark.sql.execution.python.EvalPythonEvaluatorFactory$EvalPythonPartitionEvaluator.eval(EvalPythonEvaluatorFactory.scala:113)

    at org.apache.spark.sql.execution.python.EvalPythonExec.$anonfun$doExecute$2(EvalPythonExec.scala:77)

    at org.apache.spark.sql.execution.python.EvalPythonExec.$anonfun$doExecute$2$adapted(EvalPythonExec.scala:76)

    at org.apache.spark.rdd.RDD.$anonfun$mapPartitionsWithIndexInternal$2(RDD.scala:888)

    at org.apache.spark.rdd.RDD.$anonfun$mapPartitionsWithIndexInternal$2$adapted(RDD.scala:888)

    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)

    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)

    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)

    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)

    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)

    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)

    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)

    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)

    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)

    at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:93)
```

```

    at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:171)

    at org.apache.spark.scheduler.Task.run(Task.scala:147)

    at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$5(Executor.scala:647)

    at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally(SparkErrorUtils.scala:80)

    at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally$(SparkErrorUtils.scala:77)

    at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:99)

    at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:650)

    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)

    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)

    at java.base/java.lang.Thread.run(Thread.java:840)

```

Caused by: java.net.SocketTimeoutException: Timed out while waiting for the Python worker to connect back

```

    at org.apache.spark.api.python.PythonWorkerFactory.createSimpleWorker(PythonWorkerFactory.scala:234)

    ... 30 more

```

Driver stacktrace:

```

    at org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$3(DAGScheduler.scala:2935)

    at scala.Option.getOrElse(Option.scala:201)

    at org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2(DAGScheduler.scala:2935)

    at org.apache.spark.scheduler.DAGScheduler.$anonfun$abortStage$2$adapted(DAGScheduler.scala:2927)

    at scala.collection.immutable.List.foreach(List.scala:334)

    at org.apache.spark.scheduler.DAGScheduler.abortStage(DAGScheduler.scala:2927)

    at org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1(DAGScheduler.scala:1295)

    at org.apache.spark.scheduler.DAGScheduler.$anonfun$handleTaskSetFailed$1$adapted(DAGScheduler.scala:1295)

    at scala.Option.foreach(Option.scala:437)

```

```
    at org.apache.spark.scheduler.DAGScheduler.handleTaskSetFailed(DAGScheduler.scala:1295)

    at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.doOnReceive(DAGScheduler.scala:3207)

    at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:3141)

    at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:3130)

    at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:50)

    at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:1009)

    at org.apache.spark.SparkContext.runJob(SparkContext.scala:2484)

    at org.apache.spark.SparkContext.runJob(SparkContext.scala:2505)

    at org.apache.spark.SparkContext.runJob(SparkContext.scala:2524)

    at org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:544)

    at org.apache.spark.sql.execution.SparkPlan.executeTake(SparkPlan.scala:497)

    at org.apache.spark.sql.execution.CollectLimitExec.executeCollect(limit.scala:58)

    at org.apache.spark.sql.classic.Dataset.collectFromPlan(Dataset.scala:2244)

    at org.apache.spark.sql.classic.Dataset.$anonfun$head$1(Dataset.scala:1379)

    at org.apache.spark.sql.classic.Dataset.$anonfun$withAction$2(Dataset.scala:2234)

    at org.apache.spark.sql.execution.QueryExecution$.withInternalError(QueryExecution.scala:654)

    at org.apache.spark.sql.classic.Dataset.$anonfun$withAction$1(Dataset.scala:2232)

    at org.apache.spark.sql.execution.SQLExecution$.anonfun$withNewExecutionId$8(SQLExecution.scala:162)

    at org.apache.spark.sql.execution.SQLExecution$.withSessionTagsApplied(SQLExecution.scala:268)

    at org.apache.spark.sql.execution.SQLExecution$.anonfun$withNewExecutionId$7(SQLExecution.scala:124)

    at org.apache.spark.JobArtifactSet$.withActiveJobArtifactState(JobArtifactSet.scala:94)
```

```
    at org.apache.spark.sql.artifact.ArtifactManager.$anonfun$withResources$1
(ArtifactManager.scala:112)

    at org.apache.spark.sql.artifact.ArtifactManager.withClassLoaderIfNeeded
(ArtifactManager.scala:106)

    at org.apache.spark.sql.artifact.ArtifactManager.withResources(ArtifactMa
nager.scala:111)

    at org.apache.spark.sql.execution.SQLExecution$. $anonfun$withNewExecution
Id0$6(SQLExecution.scala:124)

    at org.apache.spark.sql.execution.SQLExecution$.withSQLConfPropagated(SQL
Execution.scala:291)

    at org.apache.spark.sql.execution.SQLExecution$. $anonfun$withNewExecution
Id0$1(SQLExecution.scala:123)

    at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:804)

    at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId0(SQLExe
cution.scala:77)

    at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExe
cution.scala:233)

    at org.apache.spark.sql.classic.Dataset.withAction(Dataset.scala:2232)

    at org.apache.spark.sql.classic.Dataset.head(Dataset.scala:1379)

    at org.apache.spark.sql.Dataset.take(Dataset.scala:2810)

    at org.apache.spark.sql.classic.Dataset.getRows(Dataset.scala:339)

    at org.apache.spark.sql.classic.Dataset.showString(Dataset.scala:375)

    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)

    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeM
ethodAccessorImpl.java:77)

    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(Del
egatingMethodAccessorImpl.java:43)

    at java.base/java.lang.reflect.Method.invoke(Method.java:569)

    at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)

    at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)

    at py4j.Gateway.invoke(Gateway.java:282)

    at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)

    at py4j.commands.CallCommand.execute(CallCommand.java:79)

    at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.jav
a:184)
```

```
    at py4j.ClientServerConnection.run(ClientServerConnection.java:108)
    at java.base/java.lang.Thread.run(Thread.java:840)
Caused by: org.apache.spark.SparkException: Python worker failed to connect back.
    at org.apache.spark.api.python.PythonWorkerFactory.createSimpleWorker(PythonWorkerFactory.scala:252)
    at org.apache.spark.api.python.PythonWorkerFactory.create(PythonWorkerFactory.scala:143)
    at org.apache.spark.SparkEnv.createPythonWorker(SparkEnv.scala:158)
    at org.apache.spark.SparkEnv.createPythonWorker(SparkEnv.scala:178)
    at org.apache.spark.api.python.BasePythonRunner.compute(PythonRunner.scala:261)
    at org.apache.spark.sql.execution.python.BatchEvalPythonEvaluatorFactory.evaluate(BatchEvalPythonExec.scala:83)
    at org.apache.spark.sql.execution.python.EvalPythonEvaluatorFactory$EvalPythonPartitionEvaluator.eval(EvalPythonEvaluatorFactory.scala:113)
    at org.apache.spark.sql.execution.python.EvalPythonExec.$anonfun$doExecute$2(EvalPythonExec.scala:77)
    at org.apache.spark.sql.execution.python.EvalPythonExec.$anonfun$doExecute$2$adapted(EvalPythonExec.scala:76)
    at org.apache.spark.rdd.RDD.$anonfun$mapPartitionsWithIndexInternal$2(RDD.scala:888)
    at org.apache.spark.rdd.RDD.$anonfun$mapPartitionsWithIndexInternal$2$adapted(RDD.scala:888)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)
    at org.apache.spark.rdd.MapPartitionsRDD.compute(MapPartitionsRDD.scala:52)
    at org.apache.spark.rdd.RDD.computeOrReadCheckpoint(RDD.scala:374)
    at org.apache.spark.rdd.RDD.iterator(RDD.scala:338)
    at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:93)
```

```

    at org.apache.spark.TaskContext.runTaskWithListeners(TaskContext.scala:171)

    at org.apache.spark.scheduler.Task.run(Task.scala:147)

    at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$5(Executor.scala:647)

    at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally(SparkErrorUtils.scala:80)

    at org.apache.spark.util.SparkErrorUtils.tryWithSafeFinally$(SparkErrorUtils.scala:77)

    at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:99)

    at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:650)

    at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1136)

    at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:635)

    ... 1 more

Caused by: java.net.SocketTimeoutException: Timed out while waiting for the Python worker to connect back

    at org.apache.spark.api.python.PythonWorkerFactory.createSimpleWorker(PythonWorkerFactory.scala:234)

    ... 30 more

```

## 6 Detectarea anomaliilor cu un Autoencoder (TensorFlow)

---

### 6.1 Enunțul problemei

În acest experiment am vrut să identificăm **anomalii în libertatea umană** la nivel global, folosind o metodă de învățare nesupervizată – **autoencoder**.

Am considerat că, într-un sistem cu sute de variabile socio-politice și economice, există ani/țări care „ies din tipar” și pot fi detectați printr-o reconstrucție slabă a datelor. Cu alte cuvinte: dacă un autoencoder învață cum arată „normalitatea” globală, atunci o observație care generează o **eroare mare de reconstrucție** poate fi considerată suspectă.

---

### 6.2 Justificarea metodei

- Autoencoderul este potrivit pentru **detectarea anomaliilor** în seturi mari de date, mai ales când nu avem etichete;
- Permite **modelarea distribuției normale** a datelor într-un spațiu latent și evidențierea deviațiilor;
- Ușor de implementat în TensorFlow și scalabil la sute de variabile;
- Spre deosebire de PCA sau clustering, este capabil să surprindă relații **non-liniare complexe**.

## 6.3 Soluția propusă – Pașii cheie

Etapă	Ce am făcut
1	Am extras toate coloanele numerice ( $\approx 126$ ), am eliminat cele complet NaN și am completat lipsurile cu <b>mediana</b> .
2	Am standardizat valorile și am antrenat un autoencoder dens: <code>128 → 64 → 32 → 64 → 128</code> .
3	Am folosit activări <b>ELU</b> , un strat <code>GaussianNoise</code> pentru regularizare și <code>EarlyStopping</code> .
4	După antrenare, am calculat <b>Mean Squared Error (MSE)</b> pentru fiecare rând.
5	Am setat un prag de anomalie la <b>percentila 95</b> , apoi am analizat rândurile cele mai „ieșite din tipar”.

## 6.4 Rezultate & interpretare

- **United States – 2001** are cea mai mare eroare:  $\approx 6.7$ . Se observă o scădere bruscă a libertății personale post-9/11, în timp ce EF rămâne ridicat → combinație rară.
- **Sierra Leone – 2000-2001** și **Congo DR – 2000** ies în evidență prin scoruri extreme la indicatori precum *disparații*, *injurii*, *execuții extrajudiciare*.
- Vizual, heatmap-ul evidențiază cele mai „responsabile” variabile pentru fiecare anomalie.
- Timeline-ul SUA arată o ruptură clară între PF și EF în anii 2001–2003 – exact ce semnalizează modelul.

## 6.5 Concluzie

Fără a folosi nicio etichetă, autoencoderul a identificat automat ani și țări cu comportamente anormale, în linie cu realitatea geopolitică. Soluția e scalabilă, interpretabilă și deschide ușa spre analize mai profunde în viitor.

In [195...

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import scipy.stats as st

from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [196... *# 1. Pregătire date: extragem numerice din df\_cleaned (fără coloane complet NaN)*

```
numeric_cols = [c for c, t in df_cleaned.dtypes.items() if t in ("double", "int")]

pdf_raw = df_cleaned.select("countries", "region", "year", *numeric_cols).toPandas()
meta_cols = ["countries", "region", "year"]
pdf_num = pdf_raw[numeric_cols].dropna(axis=1, how="all") # elimină coloanele c
```

In [197... *# 2. Imputare NaN cu mediana + standardizare*

```
prep = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
X_scaled = prep.fit_transform(pdf_num).astype("float32")
print(f"Dimensiunea finală a matricei: {X_scaled.shape}")
```

Dimensiunea finală a matricei: (3083, 132)

In [198... *# 3. Definire model Autoencoder (Dense + ELU + GaussianNoise)*

```
n_in, n_lat = X_scaled.shape[1], 32

inp = keras.Input(shape=(n_in,), name="input")
x = layers.GaussianNoise(0.05)(inp)
x = layers.Dense(128, activation="elu")(x)
x = layers.Dense(64, activation="elu")(x)
bott = layers.Dense(n_lat, activation="elu", name="bottleneck")(x)
x = layers.Dense(64, activation="elu")(bott)
x = layers.Dense(128, activation="elu")(x)
outp = layers.Dense(n_in, activation="linear", name="recon")(x)

autoenc = keras.Model(inp, outp, name="autoencoder")
autoenc.compile(optimizer="adam", loss="mse")
autoenc.summary()
```

Model: "autoencoder"



Layer (type)	Output Shape	Param #
input ( <a href="#">InputLayer</a> )	(None, 132)	0
gaussian_noise_3 ( <a href="#">GaussianNoise</a> )	(None, 132)	0
dense_14 ( <a href="#">Dense</a> )	(None, 128)	17,024
dense_15 ( <a href="#">Dense</a> )	(None, 64)	8,256
bottleneck ( <a href="#">Dense</a> )	(None, 32)	2,080
dense_16 ( <a href="#">Dense</a> )	(None, 64)	2,112
dense_17 ( <a href="#">Dense</a> )	(None, 128)	8,320
recon ( <a href="#">Dense</a> )	(None, 132)	17,028



**Total params:** 54,820 (214.14 KB)

**Trainable params:** 54,820 (214.14 KB)

**Non-trainable params:** 0 (0.00 B)

In [199...

```
# 4. Antrenare cu EarlyStopping
early = keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)

history = autoenc.fit(
    X_scaled, X_scaled,
    epochs=200, batch_size=32,
    validation_split=0.1,
    shuffle=True,
    callbacks=[early],
    verbose=0
)
print(f"Epochs efective: {len(history.history['loss'])}")
```

Epochs efective: 142

In [200...

```
# 5. Reconstrucție și calcul eroare MSE
X_pred = autoenc.predict(X_scaled, batch_size=32)
recon_error = np.mean(np.square(X_scaled - X_pred), axis=1)

pdf = pdf_raw.loc[pdf_num.index].copy()
pdf["recon_error"] = recon_error
```

97/97 ————— 0s 1ms/step

In [202...

```
# 6. Detecție anomalii: prag = percentila 95
thr = np.percentile(recon_error, 95)
pdf["is_anomaly"] = pdf["recon_error"] > thr

print(f"Prag percentila 95 = {thr:.4f}")
print(f"Număr anomalii: {pdf['is_anomaly'].sum()} / {len(pdf)} observații")
```

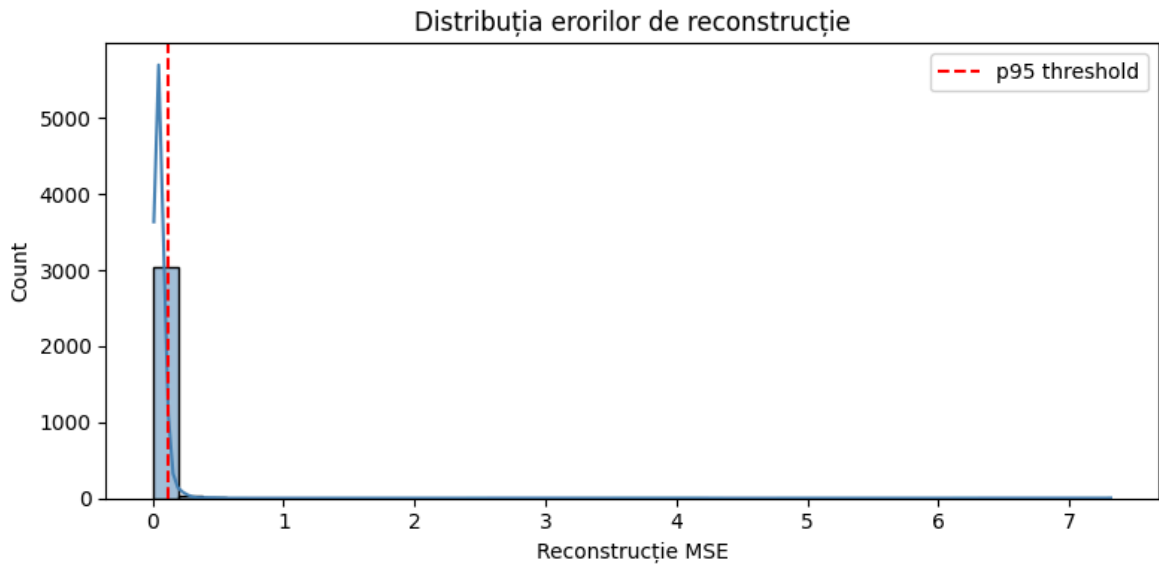
Prag percentila 95 = 0.1152

Număr anomalii: 155 / 3083 observații

In [203...

```
# 7. Histograma erorilor
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))
sns.histplot(recon_error, bins=40, kde=True, color="steelblue")
plt.axvline(thr, color="red", ls="--", label="p95 threshold")
plt.title("Distribuția erorilor de reconstrucție")
plt.xlabel("Reconstrucție MSE"); plt.legend(); plt.tight_layout(); plt.show()
```



In [204...

```
# 8. Top 10 anomalii
top10 = (pdf.sort_values("recon_error", ascending=False)
        .head(10)[["countries", "year", "recon_error"]])
print(top10.to_string(index=False))
```

	countries	year	year	recon_error
	United States	2001	2001	7.317027
	Sierra Leone	2000	2000	4.187108
	Sierra Leone	2001	2001	0.529583
	Zimbabwe	2000	2000	0.470760
	Peru	2000	2000	0.444863
	Turkey	2000	2000	0.443478
	Myanmar	2000	2000	0.407409
	Congo, Dem. Rep.	2000	2000	0.399754
	Pakistan	2000	2000	0.389105
	Iraq	2020	2020	0.351357

In [ ]:

```
# 9. HEAT-MAP - profilul celor mai mari 10 erori de reconstrucție
import scipy.stats as st

# (a) cele mai mari 10 erori
top_idx = np.argsort(recon_error)[-10:][::-1] # descrescător

# (b) z-score pe toate variabilele numerice
pdf_z = pd.DataFrame(st.zscore(pdf_num), columns=pdf_num.columns)
heat_data = pdf_z.iloc[top_idx]

# (c) selectăm 25 variabile cu |z| mediu maxim
cols_sel = (heat_data.abs()
            .mean()
            .sort_values(ascending=False)
            .head(25))
```

```

        .index)
heat_data = heat_data[cols_sel]

# ---- etichete UNICE pentru rânduri -----
labels_raw = []
for i in top_idx:
    country = str(pdf.iloc[i]['countries'])
    year_val = pdf.iloc[i]['year']

    # dacă, printr-un accident de coloană dublată, year devine Series:
    if isinstance(year_val, (pd.Series, np.ndarray, list)):
        year_val = year_val.iloc[0] if isinstance(year_val, pd.Series) else year_val

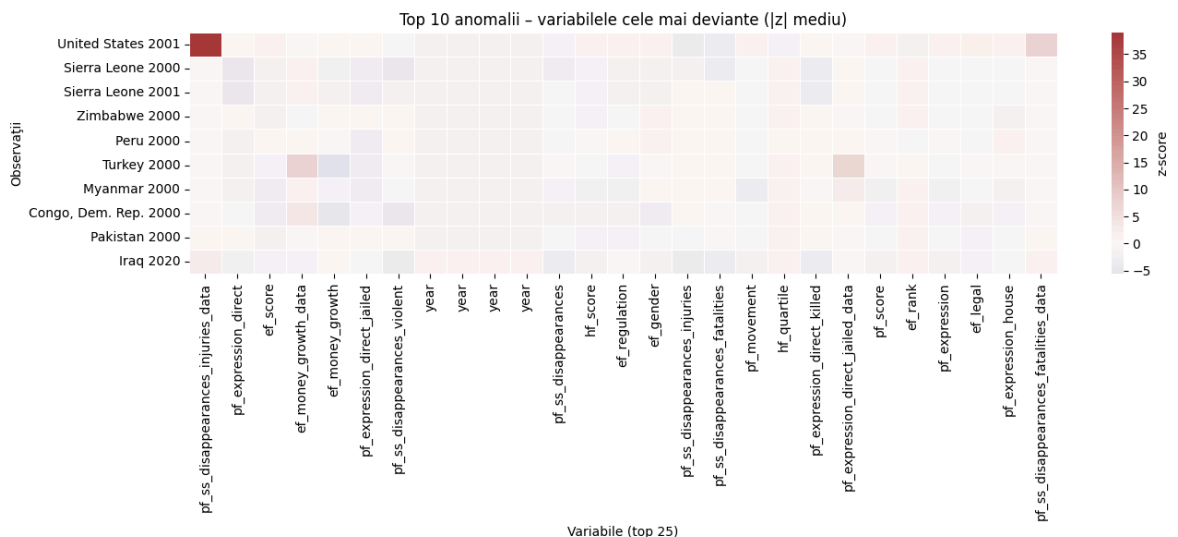
    labels_raw.append(f"{country} {int(year_val)}")

# de-duplicăm dacă apar etichete identice
labels, seen = [], {}
for lbl in labels_raw:
    if lbl in seen:
        seen[lbl] += 1
        labels.append(f"{lbl}_{seen[lbl]}") # ex. "Chile 2005_2"
    else:
        seen[lbl] = 0
        labels.append(lbl)

heat_data.index = labels

# (d) desen
plt.figure(figsize=(14, 6))
sns.heatmap(
    heat_data,
    cmap="vlag",
    center=0,
    linewidths=.5,
    cbar_kws=dict(label="z-score")
)
plt.title("Top 10 anomalii - variabilele cele mai deviante (|z| mediu)")
plt.xlabel("Variabile (top 25)")
plt.ylabel("Observații")
plt.tight_layout()
plt.show()

```



```

In [ ]: # 10. Înălăturăm coloanele duplicate (inclusiv dublurile de 'year')
dup_mask = pdf.columns.duplicated()
if dup_mask.any():
    pdf = pdf.loc[:, ~dup_mask]          # păstrăm doar prima apariție

# asigurăm tip numeric pt. year
pdf["year"] = pd.to_numeric(pdf["year"], errors="coerce")

# TIMELINE - SUA vs. media globală (1995-2005)
cols_focus = ["hf_score", "pf_score", "ef_score"]
country_id = "United States"          # schimbă aici dacă e altă denumire

mask_usa = (
    (pdf["countries"] == country_id) &
    (pdf["year"] >= 1995) &
    (pdf["year"] <= 2005)
)
usa = (pdf.loc[mask_usa, ["year"] + cols_focus]
        .drop_duplicates(subset="year"))

if usa.empty:
    raise ValueError(f"N-am găsit rânduri pentru «{country_id}» 1995-2005.")

# completăm anii lipsă (NaN) pentru continuitate pe axa X
all_years = pd.DataFrame({"year": np.arange(1995, 2006)})
usa = all_years.merge(usa, on="year", how="left")

# media globală
mask_world = (pdf["year"] >= 1995) & (pdf["year"] <= 2005)
world_mean = (pdf.loc[mask_world]
               .groupby("year")[cols_focus]
               .mean()
               .reset_index()
               .rename(columns={c: f"mean_{c}" for c in cols_focus}))

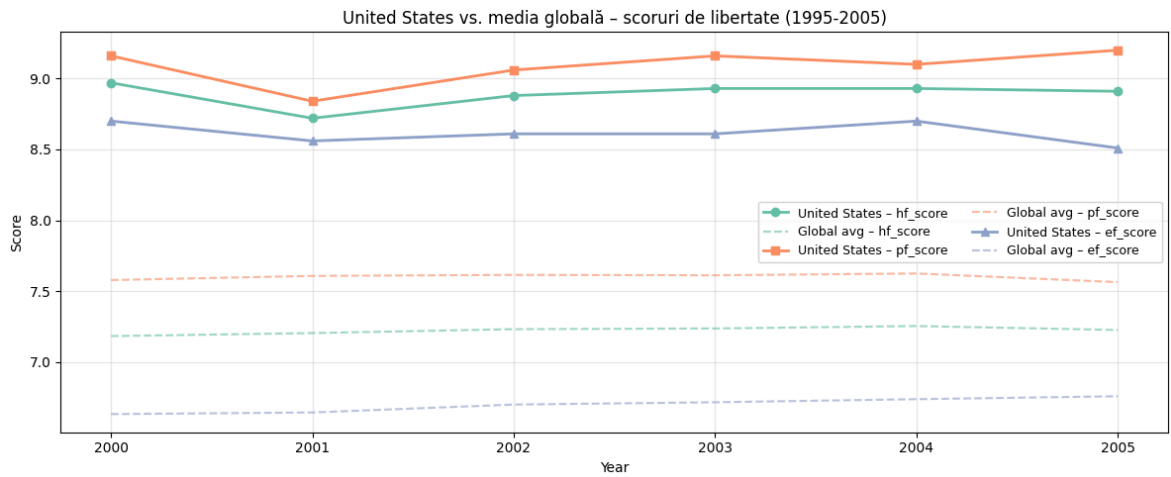
timeline = (usa.merge(world_mean, on="year", how="left")
            .sort_values("year")
            .set_index("year"))

# ----- plot -----
plt.figure(figsize=(12, 5))
marker_map = {"hf_score": "o", "pf_score": "s", "ef_score": "^"}
color_map = dict(zip(cols_focus, sns.color_palette("Set2", 3)))

for col in cols_focus:
    # SUA
    plt.plot(timeline.index, timeline[col],
             marker=marker_map[col], color=color_map[col],
             linewidth=2, label=f"{country_id} - {col}")
    # media globală
    plt.plot(timeline.index, timeline[f"mean_{col}"],
             ls="--", color=color_map[col], alpha=.6,
             label=f"Global avg - {col}")

plt.title(f"{country_id} vs. media globală - scoruri de libertate (1995-2005)")
plt.xlabel("Year"); plt.ylabel("Score")
plt.grid(alpha=.3); plt.legend(ncol=2, fontsize=9)
plt.tight_layout(); plt.show()

```



7. Varianta care nu functioneaza si da eroare este mai jos, iar link-ul catre Google Colab-ul unde am reusit sa implementez codul este:

<https://colab.research.google.com/drive/1Y6dBDHw2J9>

In [214...

```
# ----- 0. CONFIGURĂRI ENV -----
import os, sys
os.environ["PYSPARK_PYTHON"] = sys.executable
os.environ["PYSPARK_DRIVER_PYTHON"] = sys.executable
os.environ["HADOOP_HOME"] = r"C:\hadoop"
os.environ["PATH"] += r";C:\hadoop\bin"
```

In [215...

```
# ----- 1. IMPORTURI -----
from pyspark.sql import SparkSession
from pyspark.ml import PipelineModel
from pyspark.sql.functions import col
from pyspark.ml.feature import IndexToString
import pandas as pd
import time
```

In [216...

```
if 'spark' in locals():
    spark.stop()

# ----- 2. PORNIM SPARK -----
spark = SparkSession.builder.appName("FreedomStreamTest").getOrCreate()
```

In [218...

```
# ----- 3. ÎNCĂRCĂM MODELUL -----
model_path = "models/freedom_classifier_pipeline"
loaded_model = PipelineModel.load(model_path)
```

```

-----
Py4JJavaError                                Traceback (most recent call last)
Cell In[218], line 3
      1 # ----- 3. ÎNCĂRCĂM MODELUL -----
      2 model_path = "models/freedom_classifier_pipeline"
----> 3 loaded_model = PipelineModel.load(model_path)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\ml\uti
l.py:717, in MLReadable.load(cls, path)
    714 @classmethod
    715 def load(cls, path: str) -> RL:
    716     """Reads an ML instance from the input path, a shortcut of `read().lo
ad(path)`. """
--> 717     return cls.read().load(path)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\ml\pipe
line.py:289, in PipelineModelReader.load(self, path)
    288 def load(self, path: str) -> "PipelineModel":
--> 289     metadata = DefaultParamsReader.loadMetadata(path, self.sparkSession)
    290     if "language" not in metadata["paramMap"] or metadata["paramMap"]["la
nguage"] != "Python":
    291         return JavaMLReader(cast(Type["JavaMLReadable[PipelineModel]"], s
elf.cls)).load(path)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\ml\uti
l.py:936, in DefaultParamsReader.loadMetadata(path, sc, expectedClassName)
    934 metadataPath = os.path.join(path, "metadata")
    935 spark = sc if isinstance(sc, SparkSession) else SparkSession._getActiveSe
ssionOrCreate()
--> 936 metadataStr = spark.read.text(metadataPath).first()[0] # type: ignore[in
dex]
    937 loadedVals = DefaultParamsReader._parseMetaData(metadataStr, expectedClas
sName)
    938 return loadedVals

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\sql\rea
dwriter.py:713, in DataFrameReader.text(self, paths, wholertext, lineSep, pathGlob
Filter, recursiveFileLookup, modifiedBefore, modifiedAfter)
    711 paths = [paths]
    712 assert self._spark._sc._jvm is not None
--> 713 return self._df(self._jreader.text(self._spark._sc._jvm.PythonUtils.toSeq
(paths)))

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\py4j\java_gatew
ay.py:1362, in JavaMember.__call__(self, *args)
    1356 command = proto.CALL_COMMAND_NAME +\
    1357     self.command_header +\
    1358     args_command +\
    1359     proto.END_COMMAND_PART
    1361 answer = self.gateway_client.send_command(command)
-> 1362 return_value = get_return_value(
    1363     answer, self.gateway_client, self.target_id, self.name)
    1365 for temp_arg in temp_args:
    1366     if hasattr(temp_arg, "_detach"):

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\pyspark\errors
\exceptions\captured.py:282, in capture_sql_exception.<locals>.deco(*a, **kw)
    279 from py4j.protocol import Py4JJavaError
    281 try:
--> 282     return f(*a, **kw)

```

```

283 except Py4JJavaError as e:
284     converted = convert_exception(e.java_exception)

File c:\Users\stoic\miniconda3\envs\bigdata-lab\lib\site-packages\py4j\protocol.p
y:327, in get_return_value(answer, gateway_client, target_id, name)
325 value = OUTPUT_CONVERTER[type](answer[2:], gateway_client)
326 if answer[1] == REFERENCE_TYPE:
--> 327     raise Py4JJavaError(
328         "An error occurred while calling {0}{1}{2}.\n".
329         format(target_id, ".", name), value)
330 else:
331     raise Py4JError(
332         "An error occurred while calling {0}{1}{2}. Trace:\n{3}\n".
333         format(target_id, ".", name), value))

```

**Py4JJavaError:** An error occurred while calling o24921.text.  
: java.lang.UnsatisfiedLinkError: 'boolean org.apache.hadoop.io.nativeio.NativeIO\$Windows.access0(java.lang.String, int)'

```

    at org.apache.hadoop.io.nativeio.NativeIO$Windows.access0(Native Method)
    at org.apache.hadoop.io.nativeio.NativeIO$Windows.access(NativeIO.java:81
7)
    at org.apache.hadoop.fs.FileUtil.canRead(FileUtil.java:1415)
    at org.apache.hadoop.fs.FileUtil.list(FileUtil.java:1620)
    at org.apache.hadoop.fs.RawLocalFileSystem.listStatus(RawLocalFileSystem.
java:739)
    at org.apache.hadoop.fs.FileSystem.listStatus(FileSystem.java:2078)
    at org.apache.hadoop.fs.FileSystem.listStatus(FileSystem.java:2122)
    at org.apache.hadoop.fs.ChecksumFileSystem.listStatus(ChecksumFileSystem.
java:961)
    at org.apache.spark.util.HadoopFSUtils$.listLeafFiles(HadoopFSUtils.scala
a:218)
    at org.apache.spark.util.HadoopFSUtils$.anonfun$parallelListLeafFilesInte
rnal$1(HadoopFSUtils.scala:132)
    at scala.collection.immutable.List.map(List.scala:247)
    at scala.collection.immutable.List.map(List.scala:79)
    at org.apache.spark.util.HadoopFSUtils$.parallelListLeafFilesInternal(Had
oopFSUtils.scala:122)
    at org.apache.spark.util.HadoopFSUtils$.parallelListLeafFiles(HadoopFSuti
ls.scala:72)
    at org.apache.spark.sql.execution.datasources.InMemoryFileIndex$.bulkList
LeafFiles(InMemoryFileIndex.scala:178)
    at org.apache.spark.sql.execution.datasources.InMemoryFileIndex.listLeafF
iles(InMemoryFileIndex.scala:134)

```

```
    at org.apache.spark.sql.execution.datasources.InMemoryFileIndex.refresh0
(InMemoryFileIndex.scala:98)

    at org.apache.spark.sql.execution.datasources.InMemoryFileIndex.<init>(In
MemoryFileIndex.scala:70)

    at org.apache.spark.sql.execution.datasources.DataSource.createInMemoryFi
leIndex(DataSource.scala:563)

    at org.apache.spark.sql.execution.datasources.DataSource.resolveRelation
(DataSource.scala:420)

    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource.org$apache$sp
ark$sql$catalyst$analysis$ResolveDataSource$$loadV1BatchSource(ResolveDataSource.
scala:143)

    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource$$anonfun$appl
y$1.$anonfun$applyOrElse$2(ResolveDataSource.scala:61)

    at scala.Option.getOrElse(Option.scala:201)

    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource$$anonfun$appl
y$1.applyOrElse(ResolveDataSource.scala:61)

    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource$$anonfun$appl
y$1.applyOrElse(ResolveDataSource.scala:45)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.$anonfun$re
solveOperatorsUpWithPruning$3(AnalysisHelper.scala:139)

    at org.apache.spark.sql.catalyst.trees.CurrentOrigin$.withOrigin(origin.s
cala:86)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.$anonfun$re
solveOperatorsUpWithPruning$1(AnalysisHelper.scala:139)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$.allowInvok
ingTransformsInAnalyzer(AnalysisHelper.scala:416)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.resolveOper
atorsUpWithPruning(AnalysisHelper.scala:135)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.resolveOper
atorsUpWithPruning$(AnalysisHelper.scala:131)

    at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolveOperato
rsUpWithPruning(LogicalPlan.scala:37)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.resolveOper
atorsUp(AnalysisHelper.scala:112)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper.resolveOper
atorsUp$(AnalysisHelper.scala:111)

    at org.apache.spark.sql.catalyst.plans.logical.LogicalPlan.resolveOperato
rsUp(LogicalPlan.scala:37)

    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource.apply(Resolve
DataSource.scala:45)
```



```
    at org.apache.spark.sql.catalyst.analysis.ResolveDataSource.apply(ResolveDataSource.scala:43)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.$anonfun$execute$2(RuleExecutor.scala:242)

    at scala.collection.LinearSeqOps.foldLeft(LinearSeq.scala:183)

    at scala.collection.LinearSeqOps.foldLeft$(LinearSeq.scala:179)

    at scala.collection.immutable.List.foldLeft(List.scala:79)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.$anonfun$execute$1(RuleExecutor.scala:239)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.$anonfun$execute$1$adapted(RuleExecutor.scala:231)

    at scala.collection.immutable.List.foreach(List.scala:334)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.execute(RuleExecutor.scala:231)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.org$apache$spark$sql$catalyst$analysis$Analyzer$$executeSameContext(Analyzer.scala:290)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.$anonfun$execute$1(Analyzer.scala:286)

    at org.apache.spark.sql.catalyst.analysis.AnalysisContext$.withNewAnalysisContext(Analyzer.scala:234)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.execute(Analyzer.scala:286)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.execute(Analyzer.scala:249)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.$anonfun$executeAndTrack$1(RuleExecutor.scala:201)

    at org.apache.spark.sql.catalyst.QueryPlanningTracker$.withTracker(QueryPlanningTracker.scala:89)

    at org.apache.spark.sql.catalyst.rules.RuleExecutor.executeAndTrack(RuleExecutor.scala:201)

    at org.apache.spark.sql.catalyst.analysis.resolver.HybridAnalyzer.resolveInFixedPoint(HybridAnalyzer.scala:190)

    at org.apache.spark.sql.catalyst.analysis.resolver.HybridAnalyzer.$anonfun$apply$1(HybridAnalyzer.scala:76)

    at org.apache.spark.sql.catalyst.analysis.resolver.HybridAnalyzer.withTrackedAnalyzerBridgeState(HybridAnalyzer.scala:111)

    at org.apache.spark.sql.catalyst.analysis.resolver.HybridAnalyzer.apply(HybridAnalyzer.scala:71)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.$anonfun$executeAndChe
```

```
ck$1(Analyzer.scala:280)

    at org.apache.spark.sql.catalyst.plans.logical.AnalysisHelper$.markInAnal
yzer(AnalysisHelper.scala:423)

    at org.apache.spark.sql.catalyst.analysis.Analyzer.executeAndCheck(Analyz
er.scala:280)

    at org.apache.spark.sql.execution.QueryExecution.$anonfun$lazyAnalyzed$2
(QueryExecution.scala:110)

    at org.apache.spark.sql.catalyst.QueryPlanningTracker.measurePhase(QueryP
lanningTracker.scala:148)

    at org.apache.spark.sql.execution.QueryExecution.$anonfun$executePhase$2
(QueryExecution.scala:278)

    at org.apache.spark.sql.execution.QueryExecution$.withInternalError(Query
Execution.scala:654)

    at org.apache.spark.sql.execution.QueryExecution.$anonfun$executePhase$1
(QueryExecution.scala:278)

    at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:804)

    at org.apache.spark.sql.execution.QueryExecution.executePhase(QueryExecut
ion.scala:277)

    at org.apache.spark.sql.execution.QueryExecution.$anonfun$lazyAnalyzed$1
(QueryExecution.scala:110)

    at scala.util.Try$.apply(Try.scala:217)

    at org.apache.spark.util.Utils$.doTryWithCallerStacktrace(Utils.scala:137
8)

    at org.apache.spark.util.LazyTry.tryT$lzycompute(LazyTry.scala:46)

    at org.apache.spark.util.LazyTry.tryT(LazyTry.scala:46)

    at org.apache.spark.util.LazyTry.get(LazyTry.scala:58)

    at org.apache.spark.sql.execution.QueryExecution.analyzed(QueryExecution.
scala:121)

    at org.apache.spark.sql.execution.QueryExecution.assertAnalyzed(QueryExec
ution.scala:80)

    at org.apache.spark.sql.classic.Dataset$. $anonfun$ofRows$1(Dataset.scala:
115)

    at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:804)

    at org.apache.spark.sql.classic.Dataset$.ofRows(Dataset.scala:113)

    at org.apache.spark.sql.classic.DataFrameReader.load(DataFrameReader.scal
a:109)

    at org.apache.spark.sql.classic.DataFrameReader.load(DataFrameReader.scal
a:58)
```

```

        at org.apache.spark.sql.DataFrameReader.text(DataFrameReader.scala:535)
        at org.apache.spark.sql.classic.DataFrameReader.text(DataFrameReader.scala:329)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:569)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:184)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:108)
        at java.base/java.lang.Thread.run(Thread.java:840)

```

```

In [219... # ----- 4. FIȘIERE DE INTRARE -----
input_dir = "stream_input"
os.makedirs(input_dir, exist_ok=True)
processed = set()

```

```

In [220... # ----- 5. LOOP SIMPLU DE STREAMING -----
print("Streaming activ. Copiază fișiere CSV în 'stream_input/'...)

start = time.time()
timeout = 60 # rulează 1 minut

while time.time() - start < timeout:
    for file in os.listdir(input_dir):
        if file.endswith(".csv") and file not in processed:
            try:
                print(f"\nFișier detectat: {file}")
                df = pd.read_csv(os.path.join(input_dir, file)).dropna()
                sdf = spark.createDataFrame(df)
                sdf = sdf.withColumn("year", col("year").cast("int"))
                result = loaded_model.transform(sdf)

                # Convertim predicția în Label (Low/Medium/High)
                decoder = IndexToString(
                    inputCol="prediction",
                    outputCol="predicted_category",

```

```

        labels=["Low", "Medium", "High"]
    )
    final = decoder.transform(result)

    # Print stabil, fără .show()!
    for row in final.select("countries", "region", "predicted_category"):
        print(f"{row['countries']} ({row['region']}): {row['predicted_category']}")

    processed.add(file)

except Exception as e:
    print(f"Eroare la {file}: {e}")

time.sleep(5)

print("\nStreaming finalizat.")

```

Streaming activ. Copiază fişiere CSV în 'stream\_input/...'.

Fişier detectat: input\_auto\_1.csv

Eroare la input\_auto\_1.csv: name 'loaded\_model' is not defined

Fişier detectat: input\_auto\_2.csv

Eroare la input\_auto\_2.csv: name 'loaded\_model' is not defined

Fişier detectat: input\_auto\_3.csv

Eroare la input\_auto\_3.csv: name 'loaded\_model' is not defined

```

-----
KeyboardInterrupt                                Traceback (most recent call last)
Cell In[220], line 34
     31         except Exception as e:
     32             print(f"Eroare la {file}: {e}")
--> 34         time.sleep(5)
     36 print("\nStreaming finalizat.")

KeyboardInterrupt:

```