UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA: BAZE DE DATE ȘI TEHNOLOGII SOFTWARE

BIG DATA

PREDICȚIA TIPURILOR DE VIN ȘI CALITĂȚII ACESTORA ÎN FUCȚIE DE SPECIFICAȚII

PROFESOR COORDONATOR: **Letiția Ana MARIN**

STUDENT: **Valeria Gabriela SPÎNU**

GRUPA: **405**

PREDICȚIA TIPURILOR DE VIN ȘI CALITĂȚII ACESTORA ÎN FUCȚIE DE SPECIFICAȚII

1. INTRODUCERE

Vinul este o băutură alcoolică pe bază de struguri, larg răspândită pe întregul glob. Zona de cultivare și soiul strugurilor, metoda de vinificație și tipul de îmbătrânire al vinului, cantitatea de zahăr și nivelul de aciditate sunt doar câteva aspecte care au condus la apariția unei game foarte variate de vinuri. Acestea se pot clasifica în foarte multe categorii, însă principalele deosebiri se fac în funcție de culoare: vinuri roșii, albe sau roze; în funcție de cantitatea de zahăr conținută într-un litru: sec, demisec, demidulce și dulce; și în funcție de calitate : vin de masă (tărie alcoolică între 8,5 – 9,5%), vin de calitate superioară (tărie alcoolică peste 9,5%) și vinuri de origine controlată (tărie alcoolică peste 11%).

1) Prezentarea succintă a setului de date

Setul de date folosit pentru realizarea acestui proiect se referă la vinurile Vinho Verde, produse în regiunea Minho, din Portugalia. Datasetul este descărcat de pe platforma Kaggle, având în jur de 6500 de intrari. Fiecare rând conține informații precum :

- Tipul vinului : roșu sau alb,
- Aciditatea fixă : reprezentată de acizii tartric, malic, citric și succinic
- Aciditatea volatilă : în principal este reprezentată de acidul acetic
- Acidul citric : conferă prospețime vinului
- Zahărul rezidual : zahărul din struguri care rămâne după fermentație
- Cloruri: concentrația de cloruri din vin
- Dioxid de sulf liber : cunoscut și sub numele de sulfiți
- Dioxid de sulf total
- Densitatea
- Ph-ul
- Alcoolul
- Calitatea

2) Enunțarea obiectivelor

Principalele task-uri care se doresc a fi implementate în cadrul acestui proiect sunt legate de predicția calității și tipului de vin în funcție de specificațiile prezente. Cele două task-uri vor fi realizate prin clasificare.

2. SCRIPTURI SPARK

1) Pregătirea și curățarea datelor

În prima etapă s-au citit datele din fișierul csv, observându-se faptul că majoritatea atributelor sunt de tip double și numele coloanelor este format atât din cuvinte cât și din spații. În acest sens, s-a definit o nouă schemă folosită pentru a modifica tipul în float și numele atributelor.

```
#Crearea schemei
schema = tp.StructType([
    tp.StructField(name = 'type', dataType= tp.StringType(), nullable=True),
    tp.StructField(name = 'fixed_acidity', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'volatile_acidity', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'citric_acid', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'residual_sugar', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'chlorides', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'free_sulfur_dioxide', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'total_sulfur_dioxide', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'density', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'pH', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'sulphates', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'alcohol', dataType= tp.FloatType(), nullable=True),
    tp.StructField(name = 'quality', dataType= tp.IntegerType(), nullable=True),
])
```

Ulterior s-a verificat dacă există înregistrări care conțin null-uri, acestea fiind șterse. Pentru următoarele cerințe s-a folosit dataframe-ul curățat.

```
#Numararea valorilor null din fiecare coloanal
df_null = df.agg(*[f.count(f.when(f.isnull(c),c)).alias(c) for c in df.columns])
df_null.show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	10	8	3	2	2	0	0	0	9	4	0	0

```
#Afisarea numarului de recorduri din dataset
print((df.count(),len(df.columns)),"- Numarul total de recorduri")
#Stergerea recordurilor care contin valori nule
df_wo_null = df.na.drop()
print((df_wo_null.count(),len(df_wo_null.columns)),"- Numarul de recorduri care nu contin valori nule")
```

```
(6497, 13) - Numarul total de recorduri
(6463, 13) - Numarul de recorduri care nu contin valori nule
```

În final, s-au prezentat statisticile aferente setului de date, acestea fiind foarte folositoare în capitolul imediat următor.

```
#Afisarea statisticilor aferente setului de date
df_wo_null.summary().show()
```

summary	type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH
count	6463	6463	6463	6463	6463	6463	6463	6463	6463	6463
mean	null	7.217754916142849	0.33958920011962324	0.3187575435210276	5.443957910658718	0.05605011259514316	30.516865232863996	115.69449172211047	0.9946976773937697	3.218332045295558
stddev	null	1.297913064851731	0.16463923932869953	0.14525227718075692	4.756851687027275	0.03507582472555318 5	17.75881549996 71	56.52673616037642	0.003001444373663...	0.16064989975877242
min	red	3.8	0.08	0.0	0.6	0.009	1.0	6.0	0.98711	2.72
25%	null	6.4	0.23	0.25	1.8	0.038	17.0	77.0	0.99232	3.11
50%	null	7.0	0.29	0.31	3.0	0.047	29.0	118.0	0.99489	3.21
75%	null	7.7	0.4	0.39	8.1	0.065	41.0	156.0	0.997	3.32
max	white	15.9	1.58	1.66	65.8	0.611	289.0	440.0	1.03898	4.01

2) Procesarea datelor

a. Dataframes

Pentru dataframe-ul definit anterior, s-a adăugat coloana category care reușește să clasifice dacă un vin este rău, bun sau excelent. Această coloană este folosită în filtrări și query-uri SQL.

```
# Sa se adauge o coloana in care sa se specifice categoria de vin pentru fiecare record
df_wo_null = df_wo_null.withColumn("category", f.expr("case when quality <=4 then 'Bad' when quality <= 7 then 'Good' when quality>7 then 'Excellent' end"))
df_wo_null.show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.001	3.0	0.45	8.8	6	Good
white	6.3	0.3	0.34	1.6	0.049	14.0	132.0	0.994	3.3	0.49	9.5	6	Good
white	8.1	0.28	0.4	6.9	0.05	30.0	97.0	0.9951	3.26	0.44	10.1	6	Good
white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	6	Good
white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	6	Good
white	8.1	0.28	0.4	6.9	0.05	30.0	97.0	0.9951	3.26	0.44	10.1	6	Good
white	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.9949	3.18	0.47	9.6	6	Good
white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.001	3.0	0.45	8.8	6	Good
white	6.3	0.3	0.34	1.6	0.049	14.0	132.0	0.994	3.3	0.49	9.5	6	Good
white	8.1	0.22	0.43	1.5	0.044	28.0	129.0	0.9938	3.22	0.45	11.0	6	Good
white	8.1	0.27	0.41	1.45	0.033	11.0	63.0	0.9908	2.99	0.56	12.0	5	Good
white	8.6	0.23	0.4	4.2	0.035	17.0	109.0	0.9947	3.14	0.53	9.7	5	Good
white	7.9	0.18	0.37	1.2	0.04	16.0	75.0	0.992	3.18	0.63	10.8	5	Good
white	6.6	0.16	0.4	1.5	0.044	48.0	143.0	0.9912	3.54	0.52	12.4	7	Good
white	8.3	0.42	0.62	19.25	0.04	41.0	172.0	1.0002	2.98	0.67	9.7	5	Good
white	6.6	0.17	0.38	1.5	0.032	28.0	112.0	0.9914	3.25	0.55	11.4	7	Good
white	6.3	0.48	0.04	1.1	0.046	30.0	99.0	0.9928	3.24	0.36	9.6	6	Good
white	7.4	0.34	0.42	1.1	0.033	17.0	171.0	0.9917	3.12	0.53	11.3	6	Good
white	6.5	0.31	0.14	7.5	0.044	34.0	133.0	0.9955	3.22	0.5	9.5	5	Good
white	6.2	0.66	0.48	1.2	0.029	29.0	75.0	0.9892	3.33	0.39	12.8	8	Excellent

Ulterior, s-au realizat anumite filtrări și grupări pentru a analiza mai bine datele.

```
#Sa se afiseze primele 20 de inregistrari care au o cantitate de zahar de peste 10 iar alcoolul continut sa fie m ai mare de 12. Se vor afisa doar atributele type, residual_sugar, alcohol si quality
df_wo_null.filter("residual_sugar > 10 AND alcohol > 12").select(['type','residual_sugar','alcohol','quality']).s how()
```

type	residual_sugar	alcohol	quality
white	14.0	12.2	7
white	14.5	12.5	8
white	12.8	12.2	8
white	15.75	12.1	6
white	11.0	12.2	6
white	10.6	12.8	6
white	11.1	13.0	6
white	10.55	12.7	6
white	10.2	12.1	7
white	12.75	12.9	6
white	14.5	12.5	5
white	11.25	12.4	6
white	11.25	12.4	6
white	11.3	12.8	7
white	15.5	13.0	7
white	15.5	13.0	7
white	15.5	13.0	7
white	12.9	13.0	6
white	10.8	13.6	6
white	10.2	12.2	7

```
#Sa se determine cate inregistrari apartin fiecarei categorii
df_wo_null.groupBy('category').count().show()
```

category	count
Excellent	197
Good	6022
Bad	244

```
#Sa se determine cate cate inregistrari apartin fiecarui tip de vinuri
df_wo_null.groupBy('type').count().show()
```

type	count
white	4870
red	1593

```
#Sa se afiseze minimul pentru fiecare coloana, in functie de coloana quality
df_wo_null.groupBy('quality').min().show()
```

quality	min(fixed_acidity)	min(volatile_acidity)	min(citric_acid)	min(residual_sugar)	min(chlorides)	min(free_sulfur_dioxide)	min(total_sulfur_dioxide)	min(density)	min(pH)	min(sulphates)	min(alcohol)	min(quality)
6	3.8	0.08	0.0	0.7	0.015	1.0	6.0	0.98758	2.72	0.23	8.4	6
3	4.2	0.17	0.0	0.7	0.022	3.0	9.0	0.9911	2.87	0.28	8.0	3
5	4.5	0.1	0.0	0.6	0.009	2.0	6.0	0.98722	2.79	0.27	8.0	5
9	6.6	0.24	0.29	1.6	0.018	24.0	85.0	0.98965	3.2	0.36	10.4	9
4	4.6	0.11	0.0	0.7	0.013	3.0	7.0	0.9892	2.74	0.25	8.4	4
8	3.9	0.12	0.03	0.8	0.014	3.0	12.0	0.98713	2.88	0.25	8.5	8
7	4.2	0.08	0.0	0.9	0.012	3.0	7.0	0.98711	2.84	0.22	8.6	7

```
#Sa se determine inregistrarile care au total_sulfur_dioxide peste 150 si pH-ul peste 3
df_wo_null.filter((df_wo_null['pH']>3) & (df_wo_null['total_sulfur_dioxide']> 150)).show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	6	Good
white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.4	9.9	6	Good
white	7.4	0.34	0.42	1.1	0.033	17.0	171.0	0.9917	3.12	0.53	11.3	6	Good
white	7.6	0.67	0.14	1.5	0.074	25.0	168.0	0.9937	3.05	0.51	9.3	5	Good
white	7.0	0.25	0.32	9.0	0.046	56.0	245.0	0.9955	3.25	0.5	10.4	6	Good
white	5.8	0.27	0.2	14.95	0.044	22.0	179.0	0.9962	3.37	0.37	10.2	5	Good
white	6.7	0.23	0.39	2.5	0.172	63.0	158.0	0.9937	3.11	0.36	9.4	6	Good
white	6.7	0.24	0.39	2.9	0.173	63.0	157.0	0.9937	3.1	0.34	9.4	6	Good
white	7.0	0.31	0.26	7.4	0.069	28.0	160.0	0.9954	3.13	0.46	9.8	6	Good
white	6.6	0.24	0.27	1.4	0.057	33.0	152.0	0.9934	3.22	0.56	9.5	6	Good
white	6.7	0.23	0.26	1.4	0.06	33.0	154.0	0.9934	3.24	0.56	9.5	6	Good
white	7.4	0.18	0.31	1.4	0.058	38.0	167.0	0.9931	3.16	0.53	10.0	7	Good
white	6.2	0.45	0.26	4.4	0.063	63.0	206.0	0.994	3.27	0.52	9.8	4	Bad
white	6.2	0.46	0.25	4.4	0.066	62.0	207.0	0.9939	3.25	0.52	9.8	5	Good
white	7.0	0.31	0.26	7.4	0.069	28.0	160.0	0.9954	3.13	0.46	9.8	6	Good
white	7.2	0.19	0.31	1.6	0.062	31.0	173.0	0.9917	3.35	0.44	11.7	6	Good
white	6.9	0.25	0.35	1.3	0.039	29.0	191.0	0.9908	3.13	0.52	11.0	6	Good
white	7.2	0.21	0.34	11.9	0.043	37.0	213.0	0.9962	3.09	0.5	9.6	6	Good
white	7.0	0.47	0.07	1.1	0.035	17.0	151.0	0.991	3.02	0.34	10.5	5	Good
white	6.7	0.25	0.13	1.2	0.041	81.0	174.0	0.992	3.14	0.42	9.8	5	Good

```
#Sa se determine inregistrarile care fac parte din categoria 'Excellent' si sunt rosii
df_wo_null.filter((df_wo_null['type']=='red') & (df_wo_null['category']== 'Excellent')).show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
red	7.9	0.35	0.46	3.6	0.078	15.0	37.0	0.9973	3.35	0.86	12.8	8	Excellent
red	10.3	0.32	0.45	6.4	0.073	5.0	13.0	0.9976	3.23	0.82	12.6	8	Excellent
red	5.6	0.85	0.05	1.4	0.045	12.0	88.0	0.9924	3.56	0.82	12.9	8	Excellent
red	12.6	0.31	0.72	2.2	0.072	6.0	29.0	0.9987	2.88	0.82	9.8	8	Excellent
red	11.3	0.62	0.67	5.2	0.086	6.0	19.0	0.9988	3.22	0.69	13.4	8	Excellent
red	9.4	0.3	0.56	2.8	0.08	6.0	17.0	0.9964	3.15	0.92	11.7	8	Excellent
red	10.7	0.35	0.53	2.6	0.07	5.0	16.0	0.9972	3.15	0.65	11.0	8	Excellent
red	10.7	0.35	0.53	2.6	0.07	5.0	16.0	0.9972	3.15	0.65	11.0	8	Excellent
red	5.0	0.42	0.24	2.0	0.06	19.0	50.0	0.9917	3.72	0.74	14.0	8	Excellent
red	7.8	0.57	0.09	2.3	0.065	34.0	45.0	0.99417	3.46	0.74	12.7	8	Excellent
red	9.1	0.4	0.5	1.8	0.071	7.0	16.0	0.99462	3.21	0.69	12.5	8	Excellent
red	10.0	0.26	0.54	1.9	0.083	42.0	74.0	0.99451	2.98	0.63	11.8	8	Excellent
red	7.9	0.54	0.34	2.5	0.076	8.0	17.0	0.99235	3.2	0.72	13.1	8	Excellent
red	8.6	0.42	0.39	1.8	0.068	6.0	12.0	0.99516	3.35	0.69	11.7	8	Excellent
red	5.5	0.49	0.03	1.8	0.044	28.0	87.0	0.9908	3.5	0.82	14.0	8	Excellent
red	7.2	0.33	0.33	1.7	0.061	3.0	13.0	0.996	3.23	1.1	10.0	8	Excellent
red	7.2	0.38	0.31	2.0	0.056	15.0	29.0	0.99472	3.23	0.76	11.3	8	Excellent
red	7.4	0.36	0.3	1.8	0.074	17.0	24.0	0.99419	3.24	0.7	11.4	8	Excellent

b. Spark SQL

Pentru a putea folosi Spark SQL, a fost nevoie să se creeze o vizualizare temporală numită wine.

```
#Crearea unei vizualizari temporare pentru folosirea comenzilor SQL
df_wo_null.createOrReplaceTempView('wine')
```

```
#Sa se afiseze descrescator primele 10 vinuri care contin cea mai mare cantitate de alcool
output = spark.sql("SELECT * FROM wine ORDER BY alcohol DESC LIMIT 10")
output.show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
red	15.9	0.36	0.65	7.5	0.096	22.0	71.0	0.9976	2.98	0.84	14.9	5	Good
white	6.4	0.35	0.28	1.6	0.037	31.0	113.0	0.98779	3.12	0.4	14.2	7	Good
white	5.8	0.61	0.01	8.4	0.041	31.0	104.0	0.9909	3.26	0.72	14.05	7	Good
white	4.9	0.33	0.31	1.2	0.016	39.0	150.0	0.98713	3.33	0.59	14.0	8	Excellent
red	5.2	0.34	0.0	1.8	0.05	27.0	63.0	0.9916	3.68	0.79	14.0	6	Good
white	5.8	0.29	0.21	2.6	0.025	12.0	120.0	0.9894	3.39	0.79	14.0	7	Good
red	5.2	0.34	0.0	1.8	0.05	27.0	63.0	0.9916	3.68	0.79	14.0	6	Good
white	5.8	0.39	0.47	7.5	0.027	12.0	88.0	0.9907	3.38	0.45	14.0	6	Good
red	8.8	0.46	0.45	2.6	0.065	7.0	18.0	0.9947	3.32	0.79	14.0	6	Good
red	5.0	0.42	0.24	2.0	0.06	19.0	50.0	0.9917	3.72	0.74	14.0	8	Excellent

```
#Sa se afiseze vinurile grupate dupa tip si categorie
output = spark.sql("SELECT DISTINCT type, category, count(*) as count FROM wine GROUP BY type, category").show()
```

type	category	count
red	Excellent	18
red	Good	1513
white	Excellent	179
white	Bad	182
white	Good	4509
red	Bad	62

```
#Sa se afiseze primele 3 vinuri cu aciditatea cea mai mica care au o densitate peste 0.995
output = spark.sql("SELECT * FROM wine WHERE density > 0.995 ORDER BY fixed_acidity ASC").show(3)
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
white	4.2	0.215	0.23	5.1	0.041	64.0	157.0	0.99688	3.42	0.44	8.0	3	Bad
white	4.9	0.235	0.27	11.75	0.03	34.0	118.0	0.9954	3.07	0.5	9.4	6	Good
white	5.0	0.235	0.27	11.75	0.03	34.0	118.0	0.9954	3.07	0.5	9.4	6	Good

```
#Sa se afiseze media aritmetica, minimul si maximul de alcol pentru fiecare tip de vin
output = spark.sql("SELECT type, AVG(alcohol), MIN(alcohol), MAX(alcohol) FROM wine GROUP BY type").show()
```

type	avg(alcohol)	min(alcohol)	max(alcohol)
white	10.516772055968612	8.0	14.2
red	10.419617055424787	8.4	14.9

```
#Sa se afiseze vinurile care au un concentrat de cloruri peste medie, ordonandu-le descrescator dupa calitatea lor.
output = spark.sql("SELECT * FROM wine WHERE chlorides > (SELECT AVG(chlorides) FROM wine) ORDER BY quality DESC").show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
white	7.3	0.17	0.24	8.1	0.121	32.0	162.0	0.99508	3.17	0.38	10.4	8	Excellent
red	7.8	0.57	0.09	2.3	0.065	34.0	45.0	0.99417	3.46	0.74	12.7	8	Excellent
white	7.3	0.17	0.24	8.1	0.121	32.0	162.0	0.99508	3.17	0.38	10.4	8	Excellent
white	6.6	0.23	0.3	4.6	0.06	29.0	154.0	0.99142	3.23	0.49	12.2	8	Excellent
white	6.3	0.29	0.28	4.7	0.059	28.0	81.0	0.99036	3.24	0.56	12.7	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
white	7.3	0.19	0.27	13.9	0.057	45.0	155.0	0.99807	2.94	0.41	8.8	8	Excellent
red	7.9	0.35	0.46	3.6	0.078	15.0	37.0	0.9973	3.35	0.86	12.8	8	Excellent
red	10.3	0.32	0.45	6.4	0.073	5.0	13.0	0.9976	3.23	0.82	12.6	8	Excellent
red	12.6	0.31	0.72	2.2	0.072	6.0	29.0	0.9987	2.88	0.82	9.8	8	Excellent
red	11.3	0.62	0.67	5.2	0.086	6.0	19.0	0.9988	3.22	0.69	13.4	8	Excellent
red	9.4	0.3	0.56	2.8	0.08	6.0	17.0	0.9964	3.15	0.92	11.7	8	Excellent
red	10.7	0.35	0.53	2.6	0.07	5.0	16.0	0.9972	3.15	0.65	11.0	8	Excellent
red	10.7	0.35	0.53	2.6	0.07	5.0	16.0	0.9972	3.15	0.65	11.0	8	Excellent

```
#Sa se afiseze cate inregistrari au acidul citric mai mare decat 0.8
output = spark.sql("SELECT * FROM wine WHERE citric_acid > 0.8").show()
```

type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	category
white	10.2	0.44	0.88	6.2	0.049	20.0	124.0	0.9968	2.99	0.51	9.9	4	Bad
white	7.4	0.2	1.66	2.1	0.022	34.0	113.0	0.99165	3.26	0.55	12.2	6	Good
white	8.2	0.345	1.0	18.2	0.047	55.0	205.0	0.99965	2.96	0.43	9.6	5	Good
white	6.9	0.21	0.81	1.1	0.137	52.0	123.0	0.9932	3.03	0.39	9.2	6	Good
white	6.6	0.19	0.99	1.2	0.122	45.0	129.0	0.9936	3.09	0.31	8.7	6	Good
white	7.5	0.4	1.0	19.5	0.041	33.0	148.0	0.9977	3.24	0.38	12.0	6	Good
white	7.7	0.49	1.0	19.6	0.03	28.0	135.0	0.9973	3.24	0.4	12.0	6	Good
white	6.5	0.39	0.81	1.2	0.217	14.0	74.0	0.9936	3.08	0.53	9.5	5	Good
white	6.7	0.325	0.82	1.2	0.152	49.0	120.0	0.99312	2.99	0.38	9.2	5	Good
white	7.2	0.21	1.0	1.1	0.154	46.0	114.0	0.9931	2.95	0.43	9.2	6	Good
white	7.4	0.21	0.8	12.3	0.038	77.0	183.0	0.99778	2.95	0.48	9.0	5	Good
white	7.4	0.21	0.8	12.3	0.038	77.0	183.0	0.99778	2.95	0.48	9.0	5	Good
white	7.6	0.25	1.23	4.6	0.035	51.0	294.0	0.99018	3.03	0.43	13.1	6	Good
white	7.7	0.43	1.0	19.95	0.032	42.0	164.0	0.99742	3.29	0.5	12.0	6	Good
white	7.2	0.23	0.82	1.3	0.149	70.0	109.0	0.99304	2.93	0.42	9.2	6	Good
white	7.1	0.34	0.86	1.4	0.174	36.0	99.0	0.99288	2.92	0.5	9.3	5	Good
white	6.3	0.3	0.91	8.2	0.034	50.0	199.0	0.99394	3.39	0.49	11.7	6	Good
white	6.3	0.3	0.91	8.2	0.034	50.0	199.0	0.99394	3.39	0.49	11.7	6	Good
red	9.2	0.52	1.0	3.4	0.61	32.0	69.0	0.9996	2.74	2.0	9.4	4	Bad

3. METODE ML

1) Clasificarea vinurilor în funcție de calitate

Pentru implementarea acestui task a fost folosit clasificatorul DecisionTreeClassifier. Pașii folosiți sunt similari celor din cadrul cursului, dar adaptați setului de date curent. A fost necesar indexarea coloanei de tip string type, urmând mai apoi să se asambleze vectorul de caracteristici. Setul de date a fost împărțit în 80% date de training și 20% date de test. Ulterior a fost folosit un pipeline pentru antrenarea modelului și obținerea predicțiilor, la final calculându-se eroarea modelului.

```
#Setarea dataframe-ului
dataFrameQuality = df_wo_null

#Transformarea coloanei type in label numeric
typeIndexer = StringIndexer(inputCol="type", outputCol="typeNum").fit(dataFrameQuality)
dataFrameQuality = typeIndexer.transform(dataFrameQuality)

#Crearea vectorului de caracteristici
dataFrameQuality = VectorAssembler(inputCols=['typeNum','fixed_acidity','volatile_acidity', 'citric_acid', 'residu
al_sugar','chlorides',
                                   'free_sulfur_dioxide', 'total_sulfur_dioxide','density','pH','sulphates','alcohol',], outputCol
="features").transform(dataFrameQuality)


#Identificarea caracteristicilor categoriale si indexarea lor
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=10).fit(dataF
rameQuality)

#Indexarea coloanei quality-label
labelIndexer = StringIndexer(inputCol="quality", outputCol="indexedLabel").fit(dataFrameQuality)

#Partitionarea datelor
(trainingDataQuality, testDataQuality) = dataFrameQuality.randomSplit([0.8, 0.2])


#Definirea modelului
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

#Crearea pipelineului
pipeline = Pipeline(stages=[labelIndexer,featureIndexer, dt])

#Antrenarea modelului cu ajutorul pipelineului
```

```
model = pipeline.fit(trainingDataQuality)

#Realizarea predictiilor
predictions = model.transform(testDataQuality)

#Selectarea coloanelor
predictions.select("prediction", "indexedLabel", "features").show()

#Evaluarea modelului
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricNam
e="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))

#Afisarea unui sumar
treeModel = model.stages[2]
print(treeModel)
```

```
+----------+------------+--------------------+
|prediction|indexedLabel|            features|
+----------+------------+--------------------+
|       2.0|         0.0|[1.0,5.0,0.400000...|
|       0.0|         3.0|[1.0,5.0,1.019999...|
|       1.0|         1.0|[1.0,5.1999998092...|
|       0.0|         0.0|[1.0,5.1999998092...|
|       0.0|         1.0|[1.0,5.3000001907...|
|       0.0|         0.0|[1.0,5.4000000953...|
|       0.0|         1.0|[1.0,5.5999999046...|
|       0.0|         1.0|[1.0,5.5999999046...|
|       0.0|         0.0|[1.0,5.6999998092...|
|       0.0|         0.0|[1.0,5.8000001907...|
|       1.0|         1.0|[1.0,5.8000001907...|
|       2.0|         0.0|[1.0,5.9000000953...|
|       0.0|         0.0|[1.0,6.0,0.509999...|
|       0.0|         0.0|[1.0,6.0,0.509999...|
|       0.0|         0.0|[1.0,6.0,0.540000...|
|       1.0|         1.0|[1.0,6.0999999046...|
|       1.0|         2.0|[1.0,6.0999999046...|
|       0.0|         0.0|[1.0,6.0999999046...|
|       0.0|         2.0|[1.0,6.1999998092...|
|       1.0|         1.0|[1.0,6.1999998092...|
+----------+------------+--------------------+
only showing top 20 rows

Test Error = 0.466313
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_087b0b0f452d, depth=5, numNodes=41, numClasses=7, numFeatures=12
```

2) Clasificarea vinurilor în funcție de tip

Pentru acest task au fost folosiți trei clasificatori. DecisionTreeClassifier, RandomForestClassifier și LogisticRegression.

A. DecisionTreeClassifier

Și pentru cel de-al doilea task a fost folosit acest clasificator pentru a se putea compara rezultatele. Diferența față de task-ul anterior este legată de coloana label, aici folosindu-se coloana type.

```
#Crearea vectorului de caracteristici
dataFrame = VectorAssembler(inputCols=['fixed_acidity','volatile_acidity', 'citric_acid', 'residual_sugar','chloride
s', 'free_sulfur_dioxide', 'total_sulfur_dioxide','density','pH','sulphates','alcohol','quality'], outputCol="features").tr
ansform(df_wo_null)

#Indexarea coloanei type-label
```

```
labelIndexer = StringIndexer(inputCol="type", outputCol="indexedLabel").fit(dataFrame)

#Identificarea caracteristicilor categoriale si indexarea lor
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=10).fit(dataF
rame)

#Partitionarea datelor
(trainingData, testData) = dataFrame.randomSplit([0.8, 0.2])

#Definirea modelului
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

#Crearea pipelineului
pipeline = Pipeline(stages=[labelIndexer,featureIndexer, dt])

#Antrenarea modelului cu ajutorul pipelineului
model = pipeline.fit(trainingData)

#Realizarea predictiilor
predictions = model.transform(testData)

#Selectarea coloanelor
predictions.select("prediction", "indexedLabel", "features").show()

#Evaluarea modelului
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricNam
e="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test Error = %g " % (1.0 - accuracy))

#Afisarea unui sumar
treeModel = model.stages[2]
print(treeModel)
```

```
+----------+------------+--------------------+
|prediction|indexedLabel|            features|
+----------+------------+--------------------+
|       1.0|         1.0|[4.59999990463256...|
|       1.0|         1.0|[4.69999980926513...|
|       0.0|         1.0|[5.09999990463256...|
|       1.0|         1.0|[5.40000009536743...|
|       1.0|         1.0|[5.40000009536743...|
|       0.0|         1.0|[5.59999990463256...|
|       1.0|         1.0|[5.59999990463256...|
|       0.0|         1.0|[5.80000019073486...|
|       1.0|         1.0|[5.90000009536743...|
|       1.0|         1.0|[6.0,0.5,0.0,1.39...|
|       1.0|         1.0|[6.0,0.5,0.039999...|
|       1.0|         1.0|[6.0,0.5099999904...|
|       1.0|         1.0|[6.0,0.5799999833...|
|       1.0|         1.0|[6.09999990463256...|
|       1.0|         1.0|[6.19999980926513...|
|       1.0|         1.0|[6.19999980926513...|
|       1.0|         1.0|[6.19999980926513...|
|       1.0|         1.0|[6.19999980926513...|
|       1.0|         1.0|[6.19999980926513...|
|       1.0|         1.0|[6.19999980926513...|
+----------+------------+--------------------+
only showing top 20 rows

Test Error = 0.0209627
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_82f5856db87a, depth=5, numNodes=45, numClasses=2, numFeatures=12
```

B. RandomForestClassifier

Cel de-al doilea clasificator folosit pentru acest task este RandomForestClassifier. Acest clasificator este similar celui de mai sus însă mult mai bun, deoarece se creează mai multi arbori de decizie mai mici, care reușesc să mărească acuratețea modelului. Un pas suplimentar care apare în cadrul acestui clasificator este legat de conversia etichetelor indexate înapoi la etichetele originale. Din rezultate, se poate observa faptul ca acuratețea folosind acest model este mai mare față de clasificatorul anterior, test erorr fiind mai mică.

```
#Crearea vectorului de caracteristici
dataframeRFC = df_wo_null
dataframeRFC = VectorAssembler(inputCols=['fixed_acidity','volatile_acidity', 'citric_acid', 'residual_sugar','chlo
rides', 'free_sulfur_dioxide', 'total_sulfur_dioxide','density','pH','sulphates','alcohol','quality'], outputCol="features
").transform(dataframeRFC)

#Indexarea coloanei type-label
labelIndexerRFC = StringIndexer(inputCol="type", outputCol="indexedLabel").fit(dataframeRFC)

#Identificarea caracteristicilor categoriale si indexarea lor
featureIndexerRFC = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=12).fit(
dataframeRFC)

#Partitionarea datelor
(trainingDataRFC, testDataRFC) = dataframeRFC.randomSplit([0.8, 0.2])

#Definirea modelului
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# Conversia etichetelor indexate înapoi la etichetele originale
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel", labels=labelIndexerRFC.la
bels)

#Crearea pipelineului
pipeline = Pipeline(stages=[labelIndexerRFC, featureIndexerRFC, rf, labelConverter])

#Antrenarea pipeline-ului
model = pipeline.fit(trainingDataRFC)

#Realizarea predictiilor
predictionsRFC = model.transform(testDataRFC)

#Selectarea coloanelor
predictionsRFC.select("predictedLabel", "type", "features").show()

#Evaluarea modelului
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricNam
e="accuracy")
accuracy = evaluator.evaluate(predictionsRFC)
print("Test Error = %g" % (1.0 - accuracy))

#Afisarea unui sumar
rfModel = model.stages[2]
print(rfModel)
```

```
+--------------+----+--------------------+
|predictedLabel|type|            features|
+--------------+----+--------------------+
|         white| red|[5.0,1.0199999809...|
|           red| red|[5.19999980926513...|
|           red| red|[5.19999980926513...|
|           red| red|[5.30000019073486...|
|           red| red|[5.59999990463256...|
|           red| red|[5.59999990463256...|
|           red| red|[5.59999990463256...|
|           red| red|[5.69999980926513...|
|           red| red|[5.80000019073486...|
|           red| red|[6.0,0.3100000023...|
|           red| red|[6.0,0.5,0.0,1.39...|
|           red| red|[6.0,0.5099999904...|
|           red| red|[6.09999990463256...|
|           red| red|[6.09999990463256...|
|           red| red|[6.19999980926513...|
|           red| red|[6.19999980926513...|
|           red| red|[6.19999980926513...|
|           red| red|[6.19999980926513...|
|           red| red|[6.19999980926513...|
|           red| red|[6.30000019073486...|
+--------------+----+--------------------+
only showing top 20 rows

Test Error = 0.00460476
RandomForestClassificationModel: uid=RandomForestClassifier_2cf58b3f5284, numTrees=10, numClasses=2, numFeatures=12
```

C. LogisticRegression

Ultimul clasificator folosit pentru acest task este LogisticRegression, folosit în general pentru clasificarea datelor în două categorii.

```
from pyspark.ml.classification import LogisticRegression

#Setarea dataframe-ului
dataFrameLR = df_wo_null

#Transformarea coloanei type in label numeric
labelIndexer = StringIndexer(inputCol="type", outputCol="label").fit(dataFrameLR)
dataFrameLR = labelIndexer.transform(dataFrameLR)

#Crearea vectorului de caracteristici
dataFrameLR = VectorAssembler(inputCols=['fixed_acidity','volatile_acidity', 'citric_acid', 'residual_sugar','chlor
ides',
                                         'free_sulfur_dioxide', 'total_sulfur_dioxide','density','pH','sulphates','alcohol','quality'], ou
tputCol="features").transform(dataFrameLR)
#Definirea modelului
lr = LogisticRegression(maxIter=10)

#Antrenarea modelului
lrModel = lr.fit(dataFrameLR)

# Afisarea coeficientilor si interceptorilor modelului
print("Coefficients: \n" + str(lrModel.coefficientMatrix))
print("\nIntercept: " + str(lrModel.interceptVector))

trainingSummary = lrModel.summary
print("\nFalse positive rate by label:")
for i, rate in enumerate(trainingSummary.falsePositiveRateByLabel):
```

```
    print("label %d: %s" % (i, rate))

print("\nTrue positive rate by label:")
for i, rate in enumerate(trainingSummary.truePositiveRateByLabel):
    print("label %d: %s" % (i, rate))

print("\nTest Error = %g" % (1.0 - trainingSummary.accuracy))
print("\nFPR= %s" % (trainingSummary.weightedFalsePositiveRate))
print("\nTPR= %s" % (trainingSummary.weightedTruePositiveRate))
```

```
Coefficients:
DenseMatrix([[ 6.81597507e-01,  9.32577850e+00, -1.99197392e+00,
              -5.76216883e-01,  2.96878859e+01,  5.73849121e-02,
              -6.04131216e-02,  9.94256776e+02,  3.95645315e+00,
               5.66187457e+00,  7.01248516e-01,  3.77766947e-01]])

Intercept: [-1019.3413373947726]

False positive rate by label:
label 0: 0.01820464532328939
label 1: 0.003490759753593429

True positive rate by label:
label 0: 0.9965092402464065
label 1: 0.9817953546767106

Test Error = 0.00711744

FPR= 0.01457796735446289

TPR= 0.9928825622775801
```

4. METODA DL

Pentru implementarea unei metode deep learning, s-au folosit librariile pandas si tensorflow. Scopul acestei metode este cel de la task-ul 2. Pașii urmați pentru implementare sunt urmatorii: citirea datelor, transformarea coloanei type în coloană de tip numeric, ștergerea recordurilor care au valori nule, împarțirea setului de date în features(x) și label(y), apoi splitul între date de test și date de train și scalarea acestora.

```
#citirea datelor
df = pd.read_csv("/content/drive/MyDrive/BigData/proiect/winequalityN.csv")
df.head()


#transformarea coloanei type de tip string in coloana label de tip integer
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(drop='first')
encoded_type = encoder.fit_transform(df[['type']]).toarray()
df[['label']] = encoded_type


#stergerea randurilor care contin null
df = df.dropna()


#Impartirea datelor in features si label
y= df['label']
x= df.drop(['type','label'],axis =1)


#Impartirea datelor in training si test
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2)


#Scalarea datelor
from sklearn.preprocessing import MinMaxScaler
scaler =  MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Rețeaua neuronală complet conectată are input size de 12, deoarece x are 12 coloane, un prim strat ascuns cu size-ul de 12 neuroni, urmat de un alt layer de 24 de neuroni, ambele folosind ca funție de activare relu. Ultimul layer are un singur neuron iar funcția de activare este sigmoid. Optimizorul este adam iar funcția de loss folosită este binary_crossentropy.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense


# Construirea unei retele neuronale complet conectata
model = Sequential()
model.add(Dense(12, activation='relu',input_shape=(12,) ))
model.add(Dense(24, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])


#Antrenarea modelului
history = model.fit(x=X_train,
                    y=y_train,
                    epochs=50,
                    validation_data=(X_test, y_test),
                    verbose=1,batch_size=128)
```

```
41/41 [==============================] - 0s 2ms/step - loss: 0.0369 - accuracy: 0.9923 - val_loss: 0.0564 - val_accuracy: 0.9884
Epoch 30/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0368 - accuracy: 0.9925 - val_loss: 0.0562 - val_accuracy: 0.9884
Epoch 31/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0363 - accuracy: 0.9923 - val_loss: 0.0559 - val_accuracy: 0.9884
Epoch 32/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0361 - accuracy: 0.9921 - val_loss: 0.0563 - val_accuracy: 0.9876
Epoch 33/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0364 - accuracy: 0.9925 - val_loss: 0.0557 - val_accuracy: 0.9884
Epoch 34/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0357 - accuracy: 0.9923 - val_loss: 0.0552 - val_accuracy: 0.9884
Epoch 35/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0352 - accuracy: 0.9925 - val_loss: 0.0554 - val_accuracy: 0.9876
Epoch 36/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0355 - accuracy: 0.9928 - val_loss: 0.0559 - val_accuracy: 0.9876
Epoch 37/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0355 - accuracy: 0.9925 - val_loss: 0.0548 - val_accuracy: 0.9876
Epoch 38/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0350 - accuracy: 0.9925 - val_loss: 0.0541 - val_accuracy: 0.9876
Epoch 39/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0346 - accuracy: 0.9930 - val_loss: 0.0540 - val_accuracy: 0.9869
Epoch 40/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0342 - accuracy: 0.9926 - val_loss: 0.0541 - val_accuracy: 0.9884
Epoch 41/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0343 - accuracy: 0.9923 - val_loss: 0.0536 - val_accuracy: 0.9884
Epoch 42/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0339 - accuracy: 0.9926 - val_loss: 0.0534 - val_accuracy: 0.9876
Epoch 43/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0338 - accuracy: 0.9926 - val_loss: 0.0533 - val_accuracy: 0.9884
Epoch 44/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0337 - accuracy: 0.9925 - val_loss: 0.0531 - val_accuracy: 0.9884
Epoch 45/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0341 - accuracy: 0.9919 - val_loss: 0.0551 - val_accuracy: 0.9899
Epoch 46/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0335 - accuracy: 0.9926 - val_loss: 0.0531 - val_accuracy: 0.9884
Epoch 47/50
41/41 [==============================] - 0s 3ms/step - loss: 0.0335 - accuracy: 0.9923 - val_loss: 0.0526 - val_accuracy: 0.9899
Epoch 48/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0331 - accuracy: 0.9934 - val_loss: 0.0529 - val_accuracy: 0.9899
Epoch 49/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0330 - accuracy: 0.9930 - val_loss: 0.0528 - val_accuracy: 0.9884
Epoch 50/50
41/41 [==============================] - 0s 2ms/step - loss: 0.0327 - accuracy: 0.9928 - val_loss: 0.0529 - val_accuracy: 0.9884
```

```python
#Calcularea predictiilor si
y_pred = model.predict(X_test) > 0.5
y_pred = y_pred.astype(np.int32)


#Matricea de confuzie
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)


#Evaluarea modelului folosind datele de tes
model.evaluate(X_test, y_test)
```

```
array([[301,   6],
       [  9, 977]])
```

```
41/41 [==============================] - 0s 2ms/step - loss: 0.0529 - accuracy: 0.9884
[0.052926987409591675, 0.988399088382721]
```