Tema 2 - IA

Nedelcu Andrei-David 334CA

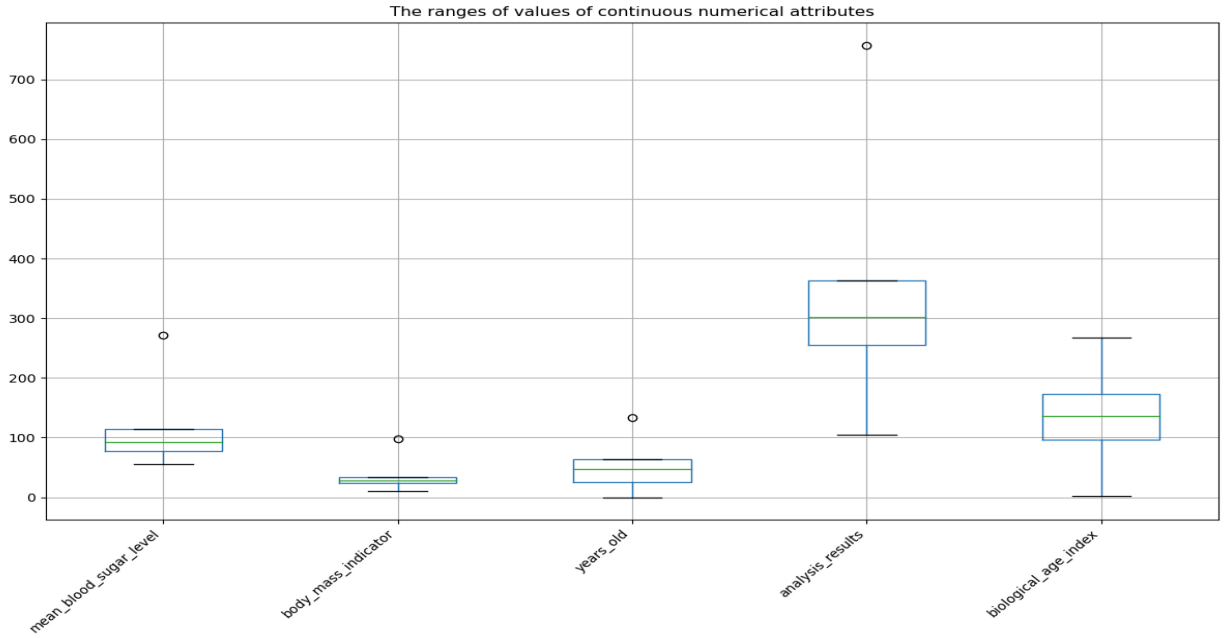
.1. Explorarea Datelor

1.1. Analiza tipului de atribute și a plajei de valori a acestora

- ***Dataset-Avc***

Valori numerice

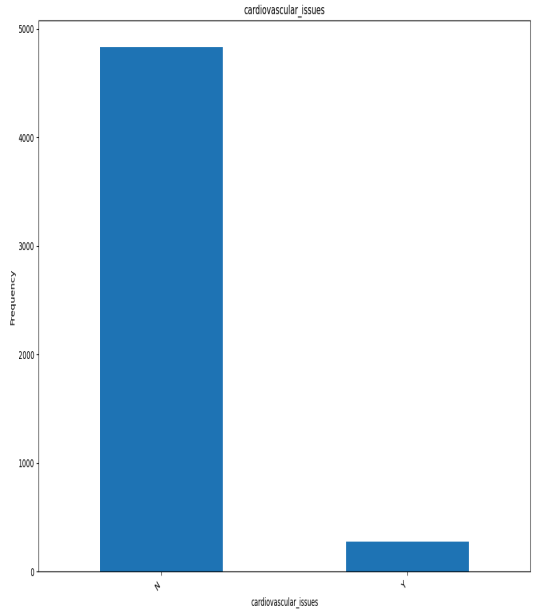
	Mean blood sugar level	Body mass indicator	Years old	Analysis results	Biological age index
count	5110	4909	5110	4599	5107
mean	106.147677	28.893237	46.568665	323.523446	134.866982
std	45.283560	7.854067	26.593912	101.577442	50.298147
min	55.120000	10.300000	0.080000	104.829714	2.915131
25%	77.245000	23.500000	26.000000	254.646209	96.807846
50%	91.885000	28.100000	47.000000	301.031628	136.386765
75%	114.090000	33.100000	63.750000	362.822769	172.535882
max	271.740000	97.600000	134.000000	756.807975	266.986321

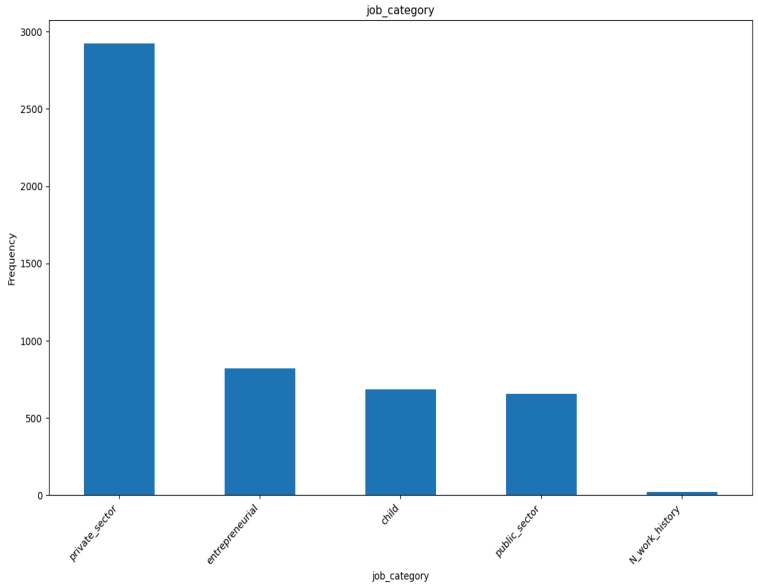


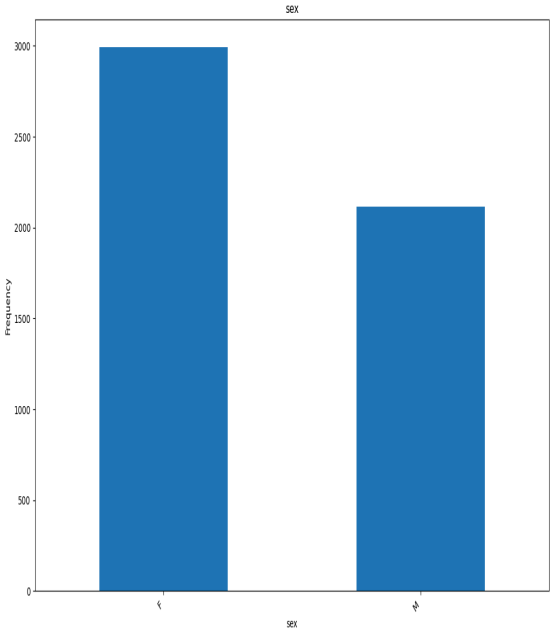
Observam ca plajele sunt foarte diferite si ca avem posibili outlieri (val indepartate de medie).

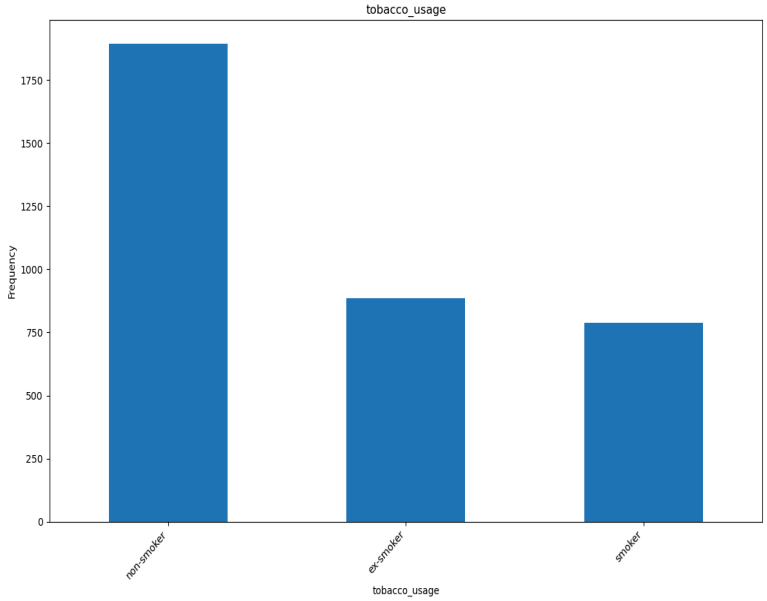
Valori tip categorii

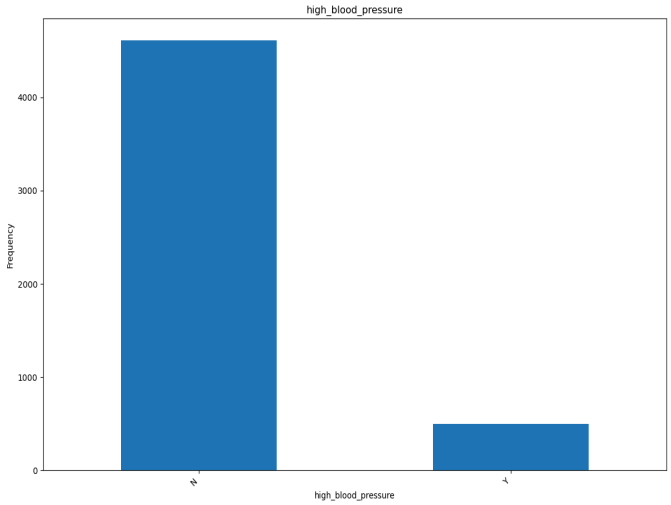
	Cardiovascular issues	Job category	Sex	Tobacco usage	High blood pressure	Married	Living area	Chaotic sleep	Cerebrovascular accident
count	5110	5110	5110	3566	5110	4599	5110	5110	5110
unique	2	5	2	3	2	2	2	2	2
top	N	private_sector	F	non-smoker	N	Y	City	N	N
freq	4834	2925	2994	1892	4612	3014	2596	4834	4861

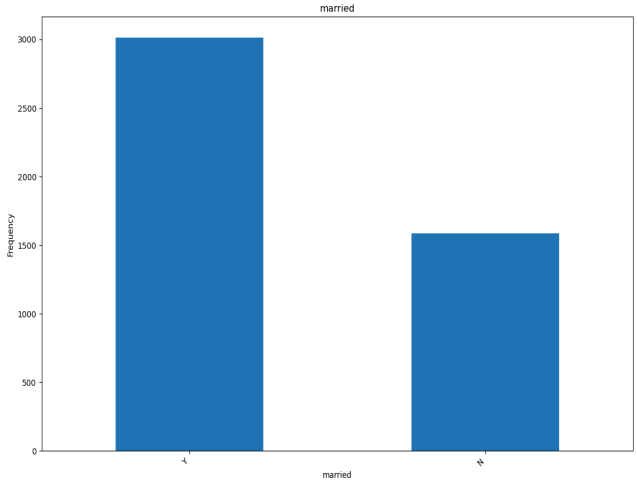


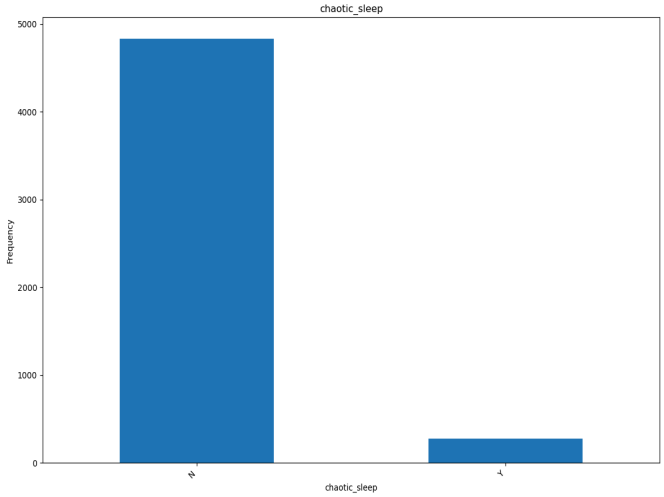


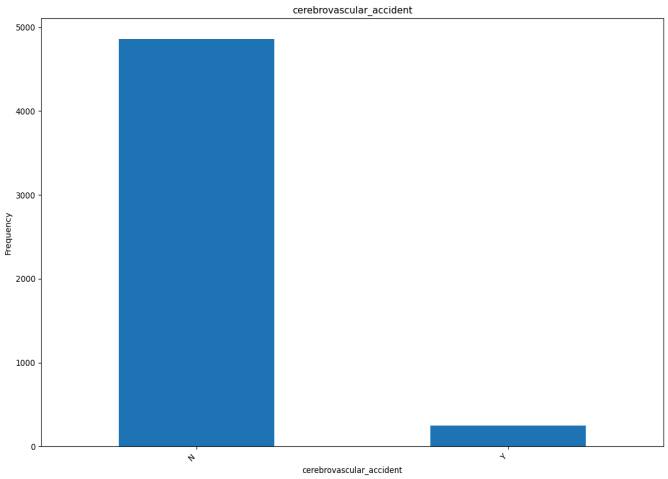


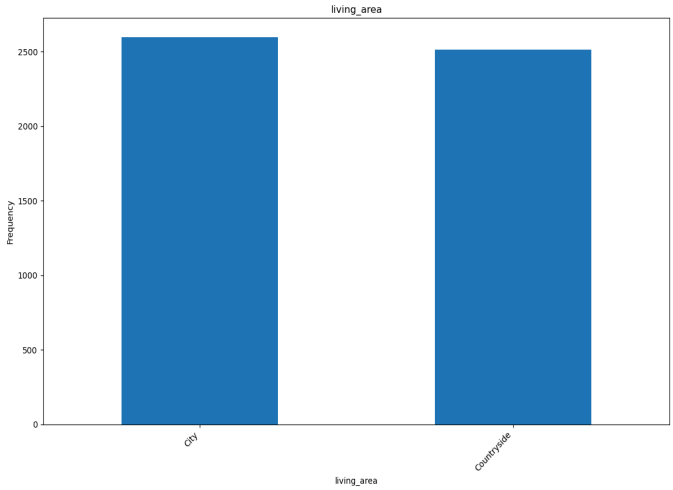



high_blood_pressure
Frequency
0
1000
2000
3000
4000
N
Y
high_blood_pressure


married
Frequency
0
500
1000
1500
2000
2500
3000
Y
N
married


chaotic_sleep
Frequency
0
1000
2000
3000
4000
5000
N
Y
chaotic_sleep

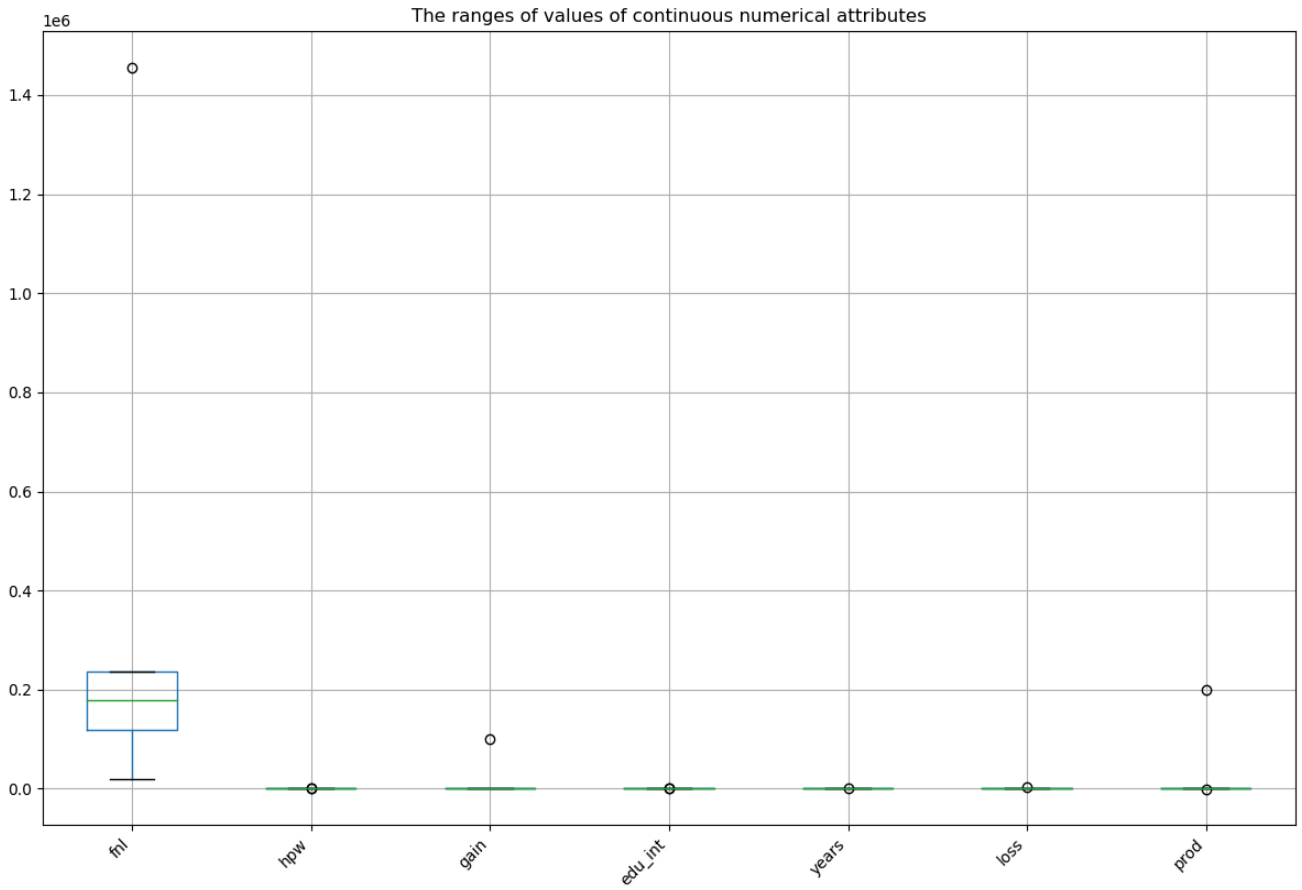

cerebrovascular_accident
Frequency
0
1000
2000
3000
4000
5000
N
Y
cerebrovascular_accident


living_area
Frequency
0
500
1000
1500
2000
2500
City
Countryside
living_area

- **Salary Prediction Dataset**

Valori Numerice

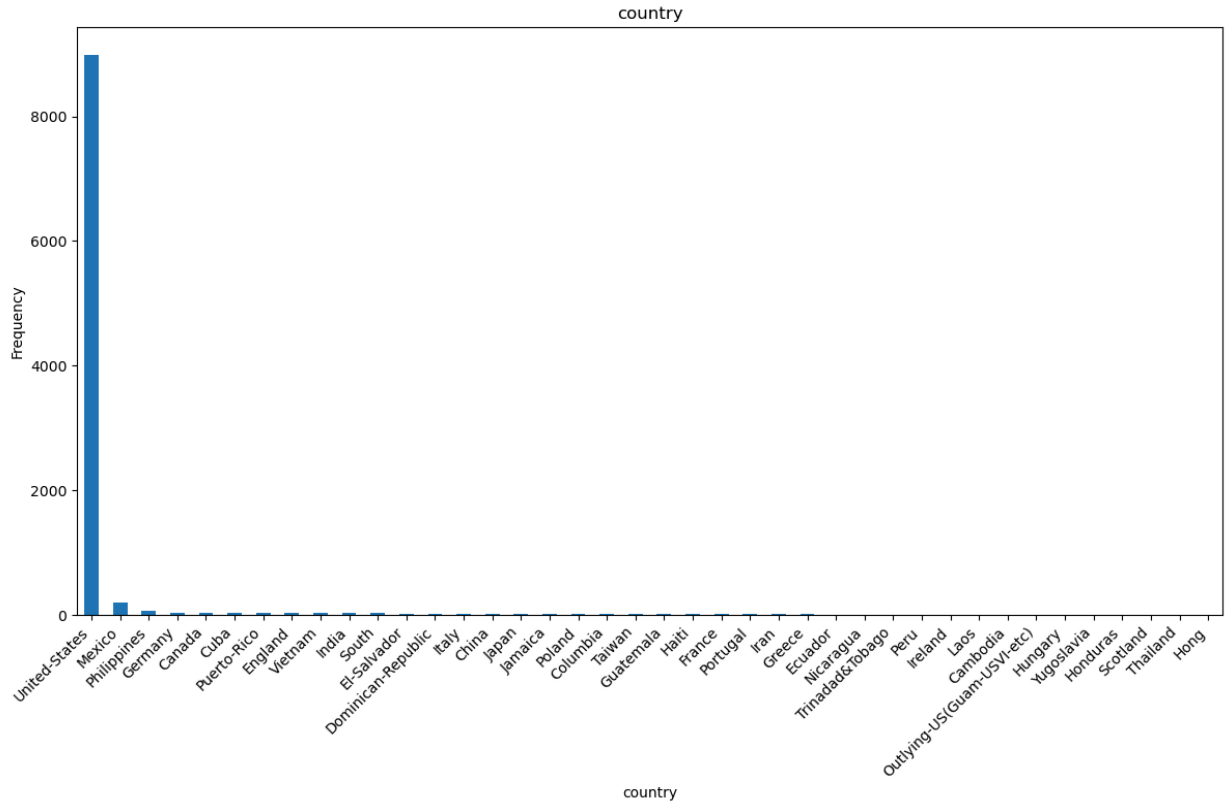
	fnl	hpw	gain	edu_int	years	loss	prod
count	9999	9199	9999	9999	9999	9999	9999
mean	190352.9	40.416241	979.853385	14.262026	38.646865	84.111411	2014.927593
std	106070.9	12.517356	7003.795382	24.770835	13.745101	394.035484	14007.604496
min	1.9214	1	0	1	17	0	-28
25%	118282.5	40	0	9	28	0	42
50%	178472	40	0	10	37	0	57
75%	237311	45	0	13	48	0	77
max	1455435	99	99999	206	90	3770	200125

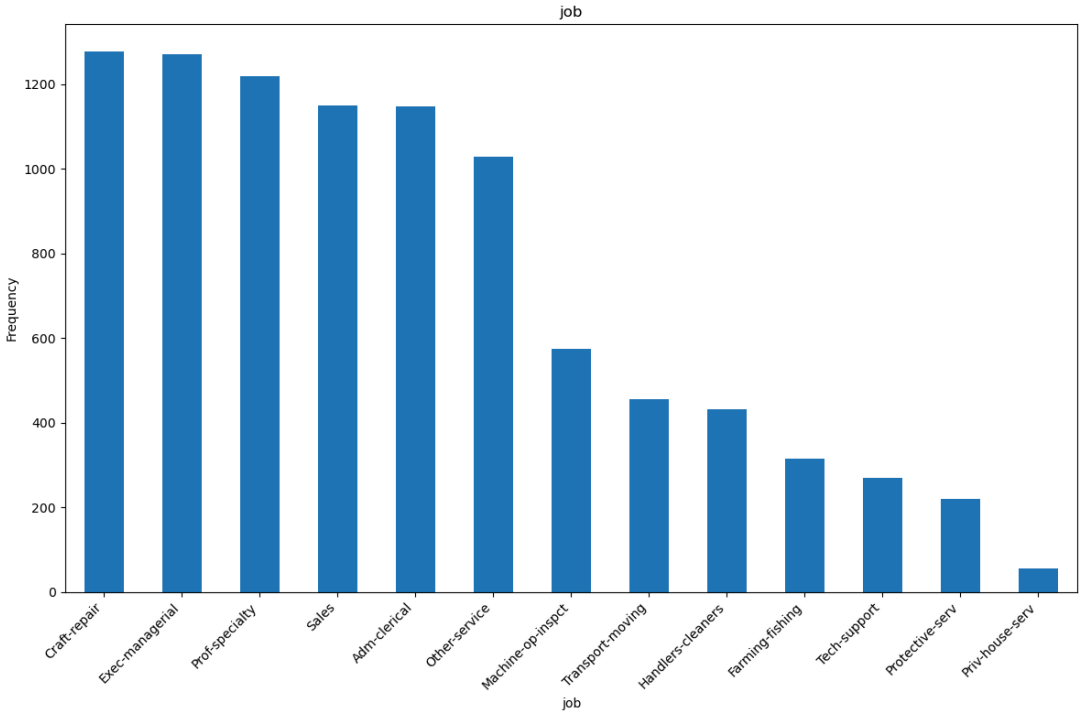


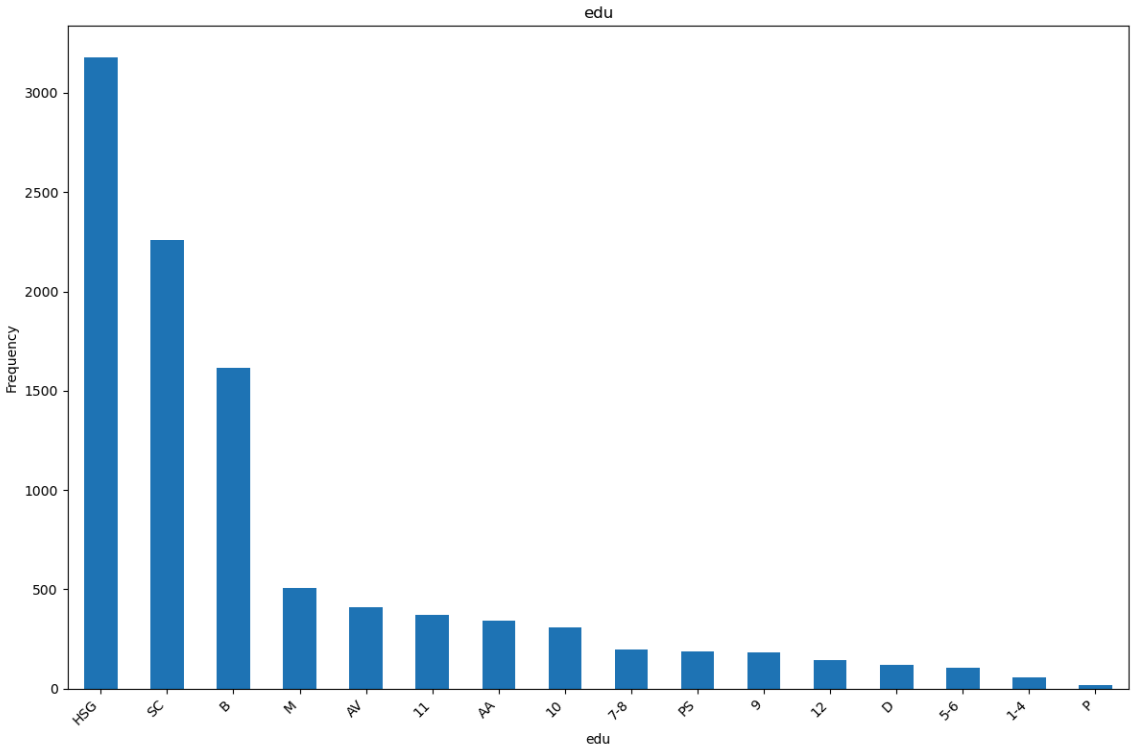
Observam ca plajele sunt foarte diferite si ca avem posibili outlieri (val indepartate de medie).

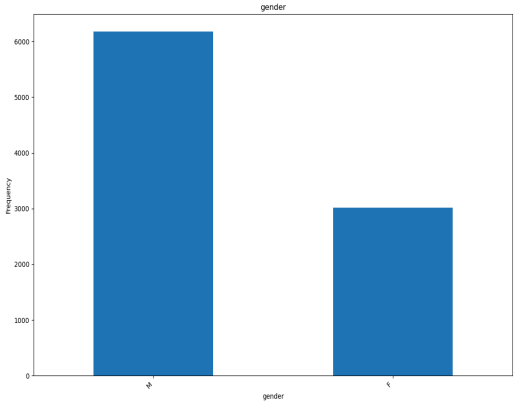
Valori tip categorii

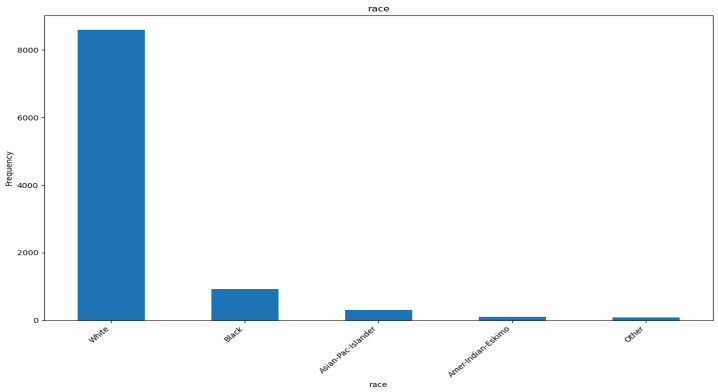
	relation	country	job	work_type	partner	edu	gender	race	gtype	money
count	9999	9841	9417	9419	9999	9999	9199	9999	9999	9999
unique	6	40	13	8	7	16	2	5	2	2
top	H	United-States	Craft-repair	Priv	MCS	HSG	M	White	AC	<=50K
freq	4097	8978	1277	6940	4667	3178	6179	8588	6711	7591

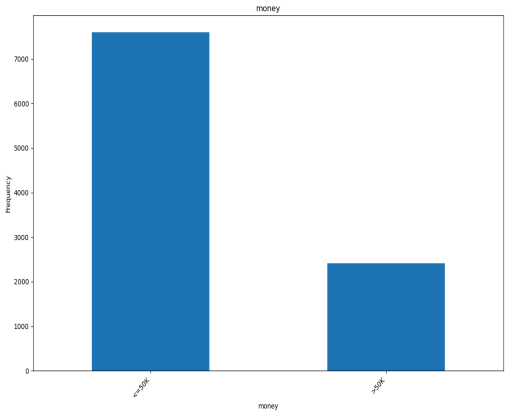


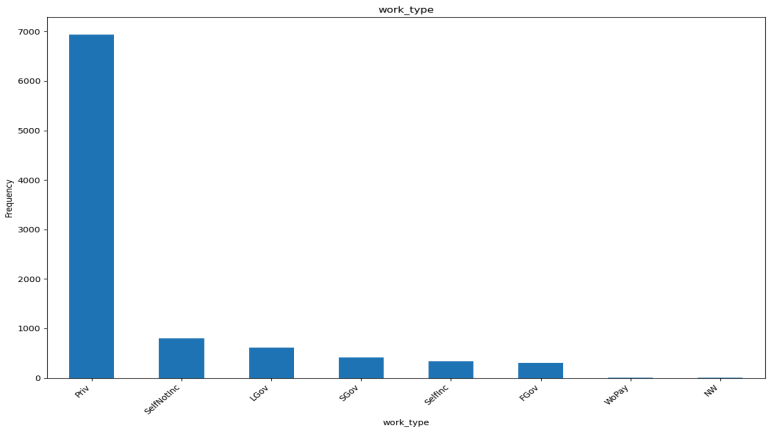

job
Frequency
0
200
400
600
800
1000
1200
Craft-repair
Exec-managerial
Prof-specialty
Sales
Adm-clerical
Other-service
Machine-op-inspct
Transport-moving
Handlers-cleaners
Farming-fishing
Tech-support
Protective-serv
Priv-house-serv
job

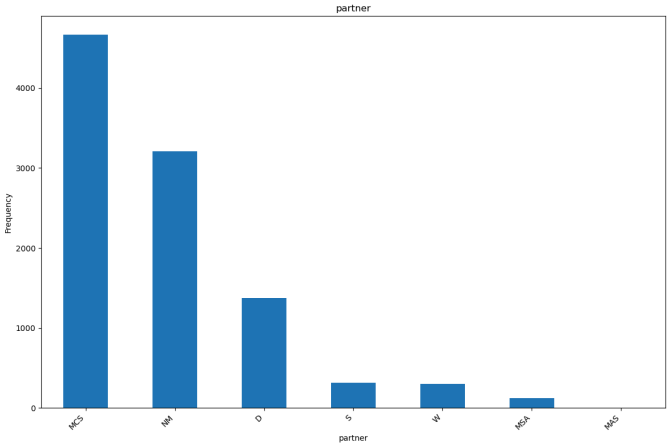

edu
Frequency
0
500
1000
1500
2000
2500
3000
HSG
SC
B
M
AV
11
AA
10
7-8
PS
9
12
D
5-6
1-4
P
edu

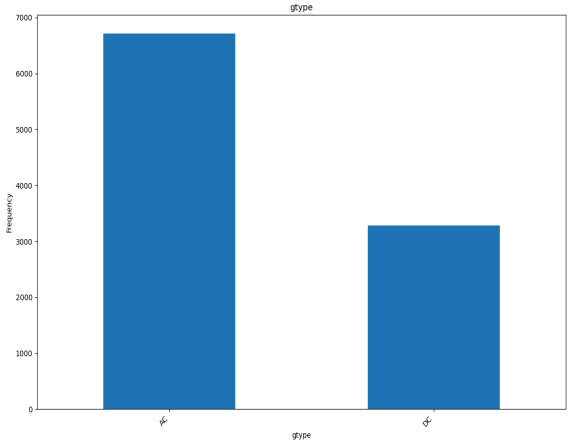

gender
Frequency
gender
M
F

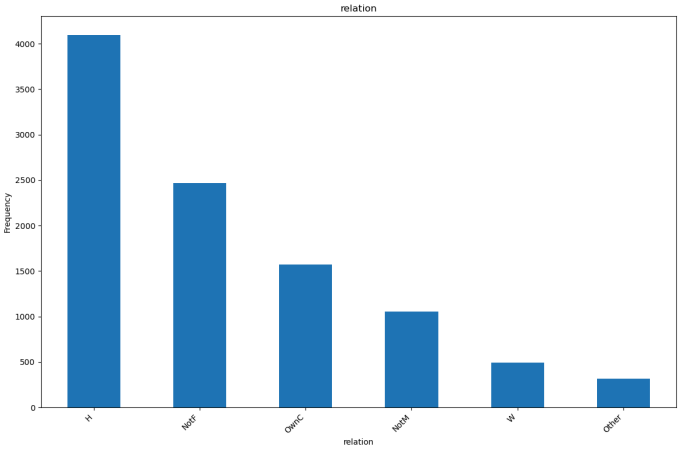

race
Frequency
race
White
Black
Asian-Pac-Islander
Amer-Indian-Eskimo
Other


money
Frequency
money
<=50k
>50k


work_type
Frequency
work_type
Priv
SelfNotInc
LGov
SGov
SelfInc
FGov
WoPay
NW


partner
Frequency
partner
MCS
NM
D
S
W
MSA
MAS

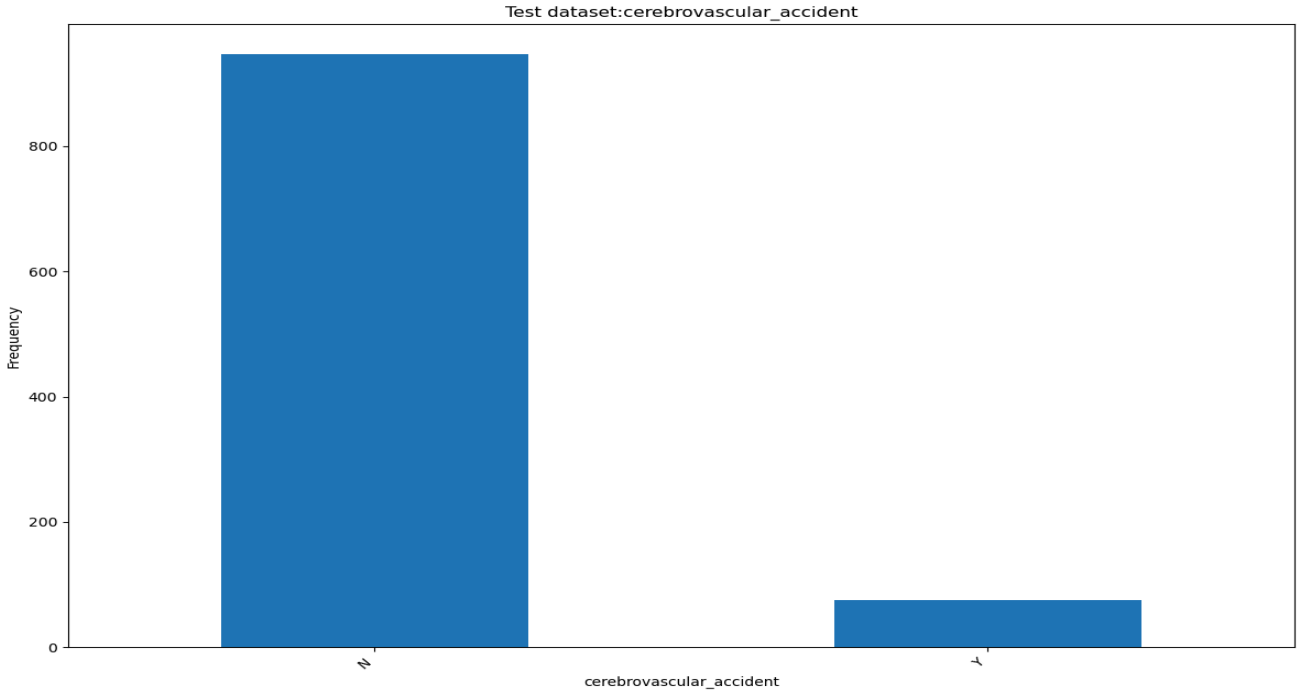

gtype
Frequency
gtype
AC
DC


relation
Frequency
relation
H
NotF
OwnC
NotM
W
Other

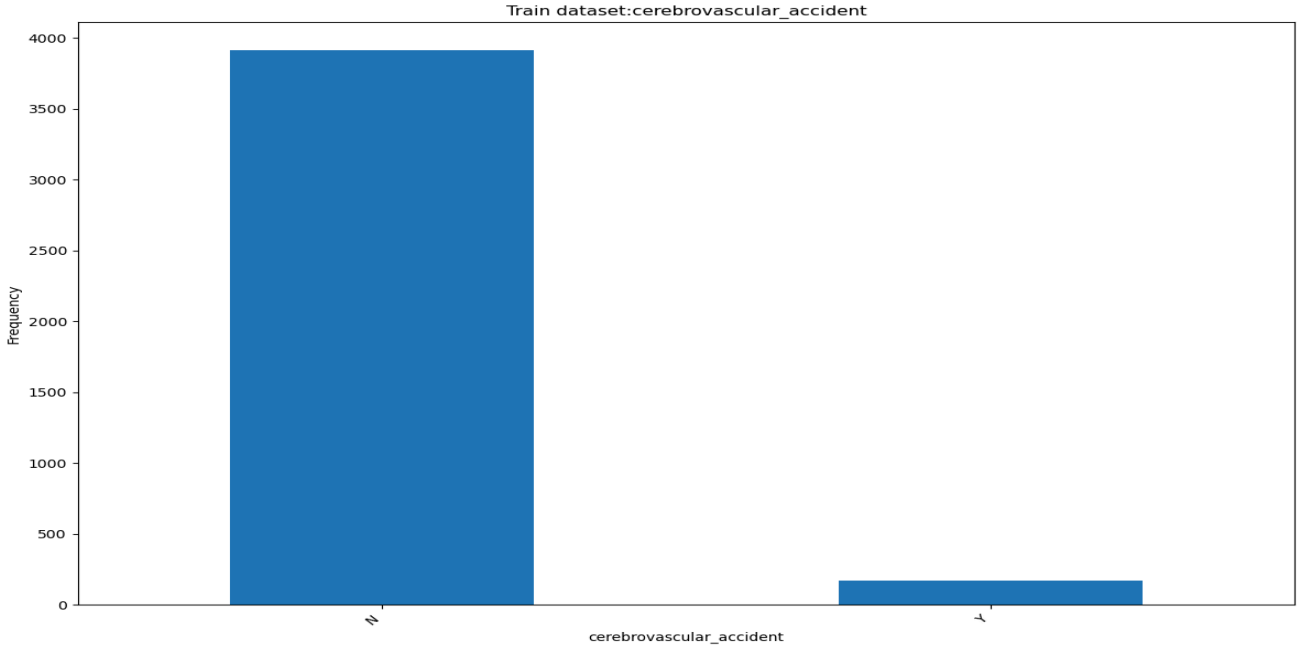
Pentru a obtine aceste date am analizat dataset-urile si am inlocuit valori care nu spuneau nimic despre categoria respectiva/ eronate (ex: tabaco_usage - not_defined, sau ‘?’, sau index biologic < 0) cu nan, pentru a le putea inputa mai tarziu.

1.2. Analiza echilibrului de clase

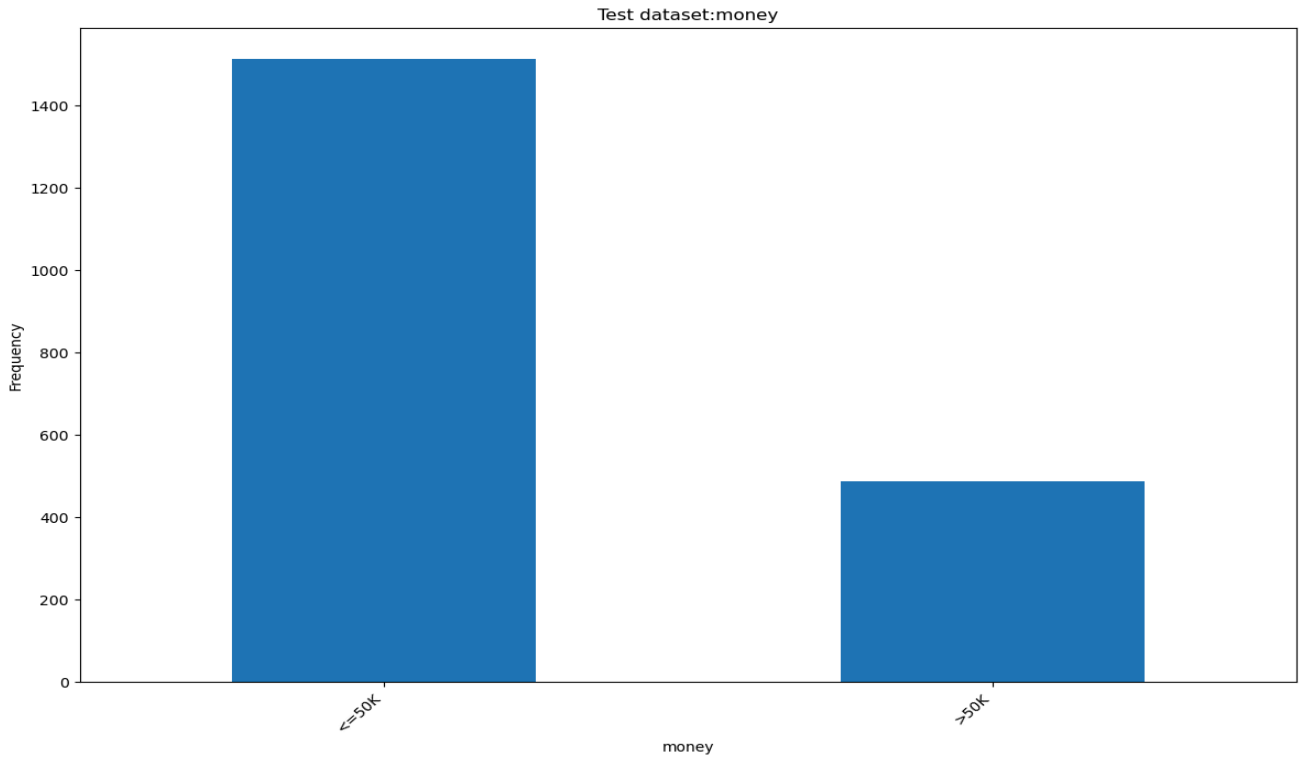
- ***Dataset-Avc***
  Test dataset:



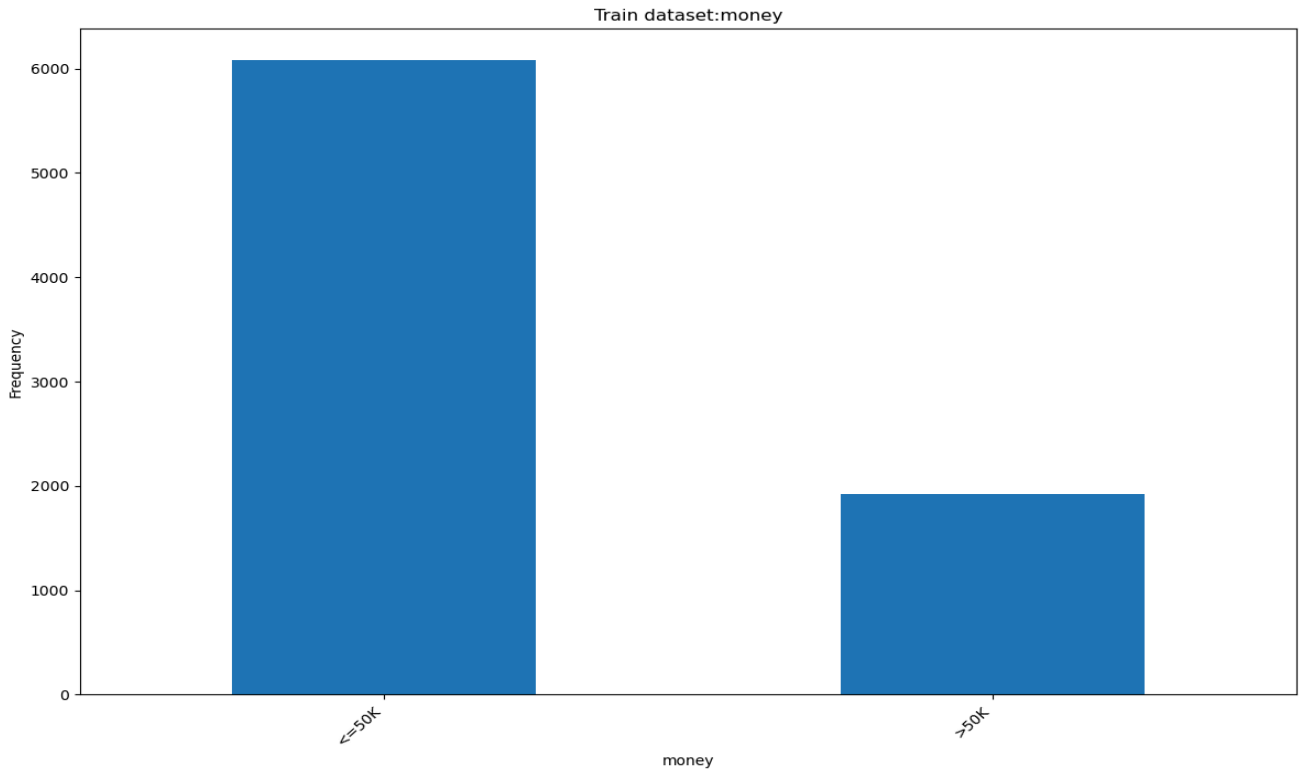
Train dataset:



- **Salary Prediction Dataset**
  Test dataset:



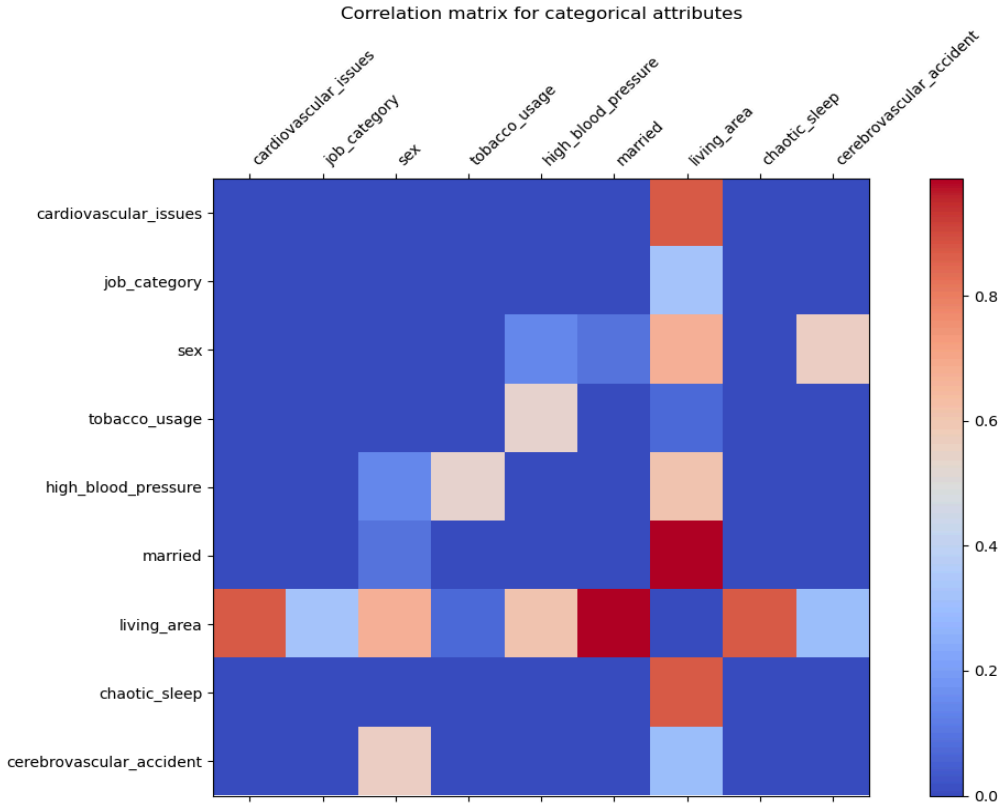
Train dataset:

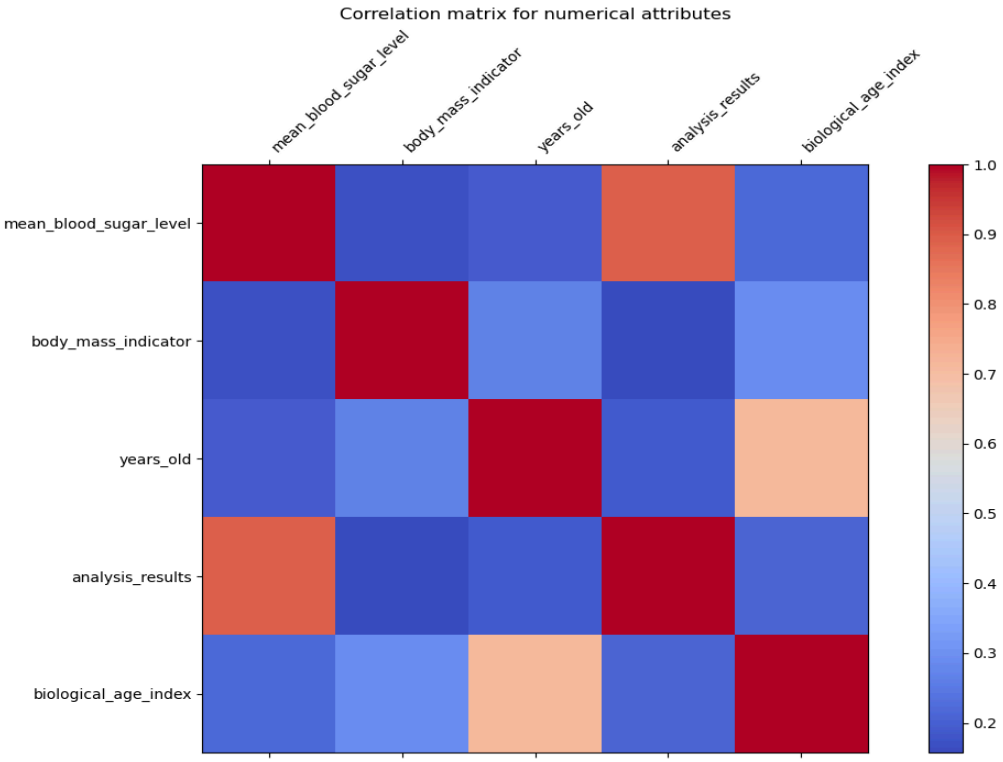


Observam ca avem un dezechilibru de clase in ambele dataseturi, astfel vom fi mai atenti atunci cand antrenam modelul la weight-ul oferit claselor si ne vom ghida mai mutl dupa metricile precision, recall, F1.

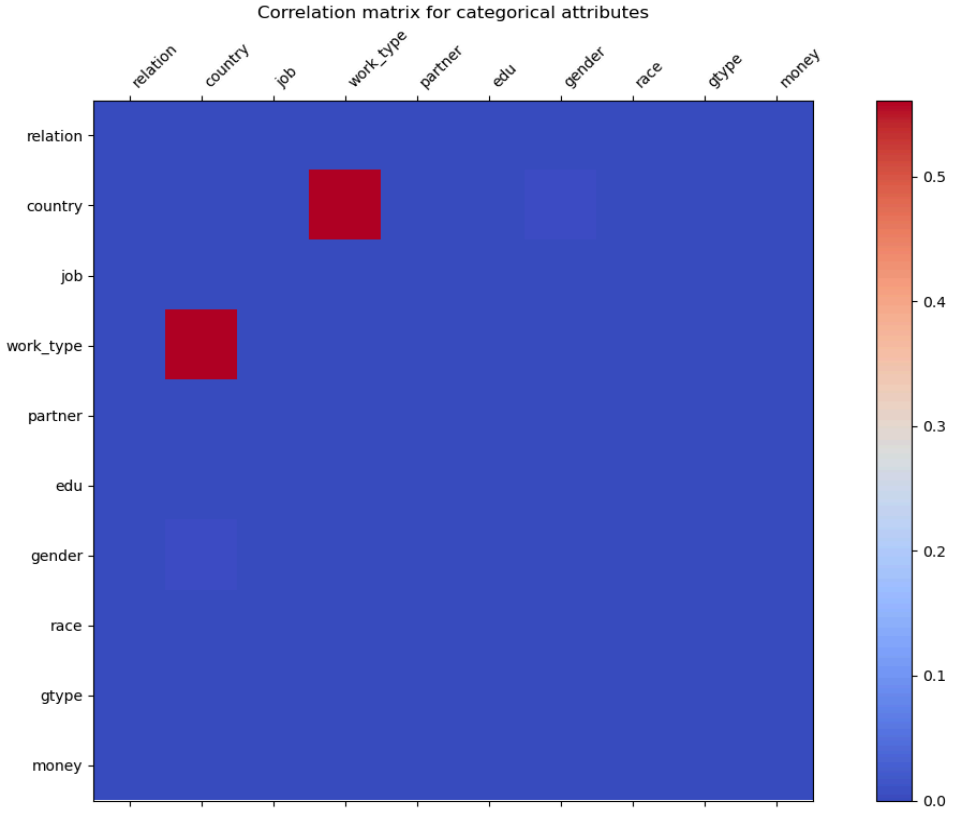
1.3. Analiza corelației între atribute

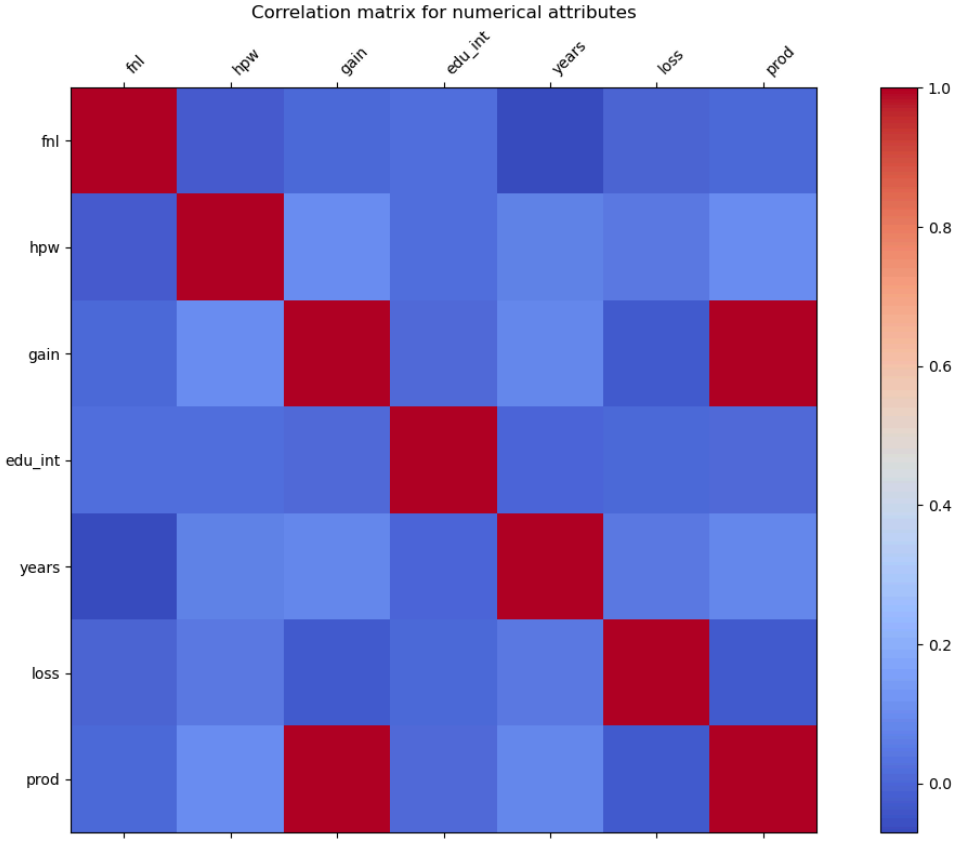
- **AVC Dataset**





- **Salary Prediction Dataset**





Observam ca avem cateva perechi de clase puternic corelate, dupa ce le-am identificat urmeaza sa eliminam cate una din fiecare pereche de atribute numarice cu o corelare de cel putin 0.7.

2. Preprocesarea datelor

2.1. Valori extreme pentru un atribut într-un eșantion

Cautam si eliminam valorile extreme prin metoda diferenței inter-cuartilă . Am ales Q1 = 0.25 si Q3 = 0.75 si dupa aplicarea algoritmului vom inlocui datele gasite cu nan in dataset pentru a le imputa mai tarziu cu date mai relevante.

2.2. Date lipsă pentru un atribut într-un eșantion

Dupa ce am eliminat variabilele redundante si cele extreme si le-am inlocuit cu nan, acum putem imputa date relevante cu ajutorul functiilor SimpleImputer(pentru date tip categorii) si IterativeImputer(pentru date numerice). Am ales sa imputez cea mai frecventa valoare pt atributele de tip categorie si o metoda iterativa bazata pe valoarea medie la atributele numerice.

2.3.. Atribute redundante (puternic corelate)

Dupa analiza corelatiei intre atributele numerice am ales sa elimin un atribut din fiecare perche cu o corelare puternica (>0.7), astfel am eliminat din datasetul AVC atributele ***analysis_results*** (atribute puternic corelate: mean_blood_sugar_level si analysis_results) si ***biological_age_index*** (atribute puternic corelate: years_old si biological_age_index), iar din datasetul Salary Prediction atributul ***prod (***atribute puternic corelate: gain si prod***).***

2.4. Plaje valorice de mărimi diferite pentru atributele numerice

Am ales sa standardizez datele folosind un StandardScaler() din biblioteca sklearn, astfel am adus toate datele la o o plaja de marimi apropiata ca sa poata fi interpretate de cate algoritmii de ML.

3 Utilizarea algoritmilor de Învățare Automată

Din pacate timpul nu mi-a permis sa implementez algoritmii manual asa ca am folosit blioteca scikit-learn pentru a antrena si testa un model de regresie logistica si unul de MLP.

Pentru a putea aplica regresia logistică/MLP , variabilele categoriale trebuie convertite într-o formă numerică. Am utilizat in acest scop clase disponibile in biblioteca scikit-learn: LabelEncoder sau OneHotEncoder.Am folosit LabelEncoder pentru a encoda variabila țintă și

OneHotEncoder pentru a encoda variabilele predictor de tip categoric cu ajutorul functiei pandas.get_dummies.

3.1 Regresie Logistica

- **Tipul de encodare folosit pentru fiecare atribut categoric:**
  - **LabelEncoder** pt variabila tinta si **OneHotEncoder** pt celelalte atribute de tip categoric

- **Setările algoritmului de optimizare de tip gradient descent folosit:**
  - În codul meu **am folosit solverul 'liblinear', care este un solver optimizat** pentru probleme de regresie logistică cu regularizare L1 sau L2. Acesta utilizează un algoritm iterativ bazat pe gradient descent.

- **Metoda de regularizare folosită:**
  - Am ales sa folosesc **regularizarea L1** dupa cateva incercari experimentale. Această regularizare ajută la prevenirea overfitting-ului prin penalizarea coeficienților modelului care sunt prea mari, forțându-i să fie zero în anumite situații. De asemenea, **am definit class weights pentru a trata dezechilibrul claselor**, acordând o greutate mai mare clasei minore adica cea in care AVC este 1 si Salriul >50k pentru a compensa subrepresentarea acesteia în setul de date.

3.2 Multi-Layered Perceptron (MLP)

- **Arhitectura rețelei neurale:**
  - Numărul de straturi: Modelul folosește **2** straturi ascunse
  - Dimensiunea fiecărui strat: Primul strat ascuns are 50 de neuroni, iar al doilea strat ascuns are, de asemenea, 50 de neuroni
  - Tipul de funcții de activare folosite: Funcția de activare pentru toate straturile este tangenta hiperbolică (tanh**),** specificată de parametrul activation='tanh'**.**
- **Configurarea optimizatorului:**
  - Tip de optimizator folosit: Solver-ul lbfgs este utilizat pentru optimizare. Acesta este un optimizator în familia metodelor cu gradient descent.
  - Learning rate: În acest caz, se folosește learning_rate='invscaling', care reduce treptat rata de învățare la fiecare pas de timp.
  - Numărul de epoci de antrenare: Modelul are un număr maxim de 300 de epoci de antrenare, specificat de parametrul max_iter=300.
  - Dimensiunea batch-urilor de antrenare: Modelul nu folosește batch-uri de antrenare, deoarece solver-ul lbfgs nu utilizează mini-batch-uri
- **Metode de regularizare folosite:**

- Utilizare early stopping: Parametrul early_stopping este setat pe False, ceea ce înseamnă că antrenarea nu se va opri prematur dacă scorul de validare nu îmbunătățește.
- Coeficient de regularizare L2 pe ponderi: Acest lucru nu este specificat direct în cod., dar solver-ul lbfgs are incorporată o formă de regularizare L2.

4. Evaluarea algoritmilor:

4.1 Descriere seturi de hiperparametrii finale pe care le-am folosit pentru fiecare algoritm:

```
clf = LogisticRegression(random_state=0, class_weight=class_weights,
solver='liblinear', penalty='l1').fit(X_train, y_train)
```

- random_state=0: Asigură reproducibilitatea rezultatelor prin fixarea seed-ului generatorului de numere aleatorii
- class_weight=class_weights: Folosește ponderi de clasă pentru a gestiona eventualele dezechilibre de clasă în date. Acest lucru ajută la îmbunătățirea performanței modelului pe clasele minoritare.
- solver='liblinear': Algoritmul de optimizare liblinear este eficient pentru seturi de date mici până la medii și suportă regularizarea L1.
- penalty='l1': Folosește regularizarea L1 pentru a induce sparsitate în coeficienții modelului. Aceasta poate duce la un model mai interpretabil, deoarece poate elimina unele dintre caracteristicile mai puțin relevante.

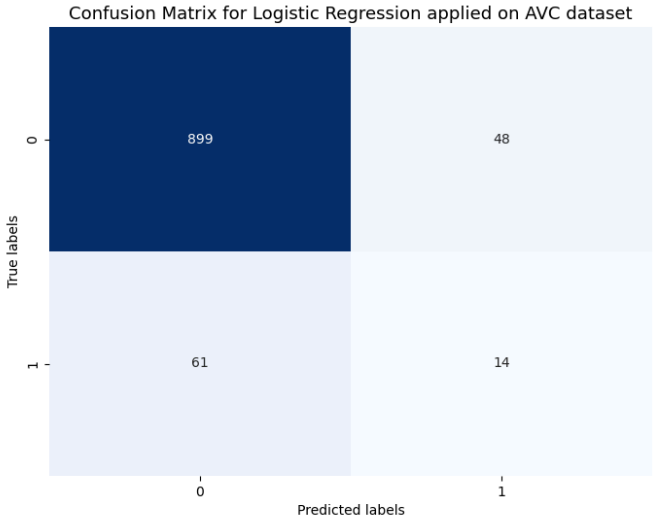
Hiperparametrii pentru LogisticRegression sunt aleși pentru a gestiona dezechilibrele de clasă și pentru a obține un model interpretabil și performant. Regularizarea L1 și ponderile de clasă contribuie la un model robust care poate face față datelor neechilibrate.

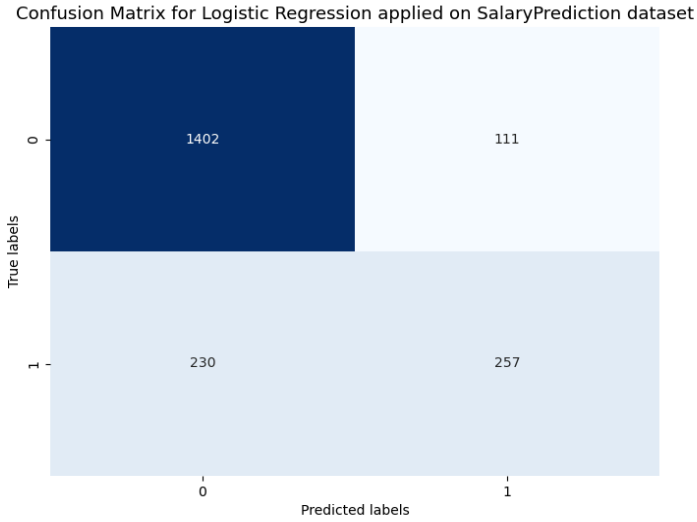
```
clf = MLPClassifier(max_iter=1000, random_state=0, hidden_layer_sizes=(50,
50), activation='tanh', solver='lbfgs', alpha=0.01,early_stopping=False,
validation_fraction=0.2, learning_rate='invscaling',
learning_rate_init=0.001).fit(X_train, y_train)
```

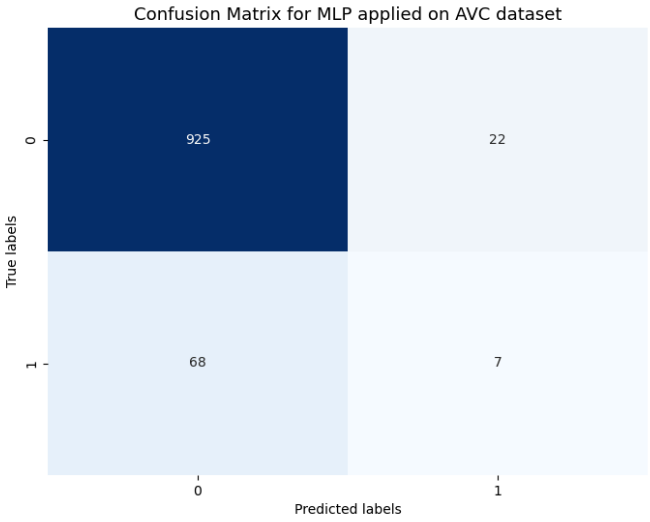
- max_iter=1000: Numărul maxim de iterații pentru antrenare. Este setat la o valoare relativ mare pentru a asigura convergența rețelei neuronale.
- random_state=0: Asigură reproducibilitatea rezultatelor prin fixarea seed-ului generatorului de numere aleatorii.
- hidden_layer_sizes=(50, 50): Rețeaua neuronală are două straturi ascunse, fiecare cu 50 de neuroni. Aceasta configurare permite modelului să învețe reprezentări complexe ale datelor.
- activation='tanh': Funcția de activare tangenta hiperbolică este folosită pentru a introduce non-linearități în model, permițând rețelei să învețe relații complexe
- solver='lbfgs': Algoritmul de optimizare L-BFGS este ales pentru convergența rapidă și performanța bună pe seturi de date mici până la medii.
- alpha=0.01: Termenul de regularizare L2 (penalizare) pentru a preveni supraînvățarea. Valoarea 0.01 este o alegere comună pentru a echilibra între învățarea modelului și prevenirea supraînvățării.
- early_stopping=False: Dezactivează oprirea timpurie. Modelul va antrena pentru toate iterațiile specificate, ceea ce poate fi util pentru explorarea completă a datelor.
- validation_fraction=0.2: Proporția datelor de antrenament folosite pentru validare. Este utilă pentru evaluarea performanței modelului în timpul antrenării.
- learning_rate='invscaling': Rata de învățare se va scădea în mod invers proporțional cu numărul de iterații, ceea ce poate ajuta la stabilizarea învățării pe măsură ce antrenarea progresează.
- learning_rate_init=0.001: Rata de învățare inițială. O valoare mică ajută la asigurarea unei învățări stabile și a unei convergențe mai precise.

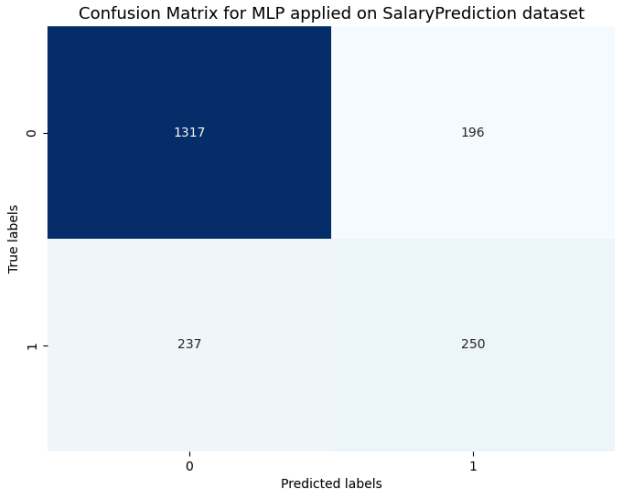
Mentionez ca a obtinut hiperparametrii cu ajutorul functiei **GridSearch**, pe un grid ce continea diversi parametrii care mi s-au parut relevanti aplicati in toate combinatiile pentru a obtine cele mai bune rezultate (in functie de scorul f1).

4.2. Matricile de confuzie pentru fiecare algoitm si dataset:









4.3. Tabele comparative al algoritmilor pe fiecare set de date:

- **AVC Dataset:**

	Accuracy	Precision	Recall	F1 score
Logistic Regresion	0.89335	0.22581	0.18667	**0.20438**
MLP	**0.91194**	**0.24138**	**0.09332**	0.13462

- **Salary Prediction Dataset:**

	Accuracy	Precision	Recall	F1 score
Logistic Regresion	**0.8295**	**0.69837**	**0.52772**	**0.60117**
MLP	0.7835	0.56054	0.51335	0.53591

Analiză AVC Dataset

1. Acuratețe:
    - MLP are o acuratețe mai mare decât Logistic Regression, ceea ce indică o capacitate mai bună de a clasifica corect instanțele din setul de date.
2. Precizie și Recall:
    - Deși MLP are o precizie puțin mai mare decât Logistic Regression, recall-ul este mult mai mic. Acest lucru indică faptul că MLP are o tendință de a rata multe instanțe pozitive (false negatives).
3. F1 Score:
    - F1 Score pentru Logistic Regression este mai mare decât pentru MLP, sugerând un echilibru mai bun între precizie și recall în cazul Logistic Regression.

Analiză Salary Prediction Dataset

1. Acuratețe:
    - Logistic Regression are o acuratețe mai mare decât MLP, ceea ce indică o performanță generală mai bună în clasificarea corectă a instanțelor.
2. Precizie și Recall:
    - Logistic Regression are o precizie și un recall mai mari decât MLP, indicând că modelul Logistic Regression este mai bun la identificarea corectă a instanțelor pozitive și la minimizarea falselor negative.
3. F1 Score:
    - F1 Score pentru Logistic Regression este mai mare decât pentru MLP, indicând un echilibru mai bun între precizie și recall în cazul Logistic Regression.

Concluzie generală

Pentru ambele seturi de date, Logistic Regression pare să ofere un echilibru mai bun între precizie și recall, reflectat în F1 Score-ul mai mare.

De ce Logistic Regression obține performanțe mai bune?

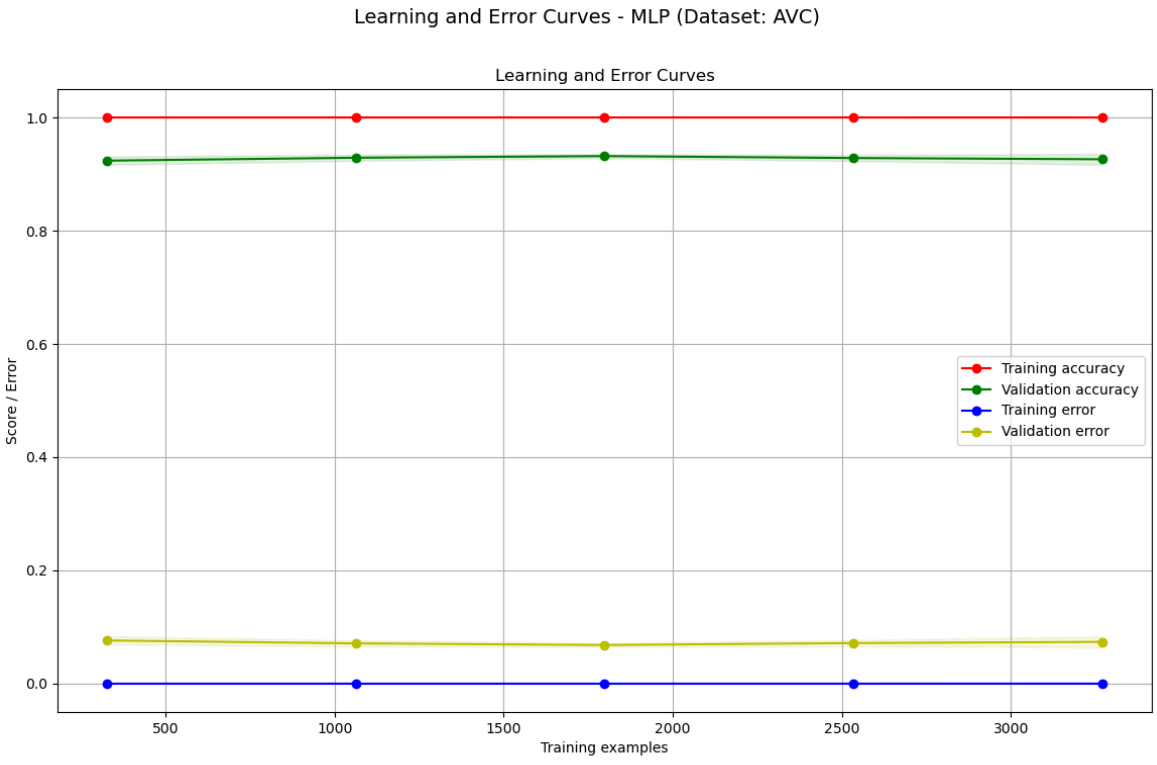
1. Complexitate Model:
    - Logistic Regression este un model liniar, ceea ce poate fi suficient pentru datele analizate. Modelul simplu reduce riscul de overfitting, ceea ce poate explica performanțele mai bune în termeni de F1 Score.
2. Generalizare:

- Fiind un model mai simplu, Logistic Regression poate generaliza mai bine pe date noi, ceea ce duce la o performanță mai bună pe setul de test.

De ce MLP nu performează la fel de bine?

1. Overfitting:
   - MLP are o structură mai complexă și poate fi predispus la overfitting, în special dacă nu sunt utilizate tehnici de regularizare eficiente sau dacă setul de date nu este suficient de mare pentru a susține un model complex.
2. Optimizare:
   - Optimizarea hiperparametrilor pentru MLP poate fi mai dificilă și sensibilă la inițializările aleatoare, ceea ce poate duce la variații în performanță.

4.4. Graficele curbelor de eroare și de performanță (acuratețea) pentru seturile de date de antrenare și test:



Learning and Error Curves - MLP (Dataset: SalaryPrediction)