Cozma Laura-Elena
Grupa 405

Big Data

Proiect – *Travel Insurance Claim Prediction*

În cadrul proiectului s-a utilizat setul de date *Travel Insurance*. Proiectul presupune analizarea cazurilor in care despagubirile cerute in urma asigurarii de calatorie au fost acordate sau nu. Predictiile vor fi calculate pe baza setului de date, ce are descrierea urmatoare:

- **Claim**, coloana target, cu valorile *Yes* sau *No*, dacă despagubirea a fost acordata sau nu a fost acordata
- **Agency**, numele agentiei alese pentru a realiza asigurarea
- **Agency Type** tipul agentiei la care s-a facut asigurarea de calatorie
- **Distribution Channel** modalitatea prin care a fost achizitionata asigurarea, *Online* sau *Offline*
- **Product Name** numele produsului solicitat
- **Duration** durata calatoriei
- **Destination** destinatia calatoriei
- **Net Sales** Valoarea vânzărilor de polițe de asigurare de călătorie
- **Commission** comisionul perceput de agentie in urma asigurarii
- **Gender** genul persoanei care a solicitat asigurarea
- **Age** varsta persoanei care a solicitat asigurarea

Explorarea datelor

Pentru inceput, vom afisa schema impreuna cu primele 5 randuri de date, pentru a putea observa mai bine structura.

```
data.printSchema()
root
 |-- Agency: string (nullable = true)
 |-- Agency Type: string (nullable = true)
 |-- Distribution Channel: string (nullable = true)
 |-- Product Name: string (nullable = true)
 |-- Claim: string (nullable = true)
 |-- Duration: integer (nullable = true)
 |-- Destination: string (nullable = true)
 |-- Net Sales: double (nullable = true)
 |-- Commision (in value): double (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Age: integer (nullable = true)
```

Se observa faptul ca schema a fost dedusa corect, insa dorim sa schimbam denumirile coloanelor pentru usurinta utilizarii lor in viitor.

```
data = data.withColumnRenamed("Agency Type","Agency_Type") \
           .withColumnRenamed("Distribution Channel",
"Distribution_Channel") \
           .withColumnRenamed("Product Name", "Product_Name") \
           .withColumnRenamed("Net Sales", "Net_Sales") \
           .withColumnRenamed("Commision (in value)", "Commission")
```

Afisarea primelor 5 randuri:

```
pd.DataFrame(data.take(10), columns = data.columns).transpose()
```

	0	1	2	3	4	5
Agency	CBH	CBH	CWT	CWT	CWT	JZI
Agency Type	Travel Agency	Travel Agency	Travel Agency	Travel Agency	Travel Agency	Airlines
Distribution Channel	Offline	Offline	Online	Online	Online	Online
Product Name	Comprehensive Plan	Comprehensive Plan	Rental Vehicle Excess Insurance	Rental Vehicle Excess Insurance	Rental Vehicle Excess Insurance	Value Plan
Claim	No	No	No	No	No	No
Duration	186	186	65	60	79	66
Destination	MALAYSIA	MALAYSIA	AUSTRALIA	AUSTRALIA	ITALY	UNITED STATES
Net Sales	-29.0	-29.0	-49.5	-39.6	-19.8	-121.0
Commision (in value)	9.57	9.57	29.7	23.76	11.88	42.35
Gender	F	F	None	None	None	F
Age	81	71	32	32	41	44

In ceea ce priveste dimensiunea datelor, setul de date este format din 11 coloane si 63326 de linii, aflate cu ajutorul liniei de cod `(data.count(), len(data.columns))`.

Verificarea valorilor nule

```
import pyspark.sql.functions as f

# Valorile null din fiecare coloană
data_agg = data.agg(*[f.count(f.when(f.isnull(c), c)).alias(c) for c in
data.columns])
data_agg.show()
```

Sectiunea de cod a avut ca rezultat urmatorul output:

Agency	Agency_Type	Distribution_Channel	Product_Name	Claim	Duration	Destination	Net_Sales	Commission	Gender	Age
0	0	0	0	0	0	0	0	0	45107	0

Se observa ca singurele valori nule sunt prezente in cadrul coloanei *Gender*, fiind 45107 de valori nule din 63326 de inregistrari. Datorita procentului mare pe care acestea il ocupa, cat si faptului ca genul unei persoane nu ar trebui sa influenteze rezultatele finale, vom elimina coloana in cauza, cu ajutorul metodei *drop*:

```
data = data.drop('Gender')
```

Valorile distincte pentru fiecare coloana

Acestea ne vor spune sub ce forma se afla datele noastre:

```
from pyspark.sql.functions import col, countDistinct
data.agg(*(countDistinct(col(c)).alias(c) for c in
data.columns)).show()
```

Agency	Agency_Type	Distribution_Channel	Product_Name	Claim	Duration	Destination	Net_Sales	Commission	Age
16	2	2	26	2	455	149	1139	1035	89

Se observa ca valorile de tip string tind sa aiba mai putine valori distincte, acestea putand fi impartite pe categorii, in timp ce la valorile numerice, in special pentru Net_Sales si Commission, valorile sunt continue.

Descrierea coloanelor numerice

Afisam descrierea coloanelor numerice din setul de date cu ajutorul metodei *describe*:

```
data.select('Duration', 'Net_Sales', 'Commission', 'Age') \
    .describe().show()
```

summary	Duration	Net_Sales	Commission	Age
count	63326	63326	63326	63326
mean	49.31707355588542	40.702017970502204	9.809991788523527	39.969980734611376
stddev	101.79156617721215	48.84563729289582	19.80438849937355	14.017009538046246
min	-2	-389.0	0.0	0
max	4881	810.0	283.5	118

Cozma Laura-Elena
Grupa 405

Majoritatea coloanelor au o deviatie standard mare, pe primul loc aflandu-se *Duration*. De asemenea, analizand valorile min si max, se observa ca exista o serie de valori incorecte pentru duration (valori negative), astfel ca acestea vor fi eliminate la pasii urmatori.

Analizarea datelor din fiecare coloana

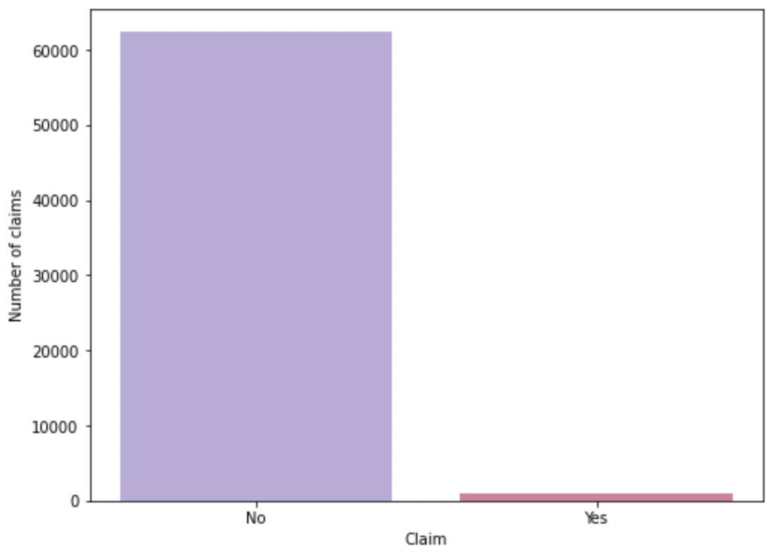
Inainte de a incepe analiza, vom crea un tabel temporar din care vom extrage date cu ajutorul Spark SQL:

```
data.createOrReplaceTempView("travel_insurance")
```

Claim

```
spark.sql("SELECT Claim, COUNT(*) as Number_of_claims FROM travel_insurance GROUP BY Claim").show()

data_plot = data.toPandas()
claim = data_plot["Claim"].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(claim.index.astype('str'), claim, color=['#BAABDA', '#D77FA1'])
plt.xlabel("Claim")
plt.ylabel("Number of claims")
plt.show()
```

```
+-----+----------------+
|Claim|Number_of_claims|
+-----+----------------+
|   No|           62399|
|  Yes|             927|
+-----+----------------+
```



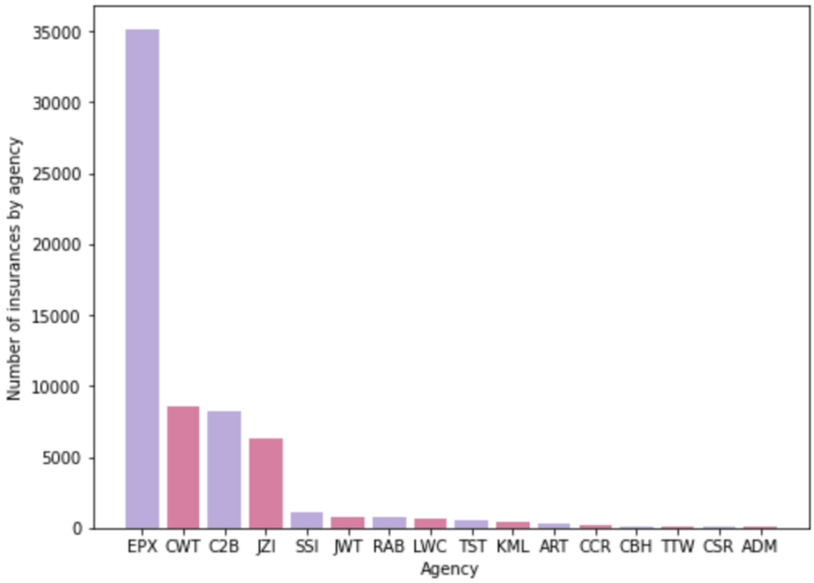
Cozma Laura-Elena
Grupa 405

Observam ca setul de date este *nebalansat*, avand mult mai multe inregistrari in cazul negativ *No*, fata de *Yes*.

Agency

```
spark.sql("SELECT Agency, COUNT(*) as Number_of_insurances FROM
travel_insurance GROUP BY Agency SORT BY Number_of_insurances
DESC").show()

agency = data_plot["Agency"].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(agency.index.astype('str'), agency, color=['#BAABDA',
'#D77FA1'])
plt.xlabel("Agency")
plt.ylabel("Number of insurances by agency")
plt.show()
```

```
+------+--------------------+
|Agency|Number_of_insurances|
+------+--------------------+
|   EPX|               35119|
|   CWT|                8580|
|   C2B|                8267|
|   JZI|                6329|
|   SSI|                1056|
|   JWT|                 749|
|   RAB|                 725|
|   LWC|                 689|
|   TST|                 528|
|   KML|                 392|
|   ART|                 331|
|   CCR|                 194|
|   CBH|                 101|
|   TTW|                  98|
|   CSR|                  86|
|   ADM|                  82|
+------+--------------------+
```



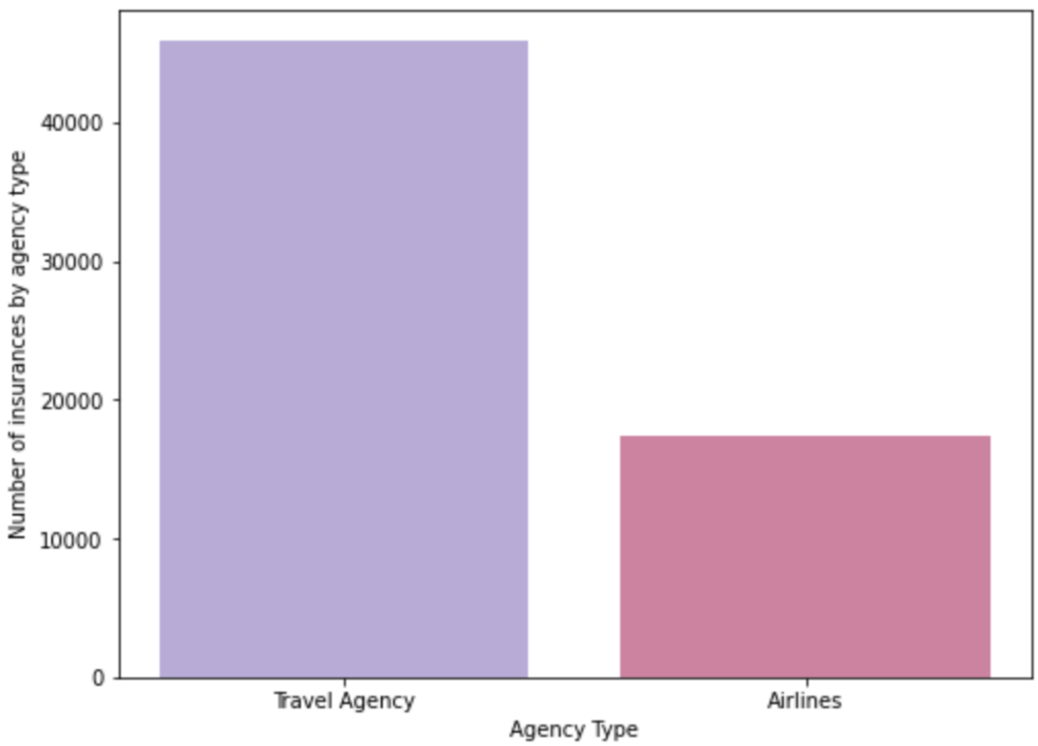
Cozma Laura-Elena
Grupa 405

Se poate observa ca exista o agentie care contine o buna parte din date (EPX) spre deosebire de proportiile urmatoare.

Agency type

```
spark.sql("SELECT Agency_Type, COUNT(*) as Number_of_insurances FROM
travel_insurance GROUP BY Agency_Type SORT BY Number_of_insurances
DESC").show()

agency_type = data_plot["Agency_Type"].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(agency_type.index.astype('str'), agency_type, color=['#BAABDA',
'#D77FA1'])
plt.xlabel("Agency Type")
plt.ylabel("Number of insurances by agency type")
plt.show()
```

```
+-------------+--------------------+
|  Agency_Type|Number_of_insurances|
+-------------+--------------------+
|Travel Agency|               45869|
|     Airlines|               17457|
+-------------+--------------------+
```

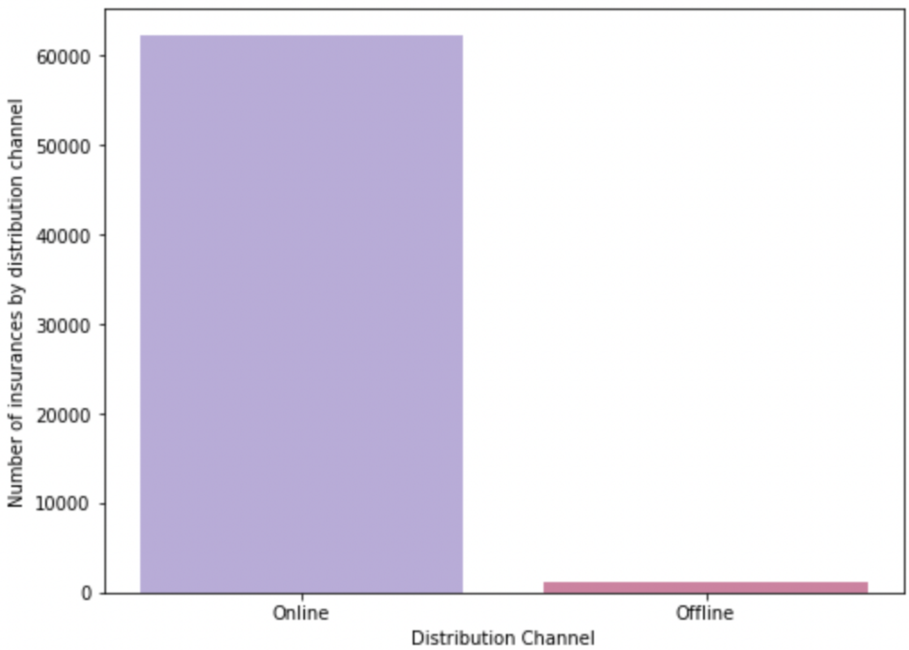


Spre deosebire de *Agency*, *Agency Type* are un numar mult mai mic de valori, cu 2 categorii, *Travel Agency* si *Airlines*. Categoria majoritara este *Travel Agency*, avand de doua ori mai multe inregistrari fara de *Airline*s.

Distribution Channel

```
spark.sql("SELECT Distribution_Channel, COUNT(*) as
Number_of_insurances FROM travel_insurance GROUP BY
Distribution_Channel SORT BY Number_of_insurances DESC").show()

distribution_channel = data_plot["Distribution_Channel"].value_counts()
plt.figure(figsize=(8, 6))
plt.bar(distribution_channel.index.astype('str'), distribution_channel,
color=['#BAABDA', '#D77FA1'])
plt.xlabel("Distribution Channel")
plt.ylabel("Number of insurances by distribution channel")
plt.show()
```

```
+--------------------+--------------------+
|Distribution_Channel|Number_of_insurances|
+--------------------+--------------------+
|              Online|               62219|
|             Offline|                1107|
+--------------------+--------------------+
```



Cozma Laura-Elena
Grupa 405

Exista doar doua canale de distributie, *Online* si *Offline*, prima categorie avand o pondere cu mult mai mare decat cea de-a doua.

Din tabelul ce contine numararea valorilor distincte pentru fiecare coloana, rezulta ca *Product Name* si *Destination* sunt categoriile care contin cele mai multe valori distincte - 26, respectiv 149 - astfel ca vom afisa doar un piechart cu primele 10 produse si destinatii selectate de persoanele asigurate.

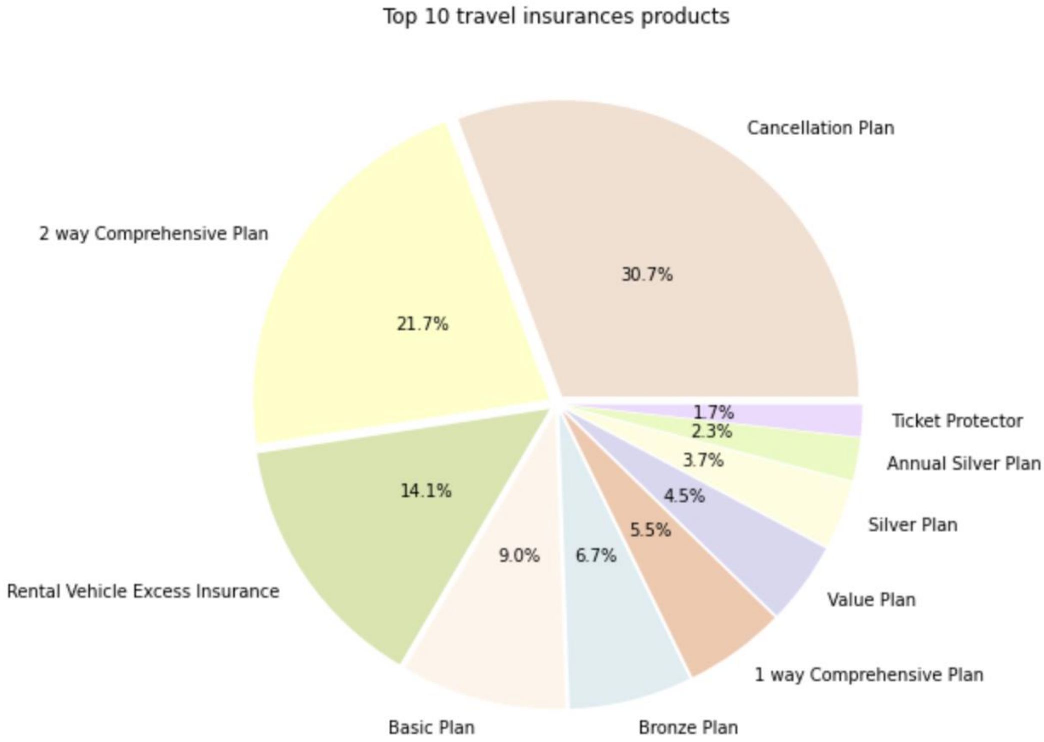
Product Name

```
product_name = spark.sql("SELECT Product_Name, COUNT(*) as
Number_of_insurances FROM travel_insurance GROUP BY Product_Name SORT
BY Number_of_insurances DESC")
product_name.show()
product_name = product_name.toPandas().head(10)
plt.figure(figsize=(10, 8))
labels = product_name['Product_Name']
plt.pie(x=product_name['Number_of_insurances'], autopct="%.1f%%",
explode=[0.03]*10, labels=labels, pctdistance=0.5, colors=['#F4DFD0',
'#FFFFC5', '#D6E4AA', '#FFF5EB', '#DEEDF0', '#F4C7AB', '#D9D7F1',
'#FFFDDE', '#E7FBBE', '#F0D9FF'])
plt.title('Top 10 travel insurances products')
plt.show()
```

```
+--------------------+--------------------+
|        Product_Name|Number_of_insurances|
+--------------------+--------------------+
|   Cancellation Plan|               18630|
|2 way Comprehensi...|               13158|
|Rental Vehicle Ex...|                8580|
|          Basic Plan|                5469|
|         Bronze Plan|                4049|
|1 way Comprehensi...|                3331|
|          Value Plan|                2715|
|         Silver Plan|                2249|
|  Annual Silver Plan|                1423|
|    Ticket Protector|                1056|
|Travel Cruise Pro...|                 527|
|  Comprehensive Plan|                 364|
|           Gold Plan|                 352|
|          24 Protect|                 247|
|Single Trip Trave...|                 204|
|        Premier Plan|                 194|
|    Annual Gold Plan|                 194|
|Single Trip Trave...|                 173|
|Annual Travel Pro...|                 100|
|Annual Travel Pro...|                  86|
+--------------------+--------------------+
only showing top 20 rows
```

Cozma Laura-Elena
Grupa 405

Se observa ca exista doua planuri care sunt achizitionate intr-o pondere mult mai mare fata de celelalte planuri: *Cancellation Plan* si *2 way Comprehensive Plan*, fiind urmate de *Rental Vehicle Excess Insurance.*

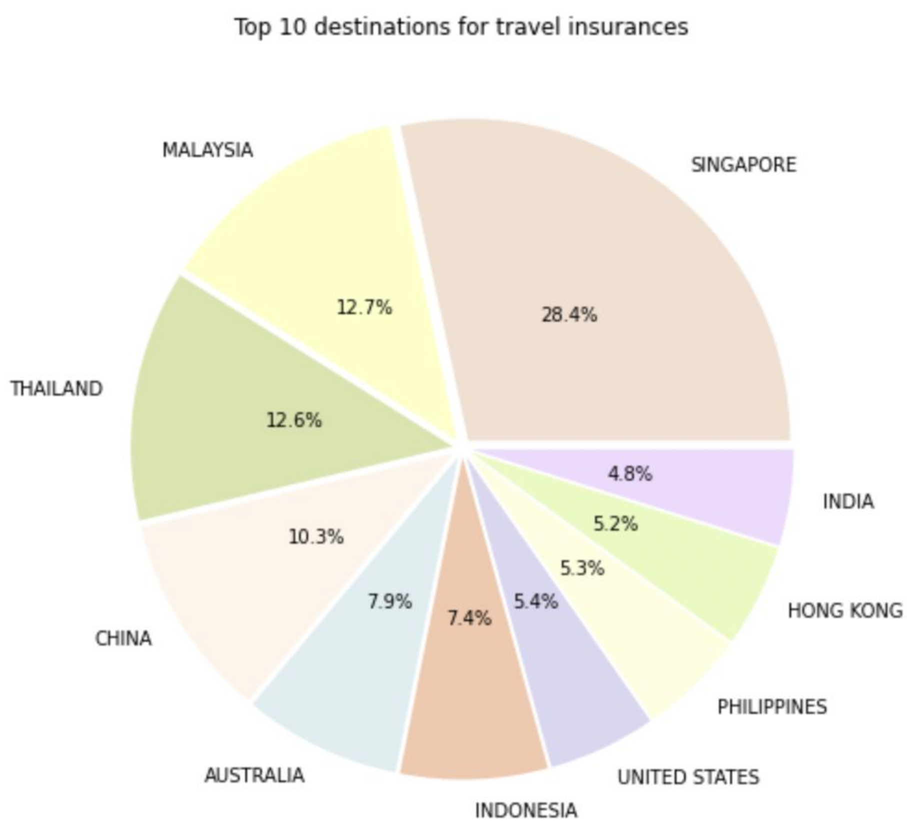


Destination

```
destination = spark.sql("SELECT Destination, COUNT(*) as
Number_of_insurances FROM travel_insurance GROUP BY Destination SORT BY
Number_of_insurances DESC")
destination.show()
destination = destination.toPandas().head(10)
plt.figure(figsize=(10, 8))
labels = destination['Destination']
plt.pie(x=destination['Number_of_insurances'], autopct="%.1f%%",
explode=[0.03]*10, labels=labels, pctdistance=0.5, colors=['#F4DFD0',
'#FFFFC5', '#D6E4AA', '#FFF5EB', '#DEEDF0', '#F4C7AB', '#D9D7F1',
'#FFFDDE', '#E7FBBE', '#F0D9FF'])
plt.title('Top 10 destinations for travel insurances')
plt.show()
```

Exista un numar foarte mare de destinatii alese, dar dintre acestea se remarca Singapore, care detine 28.4% din asigurarile facute. De asemenea, se observa ca majoritatea asigurarilor vizeaza zona Asiei.

```
+--------------------+--------------------+
|         Destination|Number_of_insurances|
+--------------------+--------------------+
|           SINGAPORE|               13255|
|            MALAYSIA|                5930|
|            THAILAND|                5894|
|               CHINA|                4796|
|           AUSTRALIA|                3694|
|           INDONESIA|                3452|
|       UNITED STATES|                2530|
|         PHILIPPINES|                2490|
|           HONG KONG|                2411|
|               INDIA|                2251|
|               JAPAN|                2061|
|            VIET NAM|                1669|
|  KOREA, REPUBLIC OF|                1479|
|      UNITED KINGDOM|                1309|
|TAIWAN, PROVINCE ...|                1090|
|             MYANMAR|                 806|
|   BRUNEI DARUSSALAM|                 780|
|         NEW ZEALAND|                 537|
|              CANADA|                 528|
|            CAMBODIA|                 493|
+--------------------+--------------------+
only showing top 20 rows
```

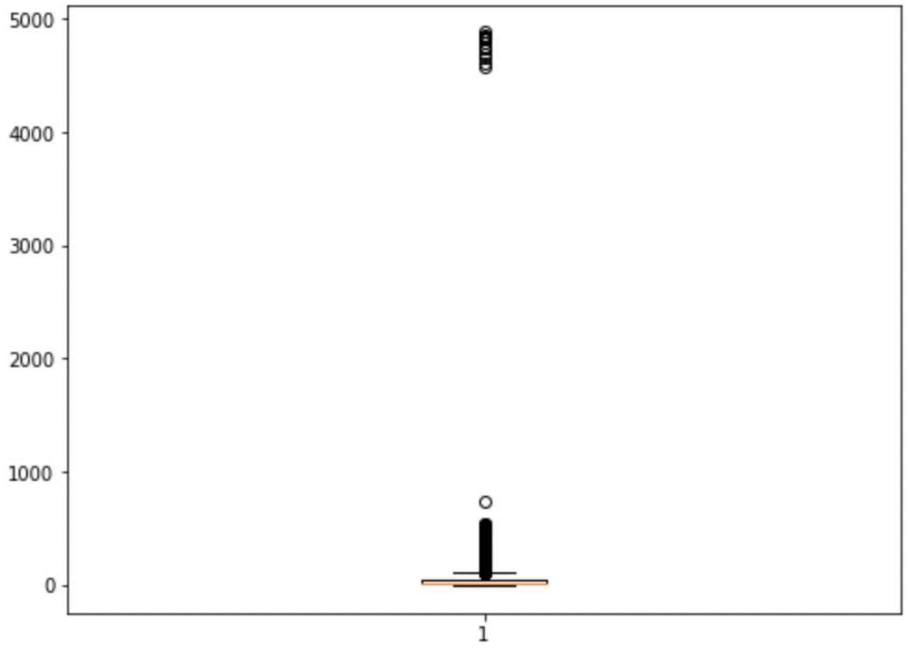


Cozma Laura-Elena
Grupa 405

Pentru analiza valorilor numerice, vom incepe prin construirea BoxPlot-urilor, pentru a vedea daca exista outliers.

Duration

```
plt.figure(figsize=(8, 6))
plt.boxplot(data_plot['Duration'])
plt.show()
```



Din Boxplot putem vedea ca exista doua categorii ale duratei, una sub 1000 de zile, dar exista si un grup distinct ce se afla peste valoarea de 4500. Se observa ca exista o serie de valori mai mici sau egale cu 0, pe care le vom elimina, deoarece durata nu poate fi negativa.

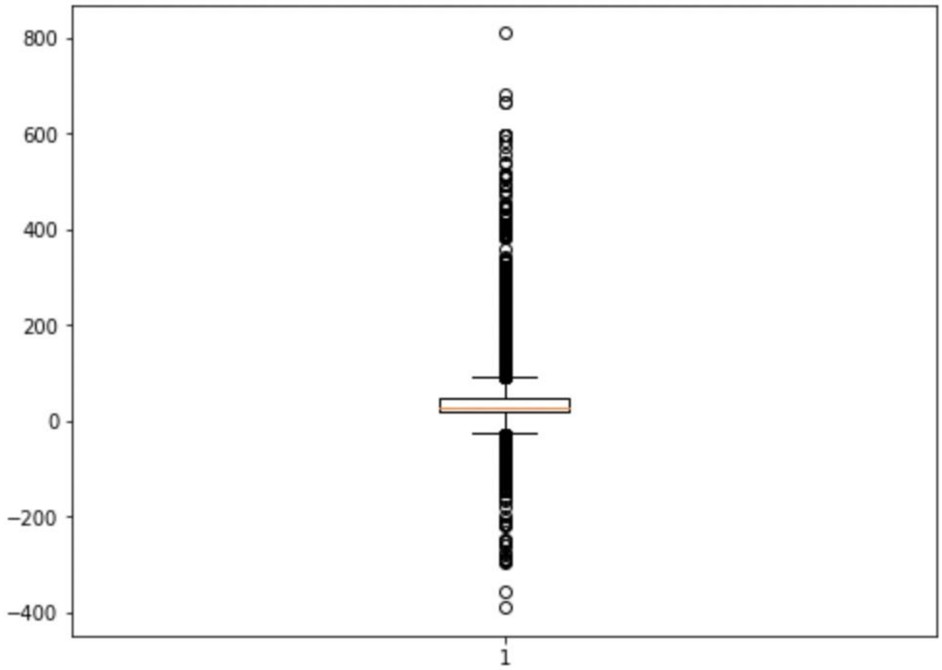
```
print("Old number of rows: " + str(data.count()))
data = data.where(data['Duration'] > 0)
print("New number of rows: " + str(data.count()))
```

```
Old number of rows: 63326
New number of rows: 63260
```

Net Sales

```
plt.figure(figsize=(8, 6))
plt.boxplot(data_plot['Net_Sales'])
plt.show()
```

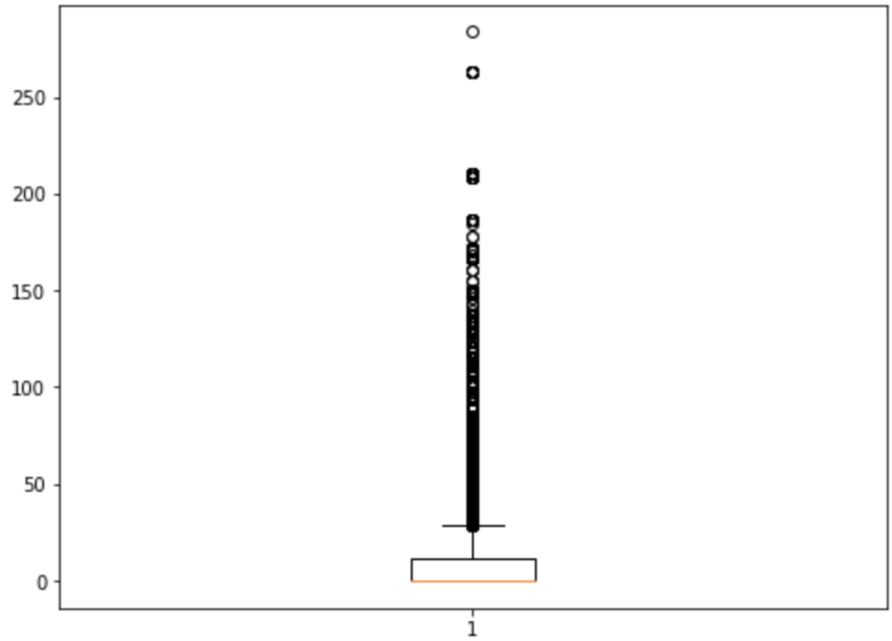
Cozma Laura-Elena
Grupa 405



Valorile Net Sales nu urmeaza niciun tipar, fiind distribuite intre -400 si 800.

Commission

```
plt.figure(figsize=(8, 6))
plt.boxplot(data_plot['Commission'])
plt.show()
```



Se observa ca exista o serie de outliers pentru valori ale comisionului mai mari de 250, astfel ca ii vom elimina:

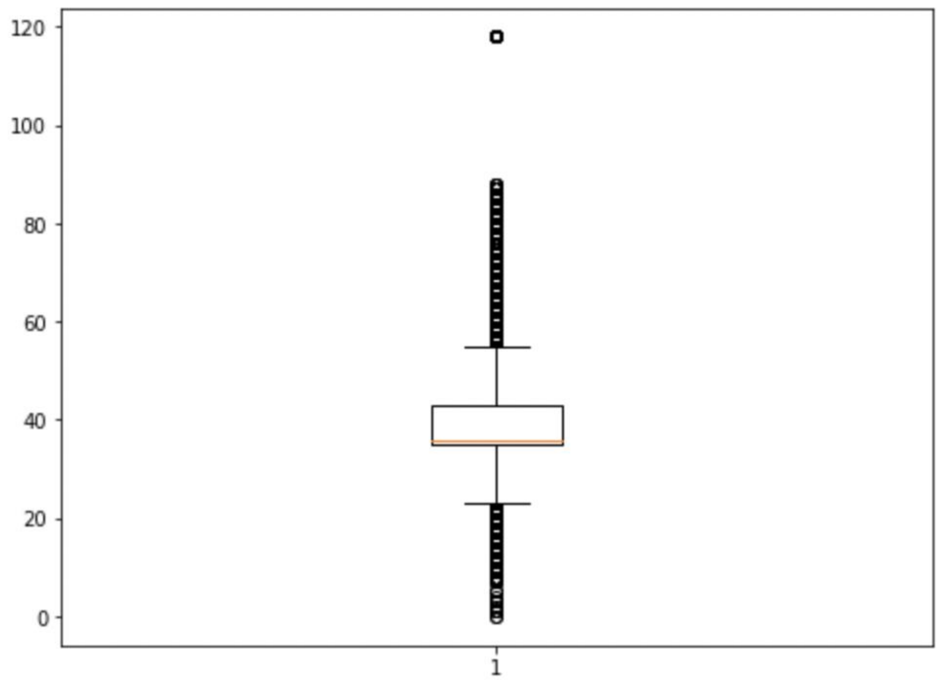
Cozma Laura-Elena
Grupa 405

```
print("Before removing commission outliers: " + str(data.count()))
data = data.where(data['Commission'] < 250)
print("After removing commission outliers: " + str(data.count()))
```

```
Before removing commission outliers: 63260
After removing commission outliers: 63251
```

Age

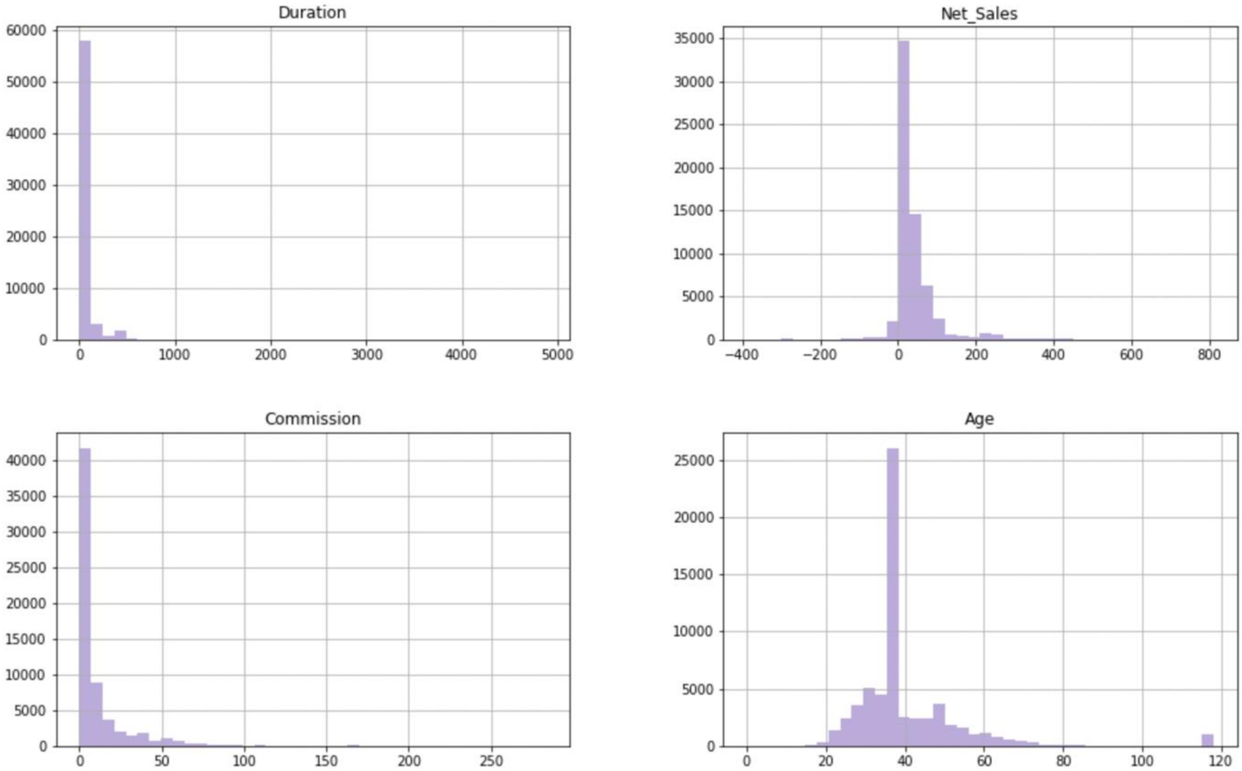
```
plt.figure(figsize=(8, 6))
plt.boxplot(data_plot['Age'])
plt.show()
```



Valoarea varstei medii este de aproximativ 40 de ani, insa se poate observa si un outlier cu valoarea de 118 ani. Desi este o varsta mare, nu este improbabila, astfel ca o vom pastra in setul de date.

De asemenea, vom construi histogramele pentru toate valorile numerice din dataset, pentru a vedea daca acestea urmeaza o anumita distributie.

```
data_plot.hist(bins=40, figsize=(16, 10), color='#BAABDA')
```



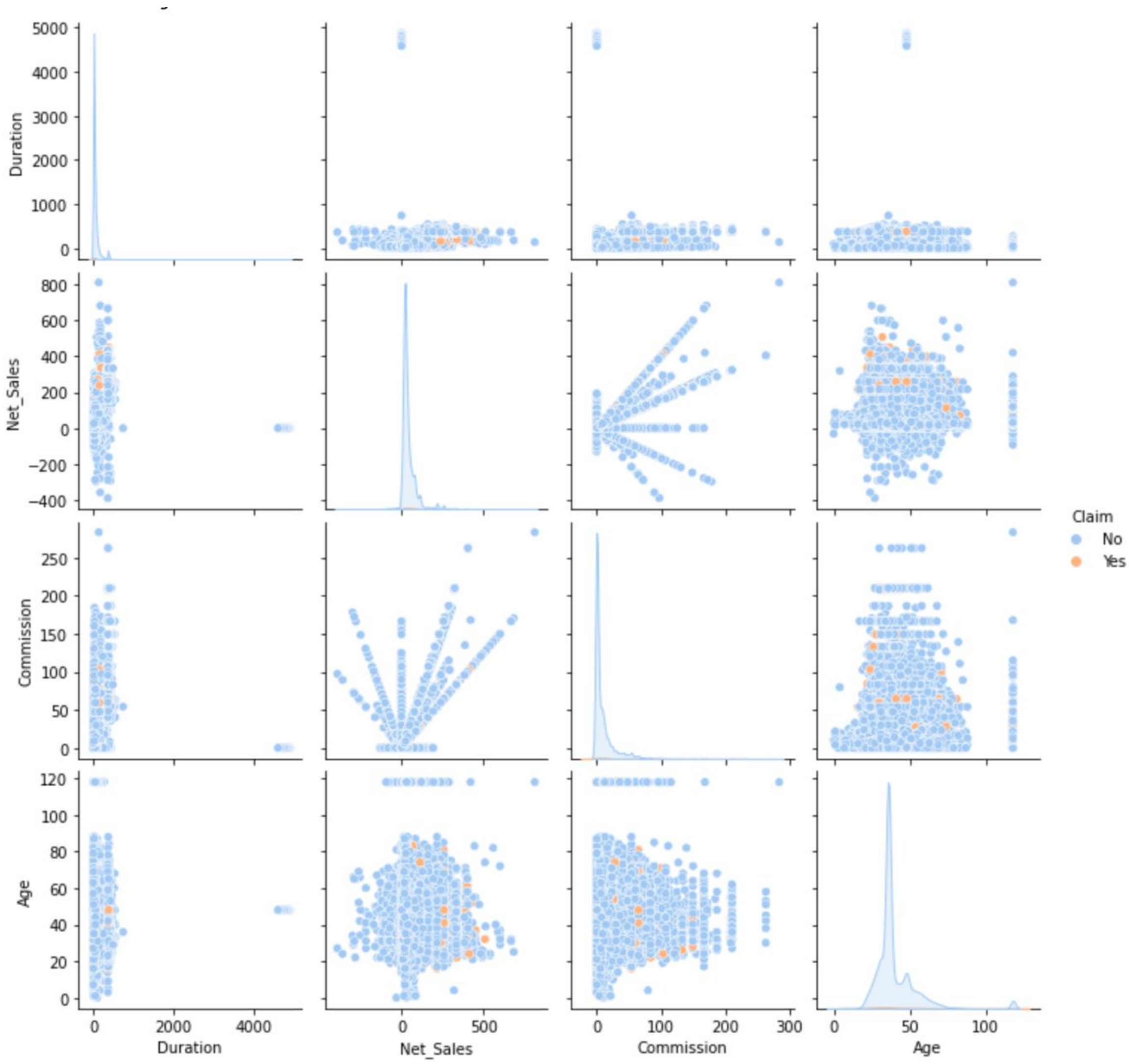
Niciuna dintre variable nu urmeaza o distributie normala.

Explorarea relatiilor dintre variabilele numerice

Aceasta se va realiza prin trasarea unui *pairplot* cu ajutorul librariei *seaborn*.

```
sns.pairplot(data_plot, hue='Claim', palette='pastel')
```

Plotul de mai jos afiseaza corelarea variabilelor pentru fiecare pereche. De asemenea, am facut distinctia dintre valorile pentru *Claim=Yes* sau *Claim=No*, in cazul in care acestea urmeaza un anumit tipar. Se observa ca nu exista variabile puternic corelate, astfel ca vor fi pastrate toate in antrenarea modelului.



Transformarea caracteristicilor

In urma analizei datelor, s-a putut observa cum coloana target *Claim* este de tipul string, astfel ca vom inlocui cu valorile numerice 1, corespunzator lui "Yes", respectiv 0, corespunzator lui "No".

```
from pyspark.sql.functions import when
data = data.withColumn("Claim", when(data['Claim'] == "Yes",
1).when(data['Claim'] == "No", 0))
```

Impartim datele in train 70% si test 30%:

```
(train_data, test_data) = data.randomSplit([0.7, 0.3])
```

Cozma Laura-Elena
Grupa 405

Deoarece setul de date este nebalansat, raportul fiind de 1:67, vom aplica Oversampling – vom duplica datele din setul de train pentru a avea echilibru intre date:

```
major_data = train_data.filter("Claim == 0")
minor_data = train_data.filter("Claim == 1")
ratio = int(major_data.count()/minor_data.count())

from pyspark.sql.functions import explode, array, lit
a = range(ratio)

oversampled_data = minor_data.withColumn("dummy", explode(array([lit(x)
for x in a]))).drop('dummy')
train_data = major_data.unionAll(oversampled_data)
major_data = train_data.filter("Claim == 0")
minor_data = train_data.filter("Claim == 1")
print("major data: {}, minor data: {}".format(major_data.count(),
minor_data.count()))
```

Vom aplica o serie de transformari asupra campurilor de tip string, precum *StringIndexer* si *OneHotEncoder*. Acesti algoritmi vor fi aplicati asupra coloanelor *Agency*, *Agency Type*, *Distribution Channel*, *Product Name* si *Destination* si vor face parte din acelasi pipeline.

String Indexer

```
from pyspark.ml.feature import StringIndexer


pipeline_stages = []
tensorflow_pipeline_stages = []
stringColumns = ['Agency', 'Agency_Type', 'Distribution_Channel',
'Product_Name', 'Destination']
for col in stringColumns:
  stringIndexer = StringIndexer(inputCol = col, outputCol = col +
'_Index', handleInvalid='keep')
  pipeline_stages += [stringIndexer]
  tensorflow_pipeline_stages += [stringIndexer]
```

One Hot Encoder

```
from pyspark.ml.feature import OneHotEncoder

stringColumns = ['Agency', 'Agency_Type', 'Distribution_Channel',
'Product_Name', 'Destination']
for col in stringColumns:
  ohe = OneHotEncoder(inputCol = col + '_Index', outputCol = col +
'_OHE', handleInvalid='keep')
  pipeline_stages += [ohe]
```

Cozma Laura-Elena
Grupa 405

Min Max Scaler

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import MinMaxScaler

stringColumns = ['Duration', 'Net_Sales', 'Commission', 'Age']
for col in stringColumns:
  assembler = VectorAssembler(inputCols=[col], outputCol=col + '_vec')
  pipeline_stages += [assembler]
  mmScaler = MinMaxScaler(inputCol = col + '_vec', outputCol = col +
'_Scaled')
  pipeline_stages += [mmScaler]
```

Vector Assembler

```
assembler = VectorAssembler(inputCols=['Agency_OHE',
                                       'Agency_Type_OHE',
                                       'Distribution_Channel_OHE',
                                       'Product_Name_OHE',
                                       'Duration_Scaled',
                                       'Destination_OHE',
                                       'Net_Sales_Scaled',
                                       'Commission_Scaled',
                                       'Age_Scaled'],
                              outputCol='features',
handleInvalid='keep')

pipeline_stages += [assembler]
```

Construirea propriu-zisa a pipeline-ului

```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages = pipeline_stages)
pipelineModel = pipeline.fit(train_data)
train_data = pipelineModel.transform(train_data)
test_data = pipelineModel.transform(test_data)
```

In ceea ce priveste adresarea problemelor datelor nebalansate, s-au analizat 3 metode:

1. Antrenarea datelor pe setul original
2. Antrenarea datelor adaugand coloana weight, ce poate fi folosita la regresie logistica. Aceasta influenteaza importanta datelor considerate, cele aflate in categoria cu un numar mai mic avand un weight mai mare
3. Utilizarea Oversampling-ului - duplicarea datelor din categoria cu mai putine date pentru a construi categorii de marime egala.

Cozma Laura-Elena
Grupa 405

Dintre cele 3 metode folosite in Logistic Regression, cea care a dat performantele cele mai mari a fost oversampling-ul, fiind metoda utilizata in continuare in notebook.

1. *Oversampling*:
- Area under ROC train: 0.8426
- Area under ROC test: 0.8321
- F1 score: 0.74
2. *Weight*:
- Area under ROC train: 0.8364
- Area under ROC test: 0.7992
- F1 score: 0.15
3. *Raw data*:
- Area under ROC train: 0.8361
- Area under ROC test: 0.7966
- F1 score: 0.09

Deoarece setul de date este in continuare nebalansat cu un raport foarte mare, orice metoda care incearca sa rezolve problema pastrand numarul de date initial, desi valoarea ROC va fi mare, in realitate F1 score este mic, deoarece exista foarte putine date cu valoarea *Claim=1*, astfel ca modelul nu va invata bine, iar numarul de valori TP va fi mult mai mic decat TN.

Antrenarea modelelor pe setul de date

S-au analizat 6 modele de clasificare - *Logistic Regression*, *SVM*, *Naive Bayes*, *Decision Tree*, *Random Forest* si *KNN* - antrenate si testate pe setul de date, dintre care le-am ales pe cele mai performante 4. Dintre acestea s-au remarcat arborii de decizie, cu un scor F1 de 0.97, impreuna cu regresia logistica, cu un scor F1 de peste 0.74.

Logistic Regression

Cozma Laura-Elena
Grupa 405

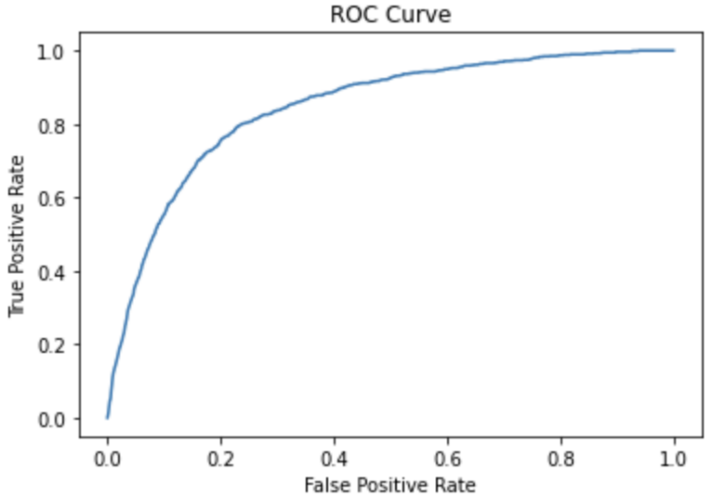
Am folosit un ParamGrid pentru CrossValidator, pentru care am oferit diverse valori pentru parametrul de regularizare, parametrul elastic net si numarul maxim de iteratii.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

lr = LogisticRegression(featuresCol = 'features', labelCol = 'Claim')


Create ParamGrid for Cross Validation
lrParamGrid = (ParamGridBuilder()
             .addGrid(lr.regParam, [0, 0.001, 0.01, 0.1, 0.5, 1.0])
             .addGrid(lr.elasticNetParam, [0.0, 0.25, 0.5, 0.75, 1.0])
             .addGrid(lr.maxIter, [5, 10, 20, 50])
             .build())

lrEvaluator = BinaryClassificationEvaluator(labelCol="Claim")

# Create 5-fold CrossValidator
lrcv = CrossValidator(estimator = lr,
                    estimatorParamMaps = lrParamGrid,
                    evaluator = lrEvaluator,
                    numFolds = 5)

lrModel = lrcv.fit(train_data)
```

Cu ajutorul output-ului din summary, am extras parametrii alesi de CrossValidator:

```
trainingSummary = lrModel.bestModel.summary
print("RegParam: " + str(lrModel.bestModel._java_obj.getRegParam()))
print("ElasticNetParam: " +
str(lrModel.bestModel._java_obj.getElasticNetParam()))
print("MaxIter: " + str(lrModel.bestModel._java_obj.getMaxIter()))

roc = trainingSummary.roc.toPandas()
plt.plot(roc['FPR'],roc['TPR'])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
print('Training set areaUnderROC: ' +
str(trainingSummary.areaUnderROC))
```

```
RegParam: 0.001
ElasticNetParam: 0.75
MaxIter: 50
```




```
Training set areaUnderROC: 0.8450509746671074
```

Am extras o metoda preluata de pe StackOverflow pentru a afisa curba ROC si pe baza output-ului din datele de test:

```
from pyspark.mllib.evaluation import BinaryClassificationMetrics

class CurveMetrics(BinaryClassificationMetrics):
    def __init__(self, *args):
        super(CurveMetrics, self).__init__(*args)

    def _to_list(self, rdd):
        points = []
        for row in rdd.collect():
            points += [(float(row._1()), float(row._2()))]
        return points

    def get_curve(self, method):
        rdd = getattr(self._java_model, method)().toJavaRDD()
        return self._to_list(rdd)
```

Dupa care am afisat Area Under ROC, Precision, Recall si valoarea F1. Setul de date fiind nebalansat, afisarea acuratetei nu se preteaza.

```
from sklearn.metrics import f1_score, confusion_matrix,
precision_score, recall_score

lrPredictions = lrModel.transform(test_data)
lrPredictions.head()
```

```
evaluator = BinaryClassificationEvaluator(labelCol="Claim")
print('Test Area Under ROC', evaluator.evaluate(lrPredictions))

y_true = lrPredictions.select("Claim").toPandas()
y_pred = lrPredictions.select("prediction").toPandas()

print('Precision: %.3f' % precision_score(y_true, y_pred))
print('Recall: %.3f' % recall_score(y_true, y_pred))
print('F1 score: %.3f' % f1_score(y_true, y_pred))

cnf_matrix = confusion_matrix(y_true, y_pred)
ax= plt.subplot()
sns.heatmap(cnf_matrix, annot=True, fmt='g', ax=ax,
cmap=sns.cm.rocket_r)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])
ax.yaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])

# Returns as a list (false positive rate, true positive rate)
preds = lrPredictions.select('Claim','probability').rdd.map(lambda row:
(float(row['probability'][1]), float(row['Claim'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(x_val, y_val)
plt.show()
```

Mai mult decat atat, am creat un grafic cu ajutorul librariei Seaborn de tip Heatmap, care reprezinta matricea de confuzie rezultata. Astfel, regresia logistica a dat rezultate destul de bune, cu un F1 score de peste 0.76 si Area under ROC de 0.838.

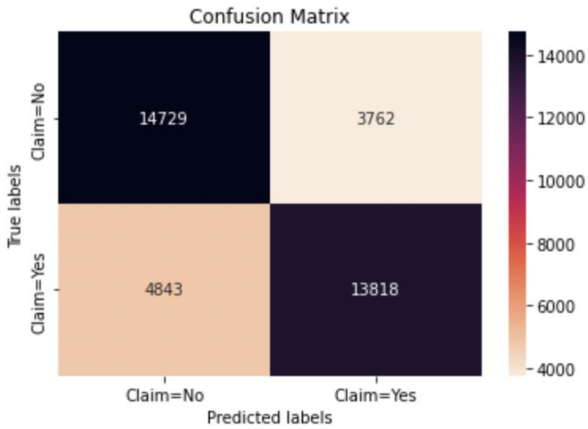
Test Area Under ROC: 0.8386250533750527

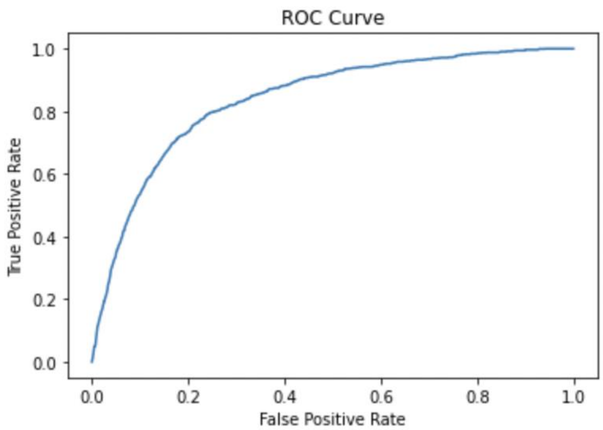
Precision: 0.786

Recall: 0.740

F1 score: 0.763

Cozma Laura-Elena
Grupa 405





Decision Tree

Am folosit din nou un ParamGrid si un CrossValidator pentru estimarea parametrilor, de data aceasta doar MaxDepth.

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(labelCol="Claim", featuresCol="features")

dtParamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [5, 10, 20, 30])
             .build())

dtEvaluator = BinaryClassificationEvaluator(labelCol = 'Claim')
dtcv = CrossValidator(estimator = dt,
                    estimatorParamMaps = dtParamGrid,
                    evaluator = dtEvaluator,
                    numFolds = 5)
dtModel = dtcv.fit(train_data)
```

Adancimea aleasa a fost si cea maxima posibila, mai exact 30.

```
print("Max depth: " + str(dtModel.bestModel._java_obj.getMaxDepth()))
```

Afisarea rezultatelor pe datele de testare:

```
dtPredictions = dtModel.transform(test_data)
```

Cozma Laura-Elena
Grupa 405

```
evaluator = BinaryClassificationEvaluator(labelCol="Claim")
print('Test Area Under ROC', evaluator.evaluate(dtPredictions))

y_true = dtPredictions.select("Claim").toPandas()
y_pred = dtPredictions.select("prediction").toPandas()

print('Precision: %.3f' % precision_score(y_true, y_pred))
print('Recall: %.3f' % recall_score(y_true, y_pred))
print('F1 score: %.3f' % f1_score(y_true, y_pred))

cnf_matrix = confusion_matrix(y_true, y_pred)
ax= plt.subplot()
sns.heatmap(cnf_matrix, annot=True, fmt='g', ax=ax,
cmap=sns.cm.rocket_r)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])
ax.yaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])

# Returns as a list (false positive rate, true positive rate)
preds = dtPredictions.select('Claim','probability').rdd.map(lambda row:
(float(row['probability'][1]), float(row['Claim'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC Curve')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.plot(x_val, y_val)
plt.show()
```

Rezultatele au fost:

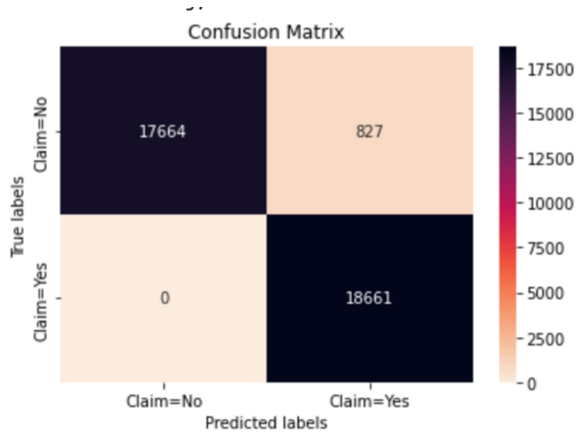
Test Area Under ROC: 0.9732101772479927

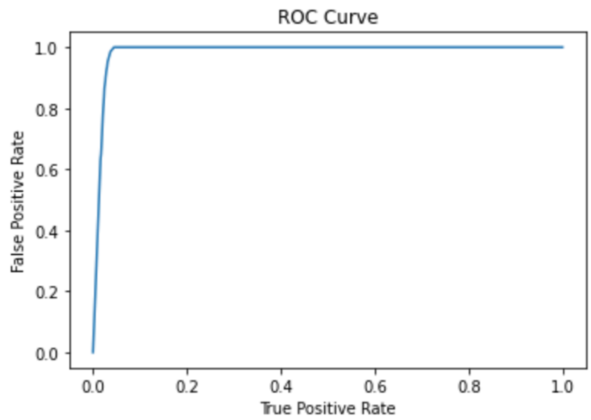
Precision: 0.958

Recall: 1.000

F1 score: 0.978

Iar matricea de confuzie impreuna cu curba ROC:





SVM

Vom folosi ParamGridBuilder pentru parametrul de regularizare si numarul maxim de iteratii.

```
from pyspark.ml.classification import LinearSVC
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator

svm = LinearSVC(featuresCol = 'features', labelCol = 'Claim')

svmParamGrid = (ParamGridBuilder()
             .addGrid(svm.regParam, [0, 0.001, 0.01, 0.1, 0.5, 1.0])
             .addGrid(svm.maxIter, [5, 10, 20, 50])
             .build())

svmEvaluator = BinaryClassificationEvaluator(labelCol = 'Claim')

svmcv = CrossValidator(estimator = svm,
                    estimatorParamMaps = svmParamGrid,
                    evaluator = svmEvaluator,
                    numFolds = 5)
svmModel = svmcv.fit(train_data)
```

Cozma Laura-Elena
Grupa 405

Afisarea parametrilor obtinuti:

```
from matplotlib import pyplot as plt
trainingSummary = svmModel.bestModel.summary
print("RegParam: " + str(svmModel.bestModel._java_obj.getRegParam()))
print("MaxIter: " + str(svmModel.bestModel._java_obj.getMaxIter()))
```

```
RegParam: 0.01
MaxIter: 20
```

Afisarea rezultatelor antrenarii pe datele de test:

```
from sklearn.metrics import f1_score, confusion_matrix,
precision_score, recall_score

svmPredictions = svmModel.transform(test_data)
# if 'rawPrediction' in svmPredictions.columns:
#   svmPredictions =
svmPredictions.withColumnRenamed("rawPrediction","predictions")
evaluator = BinaryClassificationEvaluator(labelCol="Claim")
print('Test Area Under ROC', evaluator.evaluate(svmPredictions))

y_true = svmPredictions.select("Claim").toPandas()
y_pred = svmPredictions.select("prediction").toPandas()

print('Precision: %.3f' % precision_score(y_true, y_pred))
print('Recall: %.3f' % recall_score(y_true, y_pred))
print('F1 score: %.3f' % f1_score(y_true, y_pred))

cnf_matrix = confusion_matrix(y_true, y_pred)
ax= plt.subplot()
sns.heatmap(cnf_matrix, annot=True, fmt='g', ax=ax,
cmap=sns.cm.rocket_r)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])
ax.yaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])

# Returns as a list (false positive rate, true positive rate)
preds = svmPredictions.select('Claim','rawPrediction').rdd.map(lambda
row: (float(row['rawPrediction'][1]), float(row['Claim'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC Curve')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
```

Cozma Laura-Elena
Grupa 405

```
plt.plot(x_val, y_val)
plt.show()
```

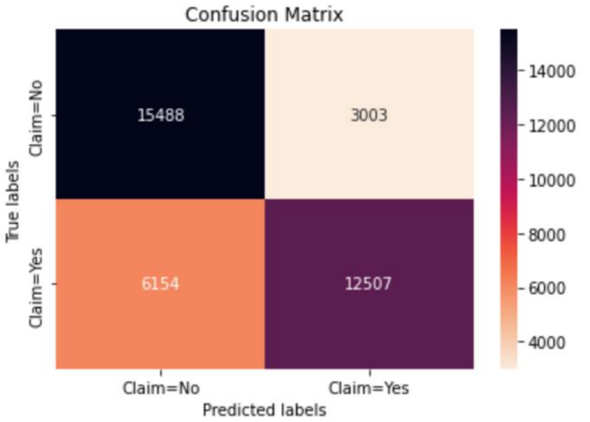
Performanta a fost similara cu cea a regresiei logistice:

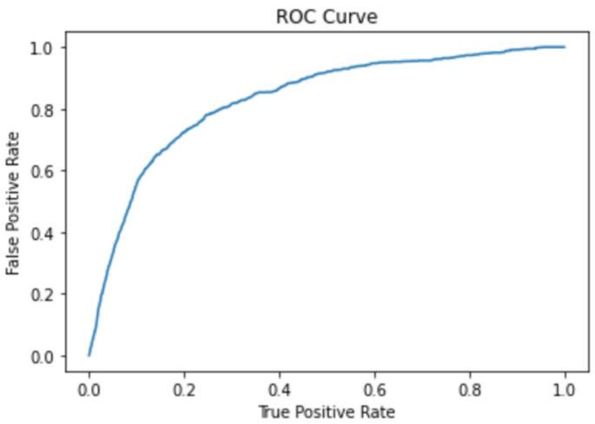
Test Area Under ROC: 0.8300577715126871

Precision: 0.806

Recall: 0.670

F1 score: 0.732





Random forest

Am folosit un ParamGridBuilder pentru maxDepth.

```
from pyspark.ml.classification import RandomForestClassifier

# Antrenarea unui model RandomForest
rf = RandomForestClassifier(labelCol="Claim", featuresCol="features",
numTrees=10)

rfParamGrid = (ParamGridBuilder()
             .addGrid(dt.maxDepth, [5, 10, 20, 30])
             .build())
rfEvaluator = BinaryClassificationEvaluator(labelCol = 'Claim')
```

Cozma Laura-Elena
Grupa 405

```
rfcv = CrossValidator(estimator = rf,
                      estimatorParamMaps = rfParamGrid,
                      evaluator = rfEvaluator,
                      numFolds = 5)
rfModel = rfcv.fit(train_data)
```

Afisam parametrul ales din bestModel:

```
from matplotlib import pyplot as plt
print("Max depth: " + str(rfModel.bestModel._java_obj.getMaxDepth()))
```

Adancimea maxima aleasa a fost 5. In continuare afisam metricile pe datele de test:

```
rfPredictions = rfModel.transform(test_data)

evaluator = BinaryClassificationEvaluator(labelCol="Claim")
print('Test Area Under ROC', evaluator.evaluate(rfPredictions))

y_true = rfPredictions.select("Claim").toPandas()
y_pred = rfPredictions.select("prediction").toPandas()

print('Precision: %.3f' % precision_score(y_true, y_pred))
print('Recall: %.3f' % recall_score(y_true, y_pred))
print('F1 score: %.3f' % f1_score(y_true, y_pred))

cnf_matrix = confusion_matrix(y_true, y_pred)
ax= plt.subplot()
sns.heatmap(cnf_matrix, annot=True, fmt='g', ax=ax,
cmap=sns.cm.rocket_r)
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])
ax.yaxis.set_ticklabels(['Claim=No', 'Claim=Yes'])

# Returns as a list (false positive rate, true positive rate)
preds = rfPredictions.select('Claim','probability').rdd.map(lambda row:
(float(row['probability'][1]), float(row['Claim'])))
points = CurveMetrics(preds).get_curve('roc')

plt.figure()
x_val = [x[0] for x in points]
y_val = [x[1] for x in points]
plt.title('ROC Curve')
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.plot(x_val, y_val)
plt.show()
```

Cozma Laura-Elena
Grupa 405

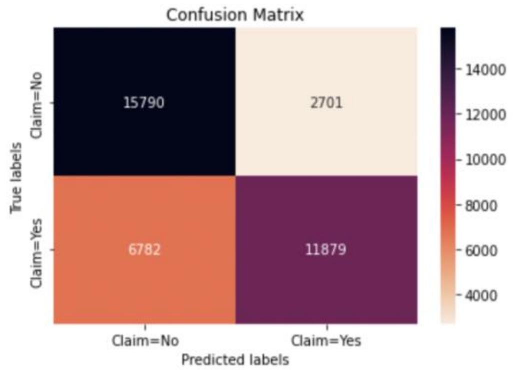
Rezultate:

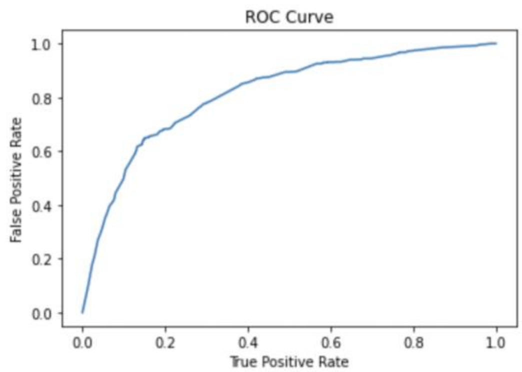
Test Area Under ROC: 0.8134097774625072

Precision: 0.815

Recall: 0.637

F1 score: 0.715





<u>Tensorflow</u>

Pentru metoda de Deep Learning am ales construirea unei retele neuronale cu 4 straturi. La inceput vom importa librariile necesare:

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

Dupa care aplicam un pipeline asupra datelor cu pasul de StringIndexer construit anterior. Pentru retea nu putem folosi OneHotEncoder si nici nu este necesar VectorAssembler.

```
pipeline = Pipeline(stages = tensorflow_pipeline_stages)
pipelineModel = pipeline.fit(data)
data_processed = pipelineModel.transform(data)
```

Cozma Laura-Elena
Grupa 405

Construim datele de test si de train, datele de test fiind formate din 25% din setul de date, luand ca seed pentru starea randomizata numarul 101. Vom construi cate doua perechi, un element din pereche avand toate coloanele mai putin labelul, iar celalalt element e format numai din coloana *Claim*.

```
data_pd = data_processed.select('Agency_Index', 'Agency_Type_Index',
'Distribution_Channel_Index', 'Product_Name_Index', 'Claim','Duration',
'Destination_Index', 'Net_Sales', 'Commission', 'Age').toPandas()
x = data_pd.drop('Claim', axis=1)
y = data_pd['Claim']


from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.25, random_state=101)
```

Vom scala datele. Acest pas nu s-a putut face in pipeline, deoarece MinMaxScaler solicita adaugarea fiecarei coloane intr-un Vector Assembler, iar acest format nu era suportat de retea.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
scaler = MinMaxScaler()

x_train_std = scaler.fit_transform(x_train[['Duration', 'Net_Sales',
'Commission', 'Age']])
x_train[['Duration', 'Net_Sales', 'Commission', 'Age']] = x_train_std
x_train.head()
x_train = np.asarray(x_train)
x_test_std = scaler.transform(x_test[['Duration', 'Net_Sales',
'Commission', 'Age']])
x_test[['Duration', 'Net_Sales', 'Commission', 'Age']] = x_test_std
x_test.head()
x_test = np.asarray(x_test)
y_train = np.asarray(y_train)
y_test = np.asarray(y_test)
```

Construim un model de tip *sequential* ce va avea 4 straturi, primele trei avand functia de activare *relu*, iar ultimul strat va avea functia de activare *sigmoid*, deoarece e vorba de o problema de clasificare binara. Vom utiliza Dropout intre straturi, pentru a ignora zgomotul si a face modelul mai robust, astfel ca erorile vor fi mai reduse.

```
model = Sequential()
model.add(Dense(units=27, activation='relu', input_shape=(9,)))
model.add(Dense(units=18, activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Dense(units=9, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(units=1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 27)                270

 dense_1 (Dense)             (None, 18)                504

 dropout (Dropout)           (None, 18)                0

 dense_2 (Dense)             (None, 9)                 171

 dropout_1 (Dropout)         (None, 9)                 0

 dense_3 (Dense)             (None, 1)                 10

=================================================================
Total params: 955
Trainable params: 955
Non-trainable params: 0
_________________________________________________________________
```

Avantajul principal la optimizorul *adam* este ca nu trebuie sa specificam learning rate-ul, deoarece e optimizat singur. Pentru ca setul de date era initial nebalansat, vom alege ca metrici alaturi de acuratete si Recall.

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

Implementam early stopping, astfel ca dupa 25 de epoci dupa care nu se va observa nicio imbunatatire, modelul va fi oprit.

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1,
patience=25)
```

Antrenam modelul:

```
history = model.fit(x=x_train,
                    y=y_train,
                    epochs=200,
                    validation_data=(x_test, y_test),
                    verbose=1,
                    callbacks=[early_stop])
```

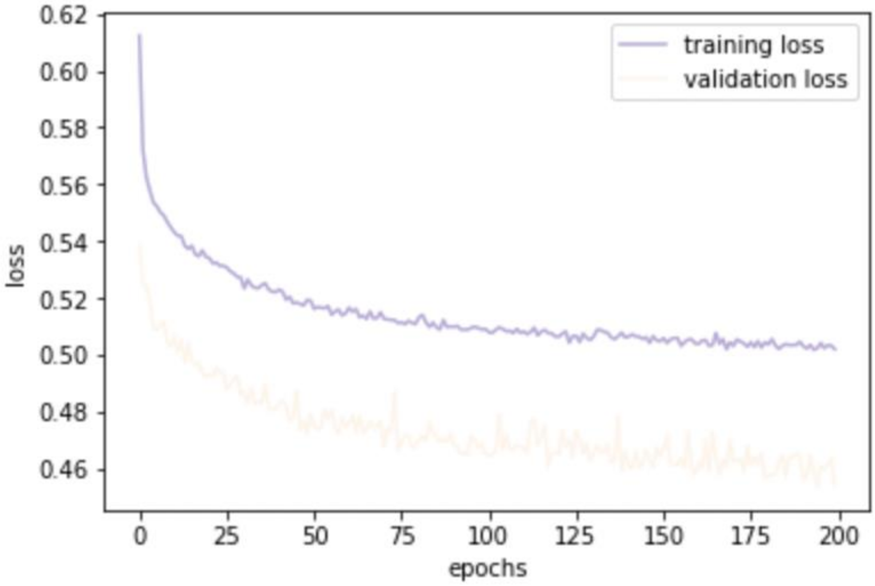
Rezultatele dupa primele 10 epoci:

Cozma Laura-Elena
Grupa 405

```
Epoch 1/200
2917/2917 [==============================] - 33s 10ms/step - loss: 0.6123 - accuracy: 0.6832 - recall: 0.5684 - val_loss: 0.5386
Epoch 2/200
2917/2917 [==============================] - 29s 10ms/step - loss: 0.5722 - accuracy: 0.7212 - recall: 0.6137 - val_loss: 0.5253
Epoch 3/200
2917/2917 [==============================] - 29s 10ms/step - loss: 0.5624 - accuracy: 0.7327 - recall: 0.6838 - val_loss: 0.5243
Epoch 4/200
2917/2917 [==============================] - 24s 8ms/step - loss: 0.5577 - accuracy: 0.7372 - recall: 0.6999 - val_loss: 0.5193
Epoch 5/200
2917/2917 [==============================] - 18s 6ms/step - loss: 0.5536 - accuracy: 0.7384 - recall: 0.7066 - val_loss: 0.5097
Epoch 6/200
2917/2917 [==============================] - 17s 6ms/step - loss: 0.5522 - accuracy: 0.7418 - recall: 0.7138 - val_loss: 0.5088
Epoch 7/200
2917/2917 [==============================] - 22s 8ms/step - loss: 0.5501 - accuracy: 0.7424 - recall: 0.7160 - val_loss: 0.5102
Epoch 8/200
2917/2917 [==============================] - 16s 5ms/step - loss: 0.5489 - accuracy: 0.7427 - recall: 0.7217 - val_loss: 0.5116
Epoch 9/200
2917/2917 [==============================] - 23s 8ms/step - loss: 0.5467 - accuracy: 0.7439 - recall: 0.7250 - val_loss: 0.5046
Epoch 10/200
2917/2917 [==============================] - 19s 6ms/step - loss: 0.5447 - accuracy: 0.7450 - recall: 0.7324 - val_loss: 0.5027
        ...
Epoch 200/200
2917/2917 [==============================] - 11s 4ms/step - loss: 0.5019 - accuracy: 0.7632 - recall: 0.8345 - val_loss: 0.4542
```

Plotam rezultatele:

```
plt.plot(history.history['loss'], c='#BAABDA', label='training loss')
plt.plot(history.history['val_loss'], c='#FFF5EB', label='validation
loss')
plt.legend()
plt.xlabel('epochs')
plt.ylabel('loss')
```



Se observa ca pana la finalul antrenarii loss-ul a ajuns pentru datele de test ls 0.52, in timp ce pe training era la 0.46, deci o diferenta de 0.06.

Afisam metricile:

```
from sklearn.metrics import classification_report

y_pred = (model.predict(x_test) > 0.5).reshape((-1,))
print(classification_report(y_test, y_pred))
```

Cozma Laura-Elena
Grupa 405

```
              precision    recall  f1-score   support

           0       0.82      0.73      0.78     15591
           1       0.76      0.84      0.80     15518

    accuracy                           0.79     31109
   macro avg       0.79      0.79      0.79     31109
weighted avg       0.79      0.79      0.79     31109
```

Spark Streaming

Am incercat transmiterea datelor din fisier incarcate anterior printr-o topica de Kafka. Am inceput cu importurile necesare.

```
!pip install kafka-python

!curl -sSOL https://dlcdn.apache.org/kafka/3.1.0/kafka_2.13-3.1.0.tgz
!tar -xzf kafka_2.13-3.1.0.tgz


!./kafka_2.13-3.1.0/bin/zookeeper-server-start.sh -daemon ./kafka_2.13-3.1.0/config/zookeeper.properties
!./kafka_2.13-3.1.0/bin/kafka-server-start.sh -daemon ./kafka_2.13-3.1.0/config/server.properties
!echo "Waiting for 10 secs until kafka and zookeeper services are up and running"
!sleep 10


!ps -ef | grep kafka
```

Am creat doua topice de kafka, una numita *insurance-train*, iar alta numita *insurance-test* la localhost cu portul 9092.

```
!./kafka_2.13-3.1.0/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 1 --topic insurance-train
!./kafka_2.13-3.1.0/bin/kafka-topics.sh --create --bootstrap-server 127.0.0.1:9092 --replication-factor 1 --partitions 2 --topic insurance-test


!./kafka_2.13-3.1.0/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic insurance-train
!./kafka_2.13-3.1.0/bin/kafka-topics.sh --describe --bootstrap-server 127.0.0.1:9092 --topic insurance-test
```

```
Topic: insurance-train  TopicId: EV4QIzJyS-mjzkpKZPnPJw PartitionCount: 1       ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: insurance-train  Partition: 0    Leader: 0       Replicas: 0     Isr: 0
Topic: insurance-test   TopicId: UJ6V3qegQRS2C3Ck4aixhQ PartitionCount: 2       ReplicationFactor: 1    Configs: segment.bytes=1073741824
        Topic: insurance-test   Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: insurance-test   Partition: 1    Leader: 0       Replicas: 0     Isr: 0
```

Modelarea datelor inainte de distribuire:

```
from sklearn.model_selection import train_test_split
x = data.toPandas().drop('Claim', axis=1)
y = data.toPandas()['Claim']
x_train_df, x_test_df, y_train_df, y_test_df = train_test_split(x, y,
test_size=0.25, random_state=101)

x_train = list(filter(None,
x_train_df.to_csv(index=False).split("\n")[1:]))
y_train = list(filter(None,
y_train_df.to_csv(index=False).split("\n")[1:]))

x_test = list(filter(None,
x_test_df.to_csv(index=False).split("\n")[1:]))
y_test = list(filter(None,
y_test_df.to_csv(index=False).split("\n")[1:]))

NUM_COLUMNS = len(x_train_df.columns)
len(x_train), len(y_train), len(x_test), len(y_test)
```

```
(93324, 93324, 31109, 31109)
```

Scriem datele in topicurile de kafka. Acestea vor fi trimise la *127.0.0.1:9092*.

```
def error_callback(exc):
    raise Exception('Error while sendig data to kafka:
{0}'.format(str(exc)))

def write_to_kafka(topic_name, items):
  count=0
  producer = KafkaProducer(bootstrap_servers=['127.0.0.1:9092'])
  for message, key in items:
    producer.send(topic_name, key=key.encode('utf-8'),
value=message.encode('utf-8')).add_errback(error_callback)
    count+=1
  producer.flush()
  print("Wrote {0} messages into topic: {1}".format(count, topic_name))

write_to_kafka("insurance-train", zip(x_train, y_train))
write_to_kafka("insurance-test", zip(x_test, y_test))
```

```
Wrote 93324 messages into topic: insurance-train
Wrote 31109 messages into topic: insurance-test
```

Cozma Laura-Elena
Grupa 405

Setam variabila PYSPARK_SUBMIT_ARGS:

```
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0'
```

Initializari pentru *consumer*:

```
kafka_topic_name = "insurance-train"
kafka_bootstrap_servers = 'localhost:9092'

from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SQLContext

sc = spark.sparkContext
ssc = StreamingContext(sc, 5)
```

Citim datele cu ajutorul lui *readStream* construind un subscriber atasat topicului definit anterior. Din pacate, urmatoarele celule de cod nu functioneaza din cauza problemelor de versiune.

```
# Subscribe to 1 topic
df = spark.readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
  .option("subscribe", kafka_topic_name) \
  .load()

df.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")
lines = df.map(lambda x: x[1])
counts = lines.flatMap(lambda line: line.split(' '))
counts = lines.flatMap(lambda line: line.split(' ')).map(lambda word: (word, 1)).reduceByKey(lambda a, b: a+b)
counts.pprint()
ssc.start()
# stream will run for 50 sec
ssc.awaitTerminationOrTimeout(50)
ssc.stop()
sc.stop()
```