CSIP5203: Big Data Analytics Applications

Assignment 2: Big Data Analytics

P-2777638

(Word counts: 2530)

Table of Contents

Task 1: Analysing Data with Spark SQL

The first task is to use Spark SQL to analyze the crime data.

Introduction of dataset

The dataset, which was obtained from the Data Police UK website, includes three cities: Nottingham, Derby, and Leicester. The information covers the years 2021 and 2022. We will be able to observe the pattern of crime rates for three distinct cities over a 24-month period. The dataset contains Crime ID, Month, Reported by, Falls within, Longitude, Latitude, LSOA code, LSOA name, crime type, last outcome category, context.

Data Analysis

As spark is used to analyse the data, I need to install and call the necessary libraries (“1.1”).

```
!pip install pyspark==3.2


#Start spark session and configuration
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext
spark = SparkSession.builder.master("local[2]").appName("dat").getOrCreate()
sc = spark.sparkContext
#create an instance of SQLContext
sqlContext = SQLContext(spark)
```

The dataset is loaded into the LNDCrime spark dataframe using spark.read.option().

```
LDNCrime = spark.read.option("header", "true").option("delimiter", ",")\
.option("inferSchema", "true").\
csv("/kaggle/input/3-cities-crime/Leicester_Derby_Nottingham Crime data 2021-22")
```

Now, first of all, we have to prepare the dataset so that it can be analysed. To make column labels more accessible, column labels are renamed by LDNCrime.withColumnRenamed().

```
# tidy up the column names

LDNCrime = LDNCrime.withColumnRenamed('Crime ID', 'Crime_ID')
LDNCrime = LDNCrime.withColumnRenamed('Reported by', 'Reported_by')
LDNCrime = LDNCrime.withColumnRenamed('Falls within', 'Falls_within')
LDNCrime = LDNCrime.withColumnRenamed('LSOA code', 'LSOA_code')
LDNCrime = LDNCrime.withColumnRenamed('LSOA name', 'LSOA_name')
LDNCrime = LDNCrime.withColumnRenamed('Crime type', 'Crime_type')
LDNCrime = LDNCrime.withColumnRenamed('Last outcome category', 'Last_outcome_category')
```

LDNCrime.printSchema() is used to print the data type of the schema of dataframe in the tree format.

```
root
 |-- Crime_ID: string (nullable = true)
 |-- Month: string (nullable = true)
 |-- Reported_by: string (nullable = true)
 |-- Falls_within: string (nullable = true)
 |-- Longitude: double (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Location: string (nullable = true)
 |-- LSOA_code: string (nullable = true)
 |-- LSOA_name: string (nullable = true)
 |-- Crime_type: string (nullable = true)
 |-- Last_outcome_category: string (nullable = true)
 |-- Context: string (nullable = true)
```

Now, let's take a look at the monthly crime rate. Using group by function, we can have a total crime rate per month.

```
LDNCrime.groupby("Month").count().orderBy("Month").show()
```

```
+-------+-----+
|  Month|count|
+-------+-----+
|2021-01|26044|
|2021-02|25316|
|2021-03|30717|
|2021-04|29992|
|2021-05|30720|
|2021-06|32975|
|2021-07|32286|
|2021-08|30060|
|2021-09|29614|
|2021-10|28585|
|2021-11|29091|
|2021-12|27046|
|2022-01|26283|
|2022-02|27338|
|2022-03|31359|
|2022-04|30723|
|2022-05|32872|
|2022-06|28952|
|2022-07|29625|
|2022-08|31867|
+-------+-----+
only showing top 20 rows
```

There are a few steps to visualize monthly crime in a graph. First, the spark dataframe is needed to be registered in a temporary table. So, for that, another temporary spark dataframe named LDNCrime_temp is created with month and count of total crime.

```
LDNCrime_temp = LDNCrime.select(LDNCrime.Month)\
.groupby(LDNCrime.Month).count()
```

And the DataFrame converted into a temporary table named tbl_LDNCrime_temp.

```
LDNCrime_temp.registerTempTable("tbl_LDNCrime_temp")
```

tbl_LDNCrime_temp is converted to a RDD and applies map() transformation to extract values in a list.

```
monthyear = sqlContext.sql("select Month from tbl_LDNCrime_temp order by Month")\
.rdd.map(lambda f:f[0]).collect()
Total = sqlContext.sql("select count from tbl_LDNCrime_temp order by Month")\
.rdd.map(lambda f:f[0]).collect()
```

Crime rate graph is plotted with monthyear and total using matplotlib.pyplot (“1.2”). And we can see a pattern of time when crime increased and decreased in Fig 1.

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


plt.figure(figsize=(20, 4))
plt.plot(monthyear, Total, linestyle='-', marker='o', color='b', label='Data Points')
#plt.subplot(2, 1, 1)
plt.plot(y, t, 'o-')
plt.title('Crime amount by year-month')
plt.ylabel('Cumulate (month)')
plt.grid(True)
```

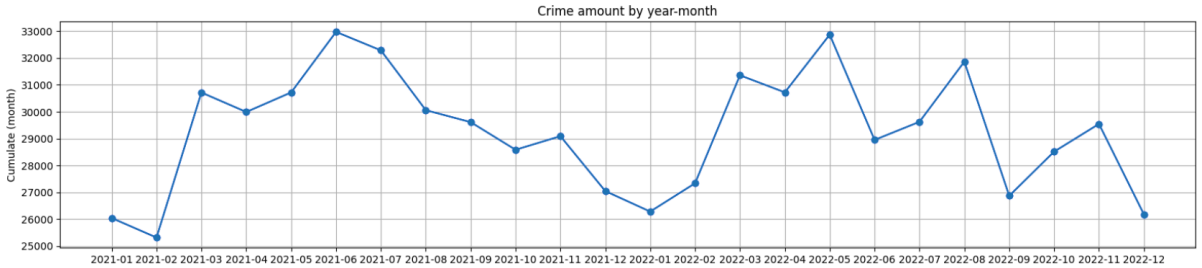


Fig 1

Now, let's find about only months. monthlyCrimeAmount, a new dataframe is created with month, total and averages.

```
monthlyCrimeAmount = sqlContext.\
sql("select substring(Month, 6, 2) as Month, \
SUM(count) as Total, avg(count) as \
Average from tbl_LDNCrime_temp group by substring(Month, 6, 2)")
```

Here is the monthly total crime rate and average rate. Fig 2 is showing the trend of crime rate. In the early and last months, there is less crime than in the middle months.

Month	Total	Average
01	52327	26163.5
02	52654	26327.0
03	62076	31038.0
04	60715	30357.5
05	63592	31796.0
06	61927	30963.5
07	61911	30955.5
08	61927	30963.5
09	56484	28242.0
10	57100	28550.0
11	58638	29319.0
12	53229	26614.5

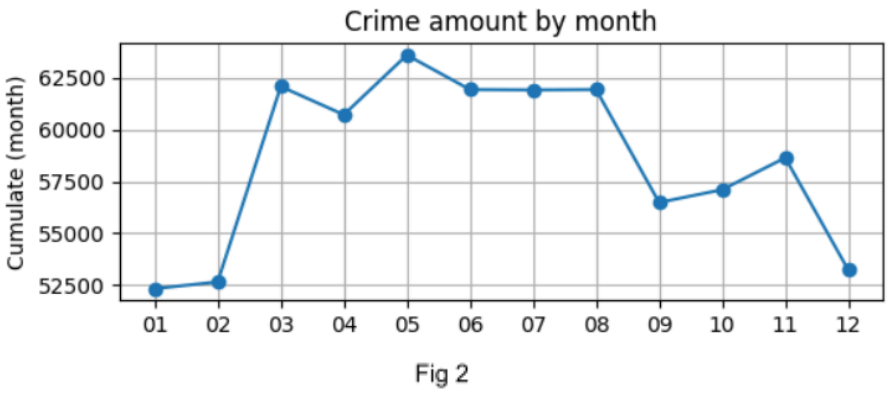


Fig 2

In the similar way of months, we can also find out about yearly crime rate. The yearlyCrimeAmount DataFrame is created to retrieve information of total and average crime by year.

```
yearlyCrimeAmount = sqlContext.sql("select substring(Month, 0, 4) \
as Year, SUM(count) as Total, avg(count) as Average from \
tbl_LDNCrime_temp group by substring(Month, 0, 4)")
```

Using the pie chart, we can visualize the difference between 2021 and 2022 (“1.3”).

```
plt.pie(totalArray, labels = yearArray, autopct='%1.1f%%')
plt.title('Crime amount between 2021 and 2022')
plt.legend()
```

The difference between 2021 and 2022 crime amount is less than 0.5%.

```
+----+------+------------------+
|Year| Total|           Average|
+----+------+------------------+
|2022|350134|29177.833333333332|
|2021|352446|           29370.5|
+----+------+------------------+
```

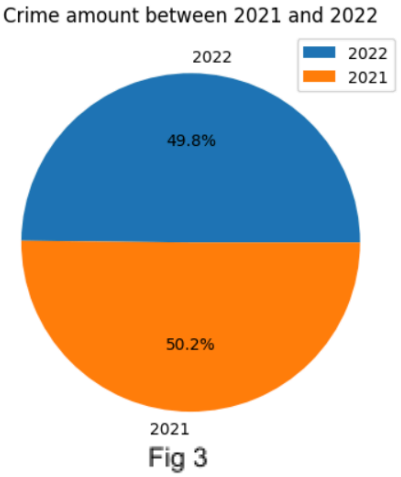


Fig 3

We have 3 non-metropolitan countries crimes: Leicestershire, Derbyshire, Nottinghamshire.

```
LDNCrime_Loc = LDNCrime.select(LDNCrime.Reported_by)\
.groupby(LDNCrime.Reported_by).count()

LDNCrime_Loc.registerTempTable("tbl_LDNCrime_loc")

Location = sqlContext.sql("select Reported_by from tbl_LDNCrime_loc")\
.rdd.map(lambda f:f[0]).collect()
Total = sqlContext.sql("select count from tbl_LDNCrime_loc")\
.rdd.map(lambda f:f[0]).collect()

 plt.figure(figsize=(10, 6))
 plt.bar(Location, Total)
 plt.title('Crime amount by non-metropolitan countries')
 plt.ylabel('Cumulate (year)')
```

From the bar chart, all three countries have a crime rate between 210000 to 260000 (“1.4”). Among three, Nottinghamshire has the most crime and leicestershire has the least (Fig 4).

Reported_by	count
Nottinghamshire P...	256640
Derbyshire Consta...	231754
Leicestershire Po...	214186

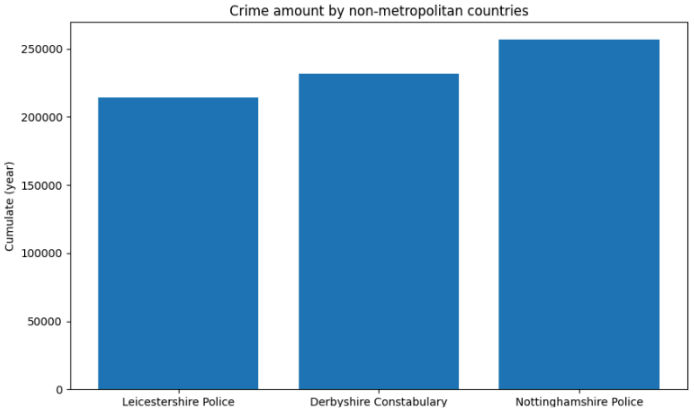


Fig 4

Now, we do also have different types of crime. We can find yearly rates for each crime and total for each.

```
LDNCrime_ct = LDNCrime.select(LDNCrime.Crime_type)\
.groupby(LDNCrime.Crime_type).count()

LDNCrime_ct.registerTempTable("Tbl_LDNCrime_ct")

sqlContext.sql("select * from Tbl_LDNCrime_ct").show()
```

Crime_type	Year	Total
Anti-social behav...	2021	80316
Robbery	2021	2146
Violence and sexu...	2022	125549
Drugs	2022	9699
Other theft	2021	20707
Shoplifting	2021	16750
Other theft	2022	25261
Shoplifting	2022	20022
Criminal damage a...	2022	31472
Anti-social behav...	2022	52384
Robbery	2022	2627
Theft from the pe...	2021	2021
Burglary	2022	14451
Drugs	2021	9979
Vehicle crime	2021	15322
Other crime	2022	7709
Public order	2022	32834
Bicycle theft	2022	3978
Criminal damage a...	2021	29841
Possession of wea...	2022	3453

Crime_type	count
Bicycle theft	7624
Public order	63799
Drugs	19678
Other crime	14844
Robbery	4773
Criminal damage a...	61313
Theft from the pe...	4723
Shoplifting	36772
Burglary	26536
Other theft	45968
Possession of wea...	6323
Violence and sexu...	244212
Vehicle crime	33315
Anti-social behav...	132700

I will only go for total rates for each here. From Fig 5, The most occurring crime is violence and sexual offences and the least is theft from the person.

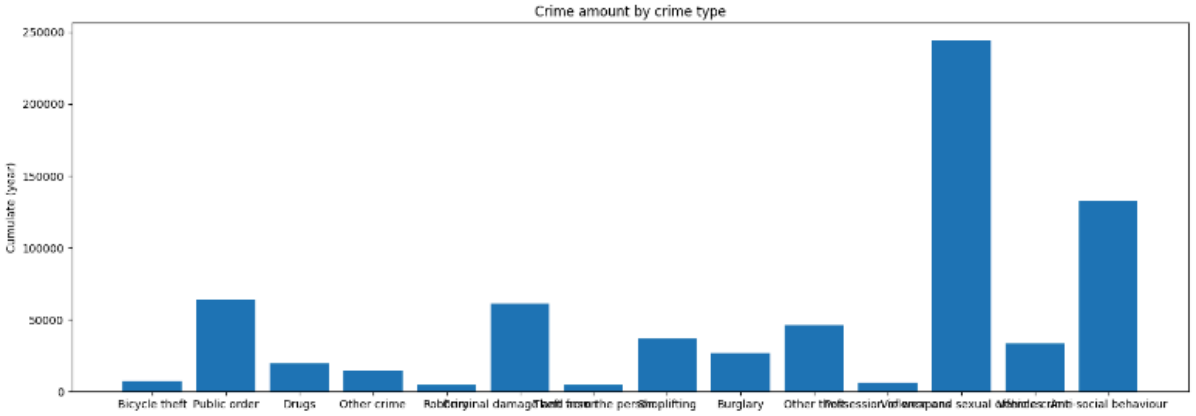


Fig 5

We have columns named LSOA code and LSOA name. Lower layer super output areas are known as LSOAs. These geographic rankings are intended to enhance the reporting of statistics for small areas in Wales and England.

The top 20 rows of LSOA names with the highest crime rates are displayed here.

```
LDNCrime.groupby("LSOA_name").count().orderBy(desc("count")).show()
```

```
[Stage 107:=============================================>             (3 + 1) / 4]
+--------------------+-----+
|           LSOA_name|count|
+--------------------+-----+
|      Leicester 041A|10069|
|          Derby 013F| 8616|
|                null| 6440|
|     Nottingham 039E| 5047|
|     Nottingham 039B| 3854|
|   Chesterfield 010F| 3426|
|          Derby 016A| 3293|
|      Bassetlaw 013C| 2713|
|     Nottingham 023H| 2706|
|      Leicester 041D| 2642|
|      Mansfield 008F| 2588|
|      Charnwood 003A| 2393|
|      Leicester 008B| 2288|
|     Nottingham 022D| 2275|
|Newark and Sherwo...| 2263|
|      Mansfield 008D| 2113|
|          Derby 008B| 1891|
|     Nottingham 002E| 1879|
|          Blaby 006C| 1801|
|      Leicester 023C| 1761|
+--------------------+-----+
only showing top 20 rows
```

Fig 6

Discussion/Findings

I began by analyzing each month of the year, then just the months and years. that displays the trends and conduct of criminal activity over months and years. According to Fig. 1, the crime rate was lowest at the beginning of both years, increased gradually until the middle of the year, and then gradually declined at the end of the year. It is also depicted in Fig 2. 2022's crime rate is lower than 2021's, indicating a decline in crime (Fig 3). Nottinghamshire has the highest rate of crime out of the three non-metropolitan nations (Fig 4). The data on the frequency of different types of crimes is presented in Fig 5. Finally, Fig. 6 shows the amount of crime that each lower area has.

Task 2: Implement machine learning algorithms for data analytics

Introduction of Problem

TItanic data is used in this task. I need to apply some classification machine learning algorithms which are available in the Pyspark Machine library(MLib) and find out which one gives the better accuracy. Before that I need to preprocess the data and do feature transformation.

Approaches

Data analysis and Preprocessing

Let's declare all necessary libraries such as pandas ("2.1"), numpy, seaborn ("2.3"), missingo, ("2.4") and sklearn,preprocessing.LabelEncoder ("2.5") here.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
from sklearn.preprocessing import LabelEncoder
```

using pd.read_csv(), csv is loaded into a pandas DataFrame. and takes a look in DataFrame.

```
df = pd.read_csv('/kaggle/input/titanic/TitanicData1.csv')
df.head()
```

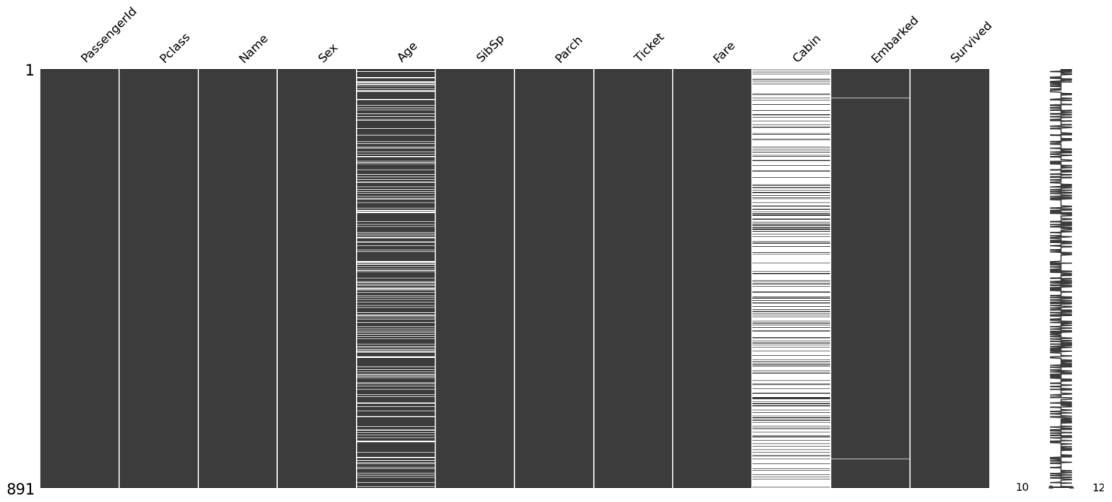
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Survived
0	1	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C	1
2	3	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
3	4	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S	1
4	5	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S	0

Now, df.info() is used to observe column names, null values, and data types.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Pclass       891 non-null    int64
 2   Name         891 non-null    object
 3   Sex          891 non-null    object
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Ticket       891 non-null    object
 8   Fare         891 non-null    float64
 9   Cabin        204 non-null    object
 10  Embarked     889 non-null    object
 11  Survived     891 non-null    int64
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Missingo provides missing data visualization plots to understand better. As we can see, out of 12 column 2 has lots of missing values and one has only 2. The white shade represents missing values. Let's handle missing values first.

```
import missingno as msno
msno.matrix(df)
```



As The column 'Cabin' has more null values than non-null values, it will be better to drop the column if it has no correlation with other columns. So, I will find the 'Cabin' correlation.

The temp_df is a temporary copied dataframe from the main dataframe df. From the Sklearn library, I import LabelEncoder that is used for encoding categorical or text data into numerical values.

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

for column in temp_df.select_dtypes(include='object').columns:
    temp_df[column] = label_encoder.fit_transform(temp_df[column])
```

Now, the DataFrame is ready. Let's plot sns.heatmap with temp_df.corr().

```
fig = plt.figure(figsize=(10,7))
ax = sns.heatmap(temp_df.corr(),vmax=1,square=True,annot=True)
```

Values closer to 1 or -1 indicate a stronger correlation and closer to 0 indicate a weaker correlation. From the heatmap in Fig 1, 'Cabin' column has the values close to 0.

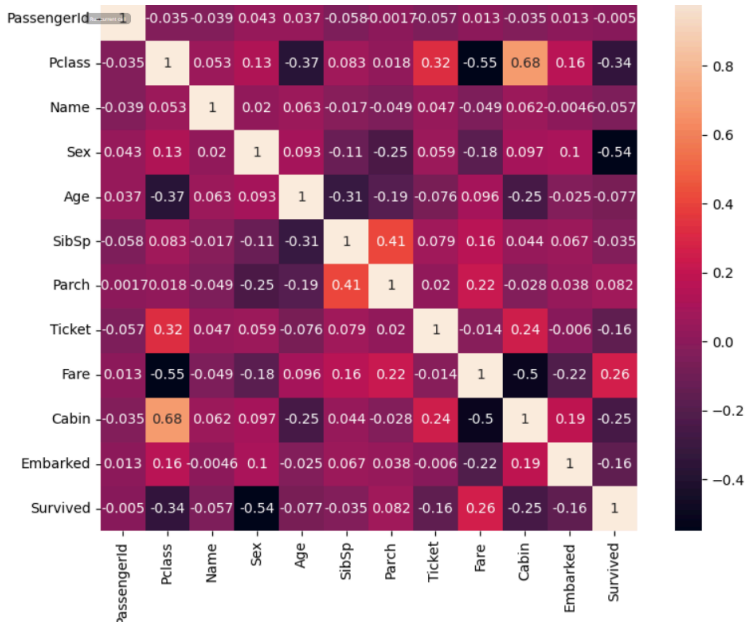


Fig 1

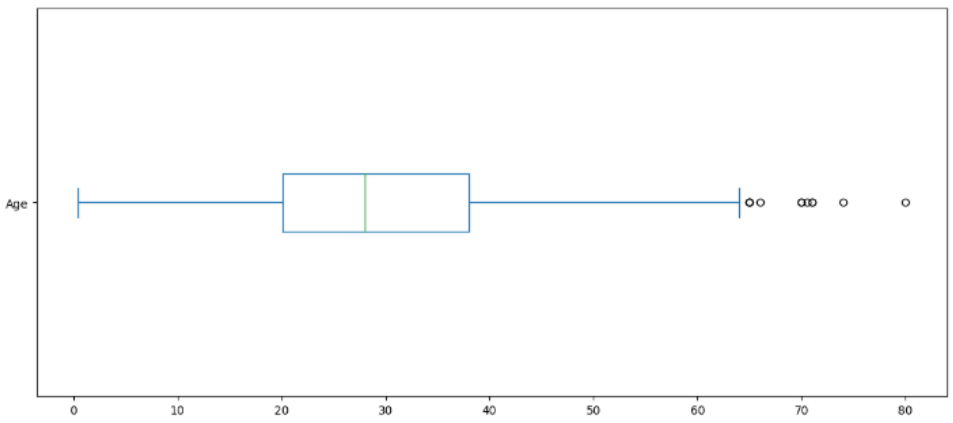
Returning to the main DataFrame df, the column ‘Cabin’ is dropped using df.drop().

```
df = df.drop('Cabin', axis=1)
```

I plot box plot to observe the distribution of data. There are also some outliers in the end.

```
df['Age'].plot(kind='box', vert=False, figsize=(14,6))
```

```
<Axes: >
```



Now about missing values in the ‘Age’ column, the median value, 28 will replace them.

```
df['Age'] = df['Age'].fillna(df['Age'].median())
df['Age'] = df['Age'].astype(int)
```

The machine learning model may have issues as a result of the outliers. So, replacing it with a median is preferable. I start by figuring out the interquartile range (IQR). Upper Age and Lower Age can be computed from the IQR. Next, using loc(), numpy NaN values are used to replace the data that are larger than upperAge and smaller than lowerAge, and fillna() is used to fill the gaps by median().

```
perAge = np.percentile(df['Age'],[0,25,50,75,100])
IQR = perAge[3]-perAge[1]
upperAge = perAge[3]+IQR*1.5
lowerAge = perAge[1]-IQR*1.5

df.loc[(df['Age'] > upperAge) | (df['Age'] < lowerAge), 'Age'] = np.nan
df.fillna(df['Age'].median(),inplace=True)

df['Age'] = df['Age'].astype(int)
```

The dataset contains two non-numerical variables: "Sex" and "Embarked." Thus, they must be changed. The data can be converted to represent ‘Male’ as 1 and ‘Female’ as 0 for the 'Sex' column.

```
df['Sex'] = df['Sex'].map( {'male':1,'female':0} )
```

For the ‘Embarked’ column, The data can be converted to represent ‘Q’ as 2, ‘S’ as 1, and ‘C’ as 0.

```
df['Embarked'] = df['Embarked'].map( {'Q':2,'S':1,'C':0} )
```

There are 2 missing values in the ‘Embarked’ column. It can be replaced by mean value and convert the datatype to integer.

```
df['Embarked'] = df['Embarked'].fillna(df['Embarked'].mean())
df['Embarked'] = df['Embarked'].astype(int)
```

Let’s do a final check for null values using df.isnull().sum().

```
df.isnull().sum()
```

```
PassengerId    0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Embarked       0
Survived       0
dtype: int64
```

Features Transformation

Now, the pyspark package needs to be installed.

```
!pip install pyspark==3.2
```

The SparkConf, SparkContext from the pyspark and SparkSession. SQLContext from pyspark.sql needs to be called. A Sparksession needs to be created as spark.

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession, SQLContext
spark = SparkSession.builder.appName("PandasToSpark").getOrCreate()
sc = spark.sparkContext
```

While converting pandas DataFrame from spark DataFrame, There is an error showing " 'DataFrame' object has no attribute 'iteritems' ". To avoid this, iteritems can be declared as items to iterate over the columns of DataFrame.

```
pd.DataFrame.iteritems = pd.DataFrame.items
```

Now, DataFrame can be loaded.

```
spark_df = spark.createDataFrame(df)
spark_df.show()
```

To perform machine learning pipelines, the PySpark MLib library's "VectorAssembler" and "StringIndexer", two crucial feature transformers, must be called ("2.6") ("2.7").

```
from pyspark.ml.feature import VectorAssembler,StringIndexer
```

A given list of columns can be combined into a single vector column using VectorAssembler. Since there is no significant correlation between the Name and Ticket columns, I include all columns except them.

```
Assembler_features = VectorAssembler(inputCols=['PassengerId', 'Pclass',
'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked'],outputCol='features')
```

StringIndexer is used to convert ''Survived column to numerical indices column.

```
Label = StringIndexer(inputCol='Survived',outputCol='survived')
```

Using 'Pipeline', Pipeline is created with stages=[Assembler_features,Label] in data_prep_pipe ("2.8").

```
from pyspark.ml import Pipeline
data_prep_pipe = Pipeline(stages=[Assembler_features,Label])
```

Now, spark_df is needed to be fit and transformed using the pipeline into clean_data.

```
cleaner = data_prep_pipe.fit(spark_df)
clean_data = cleaner.transform(spark_df)
```

We can take a look at the data and how it looks.

```
clean_data = clean_data.select(['label','features'])
clean_data.show()
```

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  0.0|[1.0,3.0,1.0,22.0...|
|  1.0|[2.0,1.0,0.0,38.0...|
|  1.0|[3.0,3.0,0.0,26.0...|
|  1.0|[4.0,1.0,0.0,35.0...|
|  0.0|[5.0,3.0,1.0,35.0...|
|  0.0|[6.0,3.0,1.0,28.0...|
|  0.0|[7.0,1.0,1.0,54.0...|
|  0.0|[8.0,3.0,1.0,28.0...|
|  1.0|[9.0,3.0,0.0,27.0...|
|  1.0|[10.0,2.0,0.0,14....|
|  1.0|[11.0,3.0,0.0,4.0...|
|  1.0|[12.0,1.0,0.0,28....|
|  0.0|[13.0,3.0,1.0,20....|
|  0.0|[14.0,3.0,1.0,39....|
|  0.0|[15.0,3.0,0.0,14....|
|  1.0|[16.0,2.0,0.0,28....|
|  0.0|[17.0,3.0,1.0,28....|
|  1.0|[18.0,2.0,1.0,28....|
|  0.0|[19.0,3.0,0.0,31....|
|  1.0|(8,[0,1,3,6],[20....|
+-----+--------------------+
only showing top 20 rows
```

Next, using randomSplit(), clean_data is split into 70% and 30% for the train-set and test-set, respectively.

```
(train_set,test_set) = clean_data.randomSplit([0.7,0.3])
```

Classification

The procedure of all classification models is almost the same.

Naive Bayes Classification (“2.9”)

From the pyspark.ml.classification, NaiveBayes is imported. For instance, NB is created from the NaiveBayes classifier. NB is used to train a Naive Bayes model. The transform method is called on the trained model with the test set. This generates predictions for the test set using the trained Naive Bayes model. The show method is used to display the predictions DataFrame. It will show the original columns along with additional columns like "rawPrediction," "probability," and "prediction," which are typical outputs of a classification model in PySpark.

```
from pyspark.ml.classification import NaiveBayes
NB = NaiveBayes()
# Train Naive bayes model using training data
model = NB.fit(train_set)

predictions = model.transform(test_set)
predictions.show()
```

```
+-----+--------------------+--------------------+--------------------+----------+
|label|            features|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|  0.0|(8,[0,1,3,6],[178...|[-258.39123220397...|[3.30705461950863...|       1.0|
|  0.0|[27.0,3.0,1.0,28....|[-127.99240061616...|[0.52566986863211...|       0.0|
|  0.0|[34.0,2.0,1.0,28....|[-139.92172375099...|[0.14741434925644...|       1.0|
|  0.0|[41.0,3.0,0.0,40....|[-177.38388187145...|[0.32867002302026...|       1.0|
|  0.0|[42.0,2.0,0.0,27....|[-170.83318895868...|[1.81958187207803...|       1.0|
|  0.0|[43.0,3.0,1.0,28....|[-132.00642506975...|[0.57558533056121...|       0.0|
|  0.0|[52.0,3.0,1.0,21....|[-119.11847063394...|[0.64994163920450...|       0.0|
|  0.0|[55.0,1.0,1.0,28....|[-297.23878529628...|[3.47632054743284...|       1.0|
|  0.0|[58.0,3.0,1.0,28....|[-131.76718696365...|[0.78432657561271...|       0.0|
|  0.0|[64.0,3.0,1.0,4.0...|[-168.92637136592...|[1.17854712795708...|       1.0|
|  0.0|[72.0,3.0,0.0,16....|[-269.81887340627...|[1.56439031931002...|       1.0|
|  0.0|[73.0,2.0,1.0,21....|[-319.37002150279...|[8.48294361888286...|       1.0|
|  0.0|[76.0,3.0,1.0,25....|[-132.95231679351...|[0.85787336160797...|       0.0|
|  0.0|[84.0,1.0,1.0,28....|[-253.78757774233...|[1.43709636684123...|       1.0|
|  0.0|[91.0,3.0,1.0,29....|[-147.39325625202...|[0.90825770645019...|       0.0|
|  0.0|[92.0,3.0,1.0,20....|[-121.29364518267...|[0.88903015240791...|       0.0|
|  0.0|[93.0,1.0,1.0,46....|[-356.31591410791...|[1.01870392964541...|       1.0|
|  0.0|[97.0,1.0,1.0,28....|[-210.67419435925...|[4.11417341198092...|       1.0|
|  0.0|[101.0,3.0,0.0,28...|[-138.91189148128...|[0.85154456678425...|       0.0|
|  0.0|[102.0,3.0,1.0,28...|[-145.40546347315...|[0.94096734378075...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
only showing top 20 rows
```

We can also observe the prediction with label.

```
#extract prediction and ground truth label from test_results
PredicationAndLabel = predictions['prediction','label']
PredicationAndLabel.show(10)
```

```
+----------+-----+
|prediction|label|
+----------+-----+
|       1.0|  0.0|
|       0.0|  0.0|
|       1.0|  0.0|
|       1.0|  0.0|
|       1.0|  0.0|
|       0.0|  0.0|
|       0.0|  0.0|
|       1.0|  0.0|
|       0.0|  0.0|
|       1.0|  0.0|
+----------+-----+
only showing top 10 rows
```

Now, to evaluate the performance of a classification model, metrics such as precision, recall, accuracy, and a confusion matrix are a good option (“2.10”). The MulticlassMetrics class is imported from the pyspark.mllib.evaluation module, which is used for evaluating the performance of multiclass classification models. The weightedPrecision, weightedRecall, and accuracy properties of the MulticlassMetrics object are used to calculate the precision, recall, and accuracy scores, respectively. The confusionMatrix method of the MulticlassMetrics object is used to obtain the confusion matrix as a Spark Matrix object.

```
# import multiclassmetrics for precision, recall, and confusion matrix calculation
from pyspark.mllib.evaluation import MulticlassMetrics
multi_metrics = MulticlassMetrics(PredicationAndLabel.rdd)
precision_score = multi_metrics.weightedPrecision
recall_score = multi_metrics.weightedRecall
accuracy_score = multi_metrics.accuracy
print('Recall_score :', recall_score)
print('Precision_score :', precision_score)
print('Accuracy_score :', accuracy_score)
print('Confusion Matrix :' )
multi_metrics.confusionMatrix().toArray()
```

Here is the result of Naive Bayes model. The recall score is approximately 70.57% which means the model correctly identifies about 70.57% of the actual positive instances. The precision score is approximately 69.73% that indicates about 69.73% of the instances predicted as positive by the model are actually positive. the overall correctness of the model, considering both true positives and true negatives is approximately 70.57%. For the confusion metrics The model correctly predicted 138 instances as negative (True Negatives). It incorrectly predicted 29 instances as positive when they were actually negative (False Positives). It incorrectly predicted 49 instances as negative when they were actually positive (False Negatives). It correctly predicted 49 instances as positive (True Positives).

```
Recall_score : 0.7056603773584905
Precision_score : 0.6973763934918208
Accuracy_score : 0.7056603773584905

Confusion Matrix :array([[138.,  29.],
                         [ 49.,  49.]])
```

Decision Tree Classification (“2.11”)

DecisionTreeClassifier is from the pyspark.ml.classification and from that, dt is created which is used to train the model. The test_set is used to transform the model.

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier()

model = dt.fit(train_set)

# Make predictions.
predictions = model.transform(test_set)

predictions.show(2)
```

```
+-----+--------------------+-------------+--------------------+----------+
|label|            features|rawPrediction|         probability|prediction|
+-----+--------------------+-------------+--------------------+----------+
|  0.0|(8,[0,1,3,6],[178...|   [1.0,87.0]|[0.01136363636363...|       1.0|
|  0.0|[27.0,3.0,1.0,28....| [233.0,27.0]|[0.89615384615384...|       0.0|
+-----+--------------------+-------------+--------------------+----------+
only showing top 2 rows
```

The precision, recall, accuracy, and a confusion matrix are used here as well.

```
# extract prediction and ground truth label from test_results
PredicationAndLabel = predictions['prediction','label']
# import multiclassmetrics for precision, recall,
# and confusion matrix calculation
from pyspark.mllib.evaluation import MulticlassMetrics
multi_metrics = MulticlassMetrics(PredicationAndLabel.rdd)
precision_score = multi_metrics.weightedPrecision
recall_score = multi_metrics.weightedRecall
accuracy_score = multi_metrics.accuracy
print(recall_score)
print(precision_score)
print(accuracy_score)
multi_metrics.confusionMatrix().toArray()
```

With a recall score of roughly 76.97%, the model accurately detects roughly 76.97% of the real positive instances. The precision score is roughly 76.86%, meaning that roughly 76.86% of the occurrences that the model predicted to be positive are in fact positive. Taking into account both true positives and true negatives, the model's overall accuracy is roughly 76.97%. The model accurately predicted 150 instances as negative (True Negatives) for the confusion metrics. False Positives were the 17 occurrences that it mispredicted as positive when they were actually negative. False Negatives: 44 cases were wrongly predicted as negative when they were actually positive. 54 occurrences were accurately predicted as positive (True Positives).

```
Recall_score : 0.7698113207547169
Precision_score : 0.7685242362558937
Accuracy_score : 0.769811320754717

Confusion Matrix : array([[150.,  17.],
                          [ 44.,  54.]])
```

Random Forest Classification (“2.12”)

The procedure is the same as Naive Bayes and Decision tree classification.

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier()

model = rf.fit(train_set)
predictions = model.transform(test_set)

predictions.show(2)
```

```
# extract prediction and ground truth label from test_results
PredicationAndLabel = predictions['prediction','label']

# import multiclassmetrics for precision, recall, and confusion
from pyspark.mllib.evaluation import MulticlassMetrics
multi_metrics = MulticlassMetrics(PredicationAndLabel.rdd)
precision_score = multi_metrics.weightedPrecision
recall_score = multi_metrics.weightedRecall
accuracy_score = multi_metrics.accuracy
print('Recall_score :', recall_score)
print('Precision_score :', precision_score)
print('Accuracy_score :', accuracy_score)
print('Confusion Matrix :' )
multi_metrics.confusionMatrix().toArray()
```

The model accurately detects approximately 78.94% real positive instances, 79.40% of the occurrences that the model predicted to be positive are in fact positive, and the model's overall accuracy is roughly 78.94%. True negative: 146, False positive: 14, False negative: 42, and True positive: 14.

```
Recall_score : 0.7894736842105263
Precision_score : 0.7940965342981021
Accuracy_score : 0.7894736842105263
Confusion Matrix : array([[146.,  14.],
                          [ 42.,  64.]])
```

Gradient-boosted tree Classifier ("2.13")

The procedure is the same as Naive Bayes and Decision tree classification and Random Forest Classification.

```
from pyspark.ml.classification import GBTClassifier

gbt = GBTClassifier()
model = rf.fit(train_set)
predictions = model.transform(test_set)

predictions.show(2)
```

```
# extract prediction and ground truth label from test_results
PredicationAndLabel = predictions['prediction','label']

# import multiclassmetrics for precision, recall, and confusion
from pyspark.mllib.evaluation import MulticlassMetrics
multi_metrics = MulticlassMetrics(PredicationAndLabel.rdd)
precision_score = multi_metrics.weightedPrecision
recall_score = multi_metrics.weightedRecall
accuracy_score = multi_metrics.accuracy
print('Recall_score :', recall_score)
print('Precision_score :', precision_score)
print('Accuracy_score :', accuracy_score)
print('Confusion Matrix :' )
multi_metrics.confusionMatrix().toArray()
```

The model produced the same result as Random Forest Classification.

```
Recall_score : 0.7894736842105263
Precision_score : 0.7940965342981021
Accuracy_score : 0.7894736842105263
Confusion Matrix : array([[146.,  14.],
                          [ 42.,  64.]])
```

Multilayer Perceptron Classifier ("2.14")

In a multilayer perceptron classifier, we need to define the layers of the neural network. The layers parameter specifies the number of nodes in each layer of the neural network. In this case, I have an input layer with num_features nodes, followed by a hidden layer with 4 nodes, another hidden layer with 2 nodes, and finally an output layer with num_classes nodes. An instance of the MultilayerPerceptronClassifier class is created, specifying parameters like the maximum number of iterations (maxIter), the structure of the neural network (layers), the block size for parallel processing (blockSize), and a random seed for reproducibility (seed). Then train and transform the model.

```
from pyspark.ml.classification import MultilayerPerceptronClassifier

feat = ['PassengerId', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare', 'Embarked']

# Define the number of features
num_features = len(feat)
# Define the number of classes in the output layer (binary classification, so 2 classes)
num_classes = 2
layers = [num_features,4,2,num_classes]

trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)

model = trainer.fit(train_set)
predictions = model.transform(test_set)
```

```
# extract prediction and ground truth label from test_results
PredicationAndLabe4 = predictions['prediction','label']

# import multiclassmetrics for precision, recall, and confusion matrix
from pyspark.mllib.evaluation import MulticlassMetrics
multi_metrics = MulticlassMetrics(PredicationAndLabe4.rdd)
precision_score = multi_metrics.weightedPrecision
recall_score = multi_metrics.weightedRecall
accuracy_score = multi_metrics.accuracy
print('Recall_score :', recall_score)
print('Precision_score :', precision_score)
print('Accuracy_score :', accuracy_score)
print('Confusion Matrix :' )
multi_metrics.confusionMatrix().toArray()
```

The model accurately detects approximately 57.14% real positive instances, 44.10% of the occurrences that the model predicted to be positive are in fact positive, and the model's overall accuracy is roughly 57.14%. True negative: 149, False positive: 11, False negative: 103, and True positive: 3.

```
Recall_score : 0.5714285714285714
Precision_score : 0.4410430839002268
Accuracy_score : 0.5714285714285714
Confusion Matrix : array([[149.,  11.],
                          [103.,   3.]])
```

Results Analysis

Random Forest Classifier and Gradient-boosted tree Classifier appear to have the highest Recall, Precision, and Accuracy among the models. Decision Tree also performs well across all metrics. Naive Bayes shows a good balance between Recall, Precision, and Accuracy. The Multilayer Perceptron Classifier has lower Recall and Precision, indicating that it may not be performing as well in this particular context.

Discussion and conclusions

Looking at precision, recall, and accuracy, Random Forest Classifier or Gradient-boosted tree Classifier or Decision Tree should be the first choice. If simplicity and assumptions of independence align with the data, Naive Bayes could be considered.

References and Bibliography

"1.1." *Apache Spark*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.SparkConf.html.

"1.2." *matplotlib.pyplot*, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html.

"1.3." *matplotlib.pyplot.pie*, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.pie.html.

"1.4." *matplotlib.pyplot.bar*, https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html.

"2.1." *pandas*, https://pandas.pydata.org/.

"2.2." *NumPy*, https://numpy.org/.

"2.3." *Seaborn*, https://seaborn.pydata.org/.

"2.4." *Missingo*, https://github.com/ResidentMario/missingno.

"2.5." *sklearn.preprocessing.LabelEncoder*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html.

"2.6." *VectorAssembler*, https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html.

"2.7." *StringIndexer*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html.

"2.8." *Pipeline*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.Pipeline.html.

"2.9." *NaiveBayes*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.NaiveBayes.html.

"2.10." *MultiClassMetrics*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.mllib.evaluation.MulticlassMetrics.html.

"2.11." *Decision Tree Classifier*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.DecisionTreeClassifier.html.

"2.12." *Random Forest Classifier*, https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.classification.RandomForestClassifier.html.

"2.13." *Gradient-boosted tree Classifier*, https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.classification.GBTClassifier.html

"2.14." *Multilayer Perceptron Classifier*, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.MultilayerPerceptronClassifier.html.

Other Resources

Appendix (if necessary)