Table of Contents

Checklist:

- Linear Regression
- Multi Linear Regression
- Logistic regression
- Cluster
- Decision Tree
- ANN

https://academy.databricks.com/instructor-led-training/scalable-machine-learning-with-apache-spark

1. Tools required.

You need to have the following tools, which can be installed as shown below:

Installation of Pip if not present:
python3 -m ensurepip --default-pip

Install pandas.
!pip install pandas
!pip install seaborn

!pip install scikit-learn matplotlib

2. Data Exploration – 60 Minutes

Notebook -> Data Exploration

You can download the data file from the following URL or get from the instructor specified /Software folder.

https://www.kaggle.com/c/santander-customer-transaction-prediction/data?select=train.csv

In this lab we will understand various data exploration commands provided in the pyspark environment.

Basic overview.
There are two kinds of variables, continuous and categorical. Each of them has different EDA requirements:
Continuous variables EDA list:

- missing values
- statistic values: mean, min, max, stddev, quantiles
- binning & distribution
- correlation

Categorical variables EDA list:

- missing values

- frequency table

Now first, Let's load the data. The data I used is from a Kaggle competition, Santander Customer Transaction Prediction.

Write a description of data here.

```
df = (spark
  .read
  .option("inferSchema","true")
  .option("header","true")
  .csv("/Software/data/train.csv"))
```

EDA for continuous variables

The built-in function describe() is extremely helpful. It computes count, mean, stddev, min and max for the selected variables. For example:

```
df.select('var_0').describe().show()
```

```
In [2]: df.select('var_0').describe().show()
[Stage 2:>                                                          (0 + 6) / 6]
+-------+-----------------+
|summary|            var_0|
+-------+-----------------+
|  count|           200000|
|   mean|10.679914251999964|
| stddev|3.0400508706688214|
|    min|           0.4084|
|    max|           20.315|
+-------+-----------------+
```

Here, the describe() function which is built in the spark data frame has done the statistic values calculation. The computed summary table is not large in size. So, we can use pandas to display it.

```
df.select('var_0','var_1','var_2','var_3','var_4','var_5','var_6','var_7','var_8','var_9','var_10','var_11','var_12','var_13','var_14').describe().toPandas()
```

Get the quantiles:

```
quantile = df.approxQuantile(['var_0'], [0.25, 0.5, 0.75], 0)
quantile_25 = quantile[0][0]
```

```
quantile_50 = quantile[0][1]
quantile_75 = quantile[0][2]
print('quantile_25: '+str(quantile_25))
print('quantile_50: '+str(quantile_50))
print('quantile_75: '+str(quantile_75))
```

In [3]:
```
df.select('var_0','var_1','var_2','var_3','var_4','var_5','var_6','var_7','var_8','var_9','var_10','var_11','var_12
```

Out[3]:

	summary	var_0	var_1	var_2	var_3	var_4	var_5	var_6
0	count	200000	200000	200000	200000	200000	200000	200000
1	mean	10.679914251999964	-1.6276216895000084	10.715191850999998	6.796529156999989	11.078333240499985	-5.065317493500007	5.408948681499987
2	stddev	3.0400508706688214	4.050044189955001	2.6408941917999007	2.043319016359727	1.623149533936836	7.8632666834767235	0.8666072662169024
3	min	0.4084	-15.0434	2.1171	-0.0402	5.0748	-32.5626	2.3473
4	max	20.315	10.3768	19.353	13.1883	16.6714	17.2516	8.4477

In [4]:
```
quantile = df.approxQuantile(['var_0'], [0.25, 0.5, 0.75], 0)
quantile_25 = quantile[0][0]
quantile_50 = quantile[0][1]
quantile_75 = quantile[0][2]
print('quantile_25: '+str(quantile_25))
print('quantile_50: '+str(quantile_50))
print('quantile_75: '+str(quantile_75))
```

```
[Stage 9:>                                                          (0 + 2) / 2]
```

```
quantile_25: 8.4537
quantile_50: 10.5247
quantile_75: 12.7582
```

Check the missings:

Introduce two functions to do the filter

```
# where
df.where(df.var_0.isNull()).count()
# filter
df.filter(df['var_0'].isNull()).count()
```
These two are the same. According to spark documentation, “where” is an alias of “filter”.

Binning:

For continuous variables, sometimes we want to bin them and check those bins distribution. For example, in financial related data, we can bin FICO scores(normally range 650 to 850) into buckets. Each bucket has an interval of 25. like 650–675, 675–700, 700–725,…And check how many people in each bucket.

Now let’s use “var_0” to give an example for binning. From previous statistic values, we know “var_0” range from 0.41 to 20.31. So we create a list of 0 to 21, with an interval of 0.5.

```
from pyspark.ml.feature import Bucketizer
from pyspark.sql.functions import udf
import numpy as np
```

```
from pyspark.sql.types import *


var = "var_0"
# create the split list ranging from 0 to  21, interval of 0.5
split_list = [float(i) for i in np.arange(0,21,0.5)]
# initialize buketizer
bucketizer = Bucketizer(splits=split_list,inputCol=var, outputCol="buckets")



# transform
df_buck = bucketizer.setHandleInvalid("keep").transform(df.select(var).dropna())
df_buck.show()
```

In [5]:
```
# where
df.where(df.var_0.isNull()).count()
# filter
df.filter(df['var_0'].isNull()).count()
```

Out[5]:
```
0
```

In [6]:
```
from pyspark.ml.feature import Bucketizer
from pyspark.sql.functions import udf
import numpy as np
from pyspark.sql.types import *
```

In [7]:
```
var = "var_0"
# create the split list ranging from 0 to  21, interval of 0.5
split_list = [float(i) for i in np.arange(0,21,0.5)]
# initialize buketizer
bucketizer = Bucketizer(splits=split_list,inputCol=var, outputCol="buckets")
```

In [8]:
```
# transform
df_buck = bucketizer.setHandleInvalid("keep").transform(df.select(var).dropna())
```

In [14]:
```
df_buck.show()
```

```
+-------+-------+
|  var_0|buckets|
+-------+-------+
| 8.9255|   17.0|
|11.5006|   23.0|
| 8.6093|   17.0|
|11.0604|   22.0|
```

the "buckets" column gives the bucket rank, not the actual bucket value(range),
use dictionary to match bucket rank and bucket value

```
bucket_names = dict(zip([float(i) for i in range(len(split_list[1:]))],split_list[1:]))


# user defined function to update the data frame with the bucket value
udf_foo = udf(lambda x: bucket_names[x], DoubleType())


bins = df_buck.withColumn("bins",
udf_foo("buckets")).groupBy("bins").count().sort("bins").toPandas()
bins
```

```
In [9]: # the "buckets" column gives the bucket rank, not the acctual bucket value(range),
        # use dictionary to match bucket rank and bu
        bucket_names = dict(zip([float(i) for i in range(len(split_list[1:]))],split_list[1:]))

In [10]: # user defined function to update the data frame with the bucket value
         udf_foo = udf(lambda x: bucket_names[x], DoubleType())

In [11]: bins = df_buck.withColumn("bins", udf_foo("buckets")).groupBy("bins").count().sort("bins").toPandas()

In [12]: bins
```

Reload this page

Out[12]:

	bins	count
0	0.5	2
1	1.0	3
2	1.5	13
3	2.0	27
4	2.5	75
5	3.0	143

Correlation:

```
from pyspark.mllib.stat import Statistics

# select variables to check correlation
```

```
df_features = df.select("var_0","var_1","var_2","var_3")

# create RDD table for correlation calculation
rdd_table = df_features.rdd.map(lambda row: row[0:])

# get the correlation matrix
corr_mat=Statistics.corr(rdd_table, method="pearson")
```

EDA for categorical variables

To check missing values, it's the same as continuous variables.

To check the frequency table:

```
freq_table = df.select(df["target"].cast("string")).groupBy("target").count().toPandas()
freq_table
```

In [18]:
```
from pyspark.mllib.stat import Statistics

# select variables to check correlation
df_features = df.select("var_0","var_1","var_2","var_3")
```

Reload this page

In [19]:
```
# create RDD table for correlation calculation
rdd_table = df_features.rdd.map(lambda row: row[0:])

# get the correlation matrix
corr_mat=Statistics.corr(rdd_table, method="pearson")
```

```
[Stage 29:>                                                         (0 + 6) / 6]
```

```
23/07/14 09:04:22 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIB
LAS
23/07/14 09:04:22 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.Fore
ignLinkerBLAS
```

In [20]:
```
freq_table = df.select(df["target"].cast("string")).groupBy("target").count().toPandas()
```

In [21]:
```
freq_table
```

Out[21]:

	target	count
0	0	179902
1	1	20098

Visualization:

Before creating visualizations we need to create a table of the dataset using SQL, in order to learn more about SQL operations using PySpark

Creating a Table using SQL and running required queries

```
#Creating Table
df.createOrReplaceTempView('CustTable')
```

In this tutorial, we'll use several different libraries to help us visualize the dataset. To do this analysis, import the following libraries:

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

By using this query, we want to understand how the average tip amounts have changed over the period we've selected. This query will also help us identify other useful insights, including the minimum/maximum tip amount per day and the average fare amount.

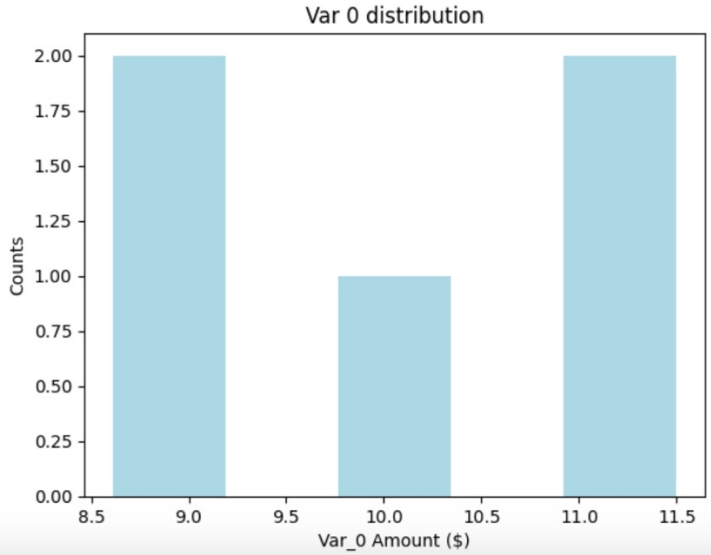
```
#Running Query
df1 = sqlContext.sql("SELECT * from CustTable limit 5").toPandas()
```

We want to understand the distribution of tips in our dataset. We'll use Matplotlib to create a histogram that shows the distribution of tip amount and count. Based on the distribution, we can see that tips are skewed toward amounts less than or equal to $10.

Look at a histogram of var_0 by count by using Matplotlib

```
ax1 = df1['var_0'].plot(kind='hist', bins=5, facecolor='lightblue')
ax1.set_title('Var 0 distribution')
ax1.set_xlabel('Var_0 Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()
```

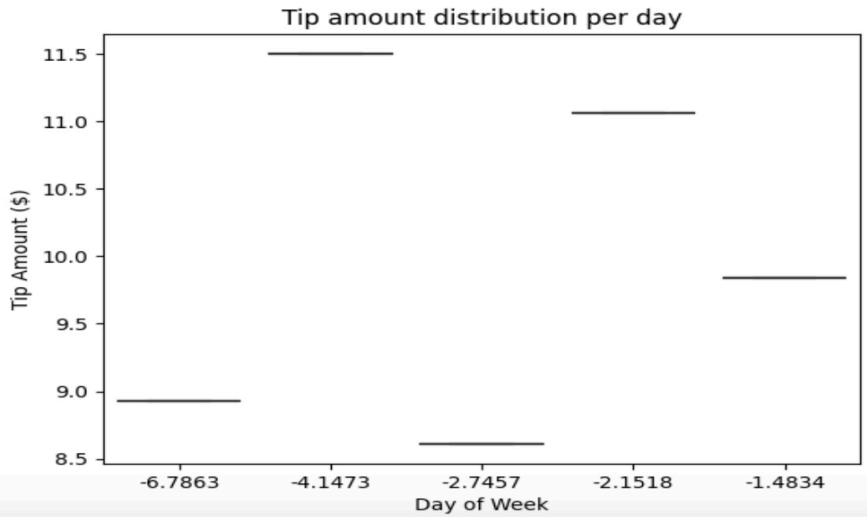
In [26]:
```
ax1 = df1['var_0'].plot(kind='hist', bins=5, facecolor='lightblue')
ax1.set_title('Var 0 distribution')
ax1.set_xlabel('Var_0 Amount ($)')
ax1.set_ylabel('Counts')
plt.suptitle('')
plt.show()
```



Next, we want to understand the relationship between the tips for a given trip and the day of the week. Use Seaborn to create a box plot that summarizes the trends for each day of the week.

```
# View the distribution of tips by day of week using Seaborn
ax = sns.boxplot(x="var_1", y="var_0",data=df1, showfliers = False)
ax.set_title('Tip amount distribution per day')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Tip Amount ($)')
plt.show()
```

```
In [27]: # View the distribution of tips by day of week using Seaborn
ax = sns.boxplot(x="var_1", y="var_0",data=df1, showfliers = False)
ax.set_title('Tip amount distribution per day')
ax.set_xlabel('Day of Week')
ax.set_ylabel('Tip Amount ($)')
plt.show()
```

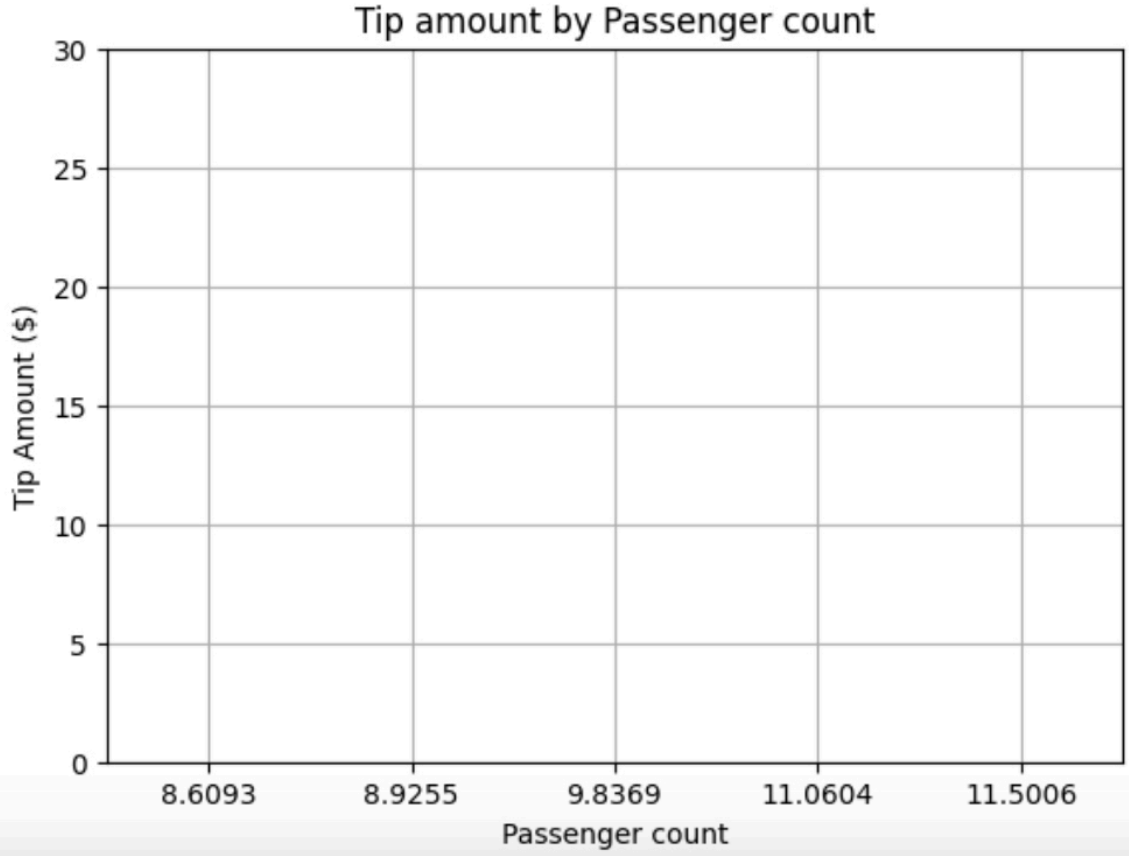


Another hypothesis of ours might be that there's a positive relationship between the number of passengers and the total taxi tip amount. To verify this relationship, run the following code to generate a box plot that illustrates the distribution of tips for each passenger count.

```
# How many passengers tipped by various amounts
ax2 = df1.boxplot(column=['var_1'], by=['var_0'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
```

```
ax2.set_ylabel('Tip Amount ($)')
ax2.set_ylim(0,30)
plt.suptitle('')
plt.show()
```

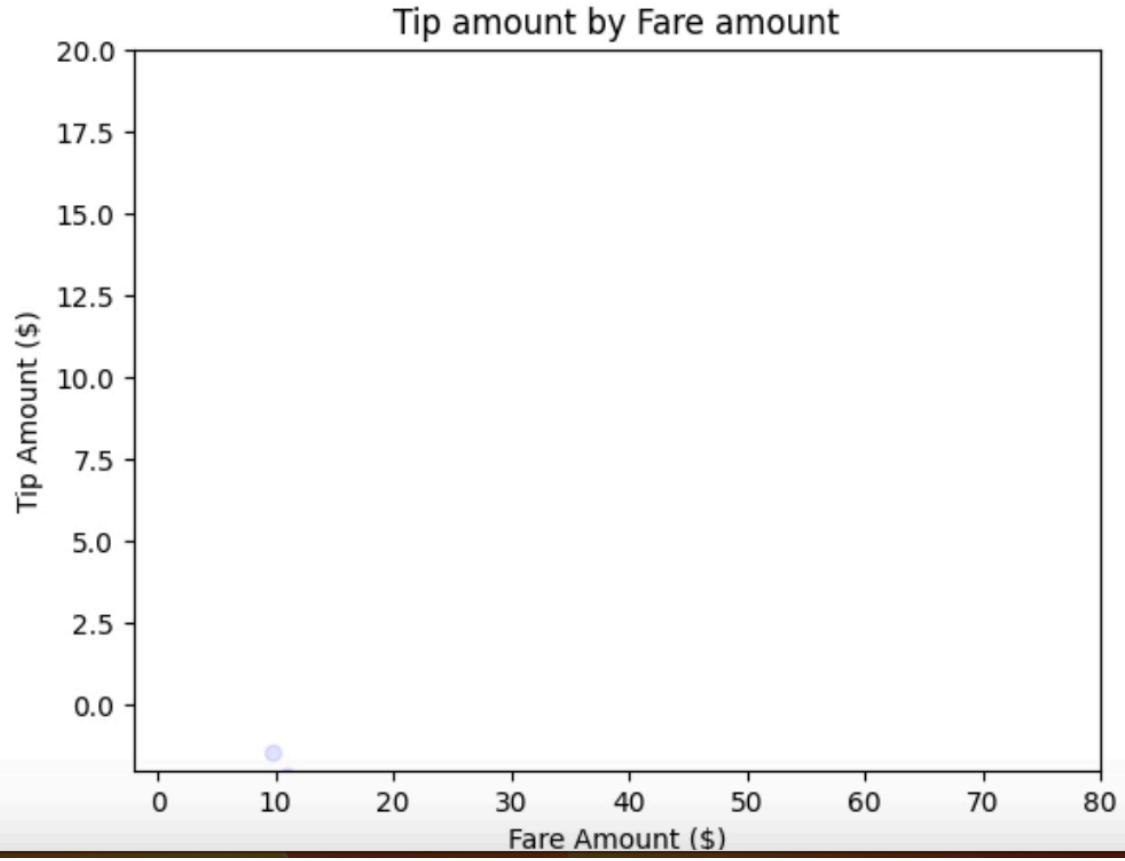
In [28]:
```
# How many passengers tipped by various amounts
ax2 = df1.boxplot(column=['var_1'], by=['var_0'])
ax2.set_title('Tip amount by Passenger count')
ax2.set_xlabel('Passenger count')
ax2.set_ylabel('Tip Amount ($)')
ax2.set_ylim(0,30)
plt.suptitle('')
plt.show()
```



Last, we want to understand the relationship between the fare amount and the tip amount. Based on the results, we can see that there are several observations where people don't tip. However, we also see a positive relationship between the overall fare and tip amounts.

```
# Look at the relationship between fare and tip amounts

ax = df1.plot(kind='scatter', x= 'var_0', y = 'var_1', c='blue', alpha = 0.10, s=2.5*(df1['var_2']))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.suptitle('')
plt.show()
```

```
ax = df1.plot(kind='scatter', x= 'var_0', y = 'var_1', c='blue', alpha = 0.10, s=2.5*(df1['var_2']))
ax.set_title('Tip amount by Fare amount')
ax.set_xlabel('Fare Amount ($)')
ax.set_ylabel('Tip Amount ($)')
plt.axis([-2, 80, -2, 20])
plt.suptitle('')
plt.show()
```



Reference:
https://towardsdatascience.com/exploratory-data-analysis-eda-with-pyspark-on-databricks-e8d6529626b1
https://towardsdatascience.com/complete-introduction-to-pyspark-part-4-62a99ce3552a
https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-data-visualization-tutorial

----------Lab Ends Here ----------

3. Understanding Regression

In this lab, we will use the Air bnb data to estimate the price of the room depending on the number of room using Linear Regression model.

Notebook: ML Regression

Import the necessary package to convert a column to vector

```
from pyspark.ml.feature import VectorAssembler

spark = SparkSession.builder \
        .master('local') \
        .appName('Air BnB Price') \
        .getOrCreate()

# Load dataset and observe a sample data format.

filePath = '/Software/data/sfb/sf-abnb-listings.csv'
airbnbDF = spark.read.option("delimiter", ",").option("header", True).csv(filePath)
# See some columns
airbnbDF.select("id", 'room_type', 'bedrooms', 'bathrooms', 'number_of_reviews', 'price')\
        .show(5)
```

```
In [10]: # Load dataset

filePath = '/Software/data/sfb/sf-abnb-listings.csv'
#airbnbDF = spark.read.parquet(filePath)
#airbnbDF = spark.read.csv(filePath)
airbnbDF = spark.read.option("delimiter", ",").option("header", True).csv(filePath)
# See some columns
airbnbDF.select("id", 'room_type', 'bedrooms', 'bathrooms', 'number_of_reviews', 'price')\
        .show(5)
```

```
+--------------------+--------------------+--------------+--------------------+--------------------+--------
------------+
|                  id|           room_type|      bedrooms|           bathrooms|   number_of_reviews|
price|
+--------------------+--------------------+--------------+--------------------+--------------------+--------
------------+
|              138592|                null|          null|                null|                null|
null|
|We love the idea ...|                null|          null|                null|                null|
null|
|",within an hour,...| ""Fire extinguis...| ""Rice maker""| ""Private entran...| ""Books and read...| ""Free
washer \u...|
|              474107|        Private room|          null|                null|         ""Blender""|
""Coffee""|
|              487019|    Entire home/apt|          null|                null| ""Clothing stora...|    ""Di
ning table""|
+--------------------+--------------------+--------------+--------------------+--------------------+--------
------------+
only showing top 5 rows
```

You can view the schema of the dataframe.

airbnbDF.printSchema()

In [11]:

```
airbnbDF.printSchema()
```

```
root
 |-- id: string (nullable = true)
 |-- listing_url: string (nullable = true)
 |-- scrape_id: string (nullable = true)
 |-- last_scraped: string (nullable = true)
 |-- source: string (nullable = true)
 |-- name: string (nullable = true)
 |-- description: string (nullable = true)
 |-- neighborhood_overview: string (nullable = true)
 |-- picture_url: string (nullable = true)
 |-- host_id: string (nullable = true)
 |-- host_url: string (nullable = true)
 |-- host_name: string (nullable = true)
 |-- host_since: string (nullable = true)
 |-- host_location: string (nullable = true)
 |-- host_about: string (nullable = true)
 |-- host_response_time: string (nullable = true)
 |-- host_response_rate: string (nullable = true)
 |-- host_acceptance_rate: string (nullable = true)
 |-- host_is_superhost: string (nullable = true)
```

As observe above, the bedroom is a string column, which we will convert into double in the further step.
Let us remove the dataset in which bedrooms and price field are Null.

```
# Filter rows that has null value on column
from pyspark.sql.functions import col
airbnFbDF = airbnbDF.filter(col("bedrooms").isNotNull()).filter(col("price").isNotNull())

# Filter rows that has numeric value on column
airbnNbDF = airbnFbDF.filter(col("bedrooms")
```

```
.cast("int").isNotNull()).filter(col("price")
.cast("int").isNotNull())
```

You can cross verify the conversion data.

```
airbnNbDF.select("bedrooms","price").show(10)
```

```
In [49]: airbnNbDF.select("bedrooms","price").show(10)
+--------+-----+
|bedrooms|price|
+--------+-----+
|      30| 30.0|
|      30|   60|
|      32|  120|
|      32|  120|
|     352|    0|
|      90|   90|
|      45| 45.0|
|       1|    1|
|       3|    3|
|   120.0|   30|
+--------+-----+
only showing top 10 rows
```

Since the regression model requires numeric parameter, use the cast column to convert into numeric data type.

```
# Cast bedrooms from string type to double type
airbnNbDF = airbnNbDF.withColumn("bedrooms",
```

```
airbnNbDF["bedrooms"]
.cast('double')).withColumn("price",
airbnNbDF["price"]
.cast('double'))
```

Verify the data types of bedrooms and Price → It should be numeric.

```
airbnNbDF.printSchema()
```

```
In [55]: # Cast bedrooms from string type to double type
         airbnNbDF = airbnNbDF.withColumn("bedrooms",
                                          airbnNbDF["bedrooms"]
                                          .cast('double')).withColumn("price",
                                          airbnNbDF["price"]
                                          .cast('double'))
         airbnNbDF.printSchema()
```

```
|-- host_listings_count: string (nullable = true)
|-- host_total_listings_count: string (nullable = true)
|-- host_verifications: string (nullable = true)
|-- host_has_profile_pic: string (nullable = true)
|-- host_identity_verified: string (nullable = true)
|-- neighbourhood: string (nullable = true)
|-- neighbourhood_cleansed: string (nullable = true)
|-- neighbourhood_group_cleansed: string (nullable = true)
|-- latitude: string (nullable = true)
|-- longitude: string (nullable = true)
|-- property_type: string (nullable = true)
|-- room_type: string (nullable = true)
|-- accommodates: string (nullable = true)
|-- bathrooms: string (nullable = true)
|-- bathrooms_text: string (nullable = true)
|-- bedrooms: double (nullable = true)
|-- beds: string (nullable = true)
|-- amenities: string (nullable = true)
|-- price: double (nullable = true)
|-- minimum_nights: string (nullable = true)
|-- maximum_nights: string (nullable = true)
```

```
# Creating Training and Test Data sets
trainDF, testDF = airbnNbDF.randomSplit([0.8, 0.2], seed = 42)

print(f'There are {trainDF.count()} rows in the training set, and {testDF.count()} in the test set')


# Preparing features with transformers

vecAssembler = VectorAssembler(inputCols=['bedrooms'], outputCol='features')

# To see the result of vector assembler
vecTrainDF = vecAssembler.transform(trainDF)
vecTrainDF.select('bedrooms', 'beds', 'features', 'price').show(5)
```

In [57]:
```
# Preparing features with transformers

vecAssembler = VectorAssembler(inputCols=['bedrooms'], outputCol='features')

# To see the result of vector assembler
vecTrainDF = vecAssembler.transform(trainDF)
vecTrainDF.select('bedrooms', 'beds', 'features', 'price').show(5)
```

```
+--------+----+--------+-----+
|bedrooms|beds|features|price|
+--------+----+--------+-----+
|    30.0|1100|  [30.0]| 30.0|
|    90.0|  32|  [90.0]| 90.0|
|    32.0|  32|  [32.0]|120.0|
|    31.0|1125|  [31.0]| 31.0|
|  1125.0|null|[1125.0]|  0.0|
+--------+----+--------+-----+
only showing top 5 rows
```

Linear Regression with MLlib

```
from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol='features', labelCol='price')
```

Pipeline

```
from pyspark.ml import Pipeline

pipeline = Pipeline(stages=[vecAssembler, lr])
```

```
pipelineModel = pipeline.fit(trainDF)
```

```
predDF = pipelineModel.transform(testDF)
predDF.select('features', 'price', 'prediction').show(10)
```

```
In [59]: predDF = pipelineModel.transform(testDF)
         predDF.select('features', 'price', 'prediction').show(10)
```

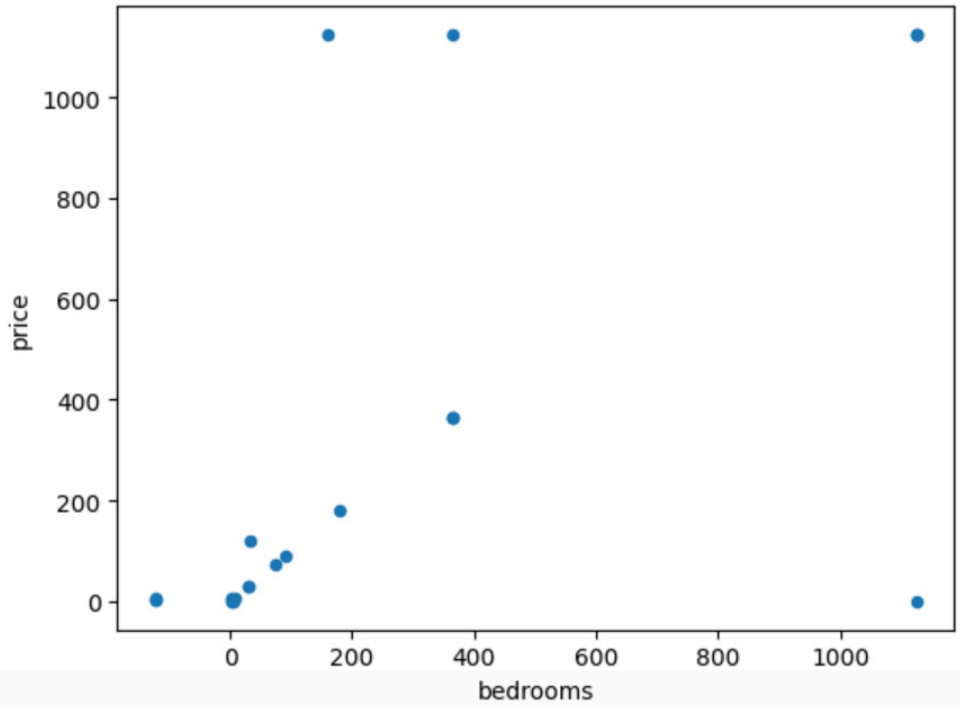
```
+------------+------+------------------+
|    features| price|        prediction|
+------------+------+------------------+
|       [32.0]| 120.0|172.12271415141336|
|       [30.0]|  30.0| 170.4987094451791|
|     [1125.0]|1125.0| 1059.641286108438|
|       [30.0]|  30.0| 170.4987094451791|
|        [3.0]|   3.0|148.57464591101655|
|[-122.43639]|   2.0|46.720002064498345|
|       [75.0]|  75.0|   207.03881533545|
|      [180.0]| 180.0|292.29906241274875|
|        [6.0]|   0.0|151.01065297036794|
|     [1125.0]|   0.0| 1059.641286108438|
+------------+------+------------------+
only showing top 10 rows
```

```
# Evaluate our model
from pyspark.ml.evaluation import RegressionEvaluator

regressionEvaluator = RegressionEvaluator(
                predictionCol = 'prediction',
```

```
                    labelCol = 'price',
                    metricName = 'rmse'
                    )

rmse = regressionEvaluator.evaluate(predDF)


print(f'RMSE is {rmse}')




slope = pipelineModel.stages[1].coefficients
intercept = pipelineModel.stages[1].intercept

print(slope, intercept)



bed_price_df = testDF.select('bedrooms', 'price').toPandas()
bed_price_df.plot.scatter('bedrooms', 'price')
```

In [62]:
```
bed_price_df = testDF.select('bedrooms', 'price').toPandas()
bed_price_df.plot.scatter('bedrooms', 'price')
```

Out[62]: <Axes: xlabel='bedrooms', ylabel='price'>



airbnbDF.select('price').summary().show()

In [63]:
```
airbnbDF.select('price').summary().show()
```

```
+-------+------------------+
|summary|             price|
+-------+------------------+
|  count|              8007|
|   mean| 238.74177381731317|
| stddev| 396.42609642996314|
|    min| ""3 in 1 - Trade...|
|    25%|               3.0|
|    50%|              30.0|
|    75%|             365.0|
|    max|    within an hour|
+-------+------------------+
```

---------- Lab Ends Here ----------

https://github.com/sniperhuni/bigdatastudy/blob/master/airBnBRentalPrice_school.ipynb

4. The classification – Logistic Regression

Notebook: Classification – Logistic Regression
& Explore ML functions

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Logistic Regression Assumptions

- Binary logistic regression requires the dependent variable to be binary.
- For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.
- Only the meaningful variables should be included.
- The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.
- The independent variables are linearly related to the log odds.
- Logistic regression requires quite large sample sizes.

Keeping the above assumptions in mind, let's look at our dataset.

Data Set Information:

This research aimed at the case of customers default payments in Taiwan and compares the predictive accuracy of probability of default among six data mining methods. From the perspective of risk management, the result of predictive accuracy of the estimated probability of default will be more valuable than the binary result of classification - credible or not credible clients. Because the real probability of default is unknown, this study presented the novel Sorting Smoothing Method to estimate the real probability of default. With the real probability of default as the response variable (Y), and the predictive probability of default as the independent variable (X), the simple linear regression result (Y = A + BX) shows that the forecasting model produced by artificial neural network has the highest coefficient of determination; its regression intercept (A) is close to zero, and regression coefficient (B) to one. Therefore, among the six data mining techniques, artificial neural network is the only one that can accurately estimate the real probability of default.

Attribute Information:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:
X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
X2: Gender (1 = male; 2 = female).
X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

X4: Marital status (1 = married; 2 = single; 3 = others).
X5: Age (year).
X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

Type the following commands in the notebook.

Machine Learning Steps Explained Using Credit Card data set

An example Spark SQL ML project that predicts the probability of a customer defaulting on their credit card bill next month using a Logistic Regression estimator and a 30,000 sample dataset of credit card customers.

Start the pyspark console.

```
%pyspark

# File location and type
file_location = "/Software/data/credit_card_default.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)



df.show()
```

In [23]:
```
# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)
```

In [24]:
```
df.show()
```

```
| 13|   630000|  2|       2|      2| 41|   -1|    0|   -1|   -1|   -1|   -1|    12137|     6500|     6500|     6
500|     6500|     2870|    1000|    6500|    6500|    6500|    2870|       0|                                    0|
| 14|    70000|  1|       2|      2| 30|    1|    2|    2|    0|    0|    2|    65802|    67369|    65701|    66
782|    36137|    36894|    3200|       0|    3000|    3000|    1500|       0|                                    1|
| 15|   250000|  1|       1|      2| 29|    0|    0|    0|    0|    0|    0|    70887|    67060|    63561|    59
696|    56875|    55512|    3000|    3000|    3000|    3000|    3000|    3000|                                    0|
| 16|    50000|  2|       3|      3| 23|    1|    2|    0|    0|    0|    0|    50614|    29173|    28116|    28
771|    29531|    30211|       0|    1500|    1100|    1200|    1300|    1100|                                    0|
| 17|    20000|  1|       1|      2| 24|    0|    0|    2|    2|    2|    2|    15376|    18010|    17428|    18
338|    17905|    19104|    3200|       0|    1500|       0|    1650|       0|                                    1|
| 18|   320000|  1|       1|      1| 49|    0|    0|    0|   -1|   -1|   -1|   253286|   246536|   194663|    70
074|     5856|   195599|   10358|   10000|   75940|   20000|  195599|   50000|                                    0|
| 19|   360000|  2|       1|      1| 49|    1|   -2|   -2|   -2|   -2|   -2|        0|        0|        0|
0|        0|        0|        0|       0|       0|       0|       0|       0|                                    0|
```

# Create a view or table
temp_table_name = "creditdata_csv"
df.createOrReplaceTempView(temp_table_name)

sql("describe table creditdata_csv").show()

In [26]:
```
sql("describe table creditdata_csv").show()
```

```
+---------+---------+-------+
| col_name|data_type|comment|
+---------+---------+-------+
|       ID|      int|   null|
|LIMIT_BAL|      int|   null|
|      SEX|      int|   null|
|EDUCATION|      int|   null|
| MARRIAGE|      int|   null|
|      AGE|      int|   null|
|    PAY_0|      int|   null|
|    PAY_2|      int|   null|
|    PAY_3|      int|   null|
|    PAY_4|      int|   null|
|    PAY_5|      int|   null|
|    PAY_6|      int|   null|
|BILL_AMT1|      int|   null|
|BILL_AMT2|      int|   null|
|BILL_AMT3|      int|   null|
|BILL_AMT4|      int|   null|
|BILL_AMT5|      int|   null|
|BILL_AMT6|      int|   null|
| PAY_AMT1|      int|   null|
| PAY_AMT2|      int|   null|
+---------+---------+-------+
only showing top 20 rows
```

#Query the created temp table in a SQL cell
sql("select * from `creditdata_csv`").show()

In [27]:
```
sql("select * from `creditdata_csv`").show()
```

```
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+-----
---+---------+---------+---------+---------+---------+---------+---------+--------------------------+
| ID|LIMIT_BAL|SEX|EDUCATION|MARRIAGE|AGE|PAY_0|PAY_2|PAY_3|PAY_4|PAY_5|PAY_6|BILL_AMT1|BILL_AMT2|BILL_AMT3|BILL_A
MT4|BILL_AMT5|BILL_AMT6|PAY_AMT1|PAY_AMT2|PAY_AMT3|PAY_AMT4|PAY_AMT5|PAY_AMT6|default_payment_next_month|
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+-----
---+---------+---------+---------+---------+---------+---------+---------+--------------------------+
|  1|    20000|  2|        2|       1| 24|    2|    2|   -1|   -1|   -2|   -2|     3913|     3102|      689|
0|        0|        0|        0|     689|        0|        0|        0|        0|                         1|
|  2|   120000|  2|        2|       2| 26|   -1|    2|    0|    0|    0|    2|     2682|     1725|     2682|     3
272|     3455|     3261|        0|    1000|    1000|    1000|        0|    2000|                         1|
|  3|    90000|  2|        2|       2| 34|    0|    0|    0|    0|    0|    0|    29239|    14027|    13559|    14
331|    14948|    15549|    1518|    1500|    1000|    1000|    1000|    5000|                         0|
|  4|    50000|  2|        2|       1| 37|    0|    0|    0|    0|    0|    0|    46990|    48233|    49291|    28
314|    28959|    29547|    2000|    2019|    1200|    1100|    1069|    1000|                         0|
|  5|    50000|  1|        2|       1| 57|   -1|    0|   -1|    0|    0|    0|     8617|     5670|    35835|    20
940|    19146|    19131|    2000|   36681|   10000|    9000|     689|     679|                         0|
|  6|    50000|  1|        1|       2| 37|    0|    0|    0|    0|    0|    0|    64400|    57069|    5760
394|    19619|    20024|    2500|    1815|     657|    1000|    1000|     800|                         0|
|  7|   500000|  1|        1|       2| 29|    0|    0|    0|    0|    0|    0|   367965|   412023|   445007|   542
653|   483003|   473944|   55000|   40000|   38000|   20239|   13750|   13770|                         0|
|  8|   100000|  2|        2|       2| 23|    0|   -1|   -1|    0|    0|   -1|    11876|      380|      601|
221|     -159|      567|     380|     601|       0|     581|    1687|    1542|                         0|
|  9|   140000|  2|        3|       1| 28|    0|    0|    2|    0|    0|    0|    11285|    14096|    12108|    12
211|    11793|     3719|    3329|       0|     432|    1000|    1000|    1000|                         0|
| 10|    20000|  1|        3|       2| 35|   -2|   -2|   -2|   -2|   -1|   -1|        0|        0|        0|
0|    13007|    13912|        0|       0|       0|   13007|    1122|       0|                         0|
| 11|   200000|  2|        3|       2| 34|    0|    0|    2|    0|    0|   -1|    11073|     9787|     5535|     2
513|     1828|     3731|    2306|      12|      50|     300|    3738|      66|                         0|
```

run cell, selec

sql("""select education,default_payment_next_month,count(*) from `creditdata_csv` \
group by education,default_payment_next_month \

```
order by count(*) desc""").show()
```

Different ways to deal with missing values:
##Delete the rows which are having the missing values.
##If the variables are continuous replace all the missing values with the mean or median of the attribute.
##If the variable in categorical replace with the most common value of that variable
##For the categorical variables replace each value of the variable and then perform clustering for best analysis of data.

```
sql("select default_payment_next_month,count(*) from creditdata_csv \
group by default_payment_next_month").show()
```

In [29]:
```
sql("""select education,default_payment_next_month,count(*) from `creditdata_csv` \
group by education,default_payment_next_month  \
order by count(*) desc""").show()
```

```
+---------+-------------------------+--------+
|education|default_payment_next_month|count(1)|
+---------+-------------------------+--------+
|        2|                        0|   10700|
|        1|                        0|    8549|
|        3|                        0|    3680|
|        2|                        1|    3330|
|        1|                        1|    2036|
|        3|                        1|    1237|
|        5|                        0|     262|
|        4|                        0|     116|
|        6|                        0|      43|
|        5|                        1|      18|
|        0|                        0|      14|
|        6|                        1|       8|
|        4|                        1|       7|
+---------+-------------------------+--------+
```

In [32]:
```
sql("select default_payment_next_month,count(*) from creditdata_csv \
group by default_payment_next_month").show()
```

```
+-------------------------+--------+
|default_payment_next_month|count(1)|
+-------------------------+--------+
|                        1|    6636|
|                        0|   23364|
+-------------------------+--------+
```

sql("select sex,count(*) from creditdata_csv \
group by sex ").show()

sql("select education,count(*) from creditdata_csv \
group by education").show()

In [33]:
```
sql("select sex,count(*) from creditdata_csv \
group by sex ").show()
```

```
+---+--------+
|sex|count(1)|
+---+--------+
|  1|   11888|
|  2|   18112|
+---+--------+
```

In [34]:
```
sql("select education,count(*) from creditdata_csv \
group by education").show()
```

```
+---------+--------+
|education|count(1)|
+---------+--------+
|        1|   10585|
|        6|      51|
|        3|    4917|
|        5|     280|
|        4|     123|
|        2|   14030|
|        0|      14|
+---------+--------+
```

```
sql("DROP VIEW IF EXISTS creditdata_final;")

sql("CREATE TEMPORARY VIEW creditdata_final as \
select *,CASE WHEN default_payment_next_month = 1  THEN 'Yes' ELSE 'No' END default_text
from creditdata_csv \
where education  not in ('0')  or education not like 'other%'" )


from pyspark.ml.regression import LinearRegression


spark.conf.set("spark.sql.execution.arrow.enabled", "true")
data = spark.sql("SELECT * from creditdata_final").show()
```

In [39]:
```
from pyspark.ml.regression import LinearRegression
```

In [41]:
```
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
data = spark.sql("SELECT * from creditdata_final").show()
```

```
23/08/15 20:57:54 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3
.0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of it.
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+------
---+---------+---------+--------+--------+--------+--------+--------+--------+--------------------------+---------
---+
| ID|LIMIT_BAL|SEX|EDUCATION|MARRIAGE|AGE|PAY_0|PAY_2|PAY_3|PAY_4|PAY_5|PAY_6|BILL_AMT1|BILL_AMT2|BILL_AMT3|BILL_A
MT4|BILL_AMT5|BILL_AMT6|PAY_AMT1|PAY_AMT2|PAY_AMT3|PAY_AMT4|PAY_AMT5|PAY_AMT6|default_payment_next_month|default_t
ext|
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+------
---+---------+---------+--------+--------+--------+--------+--------+--------+--------------------------+---------
---+
|  1|    20000|  2|        2|       1| 24|    2|    2|   -1|   -1|   -2|   -2|     3913|     3102|      689|
0|        0|        0|        0|     689|       0|       0|       0|       0|                         1|         Ye
s|
|  2|   120000|  2|        2|       2| 26|   -1|    2|    0|    0|    0|    2|     2682|     1725|     2682|     3
272|     3455|     3261|        0|    1000|    1000|    1000|       0|    2000|                         1|
Yes|
|  3|    90000|  2|        2|       2| 34|    0|    0|    0|    0|    0|    0|    29239|    14027|    13559|    14
331|    14948|    15549|    1518|    1500|    1000|    1000|    1000|    5000|                         0|
No|
|  4|    50000|  2|        2|       1| 37|    0|    0|    0|    0|    0|    0|    46990|    48233|    49291|    28
314|    28959|    29547|    2000|    2019|    1200|    1100|    1069|    1000|                         0|
No|
|  5|    50000|  1|        2|       1| 57|   -1|    0|   -1|    0|    0|    0|     8617|     5670|    35835|    20
940|    19146|    19131|    2000|   36681|   10000|    9000|     689|     679|                         0|
No|
```

sql("SELECT * from creditdata_final").show()

In [42]:
```
sql("SELECT * from creditdata_final").show()
```

```
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+------
---+---------+---------+--------+--------+--------+--------+--------+--------+-----------------------+---------
---+
| ID|LIMIT_BAL|SEX|EDUCATION|MARRIAGE|AGE|PAY_0|PAY_2|PAY_3|PAY_4|PAY_5|PAY_6|BILL_AMT1|BILL_AMT2|BILL_AMT3|BILL_A
MT4|BILL_AMT5|BILL_AMT6|PAY_AMT1|PAY_AMT2|PAY_AMT3|PAY_AMT4|PAY_AMT5|PAY_AMT6|default_payment_next_month|default_t
ext|
+---+---------+---+---------+--------+---+-----+-----+-----+-----+-----+-----+---------+---------+---------+------
---+---------+---------+--------+--------+--------+--------+--------+--------+-----------------------+---------
---+
|  1|    20000|  2|        2|       1| 24|    2|    2|   -1|   -1|   -2|   -2|     3913|     3102|      689|
0|        0|        0|       0|     689|       0|       0|       0|       0|                      1|         Ye
s|
|  2|   120000|  2|        2|       2| 26|   -1|    2|    0|    0|    0|    2|     2682|     1725|     2682|     3
272|     3455|     3261|       0|    1000|    1000|    1000|       0|    2000|                      1|
Yes|
|  3|    90000|  2|        2|       2| 34|    0|    0|    0|    0|    0|    0|    29239|    14027|    13559|    14
331|    14948|    15549|    1518|    1500|    1000|    1000|    1000|    5000|                      0|
No|
|  4|    50000|  2|        2|       1| 37|    0|    0|    0|    0|    0|    0|    46990|    48233|    49291|    28
314|    28959|    29547|    2000|    2019|    1200|    1100|    1069|    1000|                      0|
No|
|  5|    50000|  1|        2|       1| 57|   -1|    0|   -1|    0|    0|    0|     8617|     5670|    35835|    20
940|    19146|    19131|    2000|   36681|   10000|    9000|     689|     679|                      0|
No|
|  6|    50000|  1|        1|       2| 37|    0|    0|    0|    0|    0|    0|    64400|    57069|    57608|    19
394|    19619|    20024|    2500|    1815|     657|    1000|    1000|     800|                      0|
No|
|  7|   500000|  1|        1|       2| 29|    0|    0|    0|    0|    0|    0|   367965|   412023|   445007|   542
653|   483003|   473944|   55000|   40000|   38000|   20239|   13750|   13770|                      0|
No|
```

Convert string field to int using encoder :

```
from pyspark.ml.feature import (VectorAssembler,VectorIndexer,
                                OneHotEncoder,StringIndexer)

gender_indexer = StringIndexer(inputCol='default_text',outputCol='default')
gender_encoder = OneHotEncoder(inputCol='default',outputCol='defaultVec')


assembler = VectorAssembler(inputCols=[
 #'SexVec',
 #'credit_limit',
 'BILL_AMT1',
 'BILL_AMT2',
 'BILL_AMT3',
 'BILL_AMT4',
 'BILL_AMT5',
 'BILL_AMT6',
 'PAY_AMT1',
 'PAY_AMT2',
 'PAY_AMT3',
 'PAY_AMT4',
 'PAY_AMT5',
 'PAY_AMT6',],outputCol='features')
```

Pipelines

- **Transformer**: A Transformer is an algorithm which can transform one DataFrame into another DataFrame. E.g., an ML model is a Transformer which transforms a DataFrame with features into a DataFrame with predictions.
- **Estimator**: An Estimator is an algorithm which can be fit on a DataFrame to produce a Transformer. E.g., a learning algorithm is an Estimator which trains on a DataFrame and produces a model.
- **Pipeline**: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline

log_reg_titanic =
LogisticRegression(featuresCol='features',labelCol='default_payment_next_month')


pipeline = Pipeline(stages=[gender_indexer,gender_encoder,
                    assembler,log_reg_titanic])
(train_data, test_data) = data.randomSplit([0.8,.2])

fit_model = pipeline.fit(train_data)
```

```
results = fit_model.transform(test_data)
```

Evaluation

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
my_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction',
                                        labelCol='default_payment_next_month')

results.select('default_payment_next_month','prediction').show()
```

In [55]:
```
my_eval = BinaryClassificationEvaluator(rawPredictionCol='prediction',
                                        labelCol='default_payment_next_month')
```

In [65]:
```
results.select('default_payment_next_month','prediction').show()
```

restart the kernel, then re-run the whole notebook (with dialog)

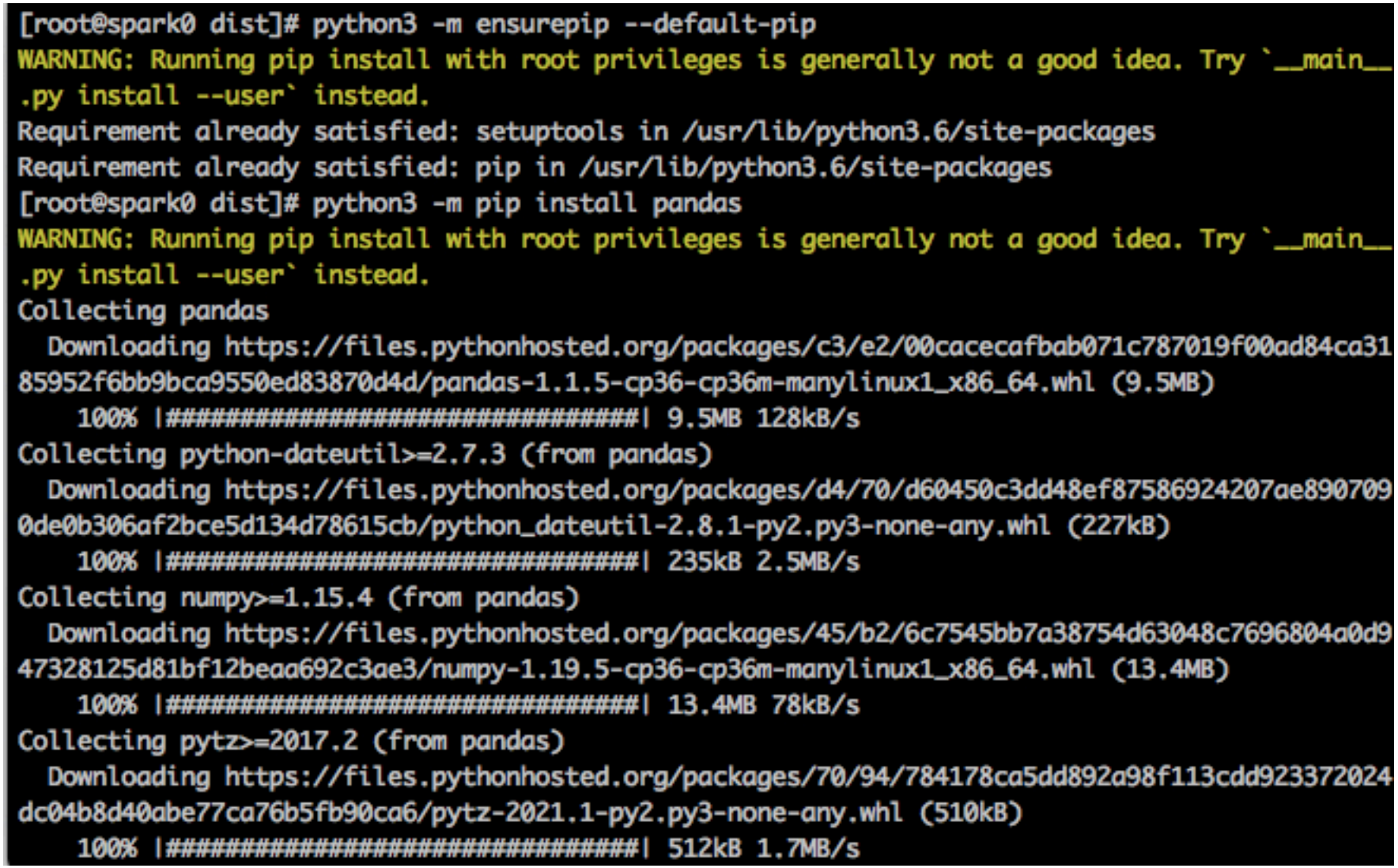
```
+--------------------------+----------+
|default_payment_next_month|prediction|
+--------------------------+----------+
|                         1|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         1|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         1|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         0|       0.0|
|                         1|       0.0|
|                         0|       0.0|
|                         1|       0.0|
|                         0|       0.0|
+--------------------------+----------+
only showing top 20 rows
```

results.select('default_payment_next_month','prediction').where('prediction == 0 ').count()

AUC = my_eval.evaluate(results)
AUC

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

```
In [66]: results.select('default_payment_next_month','prediction').where('prediction == 0 ').count()
Out[66]: 6004

In [67]: AUC = my_eval.evaluate(results)
         23/08/15 21:33:32 WARN SQLConf: The SQL config 'spark.sql.execution.arrow.enabled' has been deprecated in Spark v3
         .0 and may be removed in the future. Use 'spark.sql.execution.arrow.pyspark.enabled' instead of it.

In [68]: AUC
Out[68]: 0.5001562815468992
```


[root@spark0 dist]# python3 -m ensurepip --default-pip
WARNING: Running pip install with root privileges is generally not a good idea. Try `__main__
.py install --user` instead.
Requirement already satisfied: setuptools in /usr/lib/python3.6/site-packages
Requirement already satisfied: pip in /usr/lib/python3.6/site-packages
[root@spark0 dist]# python3 -m pip install pandas
WARNING: Running pip install with root privileges is generally not a good idea. Try `__main__
.py install --user` instead.
Collecting pandas
Downloading https://files.pythonhosted.org/packages/c3/e2/00cacecafbab071c787019f00ad84ca31
85952f6bb9bca9550ed83870d4d/pandas-1.1.5-cp36-cp36m-manylinux1_x86_64.whl (9.5MB)
100% |################################| 9.5MB 128kB/s
Collecting python-dateutil>=2.7.3 (from pandas)
Downloading https://files.pythonhosted.org/packages/d4/70/d60450c3dd48ef87586924207ae890709
0de0b306af2bce5d134d78615cb/python_dateutil-2.8.1-py2.py3-none-any.whl (227kB)
100% |################################| 235kB 2.5MB/s
Collecting numpy>=1.15.4 (from pandas)
Downloading https://files.pythonhosted.org/packages/45/b2/6c7545bb7a38754d63048c7696804a0d9
47328125d81bf12beaa692c3ae3/numpy-1.19.5-cp36-cp36m-manylinux1_x86_64.whl (13.4MB)
100% |################################| 13.4MB 78kB/s
Collecting pytz>=2017.2 (from pandas)
Downloading https://files.pythonhosted.org/packages/70/94/784178ca5dd892a98f113cdd923372024
dc04b8d40abe77ca76b5fb90ca6/pytz-2021.1-py2.py3-none-any.whl (510kB)
100% |################################| 512kB 1.7MB/s

%pyspark

```
import sys
sys.version_info
```

---------- Lab Ends Here ----------

Update the following to python3 using the Zeppelin Interpreter.

PYSPARK_PYTHON
PYSPARK_DRIVER_PYTHON

https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5526770934527230/1346075117417701/1249677559564036/latest.html

Data Transformation:
https://www.data-stats.com/spark-analytics-on-movielens-dataset/

Git
https://www.liquidlight.co.uk/blog/git-for-beginners-an-overview-and-basic-workflow/

https://hackernoon.com/building-a-machine-learning-model-with-pyspark-a-step-by-step-guide-1z2d3ycd

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5526770934527230/1346075117417701/1249677559564036/latest.html

https://www.codementor.io/@jadianes/spark-mllib-logistic-regression-du107neto

https://turingintern2018.github.io/sparkairplane.html
https://towardsdatascience.com/machine-learning-with-pyspark-and-mllib-solving-a-binary-classification-problem-96396065d2aa
https://github.com/susanli2016/PySpark-and-MLlib/blob/master/Machine%20Learning%20PySpark%20and%20MLlib.ipynb
https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8
https://www.kaggle.com/hassanamin/exploring-banking-data-using-pyspark

5. Understand classification and logistic regression

https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-machine-learning-mllib-notebook