INSERT WORKSHOP PICTURE HERE

Lab Exercise:

Experiment IBM Watson Machine Learning for z/OS
as a business analyst and/or a data scientist!

Speakers name: Guillaume Arnould
Diego Cardalliaguet
Marie-Laure Pessoa

Table of Contents

Disclaimer

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.
The development, release, and timing of any future features or functionality described for our products remains at our sole discretion I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results like those stated in this disclaimer.
Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed "as is" without any warranty, either expressed or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted per the terms and conditions of the agreements under which they are provided.
IBM products are manufactured from new parts or new and used parts.

In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.
Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.
Performance data contained herein was generally obtained in controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.
Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participants or their specific situation.
It is the customers' responsibility to insure their own compliance with legal requirements and to obtain the advice of competent, legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer's business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer follows any law.
Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products about this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a purpose.**
The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.
IBM, the IBM logo, ibm.com and [names of other referenced IBM products and services used in the presentation] are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.

Introduction

IBM Watson Machine Learning for z/OS (WMLz) is an enterprise machine learning platform that provides end to end management of the entire machine learning workflow. WMLz helps to simplify and significantly reduce the time to train and deploy machine learning models by:

- Integrating all the tools and functions needed for machine learning and automating the machine learning workflow.
- Providing a platform for better collaboration across different personas, including Data Scientist, Data Engineer, Business Analyst and Application Developers, for a successful machine learning project.
- Infusing cognitive capabilities into the machine learning workflow to help determine when model results deteriorate and need to be tuned and provide suggestions for updates or changes.

Lab Overview

Once upon a time, there was a bank offering services to private individuals. Offered services include managing of accounts, loans, etc.

The bank wants to improve their services by finding interesting groups of clients (e.g. to differentiate between good and bad clients). Bank managers have a vague idea of who could be targeted (whom to offer additional services) and who is not eligible (whom to watch carefully to minimize the bank losses). The bank stores data about their clients: accounts (transactions within several months), loans already granted, credit cards issued... Some customer behavior can be extracted by analyzing this data so that the bank managers can improve their understanding of customers and seek for specific actions to improve services.

During this hands-on-lab, based on this bank loan data you will experiment the full life cycle of predictive model development:

- Discover your environment on Machine Learning for z/OS, configure your project, your data sources and data sets to be able to start visualizing your data thru data visualization and modeler flow (Lab 1)
- Through a Jupyter notebook, you will load, manipulate and arrange data contained in 8 tables from Db2 for z/OS, before infusing data into machine learning models (Lab 2)
- Through a Jupyter notebook, you will use the dataframe prepared in Lab 2 to train and build machine learning models, we will compare random forest models, gradient boost and logistic regression to choose the one giving best results (Lab 3)
- Select the most pertinent model created in Lab 3 to deploy and test it (Lab 4)

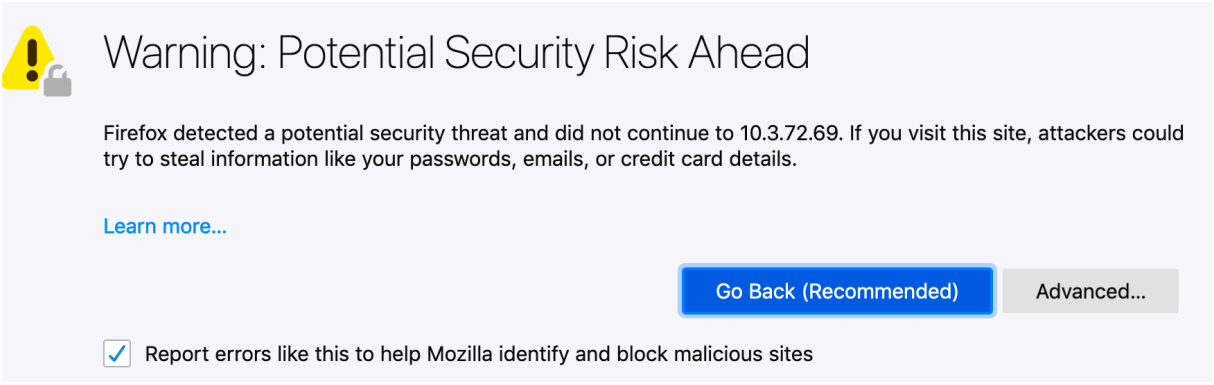
Lab Instructions

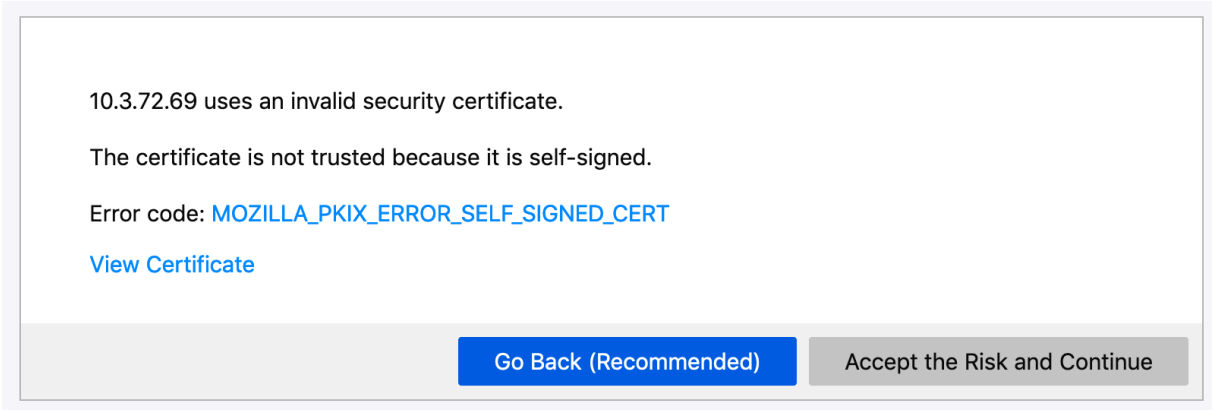
Each student should receive a separated lab worksheet. The worksheet provides information that is specific to each student to use throughout the lab.
Some of the instructions in the lab instruct the student to use a value that is marked with angle brackets (< >). The value the student should use is on their lab worksheet. Make sure to exclude the brackets when using the value for its intended purpose in the lab.

Log in to IBM Watson Machine Learning for z/OS (WMLz)

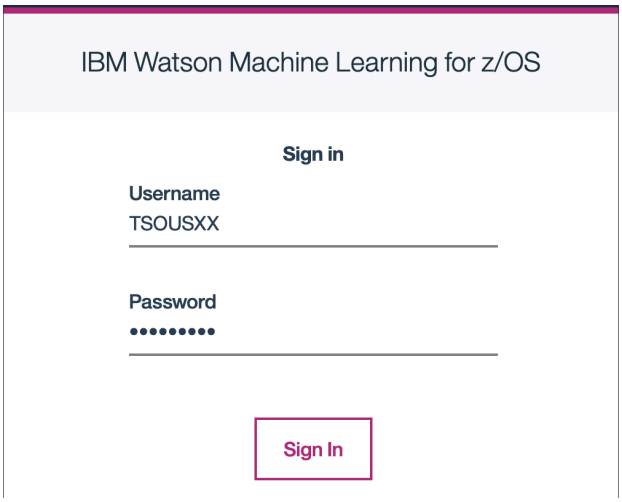
__ 1.
The URL and authentication credentials for the WMLz Web UI are provided on the separated worksheet. Contact the lab instructor if you did not receive a worksheet.

Start the Chrome or Firefox browser and enter the URL assigned to you, e.g. <login_url>. For the first-time logon, you are likely to be prompted with a message regarding the security of the page you are visiting. Click **Advanced** and the Accept the Risk and Continue.

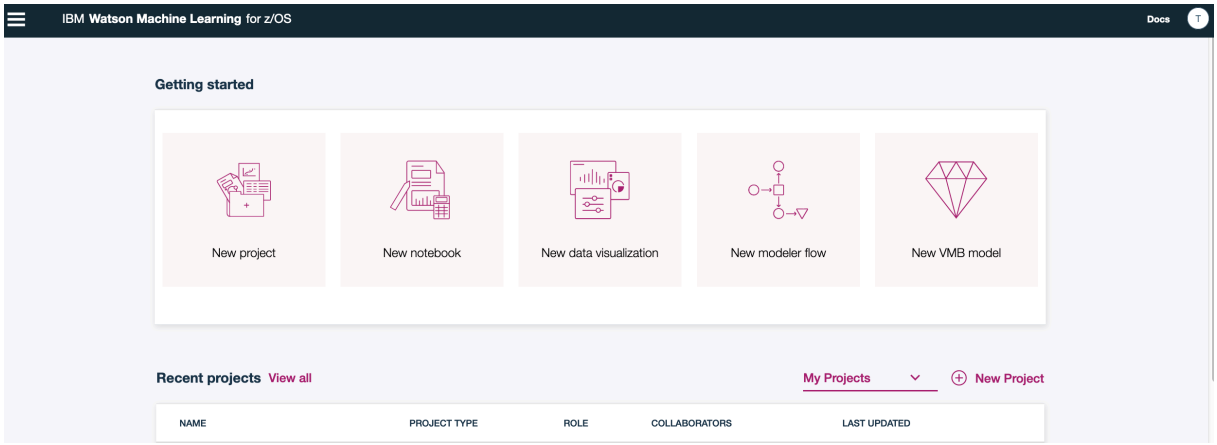




__ 2.
Enter the <**login_userid**> and <**login_password**> credentials from your lab worksheet.



The WMLz main page will be displayed.

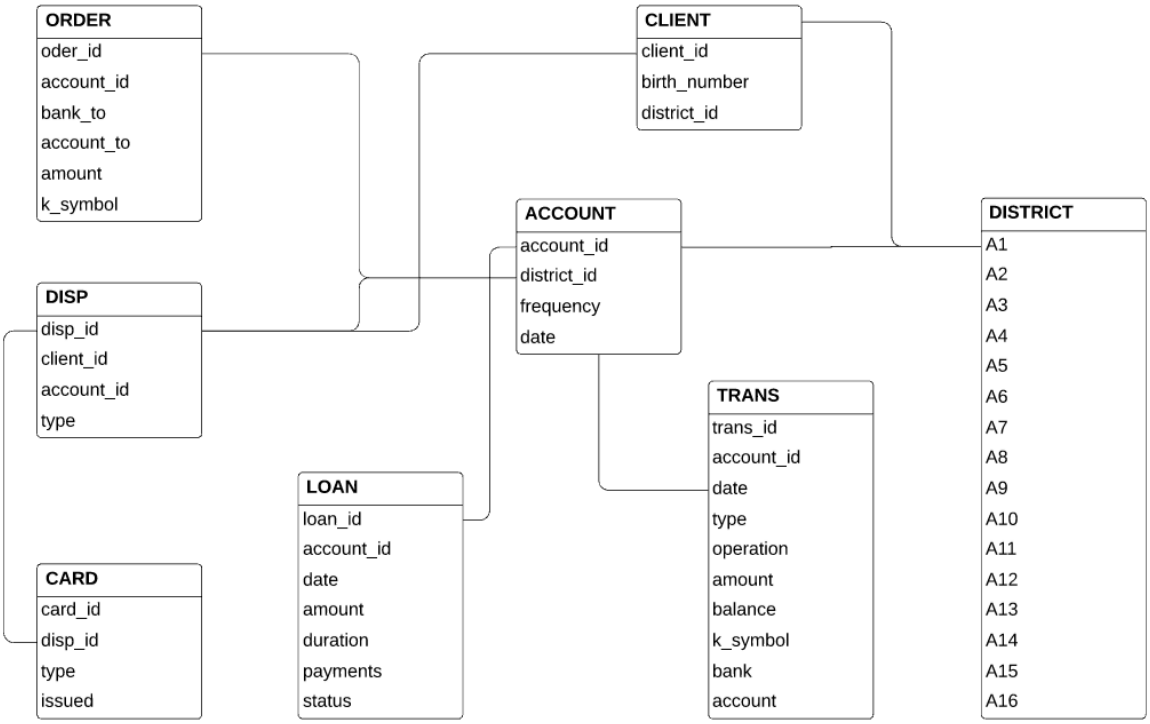


You can now proceed to the first lab.

Lab 1: Introductory lab - Dataset examination and visualization

As mentioned in the *Lab Overview*, we will work with bank loan data. This bank offers services to private persons which include managing of accounts, offering loans, etc.

Here is the entity relationship diagram of this data:



- Transactions Db2 table: TRANS

Columns name: trans_id ; account_id ; date ; type ; operation ; amount ; balance ; k_symbol ;bank ; account

- Account Db2 table: ACCOUNT

Columns name: account_id ; district_id ; frequency ; date

- Loan Db2 table: LOAN

Columns name: loan_id ; account_id ; date ; amount ; duration ; payments ; status

- Credit Card Db2 table: CARD

Columns name: card_id ; disp_id ; type ; issued

- Client Db2 table: CLIENT

Columns name: client_id ; birth_number ; district_id

- Disposition Db2 table: DISP

Columns name: disp_id ; client_id ; account_id ; type

- Demographic Db2 table: DISTRICT

Columns name: A1 ; A2 ; A3 ; A4 ; A5 ; A6 ; A7 ; A8 ; A9 ; A10 ; A11 ; A12 ; A13 ; A14 ; A15 ; A16

- Permanent order Db2 table: ORDER

Columns name: "order_id";"account_id";"bank_to";"account_to";"amount";"k_symbol"

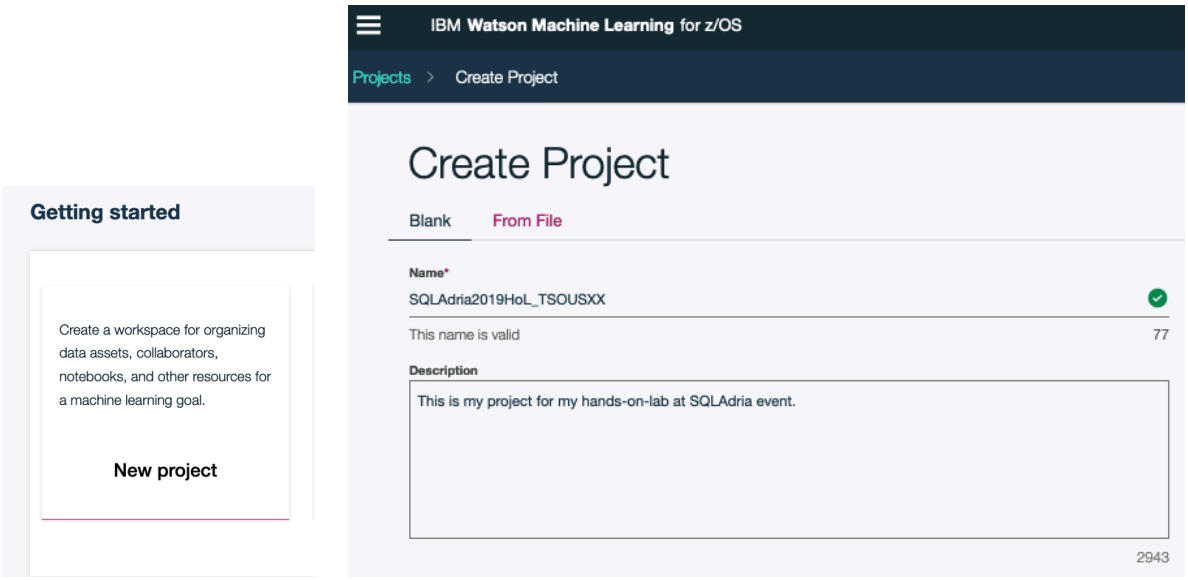
Section 1. Data visualization and exploration

In this section, you will create a project, configure a data source and its associated datasets to be able to start discovering the bank loan data.

__ 1. Project Creation

WMLz provides the data visualization tool to help data scientists and business analysts explore data visually by dragging and dropping them. In this section, you will learn how to access the data visualization tool to explore the bank loan data.

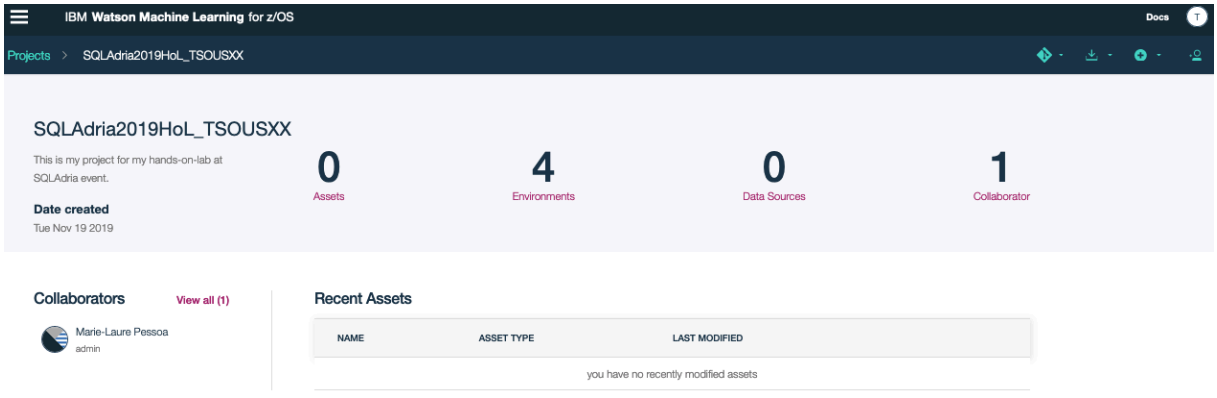
Click the **New Project** square and type a title such as SQLAdria2019HoL_**<login_userid>** as the project name, a short description if desired and then click on **Create** at the bottom right.



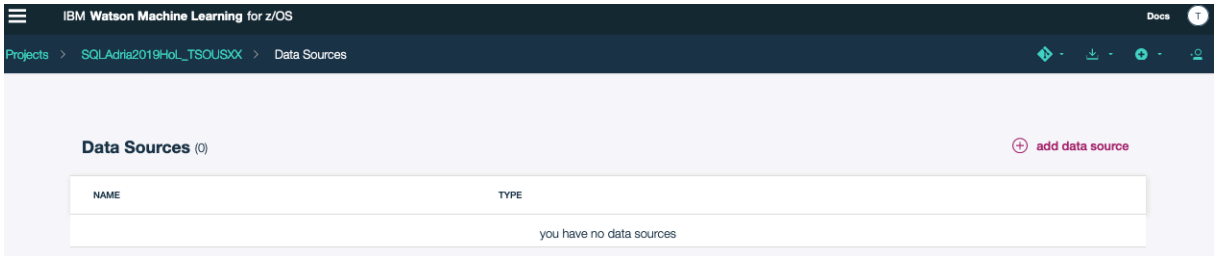
__ 2. Data source configuration

We will now create a data source for the project.

__2.1. Click on the project to open the project summary page. Then click the **Data Sources** link to open the **Data Sources** page.



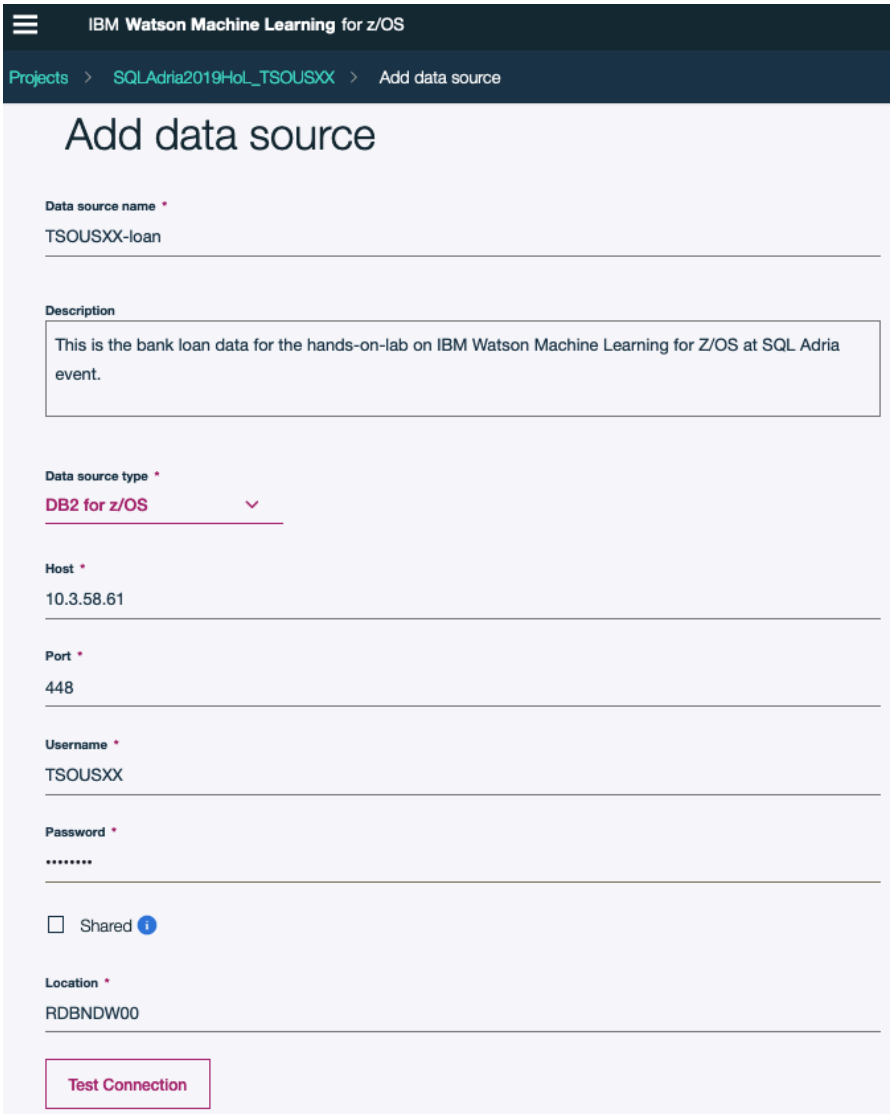
__ 2.2. Click the **add data source** button in the **Data Sources** page to create a DB2 for z/OS type data source.



__ 2.3. Fill out the data source information:

Data source name: <**login_userid>-loan**
Data source type: DB2 for z/OS
Host: <db2_host_ip>
Port: <db2_port>
Username: <login_userid>
Password: <login_password>
Location: <db2_location>

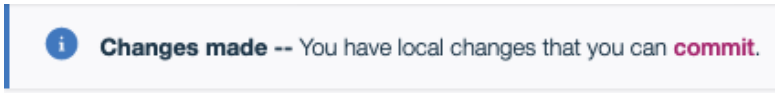
Note: Use the values specified above [Do not use the ones displayed in this image].



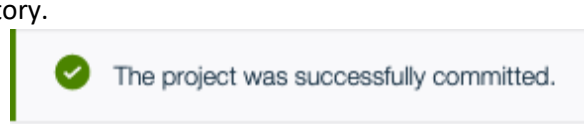
__ 2.4. Click the **Test Connection** button before creating the data source:

If the connection test is successful, **Close** the dialog box and click **Create.** After a few seconds**,** the "Data Sources" view will be displayed where you can see your newly created data source.

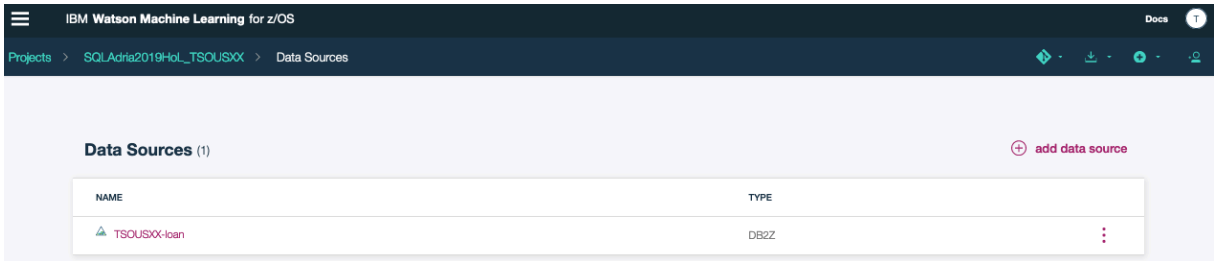
Note: At any point in time you can see this message at the top of your screen:



You can click on **commit,** enter a commit message in the new box dialog and click on **Push** to push changes to master repository.



__ 2.5. At this point you might have this view:



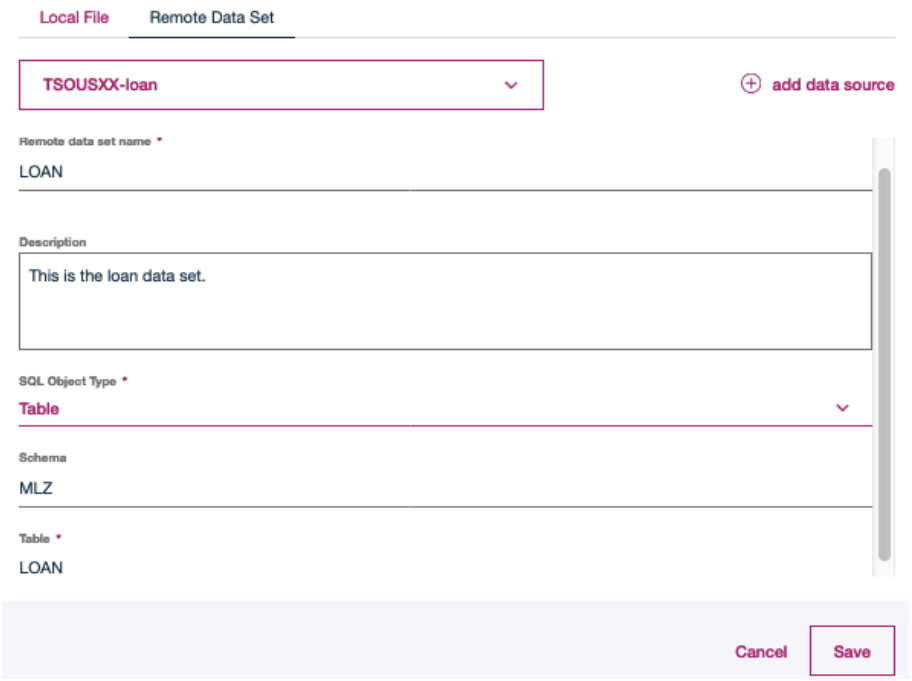
__ 3. Create a dataset that uses the new data source

__ 3.1. Click on the project name in the top left navigation bar to go back to the project overview page.

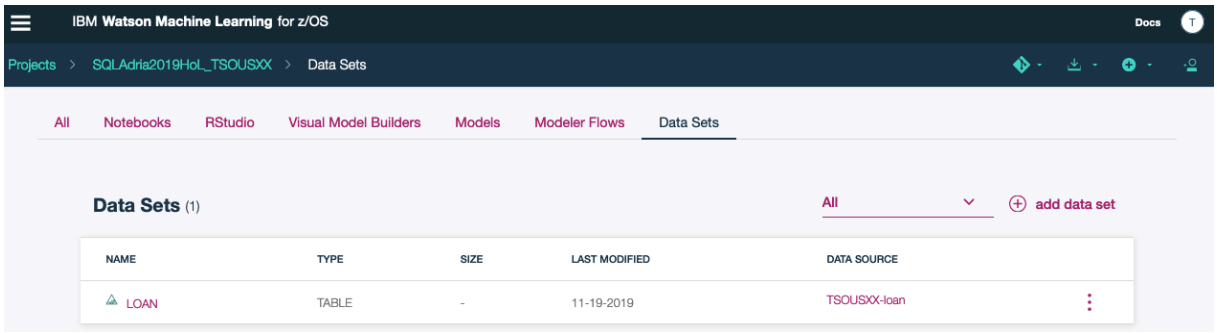
Click **Assets** to display the assets list into your project.
Select the **Data Sets** tab and click the **add data set** button. Once the dialog box to add data set is displayed, click the **Remote Data Set** tab.

__ 3.2. Select the **Data Source** created in **the** above step and fill out the data set information:

- Remote data set name: **LOAN**
- Description: if desired
- Click on the **SQL Object Type** drop-down list and choose the **Table** option
- Schema : **MLZ**
- Table: **LOAN**
- Click the **Save** button.



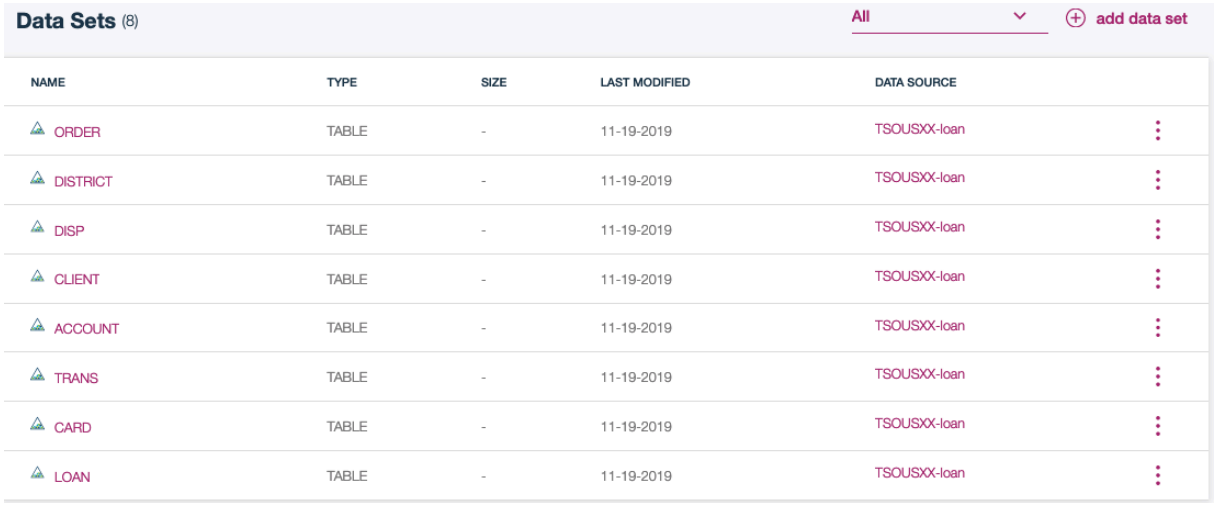
You should be able to see the newly created data set in the list of Data Sets.



__ 3.3. Repeat the above steps for the following data sets:

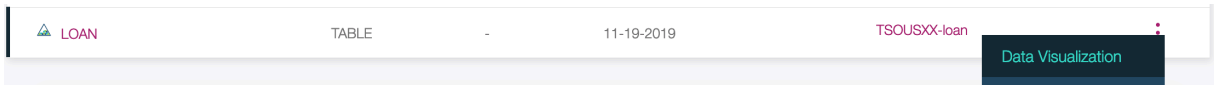
- CARD
- TRANS
- ACCOUNT
- CLIENT
- DISP
- DISTRICT
- ORDER

Once configuration completed, you should see this list of data sets:

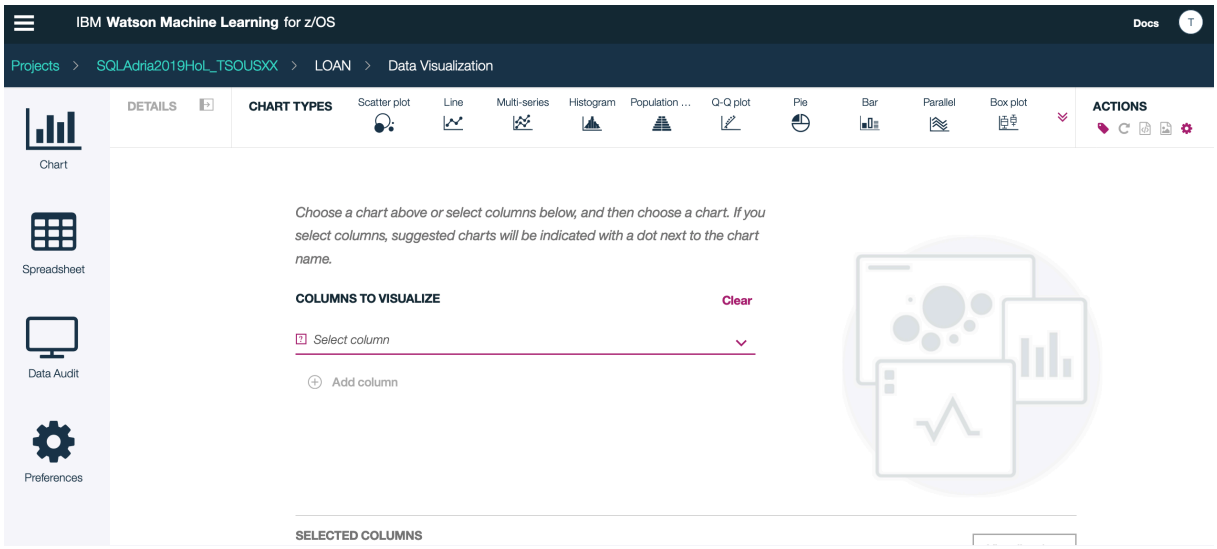


__ 4. Start discovering your data

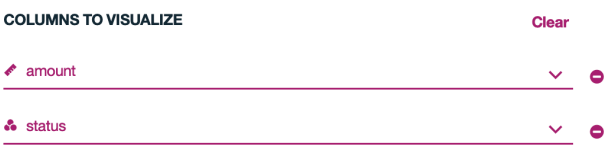
__ 4.1. We will start discovering what is inside the data sets we just configured.
Click on the three dots at the right side of your **LOAN** data set line and click on **Data Visualization:**



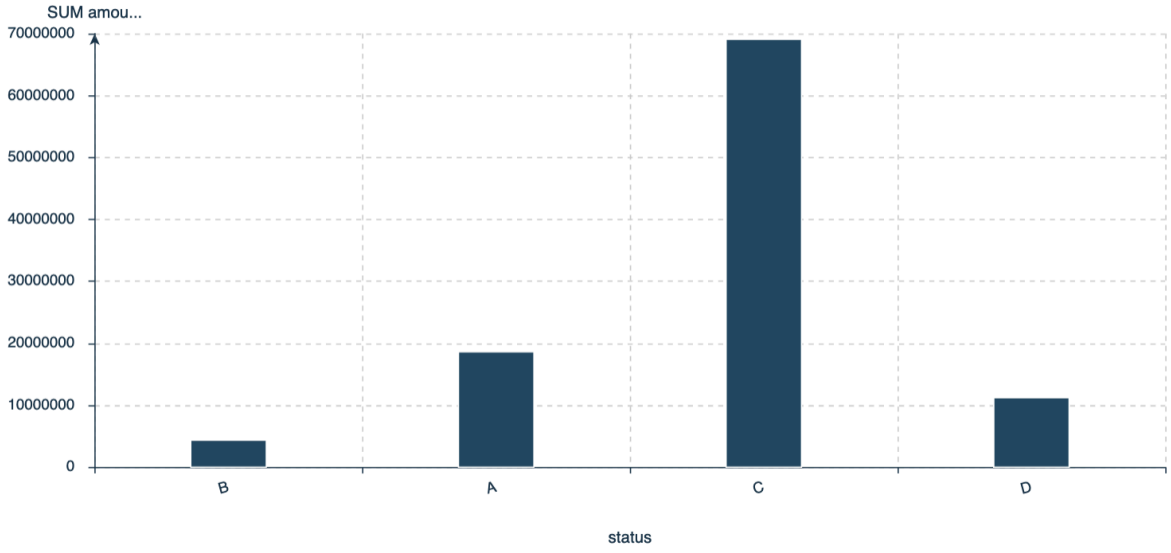
It will open the Data Visualization dashboard as shown below:



__ 4.2. Let's start to visualize 2 columns: add **amount** and **status** to the **COLUMNS TO VISUALIZE** section, then select **Bar** from **CHART TYPES** on the top bar:



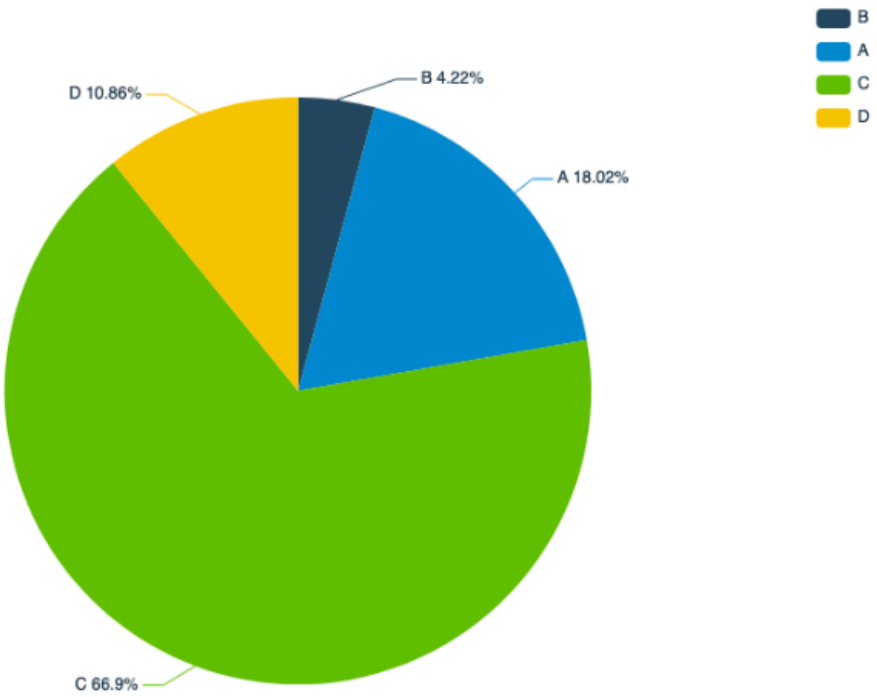
Here is the chart you should obtain:



Note:
Amount represents the amount of money borrowed.
Status represents the status of paying off the loan with 4 letters:

- 'A' stands for contract finished, no problems
- 'B' stands for contract finished, loan not payed,
- 'C' stands for running contract, OK so far,
- 'D' stands for running contract, client in debt

__ 4.3. You can then click on **Pie** to visualize your data differently – click **Continue** to switch chart types – the system chooses to show the data that makes more sense for this type of chart which is the status and it makes a count on this field to show it as percentage:



__ 4.4. You can also look at the data in this data set as a table anytime you want by clicking on **Spreadsheet** into the left icons menu of this Data Visualization Dashboard.

	loan_id	account_id	date	amount	duration	payments	status
1	5314	1787	930705	96396	12	8033	B
2	5316	1801	930711	165960	36	4610	A
3	6863	9188	930728	127080	60	2118	A
4	5325	1843	930803	105804	36	2939	A

__ 4.5. Below the table you have the SQL Editor section which allows you to enter a SQL command and run it, you can complete the 'Select * from ThisTable' with **where duration='24'** – to display loan which lasts 24 months - and click on the **Play** button, the table will be updated.

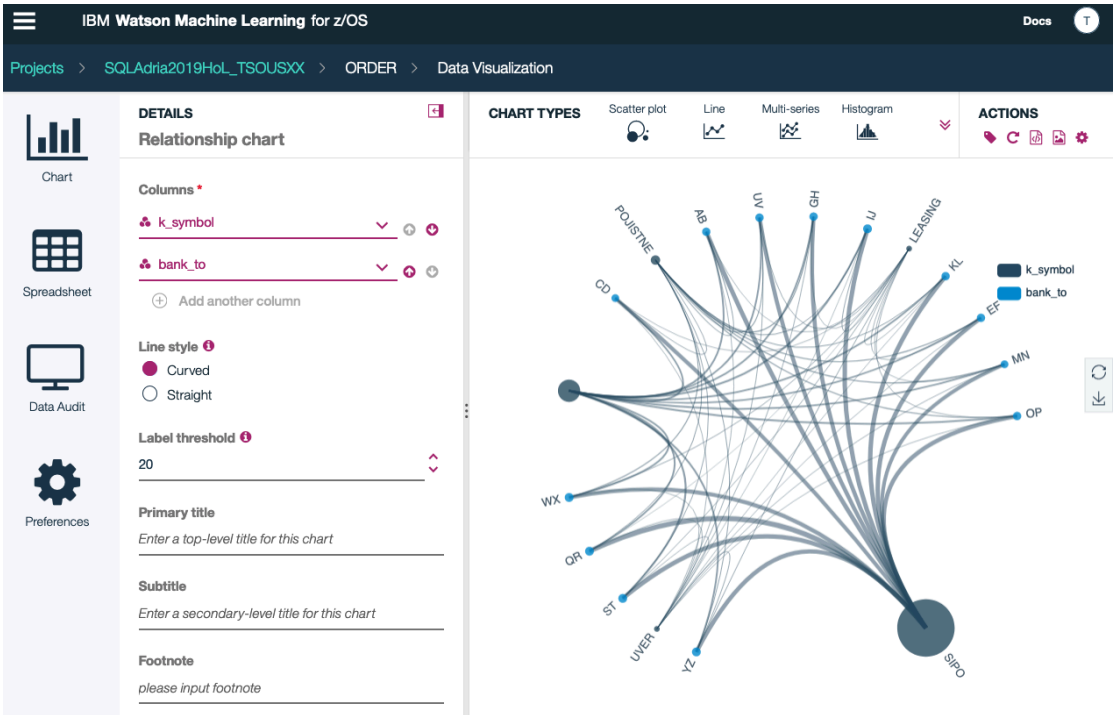
__ 4.6. Now let's have a look at the **ORDER** data set, click on the three dots at the right side of your ORDER data set line and click on **Data Visualization.**

Select the k_symbol and bank_to columns for the **COLUMNS TO VISUALIZE** section, then click on **Relationship** from **CHART TYPES** on the top bar, see below the graph you should obtain.

Note:

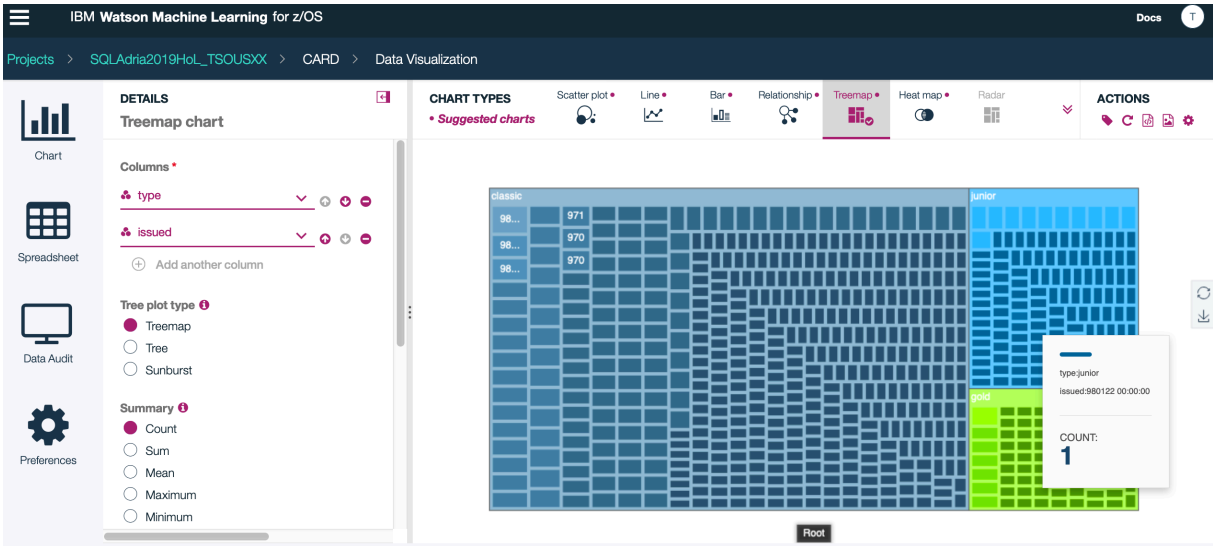
- k_symbol characterizes the payment:
  - "POJISTNE" stands for insurrance payment
  - "SIPO" stands for household payment
  - "LEASING" stands for leasing
  - "UVER" stands for loan payment
- And bank_to columns include the bank to which the payment has been made.

Here is what you may obtain:



We can notice that SIPO is one of the main payments.

__ 4.7. Let's now select the CARD data set and build the below graph by selecting the columns **type** and **issued** and choosing the **Treemap** chart type, you should get this chart:

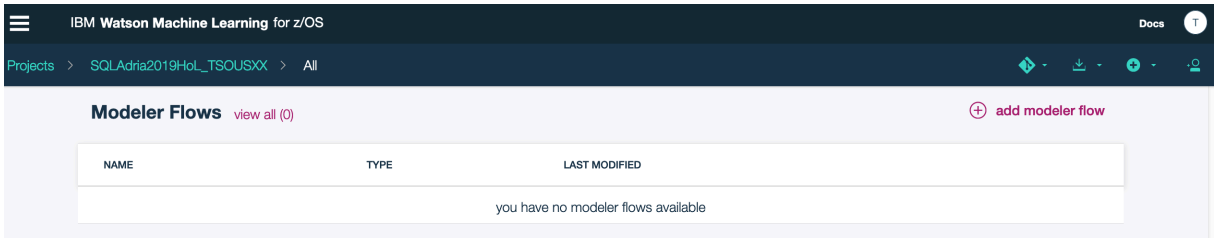


We can notice the three types of cards: classic, junior and gold, classic being the most important; their date/number issued appear in each square – as big is the square, as big the number of cards on this date has been issued.

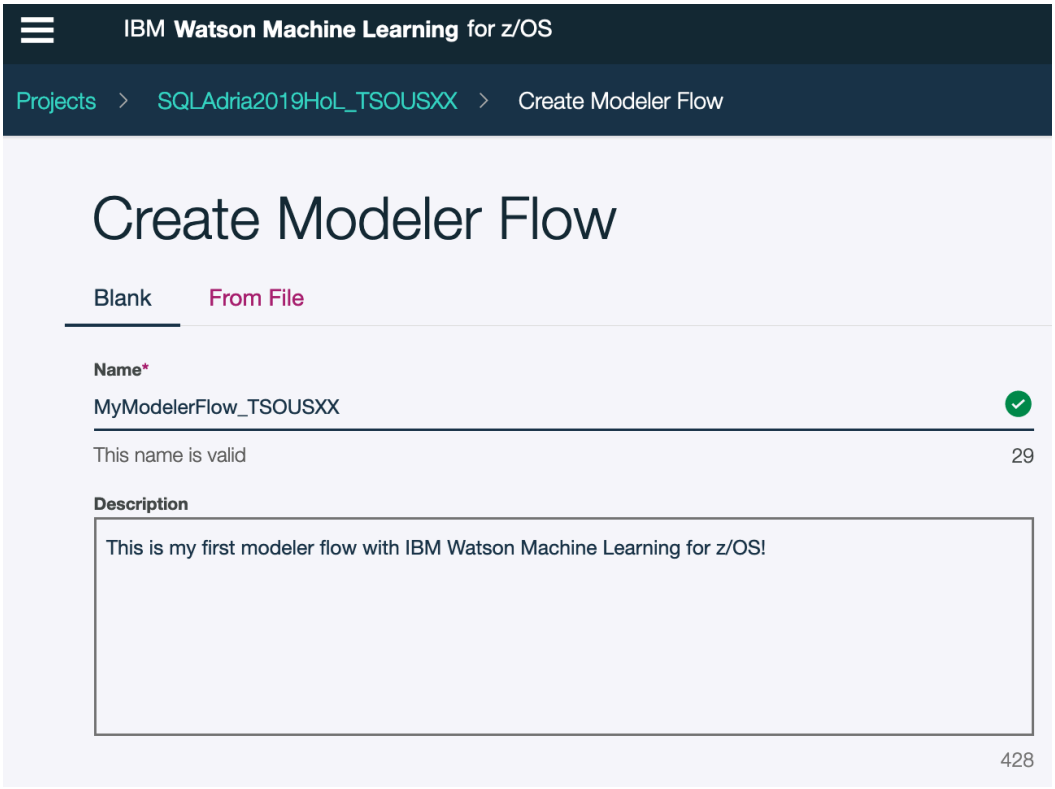
Section 2. Data merging using SPSS Modeler Flow

In this section you will learn how to use the SPSS Modeler Flow of IBM Machine Learning for z/OS to merge 3 data sets: CARD, DISP and TRANS in order to obtain the amount spent by type of cards.

__ 1.
From the project overview, click **Asset** and then on the Asset page, click **Modeler Flows** to bring up the **SPSS Modeler Streams** tab. Click **add modeler flow**:

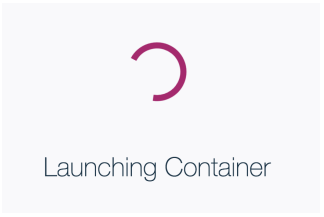


__ 2.
Give a name to the new created flow, such as **MyModelerFlow_<login_userid>**:



Then click the **Create** button.

You will see the following spinner indicating a runtime container is being created for the SPSS modeler flow. This could take a minute or two the first time you open a SPSS flow in your project:



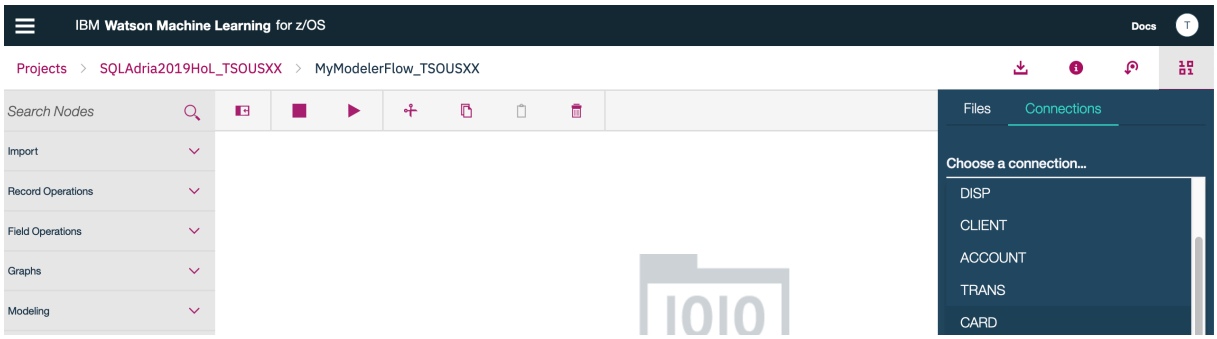
You are now in front of the Flow Editor Canva (white space in the middle) of the SPSS Modeler flow:

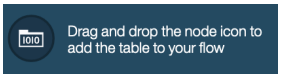
- On the left side you can open/close the Palette 

with contains all the Nodes (transformers and modeling algorithms transform and train your data) which can be added to the Canva
- And on the right side you have four options:  Download / Information on your Modeler Flow / Outputs / File or Connections already associated to your Project which can be added into the Canva.

__ 3.
As mentioned at the beginning of this section, we will do a merge between the 3 data sets: CARD, DISP and TRANS in order to obtain the amount spent by type of cards on a graph.

First select the data sets you want to include into your Flow Editor Canva: pull out the right panel by clicking the drawer button and switch to the **Connections** tab. Click the **Choose a connection...** button and select the data set **CARD** created in the previous step:



Once selected, as shown in the dialog box drag the **node icon into your** Canva: 

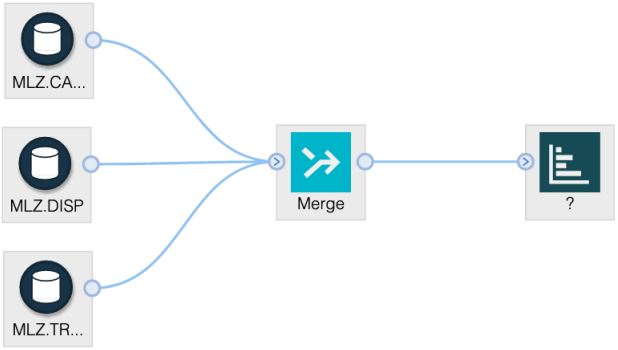
__ 4.
Repeat the above steps for data sets **DISP** and **TRANS**.

__ 5.

Pull out the **Record Operations** section from the left panel and drag and drop the **Merge** node  into the Canva
Connect the three data sets input nodes to this Merge node by putting your mouse on the little circle at the right of the node 

and dragging a line until the little circle at the left of the Merge node. Draw the line for each data set until the Merge node.
Then, as we would like to have a graph as output, pull out the Graphs section from the left panel and drag and drop the Distribution node  into the Canva as output to the flow.

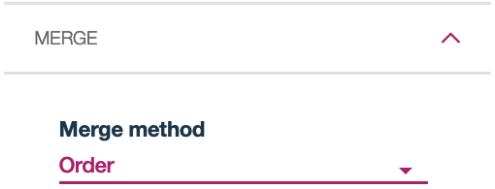
You should get the following modeler stream:

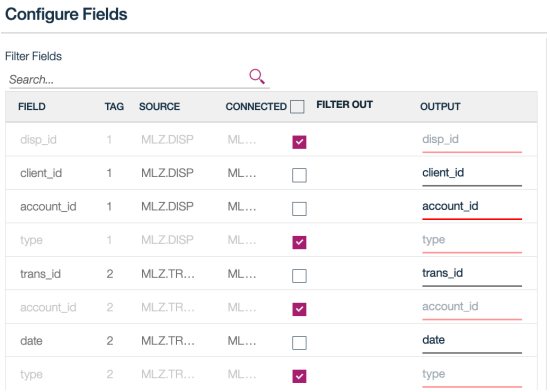


Note: Merge is used to combine data from multiple data sources and you need to make sure to have single output data in the graph.

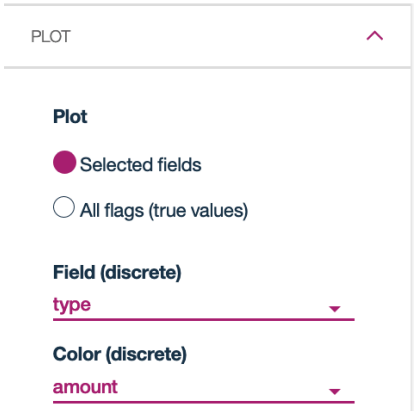
__ 6.
Configure the Merge node behavior: double click on it to open the node configuration in the right dock panel.:

- Leave the inputs section as it
- For the Merge section, leave the **Order** method
- For the Filter section, click on Configure Fields and tick the column to filter out in order to take out any duplicate, so tick: **disp_id, type** (twice as it appears 3 times so let's keep only one) and **account_id.** Click on **OK**
- Leave the Optimization and Annotations section as default.
- Click on **Save**.





__ 7.
Let's configure now the Distribution node behavior: double click on it to open the node configuration in the right dock panel.
In the Plot section, select **type** for the field and **amount** for the Color – as reminder we want to see into this graph the amount spent by type of cards.



(you will notice that the name of the node changed to "Type")
Click on **Save.**

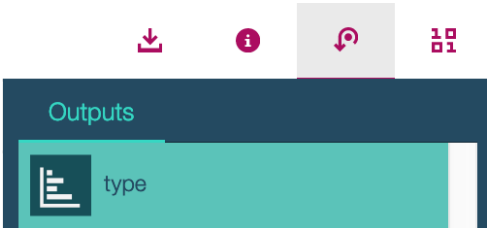
__ 8.

You are now ready to run your modeler flow, click on the Run button 

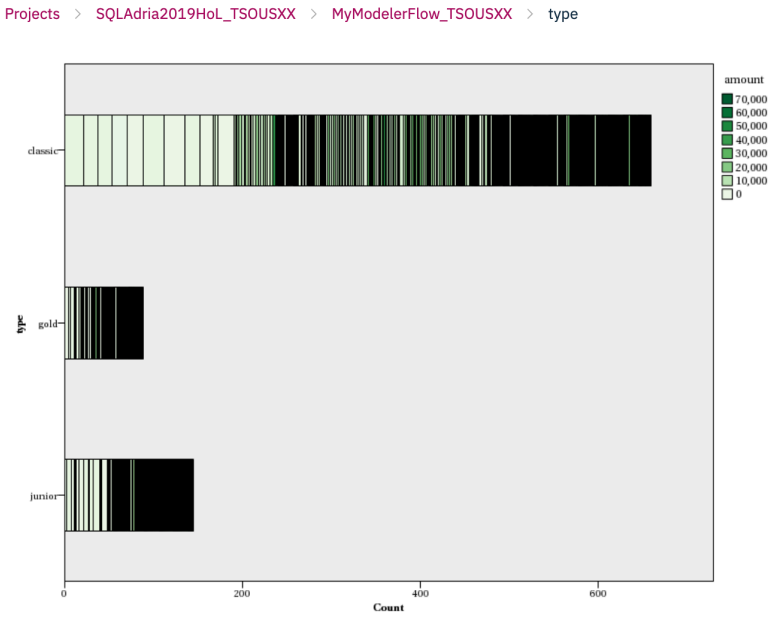
at the top of your Canva.

It takes some seconds to run the flow with the following indicator:

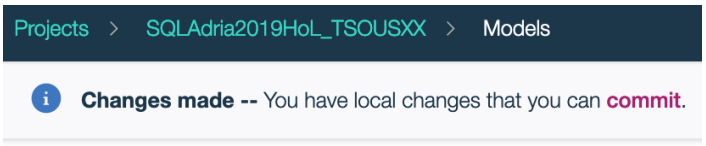
After the flow is competed, you should be able to see the graph in the right panel by clicking on the output logo:  and double clicking on the graph that has been also named "Type":



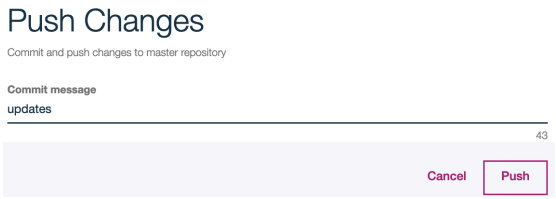
Here is the graph you should obtain:



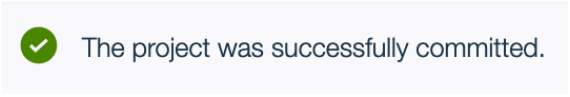
__ 9. Navigate back to the project view by clicking on your project name at the top- left of the window, you might notice the following message, you can click on commit to upload changes into your project:



Type a message before hitting the **Push** button:

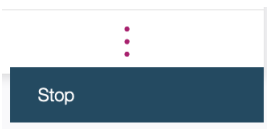


You will get this message as a confirmation:



__ 10.
To free up CPU and memory resources, please shut down your SPSS Modeler runtime.

- Click on your **project name** in the navigation bar
- Click **Environments** on the project overview page
- Click the **action menu (3 dots)** for SPSS Modeler and select **Stop**:



You completed lab 1.

Lab 2: Data Preparation through a Jupyter Notebook

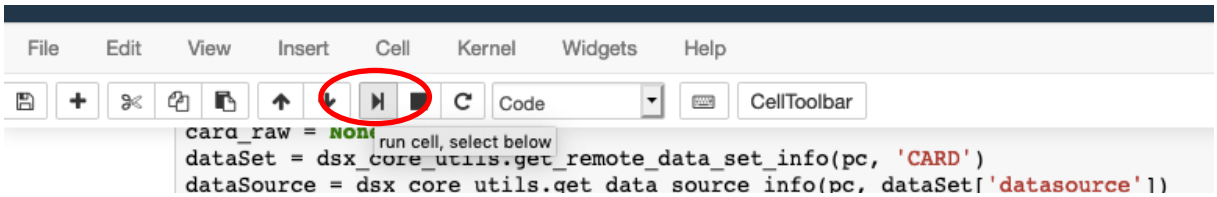
This lab will guide you through a Jupyter notebook inside the development environment of the Watson Machine Learning for z/OS. The goal of this notebook is to load, manipulate and arrange data contained in 8 tables from Db2 for z/OS.
Before infusing data into machine learning models, it is necessary to work on it in an ETL way to have the right pattern of data and to be able to inform the model with an objective to be predicted. The columns used by the models to predict are called *features* and the predicted values are called *target*.
By the end of the lab you will obtain a "dataframe" with all the information required by the machine learning models you will use in lab 3.

Note: You need your worksheet to login into the system and follow the name convention. The worksheet provides information that is specific to each student to use throughout the lab.
Some of the instructions in the lab instruct the student to use a value that is marked with angle brackets (< >). The value the student should use is on their lab worksheet.

→ Make sure to exclude the brackets when using the value for its intended purpose in the lab.

→ Along the lab you will have to execute Python code in cells, to do this you have two options

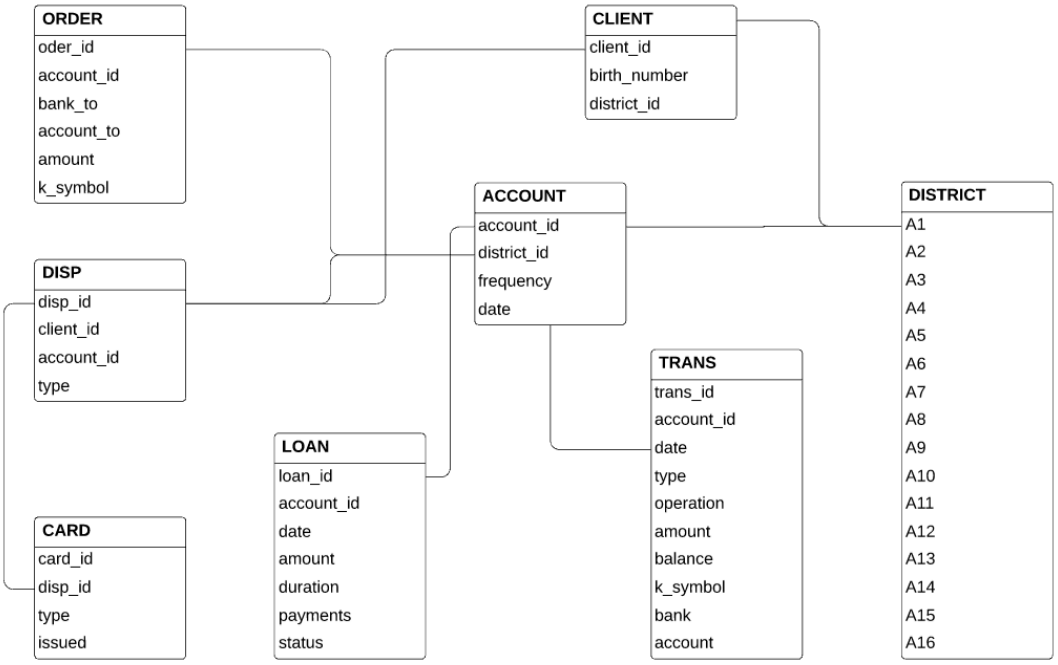
- Click on the run icon on the menu



- Pulse Shift + Return on the keyboard.

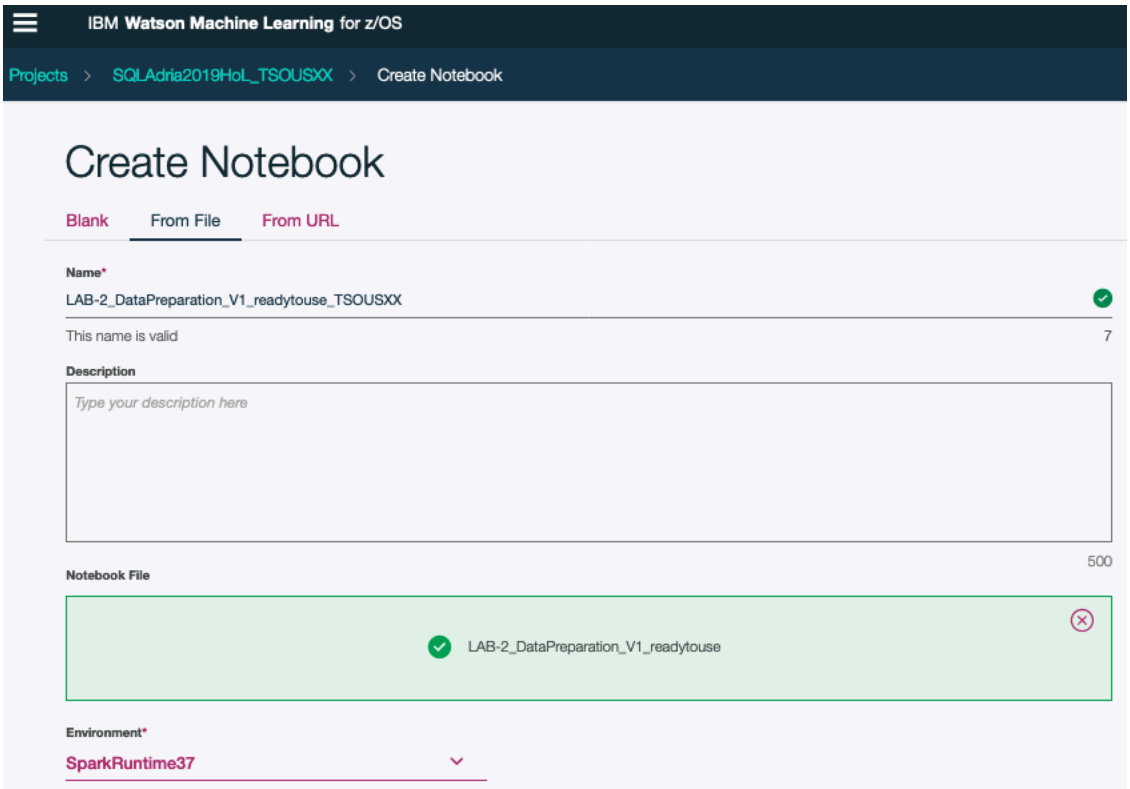
Section 1. Loading tables from Db2 for z/OS

You remember from lab 1 that the tables we will use are:



__ 1. Open your project with your <login_userid> created in lab 1 and go to your assets list. At the right side of the Notebooks section, click on ⊕ add notebook

__ 2. In the below screen:



- Select **From File** in the 3 available options
- Drag and drop the file **LAB-2_DataPreparation_V2_readytouse.ipynb** located on your desktop into the Notebook File section.

- **Once you have the Notebook File added to its section**, add _<login_userid> to the Notebook Name created by default
- Leave the Environment as it's
- And click on **Create.**

__3. Read all the introduction in the notebook until you get to cells with Python code.

__4. CELL 1. Execute the cell with the imports for all packages we will need along the notebook.
To do this place the cursor inside the cell and execute this code with the run button or with Shift + Return.

```
In [ ]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import datetime
        import time
        from sklearn import metrics
        from sklearn import neighbors
        from sklearn import ensemble
        from sklearn import tree
        from sklearn import linear_model
        from sklearn.linear_model import LogisticRegression
        from pandas.plotting import scatter_matrix
        from sklearn.metrics import classification_report
        from matplotlib import pyplot as plt
        from datetime import datetime, date, time, timedelta
        from sklearn.preprocessing import StandardScaler
        from sklearn.feature_selection import SelectKBest
        from sklearn.model_selection import train_test_split
        import matplotlib.ticker as mtick
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import f1_score
        from sklearn import svm
        plt.style.use('ggplot')
        #_____________________ Added to be able to load files from the Cloud Storage _____________________
        import dsx_core_utils, requests, os, io
        from dsx_core_utils import ProjectContext
        from pyspark.sql import SparkSession
        from pyspark import SparkContext
```

The correct execution of this cell will assign a number to it in `In [ ]:`

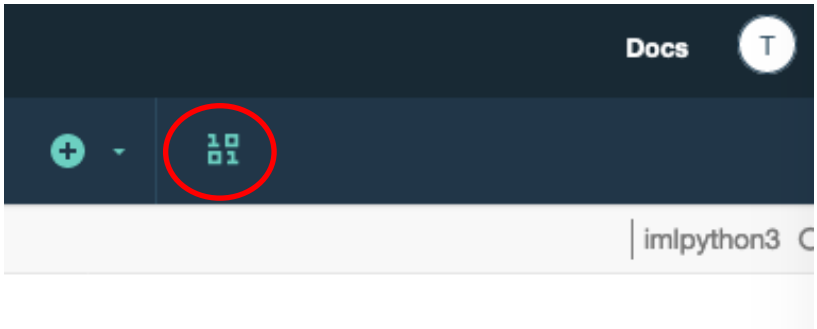
__ 5. CELL 2. Place the cursor inside this cell by clicking on it:

Import and Update tables

We load the Db2 table CARD and make some small changes like formatting the date in the issued column and the card type into numbers.

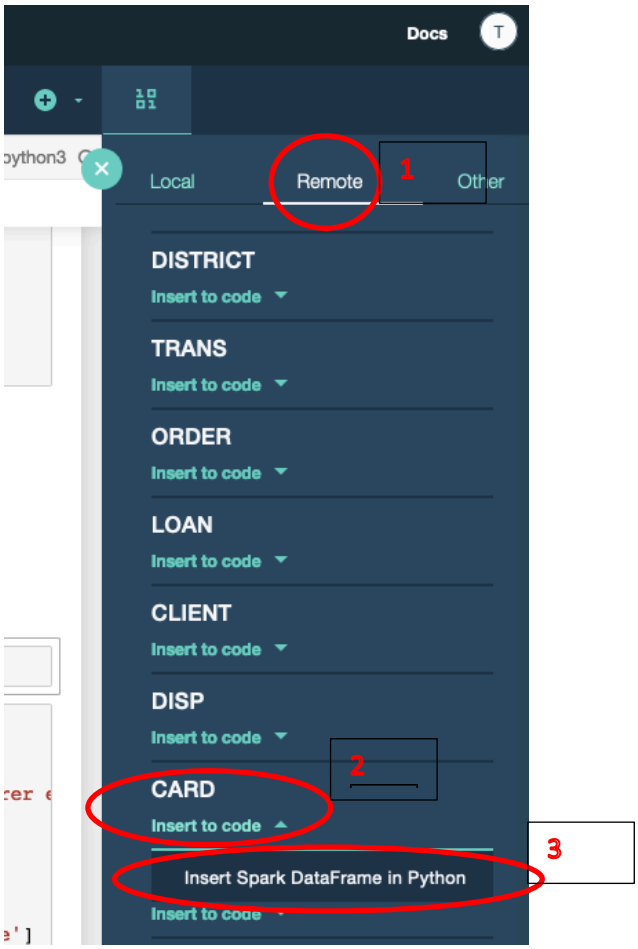
```
In [ ]: # Insert code to load CARD data in this cell
```

__ 6. Go to the upper right corner and click on an icon with 10 01



It will open a side panel.

__ 7. Select REMOTE and find the CARD table by clicking on Insert to code.



Then click into Insert Spark DataFrame in Python

__ 8. CELL 2. Code will be inserted into the cell. This code will create an authorized connection to Db2 for z/OS and copy all data in the CARD table.

```
In [ ]: # Insert code to load CARD data in this cell

import dsx_core_utils, requests, os, io
from dsx_core_utils import ProjectContext
from pyspark.sql import SparkSession
from pyspark import SparkContext
# initialize the SparkContext and projectContext
sc = SparkContext.getOrCreate()
sparkSession = SparkSession(sc).builder.getOrCreate()
pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPrediction-Copyfor-LAB2', 'Bearer eyJ0eX
# Add asset from remote connection
df_data_1 = None
dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'CARD')
dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
# Load JDBC data to Spark dataframe
dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
df_data_1 = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSou
df_data_1.show(5)
```

__ 9. CELL2. We need to edit this code now to keep a nomenclature and order in the programming logic.

Select with your mouse all code above the comment line: `# Initialize the SparkContect and projectContext`

```
In [ ]: # Insert code to load CARD data in this cell

        import dsx_core_utils, requests, os, io
        from dsx_core_utils import ProjectContext
        from pyspark.sql import SparkSession
        from pyspark import SparkContext
        # initialize the SparkContext and projectContext
        sc = SparkContext.getOrCreate()
        sparkSession = SparkSession(sc).builder.getOrCreate()
        pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPredictio
```

Delete that code.

```
In [ ]: # Insert code to load CARD data in this cell

        # initialize the SparkContext and projectContext
        sc = SparkContext.getOrCreate()
        sparkSession = SparkSession(sc).builder.getOrCreate()
        pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPredictio
        # Add asset from remote connection
        df_data_1 = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'CARD')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) !=
        df_data_1 = sparkSession.read.format("jdbc").option("driver",dataSource['driver
        df_data_1.show(5)
```

__10. CELL 2. Change the code to customize the CARD dataframe you are building with Pandas.

We load the Db2 table CARD and make some small changes like formatting the date in the issued column a

```
In [ ]:
        # initialize the SparkContext and projectContext
        sc = SparkContext.getOrCreate()
        sparkSession = SparkSession(sc).builder.getOrCreate()
        pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPrediction
        # Add asset from remote connection
        df_data_1 = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'CARD')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0
        df_data_1 = sparkSession.read.format("jdbc").option("driver",dataSource['driver_
        df_data_1.show(5)
```

Wherever `df_data_1` appears in this block of code, change it to `card_raw`.
Also delete the last line of code: `df_data_1.show()`

Add a line of code by the end of the block: `card = card_raw.toPandas()`

The result should be something like this:

Then execute the cell with the run button or with Shift + Return.

__ 11. CELL 3. Review the next cell:

This code is intended to modify the column `issued` in this dataframe. It first wipes out all the `00:00:00` and then changes the values of the `type` column from gold to 2, classic to 1 and junior to 0.

The last line shows you the first five rows of the dataframe.

Then execute the cell with the run button or with Shift + Return.

__ 12. CELL 4. Carefully study next cell:

The same process followed for previous cell has been done for you.

- Inserted code to create a Spark DataFrame from a remote source inside Python. This remote source is the table ACCOUNT in Db2 for z/OS.
- Changed the default name for the new Spark DataFrame to `account_raw`.

- Converted the Spark DataFrame to a Pandas dataframe using `account = account_raw.to Pandas()`.
- Formatted the `date` column because originally is not in the same format as the rest of the tables.
- Visualize the first 5 rows of the dataframe to have an idea of what this data looks like.

This process of loading and formatting will be followed for all the rest of the tables. Some comments in code have been introduced for you to understand some of the changes. Python is not always as readable as its fame states!

Execute the cell with the run button or with Shift + Return.

__ 13. CELL 5. Execute cell to load table DISP

```
In [ ]: # Load table DISP

        # Add asset from remote connection
        disp_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'DISP')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
        disp_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).option

        disp = disp_raw.toPandas()

        # We select all rows where the value for type is OWNER then change the name of the column
        disp = disp[disp.type == "OWNER"]
        disp.rename(columns={"type": "type_disp"}, inplace=True)
        disp.head()
```

and examine the result. Use the run button or Shift + Return.

__ 14. CELL 6. Execute the cell to load table CLIENT

```
In [ ]: # Load table CLIENT

        # Add asset from remote connection
        client_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'CLIENT')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
        client_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).option

        client = client_raw.toPandas()

        # This is an interesting extraction of information which is coded in the birth date for customers:
        # the number is in the form YYMMDD for men,
        # the number is in the form YYMM+50DD for women
        # where YYMMDD is the date of birth
        # with the following code we extract the sex information from the birth dates and add a new
        # column with the sex: 0 for female, 1 for male.
        # Also we drop some columns: birth_number, month, year

        client["month"] = client.birth_number.apply(
            lambda x: x // 100 % 100, convert_dtype=True, args=())
        client["year"] = client.birth_number.apply(
            lambda x: x // 100 // 100, convert_dtype=True, args=())
        client["age"] = 99 - client.year # age in 1999
        client["sex"] = client.month.apply(lambda x: (x - 50) < 0, convert_dtype=True, args=())
        client.sex = client.sex.astype(int)  # 0 for female, 1 for male
        client.drop(["birth_number", "month", "year"], axis=1, inplace=True)
        client.head()
```

Read code comments with attention, they explain what are the modifications done to the table. Use the run button or Shift + Return.

__ 15. CELL 7. Execute the cell to load table DISTRICT

```
In [ ]: # Load table DISTRICT

        # Add asset from remote connection
        district_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'DISTRICT')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
        district_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).option

        district = district_raw.toPandas()

        # Drop columns A2 and A3
        district.drop(["A2", "A3"], axis=1, inplace=True)
        district.head()
```

Use the run button or Shift + Return.

__ 16. CELL 8. Execute the cell to load table ORDER

```
In [ ]: # Load table ORDER

        # Add asset from remote connection
        order_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'ORDER')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
        order_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).option

        order = order_raw.toPandas()

        # This code rearranges the dataframe by dropping some columns, filling blanks and empty records,
        # renaming some other columns and resetting the index column.
        # Renaming some columns has been done for the sake of clarity in English. The names have been translated
        # from Czech to English

        order.drop(["bank_to", "account_to", "order_id"], axis=1, inplace=True)
        order.k_symbol.fillna("No_symbol")
        order.k_symbol = order.k_symbol.str.replace(" ", "No_symbol")
        order = order.groupby(["account_id", "k_symbol"]).mean().unstack()
        order = order.fillna(0)
        order.columns = order.columns.droplevel()
        order.reset_index(level="account_id", col_level=1, inplace=True)
        order.rename_axis("", axis="columns", inplace=True)
        order.rename(
            index=None,
            columns={
                "LEASING": "order_amount_LEASING_PAYMENT",
                "No_symbol": "order_amount_No_symbol",
                "POJISTNE": "order_amount_INSURANCE_PAYMENT",
                "SIPO": "order_amount_HOUSEHOLD_PAYMENT",
                "UVER": "order_amount_LOAN_PAYMENT",
            },
            inplace=True)

        order.head()
```

Read code comments with attention, they explain what are the modifications done to the table.
Use the run button or Shift + Return.

__ 17. CELL 9. Execute the cell to load table LOAN

```
In [ ]: # Load table LOAN

        # Add asset from remote connection
        loan_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'LOAN')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
        loan_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).option

        # Date formatting using lambda function as before.

        loan = loan_raw.toPandas()
        loan.date = loan.date.apply(lambda x: pd.to_datetime(str(x), format="%y%m%d"))
        loan.head()
```

Read all code comments, they contain explanations about the modifications done to the table.
Use the run button or Shift + Return.

__ 18. CELL 10. Execute the cell to check what are the values contained in column `status`

```
In [ ]: loan.status.value_counts()
```

you should see something like this:

```
[12]: C    403
      A    203
      D     45
      B     31
      Name: status, dtype: int64
```

This column will be important later when using the machine learning models.
Use the run button or Shift + Return.

__ 19. CELL 11. Finally, execute the cell to load table TRANS. This is the biggest table it will take longer than the others, give it a while.

```
In [ ]: # Load table TRANS

        # initialize the SparkContext and projectContext
        sc = SparkContext.getOrCreate()
        sparkSession = SparkSession(sc).builder.getOrCreate()
        pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPrediction-Copyfor-LAB2.ipynb
        # Add asset from remote connection
        trans_raw = None
        dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'TRANS')
        dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
        # Load JDBC data to Spark dataframe
        dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSe
        trans_raw = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("ur

        trans = trans_raw.toPandas()

        trans.loc[trans.k_symbol == "", "k_symbol"] = trans[trans.k_symbol == ""].k_symbol.apply(lambda x:
```

Read with attention the comments explaining what are the modifications done to the table. Use the run button or Shift + Return

__ 20. CELL 12. Execute the cell to start manipulating data with code. Read the comments inside the code to understand what is going on.

```
Once we have loaded all files with data and made the initial arrangements we start "changing" data into something understandable by the models: creatin

In [ ]: # create temp table trans_pv_k_symbol
        # Table is pivoted

        trans_pv_k_symbol = trans.pivot_table(values=["amount", "balance"], index=["trans_id"], columns="k_symbol")
        trans_pv_k_symbol.fillna(0, inplace=True)
        trans_pv_k_symbol.columns = ["_".join(col) for col in trans_pv_k_symbol.columns]
        trans_pv_k_symbol = trans_pv_k_symbol.reset_index()
        trans_pv_k_symbol = trans.iloc[:, :3].merge(trans_pv_k_symbol, how="left", on="trans_id")
        trans_pv_k_symbol.head()
```

Look at the result to have an idea of what is happening with the data. Use the run button or Shift + Return to execute the cell.

__21. CELL 13. As always read code comments, then insert the following code after the final comment: `get_date_loan_trans.head(10)`

```
In [ ]: # create temp table get_date_loan_trans
        # First merge loan and account

        get_date_loan_trans = pd.merge(
            loan,
            account,
            how="left",
            on="account_id",
            left_on=None,
            right_on=None,
            left_index=False,
            right_index=False,
            sort=False,
            suffixes=("_loan", "_account"),
            copy=True,
            indicator=False,
            validate=None)

        # Then merge the former with loan-account with trans. All of them use account_id as the column guiding the merging.

        get_date_loan_trans = pd.merge(
            get_date_loan_trans,
            trans,
            how="left",
            on="account_id",
            left_on=None,
            right_on=None,
            left_index=False,
            right_index=False,
            sort=False,
            suffixes=("_account", "_trans"),
            copy=True,
            indicator=False,
            validate=None)

        # Insert here the head() method to view how merging has configured the new table
```

Execute the cell using the run button or Shift + Return.
Examine the result of merging the tables. You have set the code to view 10 lines instead of 5 which is the default when you don't write a number for the `head()` method.

__22. CELL 14. Do some calculation to know the number of days between the loan date and the transfer date. Insert this code before the comment:
`(get_date_loan_trans.date_loan - get_date_loan_trans.date)`

```
In [ ]: # update table get_date_loan_trans to get the date between loan_date and trans_date. Then format it to leave a plain number of days.

        get_date_loan_trans["date_loan_trans"] =  #### Insert code before this comment ####
        get_date_loan_trans[["date_loan_trans"]] = get_date_loan_trans[["date_loan_trans"]].astype(str)

        # Format the new column to set the number of days as a number
        get_date_loan_trans.date_loan_trans = get_date_loan_trans.date_loan_trans.str.strip(" days 00:00:00.000000000")
        get_date_loan_trans.date_loan_trans = pd.to_numeric(get_date_loan_trans.date_loan_trans.str.strip(" days +"))
        get_date_loan_trans.head()
```

Execute the cell using the run button or Shift + Return.
Examine the result. You will see that there a new column called `date_loan_trans` by the end of the dataframe.

__23. CELL 15. Insert code to view the last 7 rows of the temporary table for 90 days balance. The dataframe name is `temp_90_mean`

```
In [ ]: # create temp table temp_90_mean to create new feature
        temp_90_mean = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0) & (get_date_loan_trans["date_loan_trans"] < 90)]
        temp_90_mean = temp_90_mean.drop(["trans_id", "k_symbol"], axis=1)
        temp_90_mean = temp_90_mean.groupby(["loan_id"], as_index=None).mean()
        temp_90_mean = temp_90_mean.loc[:, ["loan_id", "balance"]]
        temp_90_mean.rename(index=None, columns={"balance": "avg_balance_3M_before_loan"}, inplace=True)

        #### Insert code to view the 7 last lines of the dataframe
```

use the `tail()` method.
Execute the cell using the run button or Shift + Return.
Examine the result. A temporary dataframe with average balances for the las 90 days before requesting a loan has been created.

__24. CELL 16. Same as in step __21. This time the calculation is to average balance for the 30 days before requesting a loan. Dataframe name is `temp_30_mean`:

```
In [ ]: # create temp table temp_30_mean to create new feature
temp_30_mean = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0) & (get_date_loan_trans["date_loan_trans"] < 30)]
temp_30_mean = temp_30_mean.drop(["trans_id", "k_symbol"], axis=1)
temp_30_mean = temp_30_mean.groupby(["loan_id"], as_index=None).mean()
temp_30_mean = temp_30_mean.loc[:, ["loan_id", "balance"]]
temp_30_mean.rename(index=None, columns={"balance": "avg_balance_1M_before_loan"}, inplace=True)

#### Insert code to view the 7 last lines of the dataframe
```

Again, use the `tail()` method.
Execute the cell using the run button or Shift + Return.

__25. CELL 17. More temporary data is created out of the tables.

```
In [ ]: # create temp table temp_trans_freq to create new feature.
# A frequency of movements is created
temp_before = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0)]
temp_trans_freq = (temp_before.loc[:, ["loan_id", "trans_id"]].groupby(["loan_id"], as_index=None).count())
temp_trans_freq.rename(index=None, columns={"trans_id": "trans_freq"}, inplace=True)
temp_before = temp_before.drop(["trans_id", "k_symbol"], axis=1)
```

Execute the cell using the run button or Shift + Return.

__26. CELL 18. Minimum and average balances are calculated for each of the loans

```
In [ ]: # create temp table temp_balance_min & temp_balance_mean to create new features
# Minimun and average balances are added for each of the loans

temp_balance_min = (
    temp_before.groupby(["loan_id"], as_index=None).min().loc[:, ["loan_id", "balance"]]
)
temp_balance_min.rename(
    index=None, columns={"balance": "min_balance_before_loan"}, inplace=True
)

temp_balance_mean = (
    temp_before.groupby(["loan_id"], as_index=None)
    .mean()
    .loc[:, ["loan_id", "amount_trans", "balance"]]
)
temp_balance_mean.rename(
    index=None,
    columns={
        "amount_trans": "avg_amount_trans_before_loan",
        "balance": "avg_balance_before_loan",
    },
    inplace=True,
)

temp_balance_mean.head()
```

Execute the cell using the run button or Shift + Return.
Examine the result.

__27. CELL 19. Low balance thresholds are calculated and put in temporary tables

```
In [ ]: # create temp table times_balance_below_500 & times_balance_below_5K to create new features

times_balance_below_500 = temp_before[temp_before.balance < 500]
times_balance_below_500 = (
    times_balance_below_500.groupby(["loan_id"], as_index=None)
    .count()
    .loc[:, ["loan_id", "balance"]]
)
times_balance_below_500 = times_balance_below_500[times_balance_below_500.balance > 1]
times_balance_below_500.rename(
    index=str, columns={"balance": "times_balance_below_500"}, inplace=True
)

times_balance_below_5K = temp_before[temp_before.balance < 5000]
times_balance_below_5K = (
    times_balance_below_5K.groupby(["loan_id"], as_index=None)
    .count()
    .loc[:, ["loan_id", "balance"]]
)
times_balance_below_5K = times_balance_below_5K[times_balance_below_5K.balance > 1]
times_balance_below_5K.rename(
    index=str, columns={"balance": "times_balance_below_5K"}, inplace=True
)
```

Examine the code and then execute the cell using the run button or Shift + Return.

__28. CELL 20. Merge all former temporary tables into one. Add the following code after first comment `#### Insert code below:`

```
merge_loan_trans = merge_loan_trans.merge(times_balance_below_5K, how="left", on="loan_id")
```

```
In [ ]: # create temp table merge_loan_trans to merge the temp features above into one temp table
merge_loan_trans = loan.merge(
    temp_90_mean, how="left", on="loan_id", suffixes=("_loan", "_trans")
)
merge_loan_trans = merge_loan_trans.merge(temp_30_mean, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_trans_freq, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_balance_min, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(temp_balance_mean, how="left", on="loan_id")
merge_loan_trans = merge_loan_trans.merge(
    times_balance_below_500, how="left", on="loan_id"
)

#### Insert code below to merge the table times_balance_below_5K into the table merge_loan_trans on the left and usi



#### Visualize the resulting dataframe using head() method

```

With the code just inserted you will merge temporary dataframe `times_balance_below_5K` into the main dataframe `merge_loan_trans`, on the left using column loan_id as guide.

Then insert the code to check the resulting dataframe: `merge_loan_trans.head()`

Execute the cell using the run button or Shift + Return.

Examine the results.

__29. CELL 21. Do you remember the meaning of the status column in LOAN table. You made some visualization in lab 1.
A → Contract finished, no problem
B → Contract finished, loan not paid
C → Contract running, OK so far
D → Contract running, client in debit.

Insert the following code in the cell:
`len(loan_BorD)`

```
In [ ]: # | (pipe) operator is the union of sets when used with dataframes

        loan_BorD = loan[(loan.status == "D") | (loan.status == "B")]

        #### Insert code below. Need to know how maby rows are there after subsetting D and B status.
```

You will obtain the number of loans with status D or B from the LOAN table.

Execute the cell using the run button or Shift + Return.

__30. CELL 22. Merge LOAN table with times_balance_below_500. Insert code below the comment:
`temp.status.value_counts()`

```
In [ ]: temp = times_balance_below_500.merge(
            loan,
            how="inner",
            on="loan_id",
            left_on=None,
            right_on=None,
            left_index=False,
            right_index=False,
            sort=False,
            suffixes=("_x", "_y"),
            copy=True,
            indicator=False,
            validate=None)

        #### Insert code below
```

Execute the cell using the run button or Shift + Return.
Examine the results.

__31. CELL 23. Plot `status` vs `balance_under_500` to have an idea of the relation between these columns
Insert this code below the comment:
`plt.plot(temp.status, temp.times_balance_below_500, "ro")`

```
In [ ]: #### Insert code below to plot status vs times_balance_below_500
```

Execute the cell using the run button or Shift + Return.
Examine the results.

__32. CELL 24. Plot `status` vs `times_balance_below_500` after reordering by times balance in descending order.

```
In [ ]: # We get different views to try to understand the behaviour of payments and balances

        temp.sort_values("times_balance_below_500", ascending=False).plot(x="status", y="times_balance_below_500", kind="bar")
```

Execute the cell using the run button or Shift + Return.
Examine the results.

__33. CELL 25. Select all rows but just the columns payments and status and then check that the selection is correct by viewing the first 3 rows. Insert this code below the comment:

```
t = loan.loc[:, ["payments", "status"]]
t.head(3)
```

```
In [ ]: #### Insert code below. Select all rows but just two columns: payments and status. Check that it's correct viewing the 3 first rows
```

Execute the cell using the run button or Shift + Return.
Examine the results.

__34. CELL 26. Plot the `t` dataframe to check how payments are given a status.
Insert this code below the comment:

```
t.plot(x='status',kind="bar")
```

```
In [ ]: # Group the 4 status and take a mean of the payments made in each of the four groups

        t = t.groupby(["status"], as_index=None).mean()

        #### Insert code below. Plot the former calculation with status in x axis
```

Execute the cell using the run button or Shift + Return.
Examine the results.

Section 2. Merging all useful dataframes into one

Now that we have an idea of the behaviour of the data with related to some of the variables, we may start creating the final dataframe that will be used for the machine learning models.

__ 1. CELL 27. Create a new dataframe, `df`, that will contain all the necessary data for the machine learning models. At the same time, merge `account` dataframe with `df`.

```
In [ ]: # Merge the created merge_loan_trans with account. Have a look to the options used to merge.

df = pd.merge(
    merge_loan_trans,
    account,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_loan", "_account"),
    copy=True,
    indicator=False,
    validate=None)
```

Execute the cell using the run button or Shift + Return.

__2. CELL 28. Merge `order` dataframe with `df` by inserting this code below the comment in cell:

```
df = pd.merge(
    df,
    order,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_a", "_order"),
    copy=True,
    indicator=False,
    validate=None)
```

```
In [ ]: # Add order to the new dataframe df

#### Insert code below. Merge the already created dataframe, df, with the dataframe for the ACCOUNT table, whose name is account too.
```

Execute the cell using the run button or Shift + Return.

__3. CELL 29. Merge `disp` with `df`.

```
In [ ]: # Add disp

df = pd.merge(
    df,
    disp,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_b", "_disp"),
    copy=True,
    indicator=False,
    validate=None)
```

Execute the cell using the run button or Shift + Return.

__4. CELL 30. Merge `card` with `df`.

```
In [ ]: # Add card

df = pd.merge(
    df,
    card,
    how="left",
    on="disp_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_c", "_card"),
    copy=True,
    indicator=False,
    validate=None)
```

Execute the cell using the run button or Shift + Return.

__5. CELL 31. Merge `df` with `client` dataframe.
Insert this code below the comment in the cell.

```
df = pd.merge(
    df,
    client,
    how="left",
    on="client_id",
    left_on=None,
    right_on=None,
```

```
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_d", "_client"),
    copy=True,
    indicator=False,
    validate=None)
```

```
In [ ]: # Add client

        #### Insert code below. Merge df with client.
```

Execute the cell using the run button or Shift + Return.

__6. CELL 32. Merge `df` with `district` dataframe.
Insert this code below the comment in the cell.

```
df = pd.merge(
    df,
    district,
    how="left",
    left_on="district_id_client",
    right_on="A1",
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_e", "_district"),
    copy=True,
    indicator=False,
    validate=None)
```

```
In [ ]: # Add district

        #### Insert code below. Merge df with district.
```

Execute the cell using the run button or Shift + Return.

__7. CELL 33. Some arrangements have to be done to `before_loan_date` and `trans_pv_k_symbol` before finishing our `df` dataframe.

```
In [ ]: # Some arrangements before merging with df. We use before_loan_date and trans_pv_k_symbol after some selection from before_loan_date

before_loan_date = get_date_loan_trans[(get_date_loan_trans["date_loan_trans"] >= 0)]
before_loan_date = before_loan_date.loc[:, ["account_id", "trans_id"]]
trans_pv_k_symbol = pd.merge(
    before_loan_date,
    trans_pv_k_symbol,
    how="left",
    on="trans_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_before", "_df2"),
    copy=True,
    indicator=False,
    validate=None)

trans_pv_k_symbol.drop(["account_id_df2", "date", "trans_id"], axis=1, inplace=True)

trans_pv_k_symbol.rename(columns={"account_id_before": "account_id"}, inplace=True)

trans_pv_k_symbol = trans_pv_k_symbol.groupby(by="account_id", axis=0, as_index=False, sort=True, group_keys=True, squeeze=False).mean()
```

Examine the code to understand what are the manipulations done with it.
Execute the cell using the run button or Shift + Return.

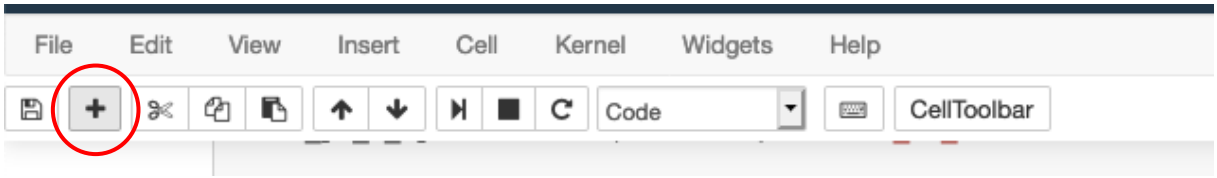
__8. CELL 34. Finally merge `df` with `trans_pv_k_symbol`.
Insert this code below the comment in the cell:

```
df = pd.merge(
    df,
    trans_pv_k_symbol,
    how="left",
    on="account_id",
    left_on=None,
    right_on=None,
    left_index=False,
    right_index=False,
    sort=False,
    suffixes=("_df", "_tt"),
    copy=True,
    indicator=False,
    validate=None)
```

```
In [ ]: # And with this merge we have our final df

        #### Insert code below. Merge df with trans_pv_k_symbol

        |
```

Execute the cell using the run button or Shift + Return.

__9. Insert a new cell below the last one by clicking on the + icon in the Jupyter menu.



Then insert the code to visualize the first 10 lines of the newly created dataframe `df`.
Execute the cell using the run button or Shift + Return.

Examine the results. Do you think there is any more work to be done on this dataframe?

Section 3. Data Cleaning

We finish this lab 2 by cleaning our final set of data for the machine learning models.

__1. CELL 35 TO 40. Look into the next 6 cells under the Data Cleaning title. Execute them in order and insert code to view how the dataframe is changing with the execution.

In the last cell you'll find the code `df.info()` which is useful to review all the dataframe parameters. Using it you can see the names of all columns, types for each of them, the number of rows and if there are nulls or NaN's in each column.

Section 4. Get Label

Last changes to the dataset before saving it for lab 3.

__1. CELL 41 TO 43. Change `status` categorical values to numeric. Check the cell and insert code below the comment:

CELL41

```
df.status.unique()
```

```
In [ ]: # Change categorical variables into numerical ones. There are ML model not admitting categoricals

m = {"A": 0, "B": 1, "C": 0, "D": 1}
df.status = df.status.map(m)

#### Insert code below. Check that the unique values are 0 and 1
```

Execute the cell using the run button or Shift + Return.

__2. CELL 44. Finally drop some columns which are not interesting for our interests. Insert code below the comment:

```
df.drop(
    [
        "loan_id",
        "account_id",
        "district_id_d",
        "disp_id",
        "client_id",
        "card_id",
        "district_id_client",
    ],
    axis=1,
    inplace=True)
```

```
In [ ]: #### Insert code below. Drop columns


# Save the shining df dataframe into a CSV file to reuse it as needed

df.to_csv('fullDataframe_EOLAB1_<login_userid>.csv', sep=';', encoding='utf-8',index = False)
```

Change the < login_userid> for the correct value of your team.

Execute the cell using the run button or Shift + Return.

You completed lab 2.

Lab 3: Modeling and training from a dataframe

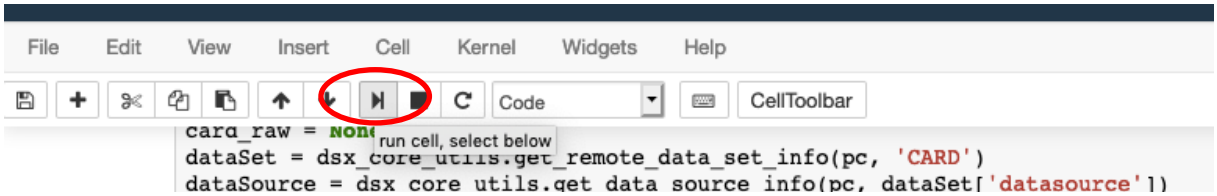
This lab will guide you through a Jupyter notebook inside the development environment of the Watson Machine Learning for z/OS. The goal of this notebook is to use the dataframe prepared in lab 2 to train and build machine learning models. We are comparing random forest models, gradient boost and logistic regression to choose the one giving best results.

Note: You need your worksheet to login into the system and follow the name convention. The worksheet provides information that is specific to each student to use throughout the lab. Some of the instructions in the lab instruct the student to use a value that is marked with angle brackets (< >). The value the student should use is on their lab worksheet.

→ Make sure to exclude the brackets when using the value for its intended purpose in the lab.

→ Along the lab you will have to execute Python code in cells, to do this you have two options

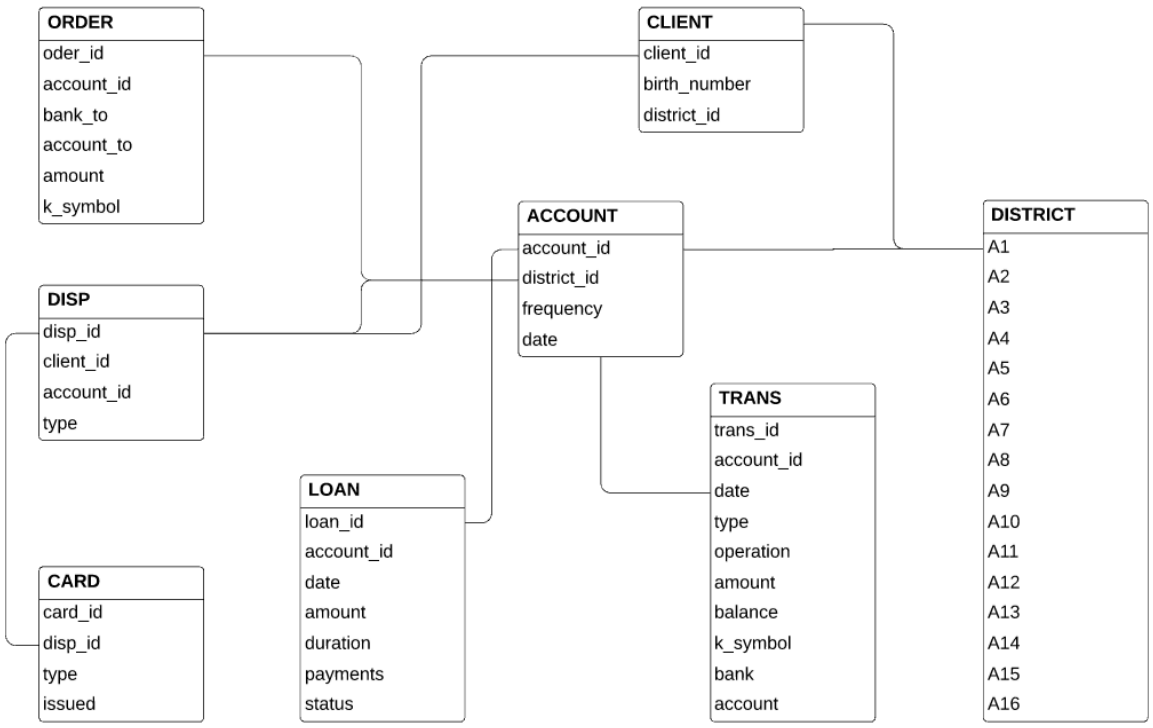
- Click on the run icon on the menu



- Pulse Shift + Return on the keyboard.

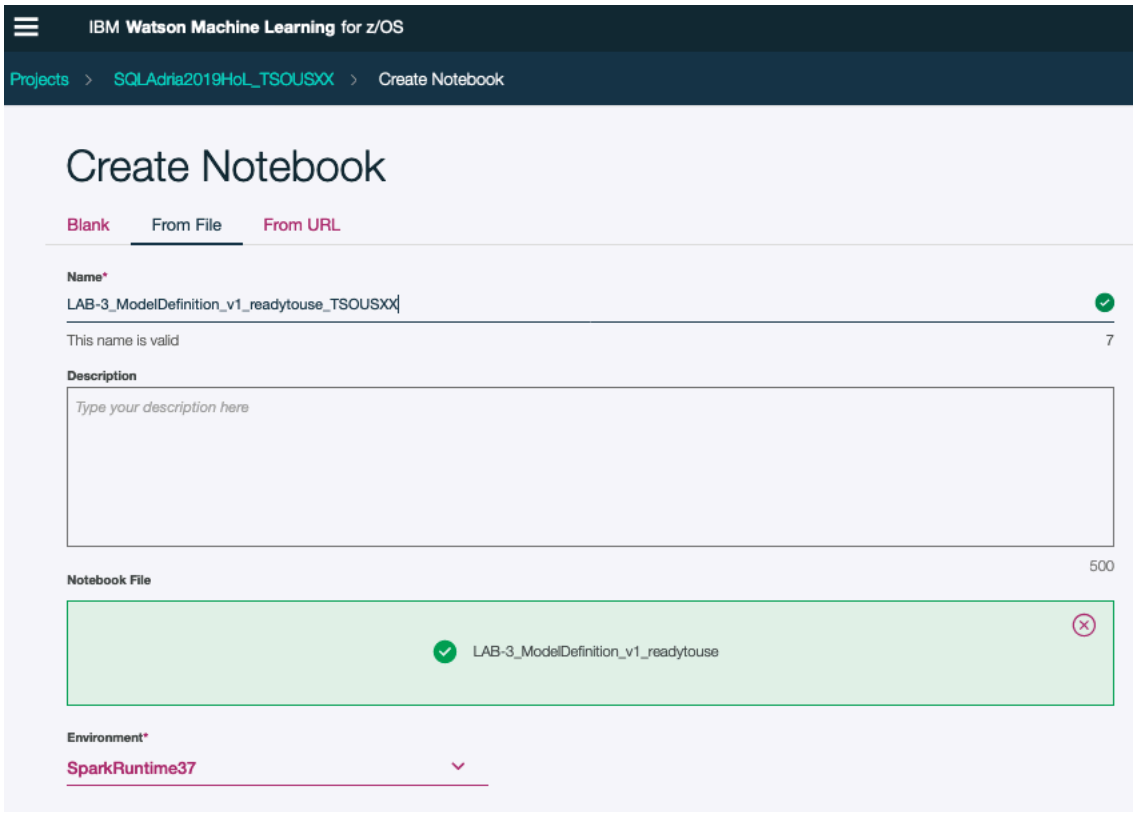
Section 1. Initial set up

You remember from lab 1 that the tables we will use are:



__ 1. Open your project with your <login_userid> created in lab 1 and go to your assets list. At the right side of the Notebooks section, click on ⊕ add notebook

__ 2. In the below screen:



- Select **From File** in the 3 available options
- Drag and drop the file **LAB-3_ModelDefinition_v2_readytouse.ipynb** located on your desktop into the Notebook File section.
- **Once you have the Notebook File added to its section**, add _<login_userid> to the Notebook Name created by default
- Leave the Environment as it's
- And click on **Create.**

__ 3. CELL 1. Read the introduction and go to the first cell in the notebook. The same `import` sentences used in lab 2 are used here.

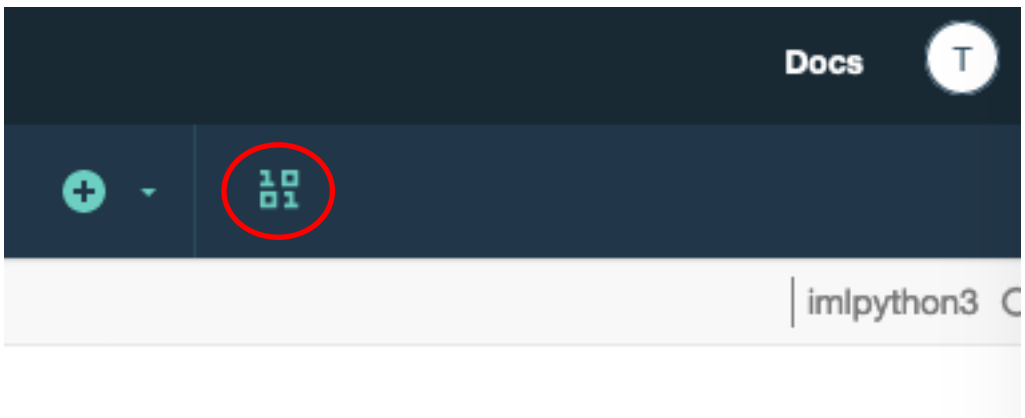
```
In [ ]: # CELL 1

import pandas as pd
import numpy as np
import seaborn as sns
import datetime
import time
from sklearn import metrics
from sklearn import neighbors
from sklearn import ensemble
from sklearn import tree
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from pandas.plotting import scatter_matrix
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
from datetime import datetime, date, time, timedelta
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
import matplotlib.ticker as mtick
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn import svm
plt.style.use('ggplot')

import pandas as pd
import dsx_core_utils
from dsx_core_utils import ProjectContext
```

Execute the cell using the Run button or Shift-Enter.

__4. CELL 2. Click on the empty cell with comments to insert code. Please click just under the first of the comments to avoid code clutter.

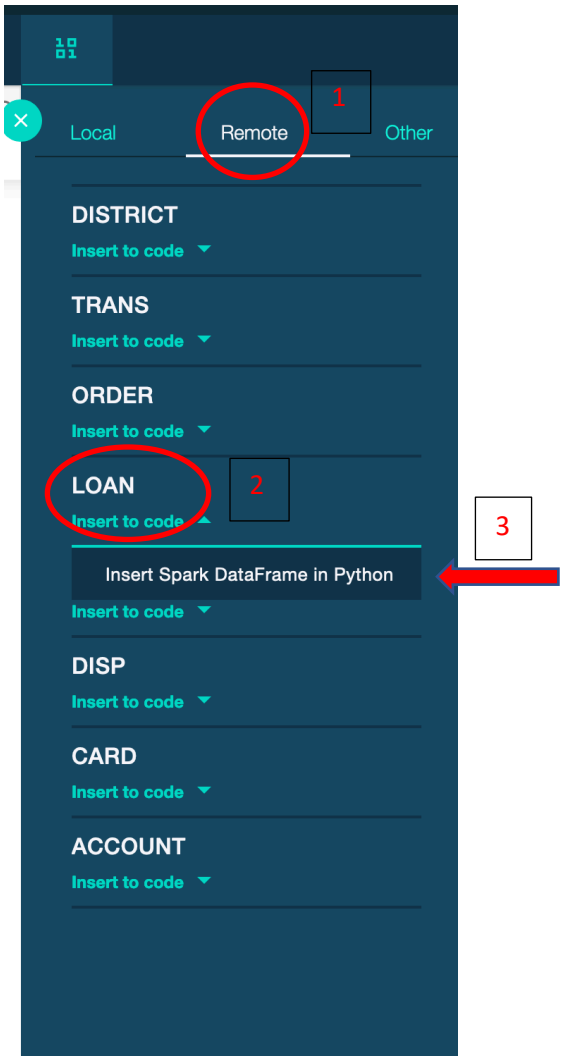
```
In [ ]: # CELL 2

#### Insert code below. Load LOAN table from Db2 as you did in LAB2.

#### Insert code from the lab script here to load convert loan_raw into loan with the required options
```

__5. Go to the upper right corner and click on an icon with 10 01



It will open a side panel.

__6. Click on the Local label of the side panel (1). You will see all the tables ready for you in the Db2 database connected to this project.
Select LOAN table by clicking on insert to code under the name of the file (2).
Finally click on Insert Spark DataFrame in Python (3).



__7. CELL 2. Delete the imports inserted by the code since they have already been included in the import group at the beginning.

```
In [ ]: # CELL 2

#### Insert code below. Load LOAN table from Db2 as you did in LAB2.
import dsx_core_utils, requests, os, io
from dsx_core_utils import ProjectContext
from pyspark.sql import SparkSession
from pyspark import SparkContext
# initialize the SparkContext and projectContext
sc = SparkContext.getOrCreate()
sparkSession = SparkSession(sc).builder.getOrCreate()
pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'LAB-3_ModelDefinition_v2_readytouse_fullcode', 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ1c2VybmFt
# Add asset from remote connection
df_data_1 = None
dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'LOAN')
dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
# Load JDBC data to Spark dataframe
dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
df_data_1 = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).opti
df_data_1.show(5)
```

__8. CELL 2. Edit the newly inserted code to adequate it to our naming conventions.

```
In [ ]: # CELL 2

#### Insert code below. Load LOAN table from Db2 as you did in LAB2.

# initialize the SparkContext and projectContext
sc = SparkContext.getOrCreate()
sparkSession = SparkSession(sc).builder.getOrCreate()
pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'LAB-3_ModelDefinition_v2_readytouse_fullcode', 'Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ1c2VybmFt
# Add asset from remote connection
df_data_1 = None
dataSet = dsx_core_utils.get_remote_data_set_info(pc, 'LOAN')
dataSource = dsx_core_utils.get_data_source_info(pc, dataSet['datasource'])
# Load JDBC data to Spark dataframe
dbTableOrQuery = (dataSet['schema'] + '.' if(len(dataSet['schema'].strip()) != 0) else '') + dataSet['table']
df_data_1 = sparkSession.read.format("jdbc").option("driver",dataSource['driver_class']).option("url", dataSource['URL']).option("dbtable",dbTableOrQuery).opti
df_data_1.show(5)


#### Insert code from the lab script here to load convert loan_raw into loan with the required options
```

Change the name `df_data_1` for `loan_raw`
Delete the line:
`loan_raw.show(5)` (or `df_data_1.show(5)` if you have not changed the name yet)

__9. CELL 2. Add the following code by the end of the cell to change date format.

```
loan = loan_raw.toPandas()
loan.date = loan.date.apply(lambda x: pd.to_datetime(str(x),
format="%y%m%d"))
loan.head()
```

```
In [ ]: #### Insert code below. Load loan.csv file from the file repository of Watson Machine Learning for z/OS

# Add asset from file system
pc = ProjectContext.ProjectContext('SQLAdria2019HoL', 'BankLoanDefaultPrediction-English-Copyfor-LAB3', 'Be
loan_raw = dsx_core_utils.get_local_dataset(pc, 'loan.csv')

#### Insert code from the lab script here to load convert loan_raw into loan with the required options
```

Execute the cell using the Run button or Shift-Enter.

__10. CELL 3. Execute cell with command %`pwd` to check that the notebook is working in this path: `/u/wmlscors.v2r1m0/imlpython/env/mlzenv37`

__11. CELL 4. Load the csv file you saved in lab 2 when data was prepared

```
In [ ]: df = pd.read_csv(
            'fullDataframe_EOLAB1_<login_userid>.csv',
            sep=";",
            delimiter=None,
            header="infer",
            names=None,
            low_memory=False)

        df.head()
```

Note: If you did not succeed in lab 2, use **test** instead of <login_userid>.

Ensure that you change <login_userid> to the value for your team.

Execute the cell and check that the dataframe looks like the one you saved.

Section 2. Standard Processing and Training

We use part of our dataframe to train models and the remaining part to evaluate.

__1. CELL 5. Insert this code to split data into 2 sets. 3% of the dataframe will be used to train the model:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
```

```
In [ ]: # Select status column as the y (our target) and all the rest as X (features)

X = df.loc[:, df.columns != "status"]
y = df.loc[:, "status"]

#### Insert code below. Split the dataframe in 2 sets. 3% of the data will be used for training
```

If you're willing to have a sight on the data splitted you may add `X.describe()` to see the basic statistics about the `X` dataframe. You can do accordingly with `X_train` and `X_test`.

The y is the target variable we would like to predict. In this case we want to know the predicted status of the customer and score it with probability of default or not default.

Execute the cell.

__2. CELL 6. Read carefully the text in the notebook, there is a clarification about why you are training a "draft model" to start.
Train your draft random forest model by inserting this code below the comment:

```
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```
In [ ]: # This is how a model is trained. Nothing strange: just invoke the algorithm with the necessary p
# you may visit the documentation centre: https://scikit-learn.org/stable/modules/generated/sklea

rf = ensemble.RandomForestClassifier(
    n_estimators=200,
    criterion="gini",
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features="auto",
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=1,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None
)

#### Insert code below. Train the model created above and get a target prediction.
```

Observe that, with this 2 lines of code, you first fit the model using the part of the dataframe marked as `X_train`, `y_train`, then you get an estimation of the target using the rest of the dataframe, `X_test`. Remember that your target is to know if any of the rows will be in a status B or D reflecting defaults.

Execute the cell.

__3. CELL 7. Print the classification reports for the model just trained. Insert this code:
`print(classification_report(y_test, y_pred))`

```
In [ ]: # For an explanation of the classification report and the following confusion matrix
        # Follow this link: https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

        #### Insert code below.
```

You'll get a table with some parameters. If you want to understand what the parameters mean you may go to this webpage (https://www.geeksforgeeks.org/confusion-matrix-machine-learning/)

Execute the cell.

__4. CELL 8. Get your confusion matrix.

```
In [9]: # CELL 8

        # A confusion matrix is a visual to help you understand how good predictions (y_pred) are for X_train compared to the test.
        cml = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
        sns.heatmap(cml,annot = True)
```

Execute this cell.

You can see that the model behaves quite well having a lot of true positives predicted and very few false positives and negatives.

__5. CELL 9. Now, for the sake of learning, check the array of importance assigned by the algorithm.
Insert this code after the comment:
`print(fi,len(fi))`

```
In [ ]: # Check how features have an "importance" assigned. It is a measure of the influence of each in the classification process

        fi = rf.feature_importances_

        #### Inset code below. Print the array of importances.
```

As you can see the array has a size of 49, all the columns of the dataframe less the status one which is our target.

Execute this cell.

__6. CELL 10. Subset the dataframe to be able to check the behaviour of the features according to importances.
Insert code below the comment:

```
feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols, "importance": rf.feature_importances_})
```

```
In [ ]: #### Insert code below. Subset the dataframe to check the behaviour of importances and features
```

Execute the cell.

__7. CELL 11. Create a dictionary to map the values for features and importance and sort values in descending order. Plot the results inserting this code under the comment:
`importance[:20].plot(x="feature", y="importance", kind="bar")`

```
In [ ]: # Study the subset created in previous cell. Using a visual aid is very useful.
        # Plot features in x axis and importance in y axis
        # Limit the visual to 20 first features in importance

        importance = pd.DataFrame(
            {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
        )
        importance.sort_values(
            by="importance",
            axis=0,
            ascending=False,
            inplace=True,
            kind="quicksort",
            na_position="last",
        )

        #### Insert code below to do the plotting
```

Execute the cell and examine the results. Does it make sense to you?

__8. CELL 12. Group and print the status of loans by sex.

```
In [ ]: # Get an idea of the distribution of defaults by sex.
        # Remember: sex 0 = women, sex 1 = men. Status 0 = all ok, Status 1 = default

        df.groupby(["sex", "status"])["status"].size()
```

Execute this cell and examine the result.

__9. CELL 13. Plot the information of status by sex.
Insert this code below the comment to set labels for the graph:

```
plt.xlabel({"1": "male", "0": "female"})
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("percentage - default or not")
plt.show()
```

```
In [ ]: # Visualize same result as in former cell using a stacked bar graph

        df.groupby(["sex", "status"])["status"].size().groupby(level=0).apply(
            lambda x: 100 * x / x.sum()
        ).unstack().plot(kind="bar", stacked=True)

        #### Insert code below. Customize the graph to make it more understandable
```

Execute the cell.
As you can see sex does not have an influence on defaults.

__10. CELL 14. Plot the relationship between age and status.
Insert code below the comment:

```
df.groupby(["age",
"status"])["status"].size().groupby(level=0).apply(
```

```
    lambda x: 100 * x / x.sum()).unstack().plot(kind="hist",
stacked=True)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.show()
```

```
In [ ]: # Visualize age related to status

        #### Insert code below. Plot the relationship between age and status

        |
```

Execute the cell and interpret the result.

__11. CELL 15. Now plot the number of years a credit card has been holded vs status. Insert this code below the comment:

```
df.groupby(["years_card_issued",
"status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()).unstack().plot(kind="bar",
stacked=True)
plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())
plt.ylabel("percentage - default or not")
plt.xlabel("years_visa_card_owned")
plt.legend(loc=1)
plt.show()
```

```
In [ ]: # Plot years of having a card vs status

        #### Insert code below

        |
```

Execute the cell and watch the result.

__12. CELL 16. Use the function binning applied to a range of ages between 20 and 50 years. Insert this code below the comment:

```
cut_points = [24, 34, 44, 50]
labels = ["20", "25", "35", "45", "50"]
df["age_bin"] = binning(df["age"], cut_points, labels)
```

```
In [ ]: # Study the influence of age vs status. To do that you need to segment by ages
        # Define a function to create age bins

        # Binning:
        def binning(col, cut_points, labels=None):
            # Define min and max values:
            minval = col.min()
            maxval = col.max()

            # create list by adding min and max to cut_points
            break_points = [minval] + cut_points + [maxval]

            # if no labels provided, use default labels 0 ... (n-1)
            if not labels:
                labels = range(len(cut_points) + 1)

            # Binning using cut function of pandas
            colBin = pd.cut(col, bins=break_points, labels=labels, include_lowest=True)
            return colBin


        # Binning age:
        #### Insert code below. Apply function binning to a sample of ages between 20 and 50 years
        |
```

Execute this cell.

__13. CELL 17. Create a grouped bar chart with the age groups created before. Group `age_bin` with `status` and then represent it versus `status`.
Insert this code below the comment:

```
df.groupby(["age_bin",
"status"])["status"].size().groupby(level=0).apply(
    lambda x: 100 * x / x.sum()).unstack().plot(kind="bar",
stacked=True)

plt.gca().yaxis.set_major_formatter(mtick.PercentFormatter())

plt.ylabel("percentage - default or not")
```

```
In [ ]: # Plot the recently created column age_bin vs status

        #### Insert code below. Create a plot grouping age_bin with status vs status
```

Execute the cell and examine the resulting graph.

__14. CELL 18. Have a look on the dataframe selecting all rows which show a status of 1 (any kind of default registered).
Insert this code below the comment:

```
df[df.status == 1].head()
```

```
In [ ]: # Have a look to the dataframe selecting all rows marked with a 1 status (any kind of default)

        #### Insert code below. Select all rows with status 1
```

Execute this cell and see the result.

__15. CELL 19. Create a graph to understand how payments and status relate to each other.
Insert this code below the comment:

```
a = loan.groupby(by="status", axis=0, level=None, as_index=True,
sort=True, group_keys=True)
a.payments.mean().plot(kind="bar")
plt.ylabel("payments")
```

```
In [ ]: # Graph status vs payments mean to have an idea of the behaviour of clients defaulting and not defaulting.

        #### Insert code below. Group averaged payments with status and plot it in a bar graph
```

Execute the cell and see the result.

__16. CELL 20. Plot a heatmap with the 10 main features we already have in the `importance` list. Add status to the plot.
Insert this code below the comment:

```
    bottom, top = hm.get_ylim()
    hm.set_ylim(bottom + 0.5, top - 0.5)
```

In [ ]:
```
# Plot a heatmap to understand the correlation of the main features. Limit it to a 10x10 matrix
import seaborn as sns

cols = list(importance.feature[:10]) # To add or remove features to the matrix change this number
cols.insert(0, "status")
corrcoef_map = np.corrcoef(df[cols].values.T)
fig, ax = plt.subplots(figsize=(12, 12))  # Sample figsize in inches
hm = sns.heatmap(
    corrcoef_map,
    cbar=True,
    annot=True,
    square=True,
    fmt=".2f",
    annot_kws={"size": 15},
    yticklabels=cols,
    xticklabels=cols,
    ax=ax,
)

#Optional. Fix for this version of the package matplotlib.

#### Insert code below to fix the view. This is a problem with the present version of matplotlib
```

Execute the cell and carefully study the correlations among features. The higher the correlation number the more correlated are the features.

Section 3. Modelling

After testing and examining features for the final models, we are in shape to define and train the final machine learning models to fit our dataframes.

__ 1. Random Forest

__1.1. CELL 21. Start with a Random Forest model. Fit the model and make a prediction test for it.
Insert this code below the comment in the cell:

```
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```
In [ ]: # Train and test a Random Forest model

rf = ensemble.RandomForestClassifier(
    n_estimators=800,
    criterion="gini",
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features="auto",
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    bootstrap=True,
    oob_score=False,
    n_jobs=1,
    random_state=None,
    verbose=0,
    warm_start=False,
    class_weight=None,
)

#### Insert code below. Run the fitting and prediction of the model created above
```

Run the cell.

__1.2. CELL 22. Print the classification report to understand how this model behaves.
Insert this code below the comment:

```
print(classification_report(y_test, y_pred))
```

```
In [ ]: # Get the classification report

#### Insert code below. Print the classification report for this fit of the model
```

Execute this cell and examine the results.

__1.3. CELL 23. Plot the confusion matrix for this model.
Insert this code below the comment:

```
cm2 = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])
sns.heatmap(cm2,annot = True)
```

```
In [ ]: # Get the confusion matrix for the model

#### Insert code below. Print the confusion matrix for this fit of the model
```

Run this cell and look at the results. What can you say about them?

__1.4. CELL 24. Arrange a new dataframe to help in understanding the behavior of features in this model.

```
In [ ]: # Build feature and importance dataframe to further check the behaviour of the model

feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols[:], "importance": rf.feature_importances_[:]}
)

importance.sort_values(
    by="importance",
    axis=0,
    ascending=False,
    inplace=True,
    kind="quicksort",
    na_position="last"
)


importance[:18].plot(x="feature", y="importance", kind="bar")
plt.ylabel("importance")
```

Run the cell. Check the influence of features in the model.

__1.5. CELL 25. Export the model to the z/OS repository to make it available to be deployed in test and production if necessary.

```
In [34]: # CELL 25

# Now you will export this model to the repository on z/OS so that it will be available for use in test and production
from repository_v3.mlrepository import MetaNames
from repository_v3.mlrepository import MetaProps
from repository_v3.mlrepositoryclient import MLRepositoryClient
from repository_v3.mlrepositoryartifact import MLRepositoryArtifact
import pprint

metaservicePath = "https://10.3.72.69:443"
client = MLRepositoryClient(metaservicePath)
client.authorize_with_token(pc.authToken)

props1 = MetaProps(
        {MetaNames.AUTHOR_NAME:"author",
         MetaNames.AUTHOR_EMAIL:"author@example.com",
         MetaNames.MODEL_META_PROJECT_ID: pc.projectName,
         MetaNames.MODEL_META_ORIGIN_TYPE: "notebook",
         MetaNames.SCOPE: "Project",
         MetaNames.MODEL_META_ORIGIN_ID: pc.nbName})

input_artifact = MLRepositoryArtifact(rf,
      name="loan_random_forest", meta_props=props1, # Change the name of the model if needed
      training_data=X_train, training_target=y_train)

client.models.save(artifact=input_artifact)
print("model saved successfully")
```

Review the values and **check that the model name corresponds with what you are just about to export.**
Execute this cell.

__ 2. Decision tree

__2.1. CELL 26. Build and run a decision tree model.
Insert the following code below the comment:
`clf = tree.DecisionTreeClassifier(`

```
    criterion="gini",
    splitter="best",
    max_depth=5,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features=None,
    random_state=None,
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    class_weight=None,
    presort=False
)

model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
In [ ]: # Create and train a decision tree model

        #### Insert code below.
```

Run this cell.

__2.2. CELL 27. Again, get the classification report for this model.
Insert the following code below the comment:
`print(classification_report(y_test, y_pred))`

```
In [ ]: # Get the classification report

        #### Insert code below. Print the classification report for this fit of the model
```

Run the cell. You might want to compare this classification report with the one for the Random Forest model.

__2.3. CELL 28. Plot your confusion matrix for this decision tree.
Insert this code below the comment:

```
cm3 = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])
sns.heatmap(cm3,annot = True)
```

```
In [ ]: # Plot the confusion matrix for this decision tree

        #### Insert code below.
```

__2.4. CELL 29. And again, do the classification of features by importance.

In [43]:
```
# CELL 29

# Classify features by order of importance for the decision tree

feature_cols = X_test.columns
importance = pd.DataFrame(
    {"feature": feature_cols[:], "importance": clf.feature_importances_[:]}
)
importance.sort_values(
    by="importance",
    axis=0,
    ascending=False,
    inplace=True,
    kind="quicksort",
    na_position="last",
)
importance[:18].plot(x="feature", y="importance", kind="bar")
```

Run the cell and examine the result.

__2.5. CELL 30. Export the decision tree model to the repository.

In [ ]:
```
# Copy the code to export the previous model and update this cell
# Delete the imports which have been done already in previous cell
# Update the name of the model before executing the cell

#### Insert code below. Export decision tree model to the repository.
```

Copy the piece of code used to export the previous model (CELL 25).
Delete all `import` sentences in it.
Change the name of the model.
Execute the cell.

__ 3. Gradient Boosting Classifier

__3.1. CELL 31. Create, train and get the classification report for the Gradient Boost Classifier model.

In [44]:
```
# CELL 31

from sklearn.ensemble import GradientBoostingClassifier

# Create the model and set parameters

gbc = GradientBoostingClassifier(
    loss="deviance",
    learning_rate=0.1,
    n_estimators=200,
    subsample=1.0,
    criterion="friedman_mse",
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_depth=3,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    init=None,
    random_state=None,
    max_features=None,
)

# Train the model and fit data
model = gbc.fit(X_train, y_train)
y_pred = gbc.predict(X_test)

# Get the classification report
print(classification_report(y_test, y_pred))
```

Run the cell. Have a look to the classification report. You might want to compare results with the two former models.

__3.2. CELL 32. Plot the confusion matrix as in former models.
Insert code below the comment:

```
cm4 = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])
sns.heatmap(cm4,annot = True)
```

In [ ]:
```
# Get the confusion matrix for this Gradient Boost Classifier

#### Insert code below. Plot the confusion matrix for this GBC model
```

Run the cell and watch the results.

__3.3. CELL 33. Export the gradient boost model to the repository.

```
In [ ]: # Copy the code to export the previous model and update this cell
        # Delete the imports which have been done already in previous cell
        # Update the name of the model before executing the cell

        #### Insert code below. Export GCB model to the repository.
```

Copy the piece of code used to export the previous model (find it in CELL 30).
Delete all `import` sentences in it.
Change the name of the model.
Execute the cell.

__ 4. Support Vector Machine (SVM)

__4.1. CELL 34. Do the feature scaling by running this code:

```
In [41]: # CELL 34

         # Standard processing for feature scaling

         sc = StandardScaler()
         X.drop(['age'], axis=1, inplace=True)
         X = sc.fit_transform(X)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Execute the cell.

__4.2. CELL 35. Create and train the SVM model. Get the classification report.

```
In [ ]: # Create, train and fit the SVM model. Print the classification model

        svc = svm.SVC(
            C=5,
            kernel="rbf",
            degree=3,
            gamma="auto",
            coef0=0.0,
            shrinking=True,
            probability=False,
            tol=0.001,
            cache_size=200,
            class_weight=None,
            verbose=False,
            max_iter=-1,
            decision_function_shape="ovr",
            random_state=None,
        )
        model = svc.fit(X_train, y_train)

        y_pred = svc.predict(X_test)

        print(classification_report(y_test, y_pred))
```

Run this cell.

__4.3. CELL 36. Get the confusion matrix from the new SVM model.
Insert this code below the comment.

```
cm5 = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])
sns.heatmap(cm5,annot = True)
```

```
In [ ]: # Get the confusion matrix as with every other model. Feel free to fix the visualization

        #### Insert code below. Get the confusion matrix for your SVM model
```

Run the cell. Compare results with other models.

__4.4. CELL 37. Export the support vector model to the repository.

```
In [ ]: # Copy the code to export the previous model and update this cell
        # Delete the imports which have been done already in previous cell
        # Update the name of the model before executing the cell

        #### Insert code below. Export SVM model to the repository.
```

Copy the piece of code used to export the previous model (you may find it in CELL 33)
Delete all `import` sentences in it.
Change the name of the model.
Execute the cell.

__ 5. Logistic Regression

And the last model for today:

__5.1. CELL 38. Create and fit a Logistic Regression model.
Insert the following code under the comment:

```
lr = LogisticRegression(penalty="l1", C=1).fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

```
In [ ]: # Create and fit a Logistic Regression model

        #### Insert code below. Invoke the model and fit it, then make your predictions
```

Run the cell.

Include this code for the classification report under the last line of code:

```
print(classification_report(y_test, y_pred))
```

Run the cell again.

__5.2. CELL 39. Get the confusion matrix for this model.

```
In [47]: # CELL 39

         # Get the confusion matrix for the Logistic Regression model

         cm6 = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
         sns.heatmap(cm6,annot = True)
```

Fix the visualization for this matrix and run the cell.

__5.3. CELL 40. Export the Logistic Regression model to the repository.

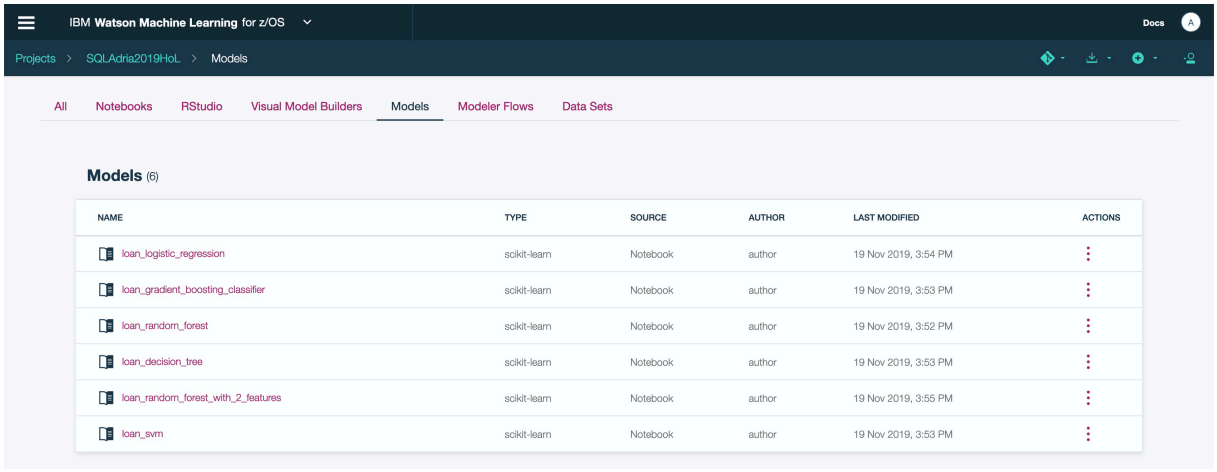
```
In [ ]: # Copy the code to export the previous model and update this cell
        # Delete the imports which have been done already in previous cell
        # Update the name of the model before executing the cell

        #### Insert code below. Export Logistic Regression model to the repository.
```

Copy the piece of code used to export the previous model.
Delete all `import` sentences in it.
Change the name of the model.
Execute the cell.
You completed lab 3.

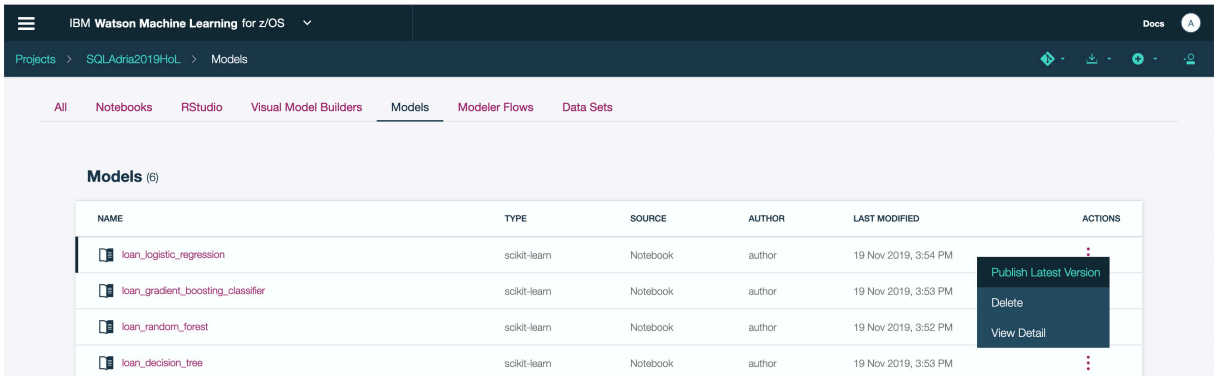
Lab 4: Deploy and test your models

Assuming you have successfully completed labs 1, 2 & 3, you have saved models generated within the Jupyter Notebook user interface. We will now manage and use these models to be called for scoring.

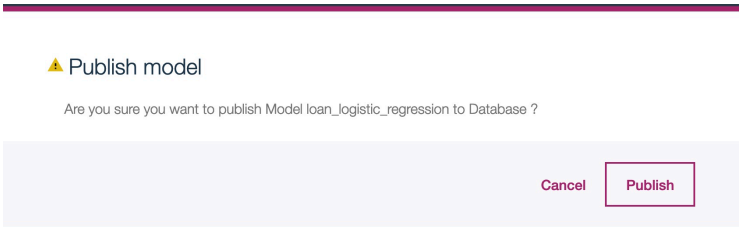
__ 1. Navigate back to the project view by clicking on your project name near the top- left of the window, and then click **Assets.** From there, select the **Models** tab and verify that the models trained from the Jupyter Notebook are saved.



__ 2. Click the **Actions** of the loan_logistic_regression model and select the **Publish Latest Version** button.



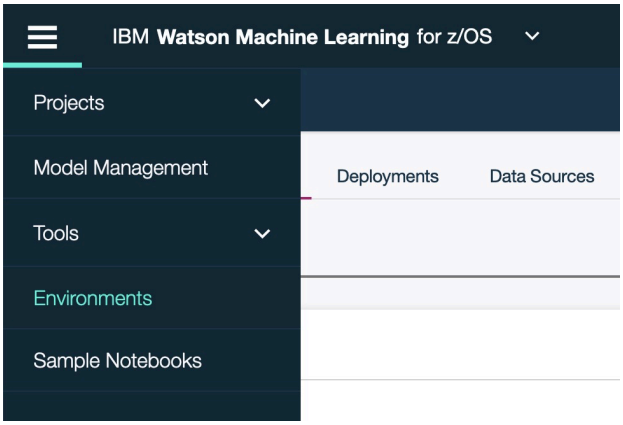
__ 3. Click the **Publish** button to publish the model so that it becomes available to everyone via the Model Management UI which is covered next.



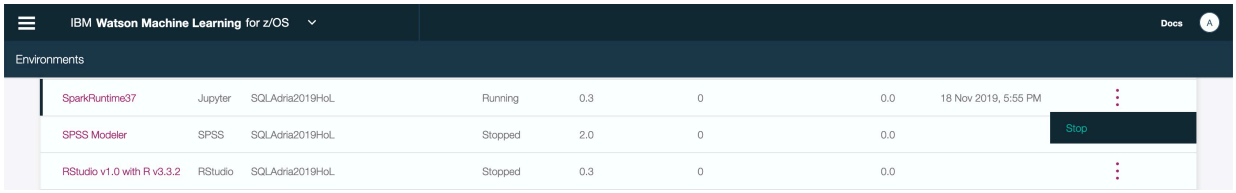
__ 4. Repeat the publish operations for all your saved models.

__ 5. Please Shutdown your Jupyter Notebook runtime to free up CPU and memory resources. To do so :

- Click on the top-left “hamburger” menu (3 horizontal bars) and select **Environments**.



- Click the **action menu (3 dots)** for SparkRuntime37 associated with your project and select **Stop**.

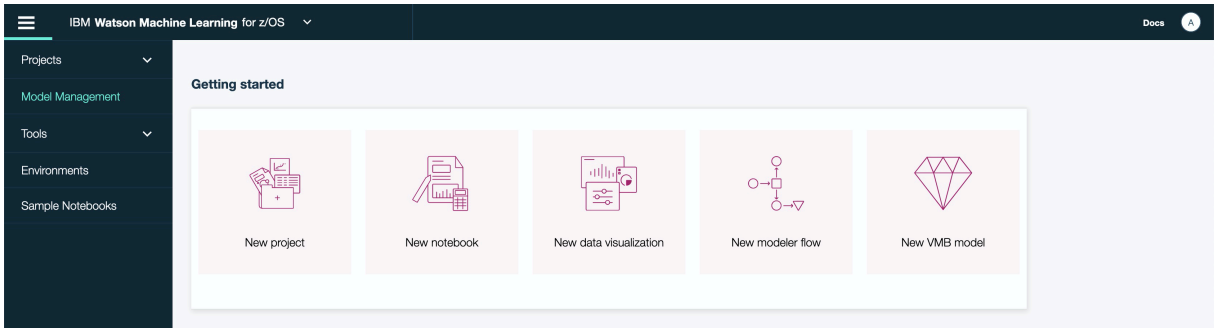


You are now ready to manage and deploy your models.
You will learn, as a z/OS System Administrator, how you can use WMLz to:

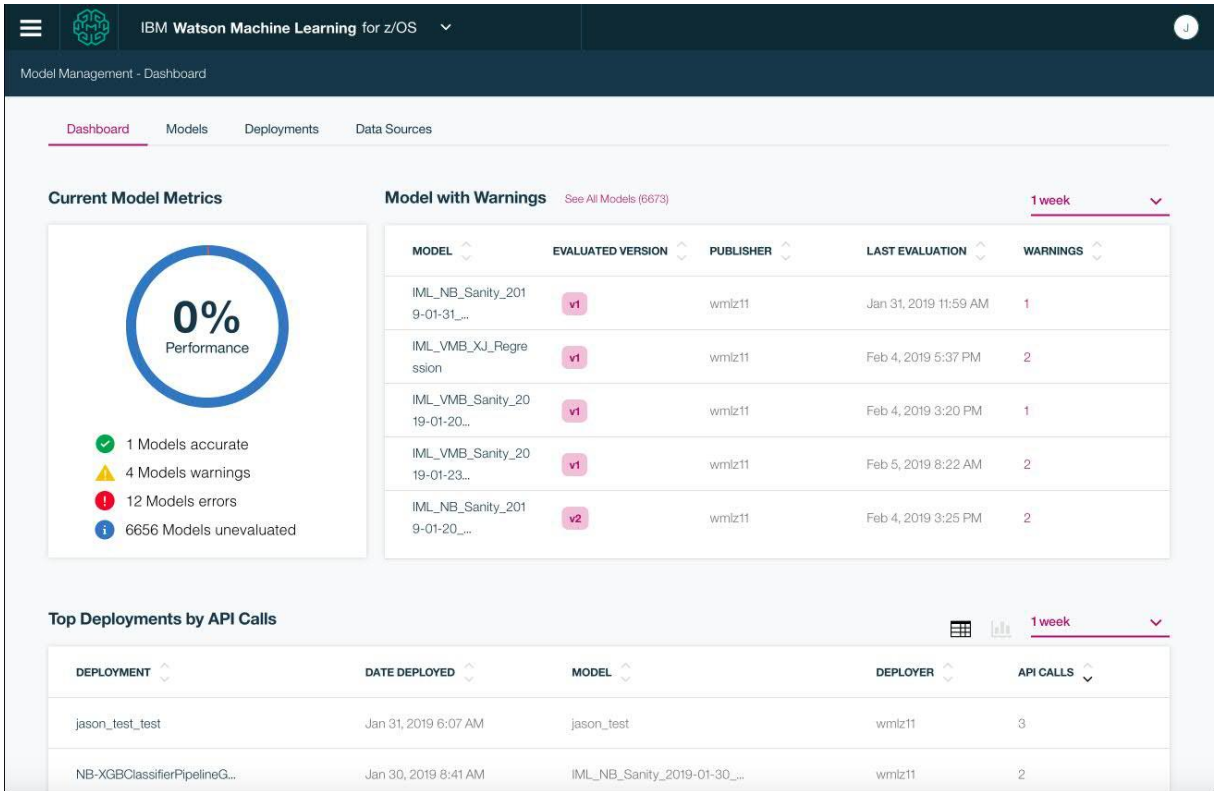
- View the details of the model that your Data Scientist has created.
- Manage the deployment of the model on your production environment.
- Schedule the model evaluation and test the model using online RESTful API.
- Work with your Application Developer to implement the model for scoring and prediction.

Section 1. Dashboard

__ 1. Open the WMLz Dashboard view UI by clicking the **hamburger menu** in the top left and selecting **Model Management** as below.

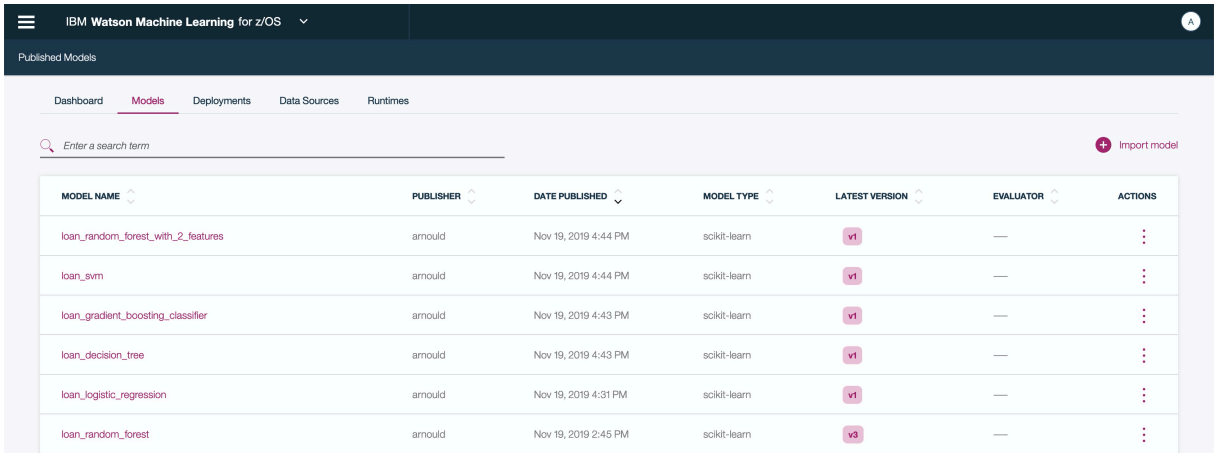


The **Dashboard** displays the statistical information of IBM Machine Learning platform, including the model performance and overall status.

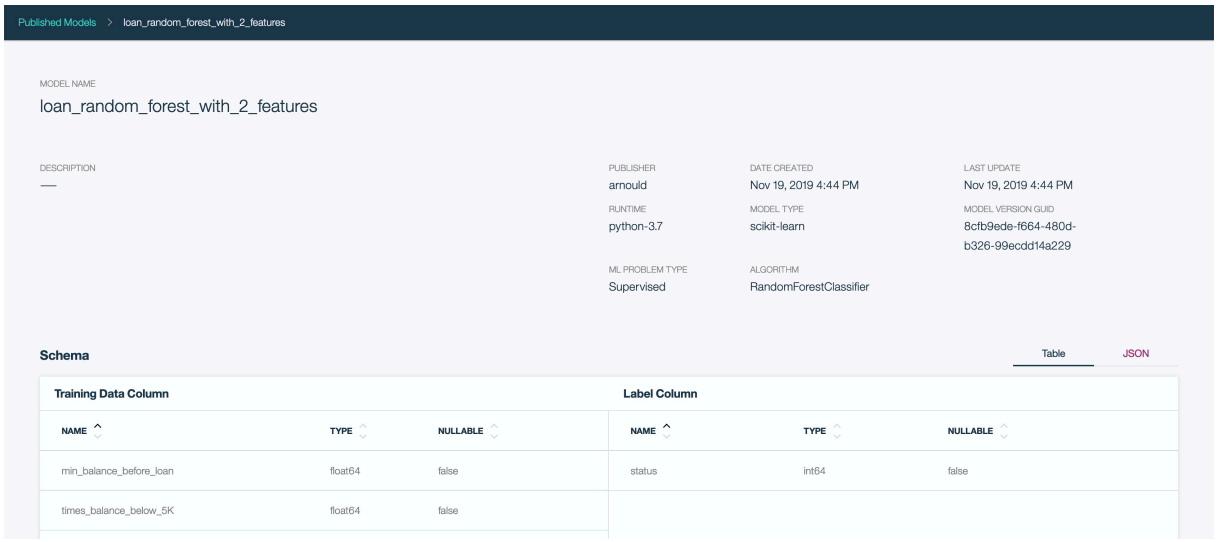


Section 2. Model Management

__ 1. Click the **Models** tab. The **Models** page is where you list and manage the Machine Learning models that have been created and exist in your environment. You can view the model details and model versions, retrain, delete and create deployment for a certain model. The page also provides a filter to help you locate a specific model.



__ 2. Click on the model ***loan_random_forest_with_2_features*** to display its detail information. The schema on the left lists the inputs that the model requires. The schema on the right displays the output from the model, which is what the model predicts.

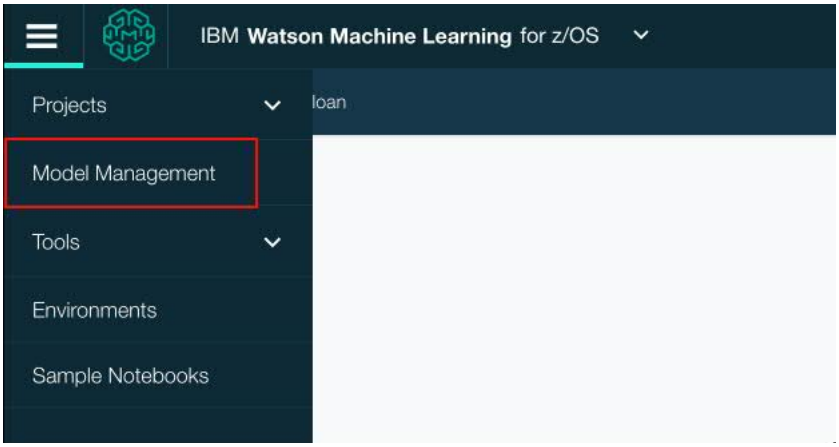


__ 3. Once you are finished viewing the details, click **Published models** in the top left of the window to return to the **Published Models** page.

Section 3. Deployment Model to Scoring Server

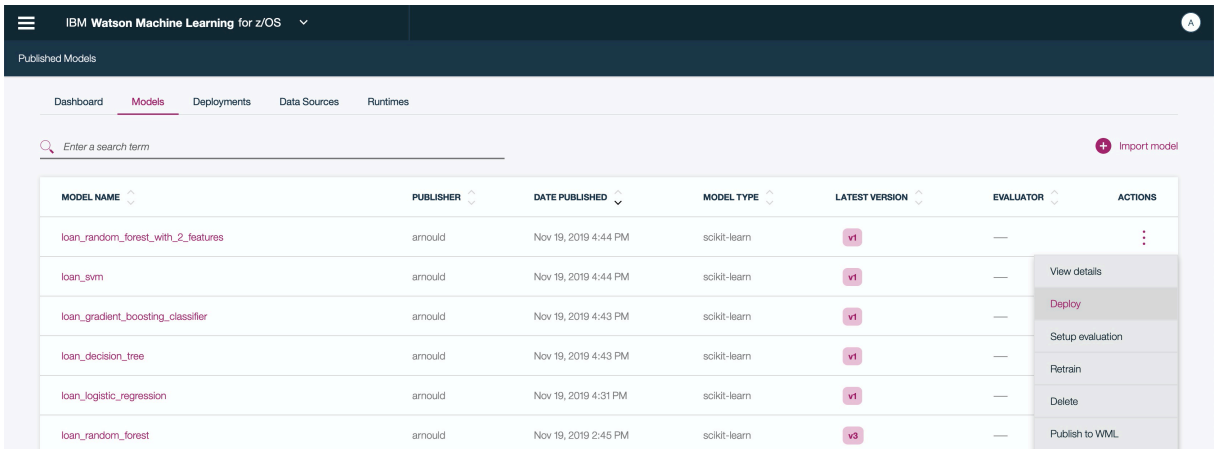
As a System Administrator, you will need to deploy the model to your z/OS environment and test it online. It may sound like a daunting task, but it can be accomplished easily in the WMLz framework:

__ 1. Navigate to the **Model Management** Page
__ 2. Click on the **3 short horizontal bars** in the top left of the web page and choose **Model Management**.

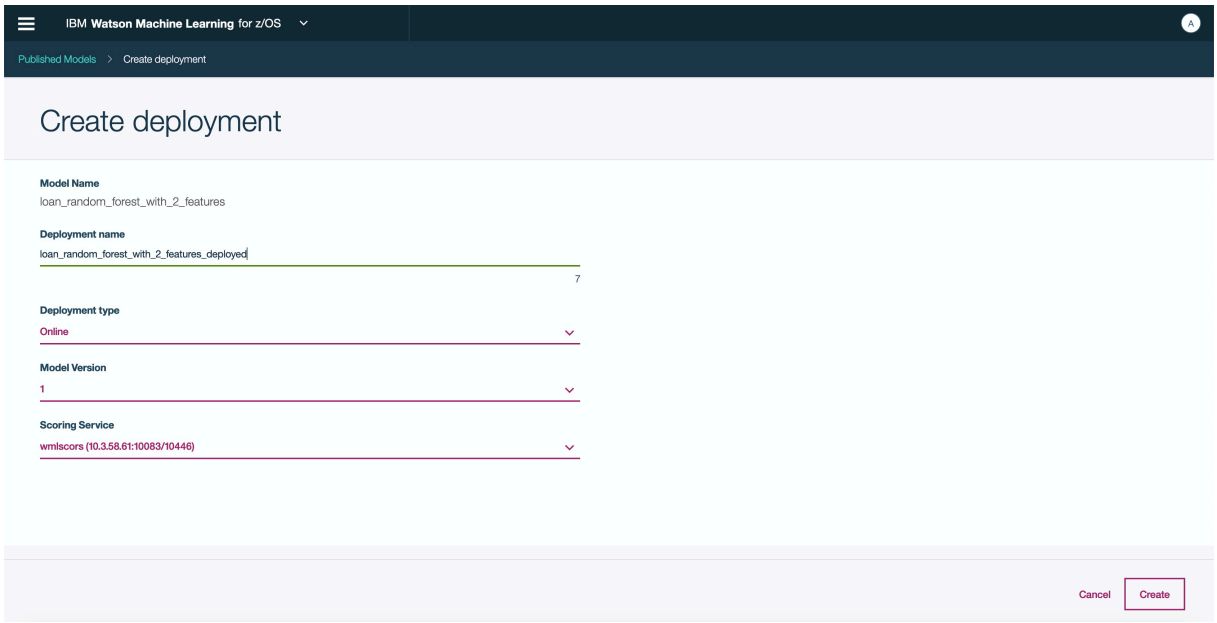


__ 3. Deploy the model :

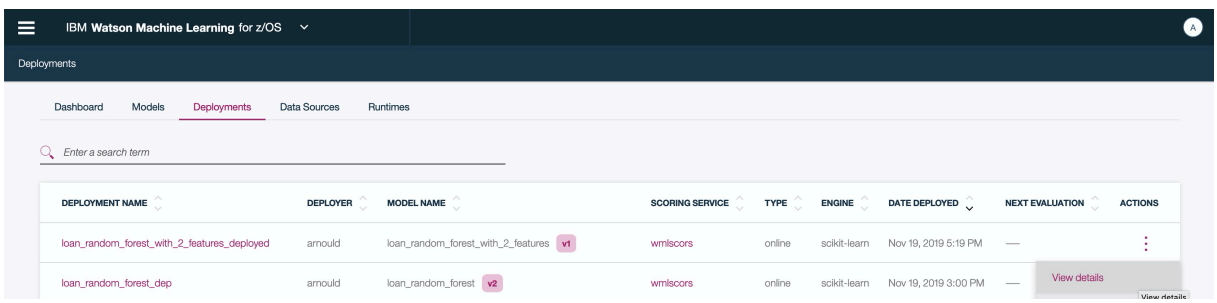
- On the row where the model **loan_random_forest_with_2_features** is located, click the **actions** menu and select the **Deploy** button under **ACTIONS.** You also have the option to deploy a model created by others.



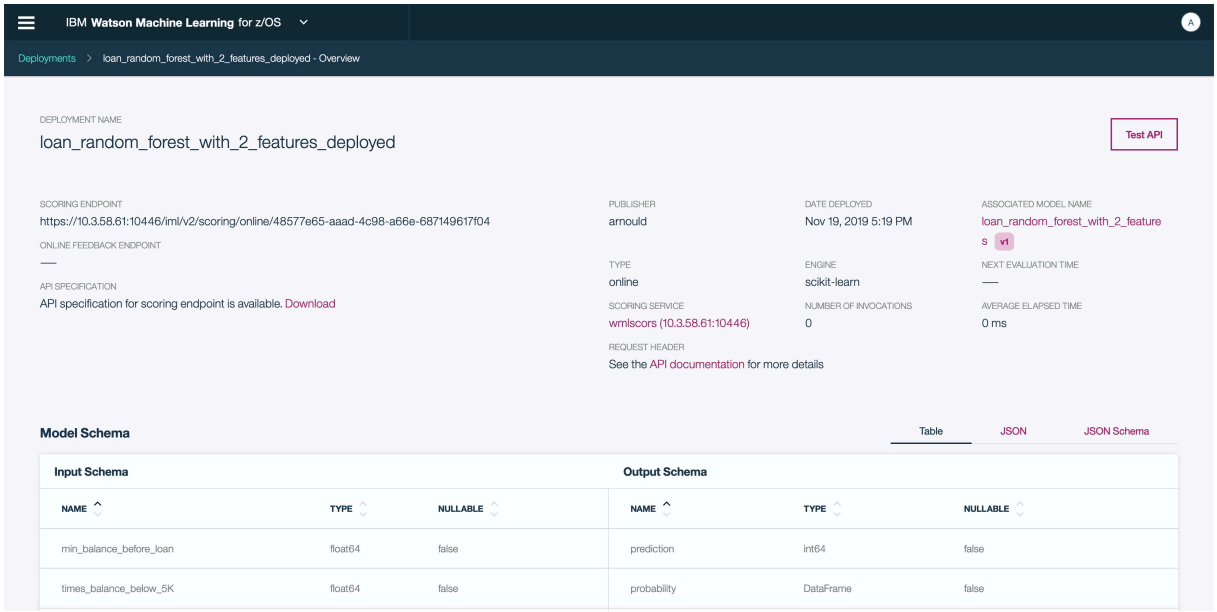
__ 4. On the **Create deployment** page, enter the name of the deployment and select **Online** for the **Type**. Choose the scoring service whose name is “wmlscors”, and then click the **Create** button to save it.



Wait until the deployment completes successfully. Then locate your deployment under the **Deployments** page. The **Deployments** page is where you view the list of models that have been deployed to your z/OS environment for scoring and prediction. You can test the model here, manage the deployment, and schedule evaluations to check the performance of the model on a regular basis.

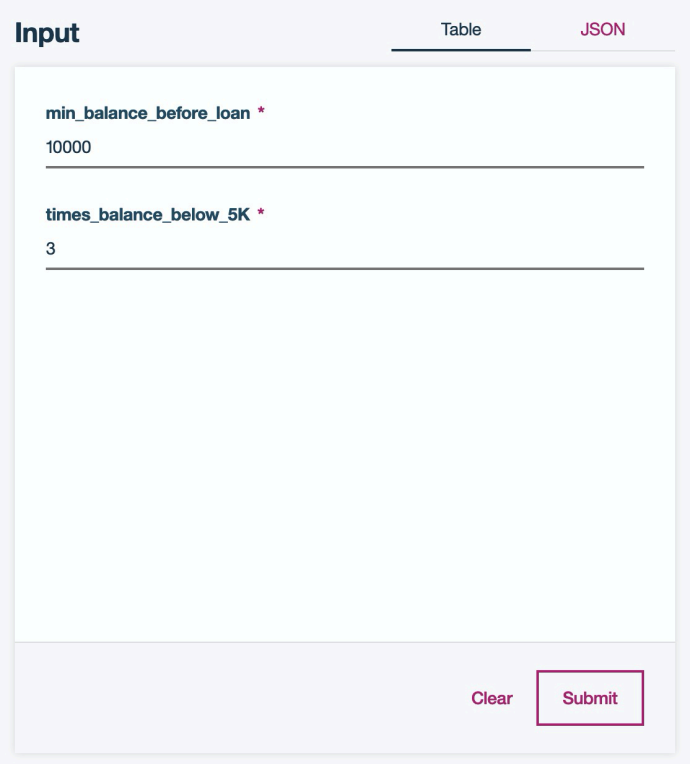


__ 5. Click the **View details** button under **ACTIONS** to check the details of the deployment.
Note the API details for the RESTful API provided by the scoring service of WMLz which allows your application to call the model for online scoring. In the next step, you will test this scoring API and the model online in the Web UI. If the test is successful, you can then have your application developer update the application to call this REST API. This allows you to implement online scoring on the same system as the transactions running under z/OS.



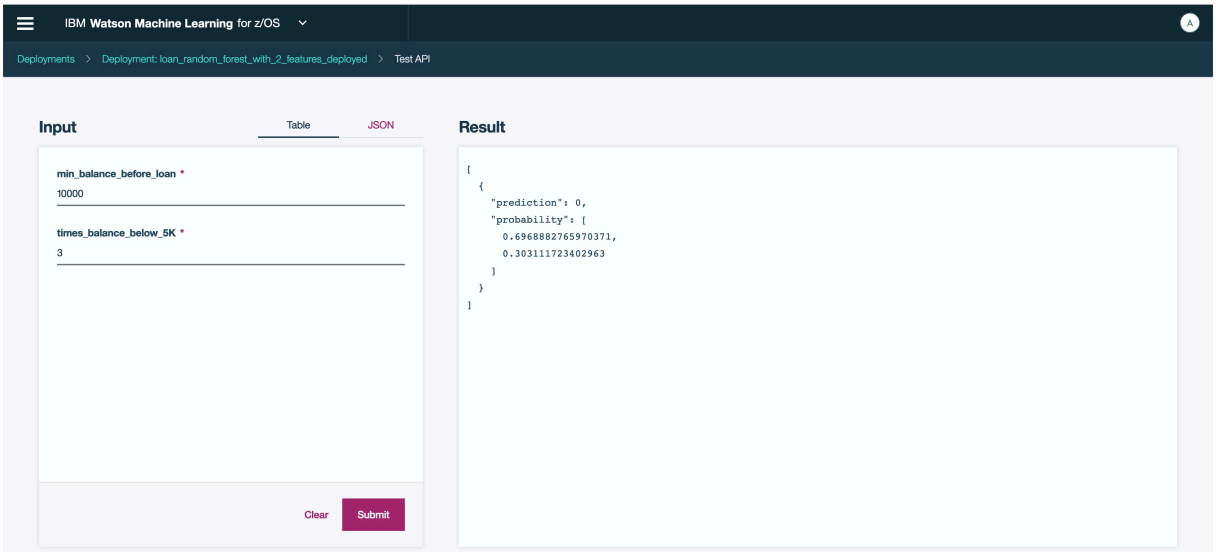
__ 6. Click the **Test API** button on the top-right corner.

__ 7. On the **Test API** page, enter a record for values according to the schema of the input record of the model as below. Input values are case sensitive!



Sample Imput Record:
min_balance_before_loan: 10000
times_balance_below_5K: 3

__ 8. Click the **Submit** button to test the prediction. The result will be displayed in the **Result** area



You completed lab 4.

Congratulations! You have successfully completed this Hands-On-Lab!
Don’t forget to submit your session feedback!

Your feedback is very important to us, we use it to continually improve the lab.

We Value Your Feedback!
Thank You.