AI on IBM Z

Credit risk assessment solution template

This solution template provides an example on how to deploy AI using an IBM Z environment, while making use of open source frameworks, Watson Machine Learning for z/OS (MLz), and more.

Within this solution template, there are various phases of the AI lifecycle included. Work through each of the following steps to deploy your own credit risk assessment solution on IBM Z.

Table of contents

Step 1

AI model training

We will build a credit risk assessment AI model by training with the provided Rapid AI on IBM Z Development Jupyter notebook. Simply point the Jupyter notebook to your dataset and run it to generate your AI model. This trained AI model can then be deployed with MLz.

All sample code for this section is within

```
ai-st-credit-risk-assessment/zST-model-training-jupyter
```

Prerequisites

1. Must have Python (3.9 or 3.10) installed

Dataset guidance

Sample open source credit risk assessment dataset can be found on Kaggle -

https://www.kaggle.com/datasets/laotse/credit-risk-dataset

Required features

- person_age
- person_income
- person_home_ownership
- person_emp_length
- loan_intent
- loan_grade
- loan_amnt
- loan_int_rate
- loan_status
- loan_percent_income
- cb_person_default_on_file
- cb_person_cred_hist_length

Access rapid AI on IBM Z development environment

1. Create and activate Python virtual environment

```
python -m venv env
source env/bin/activate
```

2. Install required Python packages

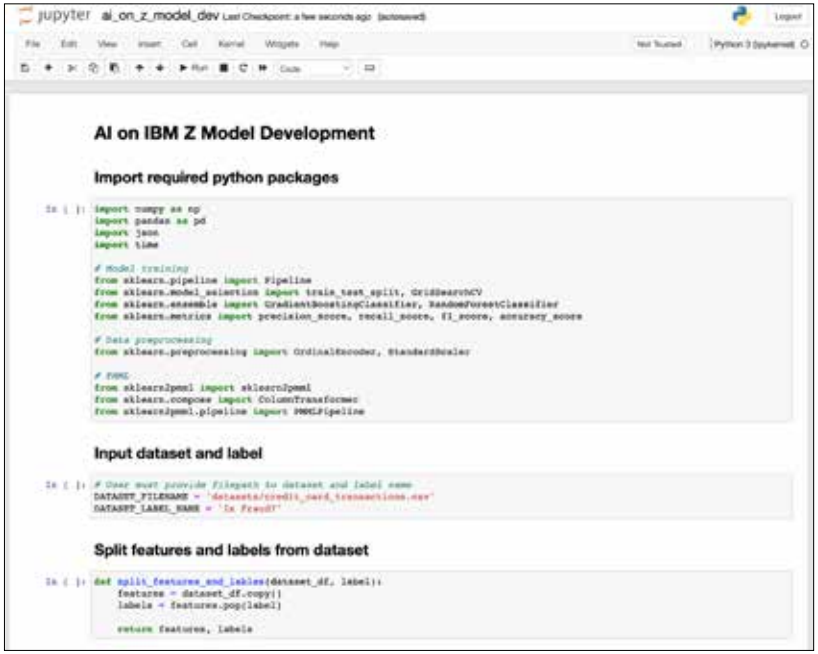
```
pip install -r requirements.txt
```

3. Run Jupyter

```
jupyter notebook
```

4. View Jupyter interface
   Go to localhost:8888 in a web browser

5. Click on ai_on_z_model_dev.ipynb in web browser

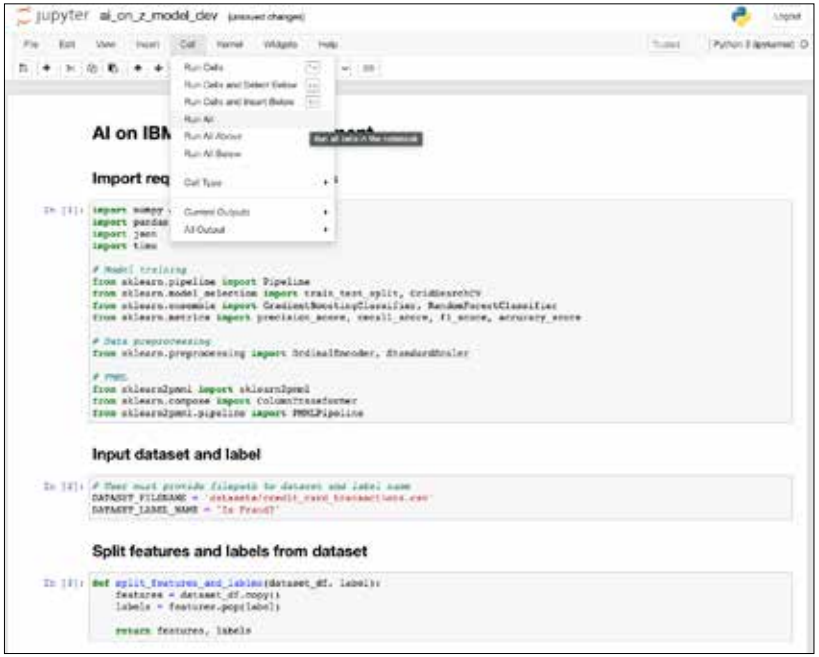


Provide data

1. Add your input dataset (csv) into datasets/ directory
2. Add input data to Jupyter notebook
   - Set DATASET_FILENAME to the path to your dataset
   - Set DATASET_LABEL_NAME to the name of the column you're predicting from the dataset

Model training

1. Step through and run Jupyter notebook from web browser



Access trained AI model

1. Once training is complete, you can find your AI models within the models/directory (choose one for the following AI model deployment step)

AI model training complete

Prerequisites

1. Must have MLz installed

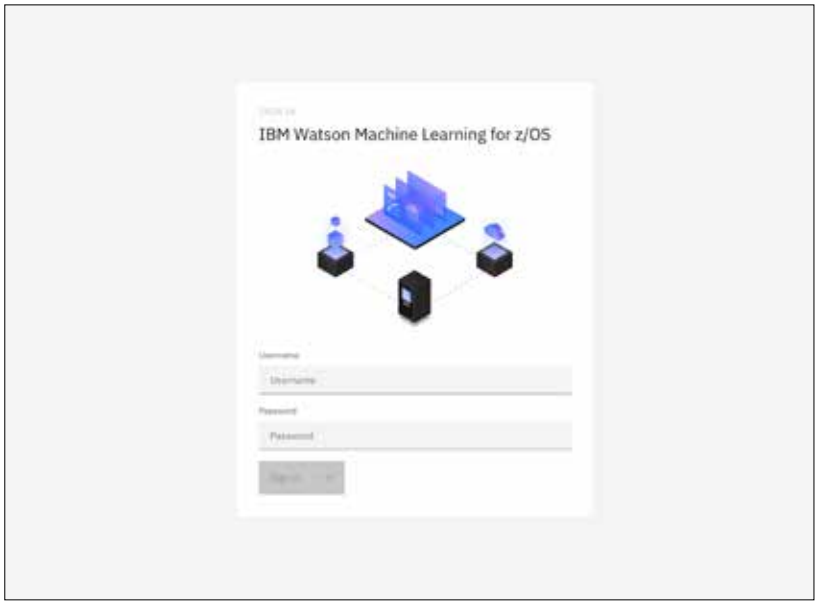
Step 2

AI model deployment

We will deploy our fraud detection AI model using MLz. We can utilize the model import functionality on the MLz UI. This deployed AI model can then be integrated into applications within the IBM Z environment.

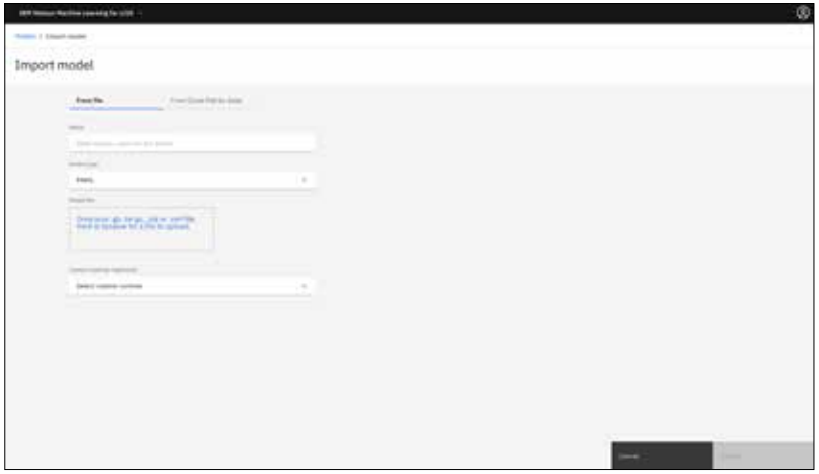
Go to MLz UI

1. Sign in with username/password



Import AI model

1. Go to models tab
2. Click import model
3. Enter model name
4. Choose model type
   Choose PMML if using your previously trained model
5. Drag and drop model file
   Use your previously trained model
6. Click import

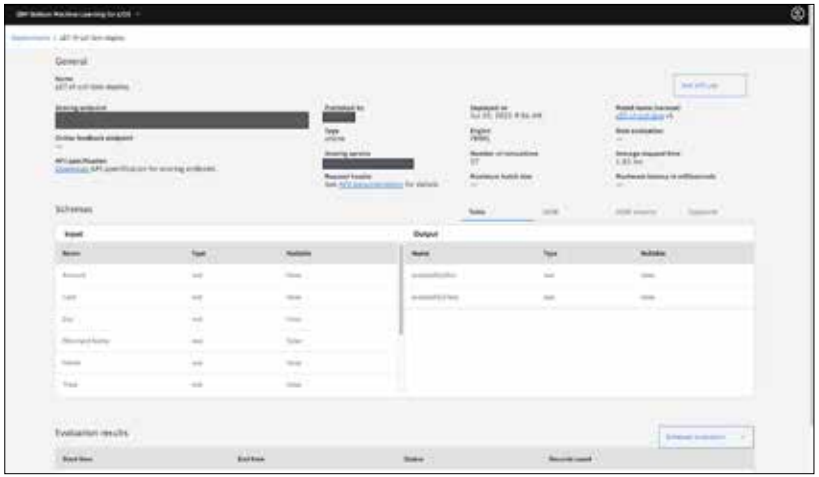


Deploy AI model

1. Go to models tab
2. Click action button for your model (on right side)
3. Click deploy
4. Enter deployment name
5. Choose deployment type
6. Choose model version
7. Choose scoring service
   Note: you should choose the correct scoring service based on your application (e.g. CICS or REST)
8. Click create

View deployed AI model

1. Go to deployments tab
2. Click on action button for your deployed model (on right side)
3. Click view details



AI model deployment complete

Step 3

AI model integration

Choose One:

Web application
CICS-COBOL application

We can use our deployed MLz credit risk assessment AI model and integrate it into different types of applications. The AI model can be analyzed and/or provide inferencing APIs using the sample AI on IBM Z Credit Risk Assessment Dashboard.

Web application

We can use our deployed MLz credit risk assessment AI model and integrate it into different types of applications. Guidance on integrating the AI model into a sample credit risk assessment application is below.

All sample code for this section is within

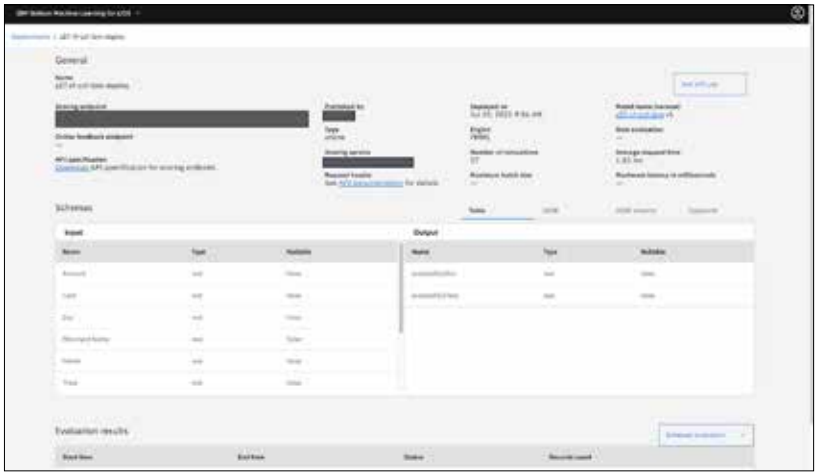
```
ai-st-credit-risk-assessment/zST-model-integration-cra
```

Prerequisites

1. Must have node.js v16 or newer installed
2. Must have Docker installed
3. Must have Git installed

Get model details for inferencing

1. Go to MLz UI
2. Go to deployments tab
3. Click on action button for your deployed model (on right side)
4. Click view details
5. Copy scoring endpoint



Configure sample application

Set the enrionment variables within env.list file

- WML_USER (username for MLz)
- WML_PASS (password for MLz user)
- SCORING_URL (scoring endpoint for deployed AI model)

Build sample application

1. Run command in terminal

```
docker build -t credit-risk-assessment .
```

Deploy sample application

1. Run command in terminal (e.g. port 9000)

```
docker run -p 9000:80 --env-file env.list
--name credit-risk-assessment-app cred-
it-risk-assessment
```

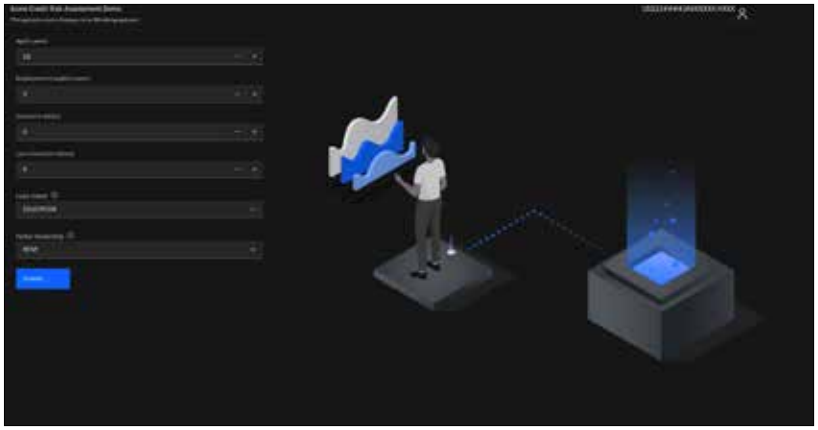
Access sample application

View the following URL in a web browser

- Credit risk assessment
  - http://{ip address}:{port}//static/cra/
- Dashboard
  - http://{ip address}:{port}//static/dashboard/

Note:

- IP address: IP of server you deployed application in
- Port: port you used with docker run



AI model integration complete

CICS-COBOL application

In this type of CICS-COBOL program, we will be inferencing the Credit Risk Assessment model deployed into the MLz using REST API calls to a hosted UI. The UI will be making call to the MLz for scoring and the result will be sent back to the CICS-COBOL program.

All sample code for this section is within

```
ai-st-credit-risk-assessment/zST-model-integration-CICS
```

Prerequisites

1. Must have access to z/OS CICS Environment
2. Must have MLz installed
3. Must have model deployed with the CICS scoring server as scoring service

Integrate into CICS Application

Integrate into CICS application

A sample COBOL file for the below integration can be found here:

```
ai-st-credit-risk-assessment/zST-model-
integration-CICS/CRAURL.cbl
```

1. Update @HOSTNAME and @PORTNUM within the sample CRAURL.cbl provided

2. Create connection with the hosted UI, using the Web Open command. Mention the host number & port number where the UI is hosted

```
move 'web open '                    to ws-step.
EXEC CICS WEB OPEN
          http
          host(ws-host)
          portnumber(ws-portnumber)
          SESSTOKEN(ws-sesstoken)
          RESP(ws-resp)
          RESP2(ws-resp2)
END-EXEC.
```

3. Using CICS ASSIGN get the application id of the UI

```
move 'ASSIGN APPLID '          to ws-step.
EXEC CICS ASSIGN
          APPLID(ws-applid)
END-EXEC.
```

Integrate into CICS Application

4. Supply all the inputs required by the UI service along with the API path of the inference

```
AIONZ.POC.COBOL(CRAURL) - 01.99                    Columns 00
=>                                                  Scroll

  *supply all the input values
      MOVE '25'                 TO ws-input-age
      MOVE '6960'               TO ws-input-income
      MOVE 'MORTGAGE'           TO ws-input-ownership
      MOVE '1'                  TO ws-input-length
      MOVE '55000'              TO ws-input-amt
      MOVE '/cra/predictwml'  to ws-path
      MOVE LENGTH OF WS-PATH TO WS-PATH-LEN
```

5. Prepare the json data for the REST API call using COBOL's String statement

```
AIONZ.POC.COBOL(CRAURL) - 01.99                    Columns 00001
d ===>                                              Scroll ===
        STRING '{"age":"' DELIMITED BY SPACES
               ws-input-age          DELIMITED BY SPACES
               '","annual_income":"' DELIMITED BY SPACES
               ws-input-income       DELIMITED BY SPACES
               '","emp_length":"'    DELIMITED BY SPACES
               ws-input-length       DELIMITED BY SPACES
               '","home_ownership":"' DELIMITED BY SPACES
               ws-input-ownership    DELIMITED BY SPACES
               '","loan_amount":"'   DELIMITED BY SPACES
               ws-input-amt          DELIMITED BY SPACES
           '"}'                        DELIMITED BY SPACES
          INTO WS-FROM.
```

6. Use the web converse command in CICS to pass the data to the UI backend service and get the response

```
AIONZ.POC.COBOL(CRAURL) - 01.99                    Co
=>
        EXEC CICS WEB CONVERSE
            SESSTOKEN    (WS-SESSTOKEN)
            POST
            MEDIATYPE    (WS-MEDIATYPE)
            PATH         (WS-PATH)
            PATHLENGTH   (WS-PATH-LEN)
            FROM         (ws-FROM)
            STATUSCODE   (WS-status)
            STATUSTEXT   (WS-statusdata)
            STATUSLEN    (Ws-statuslen)
            INTO         (Ws-recdata)
            TOLENGTH     (Ws-reclen)
            CLOSE
            RESP         (WS-RESP)
            RESP2        (WS-RESP2)
        END-EXEC.
```

Integrate into CICS Application

7. Close the web connection to the server

```
* Close the Session to the Remote Server
    EXEC CICS WEB CLOSE SESSTOKEN(ws-sesstoken)
    END-EXEC.
    PERFORM 0700-CHK-RESP.
```

8. Process the response received from API call

```
    UNSTRING ws-recdata delimited by '"loan_status":'
        into ws-str4
             ws-loan-status
    END-UNSTRING.
```

9. Handle the error codes as needed

```
    MOVE SPACES      TO WS-MESSAGE
    IF WS-RESP NOT EQUAL ZERO
       MOVE WS-RESP      TO err-resp
       MOVE WS-RESP2     TO ERR-RESP2
       STRING WS-STEP DELIMITED BY SPACES
              'failed with RESP = '
              ERR-resp delimited by spaces
              'RESP2 = '
              ERR-resp2 delimited by spaces
             into ws-message
       END-STRING
       display 'failure for ' ws-step
       display ws-message
       EXEC CICS RETURN
       END-EXEC
```

Integrate into CICS Application

10. Compile the COBOL program. Sample compile jcl provided here:

```
ai-st-credit-risk-assessment/zST-
model-integration-CICS/COMPILE.jcl
```

```
//COMP1   EXEC PGM=IGYCRCTL,REGION=0M,COND=(4,LT),
//  PARM=('NODYNAM,LIB,MAP,XREF,ADATA,CICS(''COBOL3'')')
//STEPLIB  DD DISP=SHR,DSN=IGY.SIGYCOMP
//         DD DISP=SHR,DSN=CICSTS61.CICS.SDFHLOAD
//SYSIN    DD DISP=SHR,DSN=AIONZ.POC.COBOL(CRAURL)
//SYSPRINT DD SYSOUT=*
//SYSLIN   DD DSN=&&LOADSET,DISP=(NEW,KEEP),
//            UNIT=SYSDA,SPACE=(80,(250,10))
//SYSMDECK DD SYSOUT=*
//SYSUT1   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT2   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT3   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT4   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT5   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT6   DD UNIT=SYSDA,SPACE=(460,(350,100))
//SYSUT7   DD UNIT=SYSDA,SPACE=(460,(350,100))
```

11. Define the transaction. COPY PASTE THE LINES ONE-BY-ONE

```
CEDA DEFINE TRANS(<transaction name>)
GROUP(<group name>)
```

```
PROGRAM(<program name>)
```

```
DESCRIPTION(<transaction description>)
```

```
CEDA DEFINE TRANS(R001) GROUP(TECHGRP)
PROGRAM(CRAURL)
DESCRIPTION(Transaction to execute CRAURL)
```

12. Define the program

```
CEDA DEFINE PROGRAM(<program name>)
GROUP(<group name>)
```

```
LANGUAGE(COBOL) DESCRIPTION(<program
description>)
```

```
CEDA DEFINE PROGRAM(CRAURL) GROUP(TECHGRP)
LANGUAGE(COBOL) DESCRIPTION(PROGRAM FOR CRA)
```

Integrate into CICS Application

13. Install the transaction and program in the CICS region. Execute the below mentioned command to define the transaction replacing transaction name with the transaction name, group name with the name of the group & program name with name of the COBOL program, transaction description with appropriate description for the transaction, finally, program description with appropriate description for the program

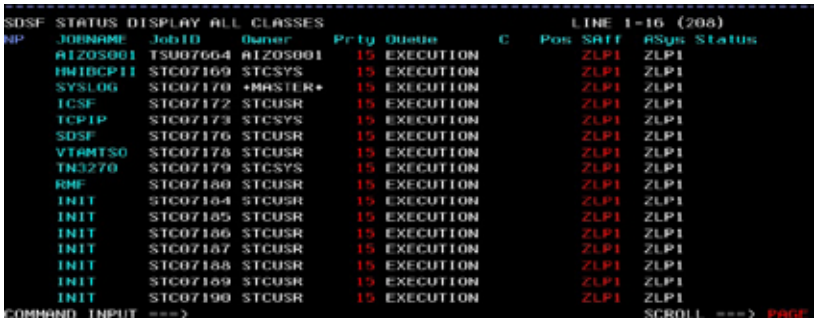
```
CEDA INS TRANS(<transaction name>)
GROUP(<group name>)
```


```
CEDA INS PROGRAM(<program name>)
GROUP(<group name>)
```

14. To invoke the transaction Type the transaction name and hit Enter

15. Verify the result

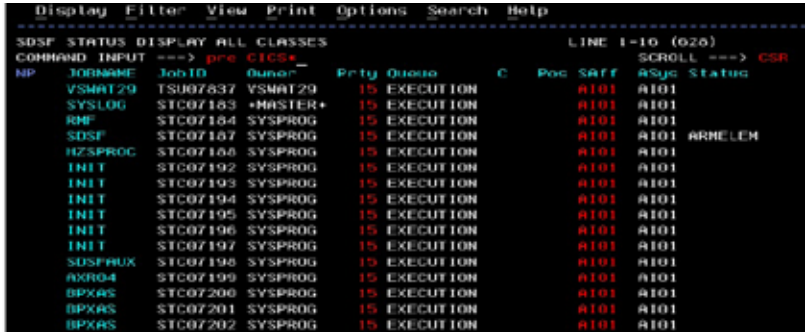
- Lets go back to the TSO screen. Navigate to the Spool



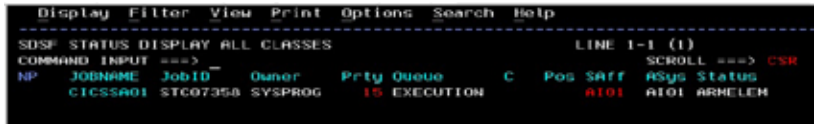
Integrate into CICS Application

- To check the started task
  - Go to Spool `pre CICS*`

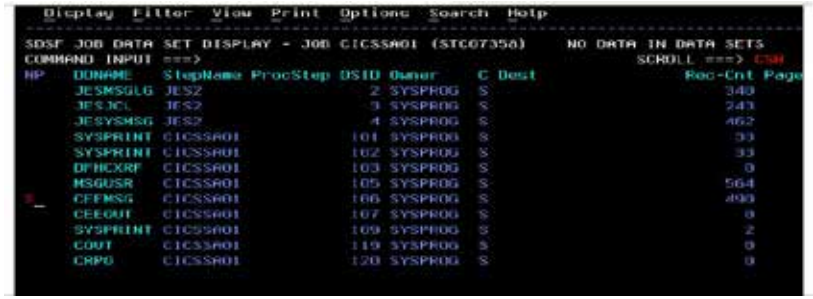
    This will list the active CICS regions



  - Put `?` to see the details of the Spool job.



  - Check the `CEEMSG` dataset name



Integrate into CICS Application

- Check the displays from the module

AI model integration complete