Credit Card Fraud Prediction using IBM Auto AI

1-INTRODUCTION:

1.1 Overview:

This project discusses building a system for creating predictions that can be used in different scenarios. It focuses on predicting fraudulent transactions, which can reduce monetary loss and risk mitigation.

1.2 Purpose:

This project aims at building a web App which automatically estimates if there is a fraud risk by taking the input values.

2-LITERATURE SURVEY:

2.1 Existing problem:

The majority of detection methods combine a variety of fraud detection datasets to form a connected overview of both valid and non-valid payment data to make a decision. This decision must consider IP address, geolocation, device identification, “BIN” data, global latitude/longitude, historic transaction patterns, and the actual transaction information. In practice, this means that merchants and issuers deploy analytically based responses that use internal and external data to apply a set of business rules or analytical algorithms to detect fraud. [2]

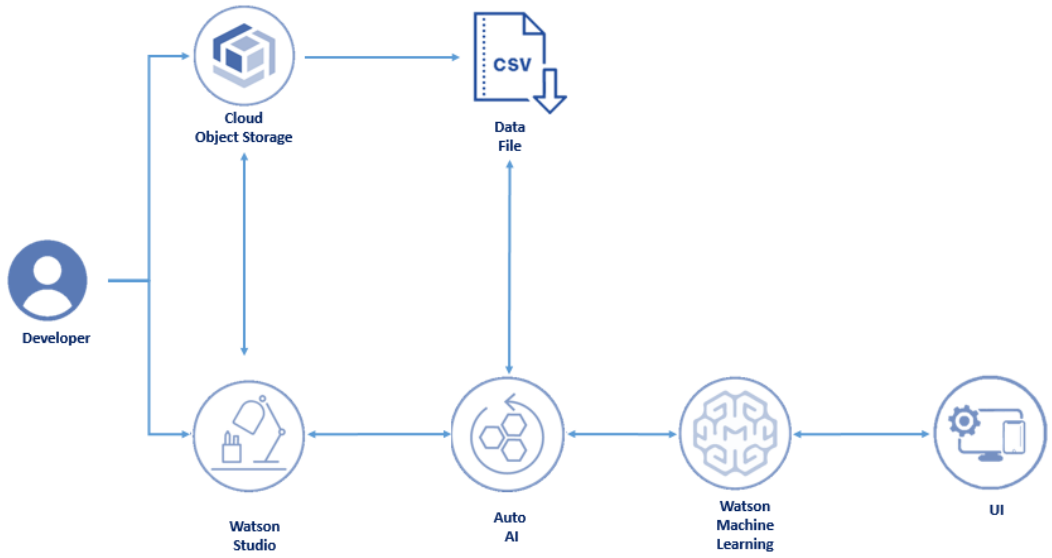
2.2 Proposed solution:

Using IBM AutoAI, all of the tasks involved in building predictive models for different requirements are automated. The model is generated from a data set that includes the gender, married, dependents, education, self employed, applicant income, co-applicant income, loan amount, loan term, credit history, housing and locality, and predicts the fraud risk.
To help simplify an AI lifecycle management cycle, AutoAI automates:

- Data preparation
- Model development
- Feature engineering
- Hyper-parameter optimization

3-THEORITICAL ANALYSIS:

3.1 Block diagram:



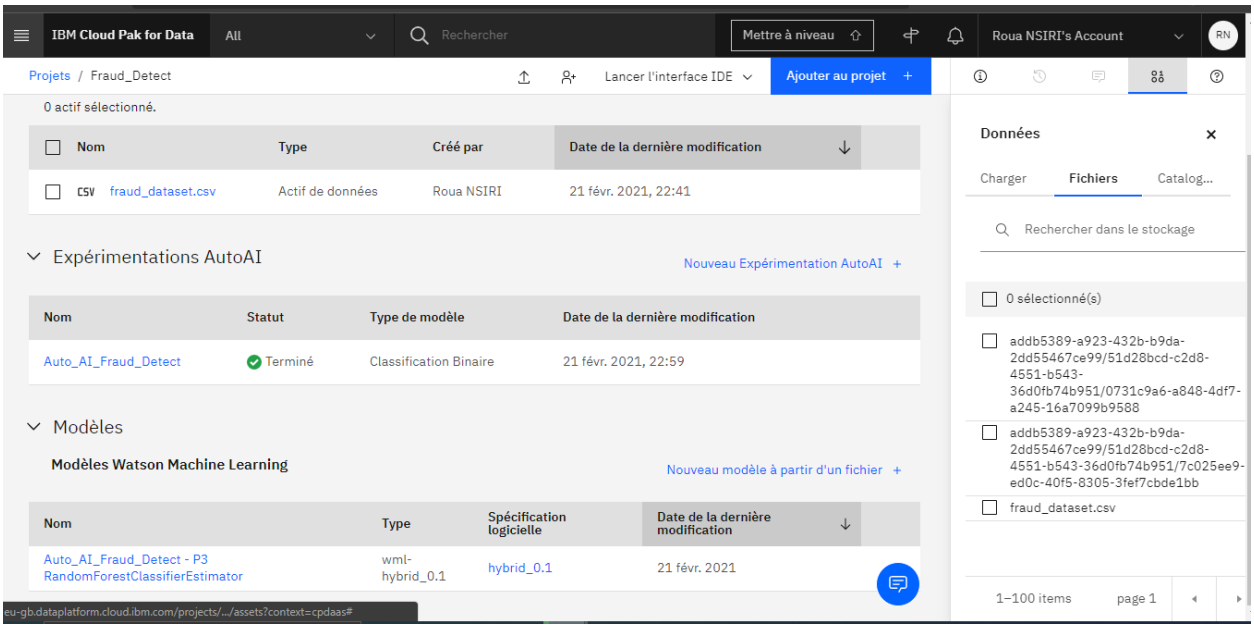
3.2 Hardware / Software designing :

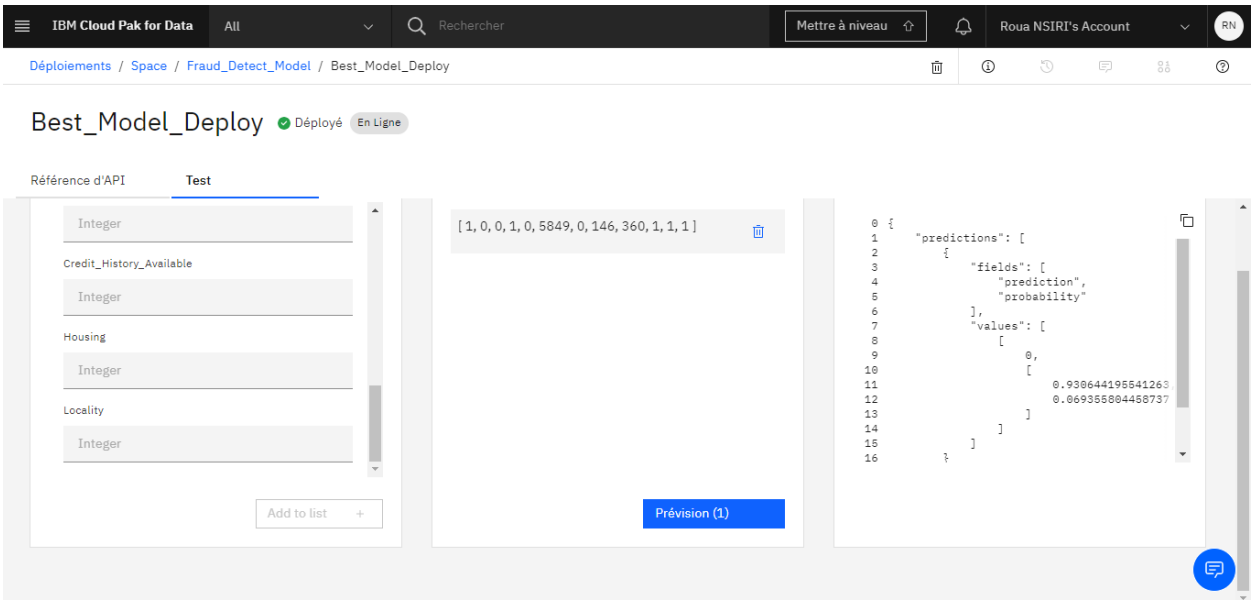
• **IBM Watson Studio:** : IBM's software platform for data science. The platform consists of a workspace that includes multiple collaboration and open-source tools for use in data science.

•**IBM Watson Machine Learning** :A service that enables you to create, train, and deploy self-learning models using an automated, collaborative workflow.

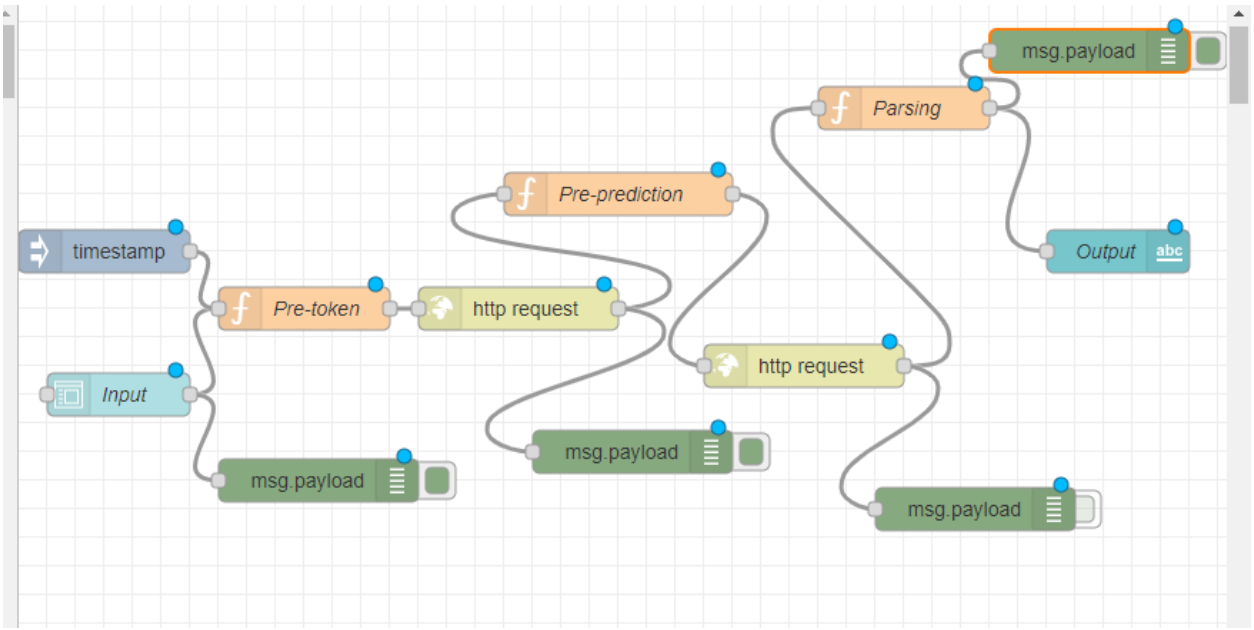
•**Node Red** : a flow-based development tool for visual programming developed originally by IBM for wiring together hardware devices, APIs and online services as part of the Internet of Things.

4-EXPERIMENTAL INVESTIGATIONS:





5-FLOWCHART:



The main flowchart is composed of:

- **Input node**: Contains the gender, married, dependents, education, self employed, applicant income, co-applicant income, loan amount, loan term, credit history, housing and locality parameters.

-**Pre-token function node**

- **Http request node**: Configured with POST method and given the URL address: "https://iam.cloud.ibm.com/identity/token". It returns a parsed JSON object.

- Pre-prediction function node

- Http request node: Configured with POST method and given the model deployment URL

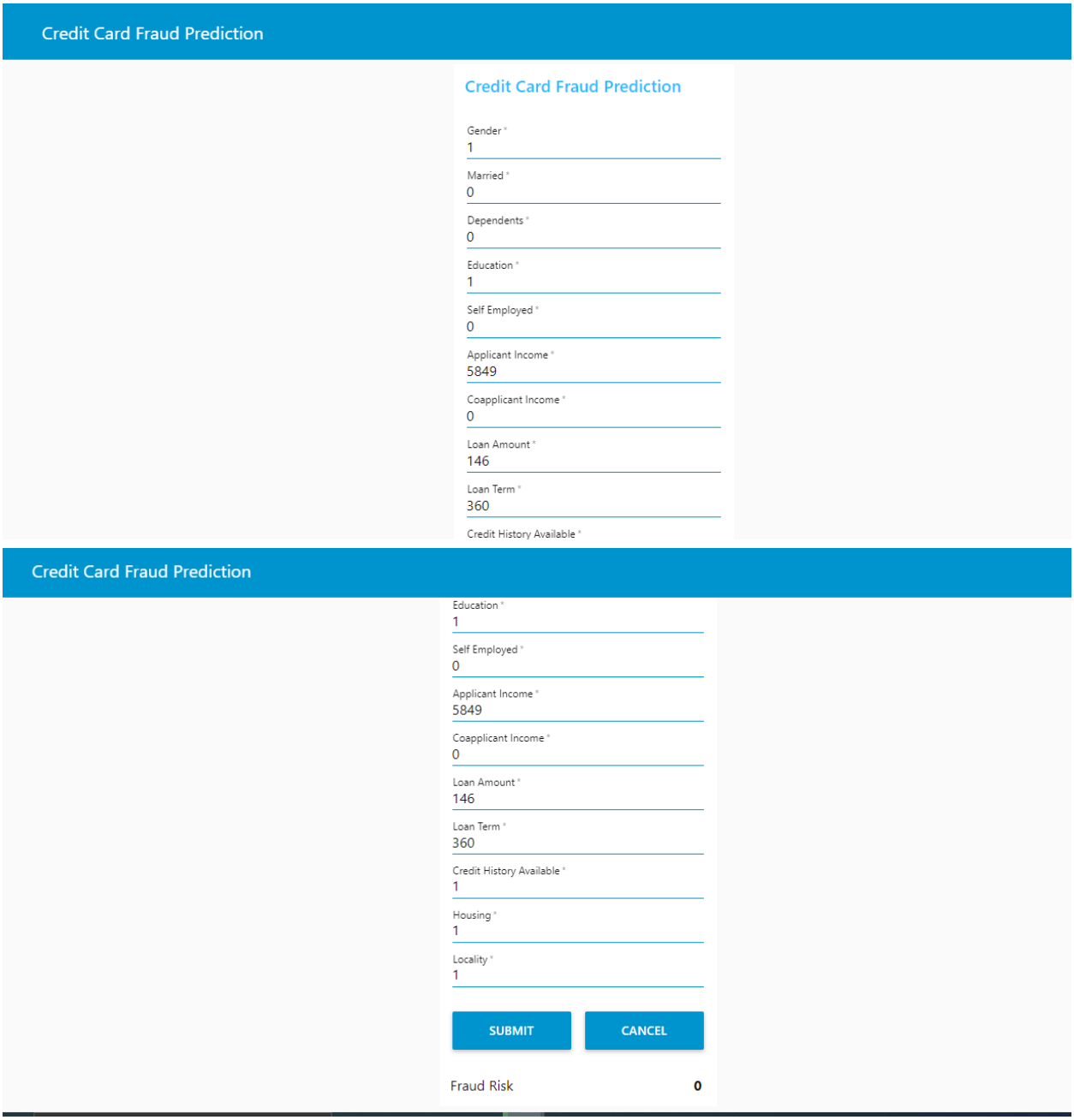
address

- Parsing function node

- Output node: Contains the fraud risk prediction value.

6-RESULT:

By testing the 1st of fraud_dataset.cv, I found the same prediction values of the fraud risk, proving the accuracy of the machine learning model.



7-ADVANTAGES & DISADVANTAGES:

7.1 Advantages:

- Fast model selection: The top performing models are quickly selected.
- Quick start
- AI lifecycle management: Enforced consistency and repeatability of end-to-end machine learning and AI development. [3]

7.2 Disadvantages:

1. Maintenance
2. Doesn't process structured data directly
3. Increasing rate of data, with limited resources

8-APPLICATIONS:

1. Healthcare
2. Legal
3. Retail
4. Financial

9-CONCLUSION:

The project focuses on predicting fraudulent transactions, which can reduce monetary loss and risk mitigation.

10-FUTURE SCOPE:

This project is a solution to predicting the risk of fraudulent transactions through credit cards, by using IBM Watson Machine Learning Service and Node-RED.

10.1 Future scope:

Machine learning techniques have been used to detect credit card frauds but no fraud detection systems have been able to offer great efficiency to date. Recent development of deep learning has been applied to solve complex problems in various areas. Experimental results show great performance of some deep learning methods against traditional machine learning models and imply that the proposed approaches can be implemented effectively for real-world credit card fraud detection systems. [4]

10.2 Bibliography :

[1] Smartinternz.com. 2021. SmartInternz. [online] Available at: [Accessed 7 March 2021].
[2] Chuprina, R., 2020. Credit Card Fraud Detection and Prevention: The Complete Guide. 8[online] DEV Community. Available at: [Accessed 7 March 2021].
[3] Syamala, M., 2020. SmartPracticeschool/SPS-6811-Credit-Card-Fraud-Prediction-using-IBM-Auto-AI. [online] GitHub. Available at: [Accessed 7 March 2021].
[4] Nguyen, T., Tahir, H., Abdelrazek, M. and Babar, A., 2021. Deep Learning Methods for Credit Card Fraud Detection. [online] arXiv.org. Available at: [Accessed 7 March 2021].

11-APPENDIX:

Package installation

```
!pip install ibm-watson-machine-learning | tail -n 1
!pip install -U autoai-libs | tail -n 1
```

AutoAI experiment metadata

```
from ibm_watson_machine_learning.helpers import DataConnection, S3Connection, S3Location

training_data_reference = [DataConnection(
   connection=S3Connection(
      api_key='hr1Z88pY2XgDrOXM42dzoE8hl2woIq5KaA1W74FWaFUT',
      auth_endpoint='https://iam.bluemix.net/oidc/token/',
      endpoint_url='https://s3.eu-geo.objectstorage.softlayer.net'
   ),
      location=S3Location(
      bucket='frauddetect-donotdelete-pr-w5ruoaoh8lotho',
      path='fraud_dataset.csv'
   )),
]
training_result_reference = DataConnection(
   connection=S3Connection(
      api_key='hr1Z88pY2XgDrOXM42dzoE8hl2woIq5KaA1W74FWaFUT',
      auth_endpoint='https://iam.bluemix.net/oidc/token/',
      endpoint_url='https://s3.eu-geo.objectstorage.softlayer.net'
   ),
   location=S3Location(
      bucket='frauddetect-donotdelete-pr-w5ruoaoh8lotho',

path='auto_ml/c1d0161e-0f41-4640-8f43-cd82d0a6bd3a/wml_data/8a2a36b1-77d3-483d-bded-0a6994a5cb61/data/automl',

model_location='auto_ml/c1d0161e-0f41-4640-8f43-cd82d0a6bd3a/wml_data/8a2a36b1-77d3-483d-bded-0a6994a5cb61/data/automl/cognito_output/Pipeline1/model.pickle',

training_status='auto_ml/c1d0161e-0f41-4640-8f43-cd82d0a6bd3a/wml_data/8a2a36b1-77d3-483d-bded-0a6994a5cb61/training-status.json'
   ))
experiment_metadata = dict(
```

```
    prediction_type='classification',
    prediction_column='Fraud_Risk',
    test_size=0.1,
    scoring='accuracy',
    project_id='7616eb24-1655-48f4-aa9f-3643549a07c2',
    space_id='None',
    deployment_url='https://eu-gb.ml.cloud.ibm.com',
    csv_separator=',',
    random_state=33,
    max_number_of_estimators=2,
    daub_include_only_estimators=None,
    training_data_reference=training_data_reference,
    training_result_reference=training_result_reference,
    positive_label=1
```

Read training data

```
df = training_data_reference[0].read(csv_separator=experiment_metadata['csv_separator'])
df.dropna('rows', how='any', subset=[experiment_metadata['prediction_column']], inplace=True)
```

Train and test data split:

```
from sklearn.model_selection import train_test_split

df.drop_duplicates(inplace=True)
X = df.drop([experiment_metadata['prediction_column']], axis=1).values
y = df[experiment_metadata['prediction_column']].values

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=experiment_metadata['test_size'],
                                                    stratify=y, random_state=experiment_metadata['random_state'])
```

Make pipeline

```
from autoai_libs.transformers.exportable import NumpyColumnSelector
from autoai_libs.transformers.exportable import CompressStrings
from autoai_libs.transformers.exportable import NumpyReplaceMissingValues
from autoai_libs.transformers.exportable import NumpyReplaceUnknownValues
from autoai_libs.transformers.exportable import boolean2float
from autoai_libs.transformers.exportable import CatImputer
from autoai_libs.transformers.exportable import CatEncoder
import numpy as np
from autoai_libs.transformers.exportable import float32_transform
from sklearn.pipeline import make_pipeline
from autoai_libs.transformers.exportable import FloatStr2Float
from autoai_libs.transformers.exportable import NumImputer
```

```
from autoai_libs.transformers.exportable import OptStandardScaler
from sklearn.pipeline import make_union
from autoai_libs.transformers.exportable import NumpyPermuteArray
from autoai_libs.cognito.transforms.transform_utils import TA2
import autoai_libs.utils.fc_methods
from autoai_libs.cognito.transforms.transform_utils import FS1
from autoai_libs.cognito.transforms.transform_utils import TAM
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
```

Pre-processing & Estimator:

```
numpy_column_selector_0 = NumpyColumnSelector(
    columns=[0, 1, 2, 3, 4, 8, 9, 10, 11]
)
compress_strings = CompressStrings(
    compress_type="hash",
    dtypes_list=[
        "int_num",
        "int_num",
        "int_num",
        "int_num",
        "int_num",
        "int_num",
        "int_num",
        "int_num",
        "int_num",
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
    misslist_list=[[], [], [], [], [], [], [], [], []],
)
numpy_replace_missing_values_0 = NumpyReplaceMissingValues(
    missing_values=[], filling_values=float("nan")
)
numpy_replace_unknown_values = NumpyReplaceUnknownValues(
    filling_values=float("nan"),
    filling_values_list=[
        float("nan"),
        float("nan"),
        float("nan"),
        float("nan"),
```

```
    float("nan"),
    float("nan"),
    float("nan"),
    float("nan"),
    float("nan"),
],
known_values_list=[
    [0, 1],
    [0, 1],
    [0, 1, 2, 3],
    [0, 1],
    [0, 1],
    [
        12,
        14,
        36,
        60,
        68,
        84,
        107,
        109,
        120,
        159,
        160,
        178,
        180,
        197,
        214,
        215,
        218,
        240,
        250,
        281,
        295,
        296,
        300,
        306,
        316,
        323,
```

```
            324,
            328,
            337,
            338,
            341,
            343,
            346,
            360,
            404,
            418,
            421,
            459,
            465,
            466,
            476,
            480,
        ],
        [0, 1],
        [0, 1],
        [1, 2, 3],
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
)
cat_imputer = CatImputer(
    strategy="most_frequent",
    missing_values=float("nan"),
    sklearn_version_family="23",
)
cat_encoder = CatEncoder(
    encoding="ordinal",
    categories="auto",
    dtype=np.float64,
    handle_unknown="error",
    sklearn_version_family="23",
)
pipeline_0 = make_pipeline(
    numpy_column_selector_0,
    compress_strings,
    numpy_replace_missing_values_0,
```

```
    numpy_replace_unknown_values,
    boolean2float(),
    cat_imputer,
    cat_encoder,
    float32_transform(),
)
numpy_column_selector_1 = NumpyColumnSelector(columns=[5, 6, 7])
float_str2_float = FloatStr2Float(
    dtypes_list=["int_num", "int_num", "int_num"],
    missing_values_reference_list=[],
)
numpy_replace_missing_values_1 = NumpyReplaceMissingValues(
    missing_values=[], filling_values=float("nan")
)
num_imputer = NumImputer(strategy="median", missing_values=float("nan"))
opt_standard_scaler = OptStandardScaler(
    num_scaler_copy=None,
    num_scaler_with_mean=None,
    num_scaler_with_std=None,
    use_scaler_flag=False,
)
pipeline_1 = make_pipeline(
    numpy_column_selector_1,
    float_str2_float,
    numpy_replace_missing_values_1,
    num_imputer,
    opt_standard_scaler,
    float32_transform(),
)
union = make_union(pipeline_0, pipeline_1)
numpy_permute_array = NumpyPermuteArray(
    axis=0, permutation_indices=[0, 1, 2, 3, 4, 8, 9, 10, 11, 5, 6, 7]
)
ta2 = TA2(
    fun=np.add,
    name="sum",
    datatypes1=[
        "intc",
        "intp",
```

```
    "int_",
    "uint8",
    "uint16",
    "uint32",
    "uint64",
    "int8",
    "int16",
    "int32",
    "int64",
    "short",
    "long",
    "longlong",
    "float16",
    "float32",
    "float64",
],
feat_constraints1=[autoai_libs.utils.fc_methods.is_not_categorical],
datatypes2=[
    "intc",
    "intp",
    "int_",
    "uint8",
    "uint16",
    "uint32",
    "uint64",
    "int8",
    "int16",
    "int32",
    "int64",
    "short",
    "long",
    "longlong",
    "float16",
    "float32",
    "float64",
],
feat_constraints2=[autoai_libs.utils.fc_methods.is_not_categorical],
col_names=[
    "Gender",
```

```
        "Married",
        "Dependents",
        "Education",
        "Self_Employed",
        "ApplicantIncome",
        "CoapplicantIncome",
        "LoanAmount",
        "Loan_Term",
        "Credit_History_Available",
        "Housing",
        "Locality",
    ],
    col_dtypes=[
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
    ],
)
fs1_0 = FS1(
    cols_ids_must_keep=range(0, 12),
    additional_col_count_to_keep=12,
    ptype="classification",
)
tam = TAM(
    tans_class=PCA(),
    name="pca",
    col_names=[
        "Gender",
        "Married",
        "Dependents",
```

```
        "Education",
        "Self_Employed",
        "ApplicantIncome",
        "CoapplicantIncome",
        "LoanAmount",
        "Loan_Term",
        "Credit_History_Available",
        "Housing",
        "Locality",
        "sum(ApplicantIncome__CoapplicantIncome)",
        "sum(ApplicantIncome__LoanAmount)",
        "sum(ApplicantIncome__Loan_Term)",
        "sum(CoapplicantIncome__LoanAmount)",
        "sum(CoapplicantIncome__Loan_Term)",
        "sum(LoanAmount__Loan_Term)",
    ],
    col_dtypes=[
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
         np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
        np.dtype("float32"),
    ],
)
fs1_1 = FS1(
    cols_ids_must_keep=range(0, 12),
```

```
    additional_col_count_to_keep=12,
    ptype="classification",
)
random_forest_classifier = RandomForestClassifier(
    class_weight="balanced",
    max_depth=4,
    max_features=0.4872195557592475,
    n_estimators=87,
    n_jobs=1,
    random_state=33,
)
```

Pipeline

```
pipeline = make_pipeline(
    union,
    numpy_permute_array,
    ta2,
    fs1_0,
    tam,
    fs1_1,
    random_forest_classifier,)
```

Train pipeline model

```
from sklearn.metrics import get_scorer
scorer = get_scorer(experiment_metadata['scoring'])
```

Fit pipeline model

```
pipeline.fit(train_X,train_y)
```

Test pipeline model

```
score = scorer(pipeline, test_X, test_y)
print(score)
```