Machine learning Model Deployment With IBM Cloud Watson Studio

Project Objective:

The project's objective is to develop a predictive model that can be used to make real-time predictions based on a given dataset. The predictive use case, in this case, could be anything from sales forecasting, fraud detection, customer churn prediction, or any other scenario where making predictions is valuable.

Design Thinking Process:

Introduction:

The "Predictive Model Deployment for Real-Time Analytics" project has set the stage for the development and deployment of a predictive model using IBM Cloud Watson Studio. To further elevate the project's success and achieve our goal of becoming proficient in predictive analytics.

Problem Statement Revisited

The primary problem statement remains focused on creating a machine learning model that can predict outcomes in real-time. However, the innovative aspect lies in improving the model's predictive accuracy and overall performance through advanced techniques.

Design Thinking Refinement:

Predictive Use Case

Enhancement through Innovation: We will explore ensemble learning techniques such as Random Forests and Gradient Boosting to improvethe model's predictive capabilities. Ensemble methods combine multiple base models to create a more robust and accurate prediction.

Dataset Selection

Enhancement through Innovation: We will consider more extensive feature engineering and data augmentation techniques to enhance thedataset's quality. This may include generating synthetic data points, which can be especially valuable for cases with imbalanced classes.

Model Training

Enhancement through Innovation: Instead of relying solely on a single machine learning algorithm, we will experiment with multiple algorithms and optimize their hyper parameters. Hyper parameter tuning, using techniques like grid search or random search, will allow usto find the best combination of parameters for each algorithm.

Model Deployment

Enhancement through Innovation: While deploying the model as a webservice, we will explore containerization solutions, such as Docker, to package the model along with its dependencies.

Integration

Enhancement through Innovation: During integration, we will focus on optimizing the API's responsiveness and scalability. We will explore loadbalancing techniques to distribute prediction requests efficiently, especially in high-traffic scenarios.

Innovative Approaches:

Ensemble Learning

Innovation: We will experiment with ensemble methods such as Random Forest, which combines multiple decision trees to improve prediction accuracy. Additionally, we will explore boosting algorithmslike AdaBoost or Gradient Boosting to further enhance model performance.

Hyperparameter Tuning

Innovation: Hyperparameter tuning will be a critical part of our modeldevelopment process. We will systematically search for the best hyperparameters using techniques like grid search and random search.This will enable us to fine-tune the model for optimal performance.

Data Augmentation

Innovation: For cases where data availability is limited, we will exploredata augmentation techniques to artificially increase the dataset's size. This includes techniques like oversampling minority classes, generating synthetic samples, and applying advanced data augmentation libraries.

Containerization

Innovation: To simplify model deployment and ensure consistent behavior across different environments, we will containerize the modelusing Docker. Containerization offers portability and scalability advantages.

Load Balancing

Innovation: For seamless integration and real-time predictions, wewill implement load balancing strategies to distribute incoming prediction requests evenly across multiple instances of the deployedmodel. This ensures optimal system performance, especially during high-demand periods.

Expected Outcomes:

By infusing innovation into the project through theimplementation of ensemble methods, hyperparameter tuning, data augmentation, containerization, and load balancing, we anticipate thefollowing outcomes.

Improved Model Accuracy: The ensemble methods and hyperparameter tuning will lead to a more accurate predictive model,resulting in better real-time predictions.

Enhanced Model Robustness: Data augmentation techniques will increase the model's ability to generalize to different scenarios, evenwith limited data.

Efficient Deployment: Containerization will simplify the deploymentprocess and make it more reliable and consistent.

Scalable Integration: Load balancing will ensure that our integrated model can handle high prediction request loads without performancedegradation.

Model Training and Deployment Process:

Setup

installation:

```
!pip install ibm-watson-machine-learning | tail -n 1
!pip install autoai-libs==1.14.13 | tail -n 1
!pip install scikit-learn==1.1.1 | tail -n 1
!pip install 'snapml==1.8.10' | tail -n 1
```

AutoAI experiment metadata:

```
from ibm_watson_machine_learning.helpers import DataConnection
from ibm_watson_machine_learning.helpers import ContainerLocation

training_data_references = [
    DataConnection(
        data_asset_id='0ca04eed-2890-4120-836e-2eaa0aef6456'
    ),
]
training_result_reference = DataConnection(
    location=ContainerLocation(
        path='auto_ml/ca8c2f3a-3c47-40ef-b7f5-b939eeb50254/wml_data/71c07cd1-54e2-4652-843e-a5524b86ad64/data/automl',
        model_location='auto_ml/ca8c2f3a-3c47-40ef-b7f5-b939eeb50254/wml_data/71c07cd1-54e2-4652-843e-a5524b86ad64/data/automl/model.zip',
        training_status='auto_ml/ca8c2f3a-3c47-40ef-b7f5-b939eeb50254/wml_data/71c07cd1-54e2-4652-843e-a5524b86ad64/training-status.json'
    )
)
```

The following cell contains input parameters provided to run the AutoAI experiment inWatson Studio

```
experiment_metadata = dict(
    prediction_type='binary',
    prediction_column='Churn',
    holdout_size=0.1,
    scoring='accuracy',
    csv_separator=',',
    random_state=33,
    max_number_of_estimators=2,
    training_data_references=training_data_references,
    training_result_reference=training_result_reference,
    deployment_url='https://eu-gb.ml.cloud.ibm.com',
    project_id='144844f6-62a4-45aa-a51a-edf35a7740db',
    positive_label='True',
    drop_duplicates=True,
    include_batched_ensemble_estimators=[]
)
```

Set n_jobs parameter to the number of available CPUs

```
import os, ast
CPU_NUMBER = 2
if 'RUNTIME_HARDWARE_SPEC' in os.environ:
    CPU_NUMBER = int(ast.literal_eval(os.environ['RUNTIME_HARDWARE_SPEC'])['num_cpu'])
```

2. Watson Machine Learning connection

This cell defines the credentials required to work with the Watson Machine Learning service.

```
api_key = 'PUT_YOUR_APIKEY_HERE'
```

```
wml_credentials = {
    "apikey": api_key,
    "url": experiment_metadata['deployment_url']
}
```

```
from ibm_watson_machine_learning import APIClient

wml_client = APIClient(wml_credentials)

if 'space_id' in experiment_metadata:
    wml_client.set.default_space(experiment_metadata['space_id'])
else:
    wml_client.set.default_project(experiment_metadata['project_id'])

training_data_references[0].set_client(wml_client)
```

3. Pipeline inspection

Read training data

```
train_X, test_X, train_y, test_y = training_data_references[0].read(experiment_metadata=experiment_metadata, with_holdout_split=True, use_flight=False)
```

Create 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 TA1
from snapml import SnapRandomForestClassifier
```

Pre-processing & Estimator.

1.

```
numpy_column_selector_0 = NumpyColumnSelector(columns=[0, 2, 3, 4, 5, 16, 18])
compress_strings = CompressStrings(
    compress_type="hash",
    dtypes_list=[
        "char_str", "int_num", "char_str", "char_str", "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"),
    ],
    missing_values_reference_list=["", "-", "?", float("nan")],
)
cat_imputer = CatImputer(
    missing_values=float("nan"),
    sklearn_version_family="1",
    strategy="most_frequent",
)
cat_encoder = CatEncoder(
    encoding="ordinal",
    categories="auto",
    dtype=np.float64,
    handle_unknown="error",
    sklearn_version_family="1",
)
pipeline_0 = make_pipeline(
```

2.

```
    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=[1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17]
)
float_str2_float = FloatStr2Float(
    dtypes_list=[
        "int_num", "float_num", "int_num", "float_num", "float_num",
        "int_num", "float_num", "float_num", "int_num", "float_num",
        "float_num", "float_num",
    ],
    missing_values_reference_list=[],
)
numpy_replace_missing_values_1 = NumpyReplaceMissingValues(
    missing_values=[], filling_values=float("nan")
)
num_imputer = NumImputer(missing_values=float("nan"), strategy="median")
opt_standard_scaler = OptStandardScaler(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)
```

3.

```
numpy_permute_array = NumpyPermuteArray(
    axis=0,
    permutation_indices=[
        0, 2, 3, 4, 5, 16, 18, 1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17,
    ],
)
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=[
        "State", "Account length", "Area code", "International plan",
        "Voice mail plan", "Number vmail messages", "Total day minutes",
        "Total day calls", "Total day charge", "Total eve minutes",
        "Total eve calls", "Total eve charge", "Total night minutes",
        "Total night calls", "Total night charge", "Total intl minutes",
        "Total intl calls", "Total intl charge", "Customer service calls",
    ],
    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"),
```

4.

```
        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, 19),
    additional_col_count_to_keep=15,
    ptype="classification",
)
ta1 = TA1(
    fun=np.sqrt,
    name="sqrt",
    datatypes=["numeric"],
    feat_constraints=[
        autoai_libs.utils.fc_methods.is_non_negative,
        autoai_libs.utils.fc_methods.is_not_categorical,
    ],
    col_names=[
        "State", "Account length", "Area code", "International plan",
        "Voice mail plan", "Number vmail messages", "Total day minutes",
        "Total day calls", "Total day charge", "Total eve minutes",
        "Total eve calls", "Total eve charge", "Total night minutes",
        "Total night calls", "Total night charge", "Total intl minutes",
        "Total intl calls", "Total intl charge", "Customer service calls",
        "sum(State__Total day minutes)",
        "sum(Total day minutes__Total day calls)",
        "sum(Total day minutes__Total day charge)",
        "sum(Total day minutes__Total eve minutes)",
        "sum(Total day minutes__Total eve calls)",
        "sum(Total day minutes__Total eve charge)",
        "sum(Total day minutes__Total night calls)",
        "sum(Total day minutes__Total night charge)",
        "sum(Total day minutes__Total intl minutes)",
```

5.

```
        "sum(Total day charge__Total eve charge)",
        "sum(Total day charge__Total night charge)",
        "sum(Total day charge__Total intl minutes)",
        "sum(Total day charge__Total intl charge)",
    ],
    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"), 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, 19),
    additional_col_count_to_keep=15,
    ptype="classification",
)
snap_random_forest_classifier = SnapRandomForestClassifier(
    gpu_ids=np.array([0], dtype=np.uint32),
    max_depth=5,
    max_features=0.7037824628016168,
    n_estimators=97,
    n_jobs=CPU_NUMBER,
    random_state=33,
)
```

4. Pipeline:

```
pipeline = make_pipeline(
    union,
    numpy_permute_array,
    ta2,
    fs1_0,
    ta1,
    fs1_1,
    snap_random_forest_classifier,
)
```

Train pipeline model

Define scorer from the optimization metric

This cell constructs the cell scorer based on the experiment metadata.

```
from sklearn.metrics import get_scorer

scorer = get_scorer(experiment_metadata['scoring'])
```

Fit pipeline model

```
pipeline.fit(train_X.values, train_y.values.ravel());
```

5.Test pipeline model

Score the fitted pipeline with the generated scorer using the holdout dataset.

```
]: score = scorer(pipeline, test_X.values, test_y.values)
   print(score)
```

```
]: pipeline.predict(test_X.values[:5])
```

Store the mode

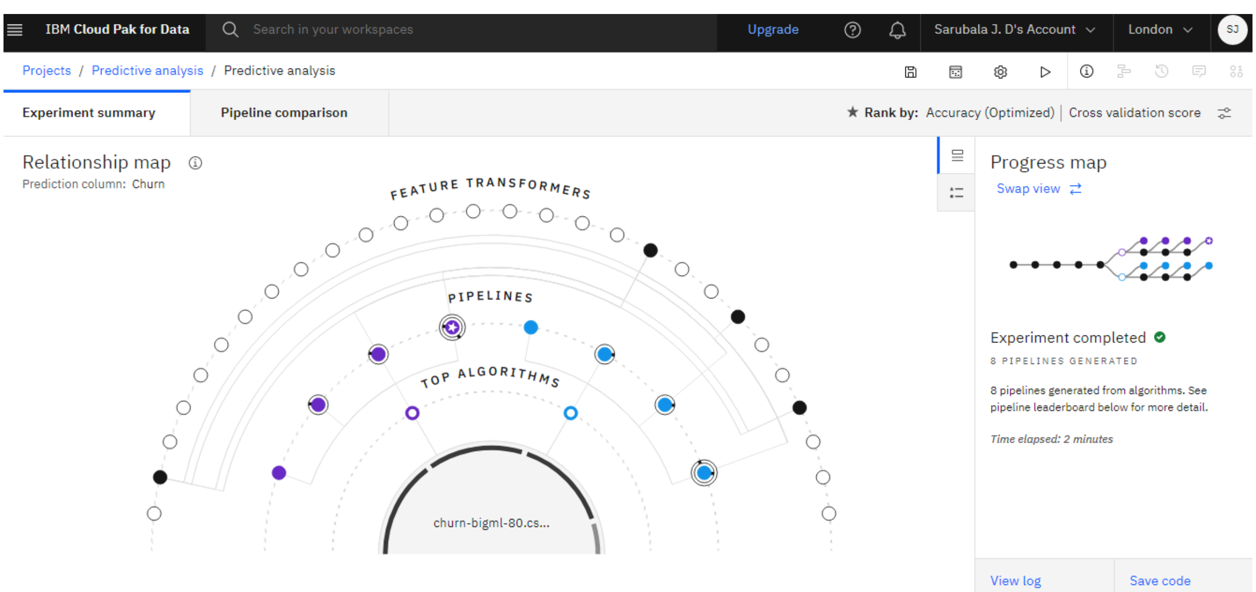
```
model_metadata = {
    wml_client.repository.ModelMetaNames.NAME: 'Trained AutoAI pipeline'
}

stored_model_details = wml_client.repository.store_model(model=pipeline, meta_props=model_metadata, experiment_metadata=experiment_metadata)
```

Inspect the stored model details.

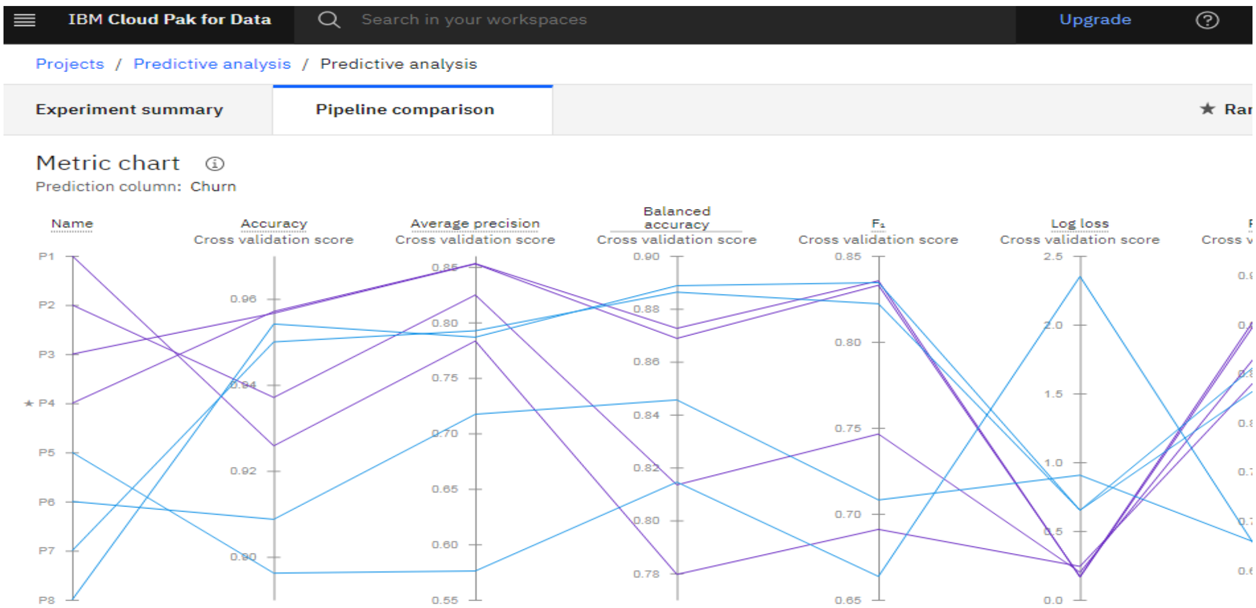
```
stored_model_details
```

DEPLOYMENT :

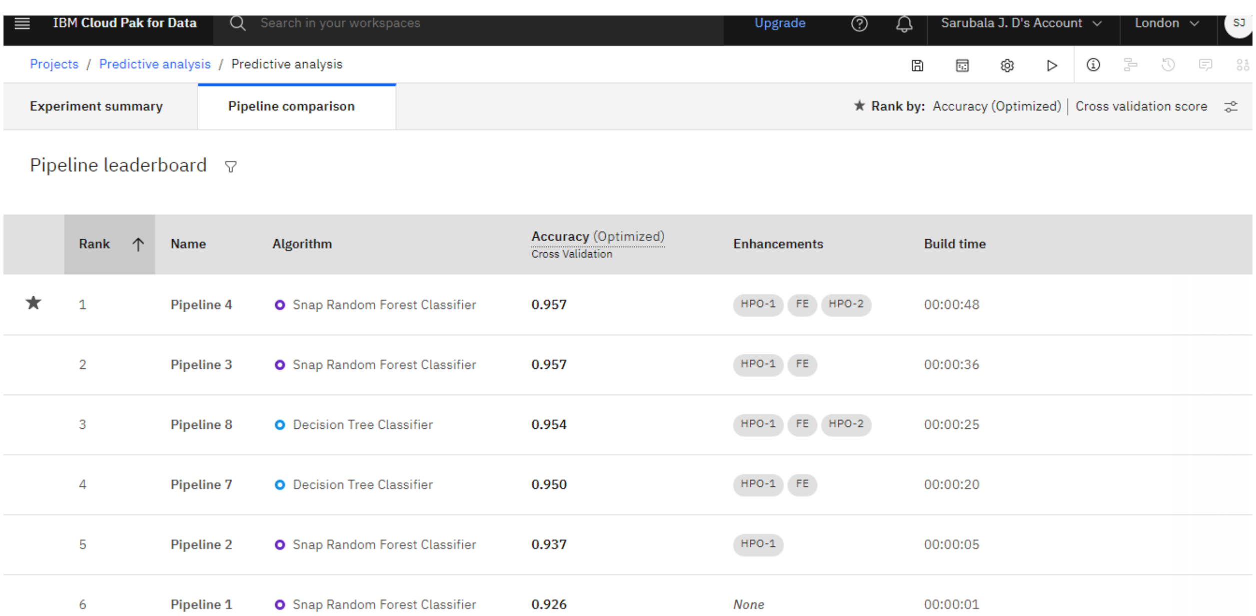
STEP 1: Experiment Summary:



STEP 2: Pipeline comparision

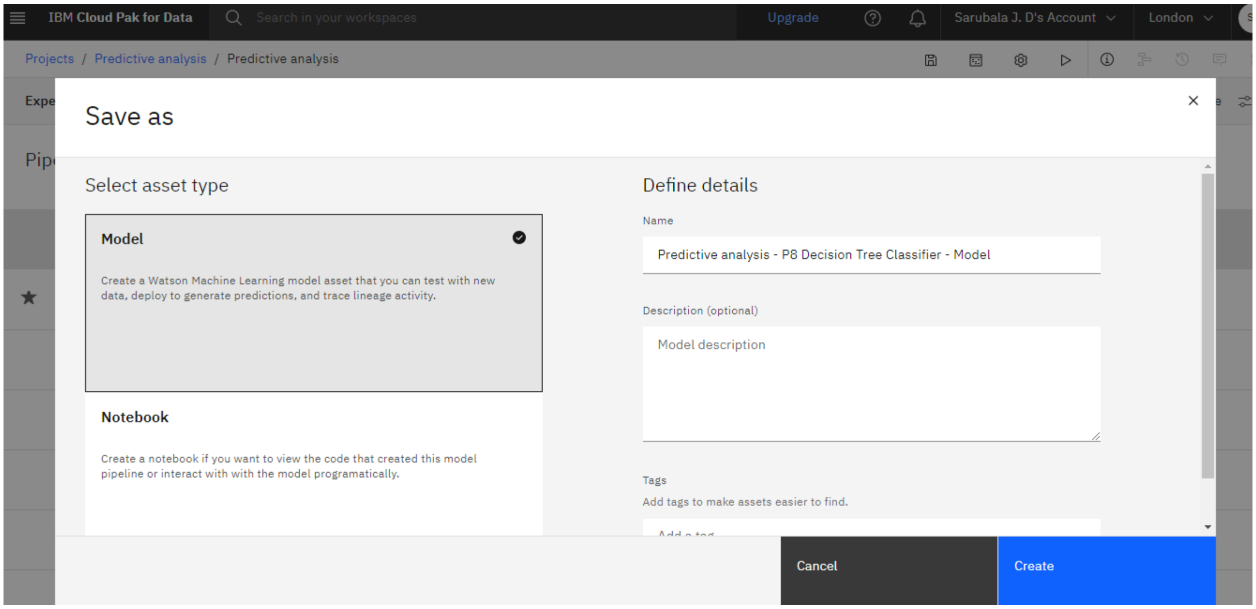


STEP 3: Pipeline Leaderboard

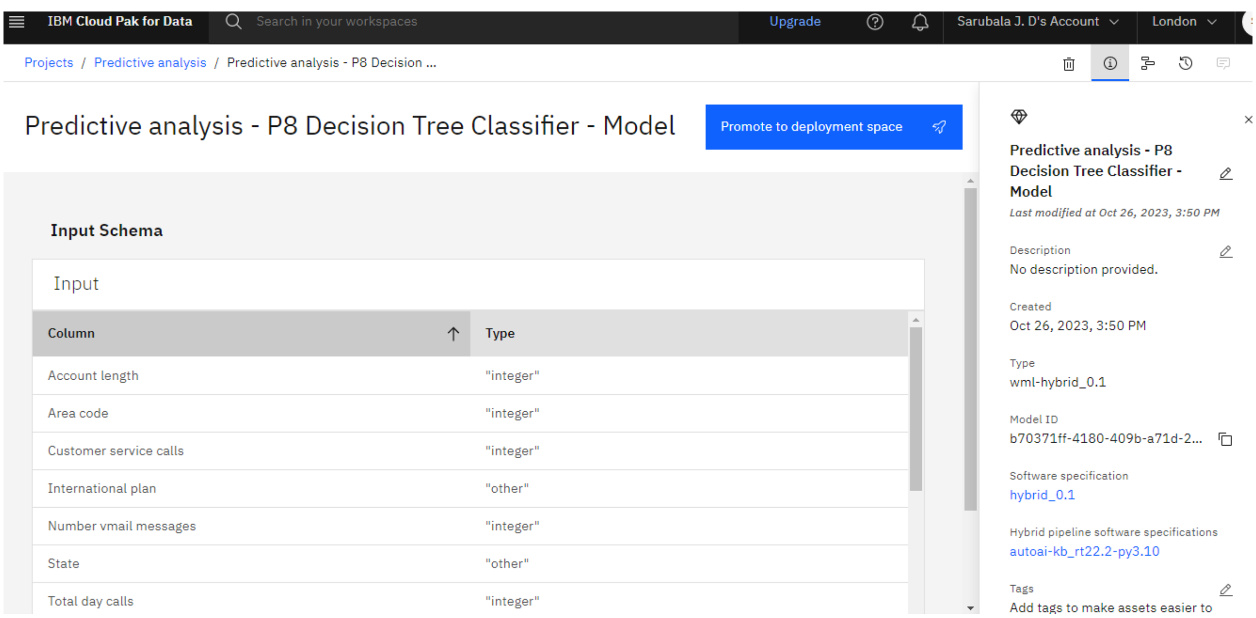


	Rank	Name	Algorithm	Accuracy (Optimized) Cross Validation	Enhancements	Build time
★	1	Pipeline 4	Snap Random Forest Classifier	0.957	HPO-1 FE HPO-2	00:00:48
	2	Pipeline 3	Snap Random Forest Classifier	0.957	HPO-1 FE	00:00:36
	3	Pipeline 8	Decision Tree Classifier	0.954	HPO-1 FE HPO-2	00:00:25
	4	Pipeline 7	Decision Tree Classifier	0.950	HPO-1 FE	00:00:20
	5	Pipeline 2	Snap Random Forest Classifier	0.937	HPO-1	00:00:05
	6	Pipeline 1	Snap Random Forest Classifier	0.926	*None*	00:00:01

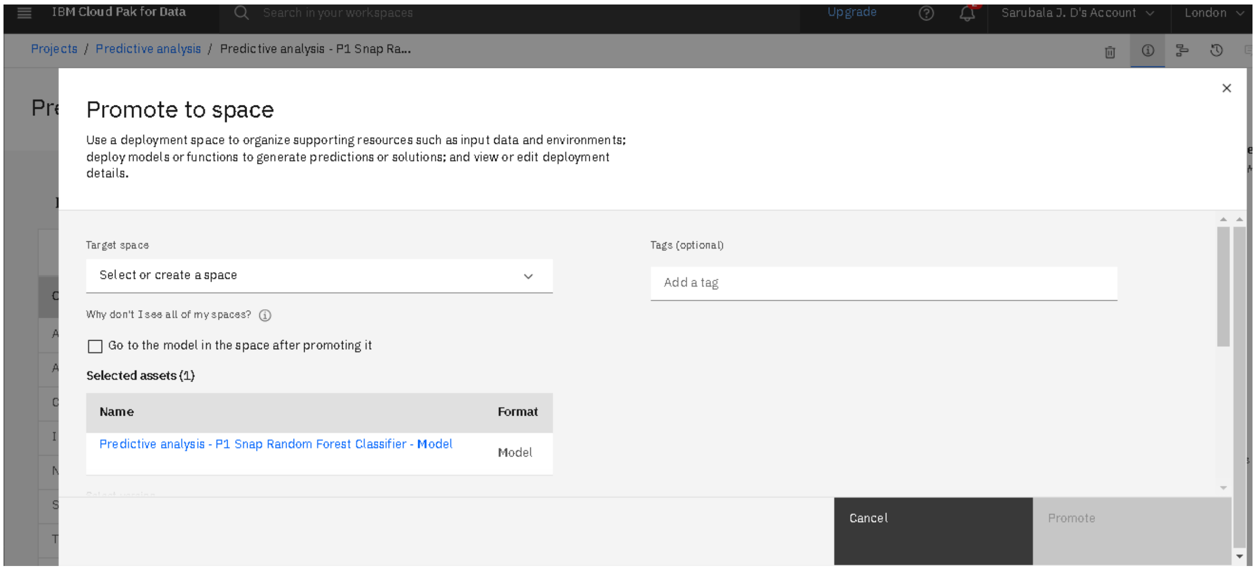
STEP 4: Model Creation

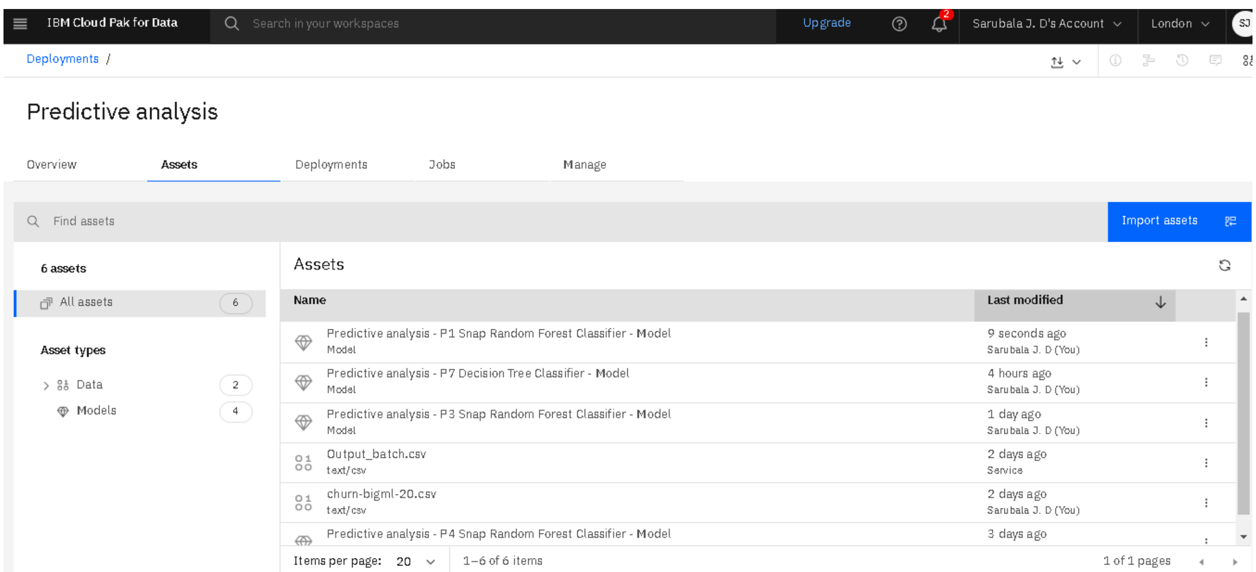


STEP 5: Deployment Promoting

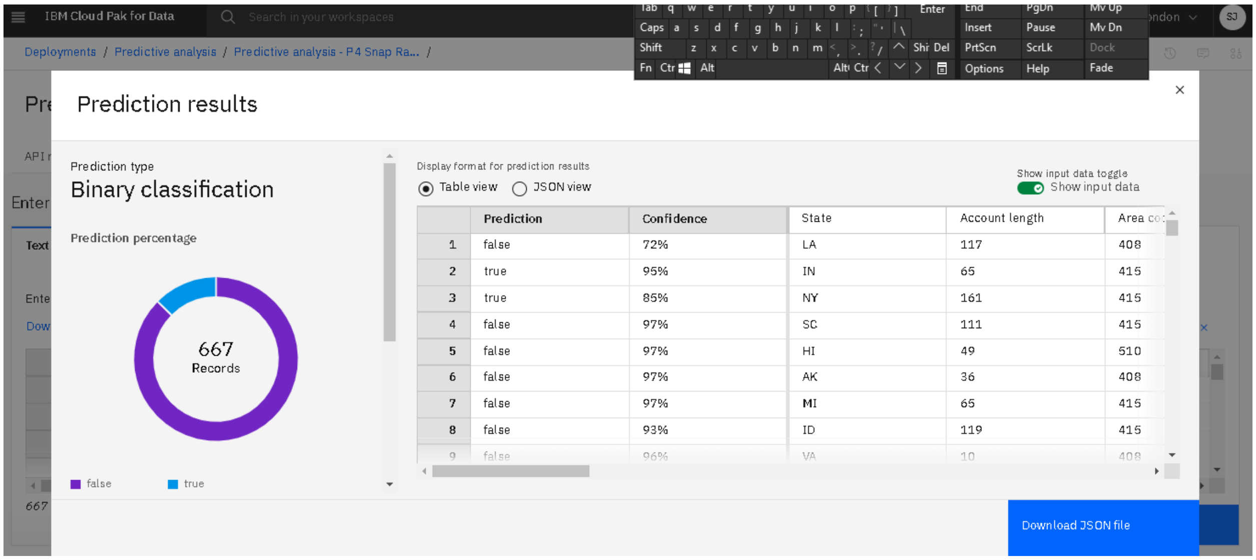


STEP 6: Deployment

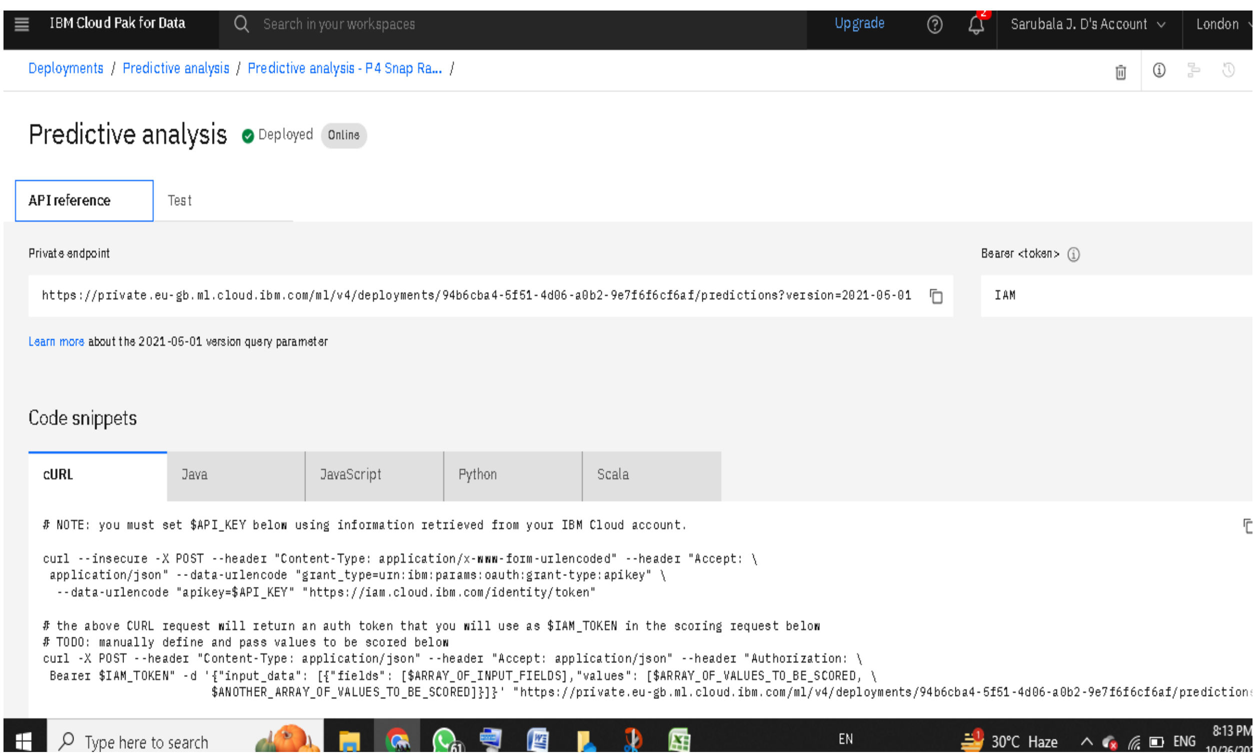




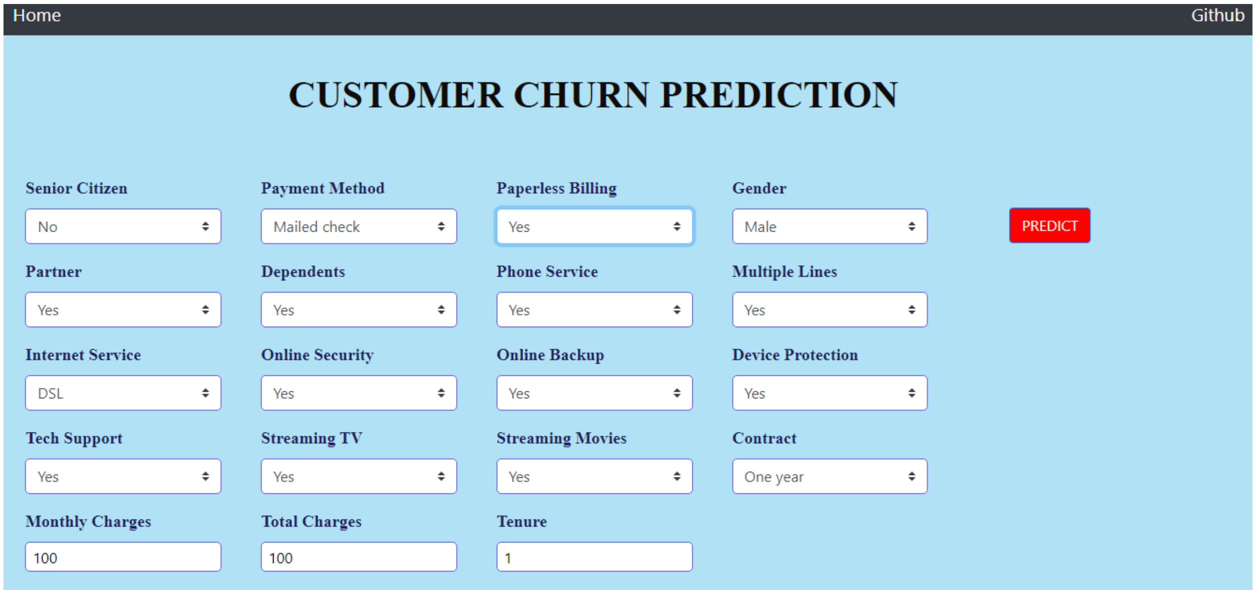
STEP 7: Testing

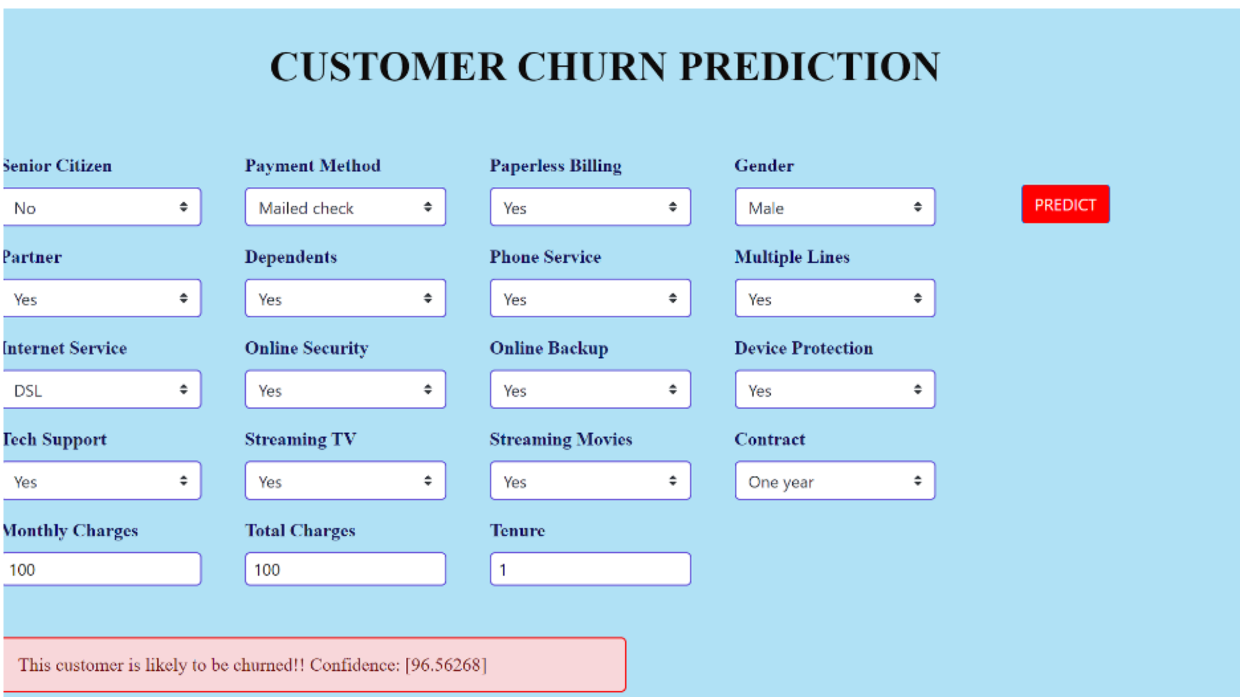


STEP 7: API reference



STEP 8: WEB Application





Access and Utilization:

Users can access the deployed model by making HTTP requests to the API, providing the necessary input data. The API will return the predicted sales values in real-time. Businesses can integrate this API into their sales management systems or use it for decision-making processes, such as inventory management and marketing budget allocation based on real-time sales predictions.