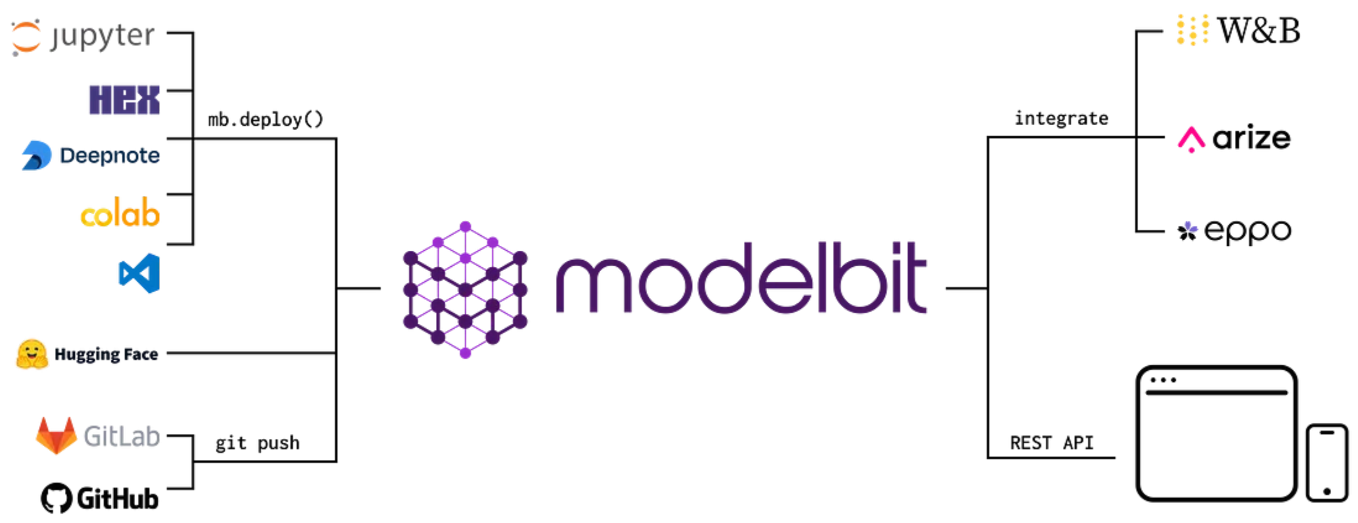
# Easily deploy machine learning models from the comfort of your Notebook

Moez Ali · Follow
9 min read · Sep 8, 2023

Streamline your Machine Learning model deployment with Modelbit.com. Seamlessly move from Jupyter notebooks to production-ready REST APIs, supported by fully customizable Python environments integrated with your git repository.



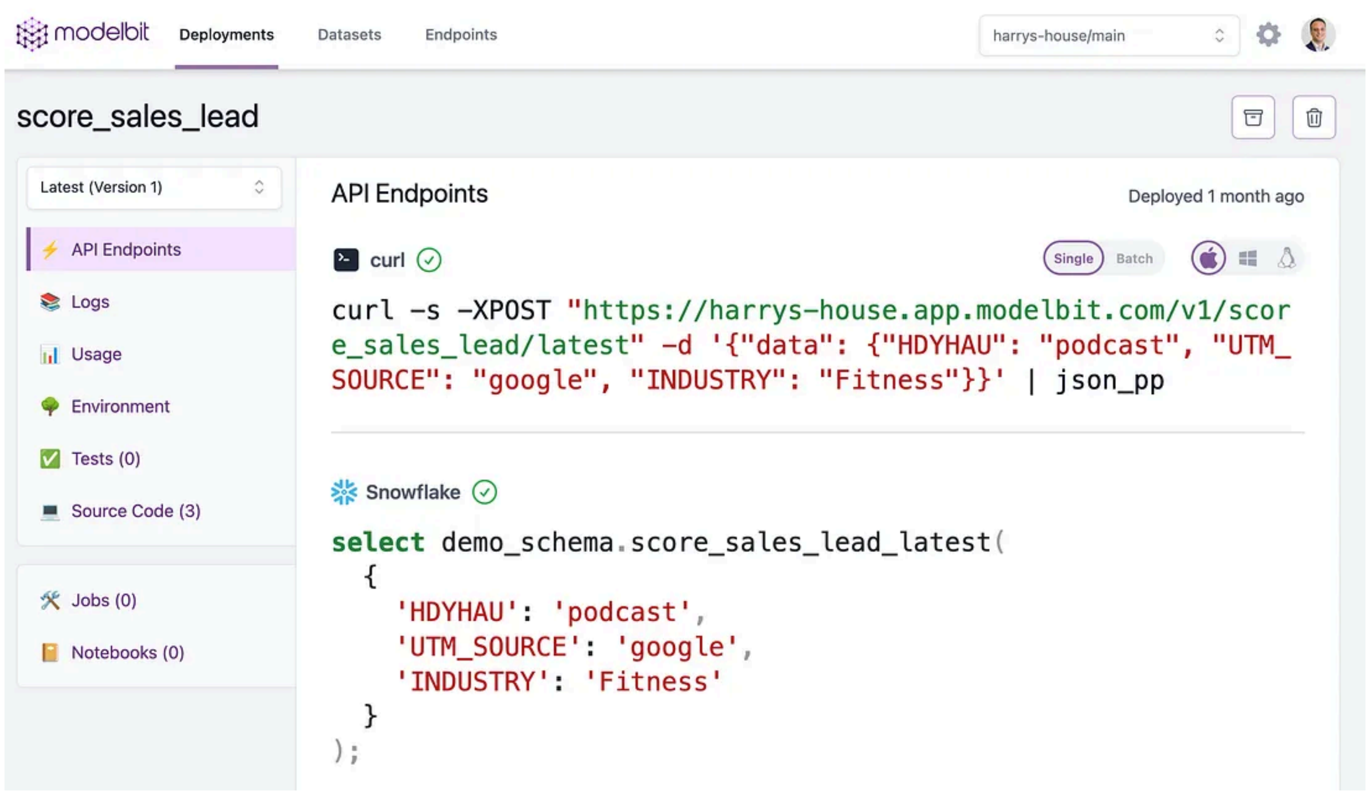
Modelbit Introduction — Image Source

## 📌 Introduction

Model deployment is a crucial step in the machine learning pipeline that involves making a trained model available for use in a production environment. Model deployment can be a complex and time-consuming

process of model deployment and make it accessible to data scientists without extensive software engineering experience.

One such tool that we will discuss in this tutorial is Modelbit, a platform that enables data scientists to deploy machine learning models quickly and easily. With Modelbit, data scientists can deploy models from Jupyter notebooks with just one line of code, without the need for special frameworks or code rewrites.



```
curl -s -XPOST "https://harrys-house.app.modelbit.com/v1/scor
e_sales_lead/latest" -d '{"data": {"HDYHAU": "podcast", "UTM_
SOURCE": "google", "INDUSTRY": "Fitness"}}' | json_pp
```

```
select demo_schema.score_sales_lead_latest(
  {
    'HDYHAU': 'podcast',
    'UTM_SOURCE': 'google',
    'INDUSTRY': 'Fitness'
  }
);
```

API endpoints shown in the UI of Modelbit — Image Source

## 🚀 Deploying ML Models: Beyond the Basics and Into the Challenges

Taking a machine learning model from training to real-world use isn't just about plugging it into an app. It's about making sure the model works well with new data, can handle lots of users at once, and gives fair and clear results.

This process has its hurdles. As data changes over time, the model might not perform as well. It's also crucial to ensure the model can quickly provide answers, especially when many people are using it.

All in all, deploying a model is more than just the initial setup; it's about constant fine-tuning and vigilance.

### 1. Environment Inconsistencies

One of the challenges of model deployment is ensuring that the deployed model is consistent with the model that was trained in the development environment. This consistency can be difficult to achieve, especially when deploying models to different environments or platforms.

docker

One solution to this problem is to use Docker containers. Docker is a platform that allows developers to package applications and their dependencies into containers, which can be run on any platform that supports Docker.

One of the main concerns when deploying machine learning models is the discrepancy between development and production environments. Libraries, dependencies, or even minor version differences can have a profound impact on model behavior.

By packaging the application along with its entire runtime environment, Docker ensures that the model sees the same environment, whether it's being tested on a developer's local machine or running in a high-throughput production setting.

This eliminates the notorious "it works on my machine" problem, ensuring that models behave consistently across different stages of their lifecycle.

> *If you want to learn more about Docker containers, check out my latest blog Deploy Machine Learning Pipeline on the cloud using Docker Container.*

🛠️ **How does Modelbit help?**

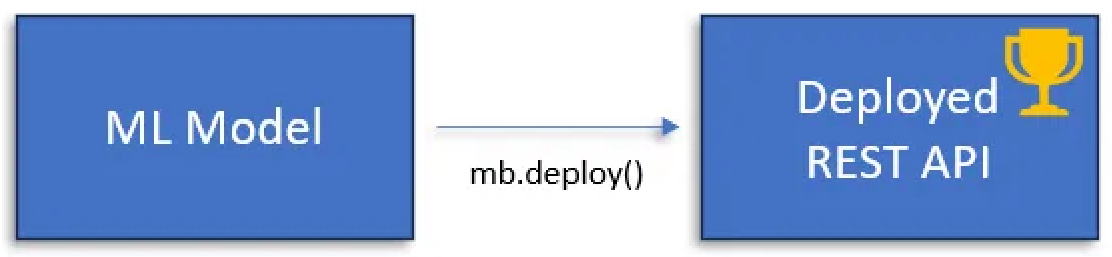


Image by Author

Modelbit automatically creates a fully customizable environment and hosts it as REST API on the cloud with just one line of code in your Jupyter Notebook.

**2. Lengthy, complex processes for creating and deploying REST API**

Deploying a trained ML model involves several steps:

- Persist the model file (.pkl, .h5, .mod, etc.)
- Create a Dockerfile, requirements.txt, and build other required files
- Build a docker container
- Create a REST API using framework like Flask, FastAPI, Django etc.
- Deploy the API on a cloud service like Azure Web Services, AWS Fargate, Google Vertex, etc.

> Modelbit simplifies the whole experience of deploying models as REST API's. With just one line of code you can have a working REST API up and running within a minute.

**3. CI/CD Processes for ML Pipeline**

Continuous Integration/Continuous Delivery (CI/CD) are a set of practices that automate the building, testing, and deployment of software. However, implementing CI/CD for machine learning presents unique challenges, as ML models depend not only on the code but also on the data and model.

> Modelbit provides an easy-to-use, comprehensive CI/CD stack with features like unit tests, model registry, and an integration with Git repo.

#### 4. API Monitoring and Reliability

API monitoring stands as a cornerstone of operational excellence. As ML models serve predictions through APIs to various applications, it becomes imperative to continuously monitor these endpoints for availability and latency.

> Modelbit has a built-in Usage Monitoring built-in to monitor the performance and latency of your API endpoints.

## 🐉 📊 🚀 End-to-end Python example of deploying a Loan Underwriting Machine Learning pipeline with Modelbit

### 🤔 Problem Statement

Financial institutions are increasingly using machine learning to enhance their loan decision-making. This shift promises more efficient and informed lending practices.

#### What is Loan Underwriting?

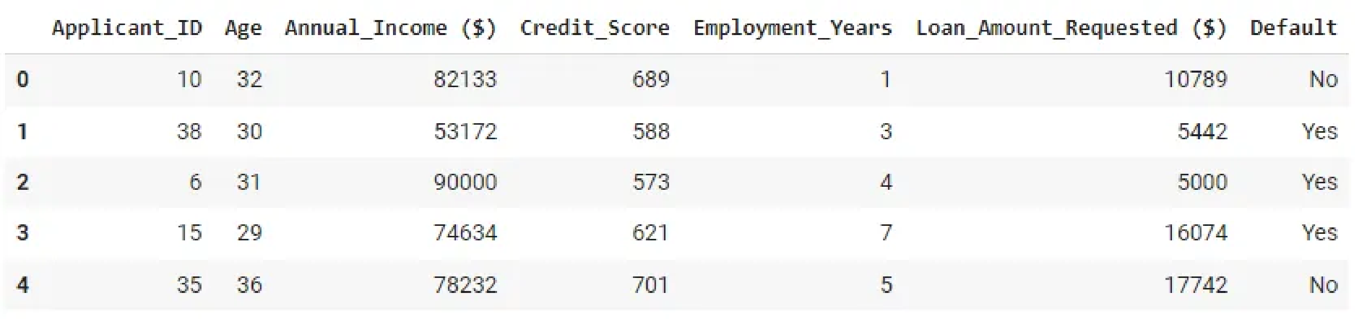
Loan underwriting is the method banks use to determine if they should give a loan, how much, and at what rate. It involves checking a person's credit history, job details, and financial records to decide on the loan's terms.

#### Why Use Machine Learning for Loan Underwriting?

1. **Fast and Accurate Analysis:** Machine learning can quickly sift through vast data, identifying trends and patterns that might be hard for humans to see, leading to better loan decisions.
2. **Continuous Learning:** Traditional loan methods stay the same unless changed, but machine learning models improve as they get more data.
3. **Fair Decision Making:** Using data-driven methods helps reduce biases that might affect human decisions. However, it's essential to train models correctly to ensure fairness.
4. **Cost Efficiency:** By automating the loan process with machine learning, banks can save money and offer better loan rates.
5. **Improved Risk Prediction:** Machine learning helps lenders predict more accurately who might struggle to repay, ensuring safer lending.

### 📈 Dataset

I have created the synthetic dataset for this demo. You can download/read the dataset from here.



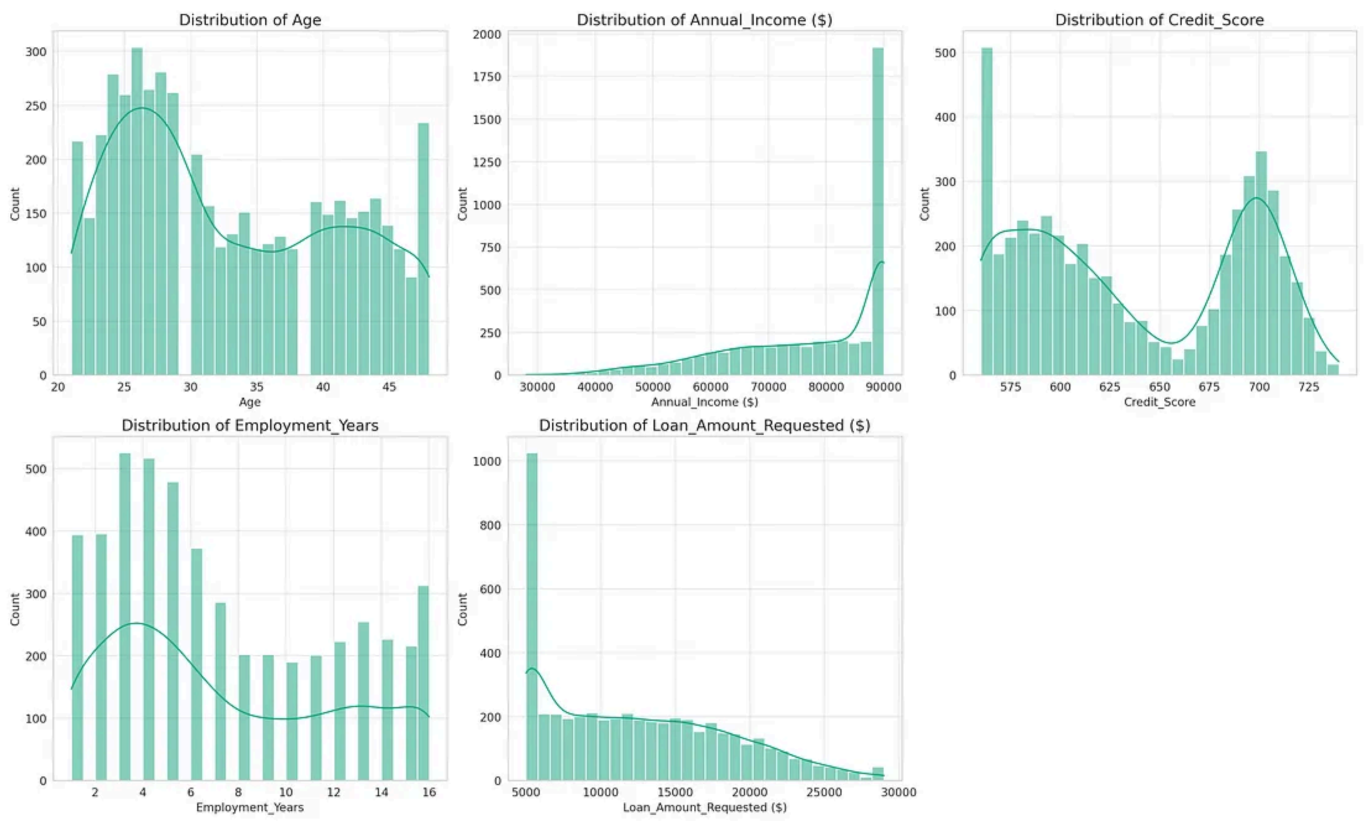
| | Applicant_ID | Age | Annual_Income ($) | Credit_Score | Employment_Years | Loan_Amount_Requested ($) | Default |
|---|---|---|---|---|---|---|---|
| **0** | 10 | 32 | 82133 | 689 | 1 | 10789 | No |
| **1** | 38 | 30 | 53172 | 588 | 3 | 5442 | Yes |
| **2** | 6 | 31 | 90000 | 573 | 4 | 5000 | Yes |
| **3** | 15 | 29 | 74634 | 621 | 7 | 16074 | Yes |
| **4** | 35 | 36 | 78232 | 701 | 5 | 17742 | No |

Synthetic dataset created by author using Gretel.AI

- **Applicant_ID** (Integer): Unique identifier for each applicant.
- Age (Integer): Age of the applicant.
- **Annual_Income** ($) (Float): Applicant's yearly income in dollars.
- **Credit_Score** (Integer): Credit score of the applicant.
- **Employment_Years** (Integer): Number of years the applicant has been employed.
- **Loan_Amount_Requested** ($) (Float): The amount in dollars the applicant has requested for the loan.
- **Default** (String: 'Yes' or 'No'): Indicates if the applicant defaulted on their loan.

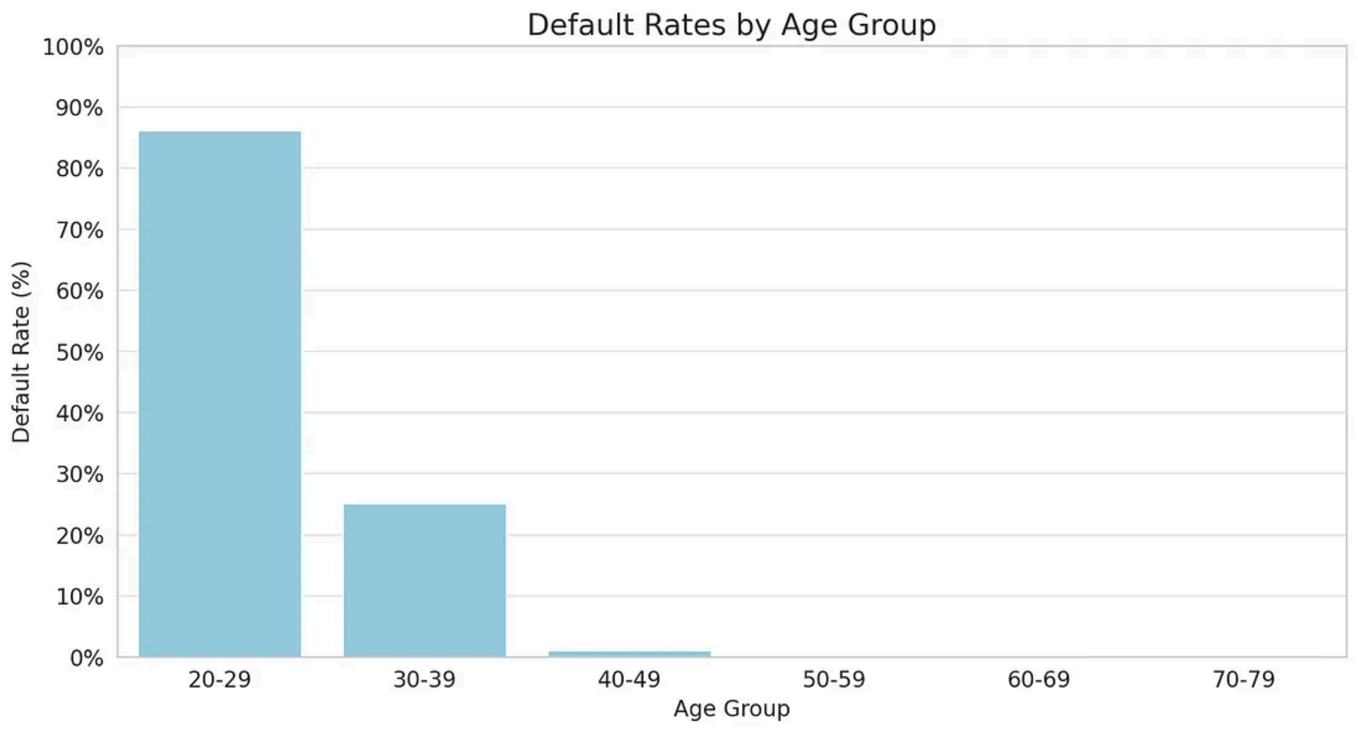
## 🔍 📊 Exploratory Data Analysis

> *Distribution of Features*



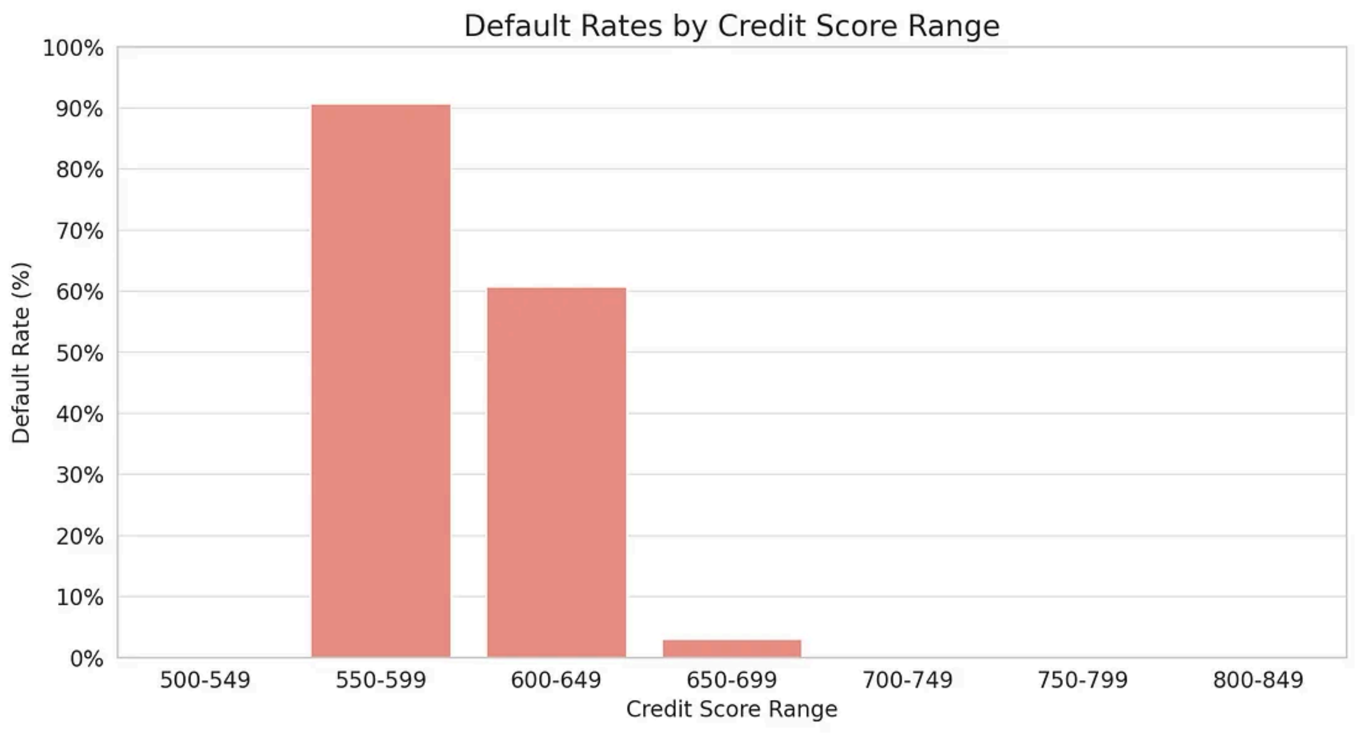
Statistical Distribution of Features

> *Default rates by Age Group*



Default Rates by Age Group

> *Default rates by Credit Score*



Default Rates by Credit Score

Observe the significant default rate when the credit score falls below 600 or when the individual is under 30 years old. The exploratory data analysis reveals significant variances in the features, which should pave the way for an effective model.

## 🤖 Model Training and Selection

The focus of this tutorial is not model training and selection, but we do need a trained pipeline to demonstrate the deployment. Let's start by building a simple sklearn pipeline consisting of some data preprocessing and a Logistic Regression model to predict loan default.

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.model_selection import train_test_split

# Features and target variable
X = data.drop(columns=['Default', 'Applicant_ID'])
y = data['Default']
```

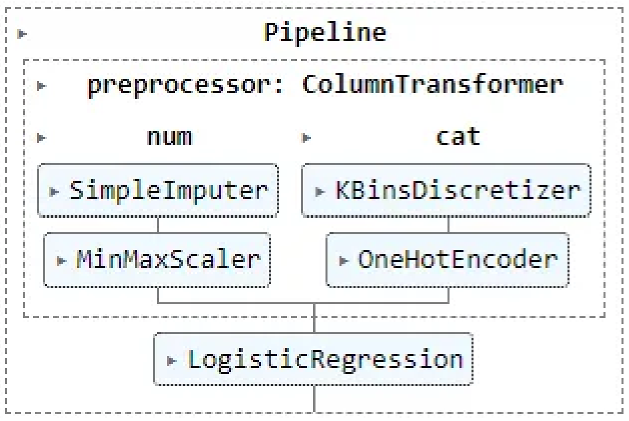
```
# Columns to be scaled
numeric_features = ['Annual_Income', 'Credit_Score', 'Employment_Years', 'Loan_A

# Column to be binned and one-hot encoded
categorical_features = ['Age']

# Create transformers
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', MinMaxScaler())
])
categorical_transformer = Pipeline(steps=[
    ('bin', KBinsDiscretizer(n_bins=6, encode='ordinal', strategy='quantile')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])

# Combine all transformers into a preprocessor using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Create and evaluate the pipeline
pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                           ('classifier', LogisticRegression())])
pipeline
```



sklearn pipeline

Now let’s fit the pipeline:

```
from sklearn.model_selection import train_test_split

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# print train and test set shape
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)

# generate predictions
y_pred = pipeline.predict(X_test)
y_pred_prob = pipeline.predict_proba(X_test)[:, 1]

# test accuracy on X_test
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

**Output:**

```
>>> X_train: (3500, 5)
>>> X_test: (1500, 5)
>>> y_train: (3500,)
>>> y_test: (1500,)

>>> 0.9606666666666667
```

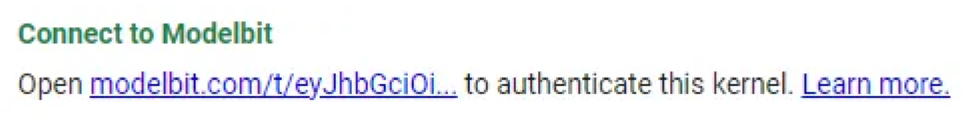
### Model Deployment

We will use Modelbit to deploy this pipeline as a REST API with just one line of code. If you haven't used Modelbit before you can start a free trial here.

You will also have to install the library using pip.

```
# install modelbit
pip install modelbit

# run on top of your notebook
import modelbit
mb = modelbit.login()
```

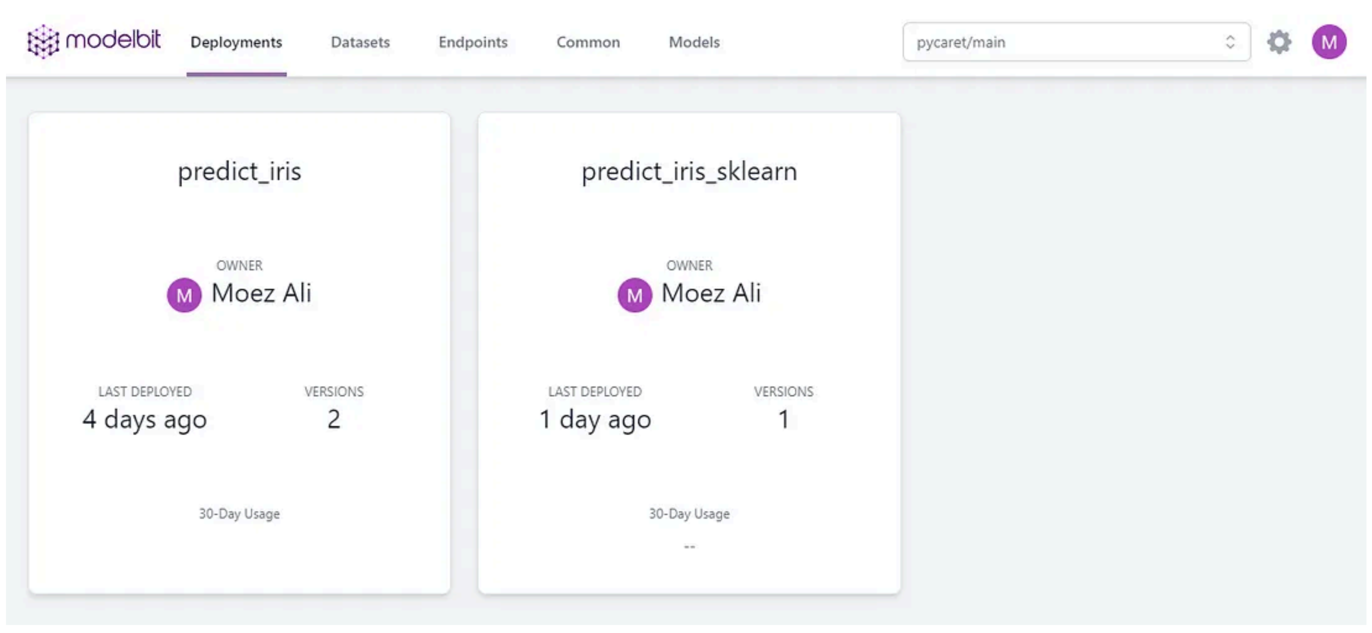


Connect to Modelbit

Open modelbit.com/t/eyJhbGciOi... to authenticate this kernel. Learn more.

Output of modelbit.login()

Click on the hyperlink generated by the code above, and it will direct you to the Deployments page.



predict_iris

OWNER
M Moez Ali

| LAST DEPLOYED | VERSIONS |
|---|---|
| 4 days ago | 2 |

30-Day Usage

predict_iris_sklearn

OWNER
M Moez Ali

| LAST DEPLOYED | VERSIONS |
|---|---|
| 1 day ago | 1 |

30-Day Usage
--

Deployments page on Modelbit.com

### Creating an Inference function

The first step is to create a Python function for inference which uses the `predict` or `predict_proba` method from sklearn.

```
import pandas as pd
import numpy as np

# first define function
def predict_loan_default(Age: int, Annual_Income: float, Credit_Score: int, Empl
  """
  Predict the probability of loan default using a pre-trained machine learning p

  Args:
      Age (int): Applicant's age.
      Annual_Income (float): Applicant's annual income.
      Credit_Score (int): Applicant's credit score.
      Employment_Years (float): Number of years employed.
      Loan_Amount_Requested (float): Requested loan amount.

  Returns:
      float: Probability of loan default.
  """

  return pipeline.predict_proba(pd.DataFrame([[Age, Annual_Income, Credit_Score,
                                             columns = ['Age', 'Annual_Income',
```

Now let's the function:

```
predict_loan_default(32, 821233, 689, 1, 10789)
```

## Output:

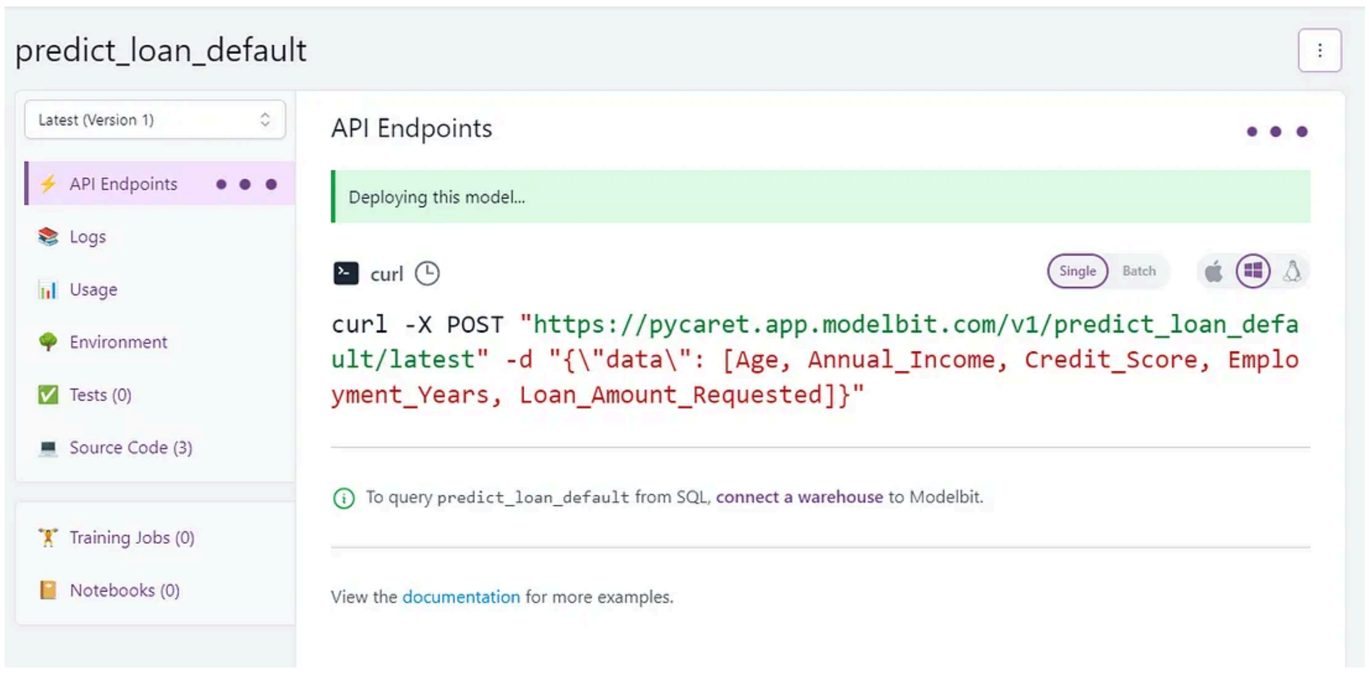
```
>>> array([4.28870351e-27])
```

✅ Success

Now to deploy this pipeline just run:

```
# deploy predict_loan_default
mb.deploy(predict_loan_default)
```

It will take few minutes for API Endpoint to be ready for use.



Deploying this model...

```
curl -X POST "https://pycaret.app.modelbit.com/v1/predict_loan_default/latest" -d "{\"data\": [Age, Annual_Income, Credit_Score, Employment_Years, Loan_Amount_Requested]}"
```

To query `predict_loan_default` from SQL, connect a warehouse to Modelbit.

View the documentation for more examples.

Once API Endpoint is available you can use `curl` or `requests` library in Python.

```
!curl -s -XPOST "https://pycaret.app.modelbit.com/v1/predict_loan_default/latest
```

**Output:**

```
{
   "data" : [
      4.28870351436048e-27
   ]
}
```

or you can also use the `requests` library:

```
import requests
import json

url = "https://pycaret.app.modelbit.com/v1/predict_loan_default/latest"
headers = {
    'Content-Type': 'application/json'
}
data = {
    "data": [30, 53172, 588, 3, 5442]
}

response = requests.post(url, headers=headers, json=data)
response_json = response.json()

print(json.dumps(response_json, indent=4))
```

**Output:**

```
{
    "data": [
        0.9969505149674829
    ]
}
```

📌 You can check out the **Google Colab Notebook here.**

## 📜 Conclusion

In the rapidly evolving landscape of machine learning, the seamless transition from model development to deployment is paramount. While the complexities of model deployment have historically been a hurdle for many data scientists, tools like Modelbit have revolutionized the process.