DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

18AMC307L - CLOUD COMPUTING

TABLE OF CONTENTS

FACULTY-IN-CHARGE **HOD/AI**

EXP NO	Study on Hosted Hypervisor and Bare Metal Hypervisor	DATE
1		

Aim

To compare the performance, efficiency, and resource utilization of Hosted Hypervisors and Bare Metal Hypervisors based on key metrics such as CPU usage, memory overhead, disk I/O, and network latency.

Algorithm

Step 1: Initialization:

- Select a Hosted Hypervisor (e.g., VMware Workstation, VirtualBox).
- Select a Bare Metal Hypervisor (e.g., VMware ESXi, Microsoft Hyper-V).
- Choose identical hardware configurations for testing.

Step 2: Setup Environment:

- Install the Hosted Hypervisor on an existing operating system.
- Install the Bare Metal Hypervisor directly on the hardware.
- Deploy identical guest VMs with the same OS, CPU, and memory allocation.

Step 3: Performance Testing:

- CPU Benchmark: Run stress tests to evaluate processing efficiency.
- Memory Usage: Measure RAM utilization and overhead.
- Disk I/O: Perform read/write operations to analyze disk performance.
- Network Performance: Use network benchmarking tools to assess latency and throughput.

Step 4: Collect Data:

- Monitor CPU usage, memory consumption, and disk latency using system monitoring tools.
- Record the network throughput and response times.

Step 5: Analyze Results:

- Compare performance metrics between the two hypervisors.
- Identify efficiency trade-offs in different scenarios.

Result:

Bare Metal Hypervisors provide better performance with lower CPU, memory, and disk overhead due to direct hardware access. Hosted Hypervisors are more flexible but have higher resource consumption due to OS dependency.

		DATE
2	**Installation of Hosted Hypervisor**	

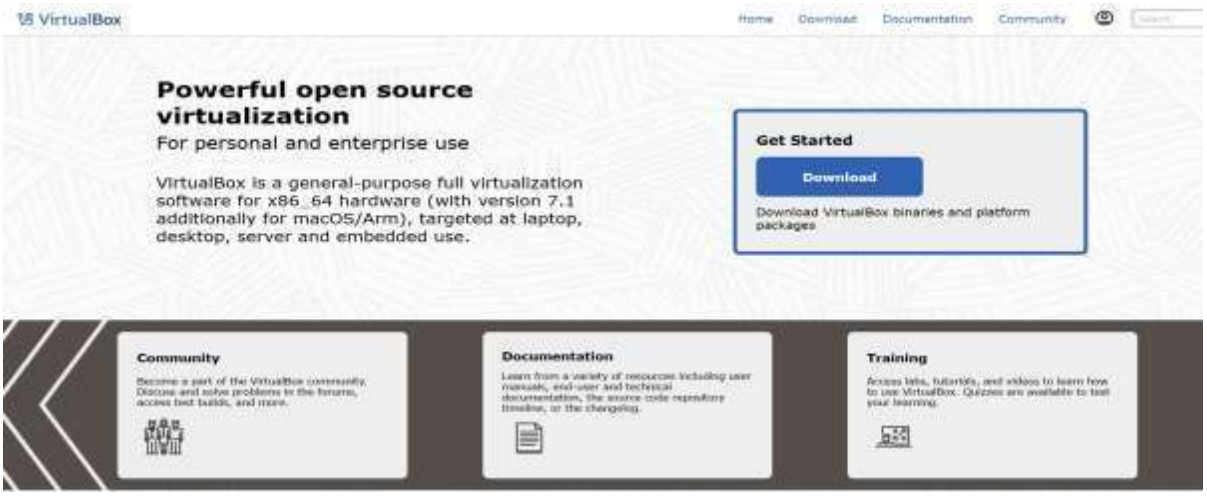
AIM:

To install and configure a Hosted Hypervisor (such as VMware Workstation or Oracle VirtualBox) on a Windows system and set up a virtual machine.

ALGORITHM:

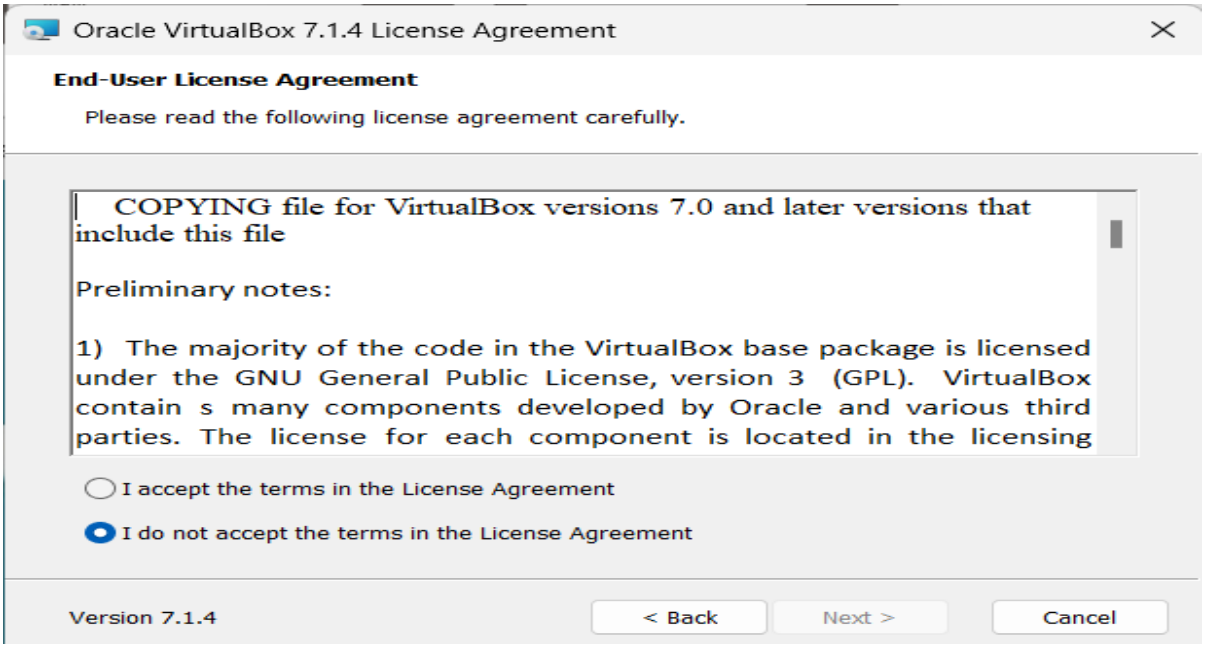
Step 1: Download the Hosted Hypervisor

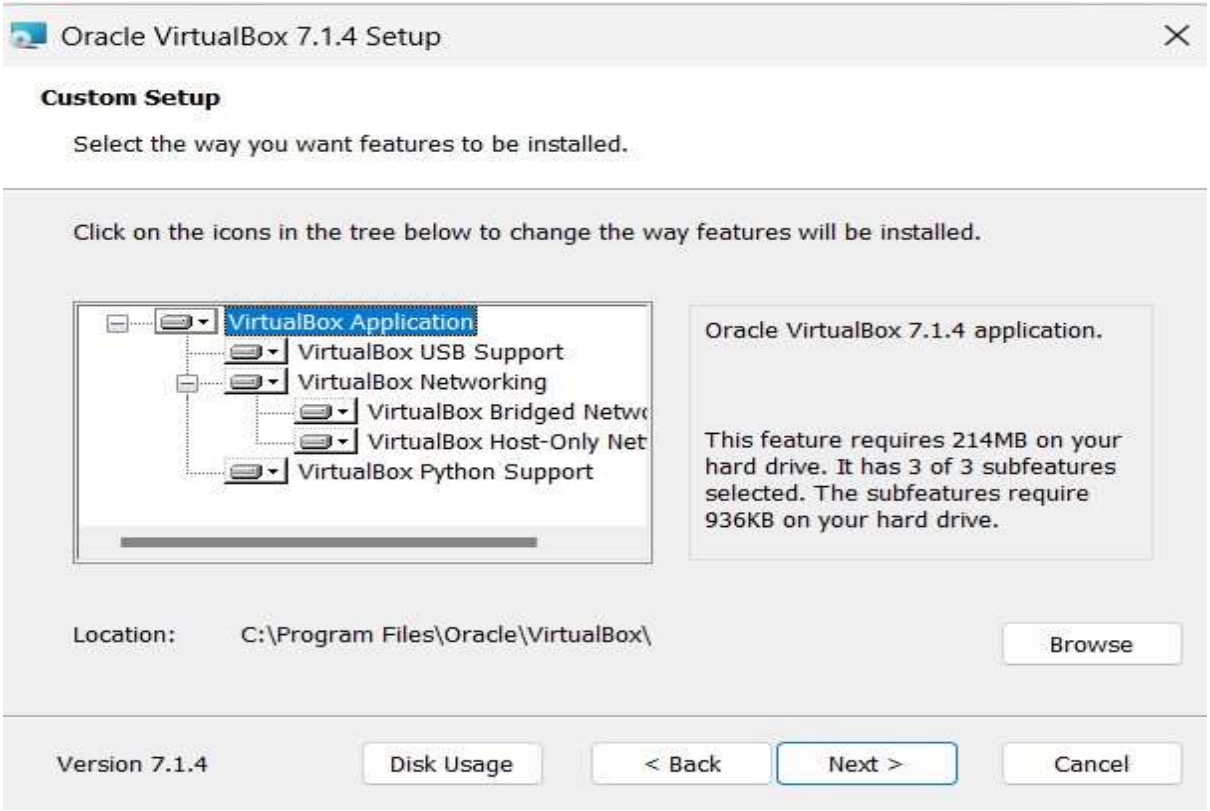
- Go to the official website of VMware Workstation or VirtualBox.
- Download the installer for Windows.



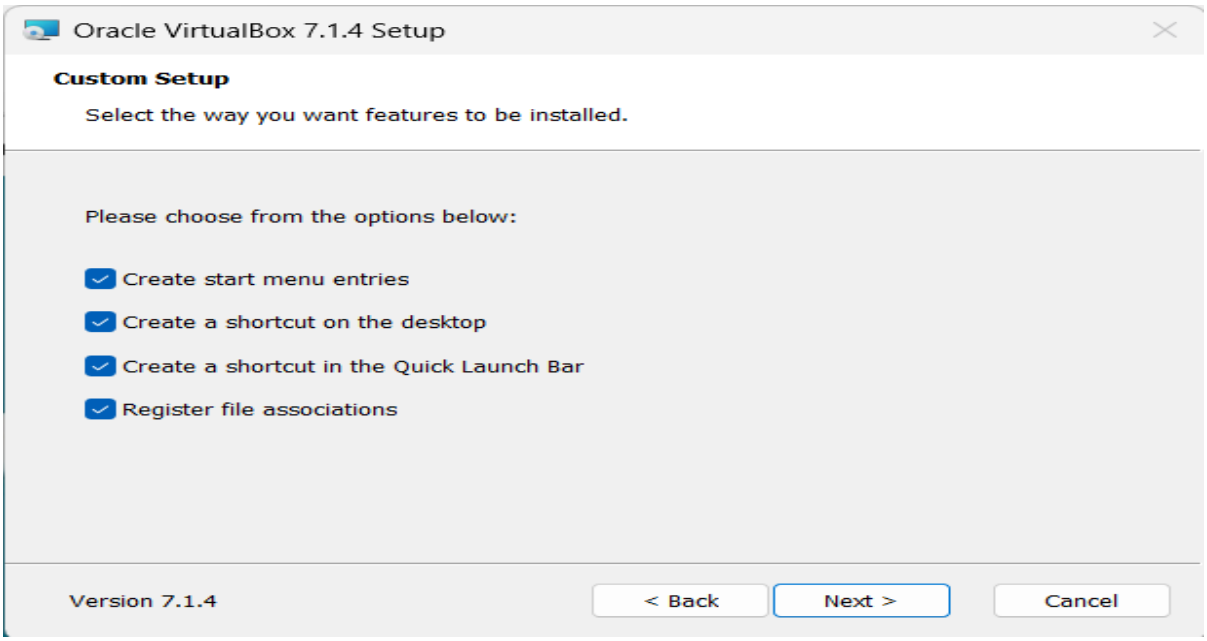
Step 2: Install the Hypervisor

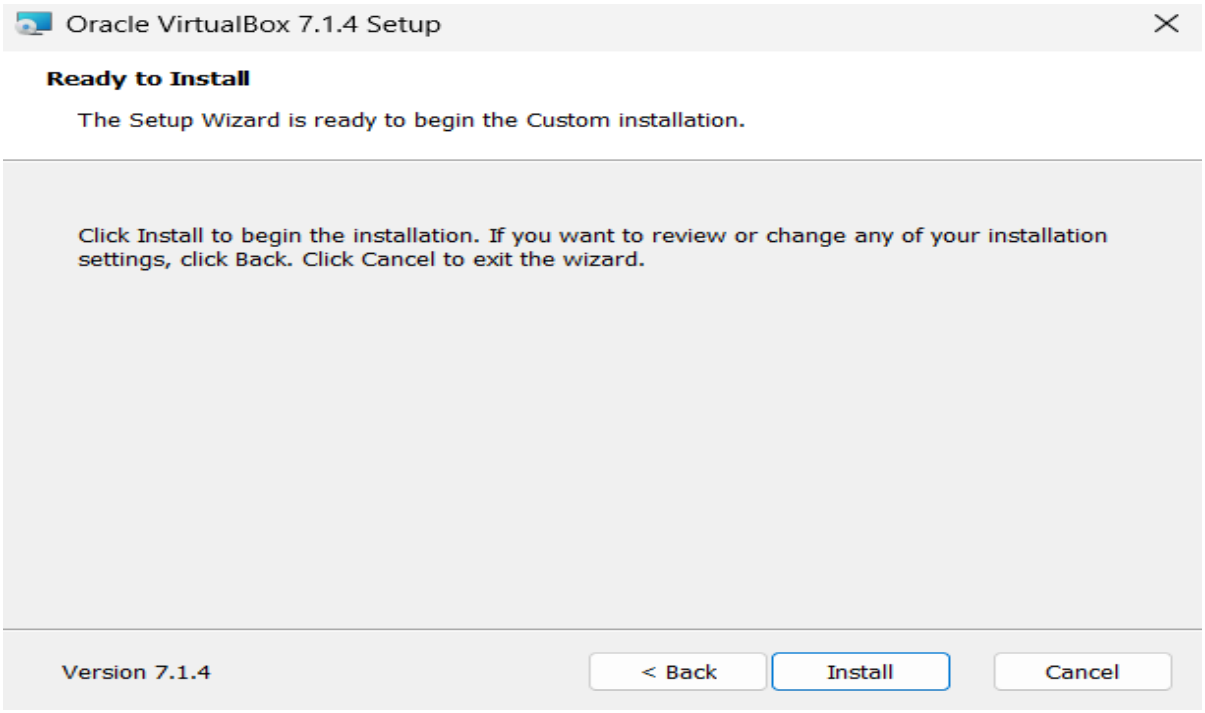
- Run the downloaded .exe file.
- Follow the setup wizard and accept the license agreement.





- Choose default settings and click Install.
- Restart the system if required.





Step 3: Verify the Installation

- Open VMware Workstation Player or VirtualBox.
- Check if the application runs successfully.

OUTPUT:

RESULT:

The Hosted Hypervisor was successfully installed and configured, enabling virtualization on the system.

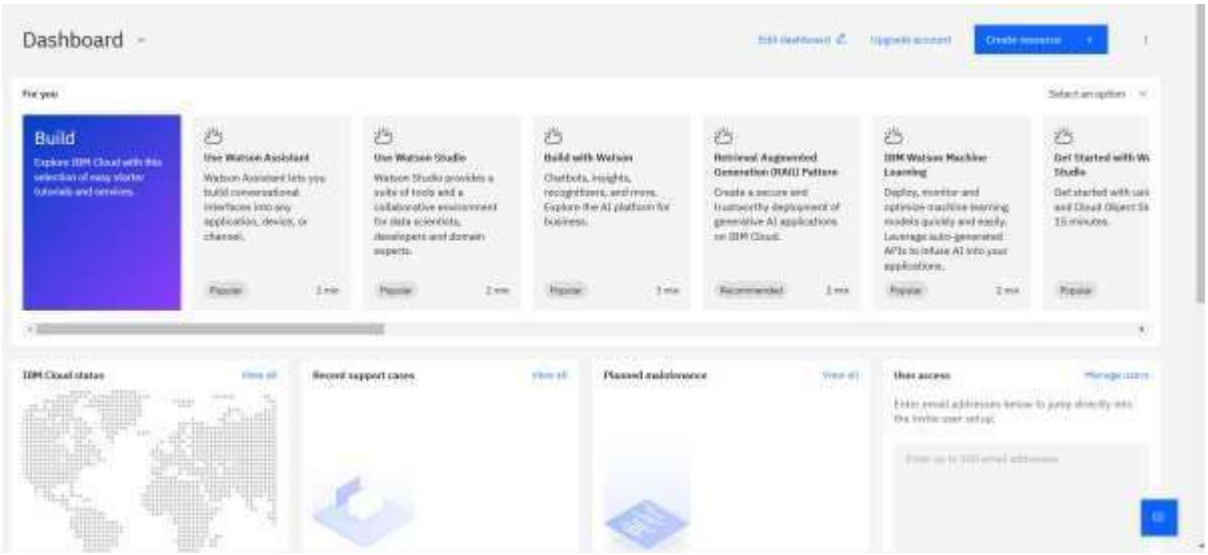
EXP NO	Installation of Bare Metal Hypervisor	DATE
3		

Aim:
To create an AI-powered chatbot using IBM Watson Assistant that can interact with users, process natural language inputs, and provide intelligent responses.

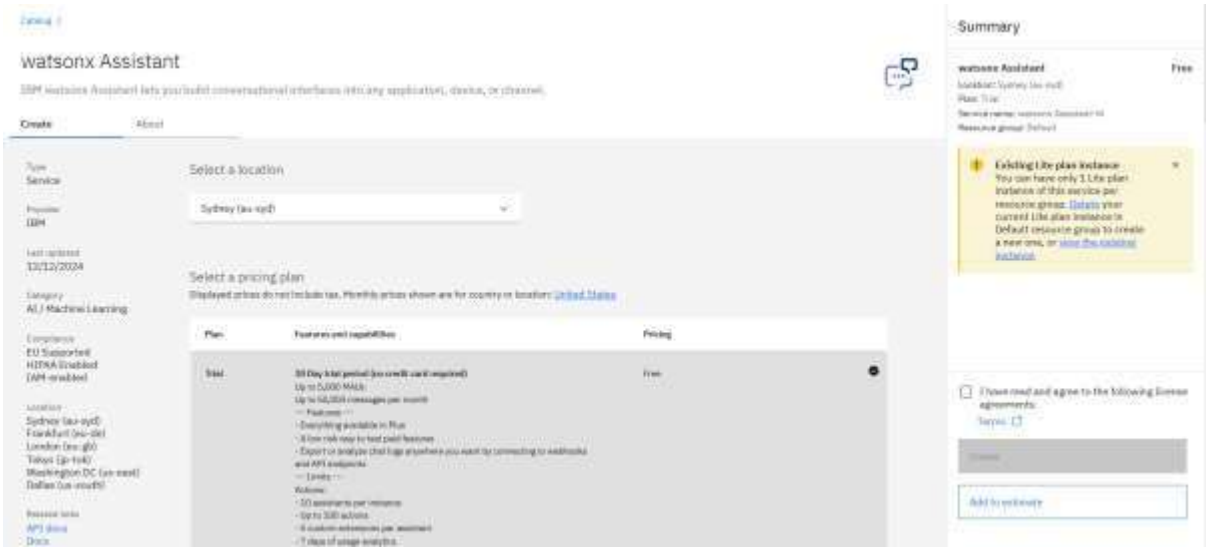
Algorithm Steps:

Step 1: Create an IBM Watson Assistant Instance

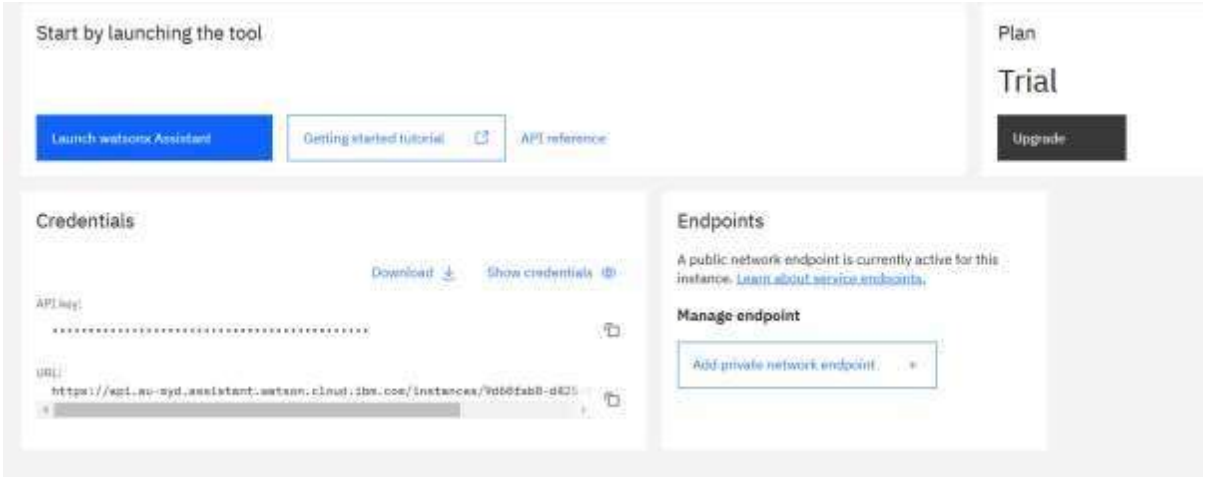
Sign up or log in to IBM Cloud.



- Navigate to "Watson Assistant" and create a new assistant instance.

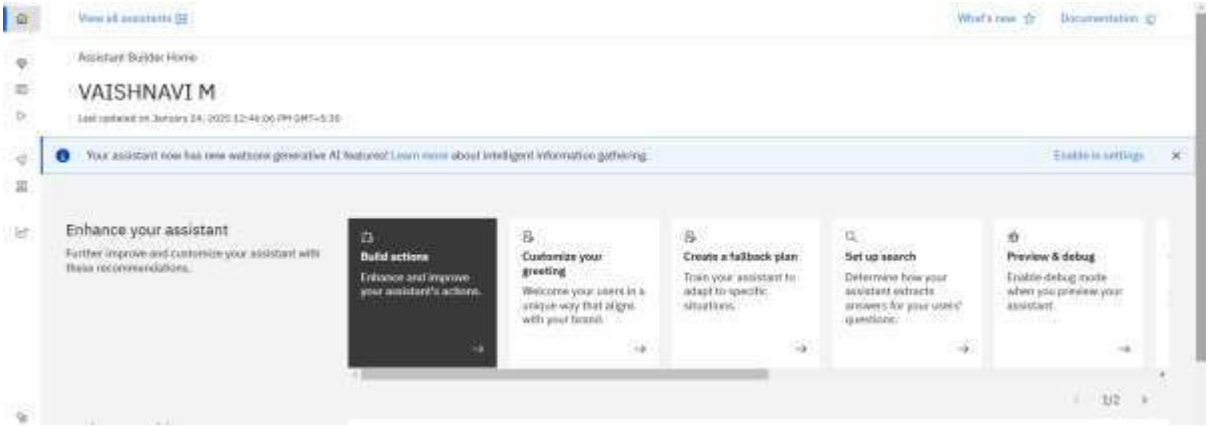


- Set up service credentials for authentication.



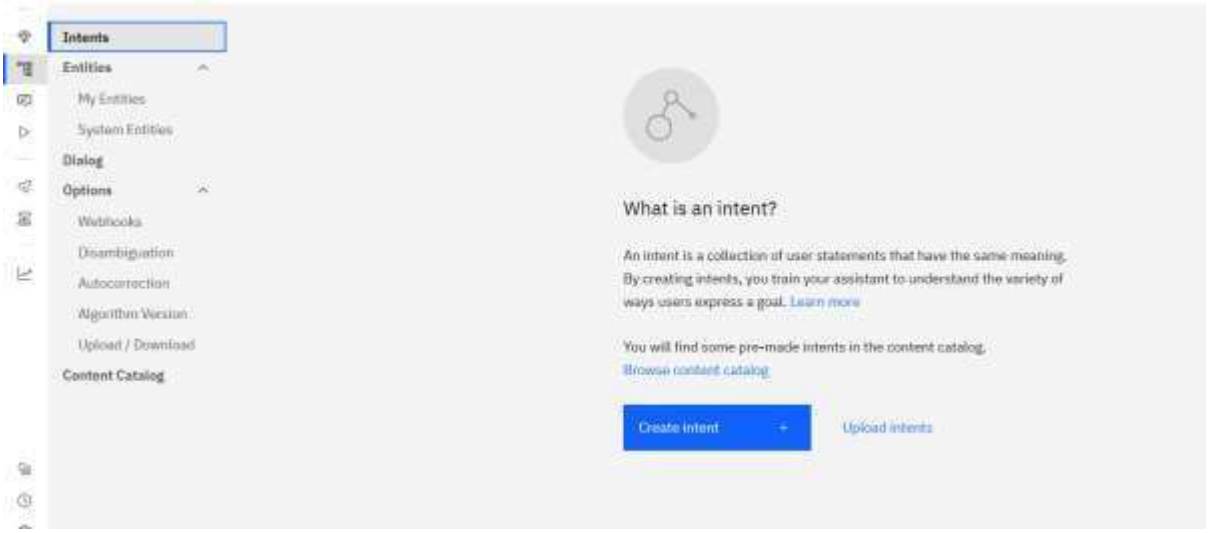
Step 2: Create a New Assistant

- Click on "Create Assistant" and give it a meaningful name.
- Define the purpose and functionality of the chatbot.



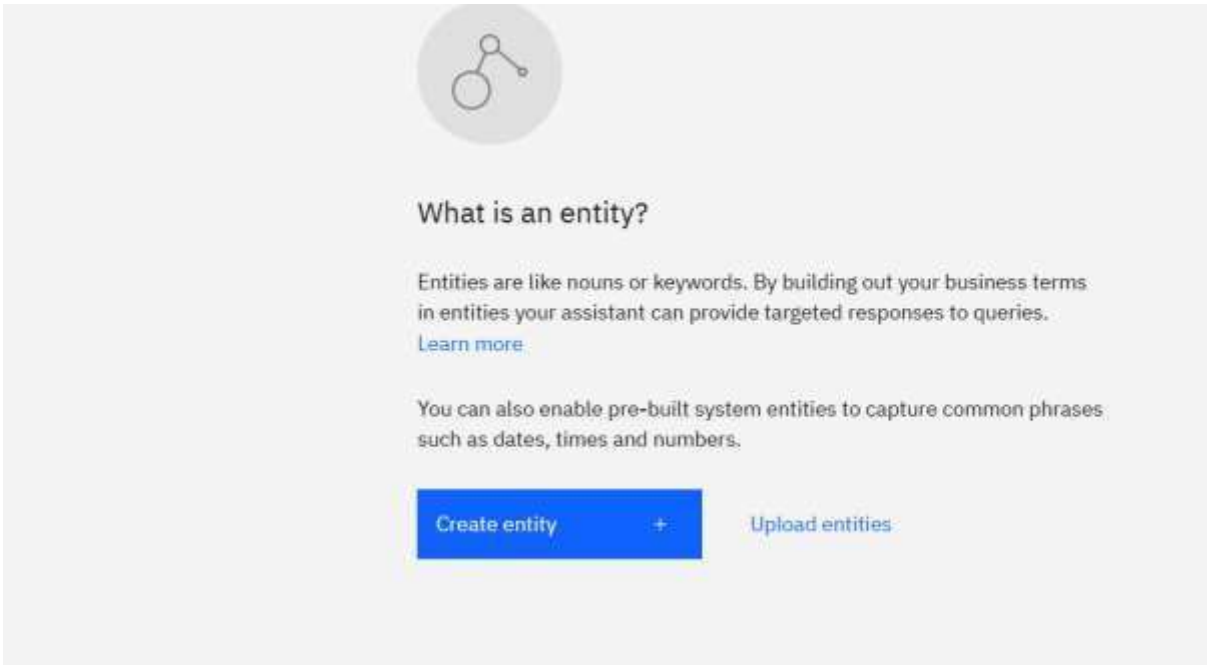
Step 3: Build Intents

- Intents represent what users want to accomplish (e.g., #greeting, #order_status).
- Add training phrases for each intent (e.g., "Hello", "Hi" for #greeting).



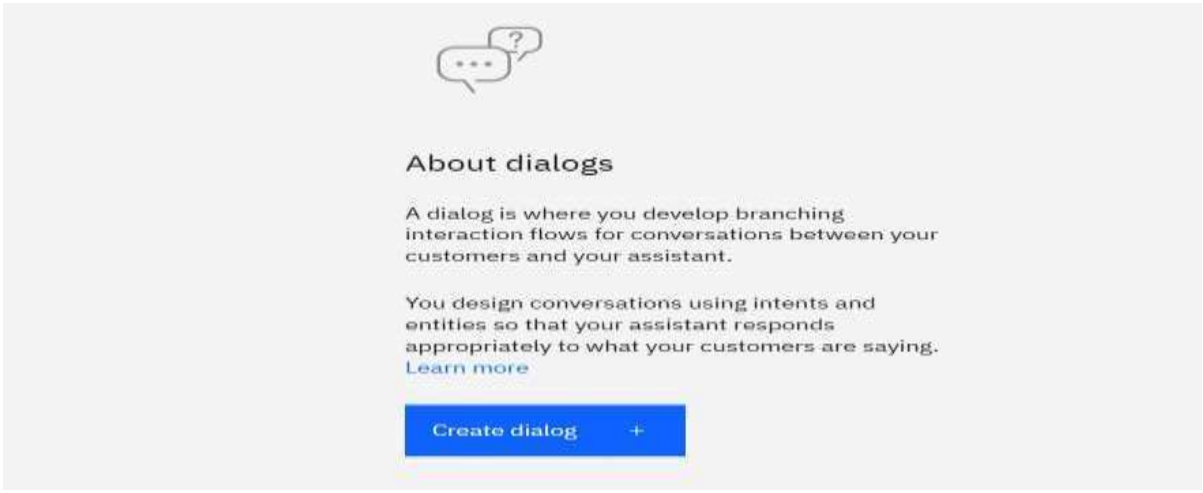
Step 4: Define Entities

- Entities help Watson understand specific keywords (e.g., @product_name, @location).
- Define possible entity values and synonyms.



Step 5: Create Dialog Flow

- Go to the "Dialog" section and create nodes for user interactions.
- Each node should check for specific intents and respond accordingly.
- Use conditions and logic to guide conversation flow.



Step 6: Train and Test the Assistant

- Click on "Train" to improve model accuracy.
- Use the "Try it" feature to test different user inputs and refine responses.

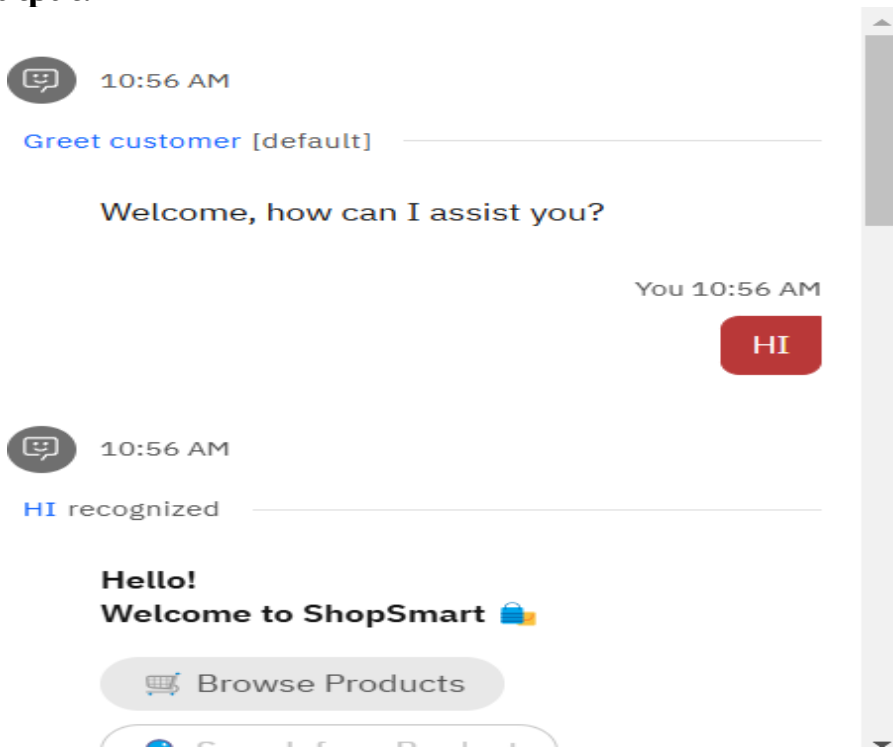
Step 7: Deploy the Assistant

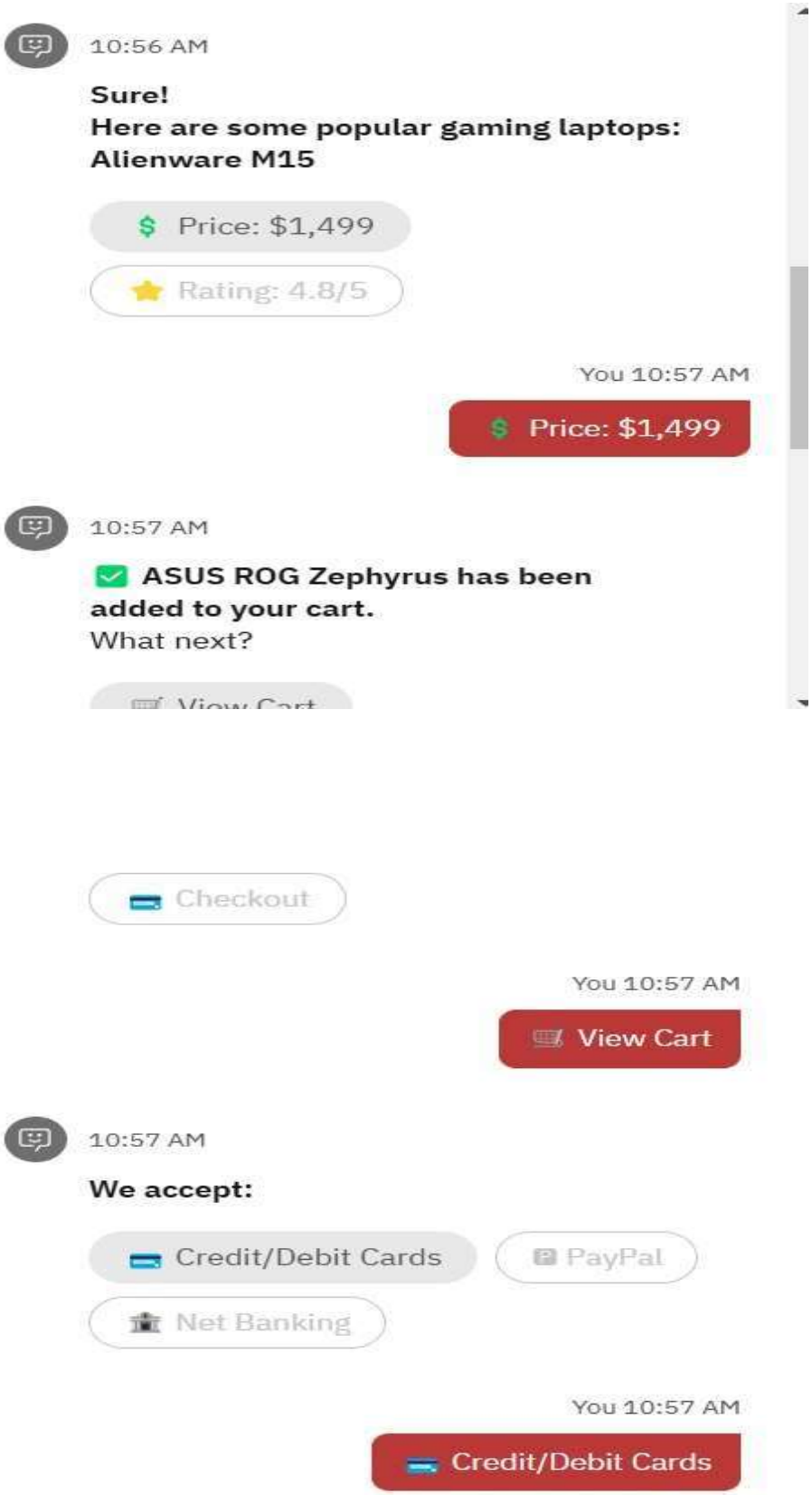
- Integrate Watson Assistant with a website, mobile app, or messaging platform.
- Obtain API credentials and connect via REST API.

Step 8: Monitor and Improve

- Use Watson's analytics to track user interactions.
- Update intents, entities, and dialog flow based on real-time feedback

Output:




10:56 AM
Sure!
Here are some popular gaming laptops:
Alienware M15
Price: $1,499
Rating: 4.8/5
You 10:57 AM
Price: $1,499
10:57 AM
ASUS ROG Zephyrus has been added to your cart.
What next?
View Cart
Checkout
You 10:57 AM
View Cart
10:57 AM
We accept:
Credit/Debit Cards
PayPal
Net Banking
You 10:57 AM
Credit/Debit Cards

Result:

By following the steps of the algorithm, we successfully created and deployed a chatbot using IBM Watson Assistant. The chatbot can:

- Understand user queries through predefined intents and entities
  Respond appropriately based on the dialog flow
  Handle multiple user interactions efficiently
  Be integrated into websites, mobile apps, or messaging platforms
  rovide real-time responses and improve with continuous training

EXP NO		DATE
4	**Implementation of Virtual Machine in Bare Metal Hypervisor**	

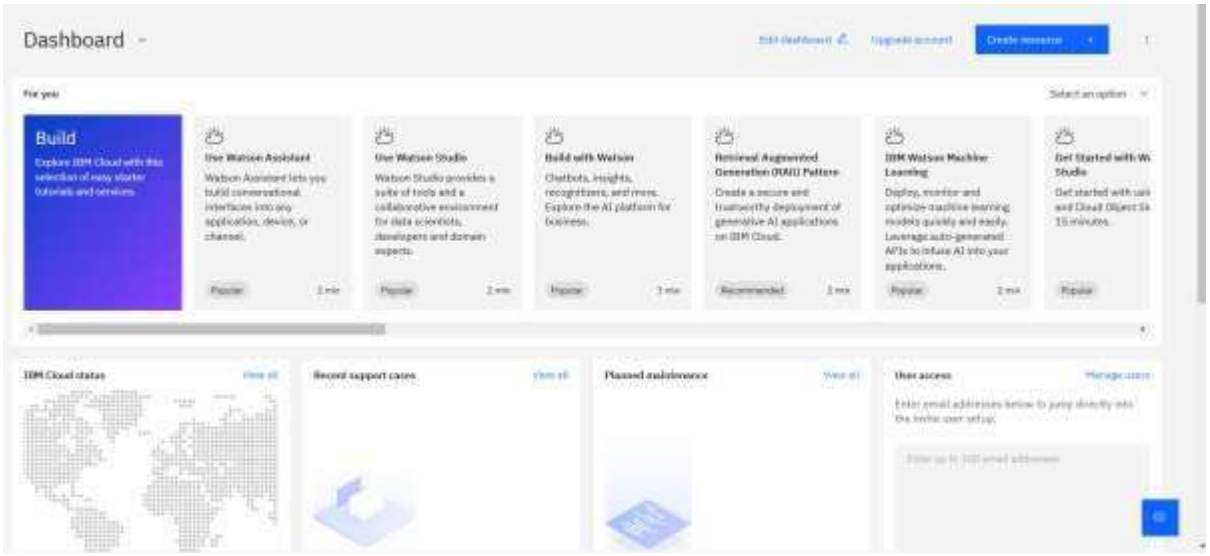
Aim:

To automate the process of data preprocessing, feature engineering, model selection, and hyperparameter tuning using IBM Watson Auto AI for building and deploying an optimized machine learning model.

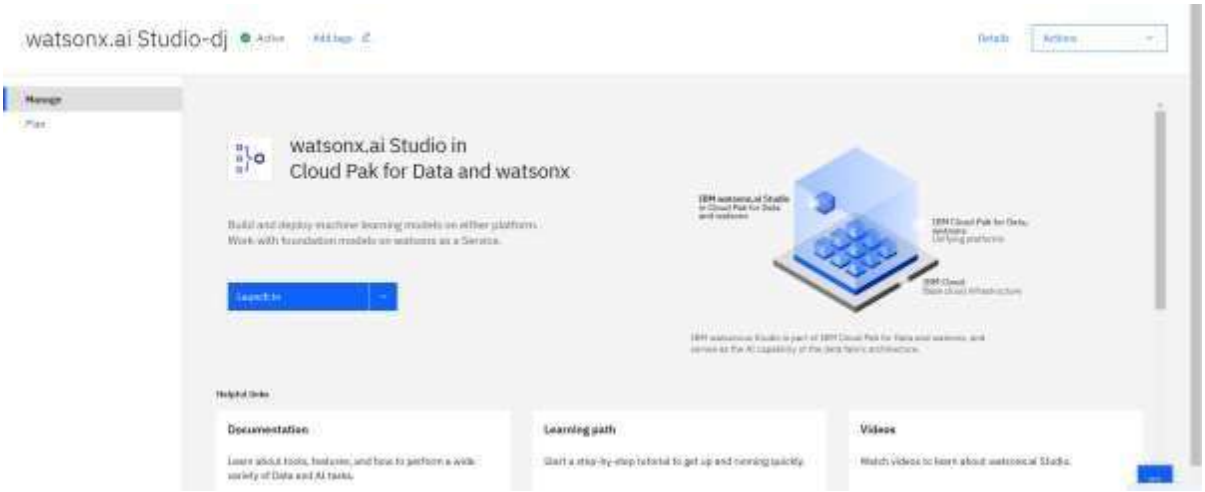
Algorithm:

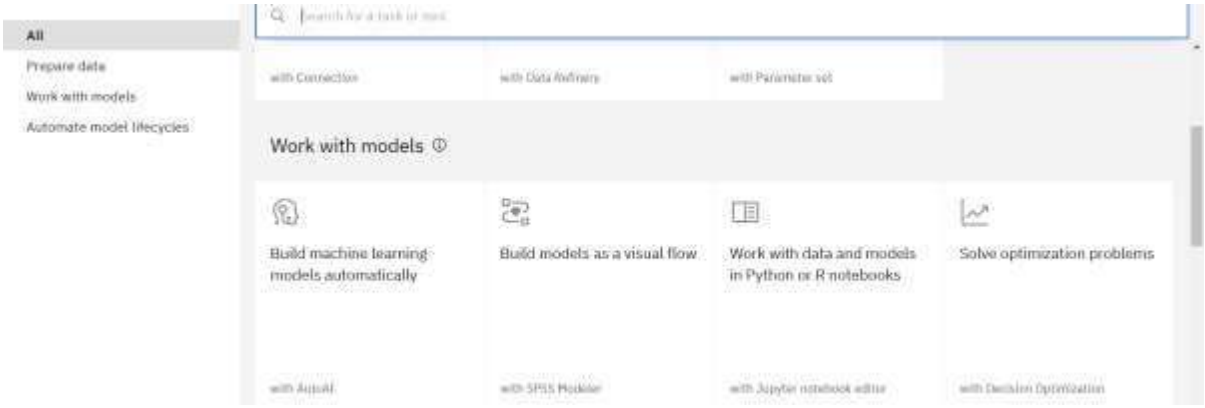
Step 1: Access Watson Studio and Create an AutoAI Experiment

- Sign in to IBM Cloud.



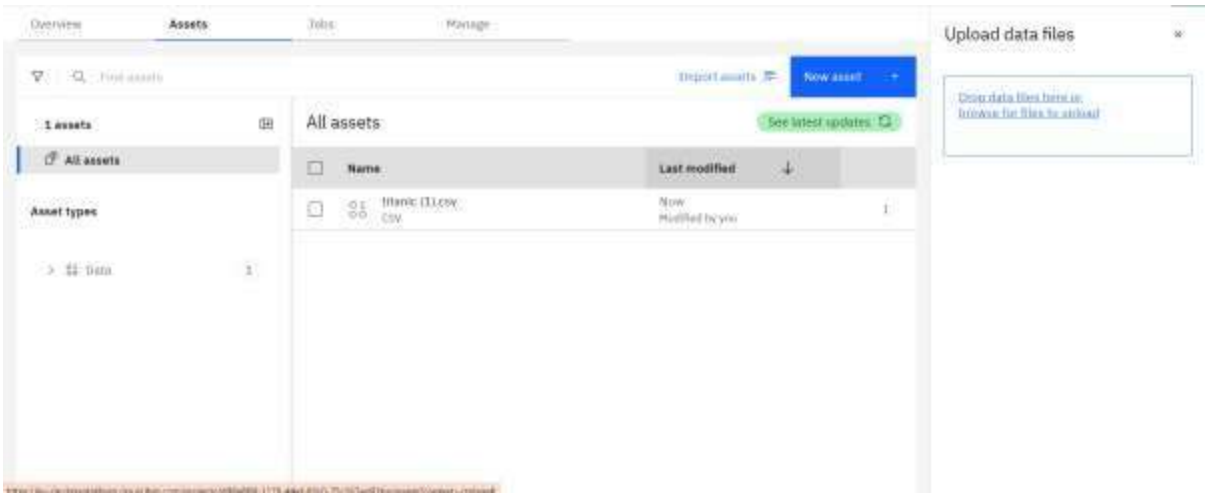
Open **Watson Studio** and create a new **AutoAI experiment** under "Projects".





Step 2: Upload the Dataset

- Click on "Add Data" and upload a structured dataset (CSV, JSON, or database connection).
- Ensure the dataset has a clear target variable for prediction.



Step 3: Configure the AutoAI Experiment

- Select the **prediction type** (classification or regression).
- Choose the **target variable** from the dataset.
- AutoAI will automatically analyze the dataset and handle missing values, data transformations, and feature engineering.

Step 4: Automated Model Selection and Training

- AutoAI will generate multiple machine learning pipelines with different algorithms.
- It will **rank** models based on performance metrics (e.g., accuracy, F1-score, RMSE).
- Select the best-performing model for deployment.

Step 5: Evaluate Model Performance

- Review performance metrics like precision, recall, confusion matrix, and ROC curves.
- Compare multiple models to choose the most accurate and efficient one.

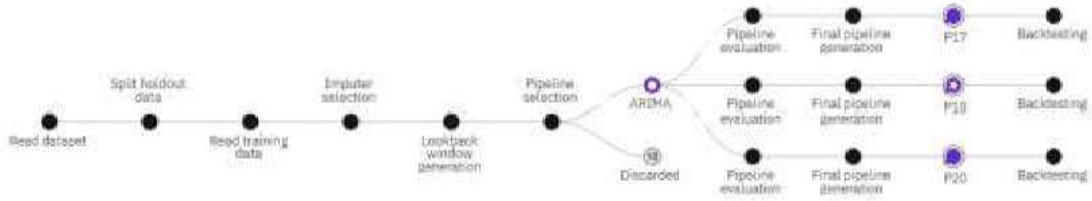
Step 6: Deploy the Model

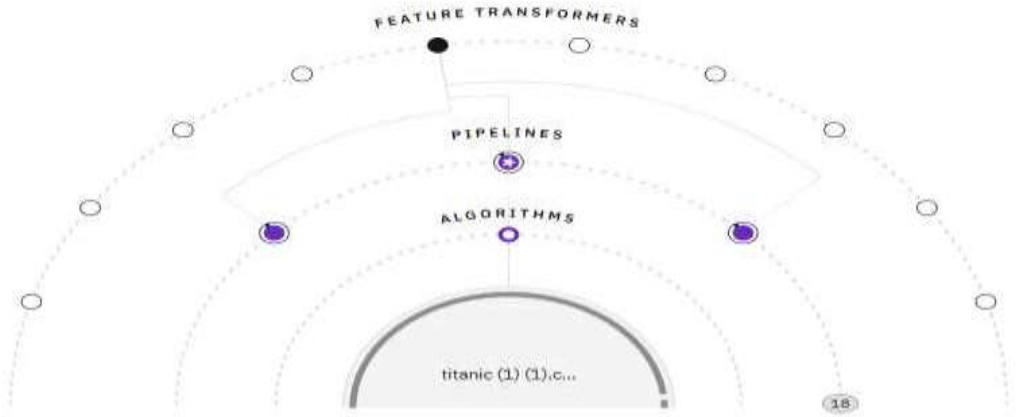
- Click on "Deploy" to create an API endpoint.
- The model can now be integrated into applications via REST API calls.

Step 7: Monitor and Improve the Model

- Use Watson Studio's monitoring tools to track model performance.
- Retrain the model periodically with updated data for better accuracy.

Output:





Result:

- AutoAI generated multiple pipelines, each representing a different ML model with optimized parameters.
- The best model was selected based on evaluation metrics (e.g., Accuracy, F1-score, RMSE, R^2).
- The optimized model was deployed for real-world predictions with minimal manual effort.

EXP NO		DATE
5	**Implementation of Virtual Dat center**	

AIM:

To create a Cloudant Service in IBM cloud Application to create aCRUD (Create, Read, Update, and Delete) in the database.

ALGORITHM:

Step 1: Using the IBM, Cloud Id, and Password, Login into the CloudId.

Step 2: Get into the IBM Cloud Dashboard.

Step 3: In a search box type Cloudant and Right click on that.

Step 4: After Opening the Cloudant, Click the IAM option.

Step 5: Create.

Step 6: After Creating a Cloudant, Click on the Database option.

Step 7: Click Cloudant.

Step 8: Launch Dashboard.

Step 9: Create a Database.

Step 10: Give Database name, Select Non-partitional work for mostwork.

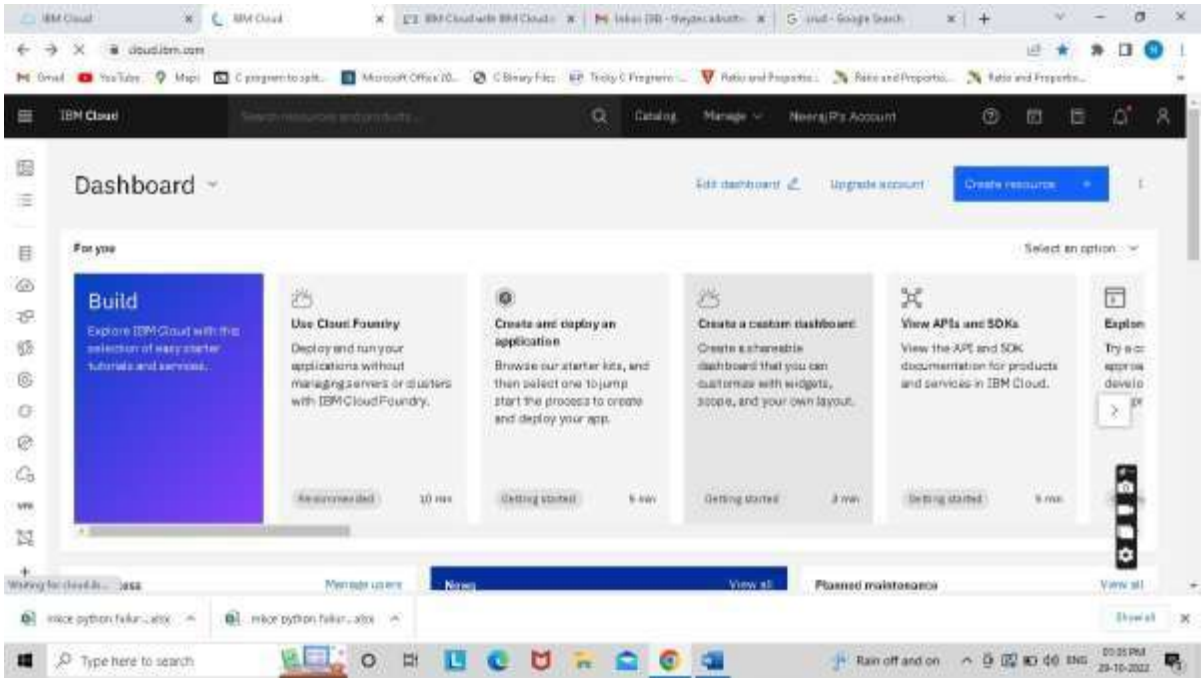
Step 11: Create a Document using Json coding.

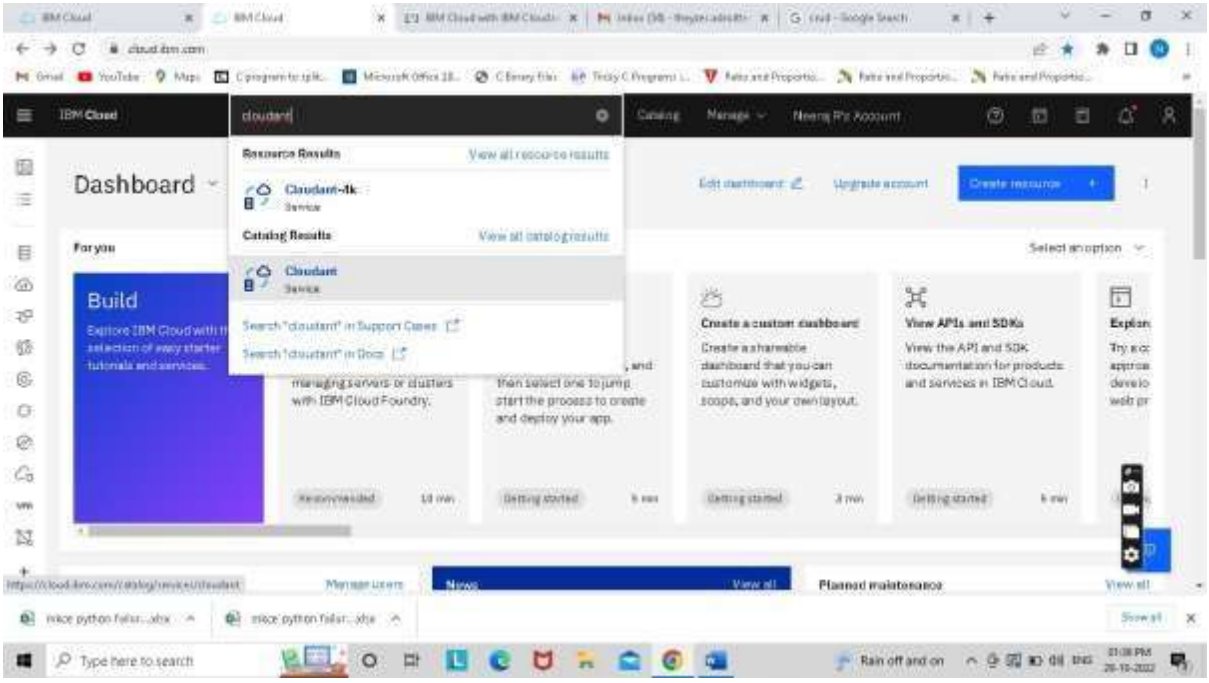
Step 12: Delete the Document.

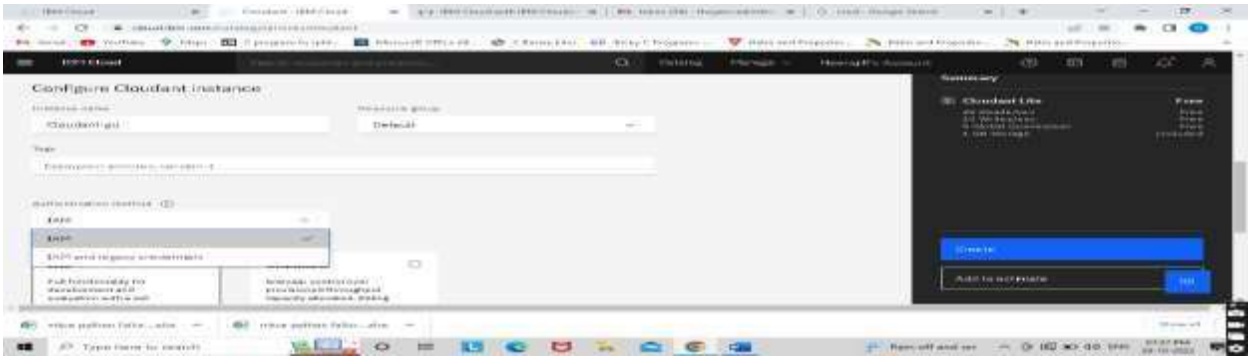
Step 13: By using the query Read the whole Database.

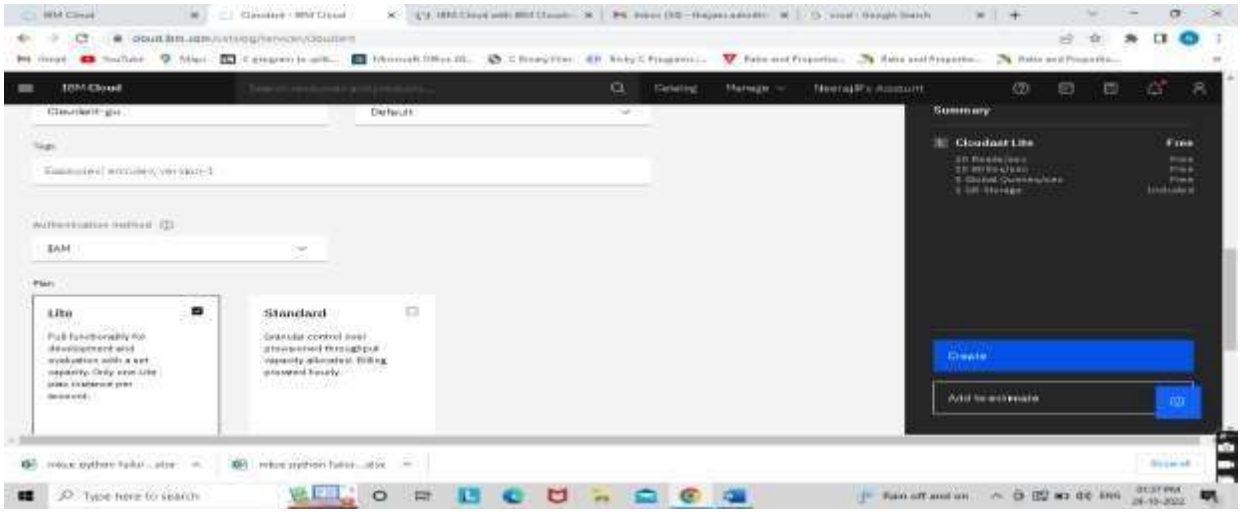
Step 14: Update the whole Database.

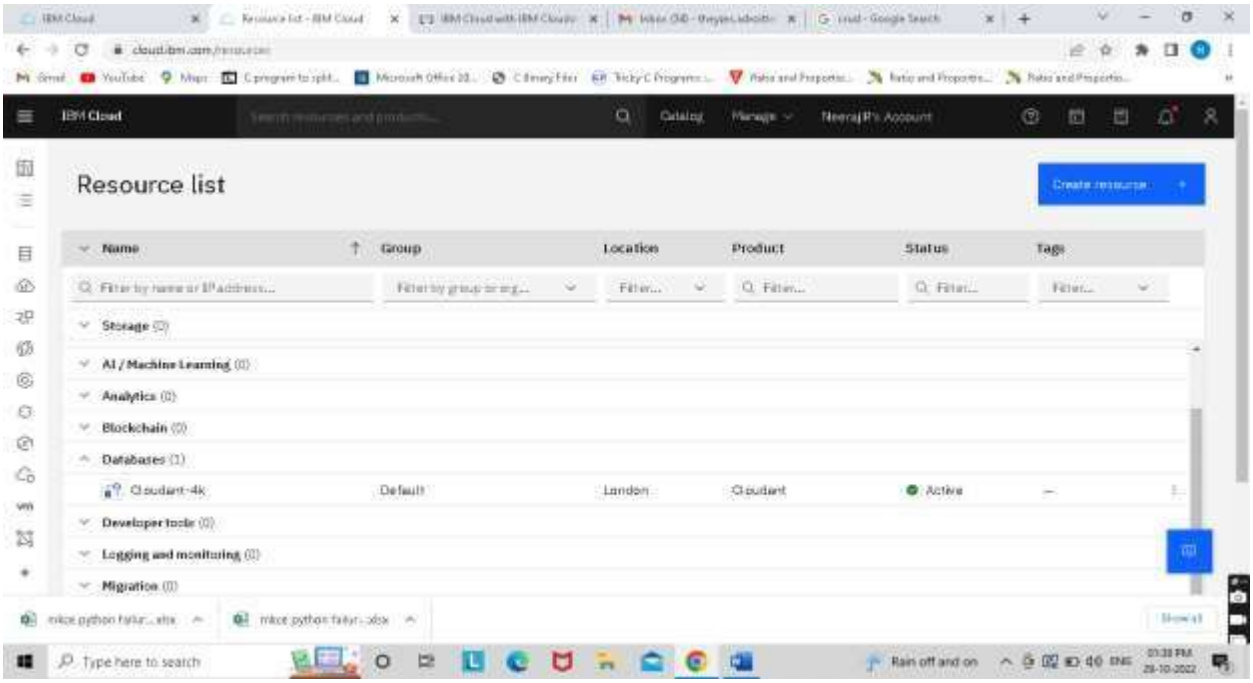
Following Screenshots:

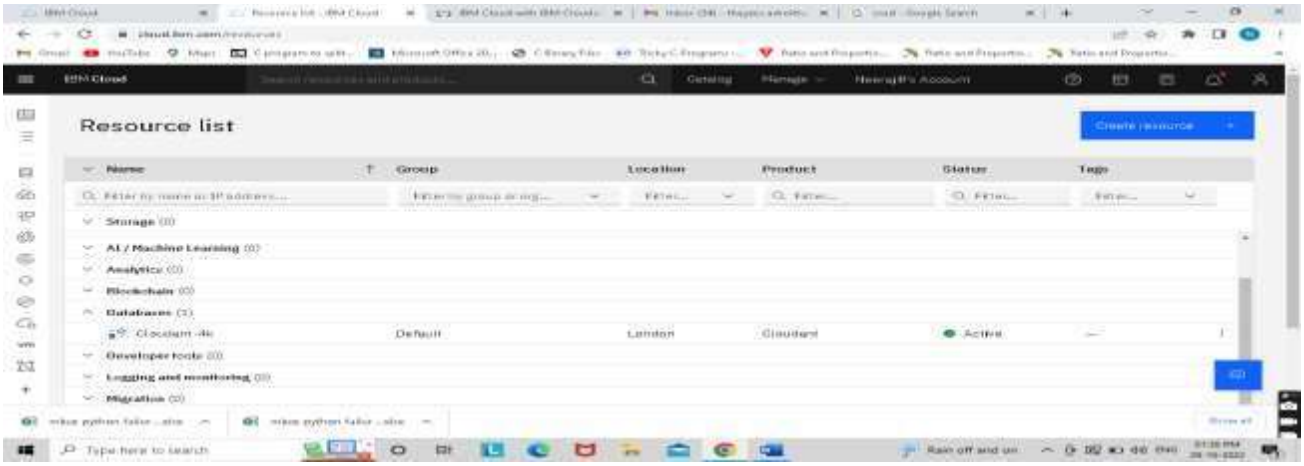


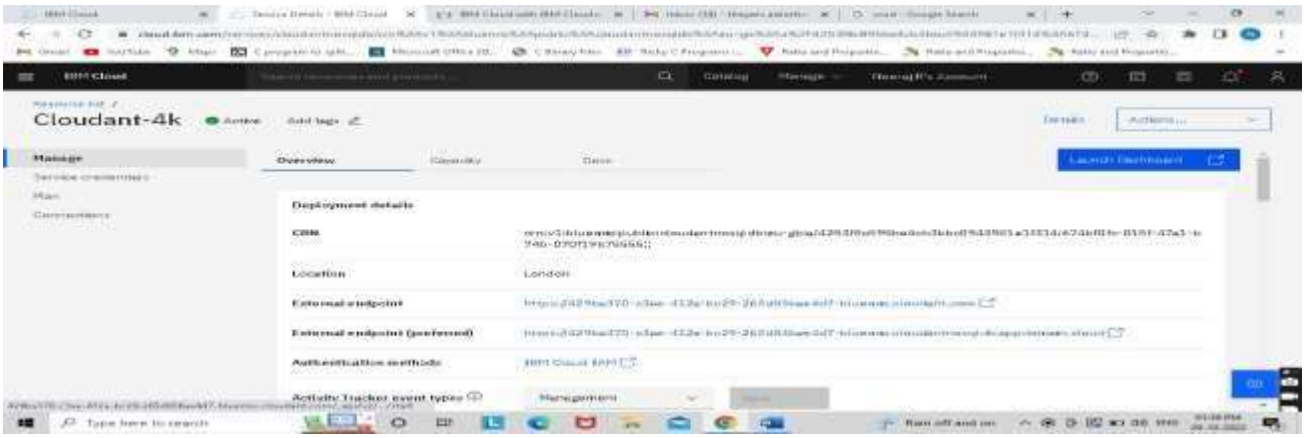


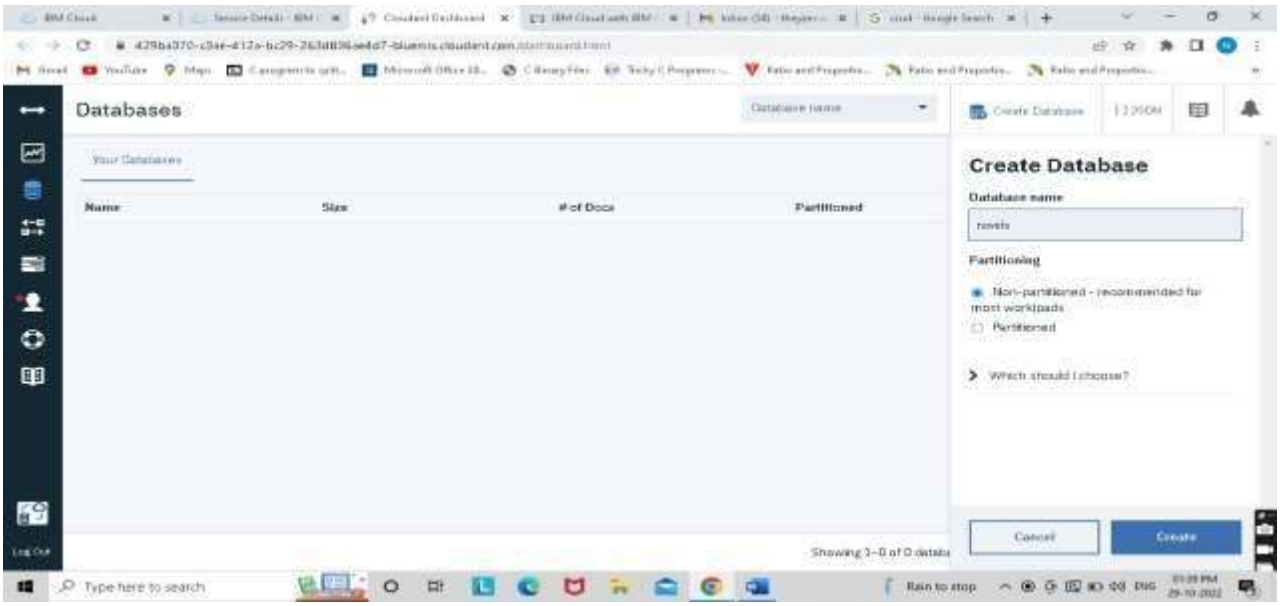

Configure Cloudant instance
Summary
Cloudant Lite
Free
Create
Add to estimate

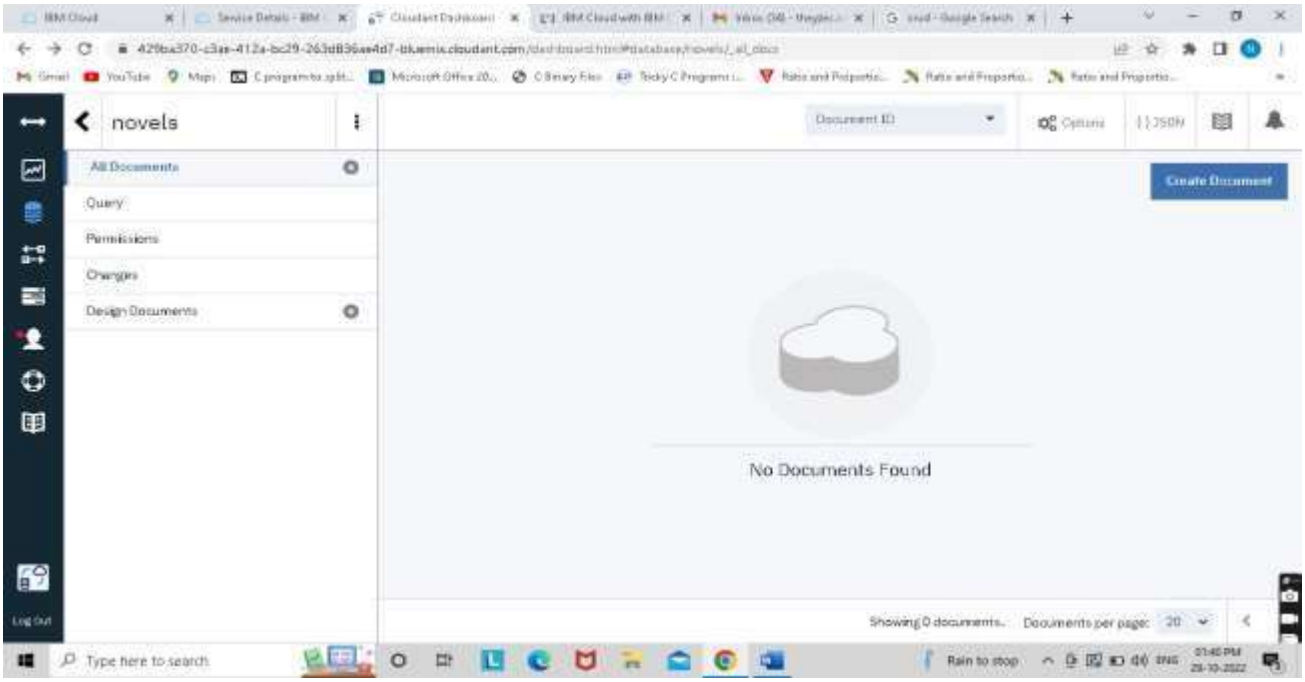

IBM Cloud
Catalog
Manage
Summary
Cloudant Lite
Free
Lite
Standard
Create
Add to estimate


IBM Cloud
Catalog
Manage
Resource list
Create resource
Name
Group
Location
Product
Status
Tags
Storage (0)
AI / Machine Learning (0)
Analytics (0)
Blockchain (0)
Databases (1)
Cloudant-4k
Default
London
Cloudant
Active
Developer tools (0)
Logging and monitoring (0)
Migration (0)


Resource list
Create resource
Name
Group
Location
Product
Status
Tags
Storage
AI / Machine Learning
Analytics
Blockchain
Databases
Cloudant-4k
Default
London
Cloudant
Active
Developer tools
Logging and monitoring
Migration


Cloudant-4k
Active
Manage
Overview
Launch Dashboard
Deployment details
CRN
Location
London
External endpoint
External endpoint (preferred)
Authentication methods
Activity Tracker event types
Management

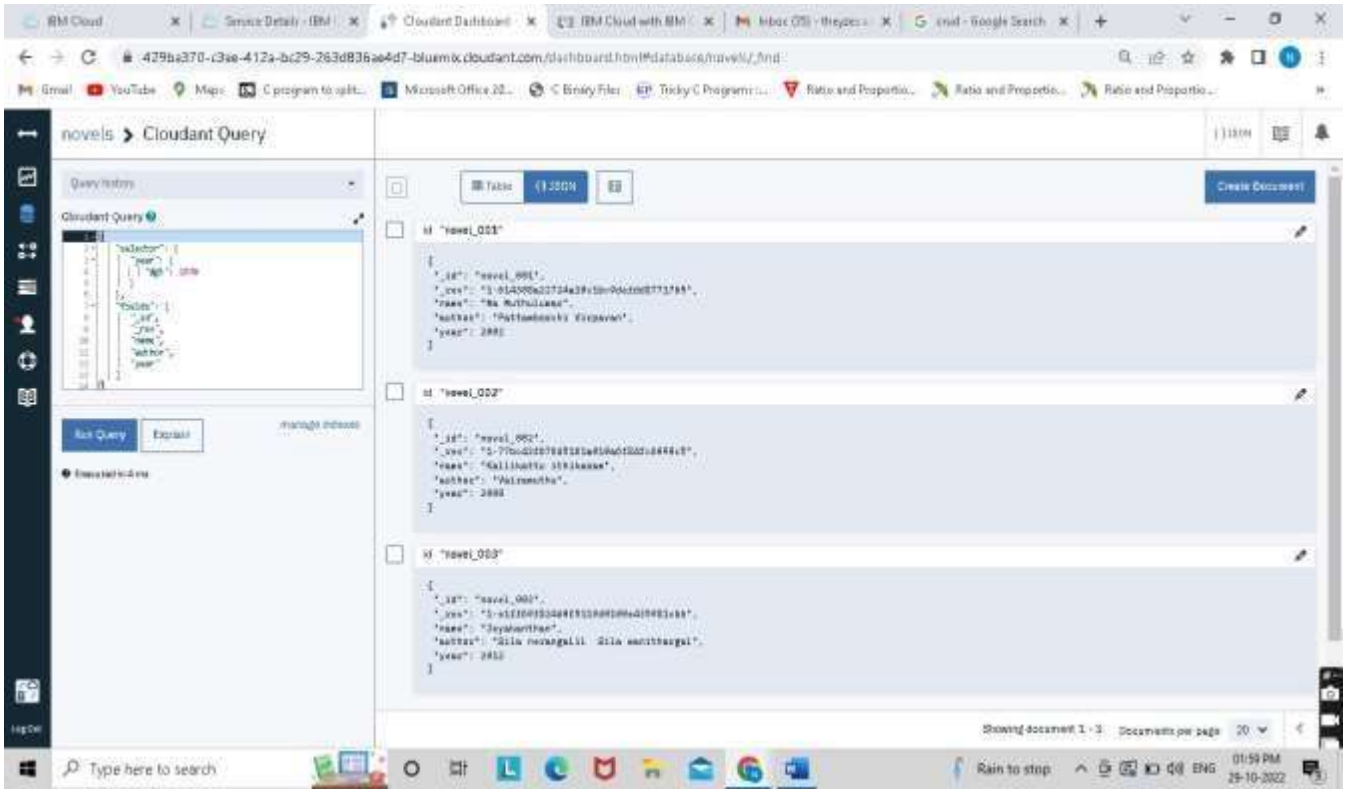

Databases
Create Database
Your Databases
Name
Size
# of Docs
Partitioned
Create Database
Database name
novels
Partitioning
Non-partitioned - recommended for most workloads
Partitioned
Which should I choose?
Cancel
Create
Showing 1–0 of 0 databases



JSON Database Code:

```
{
  "id": "novel_001",
  "title": "Pride and Prejudice",
  "author": "Jane Austen",
  "year": 1813,
  "genre": "Romance",
  "language": "English",
  "summary": "A classic novel that explores themes of love, class, and social expectations through the story of Elizabeth Bennet and Mr. Darcy."
}
```

Output:



Result:

The cloudant program was successfully Executed

EXP NO	Configuration of Virtual internetworking Components	DATE
6		

Aim:

To implement an Employee Database using IndexedDB in an HTML page, allowing users to insert, store, and display employee records using a client-side database.

Alogrithm:

Step 1: Open or Create the IndexedDB Database

1. Create an IndexedDB instance named "EmployeeDB".
2. Set the database version to 1.
3. Define an object store "Employees" with a primary key as "id".
4. Create indexes for "Name" and "Salary" fields.

Step 2: Handle Database Events

1. Use onupgradeneeded to create the database and define object stores and indexes.
2. Use onsuccess to establish the database connection.
3. Use onerror to handle connection failures.

Step 3: Insert Employee Data

1. Capture the ID, Name, and Salary from user input fields.
2. Open a readwrite transaction on the "Employees" object store.
3. Add the data using the .add() method.
4. Handle success and error events.

Step 4: Display Stored Employee Data

1. Open a readonly transaction on the "Employees" object store.
2. Use openCursor() to iterate through stored records.
3. Dynamically update the table (tblGrid) with retrieved employee data.

Coding:

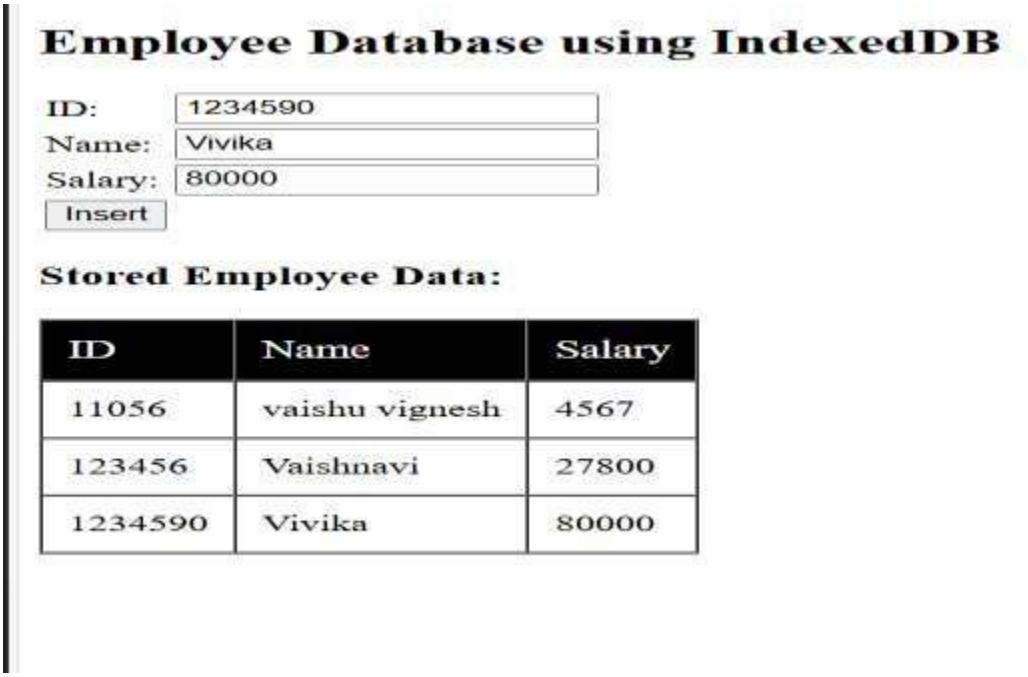
```
<!DOCTYPE HTML>
<html>
<head>
    <title>IndexedDB Example</title>
    <script>
        let db;

        // Open (or create) the database
        let request = indexedDB.open("EmployeeDB", 1);

        request.onupgradeneeded = function(event) {
            db = event.target.result;
            let objectStore = db.createObjectStore("Employees", { keyPath: "id" });
            objectStore.createIndex("Name", "Name", { unique: false });
            objectStore.createIndex("Salary", "Salary", { unique: false });
        };

        request.onsuccess = function(event) {
            db = event.target.result;
            displayData(); // Load data on page load
        };
```

```
        request.onerror = function(event) {
            console.log("Error opening database:", event.target.errorCode);
        };

        function Insert() {
            let id = document.getElementById("tbID").value;
            let Name = document.getElementById("tbName").value;
            let Salary = document.getElementById("tbSalary").value;

            let transaction = db.transaction(["Employees"], "readwrite");
            let objectStore = transaction.objectStore("Employees");

            let request = objectStore.add({ id: id, Name: Name, Salary: Salary });

            request.onsuccess = function() {
                console.log("Data inserted successfully");
                displayData(); // Refresh the table
            };
```

```
            request.onerror = function() {
                console.log("Error adding data");
            };
        }

        function displayData() {
            let table = document.getElementById("tblGrid");
            table.innerHTML = `
                <tr style="background-color:black;color:white;font-size:18px;">
                    <td>ID</td><td>Name</td><td>Salary</td>
                </tr>`; // Reset table

            let transaction = db.transaction(["Employees"], "readonly");
            let objectStore = transaction.objectStore("Employees");

            objectStore.openCursor().onsuccess = function(event) {
                let cursor = event.target.result;
                if (cursor) {
                    let row = `<tr>
                        <td>${cursor.value.id}</td>
                        <td>${cursor.value.Name}</td>
                        <td>${cursor.value.Salary}</td>
                    </tr>`;
                    table.innerHTML += row;
                    cursor.continue();
                }
            };
        }
    </script>
</head>
<body>
    <h2>Employee Database using IndexedDB</h2>
    <form id="frm">
        <table>
            <tr>
                <td>ID:</td>
                <td><input type="text" id="tbID"/></td>
            </tr>
            <tr>
                <td>Name:</td>
                <td><input type="text" id="tbName"/></td>
            </tr>
            <tr>
                <td>Salary:</td>
                <td><input type="text" id="tbSalary"/></td>
            </tr>
            <tr>
                <td><button type="button" onclick="Insert()">Insert</button></td>
            </tr>
        </table>
    </form>
```

```
    <h3>Stored Employee Data:</h3>
    <table id="tblGrid" cellpadding="10px" cellspacing="0" border="1">
        <tr style="background-color: ■black;color: □white;font-size:18px;">
            <td>ID</td>
            <td>Name</td>
            <td>Salary</td>
        </tr>
    </table>
</body>
</html>
```

OUTPUT:



Employee Database using IndexedDB

ID: 1234590
Name: Vivika
Salary: 80000
Insert

Stored Employee Data:

ID	Name	Salary
11056	vaishu vignesh	4567
123456	Vaishnavi	27800
1234590	Vivika	80000

Result:

The Employeed Database was successfully Implemented

EXP NO	Deployment in AWS	DATE
7		

Aim:

To develop an interactive College Event Invitation Webpage using HTML, CSS, and JavaScript. The page includes an animated invitation card, a real-time countdown timer, and an RSVP form for attendees. The goal is to enhance user engagement with an interactive interface**.**

Step 1: Create the HTML Structure

1. Design an invitation card with event details (title, date, time, venue).
2. Add an RSVP form with input fields for name and email.

Step 2: Style the Webpage Using CSS

3. Apply a gradient background and responsive card design.
4. Add animations for smooth transitions.

Step 3: Implement JavaScript Functionalities

5. Create a countdown timer that updates in real-time.
6. Validate the RSVP form and display a confirmation alert.
7. Clear the form after successful RSVP submission.

Coding:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>College Event Invitation</title>
    <style>
        body {
            font-family: 'Arial', sans-serif;
            background: linear-gradient(to right, #ff9966, #ff5e62);
            text-align: center;
            padding: 20px;
            color: white;
        }
        .invitation-card {
            background: white;
            color: #333;
            border-radius: 12px;
            padding: 30px;
            max-width: 500px;
            margin: auto;
            box-shadow: 0px 4px 15px rgba(0, 0, 0, 0.2);
            animation: fadeIn 1.5s ease-in-out;
        }
        @keyframes fadeIn {
            from { opacity: 0; transform: scale(0.8); }
            to { opacity: 1; transform: scale(1); }
        }
        h1 {
            color: #e74c3c;
        }
        .details {
            font-size: 18px;
            margin: 10px 0;
            color: #444;
        }
```

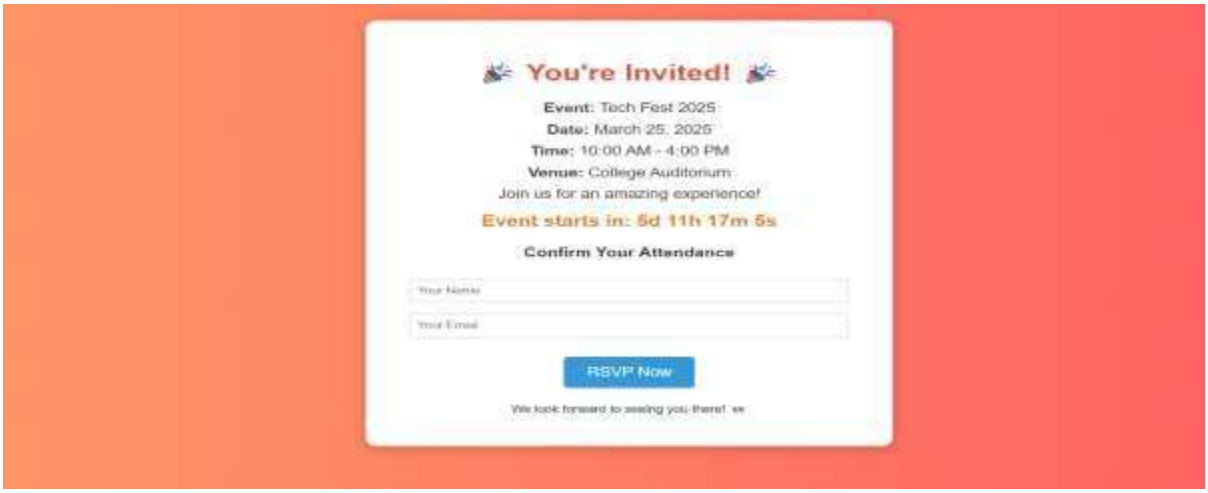
```
            color: #444;
        }
        .countdown {
            font-size: 22px;
            font-weight: bold;
            margin-top: 15px;
            color: #e67e22;
        }
        .btn {
            display: inline-block;
            margin-top: 20px;
            padding: 12px 25px;
            font-size: 18px;
            background: #3498db;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            text-decoration: none;
            transition: 0.3s;
        }
        .btn:hover {
            background: #217dbb;
        }
        .form-container {
            margin-top: 20px;
        }
        input {
            width: 90%;
            padding: 8px;
            margin: 8px 0;
            border: 1px solid #ccc;
            border-radius: 5px;
```

```
        }
        .footer {
            margin-top: 20px;
            font-size: 14px;
        }
    </style>
</head>
<body>

    <div class="invitation-card">
        <h1>🎉 You're Invited! 🎉</h1>
        <p class="details"><strong>Event:</strong> Tech Fest 2025</p>
        <p class="details"><strong>Date:</strong> March 25, 2025</p>
        <p class="details"><strong>Time:</strong> 10:00 AM - 4:00 PM</p>
        <p class="details"><strong>Venue:</strong> College Auditorium</p>
        <p class="details">Join us for an amazing experience!</p>

        <!-- Countdown Timer -->
        <p class="countdown" id="timer">Event starts in: Loading...</p>

        <!-- RSVP Form -->
        <div class="form-container">
            <h3>Confirm Your Attendance</h3>
            <input type="text" id="name" placeholder="Your Name">
            <input type="email" id="email" placeholder="Your Email">
            <button class="btn" onclick="confirmAttendance()">RSVP Now</button>
        </div>

        <p class="footer">We look forward to seeing you there! 🎶</p>
    </div>
```

```
    <p class="footer">We look forward to seeing you there! 🎶</p>
</div>

<script>
    // Countdown Timer Script
    function updateCountdown() {
        const eventDate = new Date("March 25, 2025 10:00:00").getTime();
        const now = new Date().getTime();
        const timeLeft = eventDate - now;

        const days = Math.floor(timeLeft / (1000 * 60 * 60 * 24));
        const hours = Math.floor((timeLeft % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
        const minutes = Math.floor((timeLeft % (1000 * 60 * 60)) / (1000 * 60));
        const seconds = Math.floor((timeLeft % (1000 * 60)) / 1000);

        document.getElementById("timer").innerHTML = `Event starts in: ${days}d ${hours}h ${minutes}m ${seconds}s`;

        if (timeLeft < 0) {
            document.getElementById("timer").innerHTML = "The event has started!";
        }
    }
    setInterval(updateCountdown, 1000);

    // RSVP Confirmation Script
    function confirmAttendance() {
        let name = document.getElementById("name").value;
        let email = document.getElementById("email").value;

        if (name === "" || email === "") {
            alert("Please enter your name and email to confirm your RSVP.");
```

```
        } else {
            alert(`Thank you, ${name}! Your RSVP has been confirmed. See you at the event! 🎉`);
            document.getElementById("name").value = "";
            document.getElementById("email").value = "";
        }
    }
</script>

</body>
</html>
```

Output:



Result:

A functional event invitation webpage with a countdown timer and an RSVP form. Users can enter their details and get a confirmation message upon successful RSVP.

EXP NO		DATE
8	**Install Google App engine and perform operations on it.**	

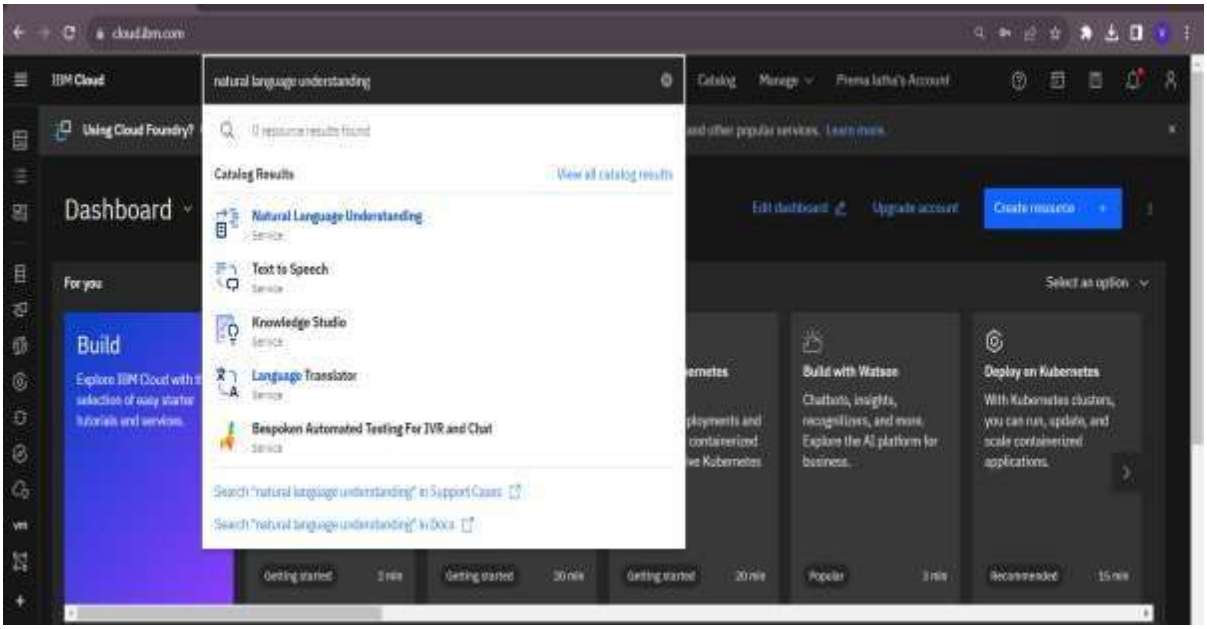
Aim:

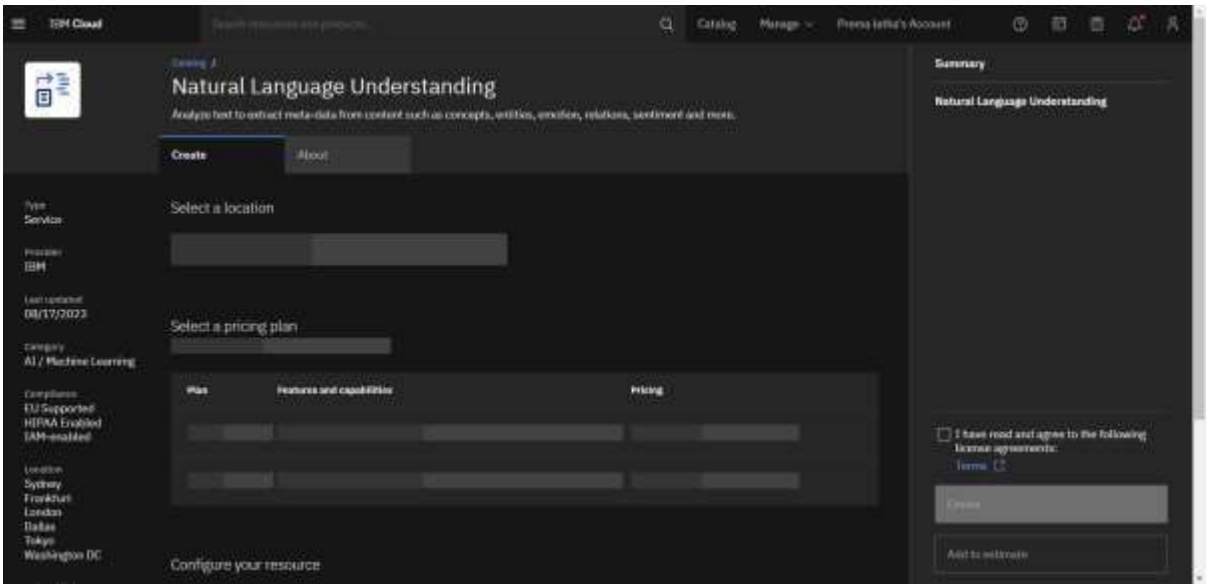
To create a Natural Language using Watson studio.

Algorithm:

Step 1: Open IBM Watson studio and Login using your account. The project and the catalog must be created by members of the same IBM Cloud account.

Step 2: click Natural language understanding.

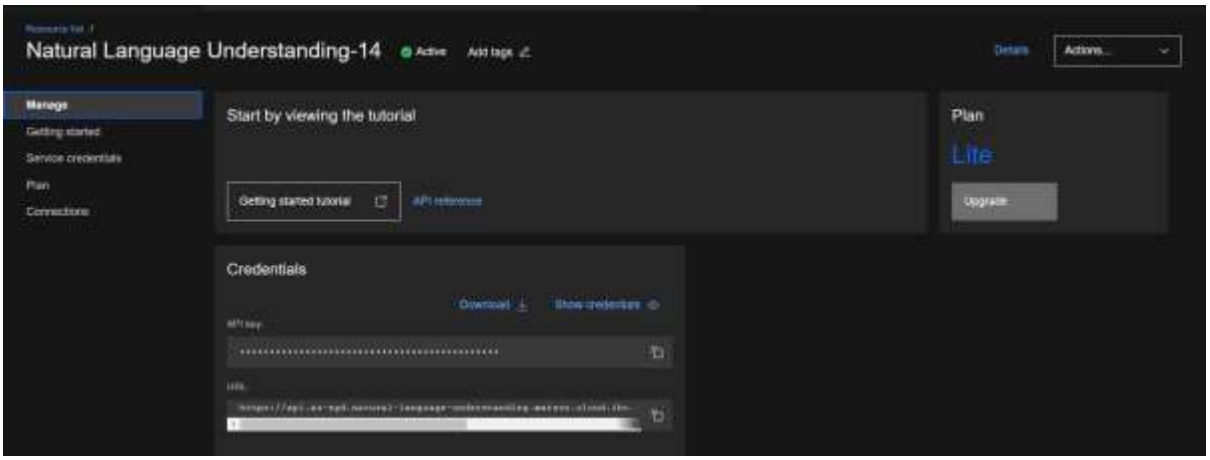




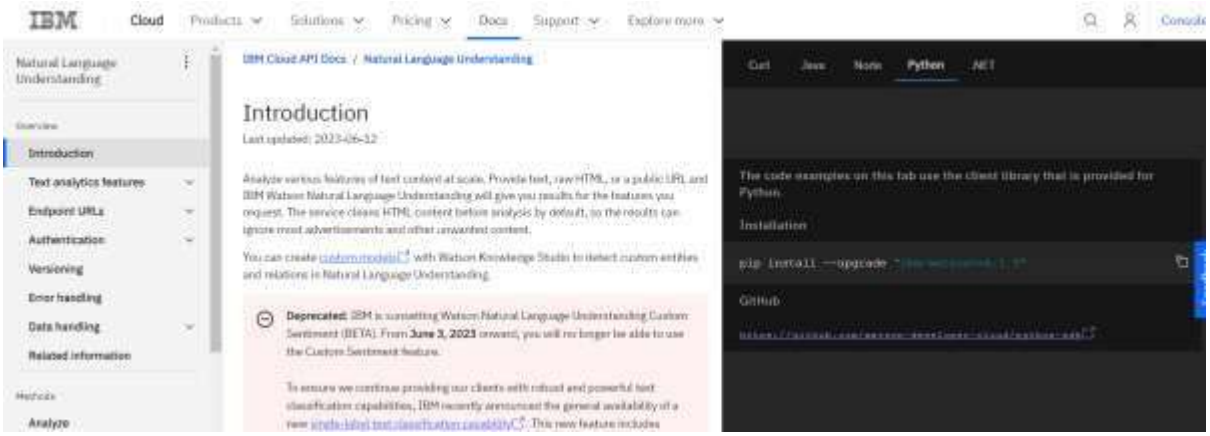
Step 3: Click the Lite version to create the Natural language page.

Step 4: Read the document How to create Natural language processing.

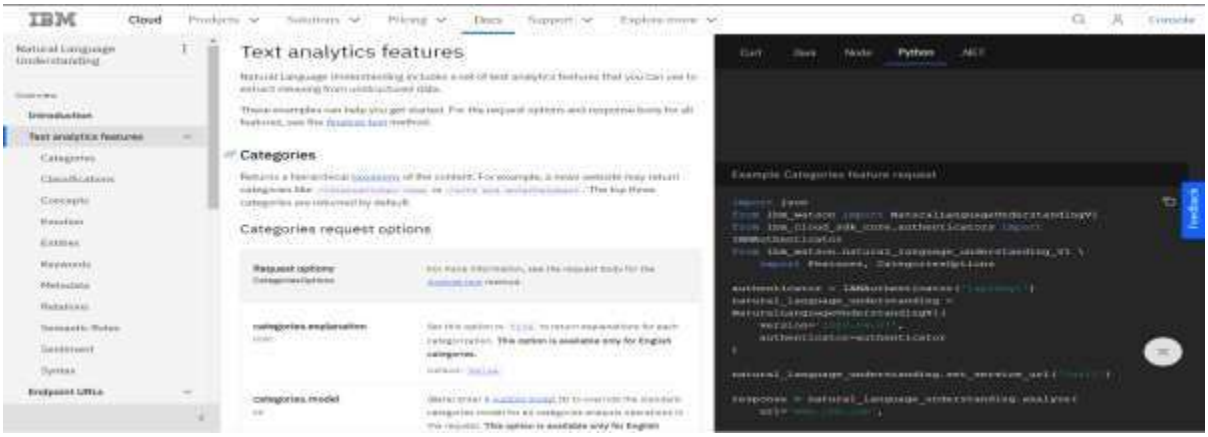
Step 5: Move on the Credentials of Natural Language processing.



Step 6: In the right-hand side under python the code will already exist, Make use of the code in Python notebook inside Watson studio

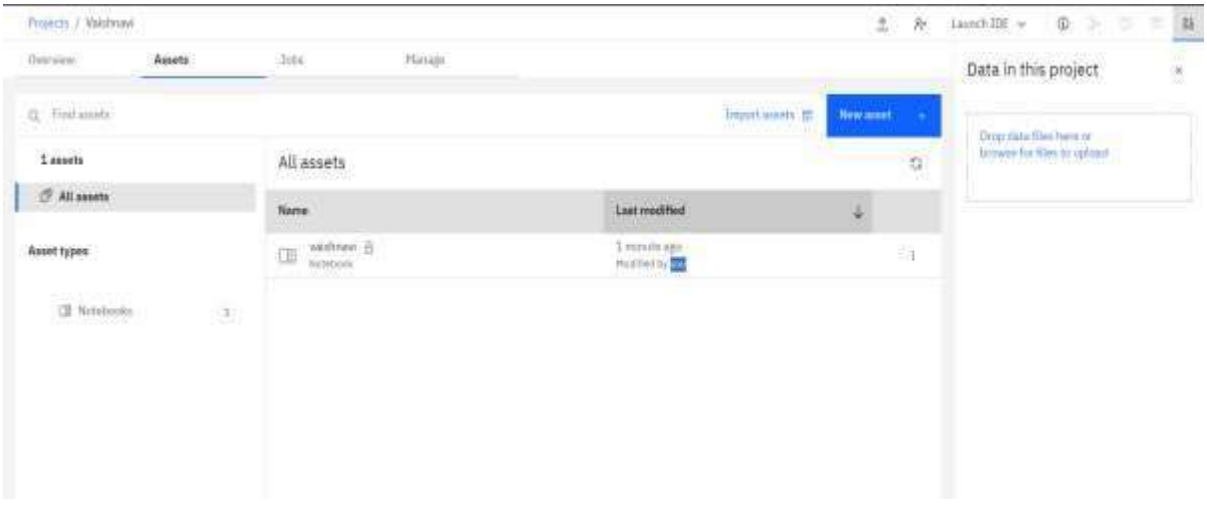


Step 7: Copy the code and run in the Watson studio.



Step 8: Open the Watson studio services create lite version.

Step 9: Click the New asset and enter the Jupyter notebook on search bar**.**



Step 10: Run the copy code in the Watson studio.

```
pip install --upgrade "ibm-watson>=6.1.0"
```

```
Collecting ibm-watson>=6.1.0
  Downloading ibm-watson-7.0.1.tar.gz (389 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 389.3/389.3 kB 25.8 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting websocket-client>=1.1.0
  Downloading websocket_client-1.6.2-py3-none-any.whl (57 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.0/57.0 kB 12.4 MB/s eta 0:00:00
Requirement already satisfied: python-dateutil>=2.5.3 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from ibm-watson>=6.1.0) (2.8.2)
Requirement already satisfied: ibm-cloud-sdk-core==3.*,>=3.3.6 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from ibm-watson>=6.1.0) (3.16.5)
Requirement already satisfied: requests<3.0,>=2.0 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from ibm-watson>=6.1.0) (2.31.0)
Requirement already satisfied: urllib3<2.0.0,>=1.26.0 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from ibm-cloud-sdk-core==3.*,>=3.3.6->ibm-watson>=6.1.0)
(1.26.11)
Requirement already satisfied: PyJWT<3.0.0,>=2.4.0 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from ibm-cloud-sdk-core==3.*,>=3.3.6->ibm-watson>=6.1.0) (2.
4.0)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from python-dateutil>=2.5.3->ibm-watson>=6.1.0) (1.16.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from requests<3.0,>=2.0->ibm-watson>=6.1.0) (3.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from requests<3.0,>=2.0->ibm-watson>=6.1.0) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python-3.10/lib/python3.10/site-packages (from requests<3.0,>=2.0->ibm-watson>=6.1.0) (2023.7.22)
Building wheels for collected packages: ibm-watson
  Building wheel for ibm-watson (pyproject.toml) ... done
  Created wheel for ibm-watson: filename=ibm_watson-7.0.1-py3-none-any.whl size=389000 sha256=6aa6a3c3fc2f350bf48a6f232c6b37dee08f72f98695071f21ec733defb0e626
  Stored in directory: /tmp/wsuser/.cache/pip/wheels/34/df/f4/f0edc5ba0637dd4bfb2029741ae20402976a49d1b6bc113553
Successfully built ibm-watson
Installing collected packages: websocket-client, ibm-watson
Successfully installed ibm-watson-7.0.1 websocket-client-1.6.2
Note: you may need to restart the kernel to use updated packages.
```

```
In [7]: import json
        from ibm_watson import NaturalLanguageUnderstandingV1
        from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
        from ibm_watson.natural_language_understanding_v1 \
            import Features, CategoriesOptions

        authenticator = IAMAuthenticator('[illegible]')
        natural_language_understanding = NaturalLanguageUnderstandingV1(
            version='2022-04-07',
            authenticator=authenticator
        )

        natural_language_understanding.set_service_url('https://api.au-syd.natural-language-understanding.watson.cloud.ibm.com/instances/2a624a18-ac2e-495e-af2c-e1e92c706926')

        response = natural_language_understanding.analyze(
            url='www.ibm.com',
            features=Features(categories=CategoriesOptions(limit=3))).get_result()

        print(json.dumps(response, indent=2))
```

OUTPUT:


```
{
  "usage": {
    "text_units": 1,
    "text_characters": 1479,
    "features": 1
  },
  "retrieved_url": "https://www.ibm.com/au-en",
  "language": "en",
  "categories": [
    {
      "score": 0.962535,
      "label": "/technology & computing/artificial intelligence"
    },
    {
      "score": 0.920983,
      "label": "/business and finance/business/business i.t."
    },
    {
      "score": 0.850616,
      "label": "/technology & computing/computing"
    }
  ]
}
```


Result :

The Natural Languaging Understanding was successfully Implemented.

EXP NO		DATE
9	**Install Cloudsim and analyse Different Algorithm in it**	

Aim:

The aim of this project in Watson Studio is to refine and visualize data for better analysis and decision-making. Using Data Refinery, raw data is cleansed, transformed, and structured to ensure accuracy and consistency. Finally, visualizations such as charts and graphs help in identifying trends and insights effectively.

Algorithm:

Phase 1: Creating a Project in Watson Studio

1. Log in to **IBM Cloud** and open **Watson Studio**.
2. Click **"Create a Project"** and select **Standard**.
3. Enter a **project name** and description.
4. Choose **Cloud Object Storage** and associate it with the project.
5. Click **"Create"** to set up the project.

Phase 2: Uploading and Preparing Data in Data Refinery

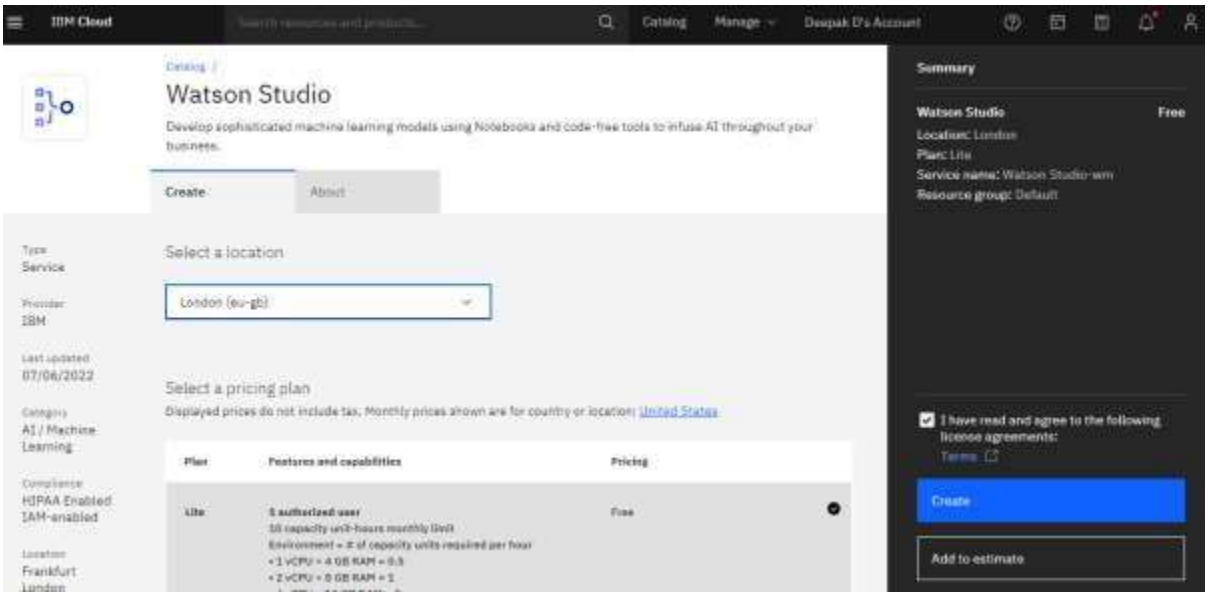
6. Open the project and go to the **Assets** tab.
7. Click **"Add Data"** and upload a dataset (CSV, Excel, JSON, etc.).
8. Select the uploaded dataset and click **"Refine"** to open **Data Refinery**.
9. Review the dataset structure (column names, data types).
10. Remove **duplicate rows** if present.
11. Handle **missing values** (fill, remove, or replace with mean/median).
12. Convert **data types** (e.g., string to integer, date formatting).
13. Normalize numerical data for consistency.

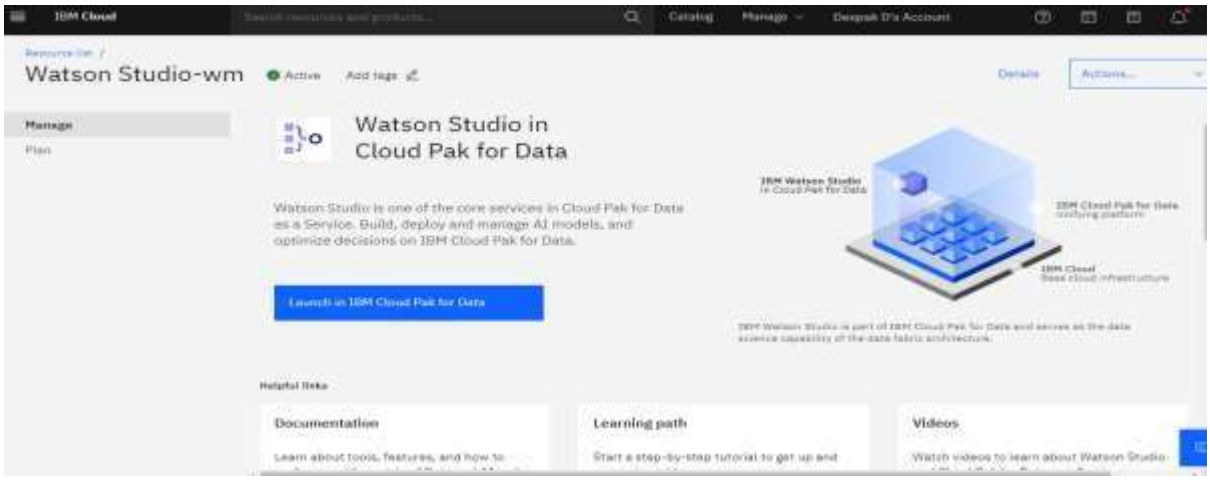
14. Rename **column names** for better readability.
15. Remove **unnecessary columns** that are not useful for analysis.
16. Filter rows based on conditions (e.g., exclude outliers).
17. Create **new columns** using transformations or calculations.
18. Use **grouping and aggregation** to summarize data.
19. Sort the dataset based on specific columns.
20. Save the cleaned dataset as a **refined version**.

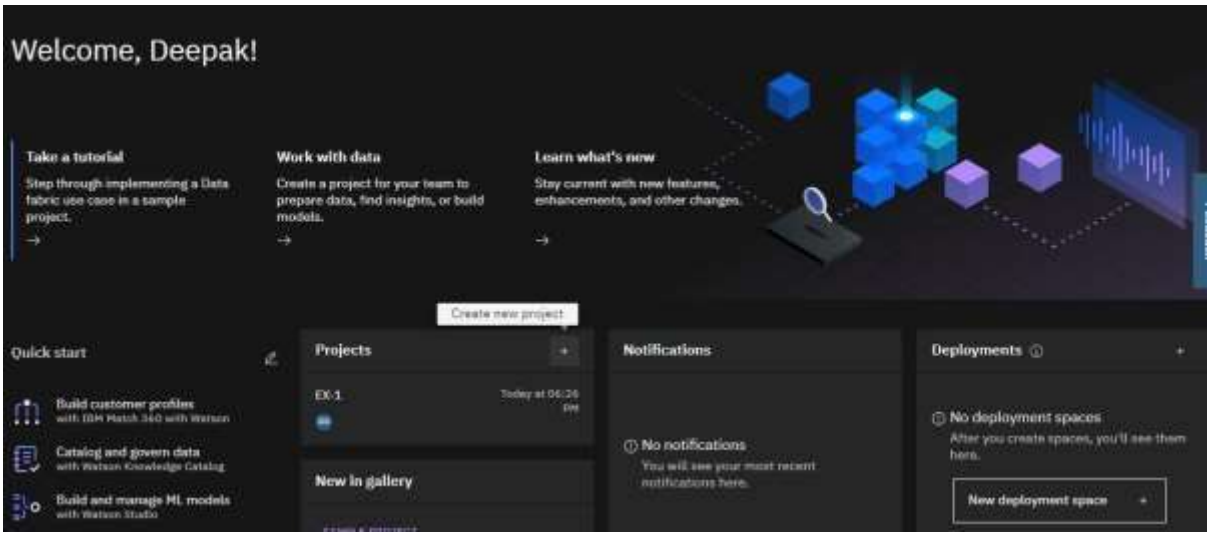
Phase 3: Visualizing Data in Watson Studio

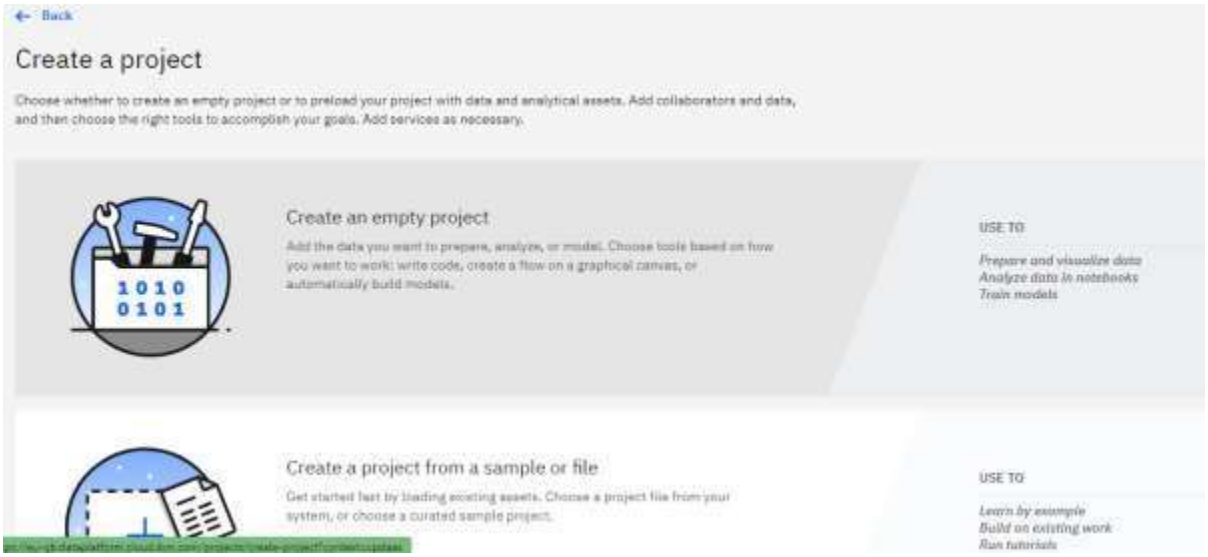
21. Click on **"Visualizations"** in Data Refinery.
22. Create **bar charts** to compare categorical data.
23. Generate **scatter plots** to analyze relationships between numerical variables.
24. Use **histograms** to understand data distribution.
25. Save visualizations for reporting and further analysis.

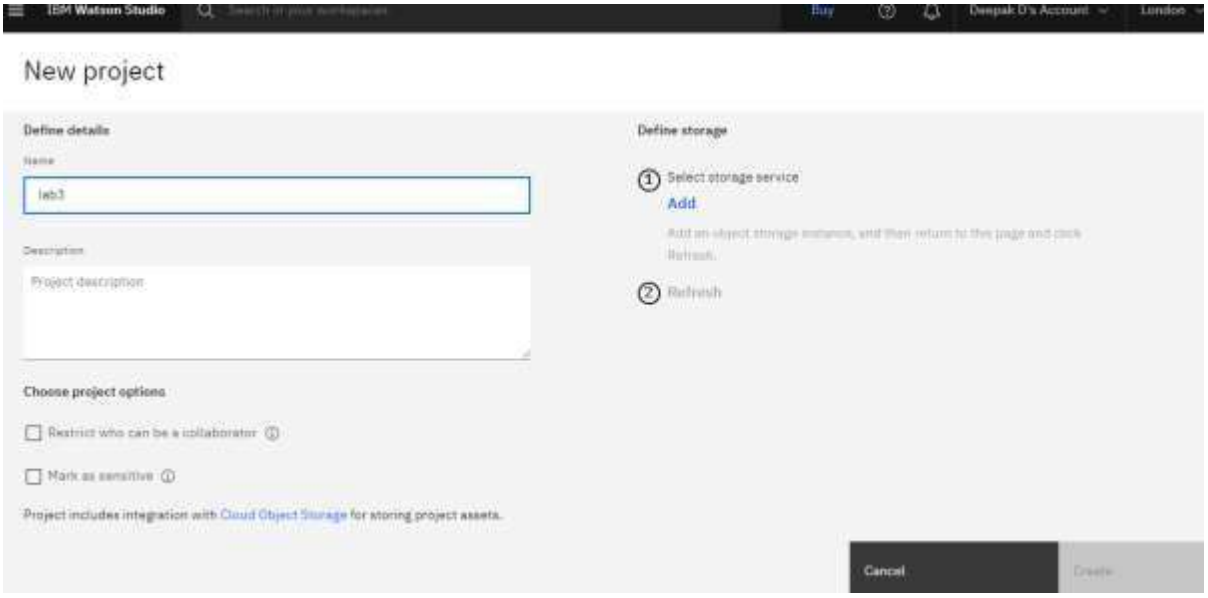
Output :

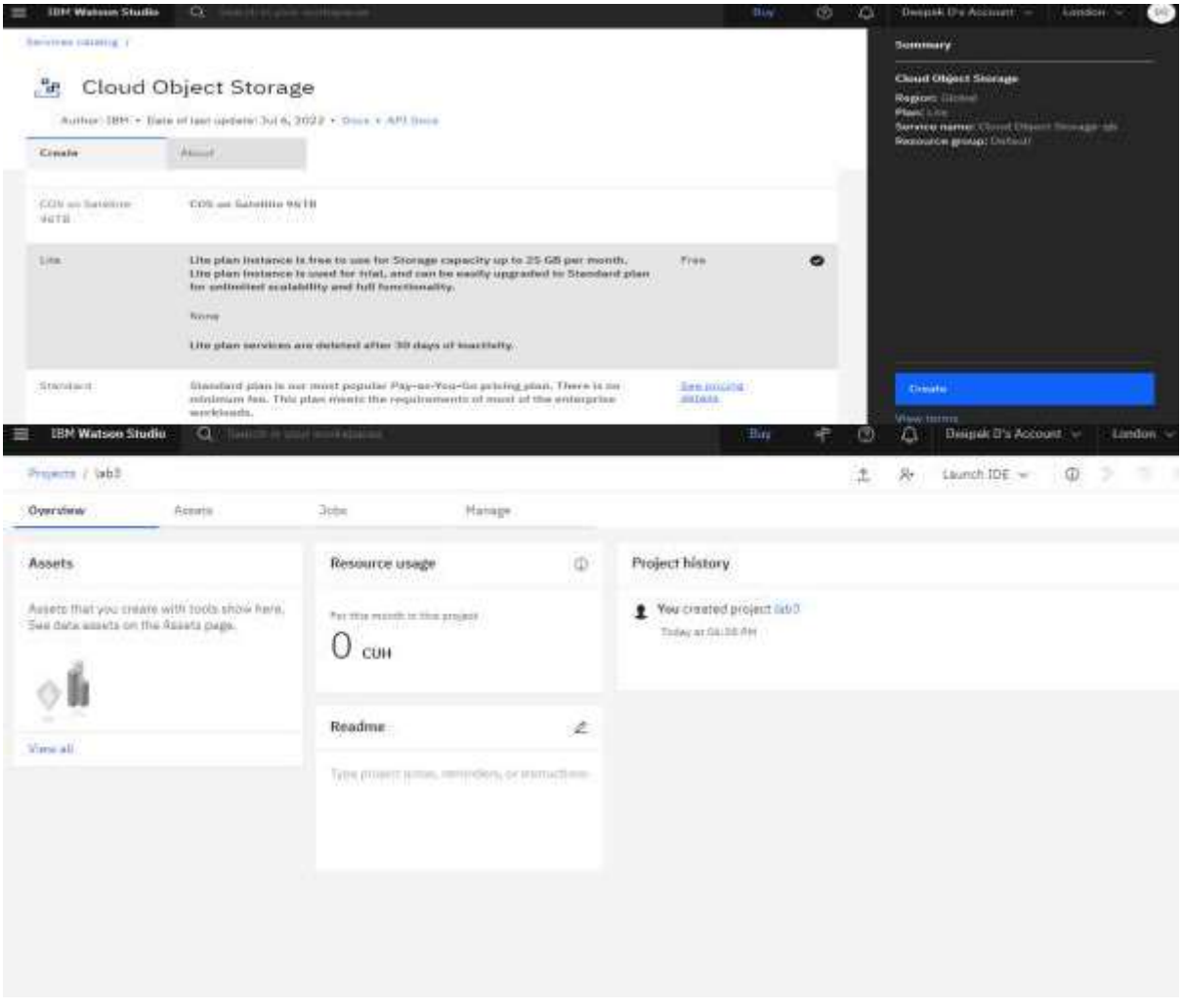


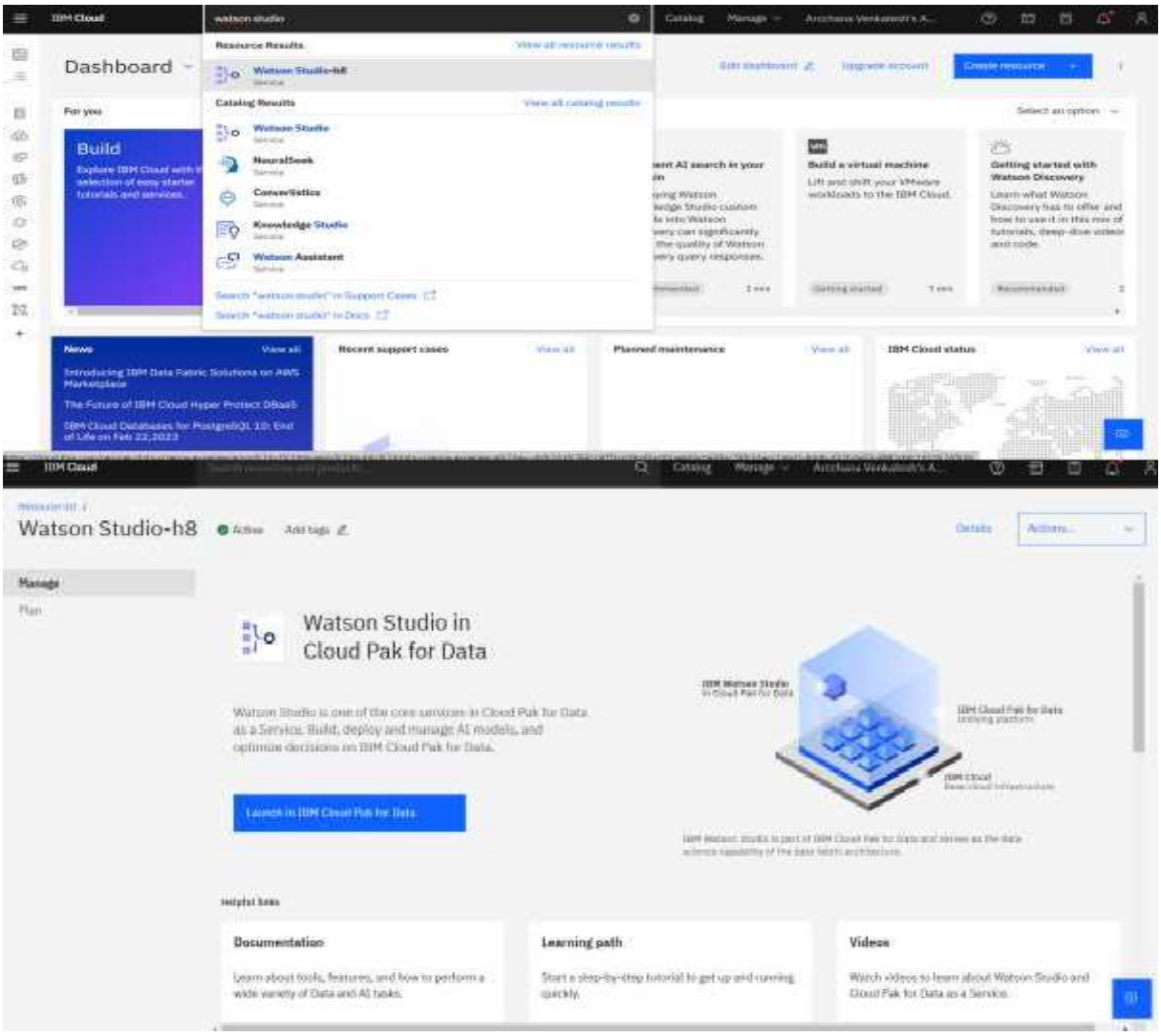

IBM Cloud
Catalog
Manage
Deepak D's Account
Watson Studio-wm
Active
Add tags
Details
Actions...
Manage
Plan
Watson Studio in
Cloud Pak for Data
Watson Studio is one of the core services in Cloud Pak for Data as a Service. Build, deploy and manage AI models, and optimize decisions on IBM Cloud Pak for Data.
Launch in IBM Cloud Pak for Data
Helpful links
Documentation
Learning path
Videos

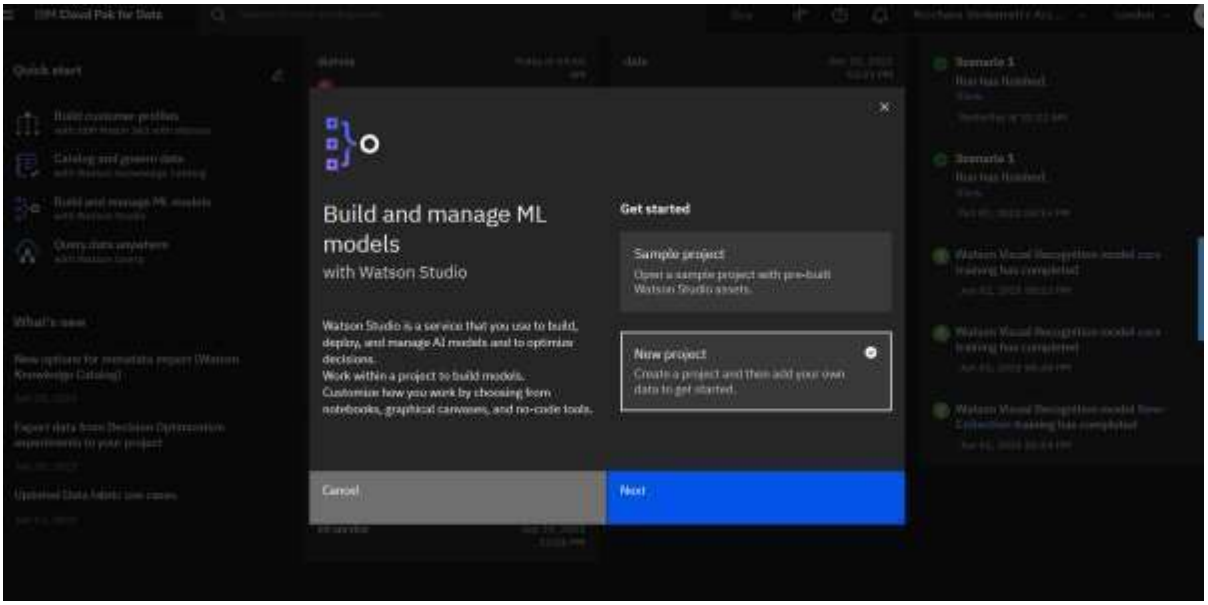

Welcome, Deepak!
Take a tutorial
Step through implementing a Data fabric use case in a sample project.
Work with data
Create a project for your team to prepare data, find insights, or build models.
Learn what's new
Stay current with new features, enhancements, and other changes.
Create new project
Quick start
Build customer profiles
Catalog and govern data
Build and manage ML models
Projects
EX-1
New in gallery
Notifications
No notifications
You will see your most recent notifications here.
Deployments
No deployment spaces
After you create spaces, you'll see them here.
New deployment space

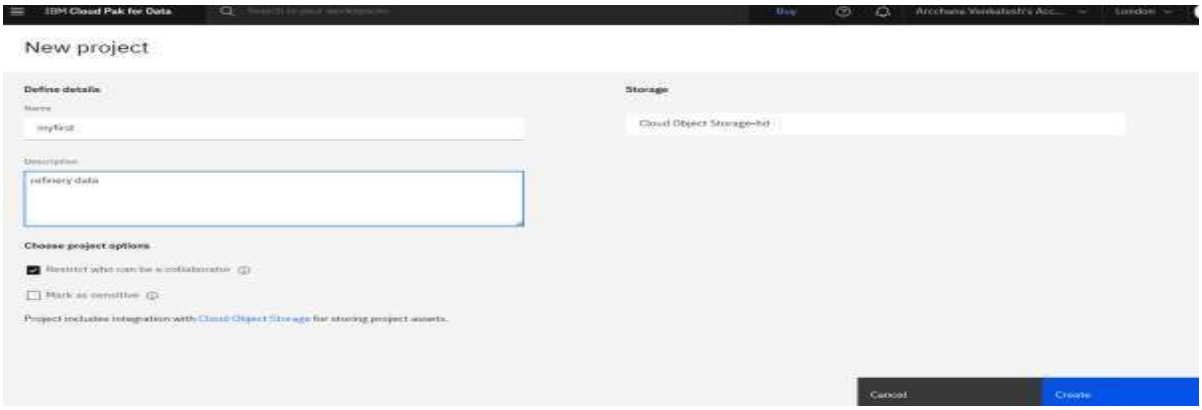

Back
Create a project
Choose whether to create an empty project or to preload your project with data and analytical assets. Add collaborators and data, and then choose the right tools to accomplish your goals. Add services as necessary.
1010
0101
Create an empty project
Add the data you want to prepare, analyze, or model. Choose tools based on how you want to work: write code, create a flow on a graphical canvas, or automatically build models.
USE TO
Prepare and visualize data
Analyze data in notebooks
Train models
Create a project from a sample or file
Get started fast by loading existing assets. Choose a project file from your system, or choose a curated sample project.
USE TO
Learn by example
Build on existing work
Run tutorials


IBM Watson Studio
New project
Define details
Name
lab3
Description
Project description
Choose project options
Restrict who can be a collaborator
Mark as sensitive
Project includes integration with Cloud Object Storage for storing project assets.
Define storage
1 Select storage service
Add
2 Refresh
Cancel

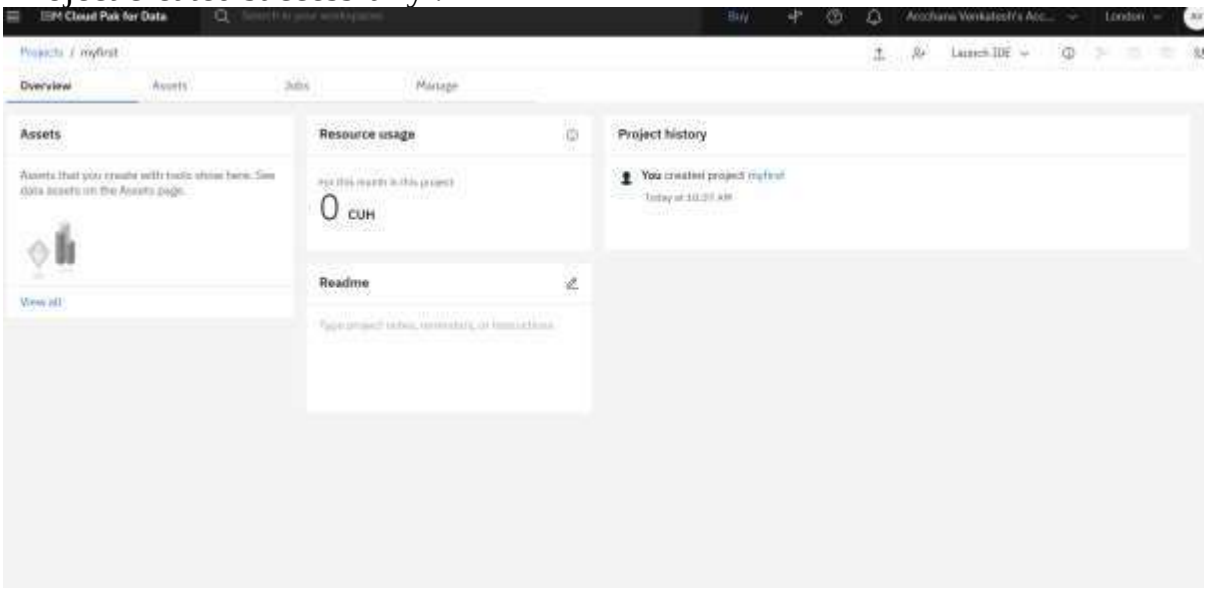

IBM Watson Studio
Cloud Object Storage
Create
Lite
Lite plan instance is free to use for Storage capacity up to 25 GB per month.
Lite plan instance is used for trial, and can be easily upgraded to Standard plan for unlimited scalability and full functionality.
Free
Lite plan services are deleted after 30 days of inactivity.
Standard
Summary
Cloud Object Storage
Create
IBM Watson Studio
Launch IDE
Overview
Assets
Jobs
Manage
Assets
Assets that you create with tools show here.
See data assets on the Assets page.
View all
Resource usage
0 CUH
Project history
You created project lab3
Readme


IBM Cloud
watson studio
Dashboard
Resource Results
Watson Studio-h8
Catalog Results
Watson Studio
NeuralSeek
Knowledge Studio
Watson Assistant
Build
Build a virtual machine
Getting started with Watson Discovery
News
Recent support cases
Planned maintenance
IBM Cloud status
Watson Studio-h8
Active
Manage
Plan
Watson Studio in Cloud Pak for Data
Launch in IBM Cloud Pak for Data
Documentation
Learning path
Videos

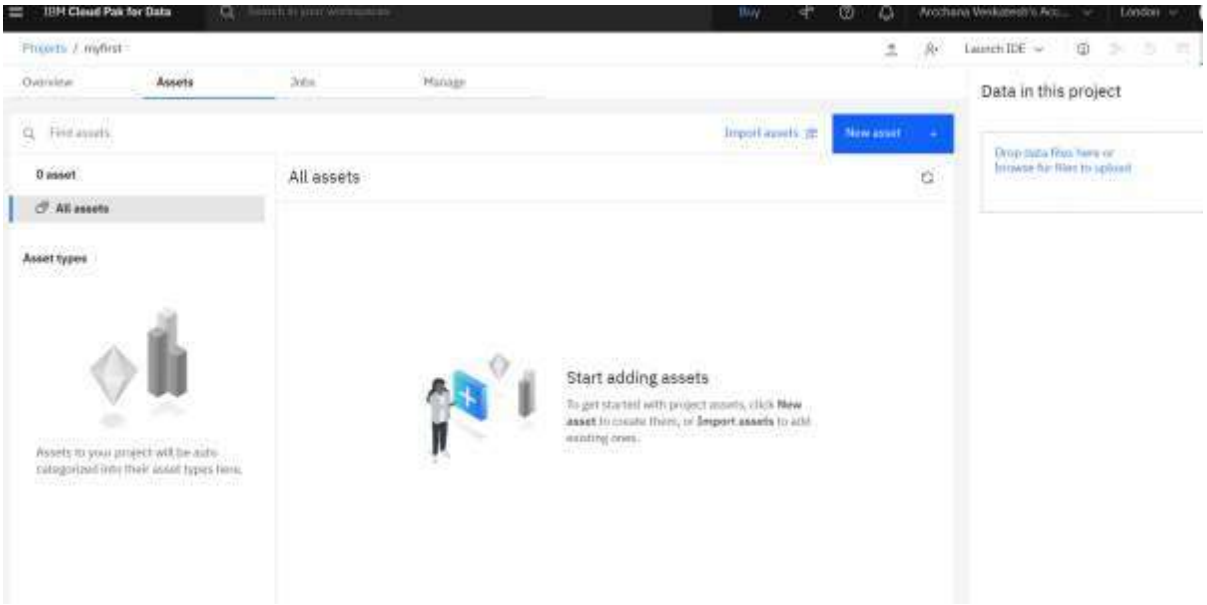

Build and manage ML models
with Watson Studio
Get started
Sample project
New project
Cancel
Next



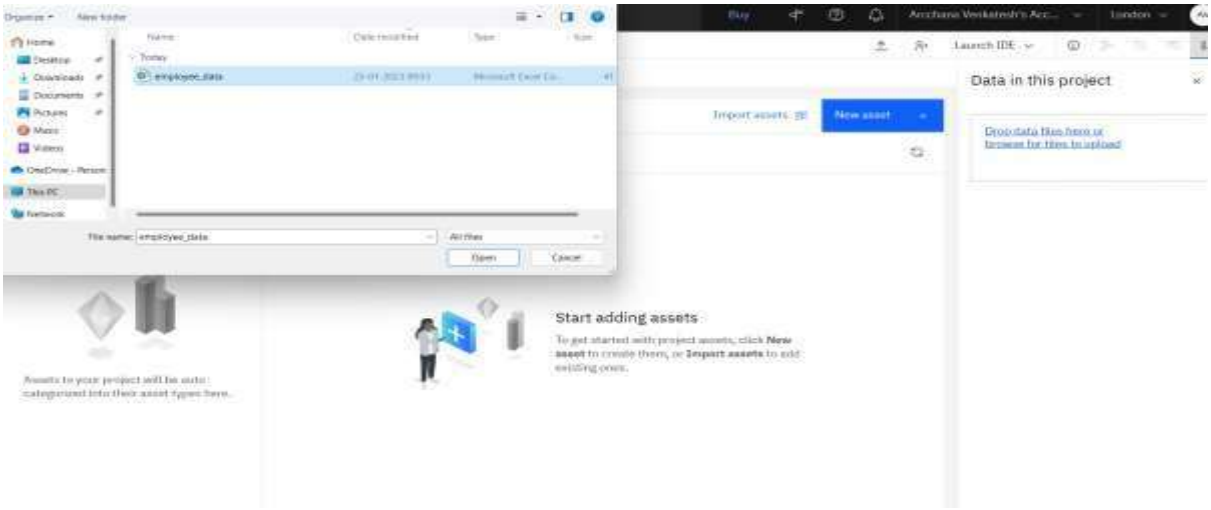
Project created successfully .



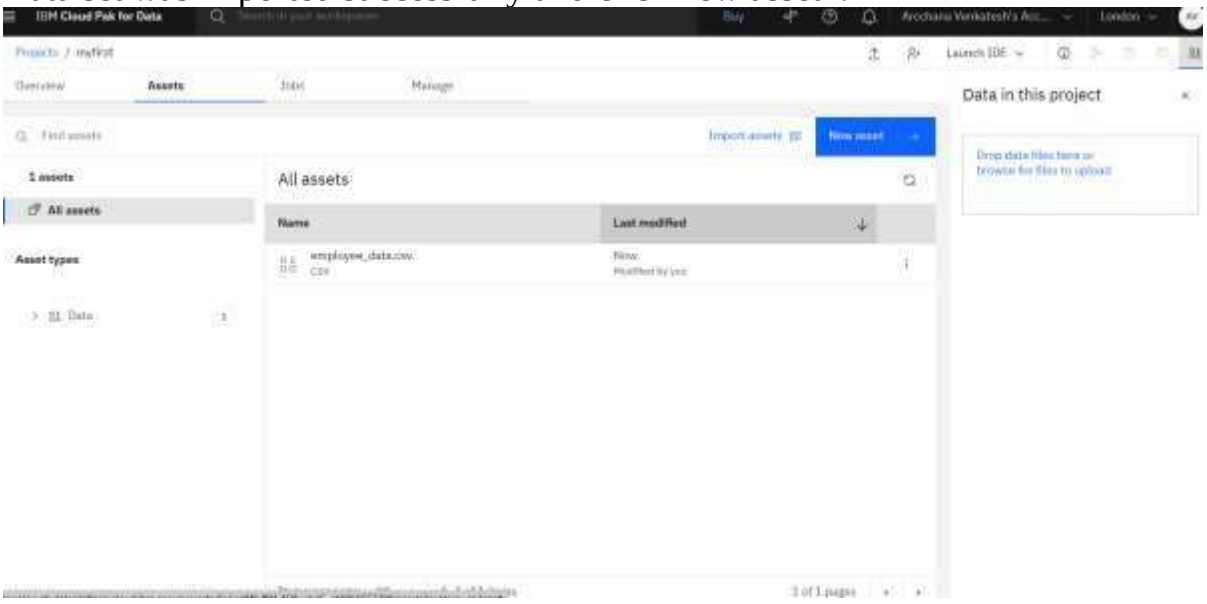
Click assets and inset data set



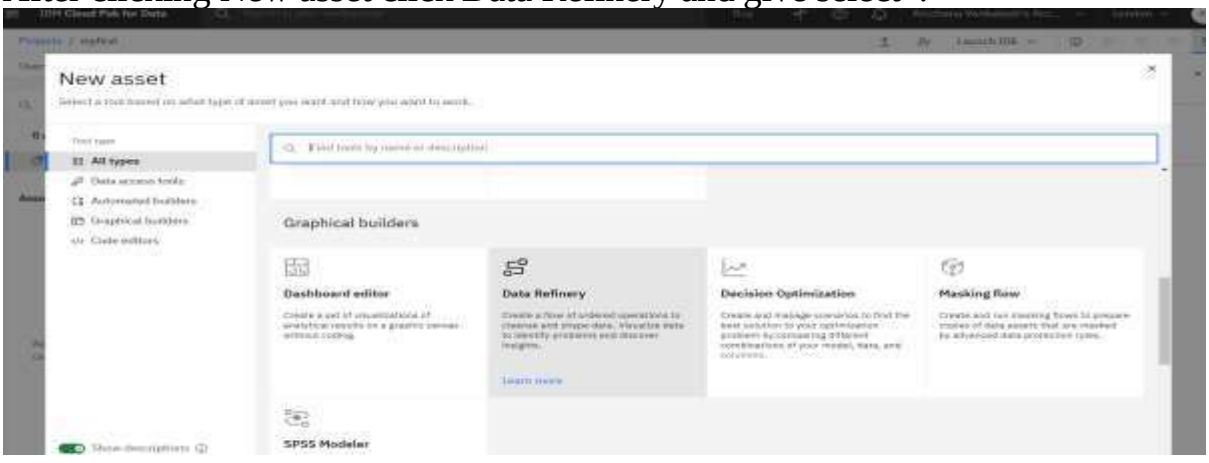
import dataset -Employee_data with the help of link

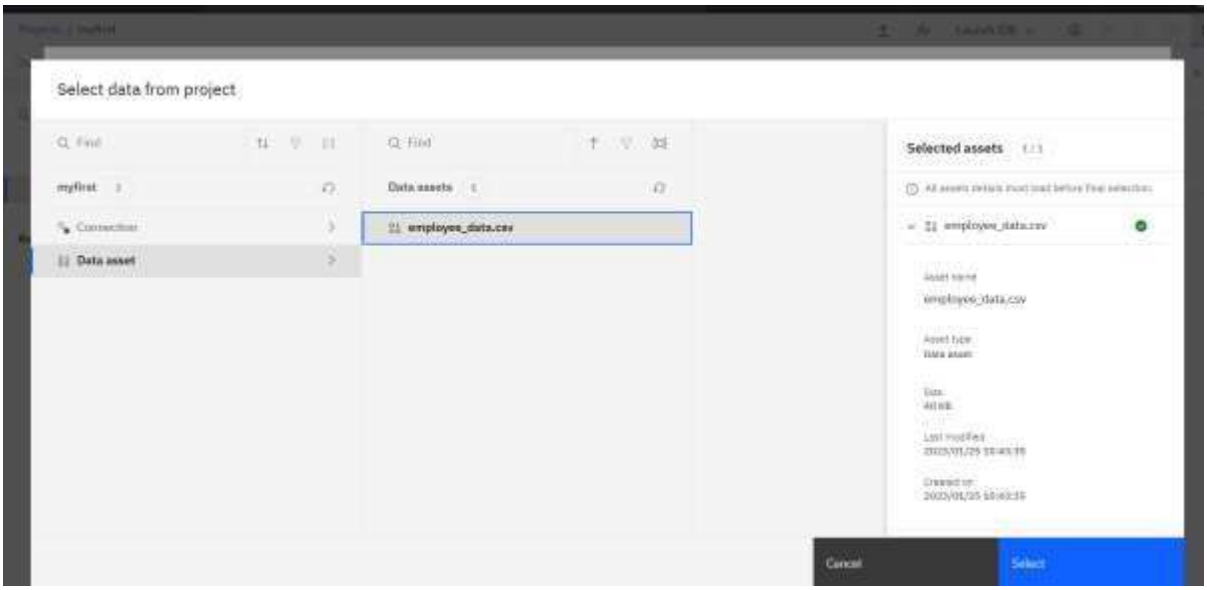


Data set was imported successfully and click new asset .

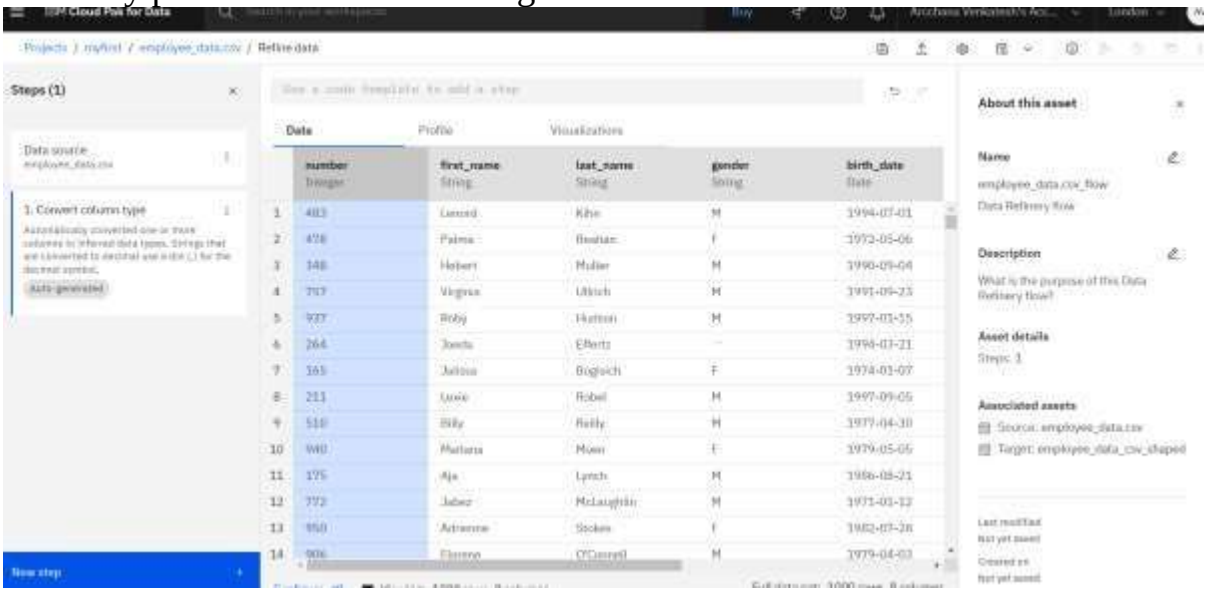


After clicking New asset click Data Refinery and give select .

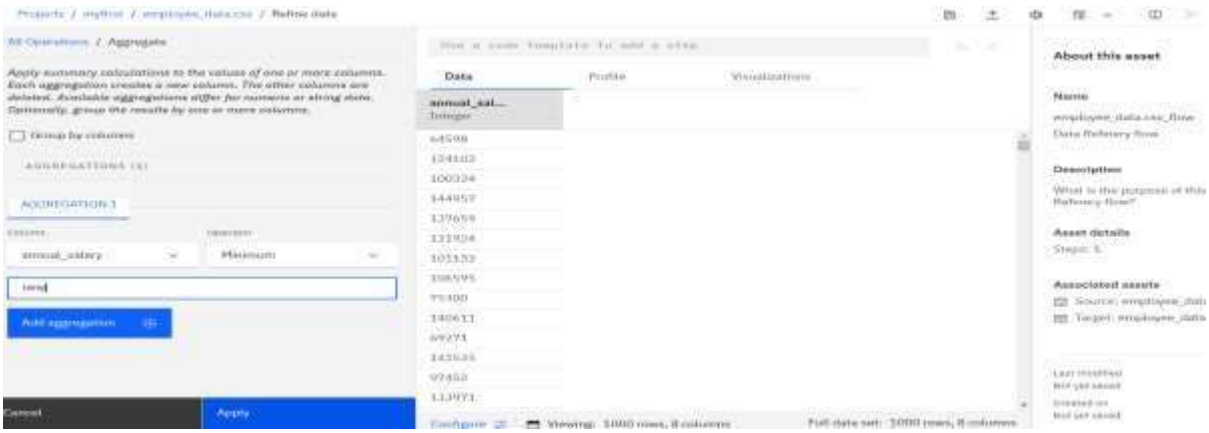




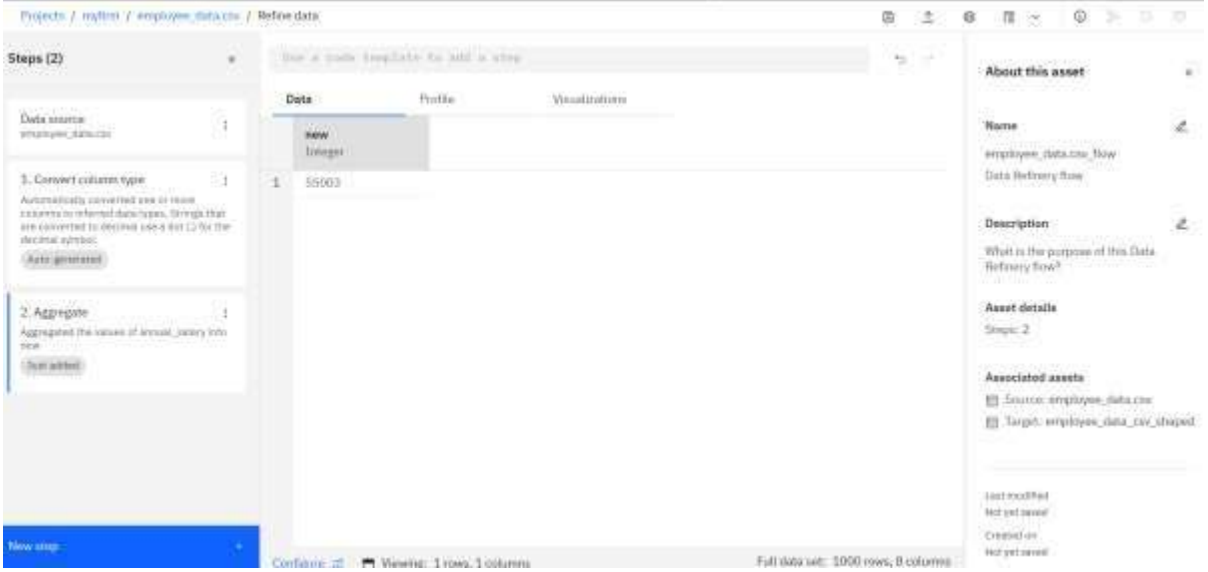
Refinery process will start working.



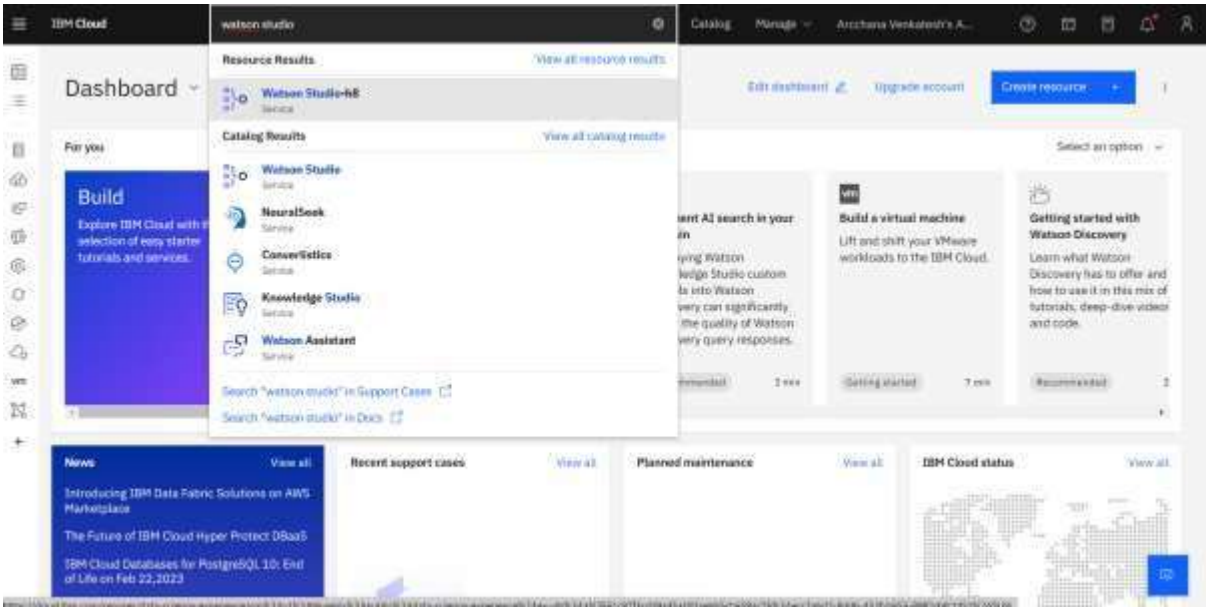
Click on a new step and aggricate and select the column and operation and create a new column name and give Apply .



After Aggregating new data was refined successfully.

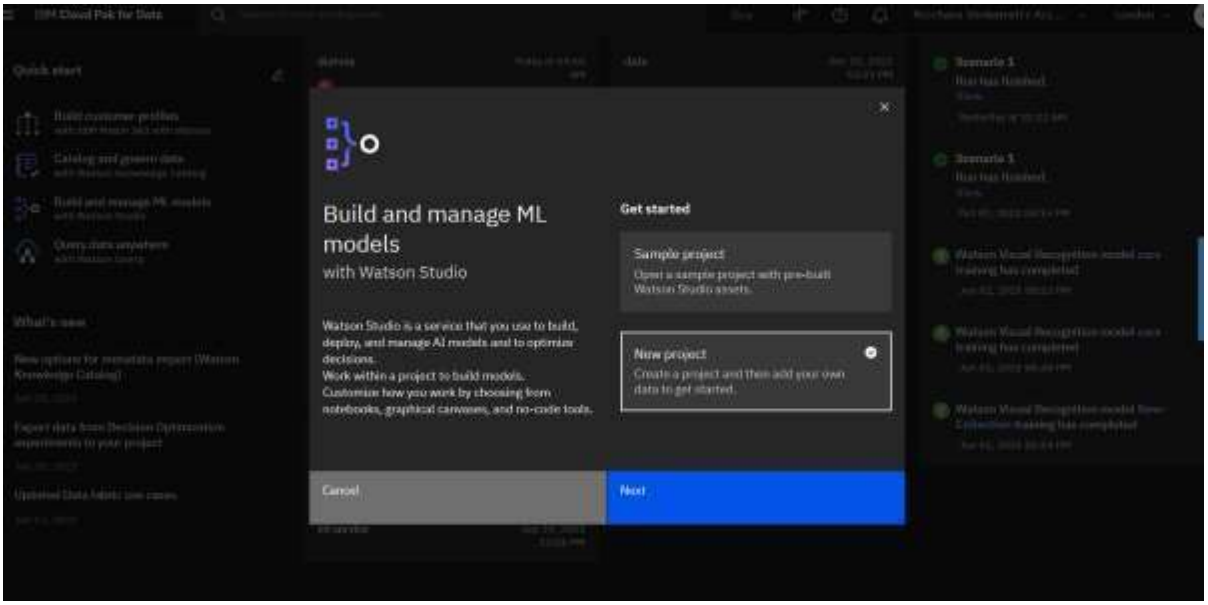


click Watson studio h8 .

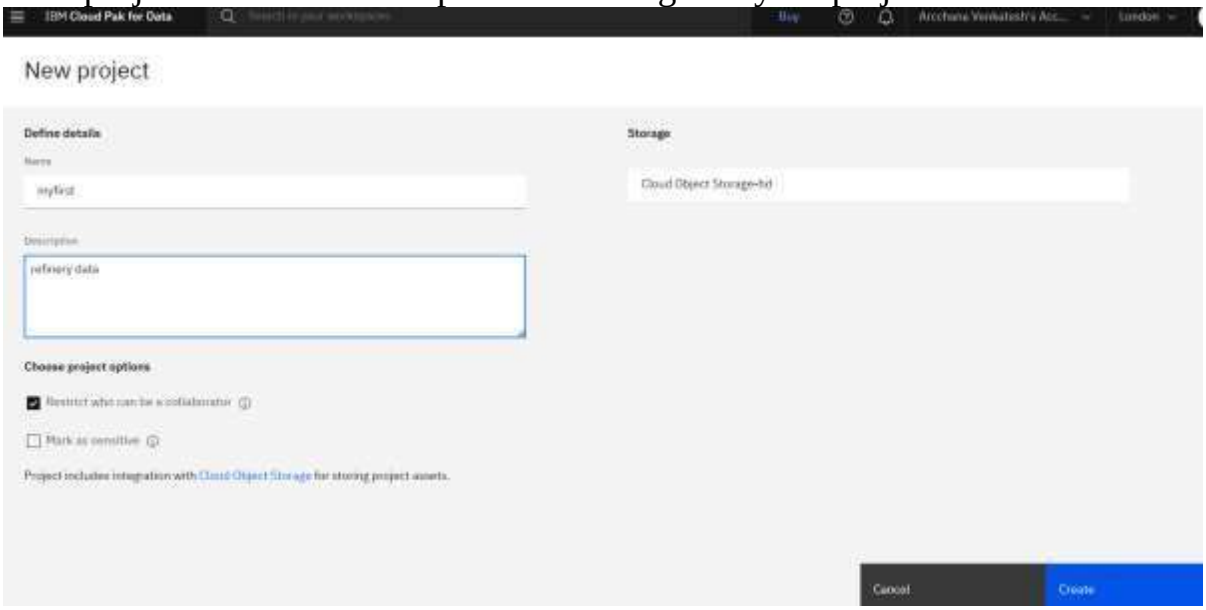


click launch in IBM cloud park data

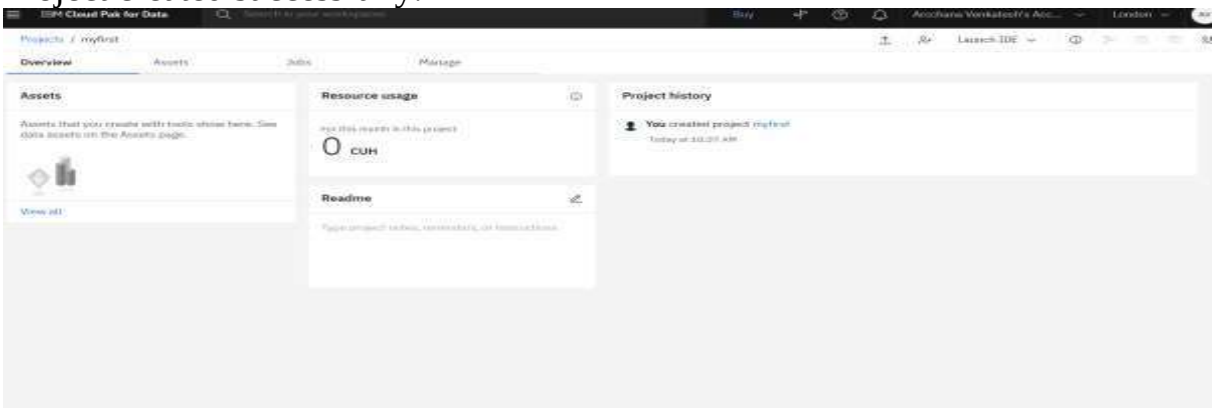
create new project and give next



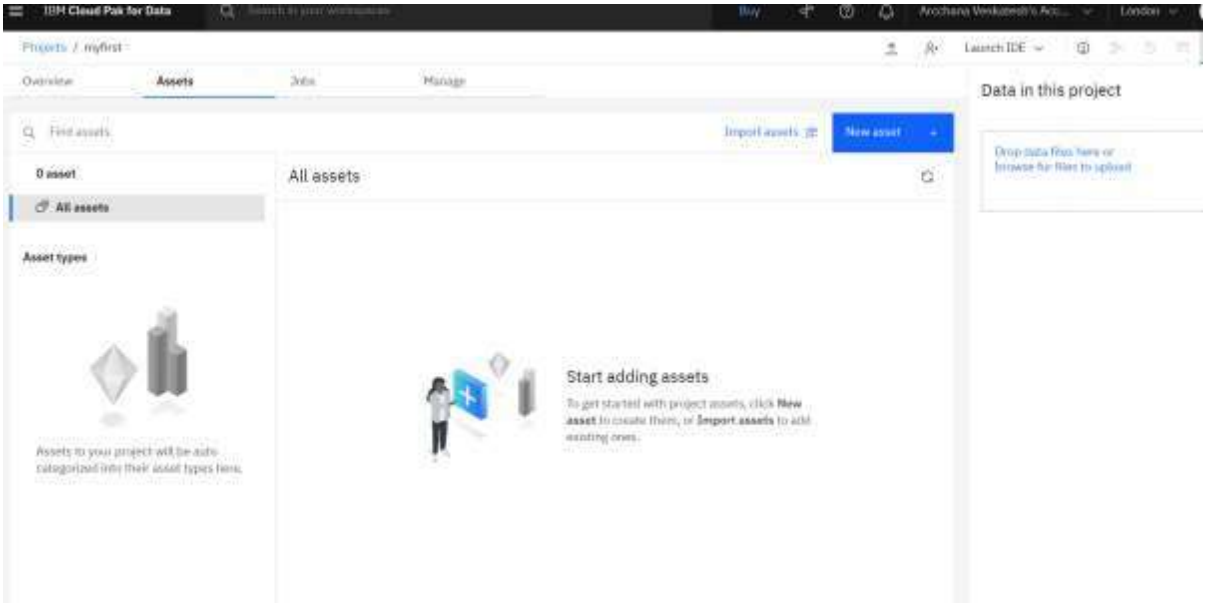
Give project name and description and storage for your project and click create



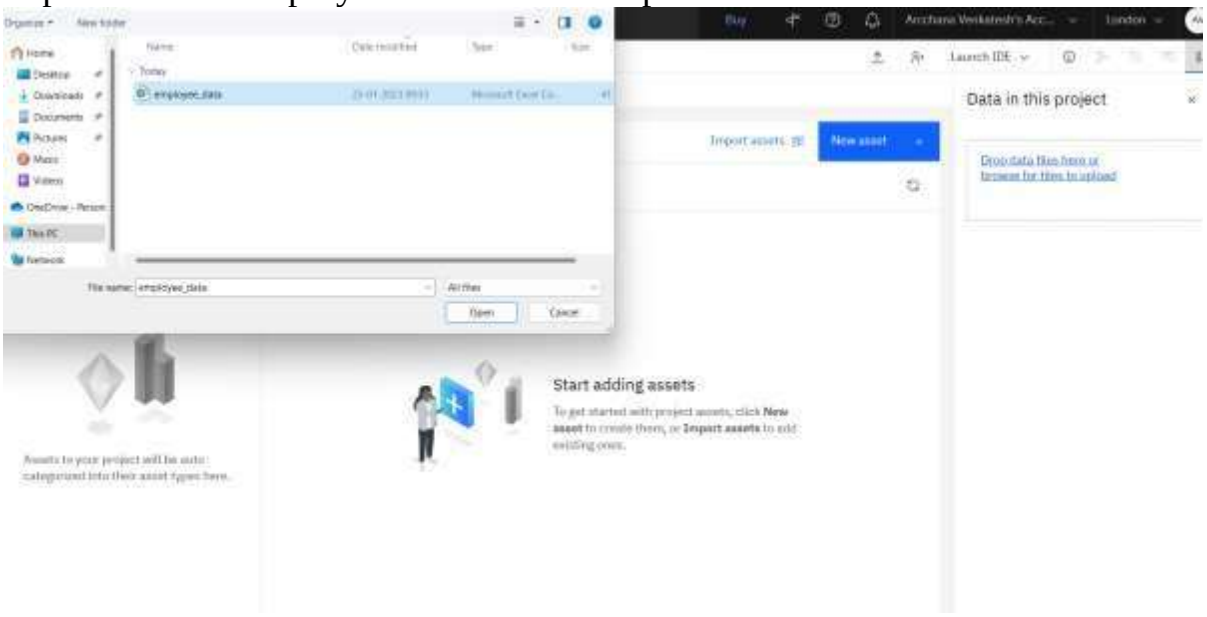
Project created successfully.



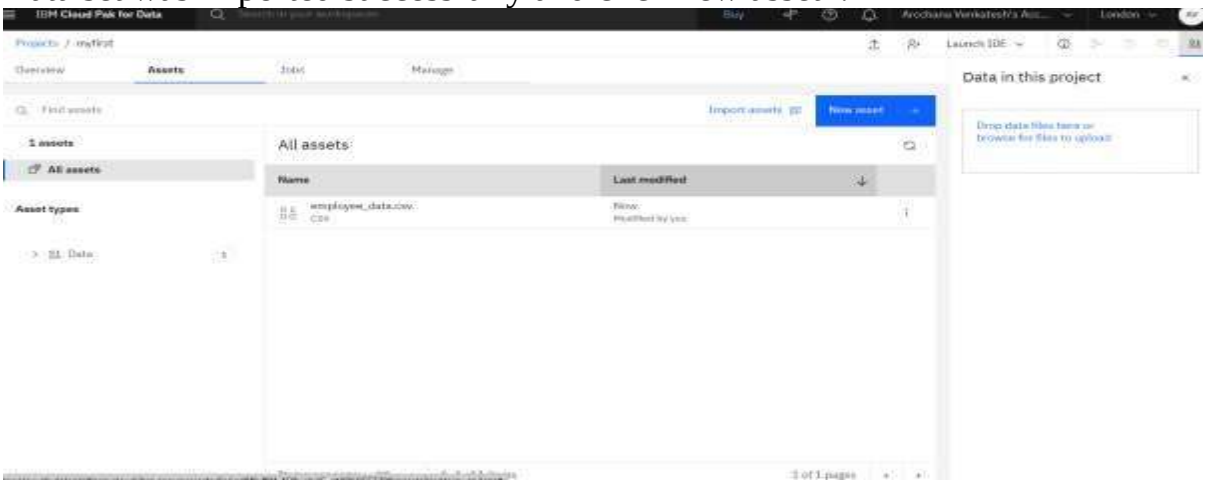
Click assets and inset data set



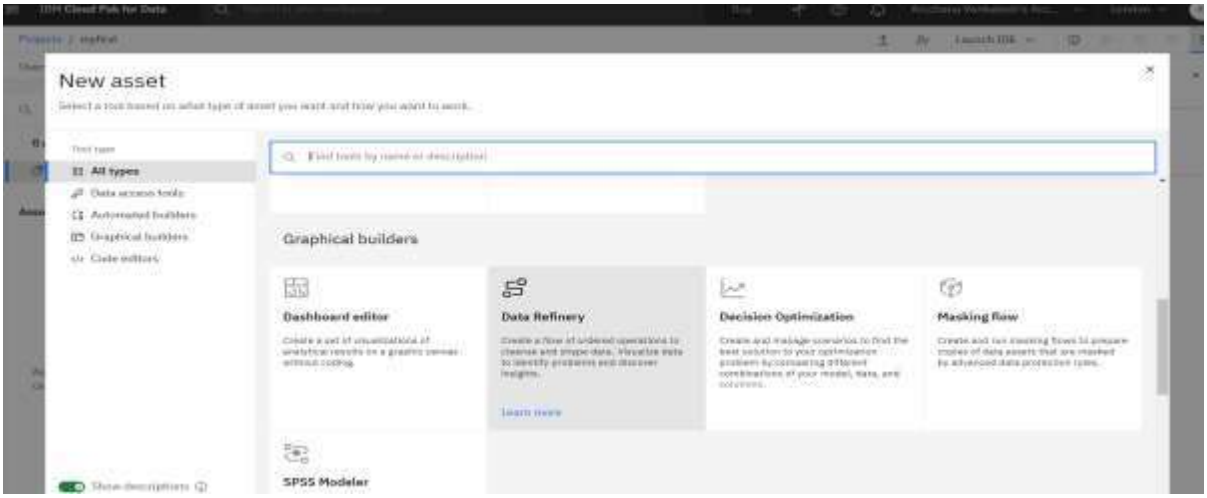
import dataset -Employee data with the help of link

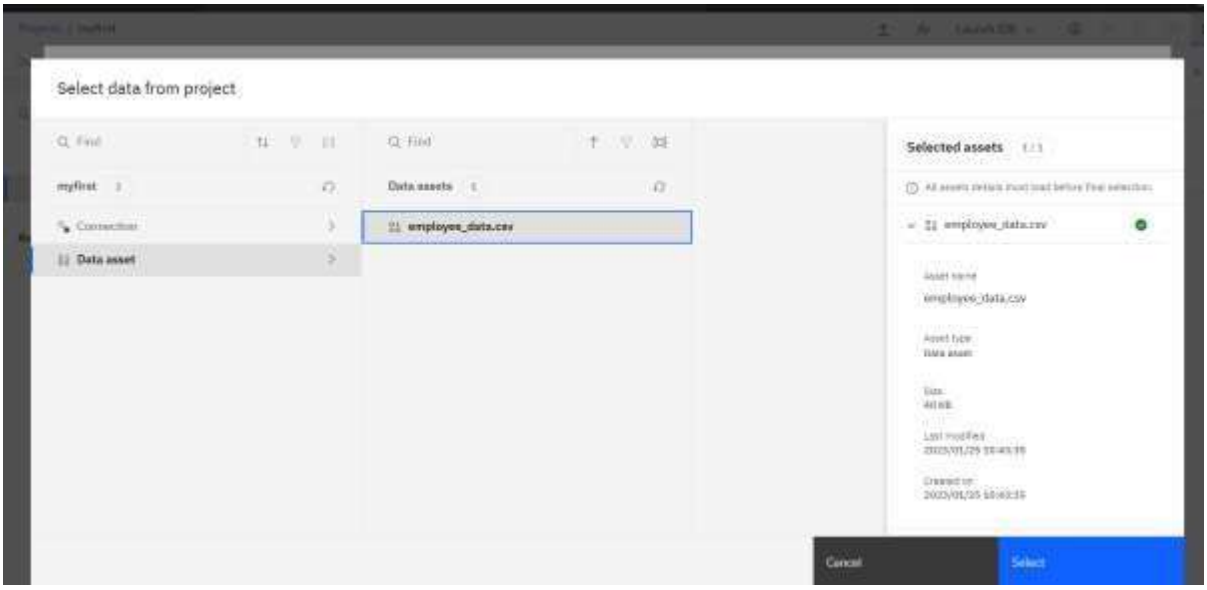


Data set was imported successfully and click new asset .

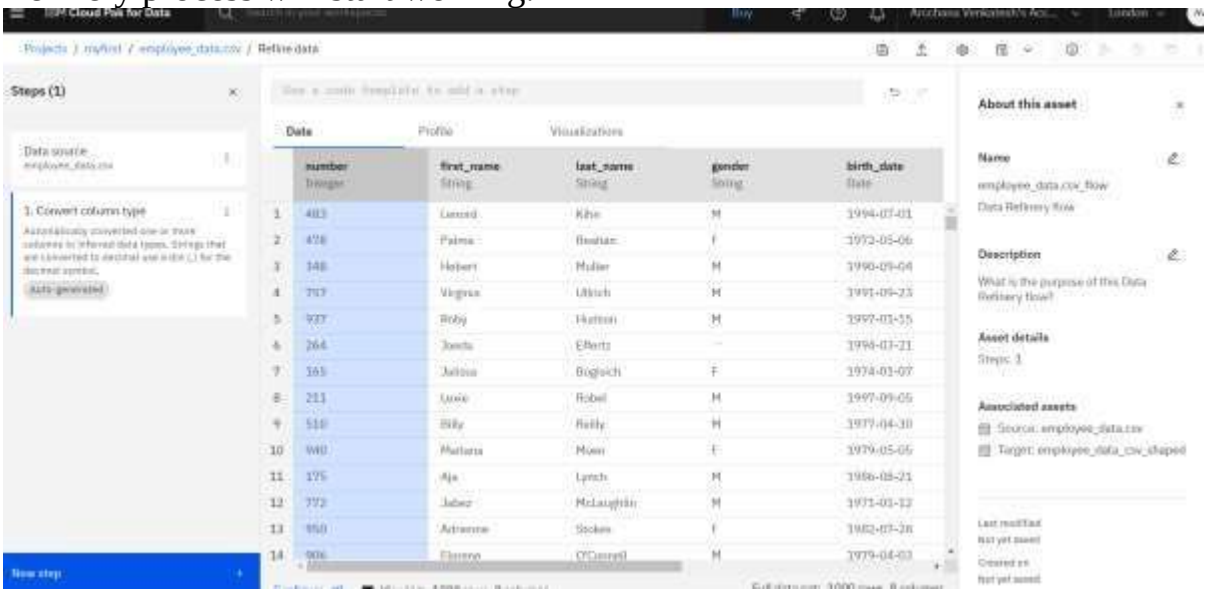


After clicking New asset click Data Refinery and give select .



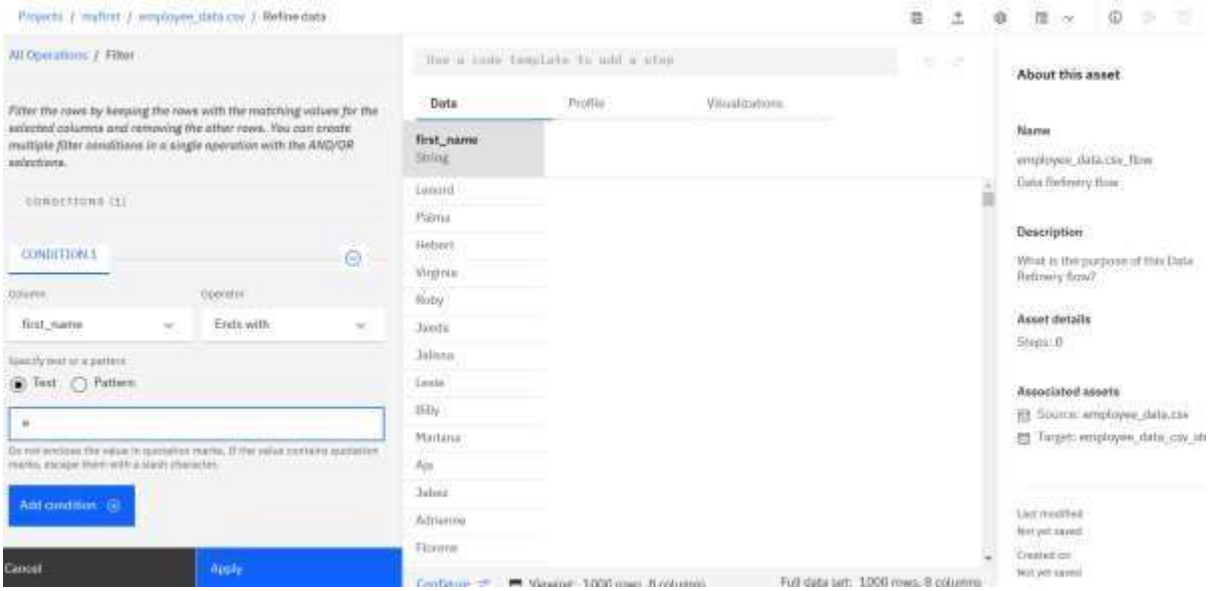


Refinery process will start working.

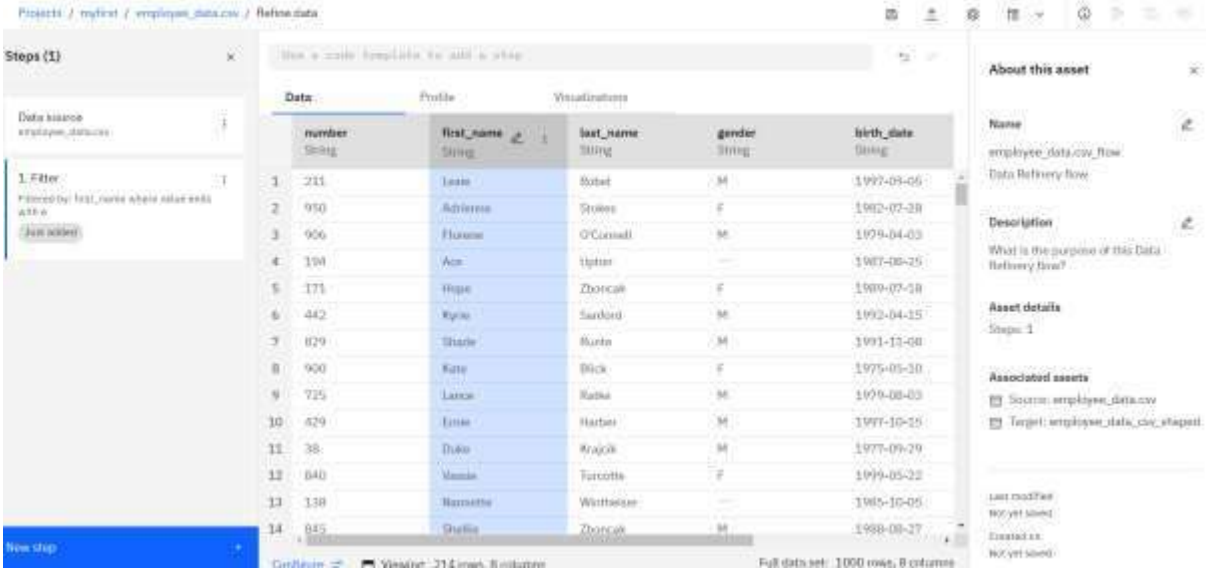


Click on a new step and do filtering, select column name as first name and select the

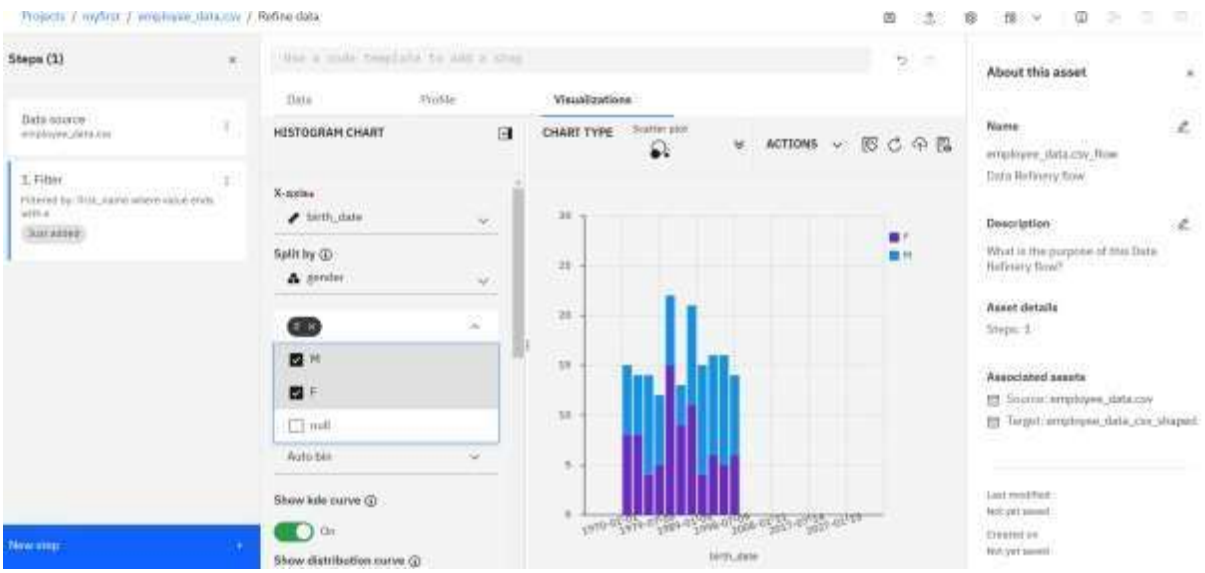
operation ends with and select text and give “e” and click Apply.



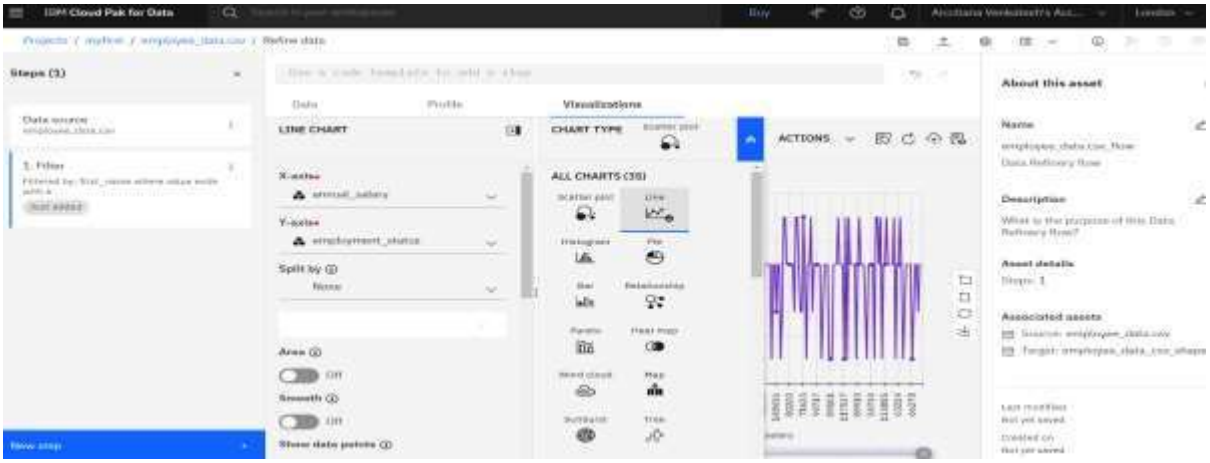
first name was end up with “e” successfully and click visualization.



Give x axis and split by and visualization will exist successfully.



Select the chart as per requirement and do visualization in Watson studio.



Result:

The experiment results in a refined and structured dataset with cleaned and transformed data. Visualizations such as charts and graphs help identify trends and patterns. This ensures accurate analysis and better decision-making.

EXP NO	Deployment of VMs in Azure	DATE
10		

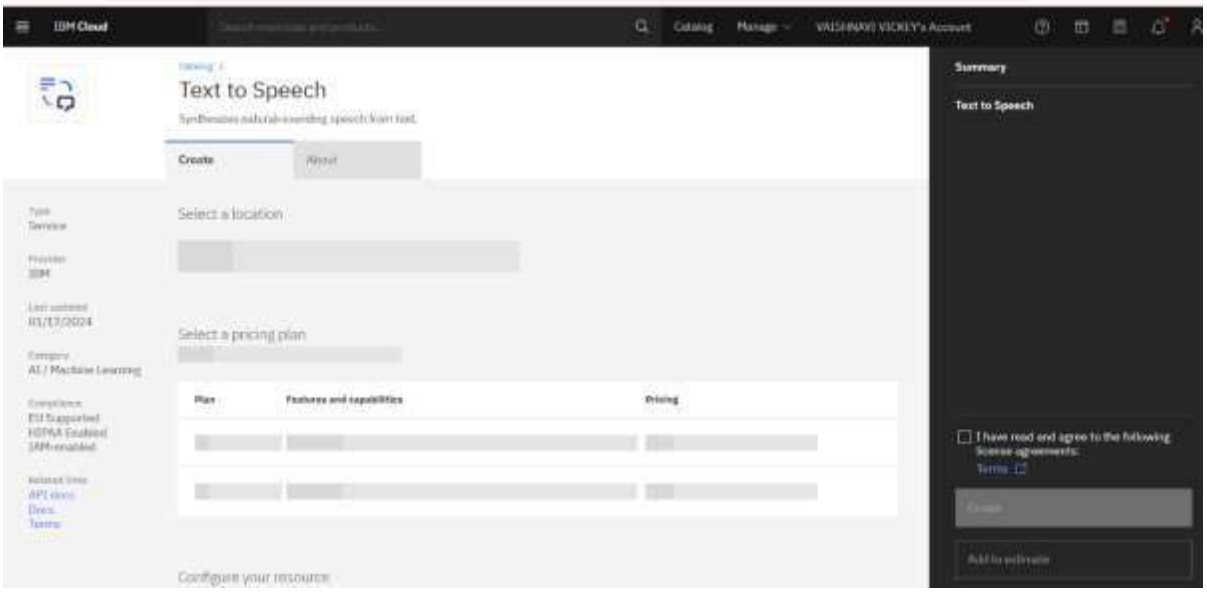
Aim

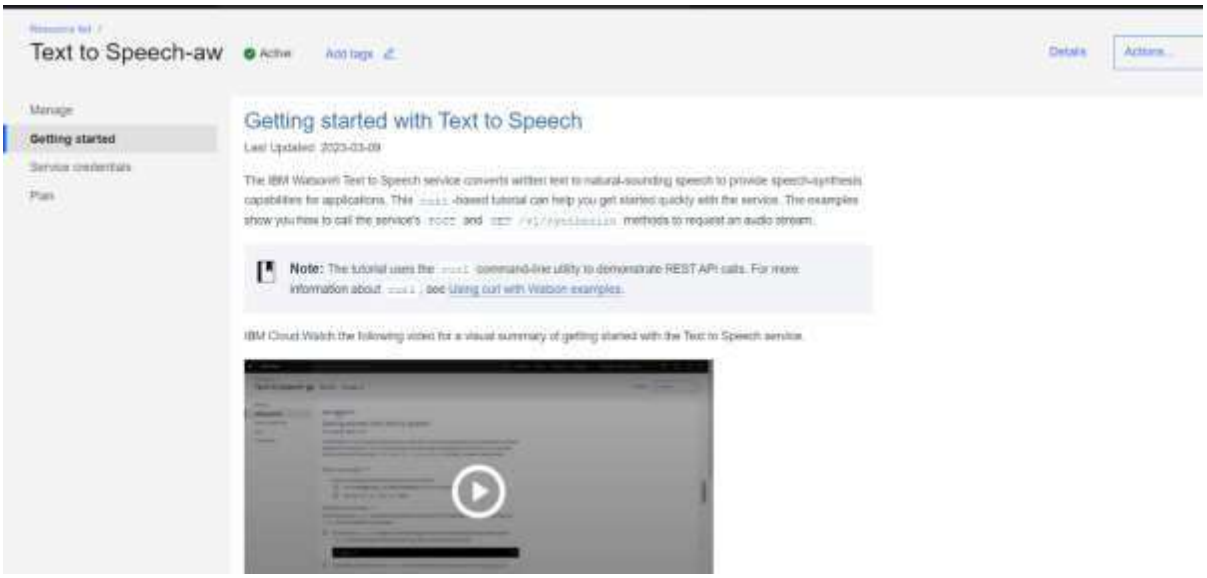
The aim of Text-to-Speech (TTS) in cloud computing is to convert text into natural-sounding speech using AI-based cloud services. It enhances accessibility, user interaction, and automation in various applications. TTS technology is widely used in virtual assistants, audiobooks, and customer service systems.

Algorithm for Text-to-Speech (TTS) in Cloud Computing

1. **Input Text Acquisition:** Accept user-provided text or extract it from a document.
2. **Preprocessing:** Clean the text by removing special characters and unnecessary spaces.
3. **Tokenization:** Break the text into individual words and sentences.
4. **Linguistic Analysis:** Identify parts of speech and sentence structure.
5. **Phonetic Conversion:** Convert words into phonemes for pronunciation.
6. **Prosody Analysis:** Adjust intonation, stress, and rhythm to make speech natural.
7. **Voice Model Selection:** Choose a voice type (gender, language, accent).
8. **Speech Synthesis:** Use cloud TTS API to generate speech from processed text.
9. **Audio Format Conversion:** Convert speech output into desired format (MP3, WAV, etc.).
10. **Output Delivery:** Stream or store the generated speech for playback.

Text To speech





Before you begin

IBM Cloud

IBM Cloud

Tip: This tutorial uses an API key to authenticate. In production, use an IAM token. For more information see Authenticating to IBM Cloud.

IBM Cloud Pak for Data

IBM Cloud Pak for Data

The Text to Speech for IBM Cloud Pak for Data must be installed and configured before beginning this tutorial. For more information, see Watson Speech services on Cloud Pak for Data.

1. Create an instance of the service by using the web client, the API, or the command-line interface. For more information about creating a service instance, see Creating a Watson Speech services instance.
2. Follow the instructions in *Creating a Watson Speech services instance* to obtain a Bearer token for the instance. This tutorial uses a Bearer token to authenticate to the service.

Synthesize text in US English

The following command use the `POST /v1/synthesize` method to synthesize US English input to audio. The request uses the voice `en-US_MichaelV3Voice`. It produces audio in the WAV format.

Tip: You can use a browser or other tools to play the audio files that are produced by the examples in this tutorial. For more information, see Playing an audio file.

Tip: You can use a browser or other tools to play the audio files that are produced by the examples in this tutorial. For more information, see Playing an audio file.

1. Issue the following command to synthesize the string "hello world". The request produces a WAV file that is named `hello_world.wav`.

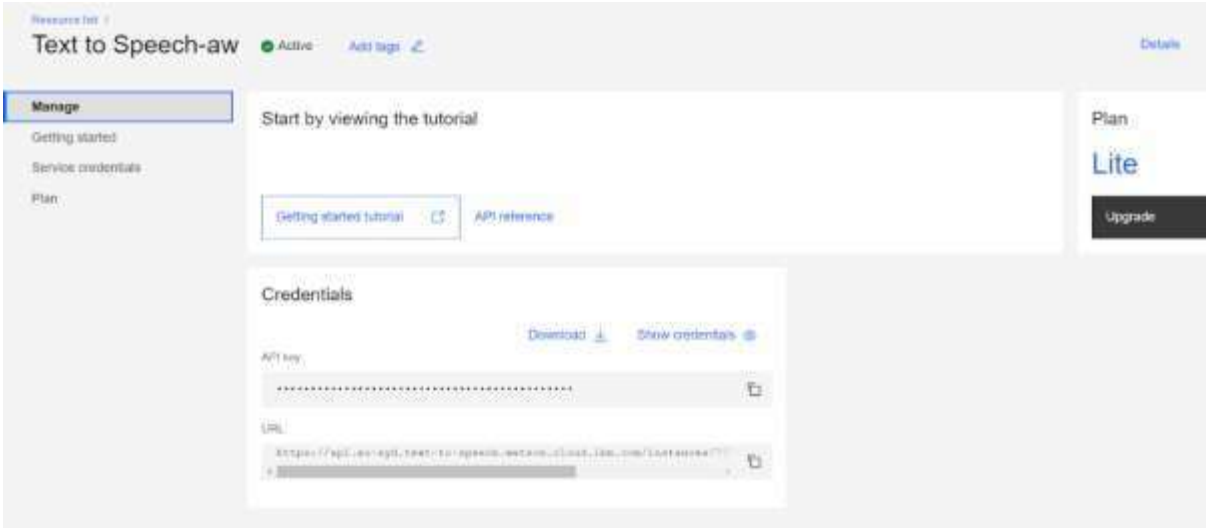
IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--header "Accept: audio/wav" \
--data "{\"text\":\"hello world\"}" \
--output hello_world.wav \
"{url}/v1/synthesize?voice=en-US_MichaelV3Voice"
```

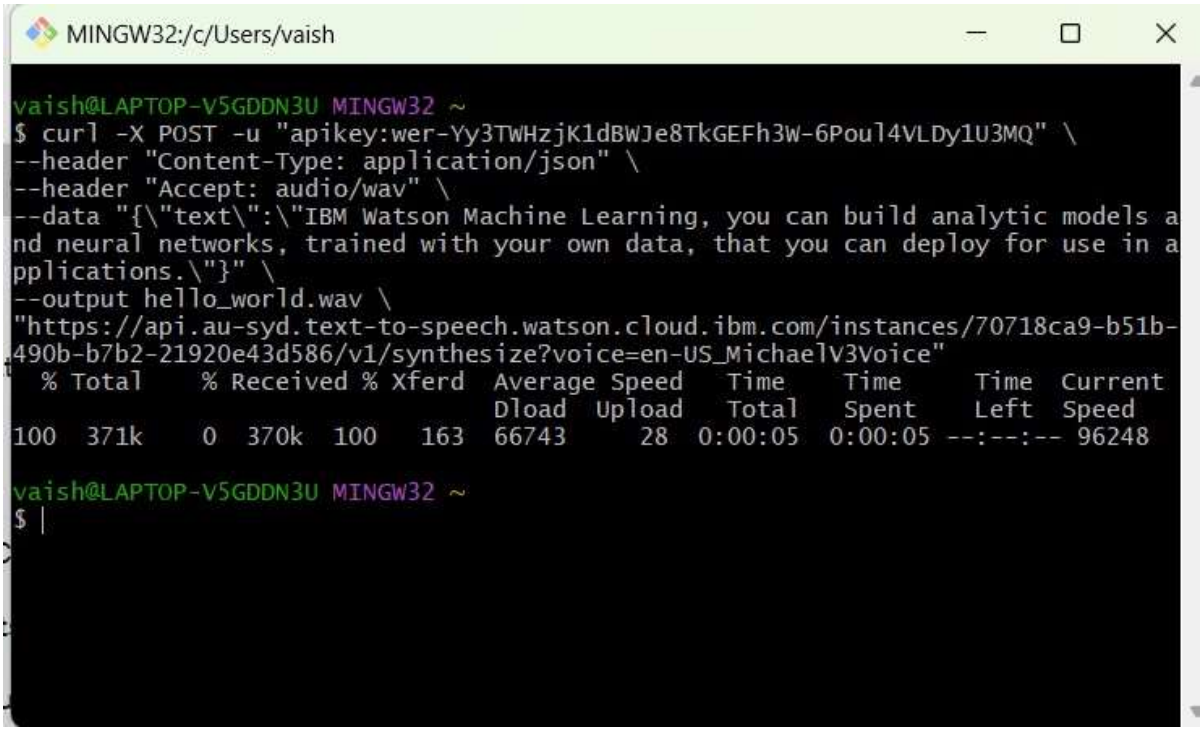
IBM Cloud Pak for Data

- Replace `{token}` and `{url}` with the access token and URL for your service instance.

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--header "Accept: audio/wav" \
--data "{\"text\":\"hello world\"}" \
--output hello_world.wav \
"{url}/v1/synthesize?voice=en-US_MichaelV3Voice"
```

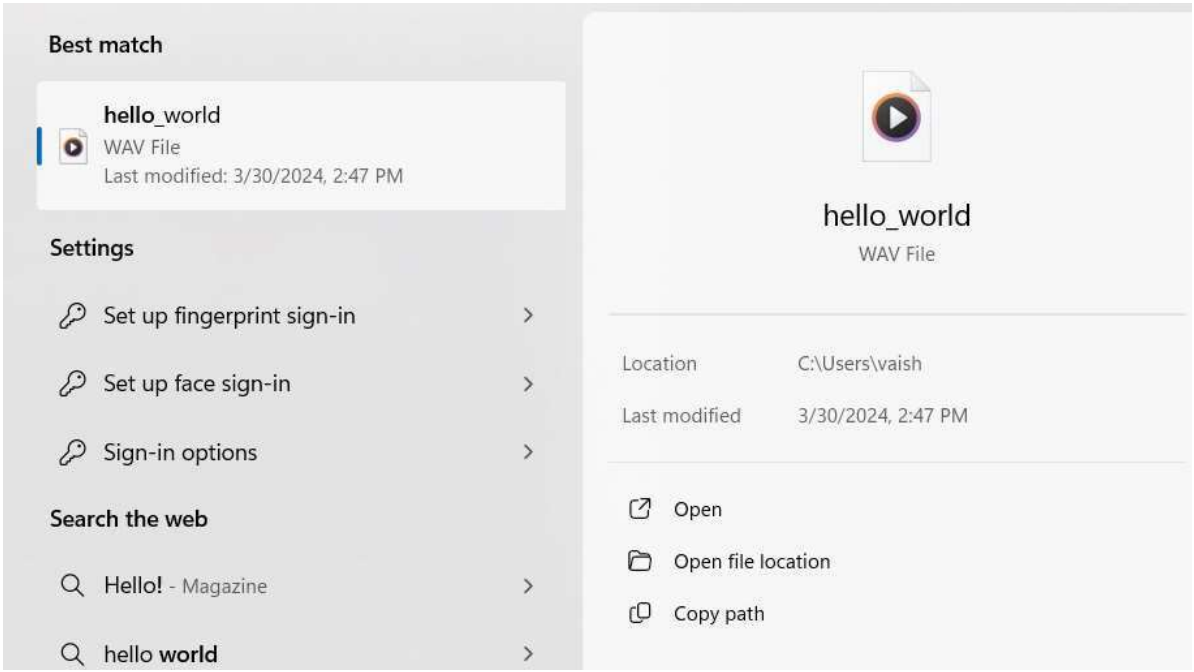

Text to Speech-aw
Active
Add tags
Details
Manage
Getting started
Service credentials
Plan
Start by viewing the tutorial
Getting started tutorial
API reference
Plan
Lite
Upgrade
Credentials
Download
Show credentials
API key
URL

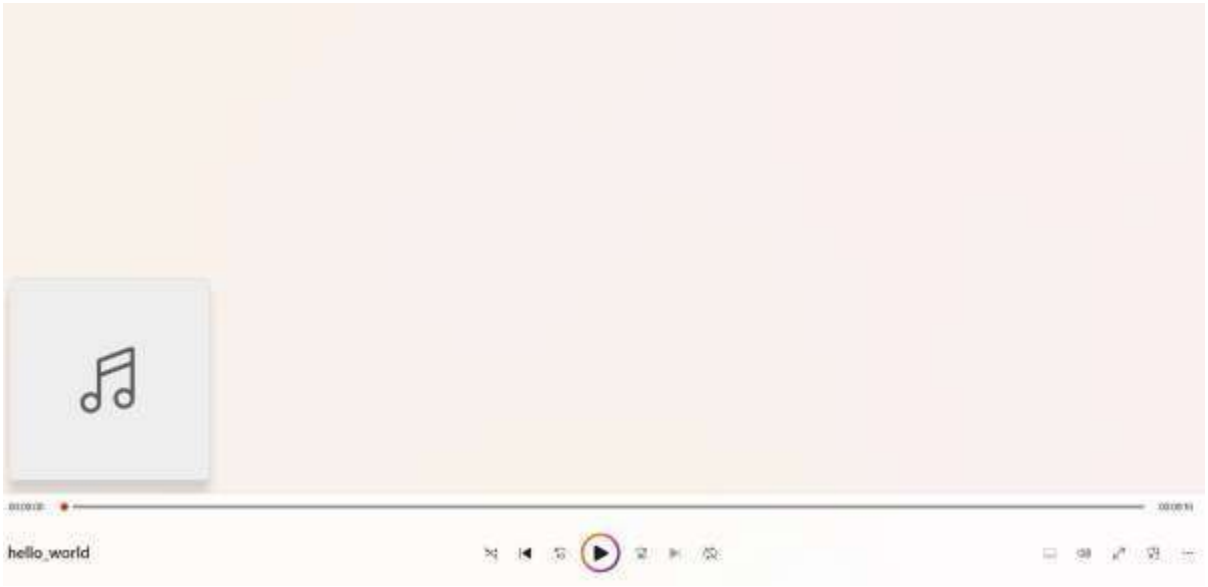

Output
let
const
Numt
data
script.js
Data v
styles
File
Edit
View
curl -X POST -u "apikey:wer-Yy3TWHzjK1dBWJe8TkGEFh3W-6Poul4VLDy1U3MQ" \
--header "Content-Type: application/json" \
--header "Accept: audio/wav" \
--data "{\"text\":\"hello world\"}" \
--output hello_world.wav \
"{url}/v1/synthesize?voice=en-US_MichaelV3Voice"


MINGW32:/c/Users/vaish
vaish@LAPTOP-V5GDDN3U MINGW32 ~
$ curl -X POST -u "apikey:wer-Yy3TWHzjK1dBWJe8TkGEFh3W-6Poul4VLDy1U3MQ" \
--header "Content-Type: application/json" \
--header "Accept: audio/wav" \
--data "{\"text\":\"IBM Watson Machine Learning, you can build analytic models a
nd neural networks, trained with your own data, that you can deploy for use in a
pplications.\"}" \
--output hello_world.wav \
"https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/70718ca9-b51b-
490b-b7b2-21920e43d586/v1/synthesize?voice=en-US_MichaelV3Voice"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 371k 0 370k 100 163 66743 28 0:00:05 0:00:05 --:--:-- 96248
vaish@LAPTOP-V5GDDN3U MINGW32 ~
$

```
Git Command :
curl -X POST -u "apikey:wer-Yy3TWHzjK1d8WJe8TkGEFh3W-6Poul4VLDy1U3MQ" \
--header "Content-Type: application/json" \
--header "Accept: audio/wav" \
--data "{\"text\":\"IBM Watson Machine Learning, you can build analytic models and neural networks, trained with your own data, that you can
deploy for use in applications.\"}" \
--output hello_world.wav \
"https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/70718ca9-b51b-490b-b7b2-21920e43d586/v1/synthesize?voice=en-
US_MichaelV3Voice"

Command PROMPT :

curl -X POST -u "apikey:wer-Yy3TWHzjK1d8WJe8TkGEFh3W-6Poul4VLDy1U3MQ" ^
More? --header "Content-Type: application/json" ^
More? --header "Accept: audio/wav" ^
More? --data "{\"text\":\"IBM Watson Machine Learning, you can build analytic models and neural networks, trained with your own data, that
you can deploy for use in applications.\"}" ^
More? --output hello_world.wav ^
More? "https://api.au-syd.text-to-speech.watson.cloud.ibm.com/instances/70718ca9-b51b-490b-b7b2-21920e43d586/v1/synthesize?voice=en-
US_MichaelV3Voice"
```

Output :





Result

The cloud-based TTS system successfully converts text into human-like speech with natural pronunciation. It provides scalable, multi-language, and real-time speech synthesis. This improves accessibility, automation, and user experience in various applications.