ZOMATO

RATING PREDICTION

CREATED BY : RAJAT SINGH

ABOUT PROJECT

This project predicts restaurant ratings using the **Zomato Bengaluru Dataset** from Kaggle. The project implements a robust **MLOps Training Pipeline** in Python 3.12, designed to be modular, scalable, and production-ready.This app predicts restaurant ratings based on user inputs, such as location, cuisine type, and more. Below is an overview of the app's functionality with visual representations of each feature.

WHAT TO DO

01

The main goal of this project is to perform extensive Exploratory Data Analysis(EDA) on the Zomato Dataset and build an appropriate Machine Learning Model that will help various Zomato Restaurants to predict their respective Ratings based on certain features

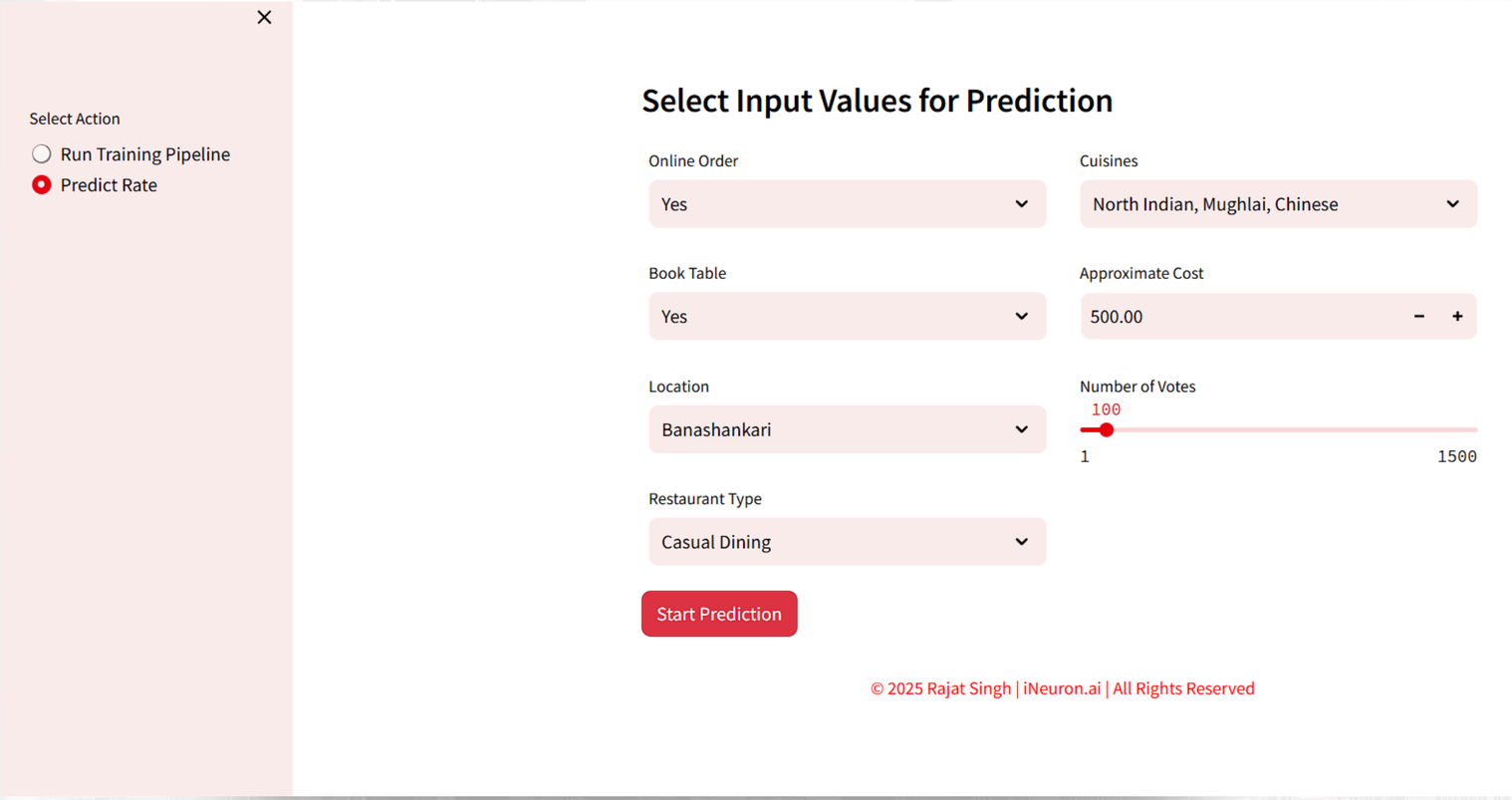
02

Approach: The classical machine learning tasks like Data Exploration, Data Cleaning,
Feature Engineering, Model Building and Model Testing. Try out different machine learning algorithms that's best fit for the above case.

03

Results: You have to build a solution that should able to predict the ratings of the restaurants listed in the dataset.

Prediction Inputs



After entering the required details, click the **Predict Rate** button to get the predicted restaurant rating. The prediction result is displayed clearly, providing actionable insights.

ABOUT APP

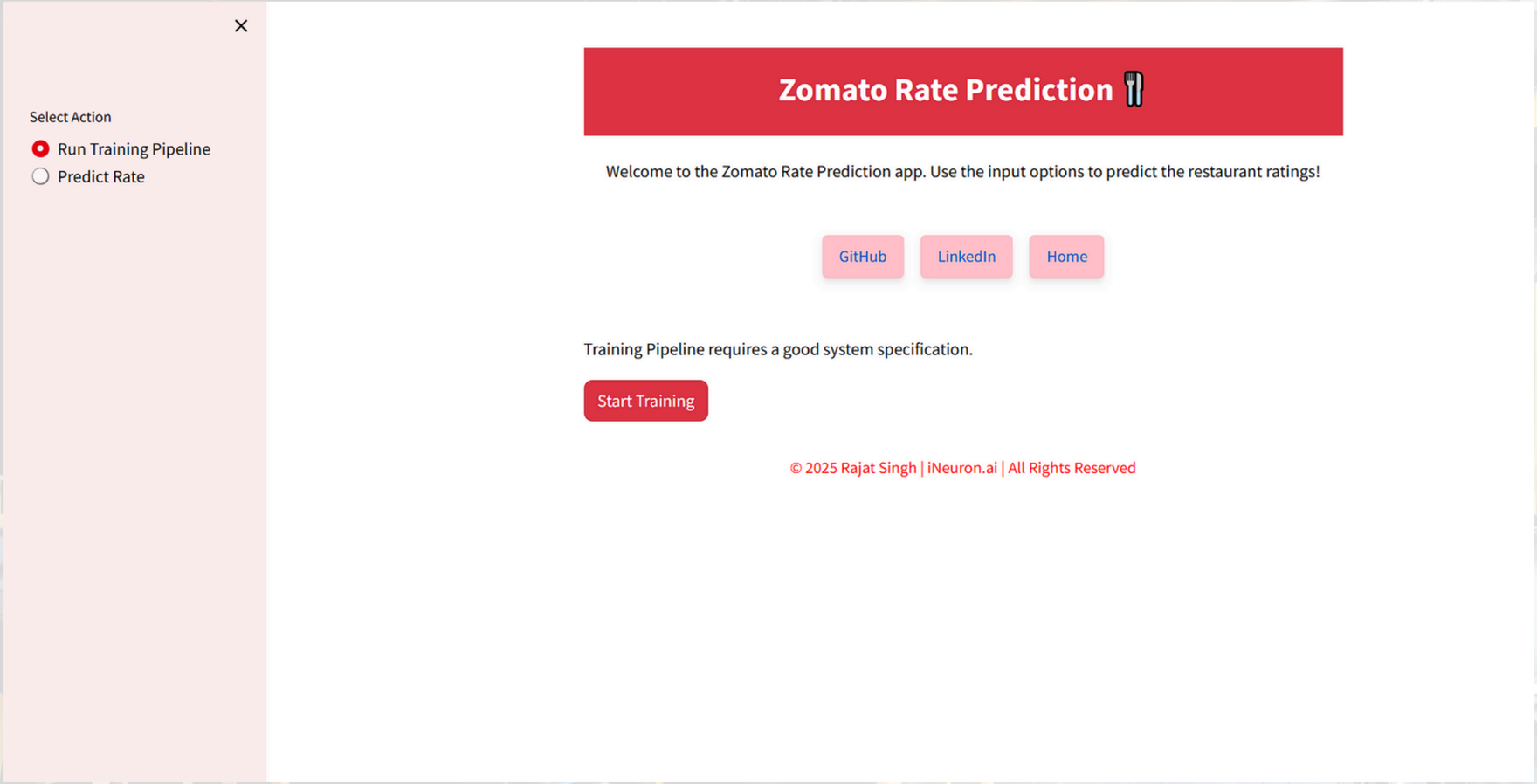
On the Prediction Page accessible via the Predict Rate button in the sidebar, users can input features to predict restaurant ratings. The input form is split into two sections for ease of use:

Input Features:
- Online Order: Select if the restaurant accepts online orders.
- Book Table: Indicate if table booking is available.
- Location Choose the restaurant's location.
- Restaurant Type: Specify the type of restaurant.

In another section:
- Cuisines: Select the type of cuisines offered.
- Approximate Cost Enter the average cost for two people.
- Votes Adjust the slider to provide the number of votes the restaurant has received.

Important Features:
The app places a higher weight on features like:
- Votes
- Approximate Cost
- Book Table

Training inputs



TRAINING BUTTON

When you click the **Run training Pipeline** button, After clicking **start training** button the app starts training the machine learning model. Please note that training can take a significant amount of time, especially if the instance resources are low. Avoid triggering it frequently in such cases.

TOOLS

Tools and Technologies Used

Project overview:

EDA

research.ipynb file

1.HANDLING MISSING DATA

The `dish_liked` column has over 20,000 missing (null) values, which is more than half of the total entries in the column. This significant number of missing values can negatively affect the analysis, so it is essential to handle them appropriately.

Steps Taken:
- Identified null values in the `dish_liked` column.
- Decided to fill these null values based on the business context or use a suitable imputation method like the mode or a predefined constant (e.g., "Unknown")

2. CLEANING IMPORTANT COLUMNS

After handling missing values, the next step is cleaning the important columns. The column may contain irrelevant or erroneous data that should be removed or corrected.

Approach:
- Drop irrelevant/extra features*that are not useful for the analysis.
- Impute missing values using the mode or a similar business rule for Categorical column.

This ensures the column contains only meaningful and clean data for further analysis.

research.ipynb file

3.TARGET COLUMN

Handling the `rating` (Target Column)
The `rating` feature, which is our target column, contains several garbage values such as "/5" and text labels like "NEW" and "nan." These should be cleaned before performing any analysis or modeling.

Steps Taken:
- Removed unnecessary symbols like "/5" from the `rating` column.
- Replaced labels such as "NEW" with `NaN` to mark them as missing values.
- Imputed missing values with the median of the column, ensuring continuity and preventing errors in analysis.

4. CLEANING THE `APPROX_COST` (FOR TWO PEOPLE) COLUMN

The `approx_cost` column, which is a continuous feature, contains some missing values that need to be addressed.

Steps Taken:
- Applied appropriate imputation strategies, such as replacing missing values with the median cost or the mean, based on the business requirements.
- Ensured that the imputation did not introduce bias or distort the data's integrity.

5. ANALYSIS OF TOP 20 RESTAURANTS BY ORDERS

To get a better understanding of the popular restaurants on Zomato, we analyzed the top 20 restaurants with the highest number of online orders.

From the data, Onesta emerges as the restaurant with the most online orders, followed closely by Empire Restaurant and Kanti Sweets

research.ipynb

ANALYSIS

01

Top Restaurants with the Highest Number of Orders by Performing EDA graph clearly shows that **Onesta** has the maximum number of online orders. This is an indication of its popularity in the Zomato dataset.

-

02

-**Top Locations for Maximum Orders**: We observe that the locations with the highest number of orders include:

- **Kormangala** (particularly 4th, 5th, 6th, and 7th blocks)
- **Jayanagar**
- **BTM Layout**

These locations are highly popular for restaurant orders, indicating strong demand in these areas.

03

The restaurant types that dominate Zomato in terms of online orders are:

- **Quick Bites**
- **Casual Dining**
- **Café**

These types are highly represented in the dataset and likely contribute to the large number of orders.

ANALYSIS OF RESTAURANT RATINGS

By analyzing the dataset, we observe that the average rating across restaurants is **3.7**. This represents the general user satisfaction level with restaurants listed on Zomato.

Top Restaurants with Highest Ratings:
- **Asia Kitchen By Mainland China** (Rating: 4.9)
- **Byg Brewski Brewing Company** (Rating: 4.9)

These restaurants have an exceptional rating of 4.9, indicating top-tier dining experiences.

RESTAURANTS WITH AVERAGE RATINGS ABOVE 4.5

- Punjab Grill
- Belgian Waffle Factory
- The Pizza Bakery
- CTR
- Maziga
- You Mee
- Burma Burma
- Chili's American Grill & Bar
- Hammered
- Brew and Barbeque - A Microbrewery Pub
- Lot Like Crepes
- Koramangala Social
- Dock Frost'd
- Sarjapur Social
- Sea Rock
- Smoke House Deli
- The Blue Wagon - Kitchen
- The Fatty Bao - Asian Gastro Bar

These restaurants have ratings above 4.5, making them standout options for users seeking a quality dining experience.

PREBOOKING FACILITY IN RESTAURANTS

From the analysis, it is observed that a majority of Zomato restaurants do not offer a prebooking facility. This suggests that many restaurants are more focused on walk-in customers, as prebooking is not a common feature.

Prebooking Facility in Top 20 Restaurants:
Only two restaurants in the top 20 list offer the prebooking facility:
- Onesta
- Mainland China

This shows that while prebooking is not widespread, it does exist among popular restaurants.

Conclusion
Through this analysis, we have gained valuable insights into the Zomato dataset, including:
- Cleaning and handling missing or erroneous data to ensure reliable analysis.
- Identifying top-performing restaurants based on orders, ratings, and other factors.
- Highlighting key locations and restaurant types that dominate the platform.
- Understanding the impact of features like prebooking on restaurant choices.

This process not only helps in cleaning the dataset but also provides an in-depth understanding of user preferences and the restaurant market on Zomato.

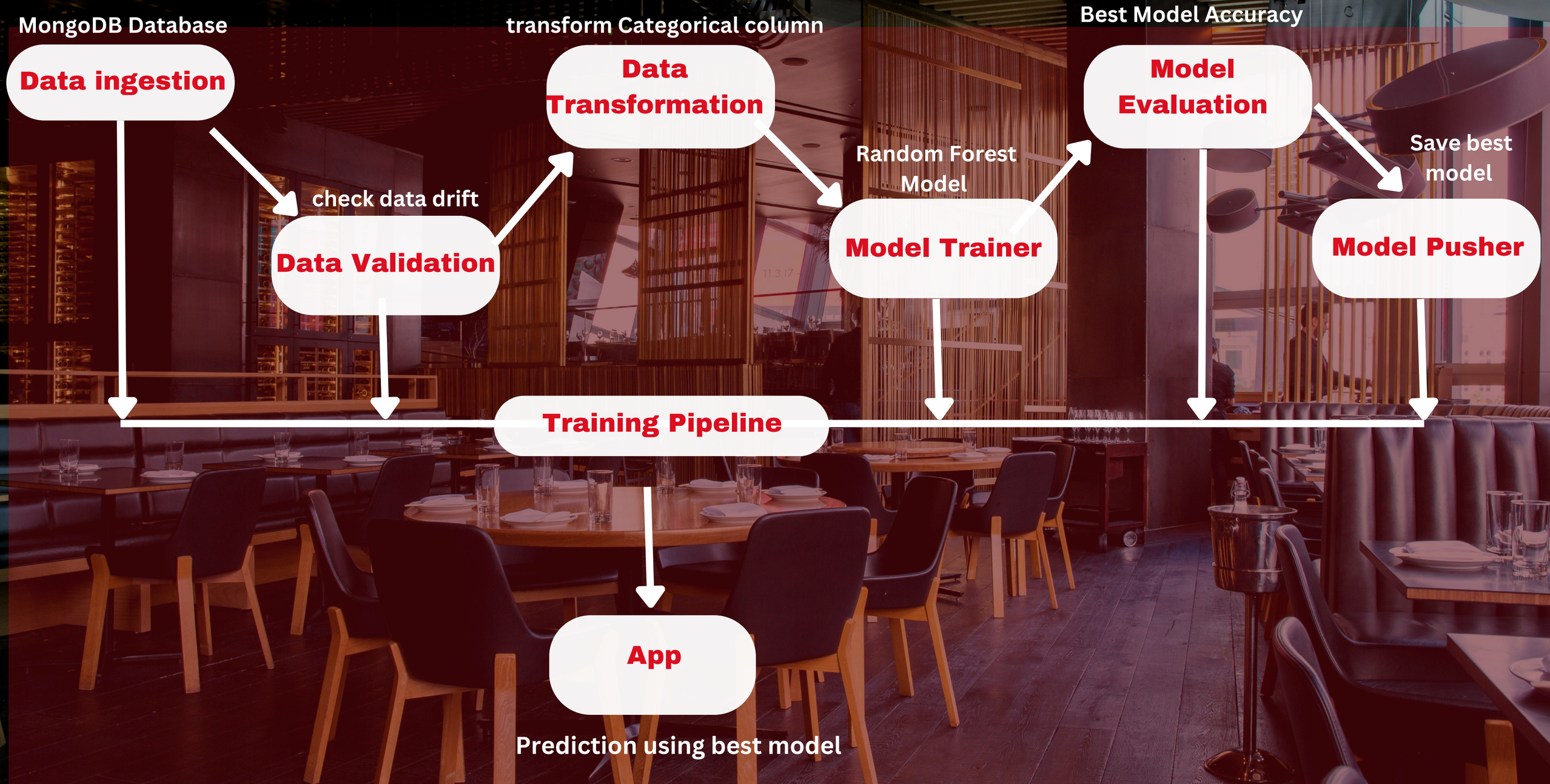
Zomato Rating Prediction

MLOPS

TRAINING PIPELINES

Zomato Rating Prediction

FLOW CHART



src/components/data_ingestion.py

DATA INGESTION

Initialization

- The DataIngestion class initializes with a configuration object that specifies database details, file paths, and test data size.
- Logs are generated to track the start of the ingestion process.

2. Fetching Data from MongoDB

- Data is exported from the MongoDB Atlas collection as a pandas DataFrame using a utility function.
- This DataFrame forms the raw dataset for further processing.

3. Saving Data to Feature Store

- The feature store is a directory where the raw dataset is saved as a CSV file.
- If the directory doesn't exist, it is created automatically.

4. Splitting Dataset

- The dataset is split into training and testing sets based on the specified test size.
- These subsets are crucial for training and validating machine learning models.

5. Saving Train and Test Data

- Both the training and testing sets are saved as separate CSV files in their respective directories.
- These directories are created if they do not exist.

6. Creating an Artifact

- An artifact is generated containing file paths for the feature store, training set, and testing set.
- This artifact is returned for use in subsequent pipeline stages.

Key Highlights

- Connects to MongoDB Atlas to fetch data efficiently.
- Stores raw data for traceability and backup.
- Splits the dataset into train and test sets for model development.
- Handles errors gracefully with custom exception handling.

This process ensures a structured and reusable approach to preparing datasets for machine learning workflows.

src/components/data_validation.py

DATA VALIDATION

. Initialization
- The DataValidation class is initialized with configuration and artifact objects.
- A dictionary is used to log validation errors.

2. Handling Missing Values
- Columns with missing values exceeding a threshold are dropped.
- Logs keep track of the dropped columns.

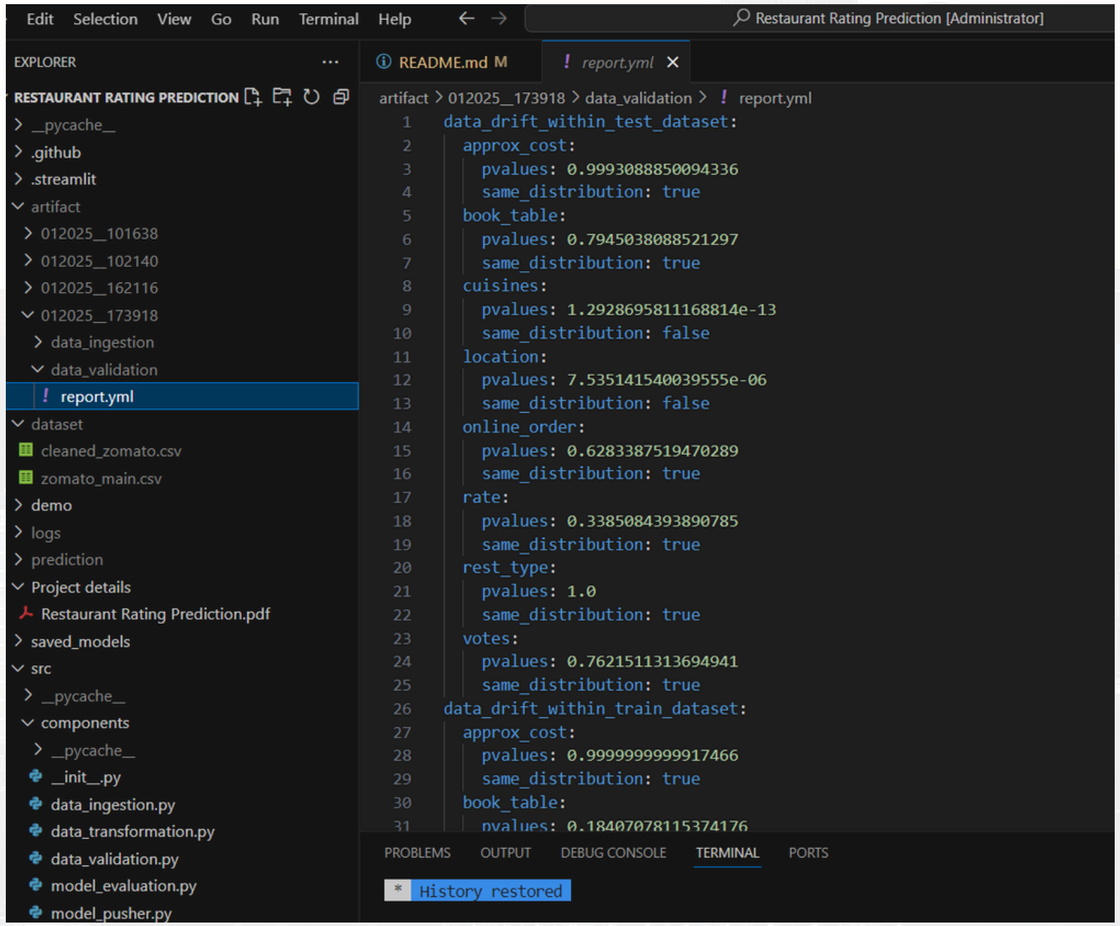
3. Column Validation
- Ensures that all required columns from the base dataset exist in the training and testing datasets.
- Logs any missing columns.

4. Data Drift Detection
- Checks for differences in the distributions of columns between the base, training, and testing datasets.
- Uses statistical tests like KS-2-sample for numerical data and Chi-square for categorical data.

5. Data Conversion
- Converts all numerical columns (excluding the target column) to a consistent data type.

6. Report Generation
- Validation results, such as missing columns and data drift, are saved to a YAML report file.
- This report serves as a reference for debugging or improving data quality.

7. Validation Artifact
- Generates an artifact containing the validation report file path, used for the next pipeline steps.



src/components/data_transformation.py

DATA TRANSFORMATION

01

This process prepares data for machine learning by encoding features and saving transformed datasets.
Steps:

1. Initialization:
2. Load configuration and artifact paths; log the process.
3. Data Loading:
4. Read training and testing datasets into pandas DataFrames.
5. Feature Encoding:

- Label Encoding: Ordinal features encoded using LabelEncoder.
- One-Hot Encoding: Nominal features encoded using OneHotEncoder.

02

- Data Cleanup:
- Drop original nominal features and concatenate encoded features.
- Feature Splitting:
- Separate input features (X) and target column (y) for both datasets.
- Saving Data:
- Save transformed training/testing data as numpy arrays.
- Save OneHotEncoder and LabelEncoder objects for reuse.
- Artifact Creation:
- Return a DataTransformationArtifact with file paths to the transformed data and encoders

Key Functions and Methods

- save_numpy_array_data(file_path, array): Saves numpy arrays to the specified file path.
- save_object(file_path, obj): Saves serialized Python objects like encoders.
- pd.concat(): Combines DataFrames horizontally (column-wise).

src/components/model_trainer.py

MODEL TRAINER

01

Purpose:
Trains and evaluates a regression model with optional hyperparameter tuning.
Components:
1. Initialization:

- Loads configuration (ModelTrainerConfig) and transformed data artifacts (DataTransformationArtifact).
- Logs initialization.

2.Model Tuning:

- Uses RandomizedSearchCV for hyperparameter tuning.
- Trains and returns the best model based on negative mean squared error (MSE).
- Parameters tuned include:
- n_estimators,max_depth, min_samples_split,min_samples_leaf, max_features.

02

.3. Model Training Pipeline:

1. Load Data:
   - Reads transformed training and testing numpy arrays.
2. Split Features:
   - Separates input features (X_train, X_test) and target values (y_train, y_test).
3. Train Model:
   - Fits a RandomForestRegressor with initial hyperparameters (max depth = 300, random state = 42).
   - Optionally applies hyperparameter tuning via model_tuning.
4. Evaluate Model:
   - Calculates R^2 scores for training and testing data.
   - Validates model performance:
     - Checks if the test R^2 score meets the expected_score.
     - Ensures no significant overfitting (difference ≤ overfitting_threshold)

03

1. Save Model:
   - Stores the trained model as a serialized object.
2. Prepare Artifact:
   - Creates ModelTrainerArtifact containing:
     - File path of the trained model.
     - Training and testing R^2 scores.

Error Handling:

- Uses a custom exception (SrcException) for logging and debugging

src/components/model_evaluation.py

MODEL EVALUATION

Short Explanation of the Code:

1. Initialization (__init__):
   - Sets up configurations and artifacts for model evaluation.
   - Initializes a ModelResolver to fetch saved model components.
2. Key Method (initiate_model_evaluation):
   - Checks for Existing Models: If no saved model exists, accepts the new model by default.
   - Loads Previous Artifacts: Retrieves and loads the saved model, transformer, and encoder.
   - Transforms Test Data: Matches the preprocessing steps of the saved model (e.g., encoding, feature alignment).
   - Model Comparison:
     - Calculates R^2 scores for both the previous and current models on test data.
     - Accepts the new model only if its R^2 score is higher.
   - Artifact Creation: Logs results and returns a ModelEvaluationArtifact indicating the decision.
3. Error Handling:
   - Ensures exceptions are logged and raised in a structured format using SrcException.

src/components/model_pusher.py

MODEL PUSHER

STEP 01

The ModelPusher class is responsible for saving the trained machine learning model, transformer, and encoder to specific directories, ensuring their availability for deployment or future use. Here's an overview of the code:

1. Initialization (__init__):
   - The class takes in three parameters: model_pusher_config, data_transformation_artifact, and model_trainer_artifact. These configurations hold paths for saving and loading artifacts.
   - It initializes a ModelResolver instance to manage model and artifact paths.

STEP 02

- Method: initiate_model_pusher():
- Loading Objects: It loads the transformer, model, and encoder using the load_object utility function.
- Saving Objects: The loaded objects are saved to the model pusher directory and the saved model directory using the save_object function.
- Model Resolver: It resolves the latest paths for the transformer, model, and encoder in the saved model directory.
- Artifact Creation: The method returns a ModelPusherArtifact, which holds paths to the saved model directory and pusher model directory.
- Error Handling: Throughout the process, errors are caught and raised as SrcException to provide detailed error information.

DEPLOYMENT PROCESS OVERVIEW

STEP 01

GitHub Secrets Setup:
The first step is to securely store necessary AWS credentials and connection information in the GitHub repository's Secrets. These credentials will be used during the CI/CD pipeline execution.
The following secrets should be added:

- AWS_ACCESS_KEY_ID: For AWS access authentication.
- AWS_SECRET_ACCESS_KEY: For AWS secret authentication.
- AWS_REGION: The AWS region where the resources are located (e.g., us-west-2).
- AWS_ECR_LOGIN_URI: The URI for AWS ECR login (e.g., 123456789012.dkr.ecr.us-west-2.amazonaws.com).
- ECR_REPOSITORY_NAME: The name of the ECR repository where the Docker images will be pushed.
- MONGO_DB_URL: The URL for connecting to the MongoDB database (if applicable).

GitHub Actions Workflow (main.yaml): The workflow file defines the sequence of steps required to build and deploy the Docker image. The key actions in the workflow are as follows:

- Checkout the Repository: The first step in the workflow is to check out the latest code from the GitHub repository.
- Set up Docker Buildx: Docker Buildx is used to enable advanced Docker building capabilities.
- Log in to AWS ECR: This step logs into AWS ECR using the provided AWS credentials.
- Build and Push Docker Image: The Docker image is built from the repository's Dockerfile, tagged, and pushed to the AWS ECR repository using the credentials.
- Deploy Docker Image to EC2: After the Docker image is successfully pushed to ECR, the workflow SSHs into the EC2 instance, pulls the image from ECR, and runs it on port 8080.
- Update EC2 Security Group: A custom inbound rule is added to allow inbound TCP traffic on port 8080, which is required for the application running in the Docker container

STEP 02

- Dockerfile Setup:
- The Dockerfile should be configured to use Python version 3.12. It is responsible for setting up the environment, installing dependencies from requirements.txt, and running the application.
- AWS EC2 Setup:
- The EC2 instance should be configured to run Docker and allow SSH access. It must also have the necessary IAM role with permissions to pull images from ECR and manage security groups. Additionally, ensure that the EC2 instance's security group allows traffic on port 8080 for the Docker container.

Key Execution Flow

- Before Execution: Ensure all GitHub Secrets are properly configured for AWS credentials, ECR repository URI, MongoDB URL, and other sensitive data.
- During Execution:
  a. GitHub Actions will run the workflow and perform tasks like checking out the repository, setting up Docker, logging into AWS, and pushing the Docker image to ECR.
  b. Afterward, it will SSH into the EC2 instance, pull the latest Docker image from ECR, and run the container.
  c. The workflow will also add an inbound rule to the EC2 security group to allow traffic on port 8080, which is necessary for the Docker container to be accessible

THANK YOU

rajat.k.singh64@gmail.com

https://github.com/RJSINGH64/restaurant_rating-prediction.git