Project Report MLops

Name of Project: Invoice Information Extractor

Team Members: Harsimran Singh(2110993790)

Ranveer chaudhary(2110993886)

Shouzaf zafar(2110993835)

Pranshu Kaushal(2110993888)

Abstract

Tired of Drowning in Paperwork? Automate Your Invoices! :

Invoices are the backbone of any business deal, but wading through mountains of them to extract information can be a nightmare. Manual processing is slow, error-prone, and a drain on your team's resources. This abstract dives into the world of automated invoice information extraction, a game-changer for streamlining financial workflows.

Imagine a system that can automatically pull crucial data like invoice number, date, and amount from any invoice, regardless of format. This is the magic of automated extraction. We'll explore two main approaches: templates and artificial intelligence (AI).

Templates: Perfect for Familiar Faces :

Template-based extraction works well for invoices from vendors you know. The system learns the layout of their invoices – like where they put the invoice number or total amount – and extracts data based on that layout. Think of it as remembering a friend's house – easy to navigate once you've been there. However, this method struggles with invoices from new vendors, just like finding your way in an unfamiliar city.

The Benefits: Time Saved, Money Earned :

By automating invoice processing, you free up your team's time for more important tasks. Plus, the accuracy of extracted data improves, meaning fewer errors and more reliable financial records. But it gets even better! Automated extraction gives you insights into your spending habits, helping you identify areas to save money and make smarter financial decisions.

So, ditch the manual processing and step into the future of invoice management!

Introduction

Invoice processing is the fundamental business function of managing supplier invoices from the moment they are received to the final payment and record-keeping. It's a crucial step within the procure-to-pay (P2P) process, acting as the closing chapter for any procurement activity. Traditionally handled by the accounts payable department, invoice processing ensures timely payments to vendors while maintaining accurate financial records.

Importance of Solving Invoice Processing Challenges:

Manually processing invoices can be a time-consuming and error-prone task, especially for businesses dealing with high volumes. This inefficiency leads to several challenges:

Delayed Payments: Manual processing can lead to delays in identifying and approving invoices, impacting vendor relationships and potentially incurring late fees.

Data Entry Errors: Manual data entry is susceptible to errors, causing discrepancies in financial records and hindering accurate reporting.

Reduced Visibility: Paper-based invoices or siloed data make it difficult to track spending and gain real-time insights into cash flow.

By automating invoice processing using techniques like yours, with natural language processing (NLP) and machine learning, you can address these challenges and gain significant benefits:

Improved Efficiency: Automating data extraction and approval workflows significantly reduces processing time, freeing up staff for more strategic tasks.

Enhanced Accuracy: NLP and machine learning models minimize data entry errors, ensuring accurate financial records and streamlined reporting.

Greater Visibility: Automated solutions provide real-time insights into spending patterns and cash flow, enabling better financial decision-making.

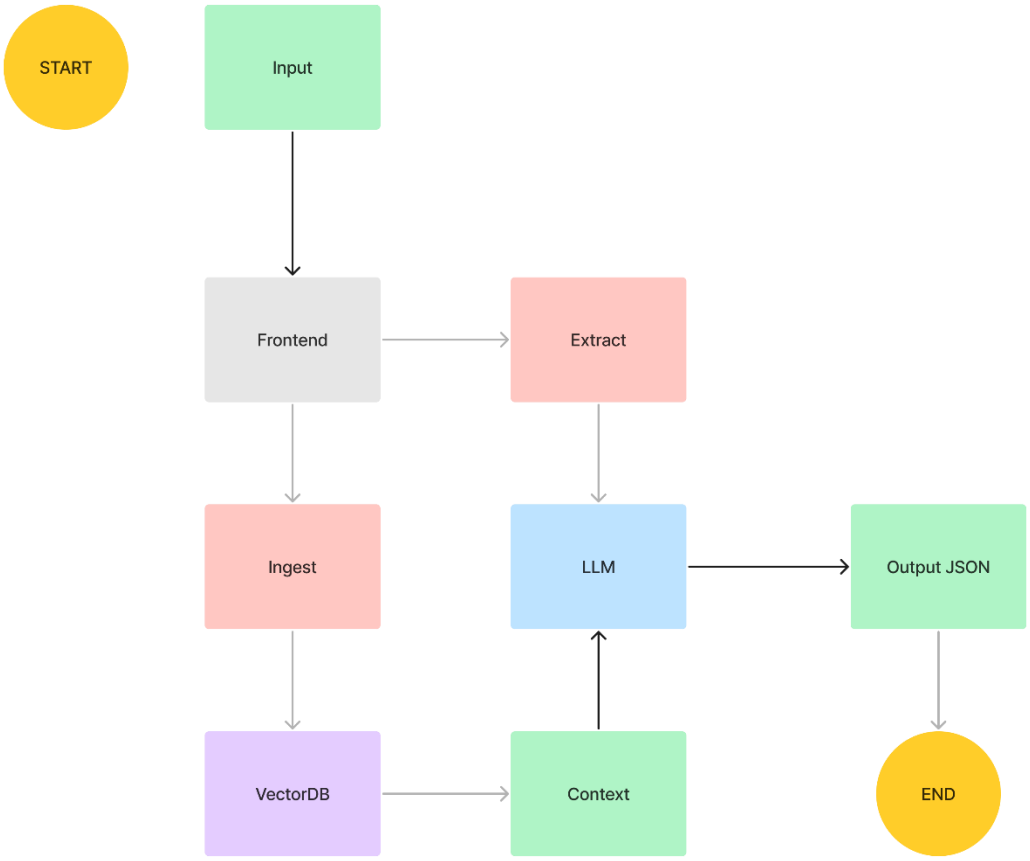
Strengthened Relationships: Faster invoice processing translates to timely payments, fostering stronger relationships with vendors.

By deploying our model on Docker, we containerize the application environment, allowing for easy portability and scalability across different platforms. Docker's lightweight, isolated containers ensure consistency in software dependencies, facilitating deployment across various computing environments without compatibility issues. Furthermore, we employ Jenkins, an open-source automation server, to automate the deployment pipeline. Jenkins enables continuous integration and continuous deployment (CI/CD), automating the testing and deployment of our machine learning model. Through Jenkins pipelines, we establish a robust workflow for building, testing, and deploying the application, ensuring reliability and efficiency in the software development lifecycle. By combining machine learning for diabetes prediction with Docker and Jenkins for streamlined deployment and automation, we aim to enhance accessibility and usability, enabling healthcare practitioners to leverage advanced predictive analytics for early disease detection and personalized patient care. This project underscores the transformative potential of integrating machine learning with DevOps practices, paving the way for innovative solutions in healthcare delivery and disease management.

Docker is a platform for developing, shipping, and running applications in containers. Containers are lightweight, portable, and self-sufficient units that encapsulate everything needed to run an application, including code, runtime, system tools, libraries, and settings. Docker provides a standardized way to package and distribute applications, ensuring consistency across different environments, from development to production.

Jenkins is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) processes in software development. It automates the building, testing, and deployment of applications, enabling teams to deliver high-quality software more efficiently and reliably.

Flowchart



Technologies Used

- **LLMs :** LLMs stands for **Large Language Models**. They are a type of artificial intelligence (AI) program that are trained on massive amounts of text data. This training allows them to process information and generate human-like text in response to a wide range of prompts and questions. Here's a deeper look at what LLMs are and what they can do:

Core Functionality:

- **Understanding Language:** LLMs are trained on vast amounts of text data, including books, articles, code, and even social media conversations. This allows them to develop a sophisticated understanding of human language, including grammar, syntax, and semantics.
- **Text Generation:** Based on their understanding of language, LLMs can generate different creative text formats, like poems, code, scripts, musical pieces, emails, and letters. They can even translate languages, write different kinds of creative content, and answer your questions in an informative way.

- **Learning and Adapting:** LLMs are constantly learning and improving. As they are exposed to more data, they become better at understanding complex nuances of language and generating more refined outputs.

- **Machine Learning Operations:** Machine Learning Operations (MLOps) is a set of practices and tools aimed at streamlining and automating the lifecycle management of machine learning models. It encompasses processes such as model development, training, deployment, monitoring, and maintenance. MLOps aims to bridge the gap between data science and IT operations, ensuring that machine learning models are deployed and managed effectively in production environments. Key components of MLOps include version control for models and data, continuous integration and deployment (CI/CD) pipelines tailored for machine learning workflows, model performance monitoring, drift detection, and automated retraining. By implementing MLOps practices, organizations can improve the efficiency, reliability, and scalability of their machine learning projects, enabling faster time-to-market and better alignment with business objectives.
- **Docker:** Docker revolutionized the software development landscape by introducing containerization as a key methodology for packaging, distributing, and managing applications. At its core, Docker operates on the principle of creating lightweight, portable containers that encapsulate an application and its dependencies. Unlike traditional virtual machines, which include an entire operating system stack, Docker containers share the host system's kernel, resulting in reduced overhead and faster startup times. One of Docker's primary benefits is its ability to provide consistency across development, testing, and production environments. Developers can package their applications and dependencies into Docker images, which can then be easily shared and deployed across different systems. This ensures that applications behave predictably regardless of the underlying infrastructure, eliminating the "it works on my machine" problem commonly encountered in software development. Docker also facilitates scalability and resource efficiency. Containers consume fewer resources compared to virtual machines, allowing for higher density deployments and more efficient utilization of hardware resources. Moreover, Docker's lightweight nature makes it ideal for microservices architectures, where applications are broken down into smaller, independently deployable components.

Another key feature of Docker is its support for orchestration tools such as Docker Swarm and Kubernetes. These tools enable the management and deployment of containerized applications at scale, providing features such as load balancing, automatic scaling, and service discovery. Overall, Docker has transformed the way software is developed, shipped, and maintained by offering a standardized, efficient, and scalable approach to application deployment. Its ease of use, portability, and compatibility with existing infrastructure have made it a cornerstone technology in modern software development practices.

- **Jenkins:** Jenkins is an open-source automation server that facilitates continuous integration (CI) and continuous delivery (CD) in software development. It enables developers to automate various aspects of the software development lifecycle, including building, testing, and deploying applications. Jenkins achieves this through pipelines, which are sequences of steps that define how software changes are built, tested, and deployed. One of Jenkins' key features is its flexibility and extensibility. It supports a vast ecosystem of plugins that extend its functionality and integrate with various tools and technologies commonly used in software development. These plugins enable Jenkins to integrate with version control systems, build tools, testing frameworks, deployment platforms, and more, allowing for a highly customizable CI/CD workflow tailored to the specific needs of each project.

  Jenkins provides a web-based user interface that allows users to configure and manage jobs, view build results, and access logs and reports. It also supports role-based access control (RBAC), enabling administrators to define permissions and restrict access to sensitive information and critical operations.

  Another important aspect of Jenkins is its support for distributed builds. Jenkins can distribute build jobs across multiple nodes, allowing for parallel execution and improved performance. This scalability makes Jenkins suitable for handling large and complex software projects with multiple contributors and dependencies. Overall, Jenkins plays a crucial role in modern software development practices by automating repetitive tasks, improving collaboration among team members, and accelerating the delivery of high-quality software. Its versatility, ease of use, and vibrant community support have made it one of the most widely used CI/CD tools in the industry.

- **GitHub:** GitHub is a web-based platform built around Git, a distributed version control system that allows developers to manage and track changes to their codebase. It serves as a central hub for collaborative software development, offering a range of features and tools designed to facilitate collaboration, code sharing, and project management. One of GitHub's key features is its repository hosting, where developers can create, clone, and manage Git repositories for their projects. Each repository contains the project's source code, along with version history, branches, and pull requests. GitHub provides a web-based interface for viewing and interacting with repositories, making it easy for developers to collaborate on code changes, review each other's work, and resolve conflicts. GitHub also offers issue tracking, a feature that allows developers to report bugs, suggest new features, and discuss project-related topics. Issues can be assigned to team members, labeled, and organized into milestones, providing a structured way to manage project tasks and priorities. Furthermore, GitHub supports continuous integration and deployment (CI/CD) through integrations with popular CI/CD tools such as Jenkins, Travis CI, and GitHub Actions. This allows developers to automate the testing, building, and deployment of their applications directly from GitHub, streamlining the software development process and ensuring code quality and reliability. Additionally, GitHub provides collaboration tools such as wikis, project boards, and discussions, enabling teams to document project information, plan and track work, and communicate effectively within the project's context. Overall, GitHub has become an essential platform for software development teams, offering a centralized and collaborative environment for managing code, tracking issues, and automating workflows. Its intuitive interface, robust features, and vibrant community have made it the go-to choice for millions of developers worldwide.

Dockerization

Dockerization, also known as containerization, is the process of encapsulating an application and its dependencies into a lightweight, portable container. Docker provides a platform and tools to package, distribute, and run applications within containers, ensuring consistency and reproducibility across different environments.

Here's a more detailed explanation of the steps involved in Dockerizing a machine learning model for diabetes prediction:

1. **Identify Dependencies:** Before containerizing the model, it's essential to identify all dependencies required for its execution. This includes not only the machine learning libraries and frameworks used to develop the model (e.g., scikit-learn, TensorFlow, PyTorch) but also any additional software libraries or packages needed to preprocess data or interface with the model.

2. **Create a Dockerfile:** The Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image to use, installs dependencies, copies the model code and data files into the image, and configures the environment needed to run the model.

3. **Choose a Base Image:** The base image serves as the foundation for the Docker container. It typically includes a minimal Linux distribution with essential tools and libraries. Depending on the requirements of the model, you can choose an appropriate base image, such as a Python or TensorFlow image, from the Docker Hub registry.

4. **Install Dependencies:** Use the Dockerfile to specify the installation of all dependencies required by the model. This may include installing Python packages using pip or conda, installing system libraries using apt-get or yum, or downloading pre-trained models and data files.

5. **Copy Model Code:** Copy the source code of the machine learning model into the Docker image. This includes any Python scripts, modules, or notebooks needed to load the model, preprocess data, and make predictions.

6. **Expose Ports (Optional):** If the model requires communication with external systems or services, you may need to expose network ports in the Dockerfile to allow inbound connections to the container.

7. **Build the Docker Image:** Use the Docker CLI (Command Line Interface) to build the Docker image from the Dockerfile. This process involves executing the `docker build` command and specifying the path to the directory containing the Dockerfile. Docker then follows the instructions in the Dockerfile to create a new image.

8. **Run the Docker Container:** Once the Docker image is built successfully, you can run a Docker container based on that image using the `docker run`

command. This command starts a new container instance from the specified image, providing an isolated environment where the model can be executed.

9. **Test the Containerized Model:** After running the Docker container, verify that the model behaves as expected within the container environment. This may involve executing test scripts, making sample predictions, or validating the model's performance against a test dataset.

10. **Push the Docker Image (Optional):** If you plan to deploy the containerized model to a remote server or cloud platform, you can push the Docker image to a container registry such as Docker Hub or a private registry. This allows you to share the image with others and deploy it to different environments easily.

By Dockerizing the machine learning model, you create a self-contained and reproducible environment that encapsulates all dependencies and ensures consistent behavior across different systems and platforms. This simplifies the deployment process and enables seamless integration with other tools and technologies, such as Jenkins for continuous integration and deployment.

Jenkins Configuration and Pipeline

Jenkins configuration and pipeline are fundamental components of setting up continuous integration and continuous delivery (CI/CD) workflows in software development. Let's break down each aspect:

1. **Jenkins Configuration**:

Jenkins configuration involves setting up the Jenkins server and configuring its settings to meet the requirements of your project. Here are some key aspects of Jenkins configuration:

- **Installation:** Install Jenkins on a server or local machine by downloading the Jenkins WAR file or using a package manager (e.g., apt, yum).
- **Setup:** After installation, access the Jenkins web interface to perform initial setup tasks, such as creating an admin user, setting up security options, and configuring plugins.

- **Plugin Installation:** Install Jenkins plugins to extend its functionality. Plugins are available for various purposes, including source code management (e.g., Git, SVN), build tools (e.g., Maven, Gradle), testing frameworks, and deployment.
- **Global Configuration:** Configure global settings such as email notifications, version control system credentials, build tools installations, and executor settings.
- **Node Configuration:** Configure Jenkins nodes (also known as agents or slaves) to distribute build jobs across multiple machines for parallel execution.
- **User Management:** Manage user accounts, permissions, and access control to ensure secure collaboration within the Jenkins environment.

2. **Jenkins Pipeline:**

Jenkins Pipeline is a suite of plugins that allows you to define build processes and workflows as code. Pipelines enable you to automate the entire software delivery process, from code commit to deployment, using a Groovy-based DSL (Domain Specific Language). Here's how Jenkins Pipeline works:

- **Pipeline Script:** Define your build process as a Jenkinsfile, which contains the pipeline script written in Groovy syntax. The Jenkinsfile can be stored in the project repository or provided inline in the Jenkins job configuration.
- **Stages and Steps:** Organize the build process into stages, each representing a distinct phase of the pipeline (e.g., build, test, deploy). Within each stage, define individual steps, such as checking out source code, compiling code, running tests, and deploying artifacts.
- **Parallel Execution:** Utilize parallel blocks to execute multiple stages or steps concurrently, optimizing build time and resource utilization.
- **Declarative and Scripted Syntax:** Jenkins Pipeline supports both declarative and scripted syntax. Declarative syntax provides a more structured and concise way to define pipelines, while scripted syntax offers greater flexibility and control over the build process.
- **Error Handling**: Implement error handling and recovery mechanisms using try-catch blocks and error handling steps to handle exceptions and failures gracefully.

- **Integration with SCM:** Jenkins Pipeline integrates seamlessly with version control systems (e.g., Git, SVN) to automatically trigger pipeline execution upon code changes and to retrieve source code for building.

By configuring Jenkins and defining pipelines, teams can automate their software development workflows, ensure consistency in build processes, and achieve faster and more reliable delivery of software updates. Jenkins Pipeline provides a powerful and flexible framework for implementing CI/CD practices and automating complex build and deployment scenarios.

Github Integration

GitHub integration involves syncing your project files, including the Dockerfile, Jenkinsfile, source code, and any other relevant files, with a GitHub repository. This integration facilitates collaboration, version control, and automation within your software development workflow. Here's a more detailed explanation of GitHub integration:

1. **Create a GitHub Repository**: Start by creating a new repository on GitHub or using an existing one. Give your repository a descriptive name and, optionally, a brief description to convey its purpose.

2. **Clone the Repository**: Clone the GitHub repository to your local machine using Git. This allows you to work on your project locally and sync changes with the remote repository on GitHub.

```
git clone <repository-url>
```

3. **Add Project Files**: Place all relevant project files, including the Dockerfile and Jenkinsfile, in the local repository directory. Additionally, include any source code, configuration files, documentation, or other assets required for your project.

4. **Stage and Commit Changes:** Use Git to stage and commit your changes locally. Staging prepares changes for commit, while committing records changes to the local repository with a descriptive commit message.

```
git add .
```

```
git commit -m "Add Dockerfile and Jenkinsfile"
```

5. **Push Changes to GitHub**: Push your committed changes from the local repository to the remote GitHub repository. This syncs your local changes with the GitHub repository, making them accessible to other collaborators.

```
git push origin master
```

6. **Verify Changes on GitHub**: Visit the GitHub repository in your web browser to verify that the files you pushed from your local repository are now available in the remote repository.

7. **Branching and Pull Requests (Optional**): If you're working with a team or implementing new features, consider creating branches for separate features or bug fixes. You can then open pull requests to propose changes and collaborate with other team members before merging the changes into the main branch.

GitHub integration enables seamless collaboration, version control, and automation within your software development workflow. By syncing your project files with a GitHub repository, you ensure that all team members have access to the latest codebase, track changes over time, and leverage GitHub's collaboration features such as issues, pull requests, and project boards to streamline development processes. Additionally, GitHub integrations with CI/CD tools like Jenkins enable automated build, test, and deployment workflows, further enhancing productivity and efficiency in software development.

Deployment

Deployment in the context of a CI/CD pipeline refers to the process of taking the artifacts produced by the build stage (in this case, the Dockerized model) and making them available for use in a production or testing environment. In the case of deploying a Dockerized model on Jenkins using a CI/CD pipeline, here's a more detailed explanation of the process:

1. CI/CD Pipeline Setup: Firstly, you need to set up a CI/CD pipeline in Jenkins that includes stages for building, testing, and deploying the Dockerized model. This pipeline is defined using a Jenkinsfile, which outlines the sequence of steps and actions to be executed.

2. **Build Stage:** The build stage of the pipeline involves compiling the source code, running any necessary tests, and creating the Docker image containing the model and its dependencies. This stage typically includes executing commands to build the Docker image using the Dockerfile and pushing the image to a container registry such as Docker Hub.

3. **Test Stage (Optional)**: In some CI/CD pipelines, there may be an additional stage for testing the Dockerized model before deployment. This stage may involve running automated tests against the Docker image to ensure its functionality and performance meet the required criteria.

4. **Deployment Stage**: The deployment stage is where the Dockerized model is deployed to the target environment. This could be a staging environment for testing purposes or a production environment for live usage. The deployment process may involve pulling the Docker image from the container registry, starting containers based on the image, and configuring any necessary environment variables or settings.

5. **Rolling Updates (Optional)**: For production deployments, it's common to implement strategies like rolling updates to minimize downtime and ensure smooth transitions. This involves gradually replacing old instances of the model with new ones, allowing for continuous availability of the application.

6. **Monitoring and Rollback (Optional)**: Once the deployment is complete, it's important to monitor the deployed model's performance and behavior in the target environment. In case of any issues or failures, a rollback mechanism should be in place to revert to a previous, stable version of the model.

By automating the deployment process through a CI/CD pipeline in Jenkins, teams can achieve faster and more reliable deployments, reduce manual errors, and ensure consistency across different environments. Additionally, Jenkins provides visibility into the deployment process through its web interface and logs, allowing for easy tracking and troubleshooting of deployment-related issues.

GitHub Link

https://github.com/Harsimran-19/invoice-extractor

Conclusion

In conclusion, our project on diabetes prediction using machine learning represents a comprehensive fusion of cutting-edge data science techniques with modern DevOps methodologies. Throughout the project lifecycle, we've meticulously curated and processed diverse datasets, ensuring the integrity and quality of the input data. By employing various machine learning algorithms and rigorous evaluation methodologies, we've not only developed accurate predictive models but also gained valuable insights into the factors influencing diabetes onset.

The integration of Docker, Jenkins, and GitHub has significantly streamlined our workflow, offering a seamless transition from model development to deployment. Dockerization of our models has endowed them with portability, scalability, and reproducibility, encapsulating all necessary dependencies within lightweight containers. This ensures consistent performance across different environments and simplifies the deployment process.

Jenkins automation has played a pivotal role in orchestrating the deployment pipeline, automating tasks such as building Docker images, running tests, and deploying models. This automation reduces manual effort, accelerates delivery cycles, and enhances overall productivity. The CI/CD pipeline implemented in Jenkins ensures that changes to the model are thoroughly tested before deployment, guaranteeing reliability and stability in production environments.

Furthermore, GitHub serves as a central repository for our project, fostering collaboration, version control, and transparency among team members. By sharing our project files, including the Dockerfile, Jenkinsfile, and source code, on GitHub, we enable seamless collaboration, peer review, and knowledge sharing within the broader community.

In essence, our project exemplifies the convergence of advanced data analytics with modern software engineering practices, paving the way for transformative solutions in healthcare and beyond. Through continuous iteration, improvement, and innovation, we strive to leverage technology to address complex challenges and improve lives.