**Relatório de Implementação da API de Modelo Preditivo e Deploy com Docker**

**Introdução e Desenvolvimento da API com FastAPI**

O presente trabalho detalha o processo de desenvolvimento e implantação de uma API RESTful para consumir modelos de machine learning previamente treinados, responsáveis pela predição de preços de serviços de transporte. A API foi construída utilizando o framework Python FastAPI, escolhido por sua alta performance, facilidade na criação de endpoints robustos, validação automática de dados de entrada e saída através de modelos Pydantic, e pela geração automática de documentação interativa (Swagger UI), o que agiliza significativamente os ciclos de desenvolvimento e teste.

O núcleo da API reside no arquivo main.py. Na inicialização, a aplicação carrega múltiplos modelos preditivos serializados (arquivos .pkl), cada um correspondendo a uma categoria de serviço de transporte (e.g., UberX, UberComfort, 99Pop). Estes arquivos .pkl, que contêm tanto o pipeline de pré-processamento quanto o modelo treinado e a lista de features esperadas, são armazenados em um diretório modelos\_final/. A API expõe um endpoint principal /predict (via método POST) que aceita como entrada um JSON contendo a distancia\_m (distância da viagem em metros) e tempo\_estim\_segundos (duração estimada da viagem em segundos). Internamente, uma função auxiliar prepare\_input\_for\_fastapi\_model é invocada para enriquecer esses dados de entrada, gerando features contextuais e temporais cruciais para os modelos, como o ano, mês, dia da semana, hora do dia (transformados em componentes seno e cosseno para capturar a ciclicidade), e um indicador de feriado. Este processo garante que os dados fornecidos aos modelos sejam consistentes com o formato utilizado durante a fase de treinamento. Após o preparo dos dados, a API itera sobre cada modelo carregado, realiza a predição de preço e compila os resultados em um objeto JSON que é retornado ao cliente, contendo os preços estimados para cada categoria de serviço.

Conteinerização com Docker para Portabilidade e Deploy

**Conteinerização com Docker para Portabilidade e Deploy**

Para garantir a portabilidade, consistência do ambiente de execução e facilitar o processo de implantação, a aplicação FastAPI foi containerizada utilizando Docker. Um arquivo Dockerfile foi criado para especificar a construção da imagem da aplicação. A imagem Docker foi baseada em uma imagem oficial do Python (python:3.9-slim), otimizada para tamanho. O Dockerfile define o diretório de trabalho dentro do contêiner, copia o arquivo requirements.txt (que lista todas as dependências Python do projeto, como FastAPI, Uvicorn, Pydantic, Scikit-learn, Joblib e Pandas) e instala essas dependências. Subsequentemente, o script main.py da API e a pasta modelos\_final/ (contendo os arquivos .pkl dos modelos) são copiados para dentro da imagem. A porta 8000 é exposta, e o comando CMD é configurado para iniciar o servidor Uvicorn, tornando a API acessível na rede do contêiner através do host 0.0.0.0 na porta 8000.

**Testes e Validação da API em Contêiner**

A funcionalidade da API em contêiner foi validada através de múltiplos testes. Inicialmente, requisições foram enviadas utilizando a ferramenta curl diretamente do terminal do Ubuntu Server onde o contêiner estava hospedado, direcionadas ao endpoint http://192.168.0.200:8001/predict. Estas requisições incluíam um payload JSON com os campos distancia\_m e tempo\_estim\_segundos. O sucesso foi confirmado pelo recebimento de uma resposta JSON contendo os preços previstos para cada categoria de serviço, como por exemplo:

{"uberX":16.74,"uberComfort":20.01,"uberBlack":25.79,"99Pop":18.7}.

**Considerações sobre Monitoramento e Implantação em Nuvem**

Embora a implantação para este projeto tenha sido realizada em um servidor Ubuntu local, simulando um ambiente de nuvem para fins acadêmicos, o uso de containers Docker prepara a aplicação para um deploy facilitado em provedores de nuvem como Google Cloud Platform (utilizando Cloud Run), AWS (com App Runner ou ECS) ou Azure (com Container Apps). Esses serviços oferecem escalabilidade automática, gerenciamento simplificado e integração com ferramentas de monitoramento. No ambiente local, o monitoramento básico do desempenho do contêiner (uso de CPU e memória) pode ser realizado com o comando docker stats, e os logs da aplicação são acessíveis via docker logs. Em um cenário de produção na nuvem, seriam utilizados os serviços de monitoramento e logging nativos do provedor escolhido (ex: Google Cloud Monitoring e Cloud Logging) para coletar métricas detalhadas de tempo de resposta, taxa de erros, e utilização de recursos, essenciais para garantir a saúde e performance da API.

**Conclusão**

O desenvolvimento da API com FastAPI e sua conteinerização com Docker demonstraram ser uma abordagem eficaz para disponibilizar modelos de machine learning como um serviço acessível via rede. Os testes confirmaram a funcionalidade e a correta operação da API em um ambiente isolado e portátil. Este projeto proporcionou experiência prática em MLOps, desde a criação da interface do modelo até sua preparação para deploy, estabelecendo uma base sólida para futuras integrações com o backend Node.js da aplicação interdisciplinar e potenciais implantações em ambientes de nuvem produtivos.