# Documento de Arquitetura do Projeto: Previsão de Ativos Financeiros com IA

## 1. Introdução à Arquitetura do Projeto

### 1.1. Recapitulação dos Objetivos do Projeto

Este documento delineia a arquitetura técnica para o desenvolvimento de um sistema de previsão de preços diários para Bitcoin (BTC-USD) e para a ação blue-chip Apple (AAPL), com um horizonte de previsão de 14 dias. Esta iniciativa visa apoiar o fundo de investimentos "Agile Capital" na otimização de sua estratégia de alocação dinâmica de ativos.[1] Os entregáveis centrais do projeto consistem em: um Jupyter Notebook detalhado, documentando a coleta de dados, análise exploratória, engenharia de features, modelagem e avaliação de desempenho; um relatório técnico abrangente; e um dashboard interativo desenvolvido com Streamlit para visualização das previsões e métricas de performance.[1]

### 1.2. Visão Geral da Arquitetura da Solução Proposta

A solução proposta é fundamentada em um pipeline de dados robusto e modular. O processo inicia-se com a coleta e subsequente sincronização de dados financeiros históricos e recentes, provenientes de fontes diversificadas como o Kaggle e a API do Yahoo Finance. Segue-se uma etapa crítica de pré-processamento, que inclui a limpeza dos dados, tratamento de valores ausentes e normalização. A engenharia de features será focada na criação de indicadores técnicos relevantes. A modelagem preditiva explorará Redes Neurais Recorrentes (RNNs), especificamente as arquiteturas SimpleRNN, LSTM (Long Short-Term Memory) e GRU (Gated Recurrent Unit). A avaliação dos modelos será multifacetada, englobando métricas de erro de previsão (RMSE, MAE, MAPE) e a simulação de estratégias de trading para cálculo de retornos e do Sharpe Ratio. Finalmente, um aplicativo Streamlit, construído de forma modular, permitirá a exploração interativa das previsões geradas, dos indicadores técnicos e da performance simulada das estratégias. Para assegurar a integridade, qualidade e robustez da solução, será implementada uma estratégia de testes abrangente, utilizando pytest para os módulos Python e nbval para a validação do Jupyter Notebook.[1]

A ênfase na modularidade, manifestada na separação clara entre o Jupyter Notebook para exploração, os scripts Python que compõem a aplicação Streamlit e os módulos de teste dedicados, transcende a mera adesão a boas práticas de engenharia de software. Esta abordagem é fundamental para a adaptabilidade e manutenção do sistema a longo prazo, especialmente em um ambiente financeiro caracterizado por

sua dinâmica e constante evolução. A capacidade de integrar novos ativos financeiros, experimentar modelos de previsão alternativos ou adicionar novas features com impacto minimizado no restante do sistema é uma consequência direta dessa modularidade. O mercado financeiro está em fluxo contínuo, com o surgimento de novos instrumentos e variações nos padrões de volatilidade. Um sistema monolítico apresentaria desafios significativos para atualizações e adaptações. Em contraste, módulos independentes para cada etapa do processo – coleta de dados, pré-processamento, modelagem e interface do usuário – permitem que cada componente seja modificado, otimizado ou substituído isoladamente. Esta separação também simplifica consideravelmente o processo de teste, permitindo a verificação focada de cada componente antes da integração.

A exigência de uma rotina de testes rigorosa, com o uso de ferramentas como pytest para testes unitários e de integração, e nbval para a validação de notebooks [Consulta do Usuário], reflete a natureza crítica dos sistemas empregados no domínio financeiro. Erros em previsões de mercado ou na lógica subjacente a estratégias de trading podem acarretar consequências financeiras adversas e significativas. No setor financeiro, a precisão, a confiabilidade e a auditabilidade dos sistemas são de importância primordial. Testes unitários asseguram que os componentes de software menores, como funções individuais, operem conforme o esperado. Testes de integração verificam se esses componentes interagem corretamente para produzir os resultados desejados. A validação de Jupyter Notebooks através de nbval garante que a lógica de análise, os cálculos e os resultados documentados no notebook permaneçam consistentes e reproduzíveis ao longo do desenvolvimento e após modificações no código base.[2] Adicionalmente, a utilização da AppTest API do Streamlit [4] permitirá a automação de testes da interface do usuário e do fluxo da aplicação, verificando se as interações e visualizações se comportam como projetado.

## 2. Fontes de Dados e Aquisição

### 2.1. Identificação das Fontes de Dados

O sucesso de qualquer modelo de previsão financeira depende intrinsecamente da qualidade e relevância dos dados de entrada. Para este projeto, os dados serão coletados de fontes públicas e robustas, combinando datasets históricos com dados atualizados via API.

- **Bitcoin (BTC-USD):**
  - **Dados históricos primários (Kaggle):**
    - A principal fonte de dados históricos para Bitcoin será o dataset "Top 10

Crypto Coin Historical Data 2014-2024" disponível no Kaggle.[6] Este dataset contém informações de preço diário (Open, High, Low, Close - OHLC) e Volume para BTC-USD, cobrindo o período de 2014 até (potencialmente) dezembro de 2024. Notavelmente, a fonte original destes dados é declarada como Yahoo Finance, o que pode simplificar o processo de sincronização e garantir consistência com os dados obtidos via API. O formato do arquivo é CSV.
        - Uma fonte alternativa, "Bitcoin Historical On-Chain Data 2014-2024" [7], também foi considerada. Embora rica em dados on-chain, como timestamp, open, close, volume e marketCap de 2014-06-27 a 2024-06-14, ela inclui muitas variáveis não explicitamente requeridas pelo escopo atual do projeto [1], podendo adicionar complexidade desnecessária na fase inicial. Portanto, o dataset de [6] será priorizado.
    - **Dados para sincronização e atualização (API):**
        - A biblioteca yfinance será utilizada para buscar os dados mais recentes de BTC-USD, utilizando o ticker "BTC-USD".[1] Isso garantirá que as análises e previsões sejam baseadas nas informações mais atuais disponíveis.
- **Apple (AAPL) ou Blue-Chip Alternativa:**
    - **Dados históricos primários (Kaggle):**
        - Para a ação da Apple (AAPL), o dataset "Apple Stock 2014-2024" de JP Kochar no Kaggle [9] será a fonte primária. Este dataset abrange o período de 2014-01-01 a 2024-01-25 e fornece dados OHLCV em formato CSV. Embora este dataset inclua indicadores técnicos pré-calculados, eles serão ignorados, pois o projeto exige o cálculo próprio desses indicadores.[1]
        - Outra opção, "Apple Stocks" de Pratham Jyot Singh [10], utiliza a Alpha Vantage API e inclui dados de dividendos e splits. Esta informação é crucial, pois destaca a necessidade de utilizar preços ajustados.
    - **Dados para sincronização e atualização (API):**
        - A API yfinance será empregada para obter os dados mais recentes e, fundamentalmente, os dados ajustados para dividendos e splits da AAPL, utilizando o ticker "AAPL".[1]

### 2.2. Estratégia de Coleta e Sincronização

A estratégia de coleta e sincronização visa criar um dataset unificado, preciso e atualizado para cada ativo:

1. **Download Inicial:** Os arquivos CSV selecionados do Kaggle ([6] para BTC-USD e [9] para AAPL) serão baixados para servirem como a base histórica.

2. **Aquisição de Dados Recentes e Ajustados via yfinance:**
    - A biblioteca yfinance será utilizada para buscar dados diários para "BTC-USD" e "AAPL".[8]
    - O período de busca via yfinance cobrirá desde a última data disponível nos CSVs do Kaggle até a data mais recente possível.
    - Para AAPL, é imperativo obter dados ajustados para splits de ações e pagamentos de dividendos. A yfinance facilita isso através do parâmetro auto_adjust=True na chamada history() ou utilizando a coluna 'Adj Close'.[12] Se os dados do Kaggle não estiverem ajustados, os dados ajustados do yfinance terão precedência ou serão usados para realizar o ajuste retrospectivo.
3. **Consolidação e Validação:**
    - Os dados históricos do Kaggle serão concatenados com os dados recentes/ajustados obtidos via yfinance.
    - Durante a concatenação, qualquer sobreposição de datas será tratada, priorizando os dados do yfinance devido à maior probabilidade de estarem ajustados e serem mais recentes.
    - Será garantido que o índice de datas seja contínuo, ordenado cronologicamente e sem duplicatas.
    - As colunas essenciais (Data, Open, High, Low, Close, Adj Close, Volume) serão padronizadas entre as fontes.

A utilização de dados do Kaggle como ponto de partida oferece um histórico extenso, mas estes são, por natureza, estáticos e podem não refletir todos os ajustes corporativos (como splits e dividendos) ou podem conter lacunas não documentadas. A API yfinance, por outro lado, é dinâmica e geralmente fornece dados já ajustados para tais eventos.[8] O estudo de caso menciona explicitamente "usar yfinance para garantir sincronização de datas" [1], o que sugere a importância de validar e, se necessário, corrigir os dados do Kaggle com uma fonte mais dinâmica. A melhor abordagem é tratar yfinance como a fonte de verdade para os preços ajustados e para o histórico mais recente. O Kaggle pode complementar para períodos mais antigos (início de 2014) caso yfinance apresente limitações na profundidade histórica para esses ativos específicos. Portanto, o pipeline de ingestão de dados deve incluir uma etapa de validação e reconciliação cuidadosa entre as duas fontes.

A necessidade de sincronizar dados de arquivos CSV estáticos com uma API dinâmica introduz uma complexidade gerenciável no pipeline de ingestão.[14] Uma política clara para a fusão de dados é essencial. Por exemplo, se um CSV do Kaggle contém dados até uma data D1 e a API yfinance fornece dados a partir de uma data D0 (onde D0 < D1), os dados do yfinance devem ser priorizados para o período de sobreposição,

dada a sua maior confiabilidade em termos de ajustes e atualizações. A frequência de execução do script de sincronização (ex: diariamente, antes de qualquer análise ou treinamento de modelo) também precisa ser definida para manter os dados relevantes.

Para a ação da Apple (AAPL), o ajuste de preços para dividendos e splits é absolutamente fundamental. Sem esse ajuste, a série temporal de preços não representaria corretamente o retorno real do investimento e poderia introduzir falsos padrões ou volatilidade que confundiriam os modelos de previsão. Saltos ou quedas abruptas nos preços devido a eventos corporativos como splits, se não ajustados, seriam interpretados erroneamente pelos modelos como movimentos de mercado genuínos. O uso da coluna 'Adj Close' (Preço de Fechamento Ajustado) é uma prática padrão em análises financeiras e será adotado.[12]

### 2.3. Tratamento de Dados e Versionamento

- Os datasets finais, combinados, limpos e ajustados para cada ativo, serão salvos em formatos eficientes como CSV ou Parquet. Estes arquivos serão armazenados em um diretório dedicado, por exemplo, data/processed/, para fácil acesso tanto pelo Jupyter Notebook quanto pelo aplicativo Streamlit.
- Para garantir a rastreabilidade e reprodutibilidade das análises e modelos, será considerado o versionamento dos datasets processados. Isso pode ser alcançado através de ferramentas como DVC (Data Version Control) ou, de forma mais simples, nomeando os arquivos de dados com timestamps ou números de versão.

A tabela a seguir resume as fontes de dados e suas características:

**Tabela 2.1: Visão Geral das Fontes de Dados**

| Ativo | Fonte Primária (Kaggle Link) | Fonte de Sincronização (Yahoo Finance Ticker) | Colunas Chave | Período Coberto Inicialmente | Estratégia de Atualização |
|---|---|---|---|---|---|
| BTC-USD | (https://www.kaggle.com/datasets/farhanali097/top-10-crypto- | BTC-USD | Date, Open, High, Low, Close, Volume | 2014 - Dez 2024 (Kaggle) | yfinance para dados recentes e validação. Priorizar |

| | | | | | |
|---|---|---|---|---|---|
| | coin-historical-data-2014-2024) [6] | | | | yfinance em sobreposiçõ es. |
| AAPL | (https://www.kaggle.com/datasets/jpkochar/apple-stock-2014-2024) [9] | AAPL | Date, Open, High, Low, Close, Adj Close, Volume | 2014 - Jan 2024 (Kaggle) | yfinance para dados recentes e preços ajustados (dividendos/ splits). Priorizar yfinance. |

Esta tabela centraliza as informações cruciais sobre a origem dos dados, o que é essencial para a reprodutibilidade, auditoria e para que qualquer membro da equipe possa entender rapidamente a proveniência e o manejo dos dados.

## 3. Configuração do Ambiente de Desenvolvimento

Uma configuração de ambiente consistente e bem definida é crucial para a colaboração eficiente e a reprodutibilidade dos resultados do projeto.

### 3.1. Ferramentas Recomendadas

- **Ambiente de Notebook:** Para a fase de exploração de dados, desenvolvimento de modelos e documentação inicial, serão utilizados Google Colab ou Kaggle Notebooks.[1] Estas plataformas são vantajosas por oferecerem ambientes pré-configurados com a maioria das bibliotecas de ciência de dados necessárias, além de facilitarem o compartilhamento e a colaboração.
- **Desenvolvimento Local:** O desenvolvimento do aplicativo Streamlit modular, incluindo a criação dos scripts Python e a suíte de testes pytest, será realizado preferencialmente em um ambiente de desenvolvimento local. IDEs como Visual Studio Code ou PyCharm são recomendados, pois oferecem ferramentas avançadas para gerenciamento de projetos, depuração e integração com controle de versão.
- **Controle de Versão:** O Git será utilizado para o controle de versão de todo o código-fonte do projeto, incluindo notebooks, scripts Python, arquivos de configuração e documentação. Isso permitirá o rastreamento de alterações, a colaboração entre desenvolvedores e a capacidade de reverter para versões anteriores, se necessário.
- **Ambiente Virtual:** Todas as dependências do projeto serão instaladas e

gerenciadas dentro de um ambiente virtual Python (utilizando venv ou conda).[15] Esta prática isola as dependências do projeto das instalações Python do sistema ou de outros projetos, prevenindo conflitos de versão e garantindo um ambiente de execução consistente.

### 3.2. Arquivo requirements.txt

Um arquivo requirements.txt será mantido na raiz do projeto para listar todas as bibliotecas Python necessárias e suas versões específicas. Este arquivo é fundamental para garantir que qualquer pessoa possa recriar o ambiente de desenvolvimento e execução de forma idêntica. Ele pode ser gerado ou atualizado usando o comando pip freeze > requirements.txt dentro do ambiente virtual ativado.

As bibliotecas principais a serem incluídas são [1]:

- **pandas**: Para manipulação, limpeza e análise de dados tabulares e séries temporais.
- **numpy**: Para operações numéricas eficientes, fundamental para arrays e matrizes.
- **yfinance**: Para download de dados históricos e recentes de mercado do Yahoo Finance.[8]
- **scikit-learn**: Para tarefas de pré-processamento (ex: MinMaxScaler, StandardScaler), divisão de dados para séries temporais (TimeSeriesSplit) e cálculo de métricas de avaliação de modelos.[1]
- **tensorflow**: Framework de deep learning que será utilizado para construir e treinar os modelos RNN (SimpleRNN, LSTM, GRU) através de sua API de alto nível Keras.[1]
- **ta ou TA-Lib**: Para o cálculo de indicadores técnicos. TA-Lib [19] é uma biblioteca C com wrappers Python, conhecida por sua abrangência e eficiência, sendo a preferida. A biblioteca ta é uma alternativa puramente Python, mais fácil de instalar, caso surjam dificuldades com TA-Lib.
- **matplotlib, plotly, seaborn**: Para a criação de visualizações estáticas e interativas. Plotly é particularmente útil para gráficos dinâmicos no dashboard Streamlit.[1]
- **streamlit**: Framework para a construção do dashboard interativo.[1]
- **pytest**: Framework para a escrita e execução de testes unitários e de integração [Consulta do Usuário].
- **pytest-cov**: Plugin para pytest que gera relatórios de cobertura de testes, ajudando a identificar partes do código não testadas.[21]
- **nbval**: Plugin para pytest que permite a validação de Jupyter Notebooks, garantindo que as células executem corretamente e produzam saídas

consistentes.[2]

A biblioteca TA-Lib, embora extremamente útil e performática para o cálculo de uma vasta gama de indicadores técnicos, pode apresentar desafios de instalação em alguns sistemas operacionais. Isso ocorre porque ela depende de uma biblioteca C subjacente que precisa ser compilada ou pré-instalada.[20] O documento de arquitetura ou um guia complementar deverá fornecer instruções claras para a instalação do TA-Lib em ambientes comuns (Windows, macOS, Linux), ou, como alternativa, sugerir o uso da biblioteca ta. A ta é uma opção puramente Python, mais simples de instalar (pip install ta), mas pode não cobrir a mesma extensão de indicadores ou ter a mesma performance otimizada do TA-Lib. Em ambientes como Google Colab e Kaggle Notebooks, a instalação e uso do TA-Lib costumam ser mais diretos.

**Tabela 3.1: Bibliotecas Principais e Finalidade**

| Biblioteca | Versão Sugerida | Finalidade no Projeto | Comando de Instalação (Observações) |
|---|---|---|---|
| pandas | >=1.5.0 | Manipulação e análise de dados, séries temporais | pip install pandas |
| numpy | >=1.23.0 | Operações numéricas, arrays | pip install numpy |
| yfinance | >=0.2.0 | Download de dados do Yahoo Finance | pip install yfinance |
| scikit-learn | >=1.2.0 | Pré-processamento, TimeSeriesSplit, métricas | pip install scikit-learn |
| tensorflow | >=2.10.0 | Modelagem RNN com Keras | pip install tensorflow |
| TA-Lib | >=0.4.24 | Cálculo de indicadores técnicos (preferencial) | Requer instalação prévia da biblioteca C. Ver documentação oficial. Ex: conda install -c conda-forge ta-lib ou brew install |

| | | | |
|---|---|---|---|
| | | | ta-lib no macOS. |
| ta | >=0.10.0 | Cálculo de indicadores técnicos (alternativa) | pip install ta |
| matplotlib | >=3.6.0 | Visualização de dados (estática) | pip install matplotlib |
| plotly | >=5.10.0 | Visualização de dados (interativa) | pip install plotly |
| seaborn | >=0.12.0 | Visualização estatística de dados | pip install seaborn |
| streamlit | >=1.25.0 | Desenvolvimento do dashboard interativo | pip install streamlit |
| pytest | >=7.0.0 | Framework de testes | pip install pytest |
| pytest-cov | >=4.0.0 | Relatórios de cobertura de testes | pip install pytest-cov |
| nbval | >=0.9.0 | Testes de Jupyter Notebooks | pip install nbval |

Esta tabela serve como um guia centralizado para as dependências do projeto, assegurando que todos os desenvolvedores utilizem versões compatíveis das bibliotecas e compreendam o papel de cada ferramenta no ecossistema do projeto.

### 3.3. Estrutura de Pastas do Projeto

Para manter o projeto organizado, facilitar a navegação e promover boas práticas de desenvolvimento, sugere-se a seguinte estrutura de pastas, inspirada em convenções comuns e nas necessidades específicas do projeto [23]:

agile_capital_forecast/

```
├── data/
│   ├── raw/          # CSVs originais do Kaggle (BTC e AAPL)
│   └── processed/    # Dados limpos, combinados e ajustados para BTC e AAPL
├── notebooks/
│   └── financial_forecasting_eda_modeling.ipynb # Notebook principal para EDA e modelagem
├── src/
│   ├── data_ingestion/
│   │   ├── __init__.py
│   │   └── loader.py         # Funções para carregar e sincronizar dados
│   ├── preprocessing/
│   │   ├── __init__.py
│   │   ├── scalers_transformers.py # Scalers e transformadores de dados
│   │   └── feature_engineering.py # Funções para cálculo de indicadores técnicos
│   ├── modeling/
│   │   ├── __init__.py
│   │   ├── rnn_models.py     # Definições das arquiteturas RNN (SimpleRNN, LSTM, GRU)
│   │   ├── prediction.py     # Funções para carregar modelos e fazer previsões
│   │   └── strategy_simulation.py # Lógica para simulação de estratégia de trading
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main_app.py       # Ponto de entrada do aplicativo Streamlit
│   │   ├── pages/            # Módulos para cada página do app Streamlit
│   │   │   ├── __init__.py
│   │   │   ├── 01_BTC_Forecast.py
│   │   │   └── 02_AAPL_Forecast.py
│   │   └── components/       # Componentes reutilizáveis da UI do Streamlit
│   │       ├── __init__.py
│   │       ├── plotting.py     # Funções para gerar gráficos
│   │       └── ui_elements.py  # Funções para criar widgets de UI
│   └── utils/
│       ├── __init__.py
│       └── helpers.py        # Funções utilitárias diversas
├── tests/
│   ├── unit/                 # Testes unitários para módulos em src/
│   │   ├── preprocessing/
│   │   │   └── test_feature_engineering.py
│   │   └── modeling/
```

```
│   │   └── test_rnn_models.py
│   ├── integration/          # Testes de integração entre módulos
│   │   └── test_app_flow.py  # Teste de fluxo da aplicação Streamlit com AppTest
│   └── notebook_tests/       # Configurações e possivelmente fixtures para nbval
│       └── conftest.py       # Pode ser usado para configurar nbval ou pytest
├── models/                   # Modelos treinados e serializados (ex:.h5)
├── reports/                  # Relatório final em PDF, figuras geradas
│   └── figures/
├──.gitignore                 # Arquivos e pastas a serem ignorados pelo Git
├── requirements.txt          # Lista de dependências Python
└── TODO.md                   # Lista de tarefas do projeto
```

Esta estrutura promove a separação de responsabilidades, facilita a localização de artefatos do projeto e suporta a modularidade e testabilidade da aplicação.

## 4. Desenvolvimento do Jupyter Notebook: Pipeline de Análise e Modelagem

O Jupyter Notebook (financial_forecasting_eda_modeling.ipynb) será o ambiente central para a exploração interativa dos dados, desenvolvimento e avaliação inicial dos modelos de previsão. Ele servirá como um documento vivo do processo de pesquisa e modelagem.

### 4.1. Coleta e Ingestão de Dados

Nesta seção inicial do notebook, serão implementados scripts Python para carregar os datasets pré-processados, que foram gerados e armazenados na pasta data/processed/ conforme descrito na Seção 2.3. Após o carregamento, uma verificação primária será realizada para confirmar o número de registros, a presença e o tipo de todas as colunas esperadas (Data, Open, High, Low, Close, Adj Close, Volume, e quaisquer indicadores técnicos já calculados se uma abordagem iterativa for usada).

### 4.2. Análise Exploratória de Dados (EDA) e Engenharia de Features

A EDA é fundamental para entender as características das séries temporais e para guiar a engenharia de features e a modelagem.

- **Visualizações:**
  - Serão gerados gráficos das séries temporais de preços (focando no 'Adj Close' para AAPL e 'Close' para BTC) e do volume de negociação para ambos os ativos.[1] Bibliotecas como matplotlib e plotly (para interatividade) serão

empregadas. Exemplos de plotagem de séries temporais podem ser encontrados em referências como [26], que utiliza seaborn (sns.lineplot).
    - A distribuição dos retornos diários (calculados como a variação percentual do preço de fechamento ajustado) será visualizada através de histogramas e box plots para identificar a forma da distribuição, outliers e volatilidade.
- **Cálculo de Indicadores Técnicos:**
    - Conforme os requisitos [1], os seguintes indicadores técnicos serão calculados e adicionados como features exógenas aos DataFrames:
        - **SMA (Médias Móveis Simples):** Diferentes janelas (ex: 20 dias, 50 dias) para capturar tendências de curto e médio prazo.
        - **EMA (Médias Móveis Exponenciais):** Semelhante à SMA, mas dando maior peso aos preços mais recentes, tornando-as mais reativas.
        - **RSI (Índice de Força Relativa):** Um oscilador de momento que mede a velocidade e a mudança dos movimentos de preços, tipicamente em uma janela de 14 dias, para identificar condições de sobrecompra ou sobrevenda.
        - **MACD (Convergência/Divergência de Médias Móveis):** Um indicador de momento que segue tendências mostrando a relação entre duas EMAs de preços.
    - A biblioteca TA-Lib é a preferida para esses cálculos devido à sua robustez e otimização.[19] Um exemplo de uso seria df = talib.SMA(df['AdjClose'], timeperiod=20).[19] Caso haja dificuldades com a instalação do TA-Lib, a biblioteca ta poderá ser utilizada como alternativa.[1]
- **Testes de Estacionariedade:**
    - A estacionariedade é uma propriedade importante para muitas técnicas de modelagem de séries temporais. O teste Augmented Dickey-Fuller (ADF) será aplicado às séries de preços (e/ou retornos) para verificar se possuem média e variância constantes ao longo do tempo.[24] A função adfuller da biblioteca statsmodels.tsa.stattools é adequada para isso.[24]
    - Se as séries originais (níveis de preço) se mostrarem não estacionárias (o que é comum para preços de ativos), será aplicada a diferenciação (calculando a diferença entre observações consecutivas, df = df['AdjClose'].diff()) e o teste ADF será reaplicado na série diferenciada até que a estacionariedade seja alcançada.[26]
- **Análise de Autocorrelação:**
    - Gráficos da Função de Autocorrelação (ACF) e da Função de Autocorrelação Parcial (PACF) serão plotados para as séries (preços, retornos, e posteriormente para os resíduos dos modelos). Estes gráficos ajudam a identificar a presença de autocorrelação e a ordem de possíveis

componentes autorregressivos (AR) e de médias móveis (MA) nos dados.[24] As funções plot_acf e plot_pacf de statsmodels.graphics.tsaplots serão utilizadas.[28]

### 4.3. Pré-processamento de Dados

Esta etapa prepara os dados para serem alimentados nos modelos de machine learning.

- **Tratamento de Datas Faltantes e Feriados:**
  - Será realizada uma verificação da continuidade do índice de datas. Dados financeiros diários podem ter lacunas devido a fins de semana e feriados de mercado.
  - Para preencher valores ausentes em colunas numéricas (OHLCV, indicadores), a técnica de forward fill (fillna(method='ffill') em pandas) será aplicada.[1] Esta abordagem é comum em dados financeiros e assume que o valor mais recente conhecido se mantém até que um novo valor seja observado. É crucial que os dados de diferentes ativos (BTC e AAPL) estejam alinhados temporalmente, especialmente se forem usados em modelos multivariados, embora o escopo inicial sugira modelos univariados por ativo.
- **Normalização de Dados:**
  - As features numéricas (preços, volume, indicadores técnicos calculados) serão normalizadas ou padronizadas para uma escala comum. Isso é importante para o bom desempenho de redes neurais, que podem ser sensíveis à escala das entradas. MinMaxScaler (escalonando para um intervalo, e.g., ) ou StandardScaler (média zero e desvio padrão unitário) da scikit-learn serão utilizados.[1]
  - Um ponto crítico é evitar o "data leakage": o scaler (seja MinMaxScaler ou StandardScaler) deve ser ajustado (fit) **exclusivamente** no conjunto de treinamento. Subsequentemente, esse scaler ajustado será usado para transformar (transform) os conjuntos de treinamento, validação e teste.[32] Aplicar fit em todo o dataset antes de dividir introduziria informação do futuro (validação/teste) no processo de treinamento.
- **Criação de Janelas de Tempo para RNNs:**
  - Modelos RNN requerem que os dados de entrada sejam formatados como sequências. Para prever os próximos 14 dias, as séries temporais serão transformadas em janelas deslizantes:
    - **Entrada (X):** Uma sequência de N observações passadas (contendo preços, volume, indicadores técnicos). O valor de N (tamanho da janela de lookback) é um hiperparâmetro a ser definido.

        - **Saída (Y):** Os 14 valores de preço (ex: 'Adj Close') para os 14 dias seguintes à última observação da janela de entrada.
    - A função tf.keras.utils.timeseries_dataset_from_array é uma ferramenta eficiente e conveniente para criar esses datasets de janelas.[34] Ela permite especificar sequence_length (o N da janela de entrada), targets (os 14 valores futuros), sampling_rate (para subamostragem dentro de uma sequência, se necessário), sequence_stride (o passo entre o início de janelas consecutivas) e batch_size.
    - Alternativamente, uma função customizada, como demonstrado em [63], poderia ser implementada para criar as sequências (def create_dataset(data, time_step=60):...), mas a utilidade do Keras é geralmente preferível por sua otimização e integração com o TensorFlow.

### 4.4. Modelagem com Redes Neurais Recorrentes (RNNs)

Serão exploradas três arquiteturas principais de RNNs para a tarefa de previsão.

- **Arquiteturas [1]:**
    - **SimpleRNN:** A forma mais básica de RNN, útil como baseline para entender o comportamento em dados sequenciais.
    - **LSTM (Long Short-Term Memory):** Uma arquitetura de RNN mais avançada, projetada para superar o problema do desaparecimento do gradiente e capturar dependências de longo prazo nos dados. É amplamente utilizada em previsão de séries temporais financeiras.[36]
    - **GRU (Gated Recurrent Unit):** Uma variação do LSTM, com uma arquitetura de portão mais simples, o que pode torná-la computacionalmente mais eficiente e mais rápida para treinar, muitas vezes com performance comparável ao LSTM.[36]
- **Estrutura dos Modelos:**
    - Os modelos consistirão em 1 ou 2 camadas recorrentes (SimpleRNN, LSTM ou GRU).
    - **Dropout:** Camadas de Dropout serão inseridas entre as camadas recorrentes e/ou antes da camada de saída para regularização, ajudando a prevenir o overfitting, um problema comum em dados financeiros ruidosos.[1]
    - **Stateful RNNs:** A opção de usar RNNs stateful será explorada.[1] Em LSTMs/GRUs stateful, o último estado de uma amostra em um batch é usado como o estado inicial para a amostra correspondente no batch seguinte. Isso pode ser útil para séries temporais muito longas onde as dependências se estendem entre batches. Se stateful=True for usado, o batch_size deve ser consistente na criação do dataset (timeseries_dataset_from_array) e na

definição da camada RNN (ex: LSTM(units, stateful=True, batch_input_shape=(batch_size, timesteps, features))). Além disso, model.reset_states() pode precisar ser chamado manualmente após cada época de treinamento ou entre sequências que não são temporalmente contíguas, para evitar que o estado de uma sequência "vaze" para outra não relacionada.
    - **Camada de Saída:** Uma camada Dense no final da rede produzirá as 14 previsões para o horizonte de 14 dias. Se a previsão for para um único valor por dia (ex: preço de fechamento), esta camada terá 14 neurônios e, tipicamente, uma função de ativação linear (padrão para regressão).
- **Implementação em TensorFlow/Keras:**
    - Os modelos serão definidos usando a API Sequencial (keras.Sequential) ou a API Funcional do Keras, que faz parte do TensorFlow. Exemplos de arquiteturas podem ser encontrados em estudos como [63] (para SimpleRNN) e [36] (para LSTM/GRU em contextos de previsão financeira).

### 4.5. Treinamento e Validação

O processo de treinamento e validação será cuidadosamente estruturado para garantir que os modelos generalizem bem para dados não vistos.

- **Estratégia de Divisão de Dados [1]:**
    - Os dados serão divididos cronologicamente em conjuntos de treinamento, validação e teste. É crucial que os dados de validação e teste sejam posteriores aos dados de treinamento para simular um cenário de previsão realista.
    - sklearn.model_selection.TimeSeriesSplit será utilizado para gerar os índices para a validação cruzada específica para séries temporais.[1] Este método garante que as dobras de treinamento sempre precedam as dobras de teste.
    - A validação em janelas deslizantes (rolling windows) também será considerada.[1] Nesta abordagem, o modelo é treinado em uma janela de dados históricos e testado na janela seguinte. A janela de treinamento então "desliza" para frente, incorporando os dados da janela de teste anterior, e o modelo é re-treinado (ou atualizado). Isso simula o re-treinamento periódico do modelo à medida que novos dados se tornam disponíveis.
- **Compilação do Modelo:**
    - **Otimizador:** Adam é uma escolha comum e robusta para o treinamento de redes neurais e será o ponto de partida.
    - **Função de Perda (Loss Function):** Sendo esta uma tarefa de regressão (prever valores de preço), o Mean Squared Error (MSE) é uma função de

perda apropriada. Mean Absolute Error (MAE) também pode ser considerado.

- **Callbacks** [1]**:**
  - EarlyStopping: Este callback monitorará uma métrica de desempenho no conjunto de validação (tipicamente val_loss). Se a métrica não apresentar melhora (ou seja, não diminuir, no caso de val_loss) por um número especificado de épocas (patience), o treinamento será interrompido prematuramente para evitar overfitting.[44] A opção restore_best_weights=True pode ser usada para carregar os pesos do modelo da época com o melhor desempenho na métrica monitorada.
  - ReduceLROnPlateau: Este callback reduzirá a taxa de aprendizado (learning rate) do otimizador quando a métrica de validação estagnar. Uma taxa de aprendizado menor pode ajudar o modelo a convergir para um mínimo melhor quando estiver perto de um platô na superfície de perda.[1]

### 4.6. Avaliação e Comparação de Modelos

Após o treinamento, os modelos serão rigorosamente avaliados em relação à sua capacidade de previsão e ao seu potencial para gerar estratégias de trading lucrativas.

- **Métricas de Forecast** [1]**:**
  - As seguintes métricas de erro serão calculadas no conjunto de teste para cada ativo (BTC, AAPL) e para cada modelo treinado (SimpleRNN, LSTM, GRU), e opcionalmente para um modelo ARIMA como baseline:
    - **RMSE (Root Mean Squared Error):** $ \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2} $. Penaliza erros maiores mais significativamente.
    - **MAE (Mean Absolute Error):** $ \frac{1}{N}\sum_{i=1}^{N}|Y_i - \hat{Y}_i| $. Mede o erro absoluto médio.
    - **MAPE (Mean Absolute Percentage Error):** $ \frac{100%}{N}\sum_{i=1}^{N}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right| $. Expressa o erro como uma porcentagem, útil para comparar a precisão entre séries de diferentes escalas.
- **Tabela 4.1: Métricas de Avaliação de Modelos**

| Métrica | Fórmula/Definição Concisa | Interpretação | Unidade |
|---|---|---|---|
| RMSE | $ \sqrt{\text{média}((Y_{\text{real}} - Y_{\text{previsto}})^2) | Erro quadrático médio, na mesma unidade da variável prevista. Menor é | Unidade da variável prevista (ex: USD) |

| | | | |
|---|---|---|---|
| | } $ | melhor. | |
| MAE | $ \text{média}( | Y_{\text{real}} - Y_{\text{previsto}} | ) $ |
| MAPE | $ \text{média}( | (Y_{\text{real}} - Y_{\text{previsto}}) / Y_{\text{real}} | ) \times 100% $ |
| Sharpe Ratio | $ (R_p - R_f) / \sigma_p $ | Retorno ajustado ao risco. Maior é melhor. | Adimensional |

*Nota: R_p é o retorno do portfólio/estratégia, R_f é a taxa livre de risco (pode ser assumida como 0), \\sigma_p é o desvio padrão do excesso de retorno.*
Esta tabela fornece um glossário claro das métricas usadas, assegurando que todos os stakeholders compreendam como o desempenho do modelo é medido e comparado.

- **Simulação de Estratégia de Trading** [1]**:**
  - Uma estratégia de trading "long-only" simples será definida com base nos sinais de previsão de 14 dias.
    - **Exemplo de Sinal:** Se a previsão média para os próximos 14 dias indicar um aumento de X% em relação ao preço atual, uma posição de compra (long) é iniciada ou mantida. Se a previsão indicar uma queda ou um aumento abaixo de um limiar Y%, a posição é fechada (se existente) e nenhuma nova posição é aberta.
    - O desempenho desta estratégia será comparado com uma estratégia de referência "buy-and-hold" (comprar o ativo no início do período de teste e mantê-lo até o final).
  - Para implementar a simulação, podem ser utilizados frameworks de backtesting como Backtesting.py [46] ou Backtrader.[47] Alternativamente, uma simulação mais simplificada pode ser implementada diretamente em pandas, calculando os retornos diários da estratégia com base nos sinais e nos preços reais.

- **Análise de Risco** [1]**:**
  - O **Sharpe Ratio** será calculado para a estratégia baseada em forecast e para a estratégia buy-and-hold.[48] Esta métrica avalia o retorno de um investimento em comparação com seu risco. Uma taxa livre de risco (ex: rendimento de títulos do governo de curto prazo) será considerada; se não houver um valor específico, pode-se assumir como 0 para simplificação, focando no excesso de retorno sobre a volatilidade.

A tarefa de prever 14 dias à frente no mercado financeiro é inerentemente desafiadora. A incerteza tende a se acumular com o aumento do horizonte de previsão, o que significa que os erros nas previsões de curto prazo podem se propagar e amplificar nas previsões de longo prazo. Um modelo pode exibir boa performance para prever o dia D+1, mas ter um desempenho significativamente inferior para o dia D+14. Esta característica pode exigir modelos com maior capacidade de capturar tendências de longo prazo ou o uso de features que reflitam dinâmicas de mercado mais amplas. A arquitetura do modelo RNN, especialmente sua camada de saída (que precisará gerar 14 valores) e a função de perda escolhida, deve ser cuidadosamente projetada para lidar com essa previsão multi-horizonte. A avaliação do modelo também deve considerar essa complexidade, possivelmente reportando métricas de erro de forma desagregada para diferentes pontos dentro do horizonte de 14 dias (ex: D+1, D+7, D+14 separadamente) para entender onde a precisão do modelo se deteriora.

O tamanho da janela de entrada (sequence_length ou time_step) para os modelos RNN é um hiperparâmetro crítico. Esta janela define a quantidade de histórico que o modelo "vê" para fazer uma previsão. Uma janela muito curta pode impedir que o modelo capture dependências temporais de mais longo prazo, que são relevantes em mercados financeiros. Por outro lado, uma janela excessivamente longa pode introduzir ruído (informações passadas irrelevantes para a previsão atual) e aumentar significativamente a complexidade computacional e o tempo de treinamento. A "memória" efetiva do modelo está ligada a essa janela. Para mercados que exibem ciclos mais longos ou são influenciados por fatores macroeconômicos com maior inércia, janelas maiores podem ser benéficas. Para estratégias de trading de prazo mais curto, janelas menores podem ser suficientes. A escolha ótima do tamanho da janela é dependente das características específicas dos dados de cada ativo e deve ser determinada empiricamente, idealmente através de um processo de validação cruzada ou experimentação sistemática.

Modelos de Redes Neurais Recorrentes, como LSTMs e GRUs, são frequentemente descritos como "caixas-pretas" devido à dificuldade em interpretar diretamente como

chegam às suas previsões. Enquanto o foco principal do projeto é a performance preditiva, o relatório final (detalhado na Seção 6) precisará abordar essa limitação de interpretabilidade. Para um cliente como um fundo de investimento, que pode precisar justificar suas decisões de alocação de capital, a incapacidade de explicar o "porquê" por trás de um sinal de trade gerado por um modelo complexo pode ser uma barreira. Modelos estatísticos mais tradicionais, como ARIMA (que pode ser usado como baseline), são geralmente mais interpretáveis. Se os modelos RNN demonstrarem um ganho de performance substancial sobre alternativas mais simples e interpretáveis, esse ganho pode justificar a troca. Técnicas emergentes de explicabilidade em IA (XAI), como SHAP (SHapley Additive exPlanations) ou LIME (Local Interpretable Model-agnostic Explanations) adaptadas para séries temporais, são áreas de pesquisa ativas, mas sua implementação robusta pode estar além do escopo inicial deste projeto. A discussão no relatório deve, no mínimo, reconhecer essa troca entre performance e interpretabilidade.

## 5. Desenvolvimento do Aplicativo Streamlit Modular

O aplicativo Streamlit fornecerá uma interface interativa para que o fundo "Agile Capital" visualize as previsões de preços, os indicadores técnicos e a performance simulada das estratégias de trading. A modularidade será um princípio chave em seu desenvolvimento.

### 5.1. Arquitetura do Aplicativo

A aplicação será estruturada de forma modular para promover a organização do código, facilitar a manutenção e permitir testes mais eficientes.[23] A estrutura de pastas proposta na Seção 3.3 será seguida:

- src/app/main_app.py: Este será o script principal executado pelo comando streamlit run. Ele conterá a configuração global da página (título, ícone, layout), a lógica de navegação entre as diferentes seções ou "páginas" do aplicativo e possivelmente elementos de UI comuns a todas as páginas, como um cabeçalho ou rodapé.
- src/app/pages/: Este diretório abrigará os scripts Python individuais para cada página principal da aplicação, aproveitando o mecanismo nativo de multipágina do Streamlit.[50] Por exemplo:
  - 01_BTC_Forecast.py: Página dedicada às previsões e análises do Bitcoin.
  - 02_AAPL_Forecast.py: Página dedicada às previsões e análises da Apple. Os prefixos numéricos nos nomes dos arquivos ajudam a controlar a ordem de aparição na barra de navegação do Streamlit.[50]
- src/app/components/: Este diretório conterá módulos com funções reutilizáveis

para construir a interface do usuário:
    - plotting.py: Funções para gerar os diversos gráficos (ex: séries temporais de preços, indicadores técnicos, retornos de estratégia) utilizando bibliotecas como Plotly para interatividade.
    - ui_elements.py: Funções para criar widgets de UI comuns que podem ser usados em múltiplas páginas (ex: seletores de data, seletores de ativos, caixas de entrada de parâmetros).
- Módulos de src/ Reutilizados:
    - src/data_ingestion/loader.py: Funções para carregar os dados processados dos ativos.
    - src/preprocessing/feature_engineering.py: Funções para calcular os indicadores técnicos necessários para visualização.
    - src/modeling/prediction.py: Funções para carregar os modelos de RNN treinados e realizar inferências (gerar previsões).

A reutilização de código entre o Jupyter Notebook e o aplicativo Streamlit é um objetivo importante. Funções de pré-processamento, engenharia de features e até mesmo partes da lógica de modelagem (como carregar um modelo salvo e fazer uma previsão) desenvolvidas e testadas no ambiente do notebook devem ser refatoradas em módulos Python dentro da estrutura src/. Isso evita a duplicação de código, o que é propenso a erros e inconsistências, e garante que tanto a análise exploratória quanto a aplicação interativa operem sobre a mesma lógica de negócios testada.

**Tabela 5.1: Módulos da Aplicação Streamlit**

| Nome do Arquivo do Módulo | Localização (Caminho) | Responsabilidade Principal | Dependências Chave (outros módulos) |
|---|---|---|---|
| main_app.py | src/app/ | Ponto de entrada, configuração global, navegação principal. | streamlit, pages/*, components/* |
| 01_BTC_Forecast.py | src/app/pages/ | UI e lógica específica para visualização das previsões de BTC. | streamlit, components/plotting.py, modeling/prediction.py, data_ingestion/loader.py |

| | | | |
|---|---|---|---|
| 02_AAPL_Forecast.py | src/app/pages/ | UI e lógica específica para visualização das previsões de AAPL. | streamlit, components/plotting.py, modeling/prediction.py, data_ingestion/loader.py |
| plotting.py | src/app/components/ | Funções para gerar gráficos (Plotly, Matplotlib). | streamlit, plotly, pandas |
| ui_elements.py | src/app/components/ | Funções para criar widgets de UI reutilizáveis. | streamlit |
| loader.py | src/data_ingestion/ | Carregamento e fornecimento de dados de ativos. | pandas |
| feature_engineering.py | src/preprocessing/ | Cálculo de indicadores técnicos para visualização. | pandas, ta / TA-Lib |
| prediction.py | src/modeling/ | Carregamento de modelos treinados e geração de previsões. | tensorflow, numpy, pandas |

Esta tabela serve como um mapa da arquitetura do código da aplicação Streamlit, auxiliando os desenvolvedores a entender a organização das funcionalidades e as interdependências entre os módulos. É um recurso valioso para a manutenção e para a integração de novos desenvolvedores ao projeto.

### 5.2. Componentes da Interface do Usuário (UI)

A interface do usuário será projetada para ser intuitiva e informativa, permitindo uma fácil exploração dos resultados do modelo.[1]

- **Navegação:** Uma barra lateral (st.sidebar) será utilizada para a navegação principal entre as diferentes seções do aplicativo, como as páginas dedicadas às previsões de BTC e AAPL.[50]
- **Seletores:**

    - Dentro de cada página de ativo (ou em uma página unificada, se essa abordagem for escolhida posteriormente), widgets como st.selectbox ou st.radio permitirão ao usuário selecionar parâmetros específicos, como o modelo de previsão a ser visualizado (SimpleRNN, LSTM, GRU), ou diferentes períodos para os indicadores técnicos.
    - Um st.date_input poderá ser incluído para permitir que o usuário selecione uma data histórica específica para a qual deseja ver a previsão que teria sido gerada (assumindo que os modelos e os dados de entrada daquela época estejam disponíveis e versionados).
- **Visualizações:**
    - **Gráfico de Previsão Principal:** Um gráfico interativo (preferencialmente com Plotly) exibirá a série temporal dos preços históricos do ativo e as previsões para os próximos 14 dias.
    - **Gráficos de Indicadores Técnicos:** Gráficos separados ou sobrepostos mostrarão os valores dos indicadores técnicos (SMA, EMA, RSI, MACD) calculados sobre os dados históricos. Se pertinente, os valores previstos desses indicadores (se o modelo os gerar ou se puderem ser derivados das previsões de preço) também poderão ser exibidos.
    - **Performance da Estratégia:** Uma tabela (st.dataframe ou st.table) ou um gráfico (ex: curva de capital) resumirá a performance da estratégia de trading simulada, incluindo o retorno acumulado, drawdowns máximos e o Sharpe Ratio, comparando-a com a estratégia buy-and-hold.
- **Interatividade:**
    - Os gráficos permitirão zoom e pan, e exibirão tooltips com valores exatos ao passar o mouse sobre os pontos de dados.
    - Controles como st.slider ou st.number_input poderão permitir que o usuário ajuste dinamicamente os períodos dos indicadores técnicos visualizados (ex: mudar a janela de uma SMA de 20 para 50 dias e ver o gráfico atualizar).

### 5.3. Lógica do Backend

O backend do aplicativo Streamlit orquestrará o carregamento de dados, a aplicação de modelos e a preparação dos resultados para exibição.

- **Carregamento de Modelos:** Os modelos RNN treinados, que foram serializados e salvos (ex: no formato H5 do Keras ou como TensorFlow SavedModel) na pasta models/, serão carregados dinamicamente quando necessário. A função responsável por isso residirá em src/modeling/prediction.py.
- **Pipeline de Inferência:**
    1. **Obtenção de Dados:** Quando o usuário selecionar um ativo e solicitar uma

previsão, os dados mais recentes para esse ativo serão obtidos utilizando as funções em src/data_ingestion/loader.py.
2. **Pré-processamento e Engenharia de Features:** As mesmas etapas de pré-processamento (normalização com os scalers ajustados no treino) e engenharia de features (cálculo de indicadores técnicos) aplicadas durante o treinamento do modelo serão replicadas nos dados recentes. Funções de src/preprocessing/ serão utilizadas.
3. **Formatação da Entrada:** Os dados processados serão formatados na estrutura de janela de entrada (sequência de dias passados) esperada pelo modelo RNN carregado.
4. **Geração de Previsões:** O modelo carregado será usado para gerar as previsões para os próximos 14 dias. Esta lógica estará em src/modeling/prediction.py.
5. **Pós-processamento:** As previsões normalizadas serão revertidas para a escala original de preços (usando o método inverse_transform do scaler). Opcionalmente, indicadores técnicos podem ser calculados sobre os preços previstos para visualização.

- **Caching:** Para otimizar a performance e a responsividade do aplicativo, o Streamlit oferece mecanismos de caching que serão extensivamente utilizados [4]:
  - @st.cache_data: Usado para armazenar em cache o resultado de funções que retornam dados serializáveis (como DataFrames do pandas). Ideal para o carregamento de dados históricos.
  - @st.cache_resource: Usado para armazenar em cache objetos globais que não são facilmente serializáveis, como conexões de banco de dados ou, crucialmente neste caso, os modelos de machine learning carregados. O uso eficaz do caching é vital, pois modelos de deep learning podem ser computacionalmente caros para carregar e para executar inferências. Se cada interação do usuário no Streamlit (como mudar um parâmetro de visualização) acionasse um recarregamento completo do modelo e uma re-inferência em todos os dados, a aplicação se tornaria excessivamente lenta e impraticável. O caching garante que essas operações custosas sejam executadas apenas uma vez ou quando os seus inputs relevantes realmente mudarem, melhorando significativamente a experiência do usuário.

Para interações mais complexas dentro do aplicativo, como manter as seleções do usuário entre diferentes execuções de script (que ocorrem a cada interação com um widget) ou armazenar resultados de simulações parciais para evitar recálculos, o st.session_state será uma ferramenta indispensável.[4] Streamlit re-executa o script da página de cima para baixo a cada interação do usuário. Para preservar informações

ou o estado da aplicação entre essas re-execuções (por exemplo, o ativo que o usuário selecionou, os resultados de uma previsão que já foi calculada, ou o estado de um formulário multi-etapas), o st.session_state atua como um dicionário persistente durante a sessão do usuário. Sem um gerenciamento de estado adequado, a experiência do usuário pode ser frustrante (ex: seleções perdidas) e a aplicação pode se tornar ineficiente ao recomputar desnecessariamente informações já disponíveis. A arquitetura da aplicação deve, portanto, prever o uso criterioso do st.session_state para gerenciar o fluxo de informações e o estado da interface.

## 6. Diretrizes para o Relatório do Modelo

O relatório técnico, com uma extensão sugerida de 8 a 10 páginas [1], será o documento formal que apresentará a metodologia, os resultados, as análises e as conclusões do projeto de previsão de ativos. Ele deve ser claro, conciso e fornecer insights acionáveis para o "Agile Capital".

### 6.1. Estrutura Sugerida

A estrutura a seguir, baseada em práticas comuns para relatórios de projetos de machine learning [53] e nos requisitos específicos do projeto [1], é recomendada:

1. **Introdução:**
    - Apresentação do contexto do problema: a necessidade do fundo "Agile Capital" de melhorar sua estratégia de alocação dinâmica através da previsão de preços de BTC-USD e AAPL.
    - Definição clara dos objetivos do projeto: desenvolver modelos de previsão para um horizonte de 14 dias e avaliar sua aplicabilidade.
    - Breve resumo da abordagem metodológica adotada (coleta de dados, tipos de modelos, avaliação).
2. **Metodologia:**
    - **Fontes de Dados e Aquisição:** Descrição detalhada das fontes de dados utilizadas (Kaggle, Yahoo Finance), o período coberto, as colunas relevantes e o processo de coleta, limpeza, ajuste (para splits/dividendos) e sincronização dos dados.
    - **Análise Exploratória de Dados (EDA):** Apresentação dos principais achados da EDA, incluindo visualizações de séries temporais, distribuições de retornos, testes de estacionariedade e análises de autocorrelação.
    - **Engenharia de Features:** Detalhamento dos indicadores técnicos calculados (SMA, EMA, RSI, MACD), suas fórmulas (ou referência à biblioteca TA-Lib/ta), os períodos utilizados e a justificativa para sua inclusão como features exógenas.

- **Pré-processamento de Dados:** Explicação dos métodos de tratamento de dados faltantes (ex: forward fill), a técnica de normalização/padronização escolhida (MinMaxScaler/StandardScaler) e como foi aplicada para evitar data leakage, e o processo de criação de janelas de tempo para alimentar os modelos RNN.
- **Arquitetura dos Modelos RNN:** Descrição pormenorizada das arquiteturas SimpleRNN, LSTM e GRU implementadas, incluindo o número de camadas recorrentes, o número de unidades por camada, o uso de camadas de Dropout (e as taxas de dropout), as funções de ativação utilizadas e a estrutura da camada de saída para a previsão de 14 dias. Se RNNs stateful foram usadas, explicar a configuração.
- **Configuração de Treinamento e Validação:** Especificação do otimizador (ex: Adam) e da função de perda (ex: MSE) utilizados. Detalhamento da estratégia de divisão de dados (treino/validação/teste cronológico, TimeSeriesSplit, validação em janelas deslizantes) e dos callbacks empregados (EarlyStopping, ReduceLROnPlateau) com seus respectivos parâmetros.

3. **Resultados e Comparativo de Modelos:**
   - Apresentação clara das métricas de performance (RMSE, MAE, MAPE) para cada modelo (SimpleRNN, LSTM, GRU, e baseline ARIMA se houver) e para cada ativo (BTC, AAPL). Os resultados devem ser organizados em tabelas comparativas e, sempre que possível, visualizados através de gráficos (ex: bar charts comparando RMSE entre modelos).
   - Discussão sobre qual modelo apresentou o melhor desempenho geral para cada ativo e em quais métricas. Análise estatística da significância das diferenças de performance, se aplicável e viável.
4. **Simulação de Estratégia de Trading:**
   - Descrição detalhada da estratégia de trading "long-only" implementada, baseada nos sinais de previsão de 14 dias (ex: critérios de compra/manutenção/venda).
   - Apresentação dos resultados da simulação de backtesting: retorno acumulado da estratégia, drawdowns máximos, número de trades, taxa de acerto (se aplicável).
   - Cálculo e interpretação do Sharpe Ratio da estratégia baseada em forecast, comparando-o com o Sharpe Ratio de uma estratégia buy-and-hold para o mesmo período e ativos.
5. **Discussão** [1]**:** Esta é uma seção crítica que vai além da simples apresentação de resultados, focando na interpretação e nas implicações.
   - **Insights Principais:** Quais são as principais conclusões extraídas dos

resultados? O que esses achados significam na prática para a "Agile Capital"? Quais modelos, features ou técnicas se mostraram mais (ou menos) eficazes e por quê?

- **Volatilidade vs. Dependência Temporal:** Análise de quão bem os modelos RNN conseguiram capturar (ou não) os padrões em dados financeiros voláteis, especialmente os picos e crashes característicos de criptomoedas como o Bitcoin.[54]
- **Relevância das Variáveis Exógenas:** Avaliação do impacto dos indicadores técnicos (features exógenas) na performance das previsões. Eles realmente adicionaram valor preditivo significativo em comparação com modelos que usariam apenas dados endógenos (preços passados)?
- **Comparativo: Modelos Tradicionais vs. Deep Learning:** Se um modelo tradicional como ARIMA foi implementado como baseline, discutir onde ele se destacou ou falhou em comparação com os modelos RNN.[57] Existem cenários ou características dos dados onde modelos mais simples ainda são competitivos ou preferíveis?
- **Aplicabilidade para Gerenciamento de Risco e Trade Real:** Com base na acurácia das previsões e na performance da estratégia simulada, discutir a viabilidade de usar os forecasts para decisões de trading reais. Quais são os riscos envolvidos? Como a incerteza das previsões (especialmente para um horizonte de 14 dias) deve ser considerada? Que frações do capital poderiam ser alocadas com base nesses sinais, considerando o risco?
- **Desafios do Overfitting em Dados Financeiros:** Dada a natureza inerentemente ruidosa e não estacionária dos dados financeiros, e a capacidade dos modelos de deep learning de se ajustarem a padrões complexos (incluindo ruído), esta subseção deve ser aprofundada. Detalhar *como* o overfitting foi monitorado (ex: comparando as curvas de perda de treino e validação) e as técnicas específicas empregadas para mitigá-lo (ex: Dropout, EarlyStopping, regularização L1/L2 se utilizada, robustez da estratégia de validação cruzada temporal).[54] Apresentar evidências (como gráficos de loss vs val_loss) do efeito dessas técnicas é altamente recomendável.
- **Limitações dos Modelos RNN:** Discussão honesta sobre as limitações inerentes às RNNs, como o problema do "vanishing/exploding gradient" (mesmo com LSTM/GRU) [55], a dificuldade de treinamento, a sensibilidade a hiperparâmetros e, crucialmente, sua natureza de "caixa-preta" que dificulta a interpretabilidade das previsões.[54]

6. **Conclusões e Recomendações:**
   - Sumário conciso dos principais achados do projeto.

    - Recomendações práticas para a "Agile Capital" sobre como as previsões e os modelos desenvolvidos poderiam (ou não) ser integrados em sua estratégia de alocação de carteira. Estas recomendações devem ir além de simplesmente declarar "o modelo X é o melhor". Devem abordar como as previsões podem ser usadas dentro de um processo de tomada de decisão, considerando a confiança associada às previsões, o perfil de risco do fundo, e possivelmente como combinar sinais de diferentes modelos ou ativos. A conexão entre os resultados técnicos e as implicações práticas para investimento é fundamental.
    - Sugestões para trabalhos futuros que poderiam aprimorar a solução (ex: inclusão de outros ativos, exploração de features alternativas como sentimento de notícias, desenvolvimento de modelos mais avançados como Transformers, otimização de portfólio baseada nas previsões).
7. **Apêndice (Opcional):**
    - Pode incluir trechos de código chave, configurações detalhadas de hiperparâmetros dos modelos finais, ou resultados de análises suplementares.

### 6.2. Estilo e Tom

O relatório deve manter um tom técnico, objetivo e rigoroso, com todas as afirmações e conclusões baseadas nas evidências apresentadas pelos dados e resultados dos modelos. Ao mesmo tempo, deve ser escrito de forma clara e acessível o suficiente para que um público com conhecimento em finanças, mas talvez não especializado em machine learning avançado, possa compreender os principais achados e suas implicações.[60] A linguagem deve ser precisa, evitando jargões desnecessários ou explicando-os quando introduzidos.

## 7. Estratégia de Testes com Pytest

Uma estratégia de testes robusta é essencial para garantir a qualidade, confiabilidade e manutenibilidade do projeto, especialmente considerando sua aplicação no domínio financeiro. Serão implementados testes em diferentes níveis, utilizando pytest como o framework principal e nbval para a validação do Jupyter Notebook.

### 7.1. Testes do Jupyter Notebook

- **Ferramenta:** nbval, um plugin do pytest.[2]
- **Objetivo:** Assegurar que o Jupyter Notebook (financial_forecasting_eda_modeling.ipynb) execute integralmente sem erros e que as saídas de células críticas (como shapes de DataFrames após

transformações, valores de métricas de avaliação de modelos após uma execução de referência, ou a ausência de exceções) permaneçam consistentes mesmo após modificações no código subjacente ou nos dados base. Isso valida o notebook como um documento reproduzível e uma ferramenta de análise confiável.

- **Implementação:**
  1. Instalação: pip install nbval.
  2. Execução: Os testes do notebook serão executados através do comando pytest --nbval notebooks/financial_forecasting_eda_modeling.ipynb.
  3. Configuração: nbval permite configurar o comportamento da validação por célula usando metadados. Por exemplo, pode-se optar por ignorar a saída de células que geram plots (que podem ter pequenas variações não funcionais) ou especificar tolerâncias para comparações numéricas.

É importante notar que, embora nbval seja excelente para garantir a integridade do notebook como um artefato de documentação e exploração, a lógica de negócios crítica (como o cálculo preciso de indicadores técnicos, as transformações de dados complexas, e a própria implementação dos modelos RNN) deve, idealmente, residir em módulos Python reutilizáveis na pasta src/. Esses módulos serão, então, importados e utilizados pelo notebook. Esta abordagem permite que a lógica central seja testada de forma mais granular e rigorosa através de testes unitários com pytest, enquanto o notebook se concentra em demonstrar o uso dessa lógica e apresentar os resultados. Se a lógica complexa estiver contida apenas dentro das células do notebook, testá-la exaustivamente com nbval torna-se mais desafiador e menos eficaz.

### 7.2. Testes do Aplicativo Streamlit

Os testes para o aplicativo Streamlit serão divididos em testes unitários para os módulos de backend e componentes, e testes de integração/UI utilizando a API AppTest do Streamlit.

- **Framework:** pytest.[15]
- **Testes Unitários:**
  - **Alvo:** Funções individuais e classes dentro dos módulos Python que compõem a aplicação Streamlit (ex: funções em src/preprocessing/feature_engineering.py, src/modeling/prediction.py, src/app/components/plotting.py, src/data_ingestion/loader.py).
  - **Exemplos de Cenários de Teste:**
    - Para feature_engineering.py: Verificar se a função de cálculo de SMA retorna os valores corretos para uma série de entrada de teste conhecida.

Testar casos de borda (ex: dados insuficientes para a janela da SMA).
        - Para scalers_transformers.py: Confirmar que uma função de normalização de dados transforma corretamente um array de entrada e que a transformação inversa retorna aos valores originais (dentro de uma tolerância).
        - Para prediction.py: Testar se a função de carregamento de modelo lida graciosamente com caminhos de arquivo inválidos ou modelos corrompidos. Verificar se a função de previsão retorna saídas com o shape esperado.
        - Para plotting.py: Testar se as funções de plotagem geram objetos de figura válidos (ex: um objeto plotly.graph_objects.Figure). Não se trata de verificar a imagem renderizada pixel a pixel, mas sim a estrutura e os dados do objeto figura.
    - **Isolamento com Mocks:** A biblioteca unittest.mock (ou equivalentes em pytest) será utilizada para criar "mocks" (objetos dublês) de dependências externas durante os testes unitários.[62] Isso permite isolar a unidade de código sendo testada. Por exemplo, ao testar uma função que normalmente leria um arquivo grande ou faria uma chamada de API, essas operações podem ser mockadas para retornar dados pré-definidos, tornando o teste mais rápido e determinístico.
- **Testes de Integração e UI com AppTest:**
    - **Alvo:** Verificar o fluxo de dados e as interações entre múltiplos módulos do aplicativo Streamlit, culminando no teste do comportamento da aplicação como um todo, de forma "headless" (sem a necessidade de um navegador real).
    - **Ferramenta Específica:** A API AppTest (streamlit.testing.v1.AppTest) é projetada para este propósito.[4] Ela permite:
        - Inicializar uma simulação da aplicação a partir de seu script principal (ex: AppTest.from_file("src/app/main_app.py")).
        - Interagir programaticamente com os widgets da UI (ex: at.selectbox(label="Select Asset").select("BTC").run()).
        - Inspecionar o estado da aplicação (at.session_state) e os elementos renderizados na tela (ex: assert at.header.value == "Bitcoin Price Forecast").
        - Verificar a presença de exceções ou warnings (assert not at.exception).
    - **Exemplos de Cenários de Teste de Integração/UI:**
        - Simular a seleção do ativo "BTC" na interface e verificar se o gráfico de previsão correspondente é carregado e se os dados exibidos (ex: último preço histórico, primeira previsão) são consistentes com o esperado.

    - Testar o fluxo completo para um ativo: carregar dados iniciais -> simular interação do usuário para acionar o cálculo de indicadores -> verificar se os indicadores são exibidos -> acionar a geração de previsão -> verificar se a previsão é exibida.
    - Verificar se o tratamento de erros (ex: seleção de uma data para a qual não há dados) resulta em mensagens apropriadas na UI.
    - Testar a navegação entre páginas (se aplicável) e a persistência de estado via st.session_state.

Embora a AppTest API seja uma ferramenta poderosa para testar aplicações Streamlit, testar interfaces de usuário (UIs) pode ser inerentemente complexo. Os testes de UI devem focar na lógica de apresentação e na correção das interações funcionais, em vez de se prenderem a detalhes de layout ou aparência visual que podem mudar frequentemente. Manter os componentes da UI relativamente simples e, crucialmente, separar a lógica de negócios principal em módulos backend (que são testados unitariamente de forma independente) ajuda a reduzir a complexidade e a fragilidade dos testes de UI. Testes de UI são mais robustos quando verificam o estado resultante da aplicação ou os dados exibidos após uma interação, em vez de dependerem da exata aparência visual dos elementos.

Os testes unitários, especialmente aqueles que lidam com funções de processamento de dados ou lógica de modelo, frequentemente necessitarão de pequenos conjuntos de dados de teste, conhecidos como "fixtures". Esses fixtures devem conter entradas de amostra e os resultados esperados correspondentes. Por exemplo, para testar uma função que calcula um indicador técnico, seria fornecida uma pequena série temporal de entrada e o valor esperado do indicador para essa série. Esses dados de teste devem ser concisos, representativos de casos típicos e de borda, e gerenciados como parte do código de teste, preferencialmente dentro do diretório tests/ ou em subdiretórios apropriados.

### 7.3. Organização e Execução dos Testes

- **Convenção de Nomenclatura:** Os arquivos de teste seguirão a convenção test_*.py ou *_test.py e serão localizados dentro da pasta tests/ na raiz do projeto.[15]
- **Estrutura de Pastas de Teste:** A estrutura de subpastas dentro de tests/ (ex: tests/unit/preprocessing/) espelhará a estrutura da pasta src/ para facilitar a organização e localização dos testes correspondentes a cada módulo da aplicação.
- **Execução:** Todos os testes (unitários, de integração, e do notebook via nbval)

devem ser executáveis através de um único comando no terminal, como pytest.

- **Relatórios de Cobertura:** pytest-cov será configurado para gerar relatórios de cobertura de código.[21] Estes relatórios são essenciais para identificar partes do código que não estão sendo exercitadas pelos testes, ajudando a direcionar esforços para aumentar a cobertura e a confiança na aplicação.

### 7.4. Tabela 7.1: Plano de Testes do Projeto

| Categoria do Teste | Ferramenta Principal | Componente(s) Alvo | Descrição do Cenário de Teste Chave | Critério de Sucesso Esperado |
|---|---|---|---|---|
| Notebook Validation | nbval (plugin pytest) | notebooks/financial_forecasting_eda_modeling.ipynb | Executar todas as células do notebook. Verificar consistência de saídas chave (ex: shapes de dataframes, métricas após processamento). | Notebook executa sem erros. Saídas especificadas correspondem às saídas de referência. |
| Unitário: Data Ingestion | pytest | src/data_ingestion/loader.py | Carregar dados de um arquivo CSV de teste. Lidar com arquivo inexistente. Sincronizar com dados de API mockada. | Dados carregados corretamente. Exceção apropriada para arquivo inexistente. Dados sincronizados conforme esperado. |
| Unitário: Feature Engineering | pytest | src/preprocessing/feature_engineering.py | Calcular SMA, EMA, RSI, MACD para uma série de entrada conhecida. | Valores dos indicadores correspondem aos valores calculados manualmente/referência. |

| | | | | |
|---|---|---|---|---|
| Unitário: Preprocessing | pytest | src/preprocessing/scalers_transformers.py | Normalizar e desnormalizar dados usando MinMaxScaler com parâmetros de treino fixos. | Dados normalizados no intervalo . Dados desnormalizados retornam aos valores originais. |
| Unitário: Model Definition | pytest | src/modeling/rnn_models.py | Criar instâncias de SimpleRNN, LSTM, GRU. Verificar a estrutura das camadas. | Modelos são criados sem erro. Número de camadas e unidades conforme especificado. |
| Unitário: Prediction Logic | pytest | src/modeling/prediction.py | Carregar um modelo mockado. Fazer uma previsão com dados de entrada formatados. | Previsão tem o shape esperado (ex: 14 dias). |
| Unitário: Streamlit Components | pytest | src/app/components/plotting.py, ui_elements.py | Gerar um objeto de gráfico Plotly com dados de entrada mockados. Criar um widget seletor. | Objeto gráfico é criado. Widget é configurado corretamente. |
| Integração: Data Pipeline | pytest | loader.py -> feature_engineering.py -> scalers_transformers.py | Simular o fluxo de dados desde o carregamento até o pré-processamento final para entrada no modelo. | DataFrame resultante tem as colunas e o formato esperados. |
| Integração/UI: Streamlit App BTC | pytest com AppTest | src/app/main_app.py, src/app/pages/01_BTC_Forecast. | Simular seleção de BTC. Verificar se o título da página é correto. Verificar | Título correto. Elemento do gráfico de previsão existe na árvore de |

| | | | | |
|---|---|---|---|---|
| | | py | se o gráfico de previsão é renderizado (presença do elemento). | elementos. |
| Integração/UI: Streamlit App AAPL | pytest com AppTest | src/app/main_app.py, src/app/pages/02_AAPL_Forecast.py | Simular seleção de AAPL. Interagir com um seletor de período de indicador. Verificar se o gráfico do indicador atualiza (inspecionando dados do gráfico se possível, ou um elemento textual que mude). | Interação com widget é bem-sucedida. Atualização esperada na UI ocorre. |
| Integração: Full App Flow | pytest com AppTest | Aplicação Streamlit completa | Simular um fluxo de usuário: selecionar ativo, rodar previsão, verificar saída numérica chave (ex: primeira previsão). | Previsão exibida corresponde a um valor esperado (dentro de uma tolerância, se a previsão for mockada ou de um modelo de referência simples). Nenhuma exceção durante o fluxo. |

Este plano de testes fornece um framework para a estratégia de qualidade do projeto, garantindo uma cobertura abrangente e que todos os tipos de testes importantes sejam considerados e planejados. Ele ajudará a rastrear o progresso da implementação dos testes e a identificar áreas que necessitam de maior atenção. A inclusão de tarefas específicas no arquivo TODO.md para a escrita e execução de cada categoria de teste, até que todos os testes passem, será um passo crucial na

gestão do projeto.

## 8. Considerações Finais e Próximos Passos

Este Documento de Arquitetura foi elaborado para servir como um guia técnico abrangente para o desenvolvimento dos três entregáveis principais do projeto: o Jupyter Notebook de análise e modelagem, o Relatório do Modelo e o Aplicativo Streamlit interativo. A estrutura detalhada, as escolhas de tecnologia e as metodologias propostas visam atender aos requisitos do fundo "Agile Capital" para a previsão de preços de Bitcoin e Apple.

A ênfase em uma abordagem modular, tanto na organização do código quanto na concepção do sistema, juntamente com uma estratégia de testes rigorosa, são pilares fundamentais para o sucesso e a sustentabilidade deste projeto. A modularidade facilitará a manutenção, a escalabilidade e a adaptação a futuras necessidades do mercado financeiro, enquanto os testes garantirão a confiabilidade e a precisão da solução.

É provável que o desenvolvimento real, especialmente no que tange à modelagem de machine learning e ao design da interface do usuário, seja um processo iterativo. Embora este documento estabeleça uma arquitetura sólida, ela deve ser encarada como uma fundação flexível, capaz de acomodar os aprendizados e o feedback que surgirão durante a implementação. A capacidade de, por exemplo, experimentar diferentes configurações de camadas RNN ou adicionar um novo tipo de visualização no Streamlit sem a necessidade de uma reengenharia completa é uma vantagem significativa da abordagem modular proposta.

Embora a escalabilidade não seja um requisito explícito imediato, as escolhas de design, como o uso eficiente de caching no Streamlit e a clara separação de componentes, foram feitas com uma visão de potencial crescimento futuro. Se o protótipo demonstrar valor, a "Agile Capital" poderá desejar expandir a solução para incluir mais ativos, atender a um número maior de usuários ou incorporar modelos preditivos mais complexos. Uma arquitetura bem pensada desde o início facilitará significativamente tais expansões. Por exemplo, se a inferência do modelo se tornar um gargalo de performance com o aumento do uso, um módulo de prediction.py bem definido e desacoplado poderá ser mais facilmente otimizado ou até mesmo migrado para um serviço de inferência dedicado.

Recomenda-se uma revisão cuidadosa deste documento por toda a equipe envolvida antes do início efetivo das atividades de desenvolvimento. O próximo passo imediato será utilizar as diretrizes e a estrutura aqui apresentadas para popular o arquivo

TODO.md com tarefas detalhadas e atribuíveis, delineando o caminho para a execução do projeto.

**Referências citadas**


1. AI Application - Estudo_de_Caso_IA.docx
2. [2001.04808] Testing with Jupyter notebooks: NoteBook VALidation (nbval) plug-in for pytest, acessado em junho 1, 2025, https://ar5iv.labs.arxiv.org/html/2001.04808
3. Jupyter Notebooks - Computational Modelling Tools Workshops, acessado em junho 1, 2025, https://computationalmodelling.bitbucket.io/tools/JupyterNotebooks/Teaching_Material/JupyterNotebooks_Manual.html
4. App testing - Streamlit Docs, acessado em junho 1, 2025, https://docs.streamlit.io/develop/api-reference/app-testing
5. st.testing.v1.AppTest - Streamlit Docs, acessado em junho 1, 2025, https://docs.streamlit.io/develop/api-reference/app-testing/st.testing.v1.apptest
6. Top 10 Crypto-Coin Historical Data (2014-2024) - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/datasets/farhanali097/top-10-crypto-coin-historical-data-2014-2024
7. Bitcoin Historical On-Chain Data (2014-2024) - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/datasets/arthurgomesbubolz/bitcoin-historical-on-chain-data-2014-2024
8. How to Retrieve Stock Market Data with Yahoo Finance API in Python - Omi AI, acessado em junho 1, 2025, https://www.omi.me/blogs/api-guides/how-to-retrieve-stock-market-data-with-yahoo-finance-api-in-python-1
9. apple-stock-2014-2024 - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/datasets/jpkochar/apple-stock-2014-2024
10. Apple Stocks - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/datasets/prathamjyotsingh/apple-stocks
11. API Reference — yfinance - GitHub Pages, acessado em junho 1, 2025, https://ranaroussi.github.io/yfinance/reference/index.html
12. backtest_tutorial/YFinance_Tutorial.ipynb at main - GitHub, acessado em junho 1, 2025, https://github.com/hudson-and-thames/backtest_tutorial/blob/main/YFinance_Tutorial.ipynb
13. How to Download Historical Stock Prices from Yahoo Finance - YouTube, acessado em junho 1, 2025, https://www.youtube.com/watch?v=S39Lx-Lh3fQ
14. Historical Market Data Sources - QuantInsti Blog, acessado em junho 1, 2025, https://blog.quantinsti.com/financial-market-data-providers/
15. Good Integration Practices - pytest documentation, acessado em junho 1, 2025, https://docs.pytest.org/en/stable/explanation/goodpractices.html

16. TongjiFinLab/FinTSB: Financial Time Series Benchmark (FinTSB): A Comprehensive and Practical Benchmark for Financial Time Series Forecasting - GitHub, acessado em junho 1, 2025, https://github.com/TongjiFinLab/FinTSBenchmark
17. Python for Finance: Time Series Analysis - MLQ.ai, acessado em junho 1, 2025, https://blog.mlq.ai/python-for-finance-time-series-analysis/
18. yfinance: Get your own Stock Data - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/code/mustafacicek/yfinance-get-your-own-stock-data
19. Introduction to using of TA-Lib - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/code/sndorburian/introduction-to-using-of-ta-lib
20. TA-Lib/ta-lib-python: Python wrapper for TA-Lib (http://ta-lib.org/). - GitHub, acessado em junho 1, 2025, https://github.com/TA-Lib/ta-lib-python
21. Setting up testing with pytest and uv - Python Developer Tooling Handbook, acessado em junho 1, 2025, https://pydevtools.com/handbook/tutorial/setting-up-testing-with-pytest-and-uv/
22. Calculate Technical Indicators in Python with TA-Lib. - TraderMade, acessado em junho 1, 2025, https://tradermade.com/tutorials/calculate-technical-indicators-in-python-with-ta-lib
23. How to Structure and Organise a Streamlit App - Towards Data Science, acessado em junho 1, 2025, https://towardsdatascience.com/how-to-structure-and-organise-a-streamlit-app-e66b65ece369/
24. Time Series Exploratory Data Analysis In Python - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/code/youssef19/time-series-exploratory-data-analysis-in-python
25. Stationarity in Python - JDEconomics, acessado em junho 1, 2025, https://www.jdeconomics.com/python-tutorials/stationarity-in-python
26. Time Series Analysis & Visualization in Python | GeeksforGeeks, acessado em junho 1, 2025, https://www.geeksforgeeks.org/time-series-data-visualization-in-python/
27. Complete Guide on Time Series Analysis in Python - Kaggle, acessado em junho 1, 2025, https://www.kaggle.com/code/prashant111/complete-guide-on-time-series-analysis-in-python/notebook
28. How to Calculate Autocorrelation in Python? - GeeksforGeeks, acessado em junho 1, 2025, https://www.geeksforgeeks.org/how-to-calculate-autocorrelation-in-python/
29. Mastering Autocorrelation Techniques in Time Series Statistical Analysis - Number Analytics, acessado em junho 1, 2025, https://www.numberanalytics.com/blog/mastering-autocorrelation-techniques
30. How to Handle Missing Data in Pandas Efficiently? - Console Flare, acessado em junho 1, 2025, https://consoleflare.com/blog/handle-missing-data-in-pandas/
31. Working with Missing Data in Pandas | GeeksforGeeks, acessado em junho 1,

2025, https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/
32. What is Feature Scaling and Why is it Important? - Analytics Vidhya, acessado em junho 1, 2025, https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/
33. Is normalizing before train-test split a data leakage in time series forecasting? - Reddit, acessado em junho 1, 2025, https://www.reddit.com/r/MLQuestions/comments/1jzwqt5/is_normalizing_before_traintest_split_a_data/
34. timeseries_dataset_from_array - TensorFlow for R, acessado em junho 1, 2025, https://tensorflow.rstudio.com/reference/keras/timeseries_dataset_from_array
35. Timeseries data loading - Keras, acessado em junho 1, 2025, https://keras.io/api/data_loading/timeseries/
36. Classification Modeling with RNN-based, Random Forest, and XGBoost for Imbalanced Data: A Case of Early Crash Detection in ASEAN - arXiv, acessado em junho 1, 2025, https://arxiv.org/pdf/2406.07888?
37. Comparative Analysis of LSTM, GRU, and ARIMA Models for Stock Market Price Prediction, acessado em junho 1, 2025, https://www.researchgate.net/publication/379175870_Comparative_Analysis_of_LSTM_GRU_and_ARIMA_Models_for_Stock_Market_Price_Prediction
38. Stateful LSTM model training in Keras - Yumi's Blog, acessado em junho 1, 2025, https://fairyonice.github.io/Stateful-LSTM-model-training-in-Keras.html
39. Understanding Keras LSTMs: Role of Batch-size and Statefulness - Stack Overflow, acessado em junho 1, 2025, https://stackoverflow.com/questions/48491737/understanding-keras-lstms-role-of-batch-size-and-statefulness
40. TimeSeriesSplit — scikit-learn 1.6.1 documentation, acessado em junho 1, 2025, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
41. Scikit-Learn Time Series Split - Rasgo, acessado em junho 1, 2025, https://www.rasgoml.com/feature-engineering-tutorials/scikit-learn-time-series-split
42. Cross-validation Tutorial - TimeGPT Foundational model for time series forecasting and anomaly detection - Nixtla, acessado em junho 1, 2025, https://www.nixtla.io/docs/tutorials-validation-tutorials-cross_validation
43. An In-Depth Guide to Time Series Cross-Validation and Sliding Forecasts - hashnode.dev, acessado em junho 1, 2025, https://saimaharana.hashnode.dev/an-in-depth-guide-to-time-series-cross-validation-and-sliding-forecasts
44. EarlyStopping - Keras, acessado em junho 1, 2025, https://keras.io/api/callbacks/early_stopping/
45. Build a Time Series Forecasting Model Using TensorFlow Keras and GridDB, acessado em junho 1, 2025, https://griddb.net/en/blog/keras-griddb/
46. Backtesting.py - Backtest trading strategies in Python, acessado em junho 1, 2025, https://kernc.github.io/backtesting.py/

47. Algorithmic_Trading_with_Python/Lecture 07_ Strategy Testing.ipynb at main - GitHub, acessado em junho 1, 2025, https://github.com/AliHabibnia/Algorithmic_Trading_with_Python/blob/main/Lecture%2007_%20Strategy%20Testing.ipynb
48. Sharpe ratio and Sortino ratio | Python, acessado em junho 1, 2025, https://campus.datacamp.com/courses/financial-trading-in-python/performance-evaluation-4?ex=8
49. Sharpe Ratio Explained: Formula, Calculation in Excel & Python, and Examples, acessado em junho 1, 2025, https://blog.quantinsti.com/sharpe-ratio-applications-algorithmic-trading/
50. Overview of multipage apps - Streamlit Docs, acessado em junho 1, 2025, https://docs.streamlit.io/develop/concepts/multipage-apps/overview
51. Building a dashboard in Python using Streamlit, acessado em junho 1, 2025, https://blog.streamlit.io/crafting-a-dashboard-app-in-python-using-streamlit/
52. Personal Finance Dashboard - Show the Community! - Streamlit, acessado em junho 1, 2025, https://discuss.streamlit.io/t/personal-finance-dashboard/73731
53. CS 391L Machine Learning Project Report Format, acessado em junho 1, 2025, https://www.cs.utexas.edu/~mooney/cs391L/paper-template.html
54. Neural Networks in Financial Forecasting: Benefits and Challenges - ResearchGate, acessado em junho 1, 2025, https://www.researchgate.net/publication/386729277_Neural_Networks_in_Financial_Forecasting_Benefits_and_Challenges
55. ROLE OF LSTM AND RNN IN PREDICTION OF S&P 500 TRENDS AND FORECASTING - ijprems.com, acessado em junho 1, 2025, https://www.ijprems.com/uploadedfiles/paper/issue_9_september_2024/36123/final/fin_ijprems1727891332.pdf
56. Why Recurrent Neural Networks (RNNs) Dominate Sequential Data Analysis - Shelf.io, acessado em junho 1, 2025, https://shelf.io/blog/recurrent-neural-networks/
57. THE COMPARISON OF ARIMA AND RNN FOR FORECASTING GOLD FUTURES CLOSING PRICES | BAREKENG - OJS UNPATTI, acessado em junho 1, 2025, https://ojs3.unpatti.ac.id/index.php/barekeng/article/download/13888/9607/
58. Comparison Between ARIMA and LSTM-RNN for VN-Index Prediction - ResearchGate, acessado em junho 1, 2025, https://www.researchgate.net/publication/338741921_Comparison_Between_ARIMA_and_LSTM-RNN_for_VN-Index_Prediction
59. Recurrent Neural Networks (RNNs) for Time Series Predictions | Encord, acessado em junho 1, 2025, https://encord.com/blog/time-series-predictions-with-recurrent-neural-networks/
60. Financial Forecasting, Risk, and Valuation: Accounting for the Future, acessado em junho 1, 2025, https://care-mendoza.nd.edu/assets/152206/penmanpaper.pdf
61. Financial Forecasting Methods with Examples, acessado em junho 1, 2025, https://www.fe.training/free-resources/financial-modeling/financial-forecasting-methods-with-examples/

62. Setup and tear down test resource in streamlit with pytest, acessado em junho 1, 2025, https://discuss.streamlit.io/t/setup-and-tear-down-test-resource-in-streamlit-with-pytest/61998
63. Time Series Forecasting using Recurrent Neural Networks (RNN) in TensorFlow, acessado em junho 1, 2025, https://www.geeksforgeeks.org/time-series-forecasting-using-recurrent-neural-networks-rnn-in-tensorflow/