Metodología para Data Science: Optimización de Estrategias de Inversión

Este repositorio contiene el código y recursos para el curso de Metodología para Data Science, enfocado en el desarrollo de estrategias óptimas de inversión para fondos provenientes de AFP utilizando técnicas de Machine Learning y optimización.

Caso de Uso: Optimización de Portafolio de Inversión

El proyecto se centra en desarrollar una estrategia de inversión óptima para fondos retirados de AFP, utilizando:

- Análisis de datos históricos de diferentes instrumentos financieros
- Algoritmos de Machine Learning para predicción de rendimientos
- Técnicas de optimización para la construcción de portafolios
- Evaluación de riesgo y retorno esperado
- Backtesting de estrategias

Objetivos

1. Analizar el comportamiento histórico de diferentes activos financieros
2. Desarrollar modelos predictivos para estimar rendimientos futuros
3. Implementar algoritmos de optimización de portafolio
4. Evaluar y comparar diferentes estrategias de inversión
5. Generar recomendaciones basadas en el perfil de riesgo del inversionista

Estructura del Repositorio

```
.
├── .github/            # Configuraciones de GitHub (workflows,
settings)
├── src/               # Código fuente del proyecto
│   ├── data/          # Datos históricos de mercados financieros
│   ├── notebooks/     # Jupyter notebooks para análisis y backtesting
│   ├── models/        # Modelos de ML y optimización
│   ├── tests/         # Tests unitarios e integración
│   │   ├── unit/     # Tests unitarios
│   │   └── integration/ # Tests de integración
│   ├── utils/         # Funciones de utilidad
│   └── visualization/ # Visualizaciones y dashboards financieros
├── .gitignore        # Archivos y directorios ignorados por git
├── requirements.txt   # Dependencias del proyecto
└── README.md         # Este archivo
```

Configuración del Entorno

1. Clonar el repositorio:

```
git clone
https://github.com/aladelca/metodologia_para_data_science_20251.git
cd metodologia_para_data_science_20251
```

2. Crear un entorno virtual:

```
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
```

3. Instalar dependencias:

```
pip install -r requirements.txt
```

4. Configurar pre-commit hooks (IMPORTANTE):

```
# Instalar pre-commit
pip install pre-commit

# Instalar los hooks en el repositorio
pre-commit install

# Verificar la instalación
pre-commit --version

# (Opcional) Ejecutar los hooks manualmente
pre-commit run --all-files
```

   Los hooks se ejecutarán automáticamente en cada commit e incluyen:

   - Formateo de código (Black)
   - Verificación de estilo (Flake8)
   - Ordenamiento de imports (isort)
   - Verificación de tipos (mypy)
   - Ejecución de tests (pytest)
   - Verificación de dependencias

Flujo de Trabajo con Git

1. Crear una nueva rama para tu trabajo:

```
git checkout -b feature/[tus-iniciales]/[descripcion]
```

2. Realizar cambios y commits:

```
git add .
git commit -m "Descripción detallada de los cambios"
```

3. Subir cambios y crear Pull Request:

```
git push origin feature/[tus-iniciales]/[descripcion]
```

4. Esperar la revisión del instructor (@aladelca)

Metodología de Desarrollo (CRISP-DM)

1. **Comprensión del Negocio**
    - Definición de objetivos de inversión
    - Identificación de restricciones y perfil de riesgo
    - Establecimiento de métricas de éxito
    - Planificación del proyecto
2. **Comprensión de los Datos**
    - Recopilación de datos financieros históricos
    - Exploración inicial de instrumentos disponibles
    - Verificación de calidad de datos
    - Identificación de fuentes adicionales de información
3. **Preparación de los Datos**
    - Limpieza y tratamiento de datos faltantes
    - Cálculo de indicadores financieros
    - Feature engineering para análisis técnico
    - Normalización y transformación de variables
    - Preparación de conjuntos de entrenamiento y validación
4. **Modelado**
    - Desarrollo de modelos predictivos
        - Predicción de rendimientos
        - Estimación de volatilidad
        - Análisis de correlaciones
    - Implementación de algoritmos de optimización
        - Estrategias de factor investing

        - Optimización con restricciones de riesgo
    - Validación cruzada y ajuste de hiperparámetros

5. **Evaluación**
    - Backtesting de estrategias
    - Evaluación de robustez
    - Análisis de costos de transacción
    - Validación con stakeholders

6. **Implementación**
    - Documentación detallada de modelos y estrategias
    - Desarrollo de pipelines de producción
    - Monitoreo de rendimiento
    - Procedimientos de rebalanceo
    - Guías de implementación práctica

Entregables por Fase

1. Comprensión del Negocio

- Documento de objetivos y restricciones
- Plan de proyecto
- Definición de KPIs

2. Comprensión de los Datos

- Informe de calidad de datos
- Catálogo de fuentes de datos
- Análisis exploratorio inicial

3. Preparación de los Datos

- Pipeline de procesamiento de datos
- Documentación de features generados
- Conjuntos de datos procesados

4. Modelado

- Notebooks con implementación de modelos
- Documentación de parámetros y decisiones
- Resultados de validación

5. Evaluación

- Reportes de rendimiento
- Análisis de riesgo
- Comparativas con benchmarks

6. Implementación

- Código documentado
- Manual de usuario
- Procedimientos de mantenimiento

Convenciones de Código y Calidad

Pre-commit Hooks

El proyecto utiliza pre-commit hooks para asegurar la calidad del código. Para configurar:

```
pip install pre-commit
pre-commit install
```

Los hooks incluyen:

- Black (formateo de código)
- Flake8 (linting)
- isort (ordenamiento de imports)
- mypy (verificación de tipos)
- pytest (pruebas unitarias)
- Verificación de dependencias

Requerimientos de Calidad

1. **Estilo de Código**
    - Seguir PEP 8
    - Usar Black para formateo automático
    - Documentar con docstrings estilo NumPy
    - Máxima longitud de línea: 88 caracteres
2. **Testing**
    - Cada feature debe incluir tests unitarios
    - Cobertura mínima requerida: 80%
    - Los tests se ejecutan automáticamente en cada commit
3. **Documentación**
    - Docstrings obligatorios para funciones y clases
    - README actualizado para nuevas funcionalidades
    - Comentarios claros en código complejo
4. **Control de Calidad**
    - Los pre-commit hooks deben pasar antes de cada commit
    - Las Pull Requests deben pasar todas las validaciones

- El pipeline de validación debe ejecutarse exitosamente

Pipeline de Validación

El pipeline de validación (`src/utils/validate_pipeline.py`) verifica:

1. Estructura correcta del proyecto
2. Disponibilidad de datos necesarios
3. Dependencias requeridas
4. Calidad del código
5. Cobertura de tests

Testing

El proyecto sigue una estructura de testing organizada:

1. **Tests Unitarios** (`src/tests/unit/`)
   - Pruebas de componentes individuales
   - Validación de funciones específicas
   - Cobertura de casos edge
2. **Tests de Integración** (`src/tests/integration/`)
   - Pruebas de flujos completos
   - Validación de interacción entre componentes
   - Escenarios de uso real
3. **Ejecución de Tests**

```
# Ejecutar todos los tests
pytest

# Ejecutar tests con reporte de cobertura
pytest --cov=src

# Ejecutar tests específicos
pytest src/tests/unit/  # Solo tests unitarios
pytest src/tests/integration/  # Solo tests de integración
```

4. **Requerimientos de Testing**
   - Cobertura mínima: 80%
   - Tests obligatorios para cada feature nueva
   - Documentación de casos de prueba
   - Fixtures compartidos en `conftest.py`

Contacto

Para dudas o consultas, contactar al instructor:

- GitHub: @aladelca