Complete Implementation Guide: Comparative Analysis of Imputation Techniques

Project: Missing Data Imputation for Machine Learning
Authors: Lucas Aparicio, Guilherme Rodrigues, Rúben Oliveira
Course: Introduction to Machine Learning 2025/26

1. Introduction and Project Overview

1.1 Project Goals

This guide provides a comprehensive, step-by-step implementation plan for comparing four imputation techniques on real-world datasets with missing values. You will compare simple Mean Imputation, K-Nearest Neighbors (KNN), and two advanced methods—Multiple Imputation by Chained Equations (MICE) and MissForest—to determine which techniques provide the best balance between accuracy and computational efficiency.

1.2 Deliverables Summary

- **Intermediate Checkpoint**: Progress report with preliminary results (due Nov 20 - Dec 7)
- **Code Submission**: Complete Python scripts in ZIP format (due Dec 31)
- **Final Report**: 6-page double-column PDF with CRediT author statement (due Dec 31)
- **Oral Presentation**: 15-minute presentation in January

1.3 Required Software and Libraries

Install the following Python packages:

Core libraries

pip install pandas numpy scipy

Machine learning

pip install scikit-learn

Imputation methods

pip install miceforest
pip install missingpy

Visualization

pip install matplotlib seaborn

Optional: for notebook development

pip install jupyter

---

2. Phase 1: Environment Setup and Data Collection

2.1 Project Structure

Create the following directory structure:

```
imputation_project/
│
├── data/
│   ├── raw/ # Original downloaded datasets
│   ├── processed/ # Cleaned and preprocessed data
│   └── results/ # Imputed datasets
│
├── notebooks/ # Jupyter notebooks for exploration
├── src/ # Python source code
│   ├── preprocessing.py
│   ├── imputation.py
│   ├── evaluation.py
│   └── visualization.py
│
├── results/ # Figures and tables
├── reports/ # Intermediate and final reports
└── requirements.txt # Dependencies
```

2.2 Dataset Acquisition

Download the three datasets specified in your proposal:

1. **Heart Disease Dataset**: Download from Kaggle at `https://www.kaggle.com/datasets/redwankarimsony/heart-disease-data`
2. **Air Quality Dataset**: Download from UCI repository at `https://archive.ics.uci.edu/dataset/360/air+quality`
3. **Handling Missing Data Example**: Download from Kaggle at `https://www.kaggle.com/datasets/prince1204/handling-missing-data-example-dataset`

Place all raw data files in `data/raw/`.

2.3 Initial Data Exploration

Create a notebook `notebooks/01_data_exploration.ipynb`:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load datasets

```
heart_data = pd.read_csv('../data/raw/heart_disease.csv')
air_quality = pd.read_csv('../data/raw/air_quality.csv')
missing_example = pd.read_csv('../data/raw/missing_data_example.csv')
```

Function to analyze missing data

```
def analyze_missing(df, dataset_name):
print(f"\n=== {dataset_name} ===")
print(f"Shape: {df.shape}")
print(f"\nMissing Values:")
missing = df.isnull().sum()
missing_pct = (missing / len(df)) * 100
```

```
missing_df = pd.DataFrame({
    'Missing_Count': missing,
    'Percentage': missing_pct
})
print(missing_df[missing_df['Missing_Count'] > 0])

# Visualize missing data pattern
plt.figure(figsize=(10, 6))
sns.heatmap(df.isnull(), cbar=False, yticklabels=False)
plt.title(f'Missing Data Pattern - {dataset_name}')
plt.tight_layout()
plt.savefig(f'../results/missing_pattern_{dataset_name}.png')

return missing_df
```

Analyze each dataset

analyze_missing(heart_data, 'Heart_Disease')
analyze_missing(air_quality, 'Air_Quality')
analyze_missing(missing_example, 'Missing_Example')

---

3. Phase 2: Data Preprocessing

3.1 Data Cleaning Script

Create src/preprocessing.py:

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

class DataPreprocessor:
def **init**(self, df, target_column=None):
self.df = df.copy()
self.target_column = target_column
self.scaler = StandardScaler()
self.label_encoders = {}

```
def handle_dtypes(self):
    """Convert columns to appropriate data types"""
    for col in self.df.columns:
        # Try to convert to numeric
        if self.df[col].dtype == 'object':
            try:
                self.df[col] = pd.to_numeric(self.df[col], errors='coerce')
            except:
                pass
    return self

def encode_categorical(self):
    """Encode categorical variables"""
    categorical_cols = self.df.select_dtypes(include=['object']).columns

    for col in categorical_cols:
        if col != self.target_column:
            le = LabelEncoder()
            # Handle missing values
            mask = self.df[col].notna()
            self.df.loc[mask, col] = le.fit_transform(
                self.df.loc[mask, col]
            )
            self.label_encoders[col] = le
    return self

def remove_excess_missing(self, threshold=0.5):
    """Remove columns with more than threshold proportion missing"""
    missing_ratio = self.df.isnull().sum() / len(self.df)
    cols_to_keep = missing_ratio[missing_ratio < threshold].index
    self.df = self.df[cols_to_keep]
    print(f"Removed {len(missing_ratio) - len(cols_to_keep)} columns")
    return self

def normalize_features(self):
    """Normalize numerical features"""
    if self.target_column:
        feature_cols = [c for c in self.df.columns
                       if c != self.target_column]
    else:
        feature_cols = self.df.columns

    numeric_cols = self.df[feature_cols].select_dtypes(
        include=[np.number]
    ).columns

    # Only normalize columns without missing values
    complete_numeric = [c for c in numeric_cols
                       if self.df[c].notna().all()]

    if complete_numeric:
        self.df[complete_numeric] = self.scaler.fit_transform(
            self.df[complete_numeric]
        )
    return self
```

```
def get_processed_data(self):
    return self.df
```

Example usage

def preprocess_dataset(input_path, output_path, target_col=None):
df = pd.read_csv(input_path)

```
preprocessor = DataPreprocessor(df, target_column=target_col)
processed_df = (preprocessor
               .handle_dtypes()
               .encode_categorical()
               .remove_excess_missing(threshold=0.5)
               .get_processed_data())

processed_df.to_csv(output_path, index=False)
print(f"Processed data saved to {output_path}")
return processed_df
```

3.2 Creating Controlled Missing Data

For proper evaluation, create additional test scenarios with artificially introduced missing values:

def introduce_missing_values(df, missing_rate=0.2, seed=42):
"""
Introduce missing values randomly (MCAR mechanism)

```
Parameters:
- df: DataFrame with complete data
- missing_rate: Proportion of values to set as missing
- seed: Random seed for reproducibility
"""
np.random.seed(seed)
df_missing = df.copy()

# Only introduce missing in numeric columns
numeric_cols = df.select_dtypes(include=[np.number]).columns

for col in numeric_cols:
    n_missing = int(len(df) * missing_rate)
    missing_indices = np.random.choice(
        df.index,
        size=n_missing,
        replace=False
    )
    df_missing.loc[missing_indices, col] = np.nan

return df_missing
```

Save ground truth for evaluation

complete_data = df[df.notna().all(axis=1)].copy()
df_with_missing = introduce_missing_values(complete_data, missing_rate=0.2)

---

4. Phase 3: Implementing Imputation Methods

4.1 Imputation Implementation

Create `src/imputation.py`:

import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer, KNNImputer
from miceforest import ImputationKernel
from missingpy import MissForest

class ImputationComparator:
def **init**(self, df_with_missing, df_complete=None):
self.df_missing = df_with_missing.copy()
self.df_complete = df_complete.copy() if df_complete is not None else None
self.results = {}

```
def mean_imputation(self):
    """Simple mean imputation"""
    print("Performing Mean Imputation...")
    imputer = SimpleImputer(strategy='mean')

    df_imputed = self.df_missing.copy()
    numeric_cols = df_imputed.select_dtypes(include=[np.number]).columns

    df_imputed[numeric_cols] = imputer.fit_transform(
        df_imputed[numeric_cols]
    )

    self.results['mean'] = df_imputed
    return df_imputed

def knn_imputation(self, n_neighbors=5):
    """KNN imputation"""
    print(f"Performing KNN Imputation (k={n_neighbors})...")
    imputer = KNNImputer(n_neighbors=n_neighbors)

    df_imputed = self.df_missing.copy()
    numeric_cols = df_imputed.select_dtypes(include=[np.number]).columns

    df_imputed[numeric_cols] = imputer.fit_transform(
        df_imputed[numeric_cols]
    )

    self.results['knn'] = df_imputed
    return df_imputed

def mice_imputation(self, n_iterations=5):
    """MICE (Multiple Imputation by Chained Equations)"""
    print(f"Performing MICE Imputation ({n_iterations} iterations)...")

    df_imputed = self.df_missing.copy()

    # Create MICE kernel
    kernel = ImputationKernel(
        df_imputed,
        save_all_iterations=True,
        random_state=42
    )

    # Run imputation
    kernel.mice(n_iterations)

    # Get completed dataset (use the last iteration)
```

```
        df_imputed = kernel.complete_data(dataset=0)

        self.results['mice'] = df_imputed
        return df_imputed

    def missforest_imputation(self, max_iter=10):
        """MissForest imputation using Random Forests"""
        print(f"Performing MissForest Imputation (max_iter={max_iter})...")

        imputer = MissForest(max_iter=max_iter, random_state=42)

        df_imputed = self.df_missing.copy()
        numeric_cols = df_imputed.select_dtypes(include=[np.number]).columns

        # MissForest expects numpy array
        imputed_array = imputer.fit_transform(
            df_imputed[numeric_cols].values
        )

        df_imputed[numeric_cols] = imputed_array

        self.results['missforest'] = df_imputed
        return df_imputed

    def run_all_methods(self):
        """Run all imputation methods"""
        self.mean_imputation()
        self.knn_imputation(n_neighbors=5)
        self.mice_imputation(n_iterations=5)
        self.missforest_imputation(max_iter=10)

        return self.results
```

Example usage

```
comparator = ImputationComparator(df_with_missing, df_complete)
imputed_results = comparator.run_all_methods()
```

---

5. Phase 4: Evaluation Metrics

5.1 Evaluation Framework

Create `src/evaluation.py`:

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
```

```
class ImputationEvaluator:
def init(self, df_true, df_missing, imputation_results):
self.df_true = df_true
self.df_missing = df_missing
self.imputation_results = imputation_results
self.metrics = {}
```

```
def calculate_imputation_metrics(self):
    """Calculate direct imputation quality metrics"""
    results = []

    # Get mask of missing values
    missing_mask = self.df_missing.isna()

    for method, df_imputed in self.imputation_results.items():
        method_metrics = {'method': method}

        # Calculate metrics only for imputed values
        for col in self.df_true.select_dtypes(include=[np.number]).columns:
            if col in df_imputed.columns:
                # Get missing positions
                mask = missing_mask[col]

                if mask.sum() > 0:  # If there were missing values
                    true_vals = self.df_true.loc[mask, col]
                    imputed_vals = df_imputed.loc[mask, col]
```

```
                        # Calculate MAE
                        mae = mean_absolute_error(true_vals, imputed_vals)

                        # Calculate RMSE
                        rmse = np.sqrt(mean_squared_error(true_vals, imputed_vals))

                        # Calculate R²
                        r2 = r2_score(true_vals, imputed_vals)

                        method_metrics[f'{col}_MAE'] = mae
                        method_metrics[f'{col}_RMSE'] = rmse
                        method_metrics[f'{col}_R2'] = r2

            # Calculate overall metrics
            mae_cols = [k for k in method_metrics.keys() if 'MAE' in k]
            rmse_cols = [k for k in method_metrics.keys() if 'RMSE' in k]
            r2_cols = [k for k in method_metrics.keys() if 'R2' in k]

            method_metrics['Overall_MAE'] = np.mean([method_metrics[k] for k in ma
            method_metrics['Overall_RMSE'] = np.mean([method_metrics[k] for k in rm
            method_metrics['Overall_R2'] = np.mean([method_metrics[k] for k in r2_co

            results.append(method_metrics)

        self.metrics['imputation'] = pd.DataFrame(results)
        return self.metrics['imputation']

    def evaluate_downstream_task(self, target_column, task='classification'):
        """Evaluate imputation quality via downstream ML task"""
        results = []

        for method, df_imputed in self.imputation_results.items():
            # Prepare features and target
            X = df_imputed.drop(columns=[target_column])
            y = df_imputed[target_column]

            if task == 'classification':
```

```
            model = RandomForestClassifier(n_estimators=100, random_state=42)
            scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

            results.append({
                'method': method,
                'accuracy_mean': scores.mean(),
                'accuracy_std': scores.std()
            })

        self.metrics['downstream'] = pd.DataFrame(results)
        return self.metrics['downstream']

    def compare_distributions(self):
        """Compare statistical distributions before and after imputation"""
        results = []

        numeric_cols = self.df_true.select_dtypes(include=[np.number]).columns

        for method, df_imputed in self.imputation_results.items():
            for col in numeric_cols:
                if col in df_imputed.columns:
                    # Calculate distribution metrics
                    true_mean = self.df_true[col].mean()
                    true_std = self.df_true[col].std()

                    imputed_mean = df_imputed[col].mean()
                    imputed_std = df_imputed[col].std()

                    results.append({
                        'method': method,
                        'column': col,
                        'mean_diff': abs(true_mean - imputed_mean),
                        'std_diff': abs(true_std - imputed_std)
                    })

        self.metrics['distribution'] = pd.DataFrame(results)
        return self.metrics['distribution']
```

```
def generate_report(self):
    """Generate comprehensive evaluation report"""
    print("\n" + "="*60)
    print("IMPUTATION EVALUATION REPORT")
    print("="*60)

    print("\n1. Direct Imputation Quality Metrics:")
    print(self.metrics['imputation'][
        ['method', 'Overall_MAE', 'Overall_RMSE', 'Overall_R2']
    ].to_string(index=False))

    if 'downstream' in self.metrics:
        print("\n2. Downstream Task Performance:")
        print(self.metrics['downstream'].to_string(index=False))

    print("\n3. Distribution Preservation:")
    dist_summary = self.metrics['distribution'].groupby('method').agg({
        'mean_diff': 'mean',
        'std_diff': 'mean'
    })
    print(dist_summary)
```

Example usage

evaluator = ImputationEvaluator(df_complete, df_with_missing, imputed_results)
evaluator.calculate_imputation_metrics()
evaluator.compare_distributions()
evaluator.generate_report()

---

6. Phase 5: Visualization and Analysis

6.1 Visualization Script

Create `src/visualization.py`:

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

class ImputationVisualizer:
def **init**(self, metrics_dict):

self.metrics = metrics_dict
sns.set_style("whitegrid")

```
def plot_mae_comparison(self, save_path='results/mae_comparison.png'):
    """Compare MAE across methods"""
    df = self.metrics['imputation']

    fig, ax = plt.subplots(figsize=(10, 6))

    methods = df['method']
    mae_values = df['Overall_MAE']

    bars = ax.bar(methods, mae_values, color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62
    ax.set_ylabel('Mean Absolute Error (MAE)', fontsize=12)
    ax.set_xlabel('Imputation Method', fontsize=12)
    ax.set_title('Imputation Quality: MAE Comparison', fontsize=14, fontweight='

    # Add value labels on bars
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.4f}',
                ha='center', va='bottom', fontsize=10)

    plt.tight_layout()
    plt.savefig(save_path, dpi=300)
    plt.close()
    print(f"Saved MAE comparison to {save_path}")

def plot_multiple_metrics(self, save_path='results/metrics_comparison.png'):
    """Compare multiple metrics across methods"""
    df = self.metrics['imputation']

    metrics_to_plot = ['Overall_MAE', 'Overall_RMSE', 'Overall_R2']

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    for idx, metric in enumerate(metrics_to_plot):
        ax = axes[idx]
        methods = df['method']
        values = df[metric]

        bars = ax.bar(methods, values, color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728
        ax.set_ylabel(metric.replace('_', ' '), fontsize=11)
        ax.set_xlabel('Method', fontsize=11)
        ax.set_title(metric.replace('_', ' '), fontsize=12, fontweight='bold')
        ax.tick_params(axis='x', rotation=45)

        # Add value labels
        for bar in bars:
            height = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2., height,
                    f'{height:.3f}',
                    ha='center', va='bottom', fontsize=9)

    plt.tight_layout()
    plt.savefig(save_path, dpi=300)
    plt.close()
    print(f"Saved metrics comparison to {save_path}")

def plot_distribution_comparison(self, df_true, imputed_results,
                                 column, save_path='results/distribution.png'):
    """Compare distributions of a specific column"""
    fig, axes = plt.subplots(2, 3, figsize=(15, 10))
    axes = axes.flatten()

    # Plot original distribution
    axes[0].hist(df_true[column].dropna(), bins=30, alpha=0.7, color='black')
    axes[0].set_title('Original (Complete Data)', fontweight='bold')
    axes[0].set_ylabel('Frequency')

    # Plot each imputation method
    methods = ['mean', 'knn', 'mice', 'missforest']
    colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728']

    for idx, (method, color) in enumerate(zip(methods, colors), start=1):
        if method in imputed_results:
            axes[idx].hist(imputed_results[method][column],
                         bins=30, alpha=0.7, color=color)
            axes[idx].set_title(f'{method.upper()} Imputation', fontweight='bold')
            axes[idx].set_ylabel('Frequency')

    # Hide extra subplot
    axes[5].axis('off')

    fig.suptitle(f'Distribution Comparison: {column}',
                fontsize=16, fontweight='bold', y=0.995)
    plt.tight_layout()
    plt.savefig(save_path, dpi=300)
    plt.close()
    print(f"Saved distribution comparison to {save_path}")

def plot_computation_time(self, time_dict, save_path='results/computation_time
    """Compare computation times"""
    fig, ax = plt.subplots(figsize=(10, 6))

    methods = list(time_dict.keys())
    times = list(time_dict.values())

    bars = ax.barh(methods, times, color=['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728'])
    ax.set_xlabel('Computation Time (seconds)', fontsize=12)
    ax.set_ylabel('Method', fontsize=12)
    ax.set_title('Computational Efficiency Comparison', fontsize=14, fontweight='

    for bar in bars:
        width = bar.get_width()
        ax.text(width, bar.get_y() + bar.get_height()/2.,
               f'{width:.2f}s',
               ha='left', va='center', fontsize=10, fontweight='bold')

    plt.tight_layout()
    plt.savefig(save_path, dpi=300)
    plt.close()
    print(f"Saved computation time plot to {save_path}")
```

Example usage

visualizer = ImputationVisualizer(evaluator.metrics)
visualizer.plot_mae_comparison()
visualizer.plot_multiple_metrics()

---

7. Phase 6: Complete Pipeline Integration

7.1 Main Execution Script

Create `main.py`:

import pandas as pd
import numpy as np
import time
from src.preprocessing import DataPreprocessor, preprocess_dataset
from src.imputation import ImputationComparator
from src.evaluation import ImputationEvaluator
from src.visualization import ImputationVisualizer

def run_complete_pipeline(dataset_path, dataset_name, target_column=None):
"""
Run complete imputation comparison pipeline

```
Parameters:
- dataset_path: Path to the dataset
- dataset_name: Name for saving results
- target_column: Target variable for downstream evaluation
"""
print(f"\n{'='*70}")
print(f"PROCESSING DATASET: {dataset_name}")
print(f"{'='*70}\n")

# Step 1: Load and preprocess data
print("Step 1: Loading and preprocessing data...")
df = pd.read_csv(dataset_path)

preprocessor = DataPreprocessor(df, target_column=target_column)
df_processed = (preprocessor
            .handle_dtypes()
```

```
                .encode_categorical()
                .remove_excess_missing(threshold=0.5)
                .get_processed_data())

# Step 2: Create evaluation scenario
print("\nStep 2: Creating evaluation scenario...")
# Use only complete cases for ground truth
df_complete = df_processed[df_processed.notna().all(axis=1)].copy()
print(f"Complete cases: {len(df_complete)}")

# Introduce controlled missing values
from src.preprocessing import introduce_missing_values
df_with_missing = introduce_missing_values(df_complete, missing_rate=0.2)

# Step 3: Run imputation methods
print("\nStep 3: Running imputation methods...")
comparator = ImputationComparator(df_with_missing, df_complete)

computation_times = {}

# Time each method
start = time.time()
comparator.mean_imputation()
computation_times['mean'] = time.time() - start

start = time.time()
comparator.knn_imputation(n_neighbors=5)
computation_times['knn'] = time.time() - start

start = time.time()
comparator.mice_imputation(n_iterations=5)
computation_times['mice'] = time.time() - start

start = time.time()
comparator.missforest_imputation(max_iter=10)
computation_times['missforest'] = time.time() - start

imputed_results = comparator.results
```

```
# Step 4: Evaluate results
print("\nStep 4: Evaluating imputation quality...")
evaluator = ImputationEvaluator(df_complete, df_with_missing, imputed_resul
evaluator.calculate_imputation_metrics()
evaluator.compare_distributions()

if target_column:
    evaluator.evaluate_downstream_task(target_column, task='classification')

evaluator.generate_report()

# Step 5: Generate visualizations
print("\nStep 5: Generating visualizations...")
visualizer = ImputationVisualizer(evaluator.metrics)
visualizer.plot_mae_comparison(f'results/{dataset_name}_mae.png')
visualizer.plot_multiple_metrics(f'results/{dataset_name}_metrics.png')
visualizer.plot_computation_time(computation_times,
                                f'results/{dataset_name}_time.png')

# Step 6: Save results
print("\nStep 6: Saving results...")
evaluator.metrics['imputation'].to_csv(
    f'results/{dataset_name}_metrics.csv', index=False
)

for method, df_imputed in imputed_results.items():
    df_imputed.to_csv(
        f'data/results/{dataset_name}_{method}_imputed.csv',
        index=False
    )

print(f"\n{'='*70}")
print(f"COMPLETED: {dataset_name}")
print(f"{'='*70}\n")

return evaluator.metrics, computation_times
```

Run for all three datasets

if **name** == "**main**":
datasets = [
('data/raw/heart_disease.csv', 'heart_disease', 'target'),
('data/raw/air_quality.csv', 'air_quality', None),
('data/raw/missing_example.csv', 'missing_example', None)
]

```
all_results = {}

for dataset_path, dataset_name, target_col in datasets:
    try:
        metrics, times = run_complete_pipeline(
            dataset_path,
            dataset_name,
            target_col
        )
        all_results[dataset_name] = {
            'metrics': metrics,
            'times': times
        }
    except Exception as e:
        print(f"Error processing {dataset_name}: {str(e)}")
        continue

# Generate summary report
print("\n" + "="*70)
print("OVERALL SUMMARY")
print("="*70)

for dataset_name, results in all_results.items():
    print(f"\n{dataset_name}:")
    print(results['metrics']['imputation'][
        ['method', 'Overall_MAE', 'Overall_RMSE']
    ].to_string(index=False))
```

---

8. Phase 7: Intermediate Checkpoint Preparation

8.1 Checkpoint Report Structure

For your intermediate checkpoint (due Nov 20 - Dec 7), prepare a document covering:

1. **Progress Summary**: What has been completed so far
2. **Preliminary Results**: Initial findings from at least one dataset
3. **Challenges Encountered**: Technical issues and solutions
4. **Remaining Work**: What still needs to be done
5. **Timeline**: Plan for completion by Dec 31

8.2 Checkpoint Deliverable Template

Intermediate Checkpoint: Imputation Techniques Comparison

Team Members

- Lucas Aparicio (up202206594)
- Guilherme Rodrigues (up202208878)
- Rúben Oliveira (up202205106)

Progress Summary (as of [Date])

Completed Tasks

- ✓ Environment setup and library installation
- ✓ Dataset acquisition and initial exploration
- ✓ Implementation of all four imputation methods
- ✓ Basic evaluation framework

Preliminary Results

[Include table with initial metrics from one dataset]

Challenges and Solutions

1. **Challenge**: MissForest compatibility issues
   Solution: [Describe your solution]
2. **Challenge**: Large dataset processing time
   Solution: [Describe your approach]

Remaining Work

- Complete evaluation on all three datasets
- Generate comprehensive visualizations
- Conduct downstream task evaluation
- Write final report
- Prepare presentation

Timeline to Completion

Task	Deadline
Complete all evaluations	Dec 15
Generate all visualizations	Dec 20
Write final report	Dec 28
Code cleanup and documentation	Dec 30
Final submission	Dec 31

---

9. Phase 8: Final Report Writing

9.1 Report Structure (6 pages, double column)

1. **Abstract** (150-200 words): Summary of objectives, methods, and key findings
2. **Introduction** (0.5 pages):
   - Problem motivation
   - Research questions
   - Contribution summary
3. **Related Work** (0.5 pages):
   - Summary of the three cited papers
   - Gap analysis
4. **Methodology** (2 pages):
   - Dataset descriptions
   - Imputation techniques explained
   - Evaluation metrics
   - Experimental setup
5. **Results** (2 pages):
   - Imputation quality metrics
   - Computational efficiency comparison
   - Distribution preservation analysis
   - Downstream task performance
6. **Discussion** (0.5 pages):
   - Interpretation of results
   - Trade-offs between methods
   - Practical recommendations
7. **Conclusion** (0.5 pages):

- Summary of findings
- Limitations
- Future work

9.2 LaTeX Template for Report

\documentclass[conference]{IEEEtran}\usepackage{cite}\usepackage{amsmath,amssymb,amsfonts}\usepackage{algorithmic}\usepackage{graphicx}\usepackage{textcomp}\usepackage{xcolor}

\begin{document}

Comparative Analysis of Various Imputation Techniques for Missing Data

\IEEEauthorblockN{Lucas Aparicio}
\IEEEauthorblockA{up202206594}

\IEEEauthorblockN{Guilherme Rodrigues}
\IEEEauthorblockA{up202208878}

\IEEEauthorblockN{Rúben Oliveira}
\IEEEauthorblockA{up202205106}

Abstract

Missing data is a pervasive challenge in real-world machine learning applications... [Your abstract here]

1. Introduction

Missing values in datasets pose significant challenges...

2. Related Work

2.1. Comparison Studies

Joel et al. conducted a comparative study...

3. Methodology

3.1. Datasets

We evaluated four imputation techniques on three real-world datasets...

3.2. Imputation Techniques

3.2.1. Mean Imputation

The simplest baseline method...

3.2.2. K-Nearest Neighbors

KNN imputation leverages...

3.2.3. MICE

Multiple Imputation by Chained Equations...

3.2.4. MissForest

An iterative imputation method using Random Forests...

3.3. Evaluation Metrics

We evaluate imputation quality using:

- Mean Absolute Error (MAE)
- Root Mean Squared Error (RMSE)
- R^2 Score
- Computation time

4. Results

Table 1: Imputation Quality Metrics

Method	MAE	RMSE	R^2
Mean	0.XXX	0.XXX	0.XXX
KNN	0.XXX	0.XXX	0.XXX
MICE	0.XXX	0.XXX	0.XXX
MissForest	0.XXX	0.XXX	0.XXX

5. Discussion

Our results demonstrate that...

6. Conclusion

This study compared four imputation techniques...
\end{document}

9.3 CRediT Author Statement

Add this section after the main content (not counted in 6-page limit):
CRediT Author Statement
Lucas Aparicio: Conceptualization, Methodology, Software (MICE and MissForest implementation), Writing - Original Draft - 40%
Guilherme Rodrigues: Software (Mean and KNN implementation), Validation, Formal Analysis, Visualization - 30%

Rúben Oliveira: Data Curation, Investigation, Writing - Review & Editing, Presentation - 30%

10. Phase 9: Presentation Preparation

10.1 Presentation Structure (15 minutes)

Section	Time	Content
Title & Introduction	1 min	Problem motivation
Background	1.5 min	Missing data types, importance
Methodology	3 min	Four techniques explained
Datasets	1 min	Three datasets overview
Results	5 min	Metrics, visualizations
Discussion	2 min	Key insights, trade-offs
Conclusion	1 min	Summary, future work
Q&A	0.5 min	Buffer time

10.2 Key Slides

1. **Title Slide**: Project title, team members
2. **Motivation**: Why missing data matters
3. **Research Questions**: What we investigated
4. **Imputation Techniques**: Visual comparison of four methods
5. **Experimental Setup**: Datasets, metrics, procedure
6. **Results - MAE Comparison**: Bar chart
7. **Results - Multiple Metrics**: Comprehensive view
8. **Results - Computation Time**: Efficiency comparison
9. **Results - Distribution Preservation**: Before/after histograms
10. **Key Findings**: Bullet points of main insights
11. **Practical Recommendations**: When to use each method
12. **Limitations & Future Work**
13. **Conclusion**: Take-home messages

11. Tips for Success

- **Version Control**: Use Git to track changes and collaborate
- **Documentation**: Comment your code thoroughly
- **Reproducibility**: Set random seeds, save configurations
- **Testing**: Test each component independently before integration
- **Time Management**: Start early, don't leave everything to the end
- **Communication**: Regular team meetings, clear task division
- **Backup**: Keep multiple copies of your work
- **Ask for Help**: Contact instructors if you encounter major blockers

12. Common Pitfalls to Avoid

1. **Not handling categorical variables properly**: Encode before imputation
2. **Normalizing before creating missing values**: Normalize after introducing missingness
3. **Evaluating on training data**: Use separate test set or cross-validation
4. **Ignoring computational constraints**: Set reasonable parameters for MICE and MissForest
5. **Comparing only MAE**: Use multiple metrics for comprehensive evaluation
6. **Poor visualization**: Make figures clear, labeled, and publication-quality
7. **Weak discussion**: Don't just present numbers, interpret and explain
8. **Missing citations**: Properly reference all three papers and methods

13. Expected Outcomes

By completing this project, you will:

- Understand strengths and weaknesses of different imputation approaches
- Gain practical experience implementing ML techniques
- Learn to evaluate and compare algorithms systematically
- Develop skills in scientific writing and presentation
- Create a portfolio-worthy project demonstrating ML competency

14. Grading Criteria Reference

Based on the course requirements:

Component	**Weight**
Project Proposal	5%
Intermediate Checkpoint	10%
Code + Final Report	60%
Oral Presentation	25%
Total	**100%**

Focus particularly on the code and report (60%) and presentation (25%) as these carry the most weight.

---

15. Resources and References

15.1 Required Papers

1. Joel, L.O., Doorsamy, W., Paul, B.S. "A comparative study of imputation techniques for missing values in Healthcare diagnostic datasets"
2. Aljuaid, T., Sasi, S. "Proper Imputation Techniques For Missing Values In Data Sets", IEEE
3. Austin, P.C., White, I.R., Lee, D.S., van Buuren, S. "Missing Data in Clinical Research: A Tutorial on Multiple Imputation"

15.2 Additional Resources

- Scikit-learn documentation: https://scikit-learn.org/
- MICE implementation: https://github.com/AnotherSamWilson/miceforest
- MissForest: https://github.com/epsilon-machine/missingpy
- Pandas missing data guide: https://pandas.pydata.org/docs/user_guide/missing_data.html

---

Conclusion

This comprehensive guide provides you with a complete roadmap from initial setup through final presentation. Follow each phase systematically, maintain good coding practices, and don't hesitate to reach out to your instructors with questions. Good luck with your project!