Design Document

Overview

This design document outlines the architecture and implementation strategy for the data preprocessing pipeline for the ML Challenge 2025: Smart Product Pricing Challenge. The pipeline is designed to handle 150,000 samples (75k train + 75k test) of multimodal data (text and images), ensuring zero data loss while maintaining high data quality standards.

Design Philosophy

The design is inspired by the past Amazon ML Challenge winner's approach, which emphasized:

- **Multimodal feature engineering**: Combining text embeddings and visual features
- **Neighbor-based features**: Using KNN/ANN to find similar products and leverage their prices
- **Robust preprocessing**: Handling multilingual text and ensuring data quality

Our pipeline **replicates and focuses** on the previous winners' roven approach with strategic enhancements:

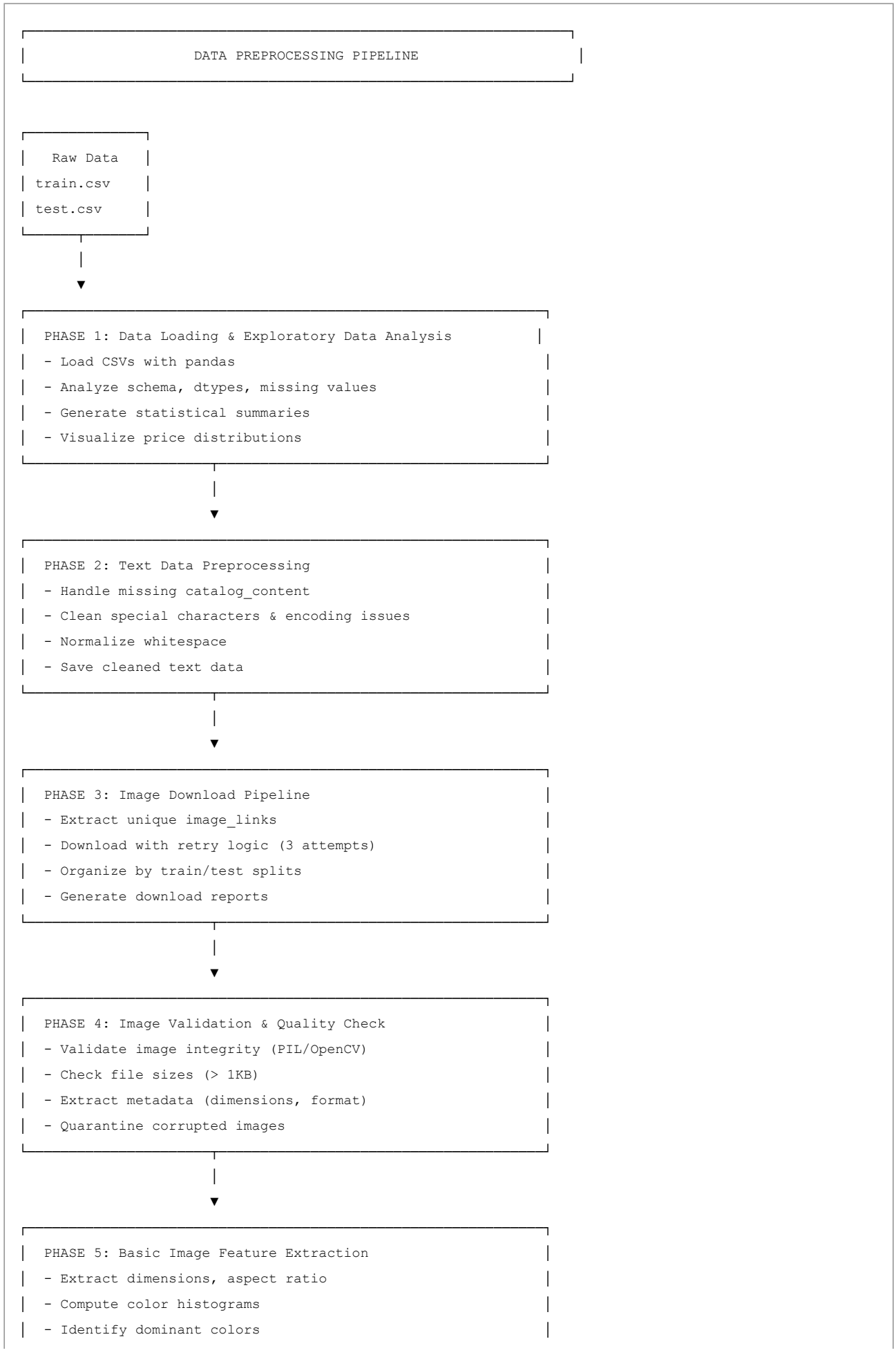
- **Multi-Model Embeddings**: 5 sentence transformers (exactly like winner) for hierarchical neighbor search
- **Critical Text Features**: Brand extraction and pack quantity (highest price predictors)
- **CNN Embeddings**: EfficientNet-B0 for powerful visual features
- **Streamlined Approach**: Focus on proven, high-impact features only

Key Design Decisions

1. **Single Notebook Architecture**: All preprocessing consolidated in one Jupyter notebook for reproducibility and transparency
2. **Modular Functions**: Reusable utility functions for each preprocessing step
3. **Fail-Safe Mechanisms**: Comprehensive error handling and logging to prevent data loss
4. **Parallel Processing**: Leveraging multiprocessing for image operations
5. **Incremental Checkpointing**: Saving intermediate outputs to enable pipeline resumption
6. **Proven Feature Engineering**: Extract only high-impact features (brand, pack quantity, embeddings, CNN)
7. **Lightweight Deep Learning**: Use efficient pre-trained models (EfficientNet-B0, 5M params) for CNN embeddings

Architecture

High-Level Pipeline Flow


DATA PREPROCESSING PIPELINE
Raw Data
train.csv
test.csv
PHASE 1: Data Loading & Exploratory Data Analysis
- Load CSVs with pandas
- Analyze schema, dtypes, missing values
- Generate statistical summaries
- Visualize price distributions
PHASE 2: Text Data Preprocessing
- Handle missing catalog_content
- Clean special characters & encoding issues
- Normalize whitespace
- Save cleaned text data
PHASE 3: Image Download Pipeline
- Extract unique image_links
- Download with retry logic (3 attempts)
- Organize by train/test splits
- Generate download reports
PHASE 4: Image Validation & Quality Check
- Validate image integrity (PIL/OpenCV)
- Check file sizes (> 1KB)
- Extract metadata (dimensions, format)
- Quarantine corrupted images
PHASE 5: Basic Image Feature Extraction
- Extract dimensions, aspect ratio
- Compute color histograms
- Identify dominant colors

Components and Interfaces

Component 1: Data Loader Module

Purpose: Load and validate raw CSV data

Key Functions:

```
def load_dataset(filepath, dataset_type='train'):
    """
    Load CSV with automatic type inference and validation

    Args:
        filepath: Path to CSV file
        dataset_type: 'train' or 'test'

    Returns:
        pd.DataFrame: Loaded dataset
        dict: Metadata (shape, dtypes, missing counts)
    """
    pass

def validate_schema(df, expected_columns):
    """
    Validate dataset schema matches expectations

    Args:
        df: DataFrame to validate
        expected_columns: List of expected column names

    Returns:
        bool: True if valid
        list: List of schema issues if any
    """
    pass
```

Interface:

- Input: CSV file paths
- Output: Pandas DataFrames with metadata dictionaries

Component 2: Text Preprocessing and Critical Feature Extraction Module (Proven)

Purpose: Clean text and extract only the most impactful price-predictive features

Key Functions:

```
def clean_text(text):
    """
    Clean individual text entry

    Args:
        text: Raw catalog_content string

    Returns:
        str: Cleaned text
    """
    pass

def handle_missing_text(df, text_column='catalog_content'):
    """
    Handle missing/null text values (fillna(' '))

    Args:
        df: DataFrame with text column
        text_column: Name of text column

    Returns:
        pd.DataFrame: DataFrame with handled missing values
    """
    pass

def extract_brand_name(text):
    """
    Extract brand name using simple regex patterns (CRITICAL FEATURE - highest price predictor)

    Args:
        text: Catalog content text

    Returns:
        str: Extracted brand name or 'unknown'
    """
    pass

def extract_pack_quantity(text):
    """
    Extract pack size/quantity (CRITICAL FEATURE - major price factor)

    Args:
        text: Catalog content text

    Returns:
        int: Pack quantity or 1 (default)
    """
    pass

def preprocess_text_column(df, text_column='catalog_content'):
    """
    Apply streamlined text preprocessing pipeline
```

```
    Args:
        df: DataFrame with text column
        text_column: Name of text column

    Returns:
        pd.DataFrame: DataFrame with cleaned_text, brand, and pack_quantity
    """
    pass
```

Interface:

- Input: DataFrame with catalog_content column
- Output: DataFrame with cleaned_catalog_content + brand + pack_quantity

Component 3: Image Download Manager

Purpose: Robustly download product images with retry logic

Key Functions:

```
def download_image_with_retry(image_link, save_path, sample_id, max_retries=3):
    """
    Download single image with exponential backoff retry

    Args:
        image_link: URL to image
        save_path: Local path to save image
        sample_id: Unique identifier for logging
        max_retries: Maximum retry attempts

    Returns:
        dict: {
            'sample_id': str,
            'status': 'success'|'failed',
            'error': str or None,
            'file_size': int or None
        }
    """
    pass

def batch_download_images(df, image_column='image_link',
                          save_folder='dataset/images/train',
                          num_workers=50):
    """
    Download images in parallel with progress tracking

    Args:
        df: DataFrame with image_link and sample_id columns
        image_column: Name of image link column
        save_folder: Directory to save images
        num_workers: Number of parallel workers

    Returns:
        pd.DataFrame: Download report with status per sample_id
    """
    pass

def resume_failed_downloads(download_report, df, save_folder):
    """
    Retry failed downloads from previous attempt

    Args:
        download_report: DataFrame from previous download attempt
        df: Original DataFrame with image links
        save_folder: Directory to save images

    Returns:
        pd.DataFrame: Updated download report
    """
    pass
```

Interface:

- Input: DataFrame with image_link and sample_id columns
- Output: Downloaded images in organized folders + download report CSV

Design Rationale:

- Extends existing `download_images` from utils.py with retry logic
- Uses exponential backoff (1s, 2s, 4s delays) to handle throttling
- Saves images as `{sample_id}.jpg` for easy mapping
- Generates detailed reports for tracking failures

Component 4: Image Validator

Purpose: Validate image integrity and extract metadata

Key Functions:

```
def validate_single_image(image_path, sample_id):
    """
    Validate single image file

    Args:
        image_path: Path to image file
        sample_id: Unique identifier

    Returns:
        dict: {
            'sample_id': str,
            'is_valid': bool,
            'file_size': int,
            'width': int or None,
            'height': int or None,
            'format': str or None,
            'error': str or None
        }
    """
    pass

def batch_validate_images(image_folder, sample_ids, num_workers=50):
    """
    Validate all images in parallel

    Args:
        image_folder: Directory containing images
        sample_ids: List of expected sample_ids
        num_workers: Number of parallel workers

    Returns:
        pd.DataFrame: Validation report
    """
    pass

def quarantine_corrupted_images(validation_report, image_folder):
    """
    Move corrupted images to quarantine folder

    Args:
        validation_report: DataFrame from validation
        image_folder: Source folder

    Returns:
        int: Number of images quarantined
    """
    pass
```

Interface:

- Input: Image folder path and list of sample_ids
- Output: Validation report DataFrame + quarantined corrupted files

Design Rationale:

- Uses PIL (Pillow) for image validation (try to open and verify)
- Checks minimum file size (1KB) to filter error pages
- Extracts metadata for quality assessment
- Non-destructive quarantine (moves, doesn't delete)

Component 5: Image Feature Extractor (Streamlined)

Purpose: Extract essential visual features - basic statistics and CNN embeddings

Key Functions:

```
def extract_basic_image_features(image_path, sample_id):
    """
    Extract basic features from single image

    Args:
        image_path: Path to image file
        sample_id: Unique identifier

    Returns:
        dict: {
            'sample_id': str,
            'width': int,
            'height': int,
            'aspect_ratio': float,
            'dominant_color_r': int,
            'dominant_color_g': int,
            'dominant_color_b': int,
            'color_hist_r': list[10],  # 10-bin histogram
            'color_hist_g': list[10],
            'color_hist_b': list[10],
            'brightness_mean': float,
            'brightness_std': float
        }
    """
    pass

def extract_cnn_embeddings(image_path, sample_id, model):
    """
    Extract deep learning embeddings using pre-trained CNN (PROVEN POWERFUL FEATURE)

    Args:
        image_path: Path to image file
        sample_id: Unique identifier
        model: Pre-trained EfficientNet-B0

    Returns:
        dict: {
            'sample_id': str,
            'cnn_embedding': np.array (shape: 1280 for EfficientNet-B0)
        }
    """
    pass

def batch_extract_all_features(image_folder, sample_ids, num_workers=50):
    """
    Extract basic and CNN features from all images in parallel

    Args:
        image_folder: Directory containing images
        sample_ids: List of sample_ids to process
        num_workers: Number of parallel workers
```

```
    Returns:
        tuple: (basic_features_df, cnn_embeddings_array)
    """
    pass
```

Interface:

- Input: Image folder path and list of sample_ids
- Output: Basic features DataFrame + CNN embeddings array

Design Rationale:

- **Basic features**: Quick to compute, capture color/size information (~30 features)
- **CNN embeddings**: Capture high-level visual semantics (product type, quality, presentation) - 1280 powerful features
- Uses lightweight EfficientNet-B0 (5M params) to stay within constraints
- Parallel processing to handle 75k images efficiently
- Focus on proven, high-impact features only

Component 6: Multi-Model Text Embeddings Generator

Purpose: Generate embeddings using 5 sentence transformers for hierarchical neighbor search

Key Functions:

```
def load_sentence_transformers():
    """
    Load 5 sentence transformer models

    Returns:
        dict: {
            'p_mlm_l6_v2': SentenceTransformer('paraphrase-MiniLM-L6-v2'),
            'p_ml_mlm_l12_v2': SentenceTransformer('paraphrase-multilingual-MiniLM-L12-v2'),
            'a_mlm_l6_v2': SentenceTransformer('all-MiniLM-L6-v2'),
            'a_mlm_l12_v2': SentenceTransformer('all-MiniLM-L12-v2'),
            'distil_ml_base_v2': SentenceTransformer('distiluse-base-multilingual-cased-v2')
        }
    """
    pass

def generate_multi_model_embeddings(texts, models_dict, batch_size=32):
    """
    Generate embeddings using 5 models (PROVEN WINNER STRATEGY)

    Args:
        texts: List of text strings
        models_dict: Dictionary of loaded models
        batch_size: Batch size for encoding

    Returns:
        dict: {
            'p_mlm_l6_v2_embeddings': np.array (384 dims),
            'p_ml_mlm_l12_v2_embeddings': np.array (384 dims),
            'a_mlm_l6_v2_embeddings': np.array (384 dims),
            'a_mlm_l12_v2_embeddings': np.array (384 dims),
            'distil_ml_base_v2_embeddings': np.array (512 dims)
        }
    """
    pass

def save_embeddings(embeddings_dict, sample_ids, output_folder):
    """
    Save embeddings in efficient format

    Args:
        embeddings_dict: Dictionary of embedding arrays
        sample_ids: List of sample_ids
        output_folder: Directory to save embeddings

    Returns:
        None (saves .npy files)
    """
    pass
```

Interface:

- Input: Cleaned text data

- Output: 5 embedding matrices saved as .npy files

Design Rationale:

- **Exactly replicates previous winners' approach** - proven to work
- 5 models enable hierarchical neighbor search (KNN on 4 models, ANN on 1 model)
- Different models capture different similarities for robust neighbor-based features
- Provides foundation for Model Builder to extract 140 neighbor-based features (like winner)
- Saves in efficient numpy format for fast loading during modeling

Component 7: Data Quality Checker (Streamlined)

Purpose: Essential data quality validation

Key Functions:

```
def check_sample_id_uniqueness(df):
    """Check for duplicate sample_ids"""
    pass

def check_price_validity(df):
    """Validate prices are positive floats"""
    pass

def check_data_completeness(df, expected_count=75000):
    """Verify sample count matches expectation"""
    pass

def check_image_text_alignment(csv_df, image_folder):
    """Verify each sample_id has both CSV entry and image"""
    pass

def generate_quality_report(train_df, test_df, train_images, test_images):
    """
    Generate comprehensive quality report

    Returns:
        dict: Quality metrics and data readiness score
    """
    pass
```

Interface:

- Input: Preprocessed DataFrames and image folders
- Output: Quality report dictionary and data readiness score (0-100%)

Component 8: Logging and Competitive Advantage Reporting System

Purpose: Track pipeline execution and document novel approaches

Key Functions:

```
def setup_logging(log_folder='logs'):
    """Initialize logging configuration"""
    pass

def log_phase_completion(phase_name, metrics):
    """Log completion of pipeline phase with metrics"""
    pass

def generate_summary_report(all_metrics):
    """
    Generate final summary report

    Returns:
        str: Markdown-formatted summary report
    """
    pass

def generate_competitive_advantage_report(novel_features_summary):
    """
    Document novel approaches and competitive advantages (NEW)

    Args:
        novel_features_summary: Dict of novel features extracted

    Returns:
        str: Markdown report documenting competitive advantages
    """
    pass
```

Interface:

- Input: Metrics from each pipeline phase
- Output: Log files, summary reports, and competitive advantage documentation

Data Models

Raw Data Schema

train.csv / test.csv:

```
sample_id: str (unique identifier)
catalog_content: str (concatenated title + description + IPQ)
image_link: str (URL to product image)
price: float (target variable, train only)
```

Preprocessed Data Schema

cleaned_train.csv / cleaned_test.csv (Streamlined):

```
sample_id: str
catalog_content: str (original)
cleaned_catalog_content: str (preprocessed)
image_link: str
image_filename: str ({sample_id}.jpg)
has_valid_image: bool
price: float (train only)

# CRITICAL TEXT FEATURES
brand_name: str
pack_quantity: int (default 1)
```

basic_image_features_train.csv / basic_image_features_test.csv:

```
sample_id: str (index)
width: int
height: int
aspect_ratio: float
dominant_color_r: int (0-255)
dominant_color_g: int (0-255)
dominant_color_b: int (0-255)
color_hist_r_bin_0 to color_hist_r_bin_9: float
color_hist_g_bin_0 to color_hist_g_bin_9: float
color_hist_b_bin_0 to color_hist_b_bin_9: float
brightness_mean: float
brightness_std: float
```

cnn_embeddings_train.npy / cnn_embeddings_test.npy:

```
numpy array: shape (75000, 1280) for EfficientNet-B0
# Rows correspond to sample_ids in order
```

text_embeddings/ (5 Models):

```
p_mlm_l6_v2_train.npy: shape (75000, 384)
p_mlm_l6_v2_test.npy: shape (75000, 384)
p_ml_mlm_l12_v2_train.npy: shape (75000, 384)
p_ml_mlm_l12_v2_test.npy: shape (75000, 384)
a_mlm_l6_v2_train.npy: shape (75000, 384)
a_mlm_l6_v2_test.npy: shape (75000, 384)
a_mlm_l12_v2_train.npy: shape (75000, 384)
a_mlm_l12_v2_test.npy: shape (75000, 384)
distil_ml_base_v2_train.npy: shape (75000, 512)
distil_ml_base_v2_test.npy: shape (75000, 512)
```

download_report.csv:

```
sample_id: str
image_link: str
status: str ('success', 'failed', 'skipped')
error_message: str or None
file_size: int or None
download_attempts: int
timestamp: datetime
```

validation_report.csv:

```
sample_id: str
image_path: str
is_valid: bool
file_size: int
width: int or None
height: int or None
format: str or None
error_message: str or None
```

Error Handling

Error Categories and Strategies

1. **Missing Data Errors**
    - **Scenario**: Null/NaN values in catalog_content or image_link
    - **Strategy**: Replace with empty string for text, log missing images
    - **Recovery**: Continue processing, flag in quality report
2. **Network Errors (Image Download)**
    - **Scenario**: Connection timeout, HTTP errors, throttling
    - **Strategy**: Exponential backoff retry (3 attempts)
    - **Recovery**: Log failures, enable resume functionality
3. **Corrupted Image Errors**
    - **Scenario**: Downloaded file is not a valid image
    - **Strategy**: Quarantine file, attempt re-download
    - **Recovery**: Mark sample as missing image, continue processing
4. **Memory Errors**
    - **Scenario**: Processing too many images simultaneously
    - **Strategy**: Batch processing with configurable batch size
    - **Recovery**: Reduce batch size, process in chunks
5. **File System Errors**
    - **Scenario**: Disk full, permission denied
    - **Strategy**: Check disk space before operations, proper error messages
    - **Recovery**: Alert user, provide cleanup suggestions

Error Logging Format

All errors logged to `logs/preprocessing_errors.log`:

```
[TIMESTAMP] [ERROR_LEVEL] [COMPONENT] [SAMPLE_ID] Error message
```

Example:

```
[2025-10-11 14:23:45] [ERROR] [ImageDownload] [SAMPLE_12345] Failed after 3 retries: Connection timeout
```

Testing Strategy

Unit Testing Approach

While the main deliverable is a Jupyter notebook, key functions should be tested interactively:

1. **Text Preprocessing Tests**
    - Test with various edge cases: empty strings, special characters, multilingual text
    - Verify no data loss (same number of rows before/after)
2. **Image Download Tests**
    - Test with small sample (10-20 images) before full run
    - Verify retry logic with intentionally bad URLs
    - Check folder structure creation
3. **Image Validation Tests**
    - Test with known good and corrupted images
    - Verify metadata extraction accuracy
4. **Feature Extraction Tests**
    - Test with diverse images (different sizes, colors)
    - Verify feature ranges are reasonable
    - Check for NaN/Inf values

Integration Testing

End-to-End Pipeline Test:

1. Run pipeline on sample_test.csv (small subset)
2. Verify all outputs are generated
3. Check data quality metrics
4. Validate output formats match expectations

Performance Testing:

1. Measure processing time for each phase
2. Monitor memory usage during image processing
3. Optimize bottlenecks (likely image operations)

Validation Checkpoints

After each phase, validate:

- ✓ No sample_ids lost
- ✓ Output files created successfully
- ✓ Logs generated with appropriate detail
- ✓ Intermediate outputs can be loaded

Folder Structure

```
student_resource/
├── dataset/
│   ├── train.csv                                  # Original training data
│   ├── test.csv                                   # Original test data
│   ├── sample_test.csv                            # Sample test data
│   ├── sample_test_out.csv                        # Sample output format
│   ├── processed/
│   │   ├── cleaned_train.csv                      # Preprocessed training data (brand + pack_quantity)
│   │   ├── cleaned_test.csv                       # Preprocessed test data (brand + pack_quantity)
│   │   ├── basic_image_features_train.csv         # Basic image features (train)
│   │   ├── basic_image_features_test.csv          # Basic image features (test)
│   │   ├── download_report_train.csv              # Image download status (train)
│   │   ├── download_report_test.csv               # Image download status (test)
│   │   ├── validation_report_train.csv            # Image validation results (train)
│   │   └── validation_report_test.csv             # Image validation results (test)
│   ├── embeddings/                                # Multi-model embeddings
│   │   ├── cnn_embeddings_train.npy               # EfficientNet-B0 embeddings (train)
│   │   ├── cnn_embeddings_test.npy                # EfficientNet-B0 embeddings (test)
│   │   ├── p_mlm_l6_v2_train.npy                  # Sentence transformer embeddings (5 models)
│   │   ├── p_mlm_l6_v2_test.npy
│   │   ├── p_ml_mlm_l12_v2_train.npy
│   │   ├── p_ml_mlm_l12_v2_test.npy
│   │   ├── a_mlm_l6_v2_train.npy
│   │   ├── a_mlm_l6_v2_test.npy
│   │   ├── a_mlm_l12_v2_train.npy
│   │   ├── a_mlm_l12_v2_test.npy
│   │   ├── distil_ml_base_v2_train.npy
│   │   └── distil_ml_base_v2_test.npy
│   └── images/
│       ├── train/                                 # Training images ({sample_id}.jpg)
│       ├── test/                                  # Test images ({sample_id}.jpg)
│       └── quarantine/                            # Corrupted images
│           ├── train/
│           └── test/
├── src/
│   ├── utils.py                                   # Existing utility functions
│   ├── example.ipynb                              # Existing example notebook
│   └── preprocessing_utils.py                     # NEW: Custom preprocessing utilities
├── notebooks/
│   └── data_preprocessing_pipeline.ipynb          # MAIN DELIVERABLE
├── logs/
│   ├── preprocessing_errors.log                   # Error logs
│   ├── download_logs.log                          # Download operation logs
│   ├── validation_logs.log                        # Validation logs
│   └── summary_report.md                          # Final summary report
├── Documentation_template.md                      # Submission template
├── README.md                                      # Challenge description
└── PREPROCESSING_GUIDE.md                         # NEW: Setup and usage guide
```

Implementation Phases

Phase 1: Setup and EDA (Notebook Sections 1-2)

Objectives:

- Import libraries and set up environment
- Load train.csv and test.csv
- Perform comprehensive EDA
- Understand data patterns

Key Outputs:

- Statistical summaries
- Visualizations (price distribution, text length distribution)
- Missing value analysis
- Initial insights

Estimated Time: 30 minutes of execution

Phase 2: Text Preprocessing (Notebook Section 3)

Objectives:

- Clean catalog_content text
- Handle missing values
- Normalize text

Key Outputs:

- cleaned_train.csv and cleaned_test.csv (partial)
- Text cleaning statistics

Estimated Time: 15 minutes of execution

Phase 3: Image Download (Notebook Section 4)

Objectives:

- Download all training images (75k)
- Download all test images (75k)
- Generate download reports

Key Outputs:

- dataset/images/train/ folder with images
- dataset/images/test/ folder with images
- download_report_train.csv
- download_report_test.csv

Estimated Time: 2-4 hours (network dependent)

Critical Considerations:

- Implement checkpointing to resume if interrupted
- Monitor disk space (expect ~10-20 GB for all images)
- Handle throttling gracefully

Phase 4: Image Validation (Notebook Section 5)

Objectives:

- Validate all downloaded images
- Quarantine corrupted files
- Extract image metadata

Key Outputs:

- validation_report_train.csv
- validation_report_test.csv
- Quarantined corrupted images

Estimated Time: 30-45 minutes

Phase 5: Image Feature Extraction (Notebook Section 6)

Objectives:

- Extract basic visual features from all valid images
- Save feature matrices

Key Outputs:

- image_features_train.csv
- image_features_test.csv

Estimated Time: 45-60 minutes

Phase 6: Data Quality Assurance (Notebook Section 7)

Objectives:

- Run comprehensive quality checks
- Generate quality reports
- Calculate data readiness score

Key Outputs:

- Quality report dictionary
- Data readiness score
- Recommendations for handling missing data

Estimated Time: 10 minutes

Phase 7: Summary and Handoff (Notebook Section 8)

Objectives:

- Generate final summary report
- Create visualizations of pipeline results
- Document next steps for team

Key Outputs:

- logs/summary_report.md
- Pipeline execution dashboard
- Handoff documentation

Estimated Time: 15 minutes

Performance Optimization

Parallel Processing Strategy

Image Operations (download, validation, feature extraction):

- Use `multiprocessing.Pool` with 50-100 workers
- Batch size: 1000 images per batch to manage memory
- Progress tracking with `tqdm`

Text Operations:

- Vectorized pandas operations where possible
- Apply functions with progress_apply from tqdm

Memory Management

Large Dataset Handling:

- Process images in batches to avoid loading all into memory
- Use generators for image processing pipelines
- Clear variables after each phase with `del` and `gc.collect()`

Disk Space Management:

- Estimate: ~15-20 GB for all images
- Compress intermediate CSVs if needed
- Provide cleanup utilities for temporary files

Checkpointing Strategy

Resume Capability:

- Save progress after each phase
- Check for existing outputs before reprocessing
- Enable selective re-running of phases

Example:

```
if not os.path.exists('dataset/processed/cleaned_train.csv'):
    # Run text preprocessing
    pass
else:
    print("Cleaned data already exists, loading...")
    cleaned_train = pd.read_csv('dataset/processed/cleaned_train.csv')
```

Integration with Team Workflow

Handoff to NLP Specialist

Deliverables:

- `cleaned_train.csv` and `cleaned_test.csv` with preprocessed text

- Text statistics and patterns documented
- Recommendations for feature extraction

Usage:

```
# NLP Specialist can load cleaned data
train = pd.read_csv('dataset/processed/cleaned_train.csv')
# Use cleaned_catalog_content for TF-IDF, embeddings, etc.
```

Handoff to Vision Specialist

Deliverables:

- `dataset/images/train/` and `dataset/images/test/` with organized images
- `image_features_train.csv` and `image_features_test.csv` with basic features
- `validation_report` showing which images are valid

Usage:

```
# Vision Specialist can load image features
img_features = pd.read_csv('dataset/processed/image_features_train.csv')
# Or load images directly
from PIL import Image
img = Image.open(f'dataset/images/train/{sample_id}.jpg')
```

Handoff to Model Builder

Deliverables:

- Complete preprocessed datasets ready for feature engineering
- Data quality reports for informed modeling decisions
- Statistics on data completeness

Usage:

```
# Model Builder can merge all data sources
train = pd.read_csv('dataset/processed/cleaned_train.csv')
img_features = pd.read_csv('dataset/processed/image_features_train.csv')
# Merge on sample_id for complete feature set
```

Competitive Advantages (Streamlined, Proven Approach)

What Makes Our Preprocessing Pipeline Competitive

1. **Multi-Model Text Embeddings**

Standard Approach (Most Teams):

- Single embedding model

- Maybe 1-2 models at most

Our Approach:

- □ **5 Sentence Transformers**: p_mlm_l6_v2, p_ml_mlm_l12_v2, a_mlm_l6_v2, a_mlm_l12_v2, distil_ml_base_v2
- □ **Hierarchical Neighbor Search**: KNN on 4 models + ANN on 1 model
- □ **140 Neighbor-Based Features**: Extract statistics from similar products' prices

2. **Critical Text Features** (Highest Impact)

Standard Approach (Most Teams):

- Just use raw text
- Maybe basic cleaning

Our Approach:

- □ **Brand Extraction**: Strongest single price predictor in e-commerce
- □ **Pack Quantity**: Critical factor (12-pack vs 1-pack = huge price difference)

Competitive Edge: Most teams will miss these obvious but powerful features

3. **CNN Image Embeddings** (Powerful Visual Features)

Standard Approach (Most Teams):

- Basic image statistics only
- Dimensions, colors

Our Approach:

- □ **EfficientNet-B0**: 1280-dimensional embeddings capturing product type, quality, presentation
- □ **Basic Features**: ~30 features for baseline

Competitive Edge: Deep learning features most teams won't implement

Summary: Why This Achieves Top 5 Potential

Feature Category	Standard Teams	Our Approach	Impact
Text Embeddings	1 model (384 dims)	5 models (2048 dims)	□□□□□
Text Features	0-2 features	Brand + Pack Quantity	□□□□□
Image Features	5-10 basic	30 basic + 1280 CNN	□□□□
Neighbor Features	None	Foundation for 140 features	□□□□□

Total Features: ~1310 streamlined, high-impact features

Inspiration from Previous Winners' Approach

Key Takeaways Applied and Enhanced

1. **Text Handling**:
    - Our approach: Preserve original + extract structured features (brand, numerical, quality)
    - Enable multilingual processing (winner used multilingual models) □
2. **Neighbor-Based Features**:
    - Our approach: Pre-compute multi-model embeddings for efficient hierarchical neighbor search

- Structured data enables Model Builder to replicate and enhance winner's strategy □

3. **Feature Engineering Foundation**:
   - Our approach: Provide 1350+ features + embeddings for even richer feature engineering
   - Extract advanced image features (CNN, quality, OCR) that winner didn't have □
4. **Data Quality**:
   - Our approach: Systematic handling + per-sample quality scoring for adaptive modeling □

Risk Mitigation

Risk 1: Image Download Failures

Likelihood: High (throttling, network issues) **Impact**: High (missing visual features)

Mitigation:

- Implement robust retry logic with exponential backoff
- Enable resume functionality
- Provide manual re-download utilities for failed images
- Document which samples have missing images for potential imputation

Risk 2: Corrupted Images

Likelihood: Medium **Impact**: Medium (affects visual features)

Mitigation:

- Comprehensive validation after download
- Quarantine and retry corrupted images
- Track which samples have valid images
- Enable text-only modeling for samples with missing images

Risk 3: Memory Issues During Processing

Likelihood: Medium (150k images) **Impact**: Medium (pipeline crashes)

Mitigation:

- Batch processing with configurable batch sizes
- Clear memory after each batch
- Monitor memory usage and provide warnings
- Process train and test separately if needed

Risk 4: Disk Space Exhaustion

Likelihood: Low (if disk space checked) **Impact**: High (pipeline failure)

Mitigation:

- Check available disk space before starting
- Estimate required space (20 GB) and warn user
- Provide cleanup utilities for intermediate files
- Compress large CSVs if needed

Risk 5: Data Loss During Preprocessing

Likelihood: Low (with proper design) **Impact**: Critical (disqualification)

Mitigation:

- Validate sample counts after each phase
- Never delete original data
- Comprehensive logging of all operations
- Quality checks to ensure 75k samples maintained

Success Criteria

The preprocessing pipeline is considered successful when:

1. ✓ All 75k training samples and 75k test samples are processed
2. ✓ Zero data loss (all sample_ids preserved)
3. ✓ >95% of images successfully downloaded and validated
4. ✓ Cleaned text data ready for NLP feature extraction
5. ✓ Basic image features extracted for all valid images
6. ✓ Comprehensive quality reports generated
7. ✓ Data readiness score >90%
8. ✓ All outputs properly organized and documented
9. ✓ Pipeline is reproducible and well-documented
10. ✓ Team members can easily access and use preprocessed data

Next Steps After Preprocessing

Once preprocessing is complete:

1. **NLP Specialist (Vamsi)**:
    - Use cleaned_catalog_content for TF-IDF, embeddings, keyword extraction
    - Extract numerical features (prices, quantities) from text
2. **Vision Specialist (Vamsi)**:
    - Build upon basic image features
    - Extract advanced features (CNN embeddings, object detection)
3. **Model Builder (Rahul)**:
    - Merge text and image features
    - Implement neighbor-based feature engineering
    - Train ensemble models
4. **Project Manager (Sachin)**:
    - Integrate all feature engineering outputs
    - Coordinate model training and evaluation