```python
import re
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# Expanded PESTEL dictionary
PESTEL_KEYWORDS = {
    'Political': ['policy', 'government', 'regulation', 'election', 'diplomacy', 'politics',
                  'parliament', 'law', 'governance', 'public', 'reform', 'voting', 'ethics'],
    'Economic': ['market', 'finance', 'trade', 'business', 'investment', 'growth', 'inflation',
'management',
                 'recession', 'tax', 'industry', 'corporation', 'employment', 'GDP', 'economy'],
    'Social': ['society', 'culture', 'education', 'health', 'community', 'demographics',
               'inequality', 'media', 'family', 'religion', 'values', 'social', 'behavior'],
    'Technological': ['technology', 'innovation', 'AI', 'digital', 'automation', 'robotics',
                      'software', 'hardware', 'internet', 'cybersecurity', 'data', 'platform',
'network'],
    'Environmental': ['environment', 'climate', 'sustainability', 'pollution', 'recycling',
'SDG',
                      'conservation', 'biodiversity', 'carbon', 'global warming', 'renewable',
'nature'],
    'Legal': ['law', 'regulation', 'compliance', 'litigation', 'intellectual property',
              'contracts', 'dispute', 'human rights', 'legal', 'taxation']
}

class DataReader:
    def __init__(self, file_paths):
        self.file_paths = file_paths

    def read_data(self):
        dfs = []
        for file_path in self.file_paths:
            try:
                df = pd.read_excel(file_path)
                dfs.append(df)
            except FileNotFoundError:
                print(f"Warning: File not found - {file_path}")
            except Exception as e:
                print(f"Error reading {file_path}: {e}")
        if dfs:
            return pd.concat(dfs, ignore_index=True)
        else:
            return pd.DataFrame()

class TextDocumentCleaner:
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, df):
        df['cleaned_documents'] = df['documents'].astype(str).apply(self.clean)
        return df

    def clean(self, text):
        text = re.sub(r'\d+', '', text)
        text = re.sub(r'[^\w\s]', '', text)
        text = re.sub(r'\s+', ' ', text).strip().lower()
        return text

class TopicModel:
    def __init__(self, num_topics=6, max_features=3000):
        self.num_topics = num_topics
        self.vectorizer = TfidfVectorizer(stop_words='english', max_features=max_features,
max_df=0.9, min_df=5)
        self.lda = LatentDirichletAllocation(n_components=num_topics, max_iter=100,
learning_method='batch', random_state=42)
        self.terms = None
    def fit(self, documents):
        X = self.vectorizer.fit_transform(documents)
        self.lda.fit(X)
        self.terms = self.vectorizer.get_feature_names_out()
        return X

    def transform(self, documents):
        return self.lda.transform(self.vectorizer.transform(documents))

    def get_topics(self, top_n=10):
        return [[self.terms[i] for i in topic.argsort()[:-top_n-1:-1]] for topic in
self.lda.components_]

class PESTELMapper:
    def __init__(self, pestel_keywords):
        self.pestel_keywords = pestel_keywords

    def fit(self, X, y=None):
        return self

    def transform(self, topics):
        return [[dim for dim, keywords in self.pestel_keywords.items() if set(terms) &
set(keywords)]
                or ['Unclassified'] for terms in topics]

class TopicEvolutionAnalyzer:
    def __init__(self, topic_model, topic_mapper):
        self.topic_model = topic_model
        self.topic_mapper = topic_mapper

    def analyze(self, df):
        # Extract topics and map to PESTEL
        topics = self.topic_model.get_topics()
        topic_mapping = self.topic_mapper.transform(topics)

        print("\n=== Identified Topics ===")
        for idx, terms in enumerate(topics):
            print(f"Topic {idx + 1}: {' '.join(terms)} --> {', '.join(topic_mapping[idx])}")

        # Track and plot topic evolution
        self.track_evolution(df)

    def track_evolution(self, df):
        df['Year'] = df['Year'].astype(int)
        years = sorted(df['Year'].unique())
        evolution_df = pd.DataFrame({
            year: self.topic_model.transform(df.loc[df['Year'] == year,
'cleaned_documents']).mean(axis=0)
            for year in years
        }, index=[f'Topic {i+1}' for i in range(self.topic_model.num_topics)]).T
        evolution_df.plot(figsize=(12, 8), marker='o', title="Evolution of Topics Over Time")
        plt.xlabel('Year')
        plt.ylabel('Average Contribution')
        plt.legend(title="Topics", bbox_to_anchor=(1.05, 1), loc='upper left')
        plt.grid(True)
        plt.show()
```