Data Analysis Report

## 1. Introduction

This report provides an analysis of experimental data from XML and JSON sources. We will parse, clean, and visualize the data to uncover key insights.

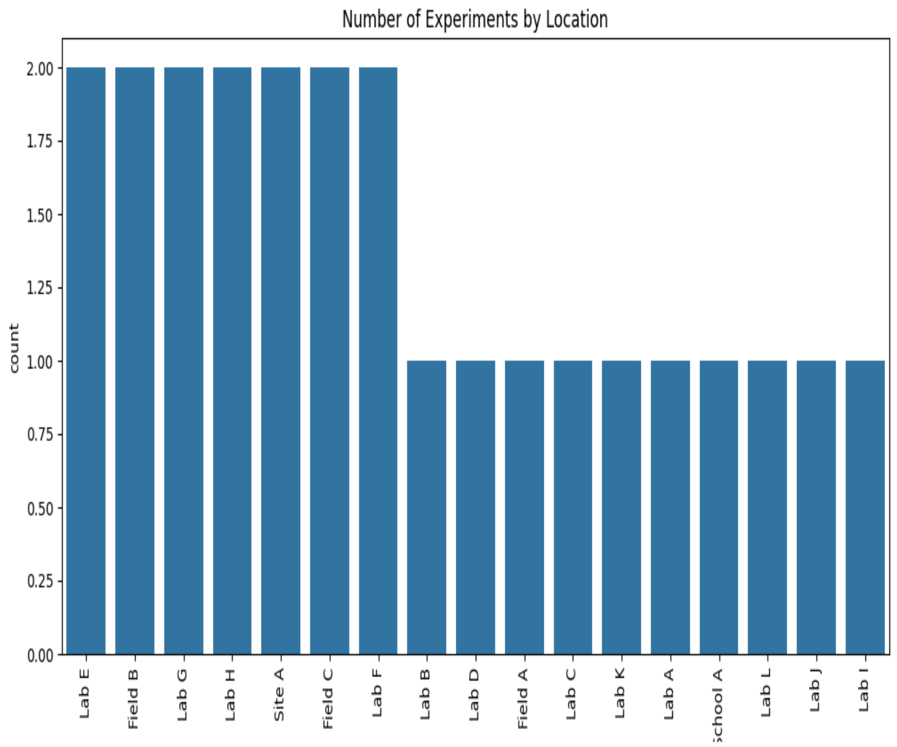
## 2. Data Processing Code

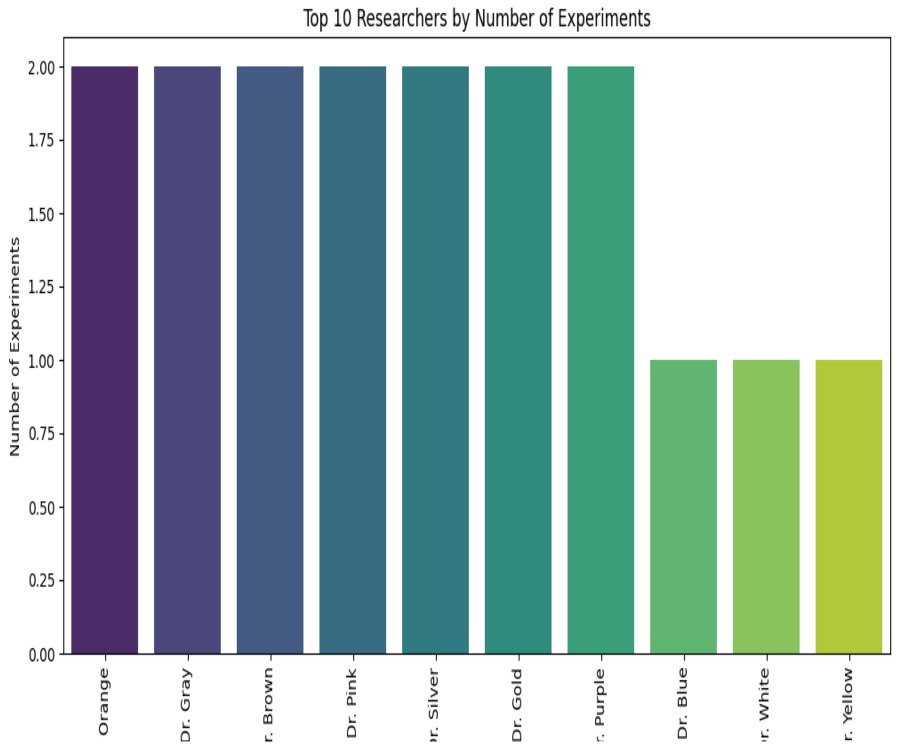
Below is the code used for data parsing, cleaning, and transformation.

```
import os import xml.etree.ElementTree as ET import json import pandas as pd
import numpy as np import matplotlib.pyplot as plt import seaborn as sns
import logging from reportlab.lib.pagesizes import letter from
reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle from
reportlab.platypus import SimpleDocTemplate, Paragraph, Image, Spacer #
Setup logging logging.basicConfig(level=logging.INFO, format='%(asctime)s -
%(levelname)s - %(message)s') # Data processing functions def
parse_xml(file_path): try: tree = ET.parse(file_path) root = tree.getroot()
xml_data = [] for experiment in root.findall('experiment'): exp_id =
experiment.find('experiment_id').text description =
experiment.find('description').text results = experiment.find('results')
metadata = experiment.find('metadata') result_dict = { 'experiment_id':
exp_id, 'description': description, 'control': results.find('control').text
if results.find('control') is not None else None, 'test':
results.find('test').text if results.find('test') is not None else None,
'reaction_time': results.find('reaction_time').text if
results.find('reaction_time') is not None else None, 'purity_increase':
results.find('purity_increase').text if results.find('purity_increase') is
not None else None, 'efficiency_gain': results.find('efficiency_gain').text
if results.find('efficiency_gain') is not None else None,
'thermal_resistance': results.find('thermal_resistance').text if
results.find('thermal_resistance') is not None else None,
'heart_health_index': results.find('heart_health_index').text if
results.find('heart_health_index') is not None else None,
'durability_increase': results.find('durability_increase').text if
results.find('durability_increase') is not None else None, 'yield_increase':
results.find('yield_increase').text if results.find('yield_increase') is not
None else None, 'air_quality_index': results.find('air_quality_index').text
if results.find('air_quality_index') is not None else None,
'bacterial_reduction_index': results.find('bacterial_reduction_index').text
if results.find('bacterial_reduction_index') is not None else None,
'battery_life_gain': results.find('battery_life_gain').text if
results.find('battery_life_gain') is not None else None,
'performance_index': results.find('performance_index').text if
results.find('performance_index') is not None else None, 'date':
metadata.find('date').text, 'researcher': metadata.find('researcher').text,
'location': metadata.find('location').text } xml_data.append(result_dict)
return pd.DataFrame(xml_data) except Exception as e: logging.error(f"Error
parsing XML file: {e}") return pd.DataFrame() def parse_json(file_path):
try: with open(file_path, 'r') as f: json_data = json.load(f)
json_parsed_data = [] for entry in json_data: result_dict = {
'experiment_id': entry['experiment_id'], 'description':
entry['description'], 'control': entry['results'].get('control', None),
'test': entry['results'].get('test', None), 'reaction_time':
entry['results'].get('reaction_time', None), 'purity_increase':
entry['results'].get('purity_increase', None), 'efficiency_gain':
entry['results'].get('efficiency_gain', None), 'thermal_resistance':
entry['results'].get('thermal_resistance', None), 'heart_health_index':
entry['results'].get('heart_health_index', None), 'durability_increase':
entry['results'].get('durability_increase', None), 'yield_increase':
entry['results'].get('yield_increase', None), 'air_quality_index':
entry['results'].get('air_quality_index', None),
'bacterial_reduction_index':
entry['results'].get('bacterial_reduction_index', None),
'battery_life_gain': entry['results'].get('battery_life_gain', None),
'performance_index': entry['results'].get('performance_index', None),
'date': entry['metadata']['date'], 'researcher':
entry['metadata']['researcher'], 'location': entry['metadata']['location'] }
json_parsed_data.append(result_dict) return pd.DataFrame(json_parsed_data)
except Exception as e: logging.error(f"Error parsing JSON file: {e}") return
pd.DataFrame() def standardize_date(date_str): try: return
pd.to_datetime(date_str, errors='coerce').strftime('%Y-%m-%d') except:
return np.nan def fill_missing_values(df): numerical_cols =
df.select_dtypes(include=np.number).columns.tolist() categorical_cols =
df.select_dtypes(include='object').columns.tolist() for col in
numerical_cols: if col in df.columns: df[col] = pd.to_numeric(df[col],
errors='coerce') df[col] = df[col].fillna(df[col].mean()) for col in
categorical_cols: if col in df.columns: df[col] = df[col].fillna('Unknown')
return df def plot_visualizations(df): try: plt.figure(figsize=(12, 10))
numerical_cols = df.select_dtypes(include=np.number).columns.tolist() if
numerical_cols: df[numerical_cols].hist(figsize=(12, 10), bins=20,
edgecolor='black') plt.tight_layout()
plt.savefig('numerical_distributions.png') plt.close() else:
logging.warning("No numerical columns to plot histograms.") if not
df[numerical_cols].isnull().all().all(): correlation_matrix =
df[numerical_cols].corr() plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap') plt.savefig('correlation_heatmap.png')
plt.close() else: logging.warning("Correlation heatmap could not be
generated due to all NaN values.") plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='location',
order=df['location'].value_counts().index) plt.xticks(rotation=90)
plt.title('Number of Experiments by Location')
plt.savefig('experiments_by_location.png') plt.close() top_researchers =
df['researcher'].value_counts().nlargest(10) plt.figure(figsize=(12, 6))
sns.barplot(x=top_researchers.index, y=top_researchers.values,
palette='viridis') plt.xticks(rotation=90) plt.title('Top 10 Researchers by
Number of Experiments') plt.ylabel('Number of Experiments')
plt.savefig('top_researchers.png') plt.close() except Exception as e:
logging.error(f"Error generating visualizations: {e}")
```

## 3. Visualizations

The following visualizations provide insights into the experimental data.





## 4. Summary

The data was successfully parsed, cleaned, and transformed. The visualizations reveal key patterns and trends in the experimental data.