Data & Methodology

Please note that the date/time and results of each web scraping process may vary. Changes in the publishing source's internet policies, the number of available news items, or the removal/restriction of past archives directly affect accessibility and results. The merged and cleaned file (merged_cleaned_20251208_174757_with_sentiment.csv) using is provided below.

1) Data Dictionaries

ESG News Headlines with Sentiment Analysis:

(v2_merged_cleaned_20251208_174757_with_sentiment.csv)

Used Headings;

category, headline, publish_year, GPT_sentiment, Rule_Based_Model_Sentiment, Hybrid_LR_Sentiment, Hybrid_NB_Sentiment, Hybrid_RF_Sentiment

Companies ESG Scores and Financial Variables:

(company_esg_financial_dataset.csv)

Used Headings;

Industry, Region, Year, Revenue,ESG_Overall, ESG_Environmental, ESG_Social, ESG_Governance, CarbonEmissions, WaterUsage, EnergyConsumption

Green Bond Variables:

File name: Green_Bonds.csv

Used Headings;

Bond_Type, F2018, F2019, F2020, F2021, F2022, F2023, F2024

Other Files:

esg_all_categories_20260104_162448.csv: ESG Today web scrapping data file

pa_news_20251208_163416.csv: Future Portfolio web scrapping data file

merged_cleaned_20251208_174757_with_sentiment.csv: just news headlines and GPT sentiment

2) General Code Flow

Part 1

Web Scrapping 1:

- Libraries & utilities

from bs4 import BeautifulSoup, import csv, from datetime import etc.

- Scrapping

Build page URL → send request → parse HTML → extract headline, link, publish date/year

- HTTP error handling

```
except requests.exceptions.HTTPError as e:
if e.response.status_code == 404:
```

- Scraping a all categories + break bot wall

```
def scrape_category(session, base_url, category_name, delay_min=1, delay_max=3):
    all_results = []
    page_num = 1
    while True:
        results, should_continue = scrape_single_page(
            session, base_url, category_name, page_num
        )
        if results:
            all_results.extend(results)
            page_num += 1
            time.sleep(random.uniform(delay_min, delay_max))
        else:
            break
    return all_results
```

- Saving and printing results

Web Scrapping 2:

- Libraries & utilities

from bs4 import BeautifulSoup, from dateutil import parser, from urllib.parse import urljoin, import requests, import random, import time etc.

- Category URLs

- True page counts (manual cap to avoid infinite loops)

- Request headers + rate limits

```
    'User-Agent': '(KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36',
    'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
    'Accept-Language': 'en-UK,en;q=0.9',
    'Connection': 'keep-alive'
DELAY_MIN, DELAY_MAX = 8.0, 12.0
```

- *Single-page request handler (break 429 rate limit)*

```
def scrape_page(url):
    try:
        resp = requests.get(url, headers=HEADERS, timeout=40)
        if resp.status_code == 429:
            return None, 429
        if resp.status_code >= 400:
            return None, resp.status_code
        return resp.text, 200
    except requests.exceptions.Timeout:
        return None, "timeout"
    except Exception as e:
        return None, str(e)
```

- Category-level scraping (true pagination logic) (Cloudflare-safe, polite scrapping // time delay)

```
def scrape_category(category, base_url):
    total_pages = MAX_PAGES[category]
    all_rows = []
    backoff = 90
    for page in range(1, total_pages + 1):
        if page == 1:
            page_url = base_url
        else:
            page_url = f"{base_url}&paged={page}"
        html, status = scrape_page(page_url)
        if status == 429:
            polite_wait(backoff)
            backoff = min(backoff * 1.5, 300)
            continue
        if status != 200:
            continue
        backoff = 90
        rows = extract_articles(html, category)
        all_rows.extend(rows)
        time.sleep(random.uniform(DELAY_MIN, DELAY_MAX))
    return all_rows
```

- Scraping a all categories
- Saving and printing results

Merge&Clean Dataset (merged_cleaned_20251208_174757_with_sentiment.csv):

- Libraries & utilities
- Load + clean + merge datasets
- Saving and printing results

Top 100 most frequent words in headlines:

- Libraries & utilities
- Basic text cleaning (NLP preprocessing)

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)      # remove URLs
    text = re.sub(r"[^a-z\s]", "", text)     # keep letters only
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

- Tokenization & stopword removal

```
stopwords = set([
    "the","a","an","and","or","to","of","in","on","for","with","at","by",
    "from","as","is","are","was","were","be","has","have","had","will",
    "its","it","this","that","these","those","after","before","over",
    "new","says","said"
])
words = []
for text in df["clean_headline"]:
    words.extend([w for w in text.split() if w not in stopwords and len(w) > 2])
```

- Visualization Top 30 words

Sentiment Analysis:

- Libraries & utilities
- Keyword lists + lowercasing

```
positive_keywords = [ ... ]
negative_keywords = [ ... ]
neutral_keywords = [ ... ]
positive_keywords = [k.lower() for k in positive_keywords]
negative_keywords = [k.lower() for k in negative_keywords]
neutral_keywords = [k.lower() for k in neutral_keywords]
```

- Rule-based labeling logic (priority: Negative > Positive > Neutral)

```
def rule_based_label(text: str) -> str:
    t = str(text).lower()
    pos = sum(1 for kw in positive_keywords if kw in t)
    neg = sum(1 for kw in negative_keywords if kw in t)
    neu = sum(1 for kw in neutral_keywords if kw in t)
    if neg > 0:
        return "Negative"
    if pos > 0:
        return "Positive"
    if neu > 0:
        return "Neutral"
    return "Neutral"
```

- Create TF-IDF features
- Train-test split
- Define ML models (LR, Naive Bayes, Random Forest)
- Evaluate models
- Model Comparison + Accurancy
- Save updated dataset (v2_merged_cleaned_20251208_174757_with_sentiment.csv) + print final status

Sentiment Distribution Across NLP Models x Year:

- Libraries & utilities
- Normalize Sentiments
- Drop NA

```
df_clean = df.dropna(subset=list(MODEL_COLS.values())).copy()
print("Rows after sentiment cleaning:", len(df_clean))
```

- Year × Sentiment × Model breakdown
- Visualize Sentiment Distribution Across NLP Models

Yearly ESG News Sentiment by Category (Hybrid_LR):

- Libraries & utilities
- Year cleaning + year range + category filtering
- Normalize Sentiments
- Plot settings + chart formatting

Scaling Sentiment Scores (Weighted Average Score):

- Libraries & utilities
- Normalize Sentiments
- Load datasets + filtering
- Weighted Average Score {"Positive": 75, "Neutral": 50, "Negative": 25}

```
pivot = out.pivot_table(index="Year", columns="Model", values="Avg_Sentiment_Score",
aggfunc="mean")
```

- Plot Yearly Average Sentiment Score

Part 2

Nominal Broad U.S. Dollar Index + Green Bond Investments:

- Libraries & utilities
- Load dataset + renaming + filtering
- Normalize bond types + mapping
- Nominal Broad U.S. Dollar Index adjustment (constant 2024 value)
- Plot total investment by years

```
yearly_totals = df[year_cols].sum(axis=0)
total_2018_2024 = yearly_totals.sum()
print(f"Total bond investment (2018–2024): {fmt2(total_2018_2024)} $B")
plt.figure(figsize=(12, 6))
plt.bar([int(y) for y in year_cols], yearly_totals.values)
plt.xlabel("Year")
plt.ylabel("Investment ($B, constant 2024 USD)")
plt.title(f"Bond Investments by Year (2018–2024) — Total: {fmt2(total_2018_2024)} $B")
plt.grid(axis="y", alpha=0.3)
plt.xticks([int(y) for y in year_cols])
plt.tight_layout()
plt.show()
```

- Plot total investment by bond type over years (2018-2024)

```
type_year = df_types.groupby("Bond_Type")[year_cols].sum()
type_total = type_year.sum(axis=1)
type_total = type_total.reindex(target_types).fillna(0.0)
print("\nTotal by Bond_Type (2018–2024):")
print(type_total.round(2).apply(fmt2))
plt.figure(figsize=(10, 5))
plt.bar(type_total.index, type_total.values)
plt.xlabel("Bond_Type")
plt.ylabel("Investment ($B, constant 2024 USD)")
plt.title("Total Investment by Bond Type (2018–2024)")
plt.grid(axis="y", alpha=0.3)
plt.tight_layout()
plt.show()
```

Part 3

Overall ESG Scores by Region (company_esg_financial_dataset.csv):

- Libraries & utilities
- Filtering and cleaning dataset
- Region order

```
region_order = (
    df.groupby("Region")["ESG_Overall"]
      .median()
      .sort_values()
      .index
      .tolist()
)
df["Region"] = pd.Categorical(
    df["Region"],
    categories=region_order,
    ordered=True
)


summary = []
for region in region_order:
    sub = df[df["Region"] == region]["ESG_Overall"]
    q1 = sub.quantile(0.25)
    q3 = sub.quantile(0.75)
    iqr = q3 - q1
    lower = q1 - 1.5 * iqr
    upper = q3 + 1.5 * iqr
    outliers = ((sub < lower) | (sub > upper)).sum()
    mode_val = sub.mode()
    mode_val = mode_val.iloc[0] if not mode_val.empty else np.nan
    summary.append({
        "Region": region,
        "Count": len(sub),
        "Min": round(sub.min(), 2),
        "Max": round(sub.max(), 2),
        "Mean": round(sub.mean(), 2),
        "Median": round(sub.median(), 2),
        "Mode": round(mode_val, 2),
        "Q1": round(q1, 2),
        "Q3": round(q3, 2),
        "IQR": round(iqr, 2),
        "Outlier_Count": int(outliers)
    })
```

- Plot ESG Overall Score by Region

ESG Overall by Industry:

- Libraries & utilities
- Filtering and cleaning dataset
- Industry order
- Plot ESG Overall Score by Industry

Year-over-year Change Overall Corporate ESG Score:

- Libraries & utilities
- Filtering and cleaning dataset
- Yearly averages

```
yearly_avg = (
  df.groupby("Year")[esg_cols]
    .mean()
    .reindex(YEARS)
    .round(2))
```

- YoY % change

```
overall_yoy = (yearly_avg["ESG_Overall"].pct_change() * 100).round(2)
yearly_avg["Overall_YoY_%_Change"] = overall_yoy
```

Year-over-year Change Overall ESG Sentiment Score:

- Libraries & utilities
- Normalize Sentiments
- Rename columns + standardized headings

```
yearly_cat = yearly_cat.rename(columns={
  "environmental": "Environmental",
  "social": "Social",
  "governmental": "Governance"
}).round(2)
```

- Overall (E+S+G combined (Sentiment+Corporate Performance) yearly average + YoY%

```
overall_yearly = (
  df.groupby("publish_year")["Sentiment_Score"]
    .mean()
    .reindex(YEARS)
)
```

```
yearly_cat["Overall_Sentiment_ESG_Score"] = overall_yearly.round(2)
yearly_cat["Overall_YoY_%_Change"] = (overall_yearly.pct_change() * 100).round(2)
yearly_cat
```

Correlation Heatmap:

- Libraries & utilities
- Clear rendering
- Numeric Columns

```
numeric_cols = [
    "Revenue", "MarketCap", "GrowthRate", "ProfitMargin",
    "ESG_Environmental", "ESG_Social", "ESG_Governance", "ESG_Overall",
    "CarbonEmissions", "WaterUsage", "EnergyConsumption"
]
for col in numeric_cols:
    df[col] = pd.to_numeric(df[col], errors="coerce")
df_num = df[numeric_cols].dropna()
corr = df_num.corr(method="pearson")
```

- Heatmap

```
fig, ax = plt.subplots(figsize=(12, 10))
data = corr.values
n = data.shape[0]
mesh = ax.pcolormesh(
    np.arange(n + 1), np.arange(n + 1), data,
    cmap="coolwarm", vmin=-1, vmax=1,
    shading="flat",
    edgecolors="none", linewidth=0,
    antialiased=False
)
mesh.set_rasterized(True)
ax.set_aspect("equal")
ax.invert_yaxis()  #

ax.set_xticks(np.arange(n) + 0.5)
ax.set_yticks(np.arange(n) + 0.5)
ax.set_xticklabels(corr.columns, rotation=90)
ax.set_yticklabels(corr.index)
ax.tick_params(which="both", bottom=False, left=False)
ax.set_title(f"Correlation Heatmap of Numeric Features ({start_year}–{end_year})")
cbar = fig.colorbar(mesh, ax=ax, fraction=0.046, pad=0.04)
```

- Plot

Revunue by ESG log scale:

- Libraries & utilities
- NBUS adjusment Revenue
- Median (50%) trend: revenue-binned median ESG

```
logx = np.log10(x)
n_bins = 35
bins = np.linspace(logx.min(), logx.max(), n_bins + 1)
df["_bin"] = pd.cut(logx, bins=bins, include_lowest=True)
binned = (
    df.groupby("_bin", observed=False)
      .agg(rev_median=("Revenue_Real_2024", "median"),
           esg_median=("ESG_Overall", "median"))
      .dropna()
      .sort_values("rev_median")
)
plt.figure(figsize=(14, 7))
sc = plt.scatter(
    x, y,
    c=y,
    cmap="viridis",
    vmin=0, vmax=100,
    s=10,
    alpha=0.45,
    linewidths=0
```

- Formatting and plotting

Overall ESG Scores (Top 500 firms and others):

- Libraries & utilities
- NBUS adjusment Revenue
- Load&clean dataset

- Select Top 500 companies by revenue (NBUS adjusted)

```
top500_ids = (
  df.groupby(company_col)["Revenue_Real_2024"]
    .mean()
    .sort_values(ascending=False)
    .head(TOP_N)
    .index
)
df_top = df[df[company_col].isin(top500_ids)].copy()
df_other = df[~df[company_col].isin(top500_ids)].copy()
```

- Yearly average ESG (Top500 vs Other)

```
avg_top = (
  df_top.groupby("Year")["ESG_Overall"]
      .mean()
      .reindex(YEARS)
)
avg_other = (
  df_other.groupby("Year")["ESG_Overall"]
       .mean()
       .reindex(YEARS)
)
```

- Autolimit nearest preposition

```
vals = pd.concat([avg_top, avg_other]).dropna()
ymin = np.floor(vals.min() / 10) * 10
ymax = np.ceil(vals.max() / 10) * 10
if ymax - ymin < 10:
  ymax = ymin + 10
```

- Plot

Average Resource Consumptions and C02 Release:

- Libraries & utilities
- Filtering + selecting
- Average values between 2018-2024 + convert percentage

```
avg_values = df[cols].mean()
percent_share = (avg_values / avg_values.sum()) * 100
```

- Plot pie chart

```
plt.pie(
    percent_share,
    labels=[
        "Carbon Emissions",
        "Water Usage",
        "Energy Consumption"
    ],
    autopct="%1.1f%%",
    startangle=140
)
plt.title("Average  Resource Consumptions and C02 Release (2018–2024)")
plt.tight_layout()
plt.show()
```

Top 500 Companies' Share of Total Resource Consumption (2018-2024):

- Libraries & utilities
- Filtering & Cleaning
- NBUS adjusted revenue
- Select Top 500 companies
- Aggregate totals (2018–2024)

```
metrics = ["CarbonEmissions", "WaterUsage", "EnergyConsumption"]
total_all = df[metrics].sum()
total_top = df_top[metrics].sum()
share_pct = (total_top / total_all) * 100
```

- Plot