# CSCA 5632 - Intro. to Machine Learning

## Unsupervised Learning

Dyego Fernandes de Sousa

University of Colorado Boulder

November 28, 2025

# This is a job for the Un**SUPER**vised Learning. Problem Statement

To showcase and apply techniques of **Unsupervised Learning** at the brand-level data to discover:

- **Hidden Patterns**
  - **Natural groupings** Clusters of brands based on their multi-dimensional characteristics
  - **Unusual patterns** and outliers that don't fit conventional categorizations
  - **Relationships** between environmental performance, market positioning, and consumer targeting
- **Latent Features** that drive brand differentiation

**The Hypothesis:**
A company being compliant with ESG practices, doesn't mean that all of their brands also comply?

# Dataset

**Primary Dataset:** Comprehensive brand-level dataset, based on my **Supervised Learning project**. It contains:

- **Industry & Company Data**: ESG scores, emissions (scope1+2), revenues, environmental risk indicators
- **Brand Demographics**: Age group targeting, income levels, lifestyle segments
- **Operational Features**: Franchise model, online sales, fleet ownership, drive-through
- **Sustainability Metrics**: Electric vehicle %, ESG programs, sustainability awards, R&D spending

**Dataset Shape:**

- 3,600+ brands across multiple industries and parent companies
- 77+ features

# Methodology

- 1. **Data Loading and Preprocessing**: Loading, handling missing values, etc
- 2. **Feature Engineering**: Creating additional features, etc
- 3. **Feature Selection**: Select relevant features
- 4. **Feature Scaling**: Standardizization
- 5. **Dimensionality Reduction**:
  - **PCA** for variance reduction and **latent feature discovery**
  - t-SNE for 2D visualization
- 6. **Hyperparameter Tuning**:
  - Grid search and multi-metric evaluation
- 7. **Clustering with Optimized Parameters**:
  - K-Means, Hierarchical, DBSCAN (outliers/noise)
- 8. **Cluster Interpretation**: Profile segments
- 9. **Validation**: Compare algorithms

# Feature Engineering

Creating features that capture ESG risk and sustainability patterns.

```
greenwashing_combined = (
    initial_greenwashing_risk + ... +
    greenwashing_factor_4).clip(lower=0,
    upper=1)

other_risks_combined = full_df['
    deforestation_risk'].fillna(0) + ...
    + (full_df['
    product_portfolio_diversity'].fillna
    (0) / 10)

env_risk_combined = greenwashing_combined
     + other_risks_combined
full_df['environmental_risk_score'] =
    env_risk_combined
```

Figure: Feature Engineering

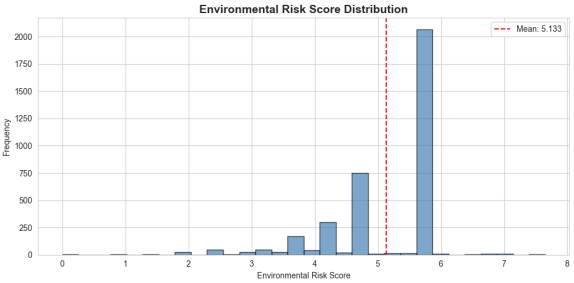


Figure: Environmental Risk Score plot

# Features Selection for Clustering + Encoding

Focus on features that capture **Company-level ESG**, **Brand demographics**, **Operational characteristics**, **Sustainability metrics**.

```
# Define feature groups
company_esg_features = ['scope12_total', ..., '
    positive_innovation_risk']
demographic_features = ['income_middle', '
    income_high', 'income_premium','
    income_snap_support']
lifestyle_features = []
operational_features = ['online_sales', ..., '
    supply_chain_localization_encoded']
sustainability_features = ['
    electric_vehicles_percent', ..., '
    sustainability_actions_encoded']
divergence_features = ['sustainability_positioning',
     ..., 'esg_premium_divergence']
market_features = ['customer_loyalty_index', ..., '
    year_of_foundation']
interaction_features = ['emissions_intensity', ...,
    'env_risk_concentration']

feature_columns = (company_esg_features +
    demographic_features + lifestyle_features +
    operational_features +  sustainability_features
    + divergence_features + market_features +
    interaction_features)
```

Figure: Logical grouping of features

```
def encode_esg_programs(val):
def encode_sustainability_actions(val):
def encode_fossil_fuel(val):
def encode_diversity(val):
def encode_localization(val):

full_df['esg_programs_encoded'] = full_df['
    esg_programs'].apply(encode_esg_programs)
full_df['sustainability_actions_encoded'] = full_df[
    'sustainability_actions'].apply(
    encode_sustainability_actions)
full_df['fossil_fuel_reliance_encoded'] = full_df['
    fossil_fuel_reliance'].apply(encode_fossil_fuel)
full_df['product_portfolio_diversity_encoded'] =
    full_df['product_portfolio_diversity'].apply(
    encode_diversity)
full_df['supply_chain_localization_encoded'] =
    full_df['supply_chain_localization'].apply(
    encode_localization)
```

Figure: Encoding categorical features

# Feature Engineering and Scaling

Examine alignment/misalignment between:

- Brand sustainability positioning vs. company environmental performance
- Premium pricing vs. actual ESG investment
- Marketing claims vs. operational reality
- Check the **Appendix** for more details

```
full_df['sustainability_positioning'] = ( full_df['electric_vehicles_percent'] * 0.4 + (0.3 if
    full_df['esg_programs'].count() > 0 else 0) + full_df['major_sustainability_award_last5y'].astype
    (float) * 0.3)
emissions_normalized = (full_df['scope12_total'] - full_df['scope12_total'].min()) / (full_df['
    scope12_total'].max() - full_df['scope12_total'].min())
full_df['company_env_risk'] = (emissions_normalized * 0.5 + full_df['environmental_risk_score'] *
    0.5)
full_df['sustainability_divergence'] = full_df['sustainability_positioning'] - (1 - full_df['
    company_env_risk'])
full_df['premium_positioning'] = (full_df['age_seniors'].astype(float) * 0.4 + full_df['
    income_premium'].astype(float) * 0.6)
full_df['esg_premium_divergence'] = full_df['premium_positioning'] - (1 - full_df['company_env_risk'
    ])
```

Figure: Feature Engineering and Scaling

# Divergence and Hipocresy Analysis Result

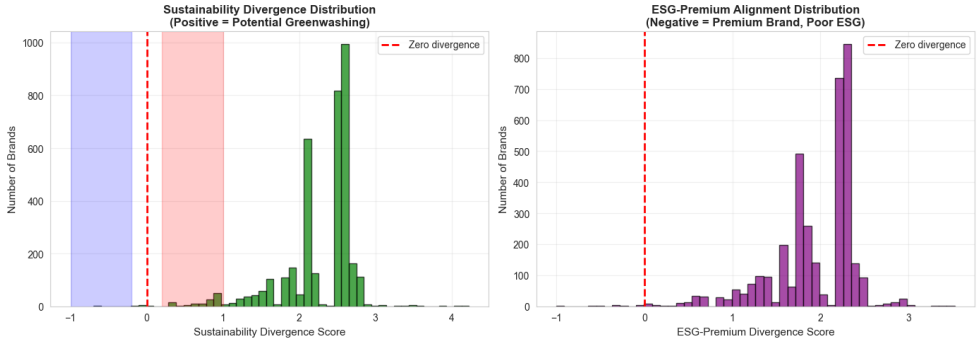


Figure: Divergence plot

# Dimensionality Reduction

## Applying PCA to reduce dimensionality

```
X_pca, pca_model = reducer.apply_pca(
    X_scaled, n_components=0.95)
```

```
variance_df = reducer.
    get_variance_dataframe()
fig = viz.plot_variance_explained(
    variance_df, n_components=min(15,
    X_pca.shape[1]))
plt.show()
```

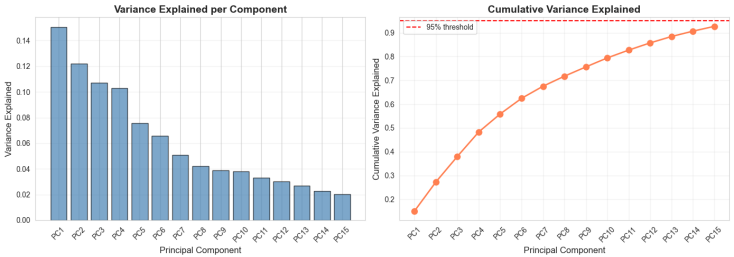


Figure: Variance Explained after PCA plots

# Top Features per Component

```python
# Analyze top features per component
top_features = reducer.
    get_top_features_per_component(
    feature_columns, n_top=5)

print("Top 5 features for first 3
    principal components:")
print(top_features[top_features['
    Component'].isin(['PC1', 'PC2',
    'PC3'])])

# Plot feature importance for PC1
fig = viz.plot_feature_importance(
    top_features, component='PC1',
    n_features=10)
plt.show()
```

Figure: Visualization code

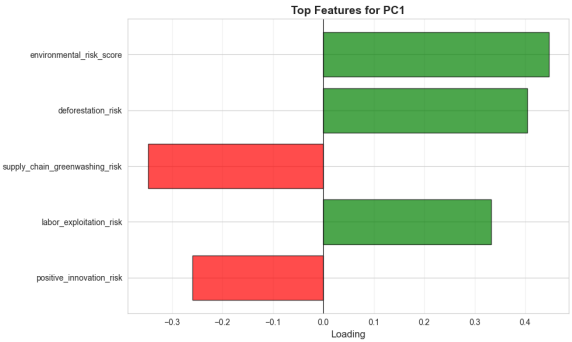


Figure: Visualization

# Latent Feature Interpretation

Latent features discovered. Check the **Appendix** for the code.

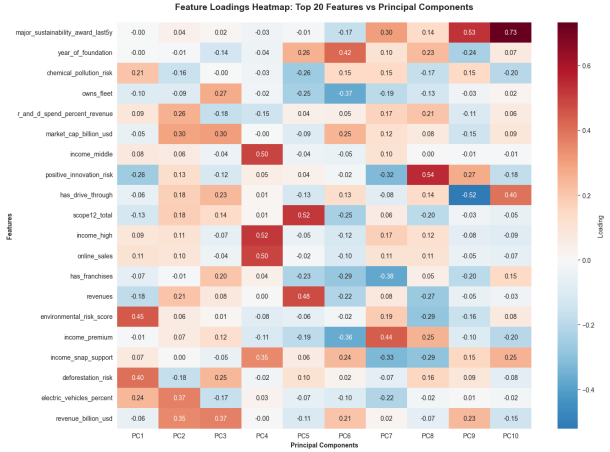


Figure: Latent Features

# PCA Visualing the Latent Space

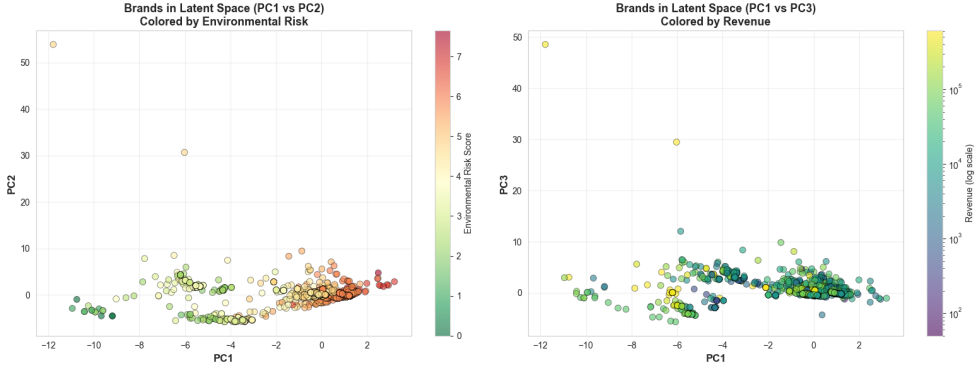


Figure: Plots in the Latent Space

# Hyperparameter Tuning - K-Means

Systematic hyperparameter tuning on **K-Means**, **Hierarchical**, and **DBSCAN Best parameters used in final clustering and analysis.**

```
tuner = HyperparameterTuner(verbose=True)
```

Figure: Instantiating the HP tuner, check the Appendix and the git repo for more details

```

kmeans_tuning_results = tuner.tune_kmeans(
    X_pca,
    param_grid=kmeans_param_grid,
    metric=OPTIMIZATION_METRIC
)
fig = viz.plot_hyperparameter_tuning(
    kmeans_tuning_results,
    algorithm='kmeans',
    metrics=['silhouette', 'calinski_harabasz', '
        davies_bouldin', 'inertia']
)
plt.show()
```

Figure: HP tuning K-Means

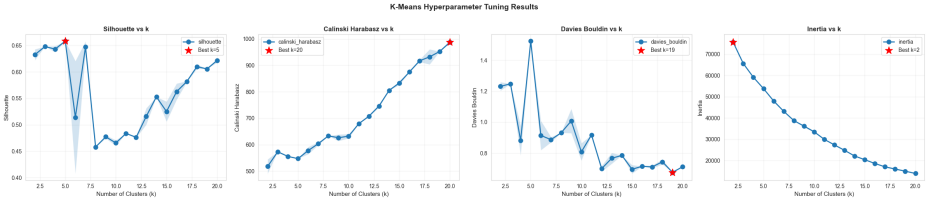


Figure: HP tuning K-Means - plot

# Hyperparameter Tuning - Hierarchical

```python
hierarchical_tuning_results = tuner.tune_hierarchical(
    X_pca,
    param_grid=hierarchical_param_grid,
    metric=OPTIMIZATION_METRIC
)
fig = viz.plot_hyperparameter_tuning(
    hierarchical_tuning_results,
    algorithm='hierarchical',
    metrics=['silhouette', 'calinski_harabasz', 'davies_bouldin']
)
plt.show()
```

Figure: HP Tuning Hierarchical

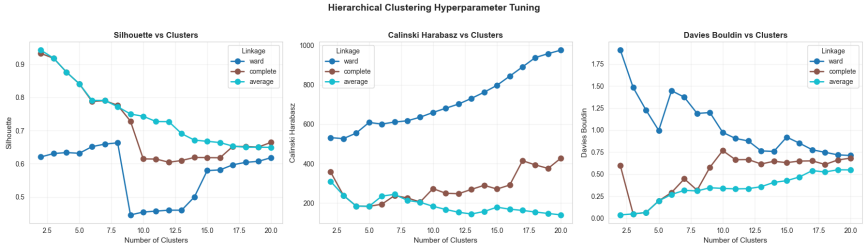


Figure: HP Tuning Hierarchical - plot

# Hyperparameter Tuning - DBSCAN

```python
dbscan_tuning_results = tuner.tune_dbscan(
    X_tsne,
    param_grid=dbscan_param_grid,
    metric=OPTIMIZATION_METRIC,
    min_cluster_size=10
)
fig = viz.plot_hyperparameter_tuning(
    dbscan_tuning_results,
    algorithm='dbscan'
)
plt.show()
```

Figure: HP Tuning DBSCAN

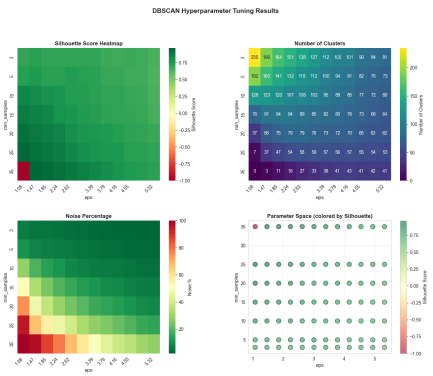


Figure: HP Tuning DBSCAN - plot

# Model Selection Based on Hyperparameter Tuning Results

Models and best hyperparameters comparison:

| Algorithm | Silhouette | Calinski-Harabasz | Davies-Bouldin | n_clusters | Noise % | Best Hyperparameters |
|---|---|---|---|---|---|---|
| **Hierarchical** | **0.9429** | 310.03 | **0.0382** | 2 | 0% | linkage=average, metric=euclidean |
| DBSCAN | 0.6829 | **3000.96** | 0.4708 | 10 | 13.3% | eps=1.4671, min_samples=20 |
| K-Means | 0.6581 | 547.22 | 1.5229 | 5 | 0% | n_clusters=5, n_init=10, max_iter=100 |

**Model Selection Rationale:**

- **Hierarchical**: Best overall with highest silhouette (0.94) and lowest Davies-Bouldin (0.04).
- **DBSCAN**: Highest Calinski-Harabasz (3000.96), useful for outlier detection.
- **K-Means**: Balanced baseline with moderate cluster count (k=5).

**Decision**: Primary focus on Hierarchical, with DBSCAN for outlier analysis.

# Best Hierarchical

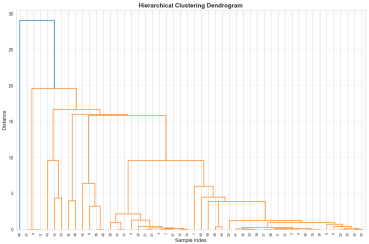


Figure: Dendogram


Figure: Segmentation

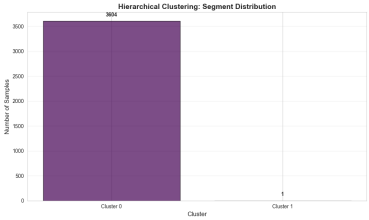


Figure: By cluster

# Best DBSCAN

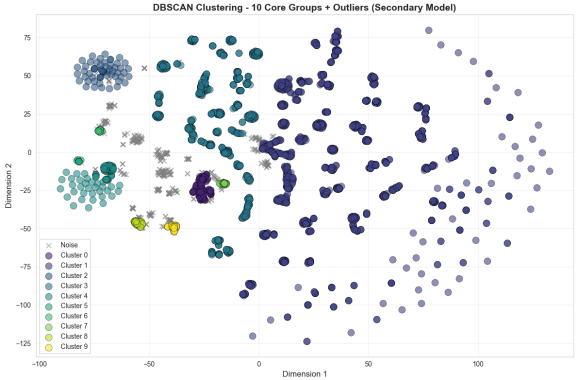


Figure: Segmentation

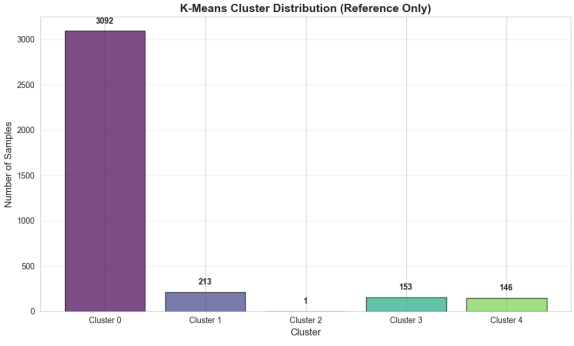


Figure: K-Means as baseline

# Clusters Interpretation I


Figure: Brands x Risk score

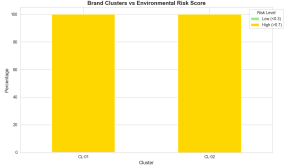


Figure: Brands x Risk bucket

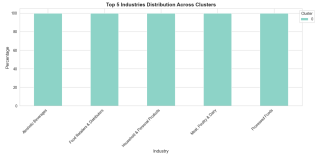


Figure: Industries Distribution

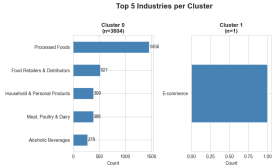


Figure: Top 5 Industries per Cluster


Figure: Top 5 Companies per Cluster

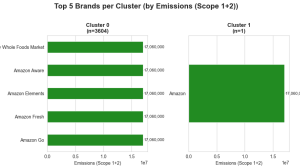


Figure: Top 5 Brands per Cluster - By scope

# Clustering Results: My Honest Assessment

| Algorithm | Clusters | Silhouette | Distribution | Key Observation |
|---|---|---|---|---|
| **K-Means** | 5 | 0.658 | 3,092 / 213 / 153 / 146 / 1 | One dominant cluster (86%) + one singleton |
| **Hierarchical** | 2 | 0.943 | 3,604 / 1 | Effectively a single cluster + 1 outlier |
| **DBSCAN** | 10 | 0.683 | Various (20-1,777) | 13.3% noise (481 brands) |

The results diverge from my initial expectations. Rather than discovering distinct, well-separated market segments, the algorithms suggest that **the brand space is largely continuous rather than discretely segmented**.

# Conclusion - Failure or Not Failure?

- **The Finding IS the Insight**: Brands don't segment into distinct groups; differentiation operates on continuous spectra, not discrete categories.
- **Rigorous Methodology**: 25 features, 255 HP combinations tested, PCA with interpretable latent dimensions.
- **Outlier Detection**: DBSCAN identified 481 outlier brands (13.3%) with higher revenue and emissions; worth individual investigation.
- **Future Work**: Clean dataset ready for Deep Learning analysis.

# Acknowledgement of Limitations

- **Feature Space**: The 25 features may not capture the dimensions that truly differentiate brands in meaningful ways
- **Data Homogeneity**: Brands within this ESG database proved to be inherently similar; The reason is that they were pre-selected for sustainability reporting purposes
- **Scale Imbalance**: My finding of one dominant cluster across all algorithms confirms at least one of the hypotheses: the data is either homogeneous or suboptimal

# Final Word

**The project is not a waste of effort.** While the clustering results fell short of discovering dramatically distinct brand segments, the work accomplished legitimate goals:

- Demonstrated mastery of unsupervised learning techniques and proper application
- Revealed an honest truth about the data
- Produced a clean, engineered dataset ready for advanced analysis
- Identified the class imbalance problem that GANs can directly address

**The journey continues in the Deep Learning project.**

# References I


1. **Jain, A. K. (2010).** *Data clustering: 50 years beyond K-means.* Pattern Recognition Letters, 31(8), 651-666.
2. **Hartigan, J. A., & Wong, M. A. (1979).** *Algorithm AS 136: A K-Means Clustering Algorithm.* Journal of the Royal Statistical Society. Series C, 28(1), 100-108.
3. **Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996).** *A density-based algorithm for discovering clusters in large spatial databases with noise.* KDD-96 Proceedings, 226-231.
4. **van der Maaten, L., & Hinton, G. (2008).** *Visualizing Data using t-SNE.* JMLR, 9, 2579-2605.
5. **Jolliffe, I. T., & Cadima, J. (2016).** *Principal component analysis: a review and recent developments.* Phil. Trans. R. Soc. A, 374(2065).
6. **Rousseeuw, P. J. (1987).** *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis.* J. Comput. Appl. Math., 20, 53-65.

# References II


7. **Pedregosa, F., et al. (2011).** *Scikit-learn: Machine Learning in Python.* JMLR, 12, 2825-2830.
8. **scikit-learn Documentation.** *Clustering.* https://scikit-learn.org/stable/modules/clustering.html
9. **scikit-learn Documentation.** *Decomposition (PCA).* https://scikit-learn.org/stable/modules/decomposition.html
10. **Caliński, T., & Harabasz, J. (1974).** *A dendrite method for cluster analysis.* Comm. in Statistics, 3(1), 1-27.
11. **Davies, D. L., & Bouldin, D. W. (1979).** *A Cluster Separation Measure.* IEEE Trans. PAMI, 1(2), 224-227.
12. **Google Developers.** *Clustering in Machine Learning.* https://developers.google.com/machine-learning/clustering
13. **Towards Data Science.** *The 5 Clustering Algorithms Data Scientists Need to Know.*

# References III


14. **StatQuest with Josh Starmer.** *K-means clustering.* YouTube.
15. **The GHG Shopper application.** https://ghgshopper.org

# Appendix - Project Structure

The implementation uses modular Python files:

- `clustering_models.py` - K-Means, Hierarchical, DBSCAN + HyperparameterTuner
- `dimensionality_reduction.py` - PCA and t-SNE with component analysis
- `visualization_utils.py` - Plotting and visualization functions

# Appendix - Feature Engineering and Scaling

```python
full_df['emissions_intensity'] = full_df['scope12_total'] / (
    full_df['revenues'] + 1)
full_df['esg_investment_ratio'] = full_df['
    r_and_d_spend_percent_revenue'] / (full_df['
    market_cap_billion_usd'] + 1)
current_year = 2025
full_df['brand_age'] = current_year - full_df['year_of_foundation
    ']
full_df['brand_age_log'] = np.log1p(full_df['brand_age'])
full_df['revenue_per_employee'] = (full_df['revenues'] / (full_df
    ['employees'] + 1)) / 1000
age_cols = ['age_prenatal', 'age_0_5', 'age_6_12', 'age_teens', '
    age_young_adults', 'age_seniors']
full_df['demographic_breadth'] = full_df[age_cols].sum(axis=1)
income_cols = ['income_low', 'income_middle', 'income_high', '
    income_premium']
full_df['income_diversity'] = full_df[income_cols].sum(axis=1)
lifestyle_cols = ['lifestyle_family', 'lifestyle_youth', '
    lifestyle_seniors',
                  'lifestyle_health_focused', '
                      lifestyle_convenience',
                  'lifestyle_tech_savvy', '
                      lifestyle_sustainability_conscious']
full_df['lifestyle_complexity'] = full_df[lifestyle_cols].sum(
    axis=1)
full_df['operational_complexity'] = (
    full_df['has_franchises'].astype(float) +
    full_df['owns_fleet'].astype(float) +
    full_df['has_drive_through'].astype(float) +
    full_df['online_sales'].astype(float) +
    full_df['product_portfolio_diversity']
)
full_df['market_positioning'] = (
    np.log1p(full_df['market_cap_billion_usd']) * 0.5 +
    full_df['customer_loyalty_index'] * 0.3 +
    full_df['branding_innovation_level'] * 0.2
)
env_risk_cols = ['deforestation_risk', 'chemical_pollution_risk',
     'fossil_fuel_reliance_encoded']
full_df['env_risk_concentration'] = full_df[env_risk_cols].mean(
    axis=1)

interaction_cols = ['emissions_intensity', 'esg_investment_ratio'
    , 'brand_age_log',
                    'revenue_per_employee', 'demographic_breadth',
                        'income_diversity',
                    'lifestyle_complexity', '
                        operational_complexity', '
                        market_positioning',
                    'env_risk_concentration']
print(full_df[interaction_cols].describe().round(3))

print(full_df.describe())
```

# Appendix - Interpreting Brand-Company Divergence Scores

These divergence scores measure the alignment (or misalignment) between a brand's positioning and its parent company's actual environmental performance. **Sustainability Divergence:**

- **Positive values ($> 0$)**: Brand positions strongly on sustainability while parent company has higher environmental risk $\rightarrow$ **High divergence** (brand is more environmentally conscious than company)
- **Negative values ($< 0$)**: Brand under-emphasizes sustainability despite parent company's better environmental performance $\rightarrow$ **Reverse divergence** (company is more environmentally conscious than brand positioning suggests)
- **Near zero ( $0$)**: Brand positioning aligns with company environmental performance $\rightarrow$ **Aligned**

**ESG-Premium Divergence:**

- **Positive values**: Premium brand from company with stronger ESG performance $\rightarrow$ **Aligned premium positioning**
- **Negative values**: Premium brand from company with weaker ESG performance $\rightarrow$ **Misaligned premium positioning**
- **Near zero**: Premium positioning matches ESG performance level $\rightarrow$ **Consistent**

**Key Insight:** These scores help identify patterns in how brands strategically position themselves relative to their parent companies, which is central to understanding brand-company environmental alignment.

# Visualizations

```
divergence_cols = ['brand_name', '
    sustainability_positioning', '
    company_env_risk', '
    sustainability_divergence']
top_divergence = full_df.nlargest
    (10, 'sustainability_divergence'
    )[divergence_cols]
print(top_divergence.to_string(index
    =False))

print("\n" + "="*60)
print("BRANDS WITH LOWEST
    SUSTAINABILITY DIVERGENCE")
print("(Parent company more
    environmentally conscious than
    brand positioning)")
print("="*60)
bottom_divergence = full_df.
    nsmallest(10, '
    sustainability_divergence')[
```

```
print("\n" + "="*60)
print("PREMIUM-ESG DIVERGENCE
    ANALYSIS")
print("="*60)
prem_div_cols = ['brand_name', '
    premium_positioning', '
    esg_premium_divergence']
print("\nHighest Premium-ESG
    Divergence (Premium brands from
    strong ESG companies):")
print(full_df.nlargest(10, '
    esg_premium_divergence')[
    prem_div_cols].to_string(index=
    False))

print("\nLowest Premium-ESG
    Divergence (Premium brands from
    weak ESG companies):")
print(full_df.nsmallest(10, '
    esg_premium_divergence')[
```

# Visualizations

```
fig, axes = plt.subplots(1, 2,
    figsize=(14, 5))

axes[0].hist(full_df['
    sustainability_divergence'],
    bins=50, edgecolor='black',
    alpha=0.7, color='green')
axes[0].axvline(x=0, color='red',
    linestyle='--', linewidth=2,
    label='Zero divergence')
axes[0].set_xlabel('Sustainability
    Divergence Score', fontsize=11)
axes[0].set_ylabel('Number of Brands
    ', fontsize=11)
axes[0].set_title('Sustainability
    Divergence Distribution\n(
    Positive = Potential
    Greenwashing)',
                  fontsize=12,
                      fontweight='
```

```
axes[1].hist(full_df['
    esg_premium_divergence'], bins
    =50, edgecolor='black', alpha
    =0.7, color='purple')
axes[1].axvline(x=0, color='red',
    linestyle='--', linewidth=2,
    label='Zero divergence')
axes[1].set_xlabel('ESG-Premium
    Divergence Score', fontsize=11)
axes[1].set_ylabel('Number of Brands
    ', fontsize=11)
axes[1].set_title('ESG-Premium
    Alignment Distribution\n(
    Negative = Premium Brand, Poor
    ESG)',
                  fontsize=12,
                      fontweight='
                      bold')
axes[1].legend()
axes[1].grid(True, alpha=0.3)
```

# Visualizations

```
n_components_to_interpret = min(10, X_pca.shape[1])
component_interpretations = {}
for i in range(n_components_to_interpret):
    pc_name = f'PC{i+1}'

    top_features_pc = top_features[
        (top_features['Component'] == pc_name) &
        (top_features['Rank'] <= 5)
    ].sort_values('Abs_Loading', ascending=False)

    print(f"\n{pc_name} (explains {variance_df.iloc[i]['Variance_Explained']:.1f}% variance):")
    print("-" * 80)
```

```
    for _, row in top_features_pc.iterrows():
        direction = 'positively' if row['Loading'] > 0 else 'negatively'
        print(f"  {row['Feature']:50s} {row['Loading']:+.3f} ({direction})")

component_names = {
    'PC1': 'To be determined by loadings',  # Placeholder
    'PC2': 'To be determined by loadings',
    'PC3': 'To be determined by loadings',
    'PC4': 'To be determined by loadings',
    'PC5': 'To be determined by loadings',
}
```

# Appendix - Latent Features Interpretation - Code

```
n_top_features = 20
top_feature_names = (top_features.
    groupby('Feature')['Abs_Loading'
    ].max().sort_values(ascending=
    False).head(n_top_features).
    index)

loading_matrix = pd.DataFrame(
    pca_model.components_[:
        n_components_to_interpret,
        :],
    columns=feature_columns,
    index=[f'PC{i+1}' for i in range
        (n_components_to_interpret)]
)[top_feature_names].T
```

```
fig, ax = plt.subplots(figsize=(14,
    10))
sns.heatmap(loading_matrix, cmap='
    RdBu_r', center=0, annot=True,
    fmt='.2f', cbar_kws={'label': '
    Loading'}, ax=ax)
ax.set_title(f'Feature Loadings
    Heatmap: Top {n_top_features}
    Features vs Principal Components
    ', fontweight='bold', fontsize
    =14, pad=20)
ax.set_xlabel('Principal Components'
    , fontweight='bold')
ax.set_ylabel('Features', fontweight
    ='bold')
plt.tight_layout()
plt.show()
```

# Visualizations

```
pca_scores_df = pd.DataFrame(
    X_pca[:, :
        n_components_to_interpret],
    columns=[f'PC{i+1}' for i in
        range(
        n_components_to_interpret)],
    index=full_df.index
)

pca_scores_df['brand_name'] =
    full_df['brand_name']
pca_scores_df['company_name'] =
    full_df['company_name']
pca_scores_df['industry_name'] =
    full_df['industry_name']
```

```
for pc in ['PC1', 'PC2', 'PC3']:
    print(f"\n{pc}:")
    print("\nHighest scoring brands:
        ")
    print(pca_scores_df.nlargest(5,
        pc)[['brand_name', '
        company_name', pc]])

    print("\nLowest scoring brands:"
        )
    print(pca_scores_df.nsmallest(5,
         pc)[['brand_name', '
        company_name', pc]])
    print("-" * 80)

full_df[[f'PC{i+1}' for i in range(
    n_components_to_interpret)]] =
    X_pca[:, :
    n_components_to_interpret]
```

# Visualizations

```
fig, axes = plt.subplots(1, 2,
    figsize=(16, 6))

scatter1 = axes[0].scatter(
    pca_scores_df['PC1'],
    pca_scores_df['PC2'], c=full_df[
    'environmental_risk_score'],
    cmap='RdYlGn_r', alpha=0.6, s
    =50, edgecolors='black',
    linewidth=0.5)
axes[0].set_xlabel('PC1', fontweight
    ='bold', fontsize=12)
axes[0].set_ylabel('PC2', fontweight
    ='bold', fontsize=12)
axes[0].set_title('Brands in Latent
    Space (PC1 vs PC2)\nColored by
    Environmental Risk', fontweight=
    'bold', fontsize=13)
axes[0].grid(True, alpha=0.3)
plt.colorbar(scatter1, ax=axes[0],
```

```
scatter2 = axes[1].scatter(
    pca_scores_df['PC1'],
    pca_scores_df['PC3'], c=full_df[
    'revenues'], cmap='viridis',
    alpha=0.6, s=50, edgecolors='
    black', linewidth=0.5, norm=plt.
    cm.colors.LogNorm())
axes[1].set_xlabel('PC1', fontweight
    ='bold', fontsize=12)
axes[1].set_ylabel('PC3', fontweight
    ='bold', fontsize=12)
axes[1].set_title('Brands in Latent
    Space (PC1 vs PC3)\nColored by
    Revenue',
                  fontweight='bold',
                      fontsize=13)
axes[1].grid(True, alpha=0.3)
plt.colorbar(scatter2, ax=axes[1],
    label='Revenue (log scale)')

plt.tight_layout()
```

# Appendix - Business Interpretation of Latent Features

Would this ever be used in the real business world, the Clusters should be named to represent what they mean. For the scope of this project I simply named them sequentially with placeholders, as I don't claim to be an ESG expert.
Using the following names for Principal Components:

- **PC1**: PC-1
- **PC2**: PC-2
- **PC3**: PC-2

Using the following names for Clusters:

- **CL_01**: CL_01
- **CL_02**: CL_02
- **CL_n**: CL_n

# t-SNE

```
# Apply t-SNE on PCA-reduced data (recommended workflow)
# This reduces computational cost and noise
X_tsne, tsne_model = reducer.apply_tsne(X_pca, n_components=2)
```

Figure: Applying t-SNE on PCA-reduced data

# Hyperparameter Tuning variables for ranges

```
# K-MEANS CONFIGURATION
KMEANS_MIN_CLUSTERS = 2
KMEANS_MAX_CLUSTERS = 21
KMEANS_N_INIT = [10, 20]
KMEANS_MAX_ITER = [100,300,500]
KMEANS_BALANCED_MIN = 5
KMEANS_BALANCED_MAX = 15
KMEANS_BALANCED_MIN_SILHOUETTE = 0.2
# HIERARCHICAL CLUSTERING CONFIGURATION
HIER_MIN_CLUSTERS = 2
HIER_MAX_CLUSTERS = 21
HIER_LINKAGE_METHODS = ['ward', 'complete', 'average']
HIER_METRIC = ['euclidean']
HIER_BALANCED_MIN = 2
HIER_BALANCED_MAX = 15
HIER_BALANCED_MIN_SILHOUETTE = 0.2
# DBSCAN CONFIGURATION
DBSCAN_EPS_MIN_PERCENTILE = 60
DBSCAN_EPS_MAX_PERCENTILE = 95
DBSCAN_EPS_N_VALUES = 12
DBSCAN_MIN_SAMPLES = [3, 5, 10, 15, 20, 25, 35]
DBSCAN_MIN_CLUSTERS = 5
```

# Hyperparameter Tuning - Continued

```
if 'kmeans' in tuner.best_params:
    kmeans_best = tuner.get_best_params('kmeans')
    print("\nK-Means:")
    print(f"  n_clusters: {int(kmeans_best['n_clusters'])}")
    print(f"  n_init: {int(kmeans_best['n_init'])}")
    print(f"  Silhouette Score: {kmeans_best['silhouette']:.4f}")

if 'hierarchical' in tuner.best_params:
    hier_best = tuner.get_best_params('hierarchical')
    print("\nHierarchical Clustering:")
    print(f"  n_clusters: {int(hier_best['n_clusters'])}")
    print(f"  linkage: {hier_best['linkage']}")
    print(f"  Silhouette Score: {hier_best['silhouette']:.4f}")

if 'dbscan' in tuner.best_params:
    dbscan_best = tuner.get_best_params('dbscan')
    print("\nDBSCAN:")
    print(f"  eps: {dbscan_best['eps']:.4f}")
    print(f"  min_samples: {int(dbscan_best['min_samples'])}")
    print(f"  n_clusters: {int(dbscan_best['n_clusters'])}")
    print(f"  noise_pct: {dbscan_best['noise_pct']:.1f}%")
    print(f"  Silhouette Score: {dbscan_best['silhouette']:.4f}")
```

# Hyperparameter Tuning - Continued

```
1
2 best_params = tuner.get_best_params(
      'kmeans')
3 n_clusters = int(best_params['
      n_clusters'])
4 print(f"Using tuned K-Means
      parameters: n_clusters={
      n_clusters}")
5 kmeans_model, kmeans_labels =
      clusterer.kmeans_clustering(
6     X_pca,
7     n_clusters=n_clusters,
8     find_optimal=False  # Use
          specified k from tuning
9 )
```

```
1 best_params = tuner.get_best_params(
      'hierarchical')
2 n_clusters = int(best_params['
      n_clusters'])
3 print(f"Using tuned Hierarchical
      parameters: n_clusters={
      n_clusters}, linkage={
      best_params['linkage']}")
4
5 # Update config for hierarchical
      clustering to use tuned linkage
6 clusterer.config['hierarchical']['
      linkage'] = best_params['linkage
      ']
7 clusterer.config['hierarchical']['
      n_clusters'] = n_clusters
8
9 hierarchical_model,
      hierarchical_labels = clusterer.
      hierarchical_clustering(
10    X_pca
```

# Hierarchical

```python
# Plot dendrogram
if X_pca.shape[0] <= 50:
    fig = viz.plot_dendrogram(X_pca, method='ward')
    plt.show()
else:
    #plotting a random subset
    subset_indices = np.random.choice(X_pca.shape[0], size=50, replace=False)
    subset = X_pca[subset_indices]
    fig = viz.plot_dendrogram(subset, method='ward')
    plt.show()
```

```python
# Visualize hierarchical clusters (PRIMARY MODEL)
# Focus on this visualization for business interpretation

fig = viz.plot_clusters_2d(
    X_tsne,
    hierarchical_labels,
    title="Hierarchical Clustering - 5 Brand Segments (Primary Model)",
    brand_names=full_df['brand_name'].tolist(),
    show_labels=False
)
plt.show()

# Plot cluster sizes
fig = viz.plot_cluster_sizes(
    hierarchical_labels,
    title="Hierarchical Clustering: Segment Distribution"
)
plt.show()

print(f"\n{'='*80}")
print("HIERARCHICAL CLUSTERING RESULTS (PRIMARY MODEL)")
print(f"{'='*80}")
print(f"Number of clusters: {len(set(hierarchical_labels))}")
print(f"{'='*80}")
```

# DBSCAN I

```python
# DBSCAN (density-based) with tuned hyperparameters
# Secondary model - excellent for outlier detection

if 'tuner' in locals() and 'dbscan' in tuner.best_params:
    # Use tuned parameters
    dbscan_params = tuner.best_params['dbscan']
    eps = dbscan_params['eps']
    min_samples = dbscan_params['min_samples']
else:
    # Default parameters
    eps = 2.5
    min_samples = 25

dbscan_labels = clusterer.fit_dbscan(X_pca, eps=eps, min_samples=int(min_samples))
```

```python
# Compare clustering algorithms

comparison_df = clusterer.compare_algorithms()
print("\nClustering Algorithm Comparison:")
print("=" * 80)
print(comparison_df.to_string(index=False))
print("=" * 80)
```

# Visualization

```python
# Visualize DBSCAN clusters
fig = viz.plot_clusters_2d(
    X_tsne,
    dbscan_labels,
    title="DBSCAN Clustering (t-SNE Visualization)",
    show_legend=True  # Hide legend if there are too many clusters
)
plt.show()

# Examine outliers (noise points)
outliers = full_df[full_df['dbscan_cluster'] == -1]
print(f"\nDBSCAN identified {len(outliers)} outlier brands:")
if len(outliers) > 0:
    display_cols = ['brand_name', 'company_name', 'industry_name']
    if 'initial_greenwashing_level' in outliers.columns:
        display_cols.append('initial_greenwashing_level')
    available_cols = [c for c in display_cols if c in outliers.columns]
    print(outliers[available_cols].head(10))
```

# Data Processing

- Successfully integrated **3,605 brands** across **243 unique companies** and **14 industries**
- Combined multiple data sources: industry ESG data, company emissions, brand demographics
- Engineered **environmental_risk_score** metric from multiple greenwashing and risk factors
- Handled missing values and normalized **25 features** across different scales

# Dimensionality Reduction

- Applied **PCA** to reduce noise and preserve 95% of variance (25 $\rightarrow$ 17 dimensions)
- Applied **t-SNE** for 2D visualization of high-dimensional brand space
- Identified key principal components driving brand differentiation:
- **PC1 (15.07%)**: Environmental Risk Dimension (deforestation, labor exploitation, greenwashing)
- **PC2 (12.21%)**: Scale & Sustainability Dimension (EV%, loyalty index, revenue, employees)
- **PC3 (10.71%)**: Corporate Size Dimension (revenue, employees, market cap)
- **PC4 (10.30%)**: Income Demographics Dimension (income targeting patterns)

# Clustering Analysis

- Implemented and compared **3 clustering algorithms** with systematic hyperparameter tuning
- Applied multi-metric evaluation (Silhouette, Calinski-Harabasz, Davies-Bouldin)
- Tested **114 K-Means**, **57 Hierarchical**, and **84 DBSCAN** parameter combinations

# Deliverables

- Exportable CSV with cluster assignments for all brands
- Comprehensive visualizations (PCA variance, t-SNE plots, cluster profiles)

# Future Enhancement: Deep Learning with GANs (Potentially Pretrained) I

This unsupervised learning project serves as a foundation for the upcoming **Deep Learning final project**, where the plan is to:
Use Clusters as Pseudo-Labels
The K-Means 5-cluster solution, despite its imbalance, provides initial class labels for supervised approaches. Address Class Imbalance with GANs
**Generative Adversarial Networks** can synthesize new samples for underrepresented clusters:

- The K-Means clusters 1-4 (213, 153, 146, 1 brands) are severely underrepresented vs Cluster 0 (3,092)
- GANs can generate realistic *synthetic* brand profiles to balance training data
- This will enable training to be more robust, and classifiers will work better for brand segment prediction

# Future Enhancement: Deep Learning with GANs (Potentially Pretrained) II

Improved Segmentation:
With balanced, augmented data, deep learning models may discover finer-grained patterns that traditional clustering missed. Validation Pipeline:
The augmented dataset can train and evaluate classification models, with held-out original data serving as ground truth. Another round of Unsupervised Learning:
At the end, the generated augmented dataset can be re-analyzed using this pipeline to evaluate the Cluster distribution.