```
#%%
# What I want to do

# Category                                  Points      Description

# Data Cleaning & Preparation               10          How well you handled missing data, transformations, and data quality

# Exploratory Analysis                      10          Depth of insights, variety of visuals, and use of statistics

# Modeling / Advanced Analysis              10          Modeling accuracy, creativity, and relevance

# Storytelling & Clarity                    10          Markdown explanations, narrative flow, and interpretation

# Creativity & Impact                       10          Unique insights, innovative metrics, and visual design quality
#%%
!pip install kagglehub
#%%
import os, glob, warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import silhouette_score
#%%
import kagglehub

# Download latest version
path = kagglehub.dataset_download("programmer3/sustainable-urban-planning-and-landscape-dataset")

print("Path to dataset files:", path)
#%%
import kagglehub
print(kagglehub.__version__)
#%%
# Install dependencies as needed:
# pip install kagglehub[pandas-datasets]
import kagglehub
from kagglehub import KaggleDatasetAdapter

# Set the path to the file you'd like to load
file_path = "urban_planning_dataset.csv"

# Load the latest version
df = kagglehub.load_dataset(
  KaggleDatasetAdapter.PANDAS,
  "programmer3/sustainable-urban-planning-and-landscape-dataset",
  file_path,
  # Provide any additional arguments like
  # sql_query or pandas_kwargs. See the
  # documenation for more information:
  # https://github.com/Kaggle/kagglehub/blob/main
```

```
/README.md#kaggledatasetadapterpandas
)

print("First 5 records:", df.head())
#%% md
Data Cleaning & Preparation
#%%
print(df.isnull().sum())
df = df.dropna()  # remove rows with missing values

duplicates = df[df.duplicated()]    # check Duplicates records
print(duplicates)

df = df.drop_duplicates()   # Remove Duplicates. and

#%%
# Initial Data Quality Check
df.info()
df.describe(include='all')
df.isnull().sum()
#%%
# Drop rows with too many missing values
df = df.dropna(thresh=df.shape[1] - 2)
#%%
# Fill Missing Values
# Numerical columns → Median
num_cols = df.select_dtypes(include=['int64','float64']).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

# Categorical columns → Mode
cat_cols = df.select_dtypes(include=['object']).columns
for col in cat_cols:
```

```python
    df[col] = df[col].fillna(df[col].mode()[0])

#%%
# Data Type Correction
# Convert date columns
date_cols = [col for col in df.columns if 'date' in col.lower()]
for col in date_cols:
    df[col] = pd.to_datetime(df[col], errors='coerce')

# Convert numeric strings to numbers
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = pd.to_numeric(df[col], errors='ignore')

#%%
# Detect and Handle Outliers (IQR Method)
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR
    df = df[(df[col] >= lower) & (df[col] <= upper)]

#%%
# Additional Data Quality Checks
print("Null values:\n", df.isnull().sum())
print("\nUnique values:\n", df.nunique())
print("\nDuplicate rows:", df.duplicated().sum())

#%%
# Save Final Cleaned Dataset
df.to_csv("cleaned_sustainable_dataset.csv",
```

```
index=False)
#%%
# Basic Data Understanding
print("Shape:", df.shape)
print("\nData Types:\n", df.dtypes)
print("\nMissing Values:\n", df.isnull().sum())
print("\nStatistical Summary:\n")
df.describe(include="all")

#%%
DATA_PATH = "urban_planning_dataset.csv"
OUTDIR = "/mnt/data/analysis_outputs"
os.makedirs(OUTDIR, exist_ok=True)

#%%
# Basic cleaning (repeat to ensure fresh state)
df_clean = df.copy()
high_missing = df_clean.columns[df_clean.isnull().mean() > 0.8].tolist()
df_clean.drop(columns=high_missing, inplace=True)

num_cols = df_clean.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = df_clean.select_dtypes(include=['object','category']).columns.tolist()

num_imp = SimpleImputer(strategy='median')
if num_cols:
    df_clean[num_cols] = num_imp.fit_transform(df_clean[num_cols])
if cat_cols:
    from sklearn.impute import SimpleImputer as SI
    cat_imp = SI(strategy='most_frequent')
    df_clean[cat_cols] = cat_imp.fit_transform(df_clean[cat_cols])
#%%
```

```python
# Cap outliers
def cap_outliers(series):
    q1 = series.quantile(0.25); q3 = series.quantile(0.75); iqr = q3-q1
    lower = q1 - 1.5*iqr; upper = q3 + 1.5*iqr
    return series.clip(lower, upper)
for c in num_cols:
    df_clean[c] = cap_outliers(df_clean[c])
#%%
# Feature engineering
if 'green_cover_percentage' in df_clean.columns:
    df_clean['urban_green_index'] = df_clean['green_cover_percentage'] / 100.0
if 'building_density' in df_clean.columns and 'green_cover_percentage' in df_clean.columns:
    df_clean['heat_stress_proxy'] = df_clean['building_density'] - df_clean['urban_green_index']
#%%
# EDA: histograms (selected)
hist_cols = ['building_density','green_cover_percentage','air_quality_index','carbon_footprint','urban_sustainability_score']
hist_cols = [c for c in hist_cols if c in df_clean.columns]
for c in hist_cols:
    plt.figure(figsize=(6,4))
    plt.hist(df_clean[c].dropna(), bins=30)
    plt.title(f"Histogram: {c}")
    plt.xlabel(c); plt.ylabel("Count")
    plt.tight_layout()
    plt.savefig(os.path.join(OUTDIR, f"hist_{c}.png"), dpi=150)
    plt.show()

#%%
# Correlation matrix
num_cols = df_clean.select_dtypes(include=[np.
```

```python
number]).columns.tolist()
corr = df_clean[num_cols].corr()
plt.figure(figsize=(10,8))
plt.imshow(corr, interpolation='nearest', aspect='auto')
plt.colorbar()
plt.xticks(range(len(num_cols)), num_cols, rotation=90)
plt.yticks(range(len(num_cols)), num_cols)
plt.title("Correlation matrix (numeric)")
plt.tight_layout()
corr_path = os.path.join(OUTDIR, "correlation_matrix.png")
plt.savefig(corr_path, dpi=150)
plt.show()
#%%
# PCA
features_for_pca = [c for c in num_cols if c != 'urban_sustainability_score']
X = df_clean[features_for_pca].fillna(0).values
scaler = StandardScaler(); Xs = scaler.fit_transform(X)
pca = PCA(n_components=min(6, Xs.shape[1])); Xp = pca.fit_transform(Xs)
plt.figure(figsize=(6,4))
plt.plot(np.cumsum(pca.explained_variance_ratio_), marker='o')
plt.xlabel("Number of components"); plt.ylabel("Cumulative explained variance")
plt.title("PCA cumulative explained variance")
plt.tight_layout()
pca_path = os.path.join(OUTDIR, "pca_cumulative.png")
plt.savefig(pca_path, dpi=150)
plt.show()
#%%
# KMeans clustering on a sample (to keep quick)
sample_idx = np.random.choice(range(Xp.shape[0
```

```
]), size=min(1000, Xp.shape[0]), replace=False)
Xp_sample = Xp[sample_idx, :3]

sil_scores = {}
for k in range(2,6):
    km = KMeans(n_clusters=k, random_state=42, n_init=5, max_iter=200)
    labs = km.fit_predict(Xp_sample)
    sil = silhouette_score(Xp_sample, labs)
    sil_scores[k] = sil

best_k = max(sil_scores, key=sil_scores.get)
# Fit on full (but limited to first 2000 rows if large)
fit_idx = np.arange(Xp.shape[0])[:min(2000, Xp.shape[0])]
km = KMeans(n_clusters=best_k, random_state=42, n_init=5, max_iter=200)
clusters_full = km.fit_predict(Xp[fit_idx, :3])
# assign cluster for those rows, default -1 for others
cluster_col = np.full(Xp.shape[0], -1, dtype=int)
cluster_col[fit_idx] = clusters_full
df_clean['cluster'] = cluster_col

plt.figure(figsize=(8,6))
plt.scatter(Xp[fit_idx,0], Xp[fit_idx,1], c=clusters_full, s=10)
plt.xlabel("PCA 1"); plt.ylabel("PCA 2")
plt.title(f"KMeans clusters (k={best_k}) on PCA components (sample)")
plt.tight_layout()
cluster_path = os.path.join(OUTDIR, "clusters_pca_sample.png")
plt.savefig(cluster_path, dpi=150)
plt.show()
#%%
```

```python
# Modeling if target exists
results = {}
if 'urban_sustainability_score' in df_clean.columns:
    target = 'urban_sustainability_score'
    features = [c for c in features_for_pca if c != target and c != 'cluster']
    X = df_clean[features].values
    y = df_clean[target].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestRegressor(n_estimators=150, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = mean_squared_error(y_test, y_pred, ); r2 = r2_score(y_test, y_pred)
    results['rmse'] = float(rmse); results['r2'] = float(r2)
    fi = pd.Series(model.feature_importances_, index=features).sort_values(ascending=False)
    plt.figure(figsize=(8,5))
    fi.head(20).plot(kind='bar')
    plt.title("Top feature importances")
    plt.tight_layout()
    fi_path = os.path.join(OUTDIR, "feature_importances.png")
    plt.savefig(fi_path, dpi=150)
    plt.show()

```