Harare Institute of Technology

success through innovation

Innovation and Technopreneurial University

PREDICTING URBAN GROWTH PATTERNS IN HARARE USING MACHINE LEARNING

About this model and dataset

About and links

↳ 1 cell hidden

1. Necessary libraries and dependables

Imports

↳ 2 cells hidden

2. Definitions and variables

Google Maps API keys and focus area coordinates

[ ] ↳ 4 cells hidden

3. Download dataset and define path variable

Dataset

```
# Download dataset to be used and keep the path variable
# Dataset source : Kaggle (see 'About' section)
path = kagglehub.dataset_download("balraj98/deepglobe-land-cover-classification-dataset")

# Show path variable to the dataset
```

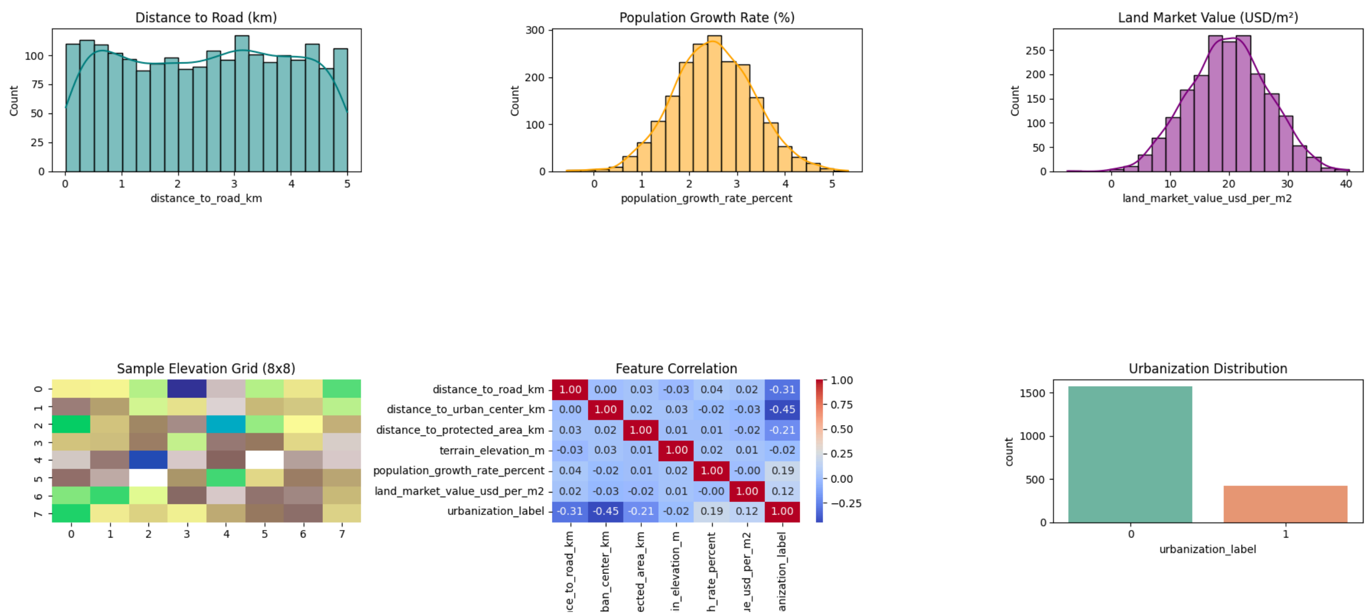
```
print("Dataset downloaded to:", path)
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/balraj98/deepglobe-land-cover-classification-dataset?dataset_version_number=2...
100%|██████████| 2.74G/2.74G [00:37<00:00, 78.0MB/s]Extracting files...

Dataset downloaded to: /root/.cache/kagglehub/datasets/balraj98/deepglobe-land-cover-classification-dataset/versions/2
```

```
warnings.filterwarnings('ignore')
urban_growth_features_df = pd.DataFrame({
    'distance_to_road_km': np.random.uniform(0, 5, num_samples),
    'distance_to_urban_center_km': np.random.uniform(0, 15, num_samples),
    'distance_to_protected_area_km': np.random.uniform(0, 10, num_samples),
    'terrain_elevation_m': np.random.normal(1350, 40, num_samples),
    'population_growth_rate_percent': np.random.normal(2.5, 0.8, num_samples),
    'land_market_value_usd_per_m2': np.random.normal(20, 7, num_samples),
})

# Create spatial elevation grids (8x8) for CNN
spatial_elevation_grids = np.zeros((num_samples, 8, 8, 1))
for idx in range(num_samples):
    base_elevation = urban_growth_features_df['terrain_elevation_m'].iloc[idx]
    spatial_elevation_grids[idx, :, :, 0] = base_elevation + np.random.normal(0, 5, (8, 8))

# Compute logistic regression logit for urbanization target variable
logit_scores = (
    -0.6 * urban_growth_features_df['distance_to_road_km'] +
    -0.3 * urban_growth_features_df['distance_to_urban_center_km'] +
    -0.2 * urban_growth_features_df['distance_to_protected_area_km'] +
    0.8 * urban_growth_features_df['population_growth_rate_percent'] +
    0.05 * urban_growth_features_df['land_market_value_usd_per_m2'] +
    np.random.normal(0, 1, num_samples)
)

# Convert logits to probabilities and generate binary urbanization labels
urbanization_probabilities = 1 / (1 + np.exp(-logit_scores))
urban_growth_features_df['urbanization_label'] = (urbanization_probabilities > 0.5).astype(int)
target_labels = urban_growth_features_df['urbanization_label'].values
```

```
# Exploratory Data Analysis (EDA) Visualizations
plt.figure(figsize=(18, 12))

plt.subplot(3, 3, 1)
sns.histplot(urban_growth_features_df['distance_to_road_km'], bins=20, kde=True, color='teal')
plt.title('Distance to Road (km)')

plt.subplot(3, 3, 2)
sns.histplot(urban_growth_features_df['population_growth_rate_percent'], bins=20, kde=True, color='orange')
```

```
plt.title('Population Growth Rate (%)')

plt.subplot(3, 3, 3)
sns.histplot(urban_growth_features_df['land_market_value_usd_per_m2'], bins=20, kde=True, color='purple')
plt.title('Land Market Value (USD/m²)')

plt.subplot(3, 3, 4)
sns.heatmap(spatial_elevation_grids[0, :, :, 0], cmap='terrain', cbar=False)
plt.title('Sample Elevation Grid (8x8)')

plt.subplot(3, 3, 5)
correlation_matrix = urban_growth_features_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Feature Correlation')

plt.subplot(3, 3, 6)
sns.countplot(x='urbanization_label', data=urban_growth_features_df, palette='Set2')
plt.title('Urbanization Distribution')

plt.subplot(3, 3, 7)
sns.scatterplot(
    x='distance_to_road_km',
    y='land_market_value_usd_per_m2',
    hue='urbanization_label',
    data=urban_growth_features_df,
    palette='Set1',
    alpha=0.6
)
plt.title('Distance to Road vs Land Market Value')

plt.subplot(3, 3, 8)
sns.boxplot(x='urbanization_label', y='population_growth_rate_percent', data=urban_growth_features_df, palette='Set3')
plt.title('Population Growth Rate by Urbanization')

plt.subplot(3, 3, 9)
sns.scatterplot(
    x='terrain_elevation_m',
    y='distance_to_protected_area_km',
    hue='urbanization_label',
    data=urban_growth_features_df,
    palette='viridis',
    alpha=0.7
)
plt.title('Elevation vs Distance to Protected Area')

plt.tight_layout()
plt.show()

print("✅ EDA visualizations completed\n")
```



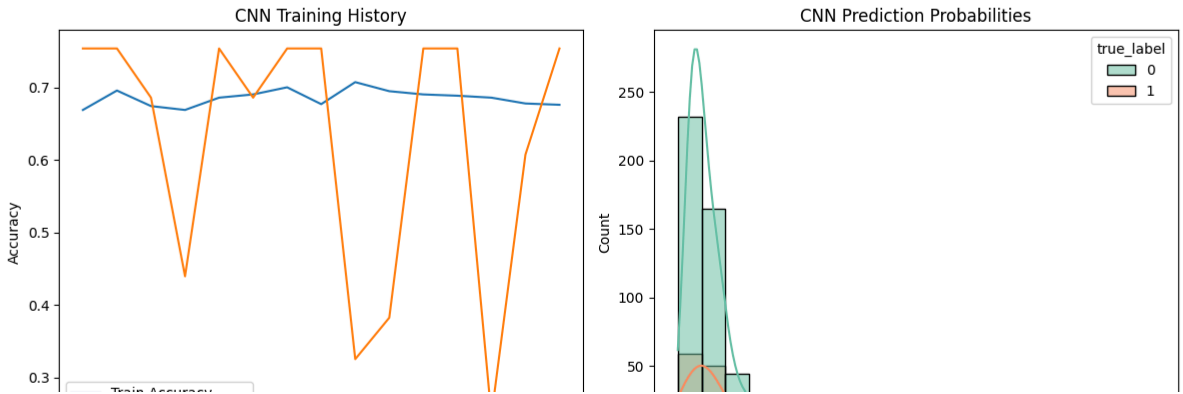
4. Model training

Train the model using specified algorithms

```
# Prepare feature matrices and target vector for model training and evaluation
X_tabular = urban_growth_features_df.drop('urbanization_label', axis=1).values  # Tabular features for GBRT and CA models
X_spatial = spatial_elevation_grids  # Spatial grid data for CNN model

# Split data into training and testing sets with stratification on target labels
X_tab_train, X_tab_test, X_spatial_train, X_spatial_test, y_train, y_test = train_test_split(
    X_tabular, X_spatial, target_labels, test_size=0.3, random_state=42, stratify=target_labels
)
```

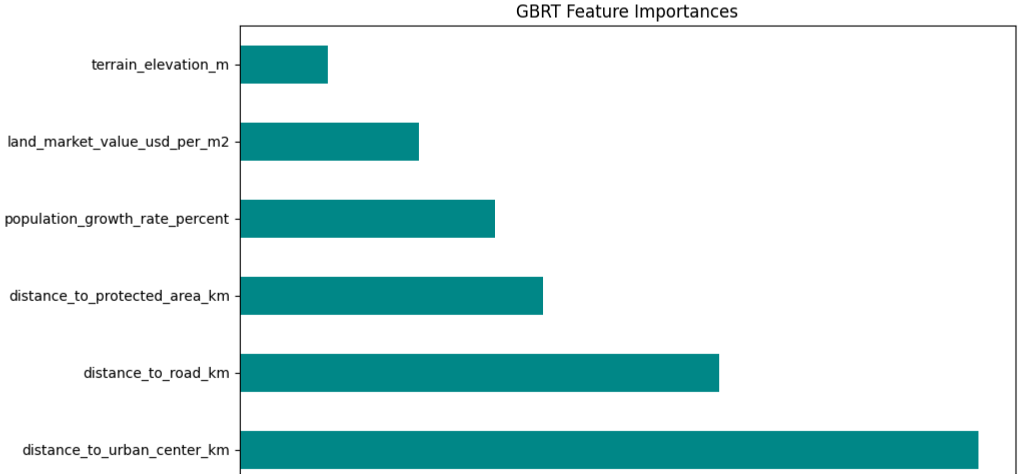
```
# -------------------------
# 4.1 Convolutional Neural Network (CNN) Model Definition and Training
# -------------------------
cnn_model = Sequential([
    InputLayer(input_shape=(8, 8, 1)),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print("Training CNN model...")

cnn_history = cnn_model.fit(
    X_spatial_train, y_train,
    epochs=15,
    batch_size=32,
    validation_split=0.2,
    verbose=0
)

# Plot CNN training and validation accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(cnn_history.history['accuracy'], label='Train Accuracy')
plt.plot(cnn_history.history['val_accuracy'], label='Validation Accuracy')
plt.title('CNN Training History')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# CNN model predictions and probabilities on test set
cnn_predictions = (cnn_model.predict(X_spatial_test) > 0.5).astype(int).flatten()
cnn_probabilities = cnn_model.predict(X_spatial_test).flatten()

# Plot histogram of CNN prediction probabilities by true label
plt.subplot(1, 2, 2)
cnn_probs_df = pd.DataFrame({
    "probability": cnn_probabilities,
    "true_label": y_test
})
sns.histplot(data=cnn_probs_df, x="probability", bins=20, kde=True, hue='true_label', palette='Set2')
plt.title('CNN Prediction Probabilities')
plt.tight_layout()
plt.show()
print("✅ CNN training complete\n")
```

```
Training CNN model...
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 30ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```




```
# ------------------------
# 4.2 Gradient Boosting Regression Trees (GBRT) Model Training
# ------------------------
gbrt_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=5)
print("Training GBRT model...")
gbrt_model.fit(X_tab_train, y_train)

# Visualize GBRT feature importances
plt.figure(figsize=(10, 6))
feature_names = urban_growth_features_df.drop('urbanization_label', axis=1).columns
feature_importances = pd.Series(gbrt_model.feature_importances_, index=feature_names)
feature_importances.nlargest(len(feature_names)).plot(kind='barh', color='darkcyan')
plt.title('GBRT Feature Importances')
plt.show()

# GBRT predictions and probabilities on test set
gbrt_predictions = gbrt_model.predict(X_tab_test)
gbrt_probabilities = gbrt_model.predict_proba(X_tab_test)[:, 1]
print("✅ GBRT training complete\n")
```

Training GBRT model...



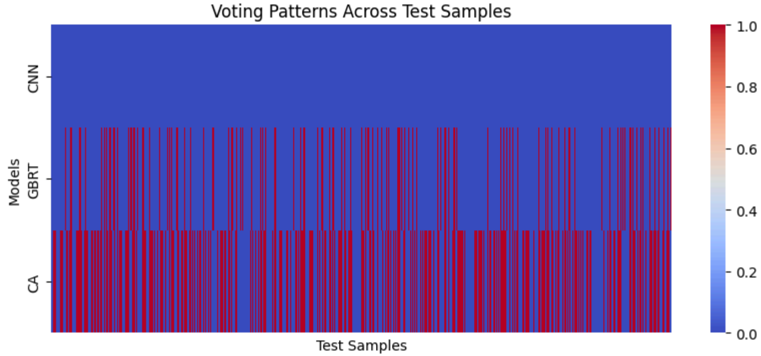

```
# ------------------------
# 4.3 Simple Cellular Automata (CA) Model Definition and Prediction
# ------------------------
class SimpleCellularAutomata:
    def predict(self, X):
        # Rule-based prediction: urbanized if at least two of three conditions met
        condition_road = (X[:, 0] < 2.5).astype(int)
        condition_population = (X[:, 4] > 2.5).astype(int)
        condition_land_value = (X[:, 5] > 20).astype(int)
        return (condition_road + condition_population + condition_land_value) >= 2

ca_model = SimpleCellularAutomata()
ca_predictions = ca_model.predict(X_tab_test)
```

```
# ------------------------
# 4.4 Voting Ensemble Model Combining CNN, GBRT, and CA Predictions
# ------------------------
class VotingEnsembleModel:
    def __init__(self, cnn, gbrt, ca):
        self.cnn_model = cnn
        self.gbrt_model = gbrt
```

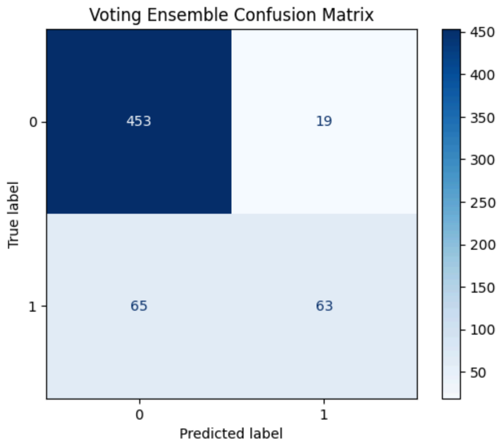
```
        self.ca_model = ca

    def predict(self, X_tab, X_spatial):
        cnn_preds = (self.cnn_model.predict(X_spatial) > 0.5).astype(int).flatten()
        gbrt_preds = self.gbrt_model.predict(X_tab)
        ca_preds = self.ca_model.predict(X_tab)
        combined_votes = np.stack([cnn_preds, gbrt_preds, ca_preds], axis=1)
        majority_vote = (combined_votes.sum(axis=1) >= 2).astype(int)
        return majority_vote, combined_votes

ensemble_model = VotingEnsembleModel(cnn_model, gbrt_model, ca_model)
ensemble_predictions, ensemble_votes = ensemble_model.predict(X_tab_test, X_spatial_test)


print("Voting Ensemble Model Combining CNN, GBRT, and CA Predictions trained successfully ! ")
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
Voting Ensemble Model Combining CNN, GBRT, and CA Predictions trained successfully !
```

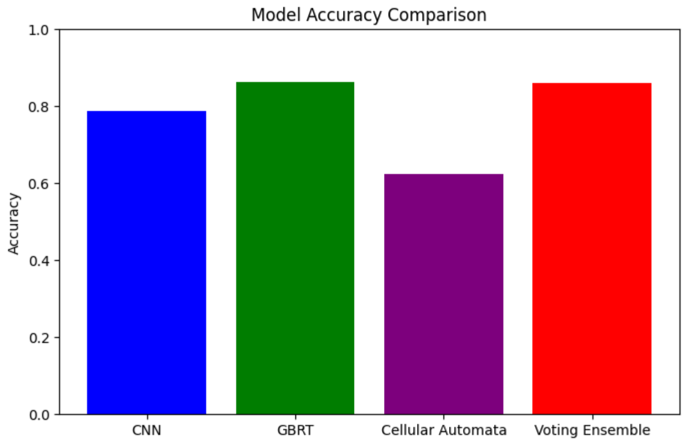
```
# -------------------------
# 4.5 Ensemble Model Visualization
# -------------------------

# Visualize voting patterns (e.g., heatmap of votes per sample)

votes = np.vstack([cnn_predictions, gbrt_predictions, ca_predictions])
plt.figure(figsize=(10, 4))
sns.heatmap(votes, cmap='coolwarm', cbar=True, xticklabels=False, yticklabels=['CNN', 'GBRT', 'CA'])
plt.title('Voting Patterns Across Test Samples')
plt.xlabel('Test Samples')
plt.ylabel('Models')
plt.show()
```



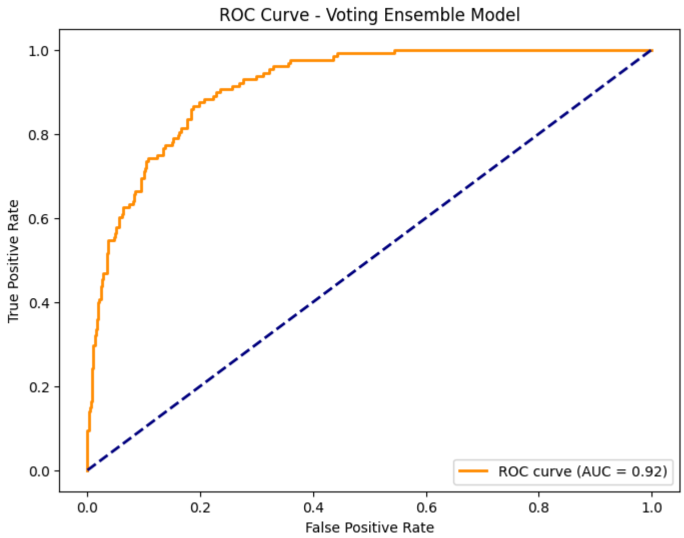

```
# Confusion matrix for ensemble predictions
ensemble_cm = confusion_matrix(y_test, ensemble_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=ensemble_cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.title('Voting Ensemble Confusion Matrix')
plt.show()
```




```
# Accuracy bar plot comparing base models and ensemble
model_accuracies = {
    'CNN': accuracy_score(y_test, cnn_predictions),
    'GBRT': accuracy_score(y_test, gbrt_predictions),
    'Cellular Automata': accuracy_score(y_test, ca_predictions),
    'Voting Ensemble': accuracy_score(y_test, ensemble_predictions)
}

plt.figure(figsize=(8, 5))
plt.bar(model_accuracies.keys(), model_accuracies.values(), color=['blue', 'green', 'purple', 'red'])
plt.ylabel('Accuracy')
plt.title('Model Accuracy Comparison')
plt.ylim(0, 1)
plt.show()
```




```
# Average the predicted probabilities from all models to get ensemble probabilities
ensemble_probabilities = (cnn_probabilities + gbrt_probabilities) / 2

# Compute ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, ensemble_probabilities)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Voting Ensemble Model')
plt.legend(loc='lower right')
plt.show()
```

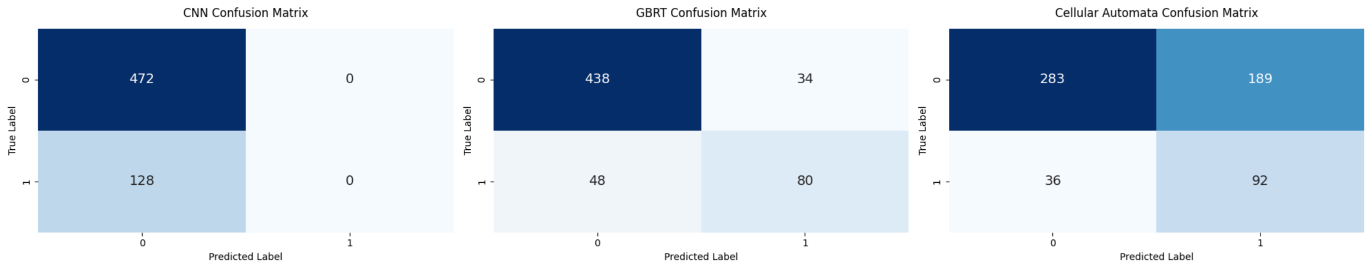


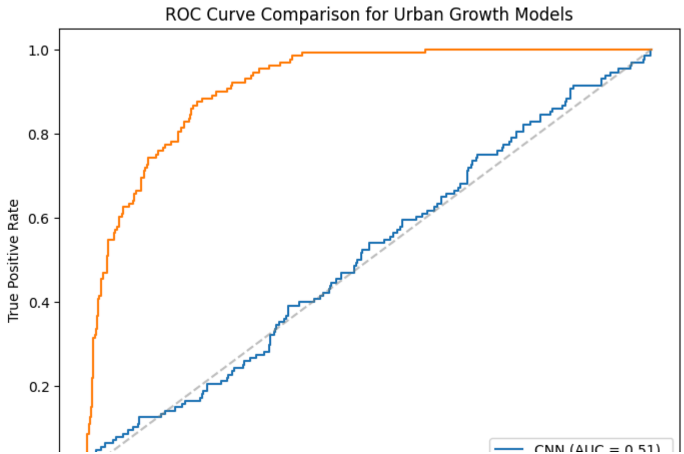
5. Model Evaluation

Evaluate the trained model

```
# Confusion matrices for CNN, GBRT, and Cellular Automata models
plt.figure(figsize=(20, 4))
for idx, (model_name, predictions) in enumerate(zip(
    ['CNN', 'GBRT', 'Cellular Automata'],
    [cnn_predictions, gbrt_predictions, ca_predictions]
)):
    plt.subplot(1, 3, idx + 1)
    sns.heatmap(
        confusion_matrix(y_test, predictions),
        annot=True,
        fmt='d',
        cmap='Blues',
        cbar=False,
```

```
        annot_kws={'size': 14}
    )
    plt.title(f'{model_name} Confusion Matrix', pad=12)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
plt.tight_layout()
plt.show()

# ROC curves comparing CNN and GBRT models
plt.figure(figsize=(8, 6))
for model_name, probabilities in zip(['CNN', 'GBRT'], [cnn_probabilities, gbrt_probabilities]):
    fpr, tpr, _ = roc_curve(y_test, probabilities)
    roc_auc_score = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc_score:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='grey', alpha=0.5)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison for Urban Growth Models')
plt.legend(loc='lower right')
plt.show()
```





Classification Reports

```
# Classification reports
print("\n📊 Classification Reports:")
for name, pred in zip(['CNN', 'GBRT', 'CA', 'Ensemble'],
                      [cnn_predictions, gbrt_predictions, ca_predictions, ensemble_predictions]):
    print(f"\n🔍 {name} Report:")
    print(classification_report(y_test, pred, target_names=['No Growth', 'Urban Growth']))  # Detailed performance metrics
```

📊 Classification Reports:

🔍 CNN Report:

	precision	recall	f1-score	support
No Growth	0.79	1.00	0.88	472
Urban Growth	0.00	0.00	0.00	128
accuracy			0.79	600
macro avg	0.39	0.50	0.44	600
weighted avg	0.62	0.79	0.69	600

🔍 GBRT Report:

	precision	recall	f1-score	support
No Growth	0.90	0.93	0.91	472
Urban Growth	0.70	0.62	0.66	128
accuracy			0.86	600
macro avg	0.80	0.78	0.79	600
weighted avg	0.86	0.86	0.86	600

🔍 CA Report:

	precision	recall	f1-score	support
No Growth	0.89	0.60	0.72	472
Urban Growth	0.33	0.72	0.45	128
accuracy			0.62	600
macro avg	0.61	0.66	0.58	600
weighted avg	0.77	0.62	0.66	600

🔍 Ensemble Report:

	precision	recall	f1-score	support
No Growth	0.87	0.96	0.92	472
Urban Growth	0.77	0.49	0.60	128
accuracy			0.86	600
macro avg	0.82	0.73	0.76	600
weighted avg	0.85	0.86	0.85	600

5. Model testing

Urban growth prediction model

```python
# --- CONFIGURATION ---
CENTER_LATITUDE = -18.0174072  # Chitungwiza Hospital Latitude
CENTER_LONGITUDE = 31.0603241  # Chitungwiza Hospital Longitude
ANALYSIS_RADIUS_KM = 1
YEARS_TO_ANALYZE = 10
CURRENT_YEAR = datetime.now().year

# --- DATA FETCHING UTILITIES ---

def get_distance_to_nearest_road(lat: float, lon: float) -> float | None:
    """
    Query OpenStreetMap Overpass API to find the nearest highway within 5km radius and compute distance in km.
    Returns None if no road found.
    """
    overpass_url = "http://overpass-api.de/api/interpreter"
    query = f"""
    [out:json];
    (
      way(around:5000,{lat},{lon})["highway"];
    );
    out geom 1;
    """
    try:
        response = requests.post(overpass_url, data={'data': query}, timeout=10)
        response.raise_for_status()
        data = response.json()
    except (requests.RequestException, ValueError):
        return None

    point = Point(lon, lat)
    min_dist = float('inf')
    for element in data.get('elements', []):
        if 'geometry' in element:
            coords = [(pt['lon'], pt['lat']) for pt in element['geometry']]
            line = LineString(coords)
            # Approximate degrees to meters conversion (1 degree ~ 111 km)
            dist_meters = point.distance(line) * 111000
            if dist_meters < min_dist:
                min_dist = dist_meters

    if min_dist == float('inf'):
        return None
    return min_dist / 1000  # Convert meters to kilometers

def get_distance_to_protected_area(lat: float, lon: float) -> float:
    return 8.0  # Fallback value in km

def get_terrain_elevation(lat: float, lon: float) -> float | None:
    """
    Query Open Elevation API to get terrain elevation in meters.
```

```
    Returns None if request fails or data unavailable.
    """
    url = f"https://api.open-elevation.com/api/v1/lookup?locations={lat},{lon}"
    try:
        r = requests.get(url, timeout=10)
        r.raise_for_status()
        results = r.json().get('results', [])
        if results:
            return results[0].get('elevation')
    except (requests.RequestException, ValueError):
        return None
    return None

def fetch_population_for_year(lat: float, lon: float, year: int) -> int:
    base_population = 350000  # Base population fall back
    growth_rate = 0.028       # 2.8% annual fall back
    years_ago = CURRENT_YEAR - year
    population = int(base_population * ((1 + growth_rate) ** (-years_ago)))
    return population

def get_population_growth_rate(lat: float, lon: float, year: int) -> float:
    """
    Falls back to 2.8% if data unavailable or invalid.
    """
    pop_now = fetch_population_for_year(lat, lon, year)
    pop_prev = fetch_population_for_year(lat, lon, year - 1)
    if pop_prev > 0:
        return 100 * (pop_now - pop_prev) / pop_prev
    return 2.8

def get_land_market_value(lat: float, lon: float) -> float:
    """
    """
    return 25.0  # Fallback average value

def extract_features(latitude: float, longitude: float, analysis_radius_km: float, year: int) -> np.ndarray:
    """
    Aggregate spatial and socioeconomic features for urban growth prediction.
    """
    dist_to_road = get_distance_to_nearest_road(latitude, longitude)
    if dist_to_road is None:
        dist_to_road = 2.5  # Fallback average in km

    dist_to_center = 0.0  # Center point distance

    dist_to_protected = get_distance_to_protected_area(latitude, longitude)

    elevation = get_terrain_elevation(latitude, longitude)
    if elevation is None:
        elevation = 1350  # Fallback average elevation in meters
```

```python
    population_growth = get_population_growth_rate(latitude, longitude, year)

    land_value = get_land_market_value(latitude, longitude)

    population = fetch_population_for_year(latitude, longitude, year)

    return np.array([
        dist_to_road,
        dist_to_center,
        dist_to_protected,
        elevation,
        population_growth,
        land_value,
        population
    ])

def ensemble_urban_growth_predictor(features: np.ndarray) -> tuple[float, float]:
    """
    Simple ensemble predictor returning urban growth likelihood and direction in radians.
    """
    base_likelihood = (0.5 + 0.1 * (5 - features[0]) + 0.1 * features[4] / 5) / 1.5
    clipped_likelihood = np.clip(base_likelihood, 0, 1)
    possible_directions_deg = [45, 90, 135, 180, 225]
    selected_direction_deg = random.choice(possible_directions_deg)
    selected_direction_rad = np.deg2rad(selected_direction_deg)
    return clipped_likelihood, selected_direction_rad

# --- MAIN ANALYSIS LOOP ---
results = []
for year in range(CURRENT_YEAR - YEARS_TO_ANALYZE + 1, CURRENT_YEAR + 1):
    features = extract_features(CENTER_LATITUDE, CENTER_LONGITUDE, ANALYSIS_RADIUS_KM, year)
    growth_score, growth_direction = ensemble_urban_growth_predictor(features)
    population = features[-1]
    results.append({
        "year": year,
        "population": population,
        "growth_score": growth_score,
        "growth_direction_rad": growth_direction
    })

# --- OUTPUT SUMMARY ---
for res in results:
    print(
        f"Year: {res['year']}, Population: {res['population']}, "
        f"Predicted Urban Growth Likelihood: {res['growth_score']:.2f}, "
        f"Direction (rad): {res['growth_direction_rad']:.2f}"
    )
```

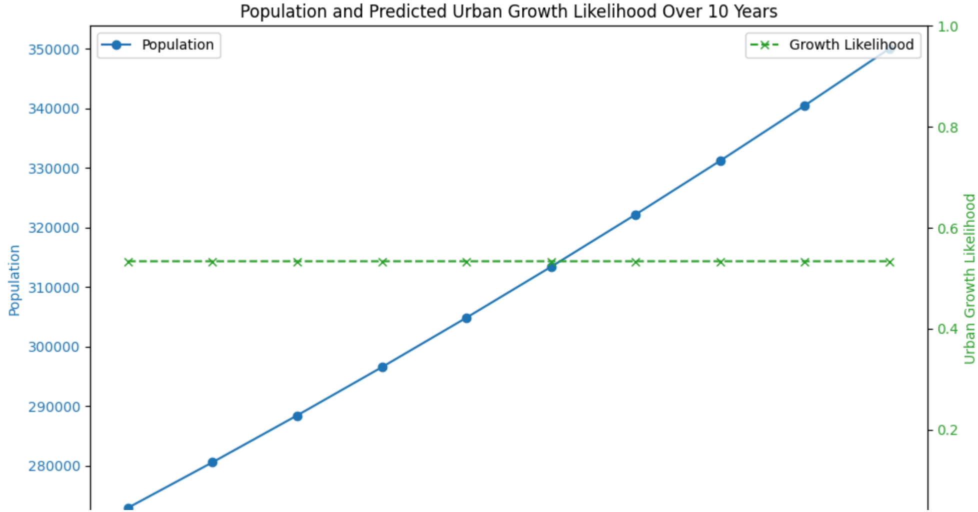
```
Year: 2016, Population: 272979.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.14
Year: 2017, Population: 280622.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 0.79
Year: 2018, Population: 288480.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.93
Year: 2019, Population: 296557.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.93
Year: 2020, Population: 304861.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 1.57
Year: 2021, Population: 313397.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.14
Year: 2022, Population: 322172.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.14
Year: 2023, Population: 331193.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.93
Year: 2024, Population: 340466.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 0.79
Year: 2025, Population: 350000.0, Predicted Urban Growth Likelihood: 0.53, Direction (rad): 3.14
```

```
# --- Helper functions for degree-km conversions accounting for latitude ---
def km_to_deg_lat(km):
    """Convert kilometers to degrees latitude approximately."""
    return km / 111.32

def km_to_deg_lon(km, latitude):
    """Convert kilometers to degrees longitude at a given latitude."""
    return km / (111.32 * np.cos(np.radians(latitude)))

# --- Overpass QL query to get admin boundaries for the 3 districts ---
overpass_query = """
[out:json][timeout:25];
area["name"="Zimbabwe"]->.searchArea;
(
  relation["boundary"="administrative"]["admin_level"="6"]["name"="Harare"](area.searchArea);
  relation["boundary"="administrative"]["admin_level"="6"]["name"="Chitungwiza"](area.searchArea);
  relation["boundary"="administrative"]["admin_level"="6"]["name"="Epworth"](area.searchArea);
);
out body;
>;
out skel qt;
"""

response = requests.post('http://overpass-api.de/api/interpreter', data={'data': overpass_query})
data = response.json()

# Convert OSM JSON to GeoJSON
geojson = json2geojson(data)

# Extract features and build GeoDataFrame
districts = []
for feature in geojson['features']:
    props = feature['properties']
    geom = shape(feature['geometry'])
    districts.append({'name': props.get('name', 'Unknown'), 'geometry': geom})

districts_gdf = gpd.GeoDataFrame(districts, crs="EPSG:4326")

# Fix invalid geometries if any
```

```
districts_gdf['geometry'] = districts_gdf['geometry'].buffer(0)
```

Data visualization

```
# Extract years, populations, and growth scores from results list
years = [res['year'] for res in results]
populations = [res['population'] for res in results]
growth_scores = [res['growth_score'] for res in results]

fig, ax1 = plt.subplots(figsize=(10,6))

# Plot population on primary y-axis
color_pop = 'tab:blue'
ax1.set_xlabel('Year')
ax1.set_ylabel('Population', color=color_pop)
ax1.plot(years, populations, marker='o', color=color_pop, label='Population')
ax1.tick_params(axis='y', labelcolor=color_pop)

# Create a secondary y-axis to plot growth likelihood
ax2 = ax1.twinx()
color_growth = 'tab:green'
ax2.set_ylabel('Urban Growth Likelihood', color=color_growth)
ax2.plot(years, growth_scores, marker='x', linestyle='--', color=color_growth, label='Growth Likelihood')
ax2.tick_params(axis='y', labelcolor=color_growth)
ax2.set_ylim(0, 1)  # Since growth likelihood is clipped between 0 and 1

# Add title and legends
plt.title('Population and Predicted Urban Growth Likelihood Over 10 Years')
fig.tight_layout()
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```




```
import folium
import numpy as np
import geopandas as gpd
from shapely.geometry import Polygon, MultiPolygon
from shapely.ops import unary_union
from folium import plugins
import branca

def create_maps(center_lat, center_lon, radius_km, pred_score, direction_rad):
    # Filter districts polygons only
    poly_mask = districts_gdf.geometry.type.isin(['Polygon', 'MultiPolygon'])
    districts_poly_gdf = districts_gdf[poly_mask].copy()

    # Convert all polygons to MultiPolygon for consistency
    districts_poly_gdf['geometry'] = districts_poly_gdf['geometry'].apply(
        lambda g: MultiPolygon([g]) if g.geom_type == 'Polygon' else g
    )

    # Compute union polygon of all districts to get the yellow boundary shape
    union_polygon = unary_union(districts_poly_gdf.geometry)

    # Prepare green urban extent boxes clipped inside districts
```

```python
def coords_to_polygon(coords):
    return Polygon(coords)

green_boxes = {
    'Harare': coords_to_polygon(harare_coords),
    'Chitungwiza': coords_to_polygon(chitungwiza_coords),
    'Epworth': coords_to_polygon(epworth_coords)
}

green_gdf = gpd.GeoDataFrame(
    {'name': list(green_boxes.keys()), 'geometry': list(green_boxes.values())},
    crs=districts_poly_gdf.crs
)

clipped_green_gdf = gpd.overlay(
    green_gdf, districts_poly_gdf, how='intersection', keep_geom_type=False, make_valid=True
)
if 'name' not in clipped_green_gdf.columns:
    name_cols = [col for col in clipped_green_gdf.columns if 'name' in col]
    clipped_green_gdf['name'] = clipped_green_gdf[name_cols[0]] if name_cols else 'Unknown'

# Initialize base map
base_map = folium.Map(location=[center_lat, center_lon], zoom_start=11, tiles=None)
folium.TileLayer(
    tiles='https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}',
    attr='Google',
    name='Google Maps',
    overlay=False,
    control=True
).add_to(base_map)
folium.TileLayer(
    tiles='https://mt1.google.com/vt/lyrs=s&x={x}&y={y}&z={z}',
    attr='Google',
    name='Google Satellite',
    overlay=False,
    control=True
).add_to(base_map)

# Add yellow/orange union polygon boundary covering all districts
folium.GeoJson(
    data=gpd.GeoSeries([union_polygon]).__geo_interface__,
    style_function=lambda x: {
        'fillColor': '#ffff00',
        'color': 'orange',
        'weight': 3,
        'fillOpacity': 0.15
    },
    tooltip="Focus Area Boundary covering all districts"
).add_to(base_map)
```

```python
# Add green urban extent boxes inside yellow boundary with detailed popups/tooltips
color_map = {
    'Harare': '#1f78b4',
    'Chitungwiza': '#33a02c',
    'Epworth': '#e31a1c'
}
for _, row in clipped_green_gdf.iterrows():
    name = row['name']
    popup_html = f"""
    <b>{name} Urban Growth Area</b><br>
    Likelihood of growth: {pred_score:.2f}<br>
    Center coordinates: ({center_lat:.4f}, {center_lon:.4f})<br>
    Radius: {radius_km} km<br>
    Model prediction based on input parameters.
    """
    iframe = branca.element.IFrame(html=popup_html, width=300, height=130)
    popup = folium.Popup(iframe, max_width=300)
    geom = row['geometry']
    if geom.geom_type == 'Polygon':
        coords = [(lat, lon) for lon, lat in geom.exterior.coords]
        folium.Polygon(
            locations=coords,
            color=color_map.get(name, 'green'),
            weight=3,
            fill=True,
            fill_color=color_map.get(name, 'green'),
            fill_opacity=0.5,
            popup=popup,
            tooltip=folium.Tooltip(f"Click for info: {name} Area"),
            highlight=True
        ).add_to(base_map)
    elif geom.geom_type == 'MultiPolygon':
        for part in geom.geoms:
            coords = [(lat, lon) for lon, lat in part.exterior.coords]
            folium.Polygon(
                locations=coords,
                color=color_map.get(name, 'green'),
                weight=3,
                fill=True,
                fill_color=color_map.get(name, 'green'),
                fill_opacity=0.5,
                popup=popup,
                tooltip=folium.Tooltip(f"Click for info: {name} Area"),
                highlight=True
            ).add_to(base_map)

# Input center circle and labeled marker with popup
folium.Circle(
    location=[center_lat, center_lon],
    radius=radius_km * 1000,
```

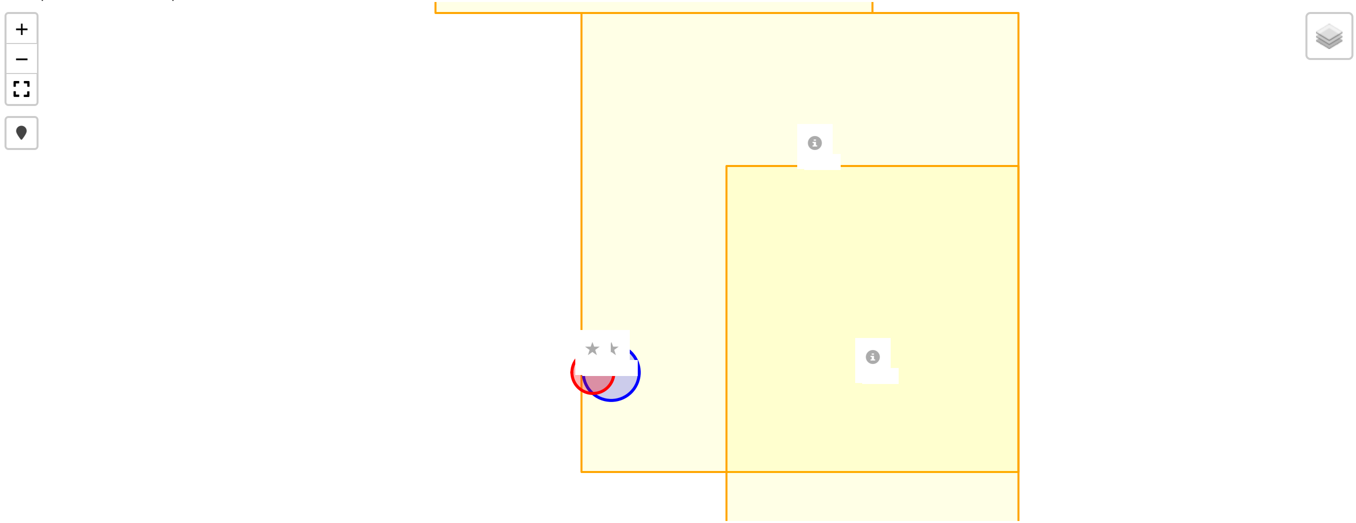
```
        color='blue',
        fill=True,
        fill_opacity=0.2,
        popup=folium.Popup(f"""
            <b>Input Area</b><br>
            Center: ({center_lat:.4f}, {center_lon:.4f})<br>
            Radius: {radius_km} km<br>
            User-selected focus area.
        """, max_width=300),
        tooltip="Input Area"
    ).add_to(base_map)
    folium.Marker(
        location=[center_lat, center_lon],
        popup="Input Center",
        tooltip="Input Center",
        icon=folium.Icon(color='blue', icon='star')
    ).add_to(base_map)

    # Predicted growth circle and labeled marker with popup
    offset_distance_deg = km_to_deg(radius_km * 0.7)
    pred_center_lat = center_lat + offset_distance_deg * np.sin(direction_rad)
    pred_center_lon = center_lon + offset_distance_deg * np.cos(direction_rad)
    growth_radius_km = radius_km * (0.5 + pred_score * 0.5)

    pred_popup_html = f"""
    <b>Predicted Urban Growth Area</b><br>
    Likelihood: {pred_score:.2f}<br>
    Center Offset: {pred_center_lat:.4f}, {pred_center_lon:.4f}<br>
    Radius: {growth_radius_km:.2f} km<br>
    Model prediction based on input parameters.
    """
    iframe = branca.element.IFrame(html=pred_popup_html, width=300, height=130)
    pred_popup = folium.Popup(iframe, max_width=300)

    folium.Circle(
        location=[pred_center_lat, pred_center_lon],
        radius=growth_radius_km * 1000,
        color='red',
        fill=True,
        fill_opacity=0.3,
        popup=pred_popup,
        tooltip=folium.Tooltip("Predicted Growth Area - Click for details")
    ).add_to(base_map)
    folium.Marker(
        location=[pred_center_lat, pred_center_lon],
        popup=pred_popup,
        tooltip="Predicted Growth Center",
        icon=folium.Icon(color='red', icon='star')
    ).add_to(base_map)
```

```
    folium.LayerControl().add_to(base_map)
    plugins.Fullscreen().add_to(base_map)
    plugins.LocateControl().add_to(base_map)

    # Second map: Google Maps base with markers and circles, zoomed to input center
    simple_map = folium.Map(location=[center_lat, center_lon], zoom_start=12, tiles=None)
    folium.TileLayer(
        tiles='https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}',
        attr='Google',
        name='Google Maps',
        overlay=False,
        control=True
    ).add_to(simple_map)

    folium.Circle(
        location=[center_lat, center_lon],
        radius=radius_km * 1000,
        color='blue',
        fill=True,
        fill_opacity=0.2,
        popup="Input Area"
    ).add_to(simple_map)
    folium.Marker(
        location=[center_lat, center_lon],
        popup="Input Center",
        tooltip="Input Center",
        icon=folium.Icon(color='blue', icon='star')
    ).add_to(simple_map)

    folium.Circle(
        location=[pred_center_lat, pred_center_lon],
        radius=growth_radius_km * 1000,
        color='red',
        fill=True,
        fill_opacity=0.3,
        popup="Predicted Growth Area"
    ).add_to(simple_map)
    folium.Marker(
        location=[pred_center_lat, pred_center_lon],
        popup="Predicted Growth Center",
        tooltip="Predicted Growth Center",
        icon=folium.Icon(color='red', icon='star')
    ).add_to(simple_map)

    folium.LayerControl().add_to(simple_map)
    plugins.Fullscreen().add_to(simple_map)
    plugins.LocateControl().add_to(simple_map)

    return base_map, simple_map
```

```
center_latitude = CENTER_LATITUDE
center_longitude = CENTER_LONGITUDE
analysis_radius_km = ANALYSIS_RADIUS_KM

print("center = " + str(center_latitude) + "," + str(center_longitude))

predicted_growth_score = growth_score
predicted_growth_direction_rad = growth_direction


output1 = widgets.Output()

with output1:
    output1.clear_output()
    detailed_map, _ = generate_urban_growth_maps(center_latitude, center_longitude, analysis_radius_km, predicted_growth_score, predicted_growth_direction_rad)
    print("\n--- Map 1: Harare Metropolitan with Districts and Predictions ---")
    display(detailed_map)

display(output1)
```

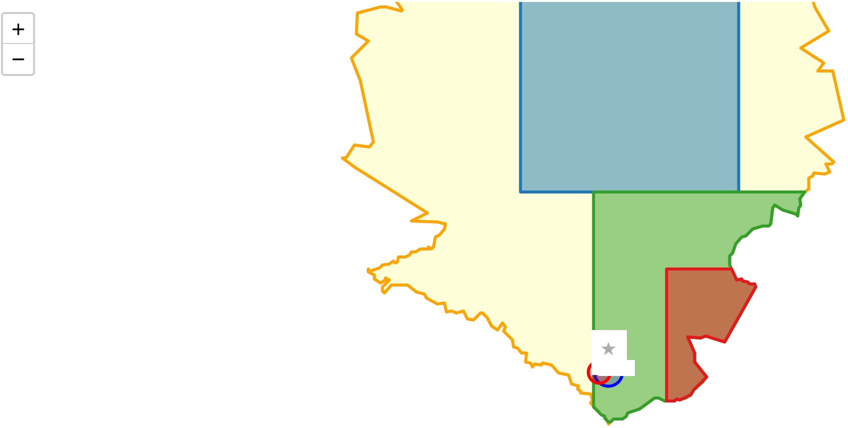
center = -18.0174072,31.0603241

--- Map 1: Harare Metropolitan with Districts and Predictions ---



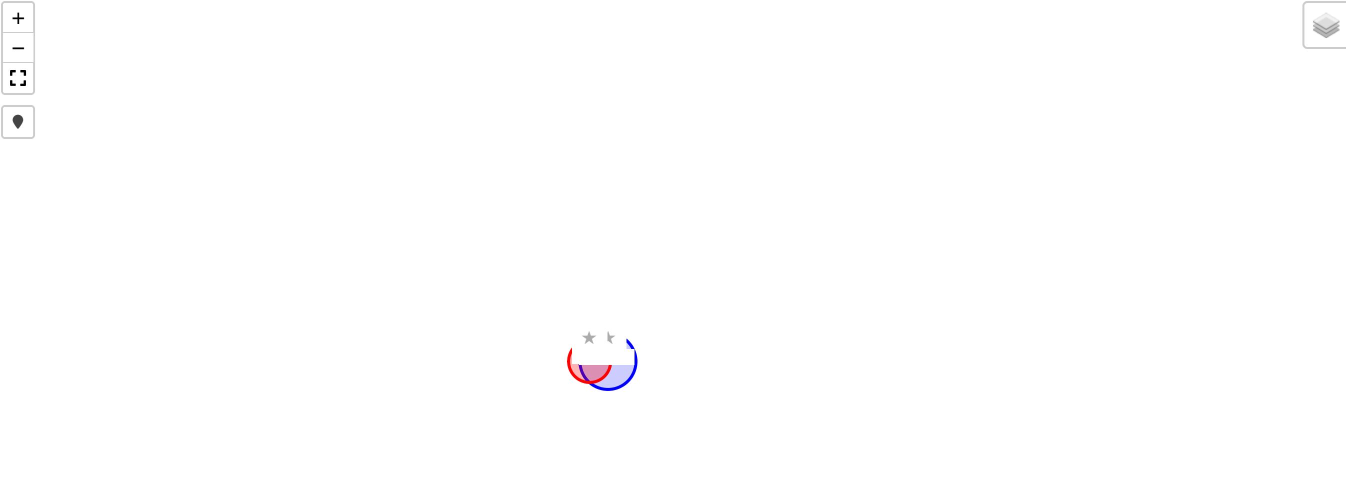
```
output1 = widgets.Output()

with output1:
    output1.clear_output()
    map1, _ = create_maps(center_latitude, center_longitude, analysis_radius_km, predicted_growth_score, predicted_growth_direction_rad)
    print("\n--- Map 1: Harare Metropolitan with Districts and Predictions ---")
    display(map1)

display(output1)
```

--- Map 1: Harare Metropolitan with Districts and Predictions ---




```
from IPython.display import display, HTML

output2 = widgets.Output()

with output2:
    output2.clear_output()
    _, map2 = create_maps(center_latitude, center_longitude, analysis_radius_km, predicted_growth_score, predicted_growth_direction_rad)
    print("\n--- Map 2: Google Maps Base with Predicted Areas ---")
    display(HTML(map2._repr_html_()))

display(output2)
```

```
--- Map 2: Google Maps Base with Predicted Areas ---
```



```
offset_deg = km_to_deg(analysis_radius_km * 0.7)
predicted_growth_lat = center_latitude + offset_deg * np.sin(predicted_growth_direction_rad)
predicted_growth_lon = center_longitude + offset_deg * np.cos(predicted_growth_direction_rad)

district_names = districts_gdf['name'].unique()
cmap = cm.get_cmap('Set2', len(district_names))
color_map = {name: colors.rgb2hex(cmap(i)) for i, name in enumerate(district_names)}

m = folium.Map(location=[center_latitude, center_longitude], zoom_start=13)

for _, district in districts_gdf.iterrows():
    color = color_map.get(district['name'], '#808080')
    gj = folium.GeoJson(
        district['geometry'],
        style_function=lambda feature, col=color: {
            'fillColor': col,
            'color': 'black',
            'weight': 2,
            'fillOpacity': 0.4,
        }
```

```
    )
    tooltip = folium.Tooltip(f"District: {district['name']}")
    popup_html = f"<b>District: {district['name']}</b>"
    popup = folium.Popup(popup_html, max_width=300)
    gj.add_child(tooltip)
    gj.add_child(popup)
    gj.add_to(m)

center_point = Point(center_longitude, center_latitude)
focused_district = None
for _, district in districts_gdf.iterrows():
    if district.geometry.contains(center_point):
        focused_district = district
        break

district_name = focused_district['name'] if focused_district is not None else "Unknown"

buffer_radius_km = 5
buffer_radius_deg = buffer_radius_km / 111

input_center_point = Point(center_longitude, center_latitude)
predicted_growth_point = Point(predicted_growth_lon, predicted_growth_lat)
input_center_buffer = input_center_point.buffer(buffer_radius_deg)
predicted_growth_buffer = predicted_growth_point.buffer(buffer_radius_deg)

def filter_nearby(gdf):
    return gdf[gdf.geometry.intersects(input_center_buffer) | gdf.geometry.intersects(predicted_growth_buffer)].copy()

urban_areas_nearby = filter_nearby(districts_gdf)
non_urban_areas_nearby = filter_nearby(districts_gdf)
expansion_areas_nearby = filter_nearby(districts_gdf)

def create_popup_html(row):
    html = f"""
    <div style="font-family: Arial; font-size: 14px; max-width: 300px;">
        <h4 style="margin-bottom: 8px;">{row.get('name', 'Unknown')}</h4>
        <b>Type:</b> {row.get('type', 'N/A')}<br>
        <b>Population:</b> {row.get('population', 'N/A')}<br>
        <b>Density (people/km²):</b> {row.get('pop_density', 'N/A')}<br>
        <b>Urban State:</b> {row.get('urban_state', 'N/A')}<br>
    """
    if 'growth_likelihood' in row and row['growth_likelihood'] is not None:
        html += f"<b>Growth Likelihood:</b> {row['growth_likelihood']:.2f}<br>"
    html += "</div>"
    return html
```