```
import numpy as np
import pandas as pd
import random
import cv2
import matplotlib.pyplot as plt

# If you're using Google Colab, you can install seaborn (for example)
and other packages as needed:
# !pip install seaborn

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error

# For displaying images in Google Colab
from google.colab.patches import cv2_imshow

def generate_synthetic_data(num_samples=100):
    """
    Generates synthetic room layout data:
    - room_width, room_height
    - furniture (list of (type, width, height))
    - obstacles (list of (x, y, w, h))
    - target_layout: actual placed furniture with coordinates
    """
    data = []

    def place_furniture(room_width, room_height, furniture,
obstacles):
        layout = []
        grid = np.zeros((room_height, room_width))
        # Mark obstacles on grid (occupied cells)
        for obs_x, obs_y, obs_w, obs_h in obstacles:
            grid[obs_y:obs_y + obs_h, obs_x:obs_x + obs_w] = 1

        # Attempt to place each furniture item without overlap
        for f_type, f_width, f_height in furniture:
            placed = False
            for _ in range(100):  # Try up to 100 random placements
                x = random.randint(0, room_width - f_width)
                y = random.randint(0, room_height - f_height)
                if np.sum(grid[y:y + f_height, x:x + f_width]) == 0:
                    layout.append({
                        "furniture_type": f_type,
                        "x": x,
                        "y": y,
                        "width": f_width,
                        "height": f_height
                    })
                    grid[y:y + f_height, x:x + f_width] = 1  # Mark as
```

```
occupied
                    placed = True
                    break
            if not placed:
                # Indicate furniture that couldn't be placed
                layout.append({
                    "furniture_type": f_type,
                    "x": -1,
                    "y": -1,
                    "width": f_width,
                    "height": f_height
                })
        return layout

    for _ in range(num_samples):
        room_width = random.randint(3, 10)
        room_height = random.randint(3, 10)

        num_furniture = random.randint(2, 5)
        furniture = [(random.choice(['Table', 'Chair', 'Sofa', 'Bed',
'Desk']),
                      random.randint(1, min(3, room_width)),
                      random.randint(1, min(3, room_height)))
                     for _ in range(num_furniture)]

        num_obstacles = random.randint(0, 2)
        obstacles = [(random.randint(0, room_width - 1),
                      random.randint(0, room_height - 1),
                      random.randint(1, min(2, room_width - 1)),
                      random.randint(1, min(2, room_height - 1)))
                     for _ in range(num_obstacles)]

        target_layout = place_furniture(room_width, room_height,
furniture, obstacles)

        data.append({
            'room_width': room_width,
            'room_height': room_height,
            'furniture': furniture,        # Original furniture
specification
            'obstacles': obstacles,
            'target_layout': target_layout # Generated layout with
placements
        })

    return pd.DataFrame(data)

# 1. Generate dataset
df = generate_synthetic_data(100)
```

```
# 2. Display a table of the DataFrame
print("First 5 rows of the dataset:")
display(df.head())

# If you want additional graphs, you can add them here. For example:
# import seaborn as sns
# sns.histplot(df['room_width'])
# plt.title("Distribution of Room Widths")
# plt.show()

# 3. Prepare data for model training
df_expanded = pd.json_normalize(df['target_layout'].explode())
df_model = pd.concat([df[['room_width', 'room_height']], df_expanded],
axis=1).dropna()

# Model training (Predict x, y from room_width, room_height)
X = df_model[['room_width', 'room_height']]
y = df_model[['x', 'y']]

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=3,
scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_

# 4. Model evaluation
y_pred = best_model.predict(X_test)
print(f'Best Model Parameters: {grid_search.best_params_}')
print(f'MSE: {mean_squared_error(y_test, y_pred)}')

# 5. Visualization (OpenCV)
def visualize_layout_opencv(room_width, room_height, target_layout,
obstacles, predictions):
    # Create a white canvas with a scaling factor (50 pixels per unit)
    canvas = np.ones((room_height * 50, room_width * 50, 3),
dtype=np.uint8) * 255

    # Draw obstacles in red
    for obs_x, obs_y, obs_w, obs_h in obstacles:
```

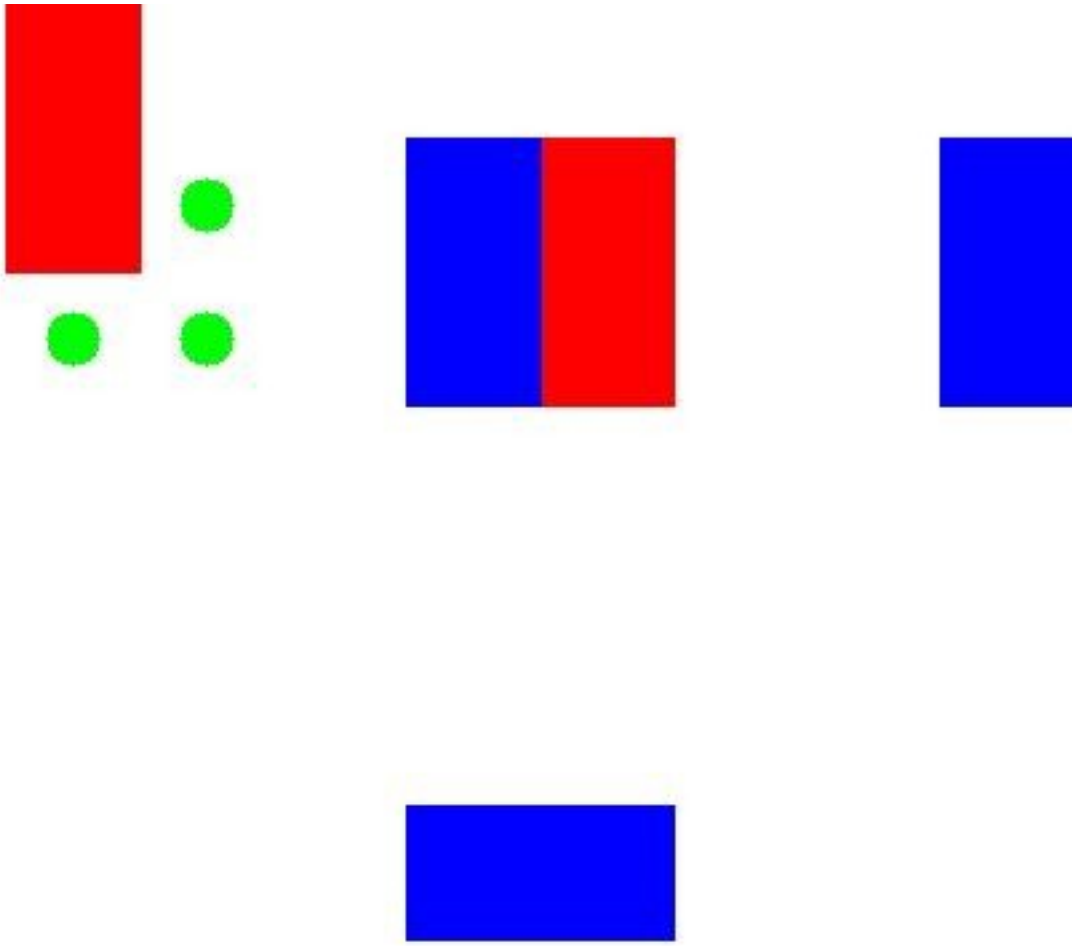
```
        cv2.rectangle(canvas, (obs_x * 50, obs_y * 50),
                      ((obs_x + obs_w) * 50, (obs_y + obs_h) * 50),
                      (0, 0, 255), -1)

    # Draw furniture (from target_layout) in blue
    for item in target_layout:
        f_type = item["furniture_type"]
        x = item["x"]
        y = item["y"]
        f_width = item["width"]
        f_height = item["height"]
        cv2.rectangle(canvas, (x * 50, y * 50),
                      ((x + f_width) * 50, (y + f_height) * 50),
                      (255, 0, 0), -1)

    # Draw predicted positions as green circles
    for i, (x, y) in enumerate(predictions):
        cv2.circle(canvas, (int(x) * 50 + 25, int(y) * 50 + 25), 10,
(0, 255, 0), -1)

    # Show layout in Colab
    cv2_imshow(canvas)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Example visualization using the first sample from the dataset
sample = df.iloc[0]
visualize_layout_opencv(
    sample['room_width'],
    sample['room_height'],
    sample['target_layout'],
    sample['obstacles'],
    y_pred[:5]  # Show the first 5 predicted positions
)
```

First 5 rows of the dataset:


```
{"summary":"{\n  \"name\": \")\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"room_width\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 5,\n
\"max\": 9,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          9,\n          7,\n          5\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"room_height\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2,\n        \"min\": 3,\n        \"max\": 10,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          7,\n
3,\n          8\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"furniture\",\n      \"properties\": {\n        \"dtype\":
```

\"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"obstacles\",\n   \"properties\": {\n   \"dtype\":
\"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"target_layout\",\n   \"properties\": {\n   \"dtype\":
\"object\",\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
Best Model Parameters: {'max_depth': 10, 'min_samples_leaf': 4,
'min_samples_split': 10, 'n_estimators': 50}
MSE: 4.769572482524676
```



```
WARNING:root:Quickchart encountered unexpected dtypes in columns:
"(['furniture', 'obstacles', 'target_layout'],)"

<google.colab._quickchart_helpers.SectionTitle at 0x7a863fd1c090>
```

```
from matplotlib import pyplot as plt
_df_16['index'].plot(kind='hist', bins=20, title='index')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_17['room_width'].plot(kind='hist', bins=20, title='room_width')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_18['room_height'].plot(kind='hist', bins=20, title='room_height')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7a8640453f10>
```

```
from matplotlib import pyplot as plt
_df_19.plot(kind='scatter', x='index', y='room_width', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_20.plot(kind='scatter', x='room_width', y='room_height', s=32,
alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7a8640453a10>
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['index']
  ys = series['room_width']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_21.sort_values('index', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('index')
_ = plt.ylabel('room_width')
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['index']
  ys = series['room_height']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])
```

```
fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_22.sort_values('index', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('index')
_ = plt.ylabel('room_height')
```

```
from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  counted = (series['index']
                .value_counts()
              .reset_index(name='counts')
              .rename({'index': 'index'}, axis=1)
              .sort_values('index', ascending=True))
  xs = counted['index']
  ys = counted['counts']
  plt.plot(xs, ys, label=series_name, color=palette[series_index %
len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = _df_23.sort_values('index', ascending=True)
_plot_series(df_sorted, '')
sns.despine(fig=fig, ax=ax)
plt.xlabel('index')
_ = plt.ylabel('count()')
```

```
<google.colab._quickchart_helpers.SectionTitle at 0x7a8640450490>
```

```
from matplotlib import pyplot as plt
_df_24['index'].plot(kind='line', figsize=(8, 4), title='index')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_25['room_width'].plot(kind='line', figsize=(8, 4),
title='room_width')
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
from matplotlib import pyplot as plt
_df_26['room_height'].plot(kind='line', figsize=(8, 4),
title='room_height')
plt.gca().spines[['top', 'right']].set_visible(False)
```