HHS Staffing Plan Analysis & Synthetic Data Generation

Objective

Analyze the HHS staffing plan dataset to identify staffing patterns and predict total staff involved. Generate synthetic data to augment the small dataset for further analysis.

Data Overview

The dataset includes staff numbers across various HHS agencies, with no missing values. It's structured for regression analysis, aiming to predict total staff involved.

Methodology

Data Preprocessing

Excluded the categorical "Staff involved" column for regression analysis.

Model Training and Evaluation

Used Linear Regression, Random Forest, and Gradient Boosting models. Evaluated models based on MAE, MSE, and R^2 Score. Linear Regression showed near-perfect accuracy.

Synthetic Data Generation

Generated synthetic data by fitting a normal distribution to each feature and sampling new values, creating 100 synthetic samples.

Results

- Linear Regression effectively predicted total staff involved.
- Synthetic data generation provided additional data for analysis.

Conclusion

The project provided insights into HHS staffing and a method for data augmentation. Future steps could explore advanced synthetic data generation techniques or further model tuning.

```
import pandas as pd
```

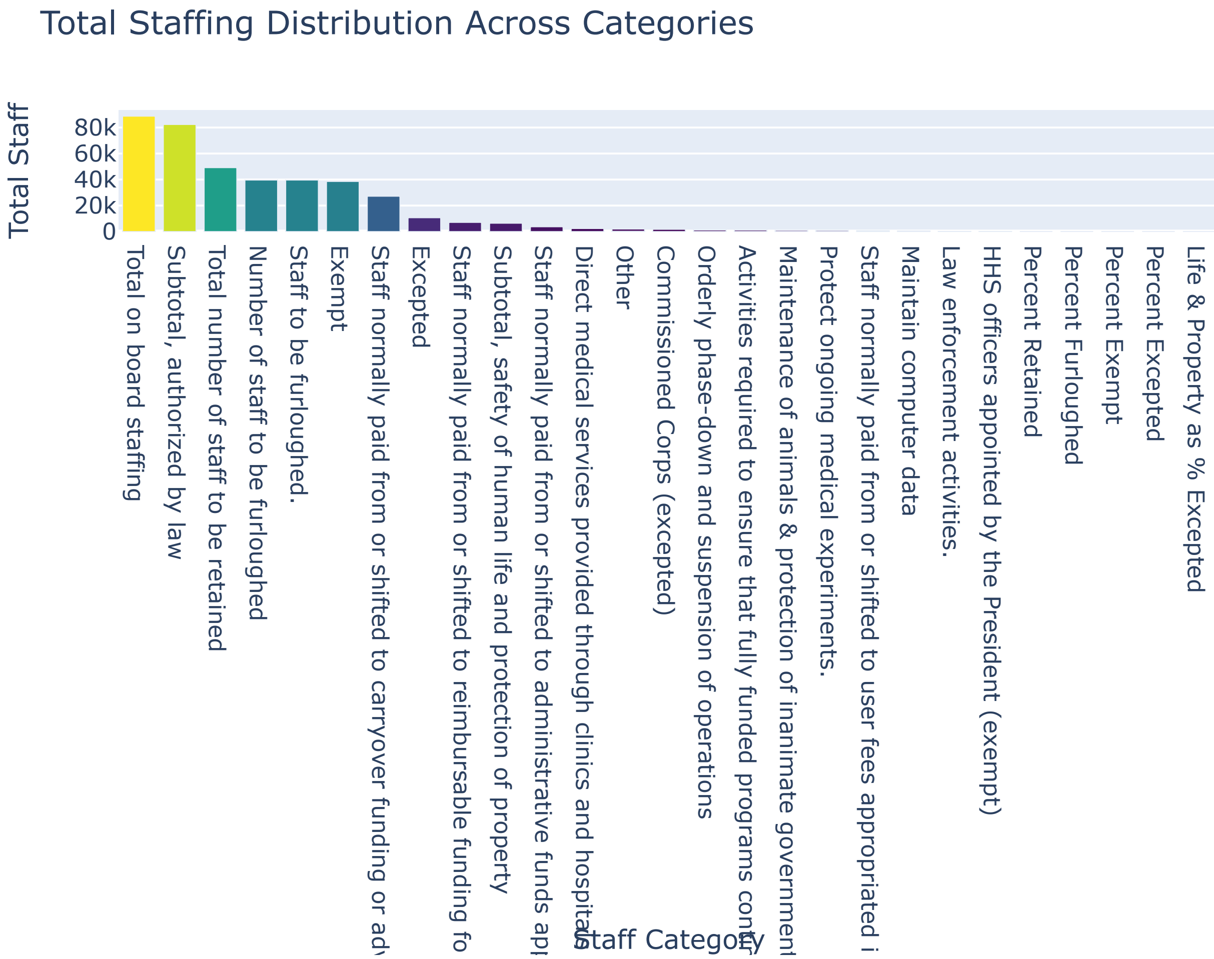
```
# Load and Read the dataset as pandas dataframe
data = pd.read_csv('/content/FY_2024_HHS_Contingency_Staffing_Plan_for_a_Lapse_in_Approp


# Display the first few rows of the dataset to understand its structure and contents
data.head()
```

	Staff involved	ACF	ACL	AHRQ	ARPA-H	ASPR	CDC	CMS	FDA	HRSA	IHS	N
0	Staff normally paid from or shifted to adminis...	40.0	0.0	0.0	0	0.0	89.0	1924.0	0.0	62.0	0	
1	Staff normally paid from or shifted to carryov...	599.0	9.0	26.0	88	531.0	2379.0	531.0	12504.0	1141.0	8233	
2	Staff normally paid from or shifted to reimbur...	13.0	3.0	0.0	0	0.0	85.0	0.0	59.0	0.0	6853	
3	Staff normally paid from or shifted to user fe...	0.0	0.0	0.0	0	0.0	0.0	556.0	31.0	38.0	0	
4	Commissioned Corps (excepted)	6.0	0.0	5.0	0	112.0	739.0	84.0	359.0	73.0	0	17

Next steps: Generate code with data View recommended plots

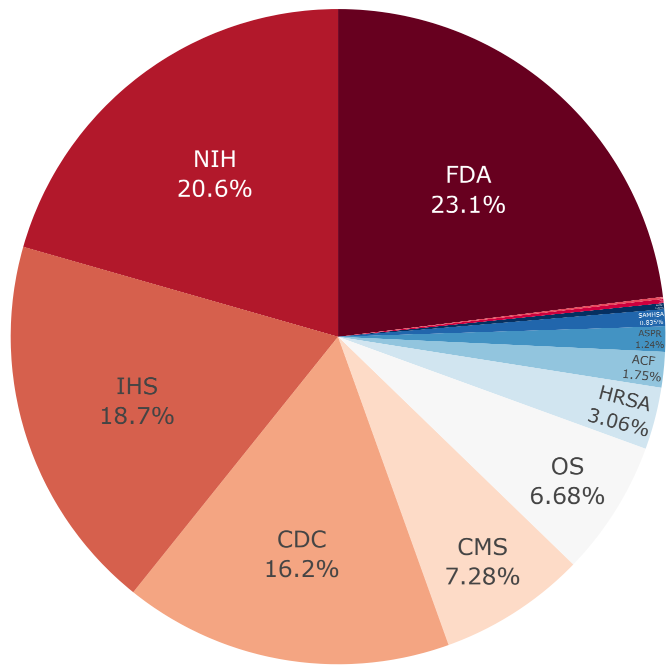
DATA VISUALIZATION

```
# Total Staffing Distribution Across Categories:

import plotly.express as px

# Total Staffing Distribution Across Categories
fig = px.bar(data, x='Staff involved', y='TOTAL', title="Total Staffing Distribution Acr
             labels={'Staff involved': 'Staff Category', 'TOTAL': 'Total Staff'},
             color='TOTAL', color_continuous_scale=px.colors.sequential.Viridis)

fig.update_layout(xaxis={'categoryorder':'total descending'}, coloraxis_colorbar=dict(ti
fig.show()
```

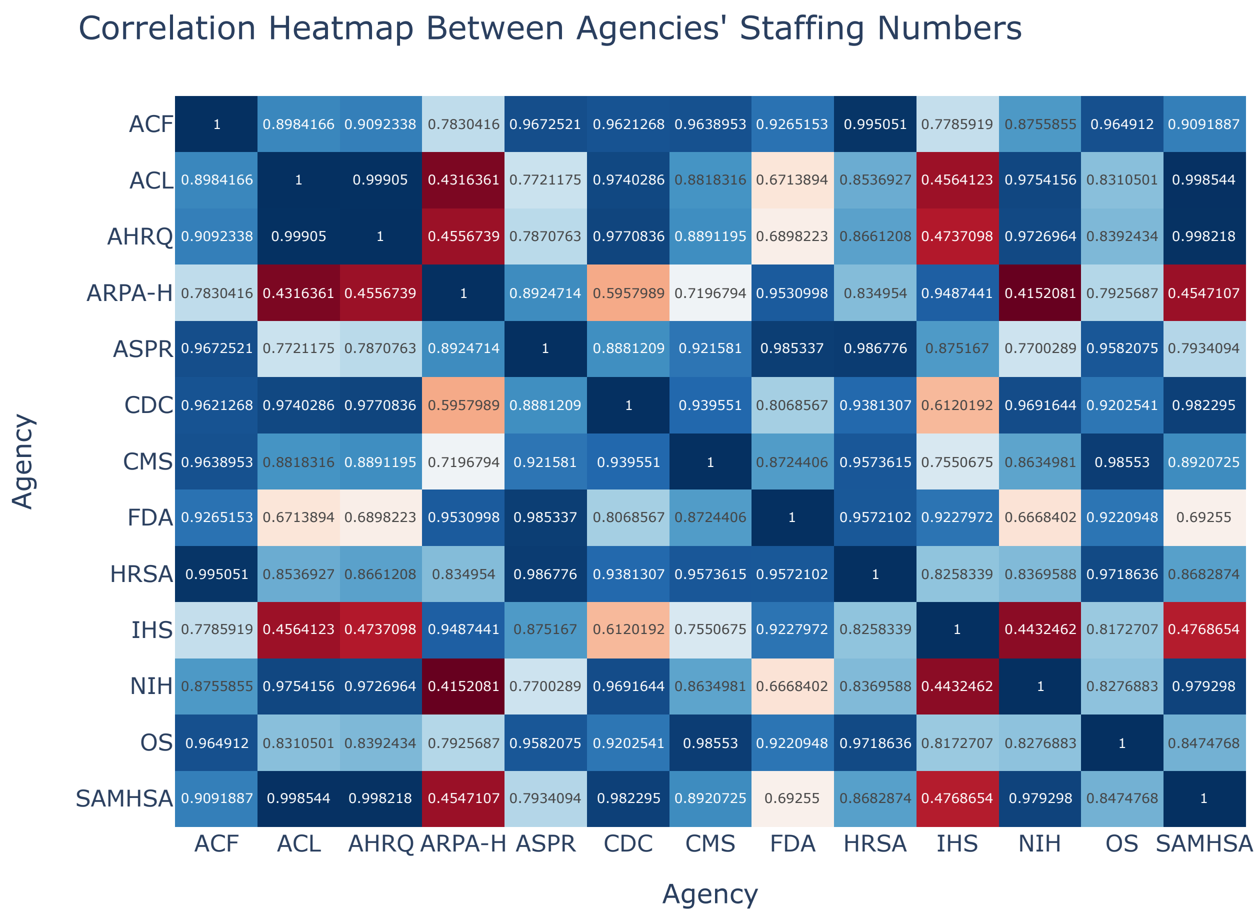



```
# Agency Contributions to Total Staffing (Pie Chart):

# Summing up the contributions from each agency (excluding 'TOTAL' column)
agency_contributions = data.drop(columns=['Staff involved', 'TOTAL']).sum().reset_index(
agency_contributions.columns = ['Agency', 'Contribution']

# Creating the pie chart
fig = px.pie(agency_contributions, values='Contribution', names='Agency',
             title='Agency Contributions to Total Staffing',
             color_discrete_sequence=px.colors.sequential.RdBu)

fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

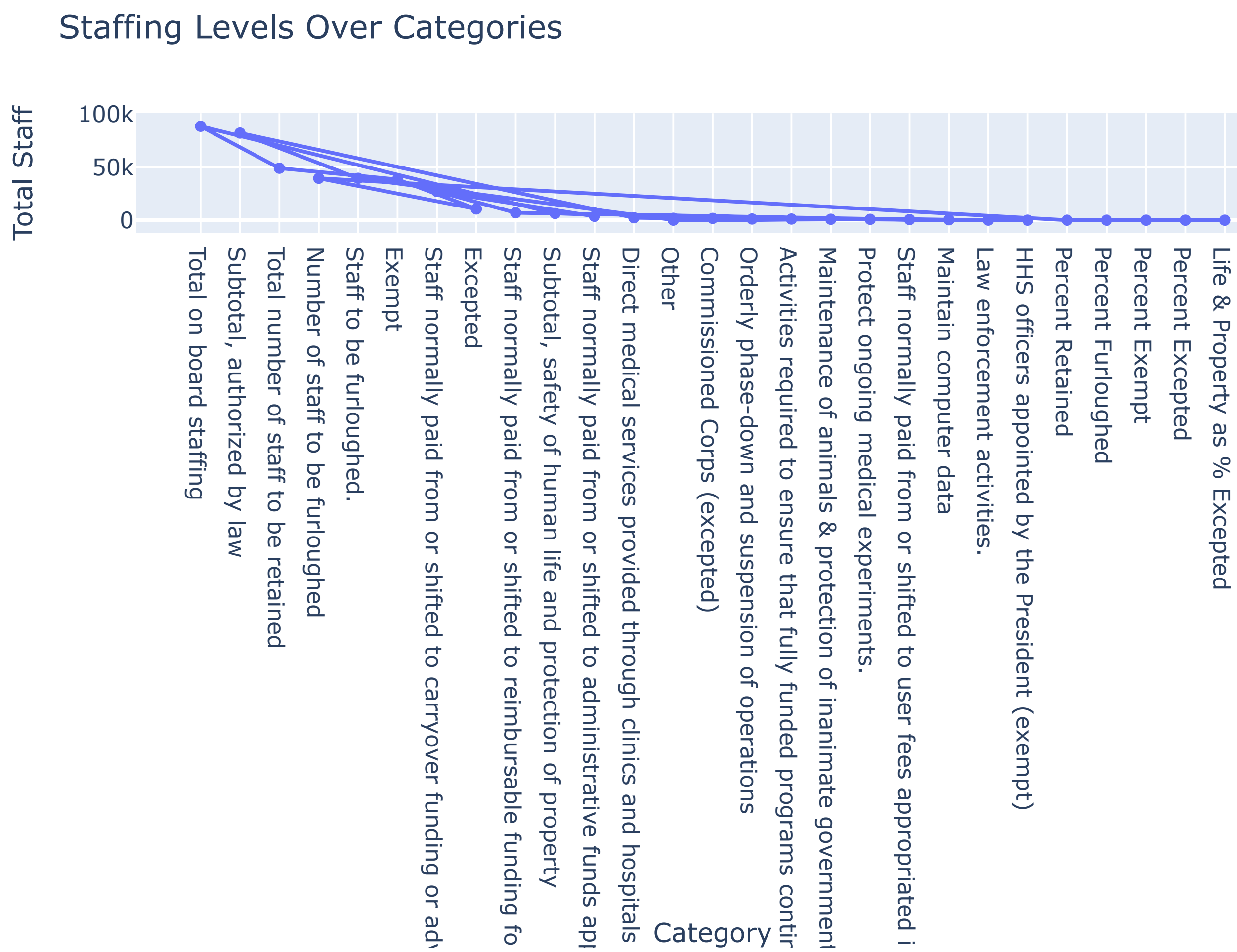
Agency Contributions to Total Staffing




```
# !pip install plotly
import plotly.express as px


# Correlation Heatmap

import numpy as np
# Calculate the correlation matrix for agency columns only
correlation_matrix = data.drop(columns=['Staff involved', 'TOTAL']).corr()

# Use Plotly to create a heatmap for the correlation matrix
fig = px.imshow(correlation_matrix,
                text_auto=True,
                aspect="auto",
                color_continuous_scale="RdBu",
                labels=dict(x="Agency", y="Agency", color="Correlation"))

fig.update_layout(title="Correlation Heatmap Between Agencies' Staffing Numbers",
                  xaxis=dict(tickmode="array", tickvals=np.arange(len(correlation_matrix
                  yaxis=dict(tickmode="array", tickvals=np.arange(len(correlation_matrix

fig.show()
```




```
import plotly.express as px

# Assuming 'data' has columns 'Staff involved' for categories and 'TOTAL' for staffing n
fig_line = px.line(data, x='Staff involved', y='TOTAL', title='Staffing Levels Over Cate
                   labels={'Staff involved': 'Category', 'TOTAL': 'Total Staff'},
                   markers=True)  # Adding markers for each point

fig_line.update_layout(xaxis_title='Category', yaxis_title='Total Staff',
                       xaxis={'categoryorder':'total descending'})  # Order categories b

fig_line.show()
```



DATA PREPROCESSING

```
# Check for missing values in the dataset
missing_values = data.isnull().sum()
missing_values
```

```
Staff involved    0
ACF               0
ACL               0
AHRQ              0
ARPA-H            0
ASPR              0
CDC               0
CMS               0
FDA               0
HRSA              0
IHS               0
NIH               0
OS                0
SAMHSA            0
TOTAL             0
dtype: int64
```

```
# Overview of data types
data_types = data.dtypes
data_types
```

```
Staff involved     object
ACF               float64
ACL               float64
AHRQ              float64
ARPA-H              int64
ASPR              float64
CDC               float64
CMS               float64
FDA               float64
HRSA              float64
IHS                 int64
NIH               float64
OS                float64
SAMHSA            float64
TOTAL             float64
dtype: object
```

Train-Test Split

```
from sklearn.model_selection import train_test_split

# Define features and target
X = data.drop(columns=['Staff involved', 'TOTAL'])  # Features
y = data['TOTAL']  # Target


# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42


# Check the shapes of the resulting splits to confirm the operation
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
((23, 13), (6, 13), (23,), (6,))
```

Model Selection and Training

Given the nature of our task (predicting a continuous variable), we should consider regression models. Based on best practices and the characteristics of our dataset, here are three models we can start with:

Linear Regression: A basic yet powerful model for regression tasks. It's a good starting point due to its simplicity and interpretability. Random Forest Regressor: An ensemble method that can handle non-linear relationships and interactions between features better than linear models. Gradient Boosting Regressor: Another powerful ensemble method that builds models sequentially to minimize errors, often providing high accuracy. We will:

Train each of the three models on the training set. Evaluate their performance on the test set using appropriate metrics, such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and R^2 score.

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score


# Initialize the models
linear_model = LinearRegression()
random_forest_model = RandomForestRegressor(random_state=42)
gradient_boosting_model = GradientBoostingRegressor(random_state=42)


# Dictionary to hold models and their performances
models = {
    "Linear Regression": linear_model,
    "Random Forest": random_forest_model,
    "Gradient Boosting": gradient_boosting_model
}


# Function to train and evaluate a model
def train_evaluate(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)  # Train the model
    predictions = model.predict(X_test)  # Make predictions on the test set
    # Calculate performance metrics
    mae = mean_absolute_error(y_test, predictions)
    mse = mean_squared_error(y_test, predictions)
    r2 = r2_score(y_test, predictions)
    return mae, mse, r2


# Results dictionary
results = {}


# Train and evaluate each model
for name, model in models.items():
    mae, mse, r2 = train_evaluate(model, X_train, X_test, y_train, y_test)
    results[name] = {"MAE": mae, "MSE": mse, "R² Score": r2}


# Print Result
results
```

```
{'Linear Regression': {'MAE': 2.2294454330840012,
  'MSE': 5.582916227673212,
  'R² Score': 0.9999996069913424},
 'Random Forest': {'MAE': 2487.6005666666665,
  'MSE': 13326854.845088443,
  'R² Score': 0.06185779633634536},
 'Gradient Boosting': {'MAE': 5329.811682508874,
  'MSE': 76502627.65728012,
  'R² Score': -4.38539246736911}}
```

```
model_performance = results
# Convert to DataFrame for easier plotting with Plotly
df_performance = pd.DataFrame(model_performance).T.reset_index()
df_performance.rename(columns={'index': 'Model'}, inplace=True)


# Plotting
for metric in ['MAE', 'MSE', 'R² Score']:
    fig = px.bar(df_performance, x='Model', y=metric, title=f'Model Performance Comparis
                 color='Model', barmode='group',
                 color_continuous_scale=px.colors.sequential.Viridis)
    fig.show()
```