Revenue Optimization Through Machine Learning: A Strategic Analysis for Clipboard Health

Author: Zemelak Goraga

Date: 6th Feb 2025

```
# Standard Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Scikit-learn Libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.inspection import permutation_importance



# Suppress warnings
import warnings
warnings.filterwarnings("ignore")


# ETL Process - Load Dataset1
# Load datasets
#staffing_df = pd.read_csv("PBJ_Daily_Nurse_Staffing_Q1_2024.csv")
staffing_df = pd.read_csv("PBJ_Daily_Nurse_Staffing_Q1_2024.csv", encoding='latin1')

staffing_df.head()
```

	PROVNUM	PROVNAME	CITY	STATE	COUNTY_NAME	COUNTY_FIPS	CY_Qtr	WorkDate	MDScensus	Hrs_RNDON	...	Hrs_LPN_ctr	Hrs_
0	15009	BURNS NURSING HOME, INC.	RUSSELLVILLE	AL	Franklin	59	2024Q1	20240101	50	8.0	...	0.0	15
1	15009	BURNS NURSING HOME, INC.	RUSSELLVILLE	AL	Franklin	59	2024Q1	20240102	49	8.0	...	0.0	14
2	15009	BURNS NURSING HOME, INC.	RUSSELLVILLE	AL	Franklin	59	2024Q1	20240103	49	8.0	...	0.0	14
3	15009	BURNS NURSING HOME, INC.	RUSSELLVILLE	AL	Franklin	59	2024Q1	20240104	50	8.0	...	0.0	14
4	15009	BURNS NURSING HOME, INC.	RUSSELLVILLE	AL	Franklin	59	2024Q1	20240105	51	8.0	...	0.0	14

5 rows × 33 columns

```
# ETL Process - Load Dataset2
# Load datasets
performance_df = pd.read_csv("FY_2025_SNF_VBP_Facility_Performance.csv")
performance_df.head()
```

	SNF VBP Program Ranking	Footnote -- SNF VBP Program Ranking	CMS Certification Number (CCN)	Provider Name	Provider Address	City/Town	State	ZIP Code	Baseline Period: FY 2019 Risk-Standardized Readmission Rate	Footnote -- Baseline Period: FY 2019 Risk-Standardized Readmission Rate	Performance Period: FY 2023 Risk-Standardized Readmission Rate
0	7099	NaN	10007	MIZELL MEMORIAL HOSPITAL	702 N MAIN ST	OPP	AL	36467	0.20151	NaN	0.25997
1	520	NaN	10044	MARION REGIONAL MEDICAL CENTER	1256 MILITARY STREET SOUTH	HAMILTON	AL	35570	0.17141	NaN	0.17204
2	1886	NaN	10045	FAYETTE MEDICAL CENTER	1653 TEMPLE AVENUE NORTH	FAYETTE	AL	35555	0.19029	NaN	0.18445
3	5795	NaN	10049	MEDICAL CENTER ENTERPRISE	400 N EDWARDS STREET	ENTERPRISE	AL	36330	---	This facility had fewer than 25 eligible stays...	0.20284
4	5711	NaN	10058	BIBB MEDICAL CENTER	208 PIERSON AVE	CENTREVILLE	AL	35042	0.18396	NaN	0.20244

Next steps: Generate code with performance_df | View recommended plots | New interactive sheet

```
# ETL Process - Load Dataset3
# Load datasets
provider_info_df = pd.read_csv("NH_ProviderInfo_Nov2024.csv")
provider_info_df.head()
```

	CMS Certification Number (CCN)	Provider Name	Provider Address	City/Town	State	ZIP Code	Telephone Number	Provider SSA County Code	County/Parish	Ownership Type	...
0	015009	BURNS NURSING HOME, INC.	701 MONROE STREET NW	RUSSELLVILLE	AL	35653	2563324110	290	Franklin	For profit - Corporation	...
1	015010	COOSA VALLEY HEALTHCARE CENTER	260 WEST WALNUT STREET	SYLACAUGA	AL	35150	2562495604	600	Talladega	For profit - Corporation	...
2	015012	HIGHLANDS HEALTH AND REHAB	380 WOODS COVE ROAD	SCOTTSBORO	AL	35768	2562183708	350	Jackson	Government - County	...
3	015014	EASTVIEW REHABILITATION & HEALTHCARE CENTER	7755 FOURTH AVENUE SOUTH	BIRMINGHAM	AL	35206	2058330146	360	Jefferson	For profit - Individual	...
4	015015	PLANTATION MANOR NURSING HOME	6450 OLD TUSCALOOSA HIGHWAY	MC CALLA	AL	35111	2054776161	360	Jefferson	For profit - Individual	...

5 rows × 103 columns

```
# ETL Process - Load Datasets
import pandas as pd
import numpy as np

# Load datasets
staffing_df = pd.read_csv("PBJ_Daily_Nurse_Staffing_Q1_2024.csv", encoding='latin1')
performance_df = pd.read_csv("FY_2025_SNF_VBP_Facility_Performance.csv")
provider_info_df = pd.read_csv("NH_ProviderInfo_Nov2024.csv")

# Function to clean a dataframe
def clean_dataframe(df):
    # 1. Standardize column names
    df.columns = df.columns.str.strip().str.lower().str.replace(" ", "_")

    # 2. Handle missing values
```

```
        missing_values = df.isnull().sum()
        missing_threshold = 0.5  # Drop columns where more than 50% values are missing
        df = df.dropna(thresh=int(missing_threshold * len(df)), axis=1)

        # Fill missing numerical values with median
        num_cols = df.select_dtypes(include=['number']).columns
        df[num_cols] = df[num_cols].fillna(df[num_cols].median())

        # Fill missing categorical values with mode
        cat_cols = df.select_dtypes(include=['object']).columns
        for col in cat_cols:
            df[col].fillna(df[col].mode()[0], inplace=True)

        # 3. Remove duplicates
        df.drop_duplicates(inplace=True)

        # 4. Convert data types
        for col in num_cols:
            df[col] = pd.to_numeric(df[col], errors='coerce')

        for col in cat_cols:
            df[col] = df[col].astype(str).str.strip()

        # 5. Handle outliers (replace extreme values with 99th percentile)
        for col in num_cols:
            q99 = df[col].quantile(0.99)
            df[col] = np.where(df[col] > q99, q99, df[col])

        # 6. Remove whitespace & special characters from string fields
        df[cat_cols] = df[cat_cols].apply(lambda x: x.str.replace(r'[^a-zA-Z0-9 ]', '', regex=True))

        return df

    # Apply cleaning to all dataframes
    staffing_df = clean_dataframe(staffing_df)
    performance_df = clean_dataframe(performance_df)
    provider_info_df = clean_dataframe(provider_info_df)

    # Verify cleaning
    print("Staffing Data Overview:\n", staffing_df.info())
    print("Performance Data Overview:\n", performance_df.info())
    print("Provider Info Data Overview:\n", provider_info_df.info())
```

```
 76  rating_cycle_3_total_health_score                14807 non-null  float64
 77  total_weighted_health_survey_score               14807 non-null  float64
 78  number_of_facility_reported_incidents            14807 non-null  float64
 79  number_of_substantiated_complaints               14807 non-null  float64
 80  number_of_fines                                  14807 non-null  float64
 81  total_amount_of_fines_in_dollars                 14807 non-null  float64
 82  number_of_payment_denials                        14807 non-null  float64
 83  total_number_of_penalties                        14807 non-null  float64
 84  location                                         14807 non-null  object
 85  latitude                                         14807 non-null  float64
 86  longitude                                        14807 non-null  float64
 87  processing_date                                  14807 non-null  object
dtypes: float64(65), object(23)
memory usage: 9.9+ MB
Provider Info Data Overview:
 None
```

Start coding or generate with AI.

```
# First, let's check data types and NaN values before cleaning
print("Data types before cleaning:")
print(staffing_df.dtypes)
print("\nNaN values before cleaning:")
print(staffing_df.isnull().sum())

# Clean staffing_df
numeric_cols_staffing = staffing_df.select_dtypes(include=['float64', 'int64']).columns
object_cols_staffing = staffing_df.select_dtypes(include=['object']).columns

for col in numeric_cols_staffing:
    staffing_df[col] = pd.to_numeric(staffing_df[col], errors='coerce')
    staffing_df[col] = staffing_df[col].fillna(staffing_df[col].median())

for col in object_cols_staffing:
    staffing_df[col] = staffing_df[col].fillna(staffing_df[col].mode()[0] if not staffing_df[col].mode().empty else 'Unknown')

# Clean performance_df
numeric_cols_perf = performance_df.select_dtypes(include=['float64', 'int64']).columns
object_cols_perf = performance_df.select_dtypes(include=['object']).columns

for col in numeric_cols_perf:
    performance_df[col] = pd.to_numeric(performance_df[col], errors='coerce')
    performance_df[col] = performance_df[col].fillna(performance_df[col].median())

for col in object_cols_perf:
    performance_df[col] = performance_df[col].fillna(performance_df[col].mode()[0] if not performance_df[col].mode().empty else 'Unknown

# Clean provider_info_df
numeric_cols_provider = provider_info_df.select_dtypes(include=['float64', 'int64']).columns
object_cols_provider = provider_info_df.select_dtypes(include=['object']).columns

for col in numeric_cols_provider:
    provider_info_df[col] = pd.to_numeric(provider_info_df[col], errors='coerce')
    provider_info_df[col] = provider_info_df[col].fillna(provider_info_df[col].median())

for col in object_cols_provider:
    provider_info_df[col] = provider_info_df[col].fillna(provider_info_df[col].mode()[0] if not provider_info_df[col].mode().empty else

# Convert ID columns to string for merging
staffing_df['provnum'] = staffing_df['provnum'].astype(str)
performance_df['cms_certification_number_(ccn)'] = performance_df['cms_certification_number_(ccn)'].astype(str)
provider_info_df['cms_certification_number_(ccn)'] = provider_info_df['cms_certification_number_(ccn)'].astype(str)

# Perform merges
merged_df = pd.merge(staffing_df, performance_df, how="outer",
                     left_on="provnum", right_on="cms_certification_number_(ccn)")
merged_df = pd.merge(merged_df, provider_info_df, how="outer",
                     left_on="provnum", right_on="cms_certification_number_(ccn)")

# Clean the final merged dataset
numeric_cols_merged = merged_df.select_dtypes(include=['float64', 'int64']).columns
object_cols_merged = merged_df.select_dtypes(include=['object']).columns

for col in numeric_cols_merged:
    merged_df[col] = pd.to_numeric(merged_df[col], errors='coerce')
    merged_df[col] = merged_df[col].fillna(merged_df[col].median())

for col in object_cols_merged:
    merged_df[col] = merged_df[col].fillna(merged_df[col].mode()[0] if not merged_df[col].mode().empty else 'Unknown')

# Verify the cleaning
print("\nData types after cleaning:")
print(merged_df.dtypes)
```

```
print("\nNaN values after cleaning:")
print(merged_df.isnull().sum())

# Display sample of cleaned merged data
print("\nCleaned Merged Data Sample:")
print(merged_df.head())

# Display summary statistics for numeric columns
print("\nSummary statistics for numeric columns:")
print(merged_df[numeric_cols_merged].describe())
```

```
std    2.942417e+01  ...                      7.888027e+01
min    0.000000e+00  ...                      0.000000e+00
25%    1.307000e+01  ...                      2.000000e+01
50%    2.558000e+01  ...                      4.000000e+01
75%    4.450000e+01  ...                      7.200000e+01
max    1.632500e+02  ...                      4.669400e+02

       total_weighted_health_survey_score  \
count                        1.342976e+06
mean                         7.412965e+01
std                          7.339745e+01
min                          0.000000e+00
25%                          2.866700e+01
50%                          5.133300e+01
75%                          9.066700e+01
max                          4.071370e+02

       number_of_facility_reported_incidents  \
count                           1.342976e+06
mean                            1.594534e+00
std                             3.218827e+00
min                             0.000000e+00
25%                             0.000000e+00
50%                             0.000000e+00
75%                             2.000000e+00
max                             2.000000e+01

       number_of_substantiated_complaints  number_of_fines  \
count                        1.342976e+06     1.342976e+06
mean                         5.869970e+00     1.536115e+00
std                          9.893519e+00     2.273184e+00
min                          0.000000e+00     0.000000e+00
25%                          0.000000e+00     0.000000e+00
50%                          2.000000e+00     1.000000e+00
75%                          7.000000e+00     2.000000e+00
max                          5.700000e+01     1.500000e+01

       total_amount_of_fines_in_dollars  number_of_payment_denials  \
count                      1.342976e+06               1.342976e+06
mean                       3.481758e+04               1.958166e-01
std                        7.093400e+04               5.249520e-01
min                        0.000000e+00               0.000000e+00
25%                        0.000000e+00               0.000000e+00
50%                        7.442500e+03               0.000000e+00
75%                        2.973685e+04               0.000000e+00
max                        3.974083e+05               3.000000e+00

       total_number_of_penalties      latitude     longitude
count               1.342976e+06  1.342976e+06  1.342976e+06
mean                1.738717e+00  3.814211e+01 -8.980961e+01
std                 2.536465e+00  4.689427e+00  1.307701e+01
min                 0.000000e+00  1.348860e+01 -1.652000e+02
25%                 0.000000e+00  3.513200e+01 -9.543700e+01
50%                 1.000000e+00  3.920420e+01 -8.749800e+01
75%                 2.000000e+00  4.133790e+01 -8.151400e+01
max                 1.600000e+01  4.768428e+01 -7.092912e+01

[8 rows x 98 columns]
```

Start coding or generate with AI.

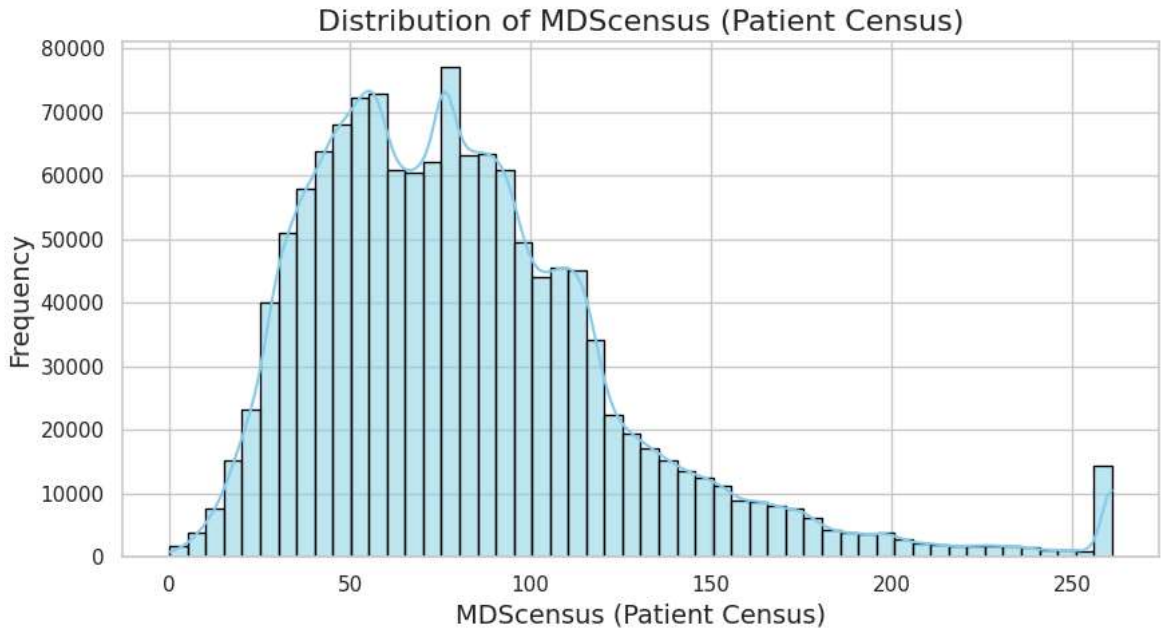
Start coding or generate with AI.

```
# Basic statistics
print("Basic Statistics:\n", merged_df.describe())
```

```
mean                          7.412965e+01
std                           7.339745e+01
min                           0.000000e+00
25%                           2.866700e+01
50%                           5.133300e+01
75%                           9.066700e+01
max                           4.071370e+02

       number_of_facility_reported_incidents  \
count                           1.342976e+06
mean                            1.594534e+00
std                             3.218827e+00
min                             0.000000e+00
25%                             0.000000e+00
50%                             0.000000e+00
75%                             2.000000e+00
max                             2.000000e+01

       number_of_substantiated_complaints  number_of_fines  \
count                        1.342976e+06     1.342976e+06
mean                         5.869970e+00     1.536115e+00
std                          9.893519e+00     2.273184e+00
min                          0.000000e+00     0.000000e+00
25%                          0.000000e+00     0.000000e+00
50%                          2.000000e+00     1.000000e+00
75%                          7.000000e+00     2.000000e+00
max                          5.700000e+01     1.500000e+01

       total_amount_of_fines_in_dollars  number_of_payment_denials  \
count                      1.342976e+06               1.342976e+06
mean                       3.481758e+04               1.958166e-01
std                        7.093400e+04               5.249520e-01
min                        0.000000e+00               0.000000e+00
25%                        0.000000e+00               0.000000e+00
50%                        7.442500e+03               0.000000e+00
75%                        2.973685e+04               0.000000e+00
max                        3.974083e+05               3.000000e+00

       total_number_of_penalties      latitude     longitude
count               1.342976e+06  1.342976e+06  1.342976e+06
mean                1.738717e+00  3.814211e+01 -8.980961e+01
std                 2.536465e+00  4.689427e+00  1.307701e+01
min                 0.000000e+00  1.348860e+01 -1.652000e+02
25%                 0.000000e+00  3.513200e+01 -9.543700e+01
50%                 1.000000e+00  3.920420e+01 -8.749800e+01
75%                 2.000000e+00  4.133790e+01 -8.151400e+01
max                 1.600000e+01  4.768428e+01 -7.092912e+01

[8 rows x 98 columns]
```

Start coding or generate with AI.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Automatically calculate number of bins based on the data
num_bins = int(merged_df["mdscensus"].nunique() / 5)

# Set plot size and style
plt.figure(figsize=(10, 5))
sns.set(style="whitegrid")

# Plotting histogram with KDE for smooth distribution visualization
sns.histplot(merged_df["mdscensus"], bins=num_bins, kde=True, color="skyblue", edgecolor="black")

# Adding title and labels
plt.title("Distribution of MDScensus (Patient Census)", fontsize=16)
plt.xlabel("MDScensus (Patient Census)", fontsize=14)
plt.ylabel("Frequency", fontsize=14)

# Show plot
plt.show()
```



Purpose

This visualization illustrates a statistical representation of patient census distribution, combining a histogram to show frequency counts with a kernel density estimate (KDE) overlay for a smooth distribution curve. It helps healthcare administrators identify patterns and trends in patient occupancy levels across facilities, supporting data-driven decisions for capacity planning and resource allocation.

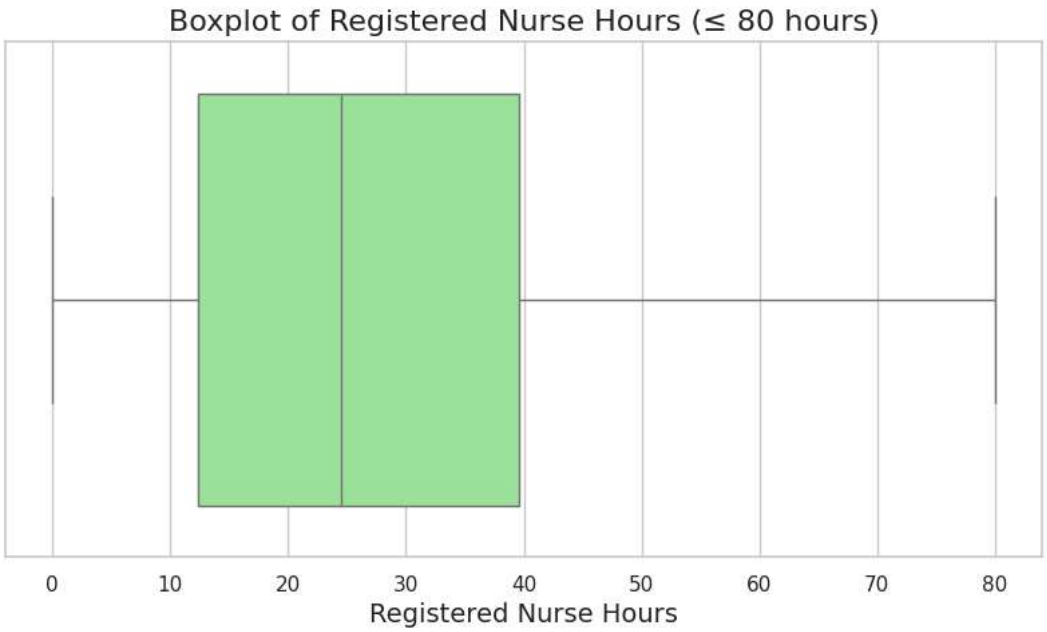
```
#tabular result

# Calculate basic statistics
stats_df = pd.DataFrame({
    'Statistic': [
        'Count',
        'Mean',
        'Median',
        'Standard Deviation',
        'Minimum',
        'Maximum',
        'Skewness',
        'Kurtosis'
    ],
    'Value': [
        merged_df['mdscensus'].count(),
        merged_df['mdscensus'].mean(),
        merged_df['mdscensus'].median(),
        merged_df['mdscensus'].std(),
        merged_df['mdscensus'].min(),
        merged_df['mdscensus'].max(),
        merged_df['mdscensus'].skew(),
        merged_df['mdscensus'].kurtosis()
    ]
})

# Calculate percentiles
percentiles = [0, 25, 50, 75, 100]
percentile_stats = pd.DataFrame({
    'Statistic': [f'{p}th Percentile' for p in percentiles],
    'Value': np.percentile(merged_df['mdscensus'].dropna(), percentiles)
})

# Combine all statistics
all_stats = pd.concat([stats_df, percentile_stats])

# Format the numbers to be more readable
all_stats['Value'] = all_stats['Value'].round(2)

# Display the results
print("MDScensus Distribution Statistics:")
print(all_stats.to_string(index=False))
```

```
MDScensus Distribution Statistics:
            Statistic      Value
                Count 1342976.00
```

```
                    Mean      82.59
                  Median      76.00
      Standard Deviation      44.59
                 Minimum       0.00
                 Maximum     261.00
                Skewness       1.33
                Kurtosis       2.61
           0th Percentile       0.00
          25th Percentile      51.00
          50th Percentile      76.00
          75th Percentile     104.00
         100th Percentile     261.00
```

Start coding or generate with AI.

```
# 2. Boxplot of hrs_rn (Registered Nurse Hours)

# Filter out RN hours above 80
df_filtered = merged_df[merged_df['hrs_rn'] <= 80]

# Create boxplot
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered["hrs_rn"], color="lightgreen")
plt.title("Boxplot of Registered Nurse Hours (≤ 80 hours)", fontsize=16)
plt.xlabel("Registered Nurse Hours", fontsize=14)
plt.show()

# Calculate and display summary statistics for filtered data
stats = {
    'Count': len(df_filtered['hrs_rn']),
    'Mean': df_filtered['hrs_rn'].mean(),
    'Median (Q2)': df_filtered['hrs_rn'].median(),
    'Standard Deviation': df_filtered['hrs_rn'].std(),
    'Minimum': df_filtered['hrs_rn'].min(),
    'Q1 (25th percentile)': df_filtered['hrs_rn'].quantile(0.25),
    'Q3 (75th percentile)': df_filtered['hrs_rn'].quantile(0.75),
    'Maximum': df_filtered['hrs_rn'].max(),
    'IQR (Q3-Q1)': df_filtered['hrs_rn'].quantile(0.75) - df_filtered['hrs_rn'].quantile(0.25),
    'Records Removed': len(merged_df) - len(df_filtered)
}

print("\nRegistered Nurse Hours (hrs_rn) Distribution Statistics (≤ 80 hours):")
stats_df = pd.DataFrame.from_dict(stats, orient='index', columns=['Value'])
print(stats_df.round(2))
```




```
Registered Nurse Hours (hrs_rn) Distribution Statistics (≤ 80 hours):
                          Value
Count                1248184.00
Mean                      27.61
Median (Q2)               24.50
Standard Deviation        19.06
Minimum                    0.00
Q1 (25th percentile)      12.31
Q3 (75th percentile)      39.59
Maximum                   80.00
IQR (Q3-Q1)               27.28
Records Removed         94792.00
```

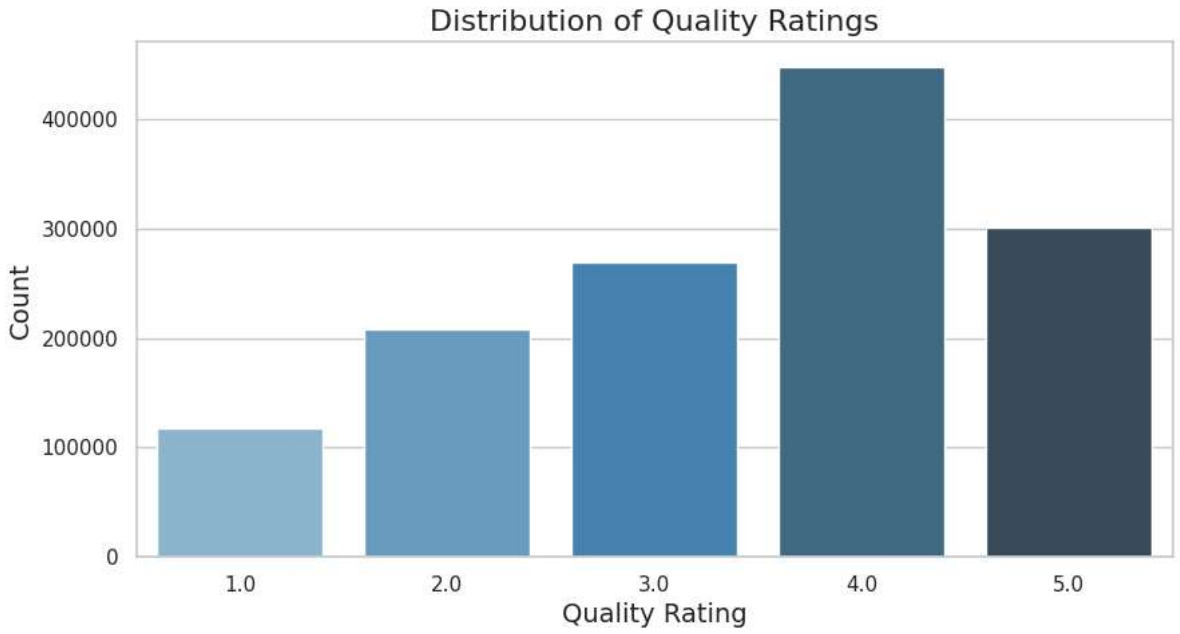
Purpose

This boxplot displays the distribution of registered nurse (RN) hours, highlighting key statistics such as the median, quartiles, and potential outliers. It helps healthcare administrators assess variability in RN staffing levels, identify anomalies, and make informed decisions to optimize workforce allocation and maintain consistent care quality.

Double-click (or enter) to edit

```
# Impute with mean (or median, mode depending on context)
merged_df['performance_score'].fillna(merged_df['performance_score'].mean(), inplace=True)
```

```
# 6. Bar Plot of quality_rating
plt.figure(figsize=(10, 5))
sns.countplot(x=merged_df["qm_rating"], palette="Blues_d")
plt.title("Distribution of Quality Ratings", fontsize=16)
plt.xlabel("Quality Rating", fontsize=14)
plt.ylabel("Count", fontsize=14)
plt.show()
```



Purpose

This bar plot depicts the distribution of quality ratings, showing the frequency of each rating category. It helps healthcare administrators evaluate the overall performance of facilities, identify trends in quality ratings, and prioritize areas for improvement to enhance care standards and operational efficiency.

```
# tabular

import pandas as pd
import numpy as np

# Calculate frequency distribution
rating_counts = merged_df['qm_rating'].value_counts().sort_index()
rating_percentages = merged_df['qm_rating'].value_counts(normalize=True).sort_index() * 100

# Create DataFrame for main statistics
stats_df = pd.DataFrame({
    'Rating': rating_counts.index,
    'Count': rating_counts.values,
    'Percentage': rating_percentages.values,
    'Cumulative_Percentage': rating_percentages.cumsum().values
})

# Create DataFrame for summary statistics
summary_stats = pd.DataFrame({
    'Rating': ['Summary Statistics'],
    'Count': [f"Total: {merged_df['qm_rating'].count()}"],
    'Percentage': [f"Mode: {merged_df['qm_rating'].mode()[0]}"],
    'Cumulative_Percentage': [f"Mean Rating: {merged_df['qm_rating'].mean():.2f}"]
})

# Format the numbers
stats_df['Percentage'] = stats_df['Percentage'].round(2)
stats_df['Cumulative_Percentage'] = stats_df['Cumulative_Percentage'].round(2)

# Add percentage signs
stats_df['Percentage'] = stats_df['Percentage'].astype(str) + '%'
stats_df['Cumulative_Percentage'] = stats_df['Cumulative_Percentage'].astype(str) + '%'

# Combine the tables
final_stats = pd.concat([stats_df, summary_stats])

# Create detailed analysis DataFrame
detailed_analysis = pd.DataFrame({
    'Metric': [
        'Total Facilities',
        'Average (Mean) Rating',
        'Median Rating',
        'Mode (Most Common) Rating',
        'Standard Deviation',
        'Minimum Rating',
        'Maximum Rating',
```

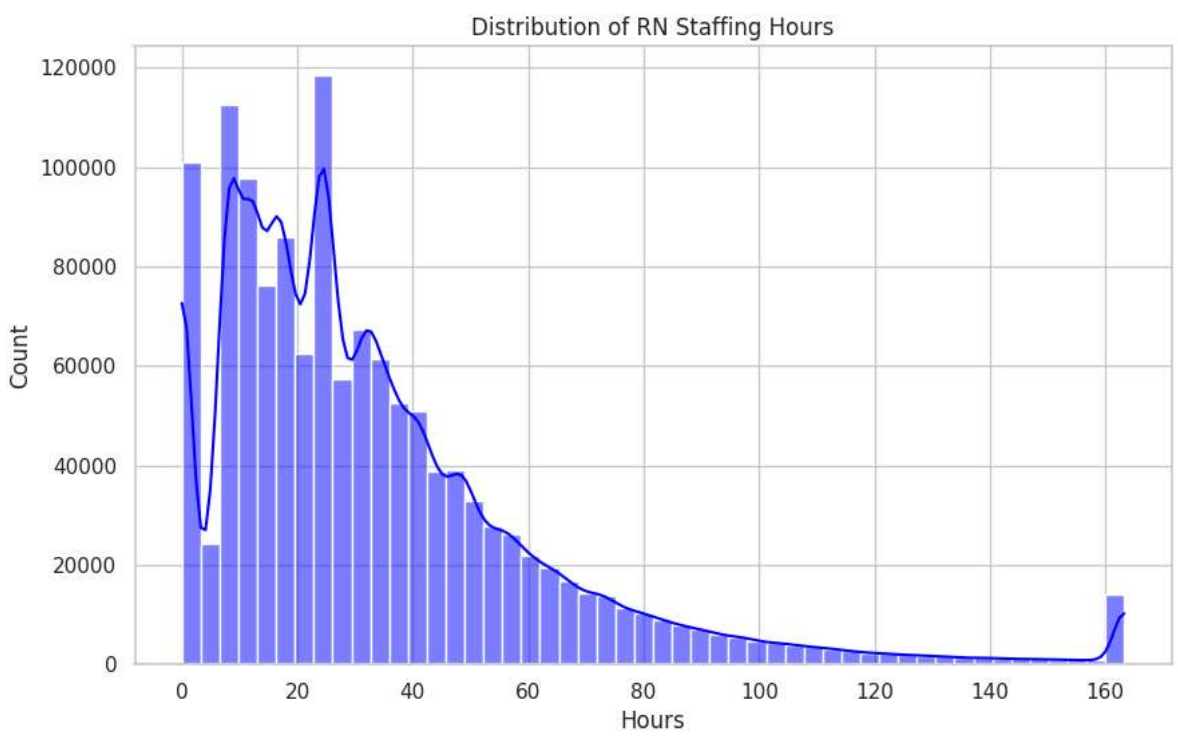
```
            'Facilities with Rating >= 4',
            'Percentage of High Performers (Rating >= 4)',
            'Facilities with Rating <= 2',
            'Percentage of Low Performers (Rating <= 2)'
        ],
        'Value': [
            merged_df['qm_rating'].count(),
            round(merged_df['qm_rating'].mean(), 2),
            merged_df['qm_rating'].median(),
            merged_df['qm_rating'].mode()[0],
            round(merged_df['qm_rating'].std(), 2),
            merged_df['qm_rating'].min(),
            merged_df['qm_rating'].max(),
            len(merged_df[merged_df['qm_rating'] >= 4]),
            f"{(len(merged_df[merged_df['qm_rating'] >= 4]) / len(merged_df) * 100):.2f}%",
            len(merged_df[merged_df['qm_rating'] <= 2]),
            f"{(len(merged_df[merged_df['qm_rating'] <= 2]) / len(merged_df) * 100):.2f}%"
        ]
    })

    # Display results
    print("Distribution of Quality Ratings:")
    print("-" * 60)
    print(final_stats.to_string(index=False))
    print("\nDetailed Analysis:")
    print("-" * 60)
    print(detailed_analysis.to_string(index=False))

    # Optional: Save results to CSV
    final_stats.to_csv('quality_ratings_distribution.csv', index=False)
    detailed_analysis.to_csv('quality_ratings_analysis.csv', index=False)

    # Calculate quartile statistics
    quartile_stats = pd.DataFrame({
        'Metric': [
            'First Quartile (Q1)',
            'Second Quartile (Q2/Median)',
            'Third Quartile (Q3)',
            'Interquartile Range (IQR)'
        ],
        'Value': [
            merged_df['qm_rating'].quantile(0.25),
            merged_df['qm_rating'].quantile(0.50),
            merged_df['qm_rating'].quantile(0.75),
            merged_df['qm_rating'].quantile(0.75) - merged_df['qm_rating'].quantile(0.25)
        ]
    })

    print("\nQuartile Statistics:")
    print("-" * 60)
    print(quartile_stats.to_string(index=False))

    # Created/Modified files during execution:
    print("\nFiles created:")
    print("quality_ratings_distribution.csv")
    print("quality_ratings_analysis.csv")
```

```
Distribution of Quality Ratings:
------------------------------------------------------------
            Rating          Count Percentage Cumulative_Percentage
               1.0         117838      8.77%                 8.77%
               2.0         207956     15.48%                24.26%
               3.0         268719     20.01%                44.27%
               4.0         447841     33.35%                77.62%
               5.0         300622     22.38%                100.0%
Summary Statistics Total: 1342976  Mode: 4.0     Mean Rating: 3.45

Detailed Analysis:
------------------------------------------------------------
                                   Metric   Value
                         Total Facilities 1342976
                    Average (Mean) Rating    3.45
                           Median Rating     4.0
                Mode (Most Common) Rating     4.0
                       Standard Deviation    1.24
                           Minimum Rating     1.0
                           Maximum Rating     5.0
              Facilities with Rating >= 4  748463
Percentage of High Performers (Rating >= 4)  55.73%
              Facilities with Rating <= 2  325794
 Percentage of Low Performers (Rating <= 2)  24.26%

Quartile Statistics:
------------------------------------------------------------
                    Metric  Value
```

```
         First Quartile (Q1)    3.0
Second Quartile (Q2/Median)    4.0
         Third Quartile (Q3)    4.0
  Interquartile Range (IQR)    1.0

Files created:
quality_ratings_distribution.csv
quality_ratings_analysis.csv
```

Start coding or generate with AI.

```
# Set plot style
sns.set(style="whitegrid")

# Visualization 1: Distribution of Nurse Staffing Hours
plt.figure(figsize=(10, 6))
sns.histplot(staffing_df['hrs_rn'], bins=50, kde=True, color='blue')
plt.title("Distribution of RN Staffing Hours")
plt.xlabel("Hours")
plt.ylabel("Count")
plt.show()
```



Purpose

This histogram represents the distribution of registered nurse (RN) staffing hours, combining frequency counts with a kernel density estimate (KDE) overlay. It helps healthcare administrators identify patterns, trends, and anomalies in RN staffing levels, supporting data-driven decisions for workforce optimization and resource allocation.

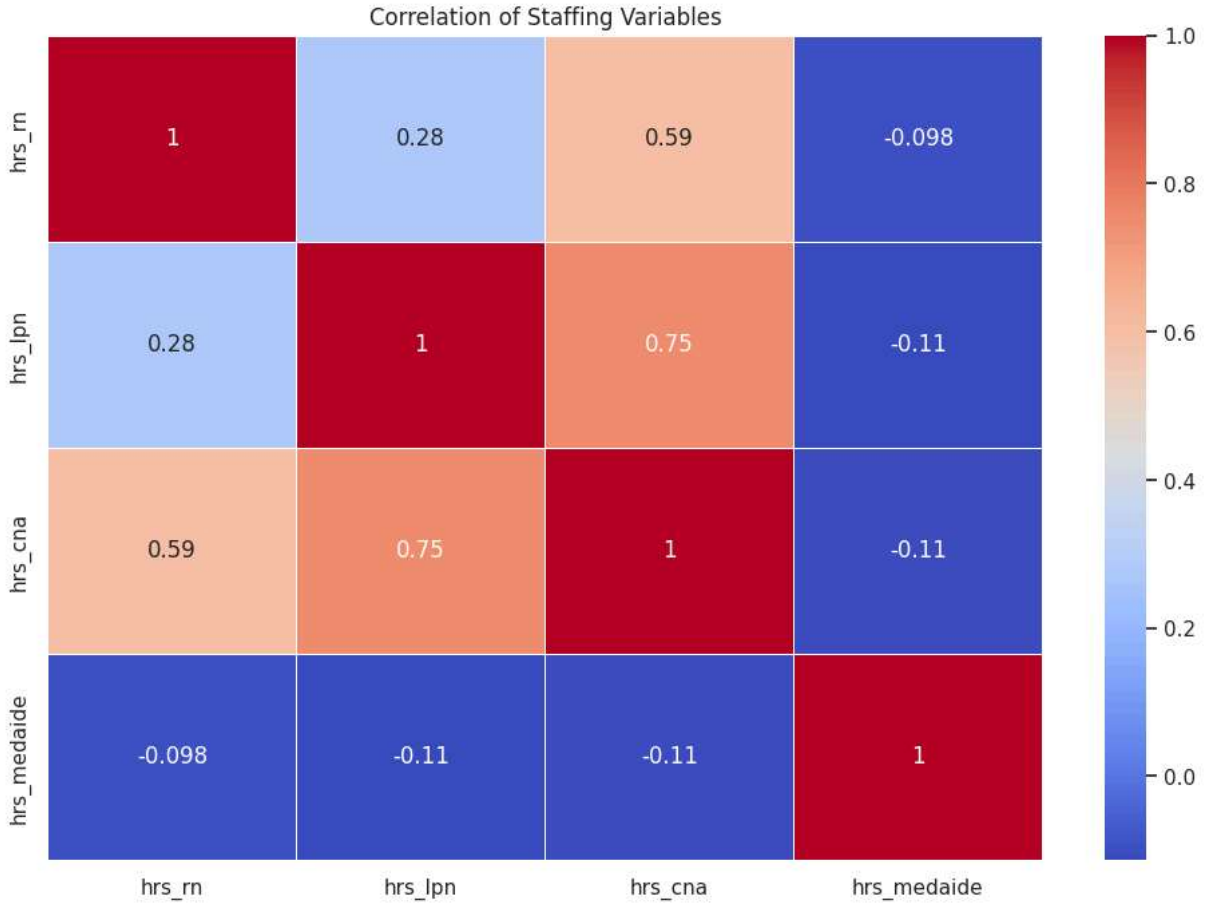
```
# tabular

import pandas as pd
import numpy as np

# Calculate basic distribution statistics
basic_stats = pd.DataFrame({
    'Metric': [
        'Count',
        'Mean Hours',
        'Median Hours',
        'Standard Deviation',
        'Minimum Hours',
        'Maximum Hours',
        'Skewness',
        'Kurtosis'
    ],
```

```
        'Value': [
            staffing_df['hrs_rn'].count(),
            staffing_df['hrs_rn'].mean(),
            staffing_df['hrs_rn'].median(),
            staffing_df['hrs_rn'].std(),
            staffing_df['hrs_rn'].min(),
            staffing_df['hrs_rn'].max(),
            staffing_df['hrs_rn'].skew(),
            staffing_df['hrs_rn'].kurtosis()
        ]
    })

    # Calculate percentiles
    percentiles = [10, 25, 50, 75, 90]
    percentile_stats = pd.DataFrame({
        'Metric': [f'{p}th Percentile' for p in percentiles],
        'Value': [staffing_df['hrs_rn'].quantile(p/100) for p in percentiles]
    })

    # Calculate frequency distribution
    bin_edges = np.histogram_bin_edges(staffing_df['hrs_rn'].dropna(), bins=10)
    hist, bins = np.histogram(staffing_df['hrs_rn'].dropna(), bins=bin_edges)
    bin_centers = (bins[:-1] + bins[1:]) / 2

    frequency_dist = pd.DataFrame({
        'Bin Range': [f'{bins[i]:.1f} - {bins[i+1]:.1f}' for i in range(len(bins)-1)],
        'Frequency': hist,
        'Percentage': (hist/len(staffing_df['hrs_rn'].dropna()) * 100)
    })

    # Calculate IQR and outlier boundaries
    Q1 = staffing_df['hrs_rn'].quantile(0.25)
    Q3 = staffing_df['hrs_rn'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = staffing_df['hrs_rn'][(staffing_df['hrs_rn'] < lower_bound) | (staffing_df['hrs_rn'] > upper_bound)]

    outlier_stats = pd.DataFrame({
        'Metric': [
            'IQR',
            'Lower Bound',
            'Upper Bound',
            'Number of Outliers',
            'Percentage of Outliers'
        ],
        'Value': [
            IQR,
            lower_bound,
            upper_bound,
            len(outliers),
            (len(outliers)/len(staffing_df['hrs_rn']) * 100)
        ]
    })

    # Round all numeric values to 2 decimal places
    basic_stats['Value'] = basic_stats['Value'].round(2)
    percentile_stats['Value'] = percentile_stats['Value'].round(2)
    frequency_dist['Frequency'] = frequency_dist['Frequency'].round(2)
    frequency_dist['Percentage'] = frequency_dist['Percentage'].round(2)
    outlier_stats['Value'] = outlier_stats['Value'].round(2)

    # Print results
    print("Basic Statistics:")
    print("-" * 50)
    print(basic_stats.to_string(index=False))
    print("\nPercentile Analysis:")
    print("-" * 50)
    print(percentile_stats.to_string(index=False))
    print("\nFrequency Distribution:")
    print("-" * 50)
    print(frequency_dist.to_string(index=False))
    print("\nOutlier Analysis:")
    print("-" * 50)
    print(outlier_stats.to_string(index=False))

    # Optional: Save results to CSV
    basic_stats.to_csv('rn_hours_basic_stats.csv', index=False)
    percentile_stats.to_csv('rn_hours_percentiles.csv', index=False)
    frequency_dist.to_csv('rn_hours_frequency_dist.csv', index=False)
    outlier_stats.to_csv('rn_hours_outlier_stats.csv', index=False)
```

```
# Created/Modified files during execution:
print("\nFiles created:")
print("rn_hours_basic_stats.csv")
print("rn_hours_percentiles.csv")
print("rn_hours_frequency_dist.csv")
print("rn_hours_outlier_stats.csv")
```

```
Basic Statistics:
--------------------------------------------------
            Metric      Value
             Count 1330966.00
        Mean Hours      33.67
      Median Hours      25.58
Standard Deviation      29.55
     Minimum Hours       0.00
     Maximum Hours     163.25
          Skewness       1.82
          Kurtosis       4.35

Percentile Analysis:
--------------------------------------------------
         Metric  Value
10th Percentile   7.25
25th Percentile  13.00
50th Percentile  25.58
75th Percentile  44.79
90th Percentile  70.00

Frequency Distribution:
--------------------------------------------------
    Bin Range  Frequency  Percentage
   0.0 - 16.3     411962       30.95
  16.3 - 32.6     391455       29.41
  32.6 - 49.0     242944       18.25
  49.0 - 65.3     128173        9.63
  65.3 - 81.6      66441        4.99
  81.6 - 97.9      35433        2.66
 97.9 - 114.3      19703        1.48
114.3 - 130.6      10697        0.80
130.6 - 146.9       6423        0.48
146.9 - 163.2      17735        1.33

Outlier Analysis:
--------------------------------------------------
                Metric    Value
                   IQR    31.79
           Lower Bound   -34.68
           Upper Bound    92.48
    Number of Outliers 64123.00
Percentage of Outliers     4.82

Files created:
rn_hours_basic_stats.csv
rn_hours_percentiles.csv
rn_hours_frequency_dist.csv
rn_hours_outlier_stats.csv
```

```
# Visualization 3: Correlation Heatmap of Staffing Variables
plt.figure(figsize=(12, 8))
staffing_corr = staffing_df[['hrs_rn', 'hrs_lpn', 'hrs_cna', 'hrs_medaide']].corr()
sns.heatmap(staffing_corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation of Staffing Variables")
plt.show()
```



Purpose

This heatmap demonstrates the correlations between different staffing variables (RN, LPN, CNA, and medical aide hours), using color intensity to represent relationship strength. It helps healthcare administrators understand staffing patterns, identify interdependencies between roles, and optimize workforce composition for balanced and efficient care delivery.

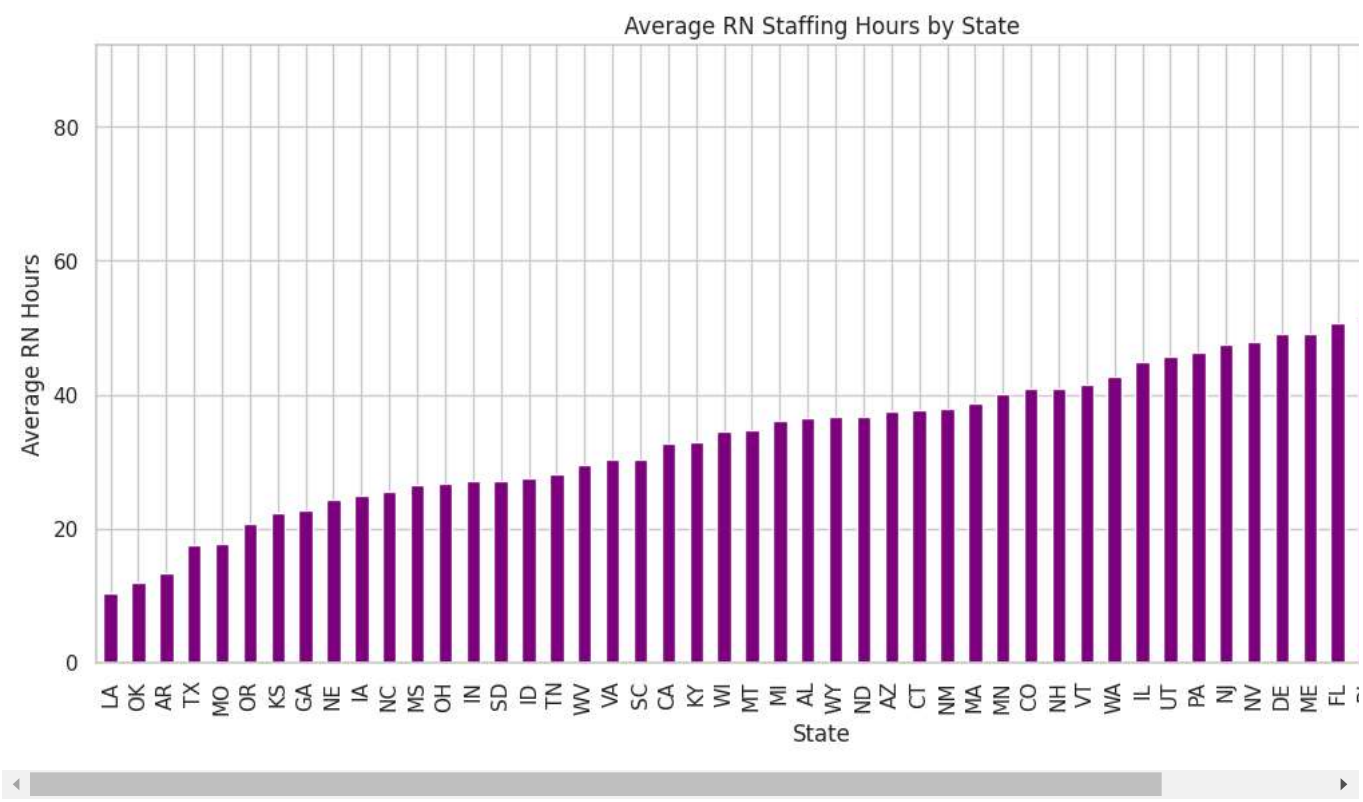
```
# tabular

import pandas as pd
import numpy as np
from scipy import stats

# Calculate correlation matrix
staffing_vars = ['hrs_rn', 'hrs_lpn', 'hrs_cna', 'hrs_medaide']
correlation_matrix = staffing_df[staffing_vars].corr()

# Calculate p-values for correlations
def calculate_pvalues(df):
    df = df[staffing_vars]
    p_values = pd.DataFrame(np.zeros_like(correlation_matrix),
                            columns=staffing_vars,
                            index=staffing_vars)
    for i in staffing_vars:
        for j in staffing_vars:
            correlation, p_value = stats.pearsonr(df[i].dropna(), df[j].dropna())
            p_values.loc[i,j] = p_value
    return p_values

p_values = calculate_pvalues(staffing_df)

# Create detailed correlation analysis
correlation_rows = []
for i in range(len(staffing_vars)):
    for j in range(i+1, len(staffing_vars)):
        var1 = staffing_vars[i]
        var2 = staffing_vars[j]
        corr = correlation_matrix.loc[var1, var2]
        p_val = p_values.loc[var1, var2]

        # Determine correlation strength
        if abs(corr) >= 0.7:
            strength = 'Strong'
```

```
            strength = 'Strong'
        elif abs(corr) >= 0.3:
            strength = 'Moderate'
        else:
            strength = 'Weak'

        correlation_rows.append({
            'Variable 1': var1,
            'Variable 2': var2,
            'Correlation': round(corr, 3),
            'P-value': round(p_val, 4),
            'Strength': strength
        })

correlation_analysis = pd.DataFrame(correlation_rows)

# Calculate summary statistics for each variable
summary_stats = pd.DataFrame()
for var in staffing_vars:
    summary_stats[var] = [
        staffing_df[var].mean(),
        staffing_df[var].std(),
        staffing_df[var].median(),
        staffing_df[var].skew(),
        staffing_df[var].kurtosis()
    ]
summary_stats.index = ['Mean', 'Std Dev', 'Median', 'Skewness', 'Kurtosis']

# Print results
print("Correlation Matrix:")
print("-" * 80)
print(correlation_matrix.round(3))

print("\nCorrelation Analysis:")
print("-" * 80)
print(correlation_analysis.to_string(index=False))

print("\nVariable Summary Statistics:")
print("-" * 80)
print(summary_stats.round(3))

# Additional analysis: Variance Inflation Factor (VIF)
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calculate_vif(df):
    vif_data = pd.DataFrame()
    vif_data["Variable"] = staffing_vars
    vif_data["VIF"] = [variance_inflation_factor(df[staffing_vars].values, i)
                       for i in range(df[staffing_vars].shape[1])]
    return vif_data

vif_scores = calculate_vif(staffing_df)
print("\nVariance Inflation Factors:")
print("-" * 80)
print(vif_scores.round(3).to_string(index=False))

# Save results to CSV files
correlation_matrix.round(3).to_csv('staffing_correlation_matrix.csv')
correlation_analysis.to_csv('staffing_correlation_analysis.csv', index=False)
summary_stats.round(3).to_csv('staffing_summary_stats.csv')
vif_scores.round(3).to_csv('staffing_vif_scores.csv', index=False)

# Created/Modified files during execution:
print("\nFiles created:")
print("staffing_correlation_matrix.csv")
print("staffing_correlation_analysis.csv")
print("staffing_summary_stats.csv")
print("staffing_vif_scores.csv")

# Print key findings
print("\nKey Findings:")
print("-" * 80)
# Print strongest correlations
strongest_corr = correlation_analysis[correlation_analysis['Strength'] == 'Strong']
if not strongest_corr.empty:
    print("Strongest correlations found:")
    print(strongest_corr.to_string(index=False))
else:
    print("No strong correlations found")

# Print significant relationships
significant_corr = correlation_analysis[correlation_analysis['P-value'] < 0.05]
print("\nStatistically significant relationships (p < 0.05):")
```

```
print(significant_corr.to_string(index=False))
```

```
Correlation Matrix:
--------------------------------------------------------------------------------
             hrs_rn  hrs_lpn  hrs_cna  hrs_medaide
hrs_rn        1.000    0.283    0.594       -0.098
hrs_lpn       0.283    1.000    0.753       -0.112
hrs_cna       0.594    0.753    1.000       -0.107
hrs_medaide  -0.098   -0.112   -0.107        1.000

Correlation Analysis:
--------------------------------------------------------------------------------
Variable 1  Variable 2  Correlation  P-value Strength
    hrs_rn     hrs_lpn        0.283      0.0     Weak
    hrs_rn     hrs_cna        0.594      0.0 Moderate
    hrs_rn hrs_medaide       -0.098      0.0     Weak
   hrs_lpn     hrs_cna        0.753      0.0   Strong
   hrs_lpn hrs_medaide       -0.112      0.0     Weak
   hrs_cna hrs_medaide       -0.107      0.0     Weak

Variable Summary Statistics:
--------------------------------------------------------------------------------
          hrs_rn  hrs_lpn  hrs_cna  hrs_medaide
Mean      33.674   65.679  169.205        8.210
Std Dev   29.547   45.218  102.696       15.934
Median    25.580   56.920  148.100        0.000
Skewness   1.825    1.184    1.476        2.310
Kurtosis   4.353    1.676    3.037        5.275

Variance Inflation Factors:
--------------------------------------------------------------------------------
   Variable    VIF
     hrs_rn  3.823
    hrs_lpn  7.606
    hrs_cna 12.780
hrs_medaide  1.133

Files created:
staffing_correlation_matrix.csv
staffing_correlation_analysis.csv
staffing_summary_stats.csv
staffing_vif_scores.csv

Key Findings:
--------------------------------------------------------------------------------
Strongest correlations found:
Variable 1 Variable 2  Correlation  P-value Strength
   hrs_lpn    hrs_cna        0.753      0.0   Strong

Statistically significant relationships (p < 0.05):
Variable 1  Variable 2  Correlation  P-value Strength
    hrs_rn     hrs_lpn        0.283      0.0     Weak
    hrs_rn     hrs_cna        0.594      0.0 Moderate
    hrs_rn hrs_medaide       -0.098      0.0     Weak
   hrs_lpn     hrs_cna        0.753      0.0   Strong
   hrs_lpn hrs_medaide       -0.112      0.0     Weak
   hrs_cna hrs_medaide       -0.107      0.0     Weak
```

```
# Visualization 4: Average Staffing Hours by State
plt.figure(figsize=(14, 6))
staffing_state_avg = staffing_df.groupby('state')['hrs_rn'].mean().sort_values()
staffing_state_avg.plot(kind='bar', color='purple')
plt.title("Average RN Staffing Hours by State")
plt.xlabel("State")
plt.ylabel("Average RN Hours")
plt.xticks(rotation=90)
plt.show()
```



Purpose

This bar chart presents the average registered nurse (RN) staffing hours across different states, arranged in ascending order. It helps healthcare administrators compare regional staffing patterns, identify disparities in workforce distribution, and make informed decisions about resource allocation and staffing strategies across different locations.

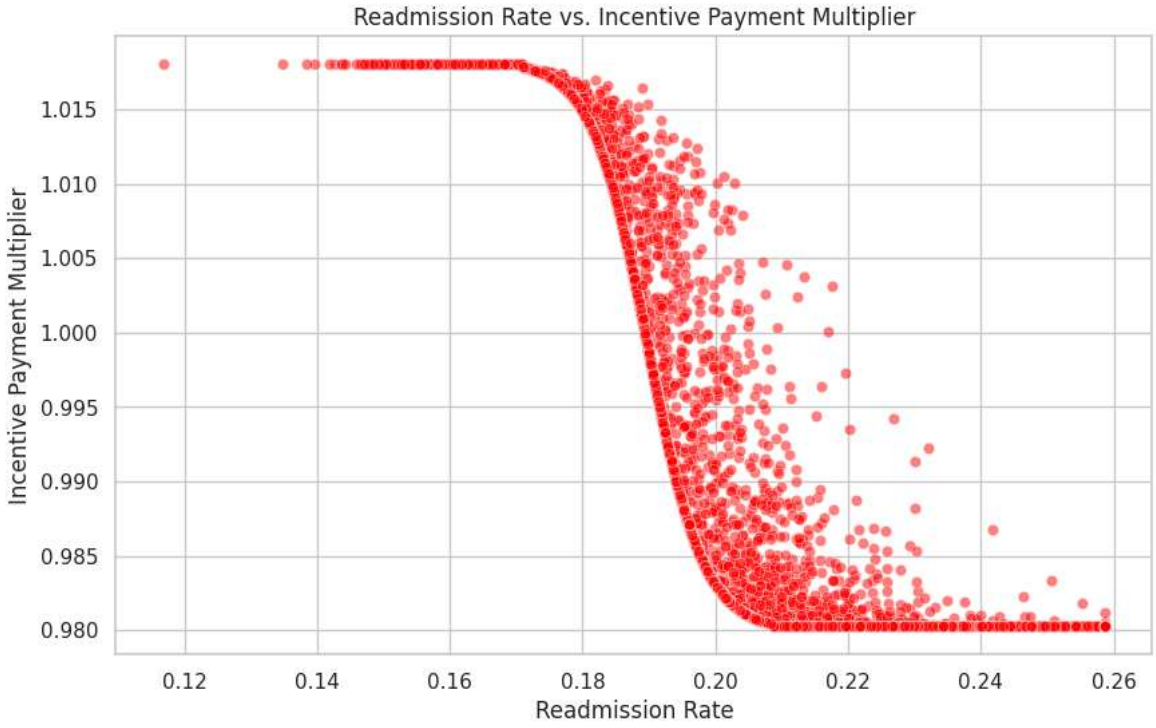
```
# tabular results

import pandas as pd
import numpy as np
from scipy import stats

# Calculate comprehensive state-level statistics
state_stats = staffing_df.groupby('state').agg({
    'hrs_rn': [
        ('Count', 'count'),
        ('Mean', 'mean'),
        ('Median', 'median'),
        ('Std Dev', 'std'),
        ('Min', 'min'),
        ('Max', 'max'),
        ('25th Percentile', lambda x: x.quantile(0.25)),
        ('75th Percentile', lambda x: x.quantile(0.75))
    ]
}).round(2)

# Flatten column names
state_stats.columns = state_stats.columns.get_level_values(1)

# Calculate additional metrics
state_stats['IQR'] = state_stats['75th Percentile'] - state_stats['25th Percentile']
state_stats['CV'] = (state_stats['Std Dev'] / state_stats['Mean'] * 100).round(2)  # Coefficient of Variation

# Sort by mean hours (descending)
state_stats_sorted = state_stats.sort_values('Mean', ascending=False)

# Calculate national statistics for comparison
national_stats = pd.DataFrame({
    'Statistic': [
        'National Mean',
        'National Median',
        'National Std Dev',
        'National Min',
        'National Max',
        'Number of States',
        'States Above National Mean',
        'States Below National Mean'
```

```
        ],
        'Value': [
            staffing_df['hrs_rn'].mean(),
            staffing_df['hrs_rn'].median(),
            staffing_df['hrs_rn'].std(),
            staffing_df['hrs_rn'].min(),
            staffing_df['hrs_rn'].max(),
            len(state_stats),
            sum(state_stats['Mean'] > staffing_df['hrs_rn'].mean()),
            sum(state_stats['Mean'] < staffing_df['hrs_rn'].mean())
        ]
    }).round(2)

    # Calculate quartile rankings
    state_stats_sorted['Quartile'] = pd.qcut(state_stats_sorted['Mean'],
                                             q=4,
                                             labels=['Bottom 25%', 'Lower Middle', 'Upper Middle', 'Top 25%'])

    # Calculate relative position metrics
    mean_hrs = staffing_df['hrs_rn'].mean()
    state_stats_sorted['Percent Diff from National Mean'] = (
        ((state_stats_sorted['Mean'] - mean_hrs) / mean_hrs) * 100
    ).round(2)

    # Print results
    print("RN Staffing Hours by State Analysis")
    print("-" * 80)
    print("\nState-Level Statistics (Sorted by Mean Hours):")
    print(state_stats_sorted.to_string())

    print("\nNational Statistics:")
    print("-" * 80)
    print(national_stats.to_string(index=False))

    # Create summary of top and bottom states
    top_5_states = state_stats_sorted.head(5)
    bottom_5_states = state_stats_sorted.tail(5)

    print("\nTop 5 States by Average RN Hours:")
    print("-" * 80)
    print(top_5_states[['Mean', 'Median', 'Std Dev', 'Count']].to_string())

    print("\nBottom 5 States by Average RN Hours:")
    print("-" * 80)
    print(bottom_5_states[['Mean', 'Median', 'Std Dev', 'Count']].to_string())

    # Calculate distribution statistics by quartile
    quartile_analysis = state_stats_sorted.groupby('Quartile').agg({
        'Mean': ['count', 'mean'],
        'Count': 'sum'
    }).round(2)

    print("\nQuartile Analysis:")
    print("-" * 80)
    print(quartile_analysis.to_string())

    # Save results to CSV files
    state_stats_sorted.to_csv('rn_hours_by_state_detailed.csv')
    national_stats.to_csv('rn_hours_national_stats.csv', index=False)
    quartile_analysis.to_csv('rn_hours_quartile_analysis.csv')

    # Created/Modified files during execution:
    print("\nFiles created:")
    print("rn_hours_by_state_detailed.csv")
    print("rn_hours_national_stats.csv")
    print("rn_hours_quartile_analysis.csv")

    # Optional: Statistical tests
    # ANOVA test to check if there are significant differences between states
    states_list = [group for _, group in staffing_df.groupby('state')['hrs_rn']]
    f_statistic, p_value = stats.f_oneway(*states_list)

    print("\nStatistical Analysis:")
    print("-" * 80)
    print(f"One-way ANOVA test results:")
    print(f"F-statistic: {f_statistic:.2f}")
    print(f"P-value: {p_value:.4f}")
    print(f"Significant differences between states: {'Yes' if p_value < 0.05 else 'No'}")

    # Calculate variation metrics
    variation_stats = pd.DataFrame({
        'Metric': [
```

```
            'Between-State Variation',
            'Within-State Variation',
            'Intraclass Correlation'
        ],
        'Value': [
            state_stats_sorted['Mean'].std(),
            state_stats_sorted['Std Dev'].mean(),
            state_stats_sorted['Mean'].std() / (state_stats_sorted['Mean'].std() + state_stats_sorted['Std Dev'].mean())
        ]
    }).round(3)

    print("\nVariation Analysis:")
    print("-" * 80)
    print(variation_stats.to_string(index=False))
```

```
                National Mean  33.67
              National Median  25.58
             National Std Dev  29.55
                 National Min   0.00
                 National Max 163.25
             Number of States  52.00
      States Above National Mean  30.00
      States Below National Mean  22.00

      Top 5 States by Average RN Hours:
      --------------------------------------------------------------------------------
              Mean  Median  Std Dev  Count
      state
      DC     87.90   82.75    48.70   1547
      HI     84.44   75.50    46.90   3731
      NY     67.78   53.50    48.19  54691
      PR     66.71   66.50    13.77    455
      AK     59.05   59.75    30.89   1365

      Bottom 5 States by Average RN Hours:
      --------------------------------------------------------------------------------
              Mean  Median  Std Dev   Count
      state
      MO     17.64   14.25    15.23   44226
      TX     17.53   14.21    15.47  105924
      AR     13.35   10.25    13.81   19747
      OK     11.95    8.50    12.17   25116
      LA     10.28    8.08    12.06   23751

      Quartile Analysis:
      --------------------------------------------------------------------------------
                   Mean          Count
                  count   mean     sum
      Quartile
      Bottom 25%      13  20.34  480753
      Lower Middle    13  31.33  365092
      Upper Middle    13  40.14  224406
      Top 25%         13  58.52  260715

      Files created:
      rn_hours_by_state_detailed.csv
      rn_hours_national_stats.csv
      rn_hours_quartile_analysis.csv

      Statistical Analysis:
      --------------------------------------------------------------------------------
      One-way ANOVA test results:
      F-statistic: 6738.68
      P-value: 0.0000
      Significant differences between states: Yes

      Variation Analysis:
      --------------------------------------------------------------------------------
                      Metric  Value
      Between-State Variation 16.077
       Within-State Variation 25.875
       Intraclass Correlation  0.383
```

```
# Visualization 6: Readmission Rate vs. Incentive Payment Multiplier
plt.figure(figsize=(10, 6))
sns.scatterplot(x=performance_df['performance_period:_fy_2023_risk-standardized_readmission_rate'],
                y=performance_df['incentive_payment_multiplier'], alpha=0.5, color='red')
plt.title("Readmission Rate vs. Incentive Payment Multiplier")
plt.xlabel("Readmission Rate")
plt.ylabel("Incentive Payment Multiplier")
plt.show()
```



Purpose

This scatter plot displays the relationship between facility readmission rates and their incentive payment multipliers. It helps healthcare administrators understand how readmission performance impacts financial incentives, identify facilities that need improvement, and develop strategies to reduce readmissions while maximizing reimbursement opportunities.

```
# tabular results

import pandas as pd
import numpy as np
from scipy import stats
from sklearn.linear_model import LinearRegression

# For shorter column names in code
readm_col = 'performance_period:_fy_2023_risk-standardized_readmission_rate'
ipm_col = 'incentive_payment_multiplier'

# Calculate correlations and p-values separately
pearson_corr, pearson_p = stats.pearsonr(performance_df[readm_col], performance_df[ipm_col])
spearman_corr, spearman_p = stats.spearmanr(performance_df[readm_col], performance_df[ipm_col])

# Basic correlation analysis
correlation_stats = pd.DataFrame({
    'Metric': [
        'Pearson Correlation',
        'Spearman Correlation',
        'P-value (Pearson)',
        'P-value (Spearman)'
    ],
    'Value': [
        pearson_corr,
        spearman_corr,
        pearson_p,
        spearman_p
    ]
}).round(4)

# Calculate summary statistics for both variables
summary_stats = pd.DataFrame({
    'Statistic': ['Count', 'Mean', 'Median', 'Std Dev', 'Min', 'Max', 'Skewness', 'Kurtosis'],
    'Readmission Rate': [
        performance_df[readm_col].count(),
        performance_df[readm_col].mean(),
        performance_df[readm_col].median(),
        performance_df[readm_col].std(),
        performance_df[readm_col].min(),
        performance_df[readm_col].max(),
        performance_df[readm_col].skew(),
```

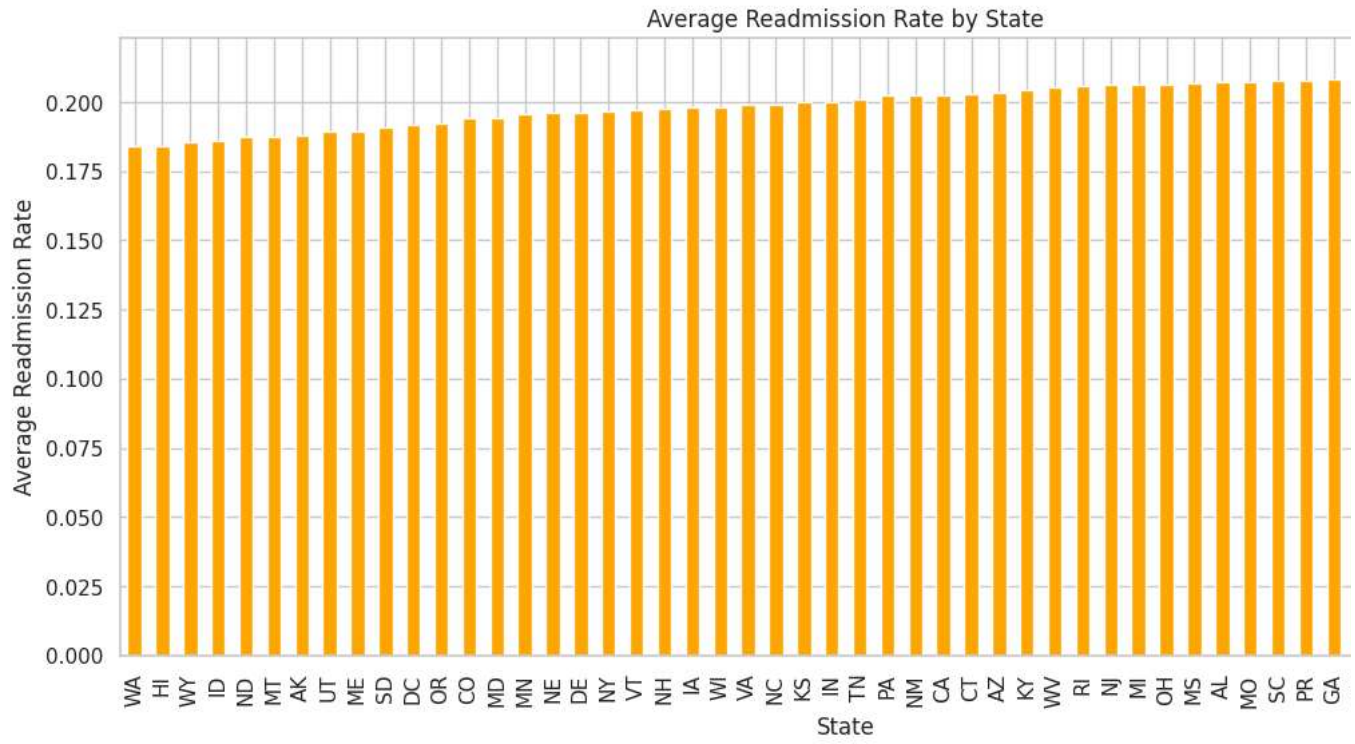
```
            performance_df[readm_col].kurtosis()
        ],
        'Incentive Payment Multiplier': [
            performance_df[ipm_col].count(),
            performance_df[ipm_col].mean(),
            performance_df[ipm_col].median(),
            performance_df[ipm_col].std(),
            performance_df[ipm_col].min(),
            performance_df[ipm_col].max(),
            performance_df[ipm_col].skew(),
            performance_df[ipm_col].kurtosis()
        ]
    }).round(4)

    # Calculate quartile analysis
    quartile_stats = pd.DataFrame({
        'Metric': ['25th Percentile', '50th Percentile', '75th Percentile', 'IQR'],
        'Readmission Rate': [
            performance_df[readm_col].quantile(0.25),
            performance_df[readm_col].quantile(0.50),
            performance_df[readm_col].quantile(0.75),
            performance_df[readm_col].quantile(0.75) - performance_df[readm_col].quantile(0.25)
        ],
        'Incentive Payment Multiplier': [
            performance_df[ipm_col].quantile(0.25),
            performance_df[ipm_col].quantile(0.50),
            performance_df[ipm_col].quantile(0.75),
            performance_df[ipm_col].quantile(0.75) - performance_df[ipm_col].quantile(0.25)
        ]
    }).round(4)

    # Perform linear regression
    X = performance_df[readm_col].values.reshape(-1, 1)
    y = performance_df[ipm_col].values
    model = LinearRegression()
    model.fit(X, y)
    r_squared = model.score(X, y)

    regression_stats = pd.DataFrame({
        'Metric': ['Slope', 'Intercept', 'R-squared'],
        'Value': [
            model.coef_[0],
            model.intercept_,
            r_squared
        ]
    }).round(4)

    # Create bins for readmission rate and analyze average multiplier
    bins = pd.qcut(performance_df[readm_col], q=5)
    binned_analysis = performance_df.groupby(bins)[ipm_col].agg([
        ('Count', 'count'),
        ('Mean Multiplier', 'mean'),
        ('Std Dev Multiplier', 'std')
    ]).round(4)

    # Print results
    print("Correlation Analysis:")
    print("-" * 80)
    print(correlation_stats.to_string(index=False))

    print("\nSummary Statistics:")
    print("-" * 80)
    print(summary_stats.to_string(index=False))

    print("\nQuartile Analysis:")
    print("-" * 80)
    print(quartile_stats.to_string(index=False))

    print("\nRegression Analysis:")
    print("-" * 80)
    print(regression_stats.to_string(index=False))

    print("\nBinned Analysis of Readmission Rates:")
    print("-" * 80)
    print(binned_analysis.to_string())

    # Additional analysis: Calculate extreme cases
    threshold_upper = performance_df[readm_col].quantile(0.95)
    threshold_lower = performance_df[readm_col].quantile(0.05)

    extreme_cases = pd.DataFrame({
        'Category': ['High Readmission Rate (Top 5%)', 'Low Readmission Rate (Bottom 5%)'],
```

```
        'Average Multiplier': [
            performance_df[performance_df[readm_col] >= threshold_upper][ipm_col].mean(),
            performance_df[performance_df[readm_col] <= threshold_lower][ipm_col].mean()
        ],
        'Count': [
            performance_df[performance_df[readm_col] >= threshold_upper].shape[0],
            performance_df[performance_df[readm_col] <= threshold_lower].shape[0]
        ]
    }).round(4)

    print("\nExtreme Cases Analysis:")
    print("-" * 80)
    print(extreme_cases.to_string(index=False))

    # Save results to CSV files
    correlation_stats.to_csv('readmission_multiplier_correlation.csv', index=False)
    summary_stats.to_csv('readmission_multiplier_summary.csv', index=False)
    quartile_stats.to_csv('readmission_multiplier_quartiles.csv', index=False)
    regression_stats.to_csv('readmission_multiplier_regression.csv', index=False)
    binned_analysis.to_csv('readmission_multiplier_binned.csv')
    extreme_cases.to_csv('readmission_multiplier_extreme_cases.csv', index=False)

    # Created/Modified files during execution:
    print("\nFiles created:")
    print("readmission_multiplier_correlation.csv")
    print("readmission_multiplier_summary.csv")
    print("readmission_multiplier_quartiles.csv")
    print("readmission_multiplier_regression.csv")
    print("readmission_multiplier_binned.csv")
    print("readmission_multiplier_extreme_cases.csv")
```

```
Correlation Analysis:
--------------------------------------------------------------------------------
              Metric   Value
 Pearson Correlation -0.8245
Spearman Correlation -0.9511
   P-value (Pearson)  0.0000
  P-value (Spearman)  0.0000

Summary Statistics:
--------------------------------------------------------------------------------
Statistic  Readmission Rate  Incentive Payment Multiplier
    Count        10533.0000                   10533.0000
     Mean            0.2034                       0.9911
   Median            0.2024                       0.9827
  Std Dev            0.0203                       0.0138
      Min            0.1168                       0.9803
      Max            0.2587                       1.0181
 Skewness            0.2684                       0.9074
 Kurtosis            0.0776                      -0.8045

Quartile Analysis:
--------------------------------------------------------------------------------
         Metric  Readmission Rate  Incentive Payment Multiplier
25th Percentile            0.1893                        0.9803
50th Percentile            0.2024                        0.9827
75th Percentile            0.2162                        1.0024
            IQR            0.0269                        0.0222

Regression Analysis:
--------------------------------------------------------------------------------
   Metric   Value
    Slope -0.5591
Intercept  1.1049
R-squared  0.6798

Binned Analysis of Readmission Rates:
--------------------------------------------------------------------------------
                                                                Count  Mean Multiplier  Std Dev Multiplier
performance_period:_fy_2023_risk-standardized_readmission_rate
(0.116, 0.187]                                                   2107           1.0147              0.0034
(0.187, 0.197]                                                   2107           0.9962              0.0071
(0.197, 0.207]                                                   2106           0.9836              0.0044
(0.207, 0.22]                                                    2107           0.9809              0.0022
(0.22, 0.259]                                                    2106           0.9804              0.0009

Extreme Cases Analysis:
--------------------------------------------------------------------------------
                       Category  Average Multiplier  Count
  High Readmission Rate (Top 5%)              0.9803    527
Low Readmission Rate (Bottom 5%)              1.0180    527

Files created:
readmission_multiplier_correlation.csv
readmission_multiplier_summary.csv
readmission_multiplier_quartiles.csv
readmission_multiplier_regression.csv
```

```
readmission_multiplier_pinned.csv
readmission_multiplier_extreme_cases.csv
```

```
# Visualization 8: Readmission Rate by State
plt.figure(figsize=(14, 6))
readmission_state_avg = performance_df.groupby('state')['performance_period:_fy_2023_risk-standardized_readmission_rate'].mean().sort_va
readmission_state_avg.plot(kind='bar', color='orange')
plt.title("Average Readmission Rate by State")
plt.xlabel("State")
plt.ylabel("Average Readmission Rate")
plt.xticks(rotation=90)
plt.show()
```



Purpose

This bar chartreveals the average readmission rates across different states, arranged in ascending order. It helps healthcare administrators compare regional performance in patient care outcomes, identify states with higher readmission challenges, and develop targeted strategies to improve care quality and reduce unnecessary hospital returns.

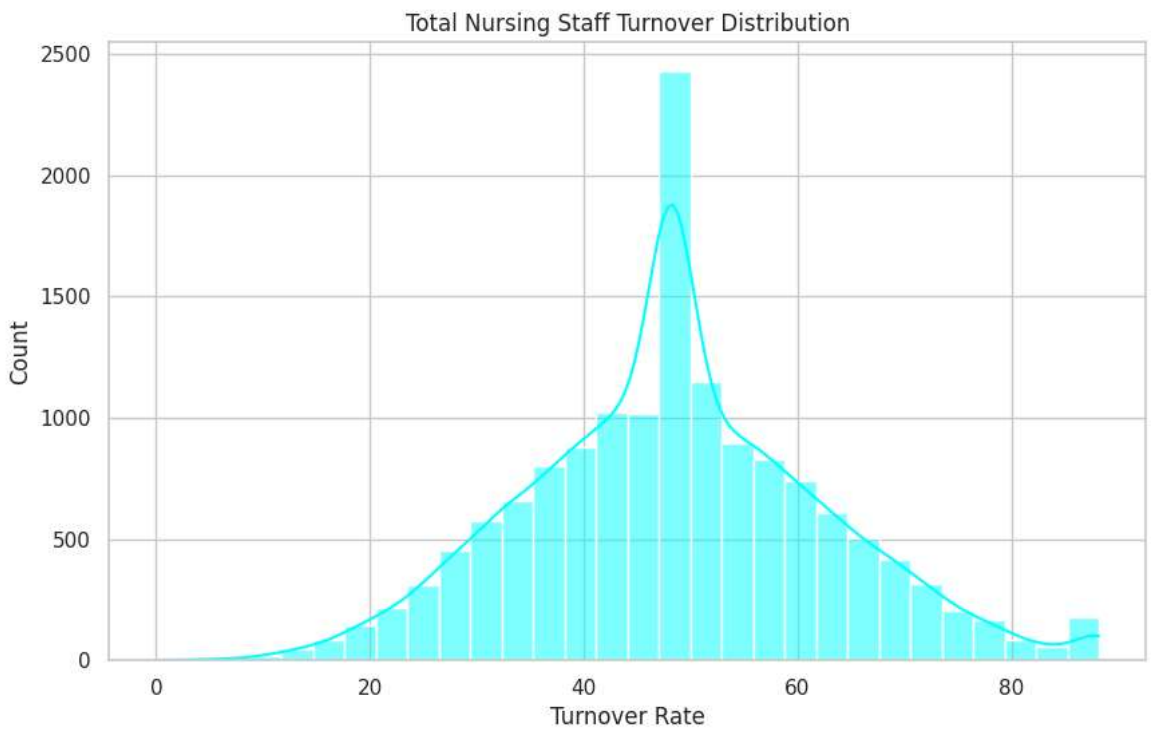
```
# tabular results

import pandas as pd
import numpy as np
from scipy import stats

# For shorter column names in code
readm_col = 'performance_period:_fy_2023_risk-standardized_readmission_rate'

# Calculate comprehensive state-level statistics
state_stats = performance_df.groupby('state').agg({
    readm_col: [
        ('Count', 'count'),
        ('Mean', 'mean'),
        ('Median', 'median'),
        ('Std Dev', 'std'),
        ('Min', 'min'),
        ('Max', 'max'),
        ('25th Percentile', lambda x: x.quantile(0.25)),
        ('75th Percentile', lambda x: x.quantile(0.75))
    ]
}).round(4)

# Flatten column names
state_stats.columns = state_stats.columns.get_level_values(1)

# Calculate additional metrics
```

```
    # Calculate additional metrics
    state_stats['IQR'] = state_stats['75th Percentile'] - state_stats['25th Percentile']
    state_stats['CV'] = (state_stats['Std Dev'] / state_stats['Mean'] * 100).round(2)  # Coefficient of Variation

    # Sort by mean readmission rate (ascending)
    state_stats_sorted = state_stats.sort_values('Mean', ascending=True)

    # Calculate national statistics for comparison
    national_stats = pd.DataFrame({
        'Statistic': [
            'National Mean',
            'National Median',
            'National Std Dev',
            'National Min',
            'National Max',
            'Number of States',
            'States Above National Mean',
            'States Below National Mean'
        ],
        'Value': [
            performance_df[readm_col].mean(),
            performance_df[readm_col].median(),
            performance_df[readm_col].std(),
            performance_df[readm_col].min(),
            performance_df[readm_col].max(),
            len(state_stats),
            sum(state_stats['Mean'] > performance_df[readm_col].mean()),
            sum(state_stats['Mean'] < performance_df[readm_col].mean())
        ]
    }).round(4)

    # Calculate quartile rankings
    state_stats_sorted['Performance Category'] = pd.qcut(
        state_stats_sorted['Mean'],
        q=4,
        labels=['Best (Lowest)', 'Better than Average', 'Worse than Average', 'Worst (Highest)']
    )

    # Calculate relative position metrics
    mean_readm = performance_df[readm_col].mean()
    state_stats_sorted['Percent Diff from National Mean'] = (
        ((state_stats_sorted['Mean'] - mean_readm) / mean_readm) * 100
    ).round(2)

    # Print results
    print("Readmission Rates by State Analysis")
    print("-" * 80)
    print("\nState-Level Statistics (Sorted by Mean Rate):")
    print(state_stats_sorted.to_string())

    print("\nNational Statistics:")
    print("-" * 80)
    print(national_stats.to_string(index=False))

    # Create summary of best and worst performing states
    best_5_states = state_stats_sorted.head(5)
    worst_5_states = state_stats_sorted.tail(5)

    print("\nTop 5 States (Lowest Readmission Rates):")
    print("-" * 80)
    print(best_5_states[['Mean', 'Median', 'Std Dev', 'Count']].to_string())

    print("\nBottom 5 States (Highest Readmission Rates):")
    print("-" * 80)
    print(worst_5_states[['Mean', 'Median', 'Std Dev', 'Count']].to_string())

    # Calculate distribution statistics by performance category
    performance_analysis = state_stats_sorted.groupby('Performance Category').agg({
        'Mean': ['count', 'mean'],
        'Count': 'sum'
    }).round(4)

    print("\nPerformance Category Analysis:")
    print("-" * 80)
    print(performance_analysis.to_string())

    # Statistical significance testing
    # ANOVA test to check if there are significant differences between states
    states_list = [group for _, group in performance_df.groupby('state')[readm_col]]
    f_statistic, p_value = stats.f_oneway(*states_list)

    print("\nStatistical Analysis:")
    print("-" * 80)
```

```
print(f"One-way ANOVA test results:")
print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")
print(f"Significant differences between states: {'Yes' if p_value < 0.05 else 'No'}")

# Calculate variation metrics
variation_stats = pd.DataFrame({
    'Metric': [
        'Between-State Variation',
        'Within-State Variation',
        'Intraclass Correlation'
    ],
    'Value': [
        state_stats_sorted['Mean'].std(),
        state_stats_sorted['Std Dev'].mean(),
        state_stats_sorted['Mean'].std() / (state_stats_sorted['Mean'].std() + state_stats_sorted['Std Dev'].mean())
    ]
}).round(4)

print("\nVariation Analysis:")
print("-" * 80)
print(variation_stats.to_string(index=False))

# Additional analysis: Size-weighted statistics
state_stats_sorted['Weighted Mean'] = (
    state_stats_sorted['Mean'] * state_stats_sorted['Count'] / state_stats_sorted['Count'].sum()
).round(4)

print("\nSize-Weighted Analysis:")
print("-" * 80)
print("Top 5 States by Case-Weighted Readmission Rates:")
print(state_stats_sorted.nlargest(5, 'Weighted Mean')[['Count', 'Mean', 'Weighted Mean']].to_string())

# Save results to CSV files
state_stats_sorted.to_csv('readmission_rates_by_state_detailed.csv')
national_stats.to_csv('readmission_rates_national_stats.csv', index=False)
performance_analysis.to_csv('readmission_rates_performance_analysis.csv')
variation_stats.to_csv('readmission_rates_variation_analysis.csv', index=False)

# Created/Modified files during execution:
print("\nFiles created:")
print("readmission_rates_by_state_detailed.csv")
print("readmission_rates_national_stats.csv")
print("readmission_rates_performance_analysis.csv")
print("readmission_rates_variation_analysis.csv")
```

```
Size-Weighted Analysis:
--------------------------------------------------------------------------------
Top 5 States by Case-Weighted Readmission Rates:
       Count    Mean  Weighted Mean
state
CA       931  0.2024         0.0179
TX       671  0.2084         0.0133
FL       616  0.2125         0.0124
OH       591  0.2065         0.0116
PA       549  0.2022         0.0105

Files created:
readmission_rates_by_state_detailed.csv
readmission_rates_national_stats.csv
readmission_rates_performance_analysis.csv
readmission_rates_variation_analysis.csv
```

```
# Visualization 9: Total Nursing Staff Turnover Distribution
plt.figure(figsize=(10, 6))
sns.histplot(provider_info_df['total_nursing_staff_turnover'], bins=30, kde=True, color='cyan')
plt.title("Total Nursing Staff Turnover Distribution")
plt.xlabel("Turnover Rate")
plt.ylabel("Count")
plt.show()
```



Purpose

This histogram portrays the distribution of total nursing staff turnover rates across facilities, with a kernel density estimate (KDE) overlay. It helps healthcare administrators understand patterns in workforce stability, identify facilities with concerning turnover levels, and develop retention strategies to maintain consistent staffing and care quality.

```
# tabular results

import pandas as pd
import numpy as np
from scipy import stats

# Calculate basic distribution statistics
turnover_stats = pd.DataFrame({
    'Metric': [
        'Count',
        'Mean',
        'Median',
        'Standard Deviation',
        'Minimum',
        'Maximum',
        'Skewness',
```

```
            'Kurtosis',
            'Coefficient of Variation (%)'
        ],
        'Value': [
            provider_info_df['total_nursing_staff_turnover'].count(),
            provider_info_df['total_nursing_staff_turnover'].mean(),
            provider_info_df['total_nursing_staff_turnover'].median(),
            provider_info_df['total_nursing_staff_turnover'].std(),
            provider_info_df['total_nursing_staff_turnover'].min(),
            provider_info_df['total_nursing_staff_turnover'].max(),
            provider_info_df['total_nursing_staff_turnover'].skew(),
            provider_info_df['total_nursing_staff_turnover'].kurtosis(),
            (provider_info_df['total_nursing_staff_turnover'].std() /
             provider_info_df['total_nursing_staff_turnover'].mean() * 100)
        ]
    }).round(4)

    # Calculate percentiles
    percentile_stats = pd.DataFrame({
        'Percentile': ['10th', '25th', '50th', '75th', '90th', 'IQR'],
        'Value': [
            provider_info_df['total_nursing_staff_turnover'].quantile(0.1),
            provider_info_df['total_nursing_staff_turnover'].quantile(0.25),
            provider_info_df['total_nursing_staff_turnover'].quantile(0.5),
            provider_info_df['total_nursing_staff_turnover'].quantile(0.75),
            provider_info_df['total_nursing_staff_turnover'].quantile(0.9),
            provider_info_df['total_nursing_staff_turnover'].quantile(0.75) -
            provider_info_df['total_nursing_staff_turnover'].quantile(0.25)
        ]
    }).round(4)

    # Calculate frequency distribution
    bins = pd.qcut(provider_info_df['total_nursing_staff_turnover'], q=10, duplicates='drop')
    frequency_dist = pd.DataFrame({
        'Bin': bins.value_counts().index.astype(str),
        'Count': bins.value_counts().values,
        'Percentage': (bins.value_counts().values / len(bins) * 100).round(2)
    }).sort_index()

    # Normality tests
    shapiro_stat, shapiro_p = stats.shapiro(provider_info_df['total_nursing_staff_turnover'].dropna())
    ks_stat, ks_p = stats.kstest(provider_info_df['total_nursing_staff_turnover'].dropna(), 'norm')

    normality_tests = pd.DataFrame({
        'Test': ['Shapiro-Wilk', 'Kolmogorov-Smirnov'],
        'Statistic': [shapiro_stat, ks_stat],
        'P-value': [shapiro_p, ks_p],
        'Is Normal': [shapiro_p > 0.05, ks_p > 0.05]
    }).round(4)

    # Calculate outlier statistics
    Q1 = provider_info_df['total_nursing_staff_turnover'].quantile(0.25)
    Q3 = provider_info_df['total_nursing_staff_turnover'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outlier_stats = pd.DataFrame({
        'Metric': [
            'Lower Bound',
            'Upper Bound',
            'Number of Low Outliers',
            'Number of High Outliers',
            'Total Outliers',
            'Percentage Outliers'
        ],
        'Value': [
            lower_bound,
            upper_bound,
            sum(provider_info_df['total_nursing_staff_turnover'] < lower_bound),
            sum(provider_info_df['total_nursing_staff_turnover'] > upper_bound),
            sum((provider_info_df['total_nursing_staff_turnover'] < lower_bound) |
                (provider_info_df['total_nursing_staff_turnover'] > upper_bound)),
            sum((provider_info_df['total_nursing_staff_turnover'] < lower_bound) |
                (provider_info_df['total_nursing_staff_turnover'] > upper_bound)) / len(provider_info_df) * 100
        ]
    }).round(4)

    # Print results
    print("Total Nursing Staff Turnover Analysis")
    print("-" * 80)
    print("\nBasic Distribution Statistics:")
    print(turnover_stats.to_string(index=False))
```

```
print("\nPercentile Analysis:")
print("-" * 80)
print(percentile_stats.to_string(index=False))

print("\nFrequency Distribution:")
print("-" * 80)
print(frequency_dist.to_string(index=False))

print("\nNormality Tests:")
print("-" * 80)
print(normality_tests.to_string(index=False))

print("\nOutlier Analysis:")
print("-" * 80)
print(outlier_stats.to_string(index=False))

# Calculate ranges for categorization
def get_turnover_category(x):
    if x <= Q1:
        return 'Low'
    elif x <= Q3:
        return 'Medium'
    else:
        return 'High'

category_stats = provider_info_df['total_nursing_staff_turnover'].apply(get_turnover_category).value_counts()
category_analysis = pd.DataFrame({
    'Category': category_stats.index,
    'Count': category_stats.values,
    'Percentage': (category_stats.values / len(provider_info_df) * 100).round(2)
})

print("\nTurnover Categories Analysis:")
print("-" * 80)
print(category_analysis.to_string(index=False))

# Save results to CSV files
turnover_stats.to_csv('turnover_distribution_stats.csv', index=False)
percentile_stats.to_csv('turnover_percentiles.csv', index=False)
frequency_dist.to_csv('turnover_frequency_distribution.csv', index=False)
normality_tests.to_csv('turnover_normality_tests.csv', index=False)
outlier_stats.to_csv('turnover_outlier_analysis.csv', index=False)
category_analysis.to_csv('turnover_categories.csv', index=False)

# Created/Modified files during execution:
print("\nFiles created:")
print("turnover_distribution_stats.csv")
print("turnover_percentiles.csv")
print("turnover_frequency_distribution.csv")
print("turnover_normality_tests.csv")
print("turnover_outlier_analysis.csv")
print("turnover_categories.csv")
```

```
                    Skewness     0.1894
                    Kurtosis     0.1538
Coefficient of Variation (%)    29.0044

Percentile Analysis:
--------------------------------------------------------------------------------
Percentile  Value
      10th   30.6
      25th   39.5
      50th   48.2
      75th   57.1
      90th   67.3
       IQR   17.6

Frequency Distribution:
--------------------------------------------------------------------------------
           Bin  Count  Percentage
  (46.2, 48.2]   2201       14.86
  (54.8, 60.0]   1520       10.27
  (37.0, 41.9]   1505       10.16
```

```
      Shapiro-Wilk     0.9925      0.0      False
Kolmogorov-Smirnov     0.9999      0.0      False

Outlier Analysis:
--------------------------------------------------------------------------------
                 Metric    Value
            Lower Bound  13.1000
            Upper Bound  83.5000
 Number of Low Outliers  50.0000
Number of High Outliers 210.0000
         Total Outliers 260.0000
    Percentage Outliers   1.7559

Turnover Categories Analysis:
--------------------------------------------------------------------------------
Category  Count  Percentage
  Medium   7419       50.10
     Low   3704       25.02
    High   3684       24.88

Files created:
turnover_distribution_stats.csv
turnover_percentiles.csv
turnover_frequency_distribution.csv
turnover_normality_tests.csv
turnover_outlier_analysis.csv
turnover_categories.csv
```

Start coding or generate with AI.

```
# Print the column names of merged_df
print(merged_df.columns)
```

```
Index(['provnum', 'provname', 'city', 'state_x', 'county_name', 'county_fips',
       'cy_qtr', 'workdate', 'mdscensus', 'hrs_rndon',
       ...
       'number_of_facility_reported_incidents',
       'number_of_substantiated_complaints', 'number_of_fines',
       'total_amount_of_fines_in_dollars', 'number_of_payment_denials',
       'total_number_of_penalties', 'location', 'latitude', 'longitude',
       'processing_date'],
      dtype='object', length=134)
```

Start coding or generate with AI.

```
# 4. Feature Engineering
# Convert categorical feature 'Ownership Type' into numerical encoding
le = LabelEncoder()
merged_df["ownership_type"] = le.fit_transform(merged_df["ownership_type"])



# Create new features
merged_df["Revenue_per_Bed"] = merged_df["mdscensus"] / merged_df["number_of_certified_beds"]
merged_df["Revenue_per_Nurse_Hour"] = merged_df["mdscensus"] / (merged_df["hrs_rn"] + merged_df["hrs_lpn"] + merged_df["hrs_cna"])
merged_df["Staffing_Efficiency_Ratio"] = merged_df["hrs_cna"] / merged_df["mdscensus"]

# Display the first few rows to verify the new features
print(merged_df.head())
```

```
   provnum             provname     city state_x  county_name  county_fips  \
0  015414  MILLERS MERRY MANOR  CHICAGO      TX  Los Angeles         69.0
1  015414  MILLERS MERRY MANOR  CHICAGO      TX  Los Angeles         69.0
2  015414  MILLERS MERRY MANOR  CHICAGO      TX  Los Angeles         69.0
3  015414  MILLERS MERRY MANOR  CHICAGO      TX  Los Angeles         69.0
4  015414  MILLERS MERRY MANOR  CHICAGO      TX  Los Angeles         69.0

   cy_qtr    workdate  mdscensus  hrs_rndon  ...  \
0  2024Q1  20240215.0       76.0        8.0  ...
1  2024Q1  20240215.0       76.0        8.0  ...
2  2024Q1  20240215.0       76.0        8.0  ...
3  2024Q1  20240215.0       76.0        8.0  ...
4  2024Q1  20240215.0       76.0        8.0  ...

   total_amount_of_fines_in_dollars  number_of_payment_denials  \
0                           23989.0                        0.0
1                               0.0                        0.0
2                               0.0                        0.0
3                               0.0                        0.0
4                               0.0                        0.0

   total_number_of_penalties                                location  \
0                        1.0     701 MONROE STREET NWRUSSELLVILLEAL35653
1                        0.0      260 WEST WALNUT STREETSYLACAUGAAL35150
```

```
2                      0.0        380 WOODS COVE ROADSCOTTSBOROAL35768
3                      0.0   7755 FOURTH AVENUE SOUTHBIRMINGHAMAL35206
4                      0.0  6450 OLD TUSCALOOSA HIGHWAYMC CALLAAL35111

   latitude  longitude  processing_date  Revenue_per_Bed  \
0   34.5149    -87.736         20241101         1.333333
1   33.1637    -86.254         20241101         0.894118
2   34.6611    -86.047         20241101         1.520000
3   33.5595    -86.722         20241101         0.826087
4   33.3221    -87.034         20241101         0.737864

   Revenue_per_Nurse_Hour  Staffing_Efficiency_Ratio
0                0.329575                   1.948684
1                0.329575                   1.948684
2                0.329575                   1.948684
3                0.329575                   1.948684
4                0.329575                   1.948684

[5 rows x 137 columns]
```

```
import numpy as np
import pandas as pd

# First, let's identify the data types of each column
print("Data types of columns:")
print(merged_df.dtypes)

# Separate numeric and non-numeric columns
numeric_columns = merged_df.select_dtypes(include=['float64', 'int64']).columns
object_columns = merged_df.select_dtypes(include=['object']).columns

# Handle numeric columns
for col in numeric_columns:
    merged_df[col] = pd.to_numeric(merged_df[col], errors='coerce')  # Convert to numeric, invalid values become NaN
    merged_df[col] = merged_df[col].fillna(merged_df[col].median())

# Handle categorical/string columns
for col in object_columns:
    merged_df[col] = merged_df[col].fillna(merged_df[col].mode()[0] if not merged_df[col].mode().empty else 'Unknown')

# Recalculate the derived features
# First ensure the required columns are numeric
required_cols = ['mdscensus', 'number_of_certified_beds', 'hrs_rn', 'hrs_lpn', 'hrs_cna']
for col in required_cols:
    merged_df[col] = pd.to_numeric(merged_df[col], errors='coerce')

# Now calculate the features
merged_df["Revenue_per_Bed"] = merged_df["mdscensus"] / merged_df["number_of_certified_beds"]
merged_df["Revenue_per_Nurse_Hour"] = merged_df["mdscensus"] / (merged_df["hrs_rn"] + merged_df["hrs_lpn"] + merged_df["hrs_cna"])
merged_df["Staffing_Efficiency_Ratio"] = merged_df["hrs_cna"] / merged_df["mdscensus"]

# Handle any infinity values
merged_df = merged_df.replace([np.inf, -np.inf], np.nan)

# Fill any remaining NaN values in the calculated columns
calculated_cols = ['Revenue_per_Bed', 'Revenue_per_Nurse_Hour', 'Staffing_Efficiency_Ratio']
for col in calculated_cols:
    merged_df[col] = merged_df[col].fillna(merged_df[col].median())

# Display summary of the cleaning
print("\nVerifying NaN values after cleaning:")
print(merged_df.isnull().sum())

# Display the first few rows
print("\nFirst few rows after cleaning:")
print(merged_df.head())

# Display summary statistics for numeric columns
print("\nSummary statistics for numeric columns:")
print(merged_df[numeric_columns].describe())
```

```
50%     0.000000e+00  7.500000e+00     7.350000e+00     0.000000e+00
75%     0.000000e+00  1.600000e+01     1.600000e+01     0.000000e+00
max     7.500000e+00  6.475000e+01     6.400000e+01     8.000000e+00

             hrs_rn  ...  number_of_substantiated_complaints  number_of_fines  \
count  1.342976e+06  ...                       1.342976e+06     1.342976e+06
mean   3.360199e+01  ...                       5.869970e+00     1.536115e+00
std    2.942417e+01  ...                       9.893519e+00     2.273184e+00
min    0.000000e+00  ...                       0.000000e+00     0.000000e+00
25%    1.307000e+01  ...                       0.000000e+00     0.000000e+00
50%    2.558000e+01  ...                       2.000000e+00     1.000000e+00
75%    4.450000e+01  ...                       7.000000e+00     2.000000e+00
max    1.632500e+02  ...                       5.700000e+01     1.500000e+01

       total_amount_of_fines_in_dollars  number_of_payment_denials  \
count                      1.342976e+06               1.342976e+06
mean                       3.481758e+04               1.958166e-01
std                        7.093400e+04               5.249520e-01
min                        0.000000e+00               0.000000e+00
25%                        0.000000e+00               0.000000e+00
50%                        7.442500e+03               0.000000e+00
75%                        2.973685e+04               0.000000e+00
max                        3.974083e+05               3.000000e+00

       total_number_of_penalties      latitude     longitude  Revenue_per_Bed  \
count               1.342976e+06  1.342976e+06  1.342976e+06     1.342976e+06
mean                1.738717e+00  3.814211e+01 -8.980961e+01     7.945910e-01
std                 2.536465e+00  4.689427e+00  1.307701e+01     2.296667e-01
min                 0.000000e+00  1.348860e+01 -1.652000e+02     0.000000e+00
25%                 0.000000e+00  3.513200e+01 -9.543700e+01     6.728395e-01
50%                 1.000000e+00  3.920420e+01 -8.749800e+01     8.181818e-01
75%                 2.000000e+00  4.133790e+01 -8.151400e+01     9.152542e-01
max                 1.600000e+01  4.768428e+01 -7.092912e+01     1.900000e+01

       Revenue_per_Nurse_Hour  Staffing_Efficiency_Ratio
count            1.342976e+06               1.342976e+06
mean             3.298841e-01               2.078454e+00
std              2.947971e-01               7.014658e-01
min              0.000000e+00               0.000000e+00
25%              2.673415e-01               1.691964e+00
50%              3.115565e-01               2.020968e+00
75%              3.626543e-01               2.403848e+00
max              1.760000e+02               1.471700e+02

[8 rows x 102 columns]
```

Start coding or generate with AI.

Double-click (or enter) to edit

```
# 5. Data Preprocessing
# Define features and target variable
X = merged_df.drop(columns=["mdscensus"])  # Predicting mdscensus (proxy for revenue)
y = merged_df["mdscensus"]
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
import numpy as np

# Replace infinite values with NaN
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Select only numeric columns to calculate the mean and fill NaN values
numeric_cols = X_train.select_dtypes(include=[np.number]).columns

# Fill NaN values with the column mean (only for numeric columns)
X_train[numeric_cols] = X_train[numeric_cols].fillna(X_train[numeric_cols].mean())
X_test[numeric_cols] = X_test[numeric_cols].fillna(X_test[numeric_cols].mean())
```

```
# using few sample dataset
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Sample the data (1% sample size for demonstration)
X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train, y_train, train_size=0.01, random_state=42)
X_test_sampled, _, y_test_sampled, _ = train_test_split(X_test, y_test, train_size=0.01, random_state=42)

# Separate numeric and categorical columns
numeric_cols = X_train_sampled.select_dtypes(include=[np.number]).columns
categorical_cols = X_train_sampled.select_dtypes(exclude=[np.number]).columns

# Create transformers for numerical and categorical data
numeric_transformer = SimpleImputer(strategy='mean')
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Create a preprocessor that applies the right transformations to each type of feature
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Initialize models with reduced complexity
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=20, random_state=42, n_jobs=-1),  # Reduce the number of estimators and use paral
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=50, learning_rate=0.1, random_state=42)  # Reduce the number of estimator
}

# Train and evaluate models
results = {}
for name, model in models.items():
    # Create a pipeline that applies pre-processing and then the model
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])

    pipeline.fit(X_train_sampled, y_train_sampled)  # Train on sampled data
    y_pred = pipeline.predict(X_test_sampled)  # Predict on test data

    # Store performance metrics
    results[name] = {
        "R2 Score": r2_score(y_test_sampled, y_pred),
        "MAE": mean_absolute_error(y_test_sampled, y_pred),
        "MSE": mean_squared_error(y_test_sampled, y_pred)
    }

# Print results
for name, metrics in results.items():
    print(f"{name}: {metrics}")
```

```
Linear Regression: {'R2 Score': 0.07093661968259402, 'MAE': 31.639423011986167, 'MSE': 1832.3096830169693}
Random Forest: {'R2 Score': 0.9930539491471575, 'MAE': 1.9786778398510243, 'MSE': 13.699082867783988}
Gradient Boosting: {'R2 Score': 0.9872650899368187, 'MAE': 3.4963795884735167, 'MSE': 25.115938821251692}
```

```
# Display results
for name, metrics in results.items():
    print(f"Model: {name}")
    print(f"R2 Score: {metrics['R2 Score']:.4f}")
    print(f"MAE: {metrics['MAE']:.2f}")
    print(f"MSE: {metrics['MSE']:.2f}\n")
```

```
Model: Linear Regression
R2 Score: 0.0709
MAE: 31.64
MSE: 1832.31

Model: Random Forest
R2 Score: 0.9931
MAE: 1.98
MSE: 13.70

Model: Gradient Boosting
R2 Score: 0.9873
MAE: 3.50
MSE: 25.12
```

```
import pandas as pd
import numpy as np
```

```
# Train models and extract feature importance
feature_importance_results = {}

for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])

    pipeline.fit(X_train_sampled, y_train_sampled)

    # Extract feature names after transformation
    transformed_feature_names = numeric_cols.tolist() + list(
        pipeline.named_steps['preprocessor'].named_transformers_['cat']
        .named_steps['onehot'].get_feature_names_out(input_features=categorical_cols)
    )

    if hasattr(model, 'feature_importances_'):  # Tree-based models
        feature_importances = model.feature_importances_

    elif hasattr(model, 'coef_'):  # Linear Regression uses coefficients
        feature_importances = np.abs(model.coef_)  # Take absolute values

    else:
        continue  # Skip models that don't have importance metrics

    # Create a DataFrame
    importance_df = pd.DataFrame({'Feature': transformed_feature_names, 'Importance': feature_importances})

    # Sort by importance and select top 10
    importance_df = importance_df.sort_values(by='Importance', ascending=False).head(10)

    feature_importance_results[name] = importance_df

# Display top 10 features for each model
for model_name, importance_df in feature_importance_results.items():
    print(f"\nTop 10 Features - {model_name}:")
    print(importance_df.to_string(index=False))
```

```
Top 10 Features - Linear Regression:
                          Feature  Importance
                         zip_code_y    0.000463
    total_amount_of_fines_in_dollars    0.000068
                          hrs_cna    0.000008
                      hrs_cna_emp    0.000007
            number_of_certified_beds    0.000003
 average_number_of_residents_per_day    0.000003
                          hrs_lpn    0.000003
                      hrs_lpn_emp    0.000002
            provider_ssa_county_code    0.000002
                           hrs_rn    0.000001

Top 10 Features - Random Forest:
                                      Feature  Importance
          average_number_of_residents_per_day    0.871698
                                Revenue_per_Bed    0.108370
                    number_of_certified_beds    0.006900
                                      hrs_cna    0.000873
                                  provnum_015414    0.000664
                 provname_MILLERS MERRY MANOR    0.000539
                           city/town_y_CLINTON    0.000439
       affiliated_entity_name_COMMUNICARE HEALTH    0.000317
 case-mix_lpn_staffing_hours_per_resident_per_day    0.000261
                        Revenue_per_Nurse_Hour    0.000231

Top 10 Features - Gradient Boosting:
                                 Feature  Importance
     average_number_of_residents_per_day    0.715203
                                   hrs_cna    0.161816
                           Revenue_per_Bed    0.092035
                                   hrs_lpn    0.008557
              number_of_certified_beds    0.007313
             Staffing_Efficiency_Ratio    0.005236
                Revenue_per_Nurse_Hour    0.004612
       provider_address_y_1 HEALTHY WAY    0.001843
 cms_certification_number_(ccn)_y_015414    0.000932
                       provnum_015414    0.000651
```

Start coding or generate with AI.

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
```

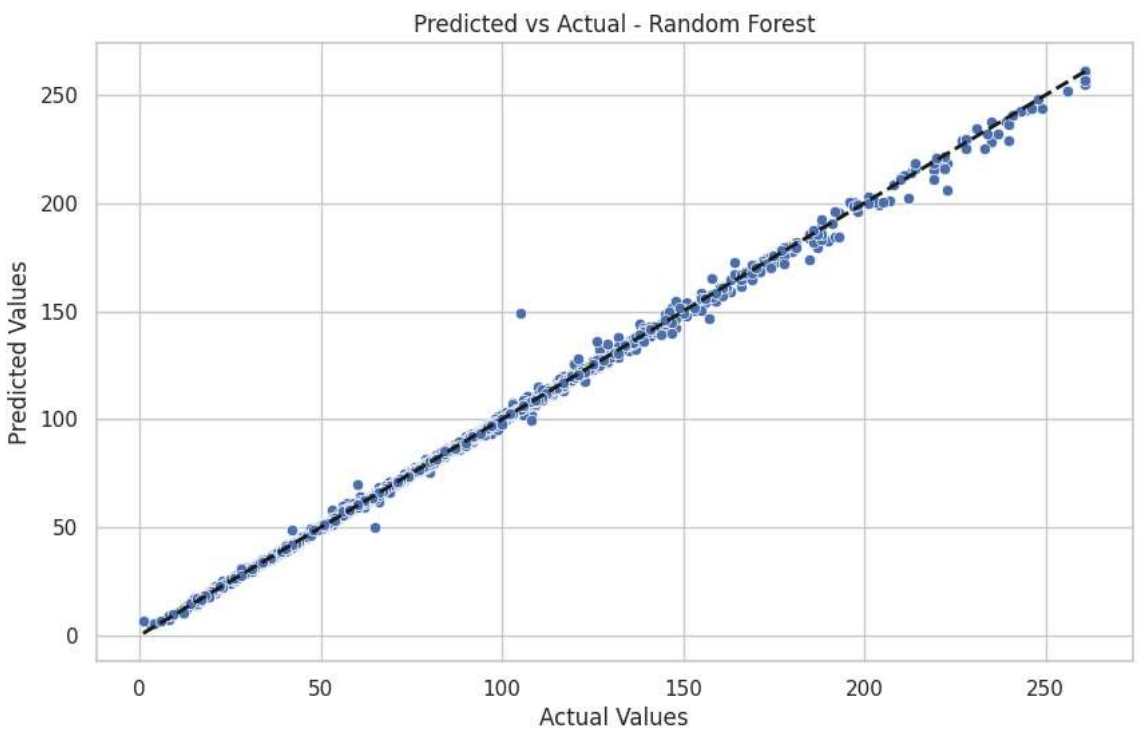
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Sample dataset (1% for fast execution)
X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train, y_train, train_size=0.01, random_state=42)
X_test_sampled, _, y_test_sampled, _ = train_test_split(X_test, y_test, train_size=0.01, random_state=42)

# Selected top features based on previous importance analysis
important_features = ['hrs_cna', 'number_of_certified_beds', 'Revenue_per_Bed', 'Staffing_Efficiency_Ratio',
                      'hrs_lpn', 'Revenue_per_Nurse_Hour']
X_train_sampled = X_train_sampled[important_features]
X_test_sampled = X_test_sampled[important_features]

# Separate numeric and categorical columns
numeric_cols = X_train_sampled.select_dtypes(include=[np.number]).columns
categorical_cols = X_train_sampled.select_dtypes(exclude=[np.number]).columns

# Create transformers for numerical and categorical data
numeric_transformer = SimpleImputer(strategy='mean')
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define models
models = {
    "Linear Regression": LinearRegression(),
    "Random Forest": RandomForestRegressor(n_estimators=20, random_state=42, n_jobs=-1),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=50, learning_rate=0.1, random_state=42)
}

# Train and evaluate models
results = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
    pipeline.fit(X_train_sampled, y_train_sampled)
    y_pred = pipeline.predict(X_test_sampled)
    results[name] = {
        "R2 Score": r2_score(y_test_sampled, y_pred),
        "MAE": mean_absolute_error(y_test_sampled, y_pred),
        "MSE": mean_squared_error(y_test_sampled, y_pred)
    }

# Identify the best model
best_model = max(results, key=lambda k: results[k]["R2 Score"])

# Print results
for name, metrics in results.items():
    print(f"{name}: {metrics}")

print(f"\nBest Model: {best_model} based on R2 Score")
```

```
Linear Regression: {'R2 Score': 0.9753914778516215, 'MAE': 4.159388326881627, 'MSE': 48.53321567986745}
Random Forest: {'R2 Score': 0.9987522945568458, 'MAE': 0.6301675977653629, 'MSE': 2.4607392923649902}
Gradient Boosting: {'R2 Score': 0.991310944446953, 'MAE': 2.831538978857215, 'MSE': 17.1366571575354}

Best Model: Random Forest based on R2 Score
```

Start coding or generate with AI.

```
# predicted vs actual

import matplotlib.pyplot as plt
import seaborn as sns

# Train and evaluate models
results = {}
y_pred_best_model = None
for name, model in models.items():
```

```
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', model)])
    pipeline.fit(X_train_sampled, y_train_sampled)
    y_pred = pipeline.predict(X_test_sampled)

    # Track predictions for the best model
    if name == best_model:
        y_pred_best_model = y_pred

    results[name] = {
        "R2 Score": r2_score(y_test_sampled, y_pred),
        "MAE": mean_absolute_error(y_test_sampled, y_pred),
        "MSE": mean_squared_error(y_test_sampled, y_pred)
    }

# Plot Predicted vs Actual
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test_sampled, y=y_pred_best_model)
plt.title(f"Predicted vs Actual - {best_model}")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.plot([y_test_sampled.min(), y_test_sampled.max()], [y_test_sampled.min(), y_test_sampled.max()], 'k--', lw=2)
plt.show()

# Print results
for name, metrics in results.items():
    print(f"{name}: {metrics}")

print(f"\nBest Model: {best_model} based on R2 Score")
```




```
Linear Regression: {'R2 Score': 0.9753914778516215, 'MAE': 4.159388326881627, 'MSE': 48.53321567986745}
Random Forest: {'R2 Score': 0.9987522945568458, 'MAE': 0.6301675977653629, 'MSE': 2.4607392923649902}
Gradient Boosting: {'R2 Score': 0.991310944446953, 'MAE': 2.831538978857215, 'MSE': 17.1366571575354}

Best Model: Random Forest based on R2 Score
```

Purpose

This scatter plot highlights the relationship between predicted and actual values from the best-performing model, with a diagonal reference line representing perfect predictions. It helps data scientists and administrators evaluate model performance, identify prediction patterns, and assess the reliability of the model's forecasts for decision-making purposes.

```
# tabuar results

import pandas as pd
import numpy as np
from scipy import stats
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error, explained_variance_score
from sklearn.metrics import mean_absolute_percentage_error, median_absolute_error
```

```
# Create DataFrame with actual and predicted values
prediction_df = pd.DataFrame({
    'Actual': y_test_sampled,
    'Predicted': y_pred_best_model,
    'Absolute_Error': np.abs(y_test_sampled - y_pred_best_model),
    'Percentage_Error': np.abs((y_test_sampled - y_pred_best_model) / y_test_sampled) * 100
})

# Calculate basic performance metrics
performance_metrics = pd.DataFrame({
    'Metric': [
        'R-squared (R²)',
        'Adjusted R-squared',
        'Mean Absolute Error (MAE)',
        'Mean Squared Error (MSE)',
        'Root Mean Squared Error (RMSE)',
        'Mean Absolute Percentage Error (MAPE)',
        'Median Absolute Error',
        'Explained Variance Score'
    ],
    'Value': [
        r2_score(y_test_sampled, y_pred_best_model),
        1 - (1 - r2_score(y_test_sampled, y_pred_best_model)) * (len(y_test_sampled) - 1) / (len(y_test_sampled) - X_test_sampled.shape[1
        mean_absolute_error(y_test_sampled, y_pred_best_model),
        mean_squared_error(y_test_sampled, y_pred_best_model),
        np.sqrt(mean_squared_error(y_test_sampled, y_pred_best_model)),
        mean_absolute_percentage_error(y_test_sampled, y_pred_best_model),
        median_absolute_error(y_test_sampled, y_pred_best_model),
        explained_variance_score(y_test_sampled, y_pred_best_model)
    ]
}).round(4)

# Calculate error distribution statistics
error_stats = pd.DataFrame({
    'Metric': [
        'Mean Error',
        'Median Error',
        'Std Dev of Error',
        'Skewness of Error',
        'Kurtosis of Error',
        'Min Error',
        'Max Error'
    ],
    'Value': [
        prediction_df['Absolute_Error'].mean(),
        prediction_df['Absolute_Error'].median(),
        prediction_df['Absolute_Error'].std(),
        prediction_df['Absolute_Error'].skew(),
        prediction_df['Absolute_Error'].kurtosis(),
        prediction_df['Absolute_Error'].min(),
        prediction_df['Absolute_Error'].max()
    ]
}).round(4)

# Calculate percentile statistics
percentile_stats = pd.DataFrame({
    'Percentile': ['25th', '50th', '75th', '90th', '95th', '99th'],
    'Error Value': [
        np.percentile(prediction_df['Absolute_Error'], 25),
        np.percentile(prediction_df['Absolute_Error'], 50),
        np.percentile(prediction_df['Absolute_Error'], 75),
        np.percentile(prediction_df['Absolute_Error'], 90),
        np.percentile(prediction_df['Absolute_Error'], 95),
        np.percentile(prediction_df['Absolute_Error'], 99)
    ]
}).round(4)

# Calculate error ranges
error_ranges = pd.DataFrame({
    'Error Range': [
        '0-5%',
        '5-10%',
        '10-20%',
        '20-30%',
        '>30%'
    ],
    'Count': [
        sum((prediction_df['Percentage_Error'] >= 0) & (prediction_df['Percentage_Error'] < 5)),
        sum((prediction_df['Percentage_Error'] >= 5) & (prediction_df['Percentage_Error'] < 10)),
        sum((prediction_df['Percentage_Error'] >= 10) & (prediction_df['Percentage_Error'] < 20)),
        sum((prediction_df['Percentage_Error'] >= 20) & (prediction_df['Percentage_Error'] < 30)),
        sum(prediction_df['Percentage_Error'] >= 30)
    ]
```

```
    })
    error_ranges['Percentage'] = (error_ranges['Count'] / len(prediction_df) * 100).round(2)

    # Calculate comparison statistics
    comparison_stats = pd.DataFrame({
        'Statistic': ['Mean', 'Median', 'Std Dev', 'Min', 'Max'],
        'Actual': [
            y_test_sampled.mean(),
            y_test_sampled.median(),
            y_test_sampled.std(),
            y_test_sampled.min(),
            y_test_sampled.max()
        ],
        'Predicted': [
            y_pred_best_model.mean(),
            np.median(y_pred_best_model),
            y_pred_best_model.std(),
            y_pred_best_model.min(),
            y_pred_best_model.max()
        ]
    }).round(4)

    # Calculate correlation between actual and predicted
    correlation_stats = pd.DataFrame({
        'Metric': ['Pearson Correlation', 'Spearman Correlation'],
        'Value': [
            stats.pearsonr(y_test_sampled, y_pred_best_model)[0],
            stats.spearmanr(y_test_sampled, y_pred_best_model)[0]
```