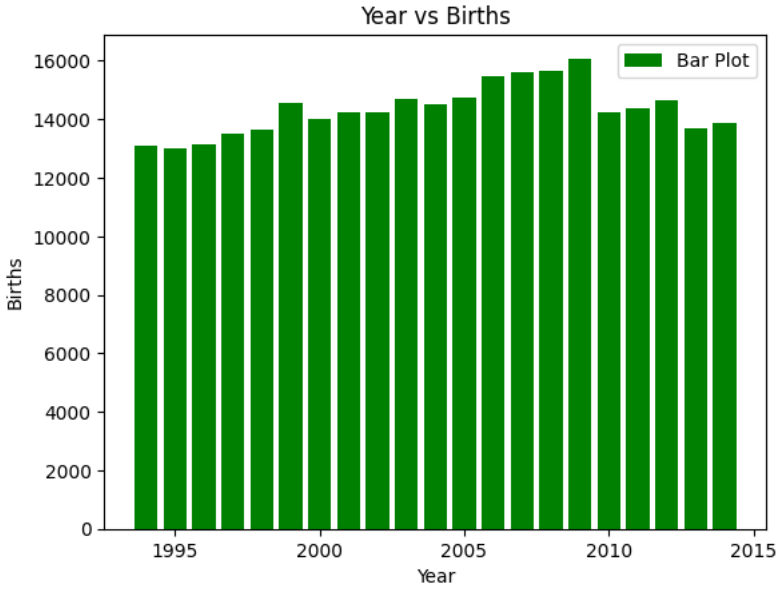
```
import pandas as pd
# Read the two CSV files into DataFrames
file1 = pd.read_csv('/content/US_births_1994-2014_CDC_NCHS.csv')
# Display the first few rows of each DataFrame
print("File 1:")
print(file1.head())
print (file1.tail())
```

```
File 1:
   year  month  date_of_month  day_of_week  births
0  1994      1              1            6    8096
1  1994      1              2            7    7772
2  1994      1              3            1   10142
3  1994      1              4            2   11248
4  1994      1              5            3   11053
      year  month  date_of_month  day_of_week  births
7665  2014     12             27            6    8656
7666  2014     12             28            7    7724
7667  2014     12             29            1   12811
7668  2014     12             30            2   13634
7669  2014     12             31            3   11990
```

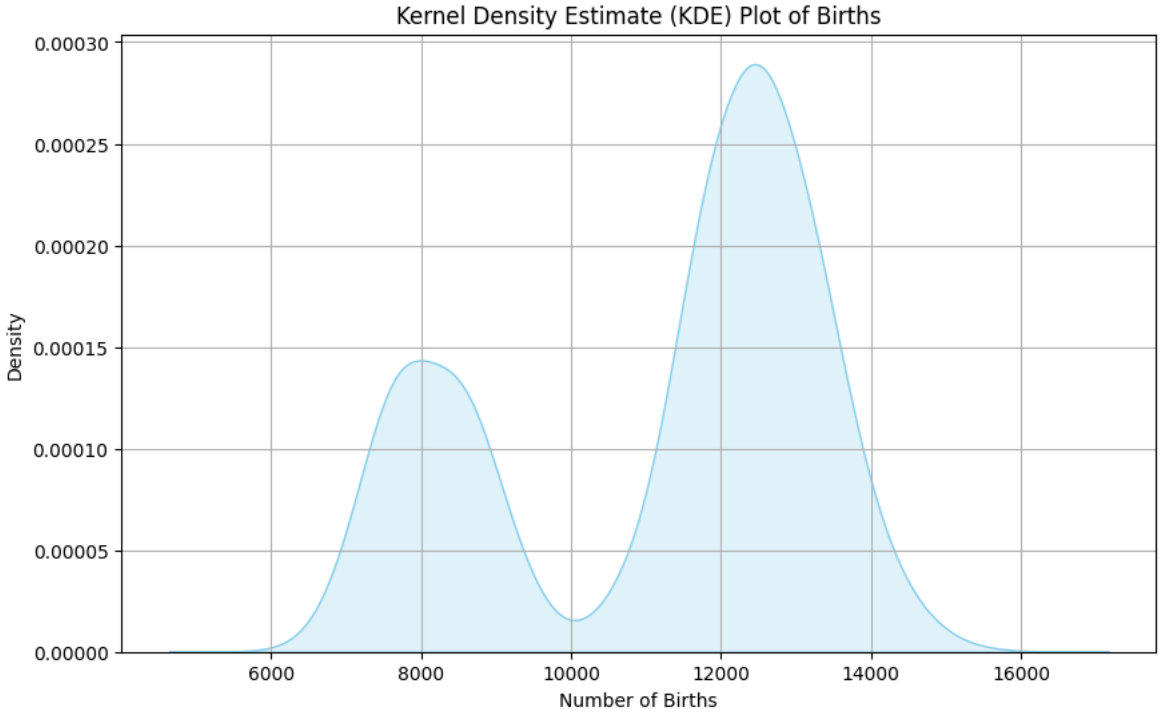
```
# Summary statistics
summary_stats = file1['births'].describe()
print("Summary Statistics for Births:")
print(summary_stats)
```

```
Summary Statistics for Births:
count     7670.000000
mean     11175.063625
std       2183.379461
min       5728.000000
25%       8786.250000
50%      11981.000000
75%      12816.000000
max      16081.000000
Name: births, dtype: float64
```

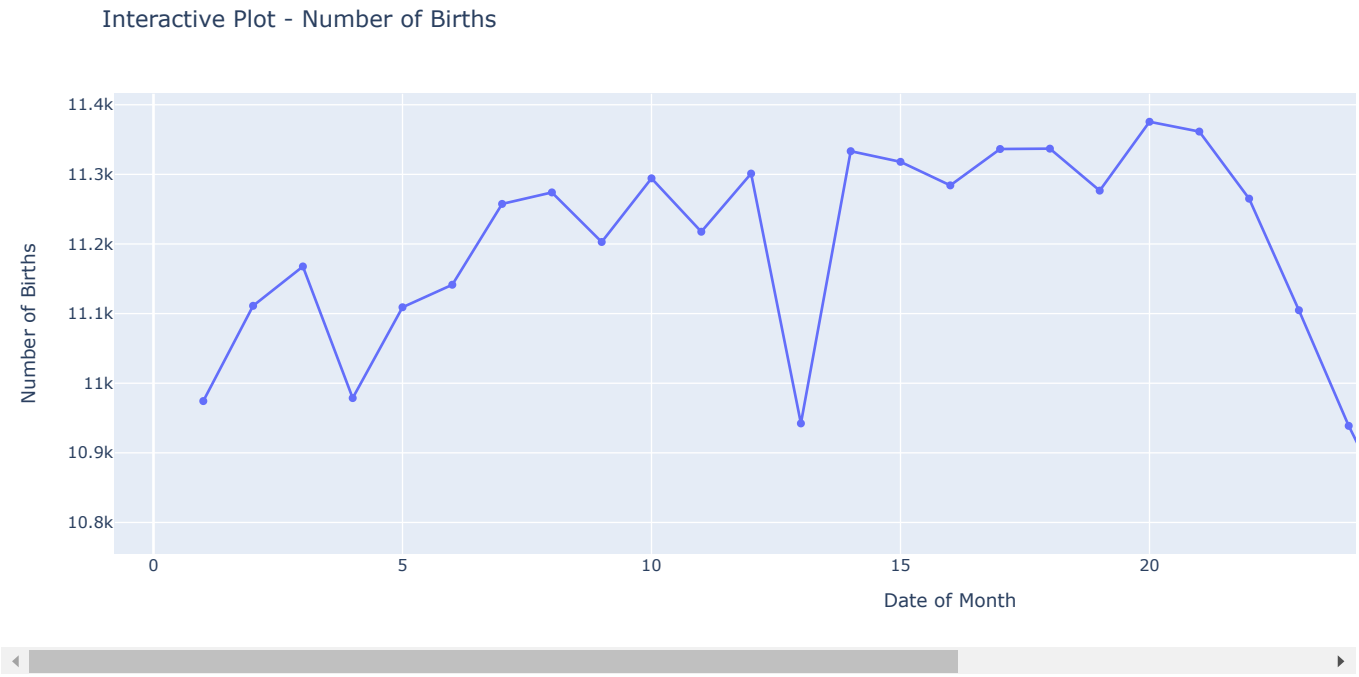
```
import matplotlib.pyplot as plt
# Bar plot
plt.bar(file1['year'], file1['births'], label='Bar Plot', color='green')
plt.title("Year vs Births")
plt.xlabel("Year")
plt.ylabel("Births")
plt.legend()
plt.show()
```



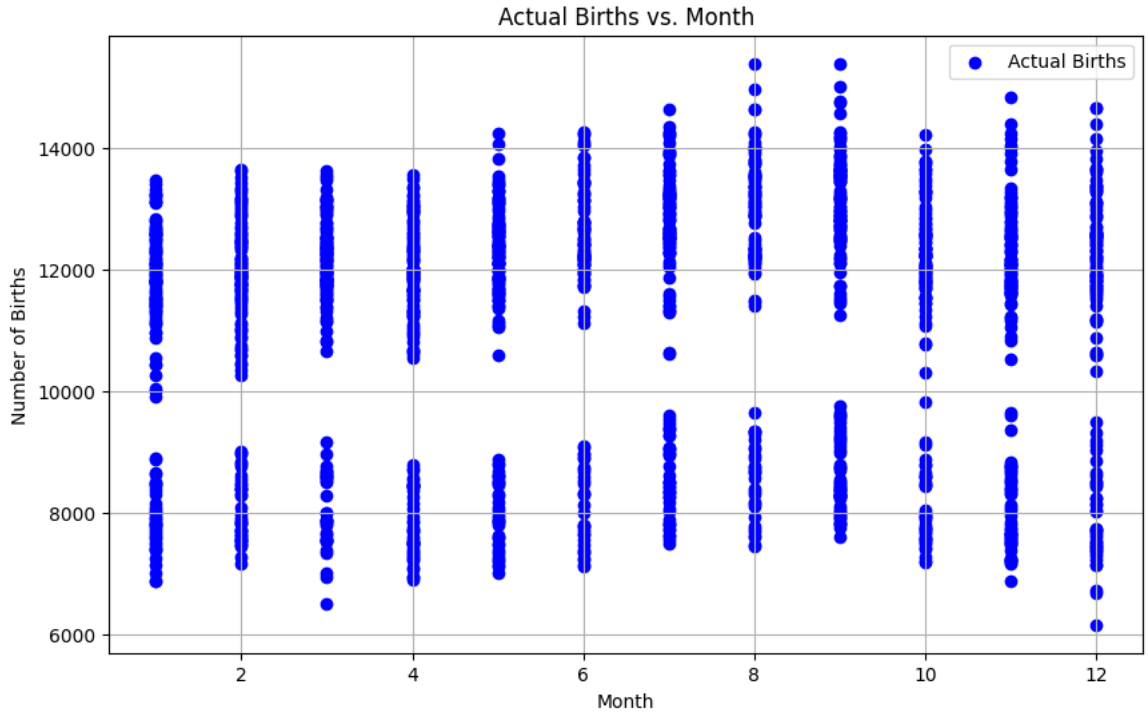
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
sns.kdeplot(file1['births'], fill=True, color='skyblue')
plt.title('Kernel Density Estimate (KDE) Plot of Births')
plt.xlabel('Number of Births')
plt.ylabel('Density')
plt.grid(True)
plt.show()
```



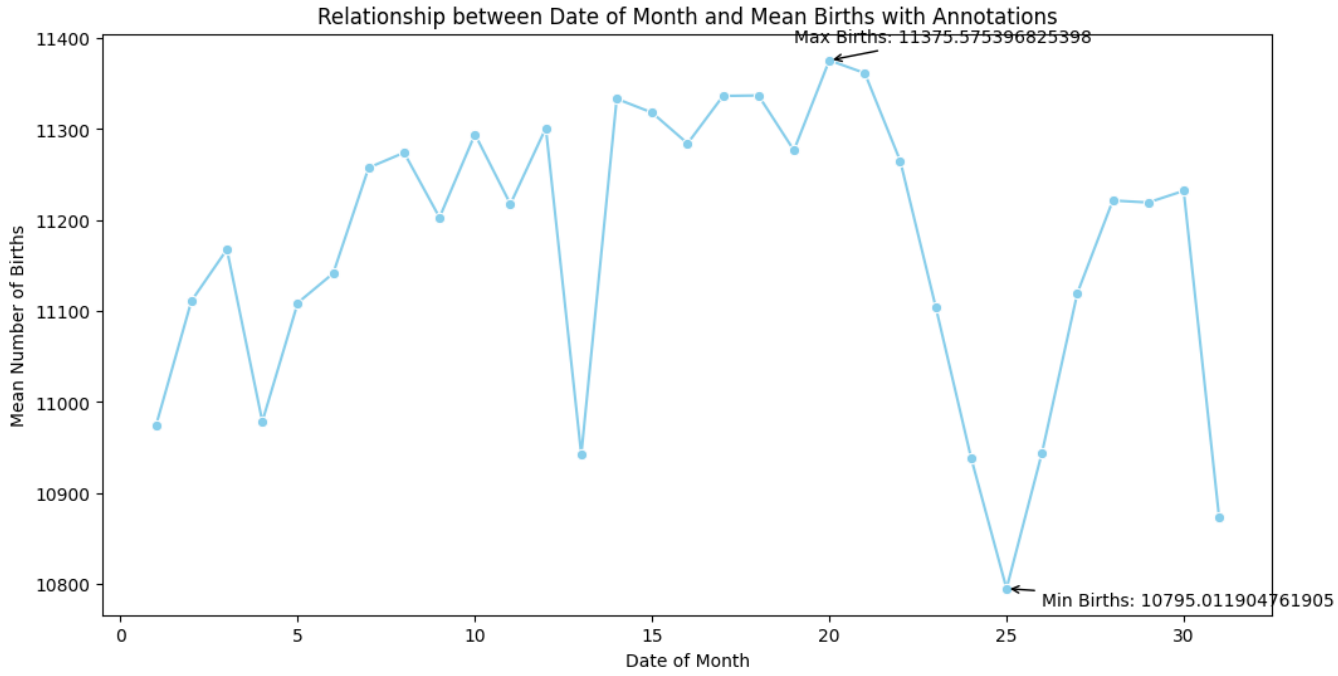

```
import pandas as pd
import plotly.express as px
# Read the dataset into a DataFrame
df = pd.read_csv("/content/US_births_1994-2014_CDC_NCHS.csv")
# Group by date_of_month and calculate the mean births for each date
daily_births = df.groupby('date_of_month')['births'].mean().reset_index()
# Create an interactive line plot using Plotly Express
fig = px.line(daily_births, x='date_of_month', y='births', markers=True, title='Interactive Plot - Number of Births')
# Add data labels to each point
fig.update_traces(text=daily_births['births'].astype(int).tolist(), textposition='top center')
# Update layout for better visibility
fig.update_layout(xaxis_title='Date of Month', yaxis_title='Number of Births', hovermode='closest')
# Show the plot
fig.show()
```




```
import matplotlib.pyplot as plt
# Create a DataFrame for actual births vs. month
actual_vs_month = pd.DataFrame({'Month': X_test['month'], 'Actual Births': y_test})
# Sort the DataFrame by month for better visualization
actual_vs_month = actual_vs_month.sort_values(by='Month')
# Plot the actual births vs. month
plt.figure(figsize=(10, 6))
plt.scatter(actual_vs_month['Month'], actual_vs_month['Actual Births'], label='Actual Births', marker='o', color='blue')
plt.xlabel('Month')
plt.ylabel('Number of Births')
plt.title('Actual Births vs. Month')
plt.legend()
plt.grid(True)
plt.show()
```



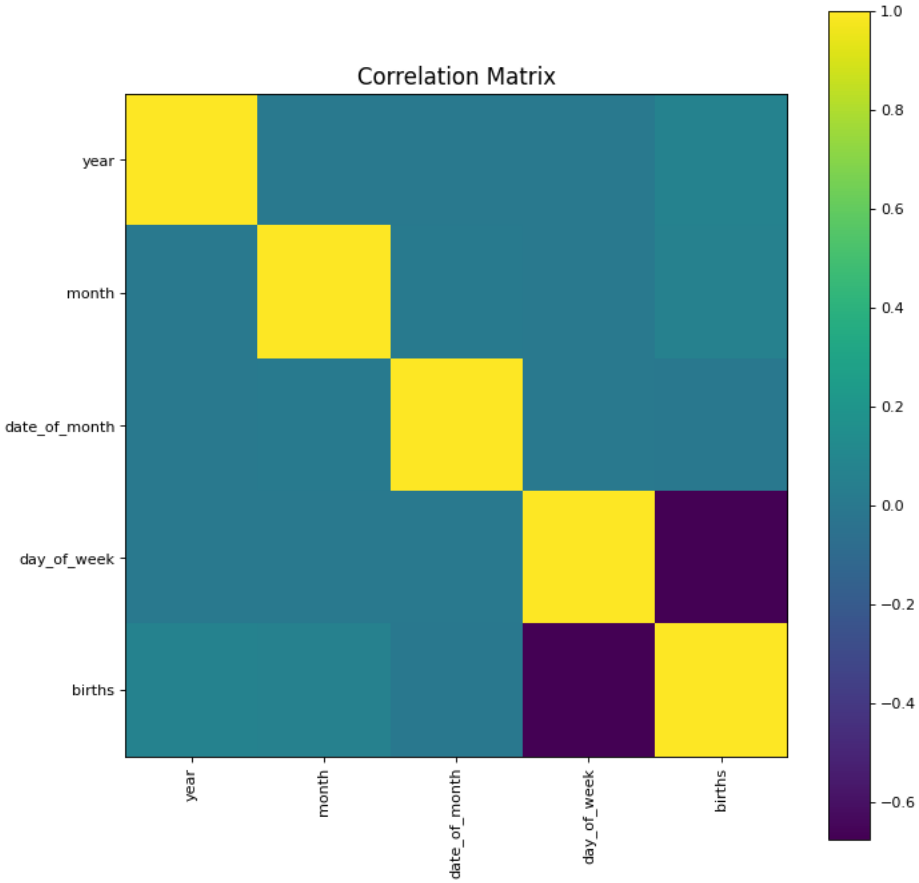
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# Read the dataset into a DataFrame
df = pd.read_csv("/content/US_births_1994-2014_CDC_NCHS.csv")
# Group by date_of_month and calculate the mean births for each date
daily_births = df.groupby('date_of_month')['births'].mean().reset_index()
# Plot the relationship between date_of_month and mean births
plt.figure(figsize=(12, 6))
sns.lineplot(x='date_of_month', y='births', data=daily_births, marker='o', color='skyblue')
# Annotate specific points (e.g., highest and lowest)
max_births_date = daily_births.loc[daily_births['births'].idxmax()]
min_births_date = daily_births.loc[daily_births['births'].idxmin()]
plt.annotate(f"Max Births: {max_births_date['births']}",
             xy=(max_births_date['date_of_month'], max_births_date['births']),
             xytext=(-20, 10),
             textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
plt.annotate(f"Min Births: {min_births_date['births']}",
             xy=(min_births_date['date_of_month'], min_births_date['births']),
             xytext=(20, -10),
             textcoords='offset points',
             arrowprops=dict(arrowstyle="->", color='black'))
plt.xlabel('Date of Month')
plt.ylabel('Mean Number of Births')
plt.title('Relationship between Date of Month and Mean Births with Annotations')
plt.show()
```



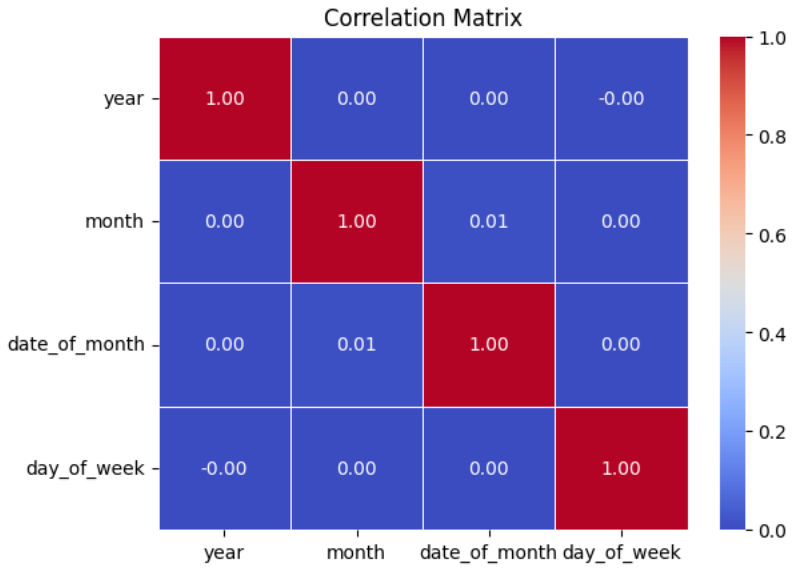
```
import pandas as pd
import matplotlib.pyplot as plt
def plotCorrelationMatrix(df, graphWidth):
    df = df.dropna('columns')  # drop columns with NaN
    df = df[[col for col in df if df[col].nunique() > 1]]  # keep columns where there are more than 1 unique values
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w', edgecolor='k')
    corrMat = plt.matshow(corr, fignum=1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title('Correlation Matrix', fontsize=15)  # Removed reference to df in the title
    plt.show()
df = pd.read_csv('/content/US_births_1994-2014_CDC_NCHS.csv')
# Select features and target variable
features = df[['year', 'month', 'date_of_month', 'day_of_week']]
target = df['births']
df_for_corr = pd.concat([features, target], axis=1)
# Use the plotCorrelationMatrix function to plot the correlation matrix
plotCorrelationMatrix(df_for_corr, 10)  # Adjust the graphWidth parameter as needed
```

```
<ipython-input-58-89c8ea6b191a>:6: FutureWarning:

In a future version of pandas all arguments of DataFrame.dropna will be keyword-only.
```



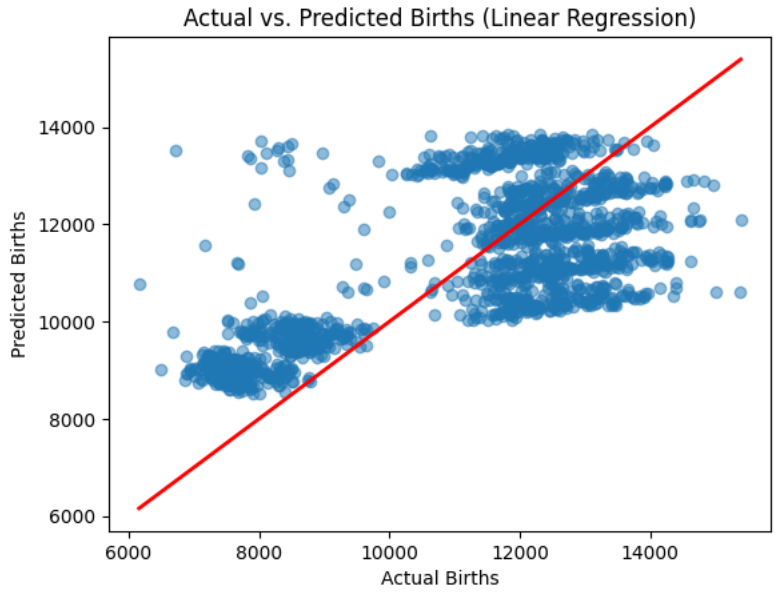

```
correlation_matrix = features.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()
```




```
# Creating X and y
file1 = pd.read_csv('/content/US_births_1994-2014_CDC_NCHS.csv')
X = file1['year']
y = file1['births']
# Splitting the varaibles as training and testing
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Print the shapes of the resulting sets
print("Training set - Features shape:", X_train.shape)
print("Training set - Target shape:", y_train.shape)
print("Testing set - Features shape:", X_test.shape)
print("Testing set - Target shape:", y_test.shape)
```

```
Training set - Features shape: (6136,)
Training set - Target shape: (6136,)
Testing set - Features shape: (1534,)
Testing set - Target shape: (1534,)
```

```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
df = pd.read_csv('/content/US_births_1994-2014_CDC_NCHS.csv')
features = df[['year', 'month', 'date_of_month', 'day_of_week']]
target = df['births']
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
# linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r-', lw=2)  # Regression line
plt.xlabel("Actual Births")
plt.ylabel("Predicted Births")
plt.title("Actual vs. Predicted Births (Linear Regression)")
plt.show()
```




```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
# Assuming df is your original DataFrame
df = pd.read_csv('/content/US_births_1994-2014_CDC_NCHS.csv')
# Extract features and target variable
features = df[['year', 'month', 'date_of_month', 'day_of_week']]
target = df['births']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
specific_year = 2025
dates_for_specific_year = pd.date_range(start=f"{specific_year}-01-01", end=f"{specific_year}-12-31", freq='D')
features_for_specific_year = pd.DataFrame({
    'year': specific_year,
    'month': dates_for_specific_year.month,
    'date_of_month': dates_for_specific_year.day,
    'day_of_week': dates_for_specific_year.dayofweek
})
predictions_for_specific_year = model.predict(features_for_specific_year)
print(f'Predicted Births for the Year {specific_year}:\n')
for date, prediction in zip(dates_for_specific_year, predictions_for_specific_year):
    print(f"{date.strftime('%Y-%m-%d')}: {prediction:.2f} births")
    # Find the date corresponding to the minimum and maximum predicted births
min_date = dates_for_specific_year[predictions_for_specific_year.argmin()]
max_date = dates_for_specific_year[predictions_for_specific_year.argmax()]
# Find the corresponding minimum and maximum predicted births
min_births = predictions_for_specific_year.min()
max_births = predictions_for_specific_year.max()
# Print the results
print(f'Minimum Predicted Births ({min_date.strftime("%Y-%m-%d")}): {min_births:.2f} births')
print(f'Maximum Predicted Births ({max_date.strftime("%Y-%m-%d")}): {max_births:.2f} births')
```

```
2025-11-27: 12599.75 births
2025-11-28: 11854.34 births
2025-11-29: 11108.94 births
2025-11-30: 10363.54 births
2025-12-01: 14900.07 births
2025-12-02: 14154.66 births
2025-12-03: 13409.26 births
2025-12-04: 12663.86 births
2025-12-05: 11918.45 births
2025-12-06: 11173.05 births
2025-12-07: 10427.65 births
2025-12-08: 14892.40 births
2025-12-09: 14147.00 births
2025-12-10: 13401.60 births
2025-12-11: 12656.19 births
2025-12-12: 11910.79 births
2025-12-13: 11165.38 births
2025-12-14: 10419.98 births
2025-12-15: 14884.74 births
2025-12-16: 14139.33 births
2025-12-17: 13393.93 births
2025-12-18: 12648.53 births
2025-12-19: 11903.12 births
2025-12-20: 11157.72 births
2025-12-21: 10412.32 births
2025-12-22: 14877.07 births
2025-12-23: 14131.67 births
2025-12-24: 13386.27 births
2025-12-25: 12640.86 births
2025-12-26: 11895.46 births
2025-12-27: 11150.05 births
2025-12-28: 10404.65 births
2025-12-29: 14869.41 births
2025-12-30: 14124.00 births
2025-12-31: 13378.60 births
Minimum Predicted Births (2025-01-26): 9978.68 births
Maximum Predicted Births (2025-12-01): 14900.07 births
```

```
# Split the data into training and testing sets with a fixed random state
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
# Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
# Make predictions on the test set
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, y_pred)
```

```
# Print the calcuate
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

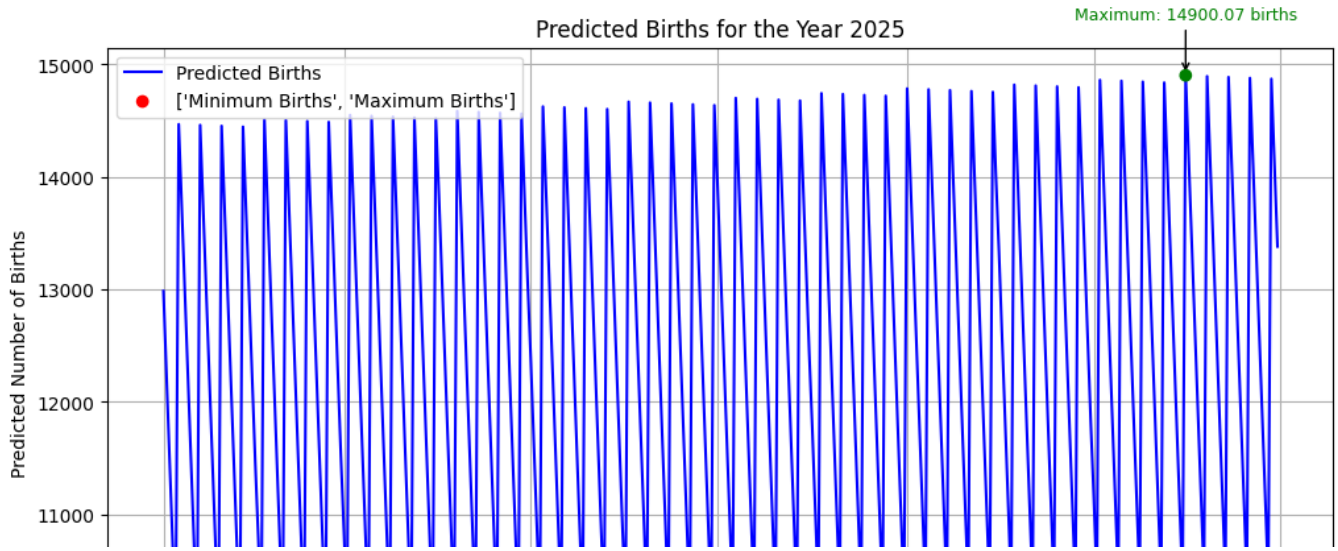
```
Mean Squared Error: 0.0038209548422804586
Mean Absolute Error: 0.03575020102522718
```

```
import matplotlib.pyplot as plt

# Plot the predicted births with arrows
plt.figure(figsize=(12, 6))
plt.plot(dates_for_specific_year, predictions_for_specific_year, label='Predicted Births', color='blue')
plt.scatter([min_date, max_date], [min_births, max_births],
            color=['red', 'green'], label=['Minimum Births', 'Maximum Births'], zorder=5)

# Draw arrows with text annotation
plt.annotate(f'Minimum: {min_births:.2f} births', xy=(min_date, min_births), xytext=(min_date, min_births - 500),
             arrowprops=dict(facecolor='red', arrowstyle='->'), color='red', fontsize=9, ha='center')
plt.annotate(f'Maximum: {max_births:.2f} births', xy=(max_date, max_births), xytext=(max_date, max_births + 500),
             arrowprops=dict(facecolor='green', arrowstyle='->'), color='green', fontsize=9, ha='center')

plt.title(f'Predicted Births for the Year {specific_year}')
plt.xlabel('Date')
plt.ylabel('Predicted Number of Births')
plt.legend()
plt.grid(True)
plt.show()
```


Predicted Births for the Year 2025
Maximum: 14900.07 births
Predicted Births
['Minimum Births', 'Maximum Births']
Predicted Number of Births
15000
14000
13000
12000
11000