Certainly! Here's a detailed documentation for each code snippet you provided:

Sales Forecasting Project Documentation

1. Data Loading and Exploration

1.1 Load the Data
- **Code:**
  ```python
  import pandas as pd
  data = pd.read_csv('sales_data.csv')
  ```

- **Description:**
  - Reads the sales data from the 'sales_data.csv' file into a Pandas DataFrame named 'data'.

1.2 Explore the Data
- **Code:**
  ```python
  print(data.head())
  print(data.info())
  ```

- **Description:**
  - Displays the first few rows of the dataset using `head()` to get an overview of the data.
  - Prints the summary information about the dataset using `info()`.

2. Data Cleaning

2.1 Handle Missing Values
- **Code:**
  ```python
  data.dropna(inplace=True)
  ```

- **Description:**
  - Drops rows with missing values from the dataset using `dropna()`.

2.2 Handle Non-Finite Values in 'Year' Column
- **Code:**
  ```python
  data['Year'].replace([np.inf, -np.inf, np.nan], -1, inplace=True)
  ```

  data['Year'] = data['Year'].astype(int)
  ```

- **Description:**
  - Replaces non-finite values in the 'Year' column with -1 and converts the column to integers.

### 2.3 Handle Non-Finite Values in 'Customer Age' Column
- **Code:**
  ```python
  data['Customer Age'].replace([np.inf, -np.inf, np.nan], -1, inplace=True)
  ```

- **Description:**
  - Replaces non-finite values in the 'Customer Age' column with -1.

2.4 Drop 'Column1' from DataFrame
- **Code:**
  ```python
  data = data.drop("Column1", axis=1)
  ```

- **Description:**
  - Drops the 'Column1' from the DataFrame using `drop()`.

3. Data Preprocessing

3.1 Encode Categorical Columns
- **Code:**
  ```python
  label_encoder = LabelEncoder()
  categorical_columns = ['Customer Gender', 'Country', 'Product Category', 'Sub Category', 'Age Group']

  for column in categorical_columns:
      data[column] = label_encoder.fit_transform(data[column])
  ```

- **Description:**
  - Encodes categorical columns using `LabelEncoder`.

3.2 Map Month Names to Numbers
- **Code:**
  ```python
  month_mapping = {'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6,

'July': 7, 'August': 8, 'September': 9, 'October': 10, 'November': 11, 'December': 12}

  data['Month'] = data['Month'].map(month_mapping)
  ```

- **Description:**
  - Maps month names to corresponding numbers for the 'Month' column.

4. Exploratory Data Analysis (EDA)

4.1 Box Plot for 'Revenue'
- **Code:**
  ```python
  sns.boxplot(x=data['Revenue'])
  plt.title('Box Plot of Revenue')
  plt.show()
  ```

- **Description:**
  - Generates a box plot to visualize the distribution of the 'Revenue' column.

4.2 Calculate Skewness of 'Revenue' Column
- **Code:**
  ```python
  skewness = data['Revenue'].skew()
  print(f'Skewness of the Revenue column: {skewness}')
  ```

- **Description:**
  - Calculates and prints the skewness of the 'Revenue' column.

4.3 Count Values Greater Than 2000 in 'Revenue'
- **Code:**
  ```python
  count_values_greater_than_2000 = (data['Revenue'] > 2000).sum()
  print(f'Total number of values greater than 2000 in the "Revenue" column:
{count_values_greater_than_2000}')
  ```

- **Description:**
  - Counts and prints the number of values greater than 2000 in the 'Revenue' column.

4.4 Box Plot for 'Quantity'
- **Code:**
  ```

  ```python
  sns.boxplot(x=data['Quantity'])
  plt.title('Box Plot of Quantity')
  plt.show()
  ```

- **Description:**
  - Generates a box plot to visualize the distribution of the 'Quantity' column.

5. Key Business Metrics

5.1 Calculate Key Metrics
- **Code:**
  ```python
  total_revenue = data['Revenue'].sum()
  average_revenue_per_sale = data['Revenue'].mean()
  max_quantity_sold = data['Quantity'].max()
  min_quantity_sold = data['Quantity'].min()

  print(f"Total Revenue: ${total_revenue:.2f}")
  print(f"Average Revenue per Sale: ${average_revenue_per_sale:.2f}")
  print(f"Maximum Quantity Sold in a Single Transaction: {max_quantity_sold}")
  print(f"Minimum Quantity Sold in a Single Transaction: {min_quantity_sold}")
  ```

- **Description:**
  - Calculates and prints key business metrics, including total revenue, average revenue per sale, and quantity statistics.

6. Model Development

6.1 Create Age Groups
- **Code:**
  ```python
  age_bins = [0, 18, 35, 50, 100]
  age_labels = ['0-18', '19-35', '36-50', '51+']


 data['Age Group'] = pd.cut(data['Customer Age'], bins=age_bins, labels=age_labels)
  ```

- **Description:**
  - Creates age groups based on predefined bins and labels.

6.2 Correlation Matrix
- **Code:**
  ```python
  correlation_matrix = data.corr()
  print(correlation_matrix)
  ```

- **Description:**
  - Calculates and prints the correlation matrix for the features in the dataset.

6.3 Log Transformation and Outlier Removal
- **Code:**
  ```python
  skewness = data['Revenue'].skew()

  if abs(skewness) > 1:
      data['Revenue'] = np.log1p(data['Revenue'])
      print("Log transformation applied.")

  Q1 = data['Revenue'].quantile(0.25)
  Q3 = data['Revenue'].quantile(0.75)
  IQR = Q3 - Q1

  outliers = ((data['Revenue'] < Q1 - 1.5 * IQR) | (data['Revenue'] > Q3 + 1.5 * IQR)).sum()

  if outliers > 0:
      print(f"{outliers} outliers detected and removed using IQR.")
      data = data[(data['Revenue'] >= Q1 - 1.5 * IQR) & (data['Revenue'] <= Q3 + 1.5 * IQR)]
  ```

- **Description:**
  - Applies log transformation to the 'Revenue' column if skewness is greater than 1.
  - Identifies and removes outliers using the interquartile range (IQR) method.

6.4 Scaling Features
- **Code:**
  ```python
  from sklearn.preprocessing import StandardScaler

  X = data.drop(['Revenue'], axis=1)
  scaler = StandardScaler()
  X_scaled = scaler.fit_transform(X)
  ```

- **Description:**
  - Scales the features (excluding 'Revenue') using StandardScaler.

6.5 Train-Test Split
- **Code:**
  ```python
  from sklearn.model_selection import train_test_split

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

- **Description:**
  - Splits the data into training and testing sets with an 80-20 split.

6.6 Random Forest Model
- **Code:**
  ```python
  from sklearn.ensemble import RandomForestRegressor

  model = RandomForestRegressor(n_estimators=100, random_state=42)
  model.fit(X_train, y_train)
  ```

- **Description:**
  - Initializes and trains a Random Forest Regressor model with 100 estimators.

6.7 Model Evaluation
- **Code:**
  ```python
  from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

  y_pred = model.predict(X_test)

  mse = mean_squared_error(y_test, y_pred)
  mae = mean_absolute_error(y_test, y_pred)
  r2 = r2_score(y_test, y_pred)

  print(f'Mean Squared Error (MSE): {mse:.4f}')
  print(f'Mean Absolute Error (MAE): {mae:.4f}')
  print(f'R-squared (R2): {r2:.4f}')
  ```

- **Description:**

  - Makes predictions on the testing set and evaluates the model using mean squared error (MSE), mean absolute error (MAE), and R-squared (R2).

6.8 Residual Analysis
- **Code:**
  ```python
  residuals = y_test.reset_index(drop=True) - pd.Series(y_pred)

  plt.scatter(y_pred, residuals, alpha=0.5)
  plt.title('Residual Analysis')
  plt.xlabel('Predicted Revenue')
  plt.ylabel('Residuals')
  plt.axhline(y=0, color='r', linestyle='-')
  plt.show()
  ```

- **Description:**
  - Conducts residual analysis by plotting residuals against predicted revenue to identify patterns or deviations.

6.9 Feature Importance
- **Code:**
  ```python
  feature_importances = model.feature_importances_

  plt.bar(features, feature_importances)
  plt.title('Feature Importance')
  plt.xlabel('Features')
  plt.ylabel('Importance Score')
  plt.xticks(rotation=45, ha='right')
  plt.show()
  ```

- **Description:**
  - Obtains feature importances from the model and creates a bar plot to visualize the importance of each feature.

6.10 Hyperparameter Tuning
- **Code:**
  ```python
  from sklearn.model_selection import GridSearchCV

  param_grid = {
      'n_estimators': [50, 100, 150],
  ```

      'max_depth': [None, 10, 20],
      'min_samples_leaf': [1, 2, 4]
  }

  model = RandomForestRegressor(random_state=42)
  grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3,
scoring='neg_mean_squared_error')
  grid_search.fit(X_train, y_train)

  best_params = grid_search.best_params_
  ```

- **Description:**
  - Performs grid search to find the best hyperparameters for the Random Forest Regressor
model.

### 6.11 Optimized Model
- **Code:**
  ```python
  optimal_model = RandomForestRegressor(
      n_estimators