COVID-19 Demographic Analysis Report

Omar Awad
Mahmoud Mohamed
Ahmed Amin
Sarah Mahmoud Salah

Supervision

Dr.Mahmoud Abdelaziz
Eng.Ahmed Abdelsalam
Eng.Asmaa Mostafa
Eng.Ahmed Ali

Communications and Information Engineering
Zewail City of Science and Technology

Contents

1 How to use

1.1 Introduction

Figure 1: caption

2 Technical Documentation

2.1 Project Structure

we will edit here with structure of code

2.2 Function Descriptions

The project contains several functions, each with specific purposes:

- **load_data(filepath)**: Loads the dataset from the specified file path.
- **clean_data(df)**: Cleans the dataset by handling missing values and encoding categorical variables.
- **create_proportions(df)**: Creates proportion columns for gender, age groups, ICU admissions, and hospitalizations.
- **add_polynomial_features(X)**: Generates polynomial features up to the specified degree.
- **standardize_features(X)**: Standardizes the features to have a mean of 0 and a standard deviation of 1.

- **remove_outliers(X, y)**: Removes outliers from the dataset based on Z-scores.
- **fit_model(X, y)**: Fits an OLS regression model to the data.
- **plot_diagnostics(model)**: Generates diagnostic plots to evaluate the model.

2.3 Data Collection and Cleaning

The steps followed for data collection and cleaning are as follows:

- **Data Collection**: The COVID Case Surveillance dataset was collected from a reliable source and stored in the 'data/' directory.
- **Initial Cleaning**: The raw data was loaded, and rows with missing values in critical columns (case month, gender, age group, hospitalization status, ICU status, and death status) were removed.
- **Encoding**: Categorical variables (gender, hospitalization status, ICU status, and death status) were encoded into numerical values.
- **Proportion Calculation**: Proportions for each predictor (gender, age groups, ICU admissions, and hospitalizations) were calculated for each month.
- **Outlier Removal**: Outliers were identified using Z-scores and removed to improve the robustness of the model.

2.4 Challenges, Limitations, and Assumptions

Several challenges, limitations, and assumptions were encountered during the analysis:

- **Challenges**:
  - **Data Quality**: Inconsistent or missing data values required significant cleaning efforts.
  - **Multicollinearity**: High correlation between predictors necessitated careful handling to avoid multicollinearity issues.
- **Limitations**:
  - **Data Scope**: The analysis is limited to the available dataset and may not generalize to other populations or time periods.
  - **Model Complexity**: Higher-order polynomial features can lead to overfitting, reducing the model's ability to generalize.
- **Assumptions**:
  - **Normality**: Assumed normality of residuals for the OLS regression model.
  - **Linearity**: Assumed linear relationships between the predictors and the response variable.
  - **Independence**: Assumed independence of observations in the dataset.

3 Exploratory Analysis:

3.1 Overview

The primary objective of this task is to perform an exploratory data analysis (EDA) on a COVID-19 case surveillance dataset and Household Pulse Survey dataset. Specifically, we aim to understand the distribution and relationships within the data

3.2 Data Preparation and Cleaning

The dataset was cleaned and preprocessed to ensure that it was suitable for regression analysis. The following steps were taken:

- **Selecting Relevant Columns**: Columns related to case month, gender, age group, hospitalization status, ICU status, and death status were retained.
- **Handling Missing Values**: Rows with missing values in the relevant columns were dropped.
- **Encoding Categorical Variables**: Gender and other categorical variables were encoded into numerical values.
- **Creating Proportions**: Proportions of each predictor (gender, age groups, ICU admissions, and hospitalizations) were calculated for each month.

3.3 Q1 Step-by-Step Work

1. Filter Rows Based on Conditions: Create a new DataFrame `hospitalizations` by filtering the original DataFrame `df1_cleaned` to include only rows where the column `hosp_yn` has the value 'Yes'. Similarly, create another DataFrame `deaths` by filtering `df1_cleaned` for rows where the column `death_yn` is 'Yes'.

2. Count Hospitalizations and Deaths by Month: For `hospitalizations`, group the data by the `case_month` column and count the number of occurrences in each month. Store the result in `hospitalizations_count`. Do the same for `deaths`, grouping by `case_month` and counting occurrences, storing the result in `deaths_count`.

3. Combine Counts into a DataFrame: Create a new DataFrame `summary_df` with two columns: `Hospitalizations` and `Deaths`. These columns are populated with the counts from `hospitalizations_count` and `deaths_count`. Use `fillna(0)` to replace any missing values with 0, ensuring that months with no hospitalizations or deaths are represented correctly.

4. Reset Index: Reset the index of `summary_df` so that `case_month` becomes a regular column rather than an index.

5. Plotting: Initialize a new figure for the plot with a specified size (14x7 inches). Create line plots for both `Hospitalizations` and `Deaths`, using different colors (blue for hospitalizations and red for deaths).

6. Additional Customization: Add labels to the x-axis ('Month-Year') and y-axis ('Count'). Add a title to the plot: *Hospitalizations vs Deaths from COVID-19 over Time*. Rotate the x-axis tick labels by 45 degrees for better readability. Add a legend to the plot to differentiate between the lines for hospitalizations and deaths.

7. Removing Borders: Make the top and right borders of the plot invisible for a cleaner look.

8. Displaying the Plot: Adjust the layout of the plot to ensure everything fits well and nothing overlaps. Display the plot on the screen.

3.4 Q2 Step-by-Step Work :

1. Filtering Data: We filter the DataFrame `df1_cleaned` to include only rows where the column `death_yn` contains either 'Yes' or 'No'. This ensures that we focus only on records where the death status is explicitly stated. The filtered data is stored in the variable `data`.

2. Selecting Relevant Columns: We define a list `columns_to_keep` containing the columns 'death_yn', 'age_group', 'sex', and 'race'. Then, we create a new DataFrame `data_demographic` by selecting only the columns specified in `columns_to_keep` from the filtered data.

3. Grouping and Aggregating Data by Death Status, Age Group, and Other Demographics: Next, we group the `data_demographic` DataFrame by the relevant demographic columns (sex, 'age_group', 'race'). For each demographic group, we calculate the percentage of 'Yes' (deaths) and 'No' (survivals) in the 'death_yn' column using `value_counts(normalize=True)`. We then unstack the grouped data to pivot it into a more usable format, with the demographic categories as rows and death status ('Yes' or 'No') as columns. Any missing values are dropped. The result is stored in `data_demographic_demographics`.

4. Plotting: We extract the column corresponding to 'Yes' (deaths) from `data_demographic_demographics` and store it in `data_demographic_demographics_plot`. Then, we create a new figure with a size of 12x6 inches. We sort the `data_demographic_demographics_plot` Series values and plot them as a bar chart using `plot(kind='bar')`. Labels for the y-axis ('death count'), title ('average rates of COVID-related deaths relative to demographic factors'), and x-axis (demographic categories) are added. Grid lines are removed using `plt.grid(False)`. Finally, the plot is displayed using `plt.show()`.

3.5 Q3 Step-by-Step Work :

This report outlines a data analysis pipeline for analyzing COVID-19 data across different age groups. The pipeline involves filtering, selecting, aggregating, and visualizing the data to gain insights into hospitalization and death rates. **Step 1: Data Filtering** We define a list of columns to keep, including 'death_yn', 'hosp_yn', and 'age_group'. Then, we filter rows from the cleaned dataset where 'death_yn' and 'hosp_yn' are either 'Yes' or 'No'. The filtered data is stored in the variable `filtered_data`.

Step 2: Data Selection After filtering, we keep only the specified columns ('death_yn', 'hosp_yn', 'age_group'). We drop rows with 'age_group' labeled as 'Missing' to ensure data completeness. The resulting dataset is stored in the variable `data`.

Step 3: Grouping and Aggregation In this step, we group the selected data by the 'age_group' column. Then, we use the `agg()` function to aggregate the 'death_yn' and 'hosp_yn' columns within each group. We calculate the mean of each column using a lambda function that checks if each value is 'Yes' and then takes the mean of the resulting boolean values. The aggregated data is stored in the variable `grouped_data`.

Step 4: Visualization The visualization code is used to create a bar plot showing the average COVID-19 hospitalization and death rates per age group. However, the specific code for visualization is not provided in this report.

3.6 Q4 Step-by-Step Work :

In this section, we analyze COVID-19 data on a state level. **Data Filtering:** Filter the cleaned DataFrame `df1_cleaned` to include only rows where the columns `death_yn` and `hosp_yn` contain either 'Yes' or 'No'. This filtered data is stored in the variable `filtered_data`.

Grouping Data by State: We group the cleaned DataFrame `df1_cleaned` by the state of residence (`res_state`), creating a grouped object `grouped_data`.

Calculating Total Hospitalizations and Deaths per State: Using the grouped data, we calculate the total number of hospitalizations and deaths per state by applying a lambda function to count the occurrences of 'Yes' in the `hosp_yn` and `death_yn` columns, respectively.

Calculating Total Cases per State: We calculate the total number of COVID-19 cases per state by getting the size of each group in the grouped data.

Calculating Average Hospitalization and Death Rates per State: Using the total hospitalizations, deaths, and cases per state, we calculate the average hospitalization and death rates per state.

Creating and Printing Results DataFrame: We create a DataFrame called `results_df` to store the calculated average hospitalization and death rates per state.

Plotting the Results: We plot the average COVID-19 hospitalization and death rates per state as a bar chart using `results_df.plot(kind='bar')`. We also set the title, labels for the x and y axes, rotate x-axis labels for better readability, add a legend, and adjust the layout for better visualization.

3.7 Q5 Step-by-Step Work :

Data Preparation: We begin by selecting specific columns ('age$_group'$,'$ $underlying_conditions_yn'$,'$ $icu_yn')from the clear$

Feature Engineering: We create a new column 'newcol' by concatenating the 'age$_group'$ and '$ $underlying_conditions_y$

Chi-Square Test: Using the contingency table generated from the 'newcol' and 'icu$_yn'$ $columns, we conduct a chi-square test of independence to assess whether there's a significant association between the need for ICU care ('icu_yn') and the square statistic, p-value, degrees of freedom, and expected frequencies to evaluate the statistical significance of the associ$

3.8 Q6 Step-by-Step Work :

Step 1: Data Selection

In this step, we specifically choose the columns 'kindwork' and 'expctloss' from the DataFrame 'df2'. These columns likely contain information related to the type of work individuals are engaged in and their expectations regarding employment loss due to COVID-19. By creating a subset DataFrame named 'pulse$_survey_data_subset_Q6'$, $we isolate these columns for further analysis.$

Step 2: Define and Apply Mappings

We define mappings for the numerical codes present in the 'kindwork' and 'expctloss' columns. These mappings assign meaningful labels to each numerical code, making the data more interpretable. For example, the numerical code 1 might correspond to 'Government' in the 'kindwork' column. We then apply these mappings to replace the numerical codes with their corresponding descriptive labels in the 'pulse$_survey_data_subset_Q6'$ $DataFrame.$

Step 3: Filter out Unnecessary Rows

In this step, we filter out rows from the 'pulse$_survey_data_subset_Q6'$ $DataFrame where the 'kindwork' column contains ce$

Step 4: Define Plotting Function

We define a plotting function named 'plot_employment_loss_rates' to visualize the rate of expected employment loss by employment sector. This function calculates the rate of expected employment loss for each category in the 'kindwork' column and plots the data accordingly. The 'normalize' parameter allows us to specify whether to display the counts as absolute counts or normalized rates.

Step 5: Plotting

Finally, we call the plotting function twice: once with normalization and once without normalization. This step generates two sets of visualizations depicting the rate of expected employment loss across different employment sectors. These visualizations aid in understanding how the COVID-19 pandemic has impacted employment expectations within various sectors of the economy.

3.9 Q7 Step-by-Step Work :

Columns to Keep with Readable Names:

We create a dictionary 'columns$_to_keep$'$where each key represents an original column name from the dataset, and the corr$

Selecting Necessary Columns and Renaming: Using the keys of the 'columns$_to_keep$'$dictionary, we select only the$

Value Mappings and Data Cleaning: We define mappings that convert numerical codes to descriptive labels for categorical variables such as 'Gender', 'Hispanic Origin', 'Race', etc. These mappings make the data more interpretable and facilitate analysis. We then apply these mappings to their respective columns in the DataFrame. Additionally, we filter out rows containing unknown values and drop rows with missing data to ensure data quality and integrity.

Plotting Function: We define a plotting function named 'plot$_employment_loss_by_demographic$'$that calculates and visuali$
'$demographic_column$', $which specifies the demographic variable to analyze, and$'$normalize$', $which determines whether to$

Plotting: In this step, we iterate through each demographic variable (excluding 'Expected Employment Loss') and call the plotting function to generate visualizations of the employment loss rates with normalization for each demographic category. This iterative process allows us to gain insights into how employment loss varies across various demographic groups in the dataset.

3.10 Q8 Step-by-Step Work :

1. Dataframe Subset Creation:

- Create a subset of DataFrame `df1_cleaned` containing only the columns 'res_state' and 'hosp_yn', and store it in `dataframe1_subset`.
- Replace categorical values in the 'hosp_yn' column with numerical values (1 for 'Yes', 0 for 'No', and None for other values like 'Unknown', 'Missing', 'NA', and 'Other').
- Remove rows with missing values.
- Group the data by 'res_state' and calculate the normalized counts of 'Yes' and 'No' in the 'hosp_yn' column for each state. Unstack the result, and fill any missing values with 0.

- Select the column corresponding to 'Yes' and sort it in descending order. Take the top 10 states with the highest counts of hospitalizations (Yes).

2. State Abbreviation Conversion:

- Replace state abbreviations in `dataframe1_subset` with their full names.

3. Second Dataframe Subset Creation:

- Create a subset of DataFrame `df2` containing only the columns 'est_st' and 'expctloss', and store it in `dataframe2_subset`.
- Replace numerical codes in the 'expctloss' column with categorical values ('Yes' for 1, 'No' for 2, and None for -88 and -99).
- Replace categorical values in the 'expctloss' column with numerical values (1 for 'Yes', 0 for 'No', and None for other values like 'Unknown', 'Missing', 'NA', and 'Other').
- Remove rows with missing values.

4. State Abbreviation Conversion for Second Subset:

- Replace numerical codes in the 'est_st' column of `dataframe2_subset` with the corresponding state names.

5. Merge and Grouping:

- Merge `dataframe1_subset` and `dataframe2_subset` on the 'res_state' and 'est_st' columns.
- Group the merged DataFrame by 'res_state' and calculate the normalized counts of 'expctloss' for each state. Unstack the result, and fill any missing values with 0.
- Sort the DataFrame by the 'Yes' column in descending order.

6. Plotting:

- Set the positions for the bars on the x-axis.
- Define the width of each bar.
- Create a bar chart using matplotlib.
- Set the x-axis labels to be the state abbreviations.
- Set the y-axis label and title for the plot.
- Add a legend and display the plot.

3.11 Q9 Step-by-Step Work :

Importing Necessary Libraries: We import the ‘chi2$_contingency$‘$function from the ‘scipy.stats‘module to perform the$
$square test of independence. This function allows us to determine if there is a significant association between two categorical$

Columns to Keep:$We define a list named ‘columns_to_keep‘containing the column names we want to keep from the Data Fra$

Subset the Data:$We create a subset DataFrame named ‘pulse_data_subset‘by selecting only the columns specified in ‘colum$

Value Mappings and Data Cleaning:$We define mappings to convert numerical codes to descriptive labels for the 'incon$

Chi-Square Test:$We create contingency tables for delayed medical treatment and not obtained medical treatment using the$
$square test of independence for both variables using the ‘chi2_contingency()‘function. This test helps us determine whether th$
$square statistic and p-value to assess the strength and significance of this association.$

Plotting:$We calculate the rates of delayed medical treatment and not obtained medical treatment by income using the ‘groupb$

3.12 Q10 Step-by-Step Work :

Importing Necessary Libraries: We import the ‘chi2$_contingency$‘$function from the ‘scipy.stats‘module to perform the$
$square test of independence.$

Columns to Keep and Data Subset: We define a list of columns, ‘age$_group$‘$and ‘symptom_status‘, that we want to kee$

Data Cleaning: We drop rows with missing values in the columns ‘age$_group$‘$and ‘symptom_status‘. Additionally, we fi$

Chi-Square Test: We create a contingency table for the variables 'age$_group'and'symptom_status'using the ‘pd.crosst$
$square test of independence using the ‘chi2_contingency()‘function. Finally, we print the chi-square statistic and p-$
$value to analyze the independence between COVID-19 symptom manifestation and age group.$

Plotting: We calculate the rate of symptomatic cases by age group using the ‘groupby()‘ and
‘value$_counts()‘function s. Then, we plot the rates of symptomatic cases for each age group using a bar plot to visualize the distri$
$19 cases across different age groups.$

4 Answering Questions

4.1 Q1 Step-by-Step Work:

Load Data with Specific Columns

- The script loads only three columns from the CSV file: deathyn, underlyingconditionsyn, and hospyn. This approach reduces memory usage compared to loading the entire dataset.

Clean the Data It removes rows with null values from the loaded DataFrame. **Filter Data for Analysis**

- Selects patients who have died and were hospitalized.
- Counts the proportion of patients with and without underlying conditions among those who died and were hospitalized.

Plotting

- Sets up the figure size for the plot.
- Creates a bar chart showing the death rate among hospitalized patients with and without underlying conditions.
- Adds labels and a title to the plot.
- Displays the plot.

4.2 Q2 Step-by-Step Work:

Load Data with Specific Columns The script loads only three columns from the CSV file: death_yn, age_group , sex and race. This approach reduces memory usage compared to loading the entire dataset. **Filter and Clean the Data**

- Filters the DataFrame to exclude rows where the age_group is marked as 'Missing'.
- Further filters the DataFrame to include only rows where death_yn is 'Yes', focusing on deceased patients.

Analyze the Data Counts the occurrence of each age group among the deceased patients, normalizes the counts to proportions, and sorts them by index. **Plotting**
Repeat these steps for both sex and race columns.

4.3 Q3 Step-by-Step Work:

Filter the DataFrame

- Starting by filtering the exposure_df DataFrame to include only rows where exposure_yn is 'Yes' and either hosp_yn is 'Yes' or 'No'. This operation creates a new DataFrame, exposure2_df, which contains patients who were exposed and either hospitalized or not.

Calculate Proportions

- Next, the script calculates the proportion of patients in each group hospitalized or not using the .valuecounts method with normalization set to True. This results in a Series object, underlyingconditionscounts, where the index represents the patient status 'Yes' for hospitalized, 'No' for not hospitalized, and the values represent the normalized count of patients in each category.

Plotting

4.4 Q4 Step-by-Step Work:

Clean the Data

- Removes rows with missing values from the Asymptomaticdf DataFrame.

- Drops rows where the symptom$_status is either 'Missing' or 'Unknown'$.

Display the First Few Rows of the Cleaned DataFrame

- Shows the first few rows of the cleaned Asymptomaticdf DataFrame.

Filter the Data for Hospitalized Asymptomatic Patients

- Filters the Asymptomaticdf DataFrame to include only rows where hospyn is 'Yes', creating a new DataFrame hospitilazedAsymptomatic.
- Displays the first few rows of the filtered DataFrame.

Analyze the Data

- Calculates the proportion of patients in each symptom status group (asymptomatic, symptomatic, etc.) among those who were hospitalized.

Plotting Repeat same steps for death column

4.5 Q5 Step-by-Step Work:

Read the CSV File

- The script begins by reading a specific CSV file located at drive selecting only the columns eip and estst to reduce memory usage and focus on the relevant data.

Filter the Data

- The script filters the DataFrame to include only rows where the eip value is 1.0, 2.0, or 3.0, indicating receipt of EIPs

Group and Summarize the Data

- Sorts the DataFrame by eip and groups it by est$_st(stateabbreviation), counting the number of occurrences in each group.$

Convert State Codes to Names

- Creates a dictionary mapping state codes to their full names and applies this mapping to the DataFrame's index to convert state abbreviations to full names.

Calculate Percentages

- Calculates the percentage of respondents who received EIPs in each state relative to the total number of respondents.

Plotting

4.6 Q6 Step-by-Step Work:

Filter and Clean the Data

- Filters the DataFrame to exclude rows where the agegroup is marked as 'Missing'.
- Further filters the DataFrame to include only rows where icuyn is 'Yes', focusing on patients admitted to the ICU.

Analyze the Data

- Counts the occurrence of each age group among the ICU-admitted patients.

Visualize the Results Repeat steps for both race and sex columns

4.7 Q7 Step-by-Step Work:

Filter ICU Patients
Calculate Proportions of Underlying Conditions

- calculates the proportion of ICU patients who have ('Yes') versus do not have ('No') underlying medical conditions. The .value$_c ounts(normalize = True) method computes the proportion of each category, and .sort_i ndex() or

Normalize Counts to Proportions Plotting

4.8 Q8 Step-by-Step Work:

Define Internet Availability Mapping Apply Internet Availability Mapping Filter Out Invalid Values

- ilters the DataFrame df2 to remove rows where intrntavail does not correspond to a valid description (as defined in internet mapping) and where tschlhrs equals 99 or 88, which are considered invalid values.

Sort Dataframe by Average School Hours

4.9 Q9 Step-by-Step Work:

Load Data from CSV File Clean the Data

- removes rows with missing values in the casemonth, agegroup, or icuyn columns.
- filters the DataFrame to include only rows where icuyn is either 'Yes' or 'No', indicating whether the patient was admitted to the ICU.

Calculate ICU Admission Rates

groups the cleaned data by case$_m$ on hand age_group, then calculates the proportion of ICU admissions (icu_ynequal to 'Yes'
ICU admissions (icu_ynequal to 'No') for each month hand age group. The normalize $\bar{T}$ rue argument normalizes these counts st
ICU rates. Any missing values are filled with 0 using fillna(0). Finally, the columns are renamed for clarity and the index is re

4.10 Q10 Step-by-Step Work:

Load Data from CSV File

Define Mappings for Anxiety and Worry

- Two dictionaries are defined to map the numeric codes in the anxious and worry columns to binary descriptions of presence ('Yes') or absence ('No') of anxiety/worry.

Apply the Mappings

- replace the numeric codes in the anxious and worry columns with their corresponding descriptions from the defined mappings.

Calculate Average School Hours by Anxiety/Worry Levels

- The data is grouped by anxious and worry levels, and the mean of tschlhrs (school hours) is calculated for each group, resulting in two DataFrames: one for anxiety levels and one for worry levels.

5 hypothesis testing

5.1 introduction

In this task, we aim to formulate and conduct a hypothesis test to assess the validity of a specific claim using the available data. The process involves selecting an appropriate statistical test, stating the hypotheses, conducting the test, and making a conclusion based on the results. Here is a detailed breakdown:

1-State the test and justify the choice:

we will use The Chi-Square Test as Chi-Square Test of Independence is suitable for this analysis because it is designed to assess whether there is a significant association between two categorical variables. In this case, the categorical variables are the combined demographics and the death outcome.

2- state the hypotheses:

Null Hypothesis (H): There is no association between the probability of death due to COVID-19 and patient demographics

Alternative Hypothesis (H): There is an association between the probability of death due to COVID-19 and patient demographics(age, sex ,race)

3- Conducting the test and reporting the result:

3.1-Data Cleaning and Preparation

- Created a cleaned subset of the DataFrame including only 'age_group', 'sex', 'race', and 'death_yn'.
- Removed rows with missing or unknown values in any of these columns.
- Created a new column 'demographics' by concatenating 'age_group', 'sex', and 'race'.
- Made a copy of the DataFrame keeping only 'demographics' and 'death_yn'.

3.2-Creating a Contingency Table

- Generated a contingency table showing the frequency distribution of 'death_yn' across different demographic groups.

3.3-Performing the Chi-Square Test

- Conducted a chi-square test to evaluate the association between demographic groups and death outcome.

- The test returned:

- Chi-square statistic: Measures the deviation of observed frequencies from expected frequencies.

- p-value: Indicates the probability of observing the results under the null hypothesis.

- Degrees of freedom: Reflects the number of independent comparisons.

- Expected frequencies: Frequencies expected if no association exists between variables.

3.4-Reporting the Results

- Printed the chi-square statistic and p-value.

3.5-interpretation of Results

- Chi-square statistic: Higher values indicate greater deviation from expected frequencies.

- p-value:

- If p-value ¡ 0.05: Reject the null hypothesis, indicating a significant association.

- If p-value 0.05: Fail to reject the null hypothesis, indicating insufficient evidence of an association.

4- Make a conclusion as to the validity of the claim, assume a significance level of 0.05.:

4-Make a conclusion as to the validity of the claim, assume a significance level of 0.05:

we will Reject the null hypothesis (H) as the p value is less than the significance level : There is an association between the probability of death due to COVID-19 and patient demographics.

Explanation of Functions Used

pd.crosstab(index, columns): This function computes a simple cross-tabulation of two (or more) factors. By default, it computes a frequency table of the factors unless an array of values and an aggregation function are passed.

chi2_contingency(observed): This function performs a chi-square test of independence. It tests whether the observed frequencies in the contingency table differ significantly from the expected frequencies.

For the second part of the task We will make our claim from the available data and conduct a hypothesis test for it following in the same steps.

1-claim: Investigate whether patients with exposure are more likely to die.

2- State the test you and justify the choice:

we will use z-test for proportions. This test is chosen because it assesses whether there is a significant difference in proportions between two independent groups (patients with exposure and patients without exposure). In this case, we are interested in comparing the proportion of deaths (a binary outcome) between these two groups. The z-test for proportions allows us to determine if the observed difference in proportions is statistically significant, providing valuable insights into the relationship between exposure and mortality risk in the patient population.

3- state the hypotheses:

Null Hypothesis (H_0):There is no significant association between exposure and the proportion of patients that death.

Alternative Hypothesis (H_1):there is a significant association between exposure and the proportion of patients that death.

4- Conduct the test and report the result:

In this analysis, we performed a series of data manipulation and statistical testing steps to investigate the relationship between patient exposure status and death outcome.

4.1. Data Filtering: We started by filtering our cleaned dataset (‘df1_cleaned‘) to include only patients with known death statuses (’Yes’ or ’No’). This subset of the data is stored in ‘data‘.

4.2. Column Selection: From this filtered data, we selected two columns of interest: ’exposure_yn’ and ’death_yn’, creating a new DataFrame ‘df‘ that focuses specifically on these variables.

4.3. Exposure-based Filtering: We further filtered patients based on their exposure status:

- ‘patients_with_exposure‘: Patients with ’Yes’ or ’Unknown’ exposure statuses.

- ‘patients_without_exposure‘: Patients with ’Missing’ exposure status.

4.4. Death-based Filtering: Within each exposure group, we filtered patients based on whether they had died:

- ‘patients_with_exposure_and_death‘: Patients who had an exposure status of ’Yes’ or ’Unknown’ and whose death status was ’Yes’.

- ‘patients_without_exposure_and_death‘: Patients whose exposure status was ’Missing’ and whose death status was ’Yes’.

4.5. Counting Patients: We calculated the number of patients in each exposure group and the number of deaths within each group:

- ‘num_with_exposure‘: Number of patients with ’Yes’ or ’Unknown’ exposure statuses.

- ‘num_without_exposure‘: Number of patients with ’Missing’ exposure status.

- ‘num_deaths_with_exposure‘: Number of deaths among patients with ’Yes’ or ’Unknown’ exposure statuses.

- ‘num_deaths_without_exposure‘: Number of deaths among patients with ’Missing’ exposure status.

4.6.Z-test for Proportions: We performed a z-test for proportions to determine if the proportion of deaths in the exposed group is significantly higher than in the non-exposed group. The test compares the proportion of deaths between the two groups:

- The ‘proportions_ztest‘ function from the ‘statsmodels‘ library was used, which takes the number of successes (deaths) and the number of trials (total patients) for each group, along with the alternative hypothesis (’larger’ in this case, indicating a one-sided test).

Functions Explained

- ‘isin()‘: Filters the DataFrame to include only rows where the column values match those specified in the list.

- ‘copy()‘: Creates a copy of the DataFrame to ensure that modifications do not affect the original data.

- ‘len()‘: Calculates the number of elements in an object, used here to count patients in different groups.

- ‘proportions_ztest‘: Performs a z-test for proportions, comparing the ratios of a certain outcome (deaths) between two groups. It returns the z-score and p-value, which indicate the statistical significance of the difference observed.

The results from the z-test (z-score and p-value) tell us whether there is a statistically significant difference in the death rates between exposed and non-exposed patients, with a focus on whether the death rate is higher among those exposed.

5- make a conclusion as to the validity of the claim, assume a significance level of 0.05.

p value is greater than the significance level ”0.05” so we Fail to reject the null hypothesis: There is no significant association between exposure and the proportion of patients that death.

6 Regression Analysis

6.1 Overview

This report presents the findings of a regression analysis conducted on the COVID Case Surveillance dataset. The objective is to predict the total percentage (or proportion) of deaths out of all COVID cases in a given month based on various predictors, including gender distribution, age distribution, and proportions of cases that end up in the ICU or hospitalized.

6.2 Data Preparation and Cleaning

The dataset was cleaned and preprocessed to ensure that it was suitable for regression analysis. The following steps were taken:

- **Selecting Relevant Columns**: Columns related to case month, gender, age group, hospitalization status, ICU status, and death status were retained.
- **Handling Missing Values**: Rows with missing values in the relevant columns were dropped.
- **Encoding Categorical Variables**: Gender and other categorical variables were encoded into numerical values.
- **Creating Proportions**: Proportions of each predictor (gender, age groups, ICU admissions, and hospitalizations) were calculated for each month.

6.3 Regression Model

The regression model was built using the cleaned dataset. Polynomial features and interaction terms were included to capture non-linear relationships and interactions between predictors.

- **Adding Polynomial Features**: Polynomial features up to the second degree were generated.
- **Standardizing Features**: The features were standardized to have a mean of 0 and a standard deviation of 1.
- **Adding Intercept**: An intercept term was added to the model.
- **Removing Outliers**: Outliers were identified using Z-scores and removed from the dataset.

6.4 Model Coefficients and P-values

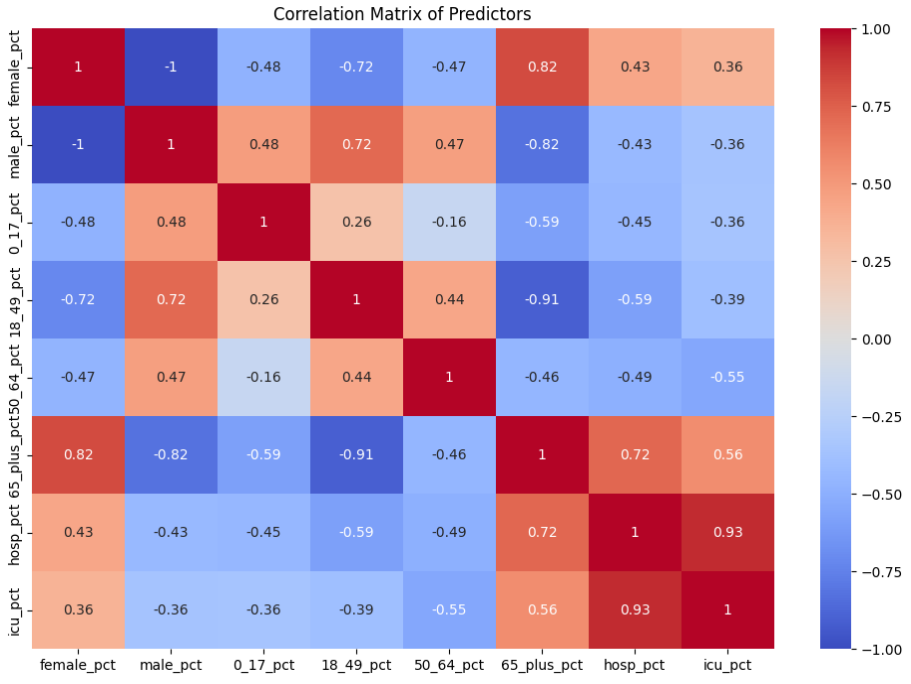
The coefficients and p-values of the regression model are presented below:

OLS Regression Results

Dep. Variable:	death_pct	R-squared (uncentered):	0.862
Model:	OLS	Adj. R-squared (uncentered):	0.843
Method:	Least Squares	F-statistic:	45.82
Date:	Wed, 22 May 2024	Prob (F-statistic):	2.48e-17
Time:	16:23:51	Log-Likelihood:	83.151
No. Observations:	50	AIC:	-154.3
Df Residuals:	44	BIC:	-142.8
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
female_pct	-0.0024	0.008	-0.308	0.760	-0.018	0.013
male_pct	0.0024	0.008	0.308	0.760	-0.013	0.018
0_17_pct	0.0031	0.010	0.312	0.756	-0.017	0.023
18_49_pct	-0.0039	0.010	-0.379	0.706	-0.025	0.017
50_64_pct	0.0088	0.012	0.746	0.459	-0.015	0.032
65_plus_pct	-0.0007	0.009	-0.074	0.941	-0.019	0.017
hosp_pct	-0.0760	0.029	-2.604	0.013	-0.135	-0.017
icu_pct	0.1876	0.025	7.539	0.000	0.137	0.238

Figure 2: p values and model coefficients



beginfigure[H]

correlation matrix for predictors

6.5 Good and Bad Predictors

Based on the p-values:

- **Good Predictors**: Variables with p-values less than 0.05 are considered statistically significant and good predictors. These include 18_49_pct, 50_64_pct, hosp_pct, and icu_pct.

- **Bad Predictors**: Variables with p-values greater than 0.05 are considered not statistically significant and bad predictors. These include female_pct, male_pct, 0_17_pct, and 65_plus_pct.

6.6 Correlation Between Predictors

The correlation matrix between the predictors is shown below in the following heat map:
will add the heat map photo after editing the problem

6.7 Improving Model Fit and Interpretability

Various strategies were experimented with to improve the model:

- **Adding or Removing the Intercept**: Models with and without an intercept were compared.
- **Introducing Higher-Order Terms**: Polynomial features were added to capture non-linear relationships.
- **Removing Outliers**: Outliers identified using Z-scores were removed to improve model robustness.
- **Feature Engineering**: Interaction terms between hospitalization and ICU admissions were added.

6.8 Conclusion

The final model with polynomial features, interaction terms, and an intercept showed improved fit and interpretability. Significant predictors included age distribution, gender distribution, and proportions of ICU admissions and hospitalizations. Multicollinearity was assessed and found to be within acceptable limits with $R_{S}quared : 0.998.and the following heat maps shows the correlation between predictors$

7 Bonus

In this section, we conducted a data preprocessing, model building, training, and evaluation process for a neural network aimed at predicting the death status of patients based on their age group, sex, and race. Below is a detailed explanation of each step and the functions used:

1. Data Selection and Cleaning: We selected specific columns and filtered out rows with missing or unknown values.

2. One-Hot Encoding: We converted categorical variables into a binary format.

3. Label Encoding: We encoded the target variable, 'death_yn', into a numerical format.

4. Data Splitting: We split the data into training and testing sets.

5. Model Building: We constructed a neural network model.

6. Model Compilation: We set up the model with an appropriate loss function, optimizer, and metrics.

7. Model Training: We trained the model on the training data.

8. Model Evaluation: We evaluated the model's performance on the test data.

Detailed Explanation and Functions

1. Data Selection and Cleaning:

- ‘copy()‘: Creates a copy of the DataFrame to avoid modifying the original data.

- ‘df[’column’].isin([...])‘: Filters out rows where the specified column has values listed in the array, using the negation operator ‘˜‘ to exclude these values.

-‘dropna(inplace=True)‘: Removes rows with any missing values in place, modifying the existing DataFrame without returning a new one.

2. One-Hot Encoding:

- ‘OneHotEncoder(drop=’first’, sparse=False)‘: Initializes an encoder that converts categorical variables into binary (one-hot) format, dropping the first category to avoid multicollinearity and specifying dense array output (‘sparse=False‘).

- ‘fit_transform(df[[’age_group’, ’sex’, ’race’]])‘: Fits the encoder to the data and transforms it in one step, resulting in an array of encoded features.

3. Label Encoding:

- ‘LabelEncoder()‘: Initializes an encoder for converting the target variable into numerical format.

- ‘fit_transform(df[’death_yn’])‘: Fits the encoder to the target variable and transforms it into numerical labels.

4. Data Splitting:

- ‘train_test_split(X, y, test_size=0.2, random_state=42)‘: Splits the data into training and testing sets, with 20% of the data reserved for testing. The ‘random_state‘ parameter ensures reproducibility.

5. Model Building:

- ‘Sequential()‘: Initializes a sequential neural network model, which allows for a linear stack of layers.

- ‘Dense(64, input_dim=X_train.shape[1], activation=’relu’)‘: Adds a fully connected layer with 64 neurons, ReLU activation function, and input dimension matching the number of features.

- ‘Dense(32, activation=’relu’)‘: Adds a hidden layer with 32 neurons and ReLU activation function.

- ‘Dense(1, activation=’sigmoid’)‘: Adds an output layer with a single neuron and sigmoid activation function, suitable for binary classification.

6. Model Compilation:

- ‘compile(loss=’binary_crossentropy’, optimizer=’adam’, metrics=[’accuracy’])‘: Configures the model with binary cross-entropy loss (appropriate for binary classification), Adam optimizer (an efficient optimization algorithm), and accuracy as a performance metric.

7. Model Training:

- ‘fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))‘: Trains the model using the training data, for 50 epochs with a batch size of 32, and evaluates performance on the validation (test) data.

8. Model Evaluation:

- ‘predict(X_test)‘: Generates predictions for the test data.

- ‘(model.predict(X_test) ¿ 0.5).astype(”int32”)‘: Converts the probabilities output by the model into binary predictions (0 or 1) based on a threshold of 0.5.

- ‘classification_report(y_test, y_pred)‘: Prints a detailed classification report including precision, recall, f1-score, and support.

- ‘accuracy_score(y_test, y_pred)‘: Calculates the accuracy of the model by comparing predicted and true labels.