U.S. Medical Insurance Costs

April 27, 2023

1 Project Description

For this project, you will be investigating a medical insurance costs dataset in a .csv file using the Python skills that you've developed. This dataset and its parameters will seem familiar if you've done any of the previous Python projects in the data science path.

However, you're now tasked with working with the actual information in the dataset and performing your own independent analysis on real-world data! We will not be providing step-by-step instructions on what to do, but we will provide you with a framework to structure your exploration and analysis.For this project, you will be investigating a medical insurance costs dataset in a .csv file using the Python skills that you've developed. This dataset and its parameters will seem familiar if you've done any of the previous Python projects in the data science path.

However, you're now tasked with working with the actual information in the dataset and performing your own independent analysis on real-world data! We will not be providing step-by-step instructions on what to do, but we will provide you with a framework to structure your exploration and analysis.

2 Project Objectives

- Work locally on your own computer
- Import a dataset into your program
- Analyze a dataset by building out functions or class methods
- Use libraries to assist in your analysis
- Optional: Document and organize your findings
- Optional: Make predictions about a dataset's features based on your findings

3 Project Requirements

- This project was built using Python 3.11 and Jupyter Notebook.
- You will need to install the following libraries:
  - matplotlib (For data visualization, this is not a requirement, but plots won't be shown if you don't have it installed)

4 Project: U.S. Medical Insurance Costs

A dataset containing information on medical insurance costs for individuals in the United States was provided by Codecademy. To learn about the dataset, I first want to explore the data and get a feel for what it contains. For that, I will use python to import the CSV file and print the headers and the number of rows.

I'm also going to save the contents of the CSV file in a list of dictionaries, where each dictionary represents a row of the dataset. I will do this to avoid having to read the CSV file multiple times.

Note: This next cell *needs* to be run first, otherwise the rest of the notebook will not work.

[342]:
```python
import csv

# Modify this if the file is in a different location
FILE_PATH = '../data/insurance.csv'

# Read the CSV file and save the contents in a list of dictionaries
with open(FILE_PATH) as insurance_csv:
    insurance_dict = csv.DictReader(insurance_csv)
    INSURANCE_DATA = list(insurance_dict)

    # Show the information of the dataset
    print('Headers:', insurance_dict.fieldnames)
    print('Number of rows:', len(INSURANCE_DATA))
```

```
Headers: ['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges']
Number of rows: 1338
```

4.1 What I found

From the headers, we can see that the data is organized by the following: (The Data type is not included in the headers, but I will include it in the table below)

Field Name	Data Type
age	int
sex	str
bmi	float
children	int
smoker	str
region	str
charges	float

There are 1338 rows in the dataset.

Additionally, Codecademy provided the following information about the dataset:

- There is no missing data (the dataset has been cleaned too).
- There are seven columns.
- Some columns are numerical while some are categorical.

4.2 What I would change about the dataset

I would change the data type of the `sex` and `smoker` fields to be `bool` instead of `str`. This would make it easier to work with the data in Python. This wasn't done in this project because the focus was on learning how to work with data in Python, not on cleaning the data.

5 Exploring the data

Now that I know how the dataset is organized, I'm going to explore the dataset by exploring different fields and their statistics.

5.0.1 Statistics (Numerical Fields)

First, I want to find the average, median, mode, and standard deviation of each field. This will give me a general idea of the data. Additionally, I will add a boxplot to visualize the data for each field.

Average, median, mode, standard deviation and percentiles To find the average, median, mode, standard deviation and percentiles of each field, I will create functions for each of these statistics.

Average

[343]:
```python
def find_average_on_numeric_field(data: list[dict], field_name: str) -> float:
    """
    Find the average of a numeric field in a list of dictionaries.
    The average is rounded to two decimal places.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the average of.

    Returns:
        float: The average of the field.
    """
    return round(sum([float(row[field_name]) for row in data]) / len(data), 2)
```

Median

[344]:
```python
def find_median_on_numeric_field(data: list[dict], field_name: str) -> float:
    """
    Find the median of a numeric field in a list of dictionaries.
    The median is rounded to two decimal places.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the median of.

    Returns:
        float: The median of the field.
```

```
    """
    sorted_data = sorted([float(row[field_name]) for row in data])
    if len(sorted_data) % 2 == 0:
        calculated_median = (sorted_data[len(sorted_data) // 2] +␣
↪sorted_data[len(sorted_data) // 2 - 1]) / 2
    else:
        calculated_median = sorted_data[len(sorted_data) // 2]

    return round(calculated_median, 2)
```

Mode

[345]:
```
def find_mode_on_numeric_field(data: list[dict], field_name: str):
    """
    Find the mode of a numeric field in a list of dictionaries.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the mode of.

    Returns:
        tuple: The mode of the field and the number of times the mode appears.
    """
    value_counts = {}
    for row in data:
        if float(row[field_name]) in value_counts:
            value_counts[float(row[field_name])] += 1
        else:
            value_counts[float(row[field_name])] = 1

    calculated_mode = max(value_counts, key=value_counts.get)
    return calculated_mode, value_counts[calculated_mode]
```

Standard Deviation

[346]:
```
def find_standard_deviation_on_numeric_field(data: list[dict], field_name: str)␣
↪-> float:
    """
    Find the standard deviation of a numeric field in a list of dictionaries.
    The standard deviation is rounded to two decimal places.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the standard deviation␣
↪of.

    Returns:
        float: The standard deviation of the field.
    """
```

```
    calculated_average = find_average_on_numeric_field(data, field_name)
    sum_of_squared_differences = sum([(float(row[field_name]) -␣
↪calculated_average) ** 2 for row in data])
    return round((sum_of_squared_differences / len(data)) ** 0.5, 2)
```

Percentiles

[347]:
```
def find_percentiles_on_numeric_field(data: list[dict], field_name: str) ->␣
↪tuple[float, float, float]:
    """
    Find the 25th, 50th, and 75th percentiles of a numeric field in a list of␣
↪dictionaries.
    The percentiles are rounded to two decimal places.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the percentiles of.

    Returns:
        tuple: The 25th, 50th, and 75th percentiles of the field.
    """
    sorted_data = sorted([float(row[field_name]) for row in data])

    percentile_25 = round(sorted_data[len(sorted_data) // 4], 2)
    percentile_50 = round(sorted_data[len(sorted_data) // 2], 2)
    percentile_75 = round(sorted_data[len(sorted_data) // 4 * 3], 2)

    return percentile_25, percentile_50, percentile_75
```

Testing the functions Now that I've established the functions, I will use them to find the statistics for each field.

[348]:
```
def find_numeric_field_statistics(numeric_fields: list[str]):
    """
    Find the average, median, mode, standard deviation, and percentiles of a␣
↪list of numeric fields.

    Args:
        numeric_fields (list): A list of numeric fields to find the statistics␣
↪of.
    """
    for numeric_field in numeric_fields:
        average = find_average_on_numeric_field(INSURANCE_DATA, numeric_field)
        median = find_median_on_numeric_field(INSURANCE_DATA, numeric_field)
        mode, mode_count = find_mode_on_numeric_field(INSURANCE_DATA,␣
↪numeric_field)
```

```
        standard_deviation =␣
↪find_standard_deviation_on_numeric_field(INSURANCE_DATA, numeric_field)
        percentiles = find_percentiles_on_numeric_field(INSURANCE_DATA,␣
↪numeric_field)

        print(f'Field: {numeric_field}'
              f'\n\tAverage: {average}'
              f'\n\tMedian: {median}'
              f'\n\tMode: {mode} ({mode_count} times)'
              f'\n\tStandard Deviation: {standard_deviation}'
              f'\n\tPercentiles:'
              f'\n\t\t25th: {percentiles[0]}'
              f'\n\t\t50th: {percentiles[1]}'
              f'\n\t\t75th: {percentiles[2]}'
              f'\n')


NUMERIC_FIELDS = ['age', 'bmi', 'children', 'charges']
find_numeric_field_statistics(NUMERIC_FIELDS)
```

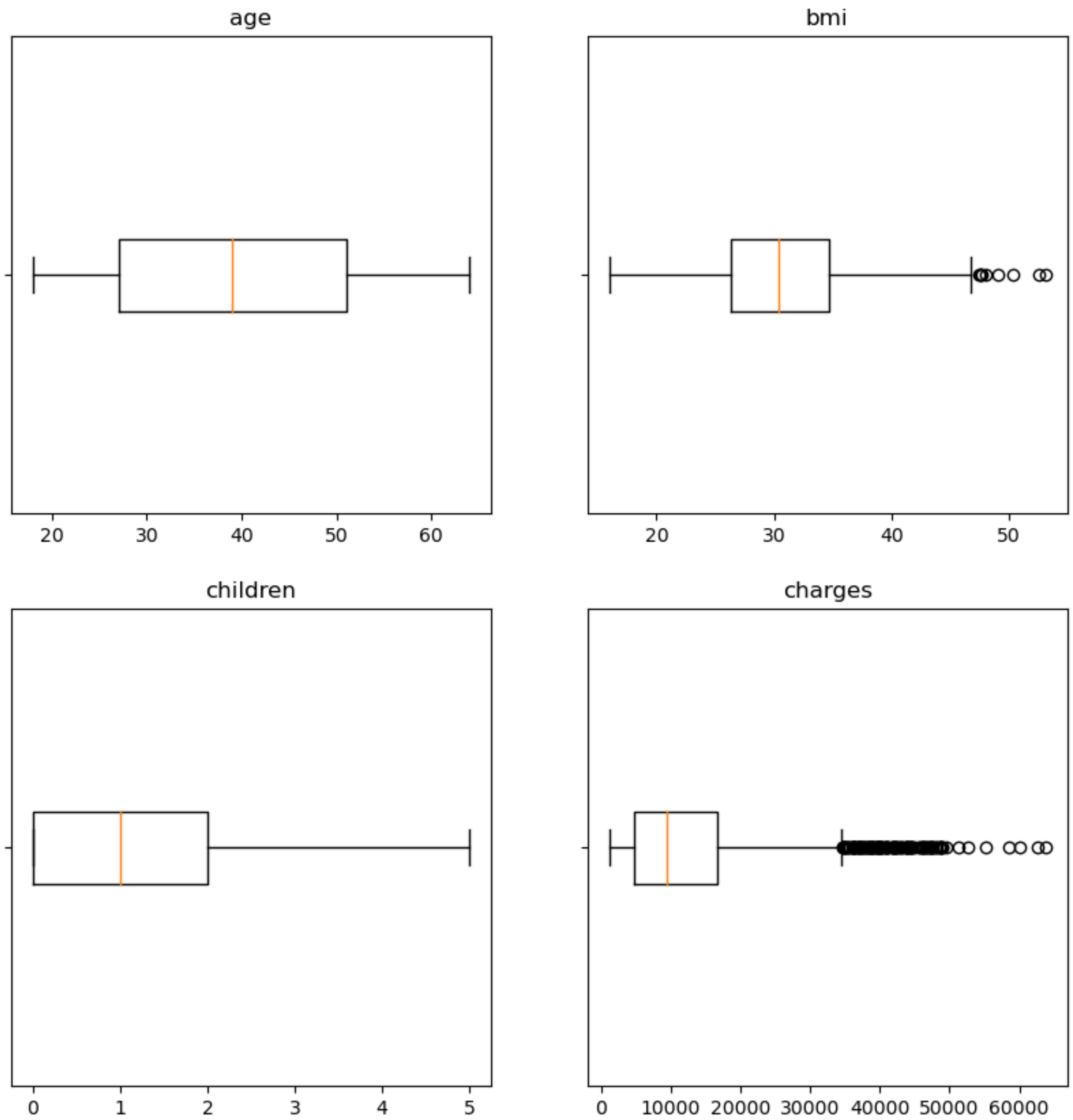
```
Field: age
        Average: 39.21
        Median: 39.0
        Mode: 18.0 (69 times)
        Standard Deviation: 14.04
        Percentiles:
                25th: 27.0
                50th: 39.0
                75th: 51.0

Field: bmi
        Average: 30.66
        Median: 30.4
        Mode: 32.3 (13 times)
        Standard Deviation: 6.1
        Percentiles:
                25th: 26.29
                50th: 30.4
                75th: 34.67

Field: children
        Average: 1.09
        Median: 1.0
        Mode: 0.0 (574 times)
        Standard Deviation: 1.21
        Percentiles:
                25th: 0.0
```

```
                50th: 1.0
                75th: 2.0

Field: charges
        Average: 13270.42
        Median: 9382.03
        Mode: 1639.5631 (2 times)
        Standard Deviation: 12105.48
        Percentiles:
                25th: 4738.27
                50th: 9386.16
                75th: 16586.5
```

Box Plots For visualization purposes (Which is not an original objective of the project), I will create box plots for each of the numeric fields.

I will use the matplotlib library to create the box plots. I will also use matplotlib to create multiple plots later on.

[349]:
```python
def plot_box_plots_for_numerical_fields(data, numeric_fields):
    from matplotlib import pyplot as plt

    fig, axes = plt.subplots(2, 2, figsize=(10, 10))

    for i, numeric_field in enumerate(numeric_fields):
        plot_row = i // 2
        plot_col = i % 2

        values = [float(row[numeric_field.lower()]) for row in data]
        axes[plot_row, plot_col].boxplot(values, vert=False)
        axes[plot_row, plot_col].set_title(numeric_field)
        axes[plot_row, plot_col].set_yticklabels([])

    plt.show()


plot_box_plots_for_numerical_fields(INSURANCE_DATA, NUMERIC_FIELDS)
```



Histograms The last visualization I will create is a histogram for each of the numeric fields. This can further help us visualize the data before finding the relationships between the fields and other tests.

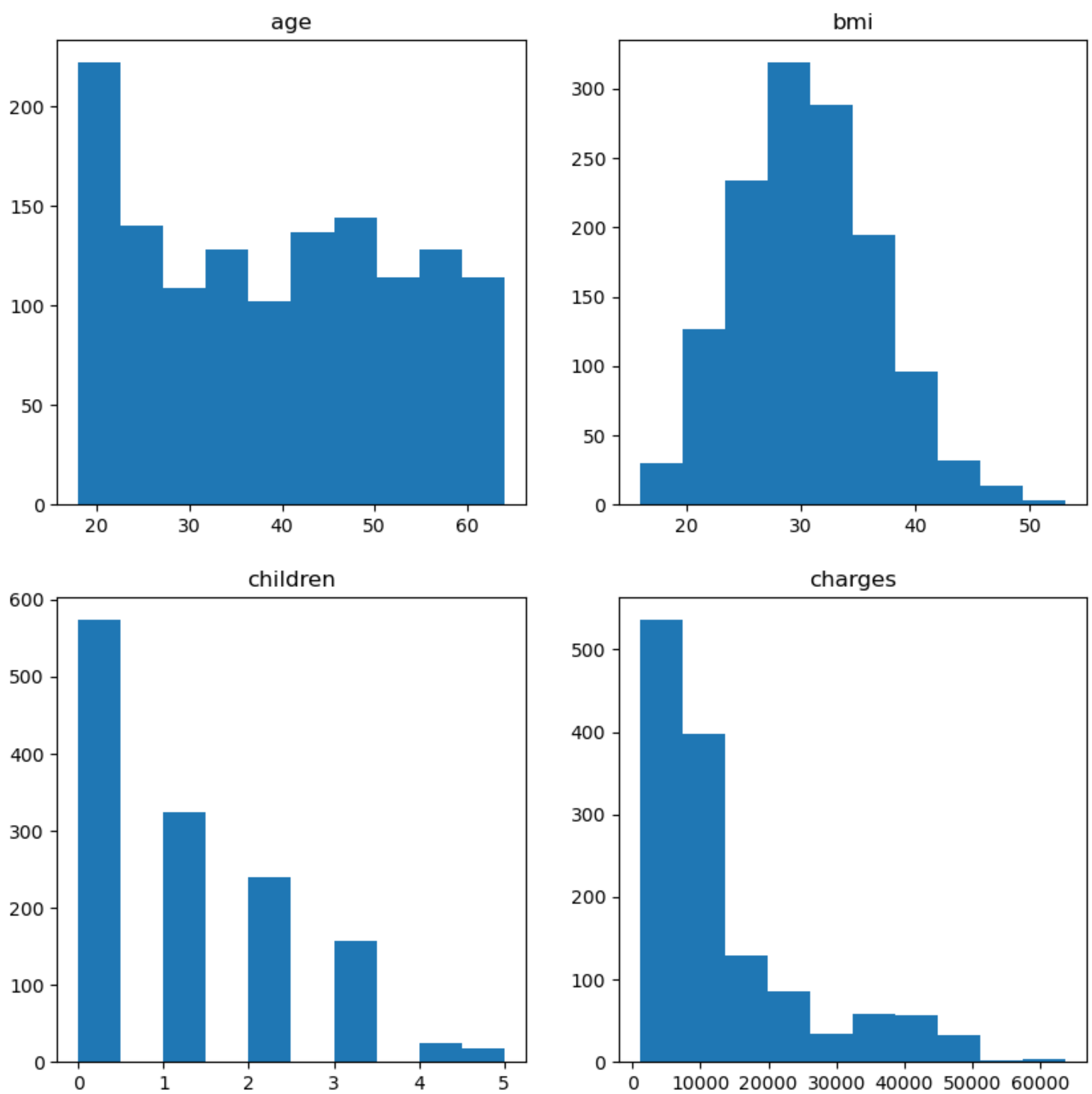
First, I will create a function to create the histograms.

[350]:
```
def plot_histograms_for_numerical_fields(data, numeric_fields):
    from matplotlib import pyplot as plt

    fig, axes = plt.subplots(2, 2, figsize=(10, 10))

    for i, numeric_field in enumerate(numeric_fields):
```

```
        plot_row = i // 2
        plot_col = i % 2

        values = [float(row[numeric_field.lower()]) for row in data]

        axes[plot_row, plot_col].hist(values)
        axes[plot_row, plot_col].set_title(numeric_field)

    plt.show()
```

Then the plots can be created.

[351]:
```
plot_histograms_for_numerical_fields(INSURANCE_DATA, NUMERIC_FIELDS)
```



5.0.2 Statistics (Categorical Fields)

Now that I've found the statistics for the numeric fields, I will find the statistics for the categorical fields.

Unlike the numeric fields, the categorical fields will not have a median, mode, or standard deviation. However, they will have a mode with its corresponding count.

For this, I will create a function to find the mode of a categorical field.

Mode

[352]:
```python
def find_mode_on_categorical_field(data: list[dict], field_name: str):
    """
    Find the mode of a categorical field in a list of dictionaries.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the mode of.

    Returns:
        tuple: The mode of the field and the number of times the mode appears.
    """
    value_counts = {}
    for row in data:
        if row[field_name] in value_counts:
            value_counts[row[field_name]] += 1
        else:
            value_counts[row[field_name]] = 1

    calculated_mode = max(value_counts, key=value_counts.get)
    return calculated_mode, value_counts[calculated_mode]
```

Now that I've created the function, I will use it to find the mode of each categorical field.

[353]:
```python
def find_categorical_field_statistics(categorical_fields: list[str]):
    """
    Find the mode of a categorical field in a list of dictionaries.

    Args:
        categorical_fields (list): A list of categorical fields to find the␣
↪mode of.
    """
    for categorical_field in categorical_fields:
        mode, mode_count = find_mode_on_categorical_field(INSURANCE_DATA,␣
↪categorical_field)
        print(f'Field: {categorical_field}'
              f'\n\tMode: {mode} ({mode_count} times)'
              f'\n')
```

```
CATEGORICAL_FIELDS = ['sex', 'smoker', 'region']
find_categorical_field_statistics(CATEGORICAL_FIELDS)
```

```
Field: sex
        Mode: male (676 times)

Field: smoker
        Mode: no (1064 times)

Field: region
        Mode: southeast (364 times)
```

5.0.3 Relationships

Now that I've found the statistics for the fields, I will find the relationships between the fields.

These relationships will be first order relationships. This means that I will only be looking at the relationship between two fields at a time.

The relationships I will be looking at are: - Age and BMI - Age and Children - Age and Charges - BMI and Children - BMI and Charges - Children and Charges

Additionally, for categorical fields, I will be looking at the relationship between the categorical field's different unique values and the charges, which are: - Sex: "male" or "female" - Smoker: "yes" or "no" - Region: "northeast", "northwest", "southeast", or "southwest"

Relationships (Numeric Fields) To find the relationships between the numeric fields, I will create a function to find the lowest and highest values of a field. This will be used to divide the leading field into groups. The leading field is the field that will be divided into groups. The trailing field is the field that will be compared to the groups of the leading fields.

[354]:
```python
def find_lowest_and_highest_values(data: list[dict], field_name: str):
    """
    Find the lowest and highest values of a field in a list of dictionaries.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the lowest and highest␣
↪values of.

    Returns:
        tuple: The lowest and highest values of the field.
    """
    values = [float(row[field_name]) for row in data]
    return min(values), max(values)
```

I will now make a function that takes a dataset, a leading field, a trailing field, and the number of groups to divide the leading field into. This function will divide the leading field into groups.

[355]:
```python
def divide_leading_field_into_groups(data: list[dict], leading_field_name: str,␣
↪num_groups: int):
    """
    Divide a leading field into groups.

    Args:
        data (list): A list of dictionaries.
        leading_field_name (str): The name of the leading field.
        num_groups (int): The number of groups to divide the leading field into.

    Returns:
        list: A list of tuples, where each tuple contains the lower and upper␣
↪bounds of a group.
    """
    lowest_value, highest_value = find_lowest_and_highest_values(data,␣
↪leading_field_name)

    group_size = (highest_value - lowest_value) // num_groups  # use integer␣
↪division to get an integer group size
    groups = [(lowest_value + (group_size * i), lowest_value + (group_size * (i␣
↪+ 1) - 1))
              for i in range(num_groups)]
    groups.append((lowest_value + (group_size * num_groups), highest_value))

    return groups
```

Now I can implement a function that takes in a dataset, a leading field, a trailing field, and the number of groups to divide the leading field into. This function will return the statistics of the trailing field for each group of the leading fields. I will also implement sub-functions to find the median, mode, standard deviation and percentiles of a list of values to find the statistics of the trailing field.

[356]:
```python
def find_median(values: list[float]):
    values.sort()
    if len(values) % 2 == 0:
        calculated_median = (values[len(values) // 2] + values[len(values) // 2␣
↪- 1]) / 2
    else:
        calculated_median = values[len(values) // 2]
    return calculated_median


def find_mode(values: list[float]):
    value_counts = {}

    for value in values:
        if value in value_counts:
```

```
            value_counts[value] += 1
        else:
            value_counts[value] = 1
    calculated_mode = max(value_counts, key=value_counts.get)
    return calculated_mode


def find_average(values: list[float]):
    return sum(values) / len(values)


def find_standard_deviation(values: list[float]):
    calculated_average = find_average(values)
    return (sum([(value - calculated_average) ** 2 for value in values]) /␣
 ↪len(values)) ** 0.5


def find_percentiles(values: list[float]):
    values.sort()
    percentile_25 = values[len(values) // 4]
    percentile_50 = values[len(values) // 2]
    percentile_75 = values[len(values) // 4 * 3]
    return percentile_25, percentile_50, percentile_75


def find_relationship_between_two_numeric_fields(data: list[dict],
                                                 leading_field_name: str,
                                                 trailing_field_name: str,
                                                 num_groups: int):
    """
    Find the relationship between two numeric fields.

    Args:
        data (list): A list of dictionaries.
        leading_field_name (str): The name of the leading field.
        trailing_field_name (str): The name of the trailing field.
        num_groups (int): The number of groups to divide the leading field into.

    Returns:
        dict: A dictionary where the keys are the groups of the leading fields␣
 ↪and the values are the statistics of the trailing field.
    """
    groups = divide_leading_field_into_groups(data, leading_field_name,␣
 ↪num_groups)
    calculated_statistics = {}

    for group_name in groups:
```

```
        values = [float(row[trailing_field_name]) for row in data if
                  group_name[0] <= float(row[leading_field_name]) <=␣
↪group_name[1]]

        if len(values) != 0:
            calculated_statistics[group_name] = [
                sum(values) / len(values),
                find_median(values),
                find_mode(values),
                find_standard_deviation(values),
                find_percentiles(values)
            ]

    return calculated_statistics
```

Finally, I created a function to standardize outputting the statistics.

```
[357]: def show_relationship_statistics(data: list[dict],
                                 leading_field_name: str,
                                 trailing_field_name: str,
                                 num_groups: int):
    """
    Show the relationship statistics between two numeric fields.

    Args:
        data (list): A list of dictionaries.
        leading_field_name (str): The name of the leading field.
        trailing_field_name (str): The name of the trailing field.
        num_groups (int): The number of groups to divide the leading field into.
    """
    calculated_statistics = find_relationship_between_two_numeric_fields(data,
                                                                         ␣
↪leading_field_name,
                                                                         ␣
↪trailing_field_name,
                                                                         ␣
↪num_groups)
    for group, statistics in calculated_statistics.items():
        print(f'Group: {group}'
              f'\n\tAverage: {statistics[0]}'
              f'\n\tMedian: {statistics[1]}'
              f'\n\tMode: {statistics[2]}'
              f'\n\tStandard Deviation: {statistics[3]}'
              f'\n\tPercentiles:'
              f'\n\t\t25th: {statistics[4][0]}'
              f'\n\t\t50th: {statistics[4][1]}'
              f'\n\t\t75th: {statistics[4][2]}'
```

```
f'\n')
```

Age and BMI

[358]:
```
show_relationship_statistics(INSURANCE_DATA, 'age', 'bmi', 10)
```

```
Group: (18.0, 21.0)
        Average: 29.81260309278351
        Median: 30.072499999999998
        Mode: 30.59
        Standard Deviation: 6.274752145447225
        Percentiles:
                25th: 25.46
                50th: 30.115
                75th: 33.88

Group: (22.0, 25.0)
        Average: 30.344687499999992
        Median: 29.877499999999998
        Mode: 23.18
        Standard Deviation: 6.346795425971242
        Percentiles:
                25th: 25.84
                50th: 29.925
                75th: 33.99

Group: (26.0, 29.0)
        Average: 29.407162162162177
        Median: 29.64
        Mode: 22.515
        Standard Deviation: 5.959988308744414
        Percentiles:
                25th: 24.75
                50th: 29.64
                75th: 33.0

Group: (30.0, 33.0)
        Average: 30.798396226415097
        Median: 29.92
        Mode: 27.645
        Standard Deviation: 6.2990107619255316
        Percentiles:
                25th: 26.62
                50th: 30.03
                75th: 35.3

Group: (34.0, 37.0)
        Average: 30.562029702970314
```

```
        Median: 29.92
        Mode: 27.74
        Standard Deviation: 5.893596339332965
        Percentiles:
                25th: 26.885
                50th: 29.92
                75th: 34.32

Group: (38.0, 41.0)
        Average: 30.164519230769237
        Median: 29.9125
        Mode: 19.95
        Standard Deviation: 5.956269544290752
        Percentiles:
                25th: 26.315
                50th: 29.925
                75th: 34.1

Group: (42.0, 45.0)
        Average: 30.27968181818181
        Median: 29.95
        Mode: 38.06
        Standard Deviation: 5.690080814143821
        Percentiles:
                25th: 25.7
                50th: 30.0
                75th: 34.96

Group: (46.0, 49.0)
        Average: 31.067695652173924
        Median: 30.3
        Mode: 32.3
        Standard Deviation: 6.026922074324594
        Percentiles:
                25th: 27.1
                50th: 30.3
                75th: 34.6

Group: (50.0, 53.0)
        Average: 31.549304347826084
        Median: 31.635
        Mode: 32.3
        Standard Deviation: 6.457158746793006
        Percentiles:
                25th: 26.41
                50th: 31.635
                75th: 36.2
```

```
Group: (54.0, 57.0)
        Average: 31.404150943396214
        Median: 31.39
        Mode: 32.775
        Standard Deviation: 5.9281545010683425
        Percentiles:
                25th: 27.645
                50th: 31.54
                75th: 34.21

Group: (58.0, 64.0)
        Average: 31.903140243902442
        Median: 32.0125
        Mode: 32.965
        Standard Deviation: 5.616709748693034
        Percentiles:
                25th: 27.55
                50th: 32.015
                75th: 36.385
```

Age and Children

[359]:
```python
show_relationship_statistics(INSURANCE_DATA, 'age', 'children', 10)
```

```
Group: (18.0, 21.0)
        Average: 0.5515463917525774
        Median: 0.0
        Mode: 0.0
        Standard Deviation: 1.015306911301437
        Percentiles:
                25th: 0.0
                50th: 0.0
                75th: 1.0

Group: (22.0, 25.0)
        Average: 0.8660714285714286
        Median: 0.0
        Mode: 0.0
        Standard Deviation: 1.2355994475953602
        Percentiles:
                25th: 0.0
                50th: 0.0
                75th: 2.0

Group: (26.0, 29.0)
        Average: 1.1441441441441442
        Median: 1.0
        Mode: 0.0
```

```
        Standard Deviation: 1.1456682761787047
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (30.0, 33.0)
        Average: 1.4433962264150944
        Median: 1.0
        Mode: 1.0
        Standard Deviation: 1.2520186619204532
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (34.0, 37.0)
        Average: 1.396039603960396
        Median: 1.0
        Mode: 1.0
        Standard Deviation: 1.126452487392309
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (38.0, 41.0)
        Average: 1.6634615384615385
        Median: 1.0
        Mode: 1.0
        Standard Deviation: 1.260422524722722
        Percentiles:
                25th: 1.0
                50th: 1.0
                75th: 2.0

Group: (42.0, 45.0)
        Average: 1.3363636363636364
        Median: 1.0
        Mode: 2.0
        Standard Deviation: 1.0977437416419413
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (46.0, 49.0)
        Average: 1.4521739130434783
```

```
        Median: 1.0
        Mode: 1.0
        Standard Deviation: 1.21041479243863
        Percentiles:
                25th: 1.0
                50th: 1.0
                75th: 2.0

Group: (50.0, 53.0)
        Average: 1.2869565217391303
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.1924780000982556
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (54.0, 57.0)
        Average: 0.9528301886792453
        Median: 0.5
        Mode: 0.0
        Standard Deviation: 1.1523730681273525
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (58.0, 64.0)
        Average: 0.6341463414634146
        Median: 0.0
        Mode: 0.0
        Standard Deviation: 1.0477873914151505
        Percentiles:
                25th: 0.0
                50th: 0.0
                75th: 1.0
```

Age and Charges

[360]:
```
show_relationship_statistics(INSURANCE_DATA, 'age', 'charges', 10)
```

```
Group: (18.0, 21.0)
        Average: 8138.613823293813
        Median: 2202.284475
        Mode: 1639.5631
        Standard Deviation: 10954.26828258034
        Percentiles:
```

```
                25th: 1705.6245
                50th: 2203.47185
                75th: 12890.05765

Group: (22.0, 25.0)
        Average: 10729.783528571428
        Median: 3232.7784
        Mode: 1664.9996
        Standard Deviation: 12817.953986850394
        Percentiles:
                25th: 2464.6188
                50th: 3238.4357
                75th: 18033.9679

Group: (26.0, 29.0)
        Average: 9445.678326756759
        Median: 4058.71245
        Mode: 2302.3
        Standard Deviation: 10619.755194189396
        Percentiles:
                25th: 3353.284
                50th: 4058.71245
                75th: 15006.57945

Group: (30.0, 33.0)
        Average: 11128.321890377358
        Median: 4990.514125
        Mode: 3260.199
        Standard Deviation: 12080.466090870115
        Percentiles:
                25th: 4347.02335
                50th: 5031.26955
                75th: 16776.30405

Group: (34.0, 37.0)
        Average: 13269.712696039604
        Median: 6198.7518
        Mode: 3935.1799
        Standard Deviation: 12752.544343047906
        Percentiles:
                25th: 5240.765
                50th: 6198.7518
                75th: 19496.71917

Group: (38.0, 41.0)
        Average: 10341.599359519229
        Median: 6867.2203
        Mode: 5383.536
```

```
        Standard Deviation: 8724.680874301683
        Percentiles:
                25th: 6282.235
                50th: 6875.961
                75th: 8162.71625

Group: (42.0, 45.0)
        Average: 15737.673376181818
        Median: 8360.443
        Mode: 5966.8874
        Standard Deviation: 13126.805586572693
        Percentiles:
                25th: 7441.053
                50th: 8410.04685
                75th: 19964.7463

Group: (46.0, 49.0)
        Average: 14849.841782869564
        Median: 9414.92
        Mode: 7147.105
        Standard Deviation: 10674.29291274941
        Percentiles:
                25th: 8556.907
                50th: 9414.92
                75th: 20878.78443

Group: (50.0, 53.0)
        Average: 16408.95999017392
        Median: 10579.711
        Mode: 8442.667
        Standard Deviation: 11447.280606787917
        Percentiles:
                25th: 9722.7695
                50th: 10579.711
                75th: 21195.818

Group: (54.0, 57.0)
        Average: 16639.69539867924
        Median: 11835.691125000001
        Mode: 9850.432
        Standard Deviation: 10998.714477941478
        Percentiles:
                25th: 10982.5013
                50th: 11840.77505
                75th: 13047.33235

Group: (58.0, 64.0)
        Average: 19766.124609512193
```

```
        Median: 13884.0765
        Mode: 11345.519
        Standard Deviation: 11859.623091045036
        Percentiles:
                25th: 12815.44495
                50th: 13887.204
                75th: 25678.77845
```

BMI and Children

[361]:
```
show_relationship_statistics(INSURANCE_DATA, 'bmi', 'children', 10)
```

```
Group: (15.96, 17.96)
        Average: 1.0666666666666667
        Median: 1.0
        Mode: 2.0
        Standard Deviation: 0.8537498983243798
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (18.96, 20.96)
        Average: 1.075
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.1042531412678886
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (21.96, 23.96)
        Average: 1.0786516853932584
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.2290922849684918
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (24.96, 26.96)
        Average: 0.9784172661870504
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.1596800966681946
        Percentiles:
```

```
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (27.96, 29.96)
        Average: 1.1575757575757575
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.3024523030946464
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (30.96, 32.96)
        Average: 1.1125827814569536
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.1011315453810788
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (33.96, 35.96)
        Average: 1.0336134453781514
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.159054766357423
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (36.96, 38.96)
        Average: 1.2531645569620253
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.1414886397565063
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (39.96, 41.96)
        Average: 0.8372093023255814
        Median: 0.0
        Mode: 0.0
```

```
        Standard Deviation: 1.2562445428901865
        Percentiles:
                25th: 0.0
                50th: 0.0
                75th: 1.0

Group: (42.96, 44.96)
        Average: 1.1428571428571428
        Median: 1.0
        Mode: 0.0
        Standard Deviation: 1.124858267715973
        Percentiles:
                25th: 0.0
                50th: 1.0
                75th: 2.0

Group: (45.96, 53.13)
        Average: 1.375
        Median: 1.0
        Mode: 1.0
        Standard Deviation: 1.2686114456365274
        Percentiles:
                25th: 1.0
                50th: 1.0
                75th: 2.0
```

BMI and Charges

[362]:
```
show_relationship_statistics(INSURANCE_DATA, 'bmi', 'charges', 10)
```

```
Group: (15.96, 17.96)
        Average: 7576.420216666668
        Median: 3732.6251
        Mode: 1621.3402
        Standard Deviation: 8086.446267652353
        Percentiles:
                25th: 2585.269
                50th: 3732.6251
                75th: 6640.54485

Group: (18.96, 20.96)
        Average: 8234.163145000002
        Median: 6304.47025
        Mode: 1241.565
        Standard Deviation: 6427.82919466062
        Percentiles:
                25th: 3208.787
                50th: 6753.038
```

```
                75th: 14571.8908

Group: (21.96, 23.96)
        Average: 10113.253969213481
        Median: 8252.2843
        Mode: 1121.8739
        Standard Deviation: 7877.219784725417
        Percentiles:
                25th: 3484.331
                50th: 8252.2843
                75th: 14426.07385

Group: (24.96, 26.96)
        Average: 10791.23693151079
        Median: 8442.667
        Mode: 1615.7667
        Standard Deviation: 8248.254337111828
        Percentiles:
                25th: 4239.89265
                50th: 8442.667
                75th: 14256.1928

Group: (27.96, 29.96)
        Average: 10743.9750006
        Median: 8516.829
        Mode: 1253.936
        Standard Deviation: 7807.74025123305
        Percentiles:
                25th: 4564.19145
                50th: 8516.829
                75th: 13770.0979

Group: (30.96, 32.96)
        Average: 14817.583683443712
        Median: 10269.46
        Mode: 1526.312
        Standard Deviation: 13286.938836876201
        Percentiles:
                25th: 5148.5526
                50th: 10269.46
                75th: 16069.08475

Group: (33.96, 35.96)
        Average: 16397.59603378151
        Median: 8596.8278
        Mode: 1137.011
        Standard Deviation: 15956.543283545772
        Percentiles:
```

```
                25th: 3987.926
                50th: 8596.8278
                75th: 34779.615

Group: (36.96, 38.96)
        Average: 16440.953760506334
        Median: 10226.2842
        Mode: 1141.4451
        Standard Deviation: 15137.013500490048
        Percentiles:
                25th: 5428.7277
                50th: 10226.2842
                75th: 20462.99766

Group: (39.96, 41.96)
        Average: 15958.655700000001
        Median: 10602.385
        Mode: 1146.7966
        Standard Deviation: 15382.15863463989
        Percentiles:
                25th: 5438.7491
                50th: 10602.385
                75th: 15555.18875

Group: (42.96, 44.96)
        Average: 13007.450939285714
        Median: 9300.212925
        Mode: 1149.3959
        Standard Deviation: 12767.368416648085
        Percentiles:
                25th: 4753.6368
                50th: 9541.69555
                75th: 11576.13

Group: (45.96, 53.13)
        Average: 18139.171181874997
        Median: 9649.23785
        Mode: 1163.4627
        Standard Deviation: 19127.060022770493
        Percentiles:
                25th: 6435.6237
                50th: 9748.9106
                75th: 44501.3982
```

Children and Charges

[363]:
```
show_relationship_statistics(INSURANCE_DATA, 'children', 'charges', 10)
```

```
Group: (0.0, 5.0)
        Average: 13270.422265141257
        Median: 9382.033
        Mode: 1639.5631
        Standard Deviation: 12105.484975561612
        Percentiles:
                25th: 4738.2682
                50th: 9386.1613
                75th: 16586.49771
```

Relationships (Categorical Fields) For this section, I will create a function that takes in the dataset and the field name and returns the statistics by using the functions for numeric fields I created earlier. This is possible since we're only looking at the relationship between the field and the charges.

[364]:
```python
def find_statistics_on_charges_for_categorical_field(data: list[dict],␣
↪field_name: str):
    """
    Find the average, median, mode, standard deviation and percentiles of the␣
↪charges for each value of a categorical field in a list of dictionaries.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the statistics of.

    Returns:
        dict: A dictionary with the values of the categorical field as keys and␣
↪a list of the average, median, mode, and standard deviation of the charges␣
↪for each value of the categorical field as values.
    """
    # Find the unique values of the field
    unique_values = set([row[field_name] for row in data])

    # Create a dictionary to store the statistics
    statistics = {}

    # Find the average, median, mode, and standard deviation of the charges for␣
↪each value of the categorical field
    for value in unique_values:
        statistics[value] = {}
        statistics[value]['average'] = find_average_on_numeric_field(
            [row for row in data if row[field_name] == value], 'charges')
        statistics[value]['median'] = find_median_on_numeric_field(
            [row for row in data if row[field_name] == value], 'charges')
        statistics[value]['mode'] = find_mode_on_numeric_field(
            [row for row in data if row[field_name] == value], 'charges')
```

```
        statistics[value]['standard deviation'] =␣
↪find_standard_deviation_on_numeric_field(
            [row for row in data if row[field_name] == value], 'charges')
        statistics[value]['percentiles'] = find_percentiles_on_numeric_field(
            [row for row in data if row[field_name] == value], 'charges')

    return statistics
```

Now that I've created the function, I will use it to find the statistics for each categorical field.

```
[365]: def show_relationship_statistics_for_categorical_field(data: list[dict],␣
↪field_name: str):
    """
    Show the relationship statistics between a categorical field and the␣
↪charges.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to find the statistics of.
    """
    calculated_statistics =␣
↪find_statistics_on_charges_for_categorical_field(data, field_name)

    for value, statistics in calculated_statistics.items():
        print(f'Value: {value}'
              f'\n\tAverage: {statistics["average"]}'
              f'\n\tMedian: {statistics["median"]}'
              f'\n\tMode: {statistics["mode"]}'
              f'\n\tStandard Deviation: {statistics["standard deviation"]}'
              f'\n\tPercentiles:'
              f'\n\t\t25th: {statistics["percentiles"][0]}'
              f'\n\t\t50th: {statistics["percentiles"][1]}'
              f'\n\t\t75th: {statistics["percentiles"][2]}'
              f'\n')
```

We can also plot the statistics for each categorical field using a box plot. For this, I will create a function.

```
[366]: def create_box_plot_for_categorical_field(data: list[dict], field_name: str):
    """
    Create a box plot for each value of a categorical field in a list of␣
↪dictionaries.

    Args:
        data (list): A list of dictionaries.
        field_name (str): The name of the field to create the box plot for.
    """
    from matplotlib import pyplot as plt
```

```python
    unique_values = set([row[field_name] for row in data])
    fig, axes = plt.subplots(1, len(unique_values), figsize=(10, 5))

    for i, value in enumerate(unique_values):
        values = [float(row['charges']) for row in data if row[field_name] ==␣
↪value]
        axes[i].boxplot(values, vert=False)
        axes[i].set_title(value)
        axes[i].set_yticklabels([])

    plt.show()


for field in CATEGORICAL_FIELDS:
    create_box_plot_for_categorical_field(INSURANCE_DATA, field)
```

5.0.4 Conclusion

In this notebook, I've found the statistics for the fields in the dataset and found the relationships between the fields.

The statistics I found were: - Average - Median - Mode - Standard Deviation - Percentiles

The relationships I explored were: - Age and BMI - Age and Children - Age and Charges - BMI and Children - BMI and Charges

I also found the relationship between the categorical fields and the charges.

This is the end of the project; I will not be analyzing the data that I've found, since that is not the purpose of this project.