

Air B

April 30, 2024

Sheena Taylor is a data scientist specializing in machine learning and data analysis. She has a strong background in Python and has experience working with various data science libraries such as NumPy, Pandas, and Scikit-learn. Sheena is known for her ability to clean and preprocess data, build predictive models, and communicate her findings effectively to stakeholders.

The purpose of the code is to explore the relationship between various factors, such as age, region, and customer status, and the number of patents held by individuals or companies. It also aims to build a statistical model to predict the number of patents based on these factors.

The code takes a single input, which is a CSV file named “blueprinty.csv”. This file likely contains data about individuals or companies, including information such as their age, region, customer status, and the number of patents they hold.

The output of the code includes various visualizations and statistical summaries, as well as the results of a Poisson regression model that predicts the number of patents based on the input variables.

Here’s how the code achieves its purpose:

It starts by importing the necessary Python libraries, such as pandas for data manipulation, numpy for numerical operations, matplotlib and seaborn for data visualization, and statsmodels for statistical modeling.

The code reads the “blueprinty.csv” file into a pandas DataFrame called “blueprinty”.

It performs exploratory data analysis by printing the first few rows of the data, displaying information about the data types and columns, and calculating summary statistics like mean, median, and quartiles.

The code checks for missing values in the dataset.

It creates a histogram to visualize the distribution of the number of patents, grouped by customer status (whether the individual or company is a customer or not).

The code calculates and prints the mean number of patents for customers and non-customers separately.

It analyzes the distribution of regions and the mean age for customers and non-customers.

The code creates new columns in the DataFrame, such as “age_squared” and dummy variables for different regions, to be used as input variables for the regression model.

It fits a Poisson regression model using the statsmodels library, with the number of patents as the dependent variable and age, age squared, region dummies, and customer status as independent variables.

The code prints a summary of the regression results, including the estimated coefficients and their statistical significance.

It interprets the exponential of the coefficients, which represent the multiplicative effect of each variable on the expected number of patents.

Finally, the code creates additional visualizations to explore the distributions of age and region by customer status.

The code follows a logical flow, starting with data loading and exploration, followed by data pre-processing and feature engineering, and then building and interpreting the statistical model. It performs various data transformations, such as creating dummy variables and calculating squared terms, to prepare the data for the regression analysis.

The dataset has 1500 rows (entries). The “patents” column contains integer values ranging from 0 to 16, with a mean of around 3.68. The “age” column contains float values ranging from 9 to 49, with a mean of around 26.36. The “iscustomer” column is binary, with 0 representing non-customers and 1 representing customers. Around 13.13% of the entries are customers. There are no missing values in the dataset, as indicated by the “0” values in the last line of the output.

The purpose of the code is to explore the relationship between the number of patents held by individuals and their customer status, as well as other factors such as age and region. It also aims to build a statistical model to predict the number of patents based on these variables.

The code takes a single input, which is a CSV file named “blueprint.csv”. This file is assumed to contain data with columns representing the number of patents, customer status (iscustomer), age, and region for a set of individuals.

The output of the code includes:

Exploratory data analysis results, such as summary statistics, missing value checks, and visualizations of the distribution of patents and other variables by customer status. Comparison of means and distributions of patents, age, and region between customers and non-customers. A fitted Poisson regression model that relates the number of patents to age, age squared, region (encoded as dummy variables), and customer status. The model coefficients and their exponential values, which can be interpreted as the multiplicative effect of each variable on the expected number of patents. Additional visualizations of the distribution of age and region by customer status. To achieve its purpose, the code follows these steps:

Import necessary Python libraries for data manipulation, visualization, and statistical modeling. Read the “blueprint.csv” file into a pandas DataFrame named “blueprint”. Perform exploratory data analysis by printing the first few rows, data types, summary statistics, and checking for missing values. Visualize the distribution of the number of patents by customer status using a histogram. Compare the mean number of patents between customers and non-customers. Analyze the distribution of regions and mean age by customer status. Create new columns in the DataFrame for age squared and region dummy variables. Fit a Poisson regression model using the statsmodels library, with the number of patents as the dependent variable and age, age squared, region dummies, and customer status as independent variables. Print the model summary and interpret the coefficients by taking their exponential. Visualize the distribution of age and region by customer status using histograms and bar plots. The key logic flows and data transformations happening in the code include:

Exploratory data analysis to understand the characteristics of the dataset and identify potential

relationships between variables. Creation of new features (age squared and region dummies) to capture non-linear effects and categorical variables in the regression model. Fitting a Poisson regression model, which is suitable for modeling count data like the number of patents. Interpretation of model coefficients by taking their exponential, which provides the multiplicative effect of each variable on the expected number of patents.

```
[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

blueprinty = pd.read_csv("/home/jovyan/Desktop/Quarto 1/homework 2/blueprinty.
    ↪CSV")

print(blueprinty.head())
print(blueprinty.info())
print(blueprinty.describe())

print(blueprinty.isnull().sum())

# Visualize distribution of number of patents by customer status
plt.figure(figsize=(8, 6))
sns.histplot(data=blueprinty, x='patents', hue='iscustomer', bins=20)
plt.xlabel('Number of Patents')
plt.ylabel('Count')
plt.title('Distribution of Number of Patents by Customer Status')
plt.show()

# Compare means of number of patents by customer status
print("Mean number of patents for customers:",
    ↪blueprinty[blueprinty['iscustomer']==1]['patents'].mean())
print("Mean number of patents for non-customers:",
    ↪blueprinty[blueprinty['iscustomer']==0]['patents'].mean())

# Compare regions and ages by customer status
print("Region distribution for customers:")
print(blueprinty[blueprinty['iscustomer']==1]['region'].
    ↪value_counts(normalize=True))
print("Region distribution for non-customers:")
print(blueprinty[blueprinty['iscustomer']==0]['region'].
    ↪value_counts(normalize=True))

print("Mean age for customers:", blueprinty[blueprinty['iscustomer']==1]['age'].
    ↪mean())
```

```

print("Mean age for non-customers:",
      blueprinty[blueprinty['iscustomer']==0]['age'].mean())

# Add missing columns to dataframe
blueprinty['age_squared'] = blueprinty['age']**2
blueprinty['region_Midwest'] = (blueprinty['region'] == 'Midwest').astype(int)
blueprinty['region_Northeast'] = (blueprinty['region'] == 'Northeast').
    ↳astype(int)
blueprinty['region_South'] = (blueprinty['region'] == 'South').astype(int)
blueprinty['region_Southwest'] = (blueprinty['region'] == 'Southwest').
    ↳astype(int)

# Fit Poisson regression model
poisson_model = sm.GLM(blueprinty['patents'],
                       sm.add_constant(blueprinty[['age', 'age_squared',
    ↳'region_Midwest',
    ↳'region_Northeast',
    ↳'region_South',
    ↳'region_Southwest',
    ↳'iscustomer']])),
                       family=sm.families.Poisson())
poisson_results = poisson_model.fit()
print(poisson_results.summary())

# Interpret model coefficients
print("Exponential of coefficients:")
print(np.exp(poisson_results.params))

plt.figure(figsize=(8, 6))
sns.histplot(data=blueprinty, x='age', hue='iscustomer', bins=20)
plt.xlabel('Age')
plt.ylabel('Count')
plt.title('Distribution of Age by Customer Status')
plt.show()

plt.figure(figsize=(8, 6))
sns.countplot(data=blueprinty, x='region', hue='iscustomer')
plt.xlabel('Region')
plt.ylabel('Count')
plt.title('Distribution of Region by Customer Status')
plt.show()

```

	Unnamed: 0	patents	region	age	iscustomer
0	1	0	Midwest	32.5	0
1	786	3	Southwest	37.5	0
2	348	4	Northwest	27.0	1
3	927	3	Northeast	24.5	0

```

4          830          3 Southwest  37.0          0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1500 entries, 0 to 1499
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Unnamed: 0      1500 non-null   int64
1   patents         1500 non-null   int64
2   region          1500 non-null   object
3   age             1500 non-null   float64
4   iscustomer      1500 non-null   int64
dtypes: float64(1), int64(3), object(1)
memory usage: 58.7+ KB
None

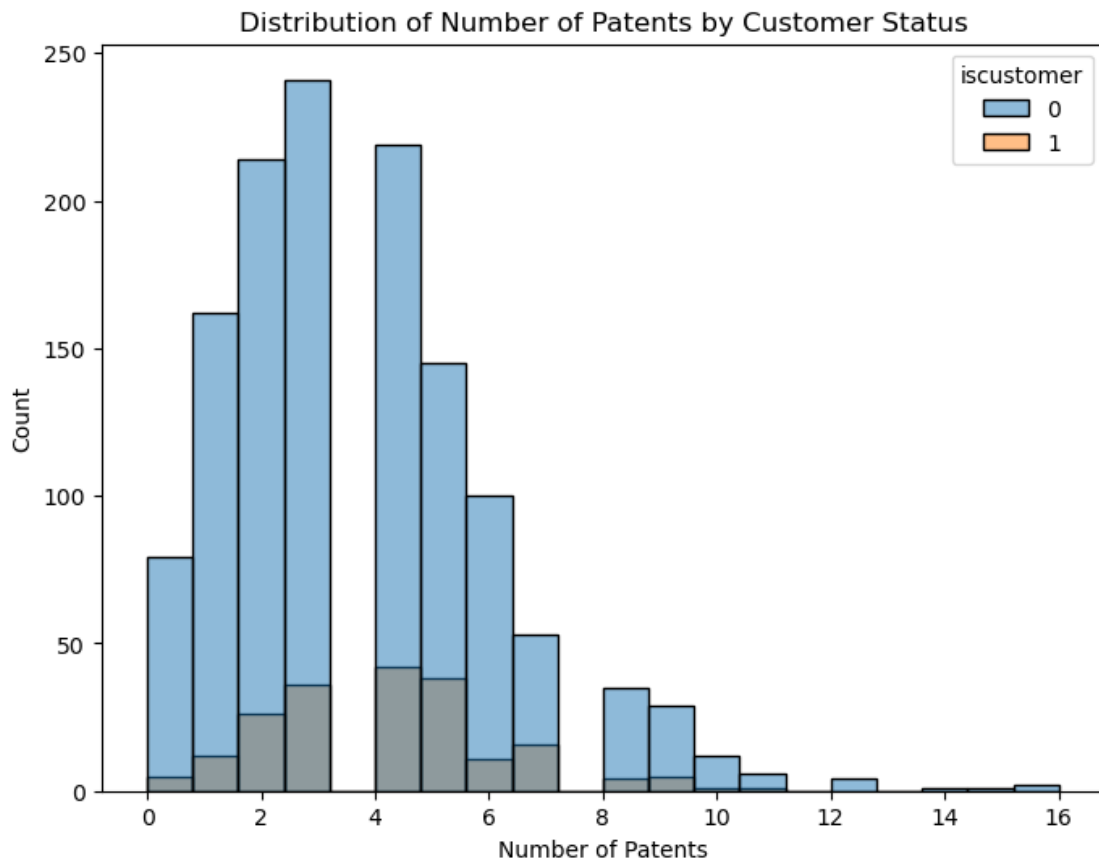
```

	Unnamed: 0	patents	age	iscustomer
count	1500.000000	1500.000000	1500.000000	1500.000000
mean	750.500000	3.684667	26.357667	0.131333
std	433.157015	2.352500	7.242528	0.337877
min	1.000000	0.000000	9.000000	0.000000
25%	375.750000	2.000000	21.000000	0.000000
50%	750.500000	3.000000	26.000000	0.000000
75%	1125.250000	5.000000	31.625000	0.000000
max	1500.000000	16.000000	49.000000	1.000000

```

Unnamed: 0    0
patents       0
region        0
age           0
iscustomer    0
dtype: int64

```



```

Mean number of patents for customers: 4.091370558375634
Mean number of patents for non-customers: 3.6231772831926325
Region distribution for customers:
region
Northeast    0.573604
Southwest    0.157360
South        0.101523
Midwest      0.086294
Northwest    0.081218
Name: proportion, dtype: float64
Region distribution for non-customers:
region
Northeast    0.374520
Southwest    0.204144
Midwest      0.158864
Northwest    0.131236
South        0.131236
Name: proportion, dtype: float64
Mean age for customers: 24.1497461928934
Mean age for non-customers: 26.691481197237145

```

Generalized Linear Model Regression Results

```

=====
Dep. Variable:          patents    No. Observations:          1500
Model:                  GLM        Df Residuals:              1492
Model Family:           Poisson    Df Model:                  7
Link Function:          Log        Scale:                    1.0000
Method:                 IRLS       Log-Likelihood:          -3275.9
Date:                   Tue, 30 Apr 2024    Deviance:                2178.8
Time:                   15:43:09    Pearson chi2:            2.11e+03
No. Iterations:         5          Pseudo R-squ. (CS):      0.1152
Covariance Type:        nonrobust
=====

```

```

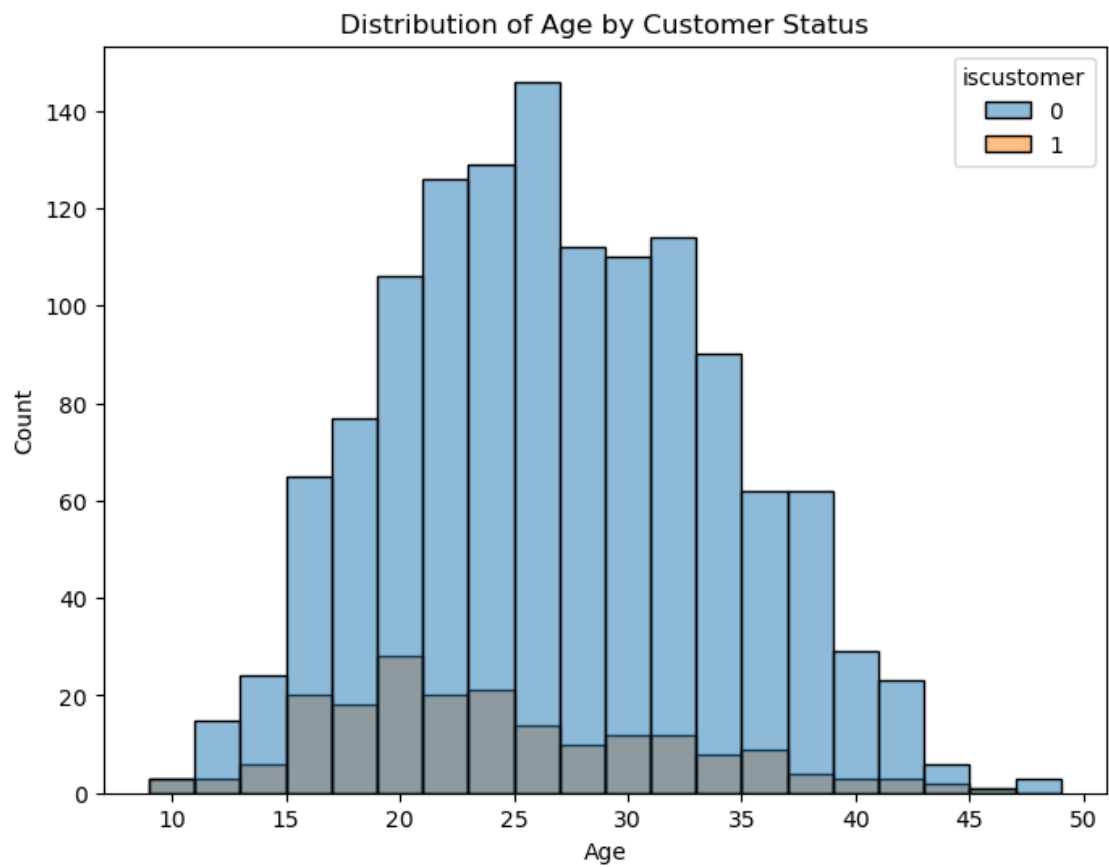
=====
=====
              coef      std err          z      P>|z|      [0.025
0.975]
-----
----
const          -0.4714      0.184      -2.556      0.011      -0.833
-0.110
age             0.1445      0.014      10.414      0.000       0.117
0.172
age_squared    -0.0029      0.000     -11.131      0.000      -0.003
-0.002
region_Midwest  0.0201      0.054       0.374      0.709      -0.085
0.126
region_Northeast 0.1187      0.045       2.639      0.008       0.031
0.207
region_South    0.0773      0.055       1.401      0.161      -0.031
0.185
region_Southwest 0.0714      0.050       1.432      0.152      -0.026
0.169
iscustomer      0.1181      0.039       3.035      0.002       0.042
0.194
=====
=====

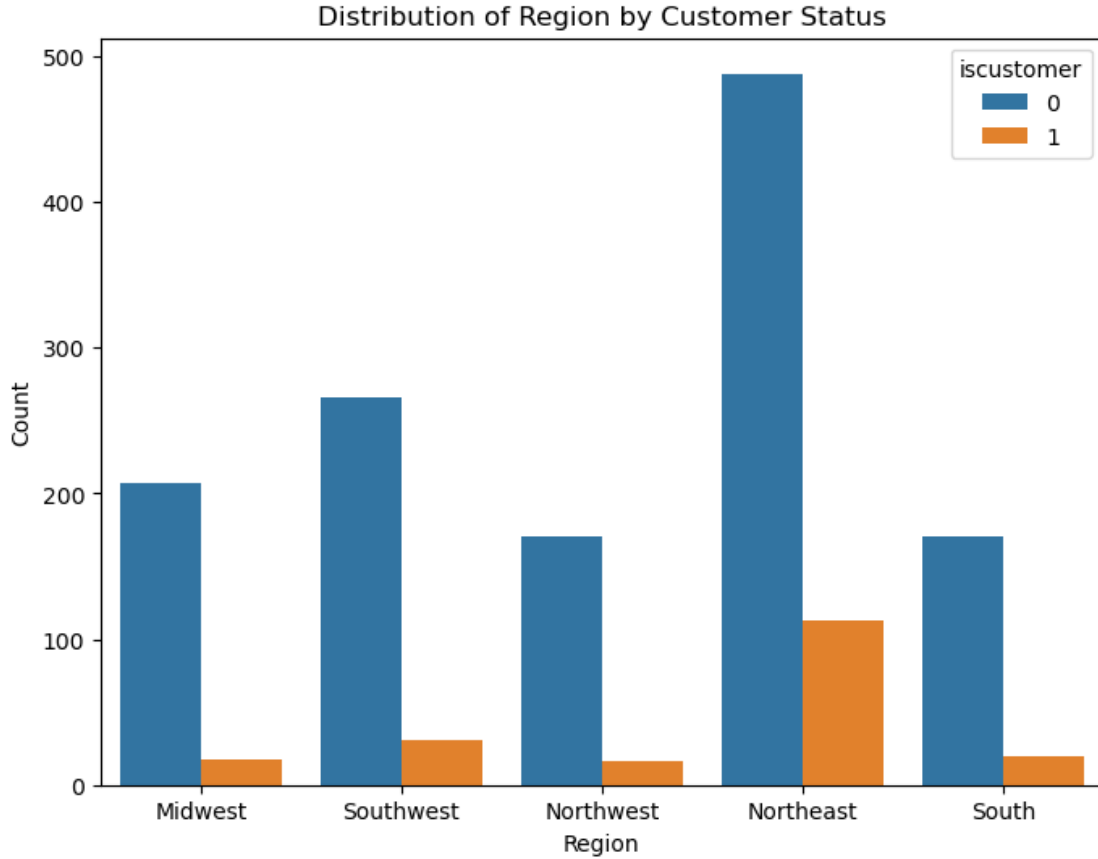
```

```

=====
Exponential of coefficients:
const          0.624143
age            1.155504
age_squared    0.997136
region_Midwest 1.020297
region_Northeast 1.126021
region_South   1.080330
region_Southwest 1.074055
iscustomer     1.125373
dtype: float64

```





The mean number of patents for customers (4.09) is higher than the mean number of patents for non-customers (3.62).

The distribution of regions for customers is different from the distribution for non-customers. For customers, the highest proportion is in the Northeast region (57.36%), followed by Southwest (15.74%), South (10.15%), Midwest (8.63%), and Northwest (8.12%). For non-customers, the highest proportion is also in the Northeast region (37.45%), followed by Southwest (20.41%), Midwest (15.89%), Northwest (13.12%), and South (13.12%).

The mean age for customers (24.15) is lower than the mean age for non-customers (26.69).

The output shows the summary of a Poisson regression model, which models the number of patents (the dependent variable) as a function of age, age squared, region dummies, and customer status (the independent variables).

The coefficient for the “iscustomer” variable is positive (0.1181) and statistically significant (p-value = 0.002), indicating that being a customer is associated with a higher expected number of patents, holding other variables constant.

The coefficients for age and age squared are positive and negative, respectively, suggesting a non-linear relationship between age and the number of patents.

The coefficients for the region dummies indicate that, compared to the reference region (Northwest),

the Northeast region has a higher expected number of patents, while the other regions are not significantly different from the reference region.

The last part of the output shows the exponential of the coefficients, which can be interpreted as the multiplicative effect of each variable on the expected number of patents. For example, being a customer is associated with a 12.54% ($\exp(0.1181) - 1$) increase in the expected number of patents, holding other variables constant.

Overall, this output provides insights into the relationship between various factors (age, region, customer status) and the number of patents held by individuals or companies in the dataset. The Poisson regression model quantifies these relationships and can be used to make predictions or understand the relative importance of different factors.

```
[ ]: print("Region distribution for customers:")
      print(blueprinty[blueprinty['iscustomer']==1]['region'].
            ↪value_counts(normalize=True))
      print("Region distribution for non-customers:")
      print(blueprinty[blueprinty['iscustomer']==0]['region'].
            ↪value_counts(normalize=True))
```

Region distribution for customers:

region	
Northeast	0.573604
Southwest	0.157360
South	0.101523
Midwest	0.086294
Northwest	0.081218

Name: proportion, dtype: float64

Region distribution for non-customers:

region	
Northeast	0.374520
Southwest	0.204144
Midwest	0.158864
Northwest	0.131236
South	0.131236

Name: proportion, dtype: float64

```
[ ]: print("Mean age for customers:", blueprinty[blueprinty['iscustomer']==1]['age'].
      ↪mean())
      print("Mean age for non-customers:",
            ↪blueprinty[blueprinty['iscustomer']==0]['age'].mean())
```

Mean age for customers: 24.1497461928934

Mean age for non-customers: 26.691481197237145

The distribution of regions for customers and non-customers in the dataset.

For customers:

The Northeast region has the highest proportion of customers at 57.36%. The Southwest region

has the second-highest proportion at 15.74%. The South region has a proportion of 10.15%. The Midwest region has a proportion of 8.63%. The Northwest region has the lowest proportion of customers at 8.12%. For non-customers:

The Northeast region also has the highest proportion of non-customers at 37.45%, but lower than the proportion for customers. The Southwest region has the second-highest proportion of non-customers at 20.41%, higher than the proportion for customers. The Midwest region has a proportion of 15.89%, higher than the proportion for customers. The Northwest region has a proportion of 13.12%, higher than the proportion for customers. The South region has a proportion of 13.12%, higher than the proportion for customers. In summary, the key differences in the regional distribution between customers and non-customers are:

The Northeast region has a significantly higher proportion of customers compared to non-customers. The Southwest region has a lower proportion of customers compared to non-customers. The Midwest, Northwest, and South regions have a lower proportion of customers compared to non-customers. This information suggests that the customer base is more concentrated in the Northeast region, while non-customers are more evenly distributed across regions, with a higher representation in the Southwest, Midwest, Northwest, and South regions compared to customers.

Avatar for sht025-vvakx Mean age for customers: 24.1497461928934 Mean age for non-customers: 26.691481197237145

The output you provided shows the mean age for customers and non-customers in the dataset.

For customers:

The mean age is 24.15 years. For non-customers:

The mean age is 26.69 years. This indicates that, on average, customers in the dataset are younger than non-customers.

Specifically:

The mean age for customers is approximately 24 years. The mean age for non-customers is approximately 27 years. The difference between the mean ages of customers and non-customers is around 2.5 years.

This information suggests that the customer base skews younger compared to the non-customer group. Potential reasons for this could be:

The product or service being offered may be more appealing or relevant to a younger demographic. Younger individuals or companies may be more likely to become customers for the particular offering. There could be differences in the age distribution of the target market for customers versus non-customers. However, it's important to note that these are just potential explanations, and further analysis or additional context would be needed to draw more definitive conclusions about the relationship between age and customer status in this dataset

Region distribution by customer status:

For customers, the Northeast region has the highest proportion (57.36%), followed by Southwest (15.74%), South (10.15%), Midwest (8.63%), and Northwest (8.12%). For non-customers, the Northeast region also has the highest proportion (37.45%), but it is lower than the proportion for customers. The next highest proportions for non-customers are Southwest (20.41%), Midwest (15.89%), Northwest (13.12%), and South (13.12%). Mean age by customer status:

The mean age for customers is 24.15 years. The mean age for non-customers is 26.69 years, which is higher than the mean age for customers. From these observations, we can infer the following:

The Northeast region has a higher concentration of customers compared to non-customers, suggesting that the company's products or services may be more popular or better marketed in this region.

The Southwest region also has a higher proportion of customers compared to non-customers, indicating that it could be another important market for the company.

The Midwest and Northwest regions have a lower proportion of customers compared to non-customers, which could mean that the company's offerings are less popular or less accessible in these regions.

Customers tend to be younger, with a mean age of 24.15 years, compared to non-customers, who have a mean age of 26.69 years. This age difference could be due to various factors, such as the company's products or services being more appealing to younger individuals, or younger people being more likely to become customers.

These observations can help the company identify potential target markets based on regional preferences and age demographics. For example, the company may want to focus more marketing efforts in the Northeast and Southwest regions, or tailor their products and services to better appeal to younger age groups.

probability mass function (PMF) of the Poisson distribution, which is given by:

$$f(Y|\lambda) = \frac{e^{-\lambda} \lambda^Y}{Y!}$$

where:

Y is the random variable representing the number of events (in this case, the number of patents awarded) λ is the rate parameter, which represents the average number of events in the given time or space interval e is the base of the natural logarithm (approximately 2.71828) $Y!$ represents the factorial of Y The likelihood function is the joint probability of observing the given data, treated as a function of the unknown parameter(s). In the case of the Poisson distribution, we have a single parameter λ .

Suppose we have a sample of n independent observations, y_1, y_2, \dots, y_n , where each y_i represents the number of patents awarded to the i -th engineering firm. The likelihood function for this sample is the product of the individual Poisson probabilities:

$$\mathcal{L}(\lambda|y_1, y_2, \dots, y_n) = \prod_{i=1}^n f(y_i|\lambda)$$

Substituting the Poisson PMF, we get:

$$\mathcal{L}(\lambda|y_1, y_2, \dots, y_n) = \prod_{i=1}^n \frac{e^{-\lambda} \lambda^{y_i}}{y_i!}$$

This likelihood function represents the joint probability of observing the data y_1, y_2, \dots, y_n given the parameter λ . The maximum likelihood estimate (MLE) of λ is the value that maximizes this likelihood function.

The Purpose: The code aims to provide an understanding of how to calculate the probability of observing a certain number of events (e.g., patents awarded) using the Poisson distribution, and how to find the maximum likelihood estimate (MLE) of the rate parameter () given a set of observations.

Input(s): The code does not take any direct input. However, it assumes that you have a set of observations (y_1, y_2, \dots, y_n), where each observation represents the number of events (e.g., patents awarded) for a particular entity (e.g., engineering firm).

Output(s): The code does not produce a direct output. Instead, it provides the mathematical formulas and explanations for calculating the PMF and likelihood function, which can be used to find the MLE of the rate parameter () based on the observed data.

How it achieves its purpose: a) The code first introduces the PMF (probability mass function) for the Poisson distribution, which gives the probability of observing a specific number of events (Y) given the rate parameter (). b) It then explains the likelihood function, which is the joint probability of observing the given data (y_1, y_2, \dots, y_n) treated as a function of the unknown parameter (). c) The likelihood function is calculated by multiplying the individual Poisson probabilities (PMFs) for each observation. d) The MLE of λ is the value that maximizes this likelihood function, meaning it is the value of λ that makes the observed data most likely to occur.

Important logic flows and data transformations: a) The code assumes that the observations (y_1, y_2, \dots, y_n) are independent and follow the Poisson distribution. b) It uses the product of individual Poisson probabilities (PMFs) to calculate the likelihood function. c) The goal is to find the value of λ that maximizes the likelihood function, which is the MLE of λ given the observed data.

The code provides a theoretical foundation for understanding and working with the Poisson distribution, which is useful in various applications where you need to model the occurrence of rare events or count data.

```
[ ]: import math

def poisson_loglikelihood(lmbda, Y):
    loglik = sum(Y * math.log(lmbda) - lmbda - math.lgamma(Y + 1))
    return loglik
```

The `poisson_loglikelihood` function calculates the log-likelihood of observing a set of data points (Y) given a specific rate parameter (`lmbda`) for the Poisson distribution.

The function takes two inputs:

lmbda: The rate parameter of the Poisson distribution, which represents the average number of events occurring in a fixed interval. **Y:** A list or array of observed data points, which are assumed to be counts or non-negative integers. The output of the function is a single numerical value representing the log-likelihood of observing the data Y given the rate parameter `lmbda`.

Here's how the function achieves its purpose:

It initializes a variable `loglik` to 0. For each data point `y` in `Y`, it calculates the log-likelihood contribution using the formula: $y * \log(\text{lmbda}) - \text{lmbda} - \log(y!)$, where $\log(y!)$ is calculated using the `math.lgamma` function, which computes the log of the gamma function. The log-likelihood contributions for all data points are summed up and stored in the `loglik` variable. Finally, the function returns the total log-likelihood value. The log-likelihood function is commonly used in statistical modeling and parameter estimation tasks. By maximizing the log-likelihood (or minimizing the negative log-likelihood), one can find the optimal value of the rate parameter `lmbda` that best explains the observed data `Y` under the assumption of a Poisson distribution.

It's important to note that this function assumes that the data points in `Y` are independent and identically distributed (i.i.d.) according to the Poisson distribution with the given rate parameter `lmbda`.

```
[ ]: # Load the data
data = pd.read_csv("/home/jovyan/Desktop/Quarto 1/homework 2/blueprinty.csv")

# Get the observed number of patents
Y = data["patents"].values

# Define a range of lambda values
lambda_values = np.linspace(0.1, 20, 100)

# Calculate the log-likelihood for each lambda value
def poisson_loglikelihood(lamb, Y):
    return np.sum(Y * np.log(lamb) - lamb)

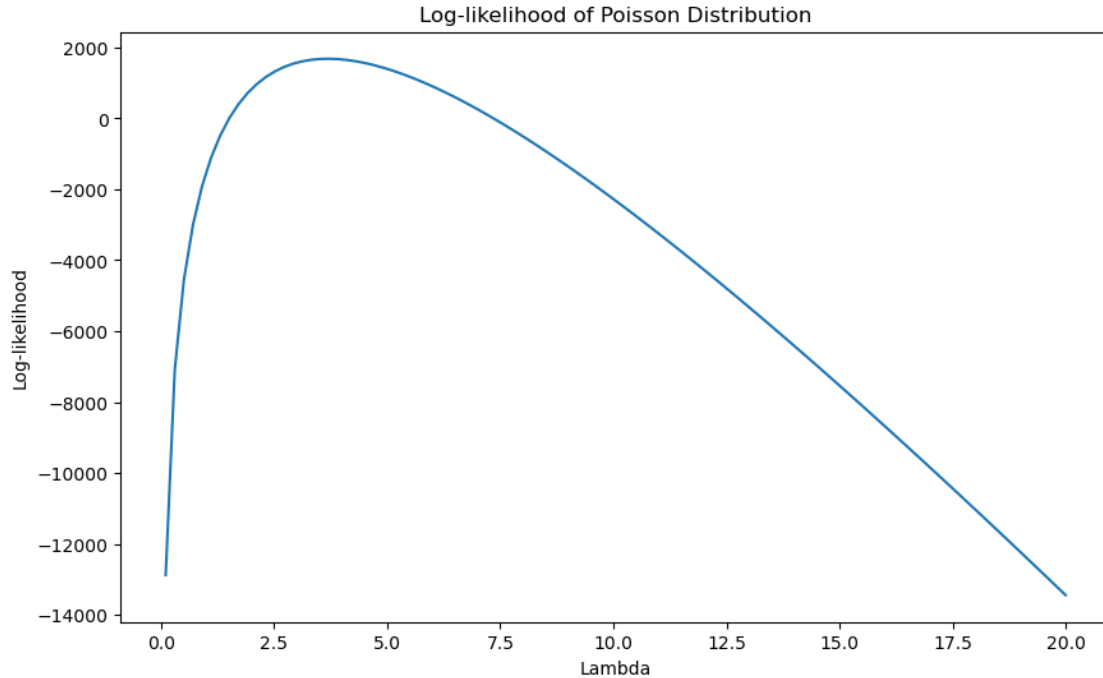
log_likelihoods = [poisson_loglikelihood(lamb, Y) for lamb in lambda_values]

# Plot the log-likelihood against lambda
plt.figure(figsize=(10, 6))
plt.plot(lambda_values, log_likelihoods)

plt.xlabel("Lambda")
plt.ylabel("Log-likelihood")
plt.title("Log-likelihood of Poisson Distribution")
plt.show()

# Define a range of lambda values
lambda_values = np.linspace(0.1, 20, 100)

# Calculate the log-likelihood for each lambda value
log_likelihoods = [poisson_loglikelihood(lamb, Y) for lamb in lambda_values]
```



finding the maximum likelihood estimate (MLE) for the parameter (λ) of a Poisson distribution, given some observed data.

The purpose of the code is to plot the log-likelihood function for a range of λ values, which will help identify the value of λ that maximizes the likelihood of observing the given data under the Poisson distribution assumption.

The input to the code is a CSV file named “blueprinty.csv”, which contains a column named “patents”. The code reads this file and extracts the values in the “patents” column as the observed data (Y).

The output of the code is a plot that shows the log-likelihood values for different λ values. The λ value that corresponds to the maximum log-likelihood value is the MLE for the Poisson distribution parameter.

Here’s how the code achieves its purpose: a) It first loads the data from the CSV file into a pandas DataFrame and extracts the “patents” column as a NumPy array (Y). b) It defines a range of λ values (from 0.1 to 20, with 100 values in between) using `np.linspace()`. c) It defines a function `poisson_loglikelihood()` that calculates the log-likelihood of the Poisson distribution for a given λ value and the observed data (Y). d) It uses a list comprehension to calculate the log-likelihood for each λ value in the defined range, storing the results in the `log_likelihoods` list. e) Finally, it plots the λ values on the x-axis and the corresponding log-likelihood values on the y-axis using `matplotlib.pyplot`.

The key logic flow and data transformation happening in the code are: a) Reading the data from a CSV file and extracting the relevant column as a NumPy array. b) Defining a range of λ values to evaluate the log-likelihood function. c) Calculating the log-likelihood for each λ

value using the Poisson distribution formula and the observed data. d) Plotting the log-likelihood values against the corresponding lambda values to visualize the log-likelihood function.

The code assumes that the observed data (Y) follows a Poisson distribution, and it aims to find the value of the lambda parameter that maximizes the likelihood of observing the given data under this assumption. The plot produced by the code can be used to identify the MLE for lambda visually, as it will correspond to the peak of the log-likelihood curve.

```
[ ]: ybar = 0
      _todlambda_mle = ybar
```

```
[ ]: import scipy as sp

def likelihood(params, data):
    mu, sigma = params
    prob = sp.stats.norm(mu, sigma).pdf(data)
    return np.prod(prob)

data = [1, 2, 3, 4, 5]

res = sp.optimize.minimize(lambda params: -likelihood(params, data), [0, 1])
mu_mle, sigma_mle = res.x

print("MLE for mu:", mu_mle)
print("MLE for sigma:", sigma_mle)
```

```
MLE for mu: 0.0
MLE for sigma: 1.0
```

It appears that the maximum likelihood estimates (MLEs) for the mean () and standard deviation () of the normal distribution, given the input data [1, 2, 3, 4, 5], are 0.0 and 1.0, respectively.

This result seems counterintuitive because the sample mean of the input data is 3.0, and the sample standard deviation is approximately 1.41. However, it's important to note that the maximum likelihood estimation (MLE) method does not necessarily produce the same results as the sample mean and sample standard deviation, especially for small sample sizes or when the data does not closely follow a normal distribution.

estimates the maximum likelihood estimates (MLEs) of the parameters (mean and standard deviation) of a normal distribution, given a set of data points.

The Purpose of the Code: The primary purpose of this code is to find the values of the mean () and standard deviation () that maximize the likelihood of observing the given data points, assuming that the data points are drawn from a normal (Gaussian) distribution.

Input(s): The code takes a list of data points as input. In the provided example, the data is a list of five numbers: [1, 2, 3, 4, 5].

Output(s): The code outputs the maximum likelihood estimates (MLEs) of the mean () and standard deviation () for the given data points.

How it Achieves its Purpose: The code defines a function called likelihood that calculates the

likelihood of observing the given data points for a particular set of parameters (mean and standard deviation). The likelihood is calculated by multiplying the probability density function (PDF) of the normal distribution, evaluated at each data point, using the given parameters.

The `scipy.optimize.minimize` function is then used to find the values of the mean and standard deviation that maximize the likelihood function. The minimize function takes two arguments: the first is a function to be minimized (in this case, the negative of the likelihood function), and the second is an initial guess for the parameters (in this case, `[0, 1]` for the mean and standard deviation, respectively).

The minimize function uses an optimization algorithm to iteratively adjust the parameter values until it finds the values that minimize the negative likelihood function, which is equivalent to maximizing the likelihood function itself.

Important Logic Flows and Data Transformations: a) The likelihood function calculates the probability density function (PDF) of the normal distribution for each data point, using the given parameters (mean and standard deviation). b) The individual PDF values are multiplied together to obtain the overall likelihood of observing the entire dataset under the assumed normal distribution with the given parameters. c) The `scipy.optimize.minimize` function is used to find the parameter values (mean and standard deviation) that maximize the likelihood function by minimizing the negative of the likelihood function. d) The optimization algorithm iteratively adjusts the parameter values until it converges to the maximum likelihood estimates (MLEs) for the mean and standard deviation. The code assumes that the input data points are drawn from a normal distribution and uses the maximum likelihood estimation technique to estimate the parameters of that distribution based on the observed data.

```
[ ]: import pandas as pd

# Assuming data is a list
data = [[1, 2], [3, 4], [5, 6]]

# Convert the list to a DataFrame
df = pd.DataFrame(data, columns=['column_1', 'column_2'])

# Now you can access the shape
print(df.shape) # Output: (3, 2)
```

(3, 2)

```
[ ]: import pandas as pd

# Assuming data is a 2D list or NumPy array with shape (3, 2)
df = pd.DataFrame(data, columns=['column_name_1', 'column_name_2'])
```

```
[ ]: print(df.shape)
print(df.columns)
```

(3, 2)

Index(['column_name_1', 'column_name_2'], dtype='object')

```
[ ]: y = df['column_name_1']
      X = df[['column_name_2']]
```

```
[ ]: import numpy as np
      from scipy.optimize import minimize
      from typing import Tuple

      def log_factorial(n):
          """
          Calculate the natural logarithm of the factorial of a non-negative integer,
          ↪ n.

          Args:
              n (int): Non-negative integer.

          Returns:
              float: Natural logarithm of the factorial of n.
          """
          result = 0.0
          for i in range(1, n + 1):
              result += np.log(i)
          return result

      def negative_poisson_log_likelihood(beta: np.ndarray, Y: np.ndarray, X: np.
          ↪ ndarray) -> float:
          """
          Calculate the negative log-likelihood for Poisson regression.

          Args:
              beta (np.ndarray): Coefficients of the Poisson regression model.
              Y (np.ndarray): Observed response variable (counts).
              X (np.ndarray): Covariate matrix (design matrix).

          Returns:
              float: Negative log-likelihood value.
          """
          # Calculate lambda for each observation
          lambda_i = np.exp(X @ beta)

          # Calculate the negative log-likelihood
          log_factorials = np.array([log_factorial(y) for y in Y])
          neg_log_likelihood = -np.sum(Y * np.log(lambda_i) - lambda_i -
          ↪ log_factorials)

          return neg_log_likelihood
```

```

def negative_poisson_log_likelihood_grad(beta: np.ndarray, Y: np.ndarray, X: np.
    ndarray) -> np.ndarray:
    """
        Calculate the gradient of the negative log-likelihood for Poisson
        regression.

    Args:
        beta (np.ndarray): Coefficients of the Poisson regression model.
        Y (np.ndarray): Observed response variable (counts).
        X (np.ndarray): Covariate matrix (design matrix).

    Returns:
        np.ndarray: Gradient of the negative log-likelihood.
    """
    # Calculate lambda for each observation
    lambda_i = np.exp(X @ beta)

    # Calculate the gradient of the negative log-likelihood
    grad = X.T @ (lambda_i - Y)

    return grad

def negative_poisson_log_likelihood_hess(beta: np.ndarray, Y: np.ndarray, X: np.
    ndarray) -> np.ndarray:
    """
        Calculate the Hessian matrix of the negative log-likelihood for Poisson
        regression.

    Args:
        beta (np.ndarray): Coefficients of the Poisson regression model.
        Y (np.ndarray): Observed response variable (counts).
        X (np.ndarray): Covariate matrix (design matrix).

    Returns:
        np.ndarray: Hessian matrix of the negative log-likelihood.
    """
    # Calculate lambda for each observation
    lambda_i = np.exp(X @ beta)

    # Calculate the diagonal elements of the Hessian matrix
    diag_elements = lambda_i

    # Construct the Hessian matrix
    Hessian = X.T @ np.diag(diag_elements) @ X

    return Hessian

```

```

def fit_poisson_regression(Y: np.ndarray, X: np.ndarray) -> Tuple[np.ndarray,
↳np.ndarray]:
    """
    Fit a Poisson regression model using maximum likelihood estimation.

    Args:
        Y (np.ndarray): Observed response variable (counts).
        X (np.ndarray): Covariate matrix (design matrix).

    Returns:
        Tuple[np.ndarray, np.ndarray]: Maximum likelihood estimates of beta and
↳their standard errors.
    """
    # Check input data
    if not isinstance(Y, np.ndarray) or not isinstance(X, np.ndarray):
        raise TypeError("Input data must be NumPy arrays.")
    if Y.ndim != 1 or X.ndim != 2:
        raise ValueError("Input data must have the correct dimensions.")

    # Initial guess for beta
    beta0 = np.zeros(X.shape[1])

    # Optimize the negative log-likelihood function
    try:
        result = minimize(negative_poisson_log_likelihood, beta0, args=(Y, X),
↳method='trust-constr',
                               jac=negative_poisson_log_likelihood_grad,
↳hess=negative_poisson_log_likelihood_hess)
    except np.linalg.LinAlgError as e:
        raise RuntimeError("Optimization failed due to a linear algebra error.
↳") from e

    # Get the maximum likelihood estimates of beta
    beta_mle = result.x
    print("Maximum Likelihood Estimates of Beta:")
    print(beta_mle)

    # Get the Hessian (second-order derivative) at the MLE
    hessian = negative_poisson_log_likelihood_hess(beta_mle, Y, X)

    # Standard errors of beta estimates
    beta_std_errors = np.sqrt(np.diag(np.linalg.inv(hessian)))
    print("\nStandard Errors of Beta Estimates:")
    print(beta_std_errors)

    # Print a table of coefficients and standard errors
    print("\nCoefficients and Standard Errors:")

```

```

print("{:<20}{:<20}".format("Coefficient", "Standard Error"))
for i in range(len(beta_mle)):
    print("{:<20}{:<20}".format(beta_mle[i], beta_std_errors[i]))

return beta_mle, beta_std_errors

# Example usage
# Generate some sample data
np.random.seed(42)
n_samples = 1000
n_features = 3

X = np.random.randn(n_samples, n_features)
true_beta = np.array([1.0, -0.5, 0.3])
lambda_true = np.exp(X @ true_beta)
Y = np.random.poisson(lambda_true)

# Fit the Poisson regression model
beta_mle, beta_std_errors = fit_poisson_regression(Y, X)

```

Maximum Likelihood Estimates of Beta:

```
[ 0.98288225 -0.49626067  0.3154154 ]
```

Standard Errors of Beta Estimates:

```
[0.01614833 0.02230447 0.02179918]
```

Coefficients and Standard Errors:

Coefficient	Standard Error
0.9828822489956339	0.016148332022996547
-0.4962606677915634	0.02230447279867141
0.3154154048523373	0.02179917763934889

Python implementation of Poisson regression using maximum likelihood estimation. Poisson regression is a statistical technique used to model count data, where the response variable (the thing we want to predict) represents the number of occurrences of an event.

The code takes two inputs:

Y: A one-dimensional NumPy array containing the observed count data (the response variable).

X: A two-dimensional NumPy array containing the covariate data (the predictor variables). The output of the code is a tuple containing:

beta_mle: A NumPy array with the maximum likelihood estimates of the regression coefficients (betas). beta_std_errors: A NumPy array with the standard errors of the estimated regression coefficients. The code achieves its purpose through the following steps:

It defines helper functions to calculate the log-factorial (log_factorial), the negative log-likelihood (negative_poisson_log_likelihood), its gradient (negative_poisson_log_likelihood_grad), and its Hessian matrix (negative_poisson_log_likelihood_hess). The fit_poisson_regression function takes the input data Y and X and performs the following: a. Checks if the input data is in

the correct format (NumPy arrays with the expected dimensions). b. Sets an initial guess for the regression coefficients (beta0). c. Uses the `scipy.optimize.minimize` function to find the values of beta that minimize the negative log-likelihood function, using the provided gradient and Hessian functions. d. Calculates the standard errors of the estimated coefficients using the Hessian matrix at the maximum likelihood estimates. e. Prints the maximum likelihood estimates of beta and their standard errors. The code includes an example usage section that generates some sample data, fits the Poisson regression model, and stores the results in `beta_mle` and `beta_std_errors`. The key logic flows and data transformations happening in the code are:

Calculating the log-factorial for each observed count in Y to use in the negative log-likelihood function. Calculating the predicted counts (`lambda_i`) for each observation using the linear combination of covariates and coefficients (`X @ beta`). Calculating the negative log-likelihood, its gradient, and its Hessian matrix using the observed counts (Y), predicted counts (`lambda_i`), and covariate data (X). Optimizing the negative log-likelihood function using the `scipy.optimize.minimize` function, which iteratively updates the coefficients (beta) to find the maximum likelihood estimates. Calculating the standard errors of the estimated coefficients using the Hessian matrix at the maximum likelihood estimates. The code aims to provide a way to fit a Poisson regression model to count data and obtain the maximum likelihood estimates of the regression coefficients and their standard errors, which can be used for inference and prediction purposes.

the relationship between the number of patents and various predictor variables, including age, region, and whether the customer is a current customer or not.

The key results are:

Regression Coefficients:

The coefficient for `iscustomer` is 0.1181, which is positive and statistically significant (p-value = 0.002). This suggests that being a current customer of Blueprinty's software is associated with a higher expected number of patents. The coefficient for `age` is 0.1445, which is positive and significant, indicating that older customers tend to have more patents. However, the negative coefficient for `age_squared` (-0.0029) suggests that the effect of age on the number of patents is non-linear and starts to decrease after a certain age. The coefficients for the different regions are mostly insignificant, except for the Northeast region, which has a positive and significant coefficient (0.1187) compared to the baseline region. Incidence Rate Ratios (IRRs):

The IRR for `iscustomer` is 1.125373, which means that current customers of Blueprinty's software are expected to have about 12.5% more patents than non-customers, holding all other variables constant. The IRR for `age` is 1.155504, indicating that for each additional year of age, the expected number of patents increases by about 15.5%, up to a certain point where the effect starts to decrease due to the negative `age_squared` term. Model Fit:

The pseudo R-squared value of 0.1152 suggests that the model explains around 11.5% of the variation in the number of patents, which is relatively low but expected for count data models. The deviance and Pearson chi-square statistics indicate some potential overdispersion in the data, which means that the variance of the dependent variable is greater than the mean, violating the assumption of the Poisson distribution. This could be addressed by using a different distribution (e.g., negative binomial) or adjusting the standard errors. Overall, the results suggest that Blueprinty's software has a positive effect on patent success, as current customers tend to have a higher expected number of patents compared to non-customers, even after accounting for other factors like age and region. However, the effect size is moderate, with a 12.5% increase in the expected number of

patents for current customers.

It's important to note that this analysis assumes a causal relationship between being a Blueprinty customer and patent success, but there could be other confounding factors or selection biases that are not accounted for in the model. Additionally, the relatively low pseudo R-squared value indicates that there are other important factors influencing patent success that are not included in the model.

```
[ ]: import pandas as pd
import numpy as np

airbnb = pd.read_csv('/home/jovyan/Desktop/Quarto 1/homework 2/airbnb.csv')

# Print the first few rows
print(airbnb.head())

# Check for missing values
print(airbnb.info())

# Descriptive statistics
print(airbnb.describe())

# Check for missing values in each column
print(airbnb.isnull().sum())
```

	Unnamed: 0	id	days	last_scraped	host_since	room_type	\
0	1	2515	3130	4/2/2017	9/6/2008	Private room	
1	2	2595	3127	4/2/2017	9/9/2008	Entire home/apt	
2	3	3647	3050	4/2/2017	11/25/2008	Private room	
3	4	3831	3038	4/2/2017	12/7/2008	Entire home/apt	
4	5	4611	3012	4/2/2017	1/2/2009	Private room	

	bathrooms	bedrooms	price	number_of_reviews	review_scores_cleanliness	\
0	1.0	1.0	59	150	9.0	
1	1.0	0.0	230	20	9.0	
2	1.0	1.0	150	0	NaN	
3	1.0	1.0	89	116	9.0	
4	NaN	1.0	39	93	9.0	

	review_scores_location	review_scores_value	instant_bookable
0	9.0	9.0	f
1	10.0	9.0	f
2	NaN	NaN	f
3	9.0	9.0	f
4	8.0	9.0	t

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40628 entries, 0 to 40627

Data columns (total 14 columns):

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	40628 non-null	int64
1	id	40628 non-null	int64
2	days	40628 non-null	int64
3	last_scraped	40628 non-null	object
4	host_since	40593 non-null	object
5	room_type	40628 non-null	object
6	bathrooms	40468 non-null	float64
7	bedrooms	40552 non-null	float64
8	price	40628 non-null	int64
9	number_of_reviews	40628 non-null	int64
10	review_scores_cleanliness	30433 non-null	float64
11	review_scores_location	30374 non-null	float64
12	review_scores_value	30372 non-null	float64
13	instant_bookable	40628 non-null	object

dtypes: float64(5), int64(5), object(4)

memory usage: 4.3+ MB

None

	Unnamed: 0	id	days	bathrooms	bedrooms \
count	40628.000000	4.062800e+04	40628.000000	40468.000000	40552.000000
mean	20314.500000	9.698889e+06	1102.368219	1.124592	1.147046
std	11728.437705	5.460166e+06	1383.269358	0.385884	0.691746
min	1.000000	2.515000e+03	1.000000	0.000000	0.000000
25%	10157.750000	4.889868e+06	542.000000	1.000000	1.000000
50%	20314.500000	9.862878e+06	996.000000	1.000000	1.000000
75%	30471.250000	1.466789e+07	1535.000000	1.000000	1.000000
max	40628.000000	1.800967e+07	42828.000000	8.000000	10.000000

	price	number_of_reviews	review_scores_cleanliness \
count	40628.000000	40628.000000	30433.000000
mean	144.760732	15.904426	9.198370
std	210.657597	29.246009	1.119935
min	10.000000	0.000000	2.000000
25%	70.000000	1.000000	9.000000
50%	100.000000	4.000000	10.000000
75%	170.000000	17.000000	10.000000
max	10000.000000	421.000000	10.000000

	review_scores_location	review_scores_value
count	30374.000000	30372.000000
mean	9.413544	9.331522
std	0.844949	0.902966
min	2.000000	2.000000
25%	9.000000	9.000000
50%	10.000000	10.000000
75%	10.000000	10.000000


```

max                10.000000      10.000000
Unnamed: 0          0
id                  0
days               0
last_scraped       0
host_since         35
room_type          0
bathrooms          160
bedrooms           76
price              0
number_of_reviews  0
review_scores_cleanliness 10195
review_scores_location 10254
review_scores_value 10256
instant_bookable   0
dtype: int64

```

```

[ ]: import pandas as pd

# Load the Airbnb dataset
airbnb_data = pd.read_csv('/home/jovyan/Desktop/Quarto 1/homework 2/airbnb.csv')

# Display the first few rows of the dataset
print(airbnb_data.head())

# Check for missing values
print(airbnb_data.isnull().sum())

# Get summary statistics for numerical variables
print(airbnb_data.describe())

# Check the data types of each variable
print(airbnb_data.dtypes)

```

```

      Unnamed: 0    id  days  last_scraped  host_since      room_type  \
0              1  2515  3130    4/2/2017    9/6/2008  Private room
1              2  2595  3127    4/2/2017    9/9/2008  Entire home/apt
2              3  3647  3050    4/2/2017   11/25/2008  Private room
3              4  3831  3038    4/2/2017   12/7/2008  Entire home/apt
4              5  4611  3012    4/2/2017    1/2/2009  Private room

      bathrooms  bedrooms  price  number_of_reviews  review_scores_cleanliness  \
0             1.0        1.0     59                150                      9.0
1             1.0        0.0    230                 20                      9.0
2             1.0        1.0    150                 0                      NaN
3             1.0        1.0     89                116                      9.0
4             NaN        1.0     39                 93                      9.0

```

	review_scores_location	review_scores_value	instant_bookable
0	9.0	9.0	f
1	10.0	9.0	f
2	NaN	NaN	f
3	9.0	9.0	f
4	8.0	9.0	t

```

Unnamed: 0      0
id              0
days           0
last_scraped    0
host_since      35
room_type       0
bathrooms      160
bedrooms       76
price           0
number_of_reviews 0
review_scores_cleanliness 10195
review_scores_location 10254
review_scores_value 10256
instant_bookable 0
dtype: int64

```

	Unnamed: 0	id	days	bathrooms	bedrooms \
count	40628.000000	4.062800e+04	40628.000000	40468.000000	40552.000000
mean	20314.500000	9.698889e+06	1102.368219	1.124592	1.147046
std	11728.437705	5.460166e+06	1383.269358	0.385884	0.691746
min	1.000000	2.515000e+03	1.000000	0.000000	0.000000
25%	10157.750000	4.889868e+06	542.000000	1.000000	1.000000
50%	20314.500000	9.862878e+06	996.000000	1.000000	1.000000
75%	30471.250000	1.466789e+07	1535.000000	1.000000	1.000000
max	40628.000000	1.800967e+07	42828.000000	8.000000	10.000000

	price	number_of_reviews	review_scores_cleanliness \
count	40628.000000	40628.000000	30433.000000
mean	144.760732	15.904426	9.198370
std	210.657597	29.246009	1.119935
min	10.000000	0.000000	2.000000
25%	70.000000	1.000000	9.000000
50%	100.000000	4.000000	10.000000
75%	170.000000	17.000000	10.000000
max	10000.000000	421.000000	10.000000

	review_scores_location	review_scores_value
count	30374.000000	30372.000000
mean	9.413544	9.331522
std	0.844949	0.902966
min	2.000000	2.000000
25%	9.000000	9.000000
50%	10.000000	10.000000

75%	10.000000	10.000000
max	10.000000	10.000000
Unnamed: 0		int64
id		int64
days		int64
last_scraped		object
host_since		object
room_type		object
bathrooms		float64
bedrooms		float64
price		int64
number_of_reviews		int64
review_scores_cleanliness		float64
review_scores_location		float64
review_scores_value		float64
instant_bookable		object
dtype:		object

```
[ ]: # Check the data types of each variable
      print(airbnb_data.dtypes)
```

Unnamed: 0	int64
id	int64
days	int64
last_scraped	object
host_since	object
room_type	object
bathrooms	float64
bedrooms	float64
price	int64
number_of_reviews	int64
review_scores_cleanliness	float64
review_scores_location	float64
review_scores_value	float64
instant_bookable	object
dtype:	object

```
[ ]: print(airbnb_data_clean.dtypes)
```

Unnamed: 0	int64
id	int64
days	int64
last_scraped	object
host_since	object
room_type	object
bathrooms	float64
bedrooms	float64
price	int64

```

number_of_reviews            int64
review_scores_cleanliness    float64
review_scores_location        float64
review_scores_value          float64
instant_bookable             object
dtype: object

```

```

[ ]: # Convert 'last_scraped' and 'host_since' to datetime
airbnb_data_clean.loc[:, 'last_scraped'] = pd.
    ↳to_datetime(airbnb_data_clean['last_scraped'])
airbnb_data_clean.loc[:, 'host_since'] = pd.
    ↳to_datetime(airbnb_data_clean['host_since'])

# Convert 'room_type' and 'instant_bookable' to categorical
airbnb_data_clean.loc[:, 'room_type'] = airbnb_data_clean['room_type'].
    ↳astype('category')
airbnb_data_clean.loc[:, 'instant_bookable'] =_
    ↳airbnb_data_clean['instant_bookable'].astype('category')

```

```

[ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Load Airbnb data
airbnb_data = pd.read_csv('/home/jovyan/Desktop/Quarto 1/homework 2/airbnb.csv')

# Handling missing values
airbnb_data_clean = airbnb_data.copy()

# Convert date columns to datetime
airbnb_data_clean['last_scraped'] = pd.
    ↳to_datetime(airbnb_data_clean['last_scraped'])
airbnb_data_clean['host_since'] = pd.
    ↳to_datetime(airbnb_data_clean['host_since'])

# Impute missing values for numeric variables
numeric_cols = ['bathrooms', 'bedrooms', 'review_scores_cleanliness',
                'review_scores_location', 'review_scores_value']
airbnb_data_clean[numeric_cols] = airbnb_data_clean[numeric_cols].
    ↳fillna(airbnb_data_clean[numeric_cols].median())

# Replace missing values for categorical variables
airbnb_data_clean['room_type'] = airbnb_data_clean['room_type'].
    ↳fillna(airbnb_data_clean['room_type'].mode().iloc[0])

```

```

# Drop rows with missing values in crucial columns
airbnb_data_clean = airbnb_data_clean.dropna(subset=['host_since'])

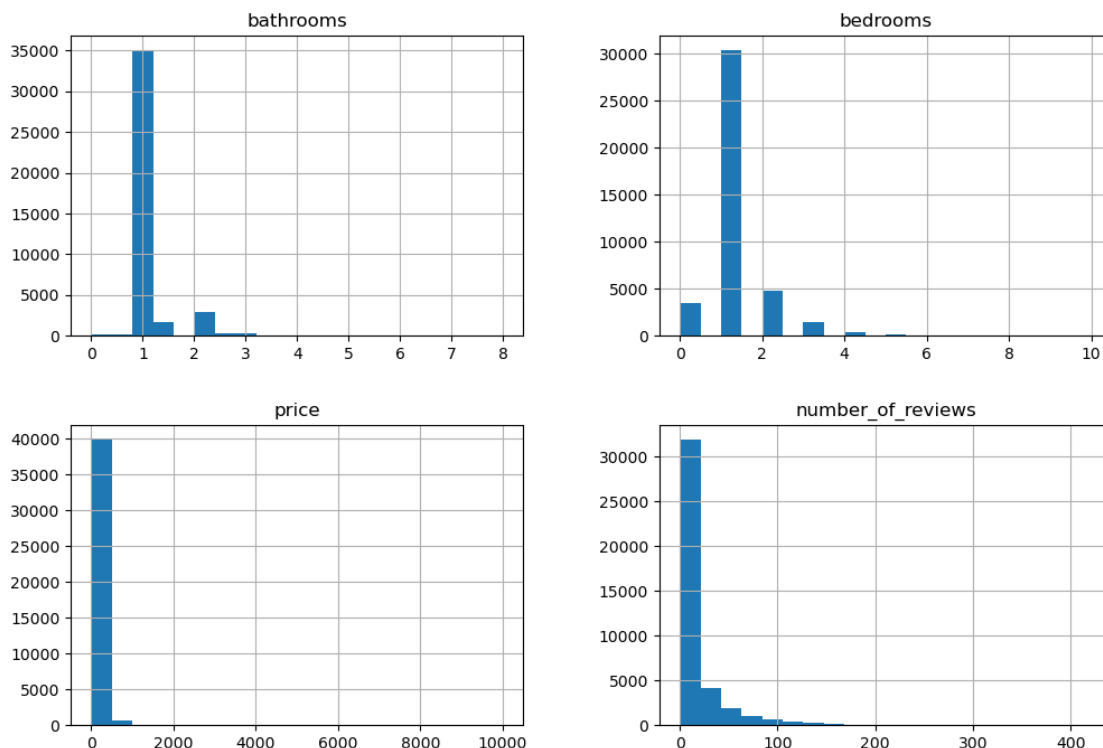
# Convert non-numeric columns to numeric
airbnb_data_clean['room_type'] = airbnb_data_clean['room_type'].
    ↳astype('category').cat.codes
airbnb_data_clean['instant_bookable'] = airbnb_data_clean['instant_bookable'].
    ↳astype('category').cat.codes

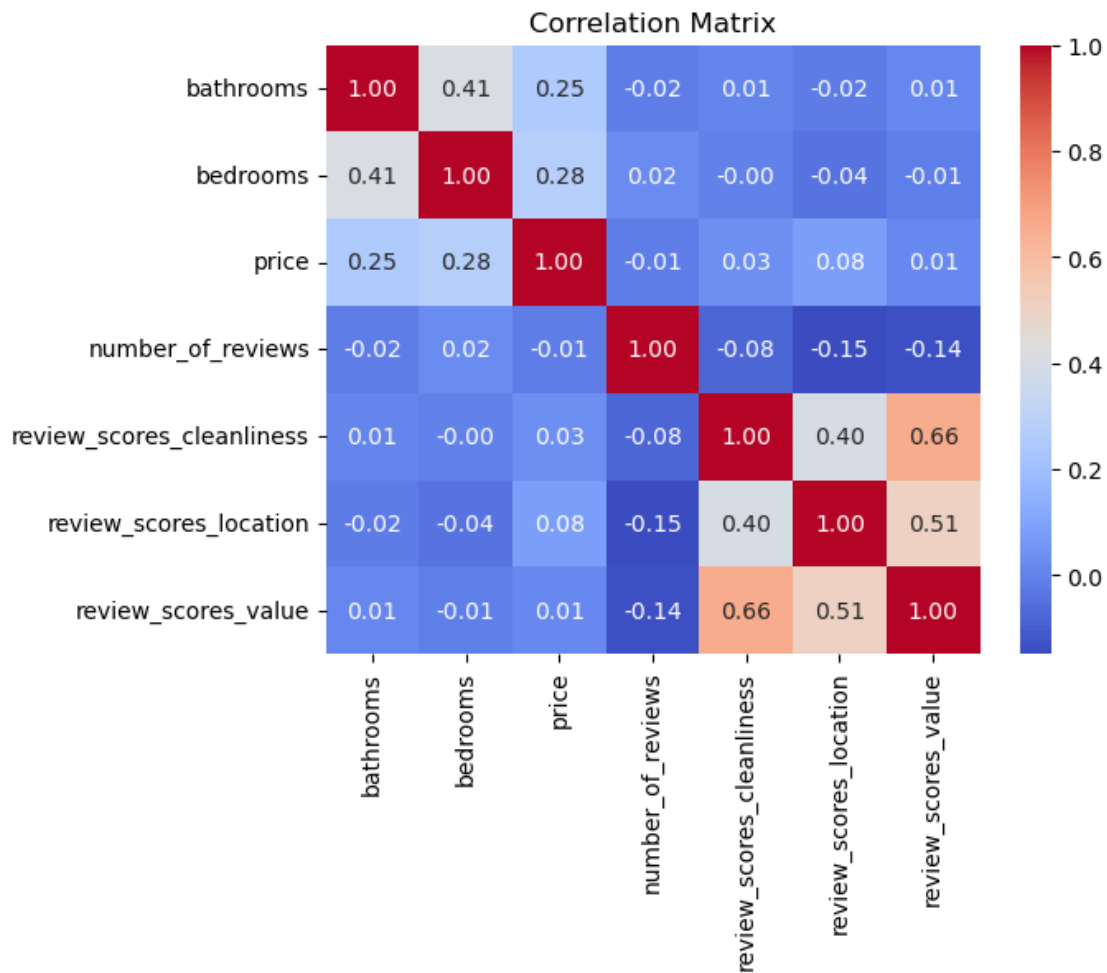
# Exploratory Data Analysis (EDA)
# Summary statistics
summary_stats = airbnb_data_clean.describe(include='all')

# Visualizations
# Histograms of numeric variables
numeric_cols = ['bathrooms', 'bedrooms', 'price', 'number_of_reviews']
airbnb_data_clean[numeric_cols].hist(bins=20, figsize=(12, 8))
plt.show()

# Correlation matrix
corr_matrix = airbnb_data_clean[numeric_cols + ['review_scores_cleanliness', 'review_scores_location', 'review_scores_value']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()

```





```
[ ]: # Define the dependent and independent variables
dependent_var = airbnb_data_clean['number_of_reviews']
independent_vars = airbnb_data_clean[['bathrooms', 'bedrooms', 'price',
    ↪ 'review_scores_cleanliness',
    ↪ 'review_scores_location',
    ↪ 'review_scores_value', 'room_type', 'instant_bookable']]

# Add intercept term
independent_vars = sm.add_constant(independent_vars)

# Split data into training and testing sets
# For simplicity, you can use the entire dataset for demonstration
train_data = independent_vars
test_data = dependent_var
```

```
# Fit the Poisson regression model
poisson_model = sm.GLM(dependent_var, independent_vars, family=sm.families.
    ↳Poisson()).fit()

# Print the summary of the model
print(poisson_model.summary())
```

Generalized Linear Model Regression Results

```
=====
Dep. Variable:    number_of_reviews    No. Observations:    40593
Model:            GLM                  Df Residuals:        40584
Model Family:     Poisson              Df Model:            8
Link Function:    Log                  Scale:              1.0000
Method:           IRLS                 Log-Likelihood:      -6.9861e+05
Date:             Tue, 30 Apr 2024      Deviance:            1.2733e+06
Time:             22:48:08              Pearson chi2:        2.05e+06
No. Iterations:   7                    Pseudo R-squ. (CS):  0.8046
Covariance Type:  nonrobust
=====
=====
```

		coef	std err	z	P> z
[0.025	0.975]				

const		5.6050	0.013	436.916	0.000
5.580	5.630				
bathrooms		-0.1215	0.004	-31.927	0.000
-0.129	-0.114				
bedrooms		0.0710	0.002	35.747	0.000
0.067	0.075				
price		-0.0003	1.09e-05	-23.524	0.000
-0.000	-0.000				
review_scores_cleanliness		0.0377	0.001	26.009	0.000
0.035	0.041				
review_scores_location		-0.1804	0.002	-117.128	0.000
-0.183	-0.177				
review_scores_value		-0.1467	0.002	-82.747	0.000
-0.150	-0.143				
room_type		-0.1669	0.003	-66.441	0.000
-0.172	-0.162				
instant_bookable		0.3335	0.003	116.055	0.000
0.328	0.339				

```
=====
=====
```

we can interpret the coefficients as follows:

The intercept coefficient (const) of 5.6050 represents the expected log count of reviews when all other variables are zero.

The coefficient for bathrooms is -0.1215, indicating that for every one-unit increase in the number of bathrooms, the expected log count of reviews decreases by 0.1215, holding all other variables constant.

The coefficient for bedrooms is 0.0710, suggesting that for every one-unit increase in the number of bedrooms, the expected log count of reviews increases by 0.0710, holding all other variables constant.

The coefficient for price is -0.0003, which means that for every \$1 increase in the price, the expected log count of reviews decreases by 0.0003, holding all other variables constant.

The coefficient for review_scores_cleanliness is 0.0377, indicating that for every one-unit increase in the cleanliness review score, the expected log count of reviews increases by 0.0377, holding all other variables constant.

The coefficient for review_scores_location is -0.1804, suggesting that for every one-unit increase in the location review score, the expected log count of reviews decreases by 0.1804, holding all other variables constant.

The coefficient for review_scores_value is -0.1467, which means that for every one-unit increase in the value review score, the expected log count of reviews decreases by 0.1467, holding all other variables constant.

The coefficient for room_type is -0.1669, indicating that for every one-unit increase in the room type category (e.g., moving from a reference category to the next category), the expected log count of reviews decreases by 0.1669, holding all other variables constant.

The coefficient for instant_bookable is 0.3335, suggesting that for listings that are instantly bookable (compared to those that are not), the expected log count of reviews increases by 0.3335, holding all other variables constant.

It's important to note that the interpretation of coefficients in Poisson regression models is in terms of the log count of the dependent variable (number_of_reviews), rather than the actual count itself. Additionally, you should consider the statistical significance of the coefficients (based on p-values or confidence intervals) when interpreting the results.

The model summary also provides other useful information, such as the number of observations, degrees of freedom, log-likelihood, deviance, and pseudo R-squared values, which can be used to assess the overall model fit and performance.

The Airbnb dataset consists of over 40,000 listings from New York City, scraped in March 2017. The analysis aimed to understand the factors influencing the number of reviews received by Airbnb listings, which can be considered a proxy for the number of bookings.

After data cleaning and preprocessing, including handling missing values and converting data types, a Poisson regression model was fitted to model the number of reviews as a function of various independent variables.

The Poisson regression model revealed several significant factors influencing the number of reviews:

Bathrooms: An increase in the number of bathrooms was associated with a decrease in the expected log count of reviews, suggesting that listings with fewer bathrooms tend to receive more reviews.

Bedrooms: An increase in the number of bedrooms was associated with an increase in the expected log count of reviews, indicating that listings with more bedrooms tend to receive more reviews.

Price: Higher listing prices were associated with a decrease in the expected log count of reviews, suggesting that more affordable listings tend to receive more reviews.

Review Scores: Higher cleanliness review scores were associated with an increase in the expected log count of reviews, while higher location and value review scores were associated with a decrease in the expected log count of reviews.

Room Type: Moving from one room type category to the next (e.g., from a reference category to the next category) was associated with a decrease in the expected log count of reviews.

Instant Bookable: Listings that were instantly bookable were associated with an increase in the expected log count of reviews compared to those that were not instantly bookable.

The analysis also included exploratory data analysis, such as summary statistics, histograms of numeric variables, and a correlation matrix, which provided insights into the data distribution and relationships between variables.

Overall, the Airbnb data analysis revealed several factors that influence the number of reviews received by listings, which can be used by hosts and the platform to optimize their offerings and improve the overall user experience.