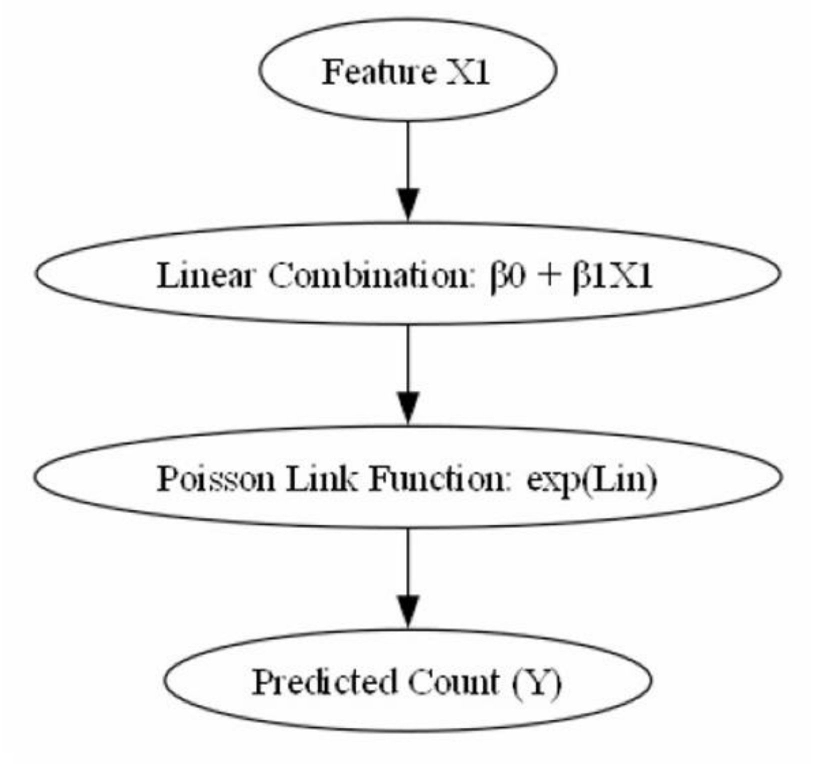
Milestone 3: Implementation

Big Data Analysis of COVID-19 Vaccinations and COVID-19 Related Deaths

System Entities

EDA (script): The purpose of this entity is to understand the structure of the data, as well as having a data cleaning section. It contains a variety of functions, including initial examination of the data, including summary statistics, a variety of visualizations showing data structure, handling for any missing or duplicated data, and different techniques of outlier handling, with a save function for the cleaned data sets.

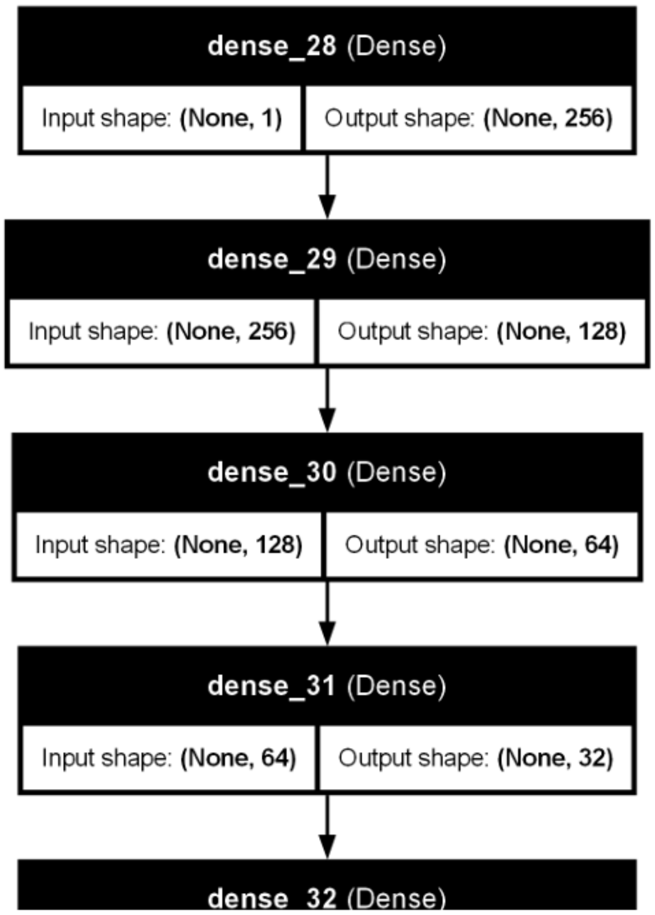
Poisson Regression (model): The purpose of this entity is to create a Poisson Regression model, with predictive analytics, predictive against observed graphical representation of the data, and performance metrics (psuedo R-squared, mean squared error, and cross validation). The following diagram can also be found in the PDF files with the source code.


Feature X1
Linear Combination: β0 + β1X1
Poisson Link Function: exp(Lin)
Predicted Count (Y)

Random Forest (model): The purpose of this entity is to create a Random Forest model, with predictive analytics, predictive against observed graphical representation of the data, and performance metrics (R-squared, mean squared error, and cross validation). The following diagram can also be found in the PDF files with the source code.

Decision Tree from the Random Forest

DNN (model): The purpose of this entity is to create a Deep Neural Network model, with predictive analytics, predictive against observed graphical representation of the data, and performance metrics (R-squared and mean squared error). The following diagram can also be found in the PDF files with the source code.



Feature Engineering (script): The purpose of this entity is to introduce feature engineering based on any data that the EDA showed seasonally or trend data, and the above models did not perform well. It also contains the same predictive against observed graphical representation of the data, and performance metrics (R-squared, mean squared error, and cross validation) as the above models.

Functional Requirements

EDA: (Testing results are available in the source code.)

Loading data	pd.read_csv(*file path*)

Checking for missing values/Removing missing values	data.isnull().sum()
	data.isnull().dropna()

Checking for duplicated values/Removing duplicated values	data.duplicated().sum()
	data.duplicated().drop_duplicates()

Checking data types	print(data.dtypes)
Converting date to date-time	pd.to_datetime(data['Day'])

Graphing data	plt.boxplot()
	fig_box1 = go.Figure()
	fig_hist1 = go.Figure()
	fig_line1 = go.Figure()
	data_rolling7.plot()
	data_grouped['COVID-19 doses (daily)'].diff().plot()
	plt.scatter()
	px.choropleth()
	plt.hist()

Outlier detection	Standard Deviations: *mean_vaccine* = statistics.mean(data['COVID-19 doses (daily)']) *vaccine_mean_outlier_high* = *mean_vaccine* + (*std_dev_vaccine* * 3) *vaccine_mean_outlier_low* = *mean_vaccine* - (*std_dev_vaccine* * 3) vaccine_hmean_outlier_count = np.sum(data['COVID-19 doses (daily)'] > vaccine _mean_outlier_high) *vaccine_lmean_outlier_count* = np.sum(data['COVID-19 doses (daily)'] < *vaccine_mean_outlier_low)* *vaccine_mean_outlier_count* = *vaccine_hmean_outlier_count* + *vaccine_lmean_outier_count* IQRs:

	statistics.median(data['COVID-19 doses (daily)']) *iqr_vaccine* = stats.iqr(data['COVID-19 doses (daily)']) *q1_vaccine* = np.quantile(data['COVID-19 doses (daily)'], 0.25) *q3_vaccine* = np.quantile(data['COVID-19 doses (daily)'], 0.75) *vaccine_median_outlier_high = q3_vaccine* + (*iqr_vaccine* * 1.5) *vaccine_median_outlier_low = q1_vaccine* - (*iqr_vaccine* * 1.5) *vaccine_hmedian_outlier_count* = np.sum(data['COVID-19 doses (daily)'] > *vaccine_median_outlier_high*) *vaccine_lmedian_outlier_coun*t = np.sum(data['COVID-19 doses (daily)'] < *vaccine_median_outlier_low*)
Outlier Handling	Standard Deviations: *data_nomean_out* = {} *data_nomean_out* = pd.DataFrame(*data_nomean_out)* *data_nomean_out['COVID-19 doses (daily, no outliers)']* = data['COVID-19 dose s (daily)'].clip(lower=*vaccine_mean_outlier_low*, upper=*vaccine_mean_outlier_high*) IQRs: *data_nomedian_out* = {}

	data_nomedian_out = pd.DataFrame(*data_nomedian_out*) *data_nomedian_out['COVID-19 doses (daily, no outliers)']* = data['COVID-19 doses (daily)’].clip(lower=v*accine_median_outlier_low*, upper=*vaccine_median_outlier_high*) Log Transform: *data_log* ={} *data_log* = pd.DataFrame(*data_log*) *data_log['COVID-19 doses (daily)']* = np.log1p(data['COVID-19 doses (daily)']))
Summary statistics of data	data.describe()
Saving graphs	plt.savefig()
Saving cleaned data	*data_log*.to_csv('*data_log*.csv', index=False)

Poisson Regression: (Model diagram can be found in the system entities, as well as in the source code. Testing results are also available in the source code.)

Loading data	pd.read_csv(*file path*)
Splitting data into training and testing groups	*X_train, X_test, y_train, y_test* = train_test_split(*X, y*, test_size=0.2, random_state=42)
Initializes model	linear_model.PoissonRegressor()
Fits model	*clf*.fit(*X_train*.values.reshape(-1, 1),*y_train*)
Generates model predictions	*clf*.predict(*X_test*.values.reshape(-1, 1))
Creates predicted against observed graph	plt.scatter(*y_test, predictions*, label='Predictions')
Finds pseudo R-squared	*clf*.score(*X_test*.values.reshape(-1, 1), *y_test*)
Finds mean squared error	mean_squared_error(*y_test, predictions*)
Cross-validates findings	cross_validate(*clf, X_train*.values.reshape(-1, 1), *y_train*, cv=5, scoring=['neg_mean_squared_error', 'r2'], return_train_score=True)

Random Forest: (Model diagram can be found in the system entities, as well as in the source code. Testing results are also available in the source code.)

Loading data	pd.read_csv(*file path*)
Splitting data into training and testing groups	*X_train, X_test, y_train, y_test* = train_test_split(*X, y*, test_size=0.2, random_state=42)
Initializes model	RandomForestRegressor(n_estimators=10, random_state=0, oob_score=False)
Fits model	*regressor*.fit(*X_train*.values.reshape(-1, 1),*y_train*)
Generates model predictions	*regressor*.predict(*X_test*.values.reshape(-1, 1))
Creates predicted against observed graph	plt.scatter(*y_test, predictions*, label='Predictions')
Finds mean squared error	mean_squared_error(*y_test, predictions_RF*)
Finds R-squared	mean_squared_error(*y_test, predictions_RF*)
Cross validating findings	cross_validate(*regressor, X_train*.values.reshape(-1, 1), *y_train*, cv=5, scoring=['neg_mean_squared_error', 'r2'], return_train_score=True)

DNN: (Model diagram can be found in the system entities, as well as in the source code. Testing results are also available in the source code.)

Loading data	pd.read_csv(*file path*)
Splits data into training and testing groups	*X_train, X_test, y_train, y_test* = train_test_split(*X, y*, test_size=0.2, random_state=42)
Initializes model	Sequential()
Adds input layer	*dnn_model*.add(KerasInput(shape=(*X_train*.shape[1],)))
Adds layer to model	*dnn_model*.add(Dense(*256,* activation='*relu*'))
Compiles model	*dnn_model*.compile(loss='mean_squared_error', optimizer='*adam*')
Fits model	*dnn_model*.fit(*X_train, y_train,* epochs=*100*, batch_size=32, validation_data=(*X_test, y_test*))
Generates model predictions	*dnn_model*.predict(*X_test*)
Graphs observed against predicted values	plt.scatter(*y_test*, *predictions_dnn_model*, color= '*red*')

Finds mean squared error	mean_squared_error(*y_test, predictions_dnn_model*)
Finds R-squared value	r2_score(*y_test, predictions_dnn_model*)

Feature Engineering: (No diagrams or testing was created for this section, as it followed structures of all past developed models.)

Loading data	pd.read_csv(*file path*)
Converts date object to datetime object	pd.to_datetime(*data*['Day'])
Log transforms data	np.log1p(*data*['COVID-19 doses (daily)'])
Sorts data by entity and time	*data*.sort_values(by=['Entity', 'Day']) *data*.groupby('Entity').cumcount()
Drops missing values	*data*.isnull().dropna()
Drops duplicated values	*data*.duplicated().drop_duplicates()
Checks data types	print(*data*.dtypes)

Adds columns to data frame for seasonality	np.sin(2 * np.pi * *data*['day_of_year'] / 365)
	np.cos(2 * np.pi * *data*['day_of_year'] / 365)

Splitting data into training and testing groups - Poisson Regression - Random Forest/DNN	*split_date* = *data*['Day'].quantile(0.8) *train_data* = *data*[data['Day'] <= *split_date*] *test_data* = *data*[data['Day'] > *split_date*] *X_train, X_test* = *X*.iloc[:-int(len(*X*)*0.2)], *X*.iloc[-int(len(*X*)*0.2):] *y_train, y_test* = *y*.iloc[:-int(len(*y*)*0.2)], *y*.iloc[-int(len(*y*)*0.2):]
Poisson Regression: - Initialization/Fit - predictions	smf.glm(formula='C*OVID_deaths_daily ~ COVID_doses_daily + time_index + sin_year + cos_year*', data=t*rain_data*, family=sm.families.Poisson()).fit() *poisson*.predict(*test_data*)
Random Forest: - Initialization - Fit - Predictions	RandomForestRegressor(n_estimators=100, random_state=42) *rf*.fit(*X_train, y_train)* *rf*.predict(*X_test)*
DNN: - Initialization - Adds input layer - Adds layer - Compiles - Fits model - Predictions	Sequential() *dnn*.add(KerasInput(shape=(*X_train*.shape[1],))) *dnn*.add(Dense(*256*, activation='*relu*')) *dnn*.compile(loss='mean_squared_error', optimizer='*adam*')

	dnn.fit(*X_train, y_train*, epochs=*100*, batch_size=32, validation_data=(*X_test, y_test*)) *dnn*.predict(*X_test*)
Graphs observed against predicted values	plt.scatter(*test_data*['COVID_deaths_daily'], *predictions_poisson*, color='*blue*')
Poisson Performance Metrics - Pseudo R-squared - Mean squared error - Cross validation	print (*poisson*.summary()) mean_squared_error(*test_data*['COVID_deaths_daily'], *predictions_poisson*) KFold(n_splits=5, shuffle=True, random_state=42) pseudo_r2_scores = [] (followed by model structure)

Random Forest Performance Metrics
- R-squared
- Mean squared error
- Cross validation
	r2_score(*y_test, predictions_rf*)
	
	mean_squared_error(*y_test, predictions_rf*)
	
	cross_validate(*rf, X_train, y_train*, cv=5, scoring=['neg_mean_squared_error', 'r2'], return_train_score=True)

DNN Performance Metrics
- R-squared
- Mean squared error
	r2_score(y*_test, predictions_dnn*)
	
	mean_squared_error(*y_test, predictions_dnn*)

Source Code Listing

This is uploaded as PDF files with comments and markdowns explaining each block of code.

Code Review

The code was periodically reviewed by fellow classmates and the professor overseeing the capstone project, and their revisions were implemented in the code and its markdowns.

Implementation Plan

Overall, this project is attempting to learn pandemic related big data and make accurate predictions based on it. Data cleaning and exploration, modeling through Poisson regression, Random Forest, and DNN, analysis of performance metrics, and further exploration of feature engineering techniques were implemented. The purpose of this is to identify potential successful techniques at analyzing worldwide pandemic related data together, instead of at a more localized analysis level. This could be utilized as a basis for other future research on similar topics.

For this project, Microsoft VS Code was used, with Python as the language and Jupyter Notebook as additional software. The data utilized was “daily-covid-19-vaccine-doses-administered" and “daily-new-confirmed-covid-19-deaths-per-million-people were utilized” merged together from Our World in Data (Appel et al., 2025).

Reference:

Appel, C., Beltekian, D., Dattani, S., Gavrilov, D., Giattino, C., Hasell, J., Macdonald, B., Mathieu, E., Ortiz-Ospina, E., Ritchie, H., Rodes-Guirao, L., & Roser, M. (2025). *COVID-19 pandemic* [Data set]. Our World in Data. https://ourworldindata.org/coronavirus

The following libraries were installed:

- Loading/saving data
  - import pandas as pd
  - from urllib.request import urlopen
- Graphing data
  - import matplotlib.pyplot as plt
  - import seaborn as sns
  - import math import graphviz
  - from sklearn.tree import plot_tree
  - from graphviz import Digraph
  - import matplotlib.image as mpimg
  - import plotly.express as px
  - import plotly.graph_objects as go
  - from plotly.subplots import make_subplots
  - from tensorflow.keras.utils import plot_model
- Modeling
  - import statsmodels.api as sm
  - import statsmodels.stats.api as sms
  - import statsmodels.formula.api as smf
  - import sklearn
  - from sklearn import linear_model
  - from sklearn.model_selection import train_test_split

    - from sklearn.impute import KNNImputer
    - from sklearn.preprocessing import StandardScaler
    - from sklearn.ensemble import RandomForestRegressor
    - import tensorflow as tf
    - from tensorflow.keras.models import Sequential
    - from keras.models import Sequential
    - from keras.layers import Dense
    - from tensorflow.keras.layers import Dense
    - from tensorflow.keras.layers import Input as KerasInput
    - from dash import Dash, dcc, html, Input, Output
    - from sklearn.pipeline import Pipeline
- Performance Metrics
    - from sklearn.metrics import f1_score
    - from sklearn.model_selection import cross_val_score, cross_validate
    - from sklearn.metrics import mean_squared_error, r2_score
    - from sklearn.model_selection import KFold
- Other:
    - import scipy.stats as stats
    - import statistics
    - import numpy as np
    - import warnings
    - import os
    - import time

Strategy for integration included initial development on a local Jupyter notebook environment, to later be developed on a Git repository. The repository will have the folders for EDA, Poisson Regression, Random Forest, DNN, and Feature Engineering, to match the modular breakdown of the scripts. Any updates to the product will be implemented on the Git repository.

Processing of data went through the following steps:

- EDA
    - The structure of the data was viewed.
    - The data was cleaned of any missing or duplicated values, and outliers were handled. Data types were also explored for appropriateness with the given data.
    - Graphical representations of the data's structure were generated.
    - Cleaned data was saved for future use.
- Poisson Regression
    - Data was split into training and testing groups.
    - The model was initialized and then fit.
    - Model predictions were generated.
    - A graph of the predicted values against original values were generated.

    - The performance metrics Pseudo R-squared, mean squared error, and cross validation were computed.
    - All performance metrics were saved to a file.
    - Analysis showed that logarithmically transformed data had the best results, but none were strong enough to use for making informed decisions upon.
- Random Forest
    - Data was split into training and testing groups.
    - The model was initialized and then fit.
    - Model predictions were generated.
    - A graph of the predicted values against original values were generated.
    - The performance metrics R-squared, mean squared error, and cross validation were computed.
    - All performance metrics were saved to a file.
    - Analysis showed that logarithmically transformed data had the best results, but none were strong enough to use for making informed decisions upon.
- DNN
    - Data was split into training and testing groups.
    - The model was initialized and then fit.
    - Model predictions were generated.
    - A graph of the predicted values against original values were generated.
    - The performance metrics R-squared and mean squared error.
    - All performance metrics were saved to a file.
    - Analysis showed that logarithmically transformed data had the best results, but none were strong enough to use for making informed decisions upon.
- Feature Engineering
    - Data was cleaned, preprocessed, and logarithmically transformed.
        - The data now included a section that allows for seasonality to be included in the model structure.
    - Data was split into training and testing groups.
    - Poisson Regression, Random Forest, and DNN models were initialized and then fit.
    - Predictions were generated for each model.
    - Graphs of the predicted values against original values were generated for each model.
    - The performance metrics for Poisson Regression included:
        - Pseudo R-squared, mean squared error, and cross validation
    - The performance metrics for Random Forest included:
        - R-squared, mean squared error, and cross validation
    - The performance metrics for DNN included:
        - R-squared and mean squared error
    - All performance metrics were saved to a file.

- Analysis showed that including seasonality as a feature engineering technique did not have a strong impact on each model's performance metrics. It is recommended that future research on similar data should analyze the data in more local forms, in order to have better success at creating quality models.

For users, an option to upload new data through a URL is available. Interactive graphs in the exploratory data analysis will allow for selection of different countries or different structures of the data. PDF files and PNG images of the outputs, reports, and generated graphs will be available for transparency to users. Additionally, a User Guide is to follow this section.

Potential risks involving users include uploading inappropriate, inconsistent, or inaccurate data could lead to poor model performance. Additionally, any newly uploaded data must be integrated into the remainder of the analysis, whether through merging or replacement of the current data. Otherwise, the data will not be included with the current structure of the product. If any data with personal identifiers is uploaded, encryption must be included to protect those at risk. The current data set did not include any identifiable information, so encryption techniques were not accounted for. Security measures to protect the integrity of the model were also not implemented, as the product is for informational purposes only. This leads to the risk of improper modification of the model, which could lead to misleading results. It is imperative that any users using the models as a basis for their own projects verify the structure of the model prior to implementation. One final risk is making misinformed decisions based on the models. The models are highly limited and have consistently shown poor performance metrics and replicability. At this stage, it is recommended that these models are poor choices for modeling big pandemic related data, and that localization of data is the preferred method of analysis. Future techniques could be developed that identify models that are able to handle the data's complexity, but were not yet identified in this product. While the models were not successful, a better understanding of how worldwide health related data should begin analysis process was found, and future researchers have a recommended starting point for their individual research. This should save researchers valuable time and allow for more efficiency in their own developing code. More detailed findings can be seen in the source code.

Big Data Analysis of COVID-19 Vaccinations and COVID-19 Related Deaths: A User Guide

By Katelyn M. Campbell

User Guide:

Big Data Analysis of COVID-19 Vaccinations and COVID-19 Related Deaths

Katelyn M. Campbell

College of Engineering and Technology

DSC-580-O500 Designing and Creating Data Products

Professor Jonathan Pollyn

April 23rd, 2025

Preface

This document is a user guide that companions the data product, Big Data Analysis of COVID-19 Vaccinations and COVID-19 Related Deaths. Its intended purpose is to serve as a guideline for worldwide health related data research, specifically pandemic related research. The audience for this product is other fellow researchers to gain insights on best handling data with similar structure. By reviewing this product, they can identify the most appropriate and efficient methods for their own research. Within the product, there is exploratory data analysis with data cleaning, Poisson Regression modeling, Random Forest modeling, Deep Neural Network modeling, and feature engineering. While a logarithmic regression was found to be most efficient at minimizing the mean squared error, none of the models, even with feature engineering, showed promise of replicable results or large R-squared values. Due to this limitation, it is recommended of future researchers to localize their data sets, giving a more centralized focus on data for models to learn on. Large, worldwide data sets are hard to create reliable models upon and often exhibit poor performance. To save time and gain efficiency, the localized data will be easier and quicker to implement for researchers. However, there are many different modeling systems available, and future techniques or research could be identified that are capable of handling big data.

Table of Contents

Note: Please use Ctrl + F to search this document as needed

General Information

This user guide contains five sections: EDA (Exploratory Data Analysis), Poisson Regression, Random Forest, Deep Neural Network, and Feature Engineering. The intended users of this product are researchers attempting to model similar data sets. This is version 1, released on April 23rd, 2025.

EDA (Exploratory Data Analysis): Within the Exploratory Data Analysis, the product explores the data through descriptive statistics, a variety of graphical representations of the data, and data cleaning procedures.

Poisson Regression: In this section, the previously cleaned data is split it into training and testing groups. Initiation and fitting of a Poisson Regression model is performed. Predictions are made from this, with a scatterplot of the observed and predicted values plotted against one another. Performance metrics, such as pseudo R-squared, mean squared error, and cross validation are computed. A discussion of the interpretation of results (based on the last run model), user interface, security, and revisions are included. Any new iterations will present a mismatch between the current interpretation. All references utilized within the work are included.

Random Forest: In this section, the previously cleaned data is split it into training and testing groups. Initiation and fitting of a Random Forest model is performed. Predictions are made from this, with a scatterplot of the observed and predicted values plotted against one another. Performance metrics, such as R-squared, mean squared error, and cross validation are computed. A discussion of the interpretation of results (based on the last run model), user interface, security, and revisions are included. Any new iterations will present a mismatch between the current interpretation. All references utilized within the work are included.

DNN (Deep Neural Network): In this section, the previously cleaned data is split it into training and testing groups. Initiation and fitting of a DNN model is performed. Predictions are made from this, with a scatterplot of the observed and predicted values plotted against one another. Performance metrics, such as R-squared and mean squared error are computed. A discussion of the interpretation of results (based on the last run model), user interface, security, and revisions are included. Any new iterations will present a mismatch between the current interpretation. All references utilized within the work are included.

Feature Engineering: Initial iterations of the model did not show ideal results for R-squared values, graphs of observed against predicted values, or cross validation results. It was identified in the EDA that there could be seasonality or trends present in the data. Therefore, feature engineering that included seasonality within the Poisson Regression, Random Forest, and DNN models, was implemented. Matching graphs and performance metrics for each were computed. Interpretations were made, but any new interations will modify the results. User interface, security, revisions, and references are included.

System Summary

The main features of the system are data input, descriptive statistics, data cleaning visualizations, modeling, predictions, and performance metrics. The system is broken into five different components: Exploratory Data Analysis (including data input, descriptive statistics, visualizations, data cleaning, and saved data files), Poisson Regression (including data input, modeling, predictions, visualizations, and saved results), Random Forest (including data input, modeling, predictions, visualizations, and saved results), Deep Neural Network (including data input, modeling, predictions, visualizations, and saved results), and Feature Engineering (including data input, data cleaning, modeling, predictions, visualizations, and saved results).

Technologies used are Microsoft VS Code, Jupyter Notebook, Python, Pandas, Matplotlib, Seaborn, Scikit-Learn, TensorFlow, Keras, SciPy, NumPy, Statsmodels, Statistics, Os, Math, Graphiz, Dash, Time, URLLib, and Plotly. Exact libraries can be found in the beginning of the source code. Python version 3.12 was used. All libraries are required for complete functionality of code.

Inputs were taken from Our World in Data, specifically their csv files "daily-covid-19-vaccine-doses-administered" and "daily-new-confirmed-covid-19-deaths-per-million-people were utilized" were merged together (Appel et al., 2025). There is an option for uploading data through URL, but the escape key can be used to bypass this. If any new data were used for input, there will likely be a need to be changes made to how the data is called to ensure accurate referencing to areas of the data frame.

Reference:

Appel, C., Beltekian, D., Dattani, S., Gavrilov, D., Giattino, C., Hasell, J., Macdonald, B., Mathieu, E., Ortiz-Ospina, E., Ritchie, H., Rodes-Guirao, L., & Roser, M. (2025). COVID-19 pandemic [Data set]. Our World in Data.

Outputs include saved cleaned data, any graphs, and performance metrics generated throughout the product.

Getting Started

1) **Installation:** To begin, first install any programs or applications not previously owned. Follow the instructions provided by the developers that are appropriate for your operating system. Once installed, install any libraries that are not already downloaded into your environment using *pip install*.
2) **Clone GitHub repository**: Into your environment, copy the GitHub repository and upload data or copy any code desired.
3) **Revisions**: If desired, make any appropriate revisions in your environment. This could be uploading your own data set and making appropriate adjustments or adding any new functionalities.
4) **Running**: Run all cells as your first iteration, in sequential order.
5) **Revisions**: If any errors or warnings are found, update with the appropriate revisions. If no changes were made the program should run completely through on the first iteration.
6) **Running**: After any corrections or adjustments are made, run your second iteration.
7) **Interpretation**: Examine all results and draw conclusions from them.

Using the System

Exploratory Data Analysis (EDA):

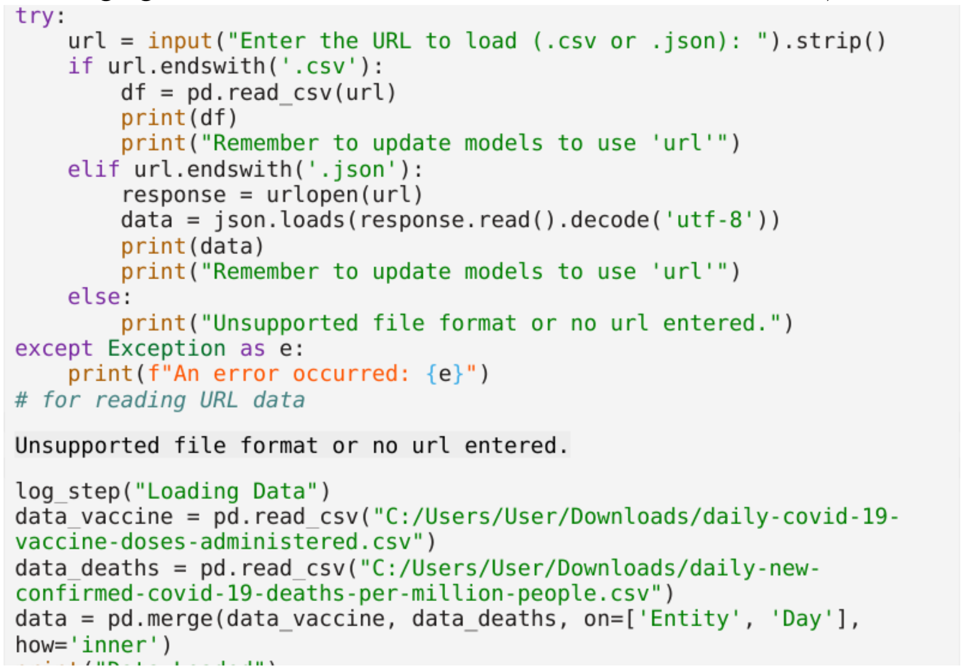
1. After the appropriate system applications are installed, install and load all of the packages. The ones used throughout the project include:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
import scipy.stats as stats
import statistics
import numpy as np
from sklearn import linear_model
from sklearn.model_selection import train_test_split
import sklearn
import warnings
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
import os
import math
import graphviz
from tensorflow.keras.models import Sequential
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input as KerasInput
from sklearn.pipeline import Pipeline
```

```python
from sklearn.tree import plot_tree
from graphviz import Digraph
import matplotlib.image as mpimg
from dash import Dash, dcc, html, Input, Output
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import time
from urllib.request import urlopen
from scipy.stats import chi2_contingency
from sklearn.model_selection import KFold
#loads necessary packages
```

2. Load the data. There is a URL option, which will include a pop-up to upload data through. If this is not desired, hit the escape button. Doing this will product the message "Unsupported file format or no url entered." If using Pandas to read a csv file that is not the originally noted data, replace with the path to your file. Depending on the structure of the data, names of the columns may need to be adjusted. (Note, this data set did require

the merging of two data sets to hold all the wanted information.)

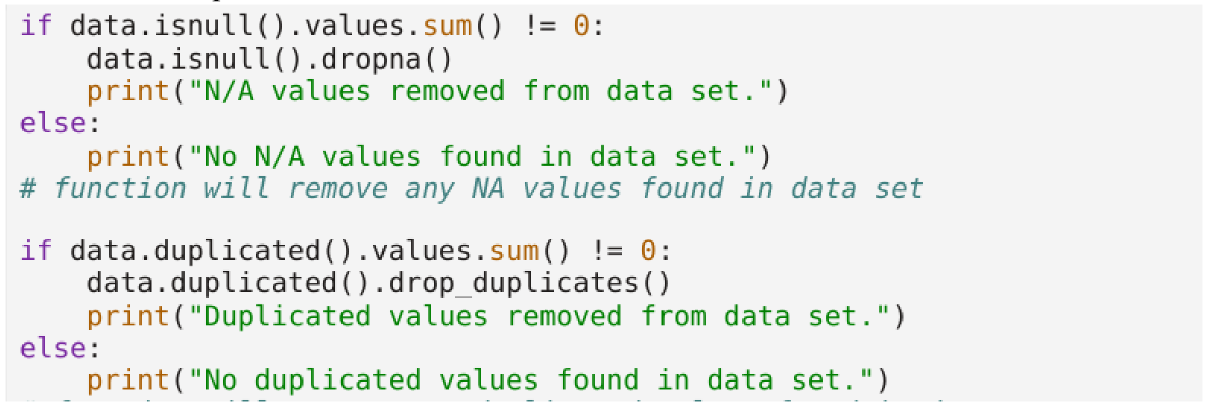


```
try:
    url = input("Enter the URL to load (.csv or .json): ").strip()
    if url.endswith('.csv'):
        df = pd.read_csv(url)
        print(df)
        print("Remember to update models to use 'url'")
    elif url.endswith('.json'):
        response = urlopen(url)
        data = json.loads(response.read().decode('utf-8'))
        print(data)
        print("Remember to update models to use 'url'")
    else:
        print("Unsupported file format or no url entered.")
except Exception as e:
    print(f"An error occurred: {e}")
# for reading URL data
```

```
Unsupported file format or no url entered.
```

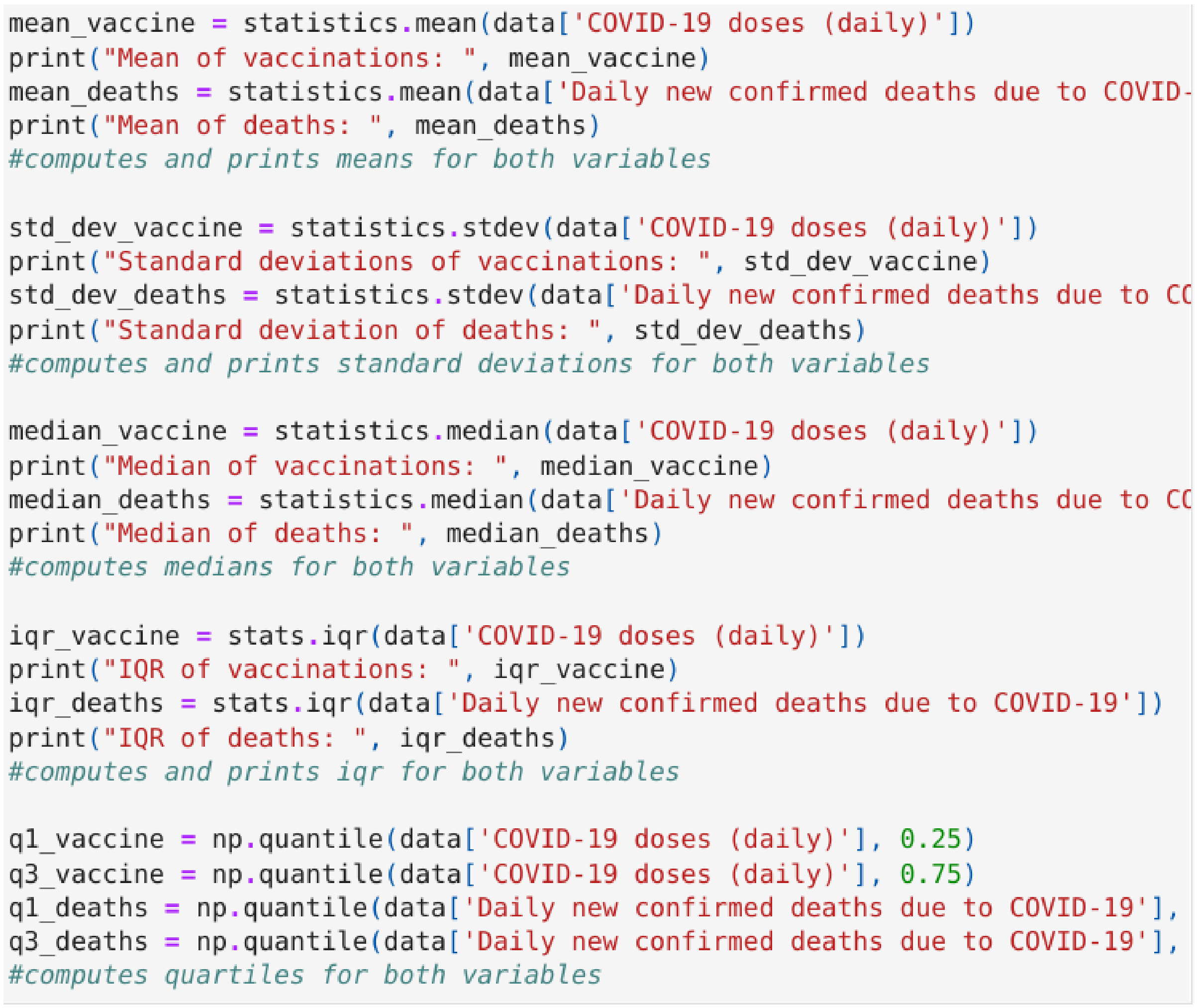
```
log_step("Loading Data")
data_vaccine = pd.read_csv("C:/Users/User/Downloads/daily-covid-19-
vaccine-doses-administered.csv")
data_deaths = pd.read_csv("C:/Users/User/Downloads/daily-new-
confirmed-covid-19-deaths-per-million-people.csv")
data = pd.merge(data_vaccine, data_deaths, on=['Entity', 'Day'],
how='inner')
```

3. As for performing data cleaning and preprocessing, outlier handling, missing value removal, and duplicated value removal are included.

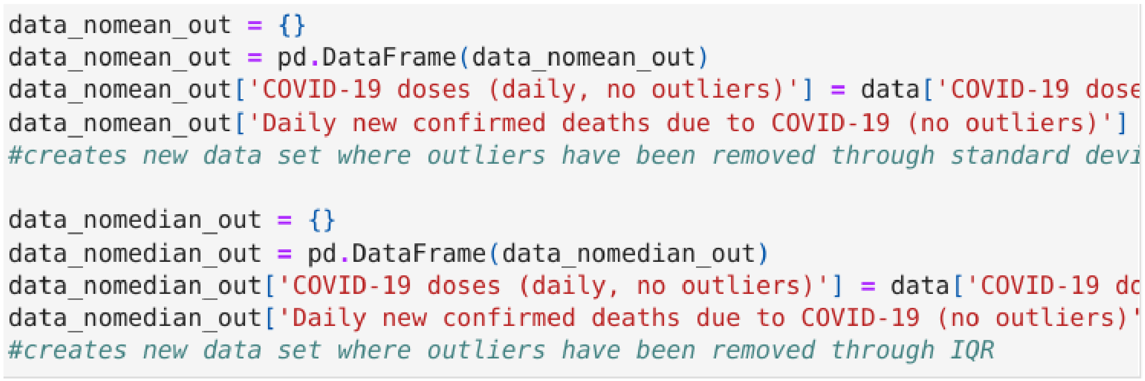


```
if data.isnull().values.sum() != 0:
    data.isnull().dropna()
    print("N/A values removed from data set.")
else:
    print("No N/A values found in data set.")
# function will remove any NA values found in data set

if data.duplicated().values.sum() != 0:
    data.duplicated().drop_duplicates()
    print("Duplicated values removed from data set.")
else:
    print("No duplicated values found in data set.")
```

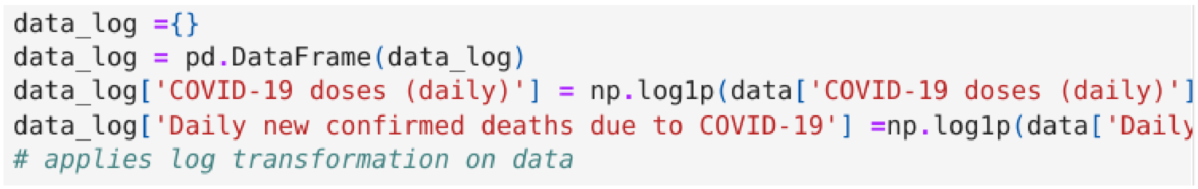
There are four different techniques in outlier handling included, keeping outliers, removing outliers through three standard deviations away, removing outliers through one and half interquartile ranges away, and logarithmic transformation. It is likely one technique is more appropriate for the data set chosen than others.



```
mean_vaccine = statistics.mean(data['COVID-19 doses (daily)'])
print("Mean of vaccinations: ", mean_vaccine)
mean_deaths = statistics.mean(data['Daily new confirmed deaths due to COVID-
print("Mean of deaths: ", mean_deaths)
#computes and prints means for both variables

std_dev_vaccine = statistics.stdev(data['COVID-19 doses (daily)'])
print("Standard deviations of vaccinations: ", std_dev_vaccine)
std_dev_deaths = statistics.stdev(data['Daily new confirmed deaths due to CC
print("Standard deviation of deaths: ", std_dev_deaths)
#computes and prints standard deviations for both variables

median_vaccine = statistics.median(data['COVID-19 doses (daily)'])
print("Median of vaccinations: ", median_vaccine)
median_deaths = statistics.median(data['Daily new confirmed deaths due to CC
print("Median of deaths: ", median_deaths)
#computes medians for both variables

iqr_vaccine = stats.iqr(data['COVID-19 doses (daily)'])
print("IQR of vaccinations: ", iqr_vaccine)
iqr_deaths = stats.iqr(data['Daily new confirmed deaths due to COVID-19'])
print("IQR of deaths: ", iqr_deaths)
#computes and prints iqr for both variables

q1_vaccine = np.quantile(data['COVID-19 doses (daily)'], 0.25)
q3_vaccine = np.quantile(data['COVID-19 doses (daily)'], 0.75)
q1_deaths = np.quantile(data['Daily new confirmed deaths due to COVID-19'],
q3_deaths = np.quantile(data['Daily new confirmed deaths due to COVID-19'],
#computes quartiles for both variables
```



```
data_nomean_out = {}
data_nomean_out = pd.DataFrame(data_nomean_out)
data_nomean_out['COVID-19 doses (daily, no outliers)'] = data['COVID-19 dose
data_nomean_out['Daily new confirmed deaths due to COVID-19 (no outliers)']
#creates new data set where outliers have been removed through standard devi

data_nomedian_out = {}
data_nomedian_out = pd.DataFrame(data_nomedian_out)
data_nomedian_out['COVID-19 doses (daily, no outliers)'] = data['COVID-19 dc
data_nomedian_out['Daily new confirmed deaths due to COVID-19 (no outliers)'
#creates new data set where outliers have been removed through IQR
```



```
data_log ={}
data_log = pd.DataFrame(data_log)
data_log['COVID-19 doses (daily)'] = np.log1p(data['COVID-19 doses (daily)']
data_log['Daily new confirmed deaths due to COVID-19'] =np.log1p(data['Daily
# applies log transformation on data
```

All could be run to use as comparisons with modeling, or turning the undesired ones into comments using the ‘#’ symbol at the beginning of the line could be done. Additionally, the data types are shown.

```
print(data.dtypes)
```

If any do not appear appropriate for their given column, apply techniques to change the values within the column to the appropriate type. For example, this data needed the date to be in a date time format.

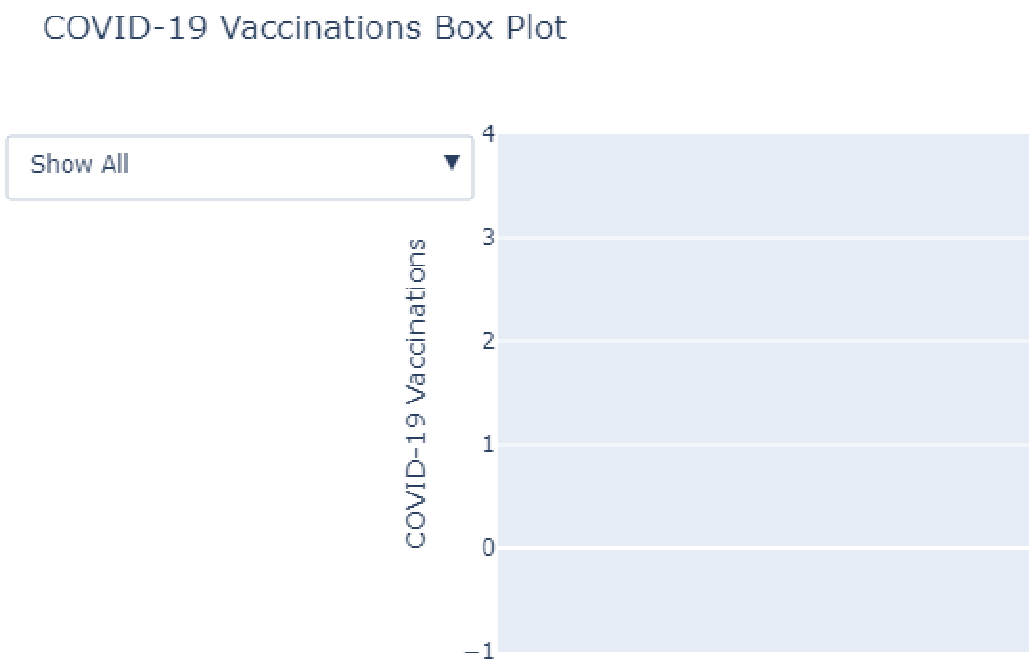
```
data['Day'] = pd.to_datetime(data['Day'])
```

4. For data exploration, a print out of summary statistics and a wide variety of graphs are included. Any graphs can be tailored to different needs.

```
data.describe()
#reports summary statistics on data
```

Currently, there are many different countries included in the data set, so they were the primary focus of interactions available for users. A filter system was included to show whatever the user wanted to focus on.

```
fig_box1 = go.Figure()
# Add traces for each country
for country in unique_countries:
 country_data = data[data["Entity"] == country]
 fig_box1.add_trace(
 go.Box(
 y=country_data["COVID-19 doses (daily)"],
 name=country,
 visible=False # Initially hide all countries
 )
 )
dropdown_buttons = [
 {
 "label": country,
 "method": "update",
 "args": [{"visible": [country == c for c in unique_countries]}]
 }
 for country in unique_countries
]
# Add a "Show All" option
dropdown_buttons.insert(0, {
 "label": "Show All",
 "method": "update",
 "args": [{"visible": [True] * len(unique_countries)}]
})
# Update layout to include dropdown
fig_box1.update_layout(
 updatemenus=[{
 "buttons": dropdown_buttons,
 "direction": "down",
 "showactive": True,
 }],
 title="COVID-19 Vaccinations Box Plot",
 yaxis_title="COVID-19 Vaccinations",
)
fig_box1.show()
# creates interactive box plot of COVID-19 vaccinations administered
```



Additionally, visualizations and descriptions of the data after outlier handling had been implemented were included.

5. Data was saved into different files after cleaning. These files are to be uploaded in the other modules of the product.

```
data.to_csv('data_clean.csv', index=False)
data_nomean_out.to_csv('data_nomean_out.csv', index=False)
data_nomedian_out.to_csv('data_nomedian_out.csv', index=False)
data_log.to_csv('data_log.csv', index=False)
# saves cleaned data, outlier removed data (by standard deviaitons and IQRs)
```

Poisson Regression:

1. Downloading the required libraries is the same as shown in the EDA module (page 24).
2. Uploading the data is also the same as shown in the EDA module (pages 24-25).
3. Splitting data into training and testing groups is important for capturing the ability of the model to accurately generate predictions. The model was split into eighty percent of the data into training, and twenty percent of the data into training. This was done for each different outlier handling data sets created.

```
X =data_clean['COVID-19 doses (daily)']
y = data_clean['Daily new confirmed deaths due to COVID-19']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. The Poisson Regression model was initialized through the following code (for each different outlier handling data sets created):

```
clf = linear_model.PoissonRegressor()
```

5. It was then fit through running (for each different outlier handling data sets created)

```
model = clf.fit(X_train.values.reshape(-1, 1),y_train)
```

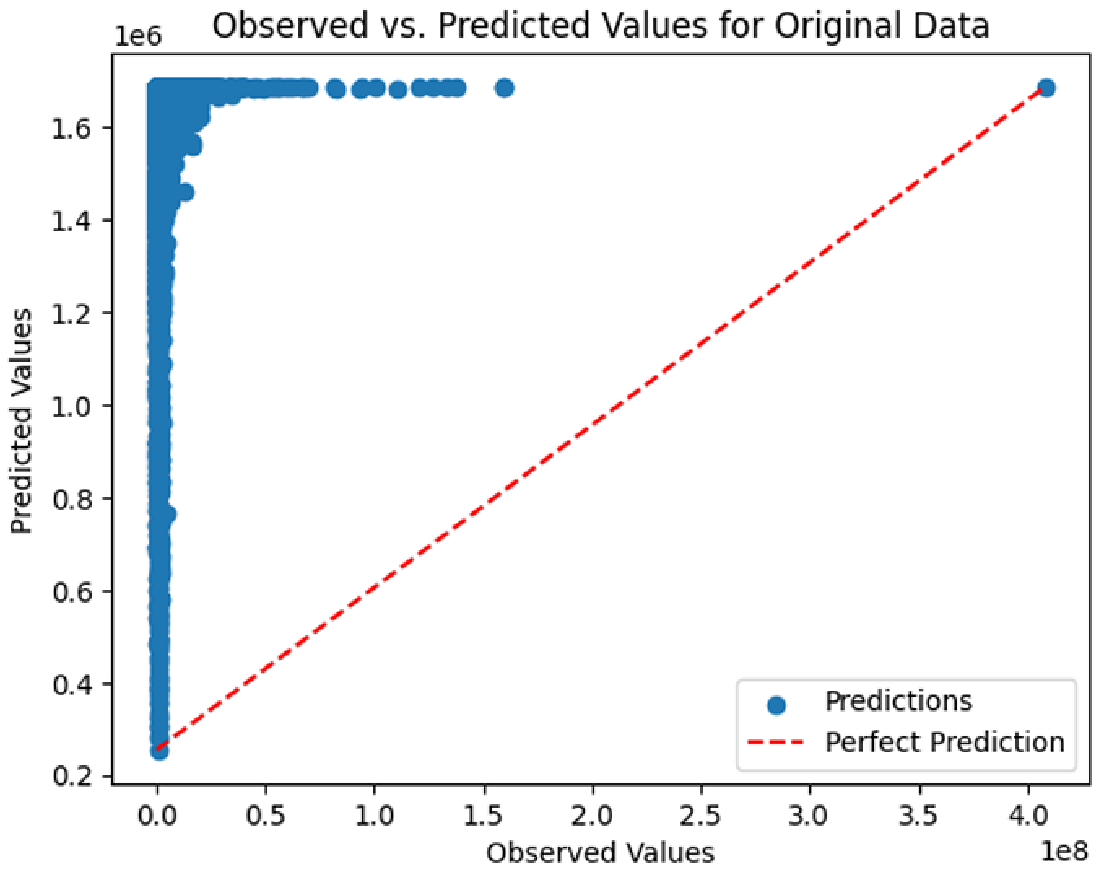
6. Then, the predictions were generated by running (for each different outlier handling data sets created):

```
predictions = clf.predict(X_test.values.reshape(-1, 1))
```

7. Scatterplots of the observed against predicted values were generated, (for each different outlier handling data sets created), to show visualizations of how well the model is predicting the data. The more data that is situated around the line of best fit, the better the model is performing.

```
plt.scatter(y_test, predictions,label='Predictions')
plt.plot([min(y_test), max(y_test)], [min(predictions),
max(predictions)], linestyle='--', color='red', label='Perfect
Prediction')
plt.xlabel('Observed Values')
plt.ylabel('Predicted Values')
plt.title('Observed vs. Predicted Values for Original Data')
plt.legend()
plt.show()
plt.savefig("poisson_observed_vs_predicted.png")
#graphs and saves observed vs. predicted values
```

A copy of one of the generated graphs is shown below:



8. The next step is generating the performance metrics of the models. The mean squared error was found using:

```
mse = mean_squared_error(y_test, predictions)
```

A lower score for this metric is ideal.

The Pseudo R-squared value was found using:

```
score = clf.score(X_test.values.reshape(-1, 1), y_test)
```

A score closer to 1 or –1 is ideal for this metric.

Cross validation was performed through:

```
cv_results = cross_validate(clf, X_train.values.reshape(-1, 1), 
y_train,
                            cv=5,
                            scoring=['neg_mean_squared_error', 'r2'],
                            return_train_score=True)
```

Values should mimic those initially computed to indicate replicability of the model.

9. All performance metrics are saved into a PDF file when the following code is run:

```
with open("poisson_model_report.txt", "w") as f:
    f.write("Poisson Model Performance Report\n")
    f.write("========================\n")
    f.write(f"Mean Squared Error (MSE): "+ str(mse) + "\n")
    f.write(f"R-squared (R²): " + str(score) + "\n")
    f.write(f"Cross-Validation (MSE): " + str(cv_results['test_r2']) +
"\n")
    f.write(f"Average CV Score (R²): " +
str(cv_results['test_neg_mean_squared_error']) + "\n")
# creates pdf of results
```

10. Interpretations should be completed based on the results found.
11. If desired, a diagram of the model's structure can be visualized by running the following code:

```
def plot_poisson_regression():
 dot = Digraph()

 dot.node('X1', 'Feature X1')

 dot.node('Lin', 'Linear Combination: β0 + β1X1')
 dot.edge('X1', 'Lin')

 dot.node('Poisson', 'Poisson Link Function: exp(Lin)')
 dot.edge('Lin', 'Poisson')

 dot.node('Y', 'Predicted Count (Y)')
 dot.edge('Poisson', 'Y')
 return dot
dot = plot_poisson_regression()
img = mpimg.imread('poisson_regression.png')
plt.figure(figsize=(6, 6))
plt.imshow(img)
plt.axis('off') # Hide axes
plt.show()
# Saves and opens the diagram
```

Random Forest:

1. Downloading the required libraries is the same as shown in the EDA module (page 24).
2. Uploading the data is also the same as shown in the EDA module (pages 24-25).
3. Splitting data into training and testing groups is important for capturing the ability of the model to accurately generate predictions. The model was split into eighty percent of the data into training, and twenty percent of the data into training. This was done for each different outlier handling data sets created.

```
X =data_clean['COVID-19 doses (daily)']
y = data_clean['Daily new confirmed deaths due to COVID-19']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. The Random Forest model was initialized through the following code (for each different outlier handling data sets created):

```
regressor = RandomForestRegressor(n_estimators=10, random_state=0,
oob_score=False)
```

5. It was then fit through running (for each different outlier handling data sets created):

```
regressor.fit(X_train.values.reshape(-1, 1), y_train)
```

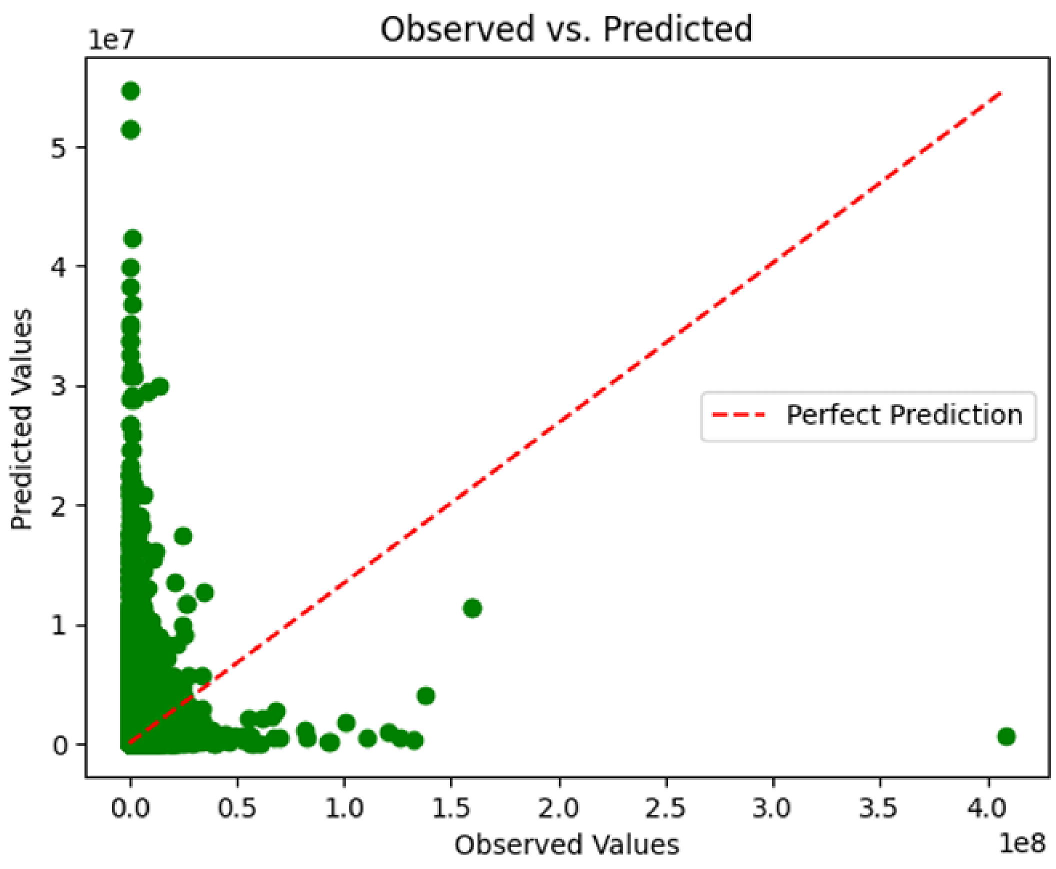
6. Then, the predictions were generated by running (for each different outlier handling data sets created):

```
predictions_RF = regressor.predict(X_test.values.reshape(-1, 1))
```

7. Scatterplots of the observed against predicted values were generated, (for each different outlier handling data sets created), to show visualizations of how well the model is predicting the data. The more data that is situated around the line of best fit, the better the model is performing.

```
plt.scatter(y_test, predictions_RF,color='green')
plt.plot([min(y_test), max(y_test)], [min(predictions_RF),
max(predictions_RF)], linestyle='--',color='red', label='Perfect
Prediction')
plt.title("Observed vs. Predicted")
plt.xlabel('Observed Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
plt.savefig("RF_observed_vs_predicted.png")
#plots and saves observed vs. predicted graph
```

A copy of one of the generated graphs is shown below:



8. The next step is generating the performance metrics of the models. The mean squared error was found using:

```
mse_RF = mean_squared_error(y_test, predictions_RF)
```

A lower score for this metric is ideal.

The R-squared value was found using:

```
r2_RF = r2_score(y_test, predictions_RF)
```

A score closer to 1 or –1 is ideal for this metric.

Cross validation was performed through:

```
cv_results = cross_validate(regressor, X_train.values.reshape(-1, 1),
y_train,
                            cv=5,
                            scoring=['neg_mean_squared_error', 'r2'],
                            return_train_score=True)
```

Values should mimic those initially computed to indicate replicability of the model.

9. All performance metrics are saved into a PDF file when the following code is run:

```
with open("RF_model_report.txt", "w") as f:
    f.write("Random Forest Model Performance Report\n")
    f.write("========================\n")
    f.write(f"Mean Squared Error (MSE): " + str(mse RF) + "\n")
    f.write(f"R-squared (R²): " + str(r2 RF) + "\n")
    f.write(f"Cross-Validation (MSE): " + str(cv_results['test_r2']) +
"\n")
    f.write(f"Average CV Score (R²): "+
str(cv_results['test_neg_mean_squared_error']) + "\n")
# creates pdf of results
```

10. Interpretations should be completed based on the results found.
11. If desired, a diagram of the model's structure can be visualized by running the following code:

```
plot_tree(regressor.estimators_[0],
 filled=True,
 rounded=True,
 max_depth=3)
plt.title("Decision Tree from the Random Forest")
plt.show()
# graphs structure of the model
```

DNN:

1. Downloading the required libraries is the same as shown in the EDA module (page 24).
2. Uploading the data is also the same as shown in the EDA module (pages 24-25).
3. Splitting data into training and testing groups is important for capturing the ability of the model to accurately generate predictions. The model was split into eighty percent of the data into training, and twenty percent of the data into training. This was done for each different outlier handling data sets created.

```
X =data_clean['COVID-19 doses (daily)']
y = data_clean['Daily new confirmed deaths due to COVID-19']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

4. The DNN model was initialized and compiled through the following code (for each different outlier handling data sets created):

```
dnn model = Sequential()
dnn model.add(KerasInput(shape= (X train.shape[1],)))
dnn_model.add(Dense(256, activation='relu'))
dnn model.add(Dense(128, activation='relu'))
dnn model.add(Dense(64, activation='relu'))
dnn model.add(Dense(32, activation='relu'))
dnn model.add(Dense(16, activation='relu'))
dnn model.add(Dense(8, activation='relu'))
dnn model.add(Dense(1))
dnn_model.compile(loss='mean_squared_error', optimizer='adam')
# compiles dnn model using adam as optimizer (Brownlee, 2022)
```

Any appropriate modifications can be made to the added layers and optimizer.

5. It was then fit through running (for each different outlier handling data sets created):

```
dnn model.fit(X train, y train, epochs=100, batch_size=32,
validation data=(X test, y test))
# fits dnn model (Brownlee, 2022)
```

   The number of epochs can be changed if desired.

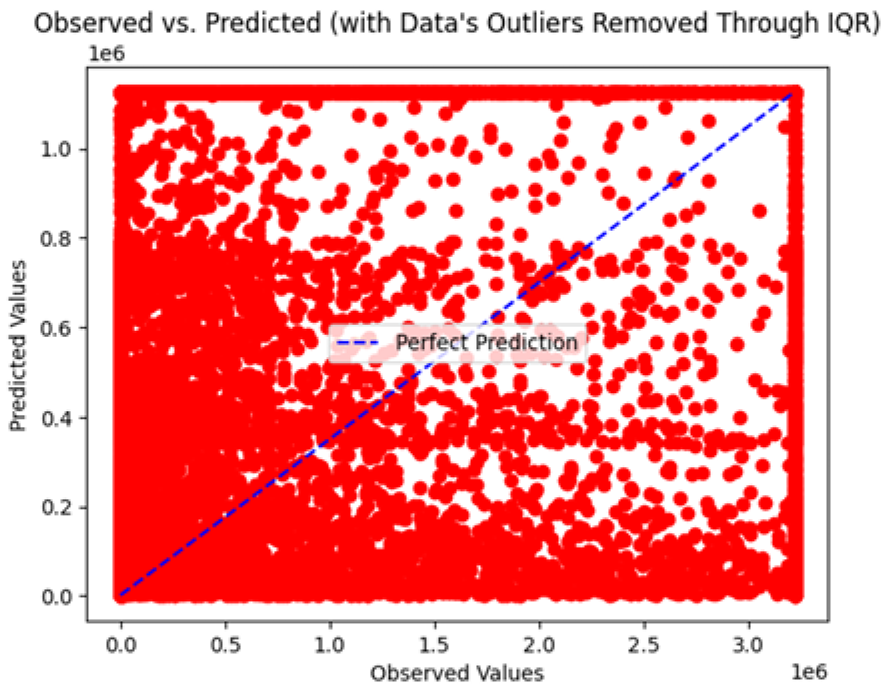
6. Then, the predictions were generated by running (for each different outlier handling data sets created):

```
predictions dnn model = dnn model.predict(X test)
# creates predictions from dnn model (Brownlee, 2022)
```

7. Scatterplots of the observed against predicted values were generated, (for each different outlier handling data sets created), to show visualizations of how well the model is predicting the data. The more data that is situated around the line of best fit, the better the model is performing.

```
plt.scatter(y  median test, predictions dnn median,color='red')
plt.plot([min(y  median test), max(y  median test)],
[min(predictions_dnn_median), max(predictions_dnn_median)],
linestyle='--', color='blue', label='Perfect Prediction')
plt.title("Observed vs. Predicted (with Data's Outliers Removed
Through IQR)")
plt.xlabel('Observed Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.show()
plt.savefig("dnn observed vs predicted median.png")
#plots and saves observed vs. predictions graph (outliers removed
through IQR)
```

   A copy of one of the generated graphs is shown below:



8. The next step is generating the performance metrics of the models. The mean squared error was found using:

```
mse dnn = mean squared error(y test, predictions_dnn_model)
```

   A lower score for this metric is ideal.
9. The R-squared value was found using:

```
r2 dnn = r2 score(y test, predictions_dnn_model)
```

   A score closer to 1 or –1 is ideal for this metric.
10. All performance metrics are saved into a PDF file when the following code is run:

```
with open("DNN model report.txt", "w") as f:
    f.write("DNN Model Performance Report\n")
    f.write("========================\n")
    f.write(f"Mean Squared Error (MSE): " +  str(mse_dnn) + "\n")
    f.write(f"R-squared (R²): " + str(r2_dnn) + "\n")
# creates pdf of results
```

11. Interpretations should be completed based on the results found.
12. If desired, a diagram of the model's structure can be visualized by running the following code:

```
plot model(dnn log, to_file='model_plot.png', show_shapes=True,
show layer names=True)
#gets diagram of model
```

Feature Engineering:

1. Downloading the required libraries is the same as shown in the EDA module (page 24).
2. Uploading the data is also the same as shown in the EDA module (pages 24-25).
3. It is necessary for the data to be appropriately prepared for seasonal analysis. While the cleaning and preprocessing are the same as the logarithmic transformed data in the EDA

module, (pages 23-27), the following addition was included:

```
data['day_of_year'] = data['Day'].dt.dayofyear
data['sin_year'] = np.sin(2 * np.pi * data['day_of_year'] / 365)
```

```
data['cos_year'] = np.cos(2 * np.pi * data['day_of_year'] / 365)
# adds seasonality columns

log_step("Data Cleaned and Preprocessed")
#prints that step was completed
```

4. The modules were adjusted to include this information:
   Poisson Regression:

```
split_date = data['Day'].quantile(0.8)
train_data = data[data['Day'] <= split_date]
test_data = data[data['Day'] > split_date]
#splits data into training and testing for Poisson

poisson = smf.glm(
    formula='COVID_deaths_daily ~ COVID_doses_daily + time_index +
sin_year + cos_year',
    data=train_data,
    family=sm.families.Poisson()
).fit()
```

   Random Forest:

```
X = data[['COVID_doses_daily', 'time_index', 'day_of_year',
'sin_year', 'cos_year']]
y = data['COVID_deaths_daily']
X_train, X_test = X.iloc[:-int(len(X)*0.2)], X.iloc[-int(len(X)*0.2):]
y_train, y_test = y.iloc[:-int(len(y)*0.2)], y.iloc[-int(len(y)*0.2):]
# splits data into training and testing for Random Forest
```

DNN (this module uses the same training and testing split as the Random Forest model):

```
dnn = Sequential()
dnn.add(KerasInput(shape= (X_train.shape[1],)))
dnn.add(Dense(256, activation='relu'))
dnn.add(Dense(128, activation='relu'))
dnn.add(Dense(64, activation='relu'))
dnn.add(Dense(32, activation='relu'))
dnn.add(Dense(16, activation='relu'))
dnn.add(Dense(8, activation='relu'))
dnn.add(Dense(1))
dnn.compile(loss='mean_squared_error', optimizer='adam')
# compiles dnn model using adam as optimizer (log transformed data)
(Brownlee, 2022)

dnn.fit(X_train, y_train, epochs=100, batch_size=32,
validation_data=(X_test, y_test))
# fits dnn model (log transformed data) (Brownlee, 2022)
```

5. After these models are generated, the predictions, visualizations, and performance metrics are computed the same as the modules of the matching models. These cells should be run. The only exception is the Poisson Regression, as it utilized a different package to generate the model. Its performance metrics were found using the following:

```
print(poisson.summary())
# prints model summary
```

```
mse_poisson = mean_squared_error(test_data['COVID_deaths_daily'],
predictions_poisson)
print(f'Mean Squared Error: {mse_poisson}')
```

```
Mean Squared Error: 33.60495405550085
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
pseudo_r2_scores = []

for train_index, test_index in kf.split(data):
    train_data = data.iloc[train_index]
    test_data = data.iloc[test_index]

    model = smf.glm(formula='COVID_deaths_daily ~ COVID_doses_daily +
time_index + sin_year + cos_year', data=train_data,
family=sm.families.Poisson()).fit()
    predictions = model.predict(test_data)

    poisson_actual = test_data["COVID_deaths_daily"]
    mse_poisson = np.mean((poisson_actual - predictions_poisson)**2)

    null_model = smf.glm(formula="COVID_deaths_daily ~ 1",
data=train_data, family=sm.families.Poisson()).fit()

    llf = model.llf
    llnull = null_model.llf
    pseudo_r2 = 1 - (llf / llnull)
    pseudo_r2_scores.append(pseudo_r2)

print("Cross-validated MSE scores:", mse_poisson)
print(f"Pseudo R²: {pseudo_r2_scores}")
# performs cross validation and finds MSE and R-squared for poisson
regression
```

```
Cross-validated MSE scores: 33.50836337221961
Pseudo R²: [0.10258297887409884, 0.10211526987769781,
0.10179679285875043, 0.10415597003880239, 0.1024017633566261]
```

6. Interpretations should be completed based on the results found.
7. The saving of any generated reports follows the same structure as those of prior modules, with the exception of the Poisson Regression model containing the structure listed above.

For any individual with poor vision, this document can be hit through the “ctrl” and “+” symbols held down simultaneously, until the document can be viewed more comfortably. Any individuals that are colorblind, a grayscale version of this document has been included in a separate file.

Troubleshooting

If any of the modules appear to be taking too long to generate, check that the program is not waiting for you to upload URL data or hit escape. The text box for this is small and can easily be missed.

If the module has an error where something is not found and new data was uploaded, it is likely that the data or column names were not accurately adjusted to those now present.

If the module has an error where something is not found and new data was not uploaded, check to ensure that all cells above the one have run. A cell that hasn't been run could contain necessary saved information for the cells to come.

There are lines of code that specify the shape of the data originally used. If the model does not run, it could be that the shape of the data does not match what was specified in the model.

It is important to use data sets that are large enough to represent the population, but not too diverse for the model to learn accurately from. A model that begins with poor data quality will have poor performance.

Again, a limitation of this product is that the models all perform poorly. The worldwide data is too diverse to learn from. Do not make any decisions based on the models generated in this product. It is recommended to use more localized data to train and analyze.

Any errors or issues not anticipated can be researched through online queries, or contacting the developer (page 43).

FAQ

Can I take this product and use it to analyze my own data?

Yes! This product is meant to be used as a resource for future health research projects.

Can new data be added?

Yes. Ensure that there is proper merging, (examples are included in the EDA and Feature Engineering modules), or replacement of the column names throughout the product.

Is log transformation always the best outlier handling technique?

No. These models performed best with logarithmic transformed data, likely due to the extent of outliers that were present in the data. In other instances, another technique may be more efficient for different data sets. This is why the other techniques were left in the product’s code.

Will the model be updated with new data as it is gathered?

At this time, there is no intention of updating the data as it is gathered. The models did not perform well with the current data. It is likely that more data could create more confusion in the model, instead of improving performance. This product had the intentions of being a resource for steps future research should take, and what models to begin with. At this time, it is recommended to not use big, worldwide data for pandemic related data. More localized data should be used, and it is not believed that more data will change this outcome.

Why did the models not perform well?

Through the logarithmic transformation, the mean squared errors were lowered to appropriate values. However, the R-squared values, predicted against observed graphs, and cross validation results did not show promising performance. This is likely because there is differences in reporting around the world. Lower populated or economic areas likely have less resources, and do not have as frequent reporting as other locations. This is likely causing the vast differences seen in the data and making it difficult for modules to learn. Therefore, more localized studies will likely improve model performance.

What are other potential methods to generate better model performance?

At this time, taking smaller portions of the data based on their location is recommended to improve model performance. However, it is possible that other feature engineering techniques could be explored and show a change in the performance of the models.

Help and Contact Details

Any additional details on the functions used in this system can be directly found in the environment using ‘help(*function_name*)’ in a new cell. Questions on this product can be sent to the developer:

Katelyn M. Campbell

KCampbell129@my.gcu.edu

Glossary

Box Plot: A box plot is a visualization of the data, showcasing the median, interquartile range, overall range, and outliers present within the data.

Correlation: Correlation is identifying the relationship between two variables.

Cross Validation: Cross validation is a model evaluation technique that creates multiple subsets of the data and varying which are a part of the training set, and which are a part of the testing set. In doing this, printing individual metrics can show if the results are replicable, and printing average results can provide the overall test metrics.

Cumulative: Cumulative is the process of adding the previous value(s) to the current value in a data set. This can be used to show how much a variable has changed over time.

Deep Neural Network (DNN): A Deep Neural Network is a model that is composed of many layers, utilized to gain a deep understanding of the data. Its name is a hint at the similarities of its structure and the biological neural composition,

Differencing: Differencing is when the difference between two successive data points is calculated. This is often used to make time series data more stationary.

Epochs: An epoch is one completion pass of training data when fitting a neural network.

Exploratory Data Analysis (EDA): Exploratory Data Analysis is the process of attempting to gain an early understanding of the data, such as showing the shape through visualizations and/or summary statistics of the data.

Feature Engineering: Feature engineering is the process of specializing a model to the structure of the data. For example, data with many outliers may want to transform the data in some manner to improve its results.

Histogram: A histogram is a visual representation of the data, where the number of occurrences of a value in the data set is expressed in continuous intervals. The structure of its distribution is often an indicator if certain modeling is appropriate for the data, or if transformations on the data are needed.

Interquartile Range (IQR): The interquartile range is the range, or distance, of the data from the seventy-fifth percentile to the twenty-fifth percentile.

Line Graph: A line graph is a visual representation of data containing a numerical dependent and independent variable. All data points are represented in a continuous line.

Logarithmic Transformation: A logarithmic transformation is when data has replaced each variable’s value with its logarithmic equivalent.

Mean Squared Error: The mean squared error is the average squared differences between the observed values and the predicted values.

Observed vs. Predicted Graph: The observed vs. Predicted graph is a scatterplot of the originally seen dependent variables against those that were predicted by a model.

Outlier: An outlier is a data point that is distinct from other data points. This can be identified through three standard deviations away from the mean, or one and a half interquartile ranges away from the median.

Pipeline: A pipeline is a structure of the processes that the data flows through in a project or product.

Poisson Regression: A Poisson Regression is a linear model that uses the Poisson distribution to analyze the probability of the response variable. It is used with count data, specifically, and expects that data to behave linearly.

Predictions: Predictions are estimates at what the dependent variable may be, given the independent variable.

Pseudo R-Squared: Pseudo R-squared is the generalization of the calculation to R-squared for non-linear models (see R-squared).

P-Value: The p-value is the probability of obtaining the observed results, or extreme results, if the null hypothesis is true.

Random Forest: Random Forest is a machine learning algorithm that uses decision trees to make predictions. It combines the information learned in multiple subsets of data in decision trees to make predictions from.

Rolling Mean: The rolling mean takes recent, prior values and averages them into a new value.

R-Squared: R-squared is the measure of variance in the dependent variable that is explained by the independent variables. This is used in linear models.

Scatterplot: A scatterplot is a visual representation of data containing a numerical dependent and independent variable. All data points are expressed as a series of coordinates.

Seasonality: Seasonality is when time data shows a repetitive pattern, either through the time of day, week, month, year, or other time period.

Standard Deviation: The standard deviation is the amount of variation a variable has away from its mean.

Train-Test Split: Train-Test split is the process of splitting the data into a training subset, to fit the model to, and a testing subset, to compare its predictive results against.

Trend: A trend is when time data shows a steady increase or decrease in the data over time.

T-Statistic: The T-Statistic is the difference between standard errors of the sample mean from the population mean.

User Interface: The user interface is the interactive points that exist for users in a data product.

This glossary's definitions were inspired from the text:

Karpatne, A., Kumar, V., Steinbach, M. & Tan, P. (2018). *Introduction to data mining* (2nd ed.). Pearson.

References

Alroy-Preis, S., Angulo, F. Anis, E., Brooks, N., Haas, E. J., Jodar, :., Khan, F., Levy, Y., McLaughlin, J. M., Mircus, G., Pan, K., Singer, S. R., Smaja, M., Southern, J., & Swerdlow, D. L. (2021). Impact and effectiveness of mRNA BNT162b2 vaccine against SARS-CoV-2 infections and COVID-19 cases, hospitalisations, and deaths following a nationwide vaccination campaign in Israel: an observational study using national surveillance data. Lancet, 397(10287), 1819-1829. doi: 10.1016/S0140-6736(21)00947-8

Appel, C., Beltekian, D., Dattani, S., Gavrilov, D., Giattino, C., Hasell, J., Macdonald, B., Mathieu, E., Ortiz-Ospina, E., Ritchie, H., Rodes-Guirao, L., & Roser, M. (2025). COVID-19 pandemic [Data set]. Our World in Data. https://ourworldindata.org/coronavirus

Barbeira, P. B., Bartolomeu, M. L., Castelli, J. M., Del Valle Juarez, M., Esperatti, M., Fuentes, N., Galligani, G., Giovacchini, C. M., Iummato, L. E., Laurora, M., Pennini, V., Pesce, M., Rearte, A. Rearte, R., Santoro, A., Tarragona, S., & Vizzotti, C. (2022). Effectiveness of rAd26-rAd5, ChAdOx1 nCoV-19, and BBIBP-CorV vaccines for risk of infection with SARS-CoV-2 and death due to COVID-19 in people older than 60 years in Argentina: a test-negative, case-control, and retrospective longitudinal study. Lancet, 399(10331), 1254-1264. doi: 10.1016/S0140 6736(22)00011-3

Barron, J. A., Buenrostro-Mariscal, R., Crossa, J., Montesinos-Lopez, A. Montesinos-Lopez, J. C., Montesinos-Lopez, O. A., & Salazar, E. (2021). Application of a Poisson deep neural network model for the prediction of count data in genome-based prediction. The Plant Genome, 14(3). https://doi.org/10.1002/tpg2.20118

Brownlee, J. (2022). Your first deep learning project in Python with Keras step-by-step. Machine Learning Mastery. https://machinelearningmastery.com/tutorial-first-neural-network-python keras/

Elliott, L., Loomis, D., & Richardson, D. B. (2005). Poisson regression analysis of ungrouped data. Occupational and Environmental Medicine, 62, 325-329. DOI: 10.1136/oem.2004.017459

Fandohan, A. B., Kakaï, R. G., & Mushaglusa, C. Z. (2022). Random forest in count data modelling: An analysis of the influence of data features and overdispersion on regression performance. Journal of Probability and Statistics, 1. https://doi.org/10.1155/2022/2833537

GeeksforGeeks. (2025). Ranom forest regression in Python. https://www.geeksforgeeks.org/random-forest-regression-in-python/

Gîrjău, M., Horton, N. J., & Prium, R. (2023). Fostering better coding practices for data scientists. HDSR. https://hdsr.mitpress.mit.edu/pub/8wsiqh1c/release/4

Katla, N. (2020). Poisson regression implementation- Python. Medium. https://medium.com/@kn12/poisson-regression-implementation-python-28d15e95dc15

Karpatne, A., Kumar, V., Steinbach, M. & Tan, P. (2018). *Introduction to data mining* (2nd ed.). Pearson.

Liu, J. (2024). Navigating the financial landscape: The power and limitations of the ARIMA model. Highlights in Science, Engineering and Technology, 88, 747-752. https://drpress.org/ojs/index.php/HSET/article/view/19082/18645

National Cancer Institute. (2023). Cleaning data: The basics. Center for Biomedical Informatics and Information Technology. https://datascience.cancer.gov/training/learn-data-science/clean data-basics

The Pennsylvania State University. (n.d.). 9: Poisson Regression. https://online.stat.psu.edu/stat504/book/export/html/782#:~:text=Interpretations,tabletop%20of%20a%20certain%20area

Plotly. (n.d.). Creating and updating figures in Python. https://plotly.com/python/creating-and updating-figures/#updating-figures

Sarahjane3102. (2022). How to split the dataset With scikit-learn's train_test_split() function GeeksforGeeks. https://www.geeksforgeeks.org/how-to-split-the-dataset-with-scikit-learns train_test_split-function