PROJECT REPORT

By: Brandon Specht, Brandon Cook, Jillian Bice, Griffin Guthery

Our project primarily focuses on the data analysis of hurricane data since 1900. For the purpose of keeping the scope manageable for the amount of time we had, for each hurricane, we cleaned the data set so that for our algorithms it would only contain magnitude, the start year, and the total adjusted damage costs in USD. For our predictive data analysis, we utilized the scikit-learn machine learning package for its numerous tools, accessibility, and compatibility with NumPy and Pandas which we also used for this project. Additionally, for all three of our visualizations, we made use of Matplotlib for its ease of use and interactivity.

The data_processor.py file is where the .csv is loaded, processed, and cleaned so that we can properly sift through the data and transfer it to our algorithms to be loaded into our visualizations. For the purpose of simplification, we are only taking in .csv files for our data. The command line interface (CLI) is able to take in different .csv files and it would work in a theoretical sense with other data so long as it pertains to hurricanes. There is sufficient error handling implemented as well so that the CLI cannot crash the program if the user decides to load in a missing file or if any other generic issues arise. Our initial .csv file contains other storms that are not tropical storms and cyclones (hurricanes). Since we are working with tropical storms and cyclones in the scope of our assignment, we need to be able to filter all of these other storms out. Our clean_data function filters and cleans it by removing rows where the disaster type does not match. After filtering the rows, the method then goes on to filter through the columns to ensure that only the relevant information such as disaster type, start year, magnitude, and others are kept. We are also sure to remove any hurricanes with a magnitude of NaN or 0 because that data would be irrelevant for the training data. Once we have it all filtered, our get_features_and_target method extracts magnitude, start year, and total damage adjusted for inflation. These three rows are then used to train our machine learning algorithms.

Our visualizer.py file is essentially where all of the code for the design of our visualizations using Matplotlib resides. For our three visualizations, we have three static methods tailored for our three types each—a line graph, scatter plot, and bar chart respectively. The first method, plot_windspeed_trend, takes in a data argument as well as a predictions argument. The data argument is a list of lists so that each inner list can contain the wind speeds, years, and damage respectively. We then use a predictions argument which creates a list of years starting from 2024 that then appends the corresponding predicted values. After that, we use the built in matplotlib.pyplot.plot method in order to create the line graph and to make a distinction between the observed and predicted wind speeds. In this case, we simply defined them with the colors red and blue. With the mplcursors library, we can achieve a level of interactivity between the user and the visualization by allowing it so that whenever the user hovers over a data point on the graph, the corresponding value for damage is displayed on the screen. This is done using the on_hover function, which displays the damage value as text with an annotation.

As for the second method, plot_clustered_data, the beginning follows a similar series of steps as the first but with a list of labels instead of predictions. Additionally, it extracts data in a

similar manner to the first, but before it actually goes and plots the data it uses a mask operation to filter out any points of data with a damage value of zero. We do this because for our clusters, we want wind speed to be associated with a damage value above zero, otherwise it would not tell us anything. Once that is done, plt.scatter is used to plot out the wind speeds versus damage. Furthermore, for the sake of readability, a legend is created using Line2D, which is a part of Matplotlib that creates labels for our two clusters. Much like the first module, there is also a cursor function that allows for interactivity, except the annotation displays the starting year of the hurricane.

Our third method, plot_anomalies functions a little differently from the other two graphs. Before elaborating further, the definition of an anomaly within the scope of this project is any hurricane that has low wind speeds and high damages as well as any hurricane that has high wind speeds and low damages. This is because we would generally expect high damages to be caused by storms with volatile and high wind speeds, but there are anomalies that exist which are good to keep track of. Our plot_anomalies method utilizes a list of booleans that indicates whether a data point is an anomaly or not. We do this because for the purpose of our visualization, the main thing that we care about is the existence of an anomaly. There are no other data points that we need for the bar chart visualization that we plot using Matplotlib. It is also important to note that for a bar graph visualization, we can highlight the exact position of the anomalies at each point they appear in the data set by iterating through a list and extracting each point where the value is -1, signifying an anomaly. It is then put into the anomalies_indices list where each position that an anomaly appears is kept track of accurately. Like the other two graphs, we use red and blue as distinct colors to differentiate. In this case, anomalies are plotted in red while the regular data points are plotted in blue. This is made possible because we iterated through the list and found each index that contains an anomaly. Lastly, much like the wind speeds visualization, mplcursors is used to create an annotation for damage values when hovering over each data point.

For our first algorithm, we aimed to create a graph displaying observed hurricane wind speeds over time, as well as predicted wind speeds for the next fifteen years. We chose ordinary least squares linear regression for its simplicity and effectiveness when modelling linear trends. In the case of wind speeds over time, we are assuming that there will be a linear trend over time that can be tracked and eventually predicted utilizing the data that we currently have. In our code, we have a WindSpeedPredictor class that inherits from the BaseEstimator and RegressorMixin classes provided by scikit-learn. The regression itself is performed by LinearRegression(). Essentially, the algorithm fits a line to the data that minimizes the sum of squared differences between the observed data points and the predicted values. The fit method trains the model using our pre-existing data that is loaded by data_processor.py, and then the predict method is actually used to predict the future wind speeds. Our findings showed that although a linear model provides a general trend line, the raw data contains significant fluctuations, with sharp dips and spikes in wind speed over time. This is expected with hurricane data, which is inherently volatile and influenced by numerous environmental factors. Our

predicted data fifteen years into the future is also consistent with this, fluctuating in a similar manner from the data points across 1990-2020.

Our second algorithm is a clustering algorithm that maps out the hurricane wind speed impact on economic damage across different severity clusters. The KMeans algorithm works by partitioning the data into a predefined number of clusters based on the similarities between data points. In our case, the primary features used for clustering include wind speeds, start year, and economic damage (adjusted for inflation). These features are used to model the severity of the hurricanes and their impact, with the idea being that hurricanes with higher wind speeds are likely to cause greater economic damage. Once the model is defined, we use the fit method which receives the feature data (X) including wind speeds, start year, and economic damage. After that, the predict method is then used to assign our data to one of the two clusters. In practice, most of our data points are assigned to a single cluster, which suggests that the majority of the hurricanes in our dataset share similar characteristics regarding their wind speed and economic consequences. We do also observe a significant number of hurricanes that fall into a second cluster. This is likely because the hurricanes that fall into this second cluster are each significant outliers in comparison to the rest of the data set that we used for this project. The data also displays a linear trend in which, generally, higher wind speeds result in higher economic damage. This is consistent with what we believed going into the project.

Lastly, when detecting anomalies within the hurricane data, we utilized a class called Local Outlier Factor (LOF). LOF is an anomaly detection method that targets outliers using density. If a data point has a significantly lower density than its neighbors, it is considered an anomaly or outlier. In the context of hurricane data, anomalies could be represented as unusually rare or catastrophic events. Unlike different methods of detecting anomalies, we thought that this one in particular would be better for our project because it accounts for the local density closest to its neighbors. This matters because when it comes to hurricane data, certain data points might only be outliers in specific contexts related to time or environment. As stated before, our measure for anomalies includes storms that have high wind speeds and low damages as well as storms that have low wind speeds and high damages in comparison to their neighbors. In our HurricaneAnomalyDetector class, we have a contamination parameter which is set to 0.1. This means that the model assumes 10% of the data points could be an anomaly. We decided to set it to this because we thought it was an appropriate number to set the threshold for what exactly constitutes an outlier. Much like the other two algorithms, the fit and predict methods work the same. The fit method takes in the feature data (wind speed and total damage) and fits the algorithm onto the data so that it can return a fitted model that can make predictions. The predict method then returns a list of values where -1 indicates an anomaly, and 1 indicates a normal data point. A point with a density significantly lower than its neighbors as set by our contamination value will return as -1, for example. In our anomaly graph, our results were around what we expected going into it. It appears as if around 10% of the indices are anomalies, which should be correct because of how we set our contamination value. Additionally, the model that we used appears to have processed the data correctly when looking over the graphs because each of the

indices that are flagged as an anomaly either have low damages and high wind speeds or low high speeds and high damages.