Nicholas Lisicin-Wilson

A01180339

Neural Network

Problem Definition

This report will examine the relationship between weather and crime rates in Vancouver. Warmer weather is often linked to increased crime rates, so this report will look for evidence of that relationship and see if crime can be accurately predicted using weather data.

Data Treatment

Data from two datasets were combined for this analysis: hourly Vancouver weather data including tables for temperature and weather type, and crime data listing crimes committed in Vancouver along with their times and details. Because the datasets were recorded over different time periods, any non-overlapping data was dropped.

Both datasets are measured by the hour but were condensed into daily values for this report. For temperature, this involved taking the average temperature; for weather, the most common weather type; and for crime, the total count of crimes that occurred.

Temperature was converted from Kelvin to Celsius for ease of reading. A back-shifted temperature column was added using the previous day's data.

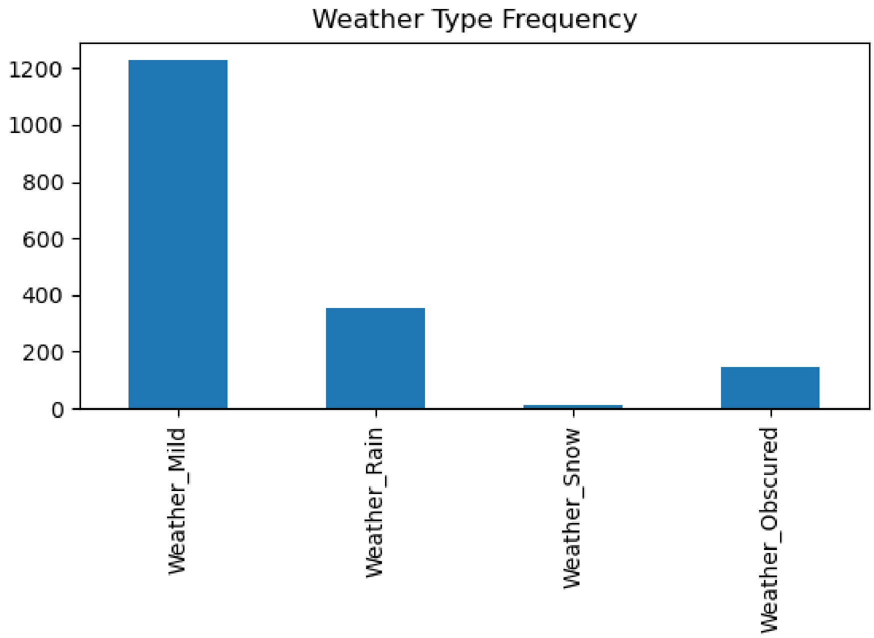
Weather was originally split into very narrow categories such as "light intensity drizzle," "light intensity drizzle rain," and "light intensity shower rain." These categories produced very small and ineffective results, so they were grouped into four broader categories: mild, rain, snow, and obscured (including dust, fog, haze, etc.). These named categories were then replaced with binary dummy variables for modelling purposes.

Exploratory Data Analysis

The following table describes the features used after data was treated:

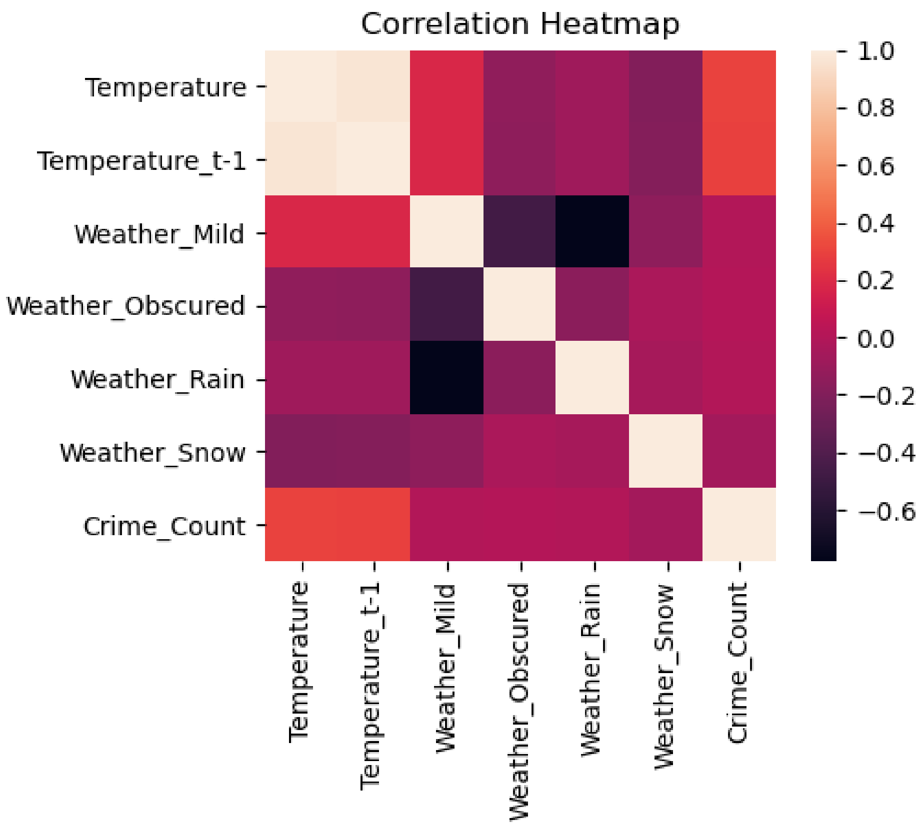
Feature	Description	Type
Crime_Count	Target variable; number of crimes that occurred in a day	Integer
Temperature	Average temperature of the day in °C	Float
Temperature_t-1	Average temperature of the previous day in °C	Float
Weather_Mild	Flag for mild weather such as clear skies and clouds	Binary integer
Weather_Rain	Flag for rainy weather such as rain and thunderstorms	Binary integer
Weather_Snow	Flag for snowy weather such as snow and sleet	Binary integer
Weather_Obscured	Flag for obscured weather such as dust and fog	Binary integer

The following bar chart shows the relative frequency of weather types:



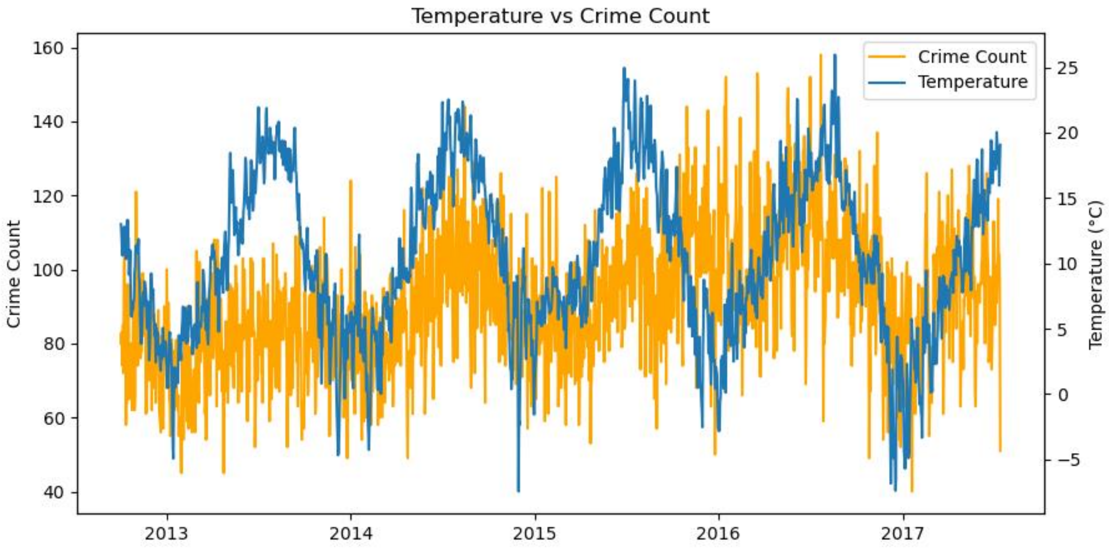
The weather frequency is likely not surprising to Vancouver residents; mild weather is by far the most common, followed by rain, and very little snow. These frequencies should be kept in mind when examining weather variables, especially snow, as the sample size is very small and it is not an effective predictor.

The following heatmap illustrates correlations between features. A correlation of 1 indicates perfect positive correlation (i.e. 1:1), a correlation of -1 indicates a perfect negative correlation (i.e. 1:-1):



Naturally, a day's temperature is very closely correlated to the temperature of the day before, and higher temperatures are more closely linked to mild weather than to snowy weather. Already we can see that temperature is positively correlated with crime count; weather does not appear to have any effect on crime, except for a slight negative correlation with snow. This relationship is likely not actually between crime and snow, but is rather due to the fact that snow is associated with colder weather, and colder weather is associated with lower crime.

The following plot illustrates the trends of temperature and crime count over time:



Though not always an exact fit, there is a clear relationship visible between the two variables. Both hit their highest and lowest values at roughly the same time, and crime count follows the same seasonal trend as temperature.

Model Development

Three models were developed: a simple linear regression model to provide a performance baseline for comparison, a neural network for more thorough predictions, and a stacked model to provide more consistent results.

Because all features are measured on very different scales, the data was scaled down to values between 0 and 1 for more effective modelling.

Evaluating the linear regression model showed that all independent variables aside from temperature were relatively insignificant. However, due to the low number of features available and the possibility for the neural networks to use them more effectively, all features were kept in the model.

Grid searching was used to determine the optimal neural network hyperparameters. Different combinations of epoch count, batch size, neuron count, layer count, kernel initializer, activation function, optimizer, and learning rate were all tested to find the set of parameters that produced the best results. To avoid outlier results, ten splits were run and their results were averaged.

The stacked model combines five individually trained neural networks with the same setup as the single network. Their predicted outputs are combined into a new set of inputs, which are then fed into another linear regression model to generate the final predicted values.

Model Evaluation

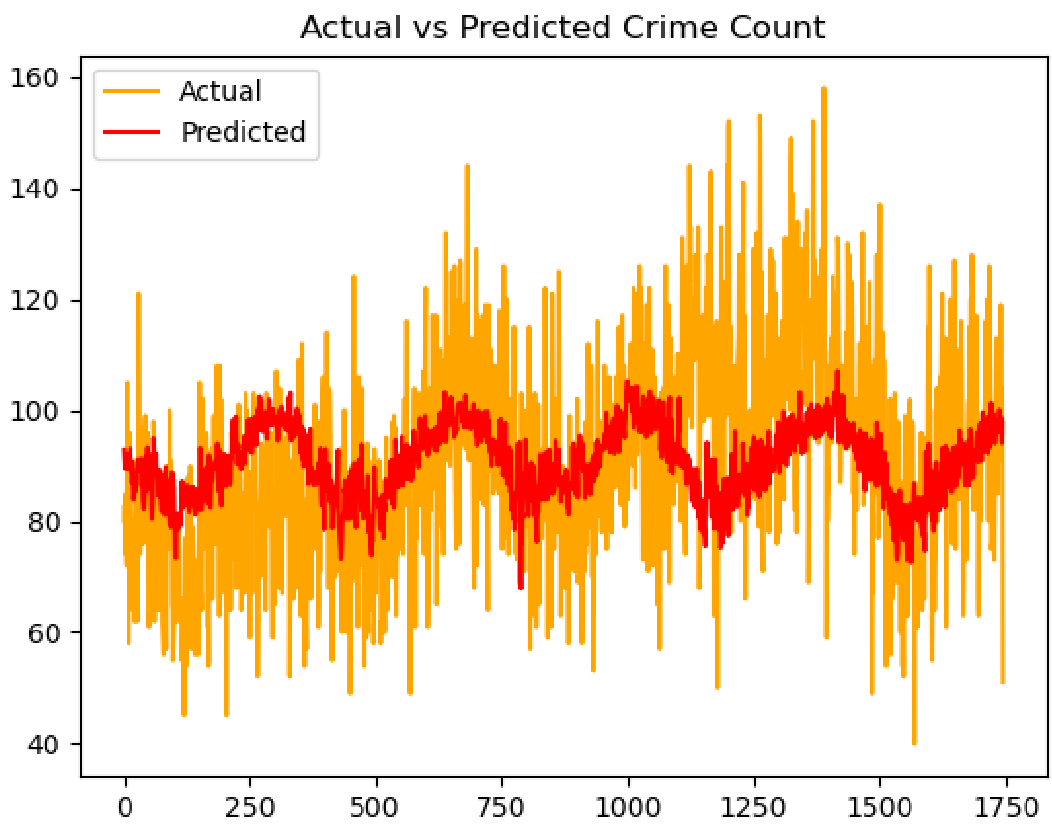
The following table shows the results of evaluating predictions for each model. Models are scored using root mean square error (RMSE), a measure of the distance between actual and predicted values where the lowest RMSE is preferable. Results were recorded for one run as well as an average across three runs, with the standard deviation (SD) indicating the degree of variance between runs:

Model	RMSE (1 run)	RMSE (3 runs)	SD
Linear Regression	0.14788	0.14525	0.00198
Neural Network	0.14858	0.14669	0.00223
Stacked	0.14659	0.14553	0.00187

Though all models produce very similar results, the stacked model emerges as the best, giving the second lowest RMSE across three runs with the least variance. The single neural network is noticeably less consistent than the stacked model despite its ten splits. For practical purposes, the linear regression model produces results on the same level as the stacked model, and without the training time. A larger or more complex dataset might better reveal the potential of the stacked model.

Back-Testing

To test the effectiveness of the model, all independent variables were shifted back a day then input into the stacked model to see how well it could recreate the actual crime count. This plot shows the results of back-testing:



The plot shows that the model produced good predictions using the previous day's features as inputs. Although it doesn't reach the highs and lows of the actual values, it clearly follows the same seasonal trend. Back-testing gave an unscaled RMSE of 17.297; quite accurate considering the scale of the crime data. All the results examined in this report suggest that weather data, namely temperature, can be used with a stacked model to produce accurate, reliable predictions of the number of crimes that will be committed in a day.

Code Appendix

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
from keras.layers import Dense
from keras.models import load_model, Sequential
from keras.optimizers import Adam
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler

# Data sourced from:
# https://www.kaggle.com/selfishgene/historical-hourly-weather-data
# https://www.kaggle.com/wosaku/crime-in-vancouver

ROOT = r"C:\Users\nlwil\Documents\My Documents\4949 - Big 
Data\Data\\"
TEMPERATURE_PATH = ROOT + "temperature.csv"
WEATHER_PATH = ROOT + "weather_description.csv"
CRIME_PATH = ROOT + "crime.csv"


def prepare_data(df_temperature, df_weather, df_crime):
    # Fix column names and types, drop missing rows
    df_temperature.index = pd.to_datetime(df_temperature.index)
    df_temperature = df_temperature.rename(
        columns={"Vancouver": "Temperature"})
    df_temperature = df_temperature.dropna()

    df_weather.index = pd.to_datetime(df_weather.index)
    df_weather = df_weather.rename(
        columns={"Vancouver": "Weather"})
    df_weather = df_weather.dropna()

    # Convert temperature to Celsius
    df_temperature = df_temperature.apply(lambda x: x - 273.15)

    # Group weather data into broader categories
    weather_feature_map = {
```

```
        "broken clouds": "Mild",
        "few clouds": "Mild",
        "overcast clouds": "Mild",
        "scattered clouds": "Mild",
        "sky is clear": "Mild",

        "drizzle": "Rain",
        "heavy intensity rain": "Rain",
        "heavy intensity shower rain": "Rain",
        "light intensity drizzle": "Rain",
        "light intensity drizzle rain": "Rain",
        "light intensity shower rain": "Rain",
        "light rain": "Rain",
        "moderate rain": "Rain",
        "proximity shower rain": "Rain",
        "proximity thunderstorm": "Rain",
        "ragged thunderstorm": "Rain",
        "shower rain": "Rain",
        "thunderstorm": "Rain",
        "thunderstorm with heavy rain": "Rain",
        "thunderstorm with light rain": "Rain",
        "thunderstorm with rain": "Rain",
        "very heavy rain": "Rain",

        "heavy shower snow": "Snow",
        "heavy snow": "Snow",
        "light rain and snow": "Snow",
        "light shower sleet": "Snow",
        "light shower snow": "Snow",
        "light snow": "Snow",
        "shower snow": "Snow",
        "sleet": "Snow",
        "snow": "Snow",

        "dust": "Obscured",
        "fog": "Obscured",
        "haze": "Obscured",
        "mist": "Obscured",
        "smoke": "Obscured",
        "volcanic ash": "Obscured"
    }
    df_weather["Weather"] =
df_weather["Weather"].map(weather_feature_map)

    # Build datetime column
    # noinspection PyTypeChecker
```

```
    df_crime["datetime"] = pd.to_datetime({
        "year": df_crime["YEAR"],
        "month": df_crime["MONTH"],
        "day": df_crime["DAY"]
    })
    df_crime.index = pd.to_datetime(df_crime["datetime"])

    # Average hourly data to daily
    df_temperature_daily =
df_temperature.groupby(pd.Grouper(freq="D")).mean()
    df_weather_daily = df_weather.groupby(
        pd.Grouper(freq="D")).agg(pd.Series.mode)

    # In case of multiple weather values, take the first
    df_weather_daily["Weather"] = df_weather_daily["Weather"].apply(
        lambda x: x[0] if isinstance(x, np.ndarray) else x)

    # Add back-shifted temperature column
    df_temperature_daily["Temperature_t-1"] = df_temperature_daily[
        "Temperature"].shift(periods=1)
    df_temperature_daily = df_temperature_daily.iloc[1:]

    # Get dummies
    df_weather_daily = pd.get_dummies(df_weather_daily,
columns=["Weather"])

    # Count crimes per day
    df_crime_daily = pd.DataFrame({
        "Crime_Count": df_crime.groupby(df_crime.index.date).size()})

df_temperature_daily.set_index(pd.to_datetime(df_temperature_daily.in
dex))

    # Get the largest date range contained in all three data sets
    start_date = max(df_temperature_daily.index.min(),
                     df_weather_daily.index.min(),
                     df_crime_daily.index.min())
    end_date = min(df_temperature_daily.index.max(),
                   df_weather_daily.index.max(),
                   df_crime_daily.index.max())
    print(f"\nRange: {start_date} to {end_date}\n")

    # Slice data
    df_temperature_daily = df_temperature_daily[start_date:end_date]
    df_weather_daily = df_weather_daily[start_date:end_date]
    df_crime_daily = df_crime_daily[start_date:end_date]
```

```
    # Concatenate dataframes
    df_combined = pd.concat([
        df_temperature_daily, df_weather_daily, df_crime_daily],
axis=1)

    return df_combined


# === Load and prepare data ===
print("\nPreparing data...")

pd.set_option("display.max_columns", None)
pd.set_option("display.width", 1000)

df_temperature = pd.read_csv(
    TEMPERATURE_PATH,
    index_col="datetime",
    parse_dates=["datetime"],
    usecols=["datetime", "Vancouver"])
df_weather = pd.read_csv(
    WEATHER_PATH,
    index_col="datetime",
    parse_dates=["datetime"],
    usecols=["datetime", "Vancouver"])
df_crime = pd.read_csv(CRIME_PATH)

FEATURES = [
    "Temperature",
    "Temperature_t-1",
    "Weather_Mild",
    "Weather_Rain",
    "Weather_Snow",
    "Weather_Obscured"
]

df_prepared = prepare_data(
    df_temperature, df_weather, df_crime)

print(df_prepared)
print(df_prepared.describe())

# Separate data
X = df_prepared.drop("Crime_Count", axis=1)
y = df_prepared["Crime_Count"]
y = np.asarray(y).reshape(-1, 1)
```

```python
scaler_X = MinMaxScaler().fit(X)
scaler_y = MinMaxScaler().fit(y)

# === Data Exploration ===
print("\nExploring data...")

# Heatmap
plt.subplots_adjust(left=0.3, bottom=0.3)
sns.heatmap(df_prepared.corr(), square=True)
plt.title("Correlation Heatmap")
plt.show()

# Bar plot
weather_features = FEATURES[2:]
plt.subplots_adjust(bottom=0.4)
df_prepared[weather_features].sum().plot(kind="bar")
plt.title("Weather Type Frequency")
plt.show()

# Temperature vs crime line plot
fig, l_axis = plt.subplots(figsize=(10, 5))
r_axis = l_axis.twinx()
l_axis.plot(df_prepared["Crime_Count"], color="orange", label="Crime
Count")
l_axis.set_ylabel("Crime Count")
r_axis.plot(df_prepared["Temperature"], label="Temperature")
r_axis.set_ylabel("Temperature (°C)")
l_lines, l_labels = l_axis.get_legend_handles_labels()
r_lines, r_labels = r_axis.get_legend_handles_labels()
l_axis.legend(l_lines + r_lines, l_labels + r_labels, loc=0)
plt.title("Temperature vs Crime Count")
plt.show()

# === Linear Regression ===
print("\nPerforming linear regression...")


def get_averages(*args):
    averages = []
    for arg in args:
        averages.append(np.mean(arg))
    return averages


rs = []
```

```
r2s = []
mses = []
rmses = []
for _ in range(3):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.7)

    # Scale data
    X_train_scaled = scaler_X.transform(X_train)
    X_test_scaled = scaler_X.transform(X_test)
    y_train_scaled = scaler_y.transform(y_train)
    y_test_scaled = scaler_y.transform(y_test)

    # Fit model
    model = LinearRegression().fit(X_train_scaled, y_train_scaled)
    y_pred_scaled = model.predict(X_test_scaled)
    y_pred = scaler_y.inverse_transform(y_pred_scaled)

    # Evaluate model
    print(f"\nIntercept: {model.intercept_[0]}")
    print(f"Coefficients:")
    lowest_coef = 1
    lowest_feature = ""
    for feature, coef in zip(FEATURES, model.coef_[0]):
        print(f"\t{feature}: {coef}")
        if abs(coef) < lowest_coef:
            lowest_feature, lowest_coef = feature, abs(coef)
    print(f"Least significant feature:\n\t{lowest_feature}:
{lowest_coef}")
    r2 = model.score(X_test_scaled, y_test_scaled)
    r = np.sqrt(r2)
    mse = mean_squared_error(y_test_scaled, y_pred_scaled)
    rmse = np.sqrt(mse)
    print(f"R: {r}")
    print(f"R2: {r2}")
    print(f"MSE: {mse}")
    print(f"RMSE: {rmse}")
    rs.append(r)
    r2s.append(r2)
    mses.append(mse)
    rmses.append(rmse)

averages = get_averages(rs, r2s, mses, rmses)
print(f"\nAverage R: {averages[0]}")
print(f"Average R2: {averages[1]}")
print(f"Average MSE: {averages[2]}")
```

```
print(f"Average RMSE: {averages[3]}")

# === Neural Network ===
print("\nTraining neural network...")


def build_model():
    model = Sequential()
    model.add(Dense(5, input_dim=len(FEATURES),
              kernel_initializer="normal", activation="relu"))
    model.add(Dense(1, kernel_initializer="normal"))
    optimizer = Adam(lr=0.001)
    model.compile(loss="mean_squared_error", optimizer=optimizer)
    return model


def fit_and_predict_neural_network(X_train_scaled, X_test_scaled,
                                   y_train_scaled, y_test_scaled):
    # Build baseline model
    estimator = KerasRegressor(build_fn=build_model, epochs=10,
                               batch_size=20, verbose=0)
    kfold = KFold(n_splits=10)
    results = cross_val_score(
        estimator, X_train_scaled, y_train_scaled, cv=kfold)
    mse = abs(results.mean())
    print(f"\nBaseline MSE: {mse}")
    print(f"Baseline RMSE: {np.sqrt(mse)}")

    # Fit model
    model = build_model()
    model.fit(X_train_scaled, y_train_scaled,
              epochs=10, batch_size=20, verbose=0,
              validation_data=(X_test_scaled, y_test_scaled))

    return model


# Prepare data
X_train, X_test, y_train, y_test = train_test_split(X, y,
train_size=0.7)

X_train_scaled = scaler_X.transform(X_train)
X_test_scaled = scaler_X.transform(X_test)
y_train_scaled = scaler_y.transform(y_train)
y_test_scaled = scaler_y.transform(y_test)
```

```
model = fit_and_predict_neural_network(X_train_scaled, X_test_scaled,
                                       y_train_scaled, y_test_scaled)
y_pred_scaled = model.predict(X_test_scaled)

# Evaluate model
mse = mean_squared_error(y_test_scaled, y_pred_scaled)
print(f"\nNeural Network MSE: {mse}")
print(f"Neural Network RMSE: {np.sqrt(mse)}")

# === Stacked Model ===
print("\nBuilding stacked model...")


def create_stacked_dataset(models, X):
    X_stacked = None
    for model in models:
        y_pred = model.predict(X, verbose=0)
        X_stacked = y_pred if X_stacked is None else np.dstack(
            (X_stacked, y_pred))
    X_stacked = X_stacked.reshape((X_stacked.shape[0],
                                   X_stacked.shape[1] *
X_stacked.shape[2]))
    return X_stacked


MODEL_FOLDER = "models"
N_MEMBERS = 5

# Build and save members
if not os.path.exists(MODEL_FOLDER):
    os.makedirs(MODEL_FOLDER)
for i in range(N_MEMBERS):
    model = fit_and_predict_neural_network(X_train_scaled,
X_test_scaled,
                                           y_train_scaled,
y_test_scaled)
    model_file_name = f"model_{i}.h5"
    model.save(f"{MODEL_FOLDER}/{model_file_name}")
    print(f"Saved {model_file_name}")

# Load members
print("")
models = []
for i in range(N_MEMBERS):
    model_file_name = f"model_{i}.h5"
    model = load_model(f"{MODEL_FOLDER}/{model_file_name}")
```

```
    models.append(model)
    print(f"Loaded {model_file_name}")

# Stack X and make predictions
X_stacked_scaled = create_stacked_dataset(models, X_test_scaled)
model = LinearRegression().fit(X_stacked_scaled, y_test_scaled)
y_pred_scaled = model.predict(X_stacked_scaled)

# Evaluate model
mse = mean_squared_error(y_test_scaled, y_pred_scaled)
print(f"\nNeural Network MSE: {mse}")
print(f"Neural Network RMSE: {np.sqrt(mse)}")

# === Back-Testing ===
print("\nBack-testing model...")

# Prepare data and make prediction
df_prepared_t1 = df_prepared.shift(periods=1).iloc[1:]
X_t1 = df_prepared_t1.drop("Crime_Count", axis=1)
X_t1_scaled = scaler_X.transform(X_t1)
X_t1_scaled_stacked = create_stacked_dataset(models, X_t1_scaled)
y_t1_pred_scaled = model.predict(X_t1_scaled_stacked)
y_pred_t1 = scaler_y.inverse_transform(y_t1_pred_scaled)

# Evaluate results
mse = mean_squared_error(y[1:], y_pred_t1)
print(f"\nBack-Shifted MSE: {mse}")
print(f"Back-Shifted RMSE: {np.sqrt(mse)}")

plt.plot(y, color="orange", label="Actual")
plt.plot(y_pred_t1, color="red", label="Predicted")
plt.legend(loc=0)
plt.title("Actual vs Predicted Crime Count")
plt.show()
```