Appendix 1
Source code for data processing

```python
# Library imports
import numpy as np
import pandas as pd

# Reading orginal dataset into excel file
original_df = pd.read_excel('Original-River-Data.xlsx', usecols='A:I', skiprows=1)
river_data = original_df.copy()

# Renaming Headers
new_columns = {'Unnamed: 0': 'Date'}
new_columns.update({col: f"{col} MDF (Cumecs)" for col in river_data.columns[1:5]})
new_columns.update({col: f"{col} DRT (mm)" for col in river_data.columns[5:]})

river_data.rename(
    columns=new_columns,
    inplace=True
)

# Converting non-numeric columns to numeric columns
river_data['Skip Bridge MDF (Cumecs)'] = pd.to_numeric(river_data['Skip Bridge MDF (Cumecs)'], errors='coerce')
river_data['Skelton MDF (Cumecs)'] = pd.to_numeric(river_data['Skelton MDF (Cumecs)'], errors='coerce')
river_data['East Cowton DRT (mm)'] = pd.to_numeric(river_data['East Cowton DRT (mm)'], errors='coerce')
river_data['Date'] = pd.to_datetime(river_data['Date'], errors='coerce')

# Dropping rows with at least 1 null value
flow_cols = list(river_data.columns[1:5])
rain_cols = list(river_data.columns[5:])

null_values = river_data.isna().any(axis=1)
river_data.dropna(how="any", inplace=True)

# Dropping rows with rainfall outliers
rainfall_outliers = river_data[(river_data[rain_cols] > 400).any(1)]
river_data.drop(rainfall_outliers.index, inplace=True)

# Dropping rows with river flow outliers
river_flow_outliers = river_data[(river_data[flow_cols] == 0).any(1)]
river_data.drop(river_flow_outliers.index, inplace=True)

# Exporting cleaned dataset to excel file
export_data = river_data.copy()
export_data["Date"] = export_data["Date"].astype("string")
export_data.to_excel('River-Data-Cleaned.xlsx')

# Lagging data
clean_df = pd.read_excel('River-Data-Cleaned.xlsx')
clean_df.drop(["Unnamed: 0"], axis=1, inplace=True)

lagged_df = pd.DataFrame()
lagged_df["Date"] = clean_df["Date"]
lagged_df[flow_cols[-1]] = clean_df[flow_cols[-1]]

## Lagging rainfall and flow columns by 1 to 3 days
for i in range(3):
    for col in flow_cols:
        col_name = col.replace("(Cumecs)", f"(t-{i+1})")
        lagged_df[col_name] = clean_df[col].shift(i+1)

for i in range(3):
    for col in rain_cols:
        col_name = col.replace("(mm)", f"(t-{i+1})")
        lagged_df[col_name] = clean_df[col].shift(i+1)

## Dropping rows with null values
lagged_df[lagged_df.isna().any(axis=1)]
lagged_df.dropna(how="any", inplace=True)
```

```
66
67  # Moving averages
68  moving_avg_df = pd.DataFrame()
69  moving_avg_df["Date"] = clean_df["Date"]
70  moving_avg_df[flow_cols[-1]] = clean_df[flow_cols[-1]]
71
72  ## Creating moving averages of between 3 and 7 days for each numerical column
73  for i in range(3, 8):
74      for col in flow_cols:
75          col_name = col.replace("(Cumecs)", f"(MA{i})")
76          moving_avg_df[col_name] = clean_df[col].rolling(i).mean()
77
78  for i in range(3, 8):
79      for col in rain_cols:
80          col_name = col.replace("(mm)", f"(MA{i})")
81          moving_avg_df[col_name] = clean_df[col].rolling(i).mean()
82
83  ## Dropping rows with null values
84  moving_avg_df[moving_avg_df.isna().any(axis=1)]
85  moving_avg_df.dropna(how="any", inplace=True)
86
87  # Lagged moving averages
88  lagged_ma_df = pd.DataFrame()
89  lagged_ma_df["Date"] = moving_avg_df["Date"]
90  lagged_ma_df[flow_cols[-1]] = moving_avg_df[flow_cols[-1]]
91
92  ## Lagging moving averages by 1 day
93  ## lagging them by more than 1 day results in much weaker correlations
94  mdf_cols = list(moving_avg_df.columns[2:22])
95  drt_cols = list(moving_avg_df.columns[22:])
96
97  for col in mdf_cols:
98      col_name = col + f" (t-1)"
99      lagged_ma_df[col_name] = moving_avg_df[col].shift(1)
100
101 for col in drt_cols:
102     col_name = col + f" (t-1)"
103     lagged_ma_df[col_name] = moving_avg_df[col].shift(1)
104
105 ## Dropping rows with null values
106 lagged_ma_df[lagged_ma_df.isna().any(axis=1)]
107 lagged_ma_df.dropna(how="any", inplace=True)
108
109 # Weighted moving averages
110 weighted_ma_df = pd.DataFrame()
111 weighted_ma_df["Date"] = clean_df["Date"]
112 weighted_ma_df[flow_cols[-1]] = clean_df[flow_cols[-1]]
113
114 ## Creating weighted moving averages of between 3 and 7 days
115 ## for each numerical column
116 for i in range(3, 8):
117     for col in flow_cols:
118         col_name = col.replace("(Cumecs)", f"(WMA{i})")
119         weighted_ma_df[col_name] = clean_df[col].ewm(span=i).mean()
120
121 for i in range(3, 8):
122     for col in rain_cols:
123         col_name = col.replace("(mm)", f"(WMA{i})")
124         weighted_ma_df[col_name] = clean_df[col].ewm(span=i).mean()
125
126 # Lagged weighted moving averages
127 lagged_wma_df = pd.DataFrame()
128 lagged_wma_df["Date"] = weighted_ma_df["Date"]
129 lagged_wma_df[flow_cols[-1]] = weighted_ma_df[flow_cols[-1]]
130
131 ## Lagging weighted moving averages
132
```

```
w_mdf_cols = list(weighted_ma_df.columns[2:22])
w_drt_cols = list(weighted_ma_df.columns[22:])

for col in w_mdf_cols:
    col_name = col + f" (t-1)"
    lagged_wma_df[col_name] = weighted_ma_df[col].shift(1)

for col in w_drt_cols:
    col_name = col + f" (t-1)"
    lagged_wma_df[col_name] = weighted_ma_df[col].shift(1)

## Dropping rows with null values
lagged_wma_df[lagged_wma_df.isna().any(1)]
lagged_wma_df.dropna(how="any", inplace=True)

# Exporting newly created datasets with lags and moving averages
lagged_df.to_excel('River-Data-Lagged.xlsx')
moving_avg_df.to_excel('River-Data-MA.xlsx')
lagged_ma_df.to_excel('River-Data-MA-Lagged.xlsx')
weighted_ma_df.to_excel('River-Data-WMA.xlsx')
lagged_wma_df.to_excel('River-Data-WMA-Lagged.xlsx')

# Utility Functions
## Functions for standardising and unstandardising values
def standardise_columns(df, cols):
    """
    This function works with dataframes to standardise values
    in multiple columns to the range [0.1, 0.9]
    """
    subset_df = df[cols]
    subset_df = 0.8 * ((subset_df - subset_df.min()) / (subset_df.max() - subset_df.min())) + 0.1
    return subset_df

def unstandardise_columns(df, cols, max_val, min_val):
    """
    This function works with numpy arrays to destandardise values
    in multiple columns
    """
    subset_df = df[cols]
    subset_df = ((subset_df - subset_df.min()) / 0.8) * (max_val - min_val) + min_val
    return subset_df

def standardise_value(x, max_val, min_val):
    """
    This function works with numpy arrays to standardise values
    in multiple arrays to the range [0.1, 0.9]
    """
    return 0.8 * ((x - min_val)) / (max_val - min_val) + 0.1

def unstandardise_value(x, max_val, min_val):
    """
    This function works with numpy arrays to destandardise values
    in multiple arrays
    """
    return ((x - 0.1) / 0.8) * (max_val - min_val) + min_val

# Function for building custom feature and target sets from larger datasets
def build_feature_set(*datasets):
    assert len(datasets) > 0, "No data sets entered"
    datasets = list(datasets)
    min_rows = min(d.shape[0] for d in datasets)

    for i, ds in enumerate(datasets):
        datasets[i] = ds.truncate(before=ds.shape[0]-min_rows).reset_index()
        datasets[i].drop(["index"], axis=1, inplace=True)

```

```
    merged_df = datasets[0].iloc[:, :2]
    for ds in datasets:
        merged_df = pd.concat([merged_df, ds.iloc[:, 2:]], axis=1)

    merged_cols = list(merged_df.columns)
    selected_cols = []

    for i in range(0, len(merged_cols), 2):
        format_str = f"{i+1}) {merged_cols[i]}"
        if i != len(merged_cols) - 1:
            second_part = f"{i+2}) {merged_cols[i+1]}"
            num_spaces = 50 - len(format_str)
            format_str += num_spaces*" " + second_part
        print(format_str)

    selected_indices = input("\nSelect columns: ")
    for index in selected_indices.split(","):
        if "-" in index:
            first_i, second_i = index.split("-")
            selected_cols += merged_cols[int(first_i) - 1: int(second_i)]
        else:
            selected_cols.append(merged_cols[int(index) - 1])

    return merged_df[selected_cols]
```