In [1]:
```
import pandas as pd
from datetime import datetime
import matplotlib.pyplot as plt
```

In [2]:
```
file_path = r'D:\Industrial immersion\Zelenograd_new.xlsx'
sheet_name = 'Zelenograd_2 precipitation'

df = pd.read_excel(file_path, sheet_name=sheet_name)
```

In [3]:
```
print(df.head())
```

```
   Число  Месяц Месяц-текст   Год      Время            Осадки  Код осадка  \
0      2      4        ММММ  2021  16:10:00  Дождь умеренный           7
1      2      4        ММММ  2021  16:20:00  Дождь умеренный           7
2      2      4        ММММ  2021  16:25:00  Дождь умеренный           7
3      2      4        ММММ  2021  16:30:00  Дождь умеренный           7
4      2      4        ММММ  2021  16:40:00               Нет           0

                 Дата  Амп. (A)  Амп. (B)  ...  Влажность  Темп.возд.  \
0 2021-04-02 16:10:00      0.05      0.03  ...      58.03        5.29
1 2021-04-02 16:20:00      0.09      0.08  ...      57.25        5.29
2 2021-04-02 16:25:00      0.09      0.08  ...      58.43        5.29
3 2021-04-02 16:30:00      0.09      0.08  ...      65.88        5.29
4 2021-04-02 16:40:00      0.09      0.08  ...      66.66        5.29

   Ветер напр.  Ветер скорость  СП (V)    (V)   (V).1  (V).2  \
0         93.0               0   20.52  12.79    4.15   4.11
1         74.0               0   20.48  12.79    4.12   4.12
2         86.0            0.02   20.84  12.81    4.12   4.13
3        112.0               0   20.91  12.82    4.12   4.12
4         76.0               0   21.04  12.86    4.12   4.12

                Unnamed: 22  Unnamed: 23
0                       NaN          NaN
1  Общий график весь период          NaN
2                       NaN          NaN
3                       NaN          NaN
4                       NaN          NaN

[5 rows x 24 columns]
```

In [4]:
```
# Convert columns to numeric data type
columns_to_convert = ['Влажность', 'Темп.возд.', 'Ветер напр.', 'Ветер скорость']
df[columns_to_convert] = df[columns_to_convert].apply(pd.to_numeric, errors='coerce')

# Verify the data types after conversion
print(df.dtypes)
```

```
Число                          int64
Месяц                          int64
Месяц-текст                   object
Год                            int64
Время                         object
Осадки                        object
Код осадка                     int64
Дата                  datetime64[ns]
Амп. (A)                     float64
Амп. (B)                     float64
Амп. (C)                     float64
Пики (A)                     float64
Пики. (B)                    float64
Пики (C)                     float64
Влажность                    float64
Темп.возд.                   float64
Ветер напр.                  float64
Ветер скорость               float64
СП (V)                       float64
 (V)                         float64
 (V).1                        object
 (V).2                       float64
Unnamed: 22                   object
Unnamed: 23                   object
dtype: object
```

In [5]:
```
# Define the new column names
new_column_names = {
    'Дата': 'Date',
    'Амп. (A)': 'Amplitude A',
    'Амп. (B)': 'Amplitude B',
    'Амп. (C)': 'Amplitude C',
    'Пики (A)': 'Peaks A',
    'Пики. (B)': 'Peaks B',
    'Пики (C)': 'Peaks C',
    'Влажность': 'Humidity',
    'Темп.возд.': 'Air Temperature',
    'Ветер напр.': 'Wind Direction',
    'Ветер скорость': 'Wind Speed',
```

```
    'V(СП),': 'SP (V)',
    '(V)': 'Voltage A',
    '(V).1': 'Voltage B',
    '(V).2': 'Voltage C'
}

# Rename the columns
df = df.rename(columns=new_column_names)

# Print the updated column names
print(df.columns)
```

```
Index(['Число', 'Месяц', 'Месяц-текст', 'Год', 'Время', 'Осадки', 'Код осадка',
       'Date', 'Amplitude A', 'Amplitude B', 'Amplitude C', 'Peaks A',
       'Peaks B', 'Peaks C', 'Humidity', 'Air Temperature', 'Wind Direction',
       'Wind Speed', 'СП (V)', ' (V)', ' (V).1', ' (V).2', 'Unnamed: 22',
       'Unnamed: 23'],
      dtype='object')
```

In [6]:

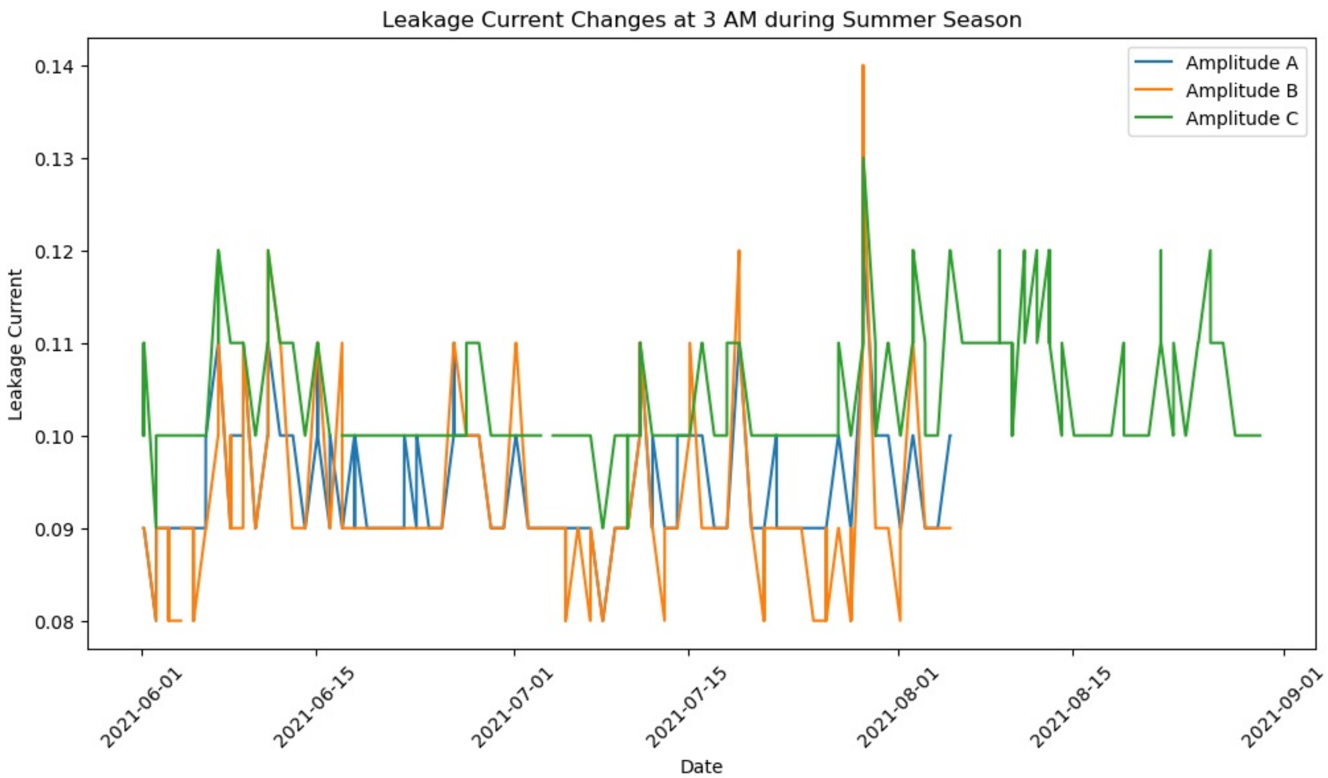
```
columns_to_drop = ['Число', 'Месяц', 'Месяц-текст', 'Год', 'Время','Unnamed: 22', 'Unnamed: 23']
df = df.drop(columns=columns_to_drop)
```

In [7]:

```
df['Date'] = pd.to_datetime(df['Date'])

summer_start = datetime.strptime('01/06/2021', '%d/%m/%Y')
summer_end = datetime.strptime('31/08/2021', '%d/%m/%Y')

summer_mask = (df['Date'] >= summer_start) & (df['Date'] <= summer_end)
summer_data = df[summer_mask]
```

In [8]:

```
# Filter the summer_data DataFrame for 3 AM readings
specific_hour = 3
summer_3am_data = summer_data[summer_data['Date'].dt.hour == specific_hour]

# Plot the changes in leakage current (Amplitude A, Amplitude B, Amplitude C) at 3 AM throughout the summer sea
plt.figure(figsize=(12, 6))
plt.plot(summer_3am_data['Date'], summer_3am_data['Amplitude A'], label='Amplitude A')
plt.plot(summer_3am_data['Date'], summer_3am_data['Amplitude B'], label='Amplitude B')
plt.plot(summer_3am_data['Date'], summer_3am_data['Amplitude C'], label='Amplitude C')
plt.xlabel('Date')
plt.ylabel('Leakage Current')
plt.title('Leakage Current Changes at 3 AM during Summer Season')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



In [9]:

```
summer_data
```

Out[9]:

	Осадки	Код осадка	Date	Amplitude A	Amplitude B	Amplitude C	Peaks A	Peaks B	Peaks C	Humidity	Air Temperature	Wind Direction	Wind Speed	СП (V
9396	Нет	0	2021-06-01 00:00:00	0.09	0.08	0.10	0.16	0.10	0.22	86.66	5.94	96.0	0.00	12.79
9397	Нет	0	2021-06-01 00:10:00	0.09	0.08	0.10	0.13	0.10	0.16	86.66	5.94	96.0	0.00	12.79
9398	Нет	0	2021-06-01 00:20:00	0.09	0.08	0.10	0.14	0.15	0.18	86.66	5.94	96.0	0.00	12.79
9399	Нет	0	2021-06-01 00:25:00	0.09	0.09	0.10	0.13	0.12	0.14	89.80	5.29	96.0	0.00	12.77
9400	Нет	0	2021-06-01 00:35:00	0.09	0.09	0.10	0.11	0.12	0.16	90.19	4.64	96.0	0.00	12.77
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21945	Дождь умеренный	7	2021-08-30 23:25:00	NaN	NaN	0.11	NaN	NaN	0.08	NaN	NaN	7.0	0.00	12.68
21946	Дождь умеренный	7	2021-08-30 23:30:00	NaN	NaN	0.11	NaN	NaN	0.09	NaN	NaN	7.0	0.00	12.68
21947	Дождь умеренный	7	2021-08-30 23:40:00	NaN	NaN	0.11	NaN	NaN	0.07	NaN	NaN	33.0	0.00	12.67
21948	Дождь умеренный	7	2021-08-30 23:50:00	NaN	NaN	0.11	NaN	NaN	0.08	NaN	NaN	7.0	0.02	12.67
21949	Дождь умеренный	7	2021-08-31 00:00:00	NaN	NaN	0.11	NaN	NaN	0.08	NaN	NaN	7.0	0.00	12.67

12554 rows × 17 columns

In [10]:

```
summer_data.isnull().sum()
```

Out[10]:

```
Осадки               0
Код осадка           0
Date                 0
Amplitude A       3298
Amplitude B       3287
Amplitude C        166
Peaks A           3262
Peaks B           3263
Peaks C            113
Humidity          1962
Air Temperature   1337
Wind Direction       0
Wind Speed           0
СП (V)               0
 (V)                 0
 (V).1               0
 (V).2               0
dtype: int64
```

In [11]:

```
summer_data.notnull().sum()
```

Out[11]:

```
Осадки             12554
Код осадка         12554
Date               12554
Amplitude A         9256
Amplitude B         9267
Amplitude C        12388
Peaks A             9292
Peaks B             9291
Peaks C            12441
Humidity           10592
Air Temperature    11217
Wind Direction     12554
Wind Speed         12554
СП (V)             12554
 (V)               12554
 (V).1             12554
 (V).2             12554
dtype: int64
```

In [12]:

```
# Drop rows with missing data for specific columns
columns_to_check = ['Amplitude A', 'Amplitude B', 'Amplitude C', 'Peaks A', 'Peaks B', 'Peaks C', 'Humidity', '
summer_data = summer_data.dropna(subset=columns_to_check)
```

In [13]:

```
summer_data
```

In [13]:
```
summer_data
```

Out[13]:

	Осадки	Код осадка	Date	Amplitude A	Amplitude B	Amplitude C	Peaks A	Peaks B	Peaks C	Humidity	Air Temperature	Wind Direction	Wind Speed	СП (V)
9396	Нет	0	2021-06-01 00:00:00	0.09	0.08	0.1	0.16	0.10	0.22	86.66	5.94	96.0	0.0	12.79
9397	Нет	0	2021-06-01 00:10:00	0.09	0.08	0.1	0.13	0.10	0.16	86.66	5.94	96.0	0.0	12.79
9398	Нет	0	2021-06-01 00:20:00	0.09	0.08	0.1	0.14	0.15	0.18	86.66	5.94	96.0	0.0	12.79
9399	Нет	0	2021-06-01 00:25:00	0.09	0.09	0.1	0.13	0.12	0.14	89.80	5.29	96.0	0.0	12.77
9400	Нет	0	2021-06-01 00:35:00	0.09	0.09	0.1	0.11	0.12	0.16	90.19	4.64	96.0	0.0	12.77
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18794	Нет	0	2021-08-05 09:25:00	0.08	0.09	0.1	6.08	18.24	3.90	23.00	27.11	201.0	0.0	20.88
18796	Нет	0	2021-08-05 09:45:00	0.08	0.08	0.1	10.06	19.97	2.65	23.00	27.11	201.0	0.0	20.88
18797	Нет	0	2021-08-05 09:55:00	0.08	0.09	0.1	6.08	1.67	6.08	23.00	27.11	201.0	0.0	20.88
18798	Нет	0	2021-08-05 10:05:00	0.09	0.09	0.1	0.22	1.65	0.40	23.00	27.11	201.0	0.0	20.88
18800	Нет	0	2021-08-05 10:25:00	0.08	0.08	0.1	0.07	99.84	0.07	20.00	28.23	91.0	0.0	20.93

7167 rows × 17 columns

In [14]:
```
summer_data.tail(5)
```

Out[14]:

	Осадки	Код осадка	Date	Amplitude A	Amplitude B	Amplitude C	Peaks A	Peaks B	Peaks C	Humidity	Air Temperature	Wind Direction	Wind Speed	СП (V)
18794	Нет	0	2021-08-05 09:25:00	0.08	0.09	0.1	6.08	18.24	3.90	23.0	27.11	201.0	0.0	20.88
18796	Нет	0	2021-08-05 09:45:00	0.08	0.08	0.1	10.06	19.97	2.65	23.0	27.11	201.0	0.0	20.88
18797	Нет	0	2021-08-05 09:55:00	0.08	0.09	0.1	6.08	1.67	6.08	23.0	27.11	201.0	0.0	20.88
18798	Нет	0	2021-08-05 10:05:00	0.09	0.09	0.1	0.22	1.65	0.40	23.0	27.11	201.0	0.0	20.88
18800	Нет	0	2021-08-05 10:25:00	0.08	0.08	0.1	0.07	99.84	0.07	20.0	28.23	91.0	0.0	20.93

In [15]:
```
summer_data.isnull().sum()
```

Out[15]:
```
Осадки             0
Код осадка         0
Date               0
Amplitude A        0
Amplitude B        0
Amplitude C        0
Peaks A            0
Peaks B            0
Peaks C            0
Humidity           0
Air Temperature    0
Wind Direction     0
Wind Speed         0
СП (V)             0
 (V)               0
 (V).1             0
 (V).2             0
dtype: int64
```

In [16]:
```
print(summer_data.columns)
```

```
In [16]: print(summer_data.columns)
```

```
Index(['Осадки', 'Код осадка', 'Date', 'Amplitude A', 'Amplitude B',
       'Amplitude C', 'Peaks A', 'Peaks B', 'Peaks C', 'Humidity',
       'Air Temperature', 'Wind Direction', 'Wind Speed', 'СП (V)', ' (V)',
       ' (V).1', ' (V).2'],
      dtype='object')
```

```
In [17]: summer_data.columns = summer_data.columns.str.strip()
summer_data.drop(['СП (V)', '(V)','(V).1', '(V).2','Peaks A','Peaks B','Peaks C'], axis=1, inplace=True)
```

```
C:\Users\shere\AppData\Local\Temp\ipykernel_5428\4186312094.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  summer_data.drop(['СП (V)', '(V)','(V).1', '(V).2','Peaks A','Peaks B','Peaks C'], axis=1, inplace=True)
```

```
In [18]: print(summer_data.tail())
```

```
      Осадки  Код осадка                Date  Amplitude A  Amplitude B  \
18794    Нет           0 2021-08-05 09:25:00         0.08         0.09
18796    Нет           0 2021-08-05 09:45:00         0.08         0.08
18797    Нет           0 2021-08-05 09:55:00         0.08         0.09
18798    Нет           0 2021-08-05 10:05:00         0.09         0.09
18800    Нет           0 2021-08-05 10:25:00         0.08         0.08

       Amplitude C  Humidity  Air Temperature  Wind Direction  Wind Speed
18794          0.1      23.0            27.11           201.0         0.0
18796          0.1      23.0            27.11           201.0         0.0
18797          0.1      23.0            27.11           201.0         0.0
18798          0.1      23.0            27.11           201.0         0.0
18800          0.1      20.0            28.23            91.0         0.0
```

```
In [19]: summer_data['Wind Direction'] = pd.to_numeric(summer_data['Wind Direction'], errors='coerce')
```

```
C:\Users\shere\AppData\Local\Temp\ipykernel_5428\145390186.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  summer_data['Wind Direction'] = pd.to_numeric(summer_data['Wind Direction'], errors='coerce')
```

```
In [42]: # Rename the 'Осадки' column to 'Precipitation'
summer_data = summer_data.rename(columns={'Осадки': 'Precipitation'})
```

```
In [43]: # Define the mapping of old names to new names
name_mapping = {
    'Нет': 'No',
    'Снег слабый': 'The snow is weak',
    'Снег умеренный': 'Snow moderate',
    'Снег сильный': 'Snow heavy',
    'Снег с дождем': 'Snow with rain',
    'Морось': 'Drizzle',
    'Дождь слабый': 'The rain is weak',
    'Дождь умеренный': 'Rain is moderate',
    'Ливень слабый': 'The shower is weak',
    'Ливень умеренный': 'Shower moderate',
    'Ливень сильный': 'Heavy rain'
}

# Replace the old names with the new names
summer_data['Precipitation'] = summer_data['Precipitation'].replace(name_mapping)

# Print the updated DataFrame
print(summer_data)
```

```
      Precipitation  Код осадка                Date  Amplitude A  Amplitude B  \
9396             No           0 2021-06-01 00:00:00         0.09         0.08
9397             No           0 2021-06-01 00:10:00         0.09         0.08
9398             No           0 2021-06-01 00:20:00         0.09         0.08
9399             No           0 2021-06-01 00:25:00         0.09         0.09
9400             No           0 2021-06-01 00:35:00         0.09         0.09
...             ...         ...                 ...          ...          ...
18794            No           0 2021-08-05 09:25:00         0.08         0.09
18796            No           0 2021-08-05 09:45:00         0.08         0.08
18797            No           0 2021-08-05 09:55:00         0.08         0.09
18798            No           0 2021-08-05 10:05:00         0.09         0.09
18800            No           0 2021-08-05 10:25:00         0.08         0.08

       Amplitude C  Humidity  Air Temperature  Wind Direction  Wind Speed
9396           0.1     86.66             5.94            96.0         0.0
9397           0.1     86.66             5.94            96.0         0.0
9398           0.1     86.66             5.94            96.0         0.0
9399           0.1     89.80             5.29            96.0         0.0
9400           0.1     90.19             4.64            96.0         0.0
...            ...       ...              ...             ...         ...
18794          0.1     23.00            27.11           201.0         0.0
18796          0.1     23.00            27.11           201.0         0.0
18797          0.1     23.00            27.11           201.0         0.0
18798          0.1     23.00            27.11           201.0         0.0
18800          0.1     20.00            28.23            91.0         0.0

[7167 rows x 10 columns]
```

In [44]:
```python
summer_data.columns
```

Out[44]:
```
Index(['Precipitation', 'Код осадка', 'Date', 'Amplitude A', 'Amplitude B',
       'Amplitude C', 'Humidity', 'Air Temperature', 'Wind Direction',
       'Wind Speed'],
      dtype='object')
```

In [45]:
```python
# Create a correlation matrix
correlation_matrix = summer_data[['Amplitude A', 'Amplitude B', 'Amplitude C', 'Air Temperature', 'Wind Directi
                                  'Humidity', 'Precipitation']].corr()
```

```
C:\Users\shere\AppData\Local\Temp\ipykernel_5428\1146063866.py:2: FutureWarning: The default value of numeric_o
nly in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  correlation_matrix = summer_data[['Amplitude A', 'Amplitude B', 'Amplitude C', 'Air Temperature', 'Wind Direc
tion', 'Wind Speed',
```

In [46]:
```python
correlation_matrix
```

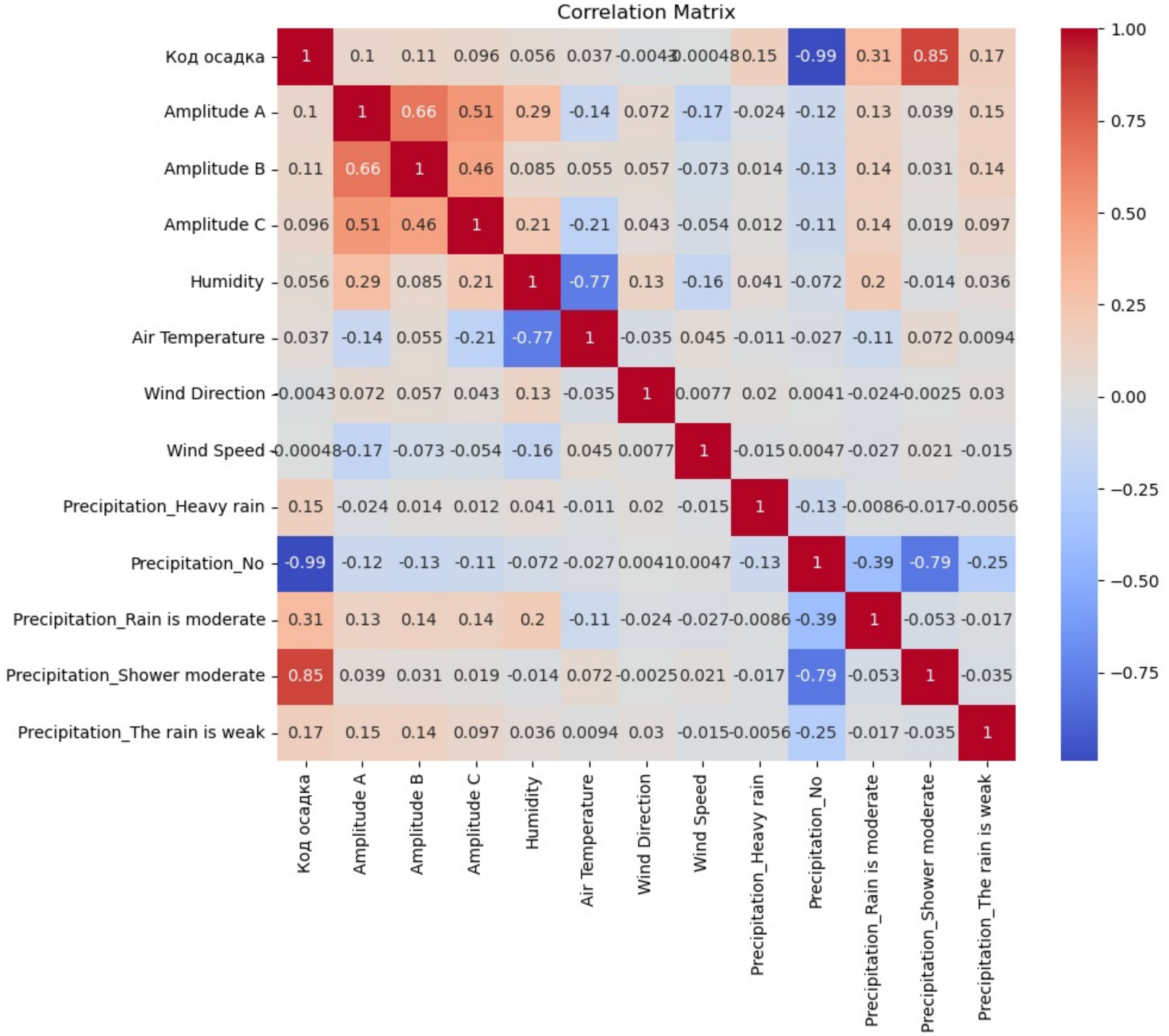
Out[46]:

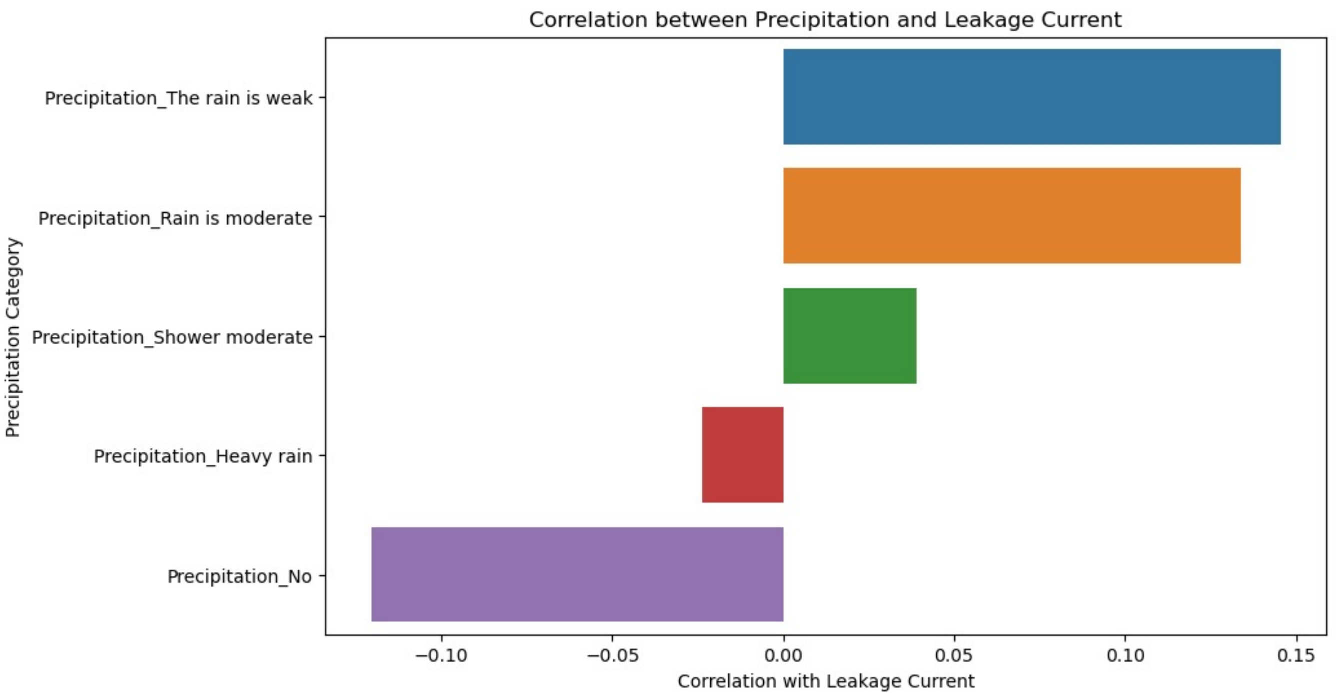
	Amplitude A	Amplitude B	Amplitude C	Air Temperature	Wind Direction	Wind Speed	Humidity
Amplitude A	1.000000	0.661013	0.509933	-0.136457	0.071932	-0.172129	0.290894
Amplitude B	0.661013	1.000000	0.457729	0.055081	0.056844	-0.073498	0.085337
Amplitude C	0.509933	0.457729	1.000000	-0.205052	0.043143	-0.054234	0.211083
Air Temperature	-0.136457	0.055081	-0.205052	1.000000	-0.035210	0.045006	-0.767505
Wind Direction	0.071932	0.056844	0.043143	-0.035210	1.000000	0.007711	0.128128
Wind Speed	-0.172129	-0.073498	-0.054234	0.045006	0.007711	1.000000	-0.158844
Humidity	0.290894	0.085337	0.211083	-0.767505	0.128128	-0.158844	1.000000

In [49]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assume you have a DataFrame named 'data_encoded' that includes the one-hot encoded 'Precipitation' columns an

# Calculate the correlation matrix
correlation_matrix = data_encoded.iloc[:, :-1].corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Extract the correlation values between precipitation and leakage current
precipitation_correlation = correlation_matrix['Amplitude A'][correlation_matrix.columns.str.startswith('Precip

# Sort the correlation values in descending order
precipitation_correlation = precipitation_correlation.sort_values(ascending=False)

# Plot the correlation values as a bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x=precipitation_correlation, y=precipitation_correlation.index)
plt.xlabel('Correlation with Leakage Current')
plt.ylabel('Precipitation Category')
plt.title('Correlation between Precipitation and Leakage Current')
```

```
plt.show()
```

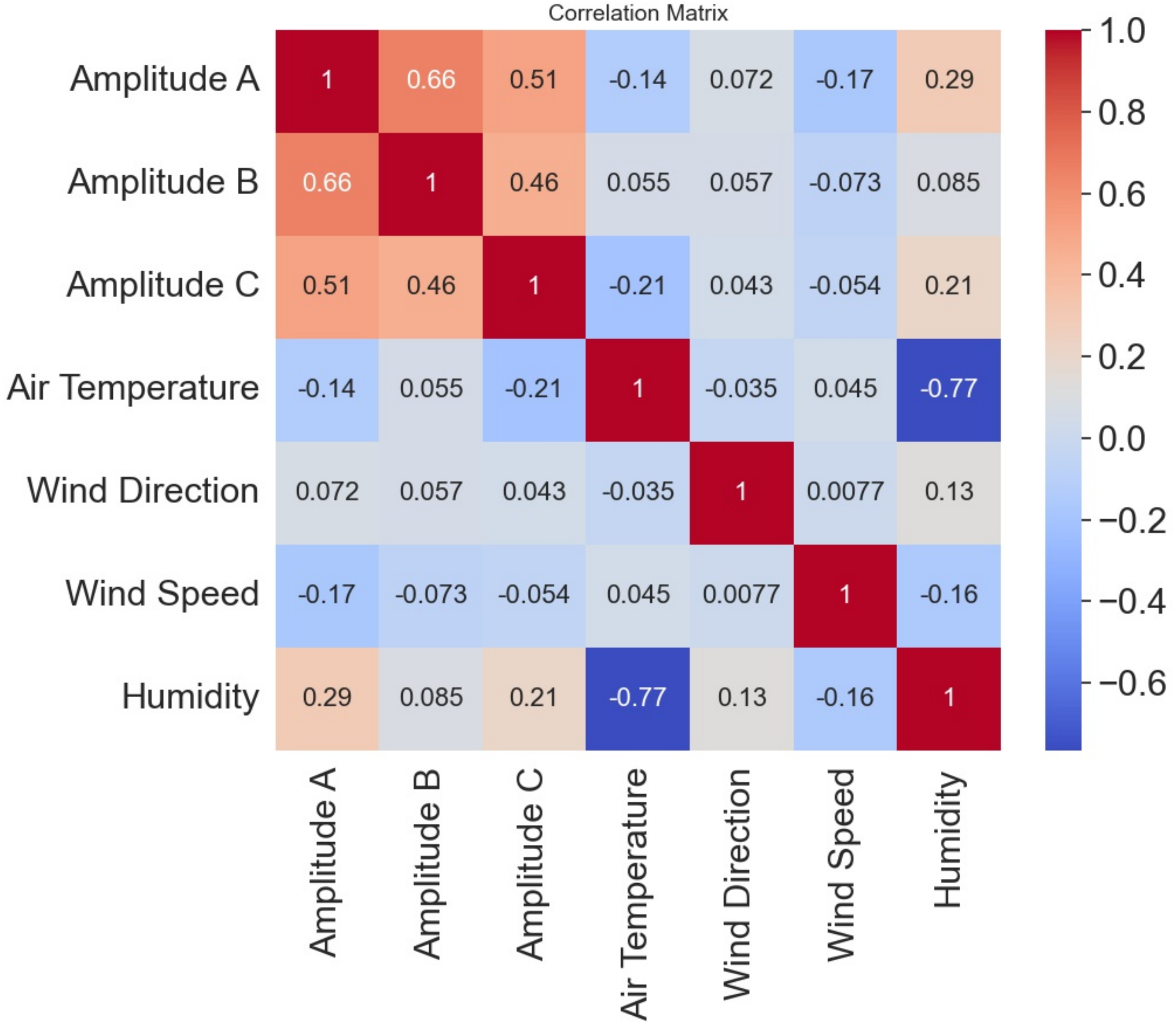
```
C:\Users\shere\AppData\Local\Temp\ipykernel_5428\2340072690.py:8: FutureWarning: The default value of numeric_o
nly in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns o
r specify the value of numeric_only to silence this warning.
  correlation_matrix = data_encoded.iloc[:, :-1].corr()
```





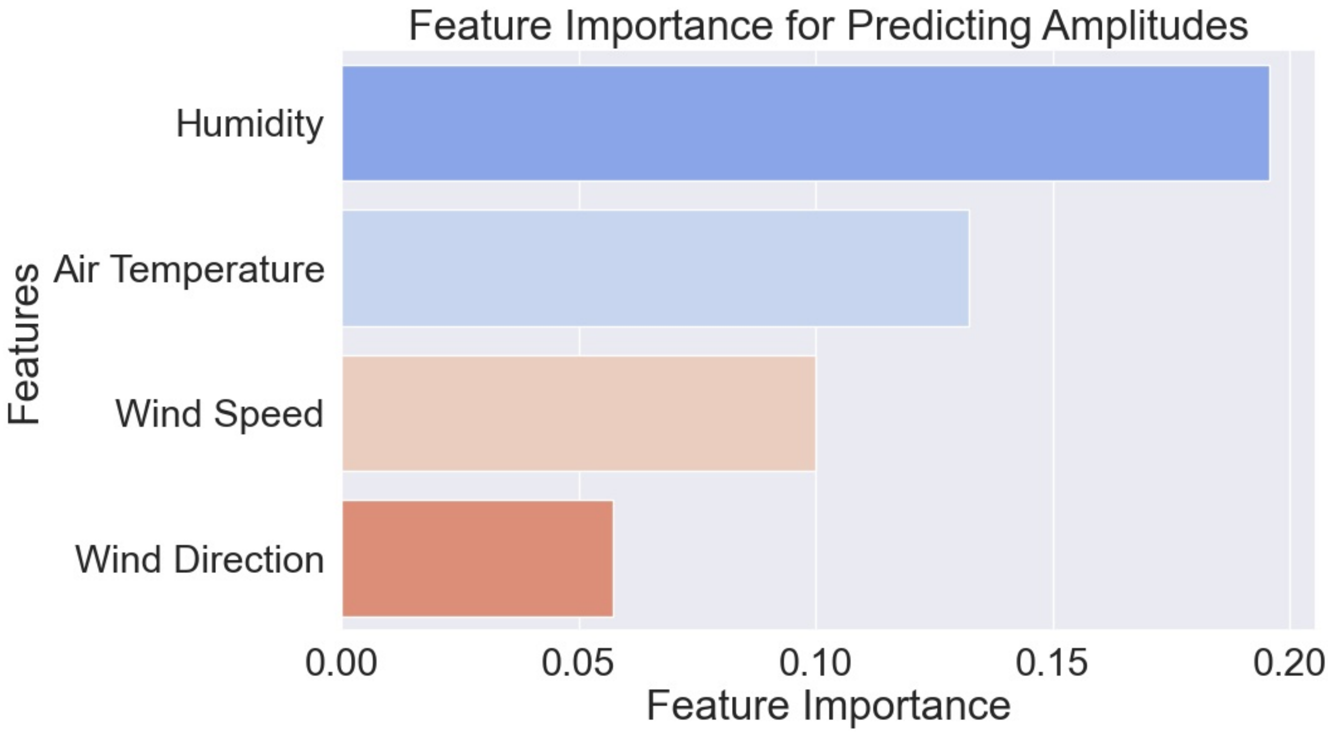
In [84]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Clean column names
# df_updated.columns = df_updated.columns.str.strip()

# Create a correlation matrix
correlation_matrix = summer_data[['Amplitude A', 'Amplitude B', 'Amplitude C', 'Air Temperature', 'Wind Directi

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', annot_kws={"fontsize": 16},
            xticklabels=correlation_matrix.columns, yticklabels=correlation_matrix.columns)
sns.set(font_scale=2)
plt.title('Correlation Matrix', fontsize=14)
plt.show()
```



In [83]:

```
# Calculate feature importances
feature_importances = correlation_matrix[['Amplitude A', 'Amplitude B', 'Amplitude C']].loc[['Air Temperature',
feature_importances = feature_importances.abs().mean(axis=1)

# Sort the feature importances in descending order
feature_importances = feature_importances.sort_values(ascending=False)

# Plot the feature importances as a bar chart
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances, y=feature_importances.index, palette='coolwarm')
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance for Predicting Amplitudes')
plt.show()
```



In [137…

```
import numpy as np

np.triu(np.ones_like(summer_data.corr()))
```

```
C:\Users\shere\AppData\Local\Temp\ipykernel_16724\2569380456.py:3: FutureWarning: The default value of numeric_
only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns
or specify the value of numeric_only to silence this warning.
  np.triu(np.ones_like(summer_data.corr()))
```

Out[137]:

```
array([[1., 1., 1., 1., 1., 1., 1., 1.],
       [0., 1., 1., 1., 1., 1., 1., 1.],
       [0., 0., 1., 1., 1., 1., 1., 1.],
       [0., 0., 0., 1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0., 1., 1., 1.],
       [0., 0., 0., 0., 0., 0., 1., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [51]:

```
summer_data
```

Out[51]:

	Precipitation	Код осадка	Date	Amplitude A	Amplitude B	Amplitude C	Humidity	Air Temperature	Wind Direction	Wind Speed
9396	No	0	2021-06-01 00:00:00	0.09	0.08	0.1	86.66	5.94	96.0	0.0
9397	No	0	2021-06-01 00:10:00	0.09	0.08	0.1	86.66	5.94	96.0	0.0
9398	No	0	2021-06-01 00:20:00	0.09	0.08	0.1	86.66	5.94	96.0	0.0
9399	No	0	2021-06-01 00:25:00	0.09	0.09	0.1	89.80	5.29	96.0	0.0
9400	No	0	2021-06-01 00:35:00	0.09	0.09	0.1	90.19	4.64	96.0	0.0
...	...	...	...	...	...	...	...	...	...	...
18794	No	0	2021-08-05 09:25:00	0.08	0.09	0.1	23.00	27.11	201.0	0.0
18796	No	0	2021-08-05 09:45:00	0.08	0.08	0.1	23.00	27.11	201.0	0.0
18797	No	0	2021-08-05 09:55:00	0.08	0.09	0.1	23.00	27.11	201.0	0.0
18798	No	0	2021-08-05 10:05:00	0.09	0.09	0.1	23.00	27.11	201.0	0.0
18800	No	0	2021-08-05 10:25:00	0.08	0.08	0.1	20.00	28.23	91.0	0.0

7167 rows × 10 columns

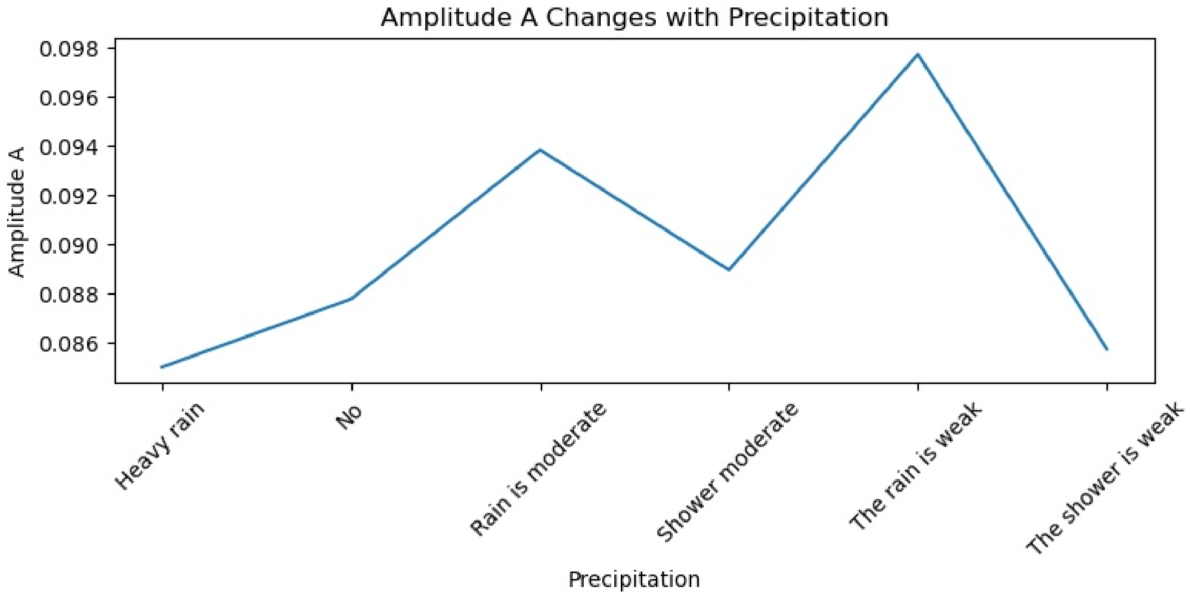
In [52]:

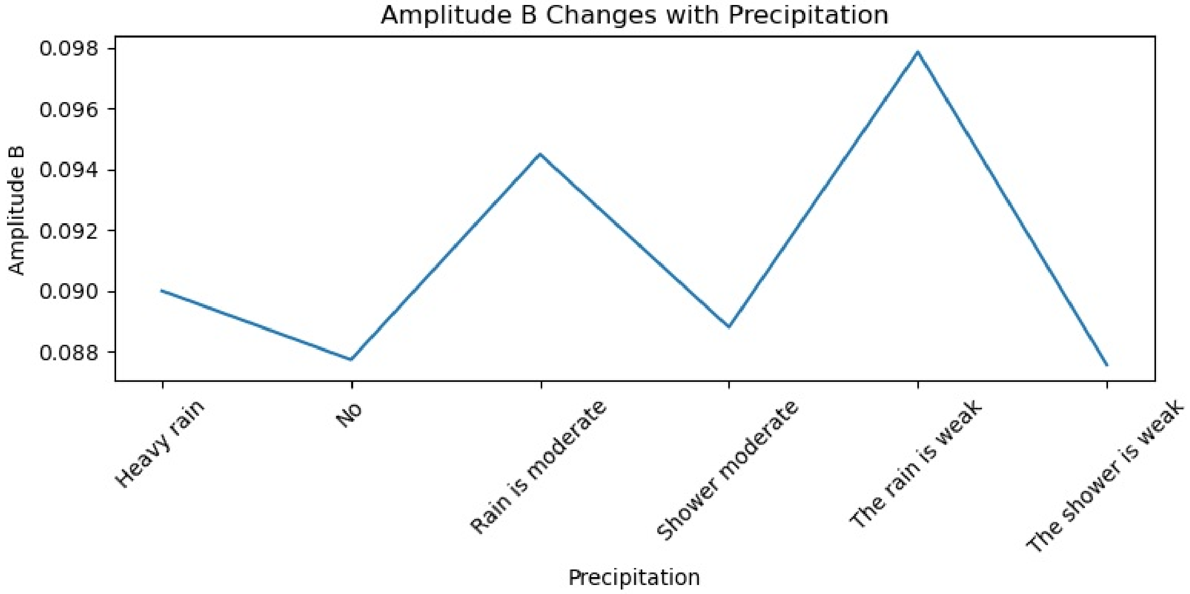
```
# Define the mapping dictionary
precipitation_mapping = {
    0: "No",
    1: "The snow is weak",
    2: "Snow moderate",
```

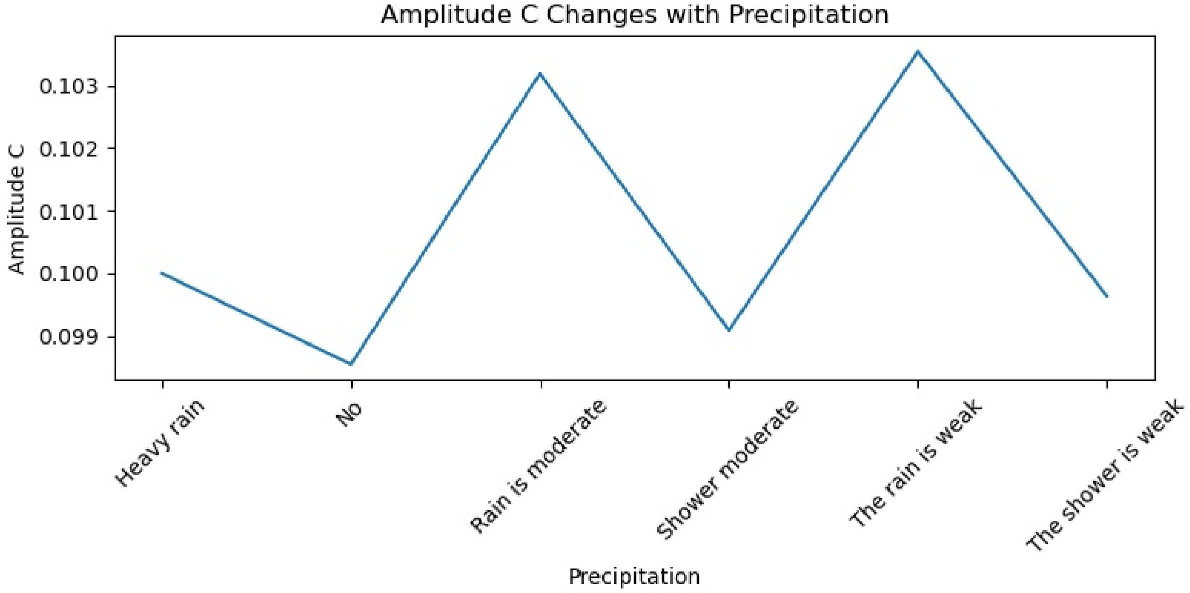
```python
    3: "Snow heavy",
    4: "Snow with rain",
    5: "Drizzle",
    6: "The rain is weak",
    7: "Rain is moderate",
    8: "The shower is weak",
    9: "Shower moderate",
    10: "Heavy rain"
}
# Create a copy of the DataFrame
summer_data_copy = summer_data.copy()

# Apply the mapping to the precipitation column using .loc on the copy
summer_data_copy.loc[:, 'Precipitation'] = summer_data_copy['Precipitation'].map(precipitation_mapping)
```

In [53]:

```python
import matplotlib.pyplot as plt

# Group the data by precipitation and calculate the mean leakage current for amplitude A, B, and C
grouped_data_A = summer_data.groupby('Precipitation')['Amplitude A'].mean()
grouped_data_B = summer_data.groupby('Precipitation')['Amplitude B'].mean()
grouped_data_C = summer_data.groupby('Precipitation')['Amplitude C'].mean()

# Create subplots for each amplitude
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(8, 12))

# Plot for amplitude A
ax1.plot(grouped_data_A.index, grouped_data_A.values)
ax1.set_xlabel('Precipitation')
ax1.set_ylabel('Amplitude A')
ax1.set_title('Amplitude A Changes with Precipitation')
ax1.tick_params(axis='x', rotation=45)

# Plot for amplitude B
ax2.plot(grouped_data_B.index, grouped_data_B.values)
ax2.set_xlabel('Precipitation')
ax2.set_ylabel('Amplitude B')
ax2.set_title('Amplitude B Changes with Precipitation')
ax2.tick_params(axis='x', rotation=45)

# Plot for amplitude C
ax3.plot(grouped_data_C.index, grouped_data_C.values)
ax3.set_xlabel('Precipitation')
ax3.set_ylabel('Amplitude C')
ax3.set_title('Amplitude C Changes with Precipitation')
ax3.tick_params(axis='x', rotation=45)

# Adjust spacing between subplots
plt.tight_layout()

# Display the plots
plt.show()
```







In [54]:

```
target_columns = ['Amplitude A','Amplitude B','Amplitude C']
columns_to_drop = target_columns + ['Date','Wind Direction','Precipitation']

# Create a DataFrame for the features by excluding the target columns
features = summer_data.drop(columns_to_drop, axis=1)


# Create separate DataFrames for each target variable
target_A = summer_data['Amplitude A']
target_B = summer_data['Amplitude B']
target_C = summer_data['Amplitude C']
```

In [55]:

```
from sklearn.model_selection import train_test_split
```

```
# Assuming you have already separated your features and targets

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, target_A, test_size=0.2, random_state=77)

# Print the indices of the train and test sets
print("Train set indices:", X_train.index)
print("Test set indices:", X_test.index)
```

```
Train set indices: Int64Index([16054, 16804,  9957, 17442, 16597, 15499, 17682, 15629,  9692,
            18651,
            ...
             9581, 14985, 16349, 10523, 11395, 12435, 12145, 15143, 10054,
            18464],
           dtype='int64', length=5733)
Test set indices: Int64Index([18388, 11820, 16194, 10576, 12338, 11171,  9779, 13715, 11904,
            12028,
            ...
            14248, 12416, 11380, 18625, 18605, 10890, 15516, 18493, 16095,
            17433],
           dtype='int64', length=1434)
```

In [56]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(5733, 4)
(1434, 4)
(5733,)
(1434,)
```

In [57]:

```
from sklearn.metrics import mean_absolute_error

# Calculate the baseline prediction for each target variable
baseline_prediction_A = target_A.mean()
baseline_prediction_B = target_B.mean()
baseline_prediction_C = target_C.mean()

# Create arrays of the same length as the respective target variables with the baseline predictions
baseline_predictions_A = [baseline_prediction_A] * len(target_A)
baseline_predictions_B = [baseline_prediction_B] * len(target_B)
baseline_predictions_C = [baseline_prediction_C] * len(target_C)

# Calculate the MAE for each target variable
baseline_mae_A = round(mean_absolute_error(target_A, baseline_predictions_A),5)
baseline_mae_B = round(mean_absolute_error(target_B, baseline_predictions_B),5)
baseline_mae_C = round(mean_absolute_error(target_C, baseline_predictions_C),5)

print("Baseline MAE for Amplitude A:", baseline_mae_A)
print("Baseline MAE for Amplitude B:", baseline_mae_B)
print("Baseline MAE for Amplitude C:", baseline_mae_C)
```

```
Baseline MAE for Amplitude A: 0.00495
Baseline MAE for Amplitude B: 0.00458
Baseline MAE for Amplitude C: 0.00317
```

In [59]:

```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_absolute_error

# Identify the target columns
target_columns = ['Amplitude A', 'Amplitude B', 'Amplitude C']
columns_to_drop = target_columns + ['Date', 'Wind Direction','Precipitation']

# Create a DataFrame for the features by excluding the target columns
features = summer_data.drop(columns_to_drop, axis=1)

# Create a Series for each target variable
target_A = summer_data['Amplitude A']
target_B = summer_data['Amplitude B']
target_C = summer_data['Amplitude C']

# Split the data into training and testing sets for each target variable
X_train_A, X_test_A, y_train_A, y_test_A = train_test_split(features, target_A, test_size=0.2, random_state=77)
X_train_B, X_test_B, y_train_B, y_test_B = train_test_split(features, target_B, test_size=0.2, random_state=77)
X_train_C, X_test_C, y_train_C, y_test_C = train_test_split(features, target_C, test_size=0.2, random_state=77)

# Apply the preprocessing steps to the numeric features
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

```
preprocessor = ColumnTransformer([
    ('numeric', numeric_transformer, features.select_dtypes(include=['float64', 'int64']).columns)
])

# Create a pipeline for the Linear Regression model for each target variable
model_lr_A = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

model_lr_B = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

model_lr_C = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Train the Linear Regression models for each target variable
model_lr_A.fit(X_train_A, y_train_A)
model_lr_B.fit(X_train_B, y_train_B)
model_lr_C.fit(X_train_C, y_train_C)
```

Out[59]:

```
Pipeline
preprocessor: ColumnTransformer
numeric
SimpleImputer
StandardScaler
LinearRegression
```

In [60]:

```
from sklearn.ensemble import RandomForestRegressor

# Create a Random Forest Regressor model for each target variable
model_rf_A = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=77))
])

model_rf_B = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=77))
])

model_rf_C = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=77))
])

# Train the Random Forest Regressor models for each target variable
model_rf_A.fit(X_train_A, y_train_A)
model_rf_B.fit(X_train_B, y_train_B)
model_rf_C.fit(X_train_C, y_train_C)
```

Out[60]:

```
Pipeline
preprocessor: ColumnTransformer
numeric
SimpleImputer
StandardScaler
RandomForestRegressor
```

In [61]:

```
X_train_A
```

Out[61]:

	Код осадка	Humidity	Air Temperature	Wind Speed
16054	0	72.00	18.10	0.00
16804	0	11.00	32.00	0.00
9957	0	37.25	21.47	0.30
17442	0	83.00	19.21	0.00
16597	0	56.00	20.32	0.00
...	...	...	...	...
12435	0	80.00	19.52	0.00
12145	0	68.23	18.23	0.20
15143	0	63.00	21.71	0.02
10054	0	49.01	18.23	0.00
18464	0	31.00	22.20	0.64

5733 rows × 4 columns

In [62]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# Normalize the data using MinMaxScaler
scaler = MinMaxScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)


# Normalize the data using MinMaxScaler
scaler = MinMaxScaler()
X_train_A_normalized = scaler.fit_transform(X_train_A)
X_test_A_normalized = scaler.transform(X_test_A)

X_train_B_normalized = scaler.fit_transform(X_train_B)
X_test_B_normalized = scaler.transform(X_test_B)

X_train_C_normalized = scaler.fit_transform(X_train_C)
X_test_C_normalized = scaler.transform(X_test_C)
```

In [63]:

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_absolute_error

# Create KNN regressor models
model_knn_A = KNeighborsRegressor()
model_knn_B = KNeighborsRegressor()
model_knn_C = KNeighborsRegressor()

# Create an imputer to fill missing values with the mean
imputer = SimpleImputer(strategy='mean')

# Impute missing values in the normalized training data
X_train_A_normalized_imputed = imputer.fit_transform(X_train_A_normalized)
X_train_B_normalized_imputed = imputer.fit_transform(X_train_B_normalized)
X_train_C_normalized_imputed = imputer.fit_transform(X_train_C_normalized)

# Train the KNN models on the imputed normalized features
model_knn_A.fit(X_train_A_normalized_imputed, y_train_A)
model_knn_B.fit(X_train_B_normalized_imputed, y_train_B)
model_knn_C.fit(X_train_C_normalized_imputed, y_train_C)

# Impute missing values in the normalized testing data
X_test_A_normalized_imputed = imputer.transform(X_test_A_normalized)
X_test_B_normalized_imputed = imputer.transform(X_test_B_normalized)
X_test_C_normalized_imputed = imputer.transform(X_test_C_normalized)

# Make predictions on the imputed normalized test features
y_pred_knn_A = model_knn_A.predict(X_test_A_normalized_imputed)
y_pred_knn_B = model_knn_B.predict(X_test_B_normalized_imputed)
y_pred_knn_C = model_knn_C.predict(X_test_C_normalized_imputed)

# Calculate the mean absolute error for each target variable
mae_knn_A = mean_absolute_error(y_test_A, y_pred_knn_A)
mae_knn_B = mean_absolute_error(y_test_B, y_pred_knn_B)
mae_knn_C = mean_absolute_error(y_test_C, y_pred_knn_C)

print("KNN Model Mean Absolute Error for Amplitude A:", mae_knn_A)
print("KNN Model Mean Absolute Error for Amplitude B:", mae_knn_B)
print("KNN Model Mean Absolute Error for Amplitude C:", mae_knn_C)
```

```
KNN Model Mean Absolute Error for Amplitude A: 0.0027294281729428166
KNN Model Mean Absolute Error for Amplitude B: 0.002595536959553695
KNN Model Mean Absolute Error for Amplitude C: 0.0022747559274755944
```

In [64]:

```
from sklearn.metrics import mean_absolute_error
```

```
In [64]: from sklearn.metrics import mean_absolute_error

# For Amplitude A
y_pred_knn_A = model_knn_A.predict(X_test_A_normalized_imputed)

non_zero_indices = y_test_A != 0
y_test_A_non_zero = y_test_A[non_zero_indices]
y_pred_knn_A_non_zero = y_pred_knn_A[non_zero_indices]

mae_knn_A = mean_absolute_error(y_test_A_non_zero, y_pred_knn_A_non_zero)
accuracy_knn_A = 100 - (mae_knn_A / y_test_A_non_zero.mean()) * 100

print("KNN Model Prediction Accuracy for Amplitude A:", round(accuracy_knn_A, 2), "%")

# For Amplitude B
y_pred_knn_B = model_knn_B.predict(X_test_B_normalized_imputed)

non_zero_indices = y_test_B != 0
y_test_B_non_zero = y_test_B[non_zero_indices]
y_pred_knn_B_non_zero = y_pred_knn_B[non_zero_indices]

mae_knn_B = mean_absolute_error(y_test_B_non_zero, y_pred_knn_B_non_zero)
accuracy_knn_B = 100 - (mae_knn_B / y_test_B_non_zero.mean()) * 100

print("KNN Model Prediction Accuracy for Amplitude B:", round(accuracy_knn_B, 2), "%")

# For Amplitude C
y_pred_knn_C = model_knn_C.predict(X_test_C_normalized_imputed)

non_zero_indices = y_test_C != 0
y_test_C_non_zero = y_test_C[non_zero_indices]
y_pred_knn_C_non_zero = y_pred_knn_C[non_zero_indices]

mae_knn_C = mean_absolute_error(y_test_C_non_zero, y_pred_knn_C_non_zero)
accuracy_knn_C = 100 - (mae_knn_C / y_test_C_non_zero.mean()) * 100

print("KNN Model Prediction Accuracy for Amplitude C:", round(accuracy_knn_C, 2), "%")
```

```
KNN Model Prediction Accuracy for Amplitude A: 96.9 %
KNN Model Prediction Accuracy for Amplitude B: 97.05 %
KNN Model Prediction Accuracy for Amplitude C: 97.7 %
```

```
In [65]: print('Linear Regression Training MAE:',round((mean_absolute_error(y_train_A, model_lr_A.predict(X_train_A))),5
print('Linear Regression Training MAE:',round((mean_absolute_error(y_train_B, model_lr_B.predict(X_train_B))),5
print('Linear Regression Training MAE:',round((mean_absolute_error(y_train_C, model_lr_C.predict(X_train_C))),5

print('Linear Regression Test MAE:',round((mean_absolute_error(y_test_A, model_lr_A.predict(X_test_A))),5))
print('Linear Regression Test MAE:',round((mean_absolute_error(y_test_B, model_lr_B.predict(X_test_B))),5))
print('Linear Regression Test MAE:',round((mean_absolute_error(y_test_C, model_lr_C.predict(X_test_C))),5))
```

```
Linear Regression Training MAE: 0.00465
Linear Regression Training MAE: 0.00442
Linear Regression Training MAE: 0.00316
Linear Regression Test MAE: 0.00456
Linear Regression Test MAE: 0.00428
Linear Regression Test MAE: 0.00324
```

```
In [66]: print('Random Forest Regressor Model Training MAE:',round((mean_absolute_error(y_train_A, model_rf_A.predict(X_
print('Random Forest Regressor Model Training MAE:',round((mean_absolute_error(y_train_B, model_rf_B.predict(X_
print('Random Forest Regressor Model Training MAE:',round((mean_absolute_error(y_train_C, model_rf_C.predict(X_

print('Random Forest Regressor Model Test MAE:',round((mean_absolute_error(y_test_A, model_rf_A.predict(X_test_
print('Random Forest Regressor Model Test MAE:',round((mean_absolute_error(y_test_B, model_rf_B.predict(X_test_
print('Random Forest Regressor Model Test MAE:',round((mean_absolute_error(y_test_C, model_rf_C.predict(X_test_
```

```
Random Forest Regressor Model Training MAE: 0.0014
Random Forest Regressor Model Training MAE: 0.00121
Random Forest Regressor Model Training MAE: 0.0011
Random Forest Regressor Model Test MAE: 0.00215
Random Forest Regressor Model Test MAE: 0.00199
Random Forest Regressor Model Test MAE: 0.00184
```

```
In [68]: import numpy as np
# For Amplitude A
y_pred_lr_A = model_lr_A.predict(X_test_A)

non_zero_indices = y_test_A != 0
y_test_A_non_zero = y_test_A[non_zero_indices]
y_pred_lr_A_non_zero = y_pred_lr_A[non_zero_indices]

mape_lr_A_error = abs(y_pred_lr_A_non_zero - y_test_A_non_zero)
mape_lr_A = 100 * (mape_lr_A_error / y_test_A_non_zero)
accuracy_lr_A = 100 - np.mean(mape_lr_A)


print("Linear Regression Model Prediction Accuracy for Amplitude A:", round(accuracy_lr_A, 2), "%")

# For Amplitude B
y_pred_lr_B = model_lr_B.predict(X_test_B)
```

```
non_zero_indices = y_test_B != 0
y_test_B_non_zero = y_test_B[non_zero_indices]
y_pred_lr_B_non_zero = y_pred_lr_B[non_zero_indices]

mape_lr_B_error = abs(y_pred_lr_B_non_zero - y_test_B_non_zero)
mape_lr_B = 100 * (mape_lr_B_error / y_test_B_non_zero)
accuracy_lr_B = 100 - np.mean(mape_lr_B)

# For Amplitude C
y_pred_lr_C = model_lr_C.predict(X_test_C)

non_zero_indices = y_test_C != 0
y_test_C_non_zero = y_test_C[non_zero_indices]
y_pred_lr_C_non_zero = y_pred_lr_C[non_zero_indices]

mape_lr_C_error = abs(y_pred_lr_C_non_zero - y_test_C_non_zero)
mape_lr_C = 100 * (mape_lr_C_error / y_test_C_non_zero)
accuracy_lr_C = 100 - np.mean(mape_lr_C)

print("Linear Regression Model Prediction Accuracy for Amplitude B:", round(accuracy_lr_B, 2), "%")
print("Linear Regression Model Prediction Accuracy for Amplitude C:", round(accuracy_lr_C, 2), "%")
```

```
Linear Regression Model Prediction Accuracy for Amplitude A: 94.83 %
Linear Regression Model Prediction Accuracy for Amplitude B: 95.19 %
Linear Regression Model Prediction Accuracy for Amplitude C: 96.68 %
```

In [69]:

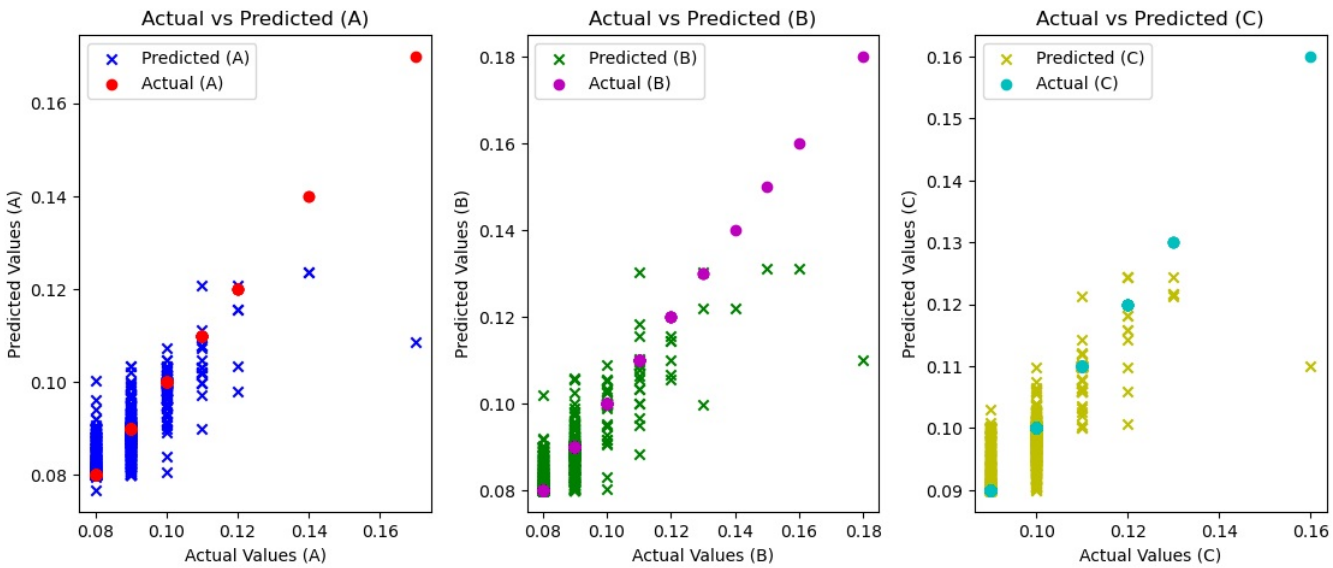
```
# For Amplitude A
y_pred_rf_A = model_rf_A.predict(X_test_A)

non_zero_indices = y_test_A != 0
y_test_A_non_zero = y_test_A[non_zero_indices]
y_pred_rf_A_non_zero = y_pred_rf_A[non_zero_indices]

mape_rf_A_error = abs(y_pred_rf_A_non_zero - y_test_A_non_zero)
mape_rf_A = 100 * (mape_rf_A_error / y_test_A_non_zero)
accuracy_rf_A = 100 - np.mean(mape_rf_A)

# For Amplitude B
y_pred_rf_B = model_rf_B.predict(X_test_B)

non_zero_indices = y_test_B != 0
y_test_B_non_zero = y_test_B[non_zero_indices]
y_pred_rf_B_non_zero = y_pred_rf_B[non_zero_indices]

mape_rf_B_error = abs(y_pred_rf_B_non_zero - y_test_B_non_zero)
mape_rf_B = 100 * (mape_rf_B_error / y_test_B_non_zero)
accuracy_rf_B = 100 - np.mean(mape_rf_B)

# For Amplitude C
y_pred_rf_C = model_rf_C.predict(X_test_C)

non_zero_indices = y_test_C != 0
y_test_C_non_zero = y_test_C[non_zero_indices]
y_pred_rf_C_non_zero = y_pred_rf_C[non_zero_indices]

mape_rf_C_error = abs(y_pred_rf_C_non_zero - y_test_C_non_zero)
mape_rf_C = 100 * (mape_rf_C_error / y_test_C_non_zero)
accuracy_rf_C = 100 - np.mean(mape_rf_C)

print("Random Forest Regressor Model Prediction Accuracy for Amplitude A:", round(accuracy_rf_A, 2), "%")
print("Random Forest Regressor Model Prediction Accuracy for Amplitude B:", round(accuracy_rf_B, 2), "%")
print("Random Forest Regressor Model Prediction Accuracy for Amplitude C:", round(accuracy_rf_C, 2), "%")
```

```
Random Forest Regressor Model Prediction Accuracy for Amplitude A: 97.57 %
Random Forest Regressor Model Prediction Accuracy for Amplitude B: 97.78 %
Random Forest Regressor Model Prediction Accuracy for Amplitude C: 98.11 %
```

In [70]:

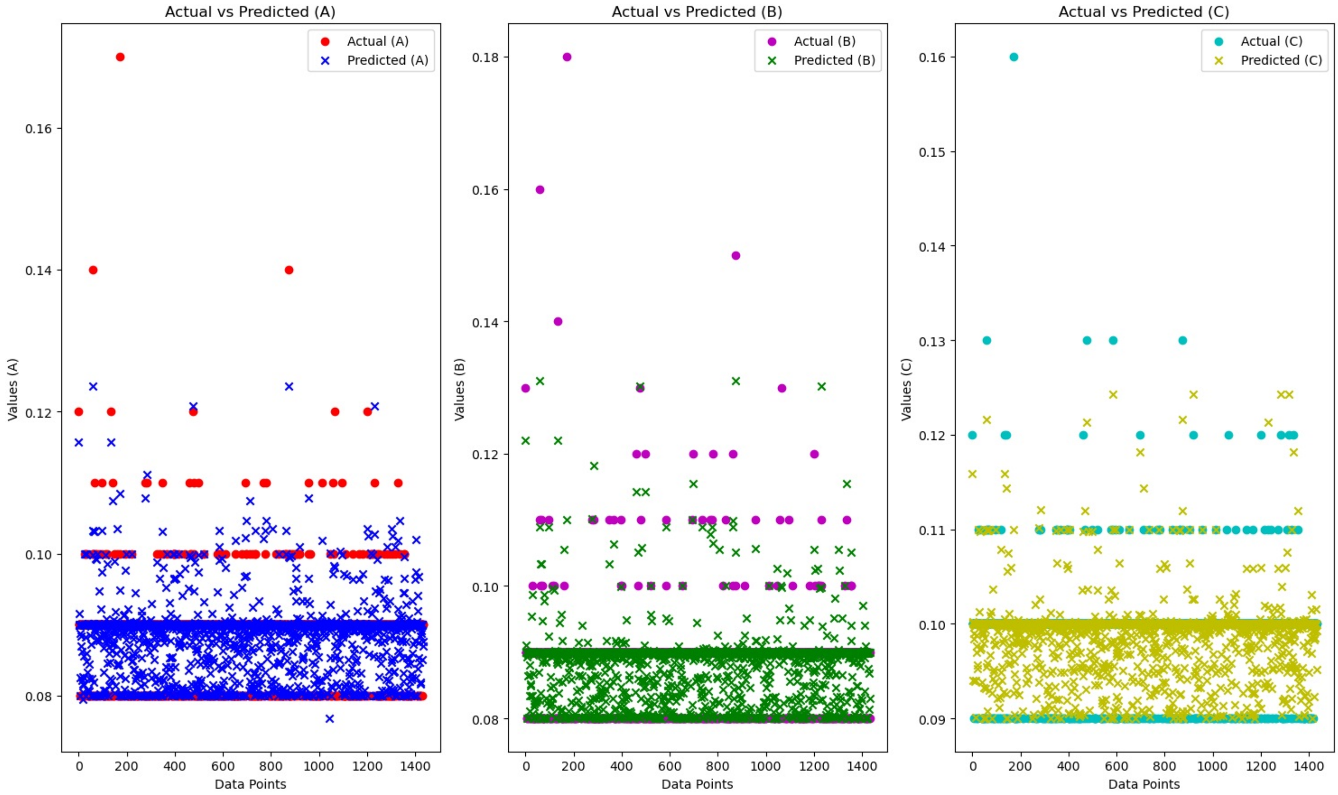
```
import matplotlib.pyplot as plt

# Assuming y_test_A, y_test_B, y_test_C are the actual values for A, B, and C respectively
# Assuming y_pred_rf_A, y_pred_rf_B, y_pred_rf_C are the predicted values for A, B, and C respectively

# Set the figure size
plt.figure(figsize=(15, 9))

# Plot for Amplitude A
plt.subplot(2, 4, 1)
plt.scatter(y_test_A, y_pred_rf_A, color='b', label='Predicted (A)', marker='x')
plt.scatter(y_test_A, y_test_A, color='r', label='Actual (A)', marker='o')
plt.xlabel('Actual Values (A)')
plt.ylabel('Predicted Values (A)')
plt.title('Actual vs Predicted (A)')
plt.legend()

# Plot for Amplitude B
plt.subplot(2, 4, 2)
plt.scatter(y_test_B, y_pred_rf_B, color='g', label='Predicted (B)', marker='x')
plt.scatter(y_test_B, y_test_B, color='m', label='Actual (B)', marker='o')
plt.xlabel('Actual Values (B)')
```

```
plt.ylabel('Predicted Values (B)')
plt.title('Actual vs Predicted (B)')
plt.legend()

# Plot for Amplitude C
plt.subplot(2, 4, 3)
plt.scatter(y_test_C, y_pred_rf_C, color='y', label='Predicted (C)', marker='x')
plt.scatter(y_test_C, y_test_C, color='c', label='Actual (C)', marker='o')
plt.xlabel('Actual Values (C)')
plt.ylabel('Predicted Values (C)')
plt.title('Actual vs Predicted (C)')
plt.legend()

plt.tight_layout()
plt.show()
```



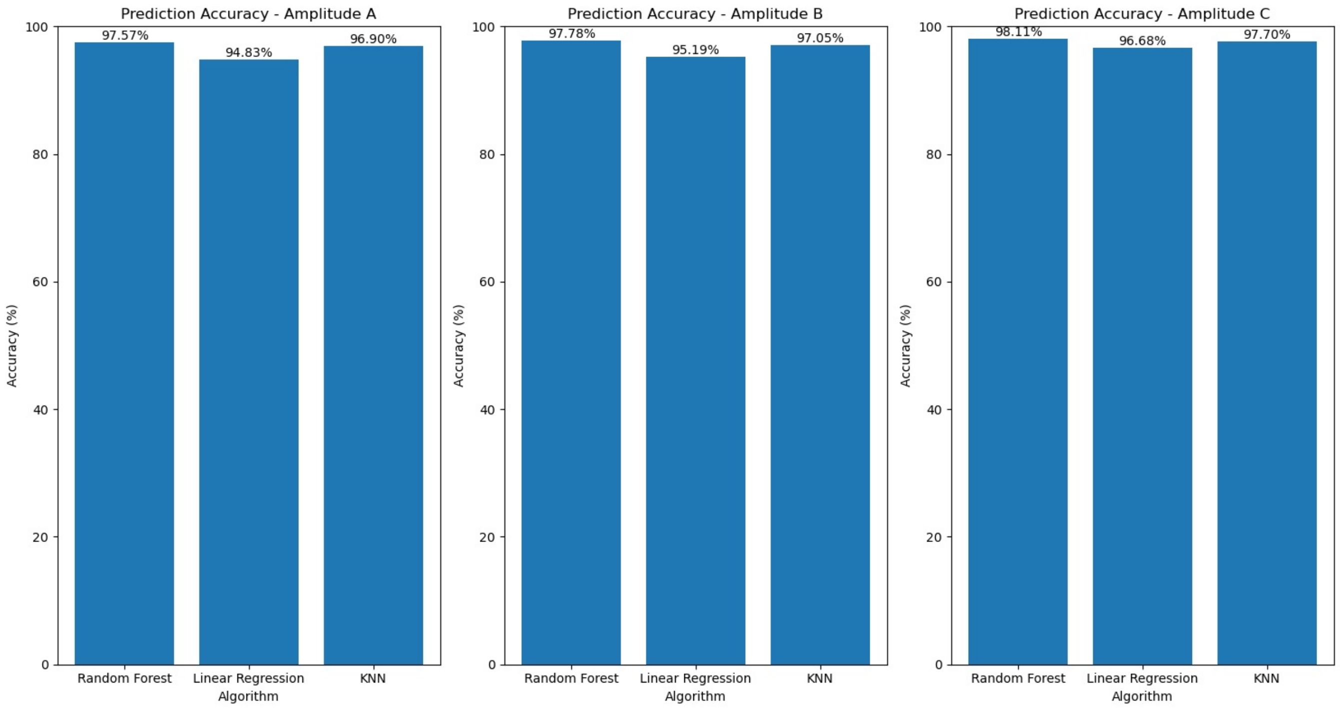
In [71]:

```
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(15, 9))

# Plot for Amplitude A
plt.subplot(1, 3, 1)
plt.scatter(range(len(y_test_A)), y_test_A, color='r', label='Actual (A)', marker='o')
plt.scatter(range(len(y_pred_rf_A)), y_pred_rf_A, color='b', label='Predicted (A)', marker='x')
plt.xlabel('Data Points')
plt.ylabel('Values (A)')
plt.title('Actual vs Predicted (A)')
plt.legend()

# Plot for Amplitude B
plt.subplot(1, 3, 2)
plt.scatter(range(len(y_test_B)), y_test_B, color='m', label='Actual (B)', marker='o')
plt.scatter(range(len(y_pred_rf_B)), y_pred_rf_B, color='g', label='Predicted (B)', marker='x')
plt.xlabel('Data Points')
plt.ylabel('Values (B)')
plt.title('Actual vs Predicted (B)')
plt.legend()

# Plot for Amplitude C
plt.subplot(1, 3, 3)
plt.scatter(range(len(y_test_C)), y_test_C, color='c', label='Actual (C)', marker='o')
plt.scatter(range(len(y_pred_rf_C)), y_pred_rf_C, color='y', label='Predicted (C)', marker='x')
plt.xlabel('Data Points')
plt.ylabel('Values (C)')
plt.title('Actual vs Predicted (C)')
plt.legend()

plt.tight_layout()
plt.show()
```



In [72]:

```python
import matplotlib.pyplot as plt

# Define the algorithms and their respective accuracies
algorithms = ['Random Forest', 'Linear Regression', 'KNN']
accuracy_A = [accuracy_rf_A, accuracy_lr_A, accuracy_knn_A]
accuracy_B = [accuracy_rf_B, accuracy_lr_B, accuracy_knn_B]
accuracy_C = [accuracy_rf_C, accuracy_lr_C, accuracy_knn_C]

# Set the figure size
plt.figure(figsize=(15, 8))

# Plot the bar chart for Amplitude A
plt.subplot(1, 3, 1)
plt.bar(algorithms, accuracy_A)
plt.ylim(0, 100)
plt.title("Prediction Accuracy - Amplitude A")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy (%)")

# Display the exact percentage values on top of each bar for Amplitude A
for i, acc in enumerate(accuracy_A):
    plt.text(i, acc, f'{acc:.2f}%', ha='center', va='bottom')

# Plot the bar chart for Amplitude B
plt.subplot(1, 3, 2)
plt.bar(algorithms, accuracy_B)
plt.ylim(0, 100)
plt.title("Prediction Accuracy - Amplitude B")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy (%)")

# Display the exact percentage values on top of each bar for Amplitude B
for i, acc in enumerate(accuracy_B):
    plt.text(i, acc, f'{acc:.2f}%', ha='center', va='bottom')

# Plot the bar chart for Amplitude C
plt.subplot(1, 3, 3)
plt.bar(algorithms, accuracy_C)
plt.ylim(0, 100)
plt.title("Prediction Accuracy - Amplitude C")
plt.xlabel("Algorithm")
plt.ylabel("Accuracy (%)")

# Display the exact percentage values on top of each bar for Amplitude C
for i, acc in enumerate(accuracy_C):
    plt.text(i, acc, f'{acc:.2f}%', ha='center', va='bottom')

# Adjust the layout
plt.tight_layout()

# Display the plot
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js