Configuration Manual

MSc Research Project
MSc in Data Analytics

Mustafa Karaburun
Student ID: x23216158

School of Computing
National College of Ireland

Supervisor: Dr David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mustafa Karaburun
Student ID:	X23216158
Programme:	MSc in Data Analytics **Year:** 2024
Module:	MSc Research Project
Lecturer:	Dr David Hamill
Submission Due Date:	12/12/2024
Project Title:	The Relationship Between Weather Factors and Stock Exchange Trading Based in Istanbul
Word Count:	758 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Mustafa Karaburun

Date: 12/12/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mustafa Karaburun
Student ID: x23216158

1 Introduction

The configuration manual provides a detailed explanation of all processes and technologies used in the research project. Having a detailed manual is important because it provides background information about the research and explains how the technology works. It also includes specific technical details that are not covered in the main report. The manual offers a step-by-step guide to the project and also includes the results obtained during the research.

2 Hardware Configuration

The hardware requirements for the project are listed below.

- Model Name: MacBook Air
- Model Identifier: Mac14,2
- Chip: Apple M2
- Total Number of Cores: 8 (4 performance and 4 efficiency)
- Memory: 8 GB
- Storage: 250 GB

3 Software Configuration

The following software is specifically used for the implementation of the project.

- Anaconda
- Jupyter Notebook
- Python

4 Data Selection

The datasets used in the research are Istanbul weather and Istanbul stock exchange (ISE) datasets. The datasets were taken from Kaggle. The weather dataset has daily weather, rainfall, average wind speed, average humidity, maximum, and minimum columns. Istanbul Stock Exchange dataset has 10 columns and is organised by working days in the ISE by Finance Yahoo.

5 Implementation

This section focuses on the implementation of the research project.

5.1 Knowledge Discovery in Databases

KDD is an interdisciplinary field that combines techniques from data mining, machine learning, statistics, database systems, and artificial intelligence. The goal is to transform raw data into meaningful information and actionable knowledge.

5.1.1. Data Selection

How weather data and stock data files are read is shown in Figure 1.

```
[ ] import pandas as pd
```

```
[ ] weather_data = pd.read_csv('Istanbul Weather Data.csv')
    stock_data = pd.read_csv('istanbul_stock_exchange.csv')
```

Figure 1

View the first few rows of both datasets to understand their structure. In Figure 2

```
print("Weather Data Sample:")
print(weather_data.head())
```

```
Weather Data Sample:
     DateTime            Condition  Rain  MaxTemp  MinTemp   SunRise  \
0  02.09.2019        Partly cloudy   0.0       27       22  06:32:00
1  01.09.2019        Partly cloudy   0.0       27       22  06:31:00
2  31.08.2019  Patchy rain possible  0.5       26       22  06:30:00
3  30.08.2019        Partly cloudy   0.0       27       22  06:29:00
4  29.08.2019        Partly cloudy   0.0       27       23  06:27:00

     SunSet MoonRise   MoonSet  AvgWind  AvgHumidity  AvgPressure
0  19:37:00  9:52:00  21:45:00       23           66         1012
1  19:38:00  8:37:00  21:13:00       21           66         1011
2  19:40:00  7:21:00  20:40:00       22           63         1015
3  19:42:00   6:4:00   20:5:00       20           64         1016
4  19:43:00  4:47:00  19:26:00       24           61         1015
```

```
print("\nStock Data Sample:")
print(stock_data.head())
```

```
Stock Data Sample:
      date  TL BASED ISE  USD BASED ISE        SP       DAX      FTSE  \
0  5-Jan-09      0.035754       0.038376 -0.004679  0.002193  0.003894
1  6-Jan-09      0.025426       0.031813  0.007787  0.008455  0.012866
2  7-Jan-09     -0.028862      -0.026353 -0.030469 -0.017833 -0.028735
3  8-Jan-09     -0.062208      -0.084716  0.003391 -0.011726 -0.000466
4  9-Jan-09      0.009860       0.009658 -0.021533 -0.019873 -0.012710

     NIKKEI   BOVESPA        EU        EM
0  0.000000  0.031190  0.012698  0.028524
1  0.004162  0.018920  0.011341  0.008773
2  0.017293 -0.035899 -0.017073 -0.020015
3 -0.040061  0.028283 -0.005561 -0.019424
4 -0.004474 -0.009764 -0.010989 -0.007802
```

Figure 2

Figure 3 shows that converting the date columns to DateTime format and then filtering both datasets for the years 2009 to 2011. Save the filtered datasets.

```
# Convert the date columns to datetime format
weather_data['DateTime'] = pd.to_datetime(weather_data['DateTime'], format='%d.%m.%Y')
stock_data['date'] = pd.to_datetime(stock_data['date'], format='%d-%b-%y')
```

```
 #Filter both datasets for the years 2009 to 2011
weather_data_filtered = weather_data[(weather_data['DateTime'].dt.year >= 2009) & (weather_data['DateTime'].dt.year <= 2011)]
stock_data_filtered = stock_data[(stock_data['date'].dt.year >= 2009) & (stock_data['date'].dt.year <= 2011)]
```

```
print(f"\nWeather Data Records (2009-2011): {weather_data_filtered.shape[0]}")
print(f"Stock Data Records (2009-2011): {stock_data_filtered.shape[0]}")
```

```
Weather Data Records (2009-2011): 1095
Stock Data Records (2009-2011): 536
```

```
import os

# Define the path
save_path = '/Users/mustafakaraburun/Desktop/data/'

# Create the directory if it does not exist
os.makedirs(save_path, exist_ok=True)

# Save the filtered datasets
weather_data_filtered.to_csv(os.path.join(save_path, 'filtered_weather_data.csv'), index=False)
stock_data_filtered.to_csv(os.path.join(save_path, 'filtered_stock_data.csv'), index=False)
```

Figure 3

5.1.2. Data Pre-processing

Figure 4 shows, Checking for missing values in weather data and stock data.

```
missing_weather = weather_data_filtered.isnull().sum()
print("Missing values in Weather Data:")
print(missing_weather)
```

```
Missing values in Weather Data:
DateTime        0
Condition       0
Rain            0
MaxTemp         0
MinTemp         0
SunRise         0
SunSet          0
MoonRise       37
MoonSet        38
AvgWind         0
AvgHumidity     0
AvgPressure     0
dtype: int64
```

```
missing_stock = stock_data_filtered.isnull().sum()
print("\nMissing values in Stock Data:")
print(missing_stock)
```

```
Missing values in Stock Data:
date            0
TL BASED ISE    0
USD BASED ISE   0
SP              0
DAX             0
FTSE            0
NIKKEI          0
BOVESPA         0
EU              0
EM              0
dtype: int64
```

Figure 4

First, merge the datasets on the date columns. Second, drop redundant columns. Third, display the first few rows of the merged dataset. In Figure 5.

```
merged_data = pd.merge(weather_data_filtered, stock_data_filtered, left_on='DateTime', right_on='date', how='inner')
```

```
merged_data = merged_data.drop(columns=['MoonRise', 'MoonSet'])
```

```
print("\nMerged Data Sample:")
print(merged_data.head())
```

```
Merged Data Sample:
    DateTime      Condition  Rain  MaxTemp  MinTemp   SunRise    SunSet  \
0 2011-02-22         Cloudy  0.25        8        5  07:50:00  18:46:00
1 2011-02-21         Cloudy  0.14        6        4  07:51:00  18:45:00
2 2011-02-18  Partly cloudy  2.16       11        8  07:55:00  18:41:00
3 2011-02-17          Sunny  0.04        9        3  07:57:00  18:40:00
4 2011-02-16  Partly cloudy  0.09        5        3  07:58:00  18:39:00

   AvgWind  AvgHumidity  AvgPressure       date  TL BASED ISE  USD BASED ISE  \
0       15           85         1011 2011-02-22     -0.007246      -0.019442
1       18           82         1016 2011-02-21     -0.013069      -0.013706
2       14           81         1013 2011-02-18      0.000191      -0.001653
3        8           72         1018 2011-02-17      0.009310       0.015977
4       25           72         1018 2011-02-16      0.008599       0.013400

         SP       DAX      FTSE    NIKKEI   BOVESPA        EU        EM
0  0.000000 -0.000473 -0.002997 -0.017920 -0.012252 -0.005465 -0.014297
1 -0.020742 -0.014239 -0.011275  0.001358 -0.011942 -0.012615 -0.000958
2  0.001923  0.002872 -0.000723  0.000568  0.005628  0.000572  0.006938
3  0.003071 -0.001186  0.000345  0.002620  0.001686 -0.000581  0.001039
4  0.006238  0.001925  0.007952  0.005717  0.018371  0.006975  0.003039
```

Figure 5

5.2 Regression Analysis

Various packages and libraries used in the regression analysis part of the research are shown in Figure 6.

```python
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
import numpy as np
```

Figure 6

Figure 7 shows, the selecting independent variables and the dependent variable (TL BASED ISE). Adding a constant to the matrix of independent variables. Calculate descriptive statistics for MaxTemp and MinTemp.

```python
x = merged_data[['MaxTemp', 'MinTemp']]
y = merged_data['TL BASED ISE']
```

```python
x = sm.add_constant(X)
```

```python
descriptive_stats = merged_data[['MaxTemp', 'MinTemp']].describe()
```

```python
print(descriptive_stats)
```

```
          MaxTemp     MinTemp
count  536.000000  536.000000
mean    18.164179   11.886194
std      8.313209    6.781235
min     -3.000000   -4.000000
25%     12.000000    6.000000
50%     18.000000   12.000000
75%     25.000000   18.000000
max     35.000000   26.000000
```

Figure 7

5.2.1. Exploratory

Figure 8 shows the correlation between MaxTemp and TL BASED ISE. Correlation between MinTemp and TL BASED ISE.

```python
# MaxTemp vs TL BASED ISE
sns.jointplot(x='MaxTemp', y='TL BASED ISE', data=merged_data, alpha=0.5)
```

```
<seaborn.axisgrid.JointGrid at 0x176cf0ad0>
```

```python
# MinTemp vsTL BASED ISE
sns.jointplot(x='MinTemp', y='TL BASED ISE', data=merged_data, alpha=0.5)
```

```
<seaborn.axisgrid.JointGrid at 0x176cd7010>
```

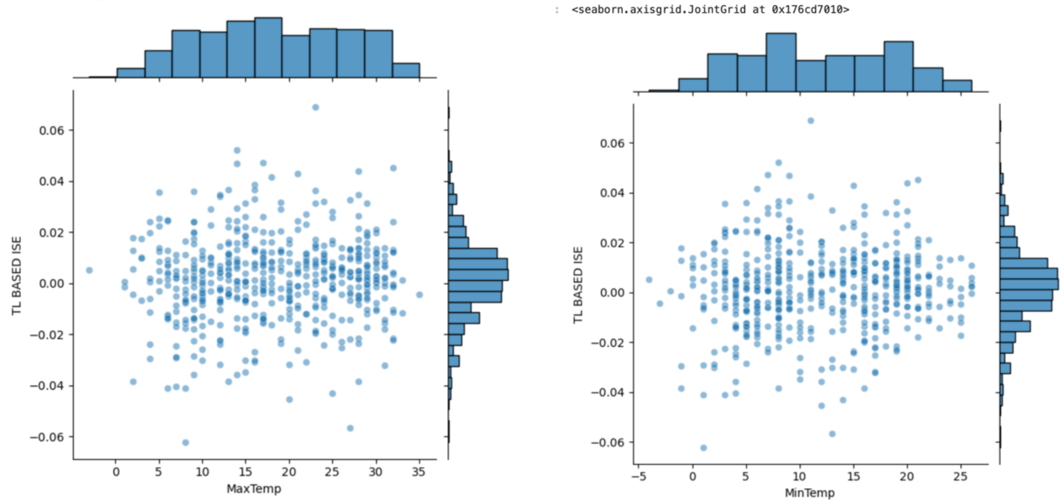


Figure 8

5.2.2. Regression using OLS

Figure 9 shows the fit of the regression model and displays the summary of the regression results.

```
# Fit the regression model
model = sm.OLS(y, x).fit()
```

```
# Display the summary of the regression results
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            TL BASED ISE   R-squared:                       0.004
Model:                             OLS   Adj. R-squared:                  0.001
Method:                  Least Squares   F-statistic:                     1.196
Date:                 Wed, 30 Oct 2024   Prob (F-statistic):              0.303
Time:                         15:59:20   Log-Likelihood:                 1448.8
No. Observations:                  536   AIC:                            -2892.
Df Residuals:                      533   BIC:                            -2879.
Df Model:                            2
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0006      0.002     -0.318      0.750      -0.004       0.003
MaxTemp     6.452e-05      0.000      0.316      0.752      -0.000       0.000
MinTemp     8.493e-05      0.000      0.339      0.735      -0.000       0.001
==============================================================================
Omnibus:                       18.195   Durbin-Watson:                   1.966
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.204
Skew:                          -0.061   Prob(JB):                     1.13e-09
Kurtosis:                       4.353   Cond. No.                         59.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 9

5.2.3. Splitting the data

Figure 10 shows how to split the data with a package that is used.

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

```
# Split the data into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Display the shapes of the train and test sets
print("Training set size:", X_train.shape, y_train.shape)
print("Test set size:", X_test.shape, y_test.shape)
```

```
Training set size: (428, 3) (428,)
Test set size: (108, 3) (108,)
```

Figure 10

5.2.4. Training the model with regression using OLS

Figure 11 shows create and fit the regression model, displays the coefficients of the model, and the coefficients in the data frame.

```python
# Create and fit the regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▸ LinearRegression
```

```python
# Display the coefficients of the model
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

```
Intercept: -0.00063362686404591 73
Coefficients: [ 0.         -0.00015669  0.00037857]
```

```python
# r squared
lm.score(X, y)
```

```
0.0004888068422301828
```

```python
# The coefficients in a dataframe
cdf = pd.DataFrame(lm.coef_,X.columns,columns=['Coef'])
print(cdf)
```

```
             Coef
const    0.000000
MaxTemp -0.000157
MinTemp  0.000379
```

Figure 11

5.2.5. Evaluation & Residuals

Figure 12 shows the evaluation of the model and the model residuals.

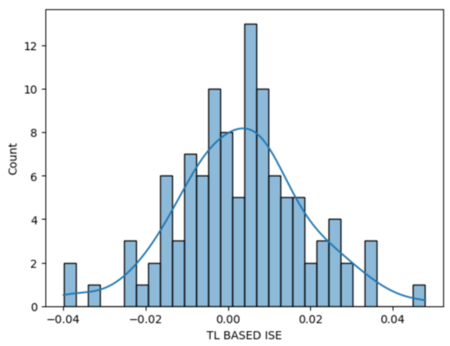
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math
```

```python
print('Mean Absolute Error:',mean_absolute_error(y_test, predictions))
print('Mean Squared Error:',mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:',math.sqrt(mean_squared_error(y_test, predictions)))
```

```
Mean Absolute Error: 0.012340730129101076
Mean Squared Error: 0.0002528007044925492
Root Mean Squared Error: 0.01589970768575791
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

residuals = y_test-predictions
sns.histplot(residuals, bins=30, kde=True)  # kde=True for the density line
plt.show()
```




```python
import pylab
import scipy.stats as stats

stats.probplot(residuals, dist="norm", plot=pylab)
pylab.show()
```

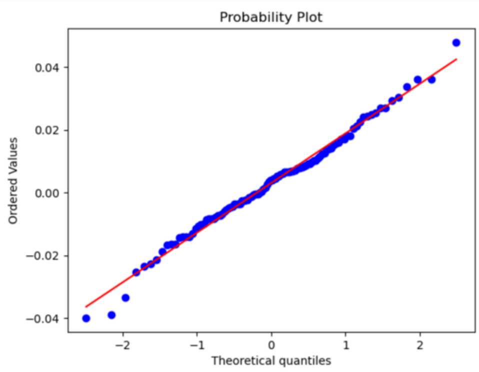


Figure 12

5.3 ANOVA (Analysis of Variance) Analysis

Figure 13 shows how to read the merged data and print it.

```
data = pd.read_csv('Merged.csv')
```

```
print("Merged Data Sample:")
print(data.head())
```

```
Merged Data Sample:
         date  TL BASED ISE  USD BASED ISE        SP       DAX      FTSE  \
0  2009-01-05      0.035754       0.038376 -0.004679  0.002193  0.003894
1  2009-01-06      0.025426       0.031813  0.007787  0.008455  0.012866
2  2009-01-07     -0.028862      -0.026353 -0.030469 -0.017833 -0.028735
3  2009-01-08     -0.062208      -0.084716  0.003391 -0.011726 -0.000466
4  2009-01-09      0.009860       0.009658 -0.021533 -0.019873 -0.012710

     NIKKEI   BOVESPA        EU        EM  ...  Rain MaxTemp  MinTemp  \
0  0.000000  0.031190  0.012698  0.028524  ...  4.32       5        3
1  0.004162  0.018920  0.011341  0.008773  ...  2.71       5        3
2  0.017293 -0.035899 -0.017073 -0.020015  ...  0.23       7        2
3 -0.040061  0.028283 -0.005561 -0.019424  ...  3.06       8        1
4 -0.004474 -0.009764 -0.010989 -0.007802  ...  1.26       3        0

    SunRise    SunSet  MoonRise  MoonSet AvgWind AvgHumidity  AvgPressure
0  08:29:00  17:50:00   0:41:00  1:57:00      15          97         1015
1  08:29:00  17:51:00  13:13:00   3:9:00      22          96         1014
2  08:29:00  17:52:00  13:53:00  4:25:00      11          88         1022
3  08:29:00  17:53:00  14:44:00  5:41:00      12          88         1024
4  08:29:00  17:54:00  15:48:00  6:53:00      23          83         1033

[5 rows x 22 columns]
```

Figure 13

Function to classify each date into a season and apply the season classification to the 'date' column. In Figure 14.

```
from datetime import datetime

# Function to classify each date into a season
def classify_season(date_str):
    date = datetime.strptime(date_str, '%Y-%m-%d')
    month = date.month
    day = date.day

    if (month == 12 and day >= 21) or month in [1, 2] or (month == 3 and day < 20):
        return 'Winter'
    elif (month == 3 and day >= 20) or month in [4, 5] or (month == 6 and day < 21):
        return 'Spring'
    elif (month == 6 and day >= 21) or month in [7, 8] or (month == 9 and day < 23):
        return 'Summer'
    else:
        return 'Autumn'

# Apply the season classification to the 'date' column
data['Season'] = data['date'].apply(classify_season)
data[['date', 'Season']].head()
```

Figure 14

Group stock returns by season in Figure 15.

```
import scipy.stats as stats

# Group stock returns by season
seasonal_data = data[['TL BASED ISE', 'Season']]

# Separate returns by season
winter_returns = seasonal_data[seasonal_data['Season'] == 'Winter']['TL BASED ISE']
spring_returns = seasonal_data[seasonal_data['Season'] == 'Spring']['TL BASED ISE']
summer_returns = seasonal_data[seasonal_data['Season'] == 'Summer']['TL BASED ISE']
autumn_returns = seasonal_data[seasonal_data['Season'] == 'Autumn']['TL BASED ISE']
```

Figure 15

Figure 16 shows the Calculation of the descriptive statistics for each season for TL BASED ISE.

```
# Calculate descriptive statistics for each season for TL BASED ISE stock returns
seasonal_descriptive_stats = data.groupby('Season')['TL BASED ISE'].describe()

# Display the descriptive statistics for each season
seasonal_descriptive_stats
```

Season	count	mean	std	min	25%	50%	75%	max
Autumn	116.0	0.000396	0.014998	-0.035850	-0.007720	0.000982	0.009202	0.038613
Spring	126.0	0.003706	0.019113	-0.056753	-0.006215	0.003409	0.012933	0.068952
Summer	130.0	0.003060	0.012603	-0.025344	-0.004546	0.002510	0.009995	0.045220
Winter	164.0	-0.000229	0.017151	-0.062208	-0.008071	0.000960	0.009773	0.046831

Figure 16

5.3.1. Assumptions

5.3.1.1. Normality

Checking normality assumption using Shapiro-Wilk in Figure 17.

```
from scipy.stats import shapiro, levene
# Checking normality assumption using Shapiro-Wilk test
normality_results = {
    'Winter': shapiro(winter_returns),
    'Spring': shapiro(spring_returns),
    'Summer': shapiro(summer_returns),
    'Autumn': shapiro(autumn_returns)
}
normality_results
```

```
{'Winter': ShapiroResult(statistic=0.9819088994377116, pvalue=0.030750346802652324),
 'Spring': ShapiroResult(statistic=0.9777476964314886, pvalue=0.035686985136607716),
 'Summer': ShapiroResult(statistic=0.9699436481067355, pvalue=0.0055080028015113604),
 'Autumn': ShapiroResult(statistic=0.9863249182737774, pvalue=0.2911781704370465)}
```

Figure 17

5.3.1.2. Homogeneity of variance

Checking homogeneity of variances using Levene's test in Figure 18.

```
levene_result = levene(winter_returns, spring_returns, summer_returns, autumn_returns)
levene_result
```

```
LeveneResult(statistic=4.799837024681618, pvalue=0.0026187706962551467)
```

Figure 18

5.3.2. Normalization

Figure 19 shows applying log transformation, a summary of the transformation, and the Shapiro-Wilk normality test on each transformation.

```
# Applying log transformation
winter_log = np.log(winter_returns + 1)  # Adding 1 to avoid log(0) issues
spring_log = np.log(spring_returns + 1)
summer_log = np.log(summer_returns + 1)
```

```
# Summary of transformations
transformations = {
    "Log Transformation": {"Winter": winter_log, "Spring": spring_log, "Summer": summer_log},
}

transformations
```

```
# Log Transformation
winter_log = np.log(winter_returns + 1)
spring_log = np.log(spring_returns + 1)
summer_log = np.log(summer_returns + 1)


# Shapiro-Wilk normality test on each transformation
normality_transformed_results = {
    "Log Transformation":
    {
        "Winter": shapiro(winter_log).pvalue,
        "Spring": shapiro(spring_log).pvalue,
        "Summer": shapiro(summer_log).pvalue
    }}

normality_transformed_results
```

```
{'Log Transformation': {'Winter': 0.013066771933123365,
  'Spring': 0.03711084459517738,
  'Summer': 0.00874685773530104}}
```

Figure 19

5.3.3. Applying Log transformation to stabilise variance

Figure 20 shows separate transformed returns by season and checks the homogeneity of variances with Levene's test.

```
# Add a small constant (e.g., 1e-5) to avoid log of zero or negative values
data['Log_TL_BASED_ISE'] = np.log(data['TL BASED ISE'] + abs(data['TL BASED ISE'].min()) + 1e-5)

# Separate transformed returns by season
winter_log = data[data['Season'] == 'Winter']['Log_TL_BASED_ISE']
spring_log = data[data['Season'] == 'Spring']['Log_TL_BASED_ISE']
summer_log = data[data['Season'] == 'Summer']['Log_TL_BASED_ISE']
autumn_log = data[data['Season'] == 'Autumn']['Log_TL_BASED_ISE']

# Check homogeneity of variances with Levene's test
levene_log_result = levene(winter_log, spring_log, summer_log, autumn_log)
levene_log_result
```

```
LeveneResult(statistic=2.262795674644082, pvalue=0.08024498629016127)
```

Figure 20

5.3.4. Applying ANOVA

Figure 21 shows the application of ANOVA with logarithmic transformation and Figure 22 shows the pie chart.

```
# Perform ANOVA
anova_log_result = stats.f_oneway(winter_log, spring_log, summer_log, autumn_log)

# Display the result
anova_result
```

```
F_onewayResult(statistic=1.96599913302753, pvalue=0.11805545131254354)
```

Figure 21

```
# Calculate the mean of log-transformed returns for each season
winter_log_mean = np.log(data[data['Season'] == 'Winter']['TL BASED ISE'] + 1).mean()
spring_log_mean = np.log(data[data['Season'] == 'Spring']['TL BASED ISE'] + 1).mean()
summer_log_mean = np.log(data[data['Season'] == 'Summer']['TL BASED ISE'] + 1).mean()
autumn_log_mean = np.log(data[data['Season'] == 'Autumn']['TL BASED ISE'] + 1).mean()

# Store means in a dictionary and take absolute values
mean_log_returns_abs = {
    'Winter': abs(winter_log_mean),
    'Spring': abs(spring_log_mean),
    'Summer': abs(summer_log_mean),
    'Autumn': abs(autumn_log_mean)
}

# Plotting the pie chart with absolute values
colors = ['#FFC300', '#FF5733', '#C70039', '#900C3F']
plt.figure(figsize=(5, 5))
plt.pie(mean_log_returns_abs.values(), labels=mean_log_returns_abs.keys(), autopct='%1.1f%%', startangle=140, colors=colors)
plt.title('Average Absolute Log-Transformed TL Based ISE Returns by Season')
plt.show()
```

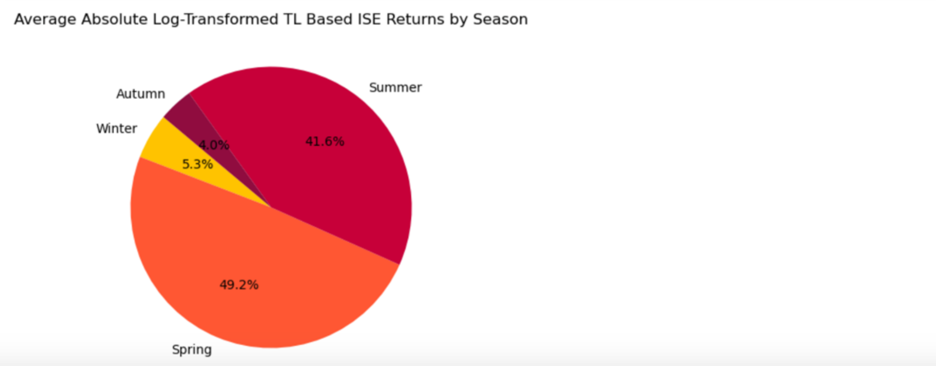


Figure 22

5.4 GARCH Model

Applying the GARCH model with exogenous variables in the variance equation in Figure 21.

```python
import pandas as pd
from arch import arch_model

# Assuming 'data' is already loaded and preprocessed
# Define the returns and rescale if necessary
ise_returns_rescaled = data['TL BASED ISE'].dropna() * 100

# Select relevant weather variables, e.g., 'Rain', 'MaxTemp', and 'AvgHumidity'
exog_variables = data[['Rain', 'MaxTemp', 'AvgHumidity']].dropna()

# Align weather variables with returns by removing corresponding NaN values in returns
ise_returns_rescaled = ise_returns_rescaled.loc[exog_variables.index]

# Apply the GARCH model with exogenous variables in the variance equation
garch_model_exog = arch_model(ise_returns_rescaled, vol='Garch', p=1, q=1, x=exog_variables)
garch_results_exog = garch_model_exog.fit(disp="off")

# Display the summary with weather variables
print(garch_results_exog.summary())
```

Figure 21

Figure 22 shows a plot of volatility, a plot of weather, and a scatter plot of conditional volatility vs. rain.

```python
# Extract the conditional volatility (sigma) from the GARCH model results
conditional_volatility = garch_results_exog.conditional_volatility

# Plot conditional volatility and selected weather variables over time
fig, ax1 = plt.subplots(figsize=(10, 4))

# Plot volatility
ax1.plot(conditional_volatility, color='blue', label='Conditional Volatility')
ax1.set_ylabel('Conditional Volatility')
ax1.set_xlabel('Date')
ax1.legend(loc='upper left')

# Plot weather variable (e.g., rainfall) on the secondary y-axis
ax2 = ax1.twinx()
ax2.plot(data['Rain'], color='green', label='Rain', alpha=0.5)
ax2.set_ylabel('Rain')
ax2.legend(loc='upper right')

plt.title('Conditional Volatility and Rainfall Over Time')
plt.show()

# Scatter Plot: Conditional Volatility vs Rain
plt.figure(figsize=(6, 4))
plt.scatter(data['Rain'], conditional_volatility, alpha=0.5, color='purple')
plt.xlabel('Rain')
plt.ylabel('Conditional Volatility')
plt.title('Scatter Plot of Conditional Volatility vs Rain')
plt.show()
```

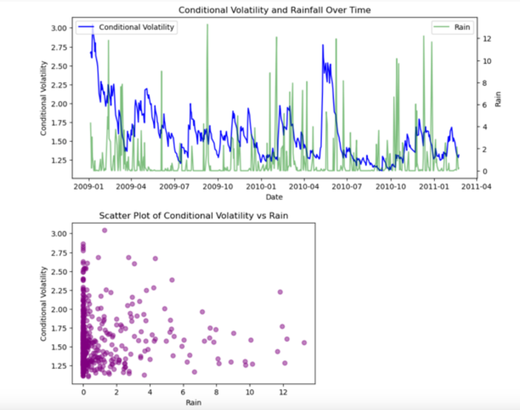


Figure 22

Figure 23 shows a plot of volatility, a plot of weather, and a scatter plot of conditional volatility vs. humidity.

```python
# Extract the conditional volatility (sigma) from the GARCH model results
conditional_volatility = garch_results_exog.conditional_volatility

# Plot conditional volatility and selected weather variables over time
fig, ax1 = plt.subplots(figsize=(10, 4))

# Plot volatility
ax1.plot(conditional_volatility, color='blue', label='Conditional Volatility')
ax1.set_ylabel('Conditional Volatility')
ax1.set_xlabel('Date')
ax1.legend(loc='upper left')


ax2 = ax1.twinx()
ax2.plot(data['AvgHumidity'], color='red', label='AvgHumidity', alpha=0.5)
ax2.set_ylabel('AvgHumidity')
ax2.legend(loc='upper right')

plt.title('Conditional Volatility and AvgHumidity')
plt.show()

# Scatter Plot: Conditional Volatility vs AvgHumidity
plt.figure(figsize=(6, 4))
plt.scatter(data['AvgHumidity'], conditional_volatility, alpha=0.5, color='orange')
plt.xlabel('Average Humidity')
plt.ylabel('Conditional Volatility')
plt.title('Scatter Plot of Conditional Volatility vs Average Humidity')
plt.show()
```

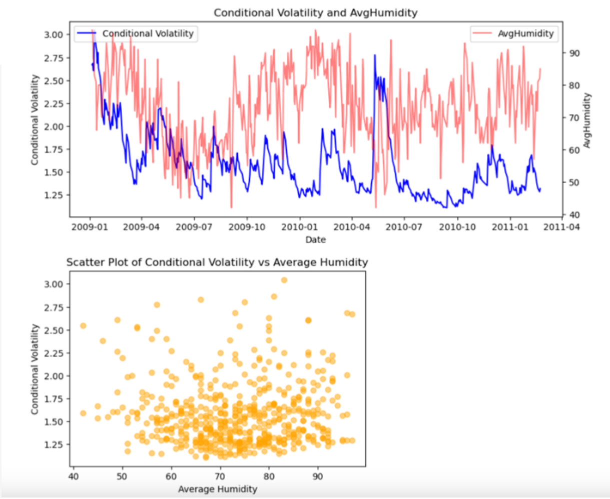


Figure 23