READING THE CSV FILE EXTRACTED FROM NOAA (National Centerers for enviromental information)

In [ ]:
```
import pandas as pd
weather = pd.read_csv("Weather.csv" , index_col="DATE")
weather
```

Out[ ]:

DATE	STATION	NAME	ACMH	ACSH	AWND	FMTM	PGTM	PRCP	SNOW	SNWD	...	WT13	WT14	WT15	WT16	WT17	WT18	W
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	80.0	90.0	NaN	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	30.0	20.0	NaN	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	80.0	100.0	NaN	NaN	NaN	0.02	0.0	0.0	...	NaN	NaN	NaN	1.0	NaN	1.0	I
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	10.0	20.0	NaN	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	1.0	I
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	30.0	10.0	NaN	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	NaN	NaN	8.05	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	NaN	NaN	10.96	NaN	NaN	0.01	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	NaN	NaN	12.30	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	NaN	NaN	6.71	NaN	551.0	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	I

	STATION	NAME	ACMH	ACSH	AWND	FMTM	PGTM	PRCP	SNOW	SNWD	...	WT13	WT14	WT15	WT16	WT17	WT18	W
DATE																		
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	NaN	NaN	NaN	NaN	NaN	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	N

19672 rows × 45 columns

Finding the null values in our data set

In [ ]:
```
weather.apply(pd.isnull).sum() #this will give us the number of null values in the data set
```

Out[ ]:
```
STATION        0
NAME           0
ACMH       10057
ACSH       10056
AWND        5116
FMTM        9548
PGTM        7403
PRCP           0
SNOW           0
SNWD           2
TAVG       13123
TMAX           0
TMIN           0
TSUN       19641
WDF1       10061
WDF2        9618
WDF5        9701
WDFG       14551
WDFM       19671
WESD       13601
WSF1       10058
WSF2        9618
WSF5        9702
WSFG       12209
WSFM       19671
WT01       12393
WT02       18395
WT03       18351
WT04       19329
WT05       19308
WT06       19491
WT07       19564
WT08       15727
WT09       19532
WT11       19658
WT13       17487
WT14       18785
WT15       19630
WT16       13095
WT17       19612
WT18       18505
WT19       19671
WT21       19667
WT22       19623
```

```
WV01        19671
dtype: int64
```

Finding the percentage of null values in our dataset so that we can extract all the data with lower percentage of null values to maintain data quality , reduce biasness , better model performance.

In [ ]:
```
null_percentage=weather.apply(pd.isnull).sum()/weather.shape[0]
null_percentage #showing the percentage of null values in each column
```

Out[ ]:
```
STATION    0.000000
NAME       0.000000
ACMH       0.511234
ACSH       0.511183
AWND       0.260065
FMTM       0.485360
PGTM       0.376322
PRCP       0.000000
SNOW       0.000000
SNWD       0.000102
TAVG       0.667090
TMAX       0.000000
TMIN       0.000000
TSUN       0.998424
WDF1       0.511438
WDF2       0.488918
WDF5       0.493137
WDFG       0.739681
WDFM       0.999949
WESD       0.691389
WSF1       0.511285
WSF2       0.488918
WSF5       0.493188
WSFG       0.620628
WSFM       0.999949
WT01       0.629982
WT02       0.935085
WT03       0.932849
WT04       0.982564
WT05       0.981497
WT06       0.990799
WT07       0.994510
WT08       0.799461
WT09       0.992883
WT11       0.999288
WT13       0.888928
WT14       0.954911
WT15       0.997865
WT16       0.665667
WT17       0.996950
WT18       0.940677
WT19       0.999949
WT21       0.999746
WT22       0.997509
```

```
WV01        0.999949
dtype: float64
```

extracting the data wtih NULL values lower than 5%

In [ ]:
```
Data_with_lower_null_percentage = weather.columns[null_percentage < 0.05]
Data_with_lower_null_percentage
```

Out[ ]:
```
Index(['STATION', 'NAME', 'PRCP', 'SNOW', 'SNWD', 'TMAX', 'TMIN'], dtype='object')
```

copying the data in the original data set to remove all the values that have less than 5% null values

In [ ]:
```
weather = weather[Data_with_lower_null_percentage].copy()
weather.columns=weather.columns.str.lower()
weather
```

Out[ ]:

	station	name	prcp	snow	snwd	tmax	tmin
DATE							
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	28	22
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	22
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	38	25
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	23
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	35	21
...	...	...	...	...	...	...	...
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42

19672 rows × 7 columns

using the ffill function to fill the NULL values with previous most data available and then finding the NULL values to make sure there are no NULL values

In [ ]:
```
weather = weather.ffill() # Fill the missing values with last known value
weather.apply(pd.isnull).sum() #this will give us the number of null values in the data set
```

Out[ ]:
```
station    0
name       0
prcp       0
snow       0
snwd       0
tmax       0
tmin       0
dtype: int64
```

checking the datatypes of the columns that we are gonna feed to our model

In [ ]:
```
weather.dtypes #For training the machine learning model we cannot give it an object type variable so
```

Out[ ]:
```
station     object
name        object
prcp       float64
snow       float64
snwd       float64
tmax         int64
tmin         int64
dtype: object
```

checking the data type of the index (weather) which is an object type here and we need to convert it to numerical format

In [ ]:
```
weather.index # here we can see the index is in object type
```

Out[ ]:
```
Index(['1970-01-01', '1970-01-02', '1970-01-03', '1970-01-04', '1970-01-05',
       '1970-01-06', '1970-01-07', '1970-01-08', '1970-01-09', '1970-01-10',
       ...
       '2023-11-01', '2023-11-02', '2023-11-03', '2023-11-04', '2023-11-05',
       '2023-11-06', '2023-11-07', '2023-11-08', '2023-11-09', '2023-11-10'],
      dtype='object', name='DATE', length=19672)
```

In [ ]:
```
# weather.index.month #this will give us an error because it is an object type in string format
```

as we saw that the datatype of our index is objectype , we cannot feed this data to our model hence we will convert it to datetime,time-series data.

In [ ]:
```
weather.index=pd.to_datetime(weather.index) # using pandas to convert it to datetime
weather.index
```

Out[ ]:
```
DatetimeIndex(['1970-01-01', '1970-01-02', '1970-01-03', '1970-01-04',
               '1970-01-05', '1970-01-06', '1970-01-07', '1970-01-08',
               '1970-01-09', '1970-01-10',
               ...
               '2023-11-01', '2023-11-02', '2023-11-03', '2023-11-04',
               '2023-11-05', '2023-11-06', '2023-11-07', '2023-11-08',
               '2023-11-09', '2023-11-10'],
              dtype='datetime64[ns]', name='DATE', length=19672, freq=None)
```

Now the data is in the correct format.

In [ ]:
```
weather.index.year
```

Out[ ]:
```
Index([1970, 1970, 1970, 1970, 1970, 1970, 1970, 1970, 1970, 1970,
       ...
       2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023, 2023],
      dtype='int32', name='DATE', length=19672)
```

In [ ]:
```
weather.index.month
```

Out[ ]:
```
Index([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
       ...
       11, 11, 11, 11, 11, 11, 11, 11, 11, 11],
      dtype='int32', name='DATE', length=19672)
```

Finding days per year and also finding leap year

In [ ]:
```
weather.index.year.value_counts().sort_index()
#tells us how many times the unique data is available in the data set , and sort them
```

Out[ ]:
```
DATE
1970    365
1971    365
1972    366
1973    365
1974    365
1975    365
1976    366
1977    365
1978    365
1979    365
1980    366
1981    365
1982    365
1983    365
1984    366
1985    365
1986    365
1987    365
1988    366
1989    365
1990    365
1991    365
1992    366
1993    365
1994    365
1995    365
1996    366
1997    365
1998    365
1999    365
2000    366
2001    365
2002    365
2003    365
2004    366
2005    365
2006    365
2007    365
2008    366
2009    365
2010    365
2011    365
2012    366
```

```
2013    365
2014    365
2015    365
2016    366
2017    365
2018    365
2019    365
2020    366
2021    365
2022    365
2023    314
Name: count, dtype: int64
```

In [ ]:
```
# weather["snwd"].plot()  # snow depth
```

In [ ]:
```
#now we are gonna start working on the model
```

In [ ]:
```
weather
```

Out[ ]:

DATE	station	name	prcp	snow	snwd	tmax	tmin
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	28	22
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	22
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	38	25
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	23
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	35	21
...	...	...	...	...	...	...	...
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42

19672 rows × 7 columns

So, overall, the line of code assigns the values in the "tmax" column shifted up by one position to the new "target" column in the original weather DataFrame. This can be useful in time series analysis or when you want to create a lag feature, where each value in the "target" column represents the next day's maximum temperature.

In [ ]:

```
#generating a target for the Total max temperature for tomorrow
weather["target"] = weather.shift(-1)["tmax"]
weather
```

Out[ ]:

	station	name	prcp	snow	snwd	tmax	tmin	target
DATE								
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	28	22	31.0
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	22	38.0
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	38	25	31.0
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	23	35.0
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	35	21	36.0
...	...	...	...	...	...	...	...	...
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	NaN

19672 rows × 8 columns

In [ ]:

```
weather = weather.ffill()
weather
```

Out[ ]:

DATE	station	name	prcp	snow	snwd	tmax	tmin	target
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	28	22	31.0
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	22	38.0
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	38	25	31.0
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	23	35.0
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	35	21	36.0
...	...	...	...	...	...	...	...	...
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	53.0

19672 rows × 8 columns

Why use Ridge Regression: Preventing Overfitting: Handling Multicollinearity:

In [ ]:

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import RidgeCV
rr = Ridge(alpha=0.1)  # Creating a Ridge regression model with a specific alpha value

predictors = weather.columns[~weather.columns.isin(["target", "name", "station"])]  # Selecting predictor columns

X_train=weather['tmax']
X_train = weather[predictors]
y_train = weather["target"]
ridge = Ridge()  # Creating a Ridge regression model without specifying alpha
param_grid = {'alpha': [0.1, 1.0, 10.0]}  # Defining a grid of alpha values to search
grid = GridSearchCV(ridge, param_grid, cv=5)  # Creating a GridSearchCV object with 5-fold cross-validation
grid.fit(X_train, y_train)  # Fitting the model with the training data

best_alpha = grid.best_params_['alpha']  # Extracting the best alpha value from the grid search results
```

```
print(f"Best alpha {best_alpha}")
alphas = [0.1, 1.0, 10.0]
ridge_cv = RidgeCV(alphas=alphas, cv=5)
ridge_cv.fit(X_train, y_train)

best_alpha_cv = ridge_cv.alpha_
print(f"Cross validation of best alpha {best_alpha_cv}")
```

```
Best alpha 10.0
Cross validation of best alpha 10.0
```

In [ ]:

```
# type(weather)
# weather.corr()
# weather.dtypes
# this works linear regression
# lambda
rr= Ridge(alpha=10.0) #alpha  parameter controls how much the co-efficients are shrunk
predictors = weather.columns[~weather.columns.isin(["target","name","station"])] # ~ looks for all columns in the lis
predictors
```

Out[ ]:

```
Index(['prcp', 'snow', 'snwd', 'tmax', 'tmin'], dtype='object')
```

In [ ]:

```
# #for predicting data till 2023 from 1980
# #taking data from first 10 years
# #taking 10 years if data 3650 total days
# #step means taking that amount of data and moving forward
# def backt_esting(weather , model ,   predictors , start=3650 , step=90):
#     all_predictions=[]

#     for i in range(start,weather.shape[0],step):
#         train = weather.iloc[ :i,:]
#         test = weather.iloc[i:(i+step),:]

#         model.fit(train[predictors],train["target"])
#         preds = model.predict(test[predictors])

#         preds = pd.Series(preds,index=test.index)

#         combined = pd.concat([test["target"],preds],axis=1)

#         combined.columns = ["actual" ,"prediction"]
#         combined["diff"] = (combined["prediction"] - combined["actual"]).abs()
```

```
#         all_predictions.append(combined)
#     return pd.concat(all_predictions)
```

In [ ]:
```
def backt_esting(weather, model, predictors, start=3650, step=90):
    all_predictions = []

    # Loop through the data starting from the specified index (default: 3650) with a specified step (default: 90)
    for i in range(start, weather.shape[0], step):
        # Split the data into training and testing sets
        train = weather.iloc[:i, :]
        test = weather.iloc[i:(i + step), :]

        # Fit the model on the training data
        model.fit(train[predictors], train["target"])

        # Make predictions on the testing data
        preds = model.predict(test[predictors])

        # Create a Pandas Series with predicted values and corresponding index (dates)
        preds = pd.Series(preds, index=test.index)

        # Combine actual target values and predicted values into a DataFrame
        combined = pd.concat([test["target"], preds], axis=1)
        combined.columns = ["actual", "prediction"]

        # Calculate the absolute difference between actual and predicted values
        combined["diff"] = (combined["prediction"] - combined["actual"]).abs()

        # Append the combined DataFrame to the list
        all_predictions.append(combined)

    # Concatenate all DataFrames in the list into a single DataFrame
    return pd.concat(all_predictions)
```

In [ ]:
```
predictions = backt_esting(weather,rr,predictors)
predictions
```

Out[ ]:

DATE	actual	prediction	diff
1979-12-30	43.0	50.222377	7.222377
1979-12-31	42.0	43.669095	1.669095
1980-01-01	41.0	41.575789	0.575789
1980-01-02	36.0	43.954768	7.954768
1980-01-03	30.0	40.196665	10.196665
...	...	...	...
2023-11-06	63.0	57.037580	5.962420
2023-11-07	53.0	65.325387	12.325387
2023-11-08	59.0	54.144181	4.855819
2023-11-09	53.0	55.805048	2.805048
2023-11-10	53.0	55.170736	2.170736

16022 rows × 3 columns

In [ ]:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

# Assuming "model" is a functional regression model
# Replace the following line with your actual functional regression model
model = make_pipeline(FunctionTransformer(), LinearRegression())

# Define the functional backtesting function
def functional_backtesting(weather, model, predictors, target_col="target", start=3650, step=90):
    all_predictions = []

    # Loop through the data starting from the specified index with a given step
    for i in range(start, weather.shape[0], step):
        # Split the data into training and testing sets
```

```
        train = weather.iloc[:i, :]
        test = weather.iloc[i:(i + step), :]

        # Fit the functional regression model using the training data
        model.fit(train[predictors], train[target_col])

        # Make predictions on the testing data
        preds = model.predict(test[predictors])

        # Create a Pandas Series with predicted values and corresponding index (dates)
        preds = pd.Series(preds, index=test.index)

        # Combine actual target values and predicted values into a DataFrame
        combined = pd.concat([test[target_col], preds], axis=1)
        combined.columns = ["actual_F", "prediction_F"]

        # Calculate Mean Absolute Error (MAE) as an example of a performance metric
        mae = mean_absolute_error(combined["actual_F"], combined["prediction_F"])
        print(f"MAE for step {i}: {mae}")

        # Calculate the absolute difference between actual and predicted values
        combined["diff_F"] = (combined["prediction_F"] - combined["actual_F"]).abs()

        # Append the combined DataFrame to the list
        all_predictions.append(combined)

    # Concatenate all DataFrames in the list into a single DataFrame
    return pd.concat(all_predictions)

# Example usage:
# Replace "your_functional_regression_model" with your actual functional regression model
# Replace "your_predictors" with the column names of your predictor variables
# Replace "your_target_column" with the name of your target variable
Functional_Backtesting = functional_backtesting(weather, model, predictors, target_col="target")
```

```
MAE for step 3650: 4.892245475400731
MAE for step 3740: 5.243163718227134
MAE for step 3830: 3.633700724434166
MAE for step 3920: 5.214827106760509
MAE for step 4010: 5.918168147688314
MAE for step 4100: 5.570812755417433
MAE for step 4190: 4.220160082942428
MAE for step 4280: 5.424108310149346
MAE for step 4370: 6.276699810028904
MAE for step 4460: 5.663657769208096
MAE for step 4550: 3.583303390402681
MAE for step 4640: 4.768332731440679
MAE for step 4730: 5.687031948328768
MAE for step 4820: 4.348743370892509
MAE for step 4910: 4.412149872194901
MAE for step 5000: 4.577656909940419
MAE for step 5090: 6.7910543963183345
MAE for step 5180: 4.681172400278418
MAE for step 5270: 3.7230144143087616
MAE for step 5360: 5.31231032959455
MAE for step 5450: 5.973633938260999
MAE for step 5540: 6.778207184295366
MAE for step 5630: 3.568434809706317
MAE for step 5720: 5.241396890835153
MAE for step 5810: 5.5465015973329965
MAE for step 5900: 6.460708746228757
MAE for step 5990: 5.038390117551201
MAE for step 6080: 5.216152095390512
MAE for step 6170: 5.190148100997851
MAE for step 6260: 5.756281312639559
MAE for step 6350: 5.356585412322728
MAE for step 6440: 4.932648541863377
MAE for step 6530: 6.062777373061294
MAE for step 6620: 5.306783460039586
MAE for step 6710: 4.686294879682417
MAE for step 6800: 4.66462284216674
MAE for step 6890: 6.032485351637287
MAE for step 6980: 5.653446887606549
MAE for step 7070: 4.491070369802418
MAE for step 7160: 4.859247676831179
MAE for step 7250: 5.591704693667175
MAE for step 7340: 6.647215836061274
MAE for step 7430: 4.751880855604678
MAE for step 7520: 3.8927504735967
```

```
MAE for step 7610: 6.353247440737792
MAE for step 7700: 5.914748668993478
MAE for step 7790: 5.965340957063862
MAE for step 7880: 4.465763452240545
MAE for step 7970: 5.811364028918725
MAE for step 8060: 5.435679374481948
MAE for step 8150: 5.110850730795418
MAE for step 8240: 3.9155789794053155
MAE for step 8330: 5.2694944675608735
MAE for step 8420: 5.322040496427273
MAE for step 8510: 5.4890415913856465
MAE for step 8600: 3.749165873493493
MAE for step 8690: 5.304502246704873
MAE for step 8780: 6.09264674472413
MAE for step 8870: 5.166237646920921
MAE for step 8960: 3.823642877044178
MAE for step 9050: 5.3369226653915
MAE for step 9140: 5.054496692008149
MAE for step 9230: 4.914273523191628
MAE for step 9320: 4.194939936522184
MAE for step 9410: 5.041189883473457
MAE for step 9500: 5.571504306775433
MAE for step 9590: 5.566533743945015
MAE for step 9680: 3.5978842406206315
MAE for step 9770: 4.968090711471082
MAE for step 9860: 7.504124348055685
MAE for step 9950: 5.458925352717525
MAE for step 10040: 4.309550323075844
MAE for step 10130: 4.9482290299670755
MAE for step 10220: 4.968189444787783
MAE for step 10310: 5.520908717705091
MAE for step 10400: 3.6213272025152334
MAE for step 10490: 5.076156151932012
MAE for step 10580: 6.084065573902563
MAE for step 10670: 5.473676339797673
MAE for step 10760: 4.002701002682273
MAE for step 10850: 5.459988115959465
MAE for step 10940: 6.199233770404553
MAE for step 11030: 6.311525731547677
MAE for step 11120: 4.17698565744739
MAE for step 11210: 5.06486687507
MAE for step 11300: 5.50990539764897
MAE for step 11390: 5.1961177976736135
MAE for step 11480: 3.5837586264998156
```

```
MAE for step 11570: 5.312441852613116
MAE for step 11660: 5.270575752954942
MAE for step 11750: 6.405219426749482
MAE for step 11840: 4.424347320060354
MAE for step 11930: 4.481191847907875
MAE for step 12020: 5.176009528973357
MAE for step 12110: 6.307406666648396
MAE for step 12200: 4.589758634178475
MAE for step 12290: 4.3453511779458145
MAE for step 12380: 6.072109250741194
MAE for step 12470: 6.310592053600292
MAE for step 12560: 3.808496075498288
MAE for step 12650: 4.112598476365553
MAE for step 12740: 6.5230459222732895
MAE for step 12830: 4.324743704029857
MAE for step 12920: 4.5086947517329845
MAE for step 13010: 3.8409491834356477
MAE for step 13100: 6.635922538788449
MAE for step 13190: 5.72222595807146
MAE for step 13280: 3.6752407246712564
MAE for step 13370: 4.16128494851614
MAE for step 13460: 5.7662596480684085
MAE for step 13550: 5.571192172297128
MAE for step 13640: 4.43948896852085
MAE for step 13730: 3.8409088828379985
MAE for step 13820: 5.791876591947357
MAE for step 13910: 6.0819167578671856
MAE for step 14000: 4.3972373114276415
MAE for step 14090: 3.6446404432369484
MAE for step 14180: 6.498989061544777
MAE for step 14270: 6.320385879015575
MAE for step 14360: 4.50610033564575
MAE for step 14450: 4.539963660827659
MAE for step 14540: 5.007221406426475
MAE for step 14630: 5.404255562874527
MAE for step 14720: 5.721905828014087
MAE for step 14810: 4.312000217196489
MAE for step 14900: 4.515760875255719
MAE for step 14990: 5.922600807321491
MAE for step 15080: 4.619763227799494
MAE for step 15170: 4.069704394127505
MAE for step 15260: 5.6277282105802255
MAE for step 15350: 6.584984389084725
MAE for step 15440: 5.024805973338188
```

```
MAE for step 15530: 3.399383207023047
MAE for step 15620: 4.182837728629862
MAE for step 15710: 5.234473495931856
MAE for step 15800: 5.014422573238856
MAE for step 15890: 3.788492331951641
MAE for step 15980: 5.383703999697413
MAE for step 16070: 6.8788119725095545
MAE for step 16160: 5.461993819145933
MAE for step 16250: 3.0136381470978426
MAE for step 16340: 5.728002392868877
MAE for step 16430: 6.179503961879066
MAE for step 16520: 5.6534193601858655
MAE for step 16610: 3.0057548624724846
MAE for step 16700: 5.261876334203516
MAE for step 16790: 6.873394966888112
MAE for step 16880: 6.02885524734633
MAE for step 16970: 3.6128251650494905
MAE for step 17060: 5.025561878734974
MAE for step 17150: 7.070199612782164
MAE for step 17240: 6.187110815572306
MAE for step 17330: 4.035157133678076
MAE for step 17420: 4.797593618982584
MAE for step 17510: 6.296188839889746
MAE for step 17600: 6.1827691144935075
MAE for step 17690: 3.985750831118977
MAE for step 17780: 5.234460973975771
MAE for step 17870: 5.968276573261668
MAE for step 17960: 6.0636535254018
MAE for step 18050: 3.69249066829285
MAE for step 18140: 4.863707358773755
MAE for step 18230: 5.827239973998073
MAE for step 18320: 5.654167231528391
MAE for step 18410: 4.124681397870691
MAE for step 18500: 4.838149912140424
MAE for step 18590: 5.0977553468326215
MAE for step 18680: 6.4047253590737565
MAE for step 18770: 4.400389571832202
MAE for step 18860: 4.440771983005348
MAE for step 18950: 6.746106948516124
MAE for step 19040: 6.106902766713401
MAE for step 19130: 4.361374050735806
MAE for step 19220: 4.555299039265511
MAE for step 19310: 5.905040537290775
MAE for step 19400: 6.050495662353673
```

```
MAE for step 19490: 3.684753835069082
MAE for step 19580: 4.247283609744148
MAE for step 19670: 2.488372590547602
```

In [ ]:
```
# import pandas as pd
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import mean_absolute_error
# from sklearn.preprocessing import FunctionTransformer
# from sklearn.linear_model import LinearRegression
# from sklearn.pipeline import make_pipeline
# # Assuming "model" is a functional regression model
# # Replace the following line with your actual functional regression model
# model = make_pipeline(FunctionTransformer(), LinearRegression())

# def functional_backtesting(weather, model, predictors, target_col="target", start=3650, step=90):
#     all_predictions = []

#     for i in range(start, weather.shape[0], step):
#         train = weather.iloc[:i, :]
#         test = weather.iloc[i:(i + step), :]

#         model.fit(train[predictors], train[target_col])
#         preds = model.predict(test[predictors])

#         preds = pd.Series(preds, index=test.index)

#         combined = pd.concat([test[target_col], preds], axis=1)
#         combined.columns = ["actual_F", "prediction_F"]

#         # You can calculate additional performance metrics if needed
#         # For example, Mean Absolute Error (MAE)
#         mae = mean_absolute_error(combined["actual_F"], combined["prediction_F"])

#         print(f"MAE for step {i}: {mae}")
#         combined["diff_F"] = (combined["prediction_F"] - combined["actual_F"]).abs()
#         all_predictions.append(combined)

#     return pd.concat(all_predictions)


# # Example usage:
# # Replace "your_functional_regression_model" with your actual functional regression model
# # Replace "your_predictors" with the column names of your predictor variables
```

```
# # Replace "your_target_column" with the name of your target variable
# Functional_Backtesting = functional_backtesting(weather, model, predictors, target_col="target")
```

In [ ]:

```
Functional_Backtesting
```

Out[ ]:

	actual_F	prediction_F	diff_F
DATE			
1979-12-30	43.0	50.229396	7.229396
1979-12-31	42.0	43.673846	1.673846
1980-01-01	41.0	41.579185	0.579185
1980-01-02	36.0	43.961961	7.961961
1980-01-03	30.0	40.204809	10.204809
...	...	...	...
2023-11-06	63.0	57.038346	5.961654
2023-11-07	53.0	65.326408	12.326408
2023-11-08	59.0	54.144923	4.855077
2023-11-09	53.0	55.805145	2.805145
2023-11-10	53.0	55.171600	2.171600

16022 rows × 3 columns

In [ ]:

```
def fahrenhit_to_celcius(temp):
    cel = (temp - 32) * 5/9
    return cel
predictions
predictions.mean()
predictions['Acutal celcius'] = predictions['actual'].apply(fahrenhit_to_celcius)
predictions['Prediction in celcius'] = predictions['prediction'].apply(fahrenhit_to_celcius)
predictions['prediction_celcius_diff'] = predictions['diff'].apply(fahrenhit_to_celcius)
predictions['prediction_celcius_diff']=predictions['prediction_celcius_diff']
predictions
# predictions["Temperature_celcius_diff"].rename()
from sklearn.metrics import mean_absolute_error
x =mean_absolute_error(predictions["actual"],predictions["prediction"])
# print(f"In Celcius :  {fahrenhit_to_celcius(x)}")
```

```
print(f"Mean Absolute Error : {x}")
# mean_absolute_error(predictions["Prediction in celcius"])
predictions
```

```
Mean Absolute Error : 5.136134268660916
```

Out[ ]:

DATE	actual	prediction	diff	Acutal celcius	Prediction in celcius	prediction_celcius_diff
1979-12-30	43.0	50.222377	7.222377	6.111111	10.123543	-13.765346
1979-12-31	42.0	43.669095	1.669095	5.555556	6.482831	-16.850503
1980-01-01	41.0	41.575789	0.575789	5.000000	5.319883	-17.457895
1980-01-02	36.0	43.954768	7.954768	2.222222	6.641538	-13.358462
1980-01-03	30.0	40.196665	10.196665	-1.111111	4.553703	-12.112964
...	...	...	...	...	...	...
2023-11-06	63.0	57.037580	5.962420	17.222222	13.909766	-14.465322
2023-11-07	53.0	65.325387	12.325387	11.666667	18.514104	-10.930341
2023-11-08	59.0	54.144181	4.855819	15.000000	12.302323	-15.080101
2023-11-09	53.0	55.805048	2.805048	11.666667	13.225027	-16.219418
2023-11-10	53.0	55.170736	2.170736	11.666667	12.872631	-16.571813

16022 rows × 6 columns

In [ ]:

```
Functional_Backtesting
```

Out[ ]:

DATE	actual_F	prediction_F	diff_F
1979-12-30	43.0	50.229396	7.229396
1979-12-31	42.0	43.673846	1.673846
1980-01-01	41.0	41.579185	0.579185
1980-01-02	36.0	43.961961	7.961961
1980-01-03	30.0	40.204809	10.204809
...	...	...	...
2023-11-06	63.0	57.038346	5.961654
2023-11-07	53.0	65.326408	12.326408
2023-11-08	59.0	54.144923	4.855077
2023-11-09	53.0	55.805145	2.805145
2023-11-10	53.0	55.171600	2.171600

16022 rows × 3 columns

In [ ]:

```
Functional_Backtesting
Functional_Backtesting.mean()
Functional_Backtesting['Actual celcius_F'] = Functional_Backtesting['actual_F'].apply(fahrenhit_to_celcius)
Functional_Backtesting['Prediction in celcius_F'] = Functional_Backtesting['prediction_F'].apply(fahrenhit_to_celcius
Functional_Backtesting['prediction_celcius_diff_F'] = Functional_Backtesting['diff_F'].apply(fahrenhit_to_celcius)
# Functional_Backtesting['prediction_celcius_diff']=Functional_Backtesting['prediction_celcius_diff']
Functional_Backtesting
# Functional_Backtesting["Temperature_celcius_diff"].rename()
from sklearn.metrics import mean_absolute_error
x =mean_absolute_error(Functional_Backtesting["actual_F"],Functional_Backtesting["prediction_F"])
# print(f"In Celcius :  {fahrenhit_to_celcius(x)}")
# print(f"Fahrenheit : {x}")
# mean_absolute_error(Functional_Backtesting["Prediction in celcius"])
print(f"{x}")
Functional_Backtesting
```

```
5.136267426300147
```

Out[ ]:

DATE	actual_F	prediction_F	diff_F	Actual celcius_F	Prediction in celcius_F	prediction_celcius_diff_F
1979-12-30	43.0	50.229396	7.229396	6.111111	10.127442	-13.761447
1979-12-31	42.0	43.673846	1.673846	5.555556	6.485470	-16.847863
1980-01-01	41.0	41.579185	0.579185	5.000000	5.321769	-17.456008
1980-01-02	36.0	43.961961	7.961961	2.222222	6.645534	-13.354466
1980-01-03	30.0	40.204809	10.204809	-1.111111	4.558227	-12.108439
...	...	...	...	...	...	...
2023-11-06	63.0	57.038346	5.961654	17.222222	13.910192	-14.465748
2023-11-07	53.0	65.326408	12.326408	11.666667	18.514671	-10.929774
2023-11-08	59.0	54.144923	4.855077	15.000000	12.302735	-15.080513
2023-11-09	53.0	55.805145	2.805145	11.666667	13.225081	-16.219364
2023-11-10	53.0	55.171600	2.171600	11.666667	12.873111	-16.571333

16022 rows × 6 columns

In [ ]:

```
Functional_Backtesting.apply(pd.isnull).sum()
```

Out[ ]:

```
actual_F                     0
prediction_F                 0
diff_F                       0
Actual celcius_F             0
Prediction in celcius_F      0
prediction_celcius_diff_F    0
dtype: int64
```

In [ ]:

```
predictions.apply(pd.isnull).sum()
```

Out[ ]:

```
actual                     0
prediction                 0
diff                       0
Acutal celcius             0
Prediction in celcius      0
prediction_celcius_diff    0
dtype: int64
```

In [ ]:
```
# predictions["Comparing_actual"]=Functional_Backtesting['actual']
# predictions["Comparing_prediction"]=Functional_Backtesting['prediction']
# predictions["Comparing_diff"]=Functional_Backtesting['diff']
# predictions.combine()
```

In [ ]:
```
# RE_order=pd.DataFrame({
#     'A':predictions['actual'],
#     'B':predictions['prediction'],
#     'C':predictions['diff'],
#     'D':predictions['Acutal celcius'],
#     'E':predictions['Prediction in celcius'],
#     'F':predictions['Comparing_prediction'],
#     'G':predictions['Comparing_actual'],
#     'H':predictions['Comparing_diff']
# })
# new_order=['A','B','C','G','F','H','D','E']
# predictions=predictions.reindex(columns=new_order)
# predictions
```

In [ ]:
```
def pct_diff(old,new):
    return (new - old) / old
def compute_rolling(weather,horizon,col):
    label=f"rolling_{horizon}_{col}"

    weather[label] = weather[col].rolling(horizon).mean()
    weather[f"{label}_percentage"] = pct_diff(weather[label] , weather[col])
    return weather
rolling_horizons=[3,4]
# weather
for horizon in rolling_horizons:
    for col in ["tmax" , "tmin" , "prcp"]:
        weather = compute_rolling(weather,horizon,col)
```

weather

In [ ]:
```
weather
```

Out[ ]:

DATE	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
1970-01-01	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	28	22	31.0	NaN	NaN	NaN	
1970-01-02	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	22	38.0	NaN	NaN	NaN	
1970-01-03	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	38	25	31.0	32.333333	0.175258	23.000000	
1970-01-04	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	31	23	35.0	33.333333	-0.070000	23.333333	
1970-01-05	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	35	21	36.0	34.666667	0.009615	23.000000	
...	...	...	...	...	...	...	...	...	...	...	...	
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0	59.666667	-0.061453	43.333333	
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0	61.333333	0.027174	46.333333	
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0	57.333333	-0.075581	45.333333	
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0	58.333333	0.011429	43.666667	

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
DATE												
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	53.0	55.000000	-0.036364	40.000000	

19672 rows × 20 columns

In [ ]:

```
weather=weather.iloc[14:,:]
weather
```

Out[ ]:

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
DATE												
1970-01-15	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	29	13	36.0	29.666667	-0.022472	18.000000	
1970-01-16	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	36	21	43.0	30.333333	0.186813	16.666667	
1970-01-17	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	43	30	42.0	36.000000	0.194444	21.333333	
1970-01-18	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.10	0.0	0.0	42	25	25.0	40.333333	0.041322	25.333333	
1970-01-19	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	25	16	24.0	36.666667	-0.318182	23.666667	
...	...	...	...	...	...	...	...	...	...	...	...	
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0	59.666667	-0.061453	43.333333	
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0	61.333333	0.027174	46.333333	
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0	57.333333	-0.075581	45.333333	
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0	58.333333	0.011429	43.666667	

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
DATE												
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	53.0	55.000000	-0.036364	40.000000	

19658 rows × 20 columns

In [ ]:
```
weather = weather.fillna(0)
weather
```

Out[ ]:

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
DATE												
1970-01-15	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	29	13	36.0	29.666667	-0.022472	18.000000	
1970-01-16	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	36	21	43.0	30.333333	0.186813	16.666667	
1970-01-17	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	43	30	42.0	36.000000	0.194444	21.333333	
1970-01-18	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.10	0.0	0.0	42	25	25.0	40.333333	0.041322	25.333333	
1970-01-19	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	25	16	24.0	36.666667	-0.318182	23.666667	
...	...	...	...	...	...	...	...	...	...	...	...	
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0	59.666667	-0.061453	43.333333	
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0	61.333333	0.027174	46.333333	
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0	57.333333	-0.075581	45.333333	
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0	58.333333	0.011429	43.666667	

DATE	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	rolling_3_tmin	rolling_3_tm
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	53.0	55.000000	-0.036364	40.000000	

19658 rows × 20 columns

In [ ]:
```
def expand_mean(df):
    return df.expanding(1).mean()
for col in ["tmax","tmin","prcp"]:
    weather[f"month_avg_{col}"]=weather[col].groupby(weather.index.month,group_keys=False).apply(expand_mean)
    weather[f"day_avg_{col}"]=weather[col].groupby(weather.index.day_of_year,group_keys=False).apply(expand_mean)
```

In [ ]:
```
weather
```

Out[ ]:

DATE	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	...	rolling_4_tmin	rolling_4
1970-01-15	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	29	13	36.0	29.666667	-0.022472	...	19.50	
1970-01-16	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	36	21	43.0	30.333333	0.186813	...	18.75	
1970-01-17	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.02	0.0	0.0	43	30	42.0	36.000000	0.194444	...	20.00	
1970-01-18	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.10	0.0	0.0	42	25	25.0	40.333333	0.041322	...	22.25	
1970-01-19	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	25	16	24.0	36.666667	-0.318182	...	23.00	
...	...	...	...	...	...	...	...	...	...	...	...	...	
2023-11-06	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	56	43	63.0	59.666667	-0.061453	...	42.25	
2023-11-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	63	53	53.0	61.333333	0.027174	...	45.75	
2023-11-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	40	59.0	57.333333	-0.075581	...	44.75	
2023-11-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	38	53.0	58.333333	0.011429	...	43.50	

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	...	rolling_4_tmin	rolling_4
DATE													
2023-11-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	53	42	53.0	55.000000	-0.036364	...	43.25	

19658 rows × 26 columns

In [ ]:
```
predictors = weather.columns[~weather.columns.isin(["target","name","station"])]
predictors
```

Out[ ]:
```
Index(['prcp', 'snow', 'snwd', 'tmax', 'tmin', 'rolling_3_tmax',
       'rolling_3_tmax_percentage', 'rolling_3_tmin',
       'rolling_3_tmin_percentage', 'rolling_3_prcp',
       'rolling_3_prcp_percentage', 'rolling_4_tmax',
       'rolling_4_tmax_percentage', 'rolling_4_tmin',
       'rolling_4_tmin_percentage', 'rolling_4_prcp',
       'rolling_4_prcp_percentage', 'month_avg_tmax', 'day_avg_tmax',
       'month_avg_tmin', 'day_avg_tmin', 'month_avg_prcp', 'day_avg_prcp'],
      dtype='object')
```

In [ ]:
```
predictions=backt_esting(weather,rr,predictors)
mean_absolute_error(predictions["actual"] , predictions["prediction"])
```

Out[ ]:
```
4.786248018944604
```

In [ ]:
```
# predictions.sort_values("diff",ascending=False)
```

In [ ]:
```
predictions['Acutal celcius'] = predictions['actual'].apply(fahrenhit_to_celcius)
predictions['Prediction in celcius'] = predictions['prediction'].apply(fahrenhit_to_celcius)
# predictions['prediction_celcius_diff'] = predictions['diff'].apply(fahrenhit_to_celcius)
# predictions['prediction_celcius_diff']=predictions['prediction_celcius_diff']
predictions
```

Out[ ]:

DATE	actual	prediction	diff	Acutal celcius	Prediction in celcius
1980-01-13	54.0	32.720164	21.279836	12.222222	0.400091
1980-01-14	51.0	47.307542	3.692458	10.555556	8.504190
1980-01-15	45.0	48.243626	3.243626	7.222222	9.024237
1980-01-16	40.0	41.882836	1.882836	4.444444	5.490464
1980-01-17	41.0	41.439741	0.439741	5.000000	5.244301
...	...	...	...	...	...
2023-11-06	63.0	56.880504	6.119496	17.222222	13.822502
2023-11-07	53.0	62.678826	9.678826	11.666667	17.043792
2023-11-08	59.0	54.065672	4.934328	15.000000	12.258707
2023-11-09	53.0	55.173638	2.173638	11.666667	12.874243
2023-11-10	53.0	56.972335	3.972335	11.666667	13.873519

16008 rows × 5 columns

In [ ]:

```
predictions.sort_values("diff",ascending=True)
```

Out[ ]:

	actual	prediction	diff	Acutal celcius	Prediction in celcius
DATE					
1993-08-20	82.0	81.999263	0.000737	27.777778	27.777368
1982-08-12	79.0	78.998711	0.001289	26.111111	26.110395
2020-04-30	62.0	62.001573	0.001573	16.666667	16.667541
1991-07-23	89.0	89.003647	0.003647	31.666667	31.668693
2012-05-07	64.0	64.003983	0.003983	17.777778	17.779990
...	...	...	...	...	...
1985-04-18	84.0	58.515796	25.484204	28.888889	14.730998
1997-02-26	71.0	45.201305	25.798695	21.666667	7.334058
2007-03-26	78.0	50.775433	27.224567	25.555556	10.430796
1998-03-26	80.0	52.633673	27.366327	26.666667	11.463152
1990-03-12	85.0	54.405369	30.594631	29.444444	12.447427

16008 rows × 5 columns

In [ ]:

```
weather.loc["1990-03-07":"1990-03-17"]
```

Out[ ]:

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	...	rolling_4_tmin	rolling_4
DATE													
1990-03-07	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	2.0	32	14	39.0	33.666667	-0.049505	...	20.50	
1990-03-08	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	1.0	39	20	43.0	35.000000	0.114286	...	19.25	
1990-03-09	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	43	29	47.0	38.000000	0.131579	...	21.25	
1990-03-10	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.01	0.0	0.0	47	39	59.0	43.000000	0.093023	...	25.50	
1990-03-11	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.05	0.0	0.0	59	41	59.0	49.666667	0.187919	...	32.25	
1990-03-12	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	59	43	85.0	55.000000	0.072727	...	38.00	
1990-03-13	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	85	41	62.0	67.666667	0.256158	...	41.00	
1990-03-14	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	62	46	55.0	68.666667	-0.097087	...	42.75	
1990-03-15	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.00	0.0	0.0	55	43	62.0	67.333333	-0.183168	...	43.25	
1990-03-16	USW00094789	JFK INTERNATIONAL	0.00	0.0	0.0	62	48	61.0	59.666667	0.039106	...	44.50	

	station	name	prcp	snow	snwd	tmax	tmin	target	rolling_3_tmax	rolling_3_tmax_percentage	...	rolling_4_tmin	rolling_4
DATE													
		AIRPORT, NY US											
1990-03-17	USW00094789	JFK INTERNATIONAL AIRPORT, NY US	0.26	0.0	0.0	61	49	59.0	59.333333	0.028090	...	46.50	

11 rows × 26 columns

In [ ]:

```
(predictions["diff"].round().value_counts()).sort_index()
```

Out[ ]:

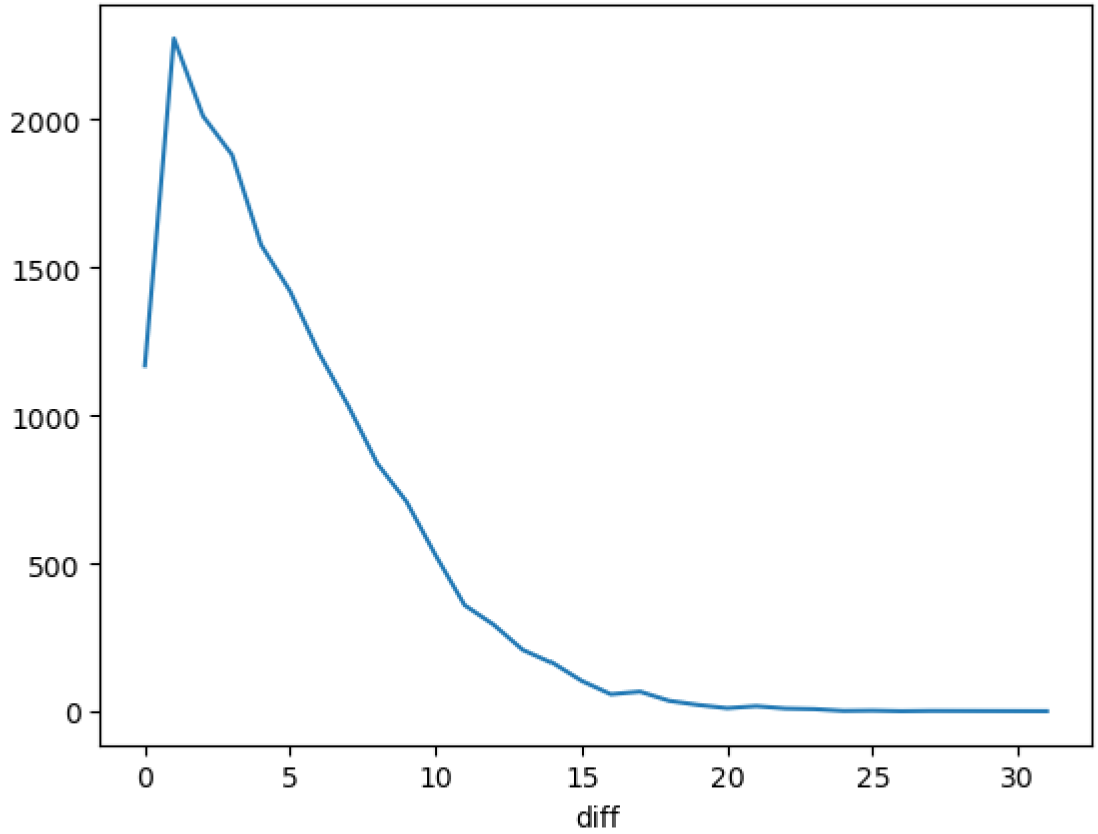
```
diff
0.0     1168
1.0     2273
2.0     2011
3.0     1880
4.0     1576
5.0     1419
6.0     1209
7.0     1032
8.0      837
9.0      709
10.0     528
11.0     359
12.0     293
13.0     208
14.0     164
15.0     103
16.0      58
17.0      67
18.0      36
19.0      22
20.0      11
21.0      18
22.0      10
23.0       8
24.0       2
25.0       3
26.0       1
27.0       2
31.0       1
Name: count, dtype: int64
```

In [ ]:

```
(predictions["diff"].round().value_counts()).sort_index().plot()
```

Out[ ]:

```
<Axes: xlabel='diff'>
```



In [ ]:
```
# from sklearn.metrics import accuracy_score
# accuracy_score(predictions["actual"].round(),predictions["prediction"].round())
# (predictions["diff"].round().value_counts()).sort_index().plot()
```

In [ ]:
```
#accuracy of the model
# (predictions["diff"].round().value_counts()).sort_index().sum() / predictions.shape[0] * 100
```

In [ ]:
```
# test = predictions["actual"] # ~ looks for all columns in the list except these columns
# pred = predictions["prediction"]
# test
```

In [ ]:
```
# pred
```

In [ ]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as  np
actual = predictions['actual']
prediction=predictions['prediction']
# Assuming 'actual' contains actual values and 'prediction' contains predicted values
mae = mean_absolute_error(actual, prediction)
mse = mean_squared_error(actual, prediction)
rmse = np.sqrt(mse)
r2 = r2_score(actual, prediction)

print(f"Mean Absolute Error: {mae}")
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")
```

```
Mean Absolute Error: 4.786248018944604
Mean Squared Error: 37.56933417066942
Root Mean Squared Error: 6.129382853980442
R-squared: 0.8741659251634186
```

In [ ]:
```
actual = Functional_Backtesting['actual_F']
prediction=Functional_Backtesting['prediction_F']
# Assuming 'actual' contains actual values and 'prediction' contains predicted values
mae_F = mean_absolute_error(actual, prediction)
mse_F = mean_squared_error(actual, prediction)
rmse_F = np.sqrt(mse)
r2_F = r2_score(actual, prediction)

print(f"Mean Absolute Error: {mae_F}")
print(f"Mean Squared Error: {mse_F}")
print(f"Root Mean Squared Error: {rmse_F}")
print(f"R-squared: {r2_F}")
```

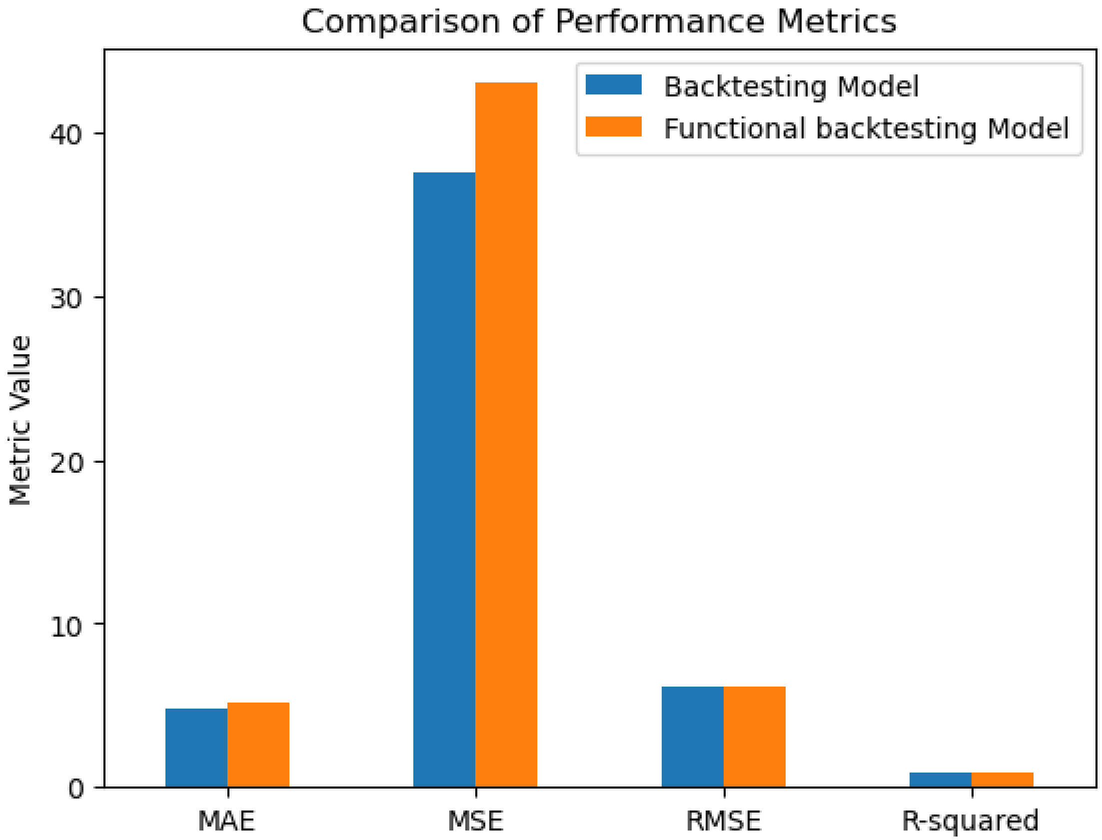
```
Mean Absolute Error: 5.136267426300147
Mean Squared Error: 42.98640103633378
Root Mean Squared Error: 6.129382853980442
R-squared: 0.8561719748777098
```

In [ ]:
```
print(f"MAE with backtesting                  : {mae}\nMAE with functional backtesting       : {mae_F}")
print(f"\nMSE with backtesting                  : {mse}\nMSE with functinoal backtesting       : {mse_F}")
print(f"\nRMSE with backtesting                 : {rmse}\nRMSE with functional backtesting      : {rmse_F}")
print(f"\nR-squared with backtesting            : {r2}\nR-squared with functional backtesting : {r2_F}")
```

```
MAE with backtesting                  : 4.786248018944604
MAE with functional backtesting       : 5.136267426300147

MSE with backtesting                  : 37.56933417066942
MSE with functinoal backtesting       : 42.98640103633378

RMSE with backtesting                 : 6.129382853980442
RMSE with functional backtesting      : 6.129382853980442

R-squared with backtesting            : 0.8741659251634186
R-squared with functional backtesting : 0.8561719748777098
```

In [ ]:

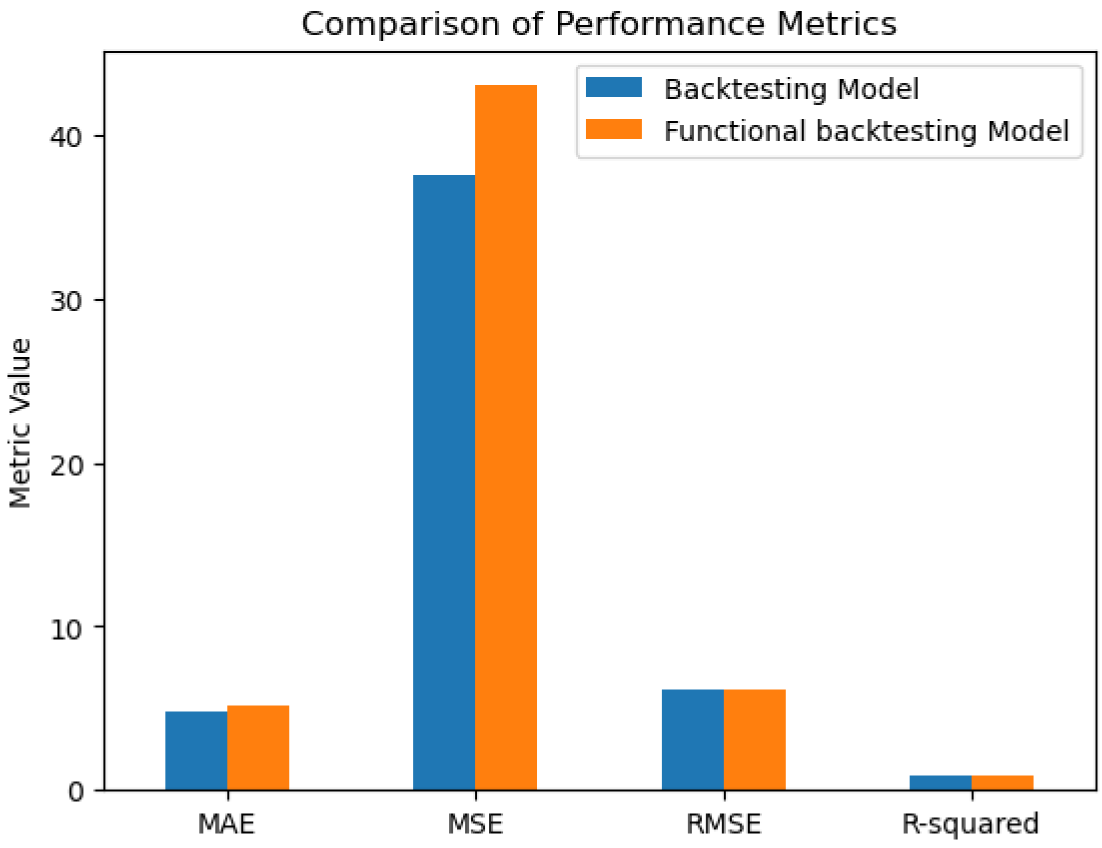
```
import matplotlib.pyplot as plt

# Performance metrics for the first set of predictions
metrics = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R-squared': r2}

# Performance metrics for the second set of predictions
metrics_F = {'MAE': mae_F, 'MSE': mse_F, 'RMSE': rmse_F, 'R-squared': r2_F}

# Combine the metrics for easy plotting
combined_metrics = pd.DataFrame({'Backtesting Model': metrics, 'Functional backtesting Model': metrics_F})

# Plotting the bar chart
combined_metrics.plot(kind='bar', rot=0)
plt.title('Comparison of Performance Metrics')
plt.ylabel('Metric Value')
plt.show()
```

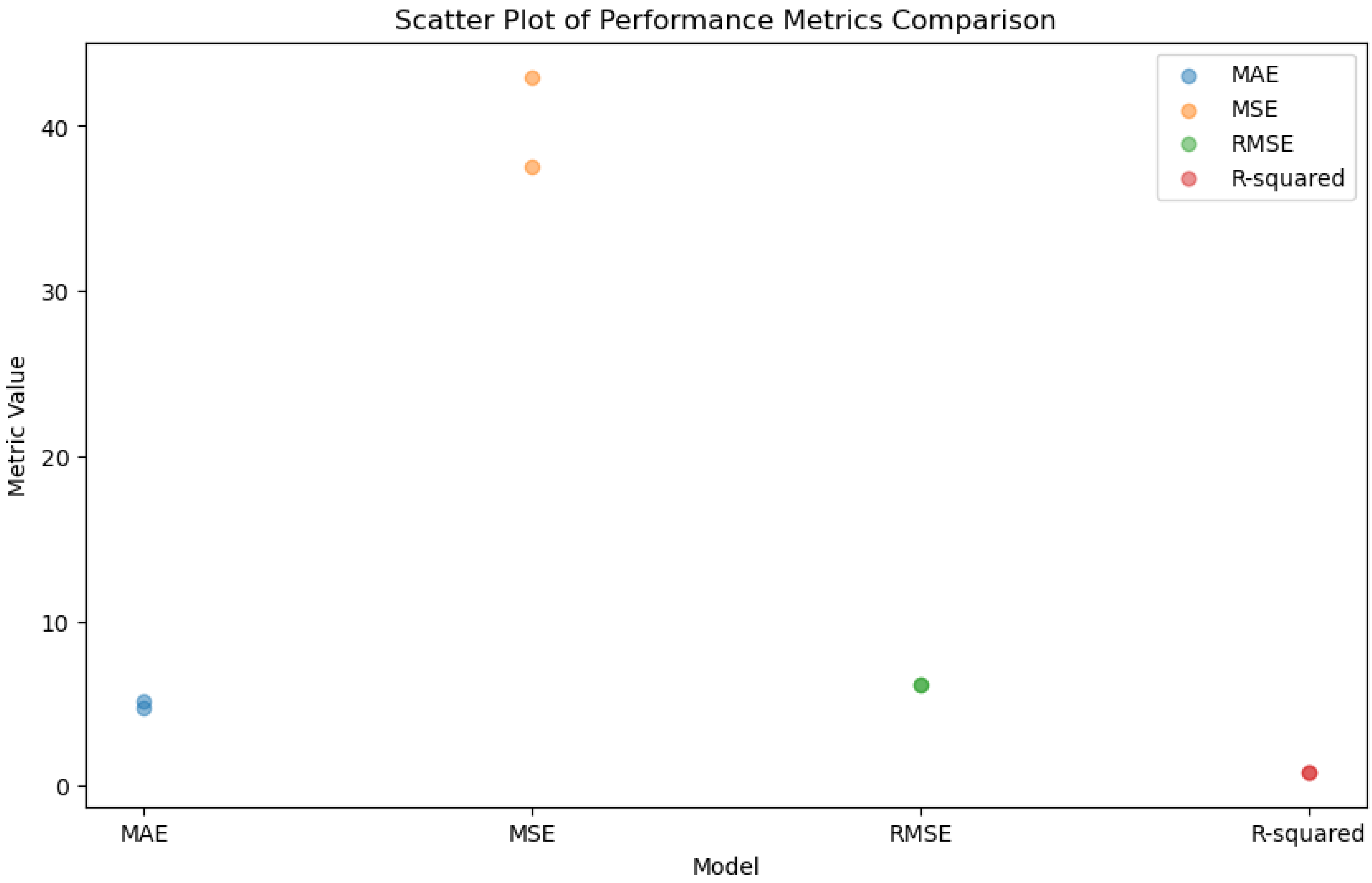


In [ ]:

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

# Performance metrics for the first set of predictions
metrics = {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R-squared': r2}

# Performance metrics for the second set of predictions
metrics_F = {'MAE': mae_F, 'MSE': mse_F, 'RMSE': rmse_F, 'R-squared': r2_F}

# Combine the metrics for easy plotting
combined_metrics = pd.DataFrame({'Backtesting Model': metrics, 'Functional backtesting Model': metrics_F})

# Plotting the bar chart
combined_metrics.plot(kind='bar', rot=0)
plt.title('Comparison of Performance Metrics')
plt.ylabel('Metric Value')
```

```
# Scatter plot for comparison
plt.figure(figsize=(10, 6))
for metric in metrics.keys():
    plt.scatter(x=[metric] * 2, y=[metrics[metric], metrics_F[metric]], label=f'{metric}', alpha=0.5)

plt.title('Scatter Plot of Performance Metrics Comparison')
plt.xlabel('Model')
plt.ylabel('Metric Value')
plt.legend()
plt.show()
```




Scatter Plot of Performance Metrics Comparison
MAE
MSE
RMSE
R-squared
40
30
20
10
0
Metric Value
MAE
MSE
RMSE
R-squared
Model