In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv(r"weatherHistory.csv")
df.head()
```

Out[2]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
0	2006-04-01 00:00:00.000 +0200	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263
1	2006-04-01 01:00:00.000 +0200	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263
2	2006-04-01 02:00:00.000 +0200	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569
3	2006-04-01 03:00:00.000 +0200	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263
4	2006-04-01 04:00:00.000 +0200	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263

In [3]:
```
df['Formatted Date']=df['Formatted Date'].apply(lambda x:x[:11])
```

In [4]:
```
df['Formatted Date'] = pd.to_datetime(df['Formatted Date'])
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 96453 entries, 0 to 96452
Data columns (total 12 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Formatted Date            96453 non-null  datetime64[ns]
 1   Summary                   96453 non-null  object
 2   Precip Type               95936 non-null  object
 3   Temperature (C)           96453 non-null  float64
 4   Apparent Temperature (C)  96453 non-null  float64
 5   Humidity                  96453 non-null  float64
 6   Wind Speed (km/h)         96453 non-null  float64
 7   Wind Bearing (degrees)    96453 non-null  float64
 8   Visibility (km)           96453 non-null  float64
 9   Loud Cover                96453 non-null  float64
 10  Pressure (millibars)      96453 non-null  float64
 11  Daily Summary             96453 non-null  object
dtypes: datetime64[ns](1), float64(8), object(3)
memory usage: 8.8+ MB
```

In [6]:
```
df[df['Precip Type'].isna()].head()
```

Out[6]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibi (k
52672	2012-04-11	Mostly Cloudy	NaN	19.016667	19.016667	0.26	14.8764	163.0	9.9
52674	2012-04-11	Mostly Cloudy	NaN	17.850000	17.850000	0.28	13.7977	169.0	9.9
52675	2012-04-11	Mostly Cloudy	NaN	16.322222	16.322222	0.32	10.8192	151.0	9.9
52677	2012-04-11	Mostly Cloudy	NaN	12.566667	12.566667	0.43	9.0160	159.0	9.9
52678	2012-04-11	Mostly Cloudy	NaN	12.927778	12.927778	0.47	17.6295	197.0	16.1

In [7]:
```
df[df['Precip Type']=='rain'].head()
```

Out[7]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
0	2006-04-01	Partly Cloudy	rain	9.472222	7.388889	0.89	14.1197	251.0	15.8263
1	2006-04-01	Partly Cloudy	rain	9.355556	7.227778	0.86	14.2646	259.0	15.8263
2	2006-04-01	Mostly Cloudy	rain	9.377778	9.377778	0.89	3.9284	204.0	14.9569
3	2006-04-01	Partly Cloudy	rain	8.288889	5.944444	0.83	14.1036	269.0	15.8263
4	2006-04-01	Mostly Cloudy	rain	8.755556	6.977778	0.83	11.0446	259.0	15.8263

In [8]:
```
df[df['Precip Type']=='snow'].head()
```

Out[8]:

	Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibili (kr
1562	2006-12-13	Foggy	snow	-0.483333	-4.150000	1.00	11.0929	219.0	0.4
1563	2006-12-13	Foggy	snow	-0.483333	-4.061111	0.96	10.7387	200.0	0.3
1564	2006-12-13	Foggy	snow	-0.922222	-3.477778	1.00	7.0679	206.0	0.1
1565	2006-12-13	Foggy	snow	-1.038889	-4.400000	1.00	9.4990	199.0	0.1
1566	2006-12-13	Foggy	snow	-1.088889	-4.438889	1.00	9.4346	219.0	0.3

In [9]:
```
df['Precip Type'] = df['Precip Type'].fillna('warm')
```

In [10]:
```
print('Number Of Missing Values: ',df.isna().sum().sum())
```

```
Number Of Missing Values:  0
```

In [11]:
```
df['year'] = df['Formatted Date'].dt.year
df['month'] = df['Formatted Date'].dt.month
df['day'] = df['Formatted Date'].dt.day
```

In [12]:
```
df[['Formatted Date','year','month','day']].head()
```

Out[12]:

	Formatted Date	year	month	day
0	2006-04-01	2006	4	1
1	2006-04-01	2006	4	1
2	2006-04-01	2006	4	1
3	2006-04-01	2006	4	1
4	2006-04-01	2006	4	1

In [13]:
```
y = df['Temperature (C)']
x = df.drop(columns =['Temperature (C)','Formatted Date'])
```

In [14]:
```
y.head()
```

Out[14]:
```
0    9.472222
1    9.355556
2    9.377778
3    8.288889
4    8.755556
Name: Temperature (C), dtype: float64
```

In [15]:
```
x.head()
```

Out[15]:

	Summary	Precip Type	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	D Summ
0	Partly Cloudy	rain	7.388889	0.89	14.1197	251.0	15.8263	0.0	1015.13	Pa clo through the
1	Partly Cloudy	rain	7.227778	0.86	14.2646	259.0	15.8263	0.0	1015.63	Pa clo through the
2	Mostly Cloudy	rain	9.377778	0.89	3.9284	204.0	14.9569	0.0	1015.94	Pa clo through the
3	Partly Cloudy	rain	5.944444	0.83	14.1036	269.0	15.8263	0.0	1016.41	Pa clo through the
4	Mostly Cloudy	rain	6.977778	0.83	11.0446	259.0	15.8263	0.0	1016.51	Pa clo through the

In [16]:
```
categorical_cols = x.select_dtypes(include='O')
numerical_cols = x.select_dtypes(include=[np.number])
```

In [17]:
```
categorical_cols.dtypes
```

Out[17]:
```
Summary          object
Precip Type      object
Daily Summary    object
dtype: object
```

In [18]:
```
numerical_cols.dtypes
```

Out[18]:
```
Apparent Temperature (C)    float64
Humidity                    float64
Wind Speed (km/h)           float64
Wind Bearing (degrees)      float64
Visibility (km)             float64
Loud Cover                  float64
Pressure (millibars)        float64
year                          int32
month                         int32
day                           int32
dtype: object
```

In [19]:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
encoded_arr = encoder.fit_transform(categorical_cols)
```

In [20]:
```
cols_name = categorical_cols.columns
cols_name
```

Out[20]:
```
Index(['Summary', 'Precip Type', 'Daily Summary'], dtype='object')
```

In [21]:
```
encoded_arr
```

Out[21]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [22]:
```
encoded_df = pd.DataFrame(
    encoded_arr,
    columns=encoder.get_feature_names_out(cols_name)
)
```

In [23]:
```
encoded_df.head()
```

Out[23]:

	Summary_Breezy	Summary_Breezy and Dry	Summary_Breezy and Foggy	Summary_Breezy and Mostly Cloudy	Summary_Breezy and Overcast	Summ
0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	

5 rows × 244 columns

In [24]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
numerical_cols.loc[:,:] = scaler.fit_transform(numerical_cols)
```

In [25]:
```
numerical_cols
```

Out[25]:

	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)	Loud Cover	Pressure (millibars)	year	mont
0	-0.324035	0.793470	0.478635	0.591256	1.306976	0.0	0.101685	-1.581343	-0.7318
1	-0.339097	0.639996	0.499594	0.665756	1.306976	0.0	0.105960	-1.581343	-0.7318
2	-0.138102	0.793470	-0.995473	0.153570	1.099586	0.0	0.108610	-1.581343	-0.7318
3	-0.459071	0.486521	0.476306	0.758881	1.306976	0.0	0.112628	-1.581343	-0.7318
4	-0.362469	0.486521	0.033841	0.665756	1.306976	0.0	0.113483	-1.581343	-0.7318
...	...	...	...	...	...	...	...	...	
96448	1.417400	-1.559811	0.026855	-1.457488	1.372265	0.0	0.095102	1.581087	0.7180
96449	1.283404	-1.304020	-0.103556	-1.559925	1.241686	0.0	0.101942	1.581087	0.7180
96450	1.045534	-0.894753	-0.264241	-1.466800	1.372265	0.0	0.106216	1.581087	0.7180
96451	0.997233	-0.690120	-0.040680	-1.559925	1.372265	0.0	0.108696	1.581087	0.7180
96452	0.895956	-0.638962	-0.713693	-1.382988	1.234005	0.0	0.110491	1.581087	0.7180

96453 rows × 10 columns

In [26]:
```
X = pd.merge(encoded_df,numerical_cols,left_index=True,right_index=True)
```

In [27]:
```
X.shape
```

Out[27]:
```
(96453, 254)
```

In [28]:
```
def plot_data(actual, predicted, method, deg=1):
    F = pd.DataFrame({
        "Actual": actual,
        "Predicted": predicted
    })

    coefficients = np.polyfit(F['Actual'], F['Predicted'], deg)

    poly_function = np.poly1d(coefficients)

    x_fit = np.linspace(F['Actual'].min(), F['Actual'].max(), 100)
    y_fit = poly_function(x_fit)

    plt.figure(figsize=(10, 6))
    plt.scatter(F['Actual'], F['Predicted'], label='Data Points')
    plt.plot(x_fit, y_fit, color='red', label=f'Fitted Polynomial (Degree {deg})'
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.title(method)
    plt.legend()
    plt.show()
```

Linear Regression

In [29]:
```
from sklearn.feature_selection import  SelectKBest,f_classif
kbest = SelectKBest(score_func=f_classif,k=20)
X = kbest.fit_transform(X,y)
```

In [30]:
```
X.shape
```

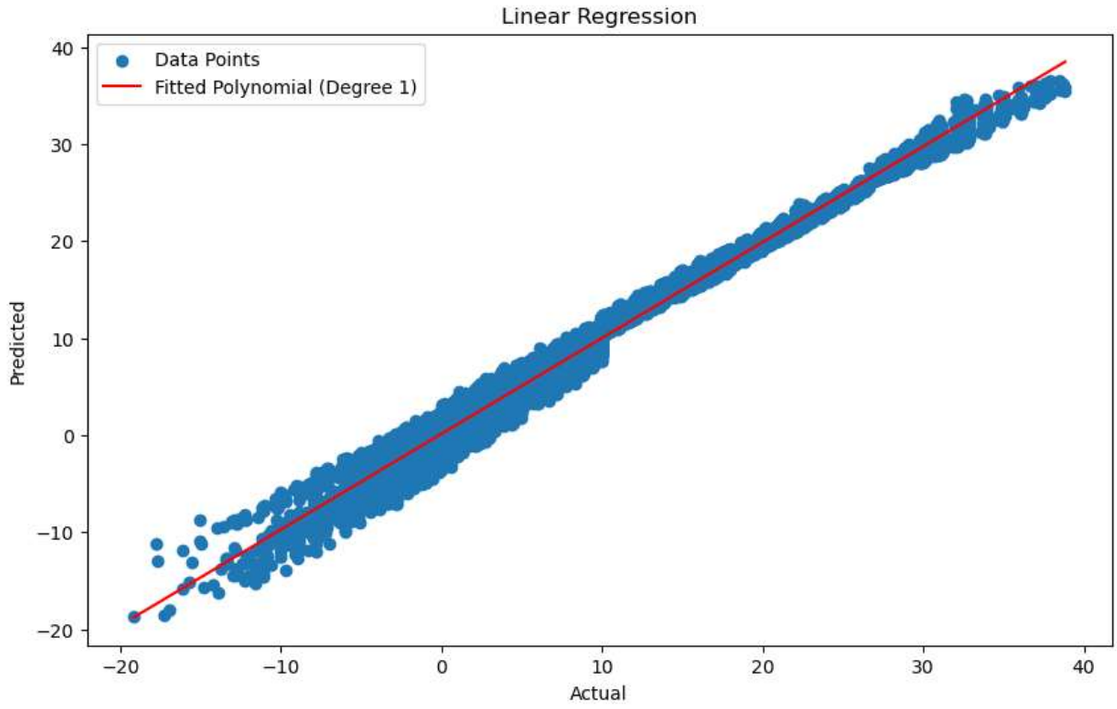
Out[30]:
```
(96453, 20)
```

In [31]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=.2,random_state=42
```

In [32]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
lr = LinearRegression()
lr.fit(x_train,y_train)
train_pred = lr.predict(x_train)
test_pred = lr.predict(x_test)
print("========Train Result========")
print(f"Mean Absolute Error : {mean_absolute_error(y_train,train_pred)}")
print(f"Mean Squared Error : {mean_squared_error(y_train,train_pred)}")
print(f"R Square Score : {r2_score(y_train,train_pred)}")
print("========Test Result========")
print(f"Mean Absolute Error : {mean_absolute_error(y_test,test_pred)}")
print(f"Mean Squared Error : {mean_squared_error(y_test,test_pred)}")
print(f"R Square Score : {r2_score(y_test,test_pred)}")
```

```
========Train Result========
Mean Absolute Error : 0.8198809941682601
Mean Squared Error : 1.1431260999458472
R Square Score : 0.9874379465582097
========Test Result========
Mean Absolute Error : 0.821629416044289
Mean Squared Error : 1.1433646334865846
R Square Score : 0.9875936021487471
```

In [33]:
```
plot_data(y_test,test_pred,method='Linear Regression')
```



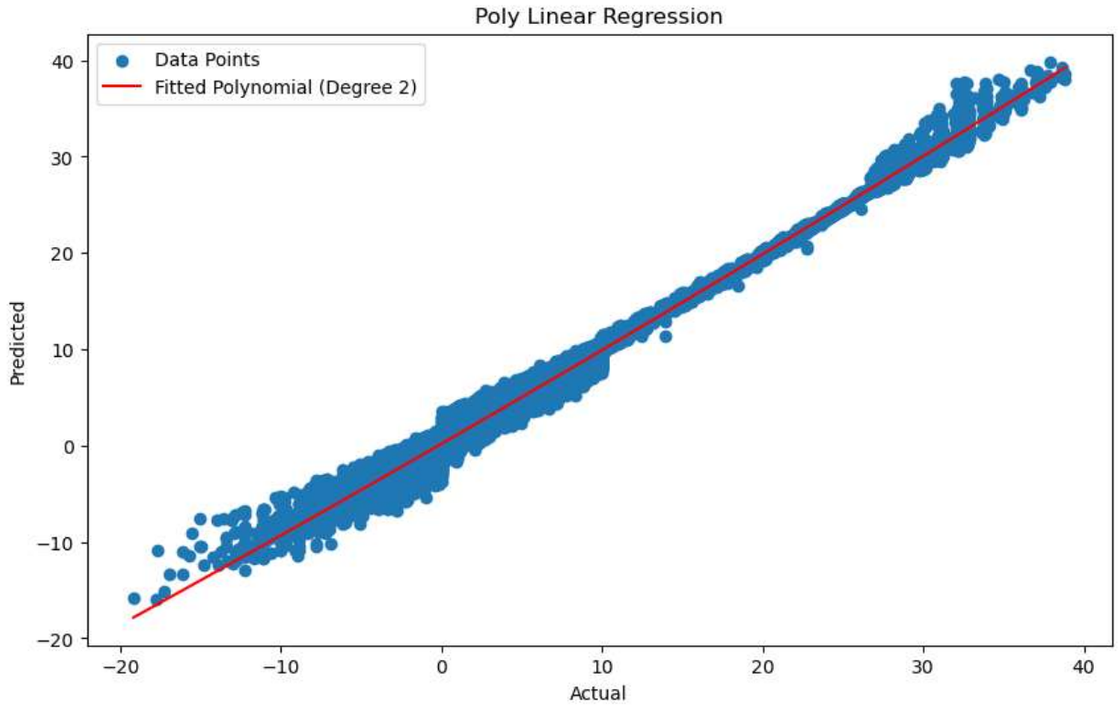
Poly Linear Regression

In [34]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_fet = PolynomialFeatures(degree=2)
x_train_poly = poly_fet.fit_transform(x_train)
x_test_poly = poly_fet.fit_transform(x_test)
```

In [35]:
```
lr = LinearRegression()
lr.fit(x_train_poly,y_train)
train_pred = lr.predict(x_train_poly)
test_pred = lr.predict(x_test_poly)
print("========Train Result========")
print(f"Mean Absolute Error : {mean_absolute_error(y_train,train_pred)}")
print(f"Mean Squared Error : {mean_squared_error(y_train,train_pred)}")
print(f"R Square Score : {r2_score(y_train,train_pred)}")
print("========Test Result========")
print(f"Mean Absolute Error : {mean_absolute_error(y_test,test_pred)}")
print(f"Mean Squared Error : {mean_squared_error(y_test,test_pred)}")
print(f"R Square Score : {r2_score(y_test,test_pred)}")
```

```
========Train Result========
Mean Absolute Error : 0.6157555736565525
Mean Squared Error : 0.8091026507181791
R Square Score : 0.9911086005833499
========Test Result========
Mean Absolute Error : 0.6231108738706855
Mean Squared Error : 0.8140282033175043
R Square Score : 0.9911671592274977
```

In [36]:
```
plot_data(y_test,test_pred,method='Poly Linear Regression',deg=2)
```



Deep Learning

In [37]:
```
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
model = Sequential([
    Dense(128,activation='relu',input_dim=X.shape[1]),
    Dense(64,activation='relu'),
    Dense(32,activation='relu'),
    Dense(1,activation=None)
])
```

In [38]:
```
from tensorflow.keras.metrics import RootMeanSquaredError
model.compile(
    optimizer='adam',
    loss=['mean_squared_error','binary_crossentropy'],
    metrics=[RootMeanSquaredError()]
)
```

In [39]:
```
model.fit(
    x_train,
    y_train,
    validation_data=(x_test,y_test),
    epochs=10,
    batch_size=32
)
```

```
Epoch 1/10
2412/2412 [==============================] - 3s 1ms/step - loss: 3.8208 - root_m
ean_squared_error: 1.9547 - val_loss: 0.7015 - val_root_mean_squared_error: 0.83
76
Epoch 2/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6835 - root_m
ean_squared_error: 0.8268 - val_loss: 0.6347 - val_root_mean_squared_error: 0.79
67
Epoch 3/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6612 - root_m
ean_squared_error: 0.8132 - val_loss: 0.6271 - val_root_mean_squared_error: 0.79
19
Epoch 4/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6560 - root_m
ean_squared_error: 0.8099 - val_loss: 0.6488 - val_root_mean_squared_error: 0.80
55
Epoch 5/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6416 - root_m
ean_squared_error: 0.8010 - val_loss: 0.6654 - val_root_mean_squared_error: 0.81
57
Epoch 6/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6324 - root_m
ean_squared_error: 0.7952 - val_loss: 0.6224 - val_root_mean_squared_error: 0.78
89
Epoch 7/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6217 - root_m
ean_squared_error: 0.7885 - val_loss: 0.6529 - val_root_mean_squared_error: 0.80
80
Epoch 8/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6169 - root_m
ean_squared_error: 0.7854 - val_loss: 0.5992 - val_root_mean_squared_error: 0.77
41
Epoch 9/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6139 - root_m
ean_squared_error: 0.7835 - val_loss: 0.6416 - val_root_mean_squared_error: 0.80
10
Epoch 10/10
2412/2412 [==============================] - 3s 1ms/step - loss: 0.6119 - root_m
ean_squared_error: 0.7822 - val_loss: 0.6237 - val_root_mean_squared_error: 0.78
97
```

Out[39]: <keras.src.callbacks.History at 0x21cbbcfc4f0>

In [40]:
```
plot_data(y_test,test_pred,method='Deep Learning')
```