In [54]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib
```

Importing the data using Pandas

In [55]:
```
tourism_data = pd.read_csv ("tanzania_tourism_stats.csv")
```

In [56]:
```
tourism_data.head(10) #Displays the first 10 rows of the dataset
```

Out[56]:

	Country	Month	Visitors	search interest	Average High Celsius	Average Low Celsius	Average Precipitation Days	Average Precipitation mm	Average Relative Humidity	Daily Mean Celsius
0	SWIZERLAND	jan	51772	646372	33.0	29.0	16	274	83	31.0
1	SWIZERLAND	feb	130947	631166	32.4	30.4	15	265	83	31.4
2	SWIZERLAND	march	20341	322640	34.0	31.0	13	230	83	NaN
3	SWIZERLAND	april	138966	1426600	34.2	31.2	9	95	84	32.7
4	SWIZERLAND	may	78025	917229	33.2	32.2	10	106	83	32.7
5	SWIZERLAND	june	138178	647755	31.6	31.6	8	145	84	NaN
6	SWIZERLAND	july	97158	891281	32.4	31.4	6	120	84	31.9
7	SWIZERLAND	aug	111152	2145048	29.6	29.2	11	190	83	29.4
8	SWIZERLAND	sept	23520	1823857	31.4	30.2	10	170	83	30.8
9	SWIZERLAND	oct	76874	2067533	33.6	31.4	12	215	83	32.5

Exploratory Data Analysis (EDA)

1. Columns with multiple categorical values require encoding
2. Some columns may need renaming (though, not necessary)

In [57]:
```
tourism_data.shape
```

Out[57]:
```
(1296, 10)
```

In [58]:
```
tourism_data.info() #checking for NULL values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296 entries, 0 to 1295
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Country                    1296 non-null   object
 1   Month                      1296 non-null   object
 2   Visitors                   1296 non-null   int64
 3   search interest            1296 non-null   int64
 4   Average High Celsius       1296 non-null   float64
 5   Average Low Celsius        1296 non-null   float64
 6   Average Precipitation Days 1296 non-null   int64
 7   Average Precipitation mm   1296 non-null   int64
 8   Average Relative Humidity  1296 non-null   int64
 9   Daily Mean Celsius         1249 non-null   float64
dtypes: float64(3), int64(5), object(2)
memory usage: 101.4+ KB
```

As can be observer, the Daily Mean Celsius column has some missing values

In [59]:
```
tourism_data.isna().sum() #Count NaN or missing values
```

Out[59]:
```
Country                      0
Month                        0
Visitors                     0
search interest              0
Average High Celsius         0
Average Low Celsius          0
Average Precipitation Days   0
Average Precipitation mm     0
Average Relative Humidity    0
Daily Mean Celsius           47
dtype: int64
```

Rename columns

In [60]:
```
tourism_data.rename(columns = {'search interest':'search_interest', 'Average High Celsius':'avg_high_celsius',
                               'Average Low Celsius':'avg_low_celsius', 'Average Precipitation Days': 'avg_ppt_days',
                               'Average Precipitation mm':'avg_ppt_mm', 'Average Relative Humidity': 'avg_relative_humidity',
                              'Daily Mean Celsius':'daily_mean_celsius'}, inplace = True)
```

In [61]:
```
tourism_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296 entries, 0 to 1295
```

```
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Country               1296 non-null   object
 1   Month                 1296 non-null   object
 2   Visitors              1296 non-null   int64
 3   search_interest       1296 non-null   int64
 4   avg_high_celsius      1296 non-null   float64
 5   avg_low_celsius       1296 non-null   float64
 6   avg_ppt_days          1296 non-null   int64
 7   avg_ppt_mm            1296 non-null   int64
 8   avg_relative_humidity 1296 non-null   int64
 9   daily_mean_celsius    1249 non-null   float64
dtypes: float64(3), int64(5), object(2)
memory usage: 101.4+ KB
```

In [62]:
```
#Find columns with missing values
cols_missing_values = tourism_data.columns[tourism_data.isnull().any()].tolist()  # to get a list instead of an Index object
print (cols_missing_values)

#count columns with missing values
print ("\nNumber of columns with missing values are: ", len(cols_missing_values))
```

```
['daily_mean_celsius']

Number of columns with missing values are:  1
```

We can observe that: Out of 10 columns in the dataset, 1 column contain missing values

Handle Missing values in columns

In [63]:
```
missing_daily_mean_celsius = tourism_data[tourism_data['daily_mean_celsius'].isnull()].index.tolist() #Index of columns with null
missing_daily_mean_celsius
```

Out[63]:
```
[2,
 5,
 10,
 17,
 22,
 26,
 33,
 39,
 44,
 146,
 290,
 440,
 603,
 809,
 903,
 1016,
 1055,
 1058,
 1063,
 1067,
 1077,
 1080,
 1082,
 1084,
 1086,
 1090,
 1092,
 1095,
 1098,
 1113,
 1114,
 1117,
 1121,
 1126,
 1133,
 1164,
 1167,
 1171,
 1185,
 1190,
 1192,
 1220,
 1227,
 1238,
 1244,
 1258,
 1276]
```

In [64]:
```
# Filling in missing Daily Mean Celsius values
for i in missing_daily_mean_celsius:
    #tourism_data.iloc[i] =
    res = (tourism_data['avg_high_celsius'][i] + tourism_data['avg_low_celsius'][i])*0.5
    res = round(res, 1)
    tourism_data['daily_mean_celsius'][i] = res
```

```
C:\Users\Nassor\AppData\Local\Temp\ipykernel_7716\1411110861.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  tourism_data['daily_mean_celsius'][i] = res
```

In [66]:
```
#Verifying the value changes
print(tourism_data.iloc[10])
print(tourism_data.iloc[146])
```

```
Country                 SWIZERLAND
```

```
Month                          nov
Visitors                     31187
search_interest             848166
avg_high_celsius              32.7
avg_low_celsius               30.4
avg_ppt_days                    12
avg_ppt_mm                     215
avg_relative_humidity           84
daily_mean_celsius            31.6
Name: 10, dtype: object
Country                   AUSTRIA
Month                       march
Visitors                   106347
search_interest            749651
avg_high_celsius             34.0
avg_low_celsius              31.0
avg_ppt_days                   13
avg_ppt_mm                    230
avg_relative_humidity          83
daily_mean_celsius           32.5
Name: 146, dtype: object
```

Check again missing values

In [67]:
```
tourism_data.isna().sum() #Count NaN or missing values
```

Out[67]:
```
Country                 0
Month                   0
Visitors                0
search_interest         0
avg_high_celsius        0
avg_low_celsius         0
avg_ppt_days            0
avg_ppt_mm              0
avg_relative_humidity   0
daily_mean_celsius      0
dtype: int64
```

Feature Engineering

One Hot Encoding

There are two categorical features (month and country) in the dataset, therefore I need to convert all categories from rows into columns. This technique is called One Hot Encoding.

In [69]:
```
countries = pd.get_dummies(tourism_data['Country'])
months = pd.get_dummies(tourism_data['Month'])

tourism_data = pd.concat([tourism_data, countries, months], axis=1)

#tourism_data = df_full.drop(df_full.columns[[0, 1]], axis=1)
tourism_data
```

Out[69]:

	Country	Month	Visitors	search_interest	avg_high_celsius	avg_low_celsius	avg_ppt_days	avg_ppt_mm	avg_relative_humidity	daily_mean_celsius
0	SWIZERLAND	jan	51772	646372	33.0	29.0	16	274	83	31.0
1	SWIZERLAND	feb	130947	631166	32.4	30.4	15	265	83	31.4
2	SWIZERLAND	march	20341	322640	34.0	31.0	13	230	83	32.5
3	SWIZERLAND	april	138966	1426600	34.2	31.2	9	95	84	32.7
4	SWIZERLAND	may	78025	917229	33.2	32.2	10	106	83	32.7
...	...	...	...	...	...	...	...	...	...	...
1291	GUINEA	aug	57775	1046312	29.6	29.2	11	190	83	29.4
1292	GUINEA	sept	135440	507198	31.4	30.2	10	170	83	30.8
1293	GUINEA	oct	141695	911982	33.6	31.4	12	215	83	32.5
1294	GUINEA	nov	68953	2451522	32.7	30.4	12	215	84	31.6
1295	GUINEA	dec	99575	1420575	33.0	32.4	14	225	83	32.7

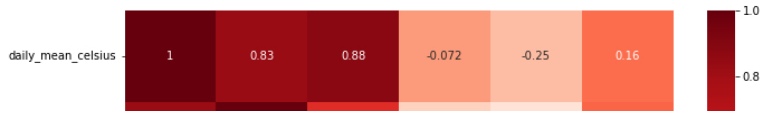
1296 rows × 130 columns

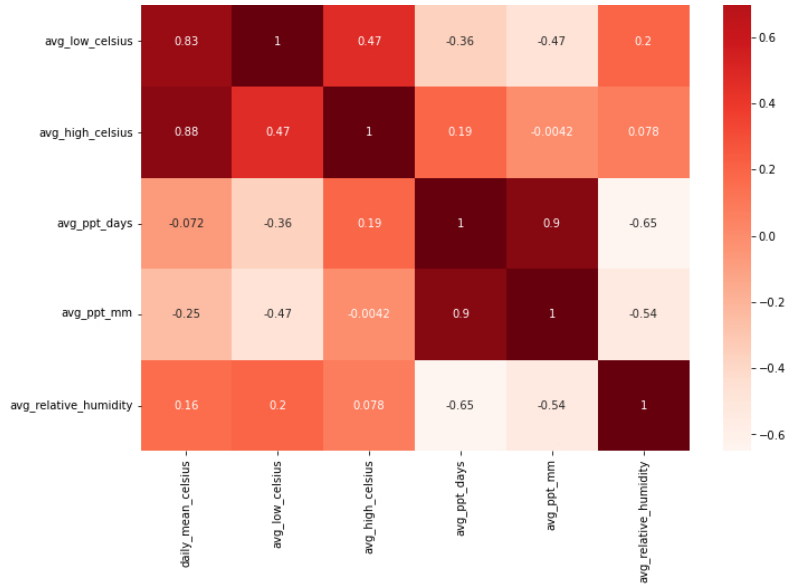
Feature Selection

Feature selection is important especially when you have too many features to begin with. By reducing the number of features hopefully it can also improve the performance of the model. There are 3 common methods of doing feature selection: Filter, wrapper, and embedded. For this project I choose a filter method by using a Pearson correlation matrix.

My goal in this step is to find features that are too similar to each other and keep only one of them. Below is the Pearson correlation heatmap to see the correlation of all features in the weather dataset.

In [73]:
```
plt.figure(figsize=(11,9))
cor = tourism_data[['daily_mean_celsius','avg_low_celsius','avg_high_celsius','avg_ppt_days',
       'avg_ppt_mm', 'avg_relative_humidity']].astype(str).astype(float).corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```





Since daily_mean_celsius, average_low_celsius, and average_high_celsius have strong positive correlation with each other, therefore I only keep daily_mean_celsius.

Same thing happens with average_precipitation_days, average_precipitation_mm, and average_relative_hummidity. 3 of them also have strong positive correlation so I decided only to keep average_precipitation_mm.

Train & Test Split

In [78]:
```
from sklearn.model_selection import train_test_split
```

In [96]:
```
LIA','CHINA','INDIA','JAPAN','MALAYSIA', 'UNITED STATES OF AMERICA', 'april', 'aug', 'dec', 'feb','jan', 'july', 'june', 'march',



))
```

Build ML Model

For building a prediction model, I choose the Gradient Boosting and Random Forest algorithm. Both are an ensemble method, a technique that creates multiple models and then combines them to produce better results, in this case both Gradient Boosting and Random Forest generated multiple decision trees. So what is the difference between them? Gradient boosting using a method called "boosting" while random forest used a method called "bagging". Boosting method is running iteratively, it evaluates what it gets right and what it gets wrong on that first tree, and then with the next iteration it places a heavier weight on those observations that it got wrong, this process continues until it has minimized the error as much as possible. The bagging method builds decision-trees simultaneously and later uses a voting method to combine the predictions of each.

In order to build the best model, I need to find the best value for each parameter of the model, this is something that is famously known as hyperparameter optimization. Usually, the value of hyperparameters is chosen randomly and then pick the hyperparameters value with the best accuracy result. But it can be a very exhausting process especially if there is more than one hyperparameter in one model, therefore it is better to use an algorithm to find the best hyperparameter combination automatically such as grid-search. By using scikit-learn library, the Grid Search algorithm can be implemented by importing a class called GridSearchCV.

In [86]:
```
# Parameters we want to try
param_grid = {
    'n_estimators': [100, 500, 1000, 3000],
    'max_depth': [5, 10, 30, 60, 100, None]

}
```

In [87]:
```
from sklearn.model_selection import GridSearchCV
from sklearn import ensemble
```

In [88]:
```
# Define the grid search we want to run. For K-Fold cross validation I use K=5
gs_cv_gradient = GridSearchCV(ensemble.GradientBoostingRegressor(), param_grid, n_jobs=-1, cv = 5)
gs_cv_randomf = GridSearchCV(ensemble.RandomForestRegressor(), param_grid, n_jobs=-1, cv = 5)
```

In [89]:
```
# Run the grid search on the training data.
gs_cv_gradient.fit(X_train, y_train)
gs_cv_randomf.fit(X_train, y_train)
```

Out[89]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'max_depth': [5, 10, 30, 60, 100, None],
                         'n_estimators': [100, 500, 1000, 3000]})
```

In [90]:
```
gs_cv_gradient.best_params_
```

Out[90]:
```
{'max_depth': 5, 'n_estimators': 100}
```

In [92]:
```
gs_cv_randomf.best_params_
```

Out[92]:
```
{'max_depth': 5, 'n_estimators': 1000}
```

Evaluation

To evaluate this model I calculate R-squared (R2) and mean absolute error (MAE). Both of this measurement are calculated using scikit-learn:

In [93]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

In [94]:
```
print('Mean Absolute Error for test set using Gradien Boost: '+str(mean_absolute_error(y_test, gs_cv_gradient.predict(X_test))))
print('r2 score Error for test set using Gradien Boost: '+str(r2_score(y_test, gs_cv_gradient.predict(X_test))))
print('Mean Absolute Error for test set using Random Forest: '+str(mean_absolute_error(y_test, gs_cv_randomf.predict(X_test))))
print('r2 score Error for test set using Random Forest: '+str(r2_score(y_test, gs_cv_randomf.predict(X_test))))
```

```
Mean Absolute Error for test set using Gradien Boost: 33544.98404302546
r2 score Error for test set using Gradien Boost: -0.21237808410440917
Mean Absolute Error for test set using Random Forest: 32106.156787919885
r2 score Error for test set using Random Forest: -0.060536505637792226
```

Feature Importance

Since both models are tree-based machine learning algorithms, we can actually rank all features based on how often they are used in determining the dependent variable.

In [95]:
```
import numpy as np
```

In [97]:
```
# These are the feature labels from our data set
feature_labels = np.array(['search_interest','avg_ppt_mm','daily_mean_celsius','AUSTRALIA','CHINA','INDIA','JAPAN','MALAYSIA', 'U

# Create a numpy array based on the model's feature importances
importance = gs_cv_gradient.best_estimator_.feature_importances_

# Sort the feature labels based on the feature importance rankings from the model
feauture_indexes_by_importance = importance.argsort()

# Print each feature label, from most important to least important (reverse order)
for index in feauture_indexes_by_importance:
    print("{} - {:.2f}%".format(feature_labels[index], (importance[index] * 100.0)))
```

```
CHINA - 0.03%
july - 0.30%
aug - 0.44%
april - 0.49%
may - 0.56%
JAPAN - 0.61%
AUSTRALIA - 0.62%
sept - 0.67%
jan - 0.89%
feb - 0.97%
INDIA - 0.98%
UNITED STATES OF AMERICA - 1.00%
nov - 1.11%
MALAYSIA - 1.12%
dec - 1.25%
oct - 1.39%
march - 1.63%
june - 1.72%
daily_mean_celsius - 4.07%
avg_ppt_mm - 5.61%
search_interest - 74.56%
```

In [98]:
```
# These are the feature labels from our data set
feature_labels = np.array(['search_interest','avg_ppt_mm','daily_mean_celsius','AUSTRALIA','CHINA','INDIA','JAPAN','MALAYSIA', 'U

# Create a numpy array based on the model's feature importances
importance = gs_cv_randomf.best_estimator_.feature_importances_

# Sort the feature labels based on the feature importance rankings from the model
feauture_indexes_by_importance = importance.argsort()

# Print each feature label, from most important to least important (reverse order)
for index in feauture_indexes_by_importance:
    print("{} - {:.2f}%".format(feature_labels[index], (importance[index] * 100.0)))
```

```
CHINA - 0.09%
JAPAN - 0.36%
aug - 0.38%
sept - 0.46%
jan - 0.62%
INDIA - 0.64%
july - 0.75%
april - 0.82%
dec - 0.86%
may - 0.91%
AUSTRALIA - 1.00%
MALAYSIA - 1.06%
oct - 1.16%
nov - 1.21%
march - 2.19%
june - 2.34%
UNITED STATES OF AMERICA - 2.66%
feb - 3.16%
daily_mean_celsius - 5.30%
avg_ppt_mm - 6.92%
search_interest - 67.13%
```

In [ ]: