```
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from pathlib import Path
import matplotlib.pyplot as plt # import matplotlib
%matplotlib inline
import seaborn as sns # seaborn data visualizer
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
d12.drop(d12[d12['HR']<47].index, inplace = True)
d12.reset_index(inplace = True)
d12
```

	index	YEAR	MN	HR	DT	SLP	MSLP	DBT	WBT	DPT	RH	VP	DD	FFF	AW
0	31	1985	1	48	1	931.7	1008.6	23.6	22.0	21.3	87	25.3	0	0	0
1	32	1985	1	48	2	934.1	1011.7	21.8	21.0	20.6	93	24.3	0	0	0
2	33	1985	1	48	3	935.8	1013.6	21.2	20.4	20.0	93	23.4	0	0	0
3	34	1985	1	48	4	935.0	1012.0	24.4	20.6	18.6	70	21.4	0	0	0
4	35	1985	1	48	5	934.1	1010.8	25.4	20.2	17.4	61	19.9	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5937	11891	2001	12	48	27	937.7	1014.8	25.0	20.0	17.3	62	19.7	14	4	8
5938	11892	2001	12	48	28	937.6	1014.7	24.4	21.0	19.3	73	22.4	14	4	6
5939	11893	2001	12	48	29	936.0	1013.0	25.0	19.0	15.5	56	17.6	14	4	6
5940	11894	2001	12	48	30	936.2	1013.4	23.8	19.2	16.6	64	18.9	14	4	4
5941	11895	2001	12	48	31	935.9	1013.1	23.4	18.8	16.1	64	18.3	14	4	4

5942 rows × 15 columns

```
#d12.drop(['YEAR'], axis = 1)
d12.drop(d12.columns[[0,1, 2, 3,4,14]], axis = 1, inplace = True)
d12
```

	SLP	MSLP	DBT	WBT	DPT	RH	VP	DD	FFF
0	931.7	1008.6	23.6	22.0	21.3	87	25.3	0	0
1	934.1	1011.7	21.8	21.0	20.6	93	24.3	0	0
2	935.8	1013.6	21.2	20.4	20.0	93	23.4	0	0
3	935.0	1012.0	24.4	20.6	18.6	70	21.4	0	0
4	934.1	1010.8	25.4	20.2	17.4	61	19.9	0	0
...	...	...	...	...	...	...	...	...	...
5937	937.7	1014.8	25.0	20.0	17.3	62	19.7	14	4
5938	937.6	1014.7	24.4	21.0	19.3	73	22.4	14	4
5939	936.0	1013.0	25.0	19.0	15.5	56	17.6	14	4
5940	936.2	1013.4	23.8	19.2	16.6	64	18.9	14	4
5941	935.9	1013.1	23.4	18.8	16.1	64	18.3	14	4

5942 rows × 9 columns

```
df = pd.concat([d11,d12],axis=1)
df
```

	YEAR	MN	DT	MAX	MIN	AW	RF	DRNRF(hrs)	DRNRF(mnts)	index	...	SLP	MSLP	DBT	WBT	DPT	RH	VP	DD	FFF	AW
0	1985	1	1	NaN	19.6	1	4.4			31.0	...	931.7	1008.6	23.6	22.0	21.3	87	25.3	0	0	0
1	1985	1	2	NaN	19.5	0	22.6	1	35	32.0	...	934.1	1011.7	21.8	21.0	20.6	93	24.3	0	0	0
2	1985	1	3	NaN	18.4	0	1.0	1	30	33.0	...	935.8	1013.6	21.2	20.4	20.0	93	23.4	0	0	0
3	1985	1	4	NaN	18.6	0	2.5			34.0	...	935.0	1012.0	24.4	20.6	18.6	70	21.4	0	0	0
4	1985	1	5	NaN	17.9	0	0.0	0	0	35.0	...	934.1	1010.8	25.4	20.2	17.4	61	19.9	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6106	2001	12	27	27.5	16.5	3	0.0	0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6107	2001	12	28	28.0	16.4	4	0.0	0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6108	2001	12	29	27.0	16.0	3	0.0	0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6109	2001	12	30	26.5	15.0	3	0.0	0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6110	2001	12	31	25.5	16.0	3	0.0	0	0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

6111 rows × 24 columns

```
#removing the null rows in data base
df=df.dropna()
```

```
features_list = list(df.drop(columns='RF').columns)
columns = list(df.columns)
print(features_list)
```

```
['YEAR', 'MN', 'DT', 'MAX', 'MIN', 'AW', 'DRNRF(hrs)', 'DRNRF(mnts)', 'index', 'YEAR', 'MN', 'HR', 'DT', 'SLP', 'MSLP', 'DBT', 'WBT', 'DPT', 'RH', 'VP', 'DD', 'FFF', 'AW']
```

```
m=[]
l=[]
for i in df['DRNRF(mnts)']:
    if(str(i).replace(" ",'')!=''):
        m+=[float(i)]
    else:
        m+=[0]
for i in df['DRNRF(hrs)']:
    if(str(i).replace(" ",'')!=''):
        l+=[float(i)*60]
    else:
        l+=[0]
for i in range(len(m)):
    m[i]+=l[i]

df['DRNF']=m

df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  app.launch_new_instance()
```

	YEAR	MN	DT	MAX	MIN	AW	RF	DRNRF(hrs)	DRNRF(mnts)	index	...	MSLP	DBT	WBT	DPT	RH	VP	DD	FFF	AW	DRNF
42	1985	2	15	32.7	18.5	5	0.0	0	0	100.0	...	1003.3	31.6	20.0	12.9	32	14.9	18	6	4	0.0
43	1985	2	16	31.8	19.1	1	0.0	0	0	101.0	...	1003.3	31.8	19.6	11.8	30	13.8	18	4	7	0.0
44	1985	2	17	30.6	18.6	2	0.0	0	0	102.0	...	1002.2	31.2	19.4	11.9	31	13.9	14	6	6	0.0
45	1985	2	18	29.8	17.6	5	0.0	0	0	103.0	...	1001.1	31.0	19.4	12.0	31	14	0	0	1	0.0
46	1985	2	19	28.8	17.4	3	0.0	0	0	104.0	...	1002.4	29.4	20.0	14.5	40	16.5	9	12	5	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
5937	2001	7	10	32.0	23.5	11	0.0	0	0	11891.0	...	1014.8	25.0	20.0	17.3	62	19.7	14	4	8	0.0
5938	2001	7	11	32.5	24.0	10	1.8	0	8	11892.0	...	1014.7	24.4	21.0	19.3	73	22.4	14	4	6	8.0
5939	2001	7	12	31.5	23.5	9	0.0	0	0	11893.0	...	1013.0	25.0	19.0	15.5	56	17.6	14	4	6	0.0
5940	2001	7	13	32.5	24.0	7	0.0	0	0	11894.0	...	1013.4	23.8	19.2	16.6	64	18.9	14	4	4	0.0
5941	2001	7	14	33.5	22.5	9	2.2	4	15	11895.0	...	1013.1	23.4	18.8	16.1	64	18.3	14	4	4	255.0

5787 rows × 25 columns

```
df=df.drop(columns=['DRNRF(hrs)', 'DRNRF(mnts)'])


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5787 entries, 42 to 5941
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   YEAR    5787 non-null   int64  
 1   MN      5787 non-null   int64  
 2   DT      5787 non-null   int64  
 3   MAX     5787 non-null   float64
 4   MIN     5787 non-null   float64
 5   AW      5787 non-null   object 
 6   RF      5787 non-null   float64
 7   index   5787 non-null   float64
 8   YEAR    5787 non-null   float64
 9   MN      5787 non-null   float64
 10  HR      5787 non-null   float64
 11  DT      5787 non-null   float64
 12  SLP     5787 non-null   float64
 13  MSLP    5787 non-null   float64
 14  DBT     5787 non-null   float64
 15  WBT     5787 non-null   float64
 16  DPT     5787 non-null   float64
 17  RH      5787 non-null   object 
 18  VP      5787 non-null   object 
 19  DD      5787 non-null   object 
 20  FFF     5787 non-null   object 
 21  AW      5787 non-null   object 
 22  DRNF    5787 non-null   float64
dtypes: float64(14), int64(3), object(6)
memory usage: 1.1+ MB
```

```
FFF=[]
AW=[]
RH=[]
VP=[]
DD=[]
for i in df['FFF']:
    if(str(i).replace(" ",'')!=''):
        FFF+=[float(i)]
    else:
        FFF+=[0]
for i in df['AW']:
    if(str(i).replace(" ",'')!='' and str(i)!='AW'):
        AW+=[float(i)]
    else:
        AW+=[0]
for i in df['RH']:
    if(str(i).replace(" ",'')!=''):
        RH+=[float(i)]
    else:
        RH+=[0]
for i in df['VP']:
    if(str(i).replace(" ",'')!=''):
        VP+=[float(i)]
    else:
        VP+=[0]
for i in df['DD']:
    if(str(i).replace(" ",'')!=''):
        DD+=[float(i)]
    else:
        DD+=[0]
df['FFF']=FFF
df['AW']=AW
df['RH']=RH
df['VP']=VP
df['DD']=DD
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/pandas/core/frame.py:3699: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[iloc] = igetitem(value, i)
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:34: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:35: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5787 entries, 42 to 5941
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   YEAR    5787 non-null   int64
 1   MN      5787 non-null   int64
 2   DT      5787 non-null   int64
 3   MAX     5787 non-null   float64
 4   MIN     5787 non-null   float64
 5   AW      5787 non-null   int64
 6   RF      5787 non-null   float64
 7   index   5787 non-null   float64
 8   YEAR    5787 non-null   float64
 9   MN      5787 non-null   float64
 10  HR      5787 non-null   float64
 11  DT      5787 non-null   float64
 12  SLP     5787 non-null   float64
 13  MSLP    5787 non-null   float64
 14  DBT     5787 non-null   float64
 15  WBT     5787 non-null   float64
 16  DPT     5787 non-null   float64
 17  RH      5787 non-null   float64
 18  VP      5787 non-null   float64
 19  DD      5787 non-null   float64
 20  FFF     5787 non-null   float64
 21  AW      5787 non-null   int64
 22  DRNF    5787 non-null   float64
dtypes: float64(18), int64(5)
memory usage: 1.1 MB
```

```
columns = list(df.columns)
print(columns)
```

```
['YEAR', 'MN', 'DT', 'MAX', 'MIN', 'AW', 'RF', 'index', 'YEAR', 'MN', 'HR', 'DT', 'SLP', 'MSLP', 'DBT', 'WBT', 'DPT', 'RH', 'VP', 'DD', 'FFF', 'AW', 'DRNF']
```

```
isnull = df.isnull().sum()
isnull
```

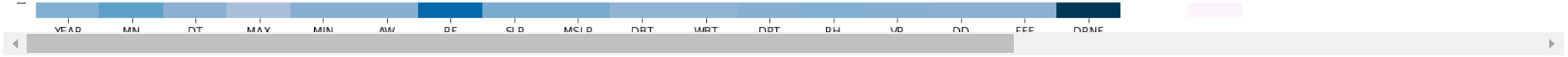
```
YEAR     0
MN       0
DT       0
MAX      0
MIN      0
AW       0
RF       0
index    0
YEAR     0
MN       0
HR       0
DT       0
SLP      0
```

```
MSLP     0
DBT      0
WBT      0
DPT      0
RH       0
VP       0
DD       0
FFF      0
AW       0
DRNF     0
dtype: int64
```

```
def standardize_var(x):
    mean = np.mean(x)
    std = np.sqrt(np.sum(np.square(x-mean))/(len(x)-1))
    return ((x-mean)/std)/np.sqrt(len(x)-1)

sdf = df.apply(standardize_var)
sdf_X = sdf[['YEAR', 'MN', 'DT', 'MAX', 'MIN', 'AW', 'SLP', 'MSLP', 'DBT', 'WBT', 'DPT', 'RH', 'VP', 'DD', 'FFF', 'DRNF']]
corr = np.array(sdf_X.corr())
corr_inv = np.linalg.inv(corr)
fit = ols('RF~YEAR+MN+DT+MAX+MIN+AW+SLP+MSLP+DBT+WBT+DPT+RH+VP+DD+FFF+DRNF',data=sdf).fit()
variables = []
reg_coef = []
vif = []
for i in range(len(sdf_X.columns)):
    col_name = sdf_X.columns[i]
    variables.append(col_name)
    reg_coef.append(fit.params[col_name])
    vif.append(corr_inv[i][i])

df_res = pd.DataFrame()
df_res['Variable'] = variables
df_res['Estimate'] = reg_coef
df_res['VIF'] = vif

df_res
```

```
colormap = plt.cm.PuBu
plt.figure(figsize=(22,18))
plt.title("Rainfall Correlation of Features", y = 1.1, size = 16)
sns.heatmap(df.astype(int).corr(), linewidths = 0.0, vmax = 1.0,
            square = True, cmap = colormap, linecolor = "white", annot = True, annot_kws = {"size" : 16})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb8d331e610>
```



```
features = df[['YEAR', 'MN', 'DT', 'MAX', 'MIN', 'AW', 'SLP', 'MSLP', 'DBT', 'WBT', 'DPT', 'RH', 'VP', 'DD', 'FFF', 'DRNF']]


Y = df['RF']
print(features)
```

```
      YEAR  MN  DT   MAX   MIN    AW    SLP    MSLP   DBT   WBT   DPT    RH  \
42    1985   2  15  32.7  18.5   5.0  928.6  1003.3  31.6  20.0  12.9  32.0
43    1985   2  16  31.8  19.1   1.0  928.6  1003.3  31.8  19.6  11.8  30.0
44    1985   2  17  30.6  18.6   2.0  927.7  1002.2  31.2  19.4  11.9  31.0
45    1985   2  18  29.8  17.6   5.0  926.4  1001.1  31.0  19.4  12.0  31.0
46    1985   2  19  28.8  17.4   3.0  927.4  1002.4  29.4  20.0  14.5  40.0
...    ...  ..  ..   ...   ...   ...    ...     ...   ...   ...   ...   ...
5937  2001   7  10  32.0  23.5  11.0  937.7  1014.8  25.0  20.0  17.3  62.0
5938  2001   7  11  32.5  24.0  10.0  937.6  1014.7  24.4  21.0  19.3  73.0
5939  2001   7  12  31.5  23.5   9.0  936.0  1013.0  25.0  19.0  15.5  56.0
5940  2001   7  13  32.5  24.0   7.0  936.2  1013.4  23.8  19.2  16.6  64.0
5941  2001   7  14  33.5  22.5   9.0  935.9  1013.1  23.4  18.8  16.1  64.0

        VP    DD   FFF   DRNF
42    14.9  18.0   6.0    0.0
43    13.8  18.0   4.0    0.0
44    13.9  14.0   6.0    0.0
45    14.0   0.0   0.0    0.0
46    16.5   9.0  12.0    0.0
...    ...   ...   ...    ...
5937  19.7  14.0   4.0    0.0
5938  22.4  14.0   4.0    8.0
5939  17.6  14.0   4.0    0.0
5940  18.9  14.0   4.0    0.0
5941  18.3  14.0   4.0  255.0

[5787 rows x 16 columns]
```

```
train_features, test_features, train_labels, test_labels = train_test_split(features, Y)


scaler = StandardScaler()
train_features = scaler.fit_transform(train_features)
test_features = scaler.transform(test_features)
from sklearn import preprocessing
lab = preprocessing.LabelEncoder()
train_labels = lab.fit_transform(train_labels)



accuracy={}
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(train_features, train_labels)
```

```
LinearRegression()
```

```
accuracy["Lin R"]=model.score(train_features, train_labels)
print("LinearRegression:",model.score(train_features, train_labels))
```

```
LinearRegression: 0.35325381892785757
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(train_features, train_labels)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
```

```
accuracy["Log R"]=model.score(train_features, train_labels)
print("LogisticRegression:",model.score(train_features, train_labels))
```

```
LogisticRegression: 0.9310289738535746
```

```
from sklearn.linear_model import ARDRegression
model = ARDRegression()
model.fit(train_features, train_labels)
```

```
ARDRegression()
```

```
print(model.score(train_features, train_labels))
```

```
0.3523723799595253
```

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(train_features, train_labels)
```

```
KNeighborsClassifier()
```

```
accuracy["KNN"]=model.score(train_features, train_labels)

print("K Nearest Neighbors:",model.score(train_features, train_labels))
```

```
K Nearest Neighbors: 0.7857142857142857
```

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion="entropy", max_depth=5)
model.fit(train_features, train_labels)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
accuracy["DTree"]=model.score(train_features, train_labels)
```

```
print("Decision Tree:",model.score(train_features, train_labels))
```

```
Decision Tree: 0.7817972350230414
```

```
from sklearn import svm
model = svm.SVC()
model.fit(train_features, train_labels)
```

```
SVC()
```

```
accuracy["SVM"]=model.score(train_features, train_labels)
```

```
print("Support Vector Machine:",model.score(train_features, train_labels))
```

```
Support Vector Machine: 0.7788018433179723
```

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
model.fit(train_features, train_labels)
```

```
GaussianNB()
```

```
GaussianNB()
```

```
accuracy["NB"]=model.score(train_features, train_labels)
```

```
print("Naïve Bayes:",model.score(train_features, train_labels))
```

```
Naïve Bayes: 0.6967741935483871
```

```
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
model.fit(train_features, train_labels)
```

```
RandomForestRegressor()
```

```
RandomForestRegressor()
```

```
accuracy["RF"]=model.score(train_features, train_labels)
```

```
print("Random Forest:",model.score(train_features, train_labels))
```

```
Random Forest: 0.9310289738535746
```

```
accuracy
```