rainfall_prediction

April 3, 2025

[2]:
```
import numpy as np
import pandas as pd
import joblib
```

[3]:
```
raw_data = pd.read_csv("train.csv")
raw_data
```

[3]:

	id	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	\
0	0	1	1017.4	21.2	20.6	19.9	19.4	87.0	
1	1	2	1019.5	16.2	16.9	15.8	15.4	95.0	
2	2	3	1024.1	19.4	16.1	14.6	9.3	75.0	
3	3	4	1013.4	18.1	17.8	16.9	16.8	95.0	
4	4	5	1021.8	21.3	18.4	15.2	9.6	52.0	
…	…	…	…	…	…	…	…	…	
2185	2185	361	1014.6	23.2	20.6	19.1	19.9	97.0	
2186	2186	362	1012.4	17.2	17.3	16.3	15.3	91.0	
2187	2187	363	1013.3	19.0	16.3	14.3	12.6	79.0	
2188	2188	364	1022.3	16.4	15.2	13.8	14.7	92.0	
2189	2189	365	1013.8	21.2	19.1	18.0	18.0	89.0	

	cloud	sunshine	winddirection	windspeed	rainfall
0	88.0	1.1	60.0	17.2	1
1	91.0	0.0	50.0	21.9	1
2	47.0	8.3	70.0	18.1	1
3	95.0	0.0	60.0	35.6	1
4	45.0	3.6	40.0	24.8	0
…	…	…	…	…	…
2185	88.0	0.1	40.0	22.1	1
2186	88.0	0.0	50.0	35.3	1
2187	79.0	5.0	40.0	32.9	1
2188	93.0	0.1	40.0	18.0	1
2189	88.0	1.0	70.0	48.0	1

[2190 rows x 13 columns]

[4]:
```
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2190 entries, 0 to 2189
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             2190 non-null   int64
 1   day            2190 non-null   int64
 2   pressure       2190 non-null   float64
 3   maxtemp        2190 non-null   float64
 4   temparature    2190 non-null   float64
 5   mintemp        2190 non-null   float64
 6   dewpoint       2190 non-null   float64
 7   humidity       2190 non-null   float64
 8   cloud          2190 non-null   float64
 9   sunshine       2190 non-null   float64
 10  winddirection  2190 non-null   float64
 11  windspeed      2190 non-null   float64
 12  rainfall       2190 non-null   int64
dtypes: float64(10), int64(3)
memory usage: 222.5 KB
```

[5]:
```
raw_data.describe()
```

[5]:

	id	day	pressure	maxtemp	temparature
count	2190.000000	2190.000000	2190.000000	2190.000000	2190.000000
mean	1094.500000	179.948402	1013.602146	26.365799	23.953059
std	632.342866	105.203592	5.655366	5.654330	5.222410
min	0.000000	1.000000	999.000000	10.400000	7.400000
25%	547.250000	89.000000	1008.600000	21.300000	19.300000
50%	1094.500000	178.500000	1013.000000	27.800000	25.500000
75%	1641.750000	270.000000	1017.775000	31.200000	28.400000
max	2189.000000	365.000000	1034.600000	36.000000	31.500000

	mintemp	dewpoint	humidity	cloud	sunshine
count	2190.000000	2190.000000	2190.000000	2190.000000	2190.000000
mean	22.170091	20.454566	82.036530	75.721918	3.744429
std	5.059120	5.288406	7.800654	18.026498	3.626327
min	4.000000	-0.300000	39.000000	2.000000	0.000000
25%	17.700000	16.800000	77.000000	69.000000	0.400000
50%	23.850000	22.150000	82.000000	83.000000	2.400000
75%	26.400000	25.000000	88.000000	88.000000	6.800000
max	29.800000	26.700000	98.000000	100.000000	12.100000

	winddirection	windspeed	rainfall
count	2190.000000	2190.000000	2190.000000
mean	104.863151	21.804703	0.753425
std	80.002416	9.898659	0.431116
min	10.000000	4.400000	0.000000

```
25%        40.000000    14.125000     1.000000
50%        70.000000    20.500000     1.000000
75%       200.000000    27.900000     1.000000
max       300.000000    59.500000     1.000000
```

[6]:
```
corr_mat = raw_data.corr(numeric_only=True)
corr_mat["rainfall"].sort_values()
```

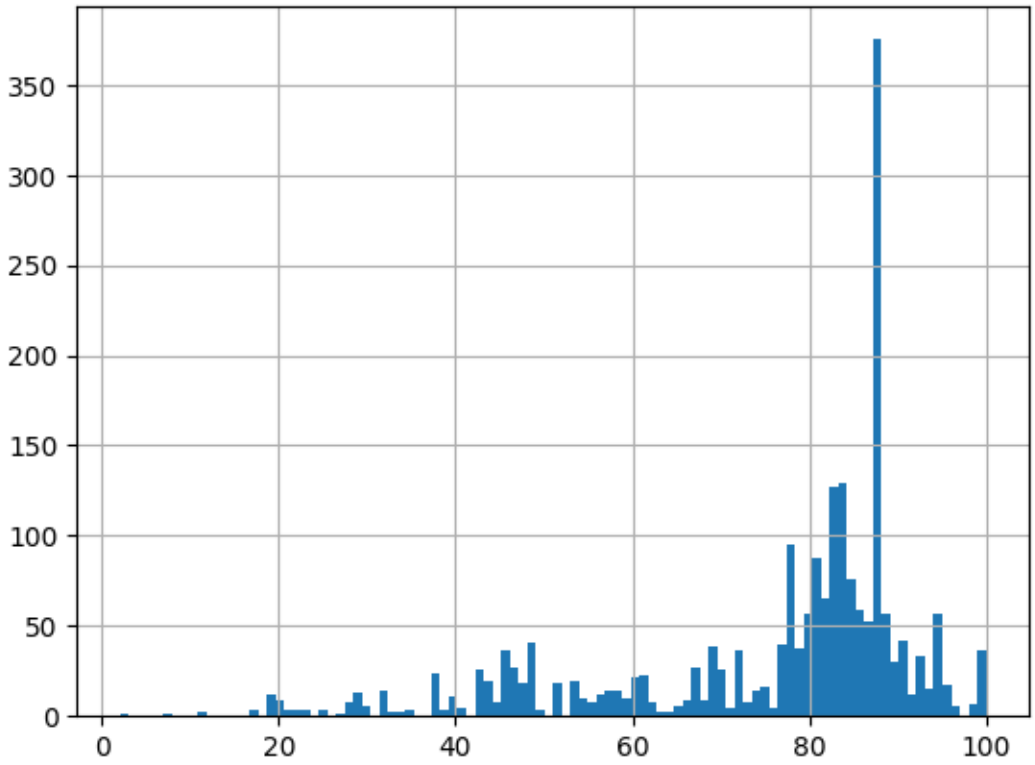
[6]:
```
sunshine        -0.555287
maxtemp         -0.079304
pressure        -0.049886
temparature     -0.049660
mintemp         -0.026841
winddirection   -0.006939
day             -0.000462
id               0.033674
dewpoint         0.081965
windspeed        0.111625
humidity         0.454213
cloud            0.641191
rainfall         1.000000
Name: rainfall, dtype: float64
```

[7]:
```
raw_data["cloud"].describe()
```

[7]:
```
count    2190.000000
mean       75.721918
std        18.026498
min         2.000000
25%        69.000000
50%        83.000000
75%        88.000000
max       100.000000
Name: cloud, dtype: float64
```

[8]:
```
raw_data["cloud"].hist(bins=100)
```

[8]:
```
<Axes: >
```

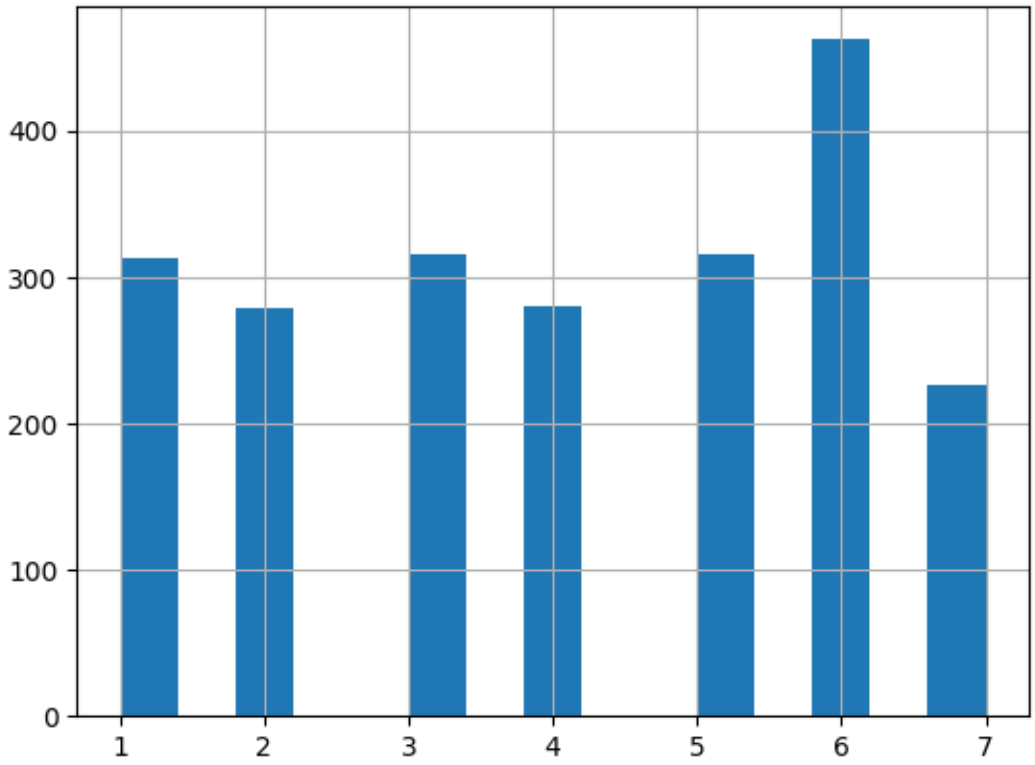


[9]:
```
raw_data["rainfall"].value_counts()
```

[9]:
```
rainfall
1    1650
0     540
Name: count, dtype: int64
```

[10]:
```
raw_data["cloud_category"] = pd.cut(raw_data["cloud"],
                                    bins=[0, 50, 70, 80, 83.5, 87.5, 90, np.
 ↪inf],
                                    labels=[1, 2, 3, 4, 5, 6, 7]
                                    )
raw_data["cloud_category"].hist(bins=15)
```

[10]:
```
<Axes: >
```



[11]:
```
attribs = list(raw_data)
attribs.remove("id")
attribs.remove("rainfall")
attribs.remove("cloud_category")
attribs
```

[11]:
```
['day',
 'pressure',
 'maxtemp',
 'temparature',
 'mintemp',
 'dewpoint',
 'humidity',
 'cloud',
 'sunshine',
 'winddirection',
 'windspeed']
```

[12]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
```

```
for train_index, test_index in split.split(raw_data,␣
 ↪raw_data["cloud_category"]):
    strat_trainset = raw_data.loc[train_index]
    strat_testset = raw_data.loc[test_index]

X_train = pd.DataFrame(strat_trainset, columns=list(raw_data))
X_test = pd.DataFrame(strat_testset, columns=list(raw_data))

y_train = X_train[["rainfall"]]
y_test = X_test[["rainfall"]]

X_train.drop(columns=["rainfall"], inplace=True)
X_test.drop(columns=["rainfall"], inplace=True)

X_train.drop(columns=["cloud_category"], inplace=True)
X_test.drop(columns=["cloud_category"], inplace=True)
```

[13]:
```
X_train
```

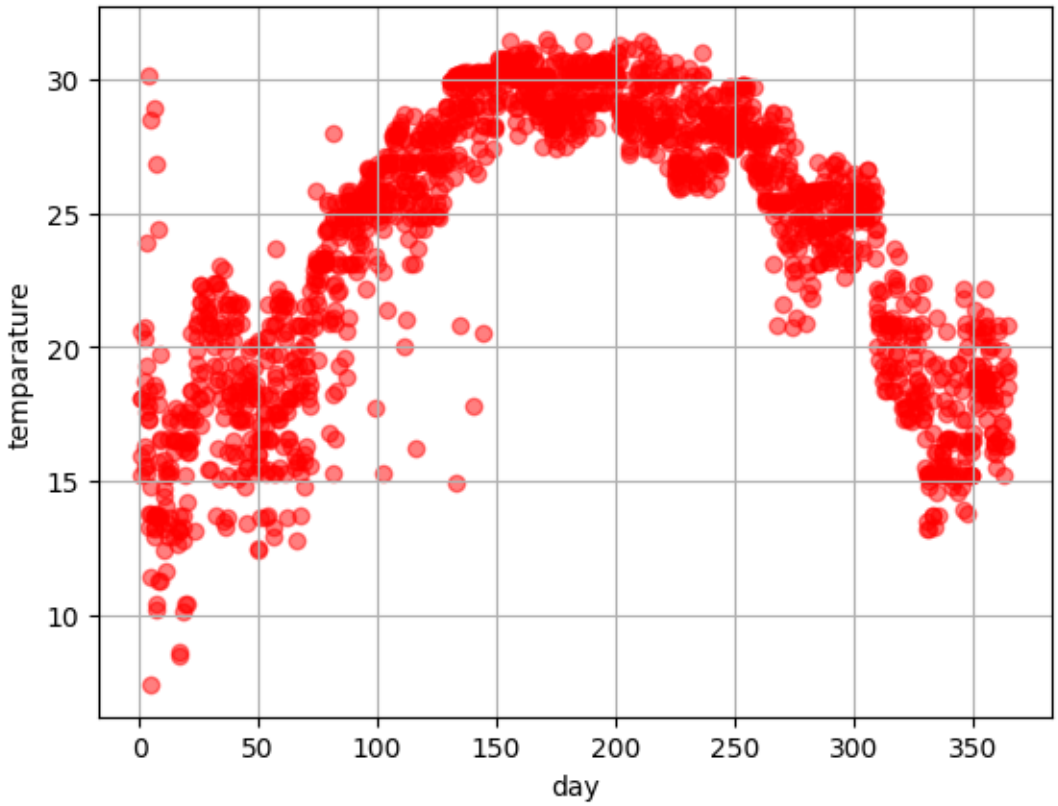
[13]:

	id	day	pressure	maxtemp	temparature	mintemp	dewpoint	humidity	\
949	949	220	1006.4	31.3	28.8	27.3	26.4	87.0	
83	83	84	1010.7	24.2	23.0	21.4	17.0	86.0	
1114	1114	20	1021.3	19.6	17.6	15.7	12.6	64.0	
1339	1339	245	1010.2	31.7	28.2	25.9	23.8	79.0	
1776	1776	317	1015.1	27.5	23.7	22.2	21.3	91.0	
…	…	…	…	…	…	…	…	…	
2008	2008	184	1008.1	31.0	28.1	26.8	25.1	81.0	
0	0	1	1017.4	21.2	20.6	19.9	19.4	87.0	
1123	1123	29	1017.6	24.8	21.3	19.8	20.1	91.0	
291	291	292	1017.1	26.0	24.9	22.3	22.2	81.0	
569	569	205	1008.8	31.3	29.2	26.4	24.3	67.0	

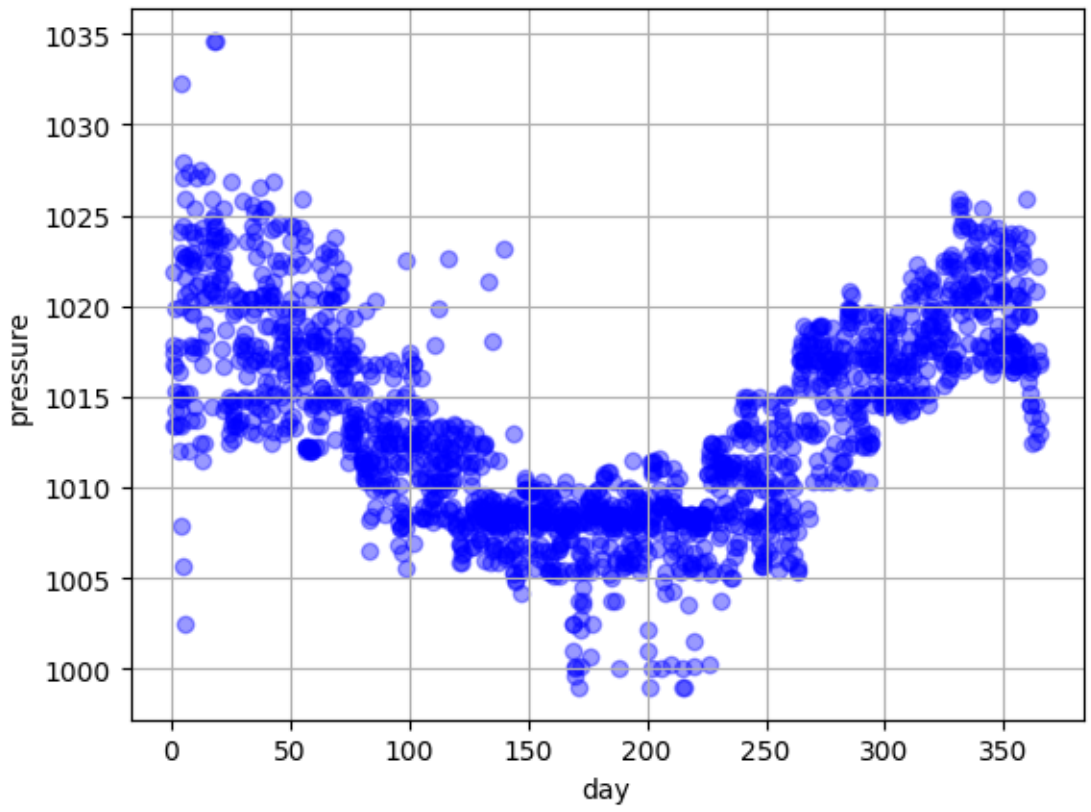
	cloud	sunshine	winddirection	windspeed
949	84.0	1.2	220.0	13.9
83	80.0	1.1	70.0	9.8
1114	73.0	5.2	60.0	20.2
1339	88.0	1.6	70.0	22.0
1776	93.0	5.9	70.0	22.5
…	…	…	…	…
2008	84.0	2.1	190.0	15.9
0	88.0	1.1	60.0	17.2
1123	89.0	0.2	50.0	37.5
291	84.0	2.4	70.0	12.0
569	46.0	8.1	230.0	18.1

[1752 rows x 12 columns]

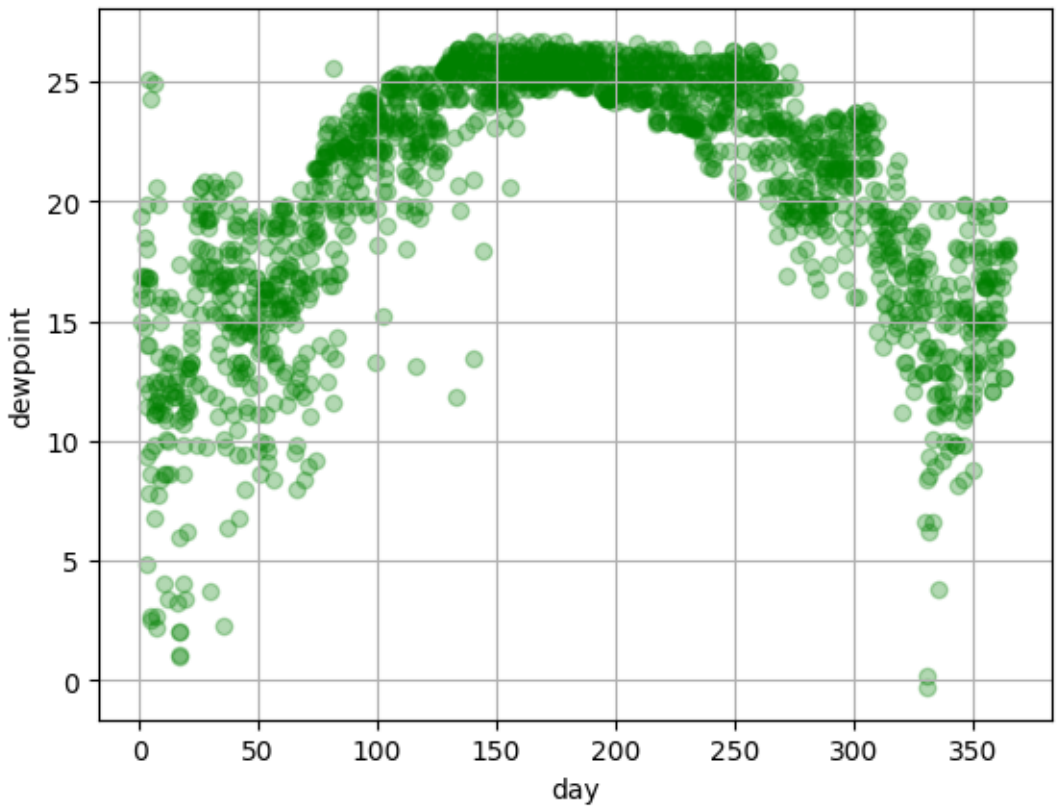
[14]:
```
%matplotlib inline
import matplotlib.pyplot as plt

def show_scatter(x_param, y_param, alpha=1, c='r'):
    plt.scatter(X_train[x_param], X_train[y_param], alpha=alpha, c=c)
    plt.xlabel(x_param)
    plt.ylabel(y_param)
    plt.grid(True)
    plt.show()


show_scatter("day", "temparature", alpha=0.5)
```



[15]:
```
show_scatter("day", "pressure", c='b', alpha=0.4)
```



[16]:
```
show_scatter("day", "dewpoint", alpha=0.3, c='g')
```



[17]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

trans_pipeline = Pipeline(
    [
        ("impute", SimpleImputer(strategy="median")),
        ("scale", StandardScaler()),
    ]
)

full_pipeline = ColumnTransformer([
    ("pipe", trans_pipeline, attribs)
])

tr_data_train = full_pipeline.fit_transform(X_train)
tr_data_test = full_pipeline.transform(X_test)
```

[18]:
```
# joblib.dump(full_pipeline, "full_pipeline.joblib")
```

[19]:
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42, loss='log_loss')
sgd_clf.fit(tr_data_train, y_train.values.ravel())
```

[19]:
```
SGDClassifier(loss='log_loss', random_state=42)
```

[20]:
```
from sklearn.metrics import roc_auc_score, roc_curve

y_train_scores = sgd_clf.decision_function(tr_data_train)

# y_train_scores
```

[21]:
```
roc_auc_score(y_train.values.ravel(), y_train_scores)
```

[21]:
```
np.float64(0.8959803680118215)
```

[22]:
```
y_test_scores = sgd_clf.decision_function(tr_data_test)
roc_auc_score(y_test.values.ravel(), y_test_scores)
```

[22]:
```
np.float64(0.8811807095343682)
```

[23]:
```
# joblib.dump(sgd_clf, "sgd_clf_transformed.pkl")
```

[24]:
```
from sklearn.ensemble import RandomForestClassifier

rfc_model = RandomForestClassifier(random_state=42)
rfc_model.fit(tr_data_train, y_train.values.ravel())
rfc_pred = rfc_model.predict_proba(tr_data_train)
roc_auc_score(y_train.values.ravel(), rfc_pred[:, 1])
```

[24]:
```
np.float64(1.0)
```

[25]:
```
from sklearn.model_selection import cross_val_predict

rfc_y_scores_train = cross_val_predict(rfc_model, tr_data_train, y_train.values.
 ↪ravel(), method="predict_proba", cv=3)
roc_auc_score(y_train.values.ravel(), rfc_y_scores_train[:, 1])
```

[25]:
```
np.float64(0.877747774689512)
```

[26]:
```
rfc_y_scores_test = rfc_model.predict_proba(tr_data_test)
roc_auc_score(y_test.values.ravel(), rfc_y_scores_test[:, 1])
```

[26]:
```
np.float64(0.8873752771618625)
```

[27]:
```
# joblib.dump(rfc_model, "random_forest_clf.pkl")
```

[28]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=295, weights="distance")

knn_clf.fit(tr_data_train, y_train.values.ravel())
knn_pred = knn_clf.predict_proba(tr_data_train)
knn_pred = cross_val_predict(knn_clf, tr_data_train, y_train.values.ravel(),␣
 ↪cv=3, method="predict_proba")
roc_auc_score(y_train.values.ravel(), knn_pred[:, 1])
```

[28]:
```
np.float64(0.8824033353270238)
```

[29]:
```
knn_pred_test = knn_clf.predict_proba(tr_data_test)
roc_auc_score(y_test.values.ravel(), knn_pred_test[:, 1])
```

[29]:
```
np.float64(0.8743625277161864)
```

[30]:
```
# joblib.dump(knn_clf, "knn_model.pkl")
```

[31]:
```
from sklearn.svm import SVC

svc_model = SVC(kernel="linear")
svc_model.fit(tr_data_train, y_train.values.ravel())
svc_proba_train = svc_model.decision_function(tr_data_train)
roc_auc_score(y_train.values.ravel(), svc_proba_train)
```

[31]:
```
np.float64(0.8959979593990781)
```

[32]:
```
svc_proba_test = svc_model.decision_function(tr_data_test)
roc_auc_score(y_test.values.ravel(), svc_proba_test)
```

[32]:
```
np.float64(0.8844235033259423)
```

[33]:
```
# joblib.dump(svc_model, "svc_model.pkl")
```

[34]:
```
from sklearn.linear_model import LogisticRegression


logistic_model = LogisticRegression()

logistic_model.fit(tr_data_train, y_train.values.ravel())
# knn_pred = logistic_model.predict_proba(tr_data_train)
logistic_pred_train = cross_val_predict(logistic_model, tr_data_train, y_train.
 ↪values.ravel(), cv=3, method="predict_proba")
roc_auc_score(y_train.values.ravel(), logistic_pred_train[:, 1])
```

[34]:
```
np.float64(0.8932114836576012)
```

[35]:
```
logistic_pred_test = logistic_model.predict_proba(tr_data_test)
roc_auc_score(y_test.values.ravel(), logistic_pred_test[:, 1])
```

[35]:
```
np.float64(0.8855044345898004)
```

[36]:
```
# joblib.dump(logistic_model, "logistic_model.pkl")
```

[37]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {
        "penalty" : ["l2", None],
        "solver": ["lbfgs", "newton-cg", "newton-cholesky", "sag", "saga"],
        "max_iter": [150, 200, 300, 500, 1000],

    },
    {
        "penalty" : ["l2", None, "l1"],
        "solver":["saga"],
        "max_iter": [150, 200, 300, 500, 1000],
    },
    {
        "penalty" : ["l2", "l1"],
        "solver":["liblinear"],
        "max_iter": [150, 200, 300, 500, 1000],
        "intercept_scaling" : [2.16135, 2.1615, 2.16165],
    },
    {
        "penalty" : ["l2"],
        "solver":["liblinear"],
        "max_iter": [150, 200, 300, 500],
        "dual" : [True, False],
    },
    {
        "penalty" : ["elasticnet"],
        "solver":["saga"],
        "max_iter": [150, 200, 300, 500, 1000],
        "l1_ratio" : [1, 0.75, 0.5, 0.25, 0]
    }
]
try:
    best_logistic_model = joblib.load("best_logistic_model.pkl")
except Exception as e:
    hyp_logistic_model = LogisticRegression()
    grid_search = GridSearchCV(hyp_logistic_model, param_grid,␣
 ↪return_train_score=True, cv=3, scoring="roc_auc")
    grid_search.fit(tr_data_train, y_train.values.ravel())
```

```
    best_logistic_model = grid_search.best_estimator_
```

[38]:
```
logistic_pred_train = cross_val_predict(best_logistic_model, tr_data_train,␣
 ↪y_train.values.ravel(), cv=3, method="predict_proba")
roc_auc_score(y_train.values.ravel(), logistic_pred_train[:, 1])
```

[38]: np.float64(0.893438412553214)

[39]:
```
logistic_pred_test = best_logistic_model.predict_proba(tr_data_test)
roc_auc_score(y_test.values.ravel(), logistic_pred_test[:, 1])
```

[39]: np.float64(0.8860864745011087)

[40]:
```
# joblib.dump(best_logistic_model, "best_logistic_model.pkl")
```

[41]:
```
param_grid_sgd = [
    # {
    #     "loss" : ["hinge", "log_loss", "modified_huber", "squared_hinge",␣
 ↪"perceptron", "squared_error", "huber", "epsilon_insensitive",␣
 ↪'squared_epsilon_insensitive'],
    #     "penalty" : ["l1", "l2", "elasticnet", None],
    #     "max_iter": [500, 1000, 1500],
    #     "learning_rate": ["optimal"],
    # },
    {
        "loss" : ["hinge", "log_loss", "modified_huber", "squared_hinge",␣
 ↪"perceptron", "squared_error", "huber", "epsilon_insensitive",␣
 ↪'squared_epsilon_insensitive'],
        "penalty" : ["l1", "l2", "elasticnet", None],
        "max_iter": [3000, 3200],
        "learning_rate": ["constant", "invscaling", "adaptive"],
        "eta0": [0.1, 0.5, 1.25, 1.5, 1.65, 1.75]
    }
]

sgd_model = SGDClassifier()
grid_search_sgd = GridSearchCV(sgd_model, param_grid_sgd,␣
 ↪return_train_score=True, cv=3, scoring="roc_auc")
grid_search_sgd.fit(tr_data_train, y_train.values.ravel())
best_sgd_model = grid_search_sgd.best_estimator_
```

```
/home/ashmit/.local/lib/python3.10/site-
packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/home/ashmit/.local/lib/python3.10/site-
```

```
packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/home/ashmit/.local/lib/python3.10/site-
packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
/home/ashmit/.local/lib/python3.10/site-
packages/sklearn/linear_model/_stochastic_gradient.py:744: ConvergenceWarning:
Maximum number of iteration reached before convergence. Consider increasing
max_iter to improve the fit.
  warnings.warn(
```

[42]:
```
grid_search_sgd.best_params_
```

[42]:
```
{'eta0': 1.65,
 'learning_rate': 'adaptive',
 'loss': 'squared_hinge',
 'max_iter': 3200,
 'penalty': 'l1'}
```

[43]:
```
sgd_pred_train = cross_val_predict(best_sgd_model, tr_data_train, y_train.
 ↪values.ravel(), cv=3, method="decision_function")
roc_auc_score(y_train.values.ravel(), sgd_pred_train)
```

[43]:
```
np.float64(0.8934243394434085)
```

[44]:
```
sgd_pred_train = best_sgd_model.decision_function(tr_data_test)
roc_auc_score(y_test.values.ravel(), sgd_pred_train)
```

[44]:
```
np.float64(0.8864467849223947)
```

[45]:
```
joblib.dump(best_sgd_model, "best_sgd_model.pkl")
```

[45]:
```
['best_sgd_model.pkl']
```

[46]:
```
trans_pipeline_no_day = Pipeline(
    [
        ("impute", SimpleImputer(strategy="median")),
        ("scale", StandardScaler()),
    ]
)

attribs.remove("day")
attribs.remove("winddirection")
```

```
full_pipeline_no_day = ColumnTransformer([
    ("pipe", trans_pipeline_no_day, attribs)
])

tr_data_train_no_day = full_pipeline_no_day.fit_transform(X_train)
tr_data_test_no_day = full_pipeline_no_day.transform(X_test)
```

[47]:
```
# joblib.dump(full_pipeline_no_day, "full_pipeline_no_day.joblib")
```

[48]:
```
param_grid = [
    {
        "penalty" : ["l2", None],
        "solver": ["lbfgs", "newton-cg", "newton-cholesky", "sag", "saga"],
        "max_iter": [300, 500, 1000],

    },
    {
        "penalty" : ["l2", None, "l1"],
        "solver":["saga"],
        "max_iter": [150, 200, 300, 500, 1000],
    },
    {
        "penalty" : ["l2", "l1"],
        "solver":["liblinear"],
        "max_iter": [150, 200, 300, 500, 1000],
        "intercept_scaling" : [2.16135, 2.1615, 2.16165],
    },
    {
        "penalty" : ["l2"],
        "solver":["liblinear"],
        "max_iter": [150, 200, 300, 500],
        "dual" : [True, False],
    },
    {
        "penalty" : ["elasticnet"],
        "solver":["saga"],
        "max_iter": [150, 200, 300, 500, 1000],
        "l1_ratio" : [1, 0.75, 0.5, 0.25, 0]
    }
]
try:
    best_logistic_model_no_day = joblib.load("best_logistic_model_no_day.pkl")
except Exception as e:
    hyp_logistic_model = LogisticRegression()
    grid_search = GridSearchCV(hyp_logistic_model, param_grid,␣
 ↪return_train_score=True, cv=3,scoring="roc_auc")
    grid_search.fit(tr_data_train_no_day, y_train.values.ravel())
```

```
    best_logistic_model_no_day = grid_search.best_estimator_
```

[49]:
```
# grid_search.best_params_
```

[50]:
```
logistic_pred_train_no_day = cross_val_predict(best_logistic_model_no_day,␣
 ↪tr_data_train_no_day, y_train.values.ravel(), cv=3, method="predict_proba")
roc_auc_score(y_train.values.ravel(), logistic_pred_train_no_day[:, 1])
```

[50]:
```
np.float64(0.8949688632445555)
```

[51]:
```
logistic_pred_test_no_day = best_logistic_model_no_day.
 ↪predict_proba(tr_data_test_no_day)
roc_auc_score(y_test.values.ravel(), logistic_pred_test_no_day[:, 1])
```

[51]:
```
np.float64(0.8876940133037694)
```

[52]:
```
# joblib.dump(best_logistic_model_no_day, "best_logistic_model_no_day.pkl")
```

[53]:
```
list(X_train)
```

[53]:
```
['id',
 'day',
 'pressure',
 'maxtemp',
 'temparature',
 'mintemp',
 'dewpoint',
 'humidity',
 'cloud',
 'sunshine',
 'winddirection',
 'windspeed']
```

[54]:
```
X_train_copy = X_train.copy()
```

[55]:
```
attribs
```

[55]:
```
['pressure',
 'maxtemp',
 'temparature',
 'mintemp',
 'dewpoint',
 'humidity',
 'cloud',
 'sunshine',
 'windspeed']
```

[114]:
```
from sklearn.base import BaseEstimator, TransformerMixin

class AddDewpointDepression(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        dew_point_depression = X[:, 3] - X[:, 5]
        temp_diff = X[:,2] - X[:,4]
        wind_chill = X[:,3] * X[:,10]
        return np.c_[X, dew_point_depression, temp_diff, wind_chill]
```

[57]:
```
add_feature = AddDewpointDepression()
X_train_more_feature = add_feature.transform(X_train.values)
```

[58]:
```
X_train_more_feature.shape
```

[58]:
```
(1752, 3)
```

[59]:
```
attribs = list(X_train)
attribs.remove("id")
# attribs
```

[60]:
```
trans_pipeline_more_feature = Pipeline(
    [
        ("impute", SimpleImputer(strategy="median")),
        ("add_feature", AddDewpointDepression()),
        ("scale", StandardScaler()),
    ]
)


full_pipeline_more_feature = ColumnTransformer([
    ("pipe", trans_pipeline_more_feature, attribs)
])

tr_data_train_more_feature = full_pipeline_more_feature.fit_transform(X_train)
tr_data_test_more_feature = full_pipeline_more_feature.transform(X_test)
```

[61]:
```
# joblib.dump(full_pipeline_more_feature, "full_pipeline_more_feature.joblib")
```

[62]:
```
tr_data_train_more_feature.shape
```

[62]:
```
(1752, 3)
```

[63]:
```
param_grid = [
    {
        "penalty" : ["l2", None],
        "solver": ["lbfgs", "newton-cg", "sag", "saga"],
        "max_iter": [300, 500, 1000],
        "random_state":[42]

    },
    {
        "penalty" : ["l2", None, "l1"],
        "solver":["saga"],
        "max_iter": [150, 300, 500, 1000],
        "random_state":[42]
    },
    {
        "penalty" : ["l2", "l1"],
        "solver":["liblinear"],
        "max_iter": [20, 30, 40, 50, 100, 200],
        "intercept_scaling" : [2.16135, 2.1615, 2.16165],
        "random_state":[42]
    },
    {
        "penalty" : ["l2"],
        "solver":["liblinear"],
        "max_iter": [150, 200, 300, 500],
        "dual" : [True, False],
        "random_state":[42]
    },
    {
        "penalty" : ["elasticnet"],
        "solver":["saga"],
        "max_iter": [150, 200, 300, 500, 1000],
        "l1_ratio" : [1, 0.75, 0.5, 0.25, 0],
        "random_state":[42]
    }
]
try:
    # best_logistic_model_more_feature = joblib.
 ↪load("best_logistic_model_dew_point_depression.pkl")
    best_logistic_model_more_feature = joblib.
 ↪load("best_logistic_model_dew_point_depression123.pkl")
except Exception as e:
    hyp_logistic_model = LogisticRegression()
    grid_search = GridSearchCV(hyp_logistic_model, param_grid,␣
 ↪return_train_score=True, cv=3,scoring="roc_auc")
    grid_search.fit(tr_data_train_more_feature, y_train.values.ravel())
    best_logistic_model_more_feature = grid_search.best_estimator_
```

[81]:
```
# grid_search.best_params_
```

[64]:
```
logistic_pred_train_more_feature =␣
 ↪cross_val_predict(best_logistic_model_more_feature,␣
 ↪tr_data_train_more_feature, y_train.values.ravel(), cv=3,␣
 ↪method="predict_proba")
roc_auc_score(y_train.values.ravel(), logistic_pred_train_more_feature[:, 1])
```

[64]:
```
np.float64(0.7533951377405623)
```

[65]:
```
logistic_pred_test_more_feature = best_logistic_model_more_feature.
 ↪predict_proba(tr_data_test_more_feature)
roc_auc_score(y_test.values.ravel(), logistic_pred_test_more_feature[:, 1])
```

[65]:
```
np.float64(0.7360587583148559)
```

[ ]:
```
# joblib.dump(best_logistic_model_more_feature,␣
 ↪"best_logistic_model_dew_point_depression.pkl")
```

[ ]:
```
['best_logistic_model_dew_point_depression.pkl']
```

[66]:
```
X_train_copy["dew_point_depression"] = X_train_copy["temparature"] -␣
 ↪X_train_copy["dewpoint"]
X_train_copy["temp_diff"] = X_train_copy["maxtemp"] - X_train_copy["mintemp"]
X_train_copy["wind_clill"] = X_train_copy["temparature"] *␣
 ↪X_train_copy["windspeed"]
X_train_copy["rainfall"] = y_train["rainfall"]
corr_matx = X_train_copy.corr(numeric_only=True)
corr_matx["rainfall"].sort_values()
```

[66]:
```
sunshine               -0.561776
dew_point_depression   -0.376055
temp_diff              -0.201654
maxtemp                -0.097131
temparature            -0.070379
mintemp                -0.048355
pressure               -0.030821
day                    -0.024052
winddirection          -0.022864
id                      0.021292
dewpoint                0.065464
wind_clill              0.085184
windspeed               0.104861
humidity                0.453765
cloud                   0.640230
rainfall                1.000000
Name: rainfall, dtype: float64
```

[115]:
```
reduce_attribs = attribs.copy()
# reduce_attribs.remove("pressure")
# reduce_attribs.remove("day")
# reduce_attribs.remove("winddirection")
# reduce_attribs.remove("dewpoint")
# reduce_attribs.remove("mintemp")

trans_pipeline_more_feature = Pipeline(
    [
        ("impute", SimpleImputer(strategy="mean")),
        ("add_feature", AddDewpointDepression()),
        ("scale", StandardScaler()),
    ]
)
trans_pipeline_reduce_feature = Pipeline(
    [
        ("add_feature", AddDewpointDepression()),
        ("scale", StandardScaler()),
    ]
)
full_pipeline_reduce_feature = ColumnTransformer([
    ("pipe", trans_pipeline_more_feature, attribs),
    # ("reduce", trans_pipeline_reduce_feature, attribs)
])

tr_data_train_reduce_feature = full_pipeline_reduce_feature.
 ↪fit_transform(X_train)
tr_data_test_reduce_feature = full_pipeline_reduce_feature.transform(X_test)
```

[116]:
```
tr_data_test_reduce_feature.shape
```

[116]:
```
(438, 14)
```

[117]:
```
joblib.dump(full_pipeline_reduce_feature, "full_pipeline_temp_diff.joblib")
```

[117]:
```
['full_pipeline_temp_diff.joblib']
```

[118]:
```
param_grid = [
    {
        "penalty" : ["l2", None],
        "solver": ["lbfgs", "newton-cg", "saga"],
        "max_iter": [5000],
        "random_state":[42]

    },
    {
        "penalty" : ["l2", None, "l1"],
```

```
        "solver":["saga"],
        "max_iter": [5000, 6000],
        "random_state":[42]
    },
    {
        "penalty" : ["l2", "l1"],
        "solver":["liblinear"],
        "max_iter": [5000],
        "intercept_scaling" : [2.16135, 2.1615, 2.16165],
        "random_state":[42]
    },
    {
        "penalty" : ["elasticnet"],
        "solver":["saga"],
        "max_iter": [5000],
        "l1_ratio" : [1, 0.75, 0.5, 0.25, 0],
        "random_state":[42]
    }
]
# try:
#     best_logistic_model = joblib.load("best_logistic_model.pkl")
# except Exception as e:
hyp_logistic_model = LogisticRegression()
grid_search = GridSearchCV(hyp_logistic_model, param_grid,␣
 ↪return_train_score=True, cv=3,scoring="roc_auc")
grid_search.fit(tr_data_train_reduce_feature, y_train.values.ravel())
best_logistic_model_reduce_feature = grid_search.best_estimator_
```

[119]:
```
grid_search.best_params_
```

[119]:
```
{'intercept_scaling': 2.16135,
 'max_iter': 5000,
 'penalty': 'l1',
 'random_state': 42,
 'solver': 'liblinear'}
```

[120]:
```
logistic_pred_train_reduce_feature =␣
 ↪cross_val_predict(best_logistic_model_reduce_feature,␣
 ↪tr_data_train_reduce_feature, y_train.values.ravel(), cv=3,␣
 ↪method="predict_proba")
roc_auc_score(y_train.values.ravel(), logistic_pred_train_reduce_feature[:, 1])
```

[120]:
```
np.float64(0.8929722407909088)
```

[121]:
```
logistic_pred_test_reduce_feature = best_logistic_model_reduce_feature.
 ↪predict_proba(tr_data_test_reduce_feature)
roc_auc_score(y_test.values.ravel(), logistic_pred_test_reduce_feature[:, 1])
```

[121]:
```
np.float64(0.8843126385809313)
```

[122]:
```
joblib.dump(best_logistic_model_reduce_feature, "best_logistic_model_temp_diff.
↪pkl")
```

[122]:
```
['best_logistic_model_temp_diff.pkl']
```

[ ]:
```

```