In [1]:

```
# In[0]: Bibliotecas
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np


from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder,MinMaxScaler



from sklearn.compose import ColumnTransformer


from sklearn.svm import LinearSVR
from sklearn.linear_model import LinearRegression,ElasticNet,Ridge
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, GradientBoostingRe
from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.tree import export_graphviz
from sklearn import tree

SEED = 42
```

In [2]:

```
# In[1]: Funcoes Gerais
def read_csv():
    df = pd.read_csv('pantanal.csv',',')

    #df['riscofogo'] = df['riscofogo'].interpolate(method='linear')

    return df


def cv_score(score):
    rmse = np.sqrt(-score)
    return (rmse)


def get_transformer():

    integer_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())])

    numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))])

    return integer_transformer, numeric_transformer,categorical_transformer


def prep_target(y,y_train, y_test):

    num_target = y.select_dtypes("float64").columns


    num_transf = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))])

    preprocessor = ColumnTransformer(
    transformers=[('num', num_transf, num_target)])

    transf = preprocessor.fit(y)

    y_train = transf.transform(y_train)
    y_test = transf.transform(y_test)

    return y_train, y_test


def prep_pipeline(X,y):

    X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.30,random_state=SE

    int_features = X.select_dtypes("int64").columns
    num_features = X.select_dtypes("float64").columns
    cat_features = X.select_dtypes(include=['object']).columns
```

```
    int_transf,num_transf,cat_transf = get_transformer()

    preprocessor = ColumnTransformer(
    transformers=[
        ('int', int_transf, int_features),
        ('num', num_transf, num_features),
        ('cat', cat_transf, cat_features)])

    transf = preprocessor.fit(X)

    X_train_prep = transf.transform(X_train)
    X_test_prep = transf.transform(X_test)

    y_train, y_test = prep_target(y,y_train, y_test)

    return X_train_prep, X_test_prep, y_train, y_test,preprocessor
```

In [7]:

```
# In[2]: Grid Search Model
def get_decisiontree_regressor(X_train,y_train):
    dt = DecisionTreeRegressor(random_state=0)

    params_dt = {
        'max_features':['auto','sqrt','log2'],
        'max_depth':[5,8,10,15,20,25,30],
        'min_samples_split':[5,8,10],
        'min_samples_leaf':[2,4]
    }


    grid_dt = GridSearchCV(estimator = dt,
                           param_grid = params_dt,
                           scoring ='neg_mean_squared_error',
                           cv = 3,
                           verbose = 1,
                           n_jobs = -1)

    grid_dt.fit(X_train, y_train.ravel())
    print('CV Score for best Decision Tree Regressor model: {:.3f}'.format(cv_score(grid_dt
    best_model_dt = grid_dt.best_estimator_
    return best_model_dt

def get_linear_regressor(X_train,y_train):
    linear = LinearRegression()

    params_linear = {
    }


    grid_linear = GridSearchCV(estimator = linear,
                           param_grid = params_linear,
                           scoring ='neg_mean_squared_error',
                           cv = 3,
                           verbose = 1,
                           n_jobs = -1)

    grid_linear.fit(X_train, y_train.ravel())
    print('CV Score for best Linear Regressor model: {:.3f}'.format(cv_score(grid_linear.be
    best_model_linear = grid_linear.best_estimator_
    return best_model_linear


def get_ridge_regression(X_train,y_train):
    ridge = Ridge(random_state=0)


    params_ridge = {
        'alpha': [1e-15, 1e-10, 1e-8, 9e-4, 7e-4, 5e-4, 3e-4,
                  1e-4, 1e-3, 5e-2, 1e-2, 0.1, 0.3, 1, 3, 5, 10, 15,
                  18, 20, 30, 50, 75, 100],
        'solver': ['auto','cholesky']
    }


    grid_ridge = GridSearchCV(estimator = ridge,
                           param_grid = params_ridge,
                           scoring ='neg_mean_squared_error',
```

```
                           cv = 3,
                           verbose = 1,
                           n_jobs = -1)

    grid_ridge.fit(X_train, y_train.ravel())
    print('CV Score for best Linear Ridge Regressor model: {:.3f}'.format(cv_score(grid_rid
    best_model_linear = grid_ridge.best_estimator_
    return best_model_linear


def get_svr_regressor(X_train,y_train):

    svr = LinearSVR()

    params_svr = {
        'epsilon': [1.5,3,5],
        'C': [1,5,7,10],
        'tol':[1e-9,1e-7,1e-5],
        'random_state': [42],
        'max_iter': [5000]
    }

    grid_svr = GridSearchCV(estimator = svr,
                           param_grid = params_svr,
                           scoring ='neg_mean_squared_error',
                           cv = 3,
                           verbose = 1,
                           n_jobs = -1)

    grid_svr.fit(X_train, y_train.ravel())
    print('CV Score for best SVR Regressor model: {:.3f}'.format(cv_score(grid_svr.best_sco
    best_model = grid_svr.best_estimator_
    return best_model

def get_rf_regressor(X_train,y_train):

    rf = RandomForestRegressor(random_state= 0)

    params_rf = {
        'n_estimators': [400,500,700],
        'max_features':['auto','sqrt'],
        'max_depth':[5,8,10,15,20,25,30],
        'min_samples_split':[5,8,10],
        'min_samples_leaf':[2,4]
    }

    grid_rf = GridSearchCV(estimator = rf,
                           param_grid = params_rf,
                           scoring ='neg_mean_squared_error',
                           cv = 3,
                           verbose = 1,
                           n_jobs = -1)

    grid_rf.fit(X_train, y_train.ravel())

    print('CV Score for best Random Forest Regressor model {:.3f}'.format(cv_score(grid_rf.
    best_model = grid_rf.best_estimator_
    return best_model


def get_extratree_regressor(X_train,y_train):
```

```
    extratrees = ExtraTreesRegressor(random_state= 0)

    params_extratrees = {
    }

    grid_extratrees = GridSearchCV(estimator = extratrees,
                       param_grid = params_extratrees,
                       scoring ='neg_mean_squared_error',
                       cv = 3,
                       verbose = 1,
                       n_jobs = -1)

    grid_extratrees.fit(X_train, y_train.ravel())

    print('CV Score for best Extra Trees Regressor model {:.3f}'.format(cv_score(grid_extra
    best_model = grid_extratrees.best_estimator_
    return best_model



def get_gbr_regressor(X_train,y_train):

    gbr = GradientBoostingRegressor(random_state=0)

    params_gbr = {
        'n_estimators': [700],
        'loss': ['ls', 'lad', 'huber', 'quantile'],
        'max_features': ['auto','sqrt','log2'],
        'max_depth':[3,5,8,10],
        'subsample':[0.8,1]
    }

    grid_gbr = GridSearchCV(estimator = gbr,
                            param_grid = params_gbr,
                            scoring ='neg_mean_squared_error',
                            cv = 3,
                            verbose = 1,
                            n_jobs = -1)

    grid_gbr.fit(X_train, y_train.ravel())

    print('CV Score for best Grandient Boost Regressor model: {:.3f}'.format(cv_score(grid_
    best_model = grid_gbr.best_estimator_
    return best_model

def rmse_dataframe(rmse):

    rmse_result = pd.DataFrame(rmse)
    rmse_result['model'] = ['linear_regression',
                            'ridge_regression',
                            'decision_tree',
                            'random_forest_regressor',
                            'extra_tree_regressor',
                            'gbr']
    rmse_result.columns = ['rmse','model']
    rmse_result = rmse_result[['model','rmse']]

    return rmse_result

def prediction_dataframe(li,y_test):
```

```
    df_predict = pd.DataFrame(li).T
    df_predict['y_Test'] = pd.DataFrame(y_test)
    df_predict.rename(columns={0:'linear_regression',
                               1:'ridge_regression',
                               2:'decision_tree',
                               3:'random_forest_regressor',
                               4:'extra_tree_regressor',
                               5:'gbr',
                               },
                              inplace=True)
    return df_predict


def model_selection(X,y, classifier):

    li = []
    rmse = []

    X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        test_size=0.30,
                                                        random_state=SEED)

    y_train, y_test = prep_target(queimadas[TARGET],y_train, y_test)

    for classifier in classifiers:
        pipe = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', classifier)])

        pipe.fit(X_train, y_train.ravel())

        print('\n',classifier)

        pred_test = pipe.predict(X_test)
        #pred_train = pipe.predict(X_train)
        rmse_result = np.sqrt(mean_squared_error(y_test, pred_test))

        rmse.append(rmse_result)
        li.append(pred_test)

        print('\n TEST - Root Mean Squared Error: : {:.3f}'.format(rmse_result))

    prediction_dataframe(li,y_test)

    return prediction_dataframe(li,y_test), rmse_dataframe(rmse)
```

In [3]:

```
#TARGET = ['frp']
#FEATURES = ['riscofogo','bioma','avg_temp_ar','avg_umd_ar','avg_vento_velo']

#FEATURES = ['diasemchuv','hora','mes','quadrimestre',
            #'avg_prep_total','avg_pressao_atm',
            #'avg_umd_ar','avg_temp_ar','avg_vento_velo']

TARGET = ['riscofogo']
FEATURES = ['diasemchuv',
                    'mes',
                    'quadrimestre',
                    'avg_pressao_atm',
                    'avg_umd_ar']
```

In [4]:

```
# In[3]: Ler dados
queimadas = read_csv()
```

In [5]:

```
# In[4]: Transformar dados
X_train, X_test, y_train, y_test, preprocessor = prep_pipeline(queimadas[FEATURES],queimada
```

In [8]:

```
# In[5]: Escolher Modelos - Regressão Linear

ln_model = get_linear_regressor(X_train,y_train)

ln_ridge_model = get_ridge_regression(X_train,y_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    1.3s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:    0.0s

CV Score for best Linear Regressor model: 0.190
Fitting 3 folds for each of 48 candidates, totalling 144 fits
CV Score for best Linear Ridge Regressor model: 0.190

[Parallel(n_jobs=-1)]: Done 144 out of 144 | elapsed:    1.2s finished
```

In [9]:

```
# In[6]: Escolher Modelos - Arvore de Decisão Regressão

dt_model = get_decisiontree_regressor(X_train,y_train)
```

```
Fitting 3 folds for each of 126 candidates, totalling 378 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  28 tasks      | elapsed:    0.0s

CV Score for best Decision Tree Regressor model: 0.153

[Parallel(n_jobs=-1)]: Done 378 out of 378 | elapsed:    0.4s finished
```

In [10]:

```
# In[7]:  Escolher Modelos - RandomForestRegressor
rf_model = get_rf_regressor(X_train,y_train)
```

```
Fitting 3 folds for each of 252 candidates, totalling 756 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    9.4s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:   58.8s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 756 out of 756 | elapsed:  5.4min finished

CV Score for best Random Forest Regressor model 0.147
```

In [11]:

```
# In[8]:  Escolher Modelos - ExtraTreeRegressor
extreg_model = get_extratree_regressor(X_train,y_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:    0.5s finished

CV Score for best Extra Trees Regressor model 0.151
```

In [12]:

```
# In[9]:  Escolher Modelos - Grandient Boost Regressor
gbr_model = get_gbr_regressor(X_train,y_train)
```

```
Fitting 3 folds for each of 96 candidates, totalling 288 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    9.5s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 288 out of 288 | elapsed:  3.3min finished

CV Score for best Grandient Boost Regressor model: 0.150
```

In [13]:

```
# In[10]:  Escolher Modelos - Grandient Boost Regressor
classifiers = [
    ln_model,
    ln_ridge_model,
    dt_model,
    rf_model,
    extreg_model,
    gbr_model
    ]

li, rmse = model_selection(queimadas[FEATURES],queimadas[TARGET],classifiers)
```

```
 LinearRegression()

 TEST - Root Mean Squared Error: : 0.189

 Ridge(alpha=1, random_state=0)

 TEST - Root Mean Squared Error: : 0.189

 DecisionTreeRegressor(max_depth=15, max_features='sqrt', min_samples_leaf=
2,
                       min_samples_split=10, random_state=0)

 TEST - Root Mean Squared Error: : 0.143

 RandomForestRegressor(max_depth=15, max_features='sqrt', min_samples_leaf=
2,
                       min_samples_split=5, n_estimators=500, random_state=0)

 TEST - Root Mean Squared Error: : 0.138

 ExtraTreesRegressor(random_state=0)

 TEST - Root Mean Squared Error: : 0.140

 GradientBoostingRegressor(max_depth=5, max_features='auto', n_estimators=70
0,
                           random_state=0, subsample=1)

 TEST - Root Mean Squared Error: : 0.139
```

In [15]:

```
# In[11]: feature Importance

feature_importances = rf_model.feature_importances_

data = {'Features':['diasemchuv',
                    'mes',
                    'quadrimestre',
                    'avg_pressao_atm',
                    'avg_umd_ar'],
        'Ratings':feature_importances}

df_data = pd.DataFrame(data, index =['0', '1', '2', '3','4'])
df_data
```

Out[15]:

	Features	Ratings
0	diasemchuv	0.191630
1	mes	0.122112
2	quadrimestre	0.262444
3	avg_pressao_atm	0.207471
4	avg_umd_ar	0.216342

In [ ]:

```
# In[10]: feature Importance
'''
fn=FEATURES
cn=TARGET
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rf_model.estimators_[0],
               feature_names = fn,
               class_names=cn,
               rounded=True,
               filled = True);
fig.savefig('rf_individualtree.png')

'''
```