Forestfires

PREDICT THE BURNED AREA OF FOREST FIRES WITH NEURAL NETWORKS

In [1]:
```
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



sns.set_style('darkgrid')

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
forestfires = pd.read_csv("forestfires.csv")
forestfires.head()
```

Out[2]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthfeb	monthjan	monthjul	monthjun	monthmar	monthmay	mo
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	0	0	1	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	0	0	0	0	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	0	0	0	0	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	0	0	1	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	0	0	1	0	

5 rows × 31 columns

In [3]:
```
forestfires.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 517 entries, 0 to 516
Data columns (total 31 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   month          517 non-null    object 
 1   day            517 non-null    object 
 2   FFMC           517 non-null    float64
 3   DMC            517 non-null    float64
 4   DC             517 non-null    float64
 5   ISI            517 non-null    float64
 6   temp           517 non-null    float64
 7   RH             517 non-null    int64  
 8   wind           517 non-null    float64
 9   rain           517 non-null    float64
 10  area           517 non-null    float64
 11  dayfri         517 non-null    int64  
 12  daymon         517 non-null    int64  
 13  daysat         517 non-null    int64  
 14  daysun         517 non-null    int64  
 15  daythu         517 non-null    int64  
 16  daytue         517 non-null    int64  
 17  daywed         517 non-null    int64  
 18  monthapr       517 non-null    int64  
 19  monthaug       517 non-null    int64  
 20  monthdec       517 non-null    int64  
 21  monthfeb       517 non-null    int64  
 22  monthjan       517 non-null    int64  
 23  monthjul       517 non-null    int64  
 24  monthjun       517 non-null    int64  
 25  monthmar       517 non-null    int64  
 26  monthmay       517 non-null    int64  
 27  monthnov       517 non-null    int64  
 28  monthoct       517 non-null    int64  
 29  monthsep       517 non-null    int64  
 30  size_category  517 non-null    object 
dtypes: float64(8), int64(20), object(3)
memory usage: 125.3+ KB
```

In [4]:
```
numerical_feature = forestfires.describe(include=["int", "float"]).columns

print(list(numerical_feature))
```

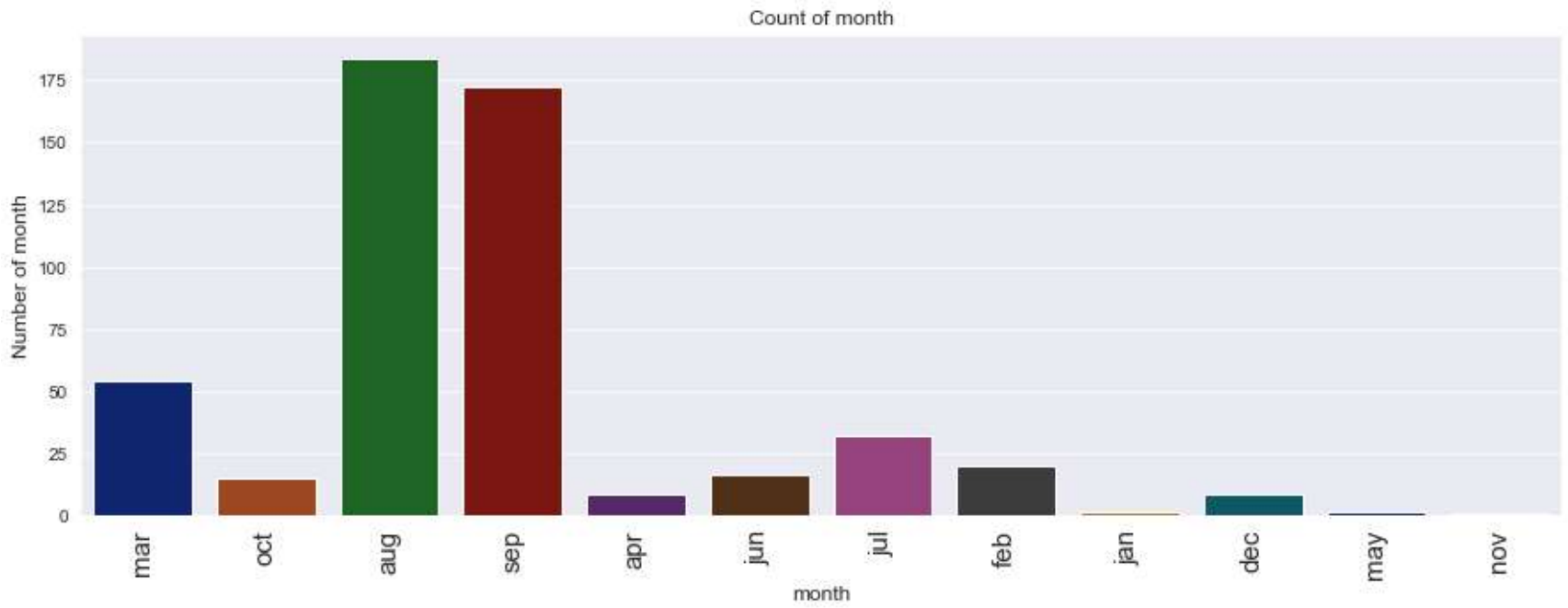
```
['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'wind', 'rain', 'area']
```

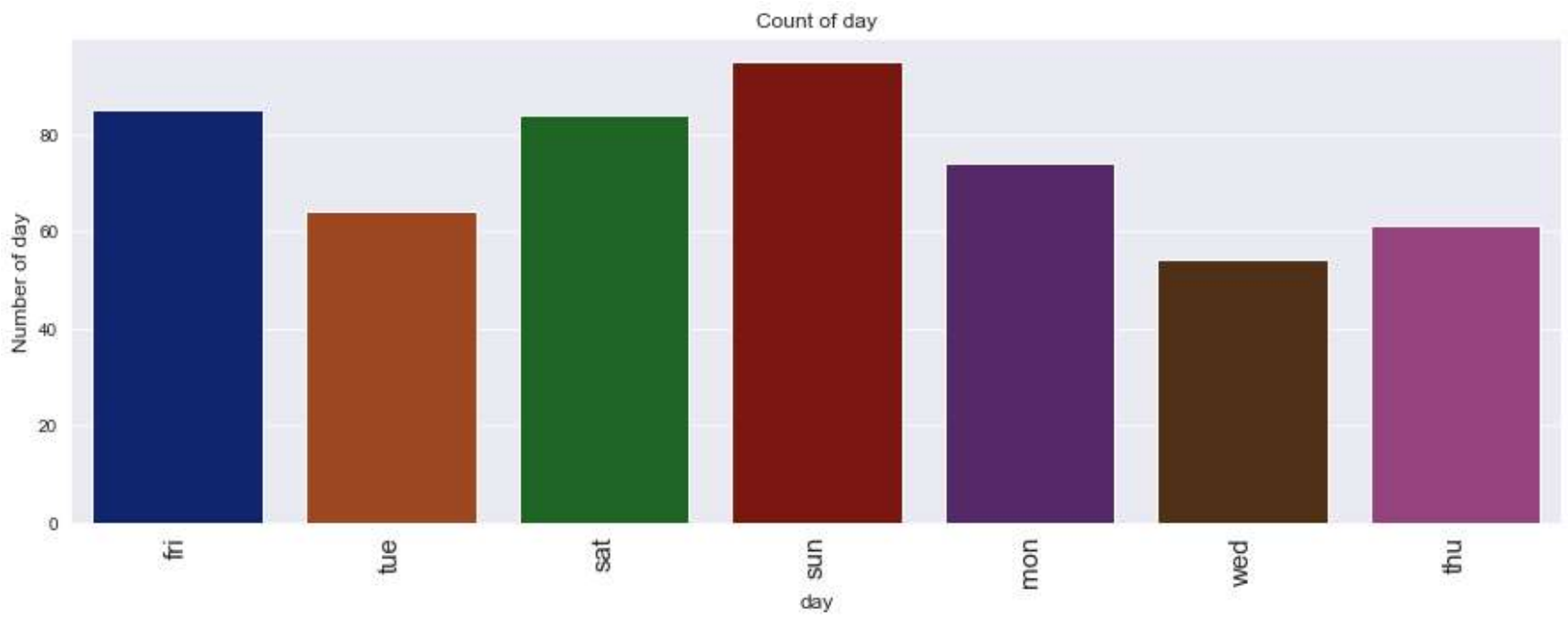
In [5]:
```
categorical_features = forestfires.describe(include=["object"]).columns

print(list(categorical_features))
```

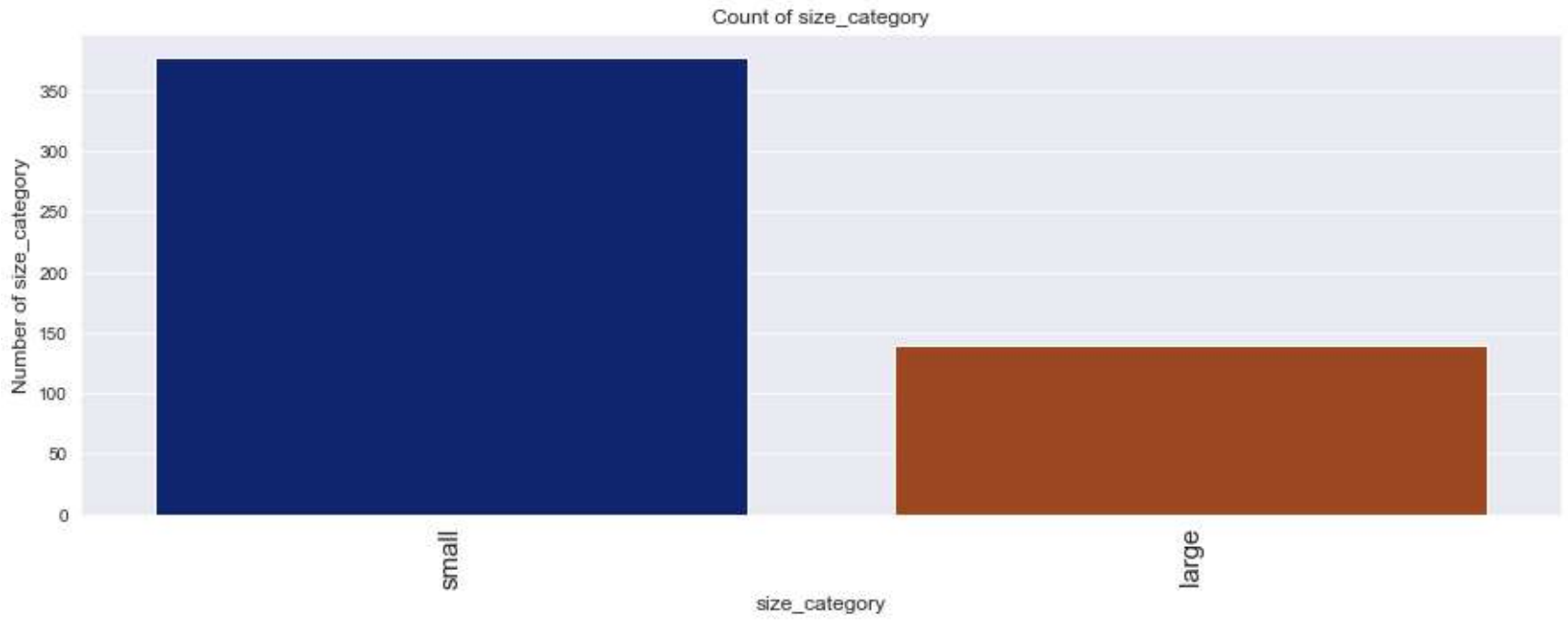
```
['month', 'day', 'size_category']
```

In [6]:
```
print(categorical_features)



for idx, column in enumerate(categorical_features):
    plt.figure(figsize=(15, 5))
    df = forestfires.copy()
    unique = df[column].value_counts(ascending=True);

    #plt.subplot(1, len(categorical_features), idx+1)
    plt.title("Count of "+ column)
    sns.countplot(data=forestfires, x=column,palette = "dark")
    #plt.bar(unique.index, unique.values);
    plt.xticks(rotation = 90, size = 15)

    plt.xlabel(column, fontsize=12)
    plt.ylabel("Number of "+ column, fontsize=12)
    plt.show()
```

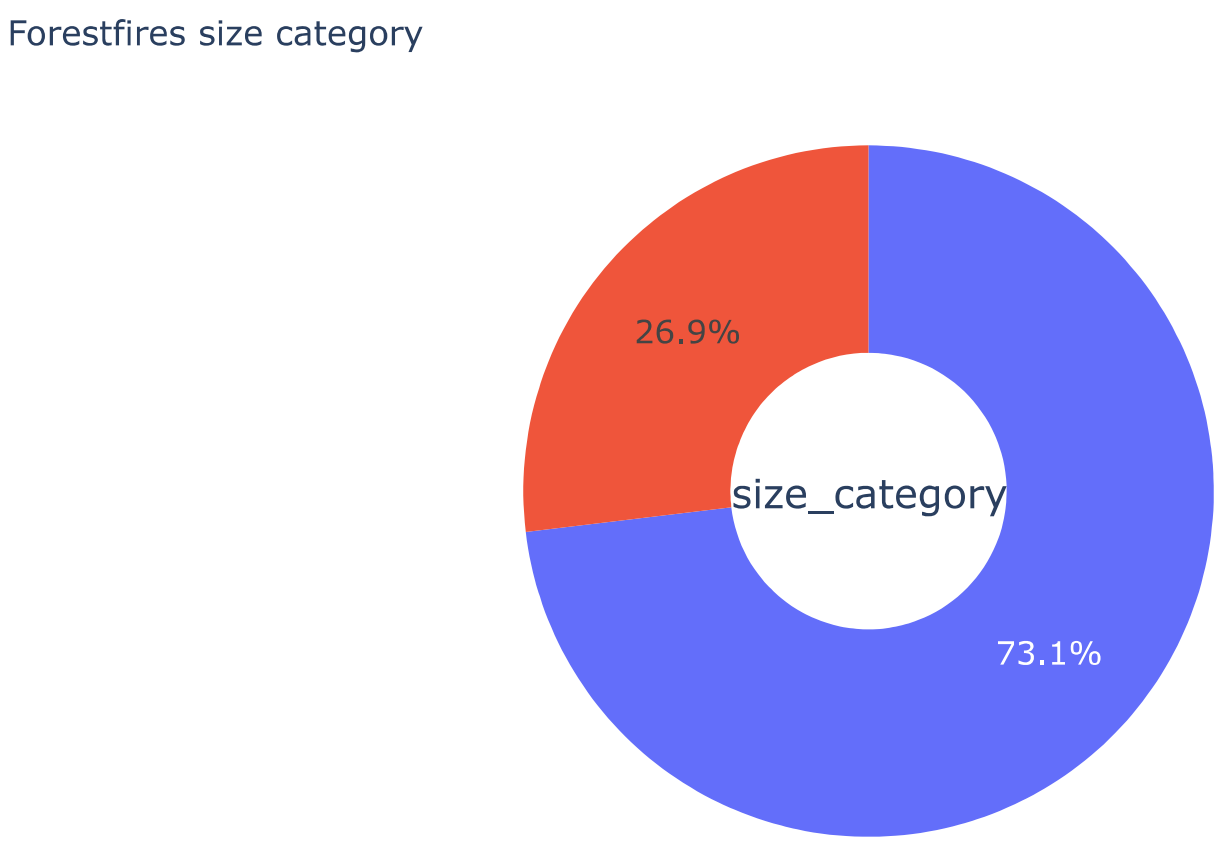
```
Index(['month', 'day', 'size_category'], dtype='object')
```



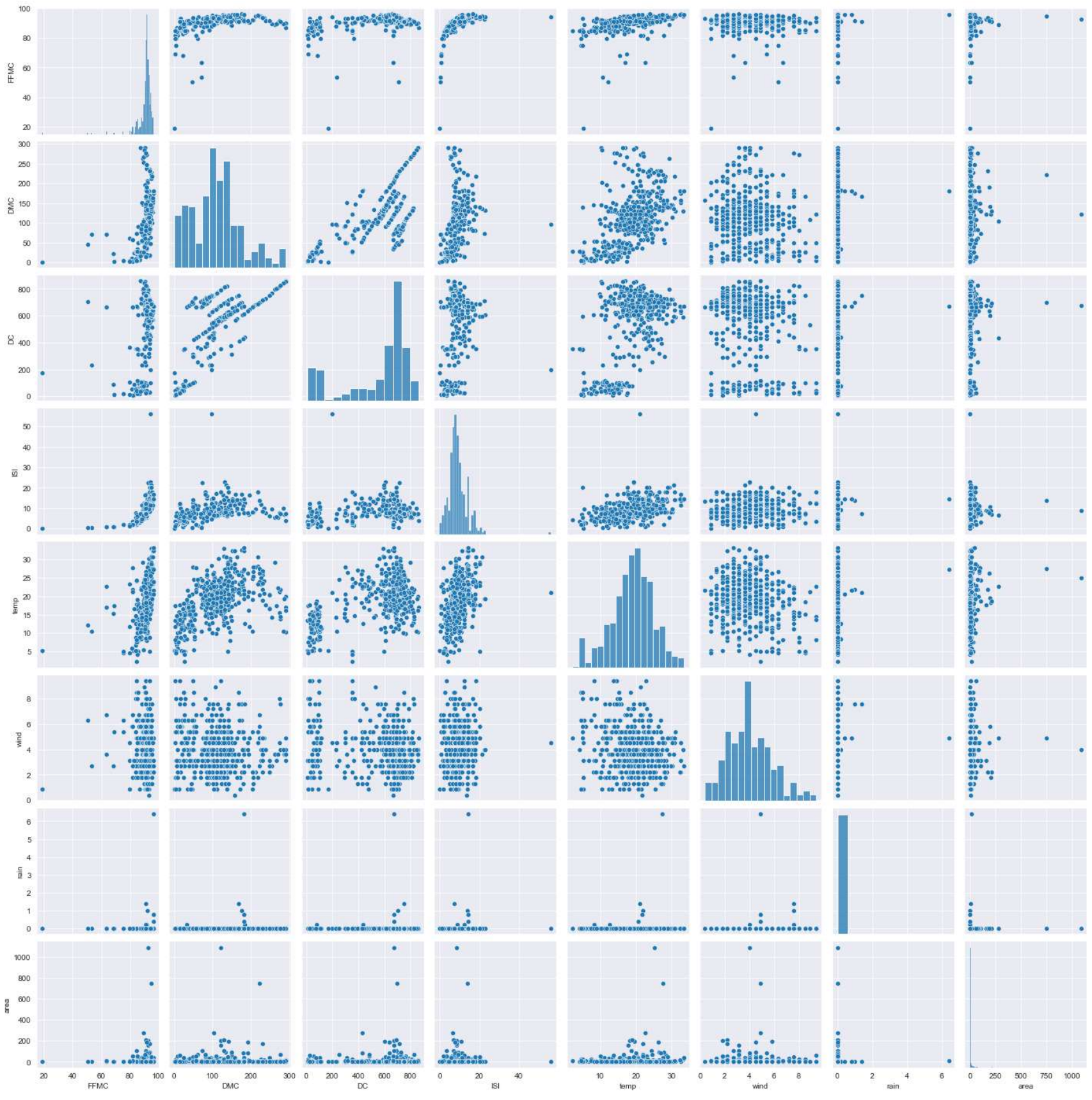




In [7]:
```
type_ = ['small', 'large']
fig = make_subplots(rows=1, cols=1)

fig.add_trace(go.Pie(labels=type_, values=forestfires['size_category'].value_counts(), name="size_category"))

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Forestfires size category",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='size_category', x=0.5, y=0.5, font_size=20, showarrow=False)])
fig.show()
```



In [8]:
```
sns.set_style('darkgrid')
sns.pairplot(forestfires[numerical_feature])
plt.show()
```



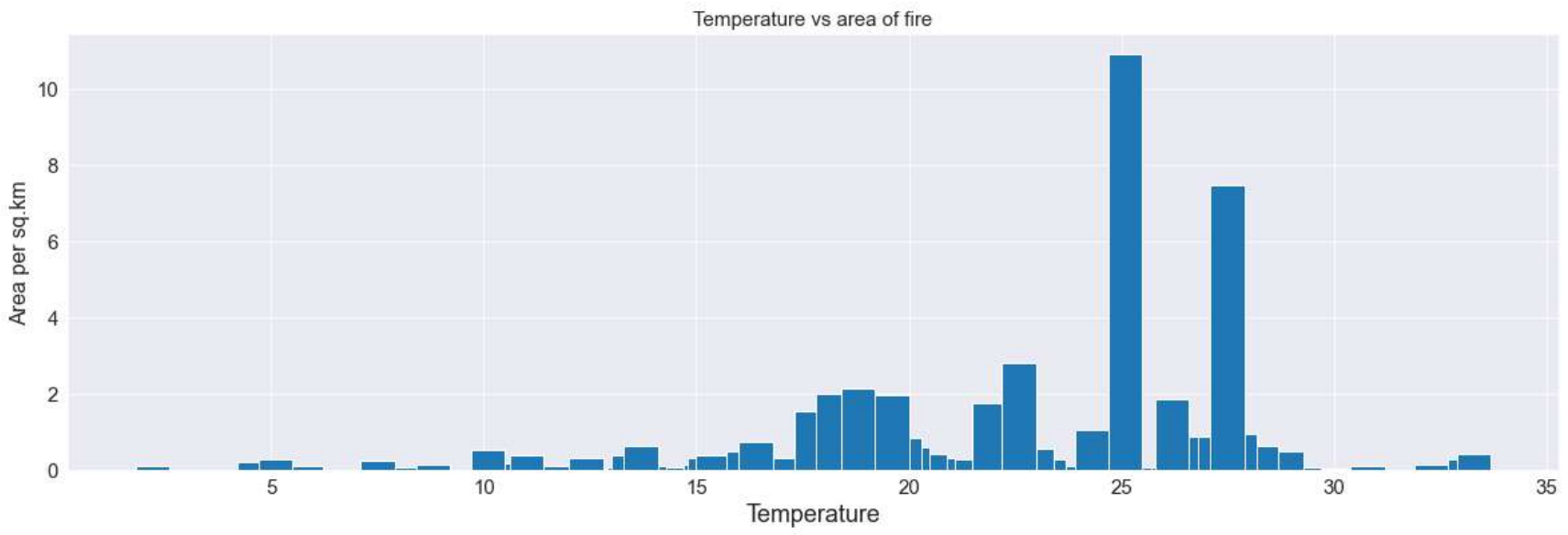
In [9]:
```
forestfires['area_km'] = forestfires['area'] / 100

forestfires.head()
```

Out[9]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	...	monthjan	monthjul	monthjun	monthmar	monthmay	monthnov	mo
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	...	0	0	0	1	0	0	
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	...	0	0	0	0	0	0	
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	...	0	0	0	0	0	0	
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	...	0	0	0	1	0	0	
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	...	0	0	0	1	0	0	

5 rows × 32 columns

In [10]:
```
highest_fire_area = forestfires.sort_values(by="area_km", ascending=True)
plt.figure(figsize=(20, 6))

plt.title("Temperature vs area of fire" , fontsize=15)
plt.bar(highest_fire_area['temp'], highest_fire_area['area_km'])

plt.xticks(size = 15)
plt.yticks(size = 15)
plt.xlabel('Temperature',fontsize=18)
plt.ylabel('Area per sq.km', fontsize=16)
plt.show()
```



In [11]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation,Layer,Lambda
```

In [12]:
```
forestfires.drop(["month","day"],axis=1,inplace = True)
```

In [39]:
```
#Label encoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
data_copy= forestfires.copy()
le = LabelEncoder()
# Label Encoding will be used for columns with 2 or less unique values
le_count = 0
for col in data_copy.columns[1:]:
    if data_copy[col].dtype == 'object':
        if len(list(data_copy[col].unique())) <= 2:
            le.fit(data_copy[col])
            data_copy[col] = le.transform(data_copy[col])
            le_count += 1
print('{} columns were label encoded.'.format(le_count))
```

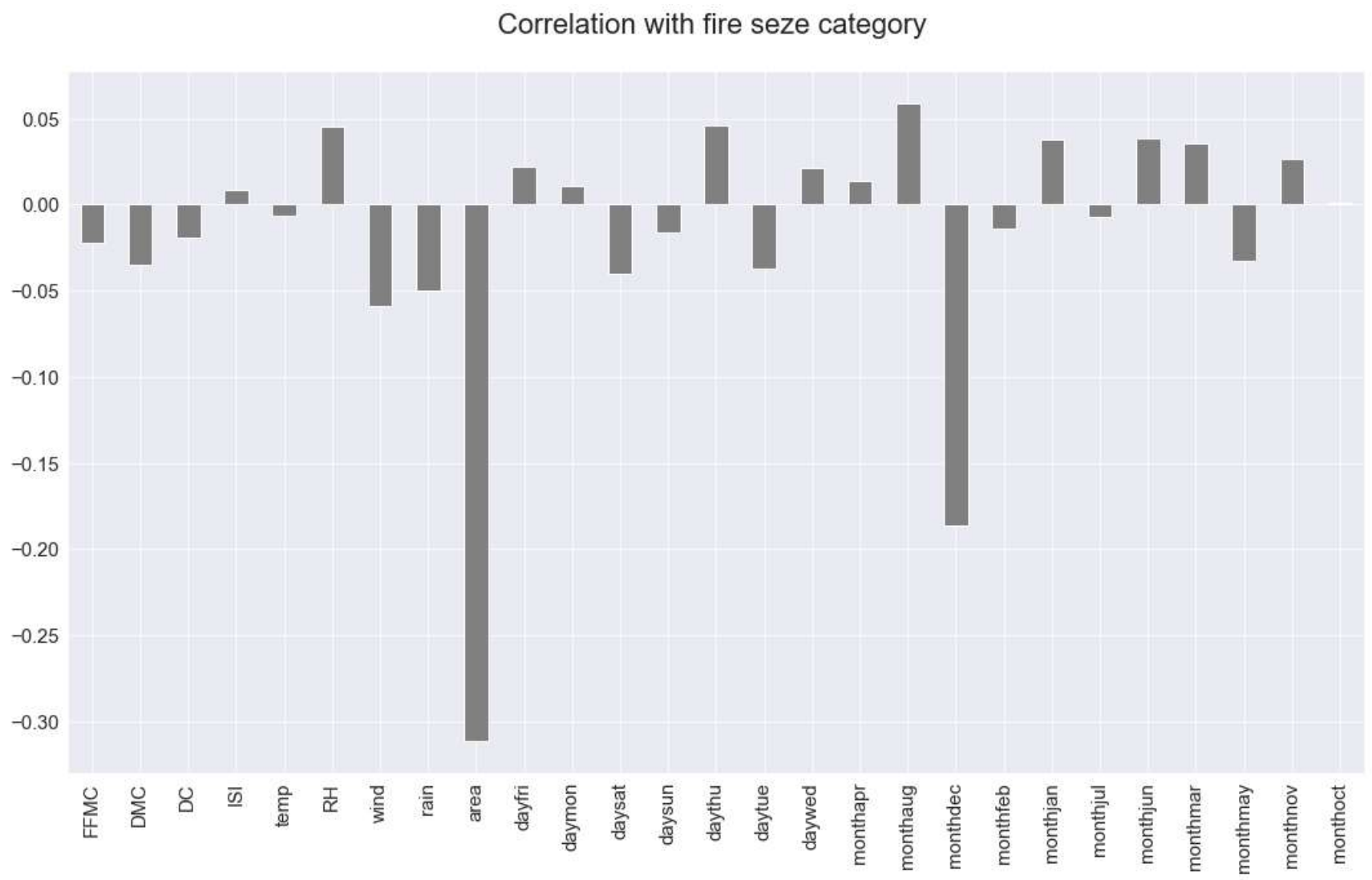
```
1 columns were label encoded.
```

In [50]:
```
data_copy.size_category.unique()
```

Out[50]:
```
0      1
1      1
2      1
3      1
4      1
      ..
512    0
513    0
514    0
515    1
516    1
Name: size_category, Length: 517, dtype: int32
```

In [41]:
```
# correlation woith forest fire size category

data2 = data_copy.iloc[:,:-3]

correlations = data2.corrwith(data_copy["size_category"])
correlations = correlations[correlations!=1]
positive_correlations = correlations[correlations >0].sort_values(ascending = False)
negative_correlations =correlations[correlations<0].sort_values(ascending = False)

correlations.plot.bar(
        figsize = (18, 10),
        fontsize = 15,
        color = 'grey',
        rot = 90, grid = True)
plt.title('Correlation with fire seze category \n',
horizontalalignment="center", fontstyle = "normal",
fontsize = "22", fontfamily = "sans-serif")
```

Out[41]: Text(0.5, 1.0, 'Correlation with fire seze category \n')

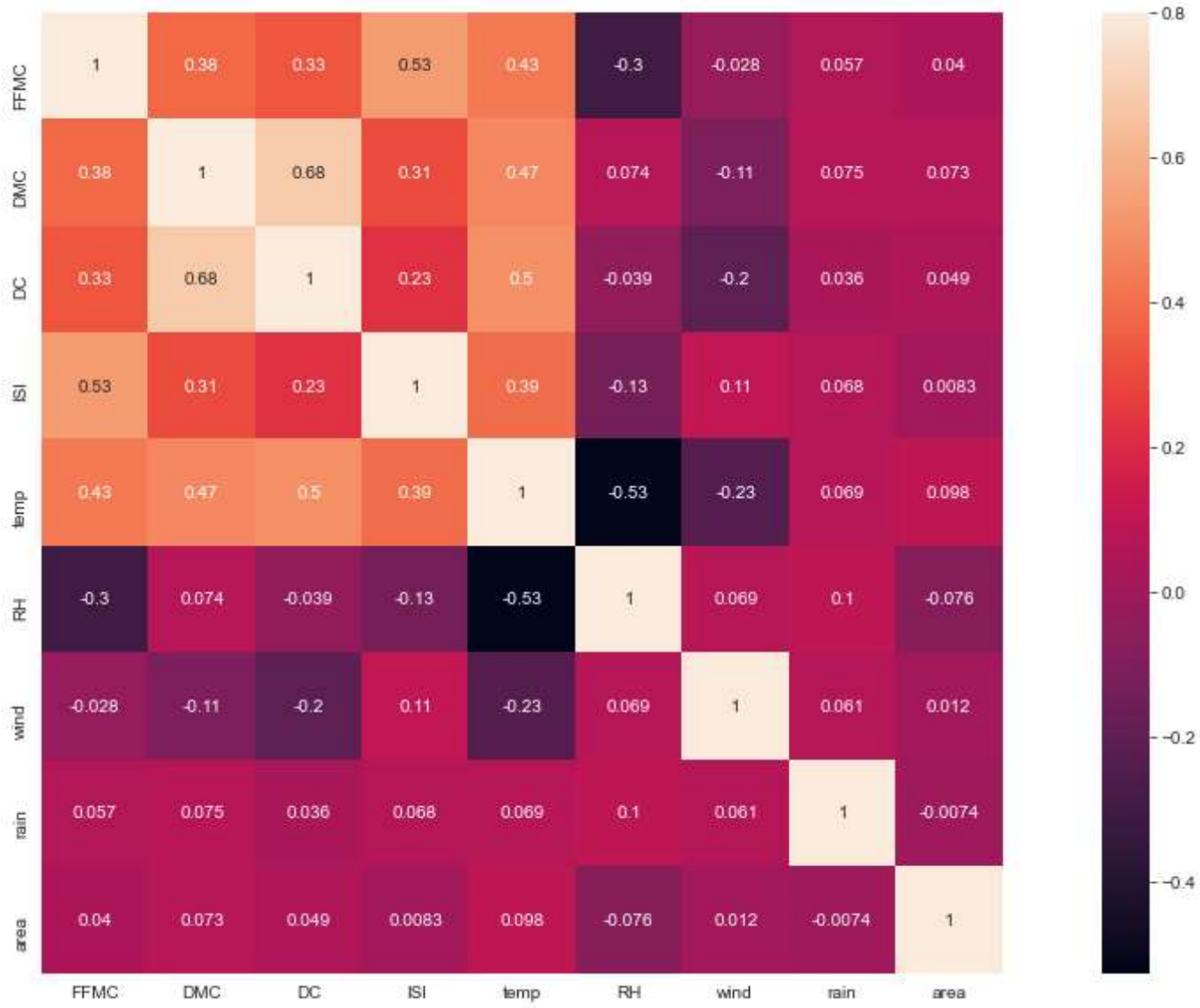


In [43]:
```
forestfires_ = pd.read_csv("forestfires.csv")
forestfires_.iloc[:,:-20].head()
```

Out[43]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

In [44]:
```
import seaborn as sns
import matplotlib.pyplot as pplt
#correlation matrix
corrmat = forestfires_.iloc[:,:-20].corr()
f, ax = pplt.subplots(figsize=(20, 10))
sns.heatmap(corrmat, vmax=.8, square=True, annot=True);
```



In [283]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.utils import np_utils
from keras.constraints import maxnorm
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
#from keras.wrappers.scikit_learn import KerasRegressor, KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, KFold, RandomizedSearchCV
from keras.optimizers import Adam
```

In [47]:
```
col = data2.columns
col
```

Out[47]:
```
Index(['FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain', 'area',
       'dayfri', 'daymon', 'daysat', 'daysun', 'daythu', 'daytue', 'daywed',
       'monthapr', 'monthaug', 'monthdec', 'monthfeb', 'monthjan', 'monthjul',
       'monthjun', 'monthmar', 'monthmay', 'monthnov', 'monthoct'],
      dtype='object')
```

In [232]:
```
"""def norm_func(i):
    x = (i-i.min())/(i.max()-i.min())
    return (x)"""


X = data_copy.iloc[:,0:28]
y = data_copy.iloc[:,28]

scaler = StandardScaler()
X = scaler.fit_transform(X)

#y =scaler.fit_transform(y.values.reshape(len(y),1))[:,0]
```

In [ ]:
```

```

In [233]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test= train_test_split(X,y, test_size=0.3,stratify = y)
```

In [93]:
```
print('Shape of X_train: ', X_train.shape)
print('Shape of X_test: ', X_test.shape)
print('Shape of y_train: ', y_train.shape)
print('Shape of y_test: ', y_test.shape)
```

```
Shape of X_train:  (361, 28)
Shape of X_test:  (156, 28)
Shape of y_train:  (361,)
Shape of y_test:  (156,)
```

In [ ]:
```
```

In [ ]:
```
# create model
model = Sequential()
model.add(Dense(28, input_dim=28, kernel_initializer='uniform', activation='relu'))
model.add(Dense(50, kernel_initializer='uniform', activation='relu'))
model.add(Dense(50, kernel_initializer='uniform', activation='relu'))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))

#Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
model.fit(X_train,y_train, epochs=500, batch_size=10)
```

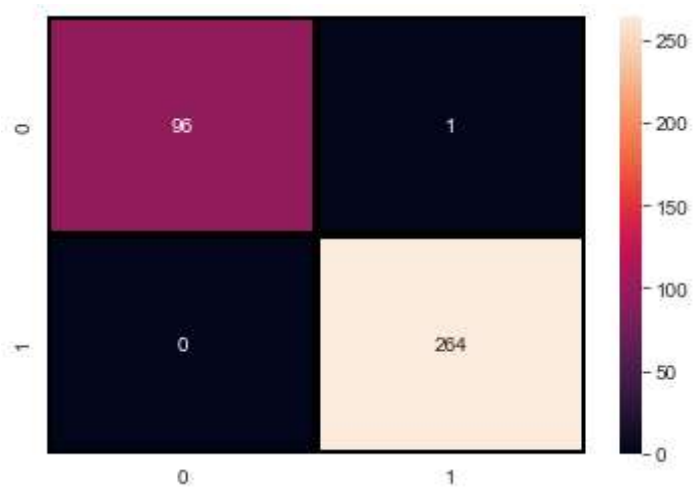
```
Epoch 1/500
37/37 [==============================] - 1s 2ms/step - loss: 0.6799 - accuracy: 0.0000e+00
Epoch 2/500
37/37 [==============================] - 0s 2ms/step - loss: 0.3923 - accuracy: 0.0000e+00
Epoch 3/500
37/37 [==============================] - 0s 2ms/step - loss: -0.3392 - accuracy: 0.0000e+00
Epoch 4/500
37/37 [==============================] - 0s 2ms/step - loss: -0.7938 - accuracy: 0.0000e+00
Epoch 5/500
37/37 [==============================] - 0s 2ms/step - loss: -2.8655 - accuracy: 0.0000e+00
Epoch 6/500
37/37 [==============================] - 0s 2ms/step - loss: -5.2578 - accuracy: 0.0000e+00
Epoch 7/500
37/37 [==============================] - 0s 2ms/step - loss: -16.8567 - accuracy: 0.0000e+00
Epoch 8/500
37/37 [==============================] - 0s 2ms/step - loss: -62.0542 - accuracy: 0.0000e+00
Epoch 9/500
37/37 [==============================] - 0s 2ms/step - loss: -138.9226 - accuracy: 0.0000e+00
Epoch 10/500
```

In [145]:
```
def base_model(hidden_dim):
    model = Sequential()
    for i in range(1,len(hidden_dim)-1):
        if (i==1):
            model.add(Dense(hidden_dim[i],input_dim=hidden_dim[0],activation="relu"))
        else:
            model.add(Dense(hidden_dim[i],activation="relu"))
    model.add(Dense(hidden_dim[-1],kernel_initializer="normal",activation="sigmoid"))
    model.compile(loss="binary_crossentropy",optimizer = "rmsprop",metrics = ["accuracy"])
    return model

#y_train = pd.DataFrame(y_train)

model_ = base_model([28,50,50,1])
model_.fit(np.array(X_train),np.array(y_train),epochs=380)
pred_train = model_.predict(np.array(X_train))
```

```
Epoch 2/380
12/12 [==============================] - 0s 1ms/step - loss: 0.6086 - accuracy: 0.7287
Epoch 3/380
12/12 [==============================] - 0s 1ms/step - loss: 0.5678 - accuracy: 0.7408
Epoch 4/380
12/12 [==============================] - 0s 1ms/step - loss: 0.5431 - accuracy: 0.7635
Epoch 5/380
12/12 [==============================] - 0s 1ms/step - loss: 0.5652 - accuracy: 0.7242
Epoch 6/380
12/12 [==============================] - 0s 1ms/step - loss: 0.5601 - accuracy: 0.7309
Epoch 7/380
12/12 [==============================] - 0s 2ms/step - loss: 0.5440 - accuracy: 0.7329
Epoch 8/380
12/12 [==============================] - 0s 2ms/step - loss: 0.5258 - accuracy: 0.7576
Epoch 9/380
12/12 [==============================] - 0s 1ms/step - loss: 0.5019 - accuracy: 0.7750
Epoch 10/380
12/12 [==============================] - 0s 1ms/step - loss: 0.4931 - accuracy: 0.7704
Epoch 11/380
12/12 [==============================] - 0s 1ms/step - loss: 0.4616 - accuracy: 0.7894
```

In [146]:
```
# Model evaluation: model 1

pred_test = model_.predict(np.array(X_test))
pred_test = pd.Series([i[0] for i in pred_test])
pred_train_ = pd.Series([i[0] for i in pred_train])

pred_y = []
pred_x = []
for i in pred_test:
    if i>0.5:
        pred_y.append(1)
    else:
        pred_y.append(0)

for i in pred_train_:
    if i>0.5:
        pred_x.append(1)
    else:
        pred_x.append(0)



#train data
from sklearn.metrics import confusion_matrix
sns.heatmap(confusion_matrix(y_train, pred_x),annot=True,fmt = "d",linecolor="k",linewidths=3)
```

Out[146]: <AxesSubplot:>

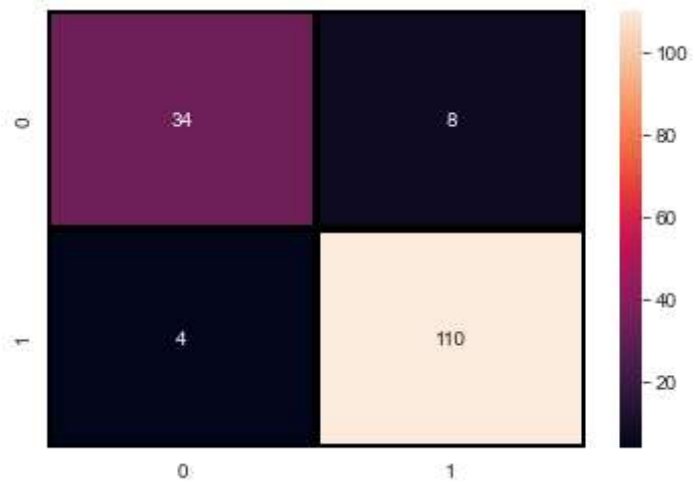


In [147]:
```
# evaluate the model
scores = model_.evaluate(X_test, y_test)
print((model_.metrics_names[1]))
```

```
5/5 [==============================] - 0s 2ms/step - loss: 1.9707 - accuracy: 0.9231
accuracy
```

In [148]:
```
#test data
sns.heatmap(confusion_matrix(y_test, pred_y),annot=True,fmt = "d",linecolor="k",linewidths=3)
```

Out[148]: <AxesSubplot:>



Hyper parameter tuning

In [229]:
```
def base_model2(hidden_dim):
    model = Sequential()
    for i in range(1,len(hidden_dim)-1):
        if (i==1):
            model.add(Dense(hidden_dim[i],input_dim=hidden_dim[0],activation="relu"))
        else:
            model.add(Dense(hidden_dim[i],activation="relu"))
    model.add(Dense(hidden_dim[-1],kernel_initializer="uniform",activation="sigmoid"))
    adam=Adam(learning_rate=0.01)
    model.compile(loss="binary_crossentropy",optimizer = "adam",metrics = ["accuracy"])
    return model

#y_train = pd.DataFrame(y_train)

"""model_2 = base_model2([28,50,50,1])
model_2.fit(np.array(X_train),np.array(y_train),epochs=380)
pred_train = model_2.predict(np.array(X_train))
    """


# Create the model
#model = base_model2([28,50,50,1])

k_model = KerasClassifier(build_fn=lambda:base_model2([28,50,50,1]), verbose=0)
```

In [ ]:
```
batch_size = [10,20,40]
epochs = [10,50,100,500,700,1000]

# Make a dictionary of the grid search parameters
param_grid = dict(batch_size = batch_size,epochs = epochs)

# Build and fit the GridSearchCV
Grid = GridSearchCV(estimator = k_model,param_grid = param_grid,cv = 10)
grid_result = Grid.fit(X_train,y_train)
```

In [207]:
```
print('Best result : {}, Using parameters{}'.format(grid_result.best_score_,grid_result.best_params_))

stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
means = grid_result.cv_results_['mean_test_score']


for mean, stdev, param in zip(means, stds, params):
  print('{},{} with: {}'.format(mean, stdev, param))
```

```
Best result : 0.9168949842453002, Using parameters{'batch_size': 20, 'epochs': 1000}
0.7838280081748963,0.03734608655221178 with: {'batch_size': 10, 'epochs': 10}
0.8752283096313477,0.02812187737273547 with: {'batch_size': 10, 'epochs': 50}
0.8835996985435486,0.024434864498214096 with: {'batch_size': 10, 'epochs': 100}
0.9057838797569275,0.010644568817571922 with: {'batch_size': 10, 'epochs': 500}
0.889193308353424,0.015226699279636447 with: {'batch_size': 10, 'epochs': 700}
0.9057838797569275,0.013800993178941281 with: {'batch_size': 10, 'epochs': 1000}
0.7422754883766174,0.025137099116127395 with: {'batch_size': 20, 'epochs': 10}
0.8530821800231934,0.019424074268920264 with: {'batch_size': 20, 'epochs': 50}
0.8696727514266968,0.032728501410022016 with: {'batch_size': 20, 'epochs': 100}
0.8947108149528503,0.02267119084256414 with: {'batch_size': 20, 'epochs': 500}
0.905821931362152,0.022195306695884435 with: {'batch_size': 20, 'epochs': 700}
0.9168949842453002,0.02152140603359116 with: {'batch_size': 20, 'epochs': 1000}
0.7311643719673157,0.028279398029079344 with: {'batch_size': 40, 'epochs': 10}
0.8281582832336426,0.0361498382504406 with: {'batch_size': 40, 'epochs': 50}
0.8503424644470214,0.0335961773110472 with: {'batch_size': 40, 'epochs': 100}
0.8780821919441223,0.020586463168486377 with: {'batch_size': 40, 'epochs': 500}
0.8864155292510987,0.022268576142584158 with: {'batch_size': 40, 'epochs': 700}
0.8835996985435486,0.031350684842788995 with: {'batch_size': 40, 'epochs': 1000}
```

Parameters selection and Hyper parameter tuning

In [278]:
```
data = pd.read_csv("forestfires.csv")
data.drop(['dayfri', 'daymon', 'daysat', 'daysun', 'daythu','daytue', 'daywed', 'monthapr', 'monthaug', 'monthde
                  'monthfeb','monthjan', 'monthjul', 'monthjun', 'monthmar', 'monthmay', 'monthnov','monthoct','

data.month.replace(('jan','feb','mar','apr','may','jun','jul','aug','sep','oct','nov','dec'),
                           (1,2,3,4,5,6,7,8,9,10,11,12), inplace=True)
data.day.replace(('mon','tue','wed','thu','fri','sat','sun'),(1,2,3,4,5,6,7), inplace=True)
data.head()
```

Out[278]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	size_category
0	3	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	small
1	10	2	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	small
2	10	6	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	small
3	3	5	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	small
4	3	7	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	small

In [279]:
```
data.size_category.replace(('small', 'large'), (0, 1), inplace = True)
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()


scaled_features=scaler.fit_transform(data.drop('size_category',axis=1))
data_head=pd.DataFrame(scaled_features,columns=data.columns[:-1])
data_head
```

Out[279]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	-1.968443	0.357721	-0.805959	-1.323326	-1.830477	-0.860946	-1.842640	0.411724	1.498614	-0.073268	-0.202020
1	1.110120	-1.090909	-0.008102	-1.179541	0.488891	-0.509688	-0.153278	-0.692456	-1.741756	-0.073268	-0.202020
2	1.110120	0.840597	-0.008102	-1.049822	0.560715	-0.509688	-0.739383	-0.692456	-1.518282	-0.073268	-0.202020
3	-1.968443	0.357721	0.191362	-1.212361	-1.898266	-0.004756	-1.825402	3.233519	-0.009834	0.603155	-0.202020
4	-1.968443	1.323474	-0.243833	-0.931043	-1.798600	0.126966	-1.291012	3.356206	-1.238940	-0.073268	-0.202020
...	...	...	...	...	...	...	...	...	...	...	...
512	0.230531	1.323474	-1.640083	-0.846648	0.474768	-1.563460	1.536084	-0.753800	-0.736124	-0.073268	-0.100753
513	0.230531	1.323474	-1.640083	-0.846648	0.474768	-1.563460	0.519019	1.638592	0.995798	-0.073268	0.651674
514	0.230531	1.323474	-1.640083	-0.846648	0.474768	-1.563460	0.398350	1.577248	1.498614	-0.073268	-0.026532
515	0.230531	0.840597	0.680957	0.549003	0.269382	0.500176	1.156839	-0.140366	-0.009834	-0.073268	-0.202020
516	1.549915	-1.090909	-2.020879	-1.685913	-1.780442	-1.739089	-1.222058	-0.815143	0.269509	-0.073268	-0.202020

517 rows × 11 columns

In [280]:
```
x_train, x_test, y_train, y_test = train_test_split(data_head,data['size_category'], test_size=0.3, random_state
```

In [281]:
```
data.head()
```

Out[281]:

	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area	size_category
0	3	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0	0
1	10	2	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0	0
2	10	6	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0	0
3	3	5	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0	0
4	3	7	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0	0

In [282]:
```
def create_model(learning_rate,dropout_rate,activation_function,init,neuron1,neuron2):
    model = Sequential()
    model.add(Dense(neuron1,input_dim = 11,kernel_initializer = init,activation = activation_function))
    model.add(Dropout(dropout_rate))
    model.add(Dense(neuron2,input_dim = neuron1,kernel_initializer = init,activation = activation_function))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1,activation = 'sigmoid'))

    adam=Adam(learning_rate = learning_rate)
    model.compile(loss = 'binary_crossentropy',optimizer = adam,metrics = ['accuracy'])
    return model

# Create the model

model = KerasClassifier(build_fn = create_model,verbose = 0)

# Define the grid search parameters

batch_size = [10,20,40]
epochs = [10,50,100]
learning_rate = [0.001,0.01,0.1]
dropout_rate = [0.0,0.1,0.2]
activation_function = ['softmax','relu','tanh','linear']
init = ['uniform','normal','zero']
neuron1 = [4,8,16]
neuron2 = [2,4,8]

# Make a dictionary of the grid search parameters

param_grids = dict(batch_size = batch_size,epochs = epochs,learning_rate = learning_rate,dropout_rate = dropout_
                   activation_function = activation_function,init = init,neuron1 = neuron1,neuron2 = neuron2)

# Build and fit the GridSearchCV

grid = GridSearchCV(estimator = model,param_grid = param_grids,cv = KFold(),verbose = 10)
grid_result = grid.fit(x_train, y_train)

# Summarize the results
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{},{} with: {}'.format(mean, stdev, param))
```

In [286]:
```
print('Best : {}, using {}'.format(grid_result.best_score_,grid_result.best_params_))
```

```
Best : 0.9944444417953491, using {'activation_function': 'tanh', 'batch_size': 40, 'dropout_rate': 0.1, 'epoch
s': 100, 'init': 'normal', 'learning_rate': 0.01, 'neuron1': 8, 'neuron2': 2}
```