Project 2: Fire in the nature park

Intelligent Data Analysis & Machine Learning I

Jaesub Kim
University of Potsdam
M. Sc. Bioinformatics

07. 08. 2024

Data Preperation

```python
import numpy as np
import pandas as pd

original_df = pd.read_csv('fires.csv',sep=',',header=0)
header = original_df.columns
print(header)
original_df.head()
```

✓ 2.4s

```
Index(['X', 'Y', 'month', 'day', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH',
       'wind', 'rain', 'area'],
      dtype='object')
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

- Fire data: 13 Features
- Two categorical features: ‘month’ and ‘day’
- Objective: prediction of the brunt forest ‘area’

Data Preprocessing

```python
from sklearn.preprocessing import LabelEncoder

# transform the 'area' column to log(area+1)
df = original_df.copy()
original_area = df['area']
df['area'] = np.log(df['area']+1)

print(df)

num_df = df.copy()

label_encoders = {} # Dictionary initialization
for column in ['month', 'day']:
    label_encoders[column] = LabelEncoder()
    num_df[column] = label_encoders[column].fit_transform(df[column])
    # Convert each string to an integer. Days of the week and months are automatically converted to integers.
```

```
     X  Y month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain      area
0    7  5   mar  fri  86.2   26.2   94.3   5.1   8.2  51   6.7   0.0  0.000000
1    7  4   oct  tue  90.6   35.4  669.1   6.7  18.0  33   0.9   0.0  0.000000
2    7  4   oct  sat  90.6   43.7  686.9   6.7  14.6  33   1.3   0.0  0.000000
3    8  6   mar  fri  91.7   33.3   77.5   9.0   8.3  97   4.0   0.2  0.000000
4    8  6   mar  sun  89.3   51.3  102.2   9.6  11.4  99   1.8   0.0  0.000000
..  .. ..   ...  ...   ...    ...    ...   ...   ...  ..   ...   ...       ...
512  4  3   aug  sun  81.6   56.7  665.6   1.9  27.8  32   2.7   0.0  2.006871
513  2  4   aug  sun  81.6   56.7  665.6   1.9  21.9  71   5.8   0.0  4.012592
514  7  4   aug  sun  81.6   56.7  665.6   1.9  21.2  70   6.7   0.0  2.498152
515  1  4   aug  sat  94.4  146.0  614.7  11.3  25.6  42   4.0   0.0  0.000000
516  6  3   nov  tue  79.5    3.0  106.7   1.1  11.8  31   4.5   0.0  0.000000

[517 rows x 13 columns]
```

- Log-transformation of feature ‘area’
- Two categorical features are transformed to integer for further process
- `LabelEncoder()` maps an unique integer value to categorical value

Data Preprocessing

```python
print("Checking for duplicated rows considering all columns:")
duplicates = num_df[num_df.duplicated(keep=False)]
if not duplicates.empty:
    print(duplicates)
else:
    print("No duplicated data found when considering all columns.")


num_df = num_df.drop_duplicates()
print("duplicate rows removed")

print("\n------------------------\n")

# to check if there is any missing value in the data
print(df.isnull().sum())
# no missing data
```

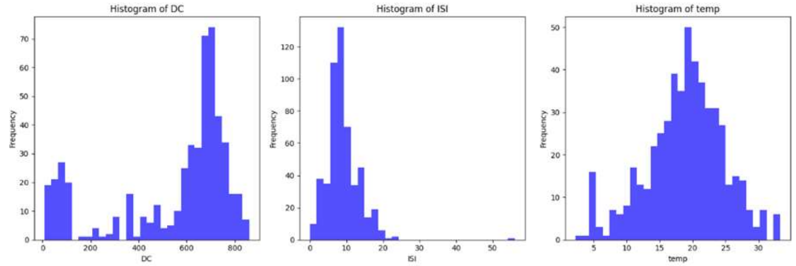
```
Checking for duplicated rows considering all columns:
     X  Y  month  day  FFMC    DMC     DC   ISI  temp  RH  wind  rain  \
52   4  3      1    6  92.1  111.2  654.1   9.6  20.4  42   4.9   0.0
53   4  3      1    6  92.1  111.2  654.1   9.6  20.4  42   4.9   0.0
99   3  4      1    3  91.4  142.4  601.4  10.6  19.8  39   5.4   0.0
100  3  4      1    3  91.4  142.4  601.4  10.6  19.8  39   5.4   0.0
214  4  4      7    2  91.7   35.8   80.8   7.8  17.0  27   4.9   0.0
215  4  4      7    2  91.7   35.8   80.8   7.8  17.0  27   4.9   0.0
302  3  6      6    0  91.1   94.1  232.1   7.1  19.2  38   4.5   0.0
303  3  6      6    0  91.1   94.1  232.1   7.1  19.2  38   4.5   0.0

         area
52   0.000000
53   0.000000
99   0.000000
100  0.000000
214  3.389799
215  3.389799
302  0.000000
303  0.000000
duplicate rows removed

------------------------

X        0
...
wind     0
rain     0
area     0
dtype: int64
```

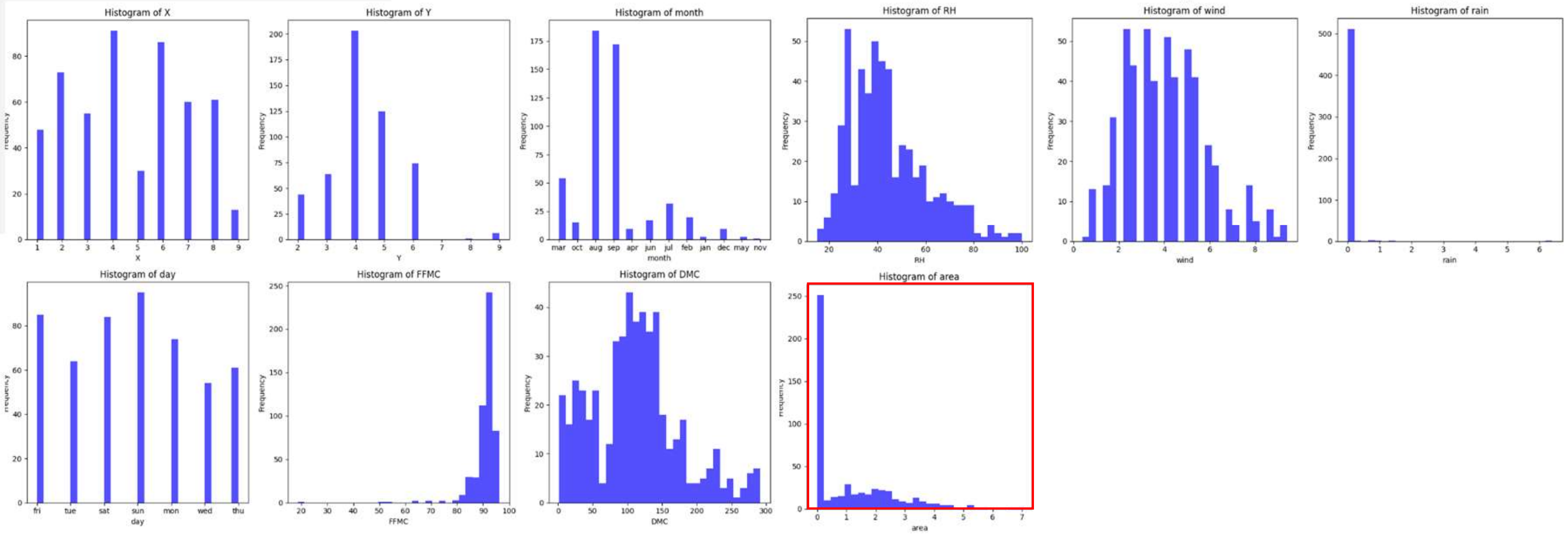
- Check if there are any missing values
- Check if there are any duplicated values
- No missing values
- Duplicated values are removed

Data Analysis

```python
import matplotlib.pyplot as plt

num_features = len(header)
fig, axes = plt.subplots(nrows=(num_features + 2) // 3, ncols=3, figsize=
(15, 5 * ((num_features + 2) // 3)))
axes = axes.flatten()   # Convert the axes in the form of a two-dimensional
array to a one-dimensional array.
                        # This makes it convenient to use a for loop.

for i, col in enumerate(df.columns):
    axes[i].hist(df[col], bins=30, color='blue', alpha=0.7)
    axes[i].set_title(f'Histogram of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

for i in range(len(df.columns), len(axes)):
    fig.delaxes(axes[i])

# Adjusting the subplot layout
plt.tight_layout()
plt.show()
```

- Histogram representation using pyplot library

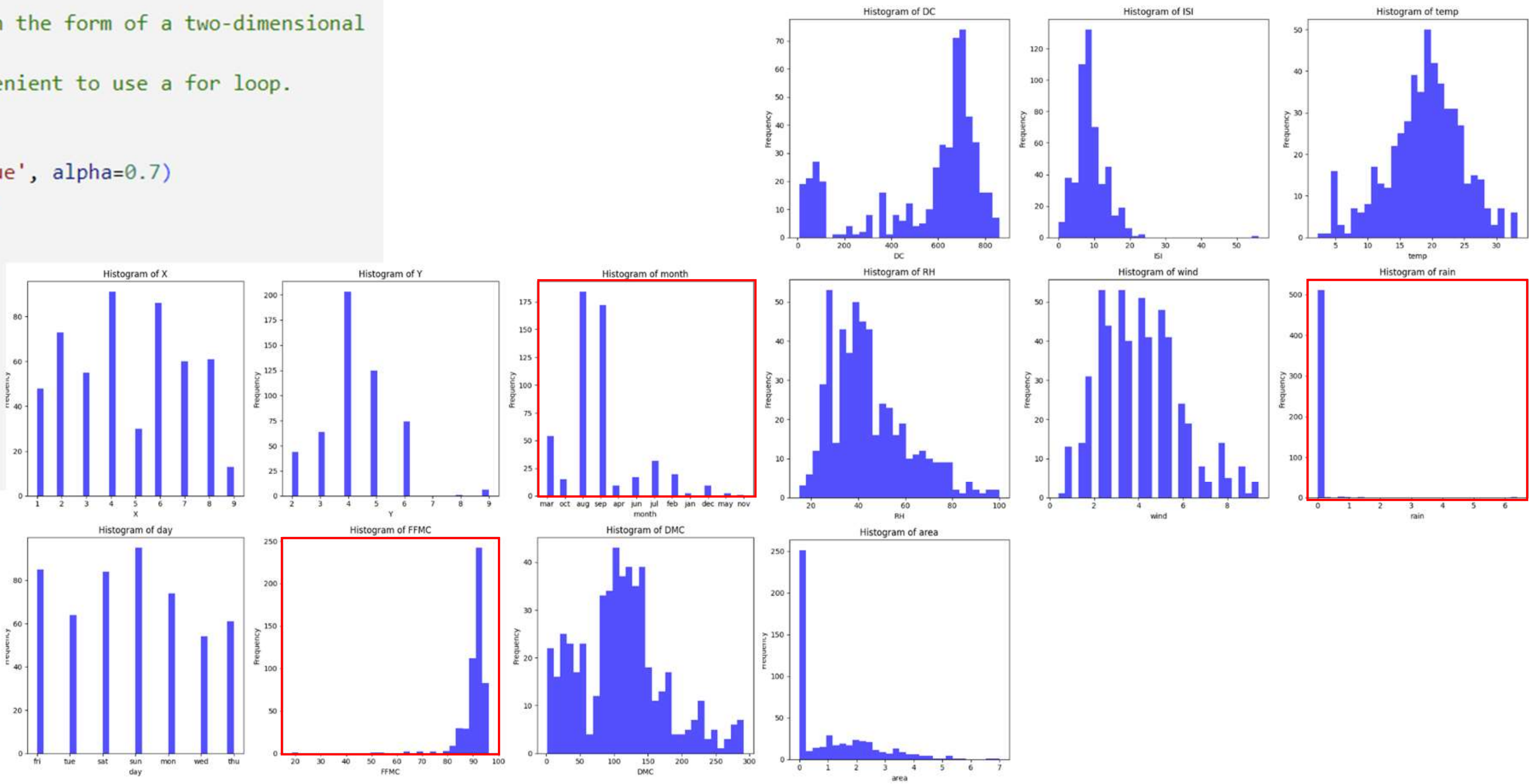




Data Analysis

```python
import matplotlib.pyplot as plt

num_features = len(header)
fig, axes = plt.subplots(nrows=(num_features + 2) // 3, ncols=3, figsize=
(15, 5 * ((num_features + 2) // 3)))
axes = axes.flatten()   # Convert the axes in the form of a two-dimensional
array to a one-dimensional array.
                        # This makes it convenient to use a for loop.

for i, col in enumerate(df.columns):
    axes[i].hist(df[col], bins=30, color='blue', alpha=0.7)
    axes[i].set_title(f'Histogram of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')

for i in range(len(df.columns), len(axes)):
    fig.delaxes(axes[i])

# Adjusting the subplot layout
plt.tight_layout()
plt.show()
```

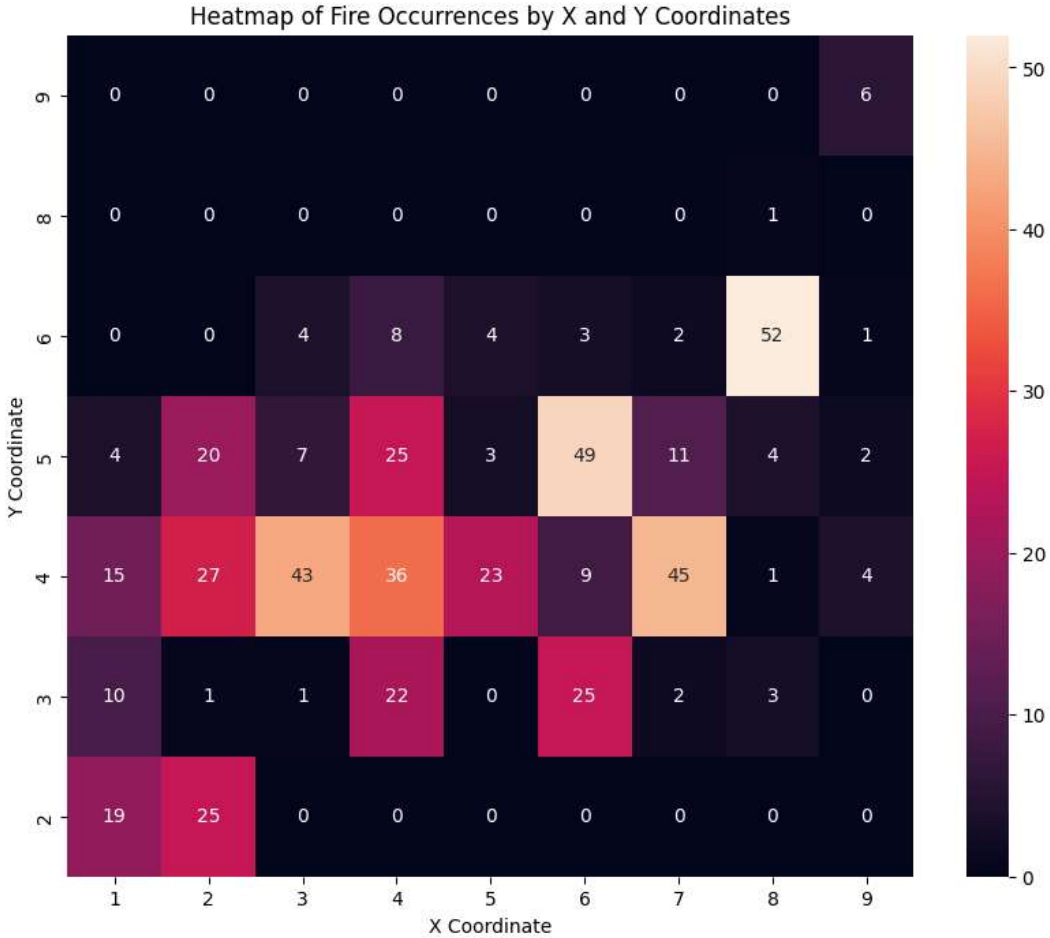
- Histogram representation using pyplot library



Data Analysis

```
import seaborn as sns

# Create a heatmap using X, Y coordinates
heatmap_data = df.pivot_table(index='Y', columns='X', values='area',
aggfunc='count', fill_value=0)
# The pivot_table function reorganizes a data frame to create a new table.

plt.figure(figsize=(10, 8))
sns.heatmap(heatmap_data, annot=True, cbar=True) # Create a heatmap using the
pivot table we created earlier.
plt.gca().invert_yaxis()
plt.title('Heatmap of Fire Occurrences by X and Y Coordinates')
plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.show()
```

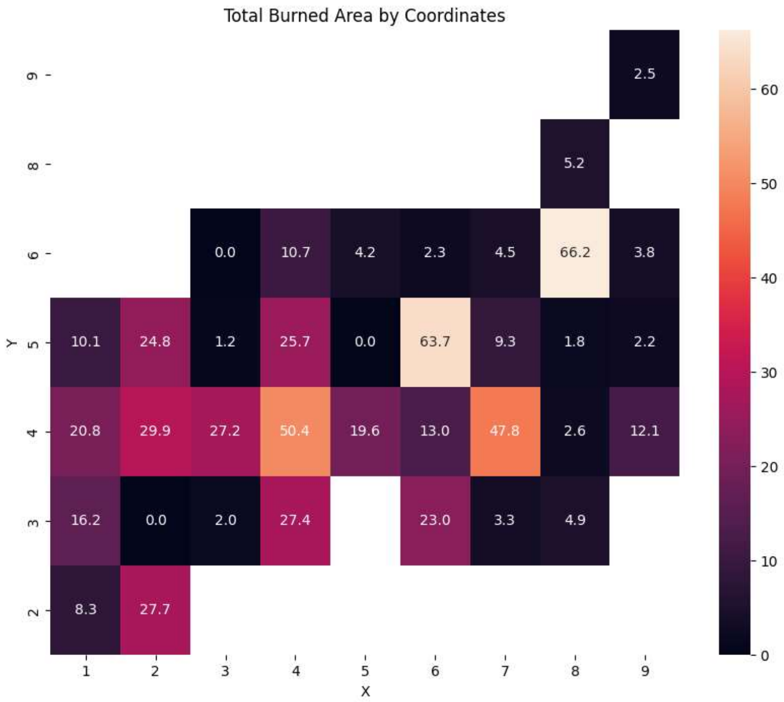
- Find relationship between coordinate and fire occurrences: location specificity
- The most frequent fires occur in (8,6), (6,5), (7,4), and (3,4) (n>40).
- Problem: most of value of 'area' are very low
- There might not be a relationship between location and 'area'

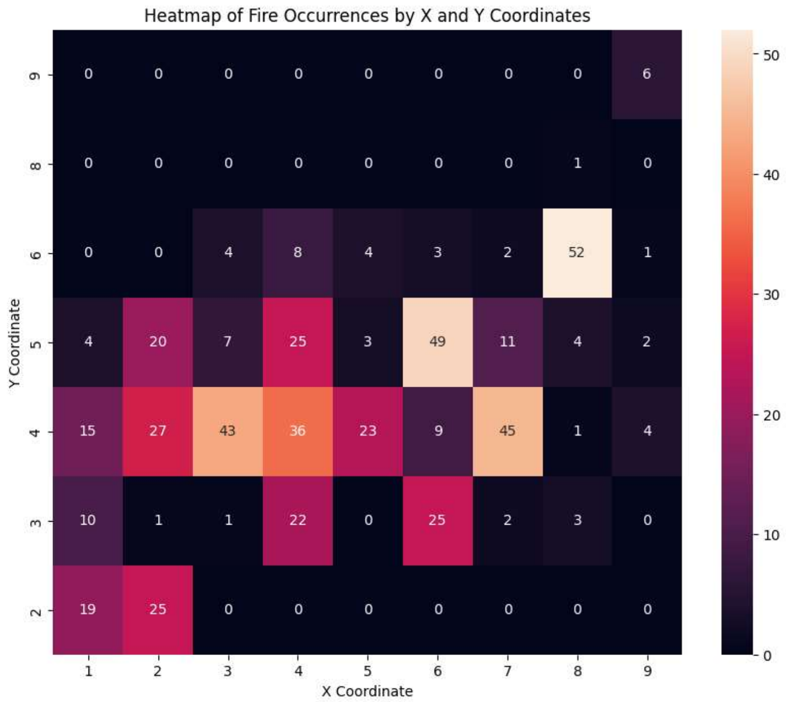


Data Analysis

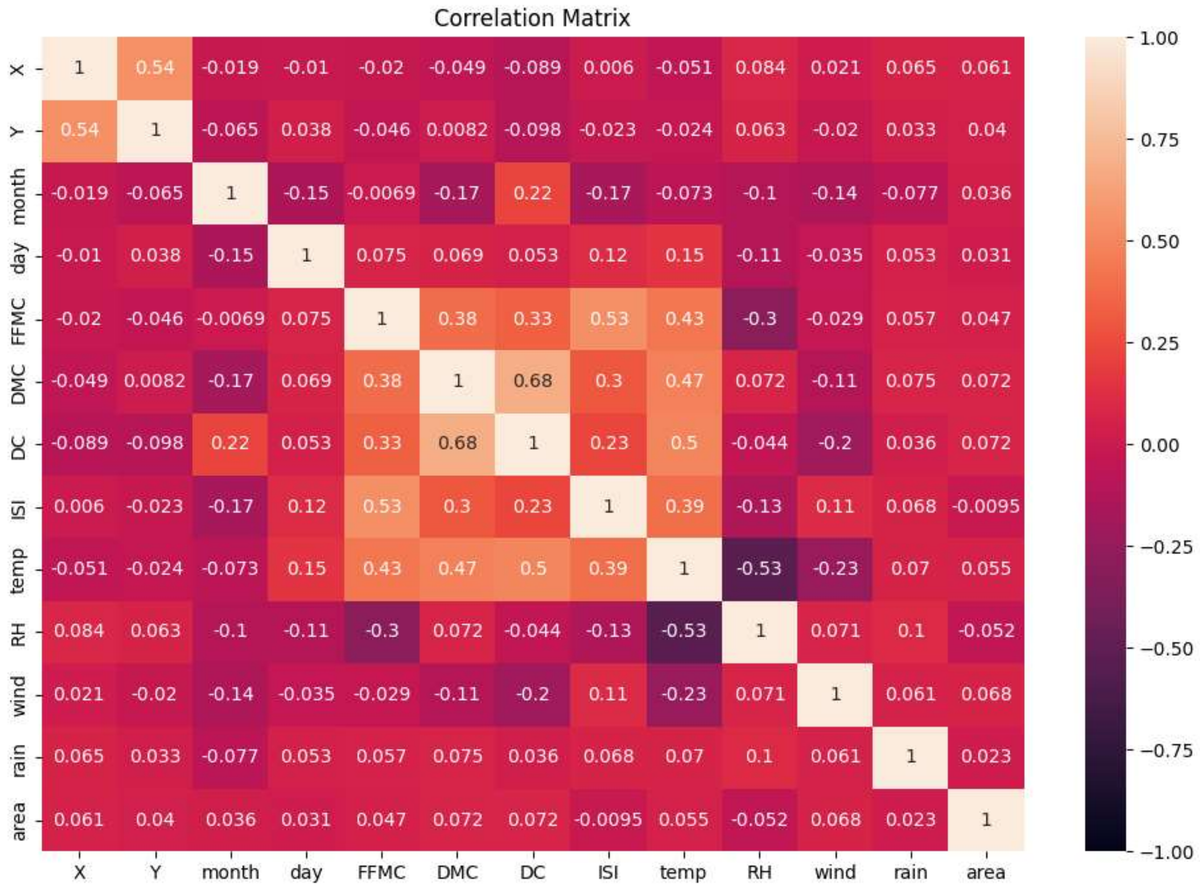
```python
area_sum_by_coordinates = df.groupby(['X', 'Y'])['area'].sum().reset_index()

pivot_table = area_sum_by_coordinates.pivot(index='Y', columns='X', values='area')

plt.figure(figsize=(10, 8))
sns.heatmap(pivot_table, annot=True, fmt=".1f")
plt.gca().invert_yaxis()
plt.title('Total Burned Area by Coordinates')
plt.show()
```

- There seems to be a correlation between occurrence and sum of ‘area’, but low





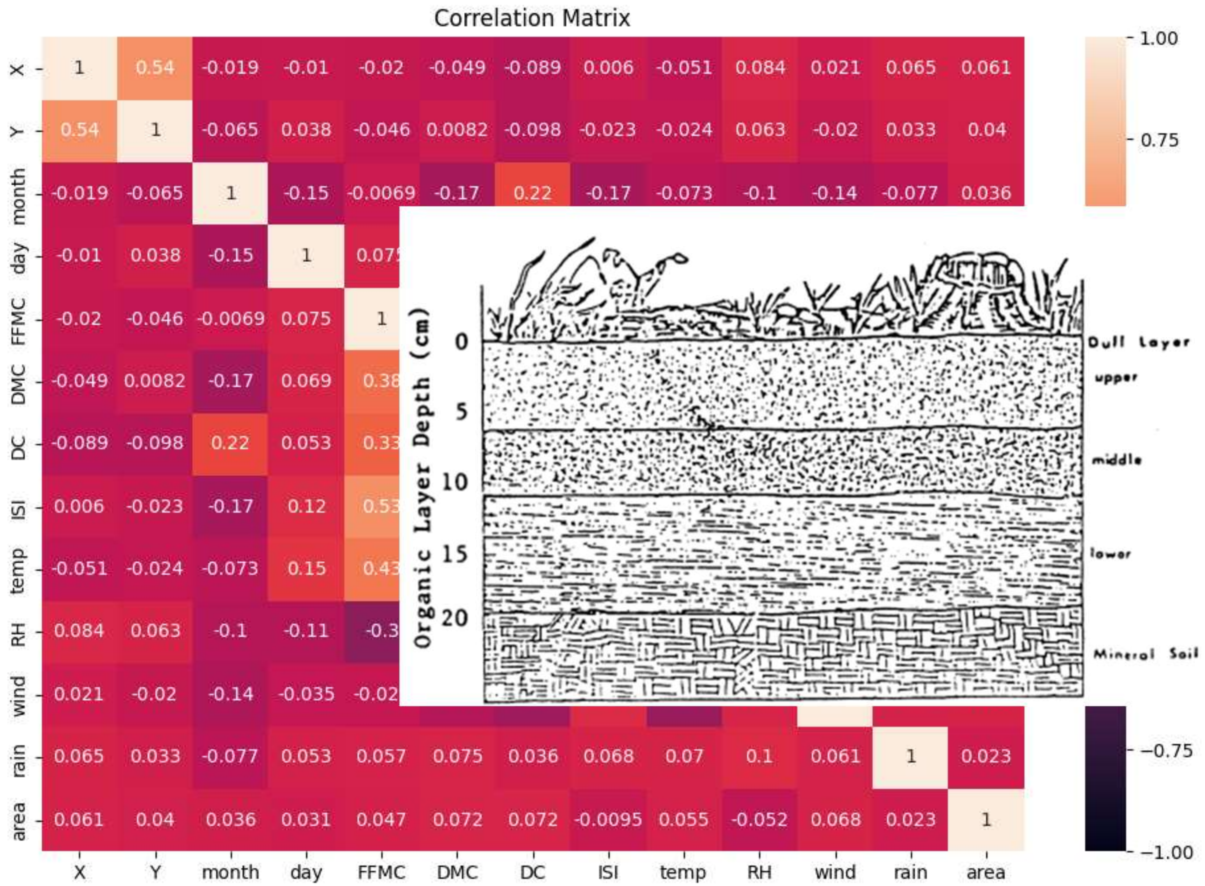
Data Analysis




```python
# Calculate correlation matrix
corr_matrix = num_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.show()
```

- DMC and DC show the highest correlations
- These two show a relatively low correlation with FFMC
- Temperature has a strong correlation with these three features, and RH

Data Analysis




```
# Calculate correlation matrix
corr_matrix = num_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, vmin=-1, vmax=1)
```

Weight	Fuel Moisture Code
5 t/ha	FFMC
50 t/ha	DMC
440 t/ha	DC

the highest

elatively low
MC

strong correlation with
, and RH

Model Selection

Random Forest

- Linear Regression, Decision Tree, SVM, etc.. Many options
- Our data have only 13 features -> no reason to use SVM
- Our data set is small: only ca. 500 size
- Decision Tree has a higher risk of overfitting than Random Forest
- Stojanova et al. (2012) compared and evaluated models such as KNN, DT, LR and SVM for prediction of forest fires
- RF showed the highest performance

Random Forest

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor

X = num_df.drop('area', axis=1)
y = num_df['area']

# split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Random Forest
RFmodel = RandomForestRegressor(random_state=42)
RFmodel.fit(X_train, y_train)

# prediction
RF_pred_test = RFmodel.predict(X_test)
RF_pred_train = RFmodel.predict(X_train)

# evaluation
RF_mse = mean_squared_error(y_test, RF_pred_test)
RF_mse_train = mean_squared_error(y_train, RF_pred_train)

results = {
    "Metric": ["Mean Squared Error"],
    "Train": [RF_mse_train],
    "Test": [RF_mse]
}

results_df = pd.DataFrame(results)
print(results_df)
```

✓ 0.1s

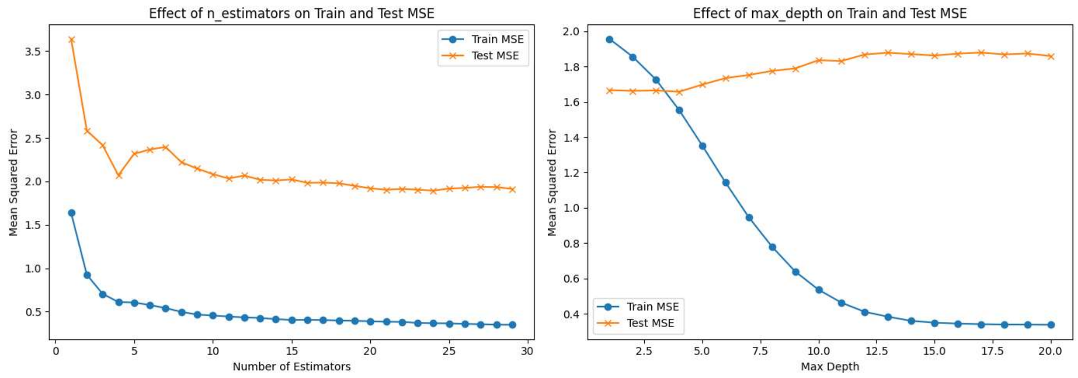
```
               Metric     Train      Test
0  Mean Squared Error  0.337283  1.869937
```

- A large difference in MSE between the training set and the test set
- It might be caused by overfitting!

Parameter modification

```python
# Evaluate model performance by changing n_estimators. Use n_estimators values
between 1 and 30.
n_estimators_range = range(1, 30)
train_mse_values = []
test_mse_values = []

for n_estimators in n_estimators_range:
    model = RandomForestRegressor(n_estimators=n_estimators, random_state=42)
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, train_pred)
    test_mse = mean_squared_error(y_test, test_pred)

    train_mse_values.append(train_mse)
    test_mse_values.append(test_mse)


# Evaluate model performance according to changes in max_depth. Use max_depth
values between 1 and 20.
max_depth_range = range(1, 21)
train_accuracies_depth = []
test_accuracies_depth = []

for max_depth in max_depth_range:
    model = RandomForestRegressor(max_depth=max_depth, random_state=42)
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, train_pred)
    test_mse = mean_squared_error(y_test, test_pred)

    train_accuracies_depth.append(train_mse)
    test_accuracies_depth.append(test_mse)
```

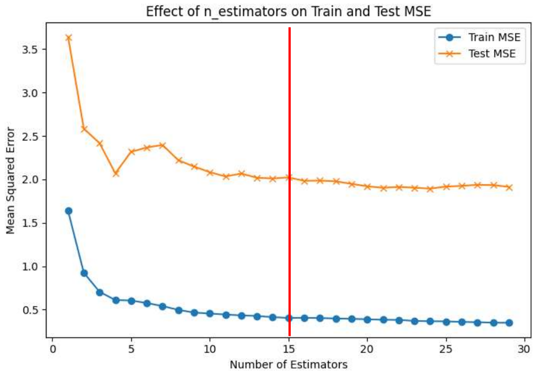
```python
# Visualization
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# First graph: MSE as a function of n_estimators
axes[0].plot(n_estimators_range, train_mse_values, label='Train MSE', marker='o')
axes[0].plot(n_estimators_range, test_mse_values, label='Test MSE', marker='x')
axes[0].set_xlabel('Number of Estimators')
axes[0].set_ylabel('Mean Squared Error')
axes[0].set_title('Effect of n_estimators on Train and Test MSE')
axes[0].legend()

# Second graph: MSE according to max_depth
axes[1].plot(max_depth_range, train_accuracies_depth, label='Train MSE',
marker='o')
axes[1].plot(max_depth_range, test_accuracies_depth, label='Test MSE',
marker='x')
axes[1].set_xlabel('Max Depth')
axes[1].set_ylabel('Mean Squared Error')
axes[1].set_title('Effect of max_depth on Train and Test MSE')
axes[1].legend()

plt.tight_layout()
plt.show()
```

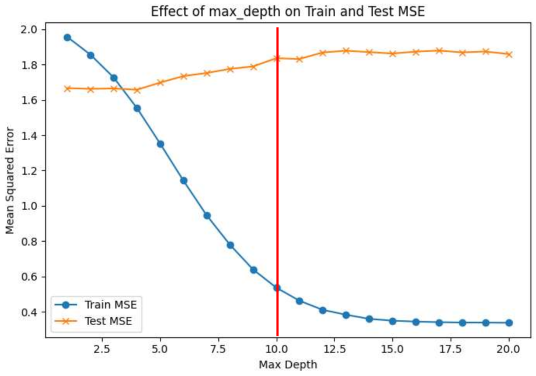


Parameter modification

```python
# Evaluate model performance by changing n_estimators. Use n_estimators values
between 1 and 30.
n_estimators_range = range(1, 30)
train_mse_values = []
test_mse_values = []

for n_estimators in n_estimators_range:
    model = RandomForestRegressor(n_estimators=n_estimators, random_state=42)
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, train_pred)
    test_mse = mean_squared_error(y_test, test_pred)

    train_mse_values.append(train_mse)
    test_mse_values.append(test_mse)


# Evaluate model performance according to changes in max_depth. Use max_depth
values between 1 and 20.
max_depth_range = range(1, 21)
train_accuracies_depth = []
test_accuracies_depth = []

for max_depth in max_depth_range:
    model = RandomForestRegressor(max_depth=max_depth, random_state=42)
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, train_pred)
    test_mse = mean_squared_error(y_test, test_pred)

    train_accuracies_depth.append(train_mse)
    test_accuracies_depth.append(test_mse)
```

```python
# Visualization
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))

# First graph: MSE as a function of n_estimators
axes[0].plot(n_estimators_range, train_mse_values, label='Train MSE', marker='o')
axes[0].plot(n_estimators_range, test_mse_values, label='Test MSE', marker='x')
axes[0].set_xlabel('Number of Estimators')
axes[0].set_ylabel('Mean Squared Error')
axes[0].set_title('Effect of n_estimators on Train and Test MSE')
axes[0].legend()

# Second graph: MSE according to max_depth
axes[1].plot(max_depth_range, train_accuracies_depth, label='Train MSE',
marker='o')
axes[1].plot(max_depth_range, test_accuracies_depth, label='Test MSE',
marker='x')
axes[1].set_xlabel('Max Depth')
axes[1].set_ylabel('Mean Squared Error')
axes[1].set_title('Effect of max_depth on Train and Test MSE')
axes[1].legend()

plt.tight_layout()
plt.show()
```





Evaluation

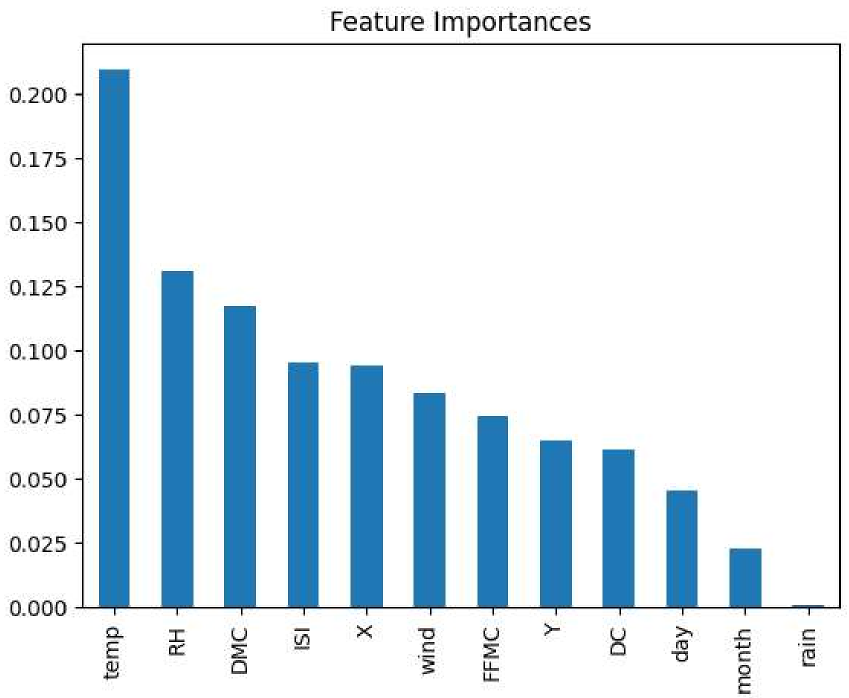
```
from sklearn.model_selection import  cross_val_score, KFold
from sklearn.metrics import make_scorer, r2_score
from sklearn.dummy import DummyRegressor

# K-fold settings
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Model setup
RFmodel = RandomForestRegressor(n_estimators= 15, max_depth=10, random_state=42)
baseline_mean = DummyRegressor(strategy='mean')
baseline_median = DummyRegressor(strategy='median')

# Setting up MSE scorer for performance evaluation
mse_scorer = make_scorer(mean_squared_error)
r2_scorer = make_scorer(r2_score)

# Perform Random Forest cross validation
mse_scores_rf = cross_val_score(RFmodel, X, y, cv=kf, scoring=mse_scorer)
r2_scores_rf = cross_val_score(RFmodel, X, y, cv=kf, scoring=r2_scorer)

# Perform Dummy Regressor cross validation
mse_scores_dummy_mean = cross_val_score(baseline_mean, X, y, cv=kf,
scoring=mse_scorer)
r2_scores_dummy_mean = cross_val_score(baseline_mean, X, y, cv=kf,
scoring=r2_scorer)

mse_scores_dummy_median = cross_val_score(baseline_median, X, y, cv=kf,
scoring=mse_scorer)
r2_scores_dummy_median = cross_val_score(baseline_median, X, y, cv=kf,
scoring=r2_scorer)

# Output
comparision = {
    "" : ["MSE", "r2_score"],
    "Random_Forest": [np.mean(mse_scores_rf), np.mean(r2_scores_rf)],
    "Baseline_Mean": [np.mean(mse_scores_dummy_mean), np.mean
    (r2_scores_dummy_mean)],
    "Baseline_Median": [np.mean(mse_scores_dummy_median), np.mean
    (r2_scores_dummy_median)]
}

print(round(pd.DataFrame(comparision),2))
```

		Random_Forest	Baseline_Mean	Baseline_Median
0	MSE	2.29	1.96	2.45
1	r2_score	-0.19	-0.01	-0.25

- Baseline Model: Dummy Regressor for prediction of mean and median
- Evaluation using MSE and r2 score
- MSE(RF) < MSE(dummy_mean) & negative r2 score
- Our model doesn’t showing good performance

Feature Importances

```
RFmodel.fit(X_train, y_train)

feature_importances = RFmodel.feature_importances_
RF_importances = pd.Series(feature_importances, index=X.columns)
RF_importances = RF_importances.sort_values(ascending=False)
RF_importances.plot(kind='bar')
plt.title('Feature Importances')
plt.show()
```



- The 3 most important role in prediction were \`temp\`, \`RH\`, and \`DMC\`
- Remarkable contribution of \`temp\`
- It is assumed to be due to high correlation with other important features.

- Very low contribution of month and rain
- Most of the data is distributed in a narrow range
- Does not play a significant role in prediction

Additional Analysis

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# normalization for linear regression

scaler = StandardScaler()
numeric_features = ['X', 'Y', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind',
'rain', 'area']
num_df[numeric_features] = scaler.fit_transform(num_df[numeric_features])

LRmodel = LinearRegression()
mse_scores_lr = cross_val_score(RFmodel, X, y, cv=kf, scoring=mse_scorer)
r2_scores_lr = cross_val_score(RFmodel, X, y, cv=kf, scoring=r2_scorer)

additional_comparision = {
    "" : ["MSE", "r2_score"],
    "Random_Forest": [np.mean(mse_scores_rf), np.mean(r2_scores_rf)],
    "Baseline_Mean": [np.mean(mse_scores_dummy_mean), np.mean
    (r2_scores_dummy_mean)],
    "Baseline_Median": [np.mean(mse_scores_dummy_median), np.mean
    (r2_scores_dummy_median)],
    "Linear_Regression": [np.mean(mse_scores_lr), np.mean(r2_scores_lr)]
}

print(round(pd.DataFrame(additional_comparision),2))
```

✓ 0.2s Py

```
                Random_Forest  Baseline_Mean  Baseline_Median  Linear_Regression
0       MSE              2.29           1.96             2.45               2.29
1  r2_score             -0.19          -0.01            -0.25              -0.19
```

- In case our RF model does not fit the data well, Linear regression is also tested
- For Linear Regression, normalization is needed
- Here, z-score normalization is used
- It shows exactly same bad performance


THANK YOU!