

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import sys
from sklearn.preprocessing import LabelEncoder
from sklearn import preprocessing
from scipy.stats import chi2_contingency
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Dataset 1

```
df = pd.read_csv("df_1.csv")
```

```
df.head()
```

	Unnamed: 0	Entity	Year	Records	Organization type	Method	Sources
0	0	21st Century Oncology	2016	2200000	healthcare	hacked	[5][6]
1	1	500px	2020	14870304	social networking	hacked	[7]
2	2	Accendo Insurance Co.	2020	175350	healthcare	poor security	[8][9]
3	3	Adobe Systems	2016	15000000	tech	hacked	[4][3]

Next steps:

Generate code with df

☒ View recommended plots

```
df.shape
```

(352, 7)

```
df.dtypes
```

```
Unnamed: 0      int64
Entity          object
Year            object
Records         object
Organization type  object
Method          object
Sources         object
dtype: object
```



```
def text_preprocessing(df):
    df = df[df['Year'] != '2014 and 2015']
    df = df[df['Year'] != '2019-2020']
    df = df[df['Year'] != '2018-2019']
    df['Records'] = [str(x).lower() for x in df['Records']]
    df = df[df['Records'] != 'unknown']
    df = df[df['Records'] != 'g20 world leaders']
    df = df[df['Records'] != '19 years of data']
    df = df[df['Records'] != '63 stores']
    df = df[df['Records'] != 'tens of thousands']
    df = df[df['Records'] != 'over 5,000,000']
    df = df[df['Records'] != 'unknown (client list)']
    df = df[df['Records'] != 'millions']
    df = df[df['Records'] != '235 gb']
    df = df[df['Records'] != '350 clients emails']
    df = df[df['Records'] != 'nan']
    df = df[df['Records'] != '2.5gb']
    df = df[df['Records'] != '250 locations']
    df = df[df['Records'] != '500 locations']
    df = df[df['Records'] != '10 locations']
    df = df[df['Records'] != '93 stores']
    df = df[df['Records'] != 'undisclosed']
    df = df[df['Records'] != 'source code compromised']
    df = df[df['Records'] != '100 terabytes']
    df = df[df['Records'] != '54 locations']
    df = df[df['Records'] != '200 stores']
    df = df[df['Records'] != '8 locations']
    df = df[df['Records'] != '51 locations']
    df = df[df['Records'] != 'tbc']
```

```
list_of_records = []
```

```
for i, x in enumerate(df['Records']):
```

```
    list_of_records.append(int(x))
```

```
df['Records'] = list_of_records
```

```
df.rename(columns={'Unnamed: 0': 'Id'}, inplace=True)
```

```
list_of_years = []
```

```
for i, x in enumerate(df['Year']):
```

```
    list_of_years.append(int(x))
```

```
df['Year'] = list_of_years
```

```
list_of_years = []
```

```
for i, x in enumerate(df['Sources']):
```

```
    list_of_years.append(str(x))
```

```
df['Sources'] = list_of_years
```

```
df.dropna(subset=['Records'], inplace=True)
```

```
df.reset_index(drop=True, inplace=True)
```

```
return df
```

```
df = text_preprocessing(df)
```

```
df.drop(['Sources'], axis=1, inplace=True)
```

```
df.isnull().any()
```

```
Id                False
Entity            False
Year              False
Records           False
Organization type False
Method            True
dtype: bool
```

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

```
Id                0
Entity            0
Year              0
Records           0
Organization type 0
Method            1
dtype: int64
```



```
df.columns = ['id', 'Entity', 'Year', 'Records', 'Organization type', 'Method']
```

```
df.head(10)
```

	id	Entity	Year	Records	Organization type	Method
0	0	21st Century Oncology	2016	2200000	healthcare	hacked
1	1	500px	2020	14870304	social networking	hacked
2	2	Accendo Insurance Co.	2020	175350	healthcare	poor security
3	3	Adobe Systems Incorporated	2013	152000000	tech	hacked
4	4	Adobe Inc.	2019	7500000	tech	poor security
5	5	Advocate Medical Group	2017	4000000	healthcare	lost / stolen media
6	6	AerServ (subsidiary of InMobi)	2018	75000	advertising	hacked
7	7	Affinity Health Plan, Inc.	2013	344579	healthcare	lost / stolen media

Next steps:

Generate code with df

 View recommended plots

```
df['Year'] = df['Year'].astype(str)
df['Year'] = df['Year'].str[:4]
df['Year'] = df['Year'].astype(int)
```

```
df.dtypes
```

```
id          int64
Entity      object
Year        int64
Records     int64
Organization type  object
Method      object
dtype: object
```

```
df_heatmap = df.copy(deep=True)
```

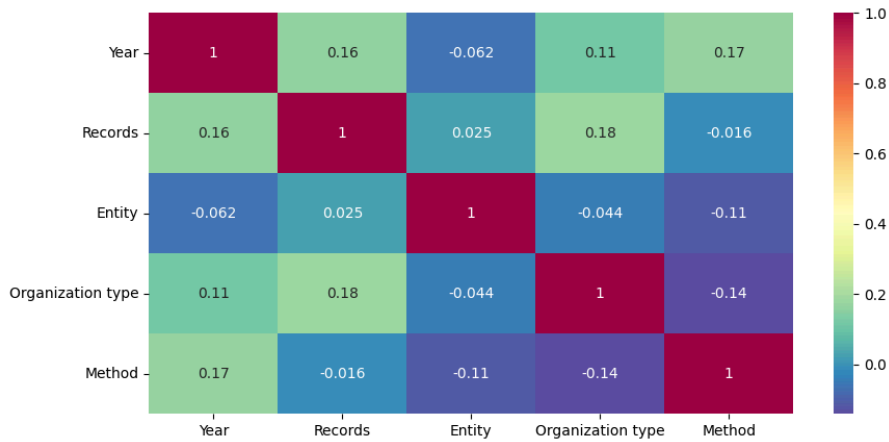
Feature Transformation

```
le = LabelEncoder()
df_heatmap['Records'] = le.fit_transform(df_heatmap['Records'])
df_heatmap['Entity'] = le.fit_transform(df_heatmap['Entity'])
df_heatmap['Organization type'] = le.fit_transform(df_heatmap['Organization type'])
df_heatmap['Method'] = le.fit_transform(df_heatmap['Method'])
```

```
df_heatmap.corr()
```

	id	Entity	Year	Records	Organization type	Method
id	1.000000	0.903278	-0.017146	0.001198	-0.001792	-0.097923
Entity	0.903278	1.000000	-0.062113	0.024731	-0.043852	-0.112576
Year	-0.017146	-0.062113	1.000000	0.162146	0.113221	0.166885
Records	0.001198	0.024731	0.162146	1.000000	0.181693	-0.015932
Organization type	-0.001792	-0.043852	0.113221	0.181693	1.000000	-0.141679

```
plt.figure(figsize=(10,5))
sns.heatmap(df_heatmap[['Year', 'Records','Entity', 'Organization type', 'Method']].corr(), cmap='Spectral_r', annot=True);
```



```
# Perform chi-square test for categorical feature "Organization type" and target variable "Records"
chi2_statistic, chi2_p_value, _, _ = chi2_contingency(pd.crosstab(df['Organization type'], df['Records']))
print("Chi-Square Statistic:", chi2_statistic)
print("Chi-Square p-value:", chi2_p_value)
```

```
Chi-Square Statistic: 14831.661706407895
Chi-Square p-value: 1.914409333443912e-16
```

```
# Perform chi-square test for categorical feature "Organization type" and target variable "Records"
chi2_statistic, chi2_p_value, _, _ = chi2_contingency(pd.crosstab(df['Method'], df['Records']))
print("Chi-Square Statistic:", chi2_statistic)
print("Chi-Square p-value:", chi2_p_value)
```

```
Chi-Square Statistic: 5437.109807616157
Chi-Square p-value: 7.956940367902997e-06
```

```
# Perform chi-square test for categorical feature "Organization type" and target variable "Records"
chi2_statistic, chi2_p_value, _, _ = chi2_contingency(pd.crosstab(df['Entity'], df['Records']))
print("Chi-Square Statistic:", chi2_statistic)
print("Chi-Square p-value:", chi2_p_value)
```

```
Chi-Square Statistic: 65330.888095238115
Chi-Square p-value: 0.05416984766035374
```

```
# Perform chi-square test for categorical feature "Organization type" and target variable "Records"
chi2_statistic, chi2_p_value, _, _ = chi2_contingency(pd.crosstab(df['Year'], df['Records']))
print("Chi-Square Statistic:", chi2_statistic)
print("Chi-Square p-value:", chi2_p_value)
```

```
Chi-Square Statistic: 4145.134008754404
Chi-Square p-value: 0.32280401338878495
```

With a p-value > 0.05, we fail to reject the null hypothesis and conclude that there is no significant association between the Year and the target variable Records.

From the above statistical test and correlation matrix we can conclude that Method, Organization type are most crucial features for predicting Records



```
df_heatmap.dropna(inplace=True)
X = df_heatmap.drop('Records', axis=1)
y = df_heatmap['Records']

print(X.shape), print(y.shape),
print()

(303, 5)
(303,)
```

```

from sklearn.decomposition import PCA

pca = PCA(n_components=2) # Specify the number of components to keep
principal_components = pca.fit_transform(X)

# Creating a DataFrame for the principal components
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

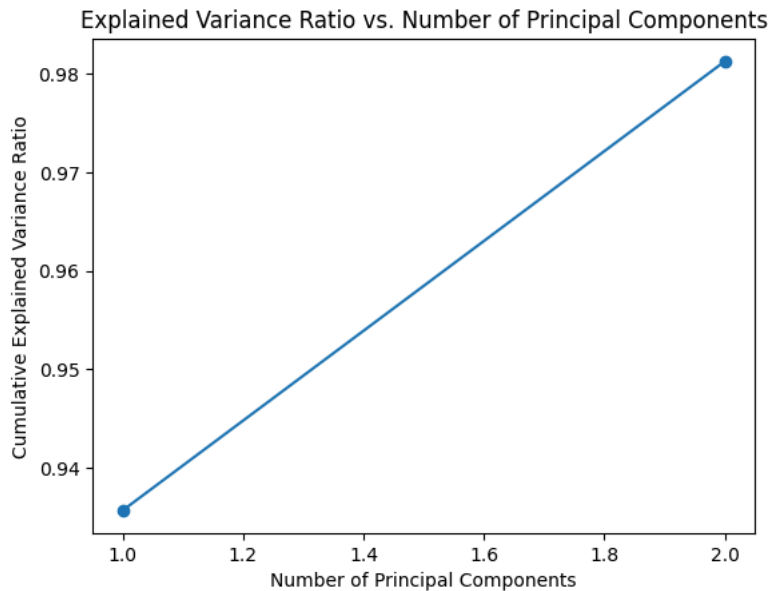
# Explained variance ratio
print ( "Components = ", pca.n_components_, "; \nTotal explained variance = ",
        round(pca.explained_variance_ratio_.sum(),5) )
explained_var_ratio = pca.explained_variance_ratio_
cumulative_var_ratio = np.cumsum(explained_var_ratio)
plt.plot(range(1, len(cumulative_var_ratio) + 1), cumulative_var_ratio, marker='o')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance Ratio')
plt.title('Explained Variance Ratio vs. Number of Principal Components')
plt.show()

```

```

Components = 2 ;
Total explained variance = 0.98129

```



```

df_encoded = pd.get_dummies(df)

X = df_encoded.drop(['Records'], axis=1)
y = np.array(df_encoded['Records'])

train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.2, random_state = 42)

print('Training Features Shape:', train_X.shape)
print('Training Labels Shape:', train_y.shape)
print('Testing Features Shape:', test_X.shape)
print('Testing Labels Shape:', test_y.shape)

Training Features Shape: (242, 370)
Training Labels Shape: (242,)
Testing Features Shape: (61, 370)
Testing Labels Shape: (61,)

mean = df_encoded['Records'].mean()
baseline_preds = [mean for i in range(len(test_y))]
baseline_errors = abs(baseline_preds - test_y)
print('Average baseline error: ', round(np.mean(baseline_errors), 2))

Average baseline error: 72172307.6

```



```

from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
import xgboost as xgb
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

class DTRegression:
    def __init__(self):
        pass

    def fit(self, train_x, train_y):
        self.model = DecisionTreeRegressor()
        self.model.fit(train_x, train_y)
        return self.model

    def predict(self, test_X):
        pred = self.model.predict(test_X)
        return pred

    def evaluate(self, test_X, test_y):
        pred = self.model.predict(test_X)
        self.errors = abs(pred - test_y)
        self.average_error = round(np.mean(self.errors), 2)
        return self.average_error

class RandomForest:
    def __init__(self, n_estimators=1000):
        self.n_estimators = n_estimators

    def fit(self, train_x, train_y):
        self.model = RandomForestRegressor(n_estimators = self.n_estimators, random_state = 42)
        self.model.fit(train_x, train_y)
        return self.model

    def predict(self, test_X):
        pred = self.model.predict(test_X)
        return pred

    def evaluate(self, test_X, test_y):
        pred = self.model.predict(test_X)
        self.errors = abs(pred - test_y)
        self.average_error = round(np.mean(self.errors), 2)
        return self.average_error

class XGBoosting:
    def __init__(self, booster='gbtree', objective = 'reg:linear', colsample_bytree = 0.3, learning_rate = 0.1, max_depth = 5, n_estimators =
        self.booster = booster
        self.objective= objective
        self.colsample_bytree = colsample_bytree
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.n_estimators = n_estimators

    def fit(self, train_X, train_y):
        self.model = xgb.XGBRegressor(booster=self.booster, objective =self.objective,
                                     colsample_bytree = self.colsample_bytree, learning_rate = self.learning_rate,
                                     max_depth = self.max_depth, n_estimators = self.n_estimators)
        self.model.fit(train_X, train_y)
        return self.model

    def predict(self, test_X):
        pred = self.model.predict(test_X)
        return pred

    def evaluate(self, test_X, test_y):
        pred = self.model.predict(test_X)
        self.errors = abs(pred - test_y)
        self.average_error = round(np.mean(self.errors), 2)
        return self.average_error

import time
lr = DTRegression()
# Record start time
start_time = time.time()
lr.fit(train_X, train_y)
# Record end time

```



```

end_time = time.time()
# Calculate duration
training_duration = end_time - start_time
print("Decision Tree Regression Training Duration:", training_duration, "seconds")
print('Mean Absolute Error:', lr.evaluate(test_X, test_y), 'degrees.')

# Record start time
start_time_prediction = time.time()

# Generate predictions using your model
lr_predictions = lr.predict(test_X)

# Record end time
end_time_prediction = time.time()

# Calculate duration
prediction_duration = end_time_prediction - start_time_prediction
print("Decision Tree Regression Prediction Duration:", prediction_duration, "seconds")

dt_accuracy = accuracy_score(test_y, lr_predictions)
dt_precision = precision_score(test_y, lr_predictions, average='weighted')
dt_recall = recall_score(test_y, lr_predictions, average='weighted')
dt_f1_score = f1_score(test_y, lr_predictions, average='weighted')

print("Decision Tree Regression Metrics:")
print("Accuracy:", dt_accuracy)
print("Precision:", dt_precision)
print("Recall:", dt_recall)
print("F1-score:", dt_f1_score)
print()

Decision Tree Regression Training Duration: 0.025714635848999023 seconds
Mean Absolute Error: 79416943.85 degrees.
Decision Tree Regression Prediction Duration: 0.002556324005126953 seconds
Decision Tree Regression Metrics:
Accuracy: 0.0
Precision: 0.0
Recall: 0.0
F1-score: 0.0

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall is ill-defined and being
_warn_prf(average, modifier, msg_start, len(result))

```

```

rf = RandomForest(n_estimators=1000)
# Record start time
start_time = time.time()
rf.fit(train_X, train_y)
# Record end time
end_time = time.time()
# Calculate duration
training_duration = end_time - start_time
print("Random Forest Training Duration:", training_duration, "seconds")
print('Mean Absolute Error:', rf.evaluate(test_X, test_y), 'degrees.')

# Record start time
start_time_prediction = time.time()

# Generate predictions using your model
rf_predictions = rf.predict(test_X)

# Record end time
end_time_prediction = time.time()

# Calculate duration
prediction_duration = end_time_prediction - start_time_prediction
print("Random Forest Prediction Duration:", prediction_duration, "seconds")

Random Forest Training Duration: 10.19991660118103 seconds
Mean Absolute Error: 62517969.45 degrees.
Random Forest Prediction Duration: 0.24471807479858398 seconds

```

```

xgboosting = XGBoosting()
# Record start time

```



```

start_time = time.time()
xgboosting.fit(train_X, train_y)
# Record end time
end_time = time.time()
# Calculate duration
training_duration = end_time - start_time
print("xgboosting Training Duration:", training_duration, "seconds")
print('Mean Absolute Error:', xgboosting.evaluate(test_X, test_y), 'degrees.')

```

```

# Record start time
start_time_prediction = time.time()

```

```

# Generate predictions using your model
xgb_predictions = xgboosting.predict(test_X)

```

```

# Record end time
end_time_prediction = time.time()

```

```

# Calculate duration
prediction_duration = end_time_prediction - start_time_prediction
print("xgboosting Prediction Duration:", prediction_duration, "seconds")

```

```

xgb_accuracy = accuracy_score(test_y, xgb_predictions)
xgb_precision = precision_score(test_y, xgb_predictions, average='weighted')
xgb_recall = recall_score(test_y, xgb_predictions, average='weighted')
xgb_f1_score = f1_score(test_y, xgb_predictions, average='weighted')

```

```

print("XGBoost Metrics:")
print("Accuracy:", xgb_accuracy)
print("Precision:", xgb_precision)
print("Recall:", xgb_recall)
print("F1-score:", xgb_f1_score)
print()

```

```

xgboosting Training Duration: 0.1085665225982666 seconds
Mean Absolute Error: 59419203.25 degrees.
xgboosting Prediction Duration: 0.037367820739746094 seconds
XGBoost Metrics:
Accuracy: 0.0
Precision: 0.0
Recall: 0.0
F1-score: 0.0

```

```

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [01:53:28] WARNING: /workspace/src/objective/regression_obj.cu
warnings.warn(msg, UserWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
_warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall is ill-defined and being
_warn_prf(average, modifier, msg_start, len(result))

```

```

def LabelEncoding(df):
    le = LabelEncoder()

    le.fit(df['Entity'])
    df['Entity'] = le.transform(df['Entity'])

    le.fit(df['Organization type'])
    df['Organization type'] = le.transform(df['Organization type'])

    le.fit(df['Method'])
    df['Method'] = le.transform(df['Method'])
    return df

```

```

import xgboost as xgb
def splitting(df):
    X = df.drop(['Records'], axis=1)
    y = np.array(df['Records'])

    X.drop(['id'], axis=1, inplace=True)
    train_X, test_X, train_y, test_y = train_test_split(X, y, test_size = 0.2, random_state = 42)
    data_dmatrix = xgb.DMatrix(data=X, label=y, enable_categorical=True)

    return train_X, test_X, train_y, test_y, data_dmatrix

```




```

df_new = LabelEncoding(df)

train_X, test_X, train_y, test_y, data_dmatrix = splitting(df_new)

params = {'booster':'gbtree', "objective":"reg:linear", 'colsample_bytree': 0.3, 'learning_rate': 0.1, 'max_depth': 5, 'alpha': 10}


xg_reg = xgb.train(params=params, dtrain=data_dmatrix, num_boost_round=10)

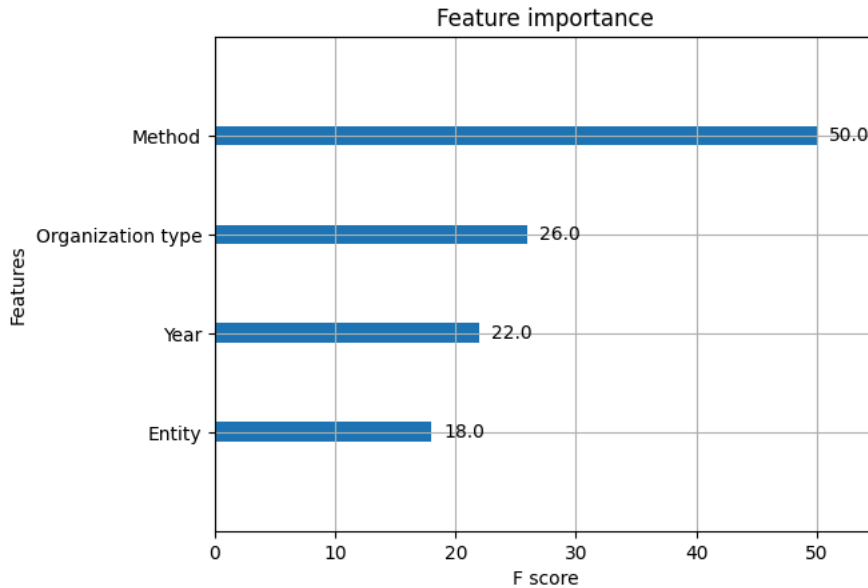
xgb.plot_importance(xg_reg)

plt.rcParams['figure.figsize'] = [5, 5]

plt.show()

```

 /usr/local/lib/python3.10/dist-packages/xgboost/core.py:160: UserWarning: [01:53:40] WARNING: /workspace/src/objective/regression_obj.cu
warnings.warn(msg, UserWarning)



```

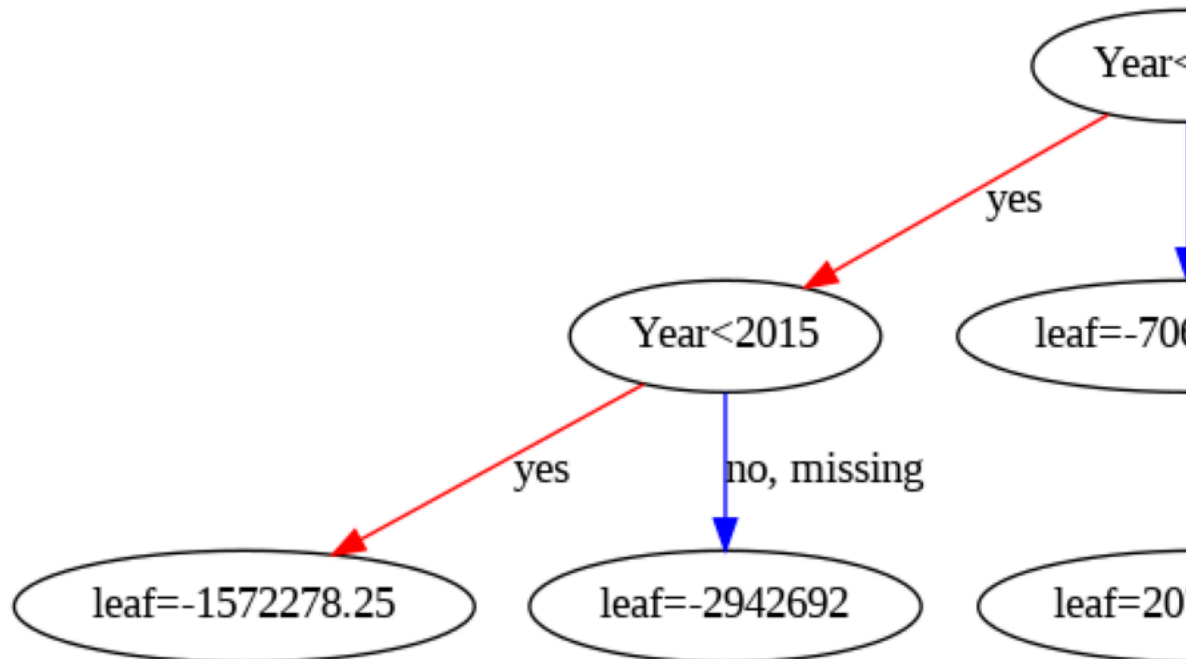
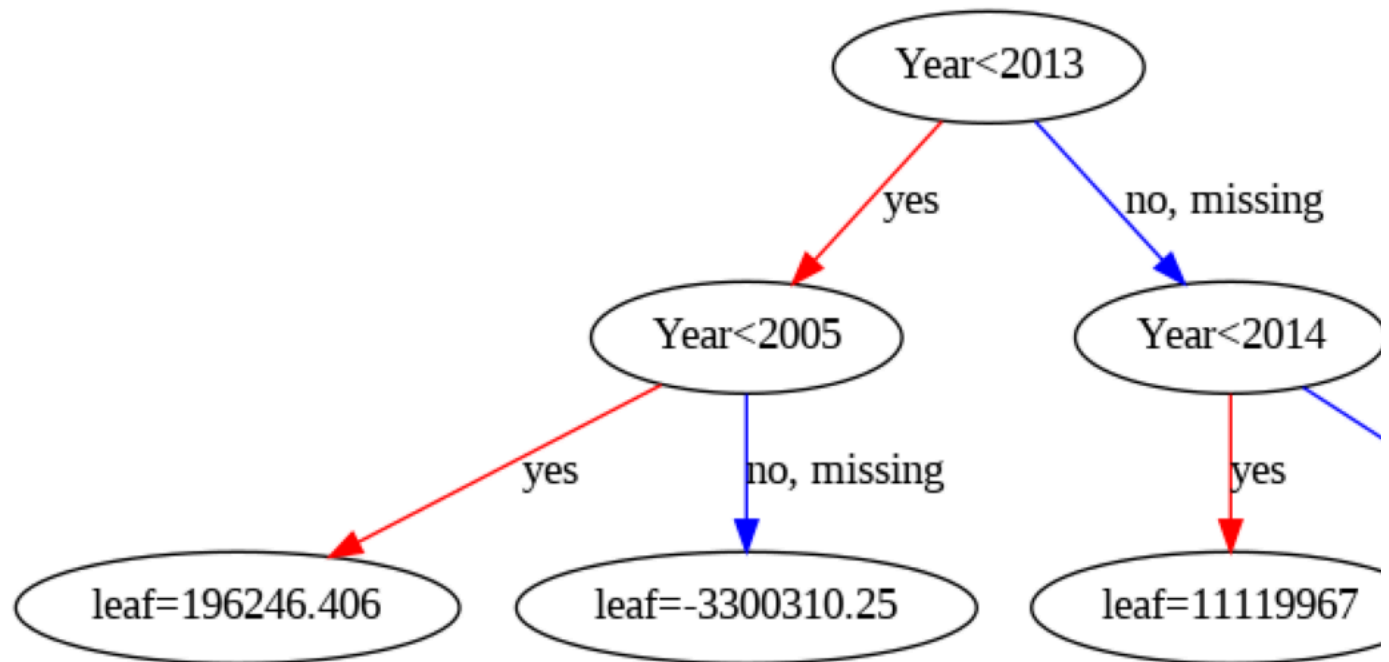
from sklearn.tree import export_graphviz
import pydot
tree = rf.model.estimators_[5]
export_graphviz(tree, out_file = 'tree.dot', feature_names = df.drop(['Records', 'id'],axis=1).columns, rounded = True, precision = 1)
(graph, ) = pydot.graph_from_dot_file('tree.dot')

graph.write_png('tree.png')

from xgboost import plot_tree
fig, ax = plt.subplots(figsize=(30, 30))
tree = xgboosting.model
plot_tree(tree, ax=ax)
plt.show()

```





```
pred = xgboosting.model.predict(test_X)
```



```

from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
model = Sequential()
model.add(Dense(1000, input_shape=(train_X.shape[1],), activation='relu')) # (features,)
model.add(Dense(500, activation='relu'))
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='linear')) # output node
model.summary() # see what your model looks like

```

```

# compile the model
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

```

```

# early stopping callback
es = EarlyStopping(monitor='val_loss',
                  mode='min',
                  patience=50,
                  restore_best_weights = True)

```

```

# fit the model!
# attach it to a new variable called 'history' in case
# to look at the learning curves
history = model.fit(train_X, train_y,
                  validation_data = (test_X, test_y),
                  callbacks=[es],
                  epochs=10,
                  batch_size=50,
                  verbose=1)

```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 1000)	5000
dense_7 (Dense)	(None, 500)	500500
dense_8 (Dense)	(None, 250)	125250
dense_9 (Dense)	(None, 1)	251

```

Total params: 631001 (2.41 MB)
Trainable params: 631001 (2.41 MB)
Non-trainable params: 0 (0.00 Byte)

```

```

Epoch 1/10
5/5 [=====] - 1s 68ms/step - loss: 43607421431578624.0000 - mae: 43308092.0000 - val_loss: 22201899281285120.00
Epoch 2/10
5/5 [=====] - 0s 24ms/step - loss: 43606150121259008.0000 - mae: 43291628.0000 - val_loss: 22199633686036480.00
Epoch 3/10
5/5 [=====] - 0s 31ms/step - loss: 43603139349184512.0000 - mae: 43263412.0000 - val_loss: 22194977941487616.00
Epoch 4/10
5/5 [=====] - 0s 35ms/step - loss: 43598066992807936.0000 - mae: 43219276.0000 - val_loss: 22188039421820928.00
Epoch 5/10
5/5 [=====] - 0s 35ms/step - loss: 43591824005261440.0000 - mae: 43162872.0000 - val_loss: 22180860282085664.00

```

```

history_dict = history.history
loss_values = history_dict['mae'] # you can change this
val_loss_values = history_dict['val_mae'] # you can also change this
epochs = range(1, len(loss_values) + 1) # range of X (no. of epochs)
plt.plot(epochs, loss_values, 'bo', label='Training MAE')
plt.plot(epochs, val_loss_values, 'orange', label='Validation MAE')

```

