Basic imports and function ceil

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller #to see if timeseries is stationary
import numpy as np
import statsmodels.formula.api as smf
import statsmodels.api as sm
import datetime
```

```
# Fonction pour majorer ou minorer un nombre à l'entier le plus proche

def my_ceil(predictions):
  for i in range(len(predictions)):
    if predictions[i]%1<=0.5:
      predictions[i] = int(predictions[i])
    else:
      predictions[i] = int(predictions[i]) + 1
  return predictions
```

```
import io
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in

```
Saving test_input.csv to test_input.csv
Saving train_input.csv to train_input.csv
Saving train_output.csv to train_output.csv
```

Dataset Exploration

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import datetime
```

```
%%capture
train_input= pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_wel
train_output= pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_we
train= train_input
train['Score']= train_output.Score
train.head()
for i in range(len(train['Date'])):
  train['Date'][i] = datetime.datetime.strptime(train['Date'][i], '%Y-%m-%d %H:%M:%S')
```

```
print(train.shape)
print(train.head())

```

Verifying if any na value in our dataframe, the result of this query shows us that there is none.

```
train.isnull().values.any()
```

Distribution of different features:

```
plt.figure("Distribution Plots")
fig = plt.figure(figsize = (24,12))
plt.subplot(2, 3,  1)
sns.distplot(np.log(train.Temperature), label = 'Temperature')
plt.subplot(2, 3, 2)
sns.distplot(np.log(train.Humidity), label = 'Humidity')
plt.subplot(2, 3, 3)
sns.distplot(train.Humex, label = 'Humex')
plt.subplot(2, 3, 4)
sns.distplot(np.log(train.CO2), label = 'CO2')
# sns.distplot(well_BE.month, label = 'month')
plt.legend()
plt.show()
```

We have the only varibale who has a normal distribution is the Humex variable which represents the a

Matrice de correlation

```
corr = train.corr()
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```

```
corr
```

Again, there is only one variable **Humidty** which is hihgly correlated with our traget variable **score**

Relation between our target variable and the variable Date

```
series = pd.Series(np.array(train.Score), index=train.Date)
groupHour = series.groupby(series.index.hour).mean()
groupHour = pd.DataFrame({'hour':groupHour.index, 'score':groupHour.values})

# plt.plot(groupHour.hour, groupHour.score)
fig, ax = plt.subplots(1,1, figsize = (12,6))
#sns.barplot(x = 'hour', y = 'score', data = groupHour)

sns.lineplot(x = 'hour', y = 'score', data = groupHour)
sns.set_style("ticks", {"xtick.major.size": 16, "ytick.major.size": 8})
```

From the graph before which reprensents the score in function of day hours, the confort subjectif dro
maybe people are working at this hours and they are less confortable.

```
series = pd.Series(np.array(train.Score), index=train.Date)
groupHour = series.groupby(series.index.month).mean()
groupHour = pd.DataFrame({'month':groupHour.index, 'score':groupHour.values})

fig, ax = plt.subplots(1,1, figsize = (12,6))
#sns.barplot(x = 'month', y = 'score', data = groupHour)
sns.lineplot(x = 'month', y = 'score', data = groupHour)
```

We did a variation of the score in function of months, but it is insignificant, since we have only few mo

Tableau software

In order to have a better idea about our dataset we used the software Tableau

```
from IPython.core.display import Image, display
display(Image('/content/score(autresvariables).PNG'))
```

The graph shows us that there is no apprent relation between our traget and features

```
display(Image('/content/variable(hours).PNG'))

```

The graph shows us that except the bright and CO2, the other features are constant during the day.

```
display(Image('/content/score(day).PNG'))

```

this graph shows us that there is no apparent seasonality concerning the score and the time.

```
display(Image('/content/score(month).PNG'))
```

This graph shows us that the score is dropping from August to Decemeber and maybe of the season

Conclusion

After exploring of our data, in the next section we will preprocess the data for modelisation.

Preprocessing phase

Train data

Import train data

```
wellB_in = pd.read_csv(io.BytesIO(uploaded['train_input.csv']))
wellB_out = pd.read_csv(io.BytesIO(uploaded['train_output.csv']))
well_B = wellB_in
```

```
# Convert Date column from String to Date
for i in range(len(well_B['Date'])):
  well_B['Date'][i] = datetime.datetime.strptime(well_B['Date'][i], '%Y-%m-%d %H:%M:%S')

train_ID = well_B.ID
train_Date = well_B.Date

well_B = well_B.drop(['ID','Date'], axis = 1)
```

Add weekdays to columns as dummies

```
well_B['weekday'] = train_Date

# Add weekday column to data
for i in range(len(well_B.weekday)):
  well_B.weekday[i] = well_B.weekday[i].weekday()
```

```
dummy_weekday = pd.get_dummies(well_B['weekday'])
dummy_weekday.columns = ['lundi','mardi','mercredi','jeudi','vendredi','samedi','dimanch

# dummy_weekday.rename(columns = {'0':'lundi','1':'mardi','2':'mercredi','3':'jeudi','4'
# dummy_weekday.head()

well_B = pd.concat([well_B,dummy_weekday], axis=1)
```

```
# Drop Weekday column with categorical values
well_B = well_B.drop('weekday', axis = 1)
```

Add hours to columns as dummy variables

```
well_B['hour'] = train_Date

for i in range(len(well_B.hour)):
  well_B.hour[i] = well_B.hour[i].hour

dummy_hour = pd.get_dummies(well_B['hour'])
```

```
dummy_hour.columns = ['midnight','AM1','AM2','AM3','AM4','AM5','AM6','AM7','AM8','AM9','

well_B = pd.concat([well_B,dummy_hour], axis = 1)

well_B = well_B.drop('hour', axis = 1)


well_B.head()
```

```
# Add objective variable to data
well_B['Score'] = wellB_out.Score
```

Test Data

Import test data

↳ 1 cellule masquée

Add weekdays to columns as dummies

↳ 3 cellules masquées

Add hours to columns as dummy variables

↳ 1 cellule masquée

K-fold Cross validation: regression and classification

Imports and functions

```
from sklearn.model_selection import KFold # import KFold
from sklearn.metrics import accuracy_score #To calculate accuracy
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

```
kf = KFold(n_splits = 7)
```

```
def cross_validation_XG_classifier(model):
  accuracy = []
  for train_index, test_index in kf.split(well_B):
    # print("TRAIN:", train_index, "TEST:", test_index)
    #train part
    X_train = np.array(well_B.drop('Score', axis = 1))[train_index]
    y_train = np.array(well_B.Score)[train_index]
    #test part
    X_test = np.array(well_B.drop('Score', axis = 1))[test_index]
    y_test = np.array(well_B.Score)[test_index]

    # Define the scaler
    # scaler = StandardScaler().fit(X_train)

    # X_train = scaler.transform(X_train)
    # X_test = scaler.transform(X_test)

    # rgr_time = RandomForestClassifier(n_estimators = 400, random_state = 0, max_depth=
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    y_pred = my_ceil(np.array(y_pred))

    accuracy.append(accuracy_score(y_pred, y_test))
    # print("The score of the " + str(cpt) + " is " + str(accuracy_score(y_pred, y_test)
    # cpt = cpt+1

  return np.average(accuracy)

# Converts floats to integers for classification
def my_ceil(predictions):
  for i in range(len(predictions)):

    if predictions[i]%1 <= +0.5:
      predictions[i] = int(predictions[i])
    else:
      predictions[i] = int(predictions[i]) + 1
  return predictions


def cross_validation_Lregressor(formula):
  accuracy = []
  for train_index, test_index in kf.split(well_B):
    # print("TRAIN:", train_index, "TEST:", test_index)
    #train part
    train = well_B.iloc[train_index]
    #test part
    test = well_B.iloc[test_index]

    # Define the scaler
    # scaler = StandardScaler().fit(X_train)

    # X_train = scaler.transform(X_train)
    # X_test = scaler.transform(X_test)

    model_lr = formula
    result_lr = smf.ols(model_lr, data = train).fit()

    y_pred = np.array(result_lr.predict(test))

    y_pred = my_ceil(y_pred)

    accuracy.append(accuracy_score(y_pred, test.Score))
    # print("The score of the " + str(cpt) + " is " + str(accuracy_score(y_pred, y_test)
    # cpt = cpt+1

  return np.average(accuracy)

```

Regression

Linear regression

↳ 10 cellules masquées

XGboost

↳ 3 cellules masquées

Classification

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from google.colab import files
```

```
def my_ceil(predictions):
  for i in range(len(predictions)):
    if predictions[i]%1<=0.5:
      predictions[i] = int(predictions[i])
    else:
      predictions[i] = int(predictions[i]) + 1
  return predictions

#Export function
def export( data_test, predictions):
  result_ = pd.DataFrame({'ID': data_test.ID, 'Score': my_ceil(predictions)})
  result_.to_csv('results_.csv', index = False)

  files.download('results_.csv')

```

Upload preprocessed dataset train and test

```
test = pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_well_bein
train = pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_well_bei
test_with_ID = pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_w
test_with_date= pd.read_csv('https://raw.githubusercontent.com/nisrineha/Challenge_data_
```

```
test_with_date.head()

```

Creation of features and the target variable Score

```
y= train['Score']
train1= train
train1= train1.drop('Score', axis= 1)
X= train1
train1.head()
```

Spliting of the data

```
X_train, X_test, y_train, y_test= train_test_split(X, y , test_size= 0.2, random_state=
```

Pipeline method

Implement of pipeline method using different transformer: numeric and categorial

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

#SimpleImputer fill any missing values
#Scaler numeric transformer

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())])

#One hot encoder to transform categorial values into integers.

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

Transform the categorical features and numeric on train dataset and test

```
#Select les columns numeric
#Select les columns categoric


integer_features = list(X.columns[X.dtypes == 'int64'])
continuous_features = list(X.columns[X.dtypes == 'float64'])
categorical_features = list(X.columns[X.dtypes == 'object'])
numeric_features = integer_features + continuous_features


from sklearn.compose import ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

integer_features_test = list(test.columns[test.dtypes == 'int64'])
continuous_features_test = list(test.columns[test.dtypes == 'float64'])
categorical_features_test = list(test.columns[test.dtypes == 'object'])
numeric_features = integer_features + continuous_features
```

```


from sklearn.compose import ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

Model selection

In this section, we chose different classifier from sklearn, to get the best classifier for our dataset, we of the dataset that we did before.

```
from sklearn.metrics import accuracy_score, log_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostin
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
classifiers = [
    KNeighborsClassifier(3),
    SVC(kernel="rbf", C=0.025, probability=True),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    LinearDiscriminantAnalysis(),
    QuadraticDiscriminantAnalysis()
    ]
pipes= []
for classifier in classifiers:
    pipe = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', classifier)])
    pipe.fit(X_train, y_train)
    pipes.append(pipe)
    print(classifier)
    print("model score: %.3f" % pipe.score(X_test, y_test))

```

From the results above, we observe that the best classifiers are **GradientBoostingClassifier** and **Rand**

we chose GradientBoostingClassifier for submission, we had our best score which is **0,6990**

```
x = test
y_pred= pipes[-3].predict(x)
export(test_with_ID, y_pred)
```

Using pipeline in GridSearch

```
param_grid = {
    'classifier__n_estimators': [ 200, 300,  400, 500],
    'classifier__max_features': ['auto', 'sqrt', 'log2'],
    'classifier__max_depth' : [10, 20, 25, 30],
    'classifier__criterion' :['gini', 'entropy']}
from sklearn.model_selection import GridSearchCV
CV = GridSearchCV(rf, param_grid, n_jobs= 1)

CV.fit(X_train, y_train)
print(CV.best_params_)
print(CV.best_score_)
```

we had from our grid search our best max depth and number of estimators for our randomforset moc

```
#Fitting the classifier
from sklearn.ensemble import RandomForestClassifier
rf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', RandomForestClassifier(n_estimators= 400, max_depth
```

```
rf.fit(X_train, y_train )

pipe = Pipeline(steps=[('preprocessor', preprocessor),
                  ('classifier',RandomForestClassifier(n_estimators= 400, max_depth=20)
pipe.fit(X_train, y_train)
print(classifier)
print("model score: %.3f" % pipe.score(X_test, y_test))
x = test
y_pred= pipe.predict(x)
export(test_with_ID, y_pred)
```

We submitted our results, but we had a score less than the one when we implemented **GradientBoost**

Random forest model

↳ 2 cellules masquées

K nearest neighbor

↳ 2 cellules masquées

Gradient Boosting Classification

↳ 3 cellules masquées

Basic deep learning model

```
# Import `Sequential` from `keras.models`
from keras.models import Sequential

# Import `Dense` from `keras.layers`
from keras.layers import Dense

# Initialize the constructor
model = Sequential()

# Add an input layer
model.add(Dense(12, activation='softmax', input_shape=(36,)))

# Add one hidden layer
model.add(Dense(12, activation='relu'))

# # Add one hidden layer
# model.add(Dense(12, activation='relu'))

# Add an output layer
model.add(Dense(output_dim = 5, activation = 'softmax'))
```

```
def dummies_categ(y):
  cat = []
  for i in range(len(y)):
    ind = np.argmax(y_pred[i]) + 1
    cat.append(ind)
  return cat
```

```
accuracy = []
for train_index, test_index in kf.split(well_B):
  # print("TRAIN:", train_index, "TEST:", test_index)
  #train part
  X_train = np.array(well_B.drop('Score', axis = 1))[train_index]
  y_train = np.array(well_B.Score)[train_index]

  y_train = pd.get_dummies(y_train)
```

```
    #test part
    X_test = np.array(well_B.drop('Score', axis = 1))[test_index]
    y_test = np.array(well_B.Score)[test_index]

    y_test = pd.get_dummies(y_test)


    # Define the scaler
    scaler = StandardScaler().fit(X_train)

    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

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

    model.fit(X_train, y_train,epochs=2, batch_size=1, verbose=1)

    y_pred = model.predict(X_test)

    y_pred = dummies_categ(y_pred)
    y_test = np.array(well_B.Score)[test_index]
    # print(y_pred)
    accuracy.append(accuracy_score(y_pred, y_test))
    # print("The score of the " + str(cpt) + " is " + str(accuracy_score(y_pred, y_test)))
    # cpt = cpt+1

np.average(accuracy)
```