Importer les packages et les données

In [3]:
```
import pyforest
import lazypredict
from lazypredict.Supervised import LazyClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
# Hide warnings
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
data = pd.read_csv('Agricol_Urbanis_Data.csv')
data = data.drop('Unnamed: 0', axis = 1)
```

In [8]:
```
data.head()
```

Out[8]:

	region	Date	Ménages exploitant moins de 3 à 5 parcelles (%)	Ménages exploitant plus de 5 à 10 parcelles (%)	Ménages exploitant plus de 5 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages	Effectif de la population	P...
0	Dakar	2017	96.80	40.36	9.93	1.21	0.46	4.90	3529306.00	3
1	Dakar	2018	100.00	0.00	0.00	1.04	0.47	5.36	3630324.00	3
2	Diourbel	2017	12.54	64.68	22.78	4.34	3.08	9.52	1692967.00	1
3	Diourbel	2018	25.74	57.75	16.51	3.82	4.65	10.41	1746496.00	1
4	Fatick	2017	37.25	52.98	9.77	3.30	2.88	7.97	813542.00	1

Créer une variable cible

In [9]:
```
deforestation_risk = [1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1]
data['deforestation_risk'] = deforestation_risk
```

- 1 : signifie que la zone est en menace de déforestation
- 0 : signifie que la zone n'est en menace de déforestation

Pour créer la variable cible, on a considéré les regions qui ont un taux d'urbanisation supérieur à 40 % et celles caractérisées par une avancée du desert menacés de déforestation.

Jeu de donnée final avec la variable cible créé

In [10]:
```
data.head()
```

Out[10]:

	region	Date	Ménages exploitant moins de 3 à 5 parcelles (%)	Ménages exploitant plus de 5 à 10 parcelles (%)	Ménages exploitant plus de 5 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages	Effectif de la population	P...
0	Dakar	2017	96.80	40.36	9.93	1.21	0.46	4.90	3529306.00	3
1	Dakar	2018	100.00	0.00	0.00	1.04	0.47	5.36	3630324.00	3
2	Diourbel	2017	12.54	64.68	22.78	4.34	3.08	9.52	1692967.00	1
3	Diourbel	2018	25.74	57.75	16.51	3.82	4.65	10.41	1746496.00	1
4	Fatick	2017	37.25	52.98	9.77	3.30	2.88	7.97	813542.00	1

Exploration des données

In [11]:
```
pd.set_option('display.float', "{:.2f}".format)
data.describe()
```

Out[11]:

Data	Ménages exploitant moins de 3 à 5 parcelles (%)	Ménages exploitant plus de 5 à 10 parcelles (%)	Ménages exploitant plus de 5 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages	Effectif de la population	Popu
count	28.00	28.00	28.00	28.00	28.00	28.00	28.00	
mean	2017.50	53.93	40.36	9.93	2.77	2.69	9.31	1156515.00	551
std	0.51	28.58	19.57	7.18	1.06	1.76	1.95	837422.52	330
min	2017.00	12.54	0.00	0.00	1.04	0.46	4.90	172402.00	126
25%	2017.00	29.62	34.40	5.13	2.16	1.03	8.53	650086.00	412
50%	2017.50	54.40	41.40	9.85	2.67	2.87	9.25	812808.50	546
75%	2018.00	71.80	56.79	11.67	3.55	3.76	10.54	1094949.00	698
max	2018.00	100.00	67.12	24.93	4.65	8.03	12.34	3630324.00	1452

In [12]:
```
data.deforestation_risk.value_counts().plot(kind="bar", color=["salmon", "lightblue"])
```

Out[12]:
<AxesSubplot:>

In [13]:
```
data.isna().sum()
```

Out[13]:
```
region                                  0
Date                                    0
Ménages exploitant moins de 3 parcelles (%)   0
Ménages exploitant plus de 3 à 5 parcelles (%)   0
Ménages exploitant plus de 4 parcelles (%)   0
Nombre moyen de parcelles par ménage    0
Superficie moyenne des parcelles par ménage (Ha)   0
Taille moyenne des ménages              0
Effectif de la population               0
Population rurale                       0
Population urbaine                      0
Taux d'urbanisation                     0
deforestation_risk                      0
dtype: int64
```

Matrice de corrélation

In [14]:
```
# Let's make our correlation matrix a little prettier
corr_matrix = data.corr()
fig, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[14]:
(12.5, -0.5)

In [15]:
```
data.drop('deforestation_risk', axis=1).corrwith(data.deforestation_risk
).plot(kind='bar', grid=True, figsize=(12, 8),
                                                    title="Correlation av
ec deforestation_risk")
```

Out[15]:
<AxesSubplot:title={'center':'Correlation avec deforestation_risk'}>

Transformation des données

In [16]:
```
X = data.drop(['region', 'Date', 'deforestation_risk'], axis=1)
y = data.deforestation_risk
```

In [17]:
```
# categorical_val.remove('target')
# dataset = pd.get_dummies(df, columns = categorical_val)
```

```
from sklearn.preprocessing import StandardScaler

s_sc = StandardScaler()
# col_to_scale = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
# dataset[col_to_scale] = s_sc.fit_transform(dataset[col_to_scale])
X = s_sc.fit_transform(X)
```

Fractionner les données

In [18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

Appliquer plusieurs modèles

In [19]:
```
clf = LazyClassifier(verbose=0,ignore_warnings=True)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)
models
```

```
100%|██████████| 29/29 [00:02<00:00, 12.57it/s]
```

Out[19]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
AdaBoostClassifier	0.89	0.92	0.92	0.89	0.26
LabelPropagation	0.89	0.92	0.92	0.89	0.01
SVC	0.89	0.92	0.92	0.89	0.02
RidgeClassifierCV	0.89	0.92	0.92	0.89	0.02
RandomForestClassifier	0.89	0.92	0.92	0.89	0.17
PassiveAggressiveClassifier	0.89	0.92	0.92	0.89	0.03
NuSVC	0.89	0.92	0.92	0.89	0.02
NearestCentroid	0.89	0.92	0.92	0.89	0.01
LogisticRegression	0.89	0.92	0.92	0.89	0.04
LabelSpreading	0.89	0.92	0.92	0.89	0.02
LinearSVC	0.89	0.92	0.92	0.89	0.02
GaussianNB	0.89	0.92	0.92	0.89	0.02
ExtraTreesClassifier	0.89	0.92	0.92	0.88	0.13
CalibratedClassifierCV	0.89	0.92	0.92	0.89	0.09
BernoulliNB	0.89	0.92	0.92	0.89	0.02
ExtraTreeClassifier	0.89	0.83	0.83	0.88	0.01
Perceptron	0.78	0.83	0.83	0.78	0.02
RidgeClassifier	0.78	0.83	0.83	0.78	0.13
SGDClassifier	0.78	0.83	0.83	0.78	0.01
KNeighborsClassifier	0.78	0.75	0.75	0.78	0.04
LinearDiscriminantAnalysis	0.67	0.75	0.75	0.67	0.17
BaggingClassifier	0.78	0.75	0.75	0.78	0.04
DecisionTreeClassifier	0.67	0.75	0.75	0.67	0.02
XGBClassifier	0.78	0.75	0.75	0.78	0.63
DummyClassifier	0.50	0.50	0.50	0.56	0.04
QuadraticDiscriminantAnalysis	0.67	0.50	0.50	0.53	0.11
LGBMClassifier	0.67	0.50	0.50	0.53	0.19

Appliquer le meilleur modèle

In [20]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classifica
tion_report

def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, o
utput_dict=True))
        print("Train Result:\n================================================
======")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f
}%")
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n
")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, ou
tput_dict=True))
        print("Test Result:\n================================================
======")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}
%")
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n"
)
```

In [21]:
```
from sklearn.neighbors import NearestCentroid
b_clf = NearestCentroid()
b_clf.fit(X_train, y_train)
print_score(b_clf, X_train, y_train, X_test, y_test, train=True)
print_score(b_clf, X_train, y_train, X_test, y_test, train=False)
```

```
Train Result:
================================================
Accuracy Score: 94.74%

_____________________________________________
CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  weighted avg
precision  1.00  0.88      0.95       0.94          0.95
recall     0.92  1.00      0.95       0.96          0.95
f1-score   0.96  0.93      0.95       0.94          0.95
support   12.00  7.00      0.95      19.00         19.00
_____________________________________________
Confusion Matrix:
 [[11  1]
 [ 0  7]]

Test Result:
================================================
Accuracy Score: 88.89%

_____________________________________________
CLASSIFICATION REPORT:
              0    1  accuracy  macro avg  weighted avg
precision  1.00  0.75      0.89       0.88          0.92
recall     0.83  1.00      0.89       0.92          0.89
f1-score   0.91  0.86      0.89       0.88          0.89
support    6.00  3.00      0.89       9.00          9.00
_____________________________________________
Confusion Matrix:
 [[5 1]
 [0 3]]
```

- Le modèle est performant à environ 95 % sur les données d'entrainement et à environ 89 % sur les données de test.