Importer les packages et les données

In [2]:

```
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 [3]:

```
data = pd.read_csv('Agricul_Urbanis_Data.csv')
data = data.drop('Unnamed: 0', axis = 1)
```

In [4]:

```
data.head()
```

Out[4]:

	region	Date	Ménages exploitant moins de 3 parcelles (%)	Ménages exploitant 3 à 5 parcelles (%)	Ménages exploitant plus de 6 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages
0	Dakar	2017	96.80	40.36	9.93	1.21	0.46	4.90
1	Dakar	2018	100.00	0.00	0.00	1.04	0.47	5.36
2	Diourbel	2017	12.54	64.68	22.78	4.34	3.08	9.52
3	Diourbel	2018	25.74	57.75	16.51	3.82	4.65	10.41
4	Fatick	2017	37.25	52.98	9.77	3.30	2.88	7.97

Créer une variable cible

In [5]:

```
risque_deforestation = [1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,
1,1,1]
data['risque_deforestation'] = risque_deforestation
```

- 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 considèré 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 crée

In [6]:

```
data.head()
```

Out[6]:

	region	Date	Ménages exploitant moins de 3 parcelles (%)	Ménages exploitant 3 à 5 parcelles (%)	Ménages exploitant plus de 6 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages
0	Dakar	2017	96.80	40.36	9.93	1.21	0.46	4.90
1	Dakar	2018	100.00	0.00	0.00	1.04	0.47	5.36
2	Diourbel	2017	12.54	64.68	22.78	4.34	3.08	9.52
3	Diourbel	2018	25.74	57.75	16.51	3.82	4.65	10.41
4	Fatick	2017	37.25	52.98	9.77	3.30	2.88	7.97

Exploration des données

In [7]:

```
pd.set_option("display.float", "{:.2f}".format)
data.describe()
```

Out[7]:

	Date	Ménages exploitant moins de 3 parcelles (%)	Ménages exploitant 3 à 5 parcelles (%)	Ménages exploitant plus de 6 parcelles (%)	Nombre moyen de parcelles par ménage	Superficie moyenne des parcelles par ménage (Ha)	Taille moyenne des ménages	
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	1
std	0.51	28.58	19.57	7.18	1.06	1.76	1.95	
min	2017.00	12.54	0.00	0.00	1.04	0.46	4.90	
25%	2017.00	29.62	34.40	5.13	2.16	1.03	8.53	
50%	2017.50	54.40	41.40	9.85	2.67	2.87	9.25	
75%	2018.00	71.80	56.79	11.67	3.55	3.76	10.54	1
max	2018.00	100.00	67.12	24.93	4.49	8.03	12.34	3

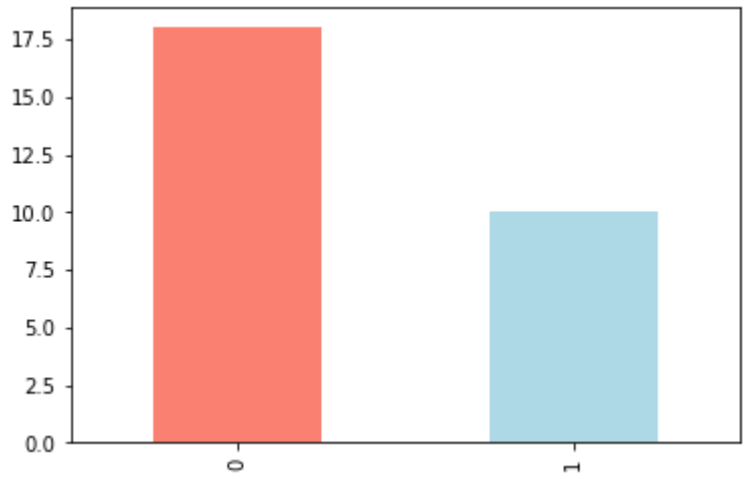
In [8]:

```
data.risque_deforestation.value_counts().plot(kind="bar", color=["salmon",
```

```
"lightblue"])
```

Out[8]:

```
<AxesSubplot:>
```



In [9]:

```
data.isna().sum()
```

Out[9]:

```
region                                                0
Date                                                  0
Ménages exploitant moins de 3 parcelles (%)           0
Ménages exploitant 3 à 5 parcelles (%)                0
Ménages exploitant plus de 6 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
risque_deforestation                                  0
dtype: int64
```

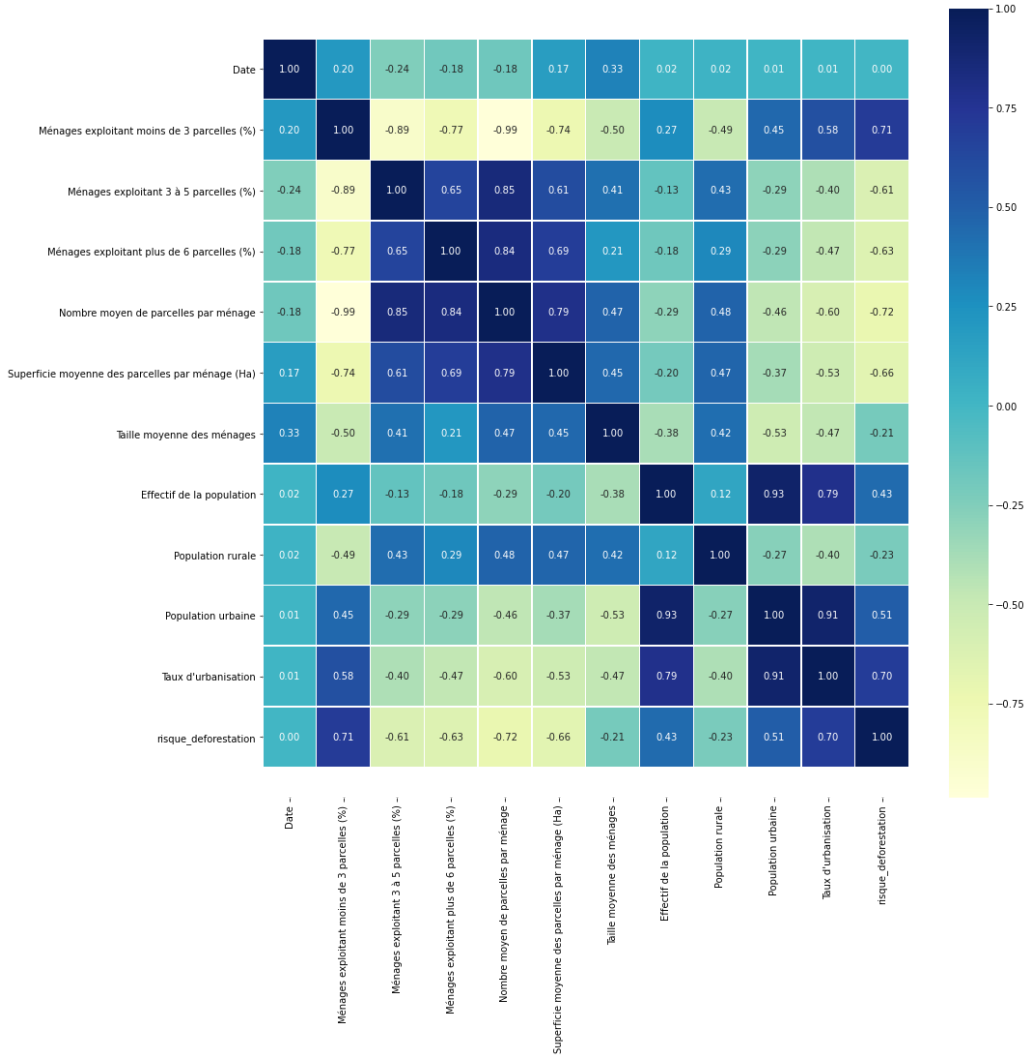
Matrice de corrélation

In [10]:

```
# 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[10]:

```
(12.5, -0.5)
```

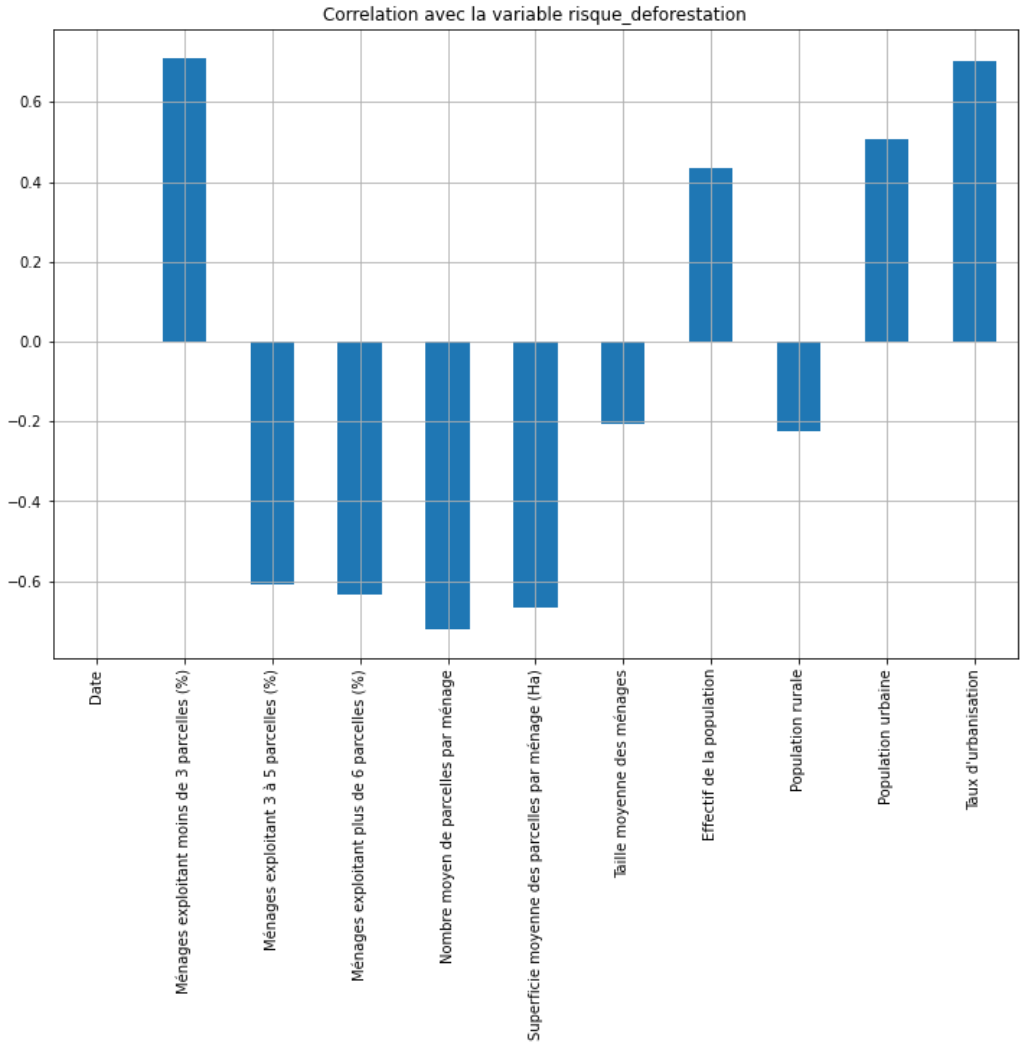


In [11]:

```
data.drop('risque_deforestation', axis=1).corrwith(data.risque_deforestation).plot(kind='bar', grid=True, figsize=(12, 8),
                                                  title="Correlation avec la variable risque_deforestation")
```

Out[11]:

```
<AxesSubplot:title={'center':'Correlation avec la variable risque_deforestation'}>
```



- On constate que les variable 'Ménages exploitant moins de 3 parcelles (%)' et 'Taux d'urbanisation' sont fortement correlée avec la variavle déforestation_risk. Ainsi on peut considèrer que ces deux variables sont de bons prédicteurs du risque de déforestation.
- Les variables 'Nombre moyen de parcelles par ménage' et 'Superficie moyenne des parcelles par ménage (Ha)' constituent aussi des bons prédicteurs du risque de déforestation puisqu'elles sont faiblement avec la variable deforestation.

Sélectionner que les variables les plus ou moins correlées à la variable cible

In [13]:

```
columns_selected  = ['Ménages exploitant moins de 3 parcelles (%)', 'Nombr
e moyen de parcelles par ménage',
                     'Superficie moyenne des parcelles par ménage (Ha)', "T
```

```
aux d'urbanisation", 'risque_deforestation']
data = data[columns_selected]
```

Transformation des données

In [14]:

```
X = data[columns_selected]
y = data.risque_deforestation
```

In [15]:

```
# 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 [16]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, r
andom_state=42)
```

Appliquer plusieurs modèles

In [17]:

```
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, 13.90it/s]
```

Out[17]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
AdaBoostClassifier	1.00	1.00	1.00	1.00	0.17
LinearDiscriminantAnalysis	1.00	1.00	1.00	1.00	0.17
XGBClassifier	1.00	1.00	1.00	1.00	0.54
SVC	1.00	1.00	1.00	1.00	0.02
RidgeClassifierCV	1.00	1.00	1.00	1.00	0.02

RidgeClassifier	1.00	1.00	1.00	1.00	0.12
RandomForestClassifier	1.00	1.00	1.00	1.00	0.19
PassiveAggressiveClassifier	1.00	1.00	1.00	1.00	0.03
NuSVC	1.00	1.00	1.00	1.00	0.03
LogisticRegression	1.00	1.00	1.00	1.00	0.04
BaggingClassifier	1.00	1.00	1.00	1.00	0.04
LinearSVC	1.00	1.00	1.00	1.00	0.02
LabelSpreading	1.00	1.00	1.00	1.00	0.01
LabelPropagation	1.00	1.00	1.00	1.00	0.01
KNeighborsClassifier	1.00	1.00	1.00	1.00	0.04
GaussianNB	1.00	1.00	1.00	1.00	0.02
ExtraTreesClassifier	1.00	1.00	1.00	1.00	0.13
ExtraTreeClassifier	1.00	1.00	1.00	1.00	0.02
DecisionTreeClassifier	1.00	1.00	1.00	1.00	0.02
CalibratedClassifierCV	1.00	1.00	1.00	1.00	0.08
BernoulliNB	1.00	1.00	1.00	1.00	0.01
NearestCentroid	0.89	0.92	0.92	0.89	0.01
Perceptron	0.89	0.92	0.92	0.89	0.02
SGDClassifier	0.89	0.92	0.92	0.89	0.02
QuadraticDiscriminantAnalysis	0.33	0.50	0.50	0.17	0.09
DummyClassifier	0.56	0.50	0.50	0.56	0.04
LGBMClassifier	0.67	0.50	0.50	0.53	0.14

Appliquer le meilleur modèle

In [18]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classificati
on_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, out
put_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, outp
```

```
ut_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 [19]:

```
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: 100.00%

_____________________________________________
CLASSIFICATION REPORT:
                0    1  accuracy  macro avg  weighted avg
precision    1.00 1.00      1.00       1.00          1.00
recall       1.00 1.00      1.00       1.00          1.00
f1-score     1.00 1.00      1.00       1.00          1.00
support     12.00 7.00      1.00      19.00         19.00

_____________________________________________
Confusion Matrix:
 [[12  0]
 [ 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 100 % sur les données d'entrainement et à environ 89 % sur les données de test.

In [ ]:

```
```