

In [4]:

```
import pandas as pd
```

In [5]:

```
df = pd.read_csv('blueberry_yield.csv')
```

In [6]:

```
df.head()
```

Out[6]:

	id	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange	MaxOfLowerTRange
0	0	25.0	0.50	0.25	0.75	0.50	69.7	42.1	58.2	50.2
1	1	25.0	0.50	0.25	0.50	0.50	69.7	42.1	58.2	50.2
2	2	12.5	0.25	0.25	0.63	0.63	86.0	52.0	71.9	62.0
3	3	12.5	0.25	0.25	0.63	0.50	77.4	46.8	64.7	55.8
4	4	25.0	0.50	0.25	0.63	0.63	77.4	46.8	64.7	55.8

In [7]:

```
df.tail()
```

Out[7]:

	id	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange	MaxOfLowe
15284	15284	12.5	0.25	0.25	0.38	0.50	77.4	46.8	64.7	
15285	15285	12.5	0.25	0.25	0.25	0.50	86.0	52.0	71.9	
15286	15286	25.0	0.50	0.25	0.38	0.75	77.4	46.8	64.7	
15287	15287	25.0	0.50	0.25	0.63	0.63	69.7	42.1	58.2	
15288	15288	25.0	0.50	0.25	0.63	0.50	77.4	46.8	64.7	

In [8]:

```
df.shape
```

Out[8]:

(15289, 18)

In [9]:

```
df.columns
```

Out[9]:

```
Index(['id', 'clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia',  
      'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',  
      'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange',  
      'RainingDays', 'AverageRainingDays', 'fruitset', 'fruitmass', 'seeds',  
      'yield'],  
      dtype='object')
```

In [10]:

```
df.duplicated().sum()
```

Out[10]:

0

In [11]:

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

Out[11]:

```
id                0
clonesize         0
honeybee          0
bumbles          0
andrena           0
osmia             0
MaxOfUpperTRange  0
MinOfUpperTRange  0
AverageOfUpperTRange  0
MaxOfLowerTRange  0
MinOfLowerTRange  0
AverageOfLowerTRange  0
RainingDays       0
AverageRainingDays 0
fruitset          0
fruitmass         0
seeds             0
yield             0
dtype: int64
```

In [12]:

```
df = df.drop('id', axis = 1)
```

In [13]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15289 entries, 0 to 15288
Data columns (total 17 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   clonesize              15289 non-null  float64
1   honeybee               15289 non-null  float64
2   bumbles                15289 non-null  float64
3   andrena                15289 non-null  float64
4   osmia                  15289 non-null  float64
5   MaxOfUpperTRange       15289 non-null  float64
6   MinOfUpperTRange       15289 non-null  float64
7   AverageOfUpperTRange   15289 non-null  float64
8   MaxOfLowerTRange       15289 non-null  float64
9   MinOfLowerTRange       15289 non-null  float64
10  AverageOfLowerTRange   15289 non-null  float64
11  RainingDays            15289 non-null  float64
12  AverageRainingDays     15289 non-null  float64
13  fruitset               15289 non-null  float64
14  fruitmass              15289 non-null  float64
15  seeds                  15289 non-null  float64
16  yield                  15289 non-null  float64
dtypes: float64(17)
memory usage: 2.0 MB
```

In [14]:

```
df.describe()
```

Out[14]:

	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRar
count	15289.000000	15289.000000	15289.000000	15289.000000	15289.000000	15289.000000	15289.000000	15289.000000
mean	19.704690	0.389314	0.286768	0.492675	0.592355	82.169887	49.673281	68.656281
std	6.595211	0.361643	0.059917	0.148115	0.139489	9.146703	5.546405	7.641151
min	10.000000	0.000000	0.000000	0.000000	0.000000	69.700000	39.000000	58.200000
25%	12.500000	0.250000	0.250000	0.380000	0.500000	77.400000	46.800000	64.700000
50%	25.000000	0.500000	0.250000	0.500000	0.630000	86.000000	52.000000	71.900000
75%	25.000000	0.500000	0.380000	0.630000	0.750000	86.000000	52.000000	71.900000
max	40.000000	18.430000	0.585000	0.750000	0.750000	94.600000	57.200000	79.000000

In [15]:

```
df.nunique()
```

Out[15]:

```
clonesize          6
honeybee           7
bumbles           11
andrena            16
osmia              14
MaxOfUpperTRange   6
MinOfUpperTRange   5
AverageOfUpperTRange 5
MaxOfLowerTRange   6
MinOfLowerTRange   7
AverageOfLowerTRange 5
RainingDays        6
AverageRainingDays 8
fruitset           1525
fruitmass          1515
seeds              2066
yield              776
dtype: int64
```

In [16]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [17]:

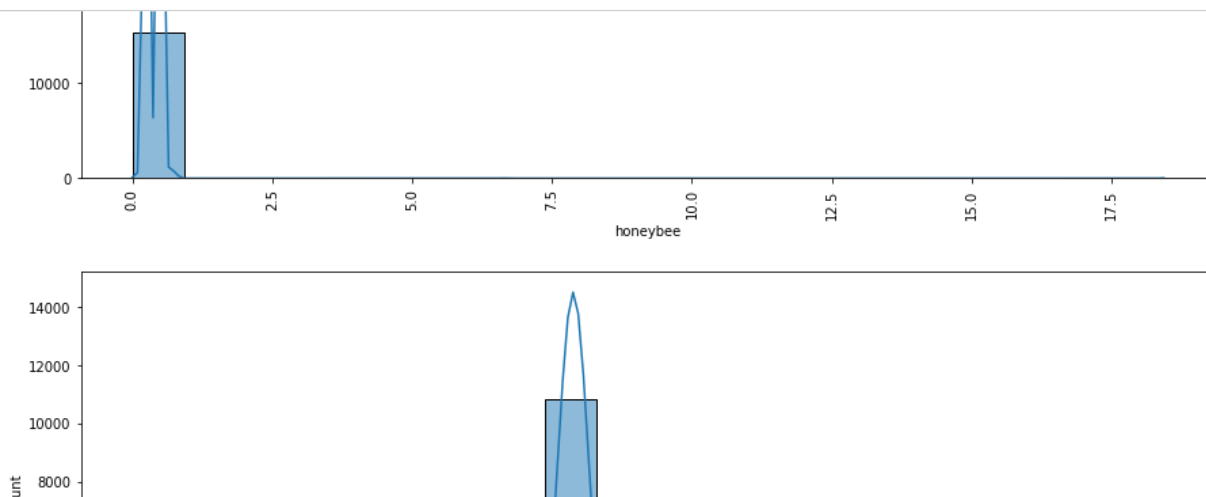
```
import numpy as np
```

In [18]:

```
import warnings
warnings.filterwarnings('ignore')
```

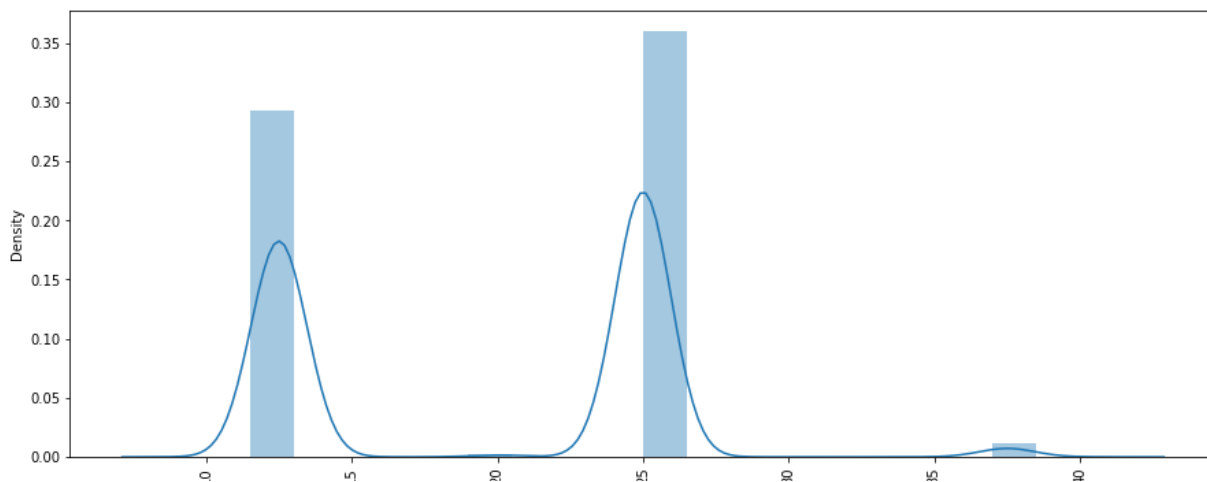
In [19]:

```
for i in df.columns:
    plt.figure(figsize=(15,6))
    sns.histplot(df[i], kde = True, bins = 20, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



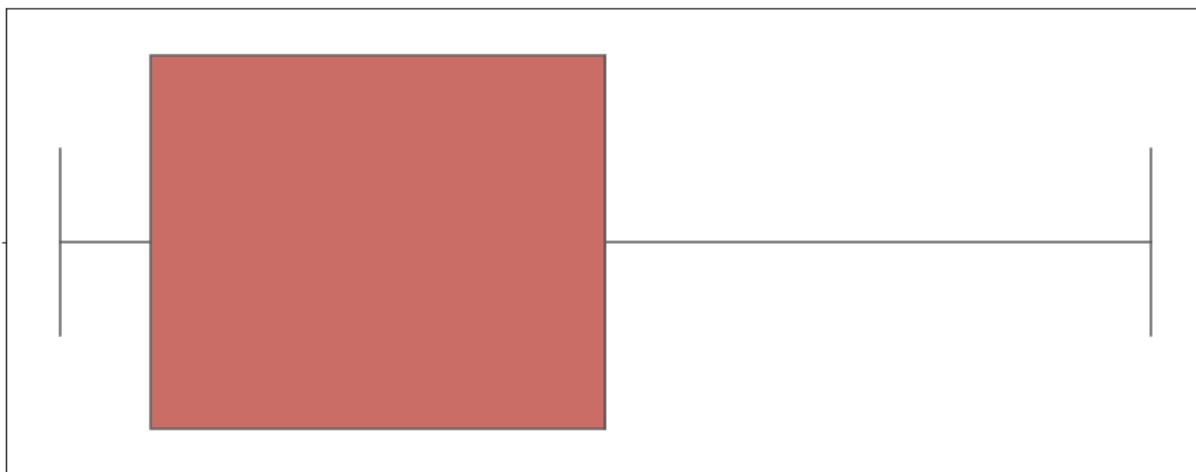
In [20]:

```
for i in df.columns:  
    plt.figure(figsize=(15,6))  
    sns.distplot(df[i], kde = True, bins = 20)  
    plt.xticks(rotation = 90)  
    plt.show()
```



In [21]:

```
for i in df.columns:  
    plt.figure(figsize=(15,6))  
    sns.boxplot(df[i], data = df, palette = 'hls')  
    plt.xticks(rotation = 90)  
    plt.show()
```



In [45]:

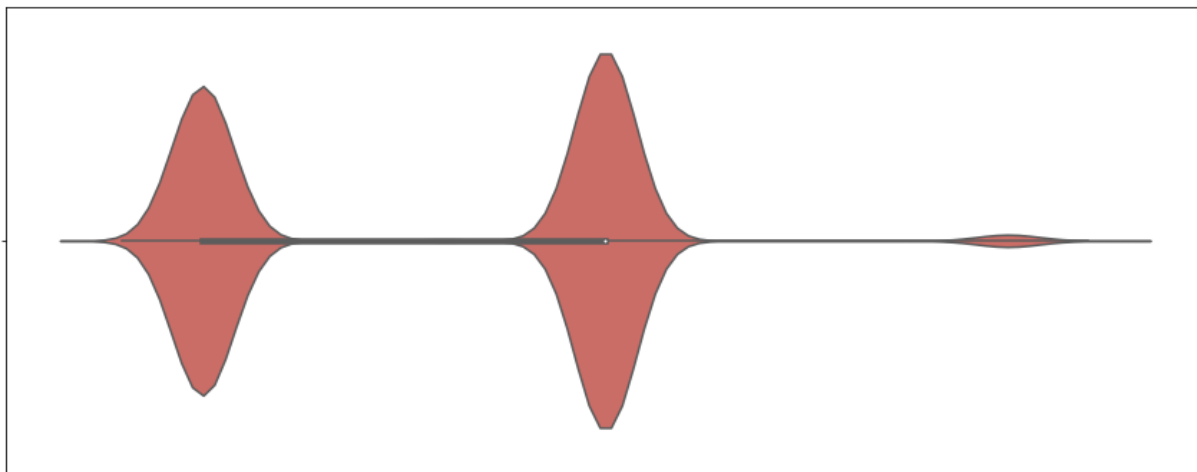
```
for i in df.columns:  
    fig = px.box(df, y=i)  
    fig.update_layout(title=i + ' Distribution Box Plot', xaxis_title=i)  
    fig.show()
```

clonesize Distribution Box Plot



In [22]:

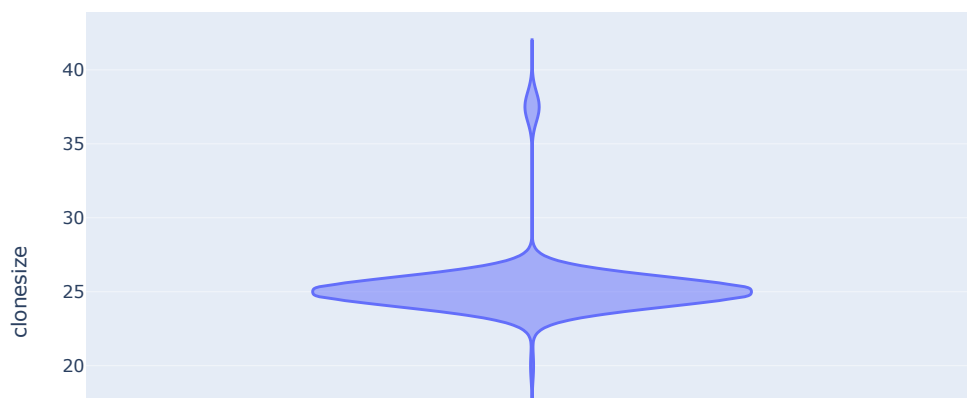
```
for i in df.columns:  
    plt.figure(figsize=(15,6))  
    sns.violinplot(df[i], data = df, palette = 'hls')  
    plt.xticks(rotation = 90)  
    plt.show()
```



In [46]:

```
for i in df.columns:  
    fig = px.violin(df, y=i)  
    fig.update_layout(title=i + ' Distribution Violin Plot', xaxis_title=i)  
    fig.show()
```

clonesize Distribution Violin Plot

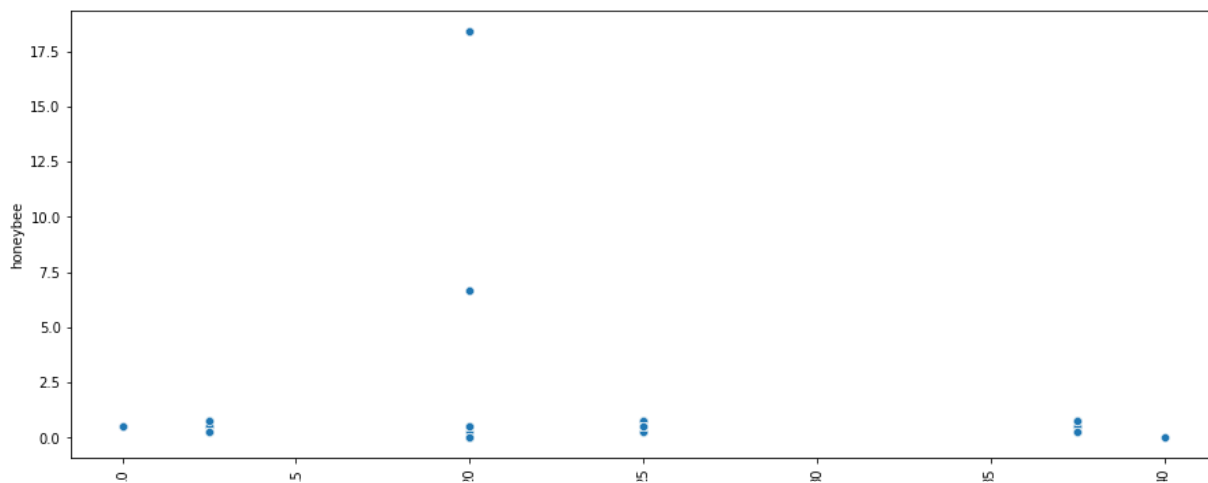


In [24]:

```

for i in df.columns:
    for j in df.columns:
        if i != j:
            plt.figure(figsize=(15,6))
            sns.scatterplot(x=i, y=j, data=df, palette='hls')
            plt.xticks(rotation=90)
            plt.show()

```

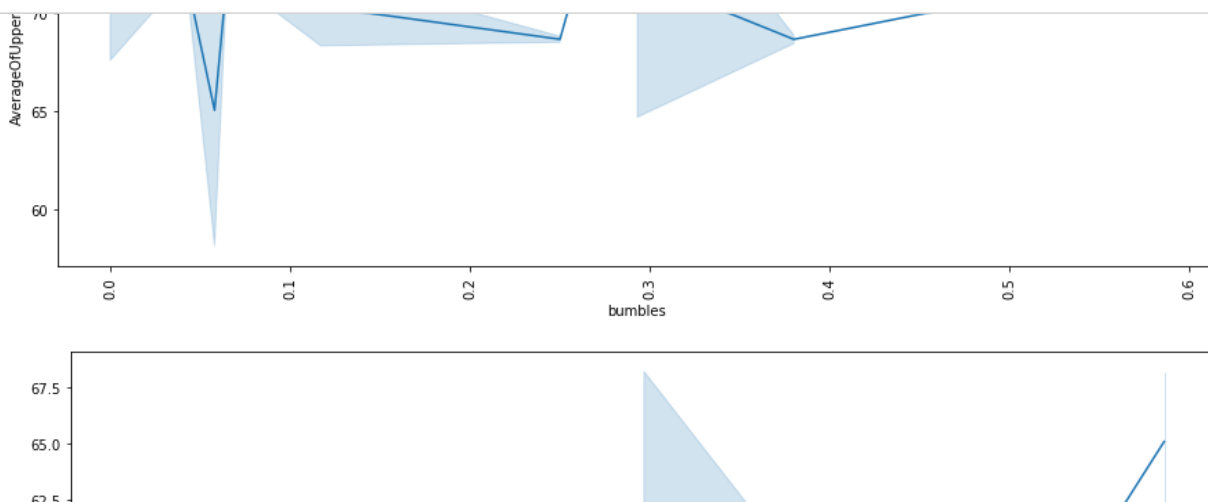


In [25]:

```

for i in df.columns:
    for j in df.columns:
        if i != j:
            plt.figure(figsize=(15,6))
            sns.lineplot(x=i, y=j, data=df, palette='hls')
            plt.xticks(rotation=90)
            plt.show()

```



In [26]:

```

df1 = df[['clonesize', 'honeybee', 'bumbles', 'andrena', 'osmia', 'MaxOfUpperTRange', 'MinOfUpperTRange', 'AverageOfUpperTRange',
          'MaxOfLowerTRange', 'MinOfLowerTRange', 'AverageOfLowerTRange', 'RainingDays', 'AverageRainingDays']]

```

In [27]:

```
for i in df1.columns:
    print(i, ':')
    print(df1[i].unique())
    print('\n')
```

```
clonesize :
[25.  12.5 37.5 20.  10.  40. ]
```

```
honeybee :
[ 0.5    0.25   0.75   0.537  0.    18.43   6.64 ]
```

```
bumbles :
[0.25  0.38  0.117 0.058 0.56  0.065 0.    0.585 0.042 0.293 0.26 ]
```

```
andrena :
[0.75  0.5   0.63  0.38  0.25  0.409 0.707 0.    0.24  0.56  0.101 0.49
 0.234 0.147 0.235 0.229]
```

```
osmia :
[0.5   0.63  0.75  0.25  0.38  0.058 0.117 0.62  0.585 0.    0.021 0.02
 0.078 0.606]
```

```
MaxOfUpperTRange :
[69.7 86.   77.4 94.6 89.   79. ]
```

```
MinOfUpperTRange :
[42.1 52.   46.8 57.2 39. ]
```

```
AverageOfUpperTRange :
[58.2 71.9 64.7 79.   65.6]
```

```
MaxOfLowerTRange :
[50.2 62.   55.8 68.2 66.   52. ]
```

```
MinOfLowerTRange :
[24.3 30.   27.   33.   28.   25.   31. ]
```

```
AverageOfLowerTRange :
[41.2 50.8 45.8 55.9 45.3]
```

```
RainingDays :
[24.   34.    1.   16.    3.77 26.   ]
```

```
AverageRainingDays :
[0.39 0.56 0.1  0.26 0.06 0.25 0.07 0.14]
```

In [28]:

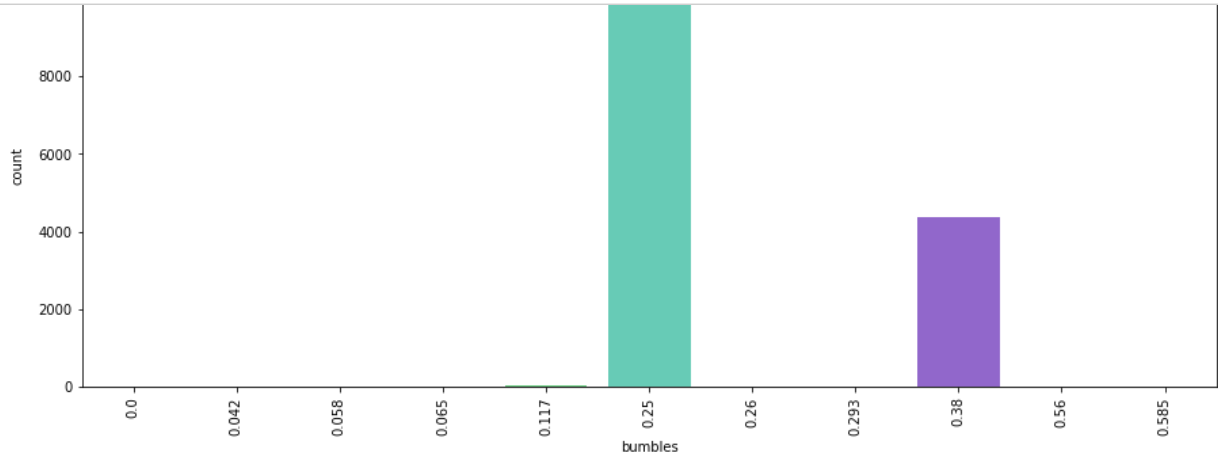
```
for i in df1.columns:
    print(i, ':')
    print(df1[i].value_counts())
    print('\n')
```

clonesize :
25.0 8245
12.5 6717
37.5 265
20.0 56
10.0 4
40.0 2
Name: clonesize, dtype: int64

honeybee :
0.500 7832
0.250 7285
0.750 110
0.537 38
0.000 16
18.430 5
6.640 3
Name: honeybee, dtype: int64

In [30]:

```
for i in df1.columns:
    print(i, ':')
    plt.figure(figsize=(15,6))
    sns.countplot(x = df1[i], data = df1, palette = 'hls')
    plt.xticks(rotation = 90)
    plt.show()
```



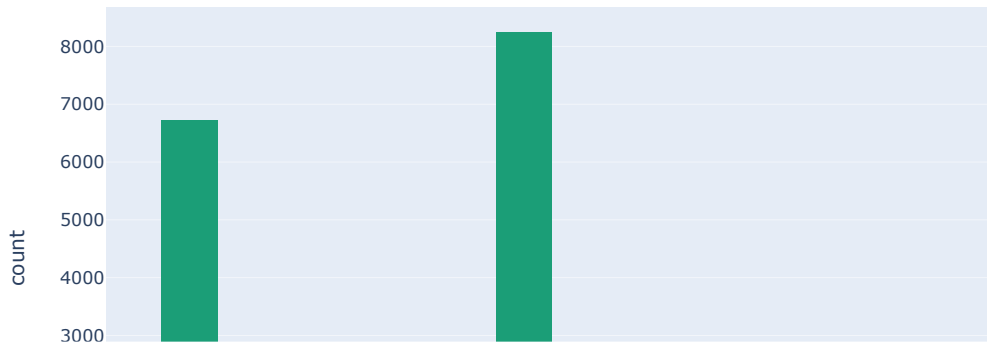
address :

In [34]:

```
import plotly.express as px

for i in df1.columns:
    fig = px.histogram(df1, x=i, title=i, color_discrete_sequence=px.colors.qualitative.Dark2)
    fig.update_xaxes(tickangle=90)
    fig.show()
```

clonesize



In [31]:

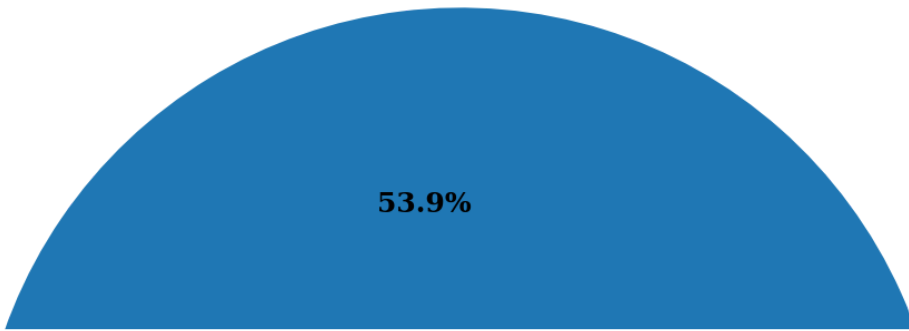
```
for i in df1.columns:
    plt.figure(figsize=(30,20))
    plt.pie(df1[i].value_counts(), labels=df1[i].value_counts().index, autopct='%1.1f%%', textprops={ 'fontsize': 25,
                                                                                                     'color': 'black',
                                                                                                     'weight': 'bold',
                                                                                                     'family': 'serif' })

    hfont = {'fontname': 'serif', 'weight': 'bold'}
    plt.title(i, size=30, **hfont)
    plt.show()
```

clonesize

25.0

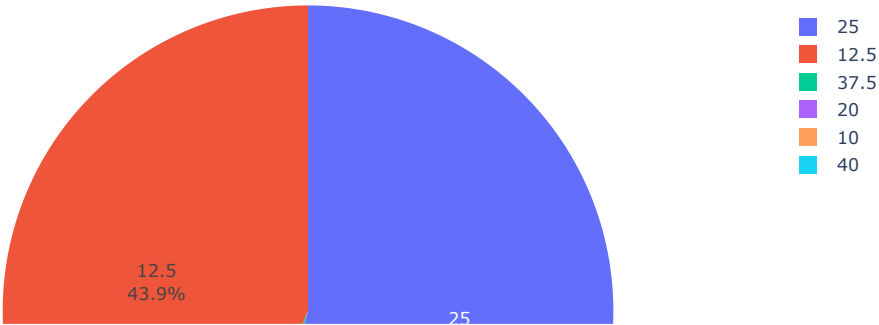
53.9%



In [32]:

```
for i in df1.columns:
    fig = px.pie(df1[i].value_counts(), values=df1[i].value_counts(), names=df1[i].value_counts().index,
                 title=i, width=800, height=600)
    fig.update_traces(textposition='inside', textinfo='percent+label')
    fig.show()
```

clonesize



In [35]:

```
df_corr = df.corr()
```

In [36]:

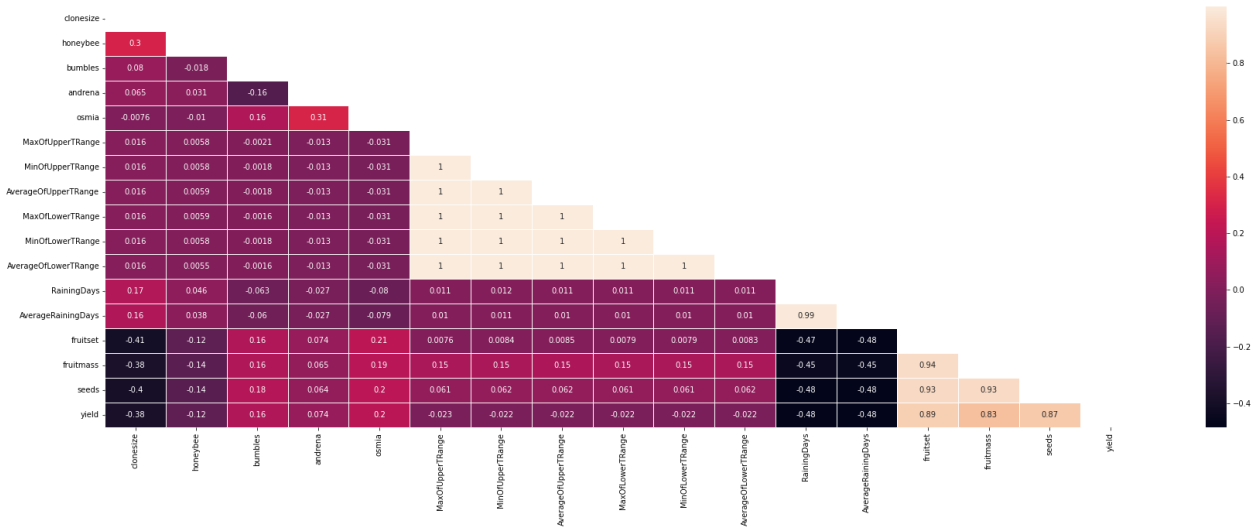
```
df_corr
```

Out[36]:

	clonesize	honeybee	bumbles	andrena	osmia	MaxOfUpperTRange	MinOfUpperTRange	AverageOfUpperTRange
clonesize	1.000000	0.304130	0.080433	0.065131	-0.007607	0.016159	0.015838	0.0160
honeybee	0.304130	1.000000	-0.017937	0.030671	-0.010394	0.005840	0.005755	0.0058
bumbles	0.080433	-0.017937	1.000000	-0.164962	0.158001	-0.002104	-0.001813	-0.0017
andrena	0.065131	0.030671	-0.164962	1.000000	0.309556	-0.013061	-0.012928	-0.0129
osmia	-0.007607	-0.010394	0.158001	0.309556	1.000000	-0.031391	-0.030819	-0.0314
MaxOfUpperTRange	0.016159	0.005840	-0.002104	-0.013061	-0.031391	1.000000	0.998599	0.9998
MinOfUpperTRange	0.015838	0.005755	-0.001813	-0.012928	-0.030819	0.998599	1.000000	0.9990
AverageOfUpperTRange	0.016057	0.005892	-0.001769	-0.012993	-0.031415	0.999806	0.999004	1.0000
MaxOfLowerTRange	0.016343	0.005942	-0.001613	-0.012924	-0.031398	0.999503	0.998199	0.9994
MinOfLowerTRange	0.016026	0.005809	-0.001804	-0.013035	-0.031486	0.999829	0.998953	0.9999
AverageOfLowerTRange	0.015987	0.005485	-0.001644	-0.013071	-0.031337	0.999772	0.999040	0.9999
RainingDays	0.165770	0.046494	-0.063294	-0.026572	-0.079874	0.011322	0.011727	0.0112
AverageRainingDays	0.164823	0.037532	-0.060232	-0.027193	-0.078720	0.010352	0.010767	0.0102
fruitset	-0.406793	-0.120492	0.160447	0.073669	0.209495	0.007580	0.008409	0.0085
fruitmass	-0.377688	-0.135310	0.163987	0.064722	0.192210	0.146237	0.147203	0.1476
seeds	-0.396898	-0.139261	0.177022	0.063504	0.200597	0.060963	0.061812	0.0620
yield	-0.382619	-0.118001	0.161145	0.073969	0.198264	-0.022517	-0.021929	-0.0219

In [38]:

```
plt.figure(figsize=(30, 10))
matrix = np.triu(df_corr)
sns.heatmap(df_corr, annot=True, linewidth=.8, mask=matrix, cmap="rocket");
plt.show()
```



In [41]:

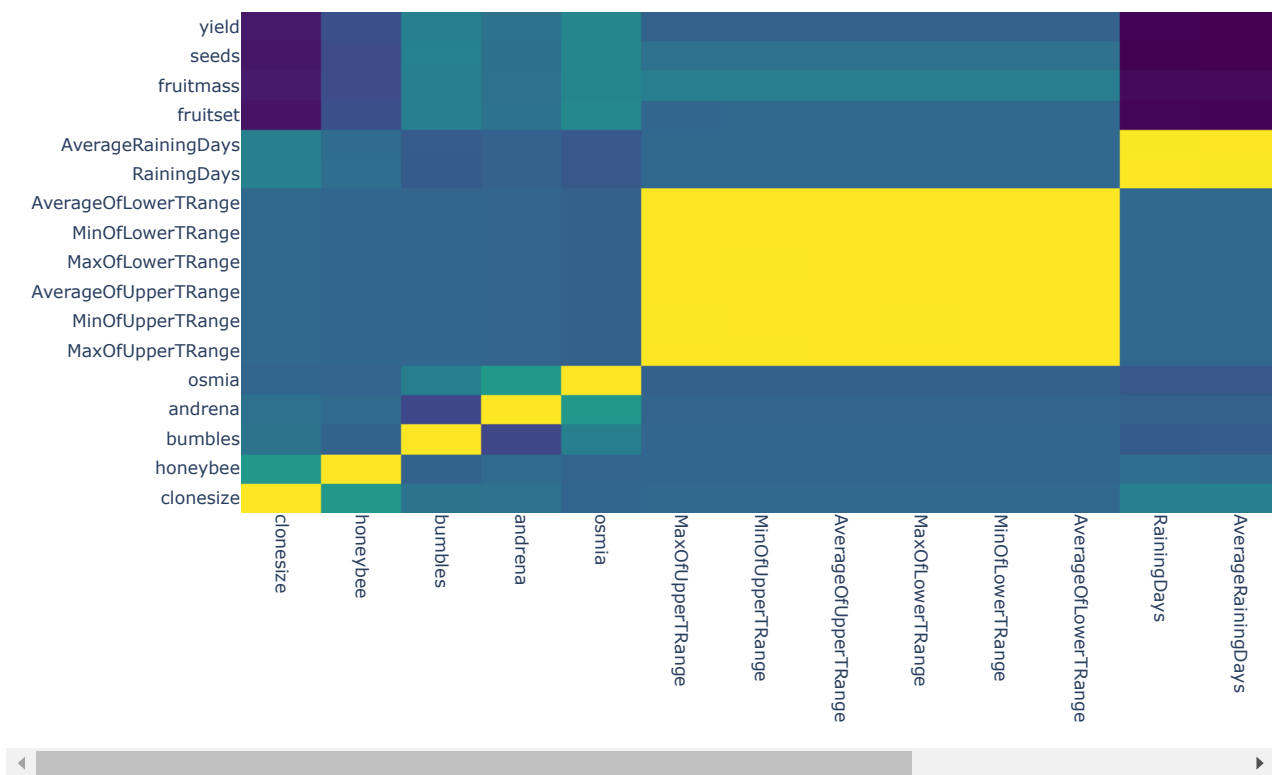
```
import plotly.graph_objects as go

fig = go.Figure(data=go.Heatmap(
    z=df_corr,
    x=df_corr.columns,
    y=df_corr.index,
    colorscale='Viridis',
    hoverongaps=False))

fig.update_layout(
    title='Correlation Matrix',
    xaxis=dict(tickangle=90),
    height=600,
    width=1200
)

fig.show()
```

Correlation Matrix



In [47]:

```
X = df.drop('yield',axis=1)
Y = df['yield']
```

In [48]:

```
from sklearn.model_selection import train_test_split
```

In [49]:

```
from sklearn.linear_model import LinearRegression
```

In [50]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [51]:

```
regressor = LinearRegression()  
regressor.fit(X_train, y_train)
```

Out[51]:

```
LinearRegression  
LinearRegression()
```

In [52]:

```
y_pred = regressor.predict(X_test)  
mse = ((y_pred - y_test) ** 2).mean()  
print("Mean squared error:", mse)
```

Mean squared error: 333169.7877732821

In [53]:

```
from sklearn.metrics import r2_score
```

In [54]:

```
r2 = r2_score(y_test, y_pred)  
print('R-squared score:', r2)
```

R-squared score: 0.810476696897496

In [55]:

```
from sklearn.tree import DecisionTreeRegressor
```

In [56]:

```
tree_reg = DecisionTreeRegressor(random_state=42)
```

In [57]:

```
tree_reg.fit(X_train, y_train)
```

Out[57]:

```
DecisionTreeRegressor  
DecisionTreeRegressor(random_state=42)
```

In [58]:

```
y_pred = tree_reg.predict(X_test)
```

In [59]:

```
r2_score = r2_score(y_test, y_pred)
```

In [60]:

```
print("R-squared score of DecisionTreeRegressor:", r2_score)
```

R-squared score of DecisionTreeRegressor: 0.6389661625506569

In [61]:

```
from xgboost import XGBRegressor
```

In [62]:

```
model = XGBRegressor()
```

In [63]:

```
model.fit(X_train, y_train)
```

Out[63]:

```
XGBRegressor
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=0,
              num_parallel_tree=1, predictor='auto', random_state=0, reg_alpha=0,
              reg_lambda=1, ...)
```

In [64]:

```
y_pred = model.predict(X_test)
```

In [66]:

```
from sklearn.metrics import r2_score
```

In [67]:

```
r2 = r2_score(y_test, y_pred)
print("R2 score:", r2)
```

```
R2 score: 0.8144045120706109
```