In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator
from scipy.stats import bernoulli
from collections import Counter
from matplotlib import colors
```

In [ ]:
```
class DummyBinaryClassifier(BaseEstimator):
    def __init__(self,method="bernoulli",p=0.5):
        self.p = 0.5 if p < 0.0 or p > 1.0 else p
        self.method = method if method in ["bernoulli","uniform_random","normal"] e
    def fit(self,X,y=None):
        return
    def predict(self,X):
        if self.method == "normal":
            return (0.5 + np.random.randn(len(X))) < self.p
        elif self.method == "bernoulli":
            return np.bool_(bernoulli.rvs(self.p, size=len(X)))
        else:
            return np.random.rand(len(X)) <= self.p
```

In [ ]:
```
def compute_priors(y):
    c = Counter(y)
    prop = {i[0]:i[1]/len(y) for i in c.items()}
    if True not in prop:
        prop[True] = 0.0
    if False not in prop:
        prop[False] = 0.0
    #print(prop)
    return prop
```

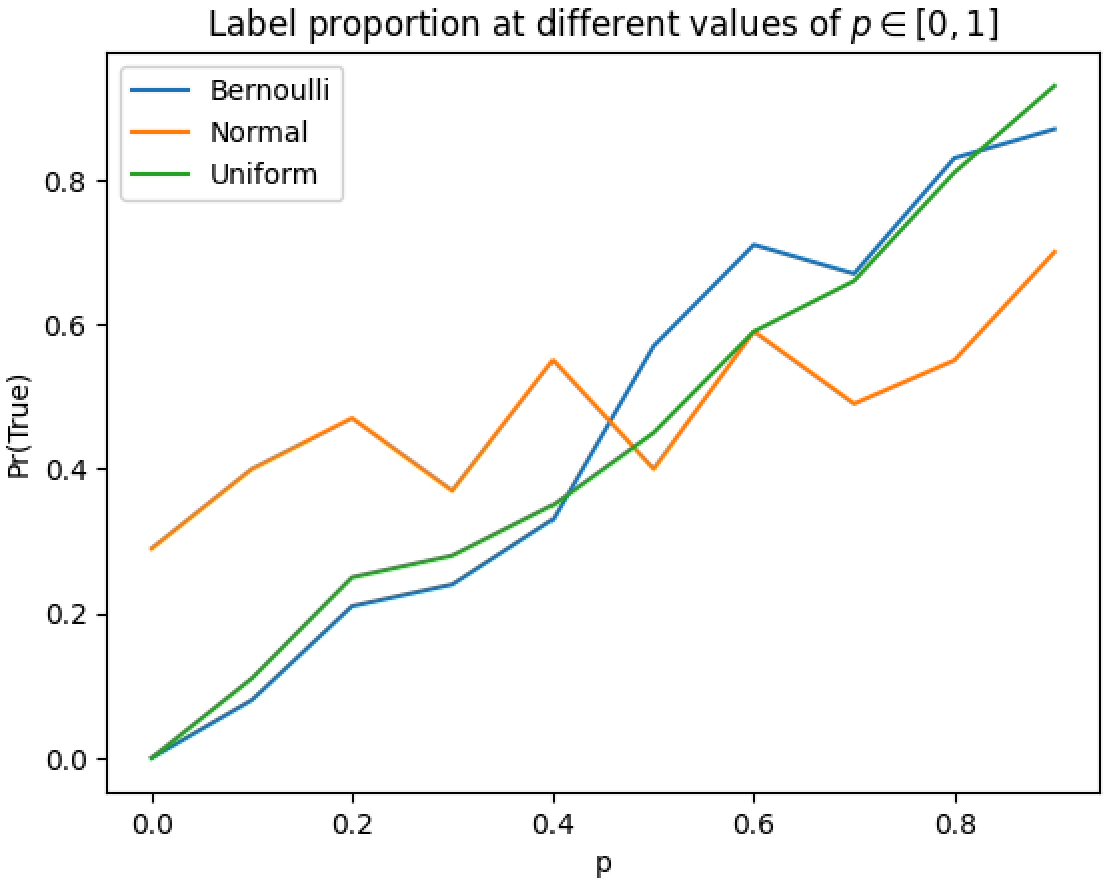
In [ ]:
```
# Input: 100 random samples
X = np.random.rand(100)
```

In [ ]:
```
p_values = np.arange(0.0,1.0,0.1)
```

In [ ]:
```
bernoulli_priors=[]
normal_priors=[]
uniform_priors=[]

for p in p_values:
    # To compute the Bernouli priors
    y_ber = DummyBinaryClassifier("bernoulli",p).predict(X)
    prop_ber = compute_priors(y_ber)
    bernoulli_priors.append(prop_ber[True])

    # To compute the Normal priors
    y_nor = DummyBinaryClassifier("normal",p).predict(X)
    prop_nor = compute_priors(y_nor)
    normal_priors.append(prop_nor[True])

    # To compute the Uniform_random priors
    y_uni = DummyBinaryClassifier("uniform_random",p).predict(X)
    prop_uni = compute_priors(y_uni)
    uniform_priors.append(prop_uni[True])
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(p_values, bernoulli_priors)
plt.plot(p_values, normal_priors)
```

```
plt.plot(p_values, uniform_priors)
plt.xlabel('p')
plt.ylabel('Pr(True)')
plt.title('Label proportion at different values of $p\in[0,1]$')
plt.legend(['Bernoulli','Normal','Uniform'], loc='upper left')
plt.show()
```



Task2 : IRIS dataset

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve, roc
```

In [ ]:
```
iris = load_iris()
X,y = iris.data,iris.target
```

To check for the no of elements of three classes of IRIS dataset.

In [ ]:
```
arr=pd.Series(iris.target)
arr.value_counts()
```

Out[ ]:
```
0    50
1    50
2    50
dtype: int64
```

Since each of the classes have equal elements, Let the first class('setosa') be True and the other two classes('versicolor', 'virginica') be False to convert the dataset into a Binary IRIS dataset.

In [ ]:
```
y[y==0] = 1
y[50:] = 0
y
```

Out[ ]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [ ]:
```
arr=pd.Series(y)
arr.value_counts() # The dataset now has two labels 0 and 1.
```

Out[ ]:
```
0    100
1     50
dtype: int64
```

Label prior of the binary IRIS dataset

In [ ]:
```
iris_prior_pred = DummyBinaryClassifier('bernoulli',0.3).predict(X)
```

In [ ]:
```
#Task 2.1
prop_iris = compute_priors(iris_prior_pred)
prop_iris
# The label prior depends on the 'p' with which we instantite the DummyBinaryClassi
```

Out[ ]:
```
{False: 0.7066666666666667, True: 0.29333333333333333}
```

In [ ]:
```
p_values = np.arange(0.0,1.1,0.1)
p_values
```

Out[ ]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])
```

In [ ]:
```
#Task 2.2

iris_prop = []
precision = []
recall = []
F1 =[]
fpr =[]
tpr =[]
auprc = []
auc_roc = []

for p in p_values:
    # The label prior for the IRIS data
    iris_pred = DummyBinaryClassifier('bernoulli',p).predict(X)
    prop_iris = compute_priors(iris_pred)
    iris_prop.append(prop_iris[True])

    # Precision, Recall and F1 for each value of p
    precision.append(precision_score(y,iris_pred))
    recall.append(recall_score(y,iris_pred))
    F1.append(f1_score(y,iris_pred))

    # True positive rate and false positive rate foe each value of p
    fpr_p, tpr_p,threshold_p = roc_curve(y,iris_pred)

    fpr.append(fpr_p)
    tpr.append(tpr_p)

    # AUPRC and AuRoC for each value of P
    auc_roc.append(roc_auc_score(y,iris_pred))
```

```
    auprc.append(average_precision_score(y,iris_pred))
```

```
C:\Users\MANOJ\.conda\envs\ProjectAutoKeras\lib\site-packages\sklearn\metrics\_cla
ssification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being se
t to 0.0 due to no predicted samples. Use `zero_division` parameter to control thi
s behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

In [ ]:
```
iris_prop # The label prior for the IRIS data
```

Out[ ]:
```
[0.0,
 0.09333333333333334,
 0.18,
 0.34,
 0.3333333333333333,
 0.47333333333333333,
 0.5666666666666667,
 0.7133333333333334,
 0.8,
 0.88,
 1.0]
```

In [ ]:
```
recall # Recall for each value of p
```

Out[ ]:
```
[0.0, 0.1, 0.22, 0.34, 0.34, 0.56, 0.48, 0.78, 0.82, 0.86, 1.0]
```

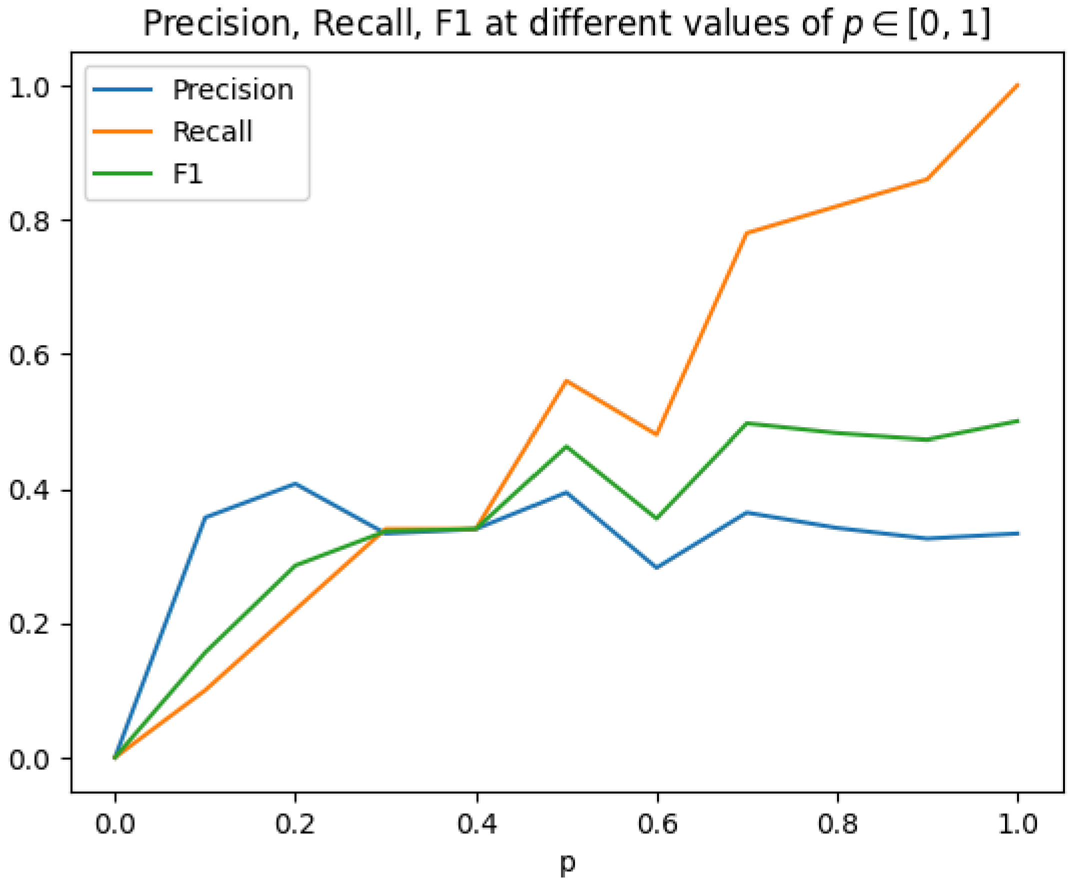
In [ ]:
```
precision # Precision for each value of p
```

Out[ ]:
```
[0.0,
 0.35714285714285715,
 0.4074074074074074,
 0.3333333333333333,
 0.34,
 0.39436619718309857,
 0.2823529411764706,
 0.3644859813084112,
 0.3416666666666667,
 0.32575757575757575,
 0.3333333333333333]
```

In [ ]:
```
F1 # F1 for each value of p
```

Out[ ]:
```
[0.0,
 0.15625,
 0.2857142857142857,
 0.33663366336633666,
 0.34,
 0.4628099173553719,
 0.35555555555555557,
 0.4968152866242038,
 0.4823529411764706,
 0.4725274725274725,
 0.5]
```

In [ ]:
```
plt.plot(p_values, precision)
plt.plot(p_values, recall)
plt.plot(p_values, F1)
plt.xlabel('p')
#plt.ylabel('Pr(True)')
plt.title('Precision, Recall, F1 at different values of $p\in[0,1]$')
plt.legend(['Precision','Recall','F1'], loc='upper left')
plt.show()
```

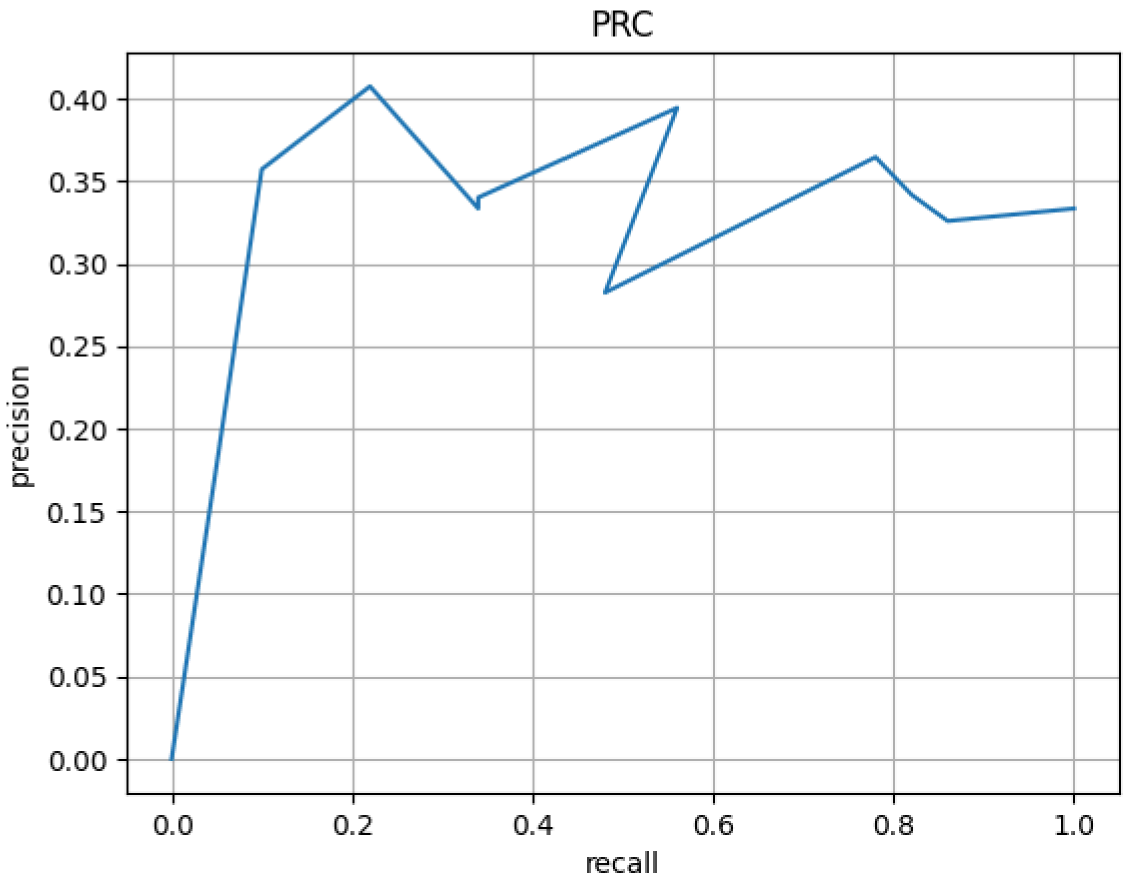


In [ ]:

```
#Task 2.3
plt.plot(recall, precision)
plt.xlabel('recall')
plt.ylabel('precision')
plt.grid(True)
plt.title('PRC')
```

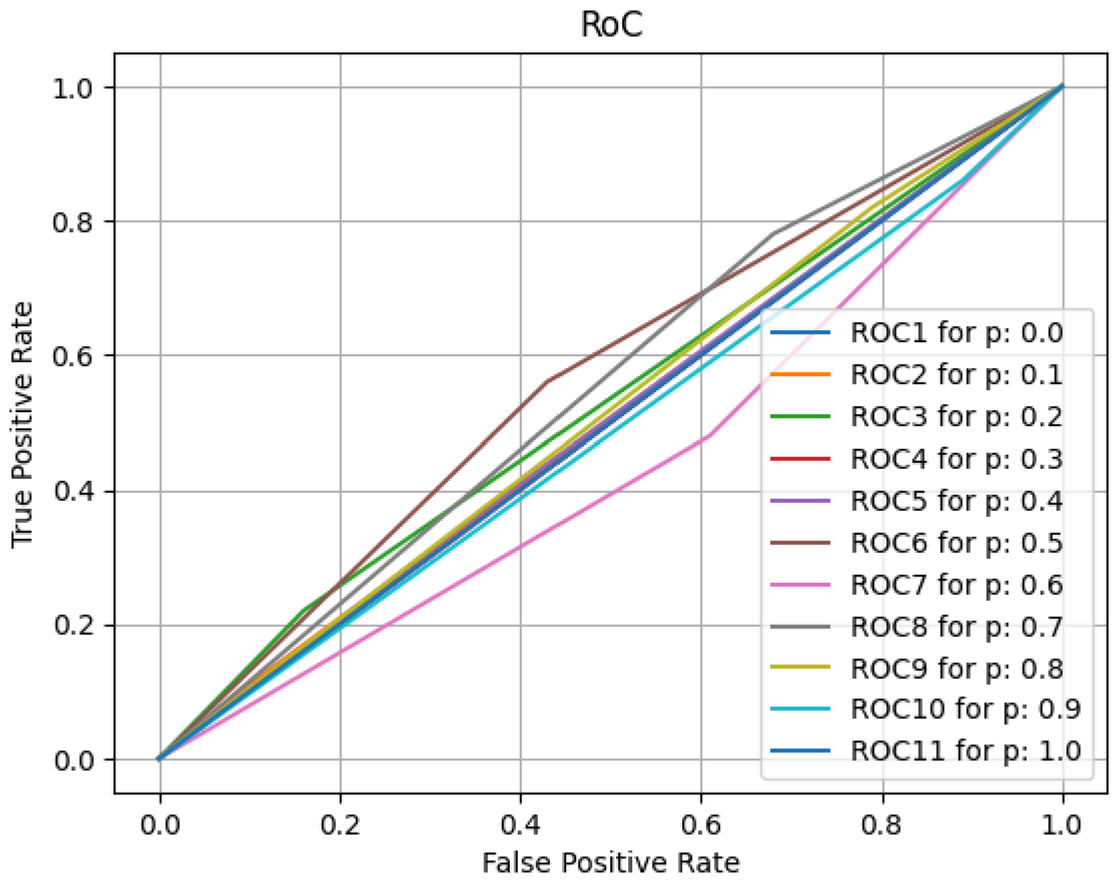
Out[ ]:

```
Text(0.5, 1.0, 'PRC')
```



In [ ]:
```
#Task 2.4
for i in range(len(fpr)):
    plt.plot(fpr[i], tpr[i],label=f'ROC{i+1} for p: {round(0.1*i,1)}')

plt.legend()
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid(True)
plt.title('RoC')
```

Out[ ]:
```
Text(0.5, 1.0, 'RoC')
```



In [ ]:
```
#Task 2.5
print("The Area under RoC curve for diff values of p:",auc_roc)
print()
print("The Area under PRC for diff values of p:",auprc)
```

```
The Area under RoC curve for diff values of p: [0.5, 0.505, 0.5299999999999999, 0.5, 0.505, 0.5650000000000001, 0.43500000000000005, 0.55, 0.515, 0.485, 0.5]

The Area under PRC for diff values of p: [0.3333333333333333, 0.3357142857142857, 0.3496296296296296, 0.3333333333333333, 0.3356, 0.3675117370892018, 0.30886274509803924, 0.3576323987538941, 0.3401666666666667, 0.32681818181818184, 0.3333333333333333]
```

Task 3: Visualization of Decision Boundaries

For visualization, using the input features of iris dataset to create the 2D grid doesn't make sense as the dummy classifier doesn't depend on the data and the classifer returns a random value of 0 and 1's based on p.

Let us consider the feature Sepal Width and Petal Width in order to create the grid

In [ ]:
```
df = pd.DataFrame(X,columns=iris.feature_names)
X_df = df[[df.columns[1],df.columns[3]]] # Sepal Width and Petal Width features
```

In [ ]:
```
x1 = X_df[X_df.columns[0]]
x2 = X_df[X_df.columns[1]]
```

In [ ]:
```
x1.head()
```

Out[ ]:
```
0    3.5
1    3.0
2    3.2
3    3.1
4    3.6
Name: sepal width (cm), dtype: float64
```

In [ ]:
```
x2.head()
```

Out[ ]:
```
0    0.2
1    0.2
2    0.2
3    0.2
4    0.2
Name: petal width (cm), dtype: float64
```

In [ ]:
```
# generating 100 points within min-max range
grid_x1 = np.linspace(x1.min(), x1.max(), 100)
grid_x2 = np.linspace(x2.min(), x2.max(), 100)
# creates a rectangular grid out of two given one-dimensional arrays
x1v, x2v = np.meshgrid(grid_x1, grid_x2)
```

In [ ]:
```
# Creating a dataframe for the synthetic mesh data and to estimate the predictions
print("Shape of x1v: ",x1v.shape)
print("Shape of x2v: ",x2v.shape)

# x1v and x2v are 2d arrays and need to be converted into a 1d array to be fed intc
# functions such as flatten() and ravel() are used.

data = pd.DataFrame(data=np.column_stack((x1v.flatten(), x2v.flatten())), columns=>
# np.column_stack((x1v.flatten(), x2v.flatten())) --> Two columns of 10000 rows eac
print("Shape of data fed into model: ",data.shape)
```

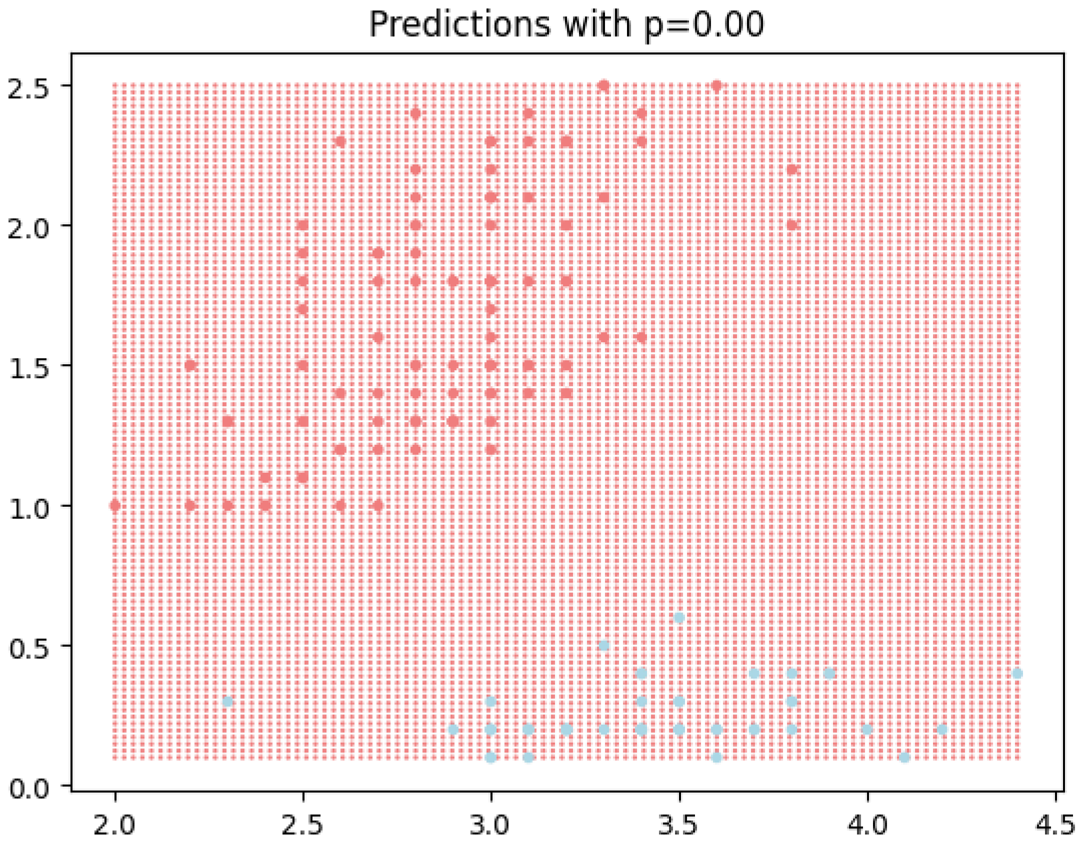
```
Shape of x1v:  (100, 100)
Shape of x2v:  (100, 100)
Shape of data fed into model:  (10000, 2)
```

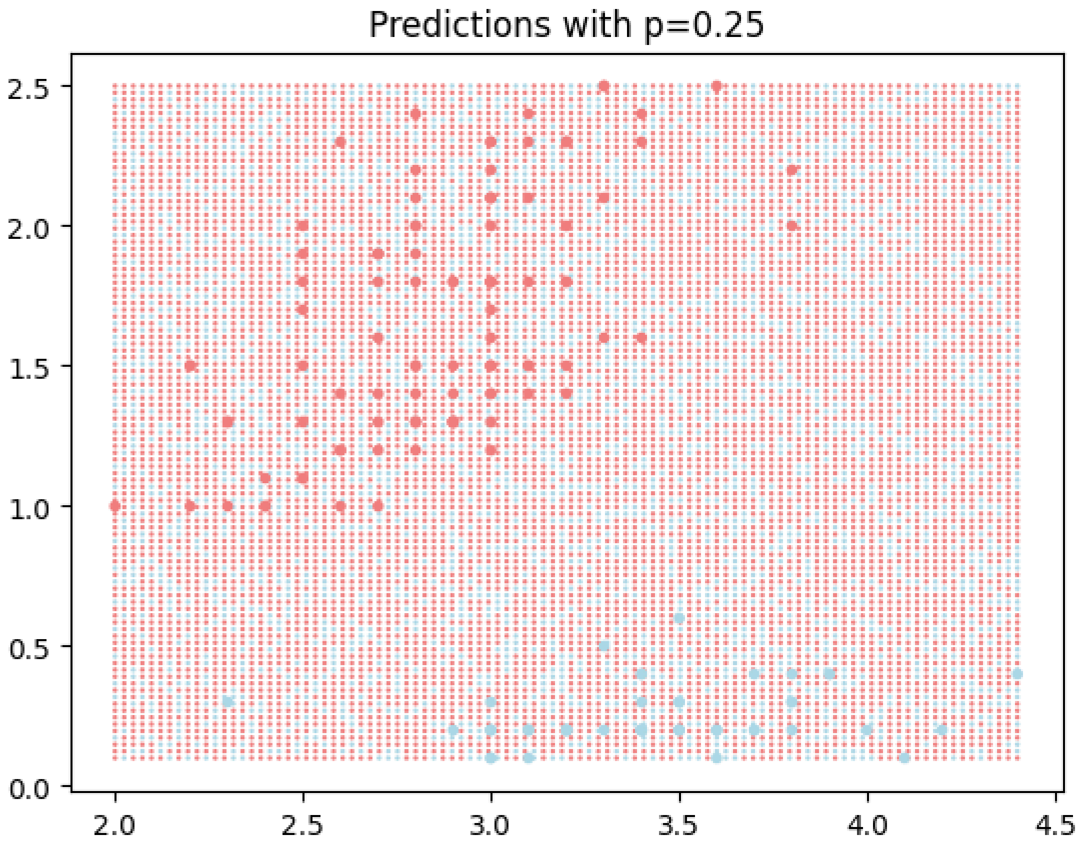
In [ ]:
```
p_value = np.arange(0,1.25,0.25)
p_value
```

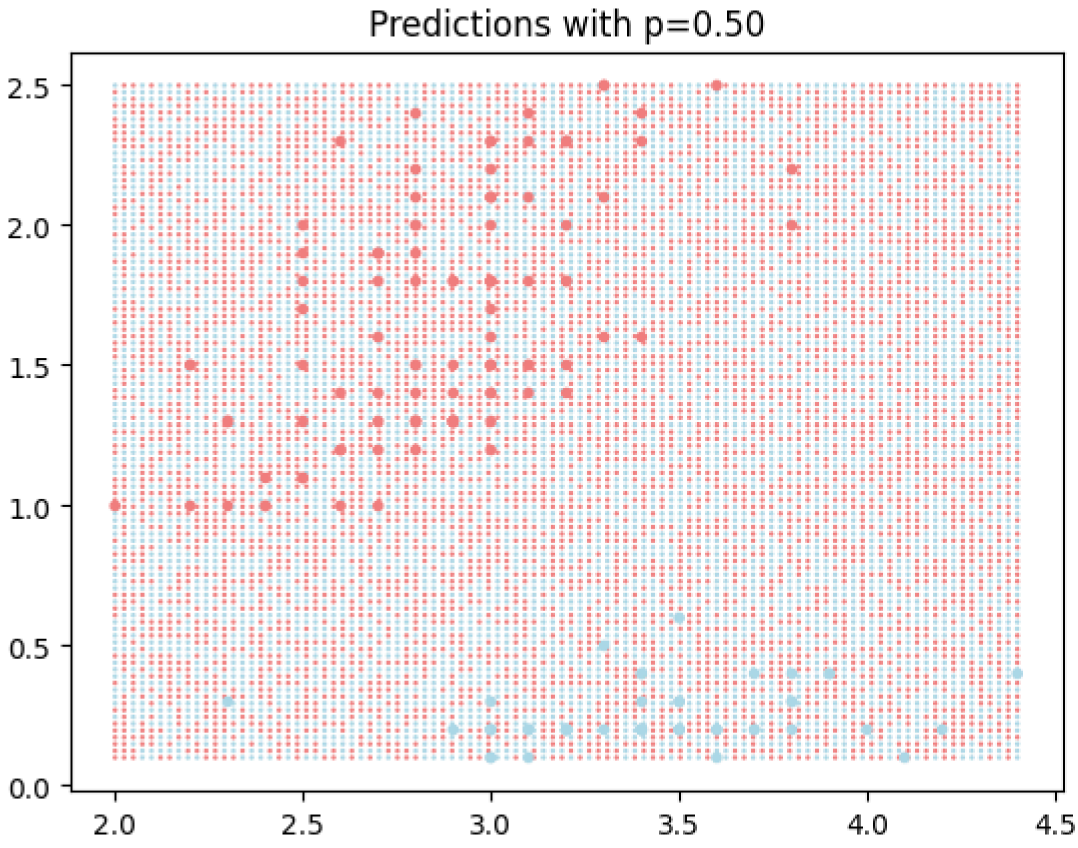
Out[ ]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

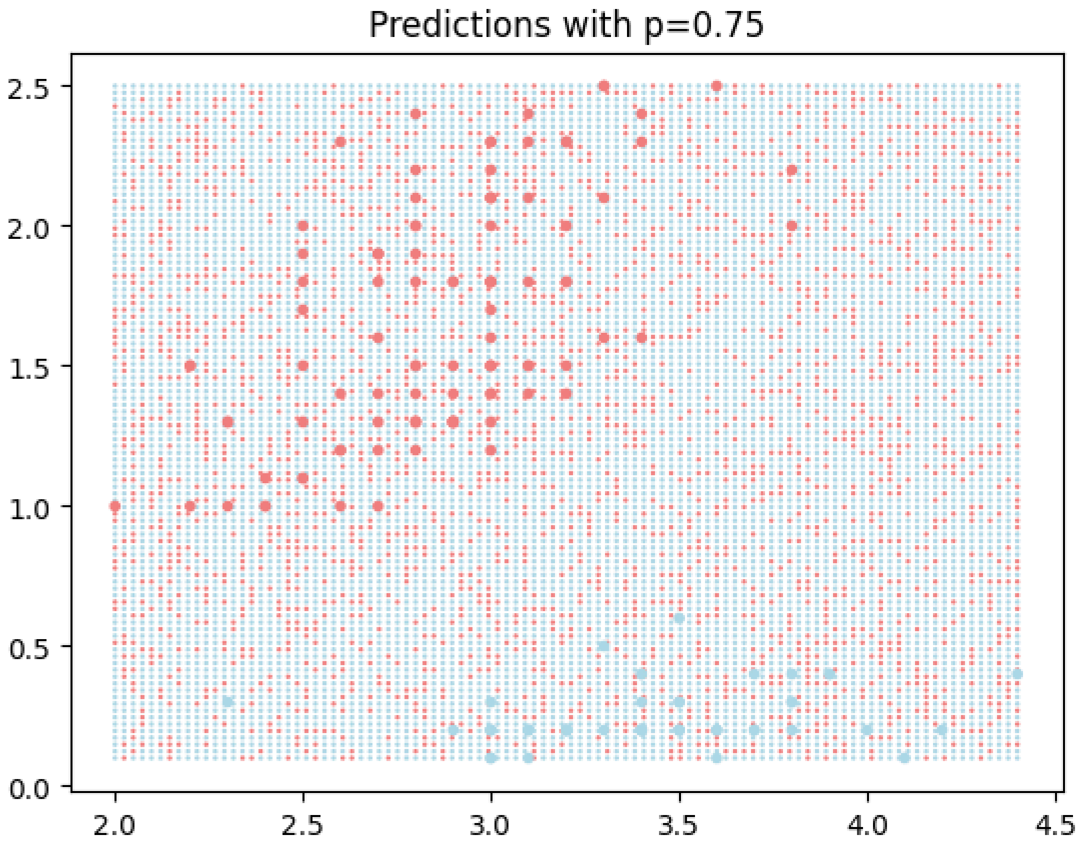
Bernoulli method

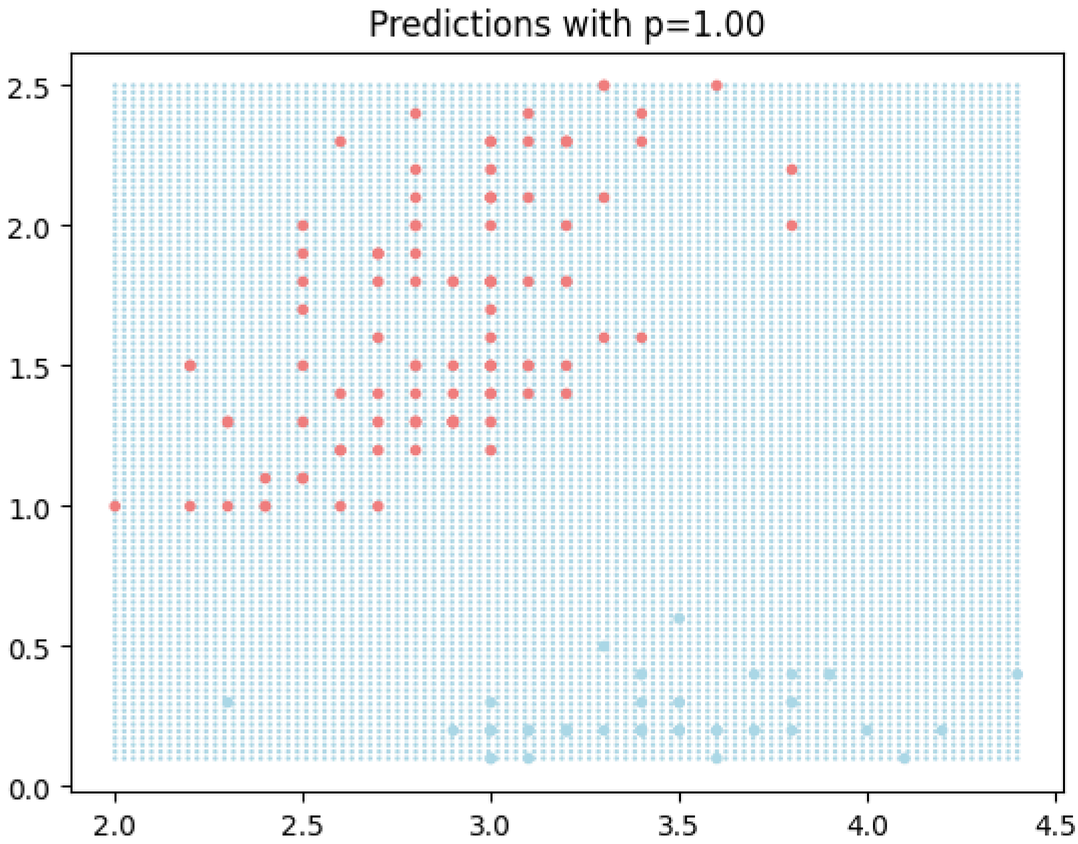
In [ ]:
```
for p in p_value:

    y_pred = DummyBinaryClassifier("bernoulli",p).predict(data)
    y_reshape = y_pred.reshape(x1v.shape)
    plt.figure()
    color_map = colors.ListedColormap(['lightcoral', 'lightblue'])
    plt.scatter(x1v, x2v, marker='.', s=2, c=y_pred, cmap=color_map,vmin=0,vmax=1)
    plt.scatter(x1, x2, marker='.', c=y, cmap=color_map)
    plt.title(f'Predictions with p={p:.2f}')
    plt.show()
```


Predictions with p=0.00
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


Predictions with p=0.25
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

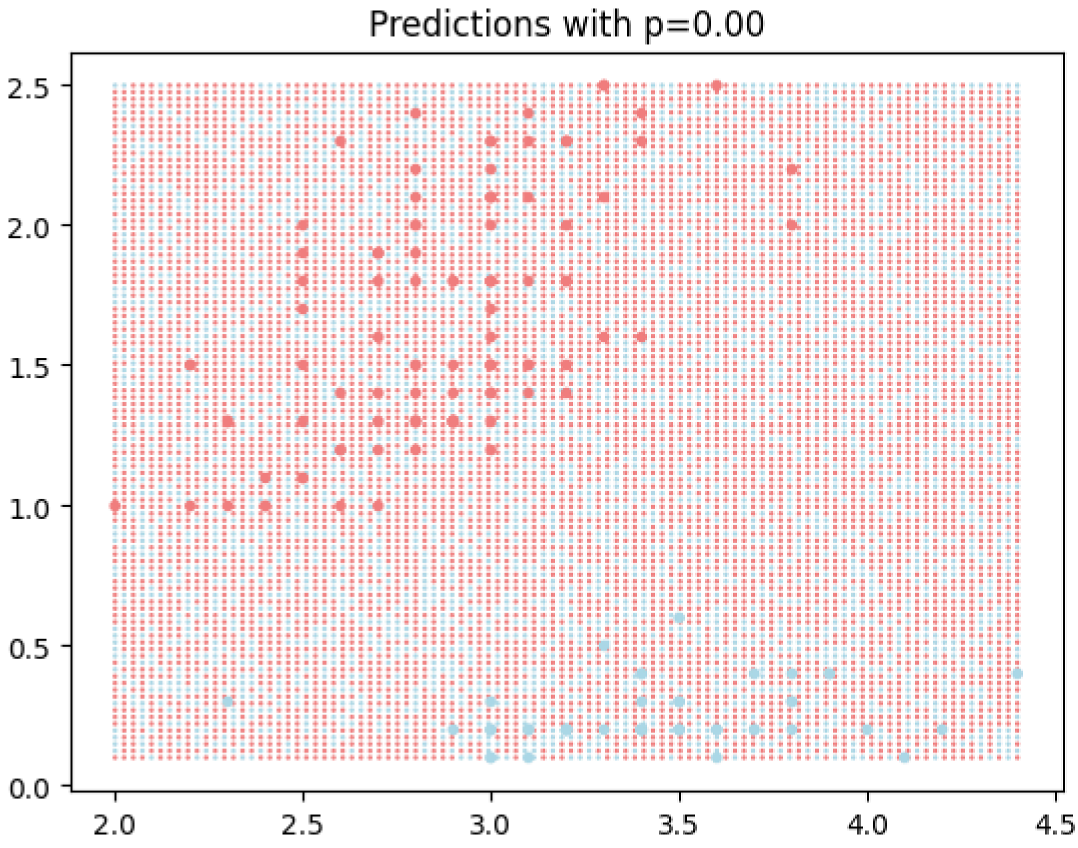

Predictions with p=0.50
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

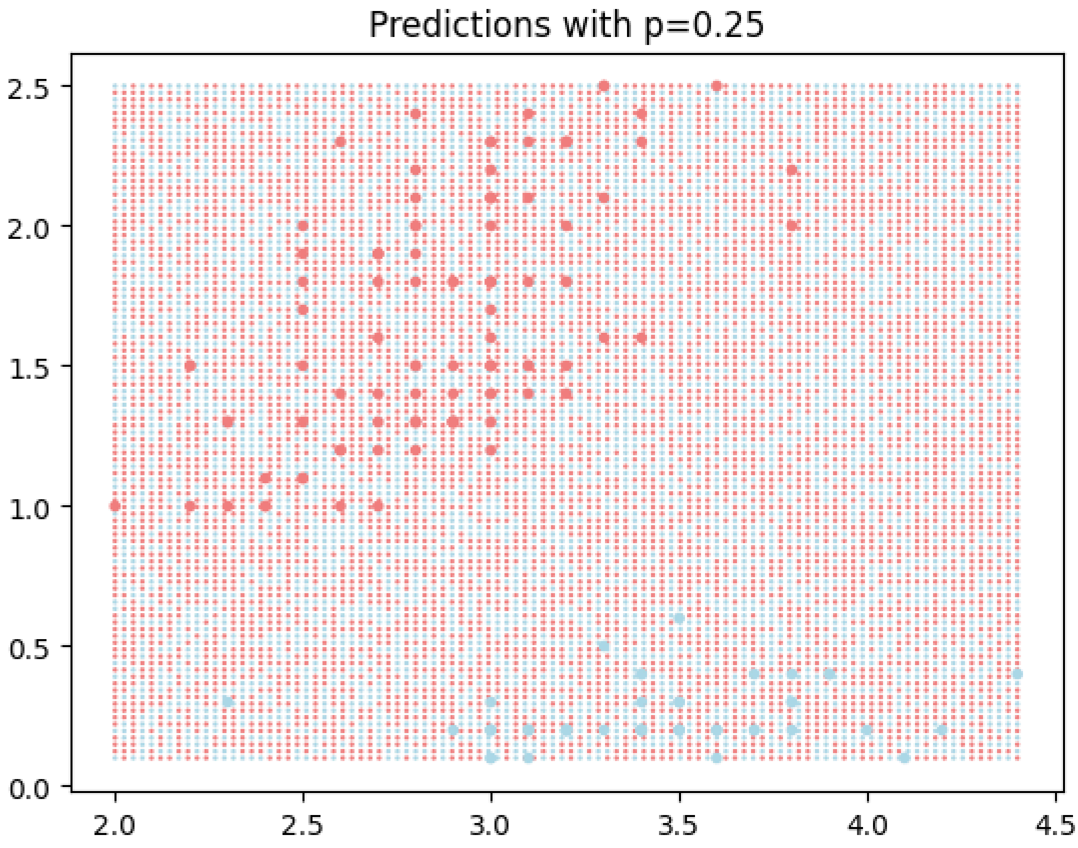

Predictions with p=0.75
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

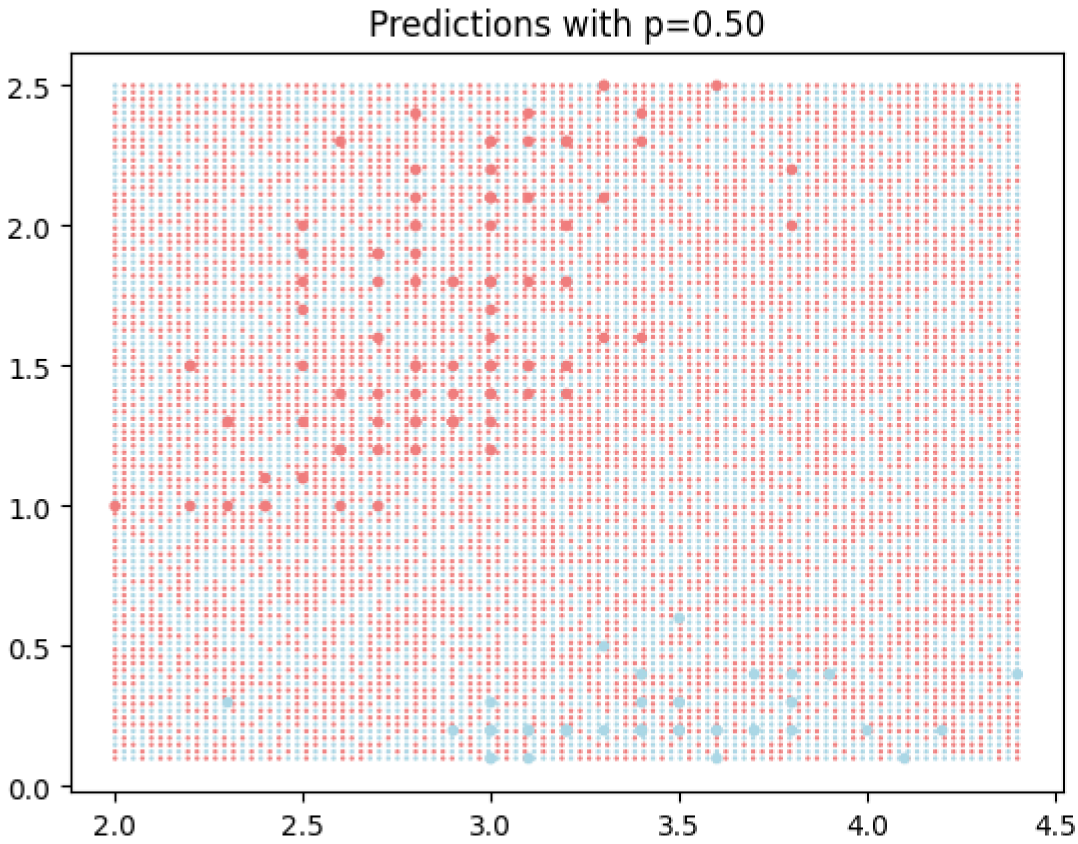


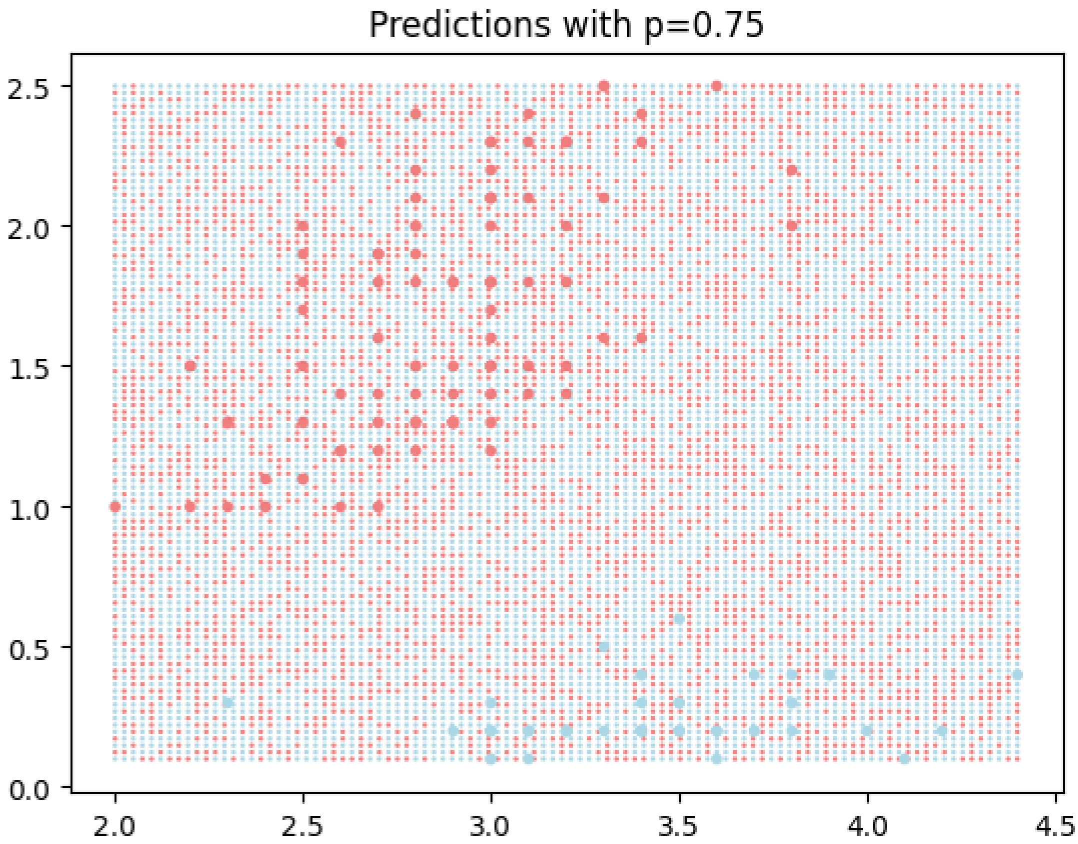
Normal method

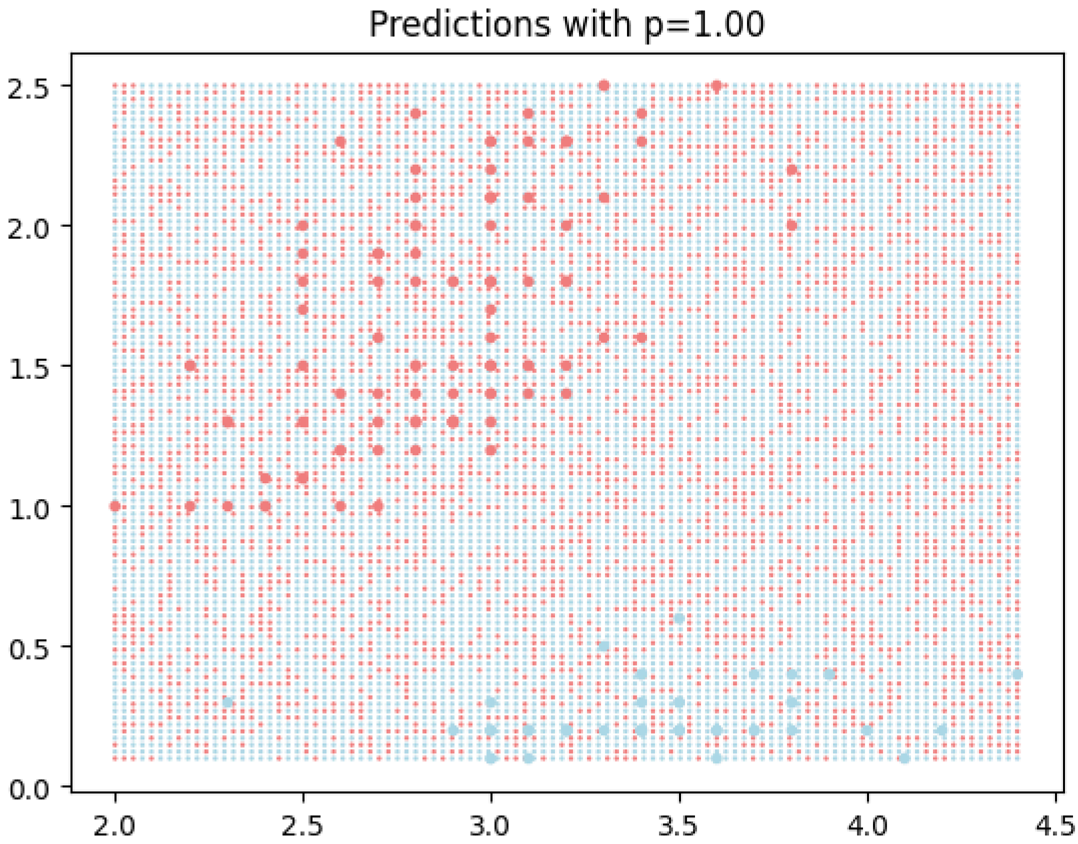
In [ ]:
```
for p in p_value:
    y_pred = DummyBinaryClassifier("normal",p).predict(data)
    y_reshape = y_pred.reshape(x1v.shape)
    plt.figure()
    color_map = colors.ListedColormap(['lightcoral', 'lightblue'])
    plt.scatter(x1v, x2v, marker='.', s=2, c=y_pred, cmap=color_map)
    plt.scatter(x1, x2, marker='.', c=y, cmap=color_map)
    plt.title(f'Predictions with p={p:.2f}')
    plt.show()
```


Predictions with p=0.00
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


Predictions with p=0.25
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

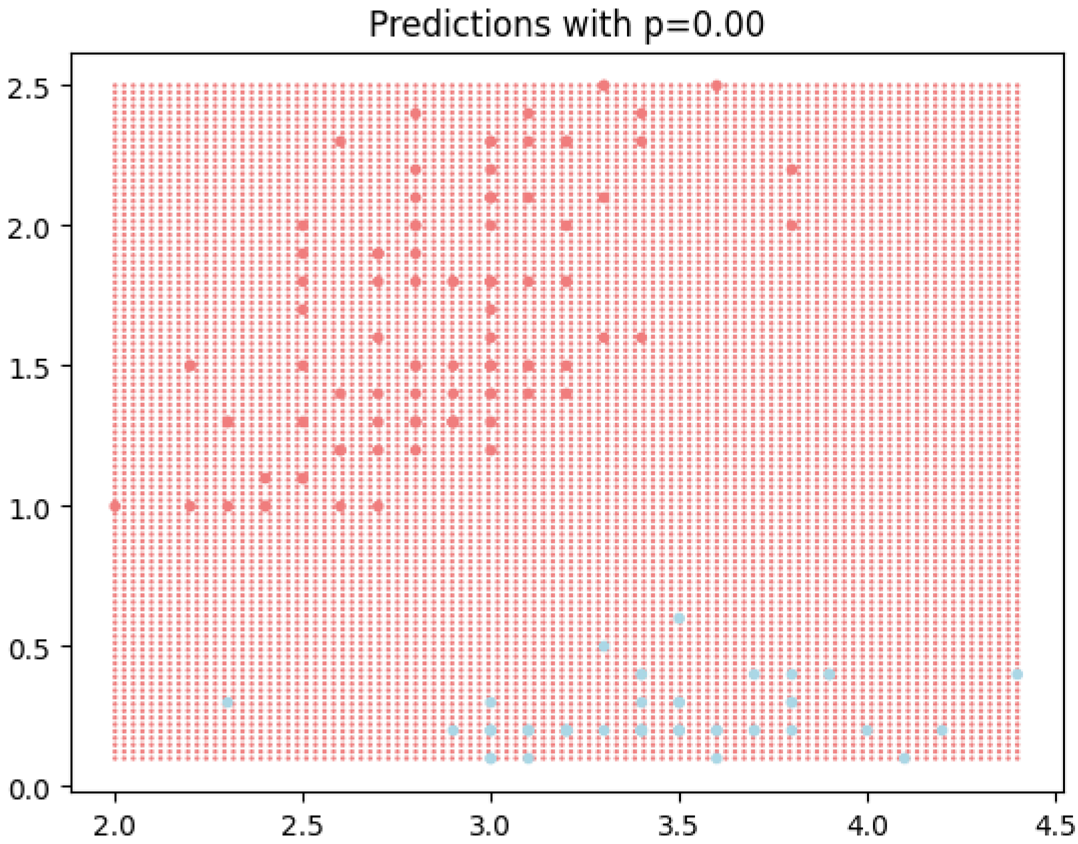

Predictions with p=0.50
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

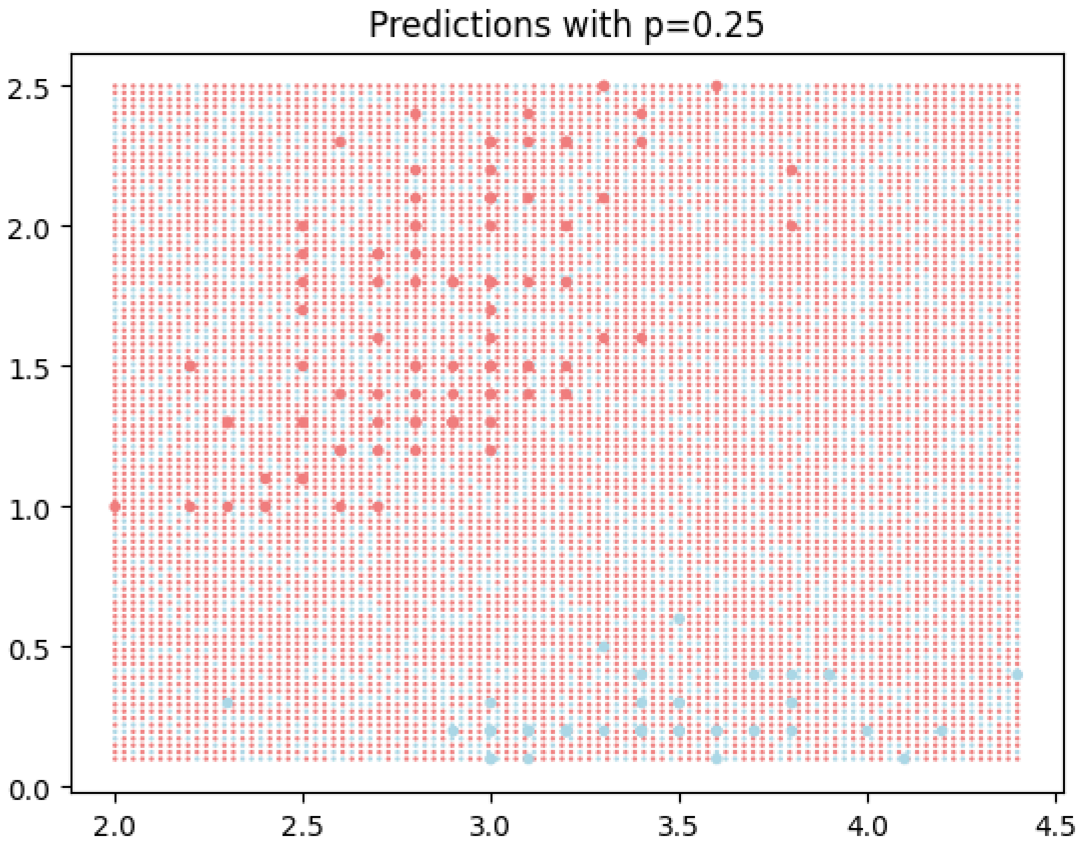

Predictions with p=0.75
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5

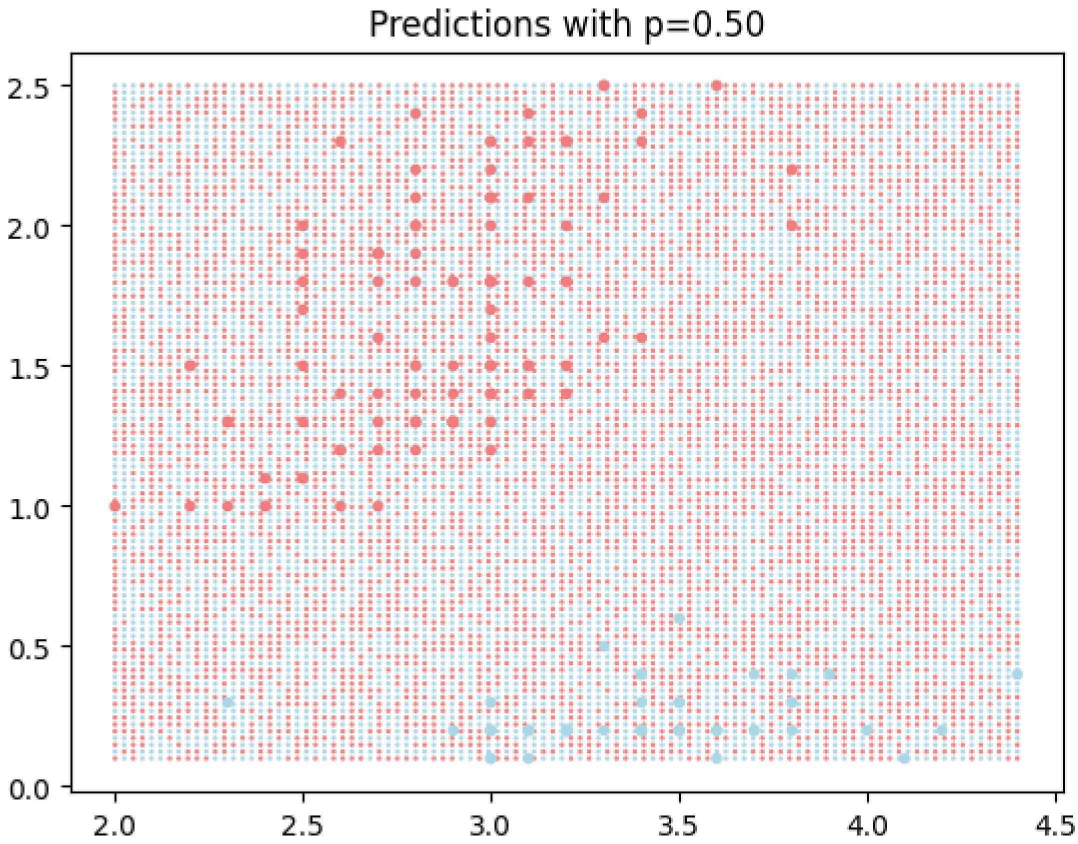


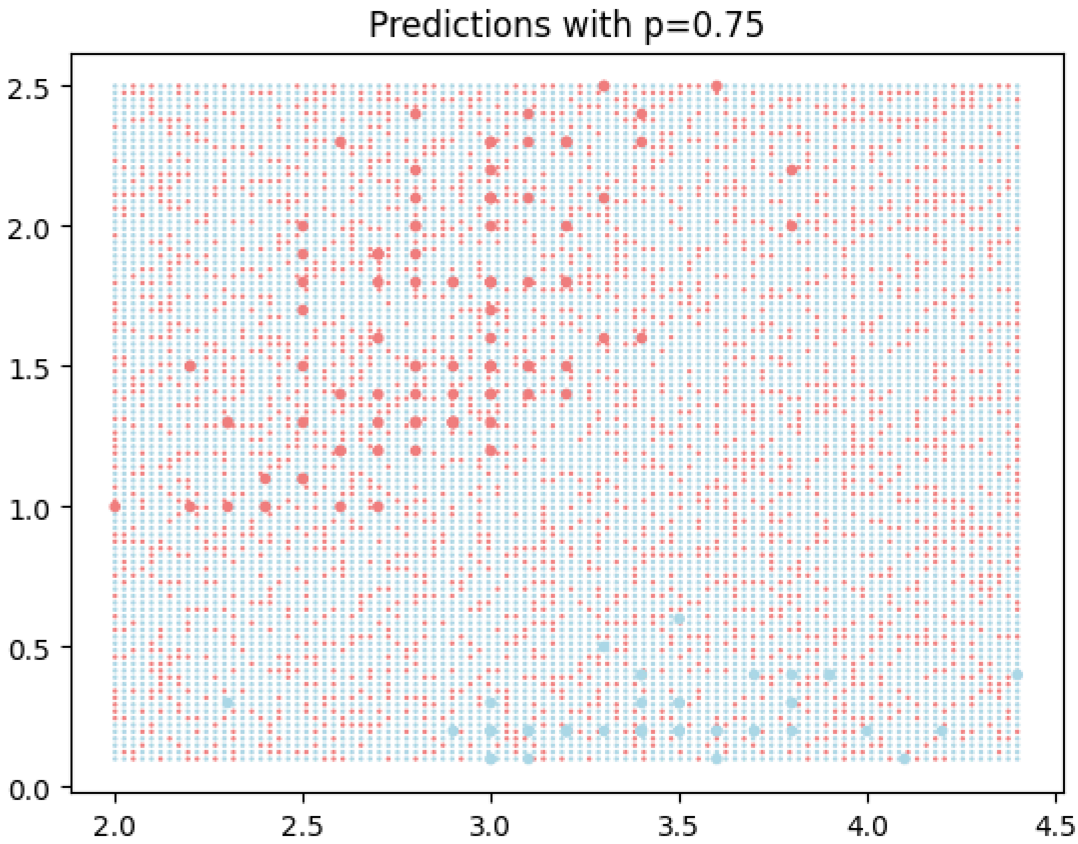
Uniform Random method

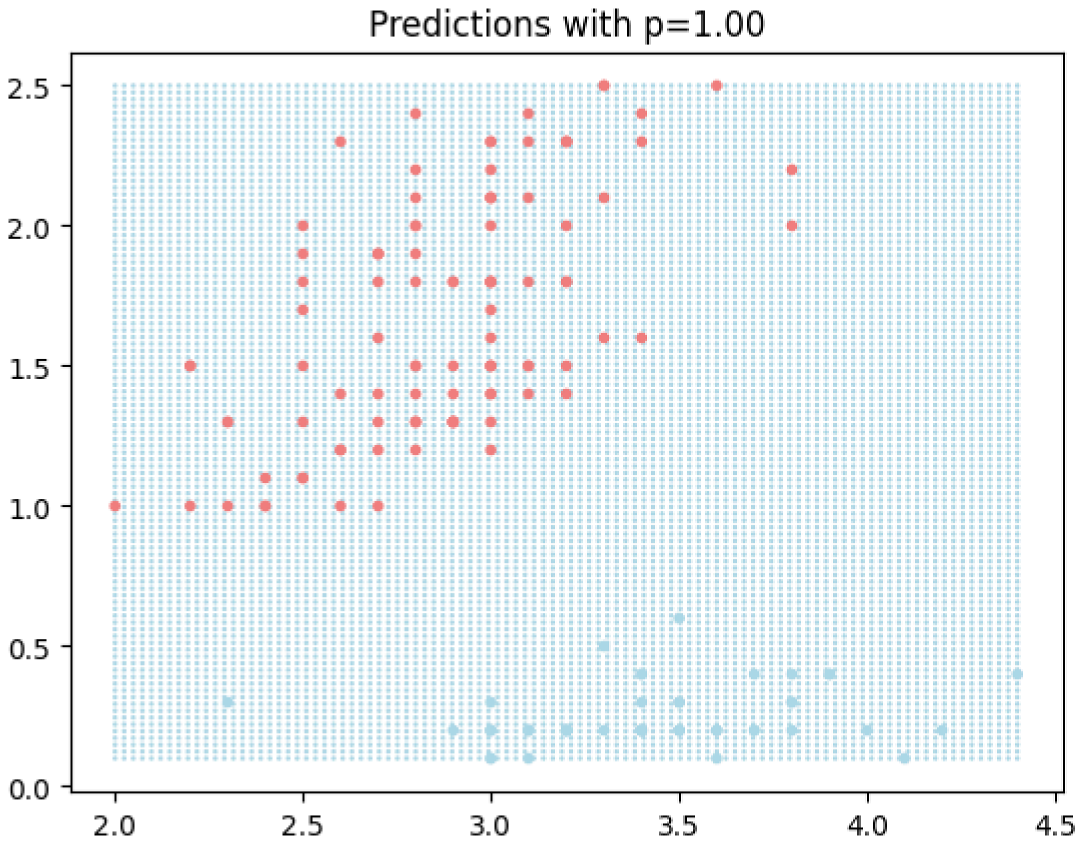
In [ ]:
```
for p in p_value:
    y_pred = DummyBinaryClassifier("uniform_random",p).predict(data)
    y_reshape = y_pred.reshape(x1v.shape)
    plt.figure()
    color_map = colors.ListedColormap(['lightcoral', 'lightblue'])
    plt.scatter(x1v, x2v, marker='.', s=2, c=y_pred, cmap=color_map,vmin=0,vmax=1)
    plt.scatter(x1, x2, marker='.', c=y, cmap=color_map)
    plt.title(f'Predictions with p={p:.2f}')
    plt.show()
```


Predictions with p=0.00
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


Predictions with p=0.25
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


Predictions with p=0.50
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5


Predictions with p=0.75
2.5
2.0
1.5
1.0
0.5
0.0
2.0
2.5
3.0
3.5
4.0
4.5




```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```