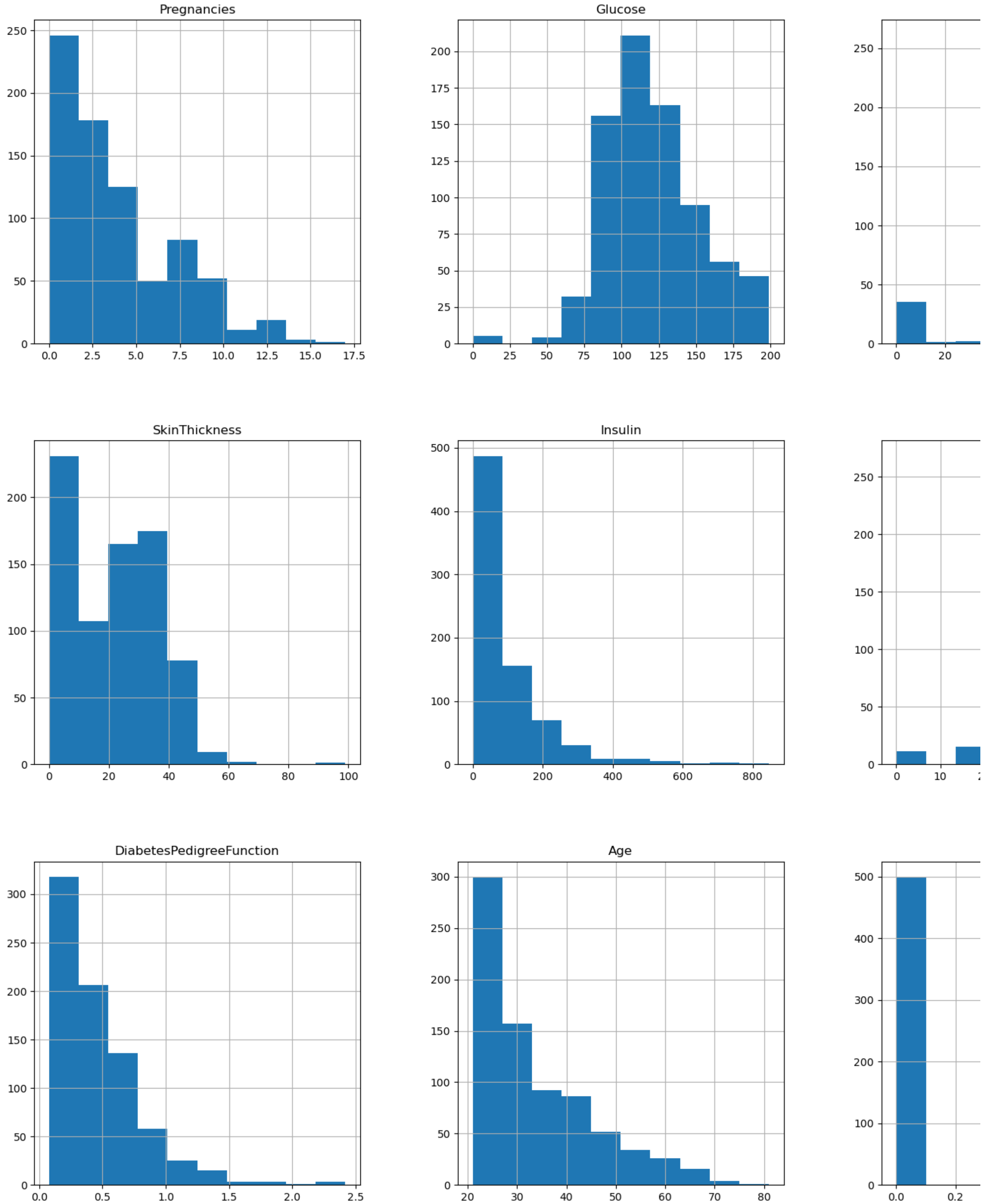
Data Exploration

In [1]:

```
import pandas as pd
diabetes_data = pd.read_csv('diabetes.csv')
p = diabetes_data.hist(figsize = (20,20))
diabetes_data.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

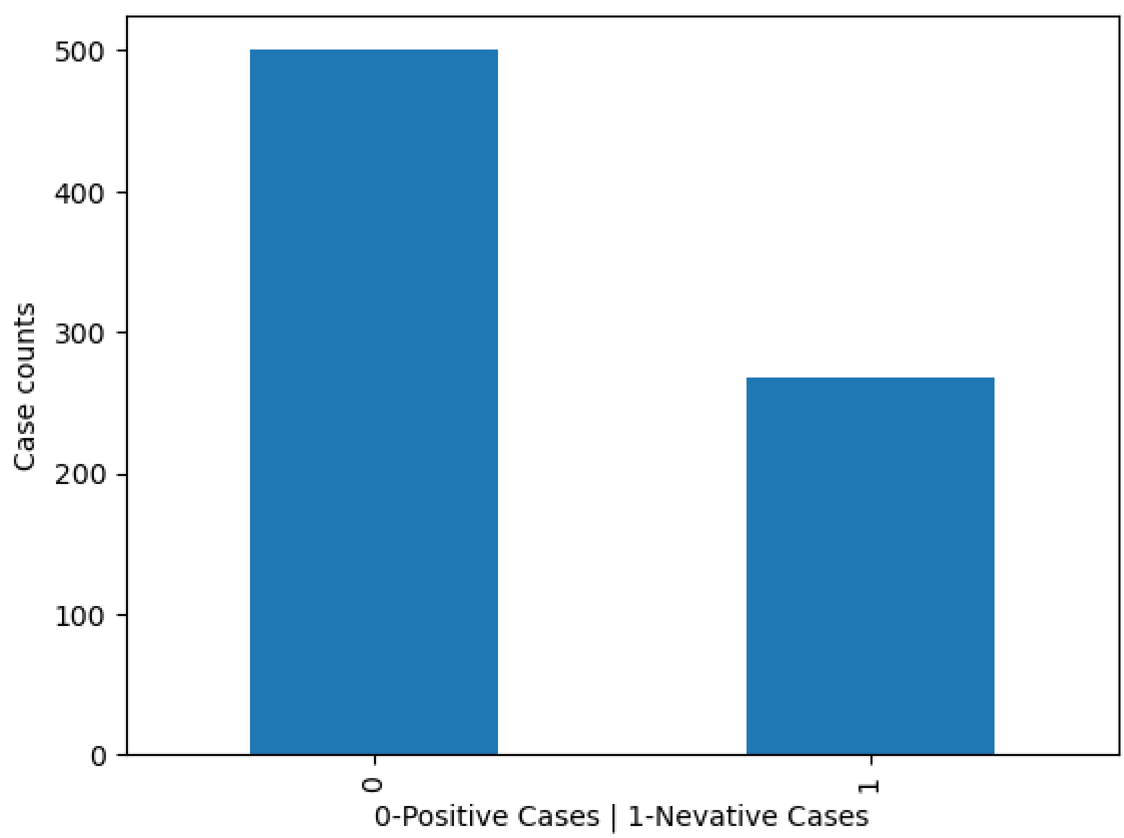

Pregnancies
0 50 100 150 200 250
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
Glucose
0 25 50 75 100 125 150 175 200
0 25 50 75 100 125 150 175 200
0 50 100 150 200 250
0 20
SkinThickness
0 50 100 150 200
0 20 40 60 80 100
Insulin
0 100 200 300 400 500
0 200 400 600 800
0 50 100 150 200 250
0 10
DiabetesPedigreeFunction
0 50 100 150 200 250 300
0.0 0.5 1.0 1.5 2.0 2.5
Age
0 50 100 150 200 250 300
20 30 40 50 60 70 80
0 100 200 300 400 500
0.0 0.2

In [2]:

```python

## checking the balance of the data by plotting the count of outcomes by the
color_wheel = {1: "#0392cf",
               2: "#7bc043"}
colors = diabetes_data["Outcome"].map(lambda x: color_wheel.get(x + 1))
print(diabetes_data.Outcome.value_counts())
p=diabetes_data.Outcome.value_counts().plot(kind="bar",)
p.set_xlabel("0-Positive Cases | 1-Nevative Cases")
p.set_ylabel("Case counts")
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

```
Text(0, 0.5, 'Case counts')
```



In [3]:

```
from sklearn.datasets import load_svmlight_file
diabetes_features,diabetes_labels = load_svmlight_file('diabetes_scale.txt')
#Since the data is of the type SVMLight file using a tht particular reader t
print(diabetes_features.shape)
```

```
(768, 8)
```

In [4]:

```
#the data in of the form of a tuple
diabetes_features = diabetes_features.toarray()
print(type(diabetes_features),type(diabetes_labels))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

In [5]:

```
import numpy as np
import random
np.random.seed(10)
weights = np.random.rand(8)
# weights = np.array([0.,0.,0.,0.,0.,0.,0.,0.])
weights
```

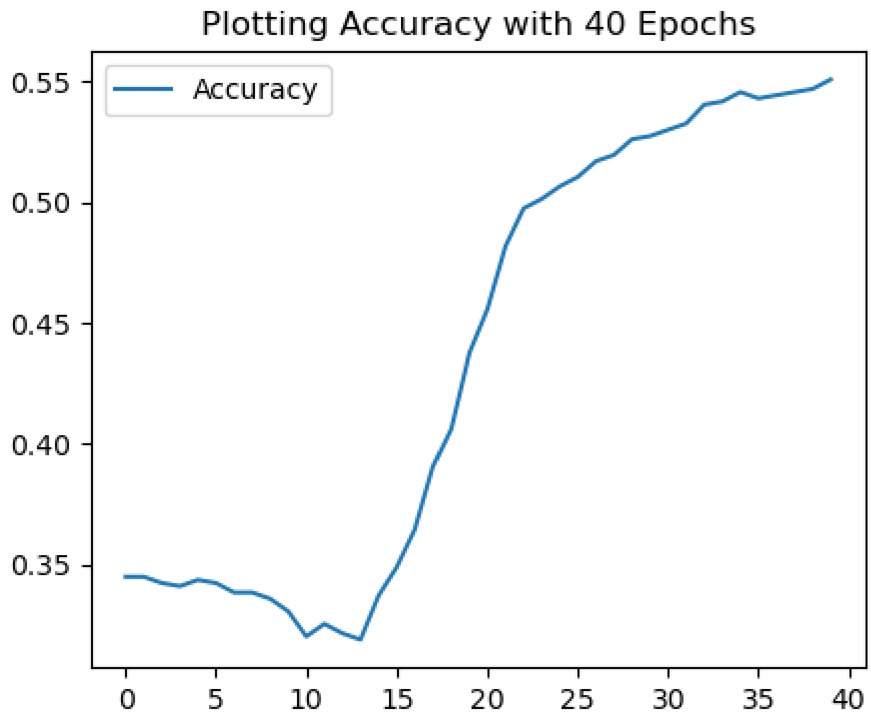
```
array([0.77132064, 0.02075195, 0.63364823, 0.74880388, 0.49850701,
       0.22479665, 0.19806286, 0.76053071])
```

In [6]:

```
def accuracy(y_true, y_pred):
    accuracy = np.mean(y_pred == y_true)
    return accuracy
#Prredicting for each row {-1,1} and then getting tht into pred array and re
def predict(features,weight):
    pred = list()
    for row in features:
        activation = 0
        for i in range(len(row)):
            activation += weights[i ] * row[i] #calc Xi*Wi
        pred.append(np.sign(activation))# Predictig for 1 row
    return (accuracy(diabetes_labels, pred))
def update_weights(weights):
    l_rate = 0.0001
    summation = 0
    for row,y in zip(diabetes_features,diabetes_labels):
        # calaculating the summation of 1*8 vector for each row
        summation+= (y*row*(1 if y*(np.dot(row,weights))<0 else 0))
    #now modifying the weights according to learning for
    weights += (l_rate*summation)
#     print("updated",weights)
    return weights
```

In [7]:

```
# print("1st accuracy" ,predict(diabetes_features,weights))
import matplotlib.pyplot as plt
best_weights = 0
best_accu = 0
acc_list,loss_list = [],[]
for i in range(40):
    acc = predict(diabetes_features,weights)
    weights = update_weights(weights)
    acc_list.append(acc)
    if acc>best_accu:
        best_accu = acc
        best_weights = weights
#     l = zero_one_loss(diabetes_features,diabetes_labels,weights)
#     loss_list.append(l)
# plt.figure(figsize=(5, 4))
# plt.title("Plotting Loss with 20 Epochs ")
# plt.plot(np.arange(20), e_train_loss_list, label = " Loss")
# plt.legend()
# plt.show()

#Plotting Validation Curve
plt.figure(figsize=(5, 4))
plt.title("Plotting Accuracy with 40 Epochs")
plt.plot(np.arange(40), acc_list, label = "Accuracy")
# plt.plot(np.arange(40), loss_list, label = "Loss")
plt.legend()
plt.show()

print("best accuracy: ",best_accu)
print("best weights: " ,best_weights)
```




```
best accuracy:  0.55078125
best weights:  [ 0.10378526 -0.10073597  0.51230559  0.24198056 -0.09956228  0.03309548
 -0.2585228   0.05204732]
```

Experimentation

In [8]:

```
#Adding an additional column to the dataset initialized as all 1 & and addit
import numpy as np
type(diabetes_features)
bias_col = np.ones(len(diabetes_features)).reshape(-1,1)
mod_diabetes_features = np.concatenate([diabetes_features, bias_col], axis=1
print(mod_diabetes_features.shape)
```

```
(768, 9)
```

In [9]:

```
port tr    est_split

n, b_y_    = train_test_split(mod_diabetes_features, diabetes_labels, test_size=0
b_y_Val    ain_test_split(X_bigtrain, y_bigtrain, test_size=0.20, random_state=42

in.shap
.shape)
al.shap


 y_test    ain_test_split(diabetes_features, diabetes_labels, test_size=0.20, rand
= train    _split(X_bigtrain, y_bigtrain, test_size=0.20, random_state=42,stratify

.shape)
hape)
.shape)
```

```
Splits With Bias
Training Size : (491, 9)
Testing Size : (154, 9)
Validation Size : (123, 9)
Splits With Bias
Training Size : (491, 8)
Testing Size : (154, 8)
Validation Size : (123, 8)
```

In [10]:

```
import seaborn as sns
def accuracy(y_true, y_pred):
    accuracy = np.mean(y_pred == y_true)
    return accuracy
#Prredicting for each row {-1,1} and then getting tht into pred array and re
def predict(features,labels,weights):
    pred = list()
    for row in features:
        activation = 0
        for i in range(len(row)):
            activation += weights[i] * row[i] #calc Xi*Wi
        pred.append(np.sign(activation))# Predictig for 1 row
    return (accuracy(labels, pred))
def accuracy_scores(features,labels,weights):
    from sklearn.metrics import accuracy_score, f1_score, precision_score, re
    from sklearn.metrics import roc_curve,roc_auc_score
    pred = list()
    for row in features:
        activation = 0
        for i in range(len(row)):
            activation += weights[i] * row[i] #calc Xi*Wi
        pred.append(np.sign(activation))# Predictig for 1 row
    print("F1 Score :",f1_score(labels, pred, average="binary"))
    print("Precision :",precision_score(labels, pred, average="binary"))
    print("Recall :",recall_score(labels, pred, average="binary"))
    print("Confusion Matrix :")
    cmat = confusion_matrix(labels, pred)
    sns.heatmap(cmat,annot=True)
    ax= plt.subplot()
    sns.heatmap(cmat, annot=True, fmt='g', ax=ax);
    # labels, title and ticks
    plt.title('Confusion Matrix')
    ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['Positive', 'Negative']); ax.yaxis.set_ticklabe

#    plt.set_xticklabels(['Positive', 'Negative'])
#    plt.set_yticklabels(['Positive', 'Negative'])
```

```
    plt.show()
    return
def update_weights(weights,festures,labels,loss,l_rate):
#     l_rate = 0.001
    summation = 0
    for row,y in zip(festures,labels):
        # calaculating the summation of 1*8 vector for each row
        if(loss == "perceptron"):
            ls = max(0,-1*y*(np.dot(row,weights))) #Perceptron Loss
            summation+= (y*row*ls)
        elif(loss == "zero_one"):
            ls = (1 if y*(np.dot(row,weights))<0 else 0) #ZeroOne Loss
            summation+= (y*row*ls)
    #now modifying the weights according to learning for
    weights += (l_rate*summation)
#     print("updated",weights)
    return weights
def zero_one_loss(features,labels,weights):
    loss =0
    for row ,y in zip(features,labels):
        loss+=(1 if y*(np.dot(row,weights))<0 else 0)
    return loss/len(labels)

def perceptron_loss(features,labels,weights):
    loss =0
    for row ,y in zip(features,labels):
        loss+= max(0,-1*y*(np.dot(row,weights)))
    return loss/len(labels)
```

SK learn baseline

In [11]:

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, f1_score, precision_score, recal
clf = Perceptron(tol=1e-3, random_state=42)
clf.fit(X_bigtrain, y_bigtrain)
Perceptron()
sk_pred = clf.predict(X_test)
print("Testing Accuracy for Sklearn Baseline : ",clf.score(X_test, y_test))
print("F1 Score :",f1_score(y_test, sk_pred, average="binary"))
print("Precision :",precision_score(y_test, sk_pred, average="binary"))
print("Recall :",recall_score(y_test, sk_pred, average="binary"))
```

```
Testing Accuracy for Sklearn Baseline :  0.7987012987012987
F1 Score : 0.8472906403940887
Precision : 0.8349514563106796
Recall : 0.86
```

In [12]:

```
#Initializing 8 weights + 1 Bias
np.random.seed(89)
b_weights = np.random.rand(9)
# weights = np.zeros(8)
weights
```

```
array([ 0.10378526, -0.10073597,  0.51230559,  0.24198056, -0.09956228,
        0.03309548, -0.2585228 ,  0.05204732])
```

Experimenting with initial weights with bias

In [13]:

```
#Experimenting with the the best Random seed for weights
all_acc=list()
overall_best_acc = 0
overall_best_weights = 0
l_rate =0.001
for j in range(202):
    np.random.seed(j)
    b_weights = np.random.rand(9)
    best_weights = 0
    best_accu = 0
    for i in range(20):
        b_weights = update_weights(b_weights,b_X_train,b_y_train,"zero_one",1
        train_acc = predict(b_X_train,b_y_train,b_weights)
        val_acc = predict(b_X_Val,b_y_Val,b_weights)
#          print("Iteration ",i)
#          print("Training accuracy:",train_acc," Validation accuracy:",val_ac
        if val_acc>best_accu:
            best_accu = val_acc
            best_weights = b_weights
#      print("for Random Seed:",j)
#      print("best accuracy: ",best_accu)
    if best_accu>overall_best_acc:
            overall_best_acc = best_accu
            overall_best_weights = best_weights
    all_acc.append((best_accu,j))
#      print("best weights: " ,best_weights)
all_acc.sort(reverse=True)
all_acc[:6]
```

```
[(0.7886178861788617, 89),
 (0.7560975609756098, 111),
 (0.7479674796747967, 195),
 (0.7398373983739838, 10),
 (0.7317073170731707, 160),
 (0.7317073170731707, 107)]
```

Experimenting with initial weights without bias

In [14]:

```python
#Experimenting with the the best Random seed for weights
all_acc=list()
overall_best_acc = 0
overall_best_weights = 0
l_rate =0.001
for j in range(202):
    np.random.seed(j)
    weights = np.random.rand(8)
    best_weights = 0
    best_accu = 0
    for i in range(20):
        b_weights = update_weights(weights,X_train,y_train,"zero_one",l_rate
        train_acc = predict(X_train,y_train,weights)
        val_acc = predict(X_Val,y_Val,weights)
#         print("Iteration ",i)
#         print("Training accuracy:",train_acc," Validation accuracy:",val_a
        if val_acc>best_accu:
            best_accu = val_acc
            best_weights = weights
#     print("for Random Seed:",j)
#     print("best accuracy: ",best_accu)
    if best_accu>overall_best_acc:
            overall_best_acc = best_accu
            overall_best_weights = best_weights
    all_acc.append((best_accu,j))
#     print("best weights: " ,best_weights)
all_acc.sort(reverse=True)
all_acc[:6]
```

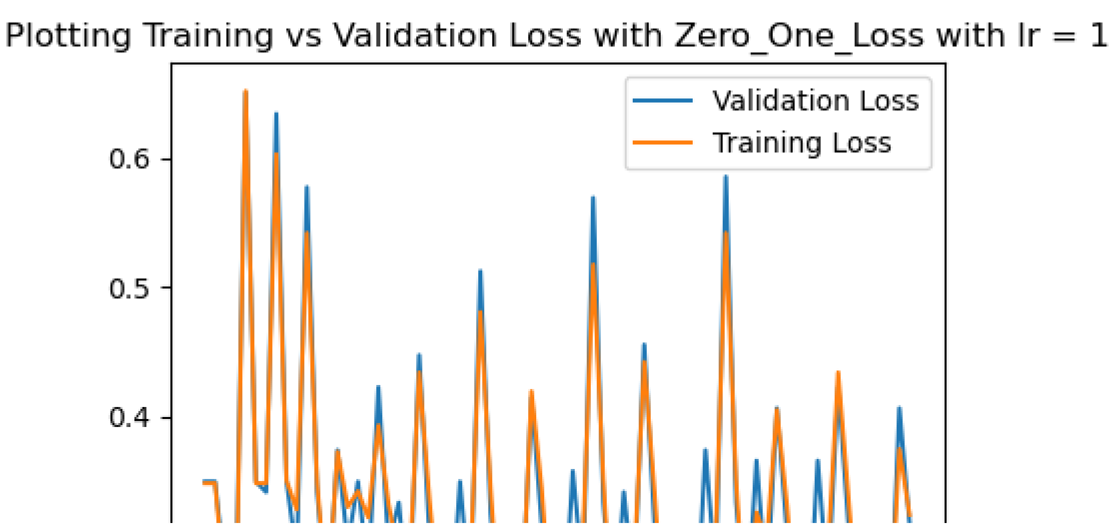
```
[(0.7723577235772358, 84),
 (0.7723577235772358, 33),
 (0.7723577235772358, 26),
 (0.7642276422764228, 195),
 (0.7642276422764228, 140),
 (0.7642276422764228, 138)]
```

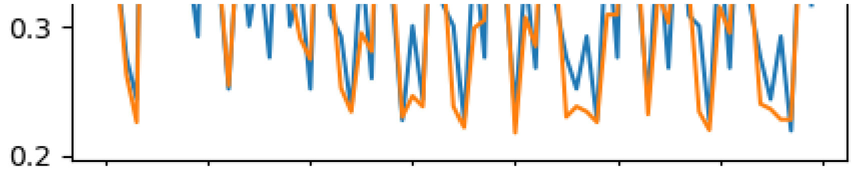
Hyper-Parameter Tuning with Bias

In [15]:

```
import matplotlib.pyplot as plt
b_accuracy_table = list()
learning_Rates = [1,.1,.01,.001,0.0001]
epoch = np.arange(10,101,10)
loss =["perceptron","zero_one"]
for e in epoch:
    for lr_rate in learning_Rates:
        for l in loss:
            best_weights = 0
            best_epoch = 0
            best_accu = 0
            e_val_loss_list = []
            e_train_loss_list = []
            e_val_acc_list = []
            e_train_acc_list = []
            np.random.seed(89)
            b_weights = np.random.rand(9)
            for i in range(e):
                b_weights = update_weights(b_weights,b_X_train,b_y_train,l,lr
                train_acc = predict(b_X_train,b_y_train,b_weights)
                val_acc = predict(b_X_Val,b_y_Val,b_weights)
                e_val_acc_list.append(val_acc)
                e_train_acc_list.append(train_acc)
                if l =="zero_one":
                    e_train_loss = zero_one_loss(b_X_train,b_y_train,b_weight
                    e_val_loss = zero_one_loss(b_X_Val,b_y_Val,b_weights)
                else:
                    e_train_loss = perceptron_loss(b_X_train,b_y_train,b_wei
                    e_val_loss = perceptron_loss(b_X_Val,b_y_Val,b_weights)
                e_val_loss_list.append(e_val_loss)
                e_train_loss_list.append(e_train_loss)
            #     print("Iteration ",i)
            #     print("Training accuracy:",train_acc," Validation accuracy
                if val_acc>best_accu:
                    best_epoch = (i+1)
                    best_accu = val_acc
                    best_weights = b_weights

```

```
        b_accuracy_table.append((best_accu,lr_rate,l,best_epoch,e,best_we
        #Plotting Loss Curve
        plt.figure(figsize=(5, 4))
        if l =="zero_one":
            plt.title("Plotting Training vs Validation Loss with Zero_One
        else:
            plt.title("Plotting Training vs Validation Loss with Percept
        plt.plot(np.arange(e), e_val_loss_list, label = "Validation Loss'
        plt.plot(np.arange(e), e_train_loss_list, label = "Training Loss'
        plt.legend()
        plt.show()

        #Plotting Validation Curve
        plt.figure(figsize=(5, 4))
        if l =="zero_one":
            plt.title("Plotting Training vs Validation Accuracy with Zer
        else:
            plt.title("Plotting Training vs Validation Accuracy with Per

        plt.plot(np.arange(e), e_val_acc_list, label = "Validation Accur
        plt.plot(np.arange(e), e_train_acc_list, label = "Training Accur
        plt.legend()
        plt.show()
        print("best accuracy: ",best_accu)
        print("best weights: " ,best_weights)
```





0.3
0.2

In [16]:

```python
import pandas as pd
b_accuracy_table.sort(reverse=True)
dfb = pd.DataFrame(b_accuracy_table, columns=["Val_Accuracy", "Learning Rate'
dfb.iloc[:45,:]
```

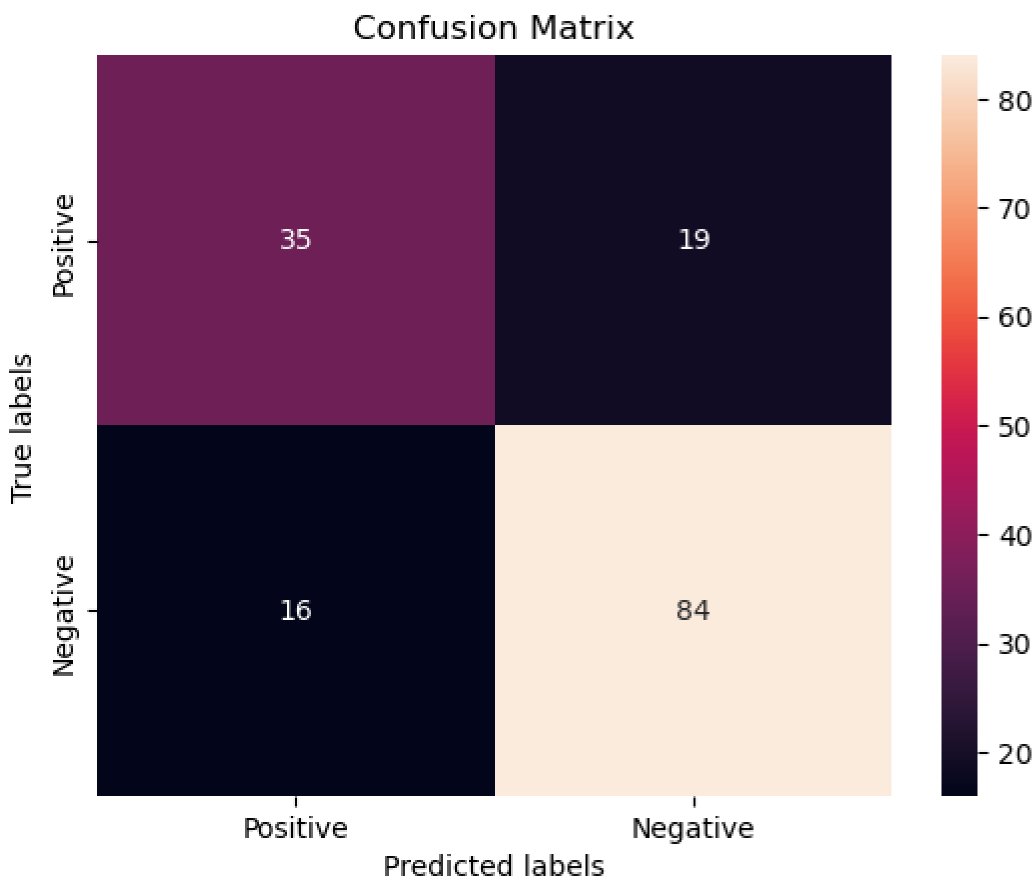
	Val_Accuracy	Learning Rate	Loss	nth Epoch	Epoch Range	
0	0.788618	1.000	zero_one	84	100	[-130.2063526795285, -43
1	0.788618	1.000	zero_one	84	90	[-202.20635517952866, -3
2	0.788618	0.100	zero_one	76	100	[-13.647380979530904, -4
3	0.788618	0.100	zero_one	76	90	[-9.370908279530937, -37
4	0.788618	0.100	zero_one	76	80	[-10.200318579530958, -3
5	0.788618	0.010	zero_one	66	100	[-0.8885427295311398, -4
6	0.788618	0.010	zero_one	66	90	[-0.9320719795311407, -4
7	0.788618	0.010	zero_one	66	80	[-1.1103071395311424, -4
8	0.788618	0.010	zero_one	66	70	[-1.9785424695311433, -3
9	0.788618	0.001	zero_one	20	100	[-0.18030591503116294, -
10	0.788618	0.001	zero_one	20	90	[-0.13636471053116314, -
11	0.788618	0.001	zero_one	20	80	[-0.14807057653116332, -
12	0.788618	0.001	zero_one	20	70	[-0.17307056103116342, -
13	0.788618	0.001	zero_one	20	60	[-0.13748229953116364, -
14	0.788618	0.001	zero_one	20	50	[-0.16436464153116373, -
15	0.788618	0.001	zero_one	20	40	[-0.16795285853116382, -
16	0.788618	0.001	zero_one	20	30	[-0.12395283553116401, -
17	0.788618	0.001	zero_one	20	20	[-0.05665868603116414, -
18	0.780488	1.000	zero_one	68	80	[-99.02984967952895, -36
19	0.780488	1.000	zero_one	68	70	[-171.20632117952903, -3
20	0.772358	1.000	zero_one	30	60	[-139.6180611795291, -43
21	0.772358	1.000	zero_one	30	50	[-95.7945091795294, -384
22	0.772358	1.000	zero_one	30	40	[-215.26510517952954, -3
23	0.772358	1.000	zero_one	30	30	[-128.73565967952982, -4
24	0.772358	0.100	zero_one	37	70	[-13.288552929530972, -3
25	0.772358	0.100	zero_one	37	60	[-19.794435179530986, -3
26	0.772358	0.100	zero_one	37	50	[-9.235607979530997, -43
27	0.772358	0.100	zero_one	37	40	[-7.653253329531025, -36
28	0.772358	0.010	zero_one	36	60	[-1.4632480595311448, -4
29	0.772358	0.010	zero_one	36	50	[-0.8938360345311473, -4
30	0.772358	0.010	zero_one	36	40	[-2.233836134531148, -3.5
31	0.764228	0.100	zero_one	10	30	[-13.800311929531034, -3

	Val_Accuracy	Learning Rate	Loss	nth Epoch	Epoch Range	
32	0.764228	0.100	zero_one	10	20	[-8.25913242953105, -40.7
33	0.764228	0.100	zero_one	10	10	[-13.812073079531075, -2
34	0.756098	1.000	zero_one	4	20	[-106.44152017953003, -4
35	0.756098	1.000	zero_one	4	10	[-140.67680517953025, -2
36	0.756098	0.010	zero_one	9	30	[-0.9632474545311513, -4
37	0.756098	0.010	zero_one	9	20	[-0.8191295845311528, -4
38	0.756098	0.010	zero_one	9	10	[-0.08089404453115911, -
39	0.682927	0.001	zero_one	9	10	[0.08345899046883516, -
40	0.650407	1.000	perceptron	1	100	[1.4688639284294427e+2
41	0.650407	1.000	perceptron	1	90	[3.7459274320849006e+2
42	0.650407	1.000	perceptron	1	80	[9.552942280671046e+22
43	0.650407	1.000	perceptron	1	70	[2.436211268701485e+19
44	0.650407	1.000	perceptron	1	60	[6.212876798970035e+17

In [17]:

```python
overall_best_weights = dfb['Best Weights'][0]
test_acc = predict(b_X_test,b_y_test,overall_best_weights)
print("Testing accuracy:",test_acc)
accuracy_scores(b_X_test,b_y_test,overall_best_weights)
```

```
Testing accuracy: 0.7727272727272727
F1 Score : 0.8275862068965517
Precision : 0.8155339805825242
Recall : 0.84
Confusion Matrix :
```

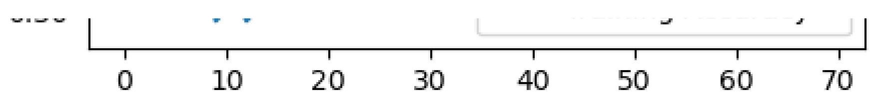


Hyperparameter Tuning Without Bias

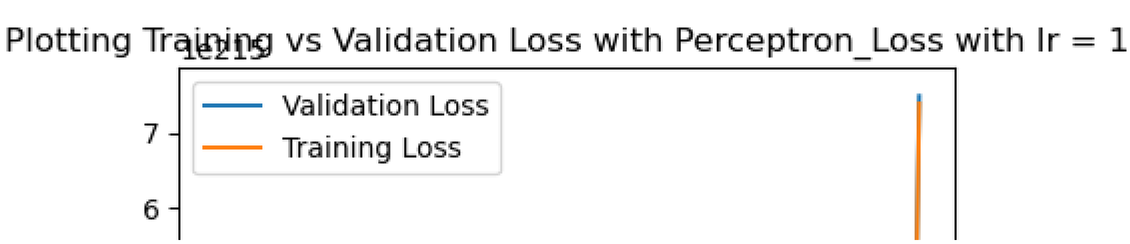
In [18]:

```
accuracy_table = list()
learning_Rates = [1,.1,.01,.001,0.0001]
epoch = np.arange(10,101,10)
loss =["perceptron","zero_one"]
for e in epoch:
    for lr_rate in learning_Rates:
        for l in loss:
            best_weights = 0
            best_accu = 0
            best_epoch = 0
            e_val_loss_list = []
            e_train_loss_list = []
            e_val_acc_list = []
            e_train_acc_list = []
            np.random.seed(84)
            weights = np.random.rand(8)
            for i in range(e):
                weights = update_weights(weights,X_train,y_train,l,lr_rate)
                train_acc = predict(X_train,y_train,weights)
                val_acc = predict(X_Val,y_Val,weights)
                e_val_acc_list.append(val_acc)
                e_train_acc_list.append(train_acc)
                if l =="zero_one":
                    e_train_loss = zero_one_loss(X_train,y_train,weights)
                    e_val_loss = zero_one_loss(X_Val,y_Val,weights)
                else:
                    e_train_loss = perceptron_loss(X_train,y_train,weights)
                    e_val_loss = perceptron_loss(X_Val,y_Val,weights)
                e_val_loss_list.append(e_val_loss)
                e_train_loss_list.append(e_train_loss)
            #     print("Iteration ",i)
            #     print("Training accuracy:",train_acc," Validation accuracy
                if val_acc>best_accu:
                    best_epoch=(i+1)
                    best_accu = val_acc
                    best_weights = weights

            accuracy_table.append((best_accu,lr_rate,l,best_epoch,e,best_wei
```

```
        #Plotting Loss Curve
        plt.figure(figsize=(5, 4))
        if l =="zero_one":
            plt.title("Plotting Training vs Validation Loss with Zero_One
        else:
            plt.title("Plotting Training vs Validation Loss with Perceptr
        plt.plot(np.arange(e), e_val_loss_list, label = "Validation Loss'
        plt.plot(np.arange(e), e_train_loss_list, label = "Training Loss'
        plt.legend()
        plt.show()

        #Plotting Validation Curve
        plt.figure(figsize=(5, 4))
        if l =="zero_one":
            plt.title("Plotting Training vs Validation Accuracy with Zero
        else:
            plt.title("Plotting Training vs Validation Accuracy with Perc

        plt.plot(np.arange(e), e_val_acc_list, label = "Validation Accura
        plt.plot(np.arange(e), e_train_acc_list, label = "Training Accura
        plt.legend()
        plt.show()
        print("best accuracy: ",best_accu)
        print("best weights: " ,best_weights)
```




```
best accuracy:  0.6910569105691057
best weights:  [-0.164712    0.07884335  0.1884986   0.06869608  0.29625329 -0.08263069
 -0.23240394 -0.06289264]
```



In [19]:

```
import pandas as pd
accuracy_table.sort(reverse=True)
df = pd.DataFrame(accuracy_table, columns=["Val_Accuracy", "Learning Rate","I
df
```

	Val_Accuracy	Learning Rate	Loss	nth Epoch	Epoch Range	
0	0.796748	0.0010	zero_one	24	100	[-0.05242974486851241, -
1	0.796748	0.0010	zero_one	24	90	[-0.14925328036851254, -
2	0.796748	0.0010	zero_one	24	80	[-0.17001797686851272, -
3	0.796748	0.0010	zero_one	24	70	[-0.1546650178685129, -0
4	0.796748	0.0010	zero_one	24	60	[-0.07995910486851306, -
...	...	...	...	...	...	...
95	0.479675	0.0001	zero_one	20	20	[-0.2744531867685141, 0.
96	0.479675	0.0001	perceptron	30	30	[-0.25348978923140203, (
97	0.341463	0.0001	perceptron	20	20	[-0.23006416383874562, (
98	0.333333	0.0001	zero_one	1	10	[-0.13851198281851426, (
99	0.333333	0.0001	perceptron	1	10	[-0.15418712858391273, (

100 rows × 6 columns

In [20]:

```
overall_best_weights = df['Best Weights'][0]
test_acc = predict(X_test,y_test,overall_best_weights)
print("Testing accuracy:",test_acc)
accuracy_scores(X_test,y_test,overall_best_weights)
```

```
Testing accuracy: 0.564935064935065
F1 Score : 0.5109489051094891
Precision : 0.9459459459459459
Recall : 0.35
Confusion Matrix :
```