In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python # For
example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output. #

Importing necessary libraries for this notebook.
import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
from plotly.offline import init_notebook_mode, iplot, plot init_notebook_mode(connected=True)
import plotly.express as px import
plotly.graph_objects as go

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split from
sklearn.neighbors import KNeighborsClassifier from
sklearn.svm import SVC
from sklearn.metrics import confusion_matrix

from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout from
keras.regularizers import l2, l1
from keras.metrics import BinaryAccuracy
```

```
/kaggle/input/occupancy-detection-data-set-uci/datatest.txt
/kaggle/input/occupancy-detection-data-set-uci/datatest2.txt
```

```
Using TensorFlow backend.
```

In [2]:

```
datatest = pd.read_csv("/kaggle/input/occupancy-detection-data-set-uci/datatest.txt") datatest2 =
pd.read_csv("/kaggle/input/occupancy-detection-data-set-uci/datatest2.txt") datatraining =
pd.read_csv("/kaggle/input/occupancy-detection-data-set-uci/datatraining.txt")
```

Exploratory Data Analysis

We have three different .txt file as datatest, datatest2 and datatraining.

In [3]:

```
print(datatest.info())
datatest.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2665 entries, 140 to 2804
Data columns (total 7 columns):
```

```
date            2665 non-null object
Temperature     2665 non-null float64
Humidity        2665 non-null float64
Light           2665 non-null float64
CO2             2665 non-null float64
HumidityRatio   2665 non-null float64
Occupancy       2665 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 166.6+ KB
None
```

Out [3]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
140	2015-02-02 14:19:00	23.7000	26.272	585.200000	749.200000	0.004764	1
141	2015-02-02 14:19:59	23.7180	26.290	578.400000	760.400000	0.004773	1
142	2015-02-02 14:21:00	23.7300	26.230	572.666667	769.666667	0.004765	1
143	2015-02-02 14:22:00	23.7225	26.125	493.750000	774.750000	0.004744	1
144	2015-02-02 14:23:00	23.7540	26.200	488.600000	779.000000	0.004767	1

In [4]:

```
print(datatest2.info())
datatest2.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9752 entries, 1 to 9752
Data columns (total 7 columns):
date            9752 non-null object
Temperature     9752 non-null float64
Humidity        9752 non-null float64
Light           9752 non-null float64
CO2             9752 non-null float64
HumidityRatio   9752 non-null float64
Occupancy       9752 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 609.5+ KB
None
```

Out [4]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-11 14:48:00	21.7600	31.133333	437.333333	1029.666667	0.005021	1
2	2015-02-11 14:49:00	21.7900	31.000000	437.333333	1000.000000	0.005009	1
3	2015-02-11 14:50:00	21.7675	31.122500	434.000000	1003.750000	0.005022	1
4	2015-02-11 14:51:00	21.7675	31.122500	439.000000	1009.500000	0.005022	1
5	2015-02-11 14:51:59	21.7900	31.133333	437.333333	1005.666667	0.005030	1

In [5]:

```
print(datatraining.info())
datatraining.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8143 entries, 1 to 8143
Data columns (total 7 columns):
date            8143 non-null object
Temperature     8143 non-null float64
Humidity        8143 non-null float64
Light           8143 non-null float64
CO2             8143 non-null float64
HumidityRatio   8143 non-null float64
Occupancy       8143 non-null int64
dtypes: float64(5), int64(1), object(1)
memory usage: 508.9+ KB
None
```

Out [5]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
1	2015-02-04 17:51:00	23.18	27.2720	426.0	721.25	0.004793	1
2	2015-02-04 17:51:59	23.15	27.2675	429.5	714.00	0.004783	1
3	2015-02-04 17:53:00	23.15	27.2450	426.0	713.50	0.004779	1
4	2015-02-04 17:54:00	23.15	27.2000	426.0	708.25	0.004772	1
5	2015-02-04 17:55:00	23.10	27.2000	426.0	704.50	0.004757	1

All text files has seven columns as date, temperature, humidity, light, CO2, humidity ratio and occupancy.

- Temperature in Celsius.
- Relative humidity as a percentage.
- Light measured in lux.
- Carbon dioxide measured in parts per million.
- Humidity ratio, derived from temperature and relative humidity measured in kilograms of water vapor per kilogram of air.
- Occupancy as either 1 for occupied or 0 for not occupied.

For training and testing the models, I will use I will use datatraining(8143 instances) as training, datatest(2665 instances) as validation and datatest2(9752 instances) as test data.

In [6]:

```
datatest['date'] = pd.to_datetime(datatest['date']) datatest2['date']
= pd.to_datetime(datatest2['date'])
```

```
datatraining['date'] = pd.to_datetime(datatraining['date'])
datatest.reset_index(drop=True, inplace=True)
datatest2.reset_index(drop=True, inplace=True)
datatraining.reset_index(drop=True, inplace=True)
```

In [7]:
```
datatraining.describe()
```

Out [7]:

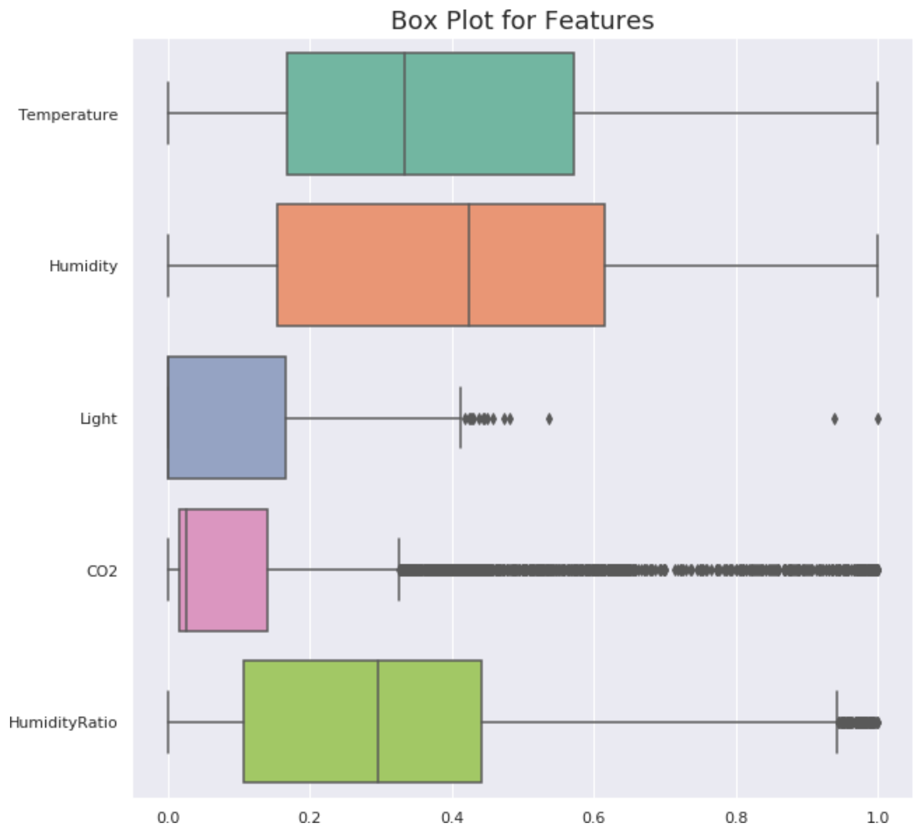
	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy
count	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000	8143.000000
mean	20.619084	25.731507	119.519375	606.546243	0.003863	0.212330
std	1.016916	5.531211	194.755805	314.320877	0.000852	0.408982
min	19.000000	16.745000	0.000000	412.750000	0.002674	0.000000
25%	19.700000	20.200000	0.000000	439.000000	0.003078	0.000000
50%	20.390000	26.222500	0.000000	453.500000	0.003801	0.000000
75%	21.390000	30.533333	256.375000	638.833333	0.004352	0.000000
max	23.180000	39.117500	1546.333333	2028.500000	0.006476	1.000000

Since we have low values like humidity_ratio and high values like light and CO2, we should normalize the data to simplfy the learning process.

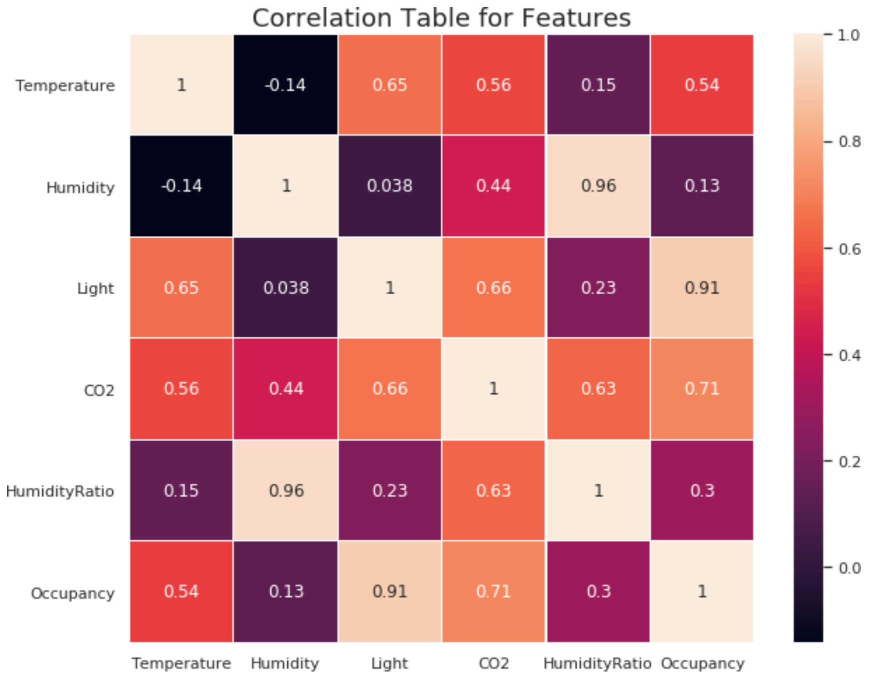
In [8]:
```
scaler = MinMaxScaler()
columns = ['Temperature', 'Humidity', 'Light', 'CO2', 'HumidityRatio']
scaler.fit(np.array(datatraining[columns]))
datatest[columns] = scaler.transform(np.array(datatest[columns]))
datatest2[columns] = scaler.transform(np.array(datatest2[columns]))
datatraining[columns] = scaler.transform(np.array(datatraining[columns]))
```

In [9]:
```
plt.figure(figsize=(10,10))
plt.title('Box Plot for Features', fontdict={'fontsize':18})
ax = sns.boxplot(data=datatraining.drop(['date', 'Occupancy'],axis=1), orient="h", palette="Set2")
print(datatraining.drop(['date', 'Occupancy'],axis=1).describe())
```

```
       Temperature     Humidity        Light          CO2  HumidityRatio
count  8143.000000  8143.000000  8143.000000  8143.000000    8143.000000
mean      0.387341     0.401676     0.077292     0.119942       0.312576
std       0.243281     0.247233     0.125947     0.194536       0.224186
min       0.000000     0.000000     0.000000     0.000000       0.000000
25%       0.167464     0.154431     0.000000     0.016246       0.106304
50%       0.332536     0.423623     0.000000     0.025220       0.296338
75%       0.571770     0.616307     0.165795     0.139925       0.441308
max       1.000000     1.000000     1.000000     1.000000       1.000000
```



In [10]:
```
plt.figure(figsize=(10,8))
plt.title('Correlation Table for Features', fontdict={'fontsize':18}) ax
= sns.heatmap(datatraining.corr(), annot=True, linewidths=.2)
```



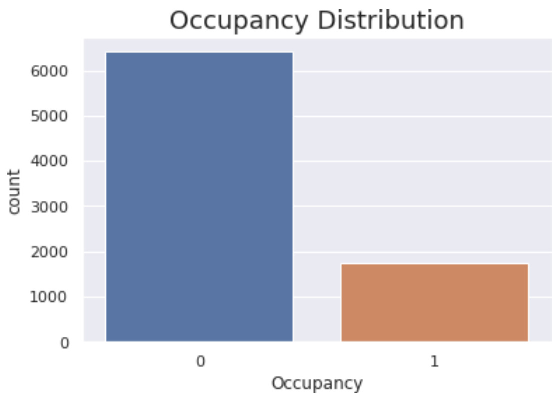
We can see the correlations between occupancy and the others. As I expected, light value is more correlated with occupancy than others.

In [11]:
```
data = datatraining.copy()
data.Occupancy = data.Occupancy.astype(str)
fig = px.scatter_3d(data, x='Temperature', y='Humidity', z='CO2', size='Light', color='Occupancy', c
fig.update_layout(scene_zaxis_type="log", title={'text': "Features and Occupancy",
                                                  'y':0.9,
                                                  'x':0.5,
                                                  'xanchor': 'center',
                                                  'yanchor': 'top'})
iplot(fig)
```

Let's look on the 4-dimensional plot for occupancy. The 4th dimension is size of dots here and I used light value as 4th dimension. The higher light will lead to bigger dots and the lower light will lead to smaller dots. You can use your mouse to change your perspective and take a closer look on the graph.

In [12]:
```
sns.set(style="darkgrid")
plt.title("Occupancy Distribution", fontdict={'fontsize':18})
```

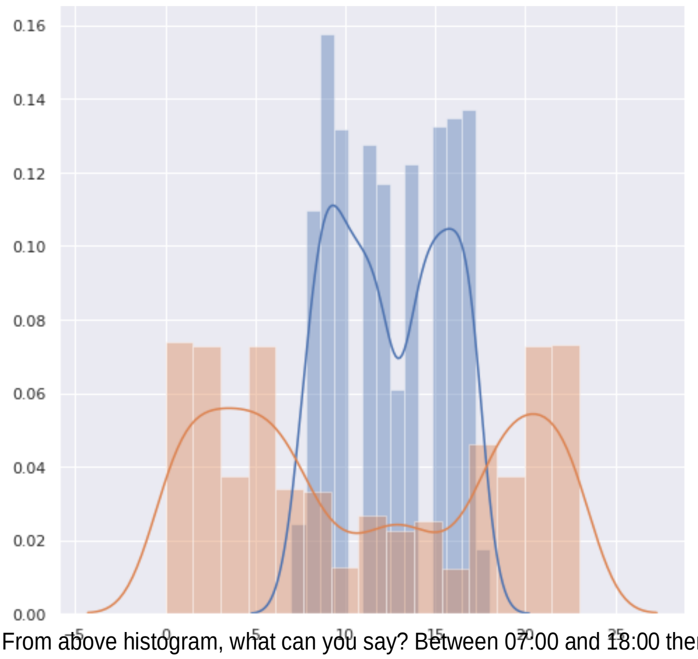
```
ax = sns.countplot(x="Occupancy", data=datatraining)
```



Our data is unbalanced, so we need to find another relations between features to strengthen our predictions. I have a question at this point, is there any relation between occupancy and the hour of the day? Let's look into it.

In [13]:
```
hours_1 = []
hours_0 = []
for date in datatraining[datatraining['Occupancy'] == 1]['date']:
    hours_1.append(date.hour)
for date in datatraining[datatraining['Occupancy'] == 0]['date']:
    hours_0.append(date.hour)
```

In [14]:
```
plt.figure(figsize=(8,8))   ax
= sns.distplot(hours_1) ax =
sns.distplot(hours_0)
```



From above histogram, what can you say? Between 07:00 and 18:00 there are occupants in the environment or not. But the time come to non-working hours, then we can absolutely say that there is no occupant. With this information, I will create a new feature from date column as day period.

07:00 - 18:00 working hour (labeled as 1)
rest of the day non-working hour (labeled as 0)

•

In [15]:
```
datatest['period_of_day'] = [1 if (i.hour >= 7 and i.hour <= 17) else 0 for i in datatest['date']]
datatest2['period_of_day'] = [1 if (i.hour >= 7and i.hour <= 17) else 0 for i in datatest2['date']]
datatraining['period_of_day'] = [1 if (i.hour >= 7 and i.hour <= 17) else 0 for i in datatraining['d
datatraining.sample(10)
```

Out [15]:

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	period_of_day
7300	2015-02-09 19:31:00	0.406699	0.847246	0.000000	0.444530	0.716502	0	0
5788	2015-02-08 18:19:00	0.069378	0.480724	0.000000	0.007582	0.296338	0	0
4860	2015-02-08 02:51:00	0.093301	0.639401	0.000000	0.014080	0.433386	0	0

	date	Temperature	Humidity	Light	CO2	HumidityRatio	Occupancy	period_of_day
7566	2015-02-09 23:57:00	0.332536	0.722092	0.000000	0.055238	0.579288	0	0
6916	2015-02-09 13:07:00	0.598086	0.739524	0.304592	0.521481	0.687086	1	1
3296	2015-02-07 00:47:00	0.239234	0.087384	0.000000	0.012945	0.005829	0	0
5793	2015-02-08 18:23:59	0.069378	0.480724	0.000000	0.008201	0.296338	0	0
660	2015-02-05 04:51:00	0.478469	0.314896	0.000000	0.021198	0.257765	0	0
3300	2015-02-07 00:51:00	0.239234	0.087384	0.000000	0.011914	0.005829	0	0
7577	2015-02-10 00:08:00	0.332536	0.724327	0.000000	0.055856	0.581253	0	0

Classification with Machine Learning Methods

In [16]:

```
X_train = datatraining.drop(columns=['date', 'Occupancy'], axis=1)
y_train = datatraining['Occupancy']
X_validation = datatest.drop(columns=['date', 'Occupancy'], axis=1) y_validation
= datatest['Occupancy']
X_test = datatest2.drop(columns=['date', 'Occupancy'], axis=1) y_test
= datatest2['Occupancy']
```

KNN (K-Nearest Neighbors)

Let's try different hyperparameters on KNN model such as n_neighbors, weights and metrics to find best options.

In [17]:

```
# parameter-tuning for knn n_neighbors_list
= [7,15,45,135] weights_list = ['uniform',
'distance'] metric_list = ['euclidean',
'manhattan'] accuracies = {}
for n in n_neighbors_list:
    for weight in weights_list: for
        metric in metric_list:
            knn_model = KNeighborsClassifier(n_neighbors=n, weights=weight, metric=metric)
            knn_model.fit(X_train, y_train)
            accuracy = knn_model.score(X_validation, y_validation)
            accuracies[str(n)+"/"+weight+"/"+metric] = accuracy
```

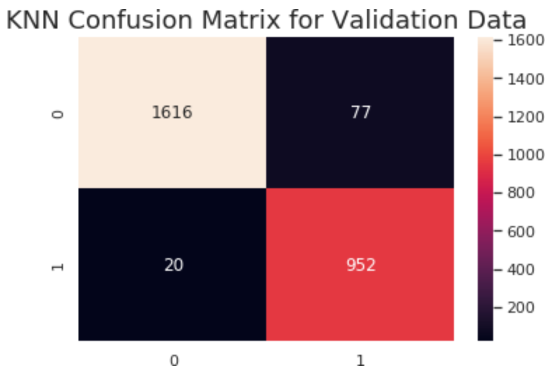
In [18]:

```
plotdata = pd.DataFrame()
plotdata['Parameters'] = accuracies.keys()
plotdata['Accuracy'] = accuracies.values()
fig = px.line(plotdata, x="Parameters", y="Accuracy")
fig.update_layout(title={'text': "Accuracies for Different Hyper-Parameters",
                         'x':0.5,
                         'xanchor': 'center',
                         'yanchor': 'top'})
iplot(fig)
```

By looking over the accuracies graph:

- 135 is enough for k-value.
- Manhattan distance performs better when k has low value. If k value is higher than usual euclidean is the better option.
- Uniform weights are better.

In [19]:
```
knn_model = KNeighborsClassifier(n_neighbors=135)
knn_model.fit(X_train, y_train)
y_pred = knn_model.predict(X_validation)
plt.title("KNN Confusion Matrix for Validation Data", fontdict={'fontsize':18}) ax
= sns.heatmap(confusion_matrix(y_validation, y_pred), annot=True, fmt="d")
```

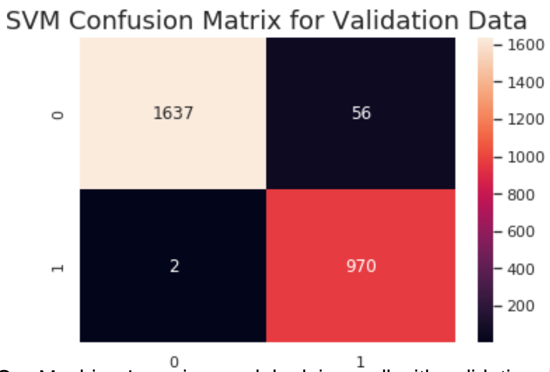


SVM (Support-Vector Machine)

In [20]:
```
svm_model = SVC()
svm_model.fit(X_train, y_train)
print("Accuracy for SVM on validation data: {}%".format(round((svm_model.score(X_validation, y_valid
```

```
Accuracy for SVM on validation data: 97.82%
```

In [21]:
```
y_pred = svm_model.predict(X_validation)
plt.title("SVM Confusion Matrix for Validation Data", fontdict={'fontsize':18}) ax
= sns.heatmap(confusion_matrix(y_validation, y_pred), annot=True, fmt="d")
```



Our Machine Learning models doing well with validation data.

Classification with Neural Networks

Firsty, I would like to try different models like with or without regularization methods. I will create four different models:

1. Without regularization
2. With 0.2 dropout regularization
3. With L1(Lasso) regularization
4. With L2(Ridge) regularization

After all models trained and evaluated with validation data, we will compare the training and validation losses.

In [22]:

```
# NN without regularization
model1 = Sequential()
model1.add(Dense(32, activation='relu', input_dim=6))
model1.add(Dense(16, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
model1.compile(optimizer='rmsprop',
               loss='binary_crossentropy', metrics=['accuracy'])
history1 = model1.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_validation, y_v
```

```
Train on 8143 samples, validate on 2665 samples Epoch
1/50
8143/8143 [==============================] - 1s 78us/step - loss: 0.4086 - accuracy: 0.8341 - val_loss: 0.1916 - val_accuracy: 0.9 Epoch
2/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1133 - accuracy: 0.9678 - val_loss: 0.1106 - val_accuracy: 0.9 Epoch
3/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0750 - accuracy: 0.9778 - val_loss: 0.0976 - val_accuracy: 0.9 Epoch
4/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0632 - accuracy: 0.9829 - val_loss: 0.0900 - val_accuracy: 0.9 Epoch
5/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0578 - accuracy: 0.9845 - val_loss: 0.0880 - val_accuracy: 0.9 Epoch
6/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0534 - accuracy: 0.9858 - val_loss: 0.0867 - val_accuracy: 0.9 Epoch
7/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0502 - accuracy: 0.9854 - val_loss: 0.0855 - val_accuracy: 0.9 Epoch
8/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0485 - accuracy: 0.9856 - val_loss: 0.0791 - val_accuracy: 0.9 Epoch
9/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0475 - accuracy: 0.9871 - val_loss: 0.0789 - val_accuracy: 0.9 Epoch
10/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0467 - accuracy: 0.9866 - val_loss: 0.0788 - val_accuracy: 0.9 Epoch
11/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0454 - accuracy: 0.9871 - val_loss: 0.0816 - val_accuracy: 0.9 Epoch
12/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0447 - accuracy: 0.9870 - val_loss: 0.0793 - val_accuracy: 0.9 Epoch
13/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0447 - accuracy: 0.9874 - val_loss: 0.0787 - val_accuracy: 0.9 Epoch
14/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0442 - accuracy: 0.9870 - val_loss: 0.0822 - val_accuracy: 0.9 Epoch
15/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0439 - accuracy: 0.9870 - val_loss: 0.0786 - val_accuracy: 0.9 Epoch
16/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0436 - accuracy: 0.9874 - val_loss: 0.0802 - val_accuracy: 0.9 Epoch
17/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0433 - accuracy: 0.9872 - val_loss: 0.0790 - val_accuracy: 0.9 Epoch
18/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0432 - accuracy: 0.9875 - val_loss: 0.0791 - val_accuracy: 0.9 Epoch
19/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0428 - accuracy: 0.9871 - val_loss: 0.0788 - val_accuracy: 0.9 Epoch
20/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0422 - accuracy: 0.9875 - val_loss: 0.0778 - val_accuracy: 0.9 Epoch
21/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0421 - accuracy: 0.9876 - val_loss: 0.0836 - val_accuracy: 0.9 Epoch
22/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0423 - accuracy: 0.9875 - val_loss: 0.0779 - val_accuracy: 0.9 Epoch
23/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0416 - accuracy: 0.9877 - val_loss: 0.0810 - val_accuracy: 0.9 Epoch
24/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0413 - accuracy: 0.9874 - val_loss: 0.0781 - val_accuracy: 0.9 Epoch
25/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0414 - accuracy: 0.9876 - val_loss: 0.0934 - val_accuracy: 0.9 Epoch
26/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0415 - accuracy: 0.9874 - val_loss: 0.0844 - val_accuracy: 0.9 Epoch
27/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0411 - accuracy: 0.9871 - val_loss: 0.0851 - val_accuracy: 0.9 Epoch
28/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0412 - accuracy: 0.9876 - val_loss: 0.0789 - val_accuracy: 0.9 Epoch
29/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0408 - accuracy: 0.9872 - val_loss: 0.0826 - val_accuracy: 0.9 Epoch
30/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0406 - accuracy: 0.9871 - val_loss: 0.0798 - val_accuracy: 0.9 Epoch
31/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0405 - accuracy: 0.9872 - val_loss: 0.0794 - val_accuracy: 0.9 Epoch
32/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0406 - accuracy: 0.9875 - val_loss: 0.0804 - val_accuracy: 0.9 Epoch
33/50
8143/8143 [==============================] - 0s 49us/step - loss: 0.0401 - accuracy: 0.9872 - val_loss: 0.0813 - val_accuracy: 0.9 Epoch
34/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0397 - accuracy: 0.9875 - val_loss: 0.0800 - val_accuracy: 0.9 Epoch
35/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0399 - accuracy: 0.9875 - val_loss: 0.0803 - val_accuracy: 0.9 Epoch
36/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0400 - accuracy: 0.9876 - val_loss: 0.0872 - val_accuracy: 0.9 Epoch
37/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0395 - accuracy: 0.9870 - val_loss: 0.0811 - val_accuracy: 0.9 Epoch
38/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0397 - accuracy: 0.9871 - val_loss: 0.0967 - val_accuracy: 0.9 Epoch
39/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0395 - accuracy: 0.9875 - val_loss: 0.0878 - val_accuracy: 0.9 Epoch
40/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0396 - accuracy: 0.9871 - val_loss: 0.0893 - val_accuracy: 0.9 Epoch
41/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0395 - accuracy: 0.9867 - val_loss: 0.0815 - val_accuracy: 0.9 Epoch
42/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0392 - accuracy: 0.9870 - val_loss: 0.0872 - val_accuracy: 0.9 Epoch
43/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0394 - accuracy: 0.9870 - val_loss: 0.0856 - val_accuracy: 0.9 Epoch
44/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0388 - accuracy: 0.9869 - val_loss: 0.0880 - val_accuracy: 0.9 Epoch
45/50
```

8143/8143 [==============================] - 0s 39us/step - loss: 0.0390 - accuracy: 0.9866 - val_loss: 0.0880 - val_accuracy: 0.9
Epoch 46/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0387 - accuracy: 0.9869 - val_loss: 0.0893 - val_accuracy: 0.9 Epoch 47/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0385 - accuracy: 0.9872 - val_loss: 0.0902 - val_accuracy: 0.9 Epoch 48/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0386 - accuracy: 0.9867 - val_loss: 0.0925 - val_accuracy: 0.9 Epoch 49/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0386 - accuracy: 0.9867 - val_loss: 0.1148 - val_accuracy: 0.9 Epoch 50/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0384 - accuracy: 0.9869 - val_loss: 0.1057 - val_accuracy: 0.9

In [23]:

```
# NN with 0.2 dropout ratio before the hidden layer.
model2 = Sequential()
model2.add(Dense(32, activation='relu', input_dim=6))
model2.add(Dropout(0.2))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))
model2.compile(optimizer='rmsprop',
               loss='binary_crossentropy',
               metrics=['accuracy'])
history2 = model2.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_validation, y_v
```

Train on 8143 samples, validate on 2665 samples Epoch 1/50
8143/8143 [==============================] - 1s 67us/step - loss: 0.3524 - accuracy: 0.8768 - val_loss: 0.1550 - val_accuracy: 0.9 Epoch 2/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1133 - accuracy: 0.9665 - val_loss: 0.0980 - val_accuracy: 0.9 Epoch 3/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0762 - accuracy: 0.9775 - val_loss: 0.0922 - val_accuracy: 0.9 Epoch 4/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0681 - accuracy: 0.9813 - val_loss: 0.0835 - val_accuracy: 0.9 Epoch 5/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0588 - accuracy: 0.9838 - val_loss: 0.0786 - val_accuracy: 0.9 Epoch 6/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0575 - accuracy: 0.9839 - val_loss: 0.0774 - val_accuracy: 0.9 Epoch 7/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0530 - accuracy: 0.9858 - val_loss: 0.0764 - val_accuracy: 0.9 Epoch 8/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0509 - accuracy: 0.9867 - val_loss: 0.0783 - val_accuracy: 0.9 Epoch 9/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0504 - accuracy: 0.9870 - val_loss: 0.0739 - val_accuracy: 0.9 Epoch 10/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0497 - accuracy: 0.9870 - val_loss: 0.0734 - val_accuracy: 0.9 Epoch 11/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0488 - accuracy: 0.9875 - val_loss: 0.0773 - val_accuracy: 0.9 Epoch 12/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0481 - accuracy: 0.9874 - val_loss: 0.0752 - val_accuracy: 0.9 Epoch 13/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0473 - accuracy: 0.9880 - val_loss: 0.0750 - val_accuracy: 0.9 Epoch 14/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0477 - accuracy: 0.9871 - val_loss: 0.0751 - val_accuracy: 0.9 Epoch 15/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0455 - accuracy: 0.9878 - val_loss: 0.0750 - val_accuracy: 0.9 Epoch 16/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0446 - accuracy: 0.9880 - val_loss: 0.0835 - val_accuracy: 0.9 Epoch 17/50
8143/8143 [==============================] - 0s 49us/step - loss: 0.0445 - accuracy: 0.9878 - val_loss: 0.0775 - val_accuracy: 0.9 Epoch 18/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0446 - accuracy: 0.9874 - val_loss: 0.0764 - val_accuracy: 0.9 Epoch 19/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0442 - accuracy: 0.9878 - val_loss: 0.0765 - val_accuracy: 0.9 Epoch 20/50
8143/8143 [==============================] - 0s 51us/step - loss: 0.0451 - accuracy: 0.9876 - val_loss: 0.0767 - val_accuracy: 0.9 Epoch 21/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0435 - accuracy: 0.9877 - val_loss: 0.0769 - val_accuracy: 0.9 Epoch 22/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0438 - accuracy: 0.9874 - val_loss: 0.0770 - val_accuracy: 0.9 Epoch 23/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0451 - accuracy: 0.9874 - val_loss: 0.0812 - val_accuracy: 0.9 Epoch 24/50
8143/8143 [==============================] - 0s 56us/step - loss: 0.0429 - accuracy: 0.9882 - val_loss: 0.0785 - val_accuracy: 0.9 Epoch 25/50
8143/8143 [==============================] - 0s 49us/step - loss: 0.0433 - accuracy: 0.9878 - val_loss: 0.0782 - val_accuracy: 0.9 Epoch 26/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0441 - accuracy: 0.9878 - val_loss: 0.0792 - val_accuracy: 0.9 Epoch 27/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0425 - accuracy: 0.9876 - val_loss: 0.0783 - val_accuracy: 0.9 Epoch 28/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0424 - accuracy: 0.9878 - val_loss: 0.0792 - val_accuracy: 0.9 Epoch 29/50
8143/8143 [==============================] - 0s 47us/step - loss: 0.0433 - accuracy: 0.9881 - val_loss: 0.0786 - val_accuracy: 0.9 Epoch 30/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0423 - accuracy: 0.9877 - val_loss: 0.0796 - val_accuracy: 0.9 Epoch 31/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0428 - accuracy: 0.9881 - val_loss: 0.0801 - val_accuracy: 0.9 Epoch 32/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0426 - accuracy: 0.9877 - val_loss: 0.0786 - val_accuracy: 0.9 Epoch 33/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0413 - accuracy: 0.9875 - val_loss: 0.0812 - val_accuracy: 0.9 Epoch 34/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0420 - accuracy: 0.9880 - val_loss: 0.0835 - val_accuracy: 0.9 Epoch 35/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0409 - accuracy: 0.9885 - val_loss: 0.0853 - val_accuracy: 0.9 Epoch 36/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0411 - accuracy: 0.9875 - val_loss: 0.0854 - val_accuracy: 0.9 Epoch 37/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0399 - accuracy: 0.9881 - val_loss: 0.0823 - val_accuracy: 0.9 Epoch 38/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0398 - accuracy: 0.9878 - val_loss: 0.0883 - val_accuracy: 0.9

Epoch 39/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0385 - accuracy: 0.9882 - val_loss: 0.0867 - val_accuracy: 0.9 Epoch 40/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0386 - accuracy: 0.9881 - val_loss: 0.0841 - val_accuracy: 0.9 Epoch 41/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0401 - accuracy: 0.9877 - val_loss: 0.0873 - val_accuracy: 0.9 Epoch 42/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0400 - accuracy: 0.9881 - val_loss: 0.0955 - val_accuracy: 0.9 Epoch 43/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0384 - accuracy: 0.9880 - val_loss: 0.0911 - val_accuracy: 0.9 Epoch 44/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0379 - accuracy: 0.9881 - val_loss: 0.0929 - val_accuracy: 0.9 Epoch 45/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0358 - accuracy: 0.9882 - val_loss: 0.0969 - val_accuracy: 0.9 Epoch 46/50
8143/8143 [==============================] - 0s 46us/step - loss: 0.0366 - accuracy: 0.9886 - val_loss: 0.0963 - val_accuracy: 0.9 Epoch 47/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0358 - accuracy: 0.9883 - val_loss: 0.1006 - val_accuracy: 0.9 Epoch 48/50
8143/8143 [==============================] - 0s 47us/step - loss: 0.0375 - accuracy: 0.9871 - val_loss: 0.0967 - val_accuracy: 0.9 Epoch 49/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0360 - accuracy: 0.9878 - val_loss: 0.0911 - val_accuracy: 0.9 Epoch 50/50
8143/8143 [==============================] - 0s 49us/step - loss: 0.0362 - accuracy: 0.9875 - val_loss: 0.1082 - val_accuracy: 0.9

In [24]:

```
# NN with L1(Lasso) regularization model3
= Sequential()
model3.add(Dense(32, activation='relu', input_dim=6, kernel_regularizer=l1(l=0.01)))
model3.add(Dense(16, activation='relu', kernel_regularizer=l1(l=0.01))) model3.add(Dense(1,
activation='sigmoid'))
model3.compile(optimizer='rmsprop',
               loss='binary_crossentropy',
               metrics=['accuracy'])
history3 = model3.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_validation, y_v
```

Train on 8143 samples, validate on 2665 samples Epoch 1/50
8143/8143 [==============================] - 0s 60us/step - loss: 1.1213 - accuracy: 0.7872 - val_loss: 0.7347 - val_accuracy: 0.6 Epoch 2/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.4839 - accuracy: 0.8287 - val_loss: 0.4557 - val_accuracy: 0.9 Epoch 3/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.3569 - accuracy: 0.9628 - val_loss: 0.3473 - val_accuracy: 0.9 Epoch 4/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.3035 - accuracy: 0.9689 - val_loss: 0.2993 - val_accuracy: 0.9 Epoch 5/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.2706 - accuracy: 0.9718 - val_loss: 0.2758 - val_accuracy: 0.9 Epoch 6/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.2440 - accuracy: 0.9759 - val_loss: 0.2469 - val_accuracy: 0.9 Epoch 7/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.2235 - accuracy: 0.9778 - val_loss: 0.2317 - val_accuracy: 0.9 Epoch 8/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.2083 - accuracy: 0.9805 - val_loss: 0.2240 - val_accuracy: 0.9 Epoch 9/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1976 - accuracy: 0.9817 - val_loss: 0.2125 - val_accuracy: 0.9 Epoch 10/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1888 - accuracy: 0.9844 - val_loss: 0.2033 - val_accuracy: 0.9 Epoch 11/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1807 - accuracy: 0.9855 - val_loss: 0.2010 - val_accuracy: 0.9 Epoch 12/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1741 - accuracy: 0.9860 - val_loss: 0.1902 - val_accuracy: 0.9 Epoch 13/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1676 - accuracy: 0.9865 - val_loss: 0.1843 - val_accuracy: 0.9 Epoch 14/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1619 - accuracy: 0.9869 - val_loss: 0.1792 - val_accuracy: 0.9 Epoch 15/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1570 - accuracy: 0.9872 - val_loss: 0.1732 - val_accuracy: 0.9 Epoch 16/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1527 - accuracy: 0.9877 - val_loss: 0.1693 - val_accuracy: 0.9 Epoch 17/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1494 - accuracy: 0.9875 - val_loss: 0.1701 - val_accuracy: 0.9 Epoch 18/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1465 - accuracy: 0.9880 - val_loss: 0.1632 - val_accuracy: 0.9 Epoch 19/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1437 - accuracy: 0.9880 - val_loss: 0.1612 - val_accuracy: 0.9 Epoch 20/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1413 - accuracy: 0.9880 - val_loss: 0.1588 - val_accuracy: 0.9 Epoch 21/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1397 - accuracy: 0.9878 - val_loss: 0.1574 - val_accuracy: 0.9 Epoch 22/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1378 - accuracy: 0.9878 - val_loss: 0.1550 - val_accuracy: 0.9 Epoch 23/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1362 - accuracy: 0.9880 - val_loss: 0.1543 - val_accuracy: 0.9 Epoch 24/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1345 - accuracy: 0.9882 - val_loss: 0.1524 - val_accuracy: 0.9 Epoch 25/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1331 - accuracy: 0.9881 - val_loss: 0.1505 - val_accuracy: 0.9 Epoch 26/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1316 - accuracy: 0.9881 - val_loss: 0.1495 - val_accuracy: 0.9 Epoch 27/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1297 - accuracy: 0.9883 - val_loss: 0.1505 - val_accuracy: 0.9 Epoch 28/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1280 - accuracy: 0.9880 - val_loss: 0.1454 - val_accuracy: 0.9 Epoch 29/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1258 - accuracy: 0.9877 - val_loss: 0.1432 - val_accuracy: 0.9 Epoch 30/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.1234 - accuracy: 0.9878 - val_loss: 0.1418 - val_accuracy: 0.9 Epoch 31/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1208 - accuracy: 0.9881 - val_loss: 0.1386 - val_accuracy: 0.9 Epoch 32/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1179 - accuracy: 0.9878 - val_loss: 0.1359 - val_accuracy: 0.9 Epoch 33/50

8143/8143 [==============================] - 0s 41us/step - loss: 0.1148 - accuracy: 0.9878 - val_loss: 0.1333 - val_accuracy: 0.9
Epoch 34/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1110 - accuracy: 0.9877 - val_loss: 0.1293 - val_accuracy: 0.9 Epoch 35/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1074 - accuracy: 0.9876 - val_loss: 0.1268 - val_accuracy: 0.9 Epoch 36/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1061 - accuracy: 0.9878 - val_loss: 0.1265 - val_accuracy: 0.9 Epoch 37/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1055 - accuracy: 0.9878 - val_loss: 0.1269 - val_accuracy: 0.9 Epoch 38/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1051 - accuracy: 0.9881 - val_loss: 0.1267 - val_accuracy: 0.9 Epoch 39/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1043 - accuracy: 0.9878 - val_loss: 0.1249 - val_accuracy: 0.9 Epoch 40/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1041 - accuracy: 0.9882 - val_loss: 0.1260 - val_accuracy: 0.9 Epoch 41/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1035 - accuracy: 0.9883 - val_loss: 0.1270 - val_accuracy: 0.9 Epoch 42/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1032 - accuracy: 0.9883 - val_loss: 0.1232 - val_accuracy: 0.9 Epoch 43/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1026 - accuracy: 0.9883 - val_loss: 0.1234 - val_accuracy: 0.9 Epoch 44/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1023 - accuracy: 0.9883 - val_loss: 0.1227 - val_accuracy: 0.9 Epoch 45/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1019 - accuracy: 0.9883 - val_loss: 0.1228 - val_accuracy: 0.9 Epoch 46/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1013 - accuracy: 0.9883 - val_loss: 0.1218 - val_accuracy: 0.9 Epoch 47/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1011 - accuracy: 0.9883 - val_loss: 0.1223 - val_accuracy: 0.9 Epoch 48/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.1007 - accuracy: 0.9883 - val_loss: 0.1222 - val_accuracy: 0.9 Epoch 49/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1003 - accuracy: 0.9883 - val_loss: 0.1217 - val_accuracy: 0.9 Epoch 50/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0999 - accuracy: 0.9883 - val_loss: 0.1215 - val_accuracy: 0.9

In [25]:

```
# NN with L2(Ridge) Regularization model4
= Sequential()
model4.add(Dense(32, activation='relu', input_dim=6, kernel_regularizer=l2(l=0.01)))
model4.add(Dense(16, activation='relu', kernel_regularizer=l2(l=0.01))) model4.add(Dense(1,
activation='sigmoid'))
model4.compile(optimizer='rmsprop',
               loss='binary_crossentropy',
               metrics=['accuracy'])
history4 = model4.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_validation, y_v
```

Train on 8143 samples, validate on 2665 samples Epoch 1/50
8143/8143 [==============================] - 0s 59us/step - loss: 0.5428 - accuracy: 0.9008 - val_loss: 0.3297 - val_accuracy: 0.9 Epoch 2/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.2548 - accuracy: 0.9681 - val_loss: 0.2347 - val_accuracy: 0.9 Epoch 3/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.2003 - accuracy: 0.9735 - val_loss: 0.2033 - val_accuracy: 0.9 Epoch 4/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1788 - accuracy: 0.9754 - val_loss: 0.1844 - val_accuracy: 0.9 Epoch 5/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1656 - accuracy: 0.9776 - val_loss: 0.1730 - val_accuracy: 0.9 Epoch 6/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1556 - accuracy: 0.9795 - val_loss: 0.1678 - val_accuracy: 0.9 Epoch 7/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1479 - accuracy: 0.9810 - val_loss: 0.1588 - val_accuracy: 0.9 Epoch 8/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1421 - accuracy: 0.9821 - val_loss: 0.1562 - val_accuracy: 0.9 Epoch 9/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1369 - accuracy: 0.9824 - val_loss: 0.1526 - val_accuracy: 0.9 Epoch 10/50
8143/8143 [==============================] - 0s 48us/step - loss: 0.1321 - accuracy: 0.9834 - val_loss: 0.1442 - val_accuracy: 0.9 Epoch 11/50
8143/8143 [==============================] - 0s 51us/step - loss: 0.1283 - accuracy: 0.9837 - val_loss: 0.1452 - val_accuracy: 0.9 Epoch 12/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.1247 - accuracy: 0.9834 - val_loss: 0.1373 - val_accuracy: 0.9 Epoch 13/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1218 - accuracy: 0.9834 - val_loss: 0.1372 - val_accuracy: 0.9 Epoch 14/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1194 - accuracy: 0.9848 - val_loss: 0.1329 - val_accuracy: 0.9 Epoch 15/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.1166 - accuracy: 0.9850 - val_loss: 0.1346 - val_accuracy: 0.9 Epoch 16/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1146 - accuracy: 0.9842 - val_loss: 0.1373 - val_accuracy: 0.9 Epoch 17/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1121 - accuracy: 0.9851 - val_loss: 0.1290 - val_accuracy: 0.9 Epoch 18/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1100 - accuracy: 0.9848 - val_loss: 0.1485 - val_accuracy: 0.9 Epoch 19/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1091 - accuracy: 0.9846 - val_loss: 0.1235 - val_accuracy: 0.9 Epoch 20/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1071 - accuracy: 0.9850 - val_loss: 0.1220 - val_accuracy: 0.9 Epoch 21/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1055 - accuracy: 0.9850 - val_loss: 0.1217 - val_accuracy: 0.9 Epoch 22/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1040 - accuracy: 0.9856 - val_loss: 0.1205 - val_accuracy: 0.9 Epoch 23/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.1030 - accuracy: 0.9856 - val_loss: 0.1189 - val_accuracy: 0.9 Epoch 24/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.1018 - accuracy: 0.9853 - val_loss: 0.1191 - val_accuracy: 0.9 Epoch 25/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.1006 - accuracy: 0.9849 - val_loss: 0.1184 - val_accuracy: 0.9 Epoch 26/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0990 - accuracy: 0.9848 - val_loss: 0.1196 - val_accuracy: 0.9 Epoch 27/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0982 - accuracy: 0.9858 - val_loss: 0.1164 - val_accuracy: 0.9

```
Epoch 28/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0971 - accuracy: 0.9859 - val_loss: 0.1141 - val_accuracy: 0.9 Epoch 29/50
8143/8143 [==============================] - 0s 45us/step - loss: 0.0962 - accuracy: 0.9851 - val_loss: 0.1149 - val_accuracy: 0.9 Epoch 30/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0958 - accuracy: 0.9856 - val_loss: 0.1111 - val_accuracy: 0.9 Epoch 31/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0949 - accuracy: 0.9860 - val_loss: 0.1335 - val_accuracy: 0.9 Epoch 32/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0937 - accuracy: 0.9853 - val_loss: 0.1125 - val_accuracy: 0.9 Epoch 33/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0927 - accuracy: 0.9858 - val_loss: 0.1103 - val_accuracy: 0.9 Epoch 34/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0923 - accuracy: 0.9864 - val_loss: 0.1095 - val_accuracy: 0.9 Epoch 35/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0915 - accuracy: 0.9856 - val_loss: 0.1119 - val_accuracy: 0.9 Epoch 36/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0910 - accuracy: 0.9860 - val_loss: 0.1081 - val_accuracy: 0.9 Epoch 37/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0906 - accuracy: 0.9856 - val_loss: 0.1081 - val_accuracy: 0.9 Epoch 38/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0900 - accuracy: 0.9861 - val_loss: 0.1072 - val_accuracy: 0.9 Epoch 39/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0890 - accuracy: 0.9855 - val_loss: 0.1118 - val_accuracy: 0.9 Epoch 40/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0877 - accuracy: 0.9866 - val_loss: 0.1699 - val_accuracy: 0.9 Epoch 41/50
8143/8143 [==============================] - 0s 44us/step - loss: 0.0881 - accuracy: 0.9860 - val_loss: 0.1063 - val_accuracy: 0.9 Epoch 42/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0875 - accuracy: 0.9859 - val_loss: 0.1049 - val_accuracy: 0.9 Epoch 43/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0871 - accuracy: 0.9855 - val_loss: 0.1046 - val_accuracy: 0.9 Epoch 44/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0865 - accuracy: 0.9862 - val_loss: 0.1114 - val_accuracy: 0.9 Epoch 45/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0862 - accuracy: 0.9861 - val_loss: 0.1162 - val_accuracy: 0.9 Epoch 46/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0854 - accuracy: 0.9860 - val_loss: 0.1062 - val_accuracy: 0.9 Epoch 47/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0857 - accuracy: 0.9864 - val_loss: 0.1035 - val_accuracy: 0.9 Epoch 48/50
8143/8143 [==============================] - 0s 42us/step - loss: 0.0841 - accuracy: 0.9865 - val_loss: 0.1045 - val_accuracy: 0.9 Epoch 49/50
8143/8143 [==============================] - 0s 43us/step - loss: 0.0847 - accuracy: 0.9867 - val_loss: 0.1024 - val_accuracy: 0.9 Epoch 50/50
8143/8143 [==============================] - 0s 41us/step - loss: 0.0844 - accuracy: 0.9861 - val_loss: 0.1020 - val_accuracy: 0.9
```

In [26]:

```python
loss1 = history1.history['loss'] val_loss1 =
history1.history['val_loss'] loss2 =
history2.history['loss'] val_loss2 =
history2.history['val_loss'] loss3 =
history3.history['loss'] val_loss3 =
history3.history['val_loss'] loss4 =
history4.history['loss'] val_loss4 =
history4.history['val_loss']


fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(len(loss1)), y=loss1,
                         name='Training Loss without Regularization', line=dict(color='royalblue')))
fig.add_trace(go.Scatter(x=np.arange(len(val_loss1)), y=val_loss1,
                         name='Validation Loss without Regularization', line = dict(color='firebrick')))

fig.add_trace(go.Scatter(x=np.arange(len(loss2)), y=loss2,
                         name='Training Loss with Dropout', line=dict(color='royalblue', dash='dash')))
fig.add_trace(go.Scatter(x=np.arange(len(val_loss2)), y=val_loss2,
                         name='Validation Loss with Dropout', line = dict(color='firebrick', dash='dash')

fig.add_trace(go.Scatter(x=np.arange(len(loss3)), y=loss3,
                         name='Training Loss with L1 Regularization', line=dict(color='royalblue', dash='
fig.add_trace(go.Scatter(x=np.arange(len(val_loss3)), y=val_loss3,
                         name='Validation Loss with L1 Regularization', line = dict(color='firebrick', da

fig.add_trace(go.Scatter(x=np.arange(len(loss4)), y=loss4,
                         name='Training Loss with L2 Regularization', line=dict(color='royalblue', dash='
fig.add_trace(go.Scatter(x=np.arange(len(val_loss4)), y=val_loss4,
                         name='Validation Loss with L2 Regularization', line = dict(color='firebrick', da


fig.update_layout(xaxis_title='Epochs',
                  yaxis_title='Loss',
                  title={'text': "Training and Validation Losses for Different Models", 'x':0.5,
                         'xanchor': 'center',
```

```
                                              'yanchor': 'top'})
iplot(fig)
```

- NN without regularization is unstabilized as expected.
- Dropout and L2 regularization doing well.
- L1 regularization is stable but it has biggest loss value.

So our best option will be a dropout layer and L2 regularization on layers. Let's train it.

P.S. You can click on the legend to close some of lines. It might be useful when examining the plot.

In [27]:
```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=6, kernel_regularizer=l2(l=0.01)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(l=0.01)))
model.add(Dense(1, activation='sigmoid')) model.compile(optimizer='rmsprop',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
history = model.fit(X_train, y_train, epochs=50, batch_size=32)
```

```
Epoch 1/50
8143/8143 [==============================] - 0s 55us/step - loss: 0.5752 - accuracy: 0.8648
Epoch 2/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.2502 - accuracy: 0.9566
Epoch 3/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.2040 - accuracy: 0.9643
Epoch 4/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1783 - accuracy: 0.9689
Epoch 5/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1618 - accuracy: 0.9756
Epoch 6/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1540 - accuracy: 0.9754
Epoch 7/50
8143/8143 [==============================] - 0s 35us/step - loss: 0.1451 - accuracy: 0.9794
Epoch 8/50
8143/8143 [==============================] - 0s 34us/step - loss: 0.1427 - accuracy: 0.9800
Epoch 9/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1335 - accuracy: 0.9805
Epoch 10/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1307 - accuracy: 0.9812
Epoch 11/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1268 - accuracy: 0.9813
Epoch 12/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.1246 - accuracy: 0.9832
Epoch 13/50
8143/8143 [==============================] - 0s 35us/step - loss: 0.1210 - accuracy: 0.9831
Epoch 14/50
8143/8143 [==============================] - 0s 35us/step - loss: 0.1189 - accuracy: 0.9834
Epoch 15/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1157 - accuracy: 0.9835
Epoch 16/50
8143/8143 [==============================] - 0s 35us/step - loss: 0.1151 - accuracy: 0.9832
Epoch 17/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1132 - accuracy: 0.9834
Epoch 18/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1106 - accuracy: 0.9838
Epoch 19/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1094 - accuracy: 0.9833
Epoch 20/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1079 - accuracy: 0.9835
```

```
Epoch 21/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.1084 - accuracy: 0.9844
Epoch 22/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1056 - accuracy: 0.9839
Epoch 23/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.1050 - accuracy: 0.9842
Epoch 24/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1035 - accuracy: 0.9845
Epoch 25/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1030 - accuracy: 0.9846
Epoch 26/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.1013 - accuracy: 0.9843
Epoch 27/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.1004 - accuracy: 0.9853
Epoch 28/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.1000 - accuracy: 0.9845
Epoch 29/50
8143/8143 [==============================] - 0s 35us/step - loss: 0.0995 - accuracy: 0.9851
Epoch 30/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0976 - accuracy: 0.9849
Epoch 31/50
8143/8143 [==============================] - 0s 38us/step - loss: 0.0981 - accuracy: 0.9850
Epoch 32/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0977 - accuracy: 0.9854
Epoch 33/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0949 - accuracy: 0.9853
Epoch 34/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0960 - accuracy: 0.9849
Epoch 35/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0961 - accuracy: 0.9849
Epoch 36/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0948 - accuracy: 0.9854
Epoch 37/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0938 - accuracy: 0.9853
Epoch 38/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0921 - accuracy: 0.9850
Epoch 39/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0920 - accuracy: 0.9855
Epoch 40/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0911 - accuracy: 0.9849
Epoch 41/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0902 - accuracy: 0.9849
Epoch 42/50
8143/8143 [==============================] - 0s 40us/step - loss: 0.0911 - accuracy: 0.9848
Epoch 43/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0899 - accuracy: 0.9855
Epoch 44/50
8143/8143 [==============================] - 0s 38us/step - loss: 0.0914 - accuracy: 0.9853
Epoch 45/50
8143/8143 [==============================] - 0s 37us/step - loss: 0.0882 - accuracy: 0.9858
Epoch 46/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0898 - accuracy: 0.9850
Epoch 47/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0897 - accuracy: 0.9859
Epoch 48/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0889 - accuracy: 0.9854
Epoch 49/50
8143/8143 [==============================] - 0s 36us/step - loss: 0.0889 - accuracy: 0.9860
Epoch 50/50
8143/8143 [==============================] - 0s 38us/step - loss: 0.0889 - accuracy: 0.9851
```

Comparing Performances of SVM and Neural Network

Let's test our models with the test data. This data has nearly 10000 instances. I will evaluate them with accuracy metric first, after then we will look into confusion matrix.

In [28]:

```python
print("Accuracy for SVM on test data: {}%\n".format(round((svm_model.score(X_test, y_test)*100),2)))
print("Accuracy for Neural Network model on test data: {}%".format(round((model.evaluate(X_test, y_t
```

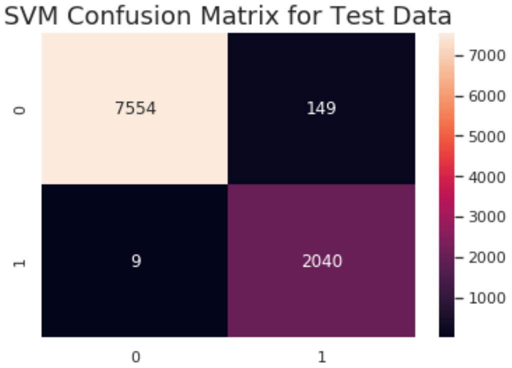
```
Accuracy for SVM on test data: 98.38%

9752/9752 [==============================] - 0s 21us/step
Accuracy for Neural Network model on test data: 97.93%
```

Seems very close right?

In [29]:

```python
y_pred = svm_model.predict(X_test)
plt.title("SVM Confusion Matrix for Test Data", fontdict={'fontsize':18}) ax
= sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d")
```



In [30]:
```
y_pred = model.predict(X_test)
threshold = 0.6
y_pred = [1 if i >= threshold else 0 for i in y_pred]
plt.title("Neural Network Confusion Matrix for Test Data", fontdict={'fontsize':18}) ax
= sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d")
```