In [ ]:
```python
import pandas as pd
import numpy as np
import altair as alt

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
```

In [ ]:
```python
df = pd.read_csv("./assets/diabetes.csv")
```

In [ ]:
```python
df.head(5)
```

Out[ ]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

In [ ]:
```python
df.describe()
```

Out[ ]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 120.894531 | 69.105469 | 20.536458 | 79.799479 | 31.992578 | 0.471876 | 33.240885 | 0.348958 |
| **std** | 3.369578 | 31.972618 | 19.355807 | 15.952218 | 115.244002 | 7.884160 | 0.331329 | 11.760232 | 0.476951 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.078000 | 21.000000 | 0.000000 |
| **25%** | 1.000000 | 99.000000 | 62.000000 | 0.000000 | 0.000000 | 27.300000 | 0.243750 | 24.000000 | 0.000000 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 30.500000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

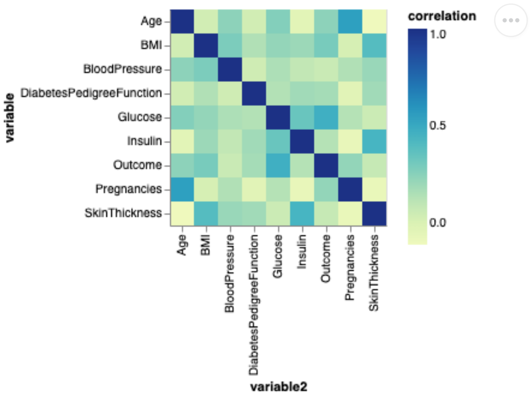
In [ ]:
```python
df.corr("pearson")
```

Out[ ]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **Pregnancies** | 1.000000 | 0.129459 | 0.141282 | -0.081672 | -0.073535 | 0.017683 | -0.033523 | 0.544341 | 0.221898 |
| **Glucose** | 0.129459 | 1.000000 | 0.152590 | 0.057328 | 0.331357 | 0.221071 | 0.137337 | 0.263514 | 0.466581 |
| **BloodPressure** | 0.141282 | 0.152590 | 1.000000 | 0.207371 | 0.088933 | 0.281805 | 0.041265 | 0.239528 | 0.065068 |
| **SkinThickness** | -0.081672 | 0.057328 | 0.207371 | 1.000000 | 0.436783 | 0.392573 | 0.183928 | -0.113970 | 0.074752 |
| **Insulin** | -0.073535 | 0.331357 | 0.088933 | 0.436783 | 1.000000 | 0.197859 | 0.185071 | -0.042163 | 0.130548 |
| **BMI** | 0.017683 | 0.221071 | 0.281805 | 0.392573 | 0.197859 | 1.000000 | 0.140647 | 0.036242 | 0.292695 |
| **DiabetesPedigreeFunction** | -0.033523 | 0.137337 | 0.041265 | 0.183928 | 0.185071 | 0.140647 | 1.000000 | 0.033561 | 0.173844 |
| **Age** | 0.544341 | 0.263514 | 0.239528 | -0.113970 | -0.042163 | 0.036242 | 0.033561 | 1.000000 | 0.238356 |
| **Outcome** | 0.221898 | 0.466581 | 0.065068 | 0.074752 | 0.130548 | 0.292695 | 0.173844 | 0.238356 | 1.000000 |

In [ ]:
```python
cor_data = df.corr("pearson").stack().reset_index().rename(columns={0: 'correlation', 'level_0': 'variable', 'level_1': 'variable2'}


base = alt.Chart(cor_data).encode(
    x='variable2:O',
    y='variable:O'
)

text = base.mark_text().encode(
    text='correlation_label',
    color=alt.condition(
        alt.datum.correlation > 0.5,
        alt.value('white'),
        alt.value('black')
    )
)

cor_plot = base.mark_rect().encode(
    color='correlation:Q'
)

cor_plot
```

Out[ ]:




In [ ]:

```python
df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].mean())
# Correcting missing values in blood pressure
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].mean()) # There are 35 records with 0 BloodPressure in data
# Correcting missing values in BMI
df['BMI'] = df['BMI'].replace(0, df['BMI'].median())
# Correct missing values in Insulin and SkinThickness

df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].median())
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())

df.describe()
```

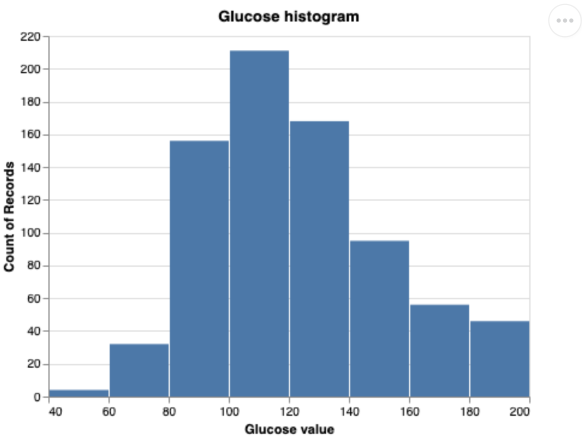
Out[ ]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 |
| **mean** | 3.845052 | 121.681605 | 72.254807 | 27.334635 | 94.652344 | 32.450911 | 0.471876 | 33.240885 | 0.348958 |
| **std** | 3.369578 | 30.436016 | 12.115932 | 9.229014 | 105.547598 | 6.875366 | 0.331329 | 11.760232 | 0.476951 |
| **min** | 0.000000 | 44.000000 | 24.000000 | 7.000000 | 14.000000 | 18.200000 | 0.078000 | 21.000000 | 0.000000 |
| **25%** | 1.000000 | 99.750000 | 64.000000 | 23.000000 | 30.500000 | 27.500000 | 0.243750 | 24.000000 | 0.000000 |
| **50%** | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 31.250000 | 32.000000 | 0.372500 | 29.000000 | 0.000000 |
| **75%** | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | 0.626250 | 41.000000 | 1.000000 |
| **max** | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | 2.420000 | 81.000000 | 1.000000 |

In [ ]:

```python
alt.Chart(df, title="Glucose histogram").mark_bar().encode(
    x= alt.X("Glucose:Q",  bin=True, title="Glucose value"),
    y='count()',
)
```

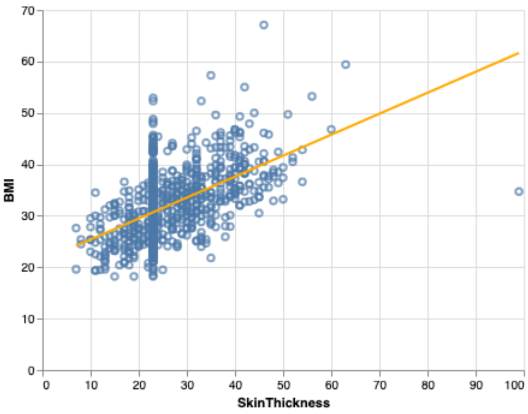
Out[ ]:

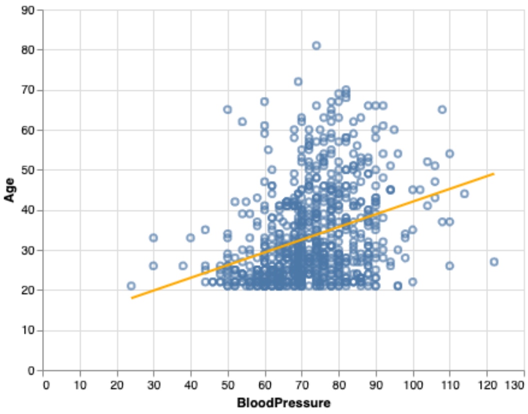


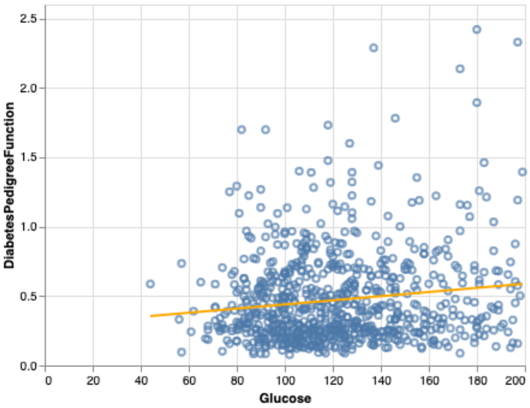

In [ ]:

```python
# Regression SkinThickness/BMI
chart = alt.Chart(df).mark_point().encode(
    x= alt.X("SkinThickness"),
    y= alt.Y("BMI"),
)

(chart + chart.transform_regression('SkinThickness', 'BMI').mark_line().encode(
    color=alt.value("#FFAA00")
)).display()


# Regression BloodPressure/Age
chart2 = alt.Chart(df).mark_point().encode(
    x= alt.X("BloodPressure"),
    y= alt.Y("Age"),
)

(chart2 + chart2.transform_regression('BloodPressure', 'Age').mark_line().encode(
    color=alt.value("#FFAA00")
)).display()

# Regression Glucose/DiabetesPedigreeFunction
chart3 = alt.Chart(df).mark_point().encode(
    x= alt.X("Glucose"),
```

```
    y= alt.Y("DiabetesPedigreeFunction")
)

(chart3 + chart3.transform_regression('Glucose', 'DiabetesPedigreeFunction').mark_line().encode(
    color=alt.value("#FFAA00")
)).display()
```



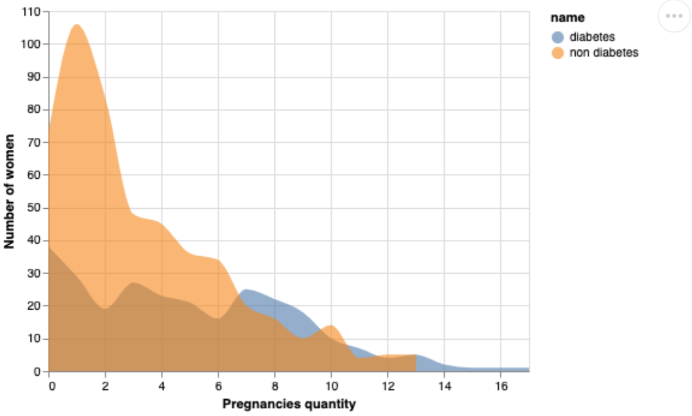




In [ ]:
```
diabetes_women = pd.DataFrame({
    'pregnancies': df["Pregnancies"][df["Outcome"] == 1].value_counts().to_frame().index.to_list(),
    'frequency': df["Pregnancies"][df["Outcome"] == 1].value_counts().values,
    'name': "diabetes"
})

non_diabetes_women = pd.DataFrame({
    'pregnancies': df["Pregnancies"][df["Outcome"] == 0].value_counts().to_frame().index.to_list(),
    'frequency': df["Pregnancies"][df["Outcome"] == 0].value_counts().values,
    'name': "non diabetes"
})

area1 = alt.Chart(diabetes_women).mark_area(
    interpolate='monotone'
).encode(
    alt.X("pregnancies", title="Pregnancies quantity"),
    alt.Y("frequency", title='Number of women'),
    opacity=alt.value(0.6),
    color="name"
)

area2 = alt.Chart(non_diabetes_women).mark_area(
    interpolate='monotone'
).encode(
    alt.X("pregnancies", scale=alt.Scale(zero=False, nice=False), title="Pregnancies quantity"),
    alt.Y("frequency", title='Number of women'),
    opacity=alt.value(0.6),
    color="name"
)
```

```
area1 + area2
```

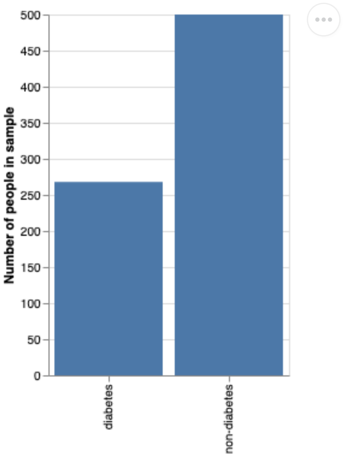
Out[ ]:




In [ ]:

```
source = pd.DataFrame({
    'a': ['diabetes', 'non-diabetes'],
    'b': [df[df["Outcome"] == 1]["Outcome"].count(), df[df["Outcome"] == 0]["Outcome"].count()]
})

alt.Chart(source).mark_bar().encode(
    alt.X("a", title=""),
    alt.Y("b", title='Number of people in sample'),
).properties(
    width=200,
    height=300)
```

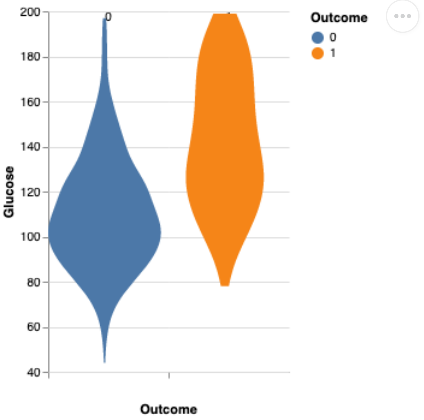
Out[ ]:




In [ ]:

```
alt.Chart(df).transform_density(
    'Glucose',
    as_=['Glucose', 'density'],
    groupby=['Outcome']
).mark_area(orient='horizontal').encode(
    y='Glucose:Q',
    color='Outcome:N',
    x=alt.X(
        'density:Q',
        stack='center',
        impute=None,
        title=None,
        axis=alt.Axis(labels=False, values=[0],grid=False, ticks=True),
    ),
    column=alt.Column(
        'Outcome:N',
        header=alt.Header(
            titleOrient='bottom',
            labelOrient='bottom',
            labelPadding=0,
        ),
    )
).properties(
    width=100
).configure_facet(
    spacing=0
).configure_view(
    stroke=None
)
```

Out[ ]:




In [ ]:
```python
# Data Transformation
q  = QuantileTransformer()
X = q.fit_transform(df)
transformedDF = q.transform(X)
transformedDF = pd.DataFrame(X)
transformedDF.columns =['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'A

transformedDF.head()
```

Out[ ]:

| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.747718 | 0.810300 | 0.516949 | 0.801825 | 0.256193 | 0.591265 | 0.750978 | 0.889831 | 1.0 |
| **1** | 0.232725 | 0.091265 | 0.290091 | 0.644720 | 0.256193 | 0.213168 | 0.475880 | 0.558670 | 0.0 |
| **2** | 0.863755 | 0.956975 | 0.233377 | 0.357888 | 0.256193 | 0.077575 | 0.782269 | 0.585398 | 1.0 |
| **3** | 0.232725 | 0.124511 | 0.290091 | 0.357888 | 0.662973 | 0.284224 | 0.106258 | 0.000000 | 0.0 |
| **4** | 0.000000 | 0.721643 | 0.005215 | 0.801825 | 0.834420 | 0.926988 | 0.997392 | 0.606258 | 1.0 |

In [ ]:
```python
features = transformedDF.drop(["Outcome"], axis=1)
labels = transformedDF["Outcome"]
# train (70%) and test (30%)
x_train, x_test, y_train, y_test = train_test_split(features, labels, test_size=0.30, random_state=7)
print (f'Shape of Train Data : {x_train.shape}')
print (f'Shape of Test Data : {x_test.shape}')
```

```
Shape of Train Data : (537, 8)
Shape of Test Data : (231, 8)
```

Validation data will be also useful to provide an unbiased **evaluation of a model** fit on the **training dataset** while tuning model hyperparameters. Model occasionally sees this data, but never does it *Learn* from this. Validation set is also called dev set.


In [ ]:
```python
x_dev, x_dev_test, y_dev, y_dev_test = train_test_split(x_test, y_test, test_size=0.5, random_state=2)

print (f'Shape of Dev/Validation Data : {x_dev.shape}')
```

```
Shape of Dev/Validation Data : (115, 8)
```

We've prepared training, test and validation data. Question is how many hidden layers and neurons should we use? This is common question and also if we googled internet in appropriate way ... this is impossible to have only one answer. In **Introduction to Neural Networks for Java**, Second Edition by *jeffheaton* we've found defnition:

**Table 5.1: Determining the Number of Hidden Layers**

| Number of Hidden Layers | Result |
|---|---|
| none | Only capable of representing linear separable functions or decisions. |
| 1 | Can approximate any function that contains a continuous mapping from one finite space to another. |
| 2 | Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy. |

Base on our case two hidden layers will be good choice. There is still an issue how many neurons per layer should we use.

1. For the first input layer should be 8 neurons because we have 8 features/columns.
2. Output layer should contain 1 neuron because our result is 1 or 0 (diabetic or not).

Using too few neurons in the hidden layers will result in something called underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in overfitting. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers.

We decided to use this formula:

$$N_h = \frac{N_s}{a * (N_i + N_o)}$$

$Ni$ = number of input neurons.
$No$ = number of output neurons.
$Ns$ = number of samples in training data set.
a = an arbitrary scaling factor usually 2-10.

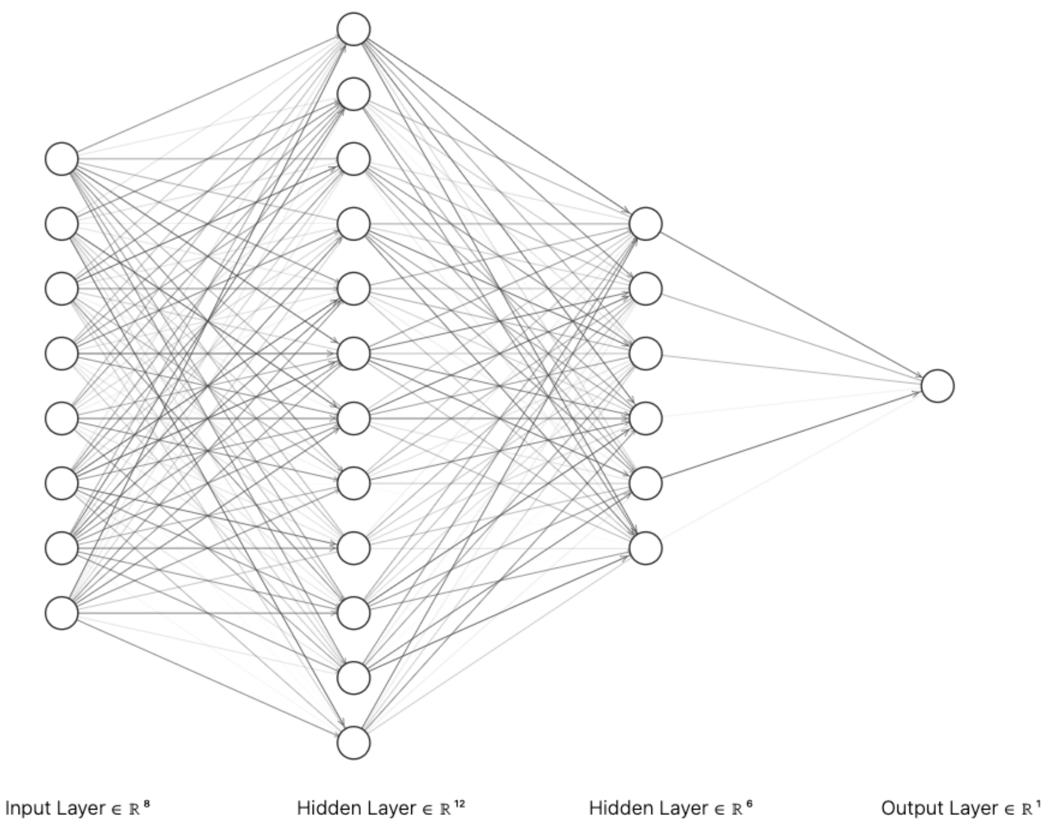
Also with formula above we wanted to follow rules:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer. <1;8>
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer. 8*2/3+1 = 6.(3)
- The number of hidden neurons should be less than twice the size of the input layer. x < 2*8 => x < 16

So we decided:

$$N_{h1} = \frac{537}{5 * (8+1)} = 12$$

$$N_{h1} = \frac{537}{10 * (8+1)} = 6$$




In [ ]:
```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Activation, Dense, Dropout, BatchNormalization, Input
from tensorflow.keras.optimizers import Adam


def create_model_and_fit(epochs):
    print(f"START CREATING MODEL FOR EPOCH={epochs}")
    inputs = Input(name='inputs', shape=[x_train.shape[1],])
    layer = Dense(12, name='FC1')(inputs)
    layer = BatchNormalization(name='BC1')(layer)
    layer = Activation('relu', name='Activation1')(layer)
    layer = Dropout(0.3, name='Dropout1')(layer)
    layer = Dense(6, name='FC2')(layer)
    layer = BatchNormalization(name='BC2')(layer)
    layer = Activation('relu', name='Activation2')(layer)
    layer = Dropout(0.3, name='Dropout2')(layer)
    layer = Dense(1, name='OutLayer')(layer)
```

```python
    layer = Activation('sigmoid', name='sigmoid')(layer)
    model = Model(inputs=inputs, outputs=layer)

    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    historic = model.fit(x=x_train, y=y_train, epochs = epochs, verbose=0)

    loss_df = pd.DataFrame({
        'loss': historic.history["loss"],
        'epoch': [i for i,x in enumerate(historic.history["loss"])],
        "name": "Error"
    })

    accuraccy_df = pd.DataFrame({
        'accuraccy': historic.history["accuracy"],
        'epoch': [i for i,x in enumerate(historic.history["accuracy"])],
        "name": "Accuraccy"
    })


    alt1 = alt.Chart(loss_df, title=f"Binary cross entropy on train dataset, EPOCH = {epochs}").mark_line(interpolate='basis').encod
        x = 'epoch',
        y = alt.Y('loss', title='value', scale=alt.Scale(domain=[min(historic.history["loss"]), max(historic.history["loss"])])),
        color = "name"
    )
    alt2 = alt.Chart(accuraccy_df).mark_line(interpolate='basis').encode(
        x='epoch',
        y = alt.Y('accuraccy', title='value', scale=alt.Scale(domain=[min(historic.history["accuracy"]), max(historic.history["accur
        color = "name"
    )

    (alt1 + alt2).display()

    return model;
```

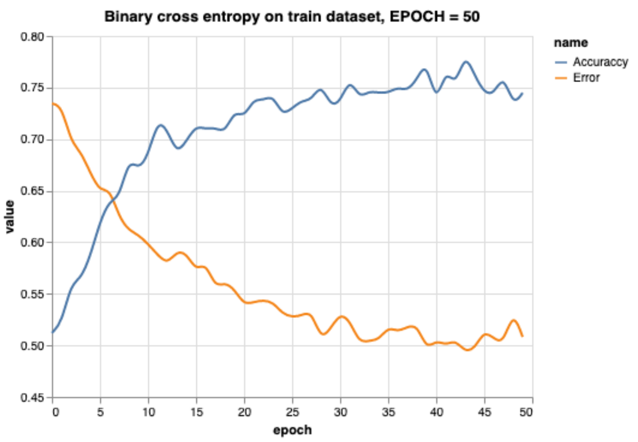
In [ ]:

```python
def show_report(x, y, name, model, epoch):
    print(f"Report for {name} set\n")
    y_prediction = model.predict(x)
    y_prediction = np.around(y_prediction)
    y_prediction = np.asarray(y_prediction)
    print('\tAccuracy:{:0.3f}\n\tClassification Report\n{}'.format(accuracy_score(y, y_prediction),
                                                              classification_report(y, y_prediction)))

    dataResult1 = pd.DataFrame({
    'result': y,
    'name': f"y_{name}",
    'item_number': [i for i, item in enumerate(y)]
    })
    dataResult2 = pd.DataFrame({
        'result':[item[0] for item in y_prediction],
        'name': f"y_{name}_prediction",
        'item_number': [i for i, item in enumerate(y)]
        })

    chart1 = alt.Chart(dataResult1, title = f"Coverage level forecasted data with the original, EPOCH = {epoch}").mark_area(
        interpolate='monotone'
    ).encode(
        x = alt.X('item_number'),
        y = alt.Y('result', scale=alt.Scale(domain=[-0.5, 1.5])),
        color='name'
    ).properties(width=1200)
    chart2 = alt.Chart(dataResult2).mark_area(
        interpolate='monotone'
    ).encode(
        x = alt.X('item_number'),
        y = alt.Y('result', scale=alt.Scale(domain=[-0.5, 1.5])),
        opacity=alt.value(0.6),
        color='name'
    ).properties(width=1200)

    (chart1 + chart2).display()
```

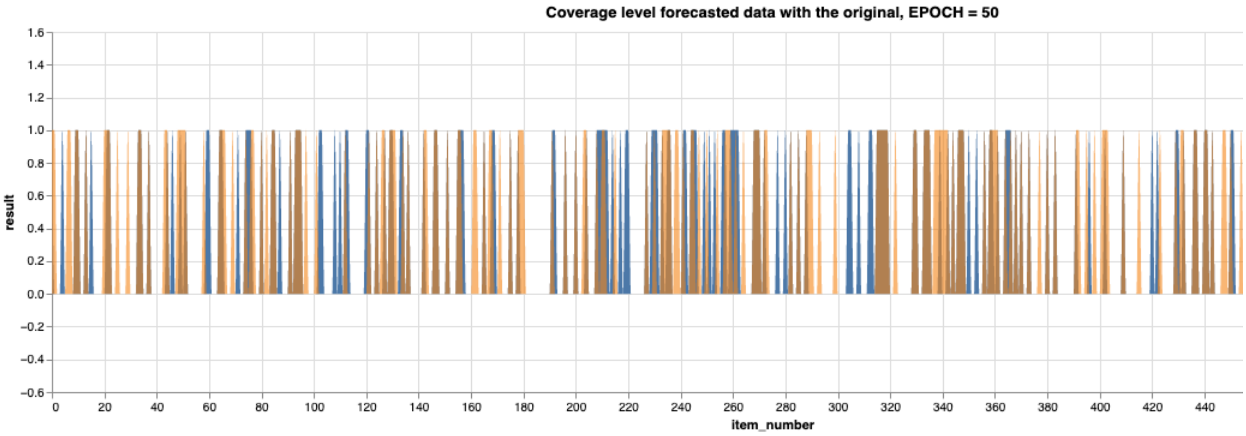
In [ ]:

```python
# Test for different epoch [50, 200, 500]

epoch_case = [50, 200, 500]

for epoch in epoch_case:
    model = create_model_and_fit(epoch)
    show_report(x_train, y_train, "train", model, epoch)
    show_report(x_test, y_test, "test", model, epoch)
```

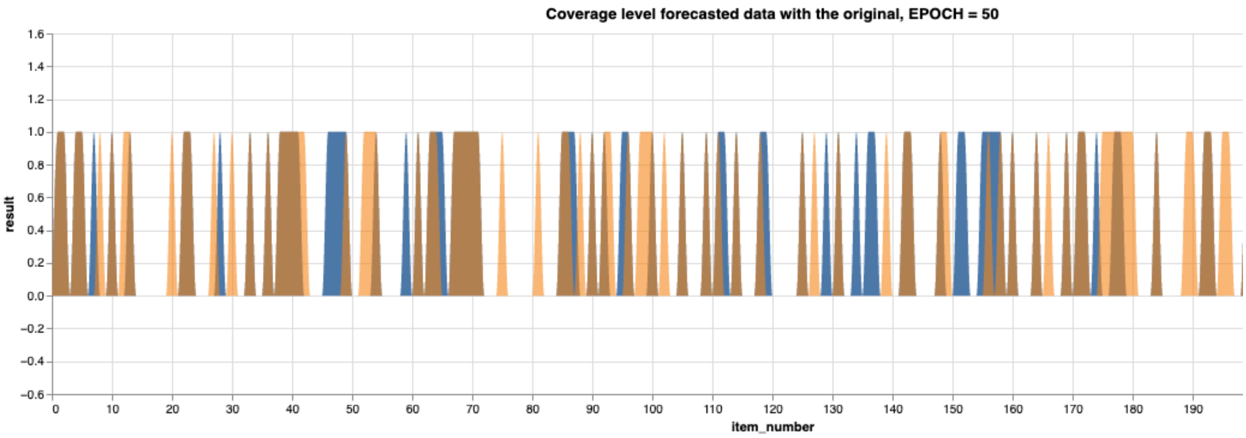
```
START CREATING MODEL FOR EPOCH=50
```



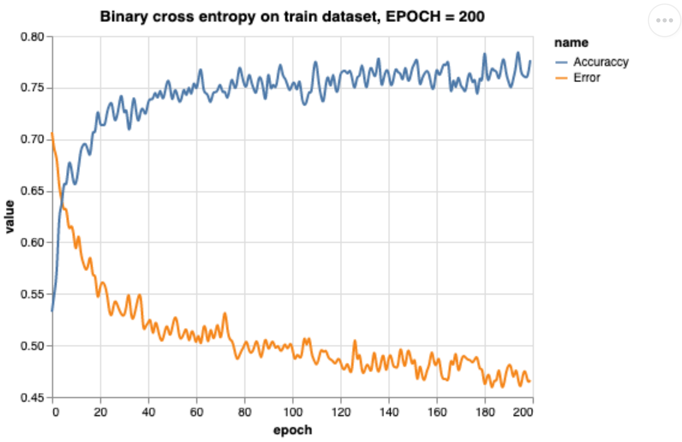

```
Report for train set

17/17 [==============================] - 0s 2ms/step
        Accuracy:0.767
        Classification Report
              precision    recall  f1-score   support

         0.0       0.83      0.81      0.82       353
         1.0       0.65      0.68      0.67       184

    accuracy                           0.77       537
   macro avg       0.74      0.75      0.74       537
weighted avg       0.77      0.77      0.77       537
```



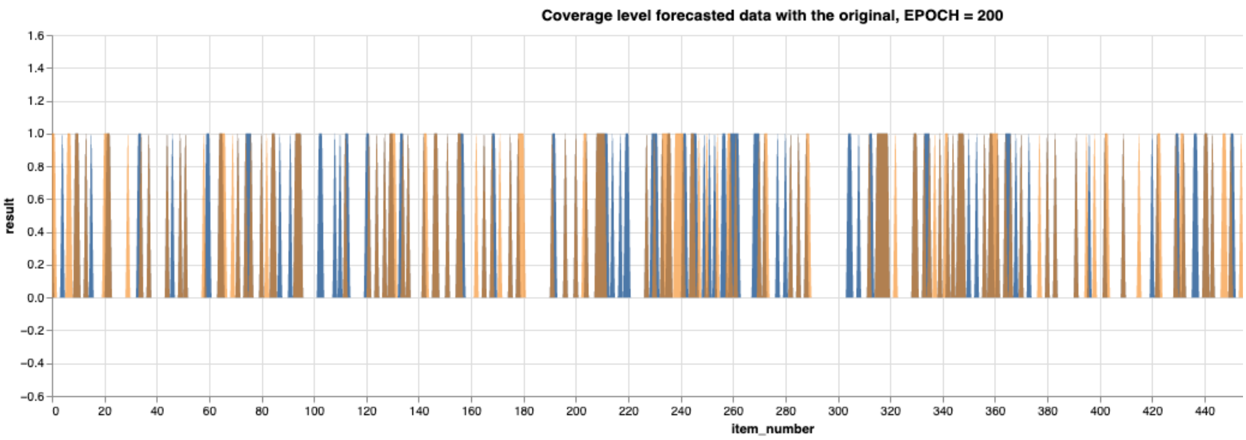

```
Report for test set

8/8 [==============================] - 0s 1ms/step
        Accuracy:0.758
        Classification Report
              precision    recall  f1-score   support

         0.0       0.83      0.78      0.80       147
         1.0       0.65      0.73      0.69        84

    accuracy                           0.76       231
   macro avg       0.74      0.75      0.74       231
weighted avg       0.77      0.76      0.76       231
```



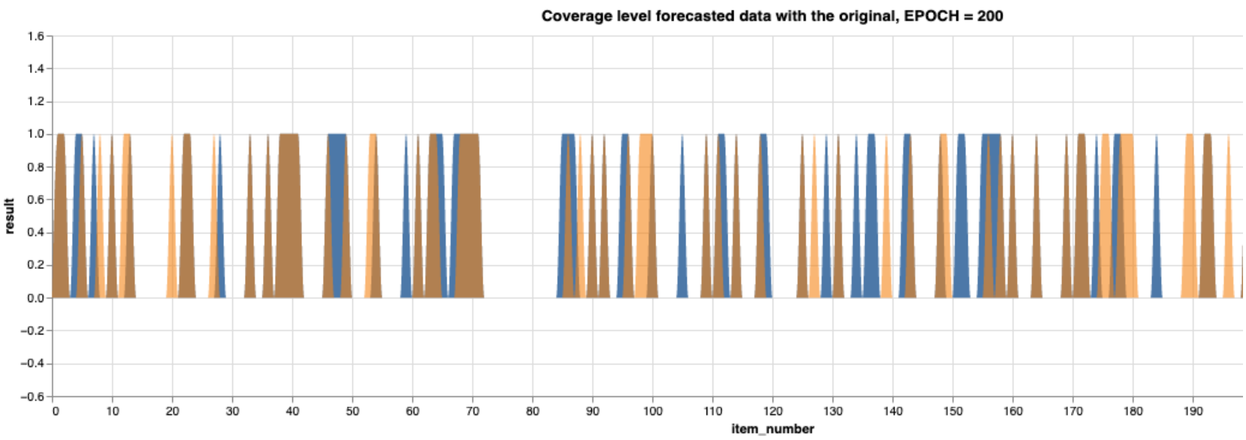

```
START CREATING MODEL FOR EPOCH=200
```



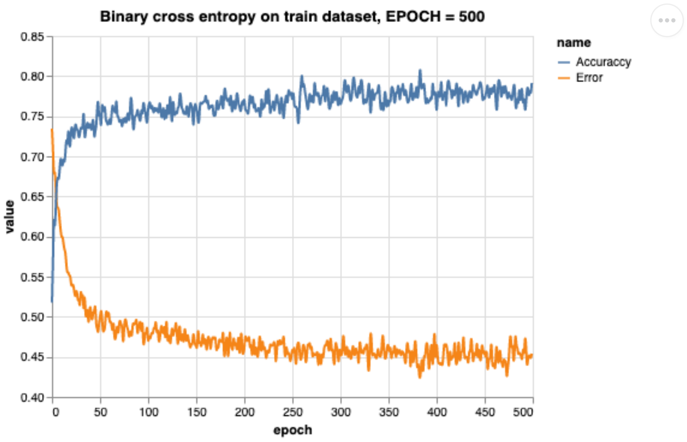

```
Report for train set

17/17 [==============================] - 0s 1ms/step
        Accuracy:0.788
        Classification Report
              precision    recall  f1-score   support

         0.0       0.82      0.87      0.84       353
         1.0       0.71      0.64      0.67       184

    accuracy                           0.79       537
   macro avg       0.77      0.75      0.76       537
weighted avg       0.78      0.79      0.78       537
```




```
Report for test set

8/8 [==============================] - 0s 1ms/step
        Accuracy:0.762
        Classification Report
              precision    recall  f1-score   support

         0.0       0.80      0.84      0.82       147
         1.0       0.69      0.63      0.66        84

    accuracy                           0.76       231
   macro avg       0.74      0.73      0.74       231
weighted avg       0.76      0.76      0.76       231
```

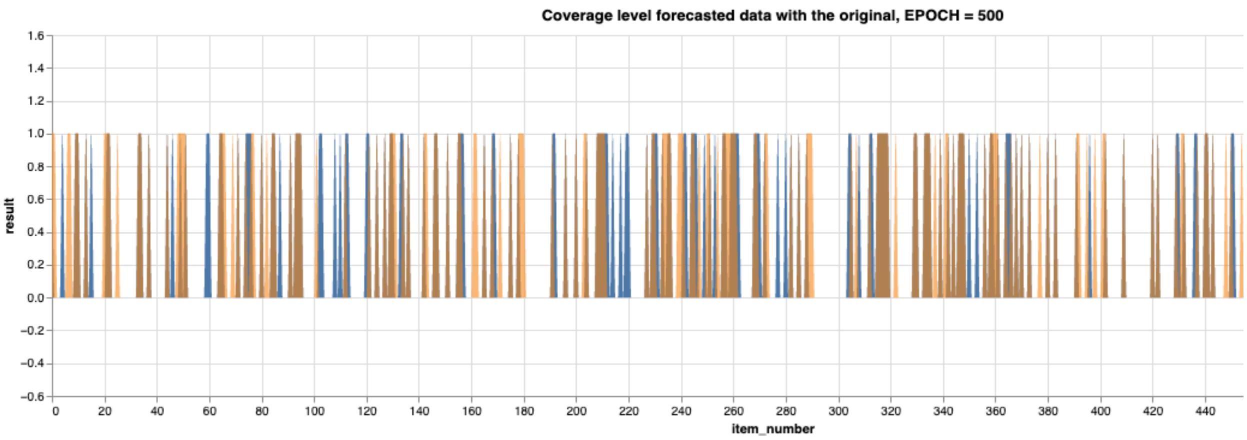



```
START CREATING MODEL FOR EPOCH=500
```




Report for train set

```
17/17 [==============================] - 0s 1ms/step
        Accuracy:0.814
        Classification Report
              precision    recall  f1-score   support

         0.0       0.86      0.86      0.86       353
         1.0       0.73      0.73      0.73       184

    accuracy                           0.81       537
   macro avg       0.79      0.79      0.79       537
weighted avg       0.81      0.81      0.81       537
```




Report for test set

```
8/8 [==============================] - 0s 1ms/step
        Accuracy:0.766
        Classification Report
              precision    recall  f1-score   support

         0.0       0.81      0.82      0.82       147
         1.0       0.68      0.67      0.67        84

    accuracy                           0.77       231
   macro avg       0.75      0.74      0.75       231
weighted avg       0.77      0.77      0.77       231
```