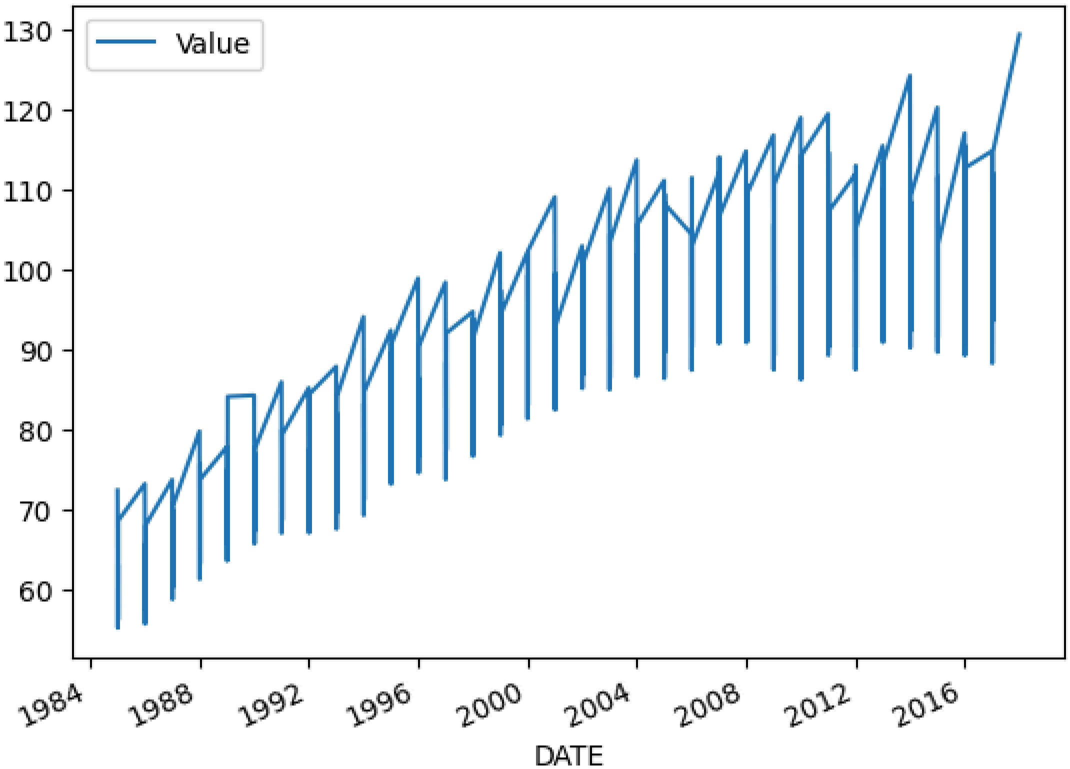
In [1]:
```
import pandas as pd

#Openning the dataset and setting date as index
df = pd.read_csv("Electric_Production.csv")
df = df.set_index("DATE")
df.index = pd.to_datetime(df.index, format='%d-%m-%Y')
# df = df.groupby(pd.Grouper(freq='m')).mean()
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 397 entries, 1985-01-01 to 2018-01-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Value   397 non-null    float64
dtypes: float64(1)
memory usage: 6.2 KB
```

Out[1]:

	Value
DATE	
1985-01-01	72.5052
1985-01-02	70.6720
1985-01-03	62.4502
1985-01-04	57.4714
1985-01-05	55.3151

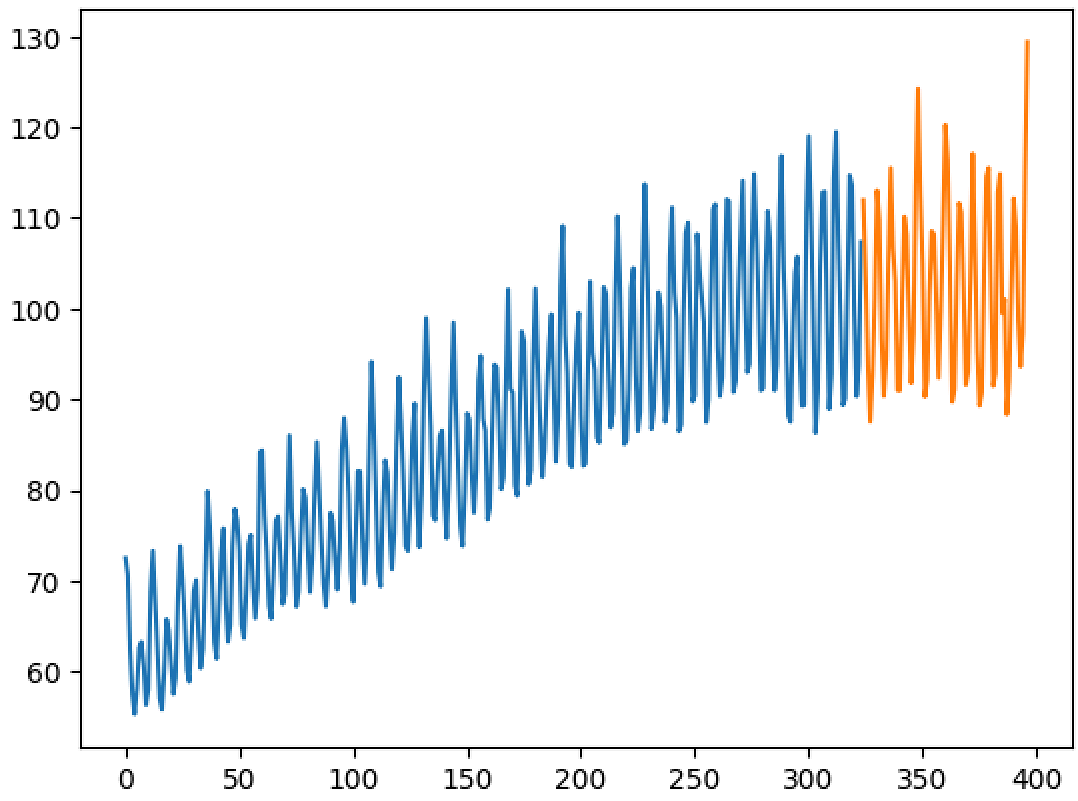
In [2]:
```
import matplotlib.pyplot as plt

df.plot(y='Value', rot=25);
```



In [3]:
```
dftm = df['Value']
print(dftm)
```

```
DATE
1985-01-01     72.5052
1985-01-02     70.6720
1985-01-03     62.4502
1985-01-04     57.4714
1985-01-05     55.3151
                ...
2017-01-09     98.6154
2017-01-10     93.6137
2017-01-11     97.3359
2017-01-12    114.7212
2018-01-01    129.4048
Name: Value, Length: 397, dtype: float64
```

In [4]:
```
import numpy as np

# Spliting the last 2 years for test
train = dftm[:12*27].values
plt.plot(np.arange(len(train)),train)
train = train.reshape((len(train), 1))
test = dftm[12*27:].values
plt.plot(np.arange(len(train), len(train)+len(test)),test)
test = test.reshape((len(test), 1))
#plt.plot(np.arange(len(df3d)),df3d)
```



In [5]:
```
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator

length = 12
generator = TimeseriesGenerator(train,train,length=length, batch_size=1)
validation_generator = TimeseriesGenerator(test,test,length=length, batch_size=1)
```

In [6]:
```
print(train[:length+1])
```

```
[[72.5052]
 [70.672 ]
 [62.4502]
 [57.4714]
 [55.3151]
 [58.0904]
 [62.6202]
 [63.2485]
 [60.5846]
 [56.3154]
 [58.0005]
 [68.7145]
 [73.3057]]
```

In [7]:
```
# Looking some TimeSeriesGenerator results
i=0
for x,y in generator:
    print(x)
    print(y)
    i = i + 1
    if i == 2:
        break
```

```
[[[72.5052]
  [70.672 ]
  [62.4502]
  [57.4714]
  [55.3151]
  [58.0904]
  [62.6202]
  [63.2485]
  [60.5846]
  [56.3154]
  [58.0005]
  [68.7145]]]
[[73.3057]]
[[[70.672 ]
  [62.4502]
  [57.4714]
  [55.3151]
  [58.0904]
  [62.6202]
  [63.2485]
  [60.5846]
  [56.3154]
  [58.0005]
  [68.7145]
  [73.3057]]]
[[67.9869]]
```

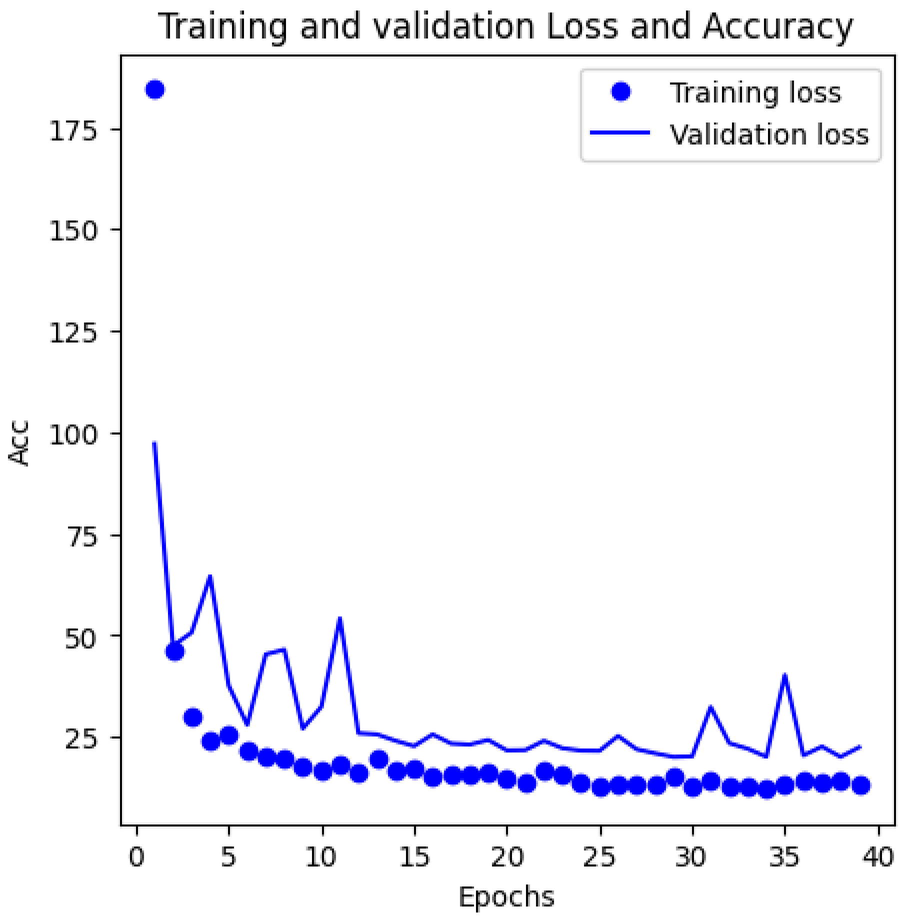
In [8]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense ,SimpleRNN, LSTM, GRU
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

model = Sequential()
model.add(SimpleRNN(10, activation='relu', input_shape=(length,1)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 10)                120

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
epochs = 100
early_stop = EarlyStopping(monitor='val_loss',patience=10)
ckpt = ModelCheckpoint('model6.hdf5', save_best_only=True, monitor='val_loss', verbose=1)
history = model.fit_generator(
    generator,
    steps_per_epoch=len(generator),
    epochs=epochs,
    validation_data=validation_generator,
    callbacks=[early_stop, ckpt])
```

```
Epoch 1/100

C:\Users\Arka Pravo Dutta\AppData\Local\Temp\ipykernel_56488\2702716593.py:4: UserWarning: `Model.fit_gen
erator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports gen
erators.
  history = model.fit_generator(

294/312 [===========================>..] - ETA: 0s - loss: 189.7519
Epoch 1: val_loss improved from inf to 97.13944, saving model to model6.hdf5
312/312 [==============================] - 3s 4ms/step - loss: 184.5096 - val_loss: 97.1394
Epoch 2/100
305/312 [============================>.] - ETA: 0s - loss: 46.7075
Epoch 2: val_loss improved from 97.13944 to 47.45446, saving model to model6.hdf5
312/312 [==============================] - 1s 3ms/step - loss: 46.3268 - val_loss: 47.4545
Epoch 3/100
302/312 [============================>.] - ETA: 0s - loss: 29.8743
Epoch 3: val_loss did not improve from 47.45446
312/312 [==============================] - 1s 3ms/step - loss: 29.7872 - val_loss: 50.6985
Epoch 4/100
300/312 [===========================>..] - ETA: 0s - loss: 24.3747
```

In [10]:
```
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs_x = range(1, len(loss_values) + 1)
plt.figure(figsize=(5,5))
#plt.subplot(2,1,1)
plt.plot(epochs_x, loss_values, 'bo', label='Training loss')
plt.plot(epochs_x, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation Loss and Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
#plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```


Training and validation Loss and Accuracy
Training loss
Validation loss
175
150
125
100
75
50
25
Acc
0
5
10
15
20
25
30
35
40
Epochs

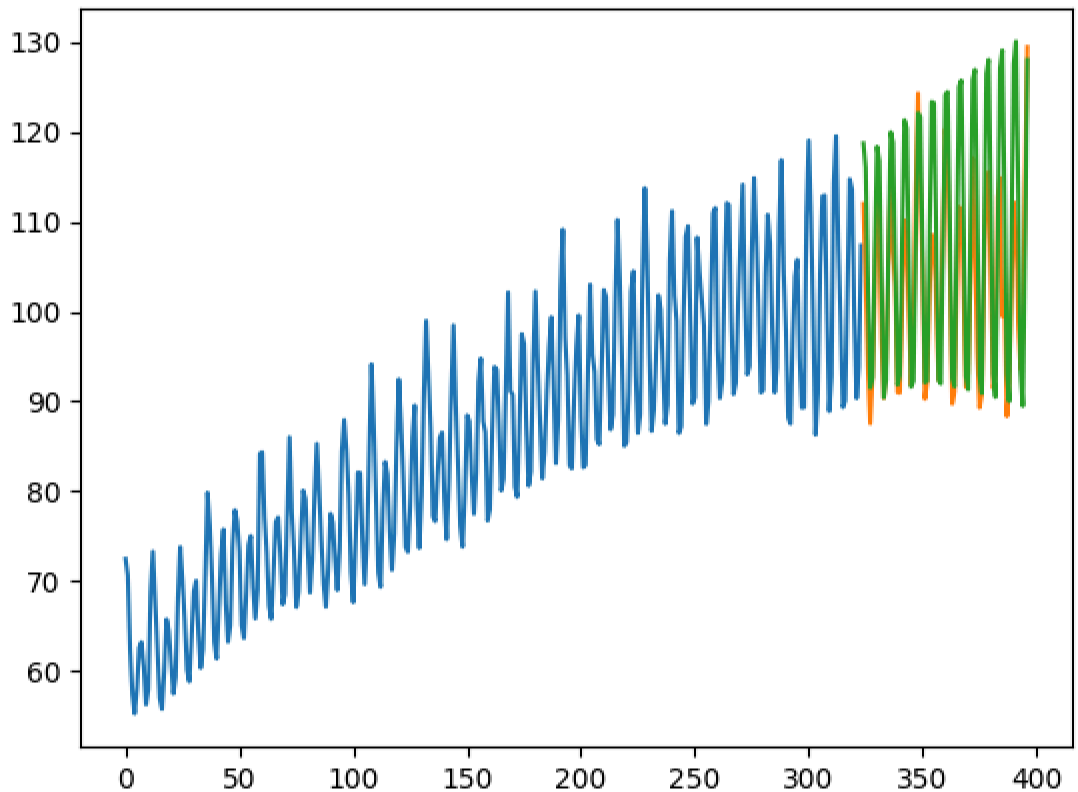
In [11]:
```
model.load_weights("model6.hdf5")

# Predicting some days ahead.
test_predictions = []
first_eval_batch = train[-length:]
current_batch = first_eval_batch.reshape((1, length, 1))
for i in range(len(test)):
    # get prediction 1 time stamp ahead ([0] is for grabbing just the number instead of [array])
    current_pred = model.predict(current_batch)[0]
    # store prediction
    test_predictions.append(current_pred)
    # update batch to now include prediction and drop first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
#prediction = scaler.inverse_transform(test_predictions)
```

```
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 15ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 13ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 12ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 10ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 14ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 16ms/step
```

In [12]:
```
# Comparing test data and predictions
plt.plot(np.arange(len(train)), train)
plt.plot(np.arange(len(train),len(train)+len(test)),test)
plt.plot(np.arange(len(train),len(train)+len(test)),test_predictions)
```

Out[12]:
```
[<matplotlib.lines.Line2D at 0x2488e88d490>]
```



In [13]:
```
# Calculating the mean squared error
loss = np.mean(np.square(test[:,0] - np.array(test_predictions)[:,0]), axis=-1)
print("Root Mean Square Error (RMSE): "+str(loss))
```

```
Root Mean Square Error (RMSE): 79.63954745174931
```

In [ ]: