3_Modeling_V2

May 3, 2021

[84]:
```
import modin.pandas as pd
import numpy as np

import pickle

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
import keras.backend as K
from keras.utils import plot_model

import matplotlib.pyplot as plt
import seaborn as sns

import shap
```

0.1 Loading training data

[2]:
```
training_dat = pd.read_pickle('./PROCESSED/training_dat2.pkl')
training_dat.columns
```

```
UserWarning: Ray execution environment not yet initialized. Initializing…
To remove this warning, run the following python code before doing dataframe
operations:

    import ray
    ray.init()

UserWarning: `read_pickle` defaulting to pandas implementation.
To request implementation, send an email to feature_requests@modin.org.
```

[2]:
```
Index(['Loan Sequence Number', 'Loan Deliquency Within Year', 'Credit Score',
       'MortgageInsuranceFlag', 'Units_1', 'Units_2', 'Units_3', 'Units_4',
       'OccupancyStatus_I', 'OccupancyStatus_P', 'OccupancyStatus_S',
```

```
       'Original Combined Loan-to-Value (CLTV)',
       'Original Debt-to-Income (DTI) Ratio', 'Original UPB',
       'Original Loan-to-Value (LTV)', 'Original Interest Rate', 'Channel_B',
       'Channel_C', 'Channel_R', 'PropertyType_CO', 'PropertyType_CP',
       'PropertyType_MH', 'PropertyType_PU', 'PropertyType_SF',
       'LoanPurpose_C', 'LoanPurpose_N', 'LoanPurpose_P', 'LoanTerm_360',
       'LoanTerm_180', 'LoanTerm_240', 'LoanTerm_120', 'LoanTerm_300',
       'Original Loan Term', 'OneBorrower', 'AffordableProgramFlag'],
      dtype='object')
```

[3]:
```
len(training_dat)
```

[3]:
```
396267
```

[4]:
```
training_dat['Loan Deliquency Within Year'].value_counts()
```

```
UserWarning: value_counts defaulting to pandas implementation.
```

[4]:
```
0.0    387832
1.0      8435
Name: Loan Deliquency Within Year, dtype: int64
```

0.2 Selecting features to feed model

Currently not excluding any columns based on the metrics. I have manually excluded a couple.

[6]:
```
metrics = training_dat.describe().transpose()
metrics.reset_index(inplace=True)
metrics.rename(columns={'index':'column'},inplace=True)
metrics.head()
```

[6]:

	column	count	mean	std	min	25%	50%	75%	max
0	Loan Deliquency Within Year	396267.0	0.021286	0.144337	0.000000	0.000000	0.000000	0.000000	1.000000
1	Credit Score	396267.0	0.000578	0.999393	-5.199338	-0.672129	-0.003764	0.670557	5.199338
2	MortgageInsuranceFlag	396267.0	0.218746	0.413397	0.000000	0.000000	0.000000	0.000000	1.000000
3	Units_1	396267.0	0.980379	0.138693	0.000000	1.000000	1.000000	1.000000	1.000000
4	Units_2	396267.0	0.013582	0.115747	0.000000	0.000000	0.000000	0.000000	1.000000

[7]:
```
tf1 = metrics['mean'] < 1.1
tf2 = metrics['mean'] > -.1
```

```
tf = tf1 & tf2
metrics[tf]
```

[7]:

	column	count	mean	std	min	25%	50%	75%	max
0	Loan Deliquency Within Year	396267.0	0.021286	0.144337	0.000000	0.000000	0.000000	0.000000	1.000000
1	Credit Score	396267.0	0.000578	0.999393	-5.199338	-0.672129	-0.003764	0.670557	5.199338
2	MortgageInsuranceFlag	396267.0	0.218746	0.413397	0.000000	0.000000	0.000000	0.000000	1.000000
3	Units_1	396267.0	0.980379	0.138693	0.000000	1.000000	1.000000	1.000000	1.000000
4	Units_2	396267.0	0.013582	0.115747	0.000000	0.000000	0.000000	0.000000	1.000000
5	Units_3	396267.0	0.003147	0.056009	0.000000	0.000000	0.000000	0.000000	1.000000
6	Units_4	396267.0	0.002892	0.053699	0.000000	0.000000	0.000000	0.000000	1.000000
7	OccupancyStatus_I	396267.0	0.072335	0.259042					
8	OccupancyStatus_P	396267.0	0.886062	0.317737					
9	OccupancyStatus_S	396267.0	0.041603	0.199681					
10	Original Combined Loan-to-Value (CLTV)	396267.0	-0.007684	0.972985					
11	Original Debt-to-Income (DTI) Ratio	396267.0	-0.004331	0.997126					
12	Original UPB	396267.0	0.000987	0.997631					
13	Original Loan-to-Value (LTV)	396267.0	-0.013095	0.972578					
14	Original Interest Rate	396267.0	-0.003350	0.995451					
15	Channel_B	396267.0	0.108051	0.310445					
16	Channel_C	396267.0	0.316756	0.465212					
17	Channel_R	396267.0	0.575193	0.494314					
18	PropertyType_CO	396267.0	0.068686	0.252920					
19	PropertyType_CP	396267.0	0.001340	0.036582					
20	PropertyType_MH	396267.0	0.003455	0.058676					
21	PropertyType_PU	396267.0	0.253067	0.434769					
22	PropertyType_SF	396267.0	0.673452	0.468951					
23	LoanPurpose_C	396267.0	0.240255	0.427239					
24	LoanPurpose_N	396267.0	0.288422	0.453029					
25	LoanPurpose_P	396267.0	0.471324	0.499178					
26	LoanTerm_360	396267.0	0.737490	0.439999					
27	LoanTerm_180	396267.0	0.187578	0.390375					
28	LoanTerm_240	396267.0	0.047826	0.213399					
29	LoanTerm_120	396267.0	0.015727	0.124417					
30	LoanTerm_300	396267.0	0.006127	0.078036					
31	Original Loan Term	396267.0	-0.064882	0.698888					
32	OneBorrower	396267.0	0.461608	0.498525					
33	AffordableProgramFlag	396267.0	0.008272	0.090575					

```
7   0.000000  0.000000  0.000000  0.000000  1.000000
8   0.000000  1.000000  1.000000  1.000000  1.000000
9   0.000000  0.000000  0.000000  0.000000  1.000000
10 -5.199338 -0.683178  0.012546  0.376283  5.199338
11 -5.199338 -0.686350 -0.023839  0.616541  5.199338
12 -5.199338 -0.676854  0.010037  0.678433  5.199338
13 -5.199338 -0.686350  0.011291  0.421111  5.199338
14 -5.199338 -0.705530 -0.089192  0.716839  5.199338
15  0.000000  0.000000  0.000000  0.000000  1.000000
16  0.000000  0.000000  0.000000  1.000000  1.000000
17  0.000000  0.000000  1.000000  1.000000  1.000000
18  0.000000  0.000000  0.000000  0.000000  1.000000
19  0.000000  0.000000  0.000000  0.000000  1.000000
20  0.000000  0.000000  0.000000  0.000000  1.000000
21  0.000000  0.000000  0.000000  1.000000  1.000000
22  0.000000  0.000000  1.000000  1.000000  1.000000
23  0.000000  0.000000  0.000000  0.000000  1.000000
24  0.000000  0.000000  0.000000  1.000000  1.000000
25  0.000000  0.000000  0.000000  1.000000  1.000000
26  0.000000  0.000000  1.000000  1.000000  1.000000
27  0.000000  0.000000  0.000000  0.000000  1.000000
28  0.000000  0.000000  0.000000  0.000000  1.000000
29  0.000000  0.000000  0.000000  0.000000  1.000000
30  0.000000  0.000000  0.000000  0.000000  1.000000
31 -5.199338 -0.736442  0.334851  0.334851  5.199338
32  0.000000  0.000000  0.000000  1.000000  1.000000
33  0.000000  0.000000  0.000000  0.000000  1.000000
```

[8]:
```
keep_cols = ['Loan Sequence Number']
exclude_cols = ['Original Combined Loan-to-Value␣
 ↪(CLTV)','PropertyType_CP','Mortgage Insurance Percentage (MI %)','Original␣
 ↪Loan Term']
good_cols = [col for col in list(metrics[tf]['column']) if col not in␣
 ↪exclude_cols]
keep_cols.extend(good_cols)
keep_cols
```

[8]:
```
['Loan Sequence Number',
 'Loan Deliquency Within Year',
 'Credit Score',
 'MortgageInsuranceFlag',
 'Units_1',
 'Units_2',
 'Units_3',
 'Units_4',
 'OccupancyStatus_I',
 'OccupancyStatus_P',
```

```
 'OccupancyStatus_S',
 'Original Debt-to-Income (DTI) Ratio',
 'Original UPB',
 'Original Loan-to-Value (LTV)',
 'Original Interest Rate',
 'Channel_B',
 'Channel_C',
 'Channel_R',
 'PropertyType_CO',
 'PropertyType_MH',
 'PropertyType_PU',
 'PropertyType_SF',
 'LoanPurpose_C',
 'LoanPurpose_N',
 'LoanPurpose_P',
 'LoanTerm_360',
 'LoanTerm_180',
 'LoanTerm_240',
 'LoanTerm_120',
 'LoanTerm_300',
 'OneBorrower',
 'AffordableProgramFlag']
```

[9]:
```
len(keep_cols)
```

[9]:
```
32
```

[10]:
```
final_df = training_dat[keep_cols].copy()
final_df.head()
```

[10]:

	Loan Sequence Number	Loan Deliquency Within Year	Credit Score	\
0	F110Q1000008	0.0	1.315958	
1	F110Q1000064	0.0	2.225823	
2	F110Q1000072	0.0	0.451469	
3	F110Q1000080	0.0	-1.733071	
4	F110Q1000096	0.0	1.033647	

	MortgageInsuranceFlag	Units_1	Units_2	Units_3	Units_4	\
0	0	1	0	0	0	
1	0	1	0	0	0	
2	0	1	0	0	0	
3	0	1	0	0	0	
4	0	1	0	0	0	

	OccupancyStatus_I	OccupancyStatus_P	…	LoanPurpose_C	LoanPurpose_N	\
0	0	1	…	0	1	
1	0	1	…	1	0	

```
2                   0                   1  …              0              1
3                   0                   1  …              0              1
4                   0                   1  …              0              1

   LoanPurpose_P  LoanTerm_360  LoanTerm_180  LoanTerm_240  LoanTerm_120  \
0              0             1             0             0             0
1              0             1             0             0             0
2              0             1             0             0             0
3              0             1             0             0             0
4              0             1             0             0             0

   LoanTerm_300  OneBorrower  AffordableProgramFlag
0             0            0                      0
1             0            1                      0
2             0            0                      0
3             0            1                      0
4             0            1                      0

[5 rows x 32 columns]
```

[11]:
```
len(final_df)
```

[11]:
```
396267
```

0.3 Splitting in to training and test datasets

[12]:
```
idx_col = 'Loan Sequence Number'

test_size = round(len(final_df[idx_col]) * .2)
test_loans = final_df[idx_col].sample(test_size)
train_loans = list(set(final_df[idx_col].unique()) - set(test_loans))

train_df = pd.DataFrame({idx_col:train_loans})
train_df = train_df.merge(final_df, on=idx_col, how='left')

test_df = pd.DataFrame({idx_col:test_loans})
test_df = test_df.merge(final_df, on=idx_col, how='left')
```

```
UserWarning: Distributing <class 'dict'> object. This may take some time.
```

[13]:
```
train_df.head()
```

[13]:
```
  Loan Sequence Number  Loan Deliquency Within Year  Credit Score  \
0         F110Q1252148                          0.0     -0.177827
1         F111Q4171377                          0.0     -0.383024
2         F114Q3148996                          0.0      0.634851
3         F118Q2045353                          0.0     -0.415634
```

```
4         F114Q4174402                          0.0      0.634851

   MortgageInsuranceFlag  Units_1  Units_2  Units_3  Units_4  \
0                      0        1        0        0        0
1                      0        1        0        0        0
2                      0        1        0        0        0
3                      0        1        0        0        0
4                      1        1        0        0        0

   OccupancyStatus_I  OccupancyStatus_P  …  LoanPurpose_C  LoanPurpose_N  \
0                  0                  1  …              1              0
1                  0                  1  …              0              1
2                  0                  1  …              1              0
3                  0                  1  …              0              0
4                  0                  1  …              0              0

   LoanPurpose_P  LoanTerm_360  LoanTerm_180  LoanTerm_240  LoanTerm_120  \
0              0             0             1             0             0
1              0             0             1             0             0
2              0             0             1             0             0
3              1             0             1             0             0
4              1             1             0             0             0

   LoanTerm_300  OneBorrower  AffordableProgramFlag
0             0            1                      0
1             0            0                      0
2             0            1                      0
3             0            0                      0
4             0            1                      0

[5 rows x 32 columns]
```

[14]:
```
test_df.head()
```

[14]:
```
  Loan Sequence Number  Loan Deliquency Within Year  Credit Score  \
0         F117Q3099461                          0.0      0.604448
1         F115Q1238542                          0.0     -0.415634
2         F119Q2306763                          0.0     -0.231641
3         F110Q2117701                          0.0     -0.705530
4         F116Q1069514                          0.0     -0.605955

   MortgageInsuranceFlag  Units_1  Units_2  Units_3  Units_4  \
0                      0        1        0        0        0
1                      1        1        0        0        0
2                      0        1        0        0        0
3                      0        1        0        0        0
4                      0        1        0        0        0
```

```
   OccupancyStatus_I  OccupancyStatus_P  …  LoanPurpose_C  LoanPurpose_N  \
0                  0                  1  …              1              0
1                  0                  1  …              0              0
2                  0                  1  …              1              0
3                  0                  1  …              1              0
4                  0                  1  …              0              0

   LoanPurpose_P  LoanTerm_360  LoanTerm_180  LoanTerm_240  LoanTerm_120  \
0              0             1             0             0             0
1              1             1             0             0             0
2              0             1             0             0             0
3              0             1             0             0             0
4              1             1             0             0             0

   LoanTerm_300  OneBorrower  AffordableProgramFlag
0             0            0                      0
1             0            1                      0
2             0            1                      0
3             0            0                      0
4             0            1                      0

[5 rows x 32 columns]
```

[15]:
```
# Build training and test datasets
X_train = train_df[train_df.columns[2:]].to_numpy()
y_train = train_df["Loan Deliquency Within Year"].to_numpy()

X_test = test_df[test_df.columns[2:]].to_numpy()
y_test = test_df["Loan Deliquency Within Year"].to_numpy()
```

[16]:
```
X_train.shape
```

[16]:
```
(317014, 30)
```

0.4 Building model

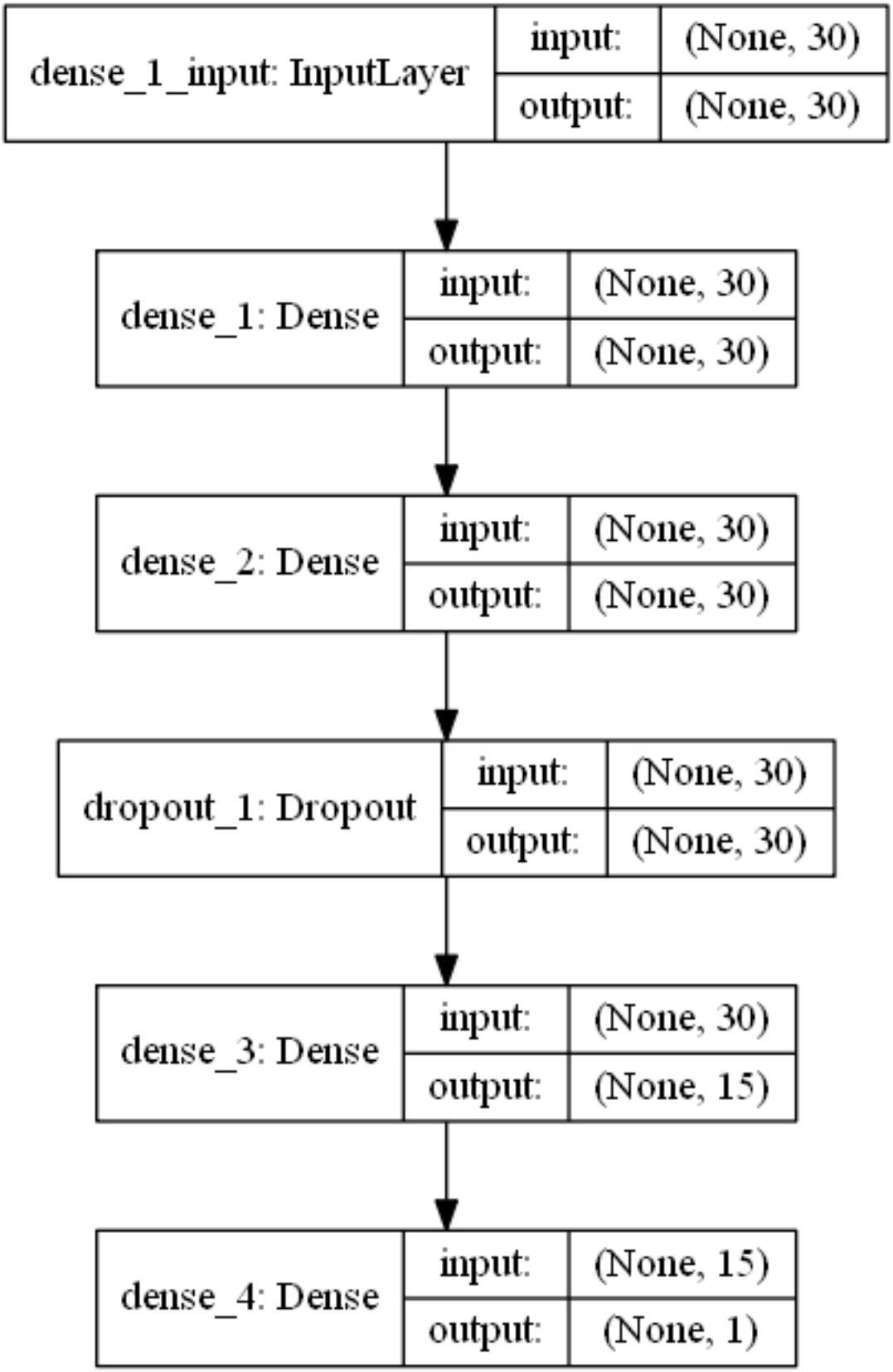
[17]:
```
# Based on https://keras.io/examples/structured_data/imbalanced_classification/

pos_count = final_df['Loan Deliquency Within Year'].sum()

weight_for_0 = 1.0 / (len(final_df)-pos_count)
weight_for_1 = 1.0 / pos_count

model = Sequential()
model.add(Dense(30,activation='relu',input_shape=(X_train.shape[-1],)))
model.add(Dense(30,activation='relu'))
```

```
model.add(Dropout(.3))
model.add(Dense(15,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

metrics = [
    keras.metrics.FalseNegatives(name="fn"),
    keras.metrics.FalsePositives(name="fp"),
    keras.metrics.TrueNegatives(name="tn"),
    keras.metrics.TruePositives(name="tp"),
    keras.metrics.BinaryAccuracy(name="binary_accuracy"),
    keras.metrics.Precision(name="precision"),
    keras.metrics.Recall(name="recall")
]
model.compile(optimizer=keras.optimizers.Adam(1e-2),␣
 ↪loss="binary_crossentropy", metrics=metrics)

class_weight = {0: weight_for_0, 1: weight_for_1}
```

```
WARNING:tensorflow:From C:\tools\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3172:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

[18]:
```
plot_model(model, show_shapes=True)
```

[18]:


dense_1_input: InputLayer
input: (None, 30)
output: (None, 30)
dense_1: Dense
input: (None, 30)
output: (None, 30)
dense_2: Dense
input: (None, 30)
output: (None, 30)
dropout_1: Dropout
input: (None, 30)
output: (None, 30)
dense_3: Dense
input: (None, 30)
output: (None, 15)
dense_4: Dense
input: (None, 15)
output: (None, 1)

0.5 Training model

[19]:
```
history = model.fit(X_train,y_train,class_weight=class_weight,batch_size=2048,
                    epochs=20, validation_split=.3, verbose=1)
```

```
WARNING:tensorflow:From C:\tools\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 221909 samples, validate on 95105 samples
Epoch 1/20
221909/221909 [==============================] - 1s 5us/step - loss: 3.5002e-06
- fn: 1266.0000 - fp: 142513.0000 - tn: 74591.0000 - tp: 3539.0000 -
binary_accuracy: 0.3521 - precision: 0.0242 - recall: 0.7365 - val_loss:
3.2969e-06 - val_fn: 771.0000 - val_fp: 34768.0000 - val_tn: 58344.0000 -
val_tp: 1222.0000 - val_binary_accuracy: 0.6263 - val_precision: 0.0340 -
val_recall: 0.6131
Epoch 2/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.2980e-06
- fn: 1926.0000 - fp: 72892.0000 - tn: 144212.0000 - tp: 2879.0000 -
binary_accuracy: 0.6628 - precision: 0.0380 - recall: 0.5992 - val_loss:
3.1935e-06 - val_fn: 869.0000 - val_fp: 26800.0000 - val_tn: 66312.0000 -
val_tp: 1124.0000 - val_binary_accuracy: 0.7091 - val_precision: 0.0403 -
val_recall: 0.5640
Epoch 3/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.2257e-06
- fn: 1896.0000 - fp: 68305.0000 - tn: 148799.0000 - tp: 2909.0000 -
binary_accuracy: 0.6836 - precision: 0.0408 - recall: 0.6054 - val_loss:
3.1776e-06 - val_fn: 901.0000 - val_fp: 25340.0000 - val_tn: 67772.0000 -
val_tp: 1092.0000 - val_binary_accuracy: 0.7241 - val_precision: 0.0413 -
val_recall: 0.5479
Epoch 4/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.2080e-06
- fn: 1800.0000 - fp: 69277.0000 - tn: 147827.0000 - tp: 3005.0000 -
binary_accuracy: 0.6797 - precision: 0.0416 - recall: 0.6254 - val_loss:
3.1605e-06 - val_fn: 827.0000 - val_fp: 28218.0000 - val_tn: 64894.0000 -
val_tp: 1166.0000 - val_binary_accuracy: 0.6946 - val_precision: 0.0397 -
val_recall: 0.5850
Epoch 5/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1852e-06
- fn: 1786.0000 - fp: 69270.0000 - tn: 147834.0000 - tp: 3019.0000 -
binary_accuracy: 0.6798 - precision: 0.0418 - recall: 0.6283 - val_loss:
3.1568e-06 - val_fn: 673.0000 - val_fp: 34625.0000 - val_tn: 58487.0000 -
val_tp: 1320.0000 - val_binary_accuracy: 0.6289 - val_precision: 0.0367 -
val_recall: 0.6623
Epoch 6/20
```

```
221909/221909 [==============================] - 1s 4us/step - loss: 3.1806e-06
- fn: 1666.0000 - fp: 73660.0000 - tn: 143444.0000 - tp: 3139.0000 -
binary_accuracy: 0.6606 - precision: 0.0409 - recall: 0.6533 - val_loss:
3.1467e-06 - val_fn: 729.0000 - val_fp: 31564.0000 - val_tn: 61548.0000 -
val_tp: 1264.0000 - val_binary_accuracy: 0.6604 - val_precision: 0.0385 -
val_recall: 0.6342
Epoch 7/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1780e-06
- fn: 1698.0000 - fp: 72418.0000 - tn: 144686.0000 - tp: 3107.0000 -
binary_accuracy: 0.6660 - precision: 0.0411 - recall: 0.6466 - val_loss:
3.1462e-06 - val_fn: 748.0000 - val_fp: 30406.0000 - val_tn: 62706.0000 -
val_tp: 1245.0000 - val_binary_accuracy: 0.6724 - val_precision: 0.0393 -
val_recall: 0.6247
Epoch 8/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1696e-06
- fn: 1680.0000 - fp: 72347.0000 - tn: 144757.0000 - tp: 3125.0000 -
binary_accuracy: 0.6664 - precision: 0.0414 - recall: 0.6504 - val_loss:
3.1552e-06 - val_fn: 894.0000 - val_fp: 24494.0000 - val_tn: 68618.0000 -
val_tp: 1099.0000 - val_binary_accuracy: 0.7331 - val_precision: 0.0429 -
val_recall: 0.5514
Epoch 9/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1630e-06
- fn: 1756.0000 - fp: 69704.0000 - tn: 147400.0000 - tp: 3049.0000 -
binary_accuracy: 0.6780 - precision: 0.0419 - recall: 0.6345 - val_loss:
3.1487e-06 - val_fn: 855.0000 - val_fp: 26170.0000 - val_tn: 66942.0000 -
val_tp: 1138.0000 - val_binary_accuracy: 0.7158 - val_precision: 0.0417 -
val_recall: 0.5710
Epoch 10/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1581e-06
- fn: 1685.0000 - fp: 72752.0000 - tn: 144352.0000 - tp: 3120.0000 -
binary_accuracy: 0.6646 - precision: 0.0411 - recall: 0.6493 - val_loss:
3.1425e-06 - val_fn: 807.0000 - val_fp: 27579.0000 - val_tn: 65533.0000 -
val_tp: 1186.0000 - val_binary_accuracy: 0.7015 - val_precision: 0.0412 -
val_recall: 0.5951
Epoch 11/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1548e-06
- fn: 1732.0000 - fp: 71073.0000 - tn: 146031.0000 - tp: 3073.0000 -
binary_accuracy: 0.6719 - precision: 0.0414 - recall: 0.6395 - val_loss:
3.1622e-06 - val_fn: 896.0000 - val_fp: 24234.0000 - val_tn: 68878.0000 -
val_tp: 1097.0000 - val_binary_accuracy: 0.7358 - val_precision: 0.0433 -
val_recall: 0.5504
Epoch 12/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1561e-06
- fn: 1624.0000 - fp: 74023.0000 - tn: 143081.0000 - tp: 3181.0000 -
binary_accuracy: 0.6591 - precision: 0.0412 - recall: 0.6620 - val_loss:
3.1366e-06 - val_fn: 744.0000 - val_fp: 30436.0000 - val_tn: 62676.0000 -
val_tp: 1249.0000 - val_binary_accuracy: 0.6722 - val_precision: 0.0394 -
val_recall: 0.6267
```

```
Epoch 13/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1451e-06
- fn: 1632.0000 - fp: 73668.0000 - tn: 143436.0000 - tp: 3173.0000 -
binary_accuracy: 0.6607 - precision: 0.0413 - recall: 0.6604 - val_loss:
3.1423e-06 - val_fn: 766.0000 - val_fp: 29293.0000 - val_tn: 63819.0000 -
val_tp: 1227.0000 - val_binary_accuracy: 0.6839 - val_precision: 0.0402 -
val_recall: 0.6157
Epoch 14/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1463e-06
- fn: 1617.0000 - fp: 75627.0000 - tn: 141477.0000 - tp: 3188.0000 -
binary_accuracy: 0.6519 - precision: 0.0404 - recall: 0.6635 - val_loss:
3.1450e-06 - val_fn: 800.0000 - val_fp: 28068.0000 - val_tn: 65044.0000 -
val_tp: 1193.0000 - val_binary_accuracy: 0.6965 - val_precision: 0.0408 -
val_recall: 0.5986
Epoch 15/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1409e-06
- fn: 1642.0000 - fp: 73344.0000 - tn: 143760.0000 - tp: 3163.0000 -
binary_accuracy: 0.6621 - precision: 0.0413 - recall: 0.6583 - val_loss:
3.1374e-06 - val_fn: 826.0000 - val_fp: 27043.0000 - val_tn: 66069.0000 -
val_tp: 1167.0000 - val_binary_accuracy: 0.7070 - val_precision: 0.0414 -
val_recall: 0.5855
Epoch 16/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1386e-06
- fn: 1655.0000 - fp: 72319.0000 - tn: 144785.0000 - tp: 3150.0000 -
binary_accuracy: 0.6666 - precision: 0.0417 - recall: 0.6556 - val_loss:
3.1612e-06 - val_fn: 867.0000 - val_fp: 25134.0000 - val_tn: 67978.0000 -
val_tp: 1126.0000 - val_binary_accuracy: 0.7266 - val_precision: 0.0429 -
val_recall: 0.5650
Epoch 17/20
221909/221909 [==============================] - 1s 4us/step - loss: 3.1341e-06
- fn: 1625.0000 - fp: 73162.0000 - tn: 143942.0000 - tp: 3180.0000 -
binary_accuracy: 0.6630 - precision: 0.0417 - recall: 0.6618 - val_loss:
3.1424e-06 - val_fn: 728.0000 - val_fp: 30863.0000 - val_tn: 62249.0000 -
val_tp: 1265.0000 - val_binary_accuracy: 0.6678 - val_precision: 0.0394 -
val_recall: 0.6347
Epoch 18/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1311e-06
- fn: 1603.0000 - fp: 74621.0000 - tn: 142483.0000 - tp: 3202.0000 -
binary_accuracy: 0.6565 - precision: 0.0411 - recall: 0.6664 - val_loss:
3.1402e-06 - val_fn: 696.0000 - val_fp: 32201.0000 - val_tn: 60911.0000 -
val_tp: 1297.0000 - val_binary_accuracy: 0.6541 - val_precision: 0.0387 -
val_recall: 0.6508
Epoch 19/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1378e-06
- fn: 1638.0000 - fp: 73479.0000 - tn: 143625.0000 - tp: 3167.0000 -
binary_accuracy: 0.6615 - precision: 0.0413 - recall: 0.6591 - val_loss:
3.1380e-06 - val_fn: 649.0000 - val_fp: 34746.0000 - val_tn: 58366.0000 -
val_tp: 1344.0000 - val_binary_accuracy: 0.6278 - val_precision: 0.0372 -
```

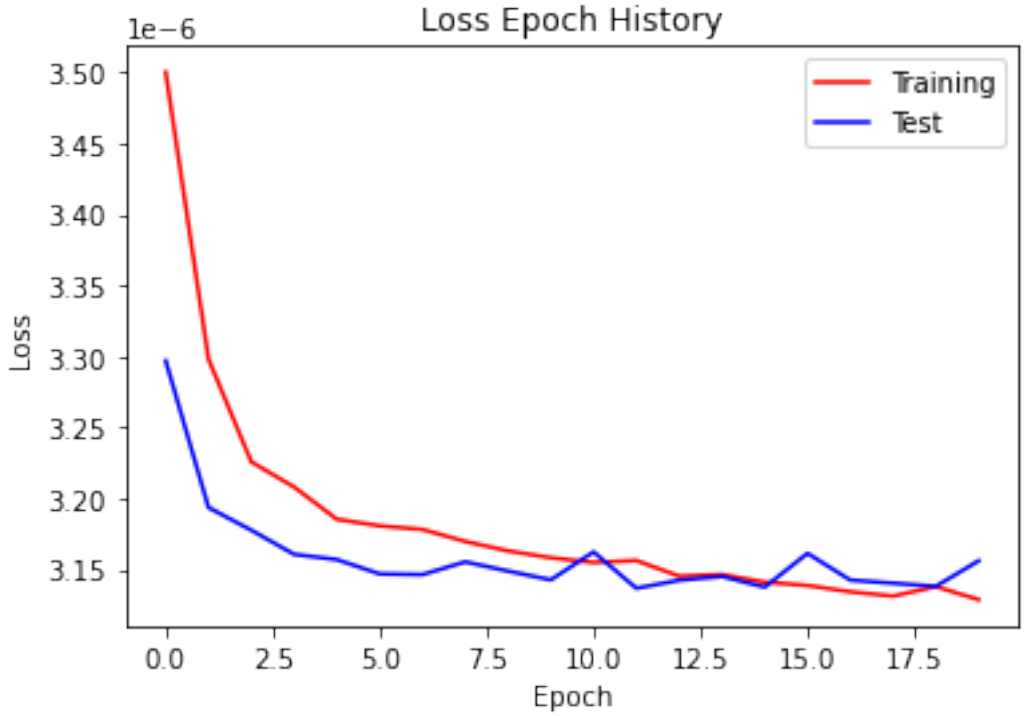
```
val_recall: 0.6744
Epoch 20/20
221909/221909 [==============================] - 1s 3us/step - loss: 3.1286e-06
- fn: 1592.0000 - fp: 75259.0000 - tn: 141845.0000 - tp: 3213.0000 -
binary_accuracy: 0.6537 - precision: 0.0409 - recall: 0.6687 - val_loss:
3.1560e-06 - val_fn: 804.0000 - val_fp: 27750.0000 - val_tn: 65362.0000 -
val_tp: 1189.0000 - val_binary_accuracy: 0.6998 - val_precision: 0.0411 -
val_recall: 0.5966
```

[20]:
```
history.history.keys()
```

[20]:
```
dict_keys(['val_loss', 'val_fn', 'val_fp', 'val_tn', 'val_tp',
'val_binary_accuracy', 'val_precision', 'val_recall', 'loss', 'fn', 'fp', 'tn',
'tp', 'binary_accuracy', 'precision', 'recall'])
```

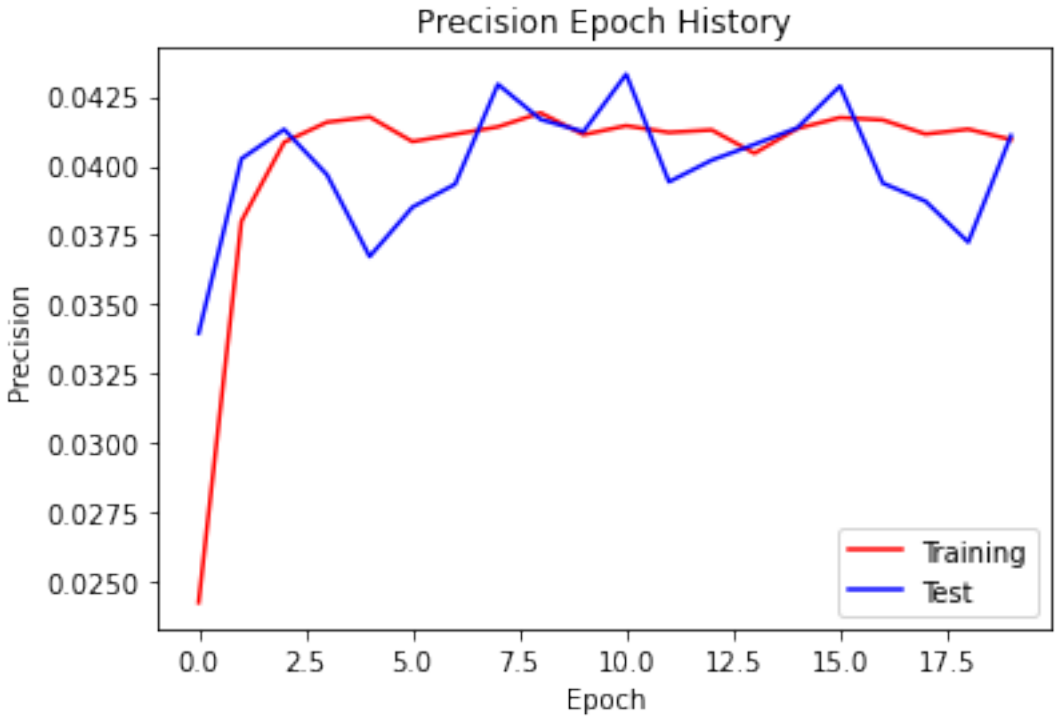
[21]:
```
# Inspired by https://machinelearningmastery.com/
 ↪,→display-deep-learning-model-training-history-in-keras/
plt.plot(history.history['loss'],'r')
plt.plot(history.history['val_loss'],'b')
plt.title('Loss Epoch History')
plt.legend(['Training','Test'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

[21]:
```
Text(0, 0.5, 'Loss')
```



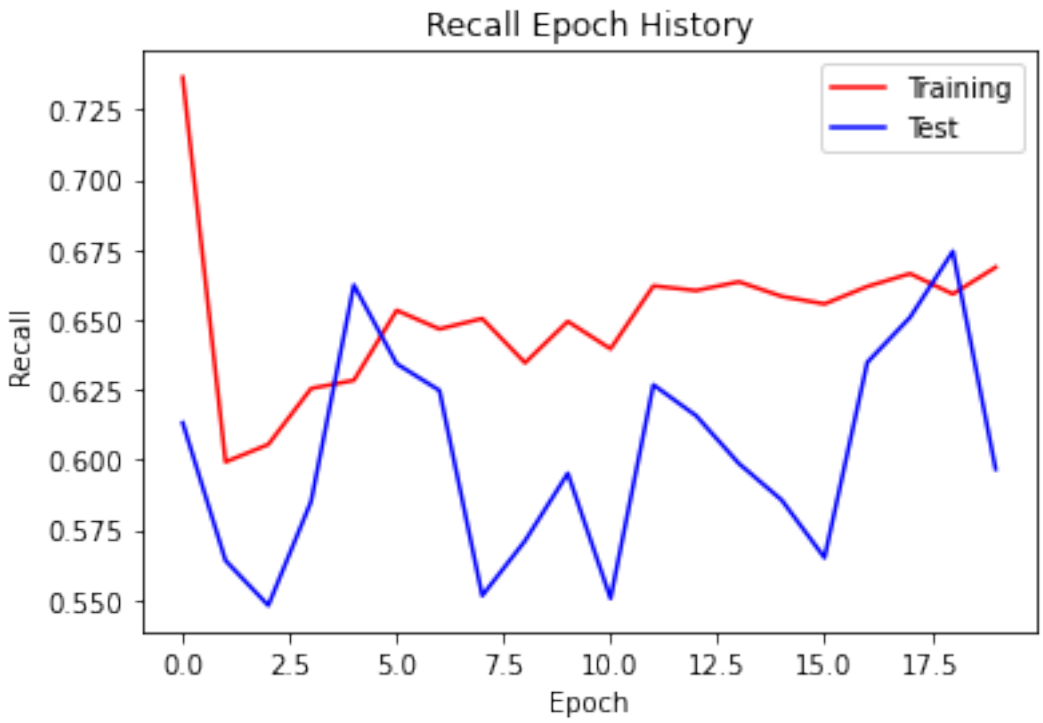
[22]:
```
# Inspired by https://machinelearningmastery.com/
 ↪,→display-deep-learning-model-training-history-in-keras/
plt.plot(history.history['precision'],'r')
plt.plot(history.history['val_precision'],'b')
plt.legend(['Training','Test'])
plt.title('Precision Epoch History')
plt.xlabel('Epoch')
plt.ylabel('Precision')
```

[22]:
```
Text(0, 0.5, 'Precision')
```



[23]:
```
# Inspired by https://machinelearningmastery.com/
 ↪,→display-deep-learning-model-training-history-in-keras/
plt.plot(history.history['recall'],'r')
plt.plot(history.history['val_recall'],'b')
plt.legend(['Training','Test'])
plt.title('Recall Epoch History')
plt.xlabel('Epoch')
plt.ylabel('Recall')
```

[23]:
```
Text(0, 0.5, 'Recall')
```



0.6 Testing model

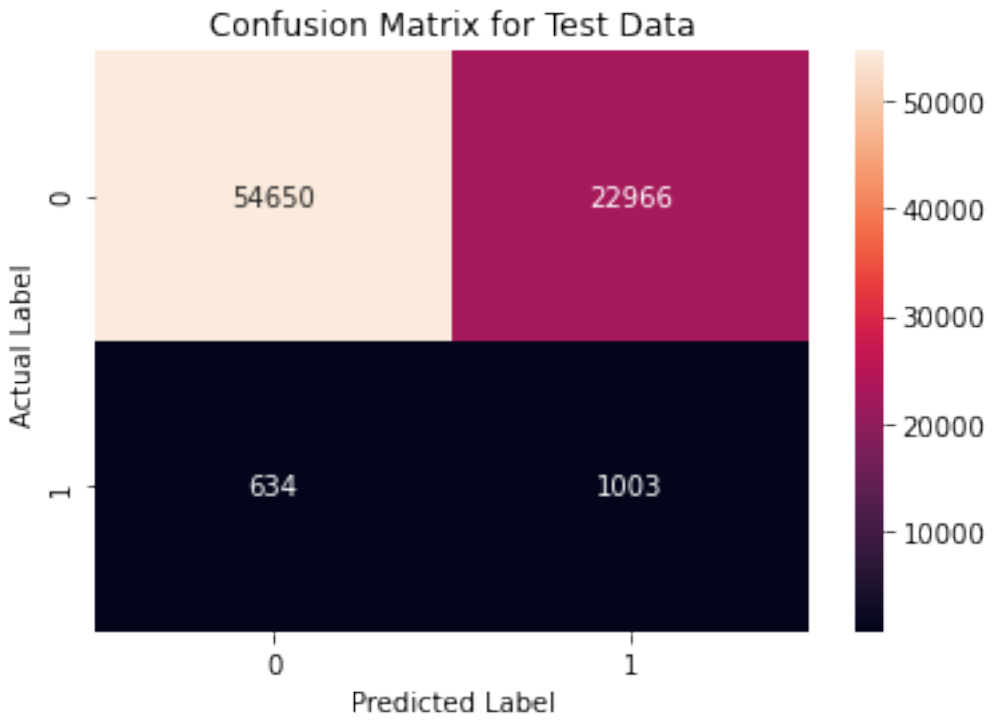
[25]:
```
preds = model.predict(X_test)
y_pred = np.where(preds>0.5,1,0)
conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix
```

[25]:
```
array([[54650, 22966],
       [  634,  1003]], dtype=int64)
```

[26]:
```
conf_df = pd.DataFrame(conf_matrix, index = ['Actual Negative','Actual␣
↪Positive'],
                       columns = ['Predict Negative','Predict Positive'])
sns.heatmap(conf_df, annot=True, fmt='.5g')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title('Confusion Matrix for Test Data')
```

```
UserWarning: Distributing <class 'numpy.ndarray'> object. This may take some
time.
```

[26]:
```
Text(0.5, 1.0, 'Confusion Matrix for Test Data')
```



0.7 Leveraging SHAP (SHapley Additive exPlanations) to explain test predictions

[145]:
```
shap.initjs()
```

```
<IPython.core.display.HTML object>
```

[146]:
```
# inspired by https://shap.readthedocs.io/en/latest/example_notebooks/
 ↪tabular_examples/neural_networks/
 ↪Census%20income%20classification%20with%20Keras.html
def f(X):
    return model.predict([X[:,i] for i in range(X.shape[1])]).flatten()
```

[147]:
```
with open("./PROCESSED/scalers.pkl","rb") as f:
    scaler_dct = pickle.load(f)
```

[148]:
```
test_df.columns
```

[148]:
```
Index(['Loan Sequence Number', 'Loan Deliquency Within Year', 'Credit Score',
       'MortgageInsuranceFlag', 'Units_1', 'Units_2', 'Units_3', 'Units_4',
       'OccupancyStatus_I', 'OccupancyStatus_P', 'OccupancyStatus_S',
       'Original Debt-to-Income (DTI) Ratio', 'Original UPB',
       'Original Loan-to-Value (LTV)', 'Original Interest Rate', 'Channel_B',
```

```
       'Channel_C', 'Channel_R', 'PropertyType_CO', 'PropertyType_MH',
       'PropertyType_PU', 'PropertyType_SF', 'LoanPurpose_C', 'LoanPurpose_N',
       'LoanPurpose_P', 'LoanTerm_360', 'LoanTerm_180', 'LoanTerm_240',
       'LoanTerm_120', 'LoanTerm_300', 'OneBorrower', 'AffordableProgramFlag'],
      dtype='object')
```

[149]:
```
test_df[test_df['OneBorrower'] == 1].head()
```

[149]:

	Loan Sequence Number	Loan Deliquency Within Year	Credit Score	\
1	F115Q1238542	0.0	-0.415634	
2	F119Q2306763	0.0	-0.231641	
4	F116Q1069514	0.0	-0.605955	
5	F111Q3042379	0.0	1.858747	
8	F117Q4058814	0.0	0.899534	

	MortgageInsuranceFlag	Units_1	Units_2	Units_3	Units_4	\
1	1	1	0	0	0	
2	0	1	0	0	0	
4	0	1	0	0	0	
5	0	1	0	0	0	
8	0	1	0	0	0	

	OccupancyStatus_I	OccupancyStatus_P	…	LoanPurpose_C	LoanPurpose_N	\
1	0	1	…	0	0	
2	0	1	…	1	0	
4	0	1	…	0	0	
5	0	1	…	0	0	
8	1	0	…	1	0	

	LoanPurpose_P	LoanTerm_360	LoanTerm_180	LoanTerm_240	LoanTerm_120	\
1	1	1	0	0	0	
2	0	1	0	0	0	
4	1	1	0	0	0	
5	1	1	0	0	0	
8	0	1	0	0	0	

	LoanTerm_300	OneBorrower	AffordableProgramFlag
1	0	1	0
2	0	1	0
4	0	1	0
5	0	1	0
8	0	1	0

[5 rows x 32 columns]

[150]:
```
test_df[test_df['Credit Score'] < -5].head()
```

[150]:
```
      Loan Sequence Number  Loan Deliquency Within Year  Credit Score  \
16569         F110Q1242556                          0.0     -5.199338

       MortgageInsuranceFlag  Units_1  Units_2  Units_3  Units_4  \
16569                      0        1        0        0        0

       OccupancyStatus_I  OccupancyStatus_P  …  LoanPurpose_C  \
16569                  0                  1  …              1

       LoanPurpose_N  LoanPurpose_P  LoanTerm_360  LoanTerm_180  LoanTerm_240  \
16569              0              0             0             1             0

       LoanTerm_120  LoanTerm_300  OneBorrower  AffordableProgramFlag
16569             0             0            0                      0

[1 rows x 32 columns]
```

[151]:
```
# Converting features back to original scale for display purposes
def get_original(col,val):
    if col in scaler_dct:
        scaler = scaler_dct[col]
        nval = scaler.inverse_transform(np.array([[val]]))[0][0]
    else:
        nval = val
    return nval
```

[152]:
```
rec = X_test[4]
shap_values = explainer.shap_values(rec, nsamples=2500)
feature_vals = [get_original(col_name,value) for col_name,value in␣
 ↪zip(keep_cols[2:],rec)]
feature_vals = np.array(feature_vals)
shap.force_plot(explainer.expected_value, shap_values[0],
                features=feature_vals, feature_names=keep_cols[2:])
```

[152]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x1688249cf98>
```

[153]:
```
shap_values100 = explainer.shap_values(X_test[100:110], nsamples=500)
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

[154]:
```
shap.force_plot(explainer.expected_value, shap_values100[0], X_test[100:110],␣
 ↪feature_names=keep_cols[2:])
```

[154]:
```
<shap.plots._force.AdditiveForceArrayVisualizer at 0x16892859320>
```

[ ]:
```

```