Open in Colab

In [ ]:

```
%cd /content/drive/MyDrive/CGM
!ls
# !gzip "/content/drive/MyDrive/CGM/dataset/HIGGS_6M.csv.gz" -d "/content/drive/MyDrive/C
GM/dataset"
```

In [ ]:

```
import xgboost
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

EDA

In [ ]:

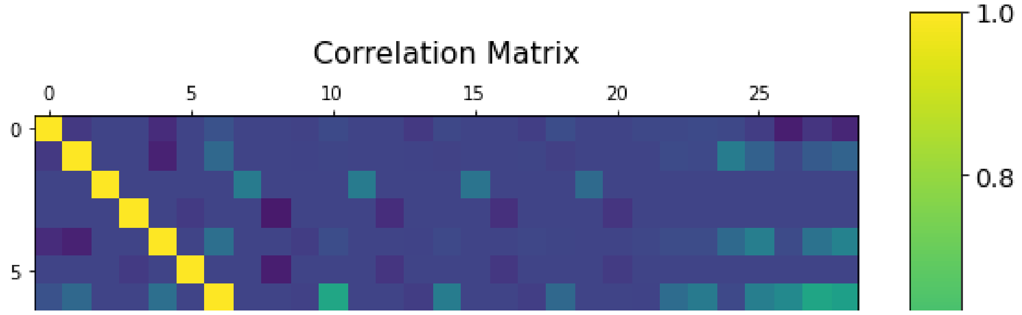
```
df=pd.read_csv("dataset/HIGGS_6M.csv")
df.head()
```

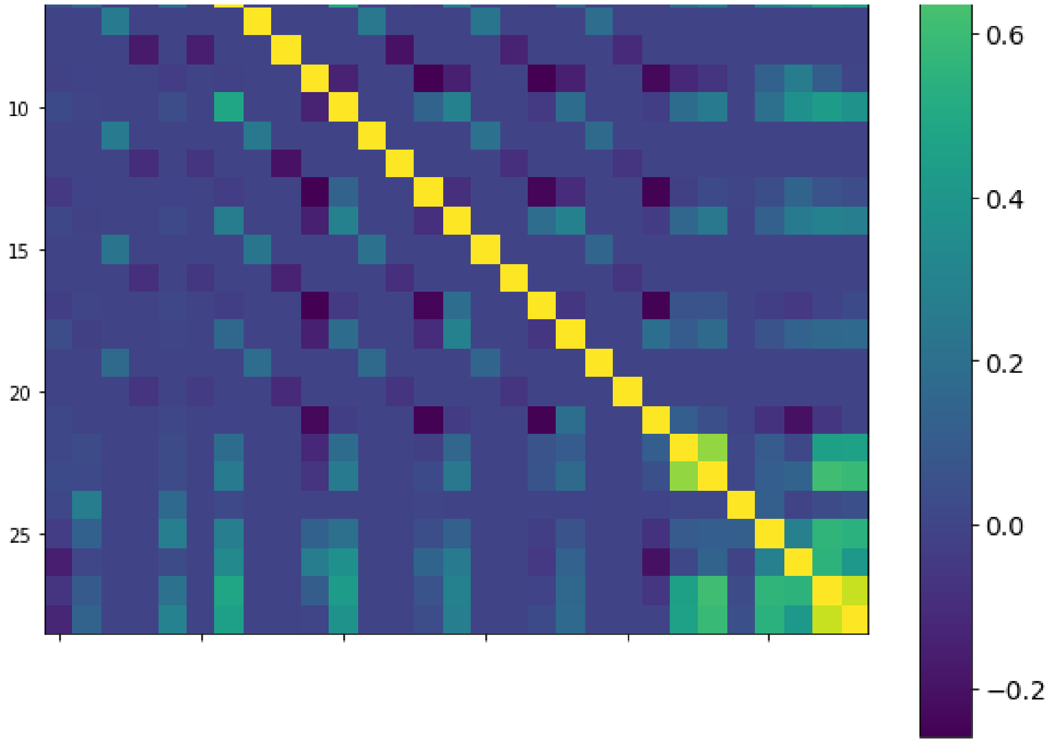
Out[ ]:

	1.000000000000000000e+00	8.692932128906250000e-01	-6.350818276405334473e-01	2.256902605295181274e-01	3.2747006416320800
0	1.0	0.907542	0.329147	0.359412	1.497
1	1.0	0.798835	1.470639	-1.635975	0.453
2	0.0	1.344385	-0.876626	0.935913	1.992
3	1.0	1.105009	0.321356	1.522401	0.882
4	0.0	1.595839	-0.607811	0.007075	1.818

In [ ]:

```
f = plt.figure(figsize=(10, 10))
plt.matshow(df.corr(), fignum=f.number)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=14)
plt.title('Correlation Matrix', fontsize=16);
```



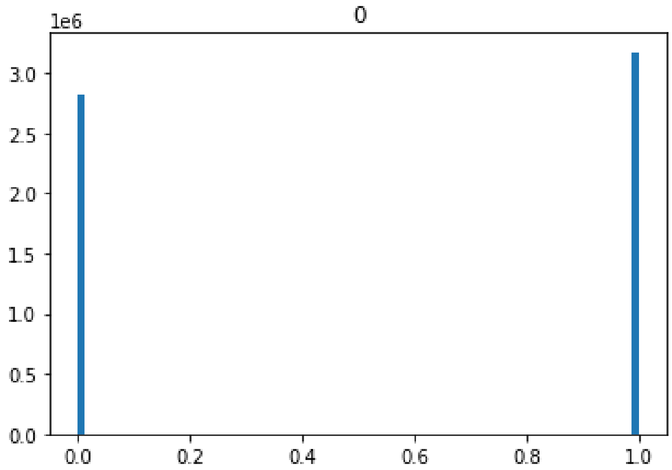


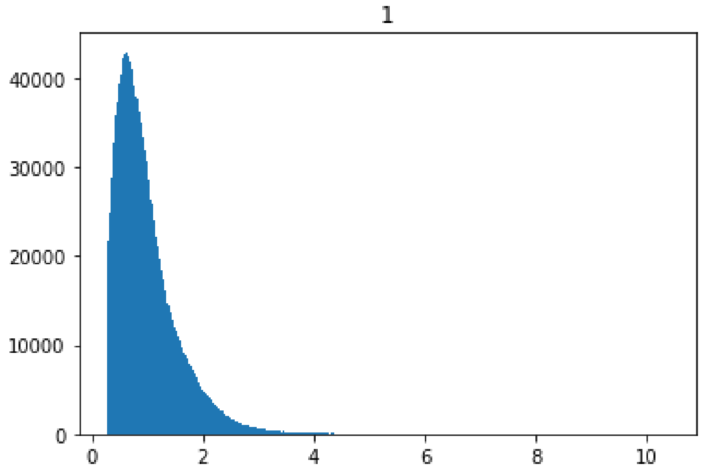
In [ ]:

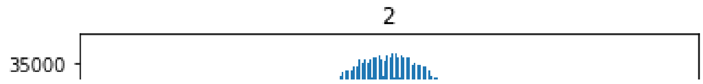
```
dataset = pd.read_csv("dataset/HIGGS_6M.csv").to_numpy()
```

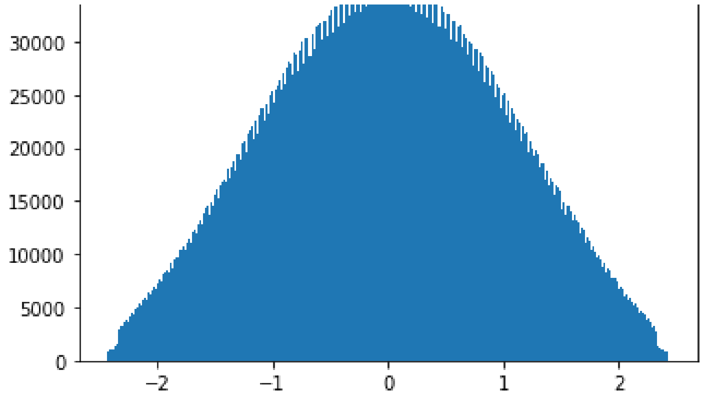
In [ ]:

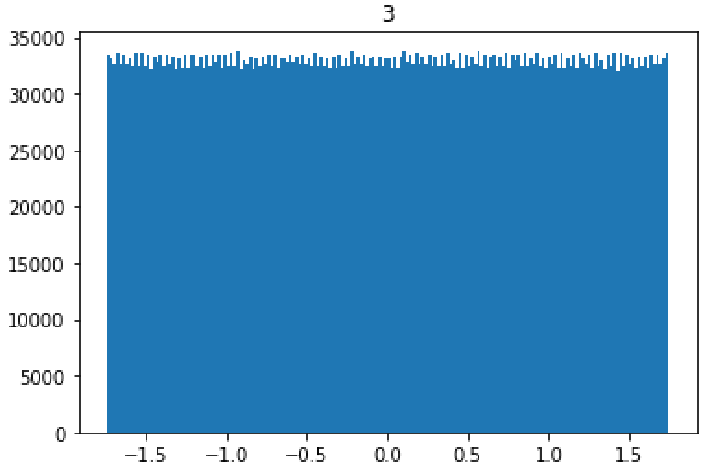
```
for i in range(29):
  plt.hist(dataset[:,i],bins='auto')
  plt.title(str(i))
  plt.show()
```

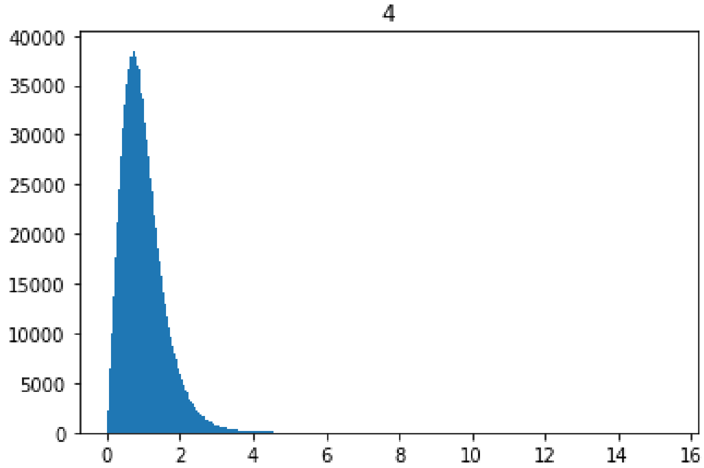


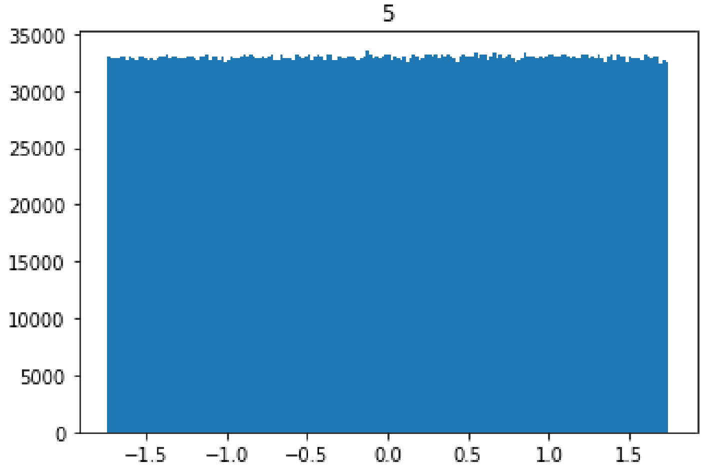


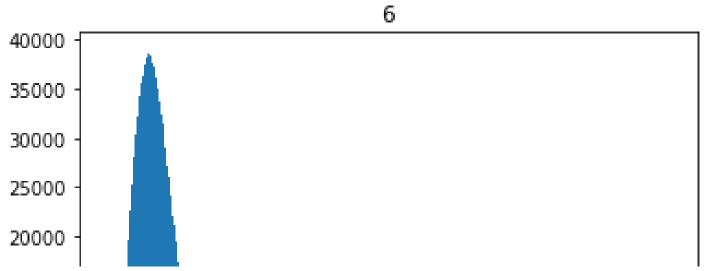


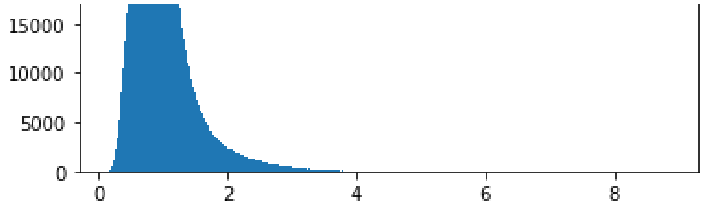

30000
25000
20000
15000
10000
5000
0
−2
−1
0
1
2

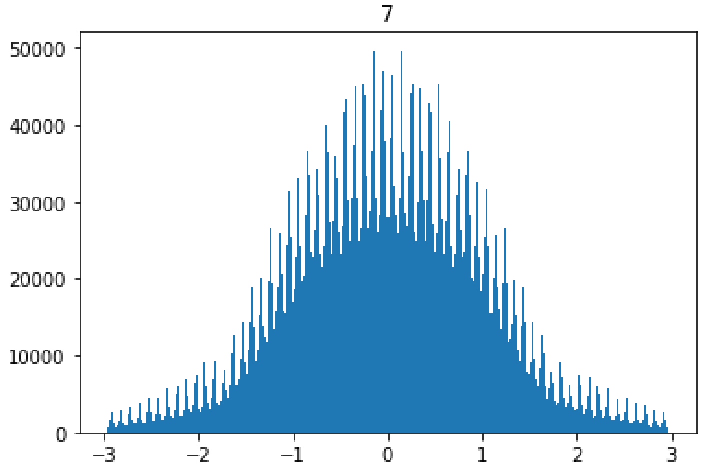

3
35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

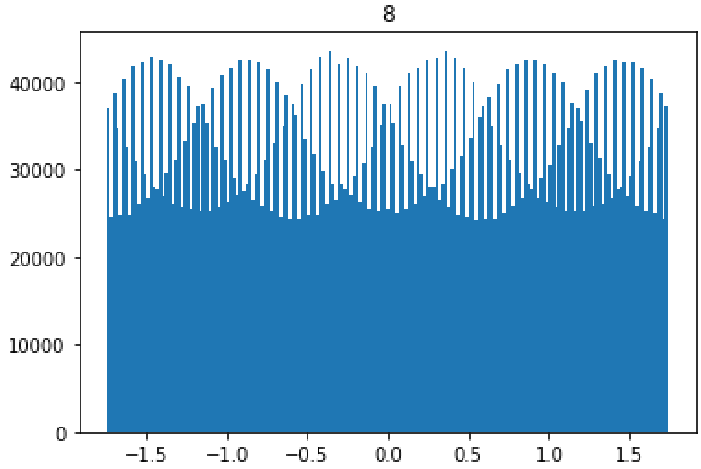

4
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16

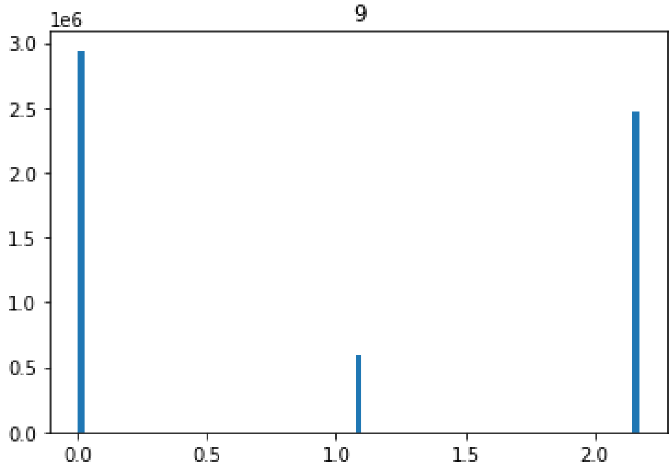

5
35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

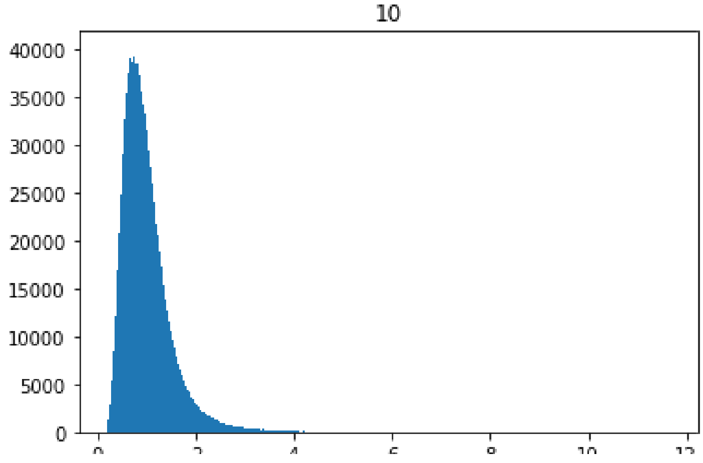

6
40000
35000
30000
25000
20000


15000
10000
5000
0
0
2
4
6
8

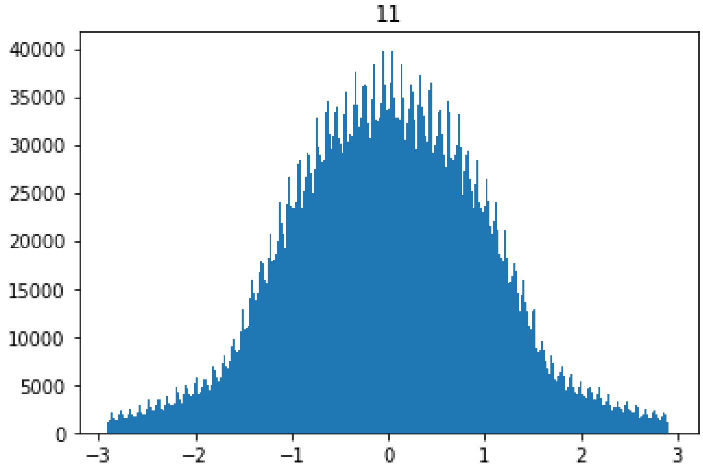

7
50000
40000
30000
20000
10000
0
−3
−2
−1
0
1
2
3

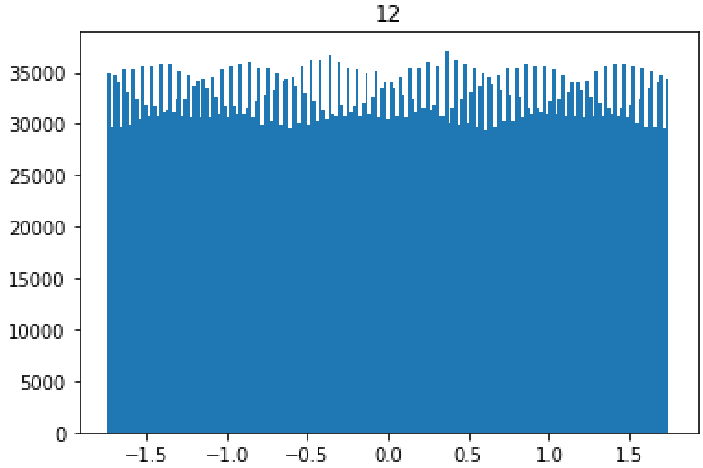

8
40000
30000
20000
10000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

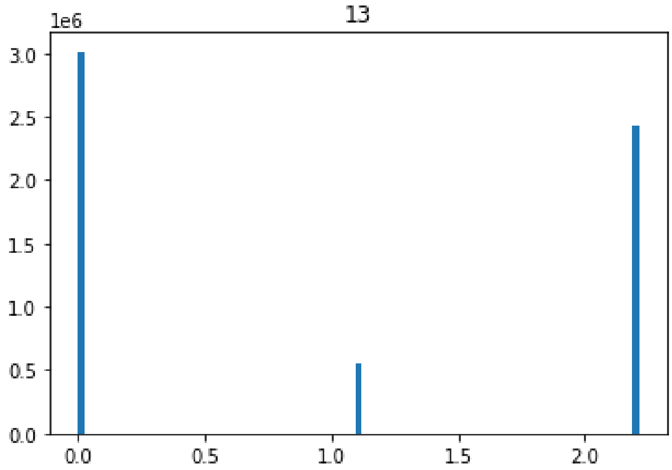

9
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0

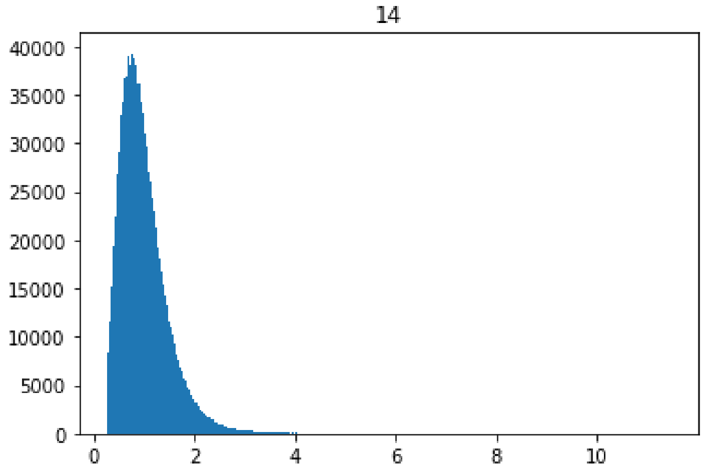

10
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12

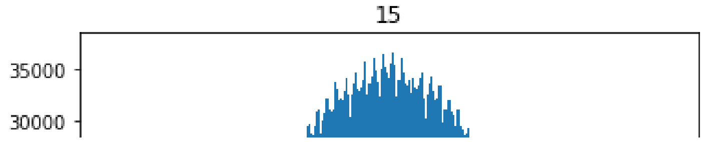
0
2
4
6
8
10
12

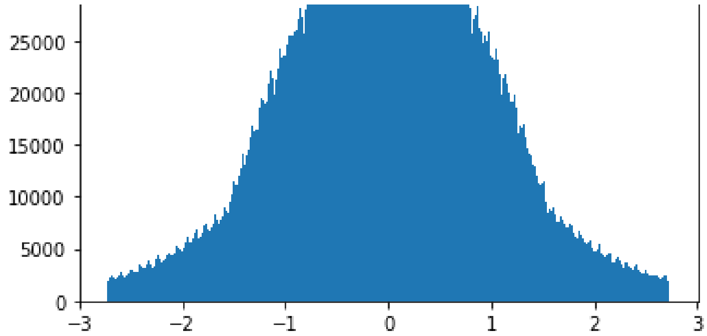

11
40000
35000
30000
25000
20000
15000
10000
5000
0
−3
−2
−1
0
1
2
3


12
35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

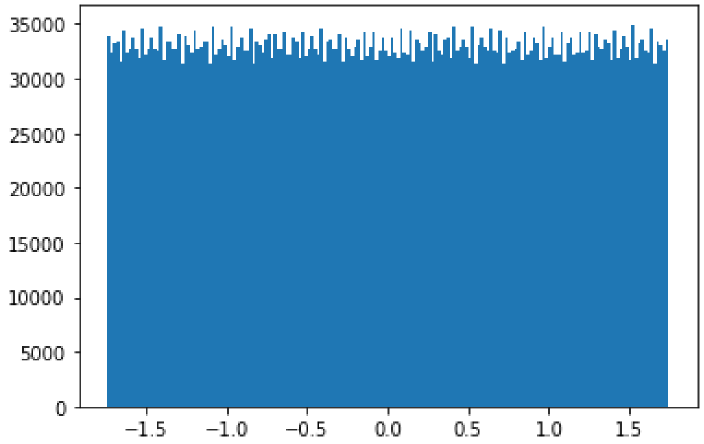

13
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0


14
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10

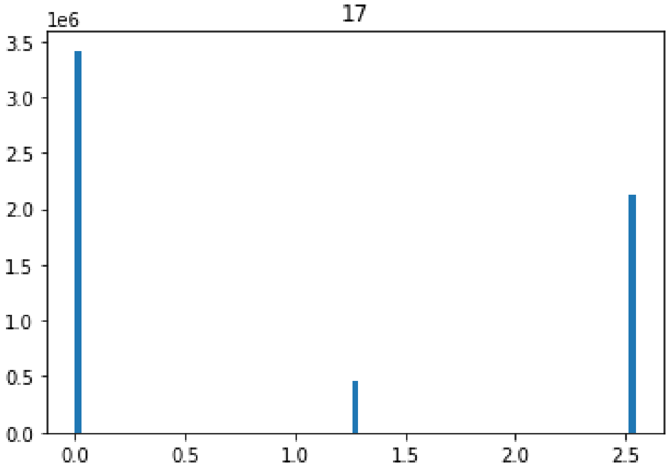

15
35000
30000


25000
20000
15000
10000
5000
0
−3
−2
−1
0
1
2
3

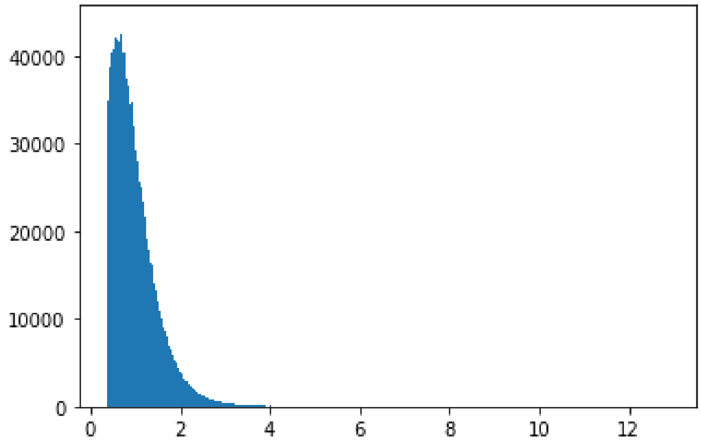
16


35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

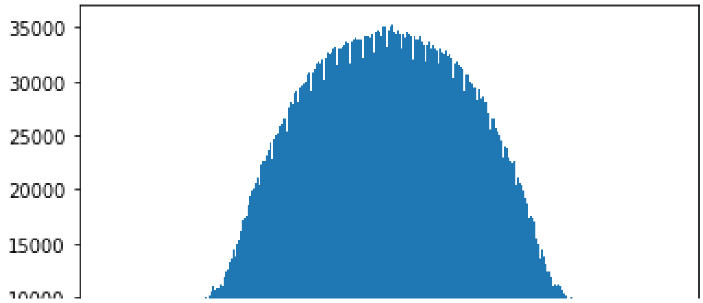
17

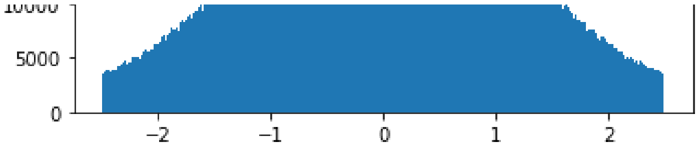

1e6
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5

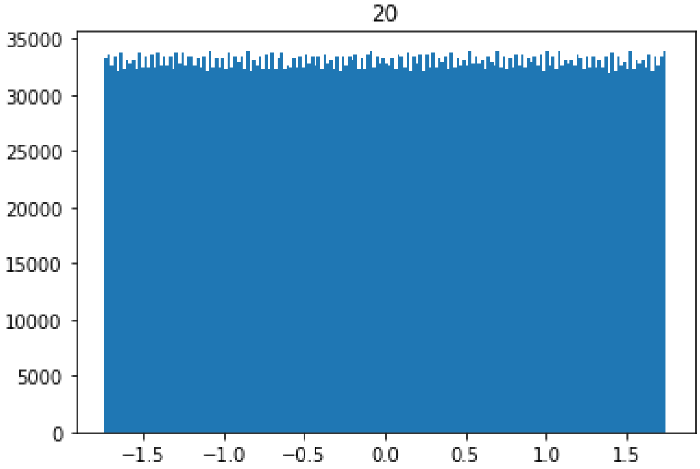
18

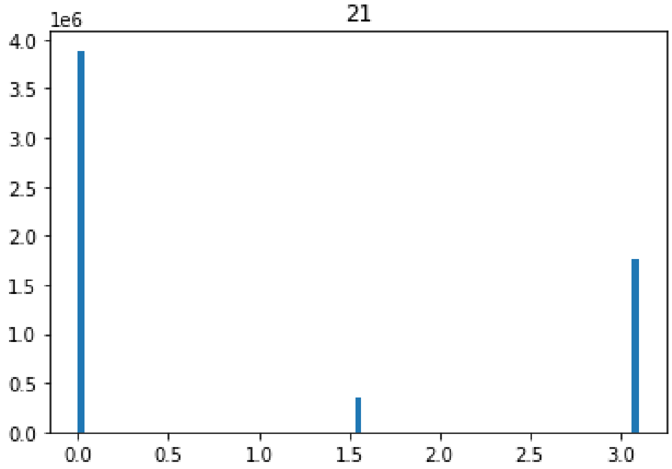

40000
30000
20000
10000
0
0
2
4
6
8
10
12

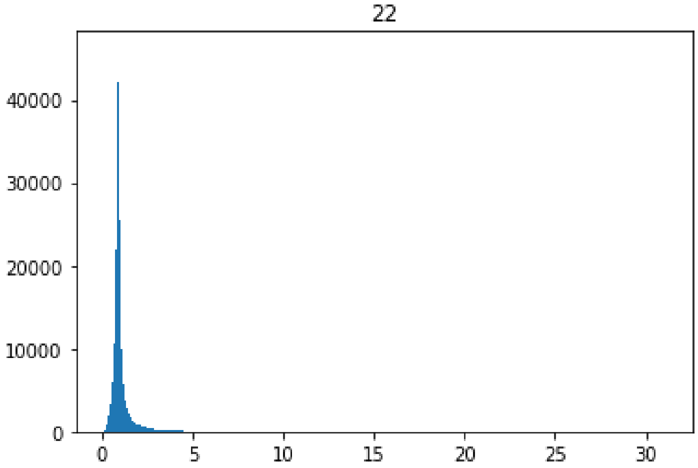
19

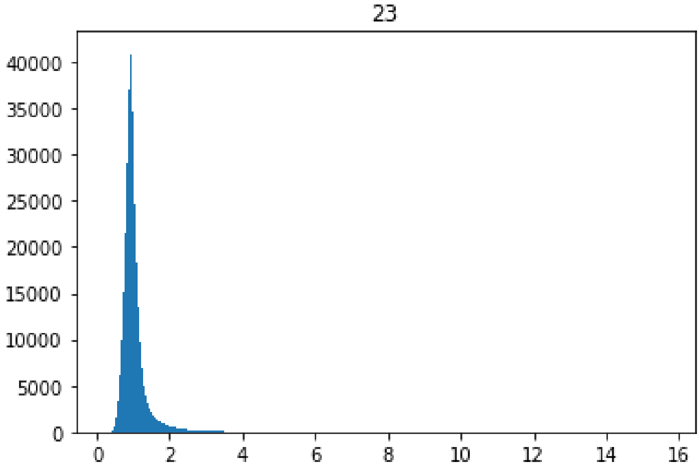

35000
30000
25000
20000
15000
10000

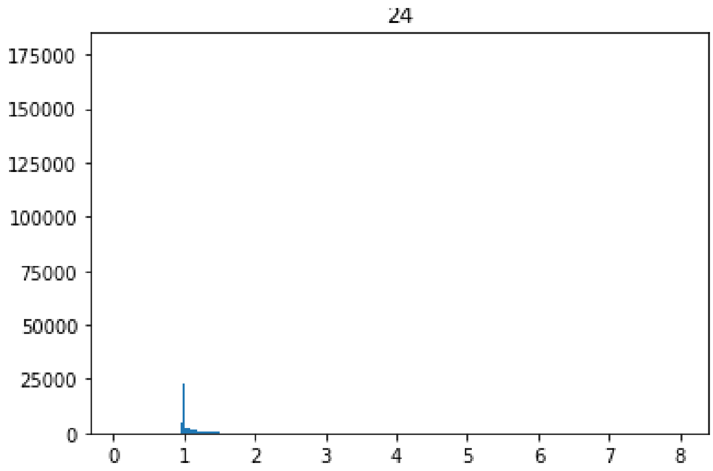

10000
5000
0
−2
−1
0
1
2

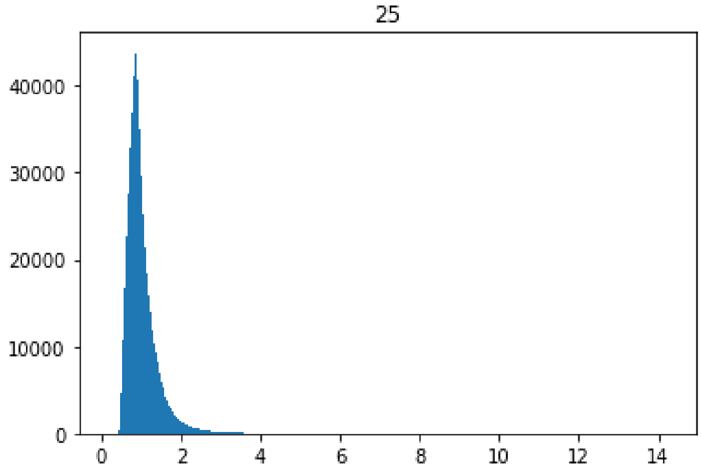

20
35000
30000
25000
20000
15000
10000
5000
0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

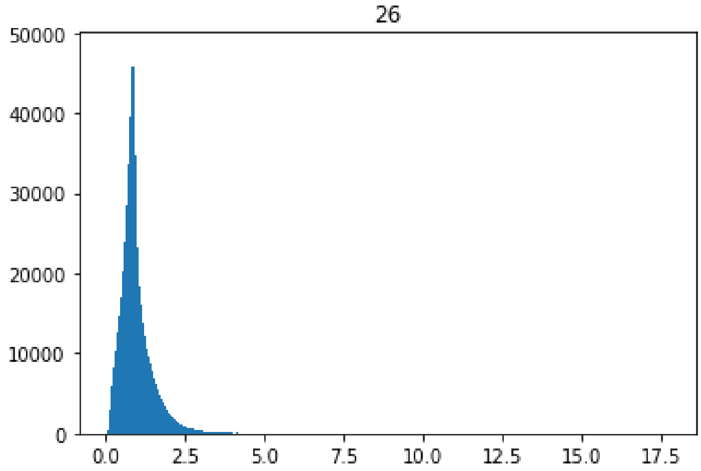

21
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


22
40000
30000
20000
10000
0
0
5
10
15
20
25
30

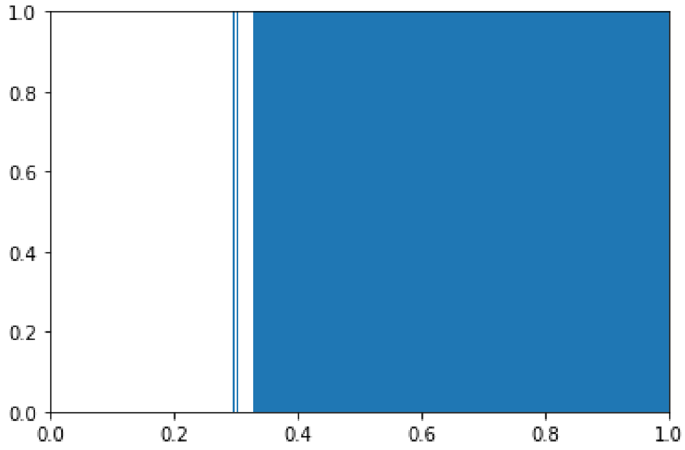

23
40000
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16







```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-9-8e3c99999bc5> in <module>()
      1 for i in range(29):
----> 2   plt.hist(dataset[:,i],bins='auto')
      3   plt.title(str(i))
      4   plt.show()

/usr/local/lib/python3.7/dist-packages/matplotlib/pyplot.py in hist(x, bins, range, densi
ty, weights, cumulative, bottom, histtype, align, orientation, rwidth, log, color, label,
stacked, data, **kwargs)
   2608         align=align, orientation=orientation, rwidth=rwidth, log=log,
   2609         color=color, label=label, stacked=stacked, **({"data": data}
-> 2610         if data is not None else {}), **kwargs)
   2611 
   2612 

/usr/local/lib/python3.7/dist-packages/matplotlib/__init__.py in inner(ax, data, *args, *
*kwargs)
   1563     def inner(ax, *args, data=None, **kwargs):
   1564         if data is None:
-> 1565             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1566 
   1567         bound = new_sig.bind(ax, *args, **kwargs)

/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_axes.py in hist(self, x, bins, range, density, weights, cumulative, bottom, histtype, align, orientation, rwidth, log, color, label, stacked, **kwargs)
   6727                 patch = _barfunc(bins[:-1]+boffset, height, width,
   6728                                  align='center', log=log,
-> 6729                                  color=c, **{bottom_kwarg: bottom})
   6730                 patches.append(patch)
   6731                 if stacked:

/usr/local/lib/python3.7/dist-packages/matplotlib/__init__.py in inner(ax, data, *args, **kwargs)
   1563     def inner(ax, *args, data=None, **kwargs):
   1564         if data is None:
-> 1565             return func(ax, *map(sanitize_sequence, args), **kwargs)
   1566 
   1567         bound = new_sig.bind(ax, *args, **kwargs)

/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_axes.py in bar(self, x, height, width, bottom, align, **kwargs)
   2402             elif orientation == 'horizontal':
   2403                 r.sticky_edges.x.append(l)
-> 2404             self.add_patch(r)
   2405             patches.append(r)
   2406 

/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_base.py in add_patch(self, p)
   1917         if p.get_clip_path() is None:
   1918             p.set_clip_path(self.patch)
-> 1919         self._update_patch_limits(p)
   1920         self.patches.append(p)
   1921         p._remove_method = self.patches.remove

/usr/local/lib/python3.7/dist-packages/matplotlib/axes/_base.py in _update_patch_limits(self, patch)
   1943                 xys = patch_to_data.transform(xys)
   1944 
-> 1945             updatex, updatey = patch.get_transform().\
   1946                 contains_branch_seperately(self.transData)
   1947             self.update_datalim(xys, updatex=updatex,

/usr/local/lib/python3.7/dist-packages/matplotlib/patches.py in get_transform(self)
    260     def get_transform(self):
    261         """Return the `~.transforms.Transform` applied to the `Patch`."""
--> 262         return self.get_patch_transform() + artist.Artist.get_transform(self)
    263 
    264     def get_data_transform(self):

/usr/local/lib/python3.7/dist-packages/matplotlib/patches.py in get_patch_transform(self)
    775 
    776     def get_patch_transform(self):
--> 777         self._update_patch_transform()
    778         return self._rect_transform
    779 

/usr/local/lib/python3.7/dist-packages/matplotlib/patches.py in _update_patch_transform(self)
    754         """
    755         x0, y0, x1, y1 = self._convert_units()
--> 756         bbox = transforms.Bbox.from_extents(x0, y0, x1, y1)
    757         rot_trans = transforms.Affine2D()
    758         rot_trans.rotate_deg_around(x0, y0, self.angle)

/usr/local/lib/python3.7/dist-packages/matplotlib/transforms.py in from_extents(*args)
    787         The *y*-axis increases upwards.
    788         """
--> 789         points = np.array(args, dtype=float).reshape(2, 2)
    790         return Bbox(points)
    791 

KeyboardInterrupt: 
```



In [ ]:

```
X = dataset[:,1:]
Y = dataset[:,0].astype(int)
print(X[0],Y[0])
print(np.shape(X),np.shape(X[0]),np.shape(Y),np.shape(Y[0]))
```

```
[ 9.07542109e-01  3.29147279e-01  3.59411865e-01  1.49796987e+00
 -3.13009530e-01  1.09553063e+00 -5.57524920e-01 -1.58822978e+00
  2.17307615e+00  8.12581182e-01 -2.13641927e-01  1.27101457e+00
  2.21487212e+00  4.99993950e-01 -1.26143181e+00  7.32156157e-01
  0.00000000e+00  3.98700893e-01 -1.13893008e+00 -8.19110195e-04
  0.00000000e+00  3.02219898e-01  8.33048165e-01  9.85699654e-01
  9.78098392e-01  7.79732168e-01  9.92355764e-01  7.98342586e-01] 1
(5999999, 28) (28,) (5999999,) ()
```

Model on Raw dataset (no preprocessing)

You can skip training, jump to the last cell to load from drive

In [ ]:

```
seed = 7
test_size = 0.08
X_train, X_valid, y_train, y_valid = train_test_split(X, Y, test_size=test_size, random_
state=seed)
```

In [ ]:

```
eval_set = [(X_valid, y_valid)]
model = XGBClassifier()
model.fit(X_train, y_train, eval_metric="auc", eval_set=eval_set, verbose=True)
```

```
[20:38:34] WARNING: /workspace/src/learner.cc:686: Tree method is automatically selected
to be 'approx' for faster speed. To use old behavior (exact greedy algorithm on single ma
chine), set tree_method to 'exact'.
[0] validation_0-auc:0.679592
[1] validation_0-auc:0.684988
[2] validation_0-auc:0.688146
[3] validation_0-auc:0.692478
[4] validation_0-auc:0.708175
[5] validation_0-auc:0.71577
[6] validation_0-auc:0.723524
[7] validation_0-auc:0.728979
[8] validation_0-auc:0.730521
[9] validation_0-auc:0.731517
[10] validation_0-auc:0.736425
[11] validation_0-auc:0.736944
[12] validation_0-auc:0.740219
[13] validation_0-auc:0.743769
[14] validation_0-auc:0.745711
[15] validation_0-auc:0.745926
[16] validation_0-auc:0.747654
```

```
[17] validation_0-auc:0.749516
[18] validation_0-auc:0.751581
[19] validation_0-auc:0.753539
[20] validation_0-auc:0.755207
[21] validation_0-auc:0.756212
[22] validation_0-auc:0.758195
[23] validation_0-auc:0.759694
[24] validation_0-auc:0.760278
[25] validation_0-auc:0.761302
[26] validation_0-auc:0.762495
[27] validation_0-auc:0.763558
[28] validation_0-auc:0.764722
[29] validation_0-auc:0.76556
[30] validation_0-auc:0.766682
[31] validation_0-auc:0.767155
[32] validation_0-auc:0.768227
[33] validation_0-auc:0.768552
[34] validation_0-auc:0.76924
[35] validation_0-auc:0.770104
[36] validation_0-auc:0.770717
[37] validation_0-auc:0.771381
[38] validation_0-auc:0.77207
[39] validation_0-auc:0.772801
[40] validation_0-auc:0.77311
[41] validation_0-auc:0.773857
[42] validation_0-auc:0.774087
[43] validation_0-auc:0.774715
[44] validation_0-auc:0.775203
[45] validation_0-auc:0.775696
[46] validation_0-auc:0.776102
[47] validation_0-auc:0.776552
[48] validation_0-auc:0.777135
[49] validation_0-auc:0.777567
[50] validation_0-auc:0.777888
[51] validation_0-auc:0.778765
[52] validation_0-auc:0.77896
[53] validation_0-auc:0.779252
[54] validation_0-auc:0.779619
[55] validation_0-auc:0.779822
[56] validation_0-auc:0.779892
[57] validation_0-auc:0.780381
[58] validation_0-auc:0.780899
[59] validation_0-auc:0.781338
[60] validation_0-auc:0.781614
[61] validation_0-auc:0.781701
[62] validation_0-auc:0.781829
[63] validation_0-auc:0.78201
[64] validation_0-auc:0.782281
[65] validation_0-auc:0.782884
[66] validation_0-auc:0.783448
[67] validation_0-auc:0.783688
[68] validation_0-auc:0.784032
[69] validation_0-auc:0.784123
[70] validation_0-auc:0.784336
[71] validation_0-auc:0.784749
[72] validation_0-auc:0.785088
[73] validation_0-auc:0.785265
[74] validation_0-auc:0.785427
[75] validation_0-auc:0.785509
[76] validation_0-auc:0.785724
[77] validation_0-auc:0.786082
[78] validation_0-auc:0.786251
[79] validation_0-auc:0.786409
[80] validation_0-auc:0.786623
[81] validation_0-auc:0.78675
[82] validation_0-auc:0.787107
[83] validation_0-auc:0.787251
[84] validation_0-auc:0.787678
[85] validation_0-auc:0.787871
[86] validation_0-auc:0.78826
[87] validation_0-auc:0.788387
[88] validation_0-auc:0.788568
```

```
[89] validation_0-auc:0.788789
[90] validation_0-auc:0.788916
[91] validation_0-auc:0.789111
[92] validation_0-auc:0.789268
[93] validation_0-auc:0.789334
[94] validation_0-auc:0.789485
[95] validation_0-auc:0.789628
[96] validation_0-auc:0.789814
[97] validation_0-auc:0.789979
[98] validation_0-auc:0.790176
[99] validation_0-auc:0.7904
```

Out[ ]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [ ]:

```
y_pred = model.predict(X_valid)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_valid, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 71.40%
```

In [ ]:

```
mod_dataset=dataset
logtransform=[1,4,6,10,14,18,22,23,24,25,26,27,28]
for index in logtransform:
  mod_dataset[:,index]=np.log(mod_dataset[:,index])
```

In [ ]:

```
for i in range(29):
  plt.hist(mod_dataset[:,i],bins='auto')
  plt.title(str(i))
  plt.show()
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
categorical=[0,9,13,17,21]
non_categorical=[]
for i in range(29):
  if i not in categorical:
    non_categorical.append(i)
print(non_categorical)
```

In [ ]:

```
for index in non_categorical:
  mod_dataset[:,index]=scaler.fit_transform(mod_dataset[:,index].reshape(-1,1)).reshape(
-1)
```

In [ ]:

```
for i in range(29):
  plt.hist(mod_dataset[:,i],bins='auto')
  plt.title(str(i))
  plt.show()
```

In [ ]:

```
X = mod_dataset[:,1:]
```

```
Y = mod_dataset[:,0].astype(int)
print(X[0],Y[0])
print(np.shape(X),np.shape(X[0]),np.shape(Y),np.shape(Y[0]))
```

```
[ 9.91932443e-02  3.26245081e-01  3.56807259e-01  8.94439464e-01
 -3.11098036e-01  4.54615981e-01 -5.51916670e-01 -1.57885773e+00
  2.17307615e+00 -1.99794457e-01 -2.11991992e-01  1.26350201e+00
  2.21487212e+00 -1.24524704e+00 -1.25092734e+00  7.27702034e-01
  0.00000000e+00 -1.69832846e+00 -1.13024130e+00 -8.23977171e-04
  0.00000000e+00 -2.77900602e+00 -6.04251123e-01 -4.34355339e-01
  8.12042371e-02 -1.89601671e-01  2.57272610e-02 -5.25906146e-01] 1
(5999999, 28) (28,) (5999999,) ()
```

Model on Processed Dataset

- **To some features logarithm transform applied**
- **Standard Scaled**
- **Min Max Scaled**
- **Categorical features rounded to integers**

You can skip training, jump to the last cell to load from drive

In [ ]:

```
seed = 7
test_size = 0.08
X_train, X_valid, y_train, y_valid = train_test_split(X, Y, test_size=test_size, random_
state=seed)
```

In [ ]:

```
eval_set = [(X_valid, y_valid)]
model = XGBClassifier()
eval_set = [(X_valid, y_valid)]
model.fit(X_train, y_train, eval_metric="auc", eval_set=eval_set, verbose=True)
```

In [ ]:

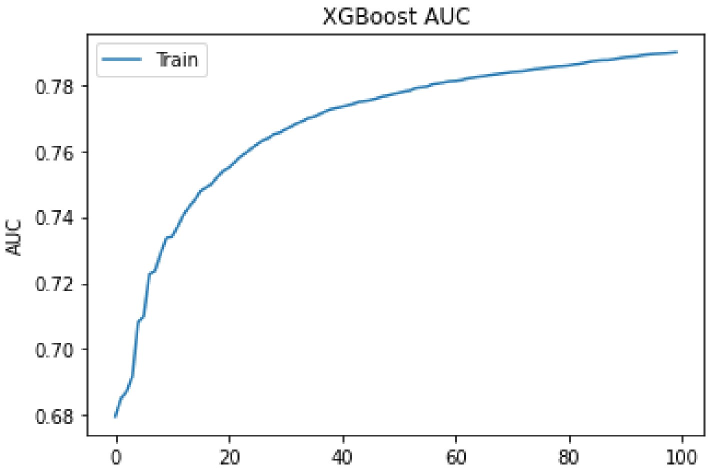
```
y_pred = model.predict(X_valid)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_valid, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 71.38%
```

In [ ]:

```
from matplotlib import pyplot

results = model.evals_result()
print(results)
epochs = len(results['validation_0']['auc'])
x_axis = range(0, epochs)
# plot log loss
fig, ax = pyplot.subplots()
ax.plot(x_axis, results['validation_0']['auc'], label='Train')
ax.legend()
pyplot.ylabel('AUC')
pyplot.title('XGBoost AUC')
pyplot.show()
```

```
{'validation_0': {'auc': [0.679592, 0.68499, 0.687248, 0.691798, 0.708075, 0.709898, 0.72
2665, 0.723656, 0.729072, 0.733647, 0.734079, 0.737091, 0.740615, 0.74296, 0.745188, 0.74
7834, 0.748963, 0.750014, 0.752192, 0.753838, 0.754897, 0.756489, 0.758168, 0.759406, 0.7
60729, 0.762003, 0.763188, 0.763876, 0.765118, 0.765563, 0.76654, 0.767364, 0.768302, 0.7
68978, 0.769943, 0.770313, 0.771099, 0.771922, 0.772615, 0.773035, 0.773421, 0.773835, 0.
774293, 0.77491, 0.775118, 0.775384, 0.775844, 0.77643, 0.776753, 0.777219, 0.777556, 0.7
78001, 0.778301, 0.779054, 0.779275, 0.779472, 0.780207, 0.780469, 0.780745, 0.781102, 0.
78121, 0.781446, 0.781914, 0.78219, 0.782477, 0.782645, 0.782943, 0.783189, 0.783428, 0.7
83651, 0.783935, 0.784048, 0.784195, 0.784507, 0.784786, 0.784964, 0.785211, 0.785401, 0.
785639, 0.785742, 0.785923, 0.786196, 0.78641, 0.78677, 0.78718, 0.787388, 0.78755, 0.787
582, 0.787816, 0.788135, 0.788397, 0.788557, 0.788672, 0.789009, 0.789217, 0.78939, 0.789
482, 0.78959, 0.789736, 0.789893]}}
```



Load Model from Drive

In [ ]:

```
import pickle
pickle.dump(model, open("xgb.pickle.dat", "wb"))
```

In [ ]:

```
"""
Checking whether load and save is working okay
"""
loaded_model = pickle.load(open("xgb.pickle.dat", "rb"))
y_pred = loaded_model.predict(X_valid)
predictions = [round(value) for value in y_pred]
accuracy = accuracy_score(y_valid, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

In [ ]:

```
"""
too slow abandoned after 10 minutes
"""
# df = pd.DataFrame(dataset, columns=np.arange(29)).sample(10)
# sns.pairplot(df)
```

In [ ]:

```
"""
took too long to run, abandoned after 14 minutes
"""
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn import metrics
# knn=KNeighborsClassifier(n_neighbors=2)
# knn.fit(X_train,y_train)
# y_pred=knn.predict(X_valid)
# score=metrics.accuracy_score(y_valid,y_pred)
# print(score)
```