Milestone-4 AutoEncoder Deep learning Approach

```
# import library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Tasks of Milestone - 4

- Data Preprocessing
- Anomaly Detection with Auto encoders
- visualization of results when using AutoEncoder approach

Read the data

```
# Preprocessed dataset
org_df=pd.read_csv('/content/preprocessed.csv')
org_df.head()
```

	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	Part
0	MD	F	I	SAINT LOUIS	MO	US	Internal Medicine	
1	MD	F	I	FAYETTEVILLE	NC	US	Obstetrics & Gynecology	
2	DPM	M	I	NORTH HAVEN	CT	US	Podiatry	
3	MD	M	I	KANSAS CITY	MO	US	Internal Medicine	
4	DO	M	I	JUPITER	FL	US	Internal Medicine	

Next steps: Generate code with org_df View recommended plots

```
org_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column                                                  Non-Null Count   Dtype
---  ------                                                  --------------   -----
 0   Credentials of the Provider                             100000 non-null  object
 1   Gender of the Provider                                  100000 non-null  object
 2   Entity Type of the Provider                             100000 non-null  object
 3   City of the Provider                                    100000 non-null  object
 4   State Code of the Provider                              100000 non-null  object
 5   Country Code of the Provider                            100000 non-null  object
 6   Provider Type                                           100000 non-null  object
 7   Medicare Participation Indicator                        100000 non-null  object
 8   Place of Service                                        100000 non-null  object
 9   HCPCS Code                                              100000 non-null  object
 10  HCPCS Drug Indicator                                    100000 non-null  object
 11  Number of Services                                      100000 non-null  float64
 12  Number of Medicare Beneficiaries                        100000 non-null  float64
 13  Number of Distinct Medicare Beneficiary/Per Day Services  100000 non-null  float64
```

```
 14  Average Medicare Allowed Amount                      100000 non-null  float64
 15  Average Submitted Charge Amount                      100000 non-null  float64
 16  Average Medicare Payment Amount                      100000 non-null  float64
 17  Average Medicare Standardized Amount                 100000 non-null  float64
dtypes: float64(7), object(11)
memory usage: 13.7+ MB
```

```
# Make a copy of original data for the visualization
df=org_df.copy()
```

1 - Data Preprocessing

```
columns=['Credentials of the Provider', 'Gender of the Provider',
       'Entity Type of the Provider', 'City of the Provider',
       'State Code of the Provider', 'Country Code of the Provider',
       'Provider Type', 'Medicare Participation Indicator', 'Place of Service',
       'HCPCS Code', 'HCPCS Drug Indicator']
for i in columns:
    if org_df[i].nunique() >5:
        print(f"categorical values in {i} :",org_df[i].nunique())
print("\n")
for i in columns:
    if org_df[i].nunique() <5:
        print(f"categorical values in {i} :",org_df[i].nunique())
```

```
categorical values in Credentials of the Provider : 1539
categorical values in City of the Provider : 5846
categorical values in State Code of the Provider : 58
categorical values in Provider Type : 90
categorical values in HCPCS Code : 2631


categorical values in Gender of the Provider : 3
categorical values in Entity Type of the Provider : 2
categorical values in Country Code of the Provider : 4
categorical values in Medicare Participation Indicator : 2
categorical values in Place of Service : 2
categorical values in HCPCS Drug Indicator : 2
```

Frequency encoding of columns which having more than two categorical values

```
# Columns to be frequency encoded
frequency_encode_cols = ['Credentials of the Provider', 'City of the Provider','State Code of the
                         'Provider Type', 'HCPCS Code','Gender of the Provider','Country Code of t

# Function to perform frequency encoding
def frequency_encoding(df, columns):
    for column in columns:
        frequency = df[column].value_counts()
        df[column] = df[column].map(frequency)
    return df

# Apply frequency encoding
encoded_data = frequency_encoding(org_df, frequency_encode_cols)
encoded_data.head()
```

	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Drug Indicator	Number of Services	N Benef
0	73827	29105	I	500	1997	99994	11366	Y	F	1297	N	27.0	
1	73827	29105	I	209	3725	99994	1028	Y	O	243	N	175.0	
2	1915	66641	I	10	1403	99994	2027	Y	O	44	N	32.0	
3	73827	66641	I	317	1997	99994	11366	Y	O	460	N	20.0	
4	6176	66641	I	51	7263	99994	11366	Y	O	732	N	33.0	

Next steps: Generate code with encoded_data View recommended plots

One Hot Encoding for binary categorical columns

```
cols=['Entity Type of the Provider','Medicare Participation Indicator','Place of Service',
'HCPCS Drug Indicator']
```

```
new_df=pd.get_dummies(encoded_data,columns=cols,dtype=float)
new_df.head()
```

	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	HCPCS Code	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	...	Average Medicare Payment Amount	S
0	73827	29105	500	1997	99994	11366	1297	27.0	24.0	27.0	...	157.262222	
1	73827	29105	209	3725	99994	1028	243	175.0	175.0	175.0	...	118.830000	
2	1915	66641	10	1403	99994	2027	44	32.0	13.0	32.0	...	64.439688	
3	73827	66641	317	1997	99994	11366	460	20.0	18.0	20.0	...	3.430000	
4	6176	66641	51	7263	99994	11366	732	33.0	24.0	31.0	...	19.539394	

5 rows × 22 columns

Standardized the data

```
# Standardized the data
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()

numerical_cols=new_df.iloc[:,:14].columns
scaled_data=ss.fit_transform(new_df[numerical_cols])
temp_df=pd.DataFrame(scaled_data,columns=new_df.iloc[:,:14].columns)
temp_df.head()
```

	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	HCPCS Code	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount	Av Subm C A
0	0.594983	-1.211160	1.571686	-0.737342	0.007746	1.336743	0.397579	-0.085301	-0.059308	-0.070183	0.385450	-0.0
1	0.594983	-1.211160	0.189180	-0.004973	0.007746	-0.940500	-0.439989	-0.025939	0.076775	0.020049	0.086673	0.1
2	-1.684316	0.686478	-0.756245	-0.989093	0.007746	-0.720441	-0.598126	-0.083296	-0.069222	-0.067135	-0.041922	-0.1
3	0.594983	0.686478	0.702275	-0.737342	0.007746	1.336743	-0.267549	-0.088109	-0.064716	-0.074451	-0.380709	-0.3
4	-1.549260	0.686478	-0.561459	1.494517	0.007746	1.336743	-0.051402	-0.082895	-0.059308	-0.067744	-0.291221	-0.2

Next steps: Generate code with temp_df View recommended plots

```
# creating final transformed dataset after scaling
scaled_df=temp_df.join(new_df.iloc[:,14:])
scaled_df.head()
```

	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	HCPCS Code	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	...	Average Medicare Payment Amount
0	0.594983	-1.211160	1.571686	-0.737342	0.007746	1.336743	0.397579	-0.085301	-0.059308	-0.070183	...	0.400082
1	0.594983	-1.211160	0.189180	-0.004973	0.007746	-0.940500	-0.439989	-0.025939	0.076775	0.020049	...	0.207649
2	-1.684316	0.686478	-0.756245	-0.989093	0.007746	-0.720441	-0.598126	-0.083296	-0.069222	-0.067135	...	-0.064687
3	0.594983	0.686478	0.702275	-0.737342	0.007746	1.336743	-0.267549	-0.088109	-0.064716	-0.074451	...	-0.370166
4	-1.549260	0.686478	-0.561459	1.494517	0.007746	1.336743	-0.051402	-0.082895	-0.059308	-0.067744	...	-0.289505

5 rows × 22 columns

```
scaled_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 22 columns):
 #   Column                                                   Non-Null Count   Dtype
---  ------                                                   --------------   -----
 0   Credentials of the Provider                              100000 non-null  float64
 1   Gender of the Provider                                   100000 non-null  float64
 2   City of the Provider                                     100000 non-null  float64
 3   State Code of the Provider                               100000 non-null  float64
 4   Country Code of the Provider                             100000 non-null  float64
 5   Provider Type                                            100000 non-null  float64
 6   HCPCS Code                                               100000 non-null  float64
 7   Number of Services                                       100000 non-null  float64
 8   Number of Medicare Beneficiaries                         100000 non-null  float64
 9   Number of Distinct Medicare Beneficiary/Per Day Services 100000 non-null  float64
 10  Average Medicare Allowed Amount                          100000 non-null  float64
 11  Average Submitted Charge Amount                          100000 non-null  float64
 12  Average Medicare Payment Amount                          100000 non-null  float64
 13  Average Medicare Standardized Amount                     100000 non-null  float64
 14  Entity Type of the Provider_I                            100000 non-null  float64
 15  Entity Type of the Provider_O                            100000 non-null  float64
 16  Medicare Participation Indicator_N                       100000 non-null  float64
 17  Medicare Participation Indicator_Y                       100000 non-null  float64
 18  Place of Service_F                                       100000 non-null  float64
 19  Place of Service_O                                       100000 non-null  float64
 20  HCPCS Drug Indicator_N                                   100000 non-null  float64
 21  HCPCS Drug Indicator_Y                                   100000 non-null  float64
dtypes: float64(22)
memory usage: 16.8 MB
```

2 - Auto Encoders – Deep Learning

```
# spliting the data
from sklearn.model_selection import train_test_split
X_train,X_test=train_test_split(scaled_df,test_size=0.2,random_state=42)
```

Model Building

```
from keras.models import Model, Sequential
from keras.layers import Input, Dense
from keras import regularizers,optimizers
from keras.layers import Dropout
```

```
# Build the autoencoder model
input_dim = X_train.shape[1]
encoding_dim = 16
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = int(encoding_dim / 2)
hidden_dim3 = int(encoding_dim / 2)

# Input layer
input_layer = Input(shape=(input_dim,))

# Encoding layer
encoder = Dense(encoding_dim, activation='relu', activity_regularizer=regularizers.l1(10e-5))(inpu
encoder = Dense(hidden_dim1, activation='relu')(encoder)
encoder = Dense(hidden_dim2, activation='relu')(encoder)
encoder = Dense(hidden_dim3, activation='relu')(encoder)
encoder = Dropout(0.2)(encoder)

# # Bottleneck layer
# bottleneck = Dense(encoding_dim, activation='relu')(encoder)

# Decoding layer
decoder = Dense(hidden_dim2, activation='relu')(encoder)
decoder = Dense(hidden_dim1, activation='relu')(decoder)
decoder = Dense(encoding_dim, activation='relu')(decoder)
decoder = Dense(input_dim, activation='sigmoid')(decoder)

# Autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)
```

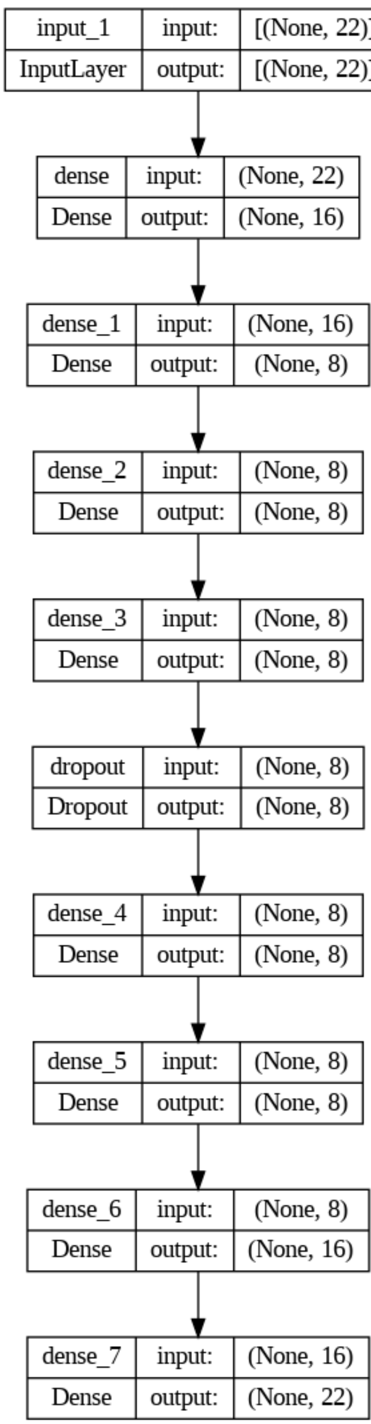
```
# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse'])
```

```
# summary of the model
autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 22)]              0         
                                                                 
 dense (Dense)               (None, 16)                368       
                                                                 
 dense_1 (Dense)             (None, 8)                 136       
                                                                 
 dense_2 (Dense)             (None, 8)                 72        
                                                                 
 dense_3 (Dense)             (None, 8)                 72        
                                                                 
 dropout (Dropout)           (None, 8)                 0         
```

```
 dense_4 (Dense)             (None, 8)                 72

 dense_5 (Dense)             (None, 8)                 72

 dense_6 (Dense)             (None, 16)                144

 dense_7 (Dense)             (None, 22)                374

=================================================================
Total params: 1310 (5.12 KB)
Trainable params: 1310 (5.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

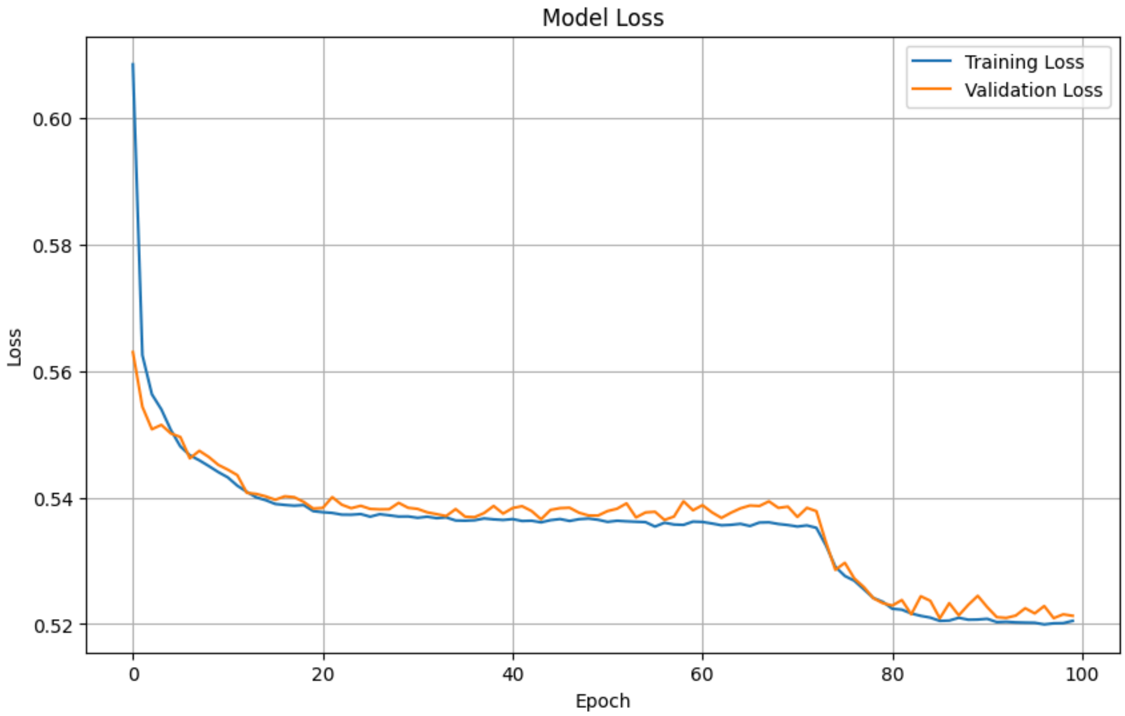
```
# plot the model
import tensorflow as tf
tf.keras.utils.plot_model(autoencoder, to_file='model.png',show_shapes=True)
```



```
# Train the autoencoder
history = autoencoder.fit(X_train, X_train, epochs=100, batch_size=32, shuffle=True, validation_da
```

```
Epoch 55/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5361 - mse: 0.5358 - val_loss: 0.5377 - val_mse: 0.5373
Epoch 56/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5354 - mse: 0.5351 - val_loss: 0.5378 - val_mse: 0.5374
Epoch 57/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5360 - mse: 0.5357 - val_loss: 0.5365 - val_mse: 0.5361
Epoch 58/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5357 - mse: 0.5354 - val_loss: 0.5370 - val_mse: 0.5366
Epoch 59/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5357 - mse: 0.5353 - val_loss: 0.5394 - val_mse: 0.5390
Epoch 60/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5362 - mse: 0.5358 - val_loss: 0.5380 - val_mse: 0.5376
Epoch 61/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5361 - mse: 0.5358 - val_loss: 0.5388 - val_mse: 0.5384
Epoch 62/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5359 - mse: 0.5355 - val_loss: 0.5377 - val_mse: 0.5373
Epoch 63/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5356 - mse: 0.5353 - val_loss: 0.5368 - val_mse: 0.5364
Epoch 64/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5357 - mse: 0.5353 - val_loss: 0.5376 - val_mse: 0.5372
Epoch 65/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5358 - mse: 0.5355 - val_loss: 0.5383 - val_mse: 0.5379
Epoch 66/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5355 - mse: 0.5351 - val_loss: 0.5387 - val_mse: 0.5384
Epoch 67/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5361 - mse: 0.5357 - val_loss: 0.5387 - val_mse: 0.5383
Epoch 68/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5361 - mse: 0.5357 - val_loss: 0.5394 - val_mse: 0.5391
Epoch 69/100
2500/2500 [==============================] - 13s 5ms/step - loss: 0.5358 - mse: 0.5355 - val_loss: 0.5384 - val_mse: 0.5380
Epoch 70/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5357 - mse: 0.5353 - val_loss: 0.5386 - val_mse: 0.5382
Epoch 71/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5354 - mse: 0.5351 - val_loss: 0.5369 - val_mse: 0.5366
Epoch 72/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5356 - mse: 0.5352 - val_loss: 0.5384 - val_mse: 0.5380
Epoch 73/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5352 - mse: 0.5349 - val_loss: 0.5379 - val_mse: 0.5375
Epoch 74/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5324 - mse: 0.5321 - val_loss: 0.5330 - val_mse: 0.5326
Epoch 75/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5291 - mse: 0.5287 - val_loss: 0.5286 - val_mse: 0.5282
Epoch 76/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5276 - mse: 0.5272 - val_loss: 0.5297 - val_mse: 0.5293
Epoch 77/100
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5269 - mse: 0.5265 - val_loss: 0.5272 - val_mse: 0.5268
Epoch 78/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5255 - mse: 0.5251 - val_loss: 0.5259 - val_mse: 0.5254
Epoch 79/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5241 - mse: 0.5237 - val_loss: 0.5241 - val_mse: 0.5237
Epoch 80/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5235 - mse: 0.5231 - val_loss: 0.5233 - val_mse: 0.5229
Epoch 81/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5224 - mse: 0.5220 - val_loss: 0.5229 - val_mse: 0.5226
Epoch 82/100
2500/2500 [==============================] - 8s 3ms/step - loss: 0.5223 - mse: 0.5219 - val_loss: 0.5238 - val_mse: 0.5234
Epoch 83/100
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5217 - mse: 0.5213 - val_loss: 0.5215 - val_mse: 0.5212
Epoch 84/100
```

```
# Plot the training loss and validation loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```




```
# Predict the reconstruction
y_pred = autoencoder.predict(scaled_df.values)
```

```
3125/3125 [==============================] - 6s 2ms/step
```

```
y_pred.shape, scaled_df.shape
```

```
((100000, 22), (100000, 22))
```

```
y_pred
```

```
array([[1.8186918e-12, 0.0000000e+00, 9.9999946e-01, ..., 4.8804933e-09,
        1.0000000e+00, 9.1591036e-15],
       [5.4847018e-20, 0.0000000e+00, 0.0000000e+00, ..., 9.9999821e-01,
        1.0000000e+00, 0.0000000e+00],
       [1.4770133e-23, 0.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       ...,
       [3.8552137e-11, 0.0000000e+00, 9.9999893e-01, ..., 8.9694424e-10,
        1.0000000e+00, 5.8933674e-13],
       [6.7573482e-01, 0.0000000e+00, 0.0000000e+00, ..., 1.0000000e+00,
        1.0000000e+00, 0.0000000e+00],
       [8.6119712e-21, 0.0000000e+00, 4.0032431e-36, ..., 9.9961191e-01,
        1.0000000e+00, 0.0000000e+00]], dtype=float32)
```

```
scaled_df.values
```

```
array([[ 0.5949835 , -1.21116044,  1.57168633, ...,  0.        ,
         1.        ,  0.        ],
       [ 0.5949835 , -1.21116044,  0.18918015, ...,  1.        ,
         1.        ,  0.        ],
       [-1.68431551,  0.68647828, -0.75624504, ...,  1.        ,
         1.        ,  0.        ],
       ...,
       [ 0.5949835 ,  0.68647828,  1.57168633, ...,  0.        ,
         1.        ,  0.        ],
       [ 0.5949835 , -1.21116044, -0.77524856, ...,  1.        ,
         1.        ,  0.        ],
       [ 0.5949835 , -1.21116044, -0.4759431 , ...,  1.        ,
         1.        ,  0.        ]])
```

```
# Calculate the reconstruction error
mse = np.mean(np.power(scaled_df.values -  y_pred, 2), axis=1)
```

```
# Determine the threshold for anomaly detection
threshold = np.percentile(mse, 98)
```

```
# Classify anomalies
y_predict = mse > threshold
```

```
y_predict.shape
```

```
(100000,)
```

```
# creating a new columns for the Anomaly labels
df['Anomaly']=y_predict
```

```
# To get the rows which are anomalous
df[y_predict==1]
```

vider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Drug Indicator	Nu Serv
rology	Y	O	95811	N	
:ology- cology	Y	O	J1439	Y	15
opedic urgery	Y	F	27130	N	
trics & cology	Y	F	57425	N	
urgery	Y	F	22551	N	
...	...	...	...	...	
opedic urgery	Y	F	27447	N	
ıtology	Y	O	J0717	Y	43
ılatory urgical Center	Y	F	0191T	N	
ardiac urgery	Y	F	33533	N	
jnostic liology	Y	O	37227	N	

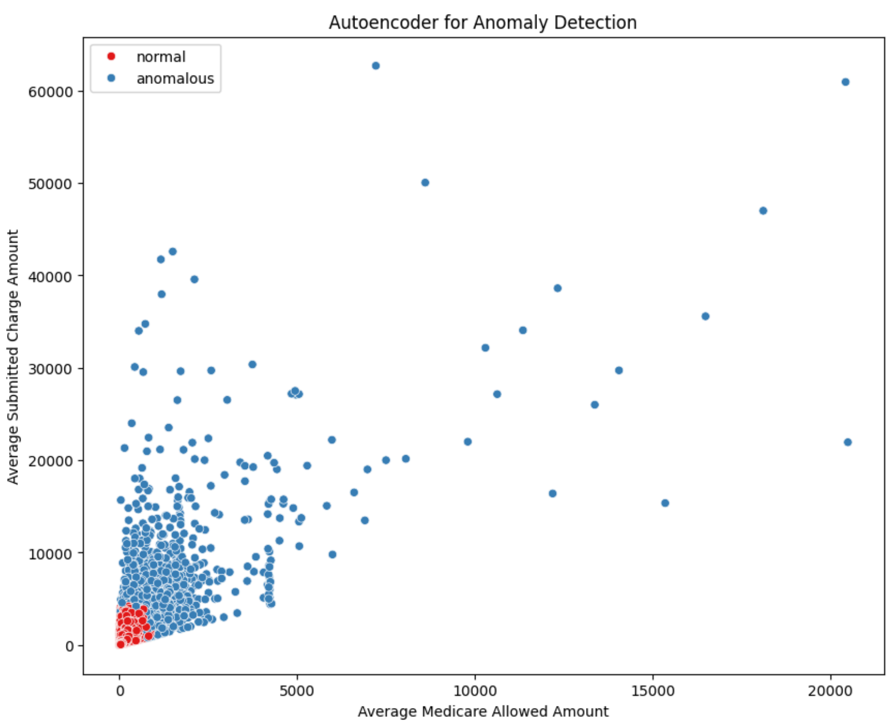
```
# assign labels as 'normal' and 'anomalous' for Normal and Anomalous data points
df=df.replace({True:'anomalous',False:'normal'})
df.head()
```

Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Drug Indicator	Number of Services
Internal Medicine	Y	F	99223	N	27.0
Obstetrics & Gynecology	Y	O	G0202	N	175.0
Podiatry	Y	O	99348	N	32.0
Internal Medicine	Y	O	81002	N	20.0
Internal Medicine	Y	O	96372	N	33.0

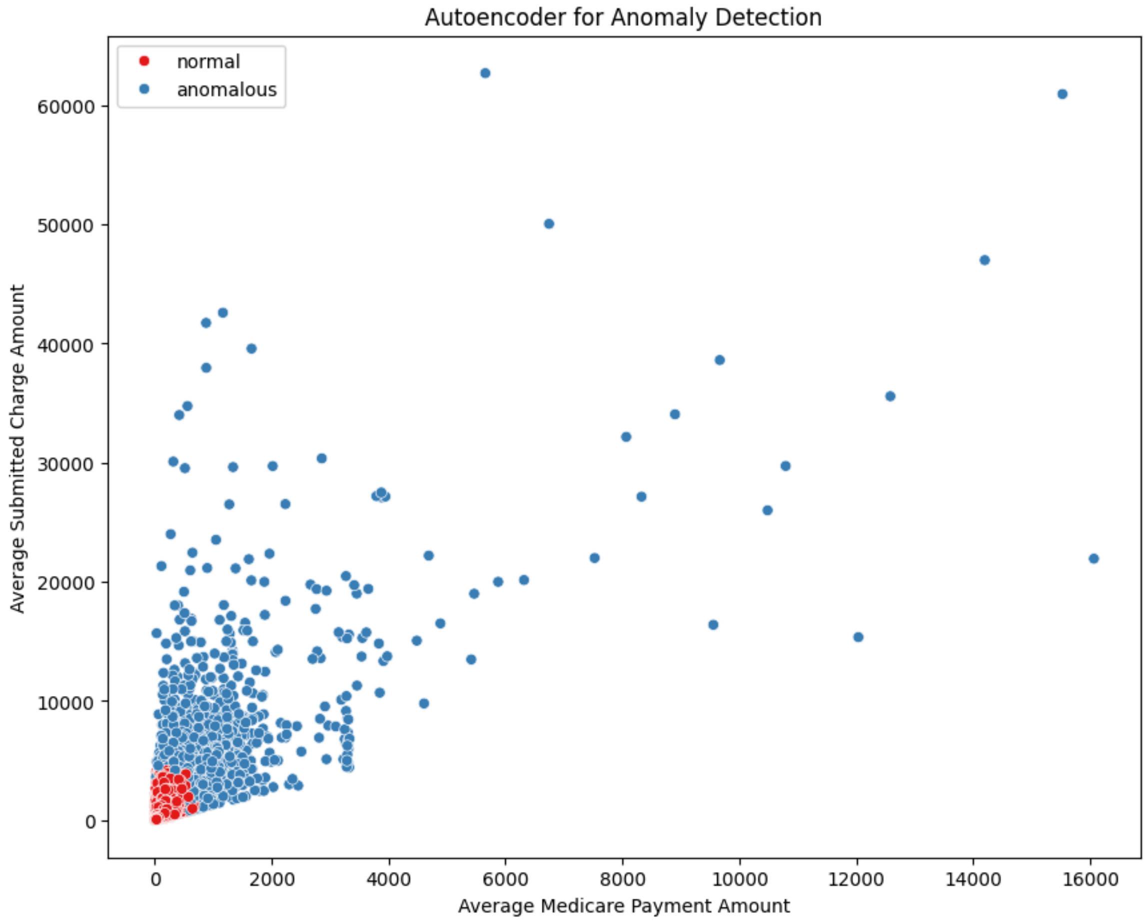
Next steps: Generate code with df View recommended plots

Visualizations

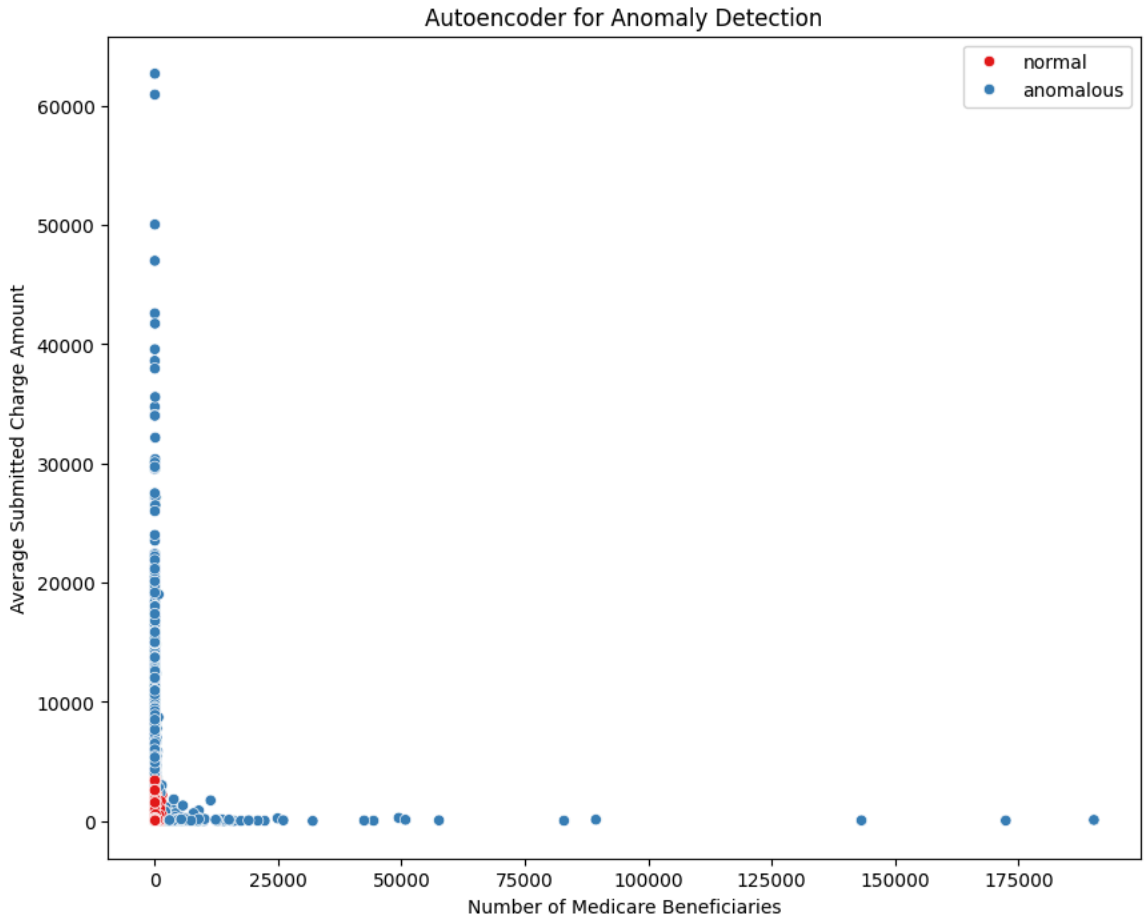
```
# Plot the data
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Average Medicare Allowed Amount', y='Average Submitted Charge Amount', data=df,
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Average Medicare Allowed Amount')
plt.ylabel ('Average Submitted Charge Amount')
plt.legend()
plt.show()
```



```
# Plot the data
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Average Medicare Payment Amount', y='Average Submitted Charge Amount', data=df,
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Average Medicare Payment Amount')
plt.ylabel ('Average Submitted Charge Amount')
plt.legend()
plt.show()
```



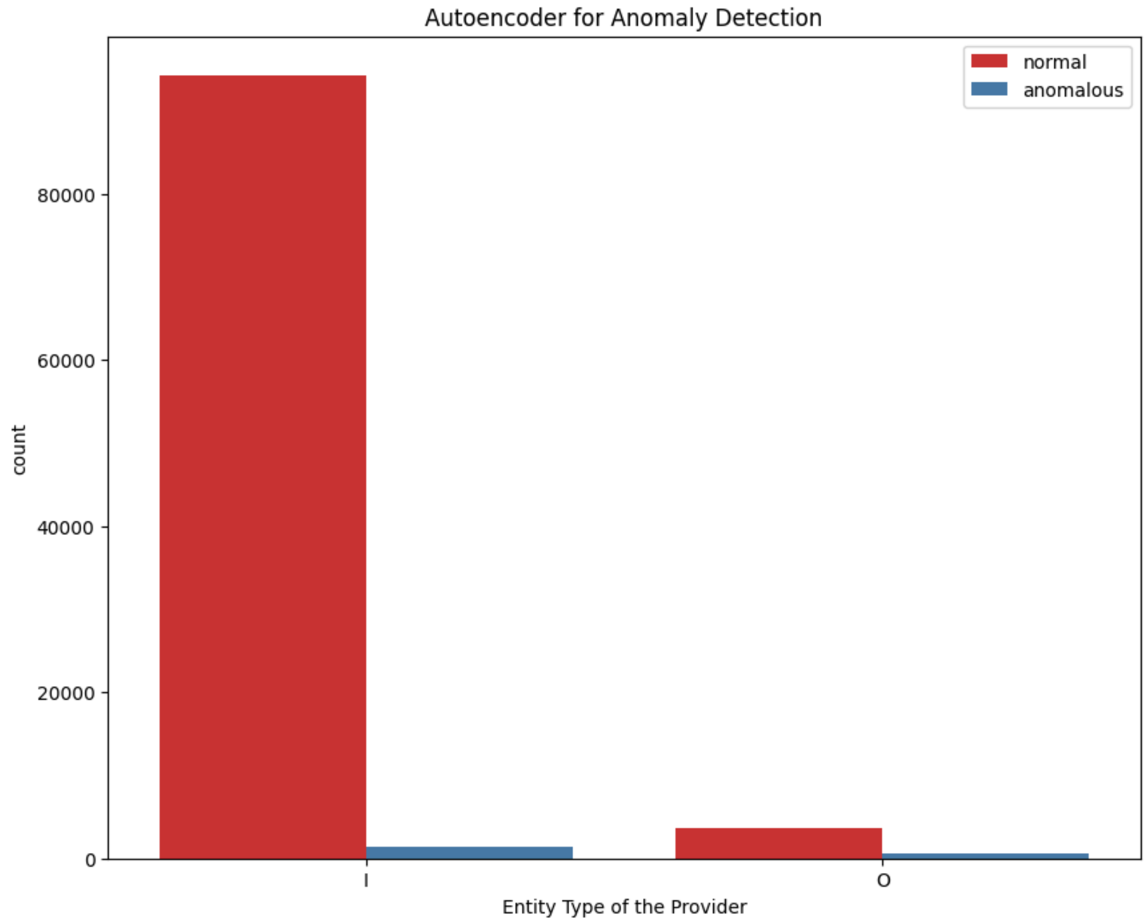

```
# Plot the data
plt.figure(figsize=(10, 8))
sns.scatterplot(x='Number of Medicare Beneficiaries', y='Average Submitted Charge Amount', data=df
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Number of Medicare Beneficiaries')
plt.ylabel ('Average Submitted Charge Amount')
plt.legend()
plt.show()
```



- Above scatter plots are between the different numerical columns.
- These plot shows the normal and anamalous datapoints as we can see in the plots.

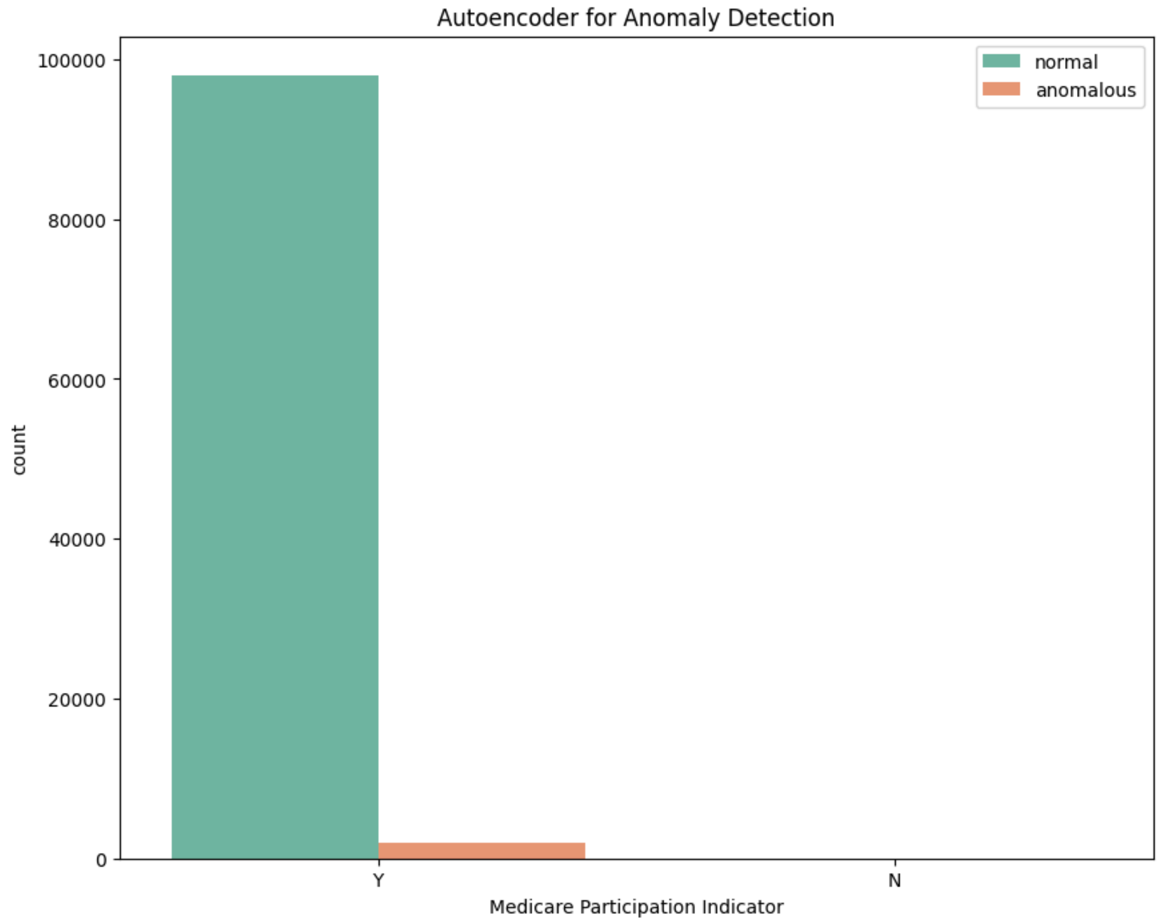
Visualization for categorical features

```
# Plot the data
plt.figure(figsize=(10, 8))
sns.countplot(x='Entity Type of the Provider', data=df, hue='Anomaly', palette='Set1')
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Entity Type of the Provider')
plt.legend()
plt.show()

# for the value counts of different categories
df[y_predict==1]['Entity Type of the Provider'].value_counts()
```



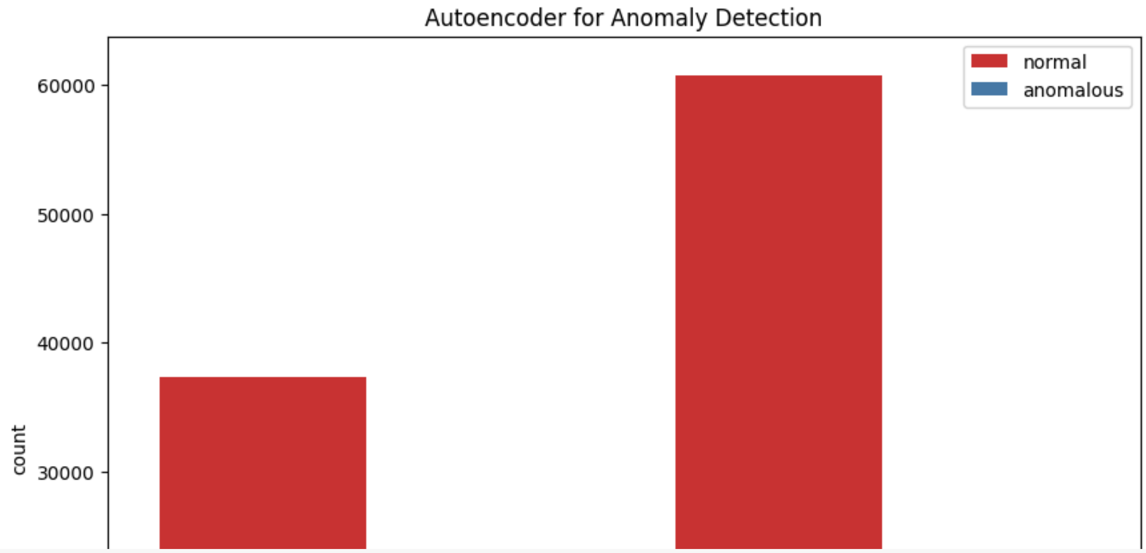

```
Entity Type of the Provider
I    1472
O     528
Name: count, dtype: int64
```

```
# Plot the data
plt.figure(figsize=(10, 8))
sns.countplot(x='Medicare Participation Indicator', data=df, hue='Anomaly', palette='Set2')
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Medicare Participation Indicator')
plt.legend()
plt.show()

# for the value counts of different categories
df[y_predict==1]['Medicare Participation Indicator'].value_counts()
```




```
Medicare Participation Indicator
Y    1999
N       1
Name: count, dtype: int64
```

```
# Plot the data
plt.figure(figsize=(10, 8))
sns.countplot(x='Place of Service', data=df, hue='Anomaly', palette='Set1')
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('Place of Service')
plt.legend()
plt.show()

# for the value counts of different categories
df[y_predict==1]['Place of Service'].value_counts()
```




```
# Plot the data
plt.figure(figsize=(10, 8))
sns.countplot(x='HCPCS Drug Indicator', data=df, hue='Anomaly', palette='Set1')
plt.title('Autoencoder for Anomaly Detection')
plt.xlabel('HCPCS Drug Indicator')
plt.legend()
plt.show()

# for the value counts of different categories
df[y_predict==1]['HCPCS Drug Indicator'].value_counts()
```