PROJECT - **Unsupervised Anamoly Detection**

DATASET - **Healthcare Providers Data For Anomaly Detection**

NAME - **Shrikar Gaikar**

Overview

Healthcare fraud is considered a challenge for many societies. Health care funding that could be spent on medicine, care for the elderly, or emergency room visits is instead lost to fraudulent activities by materialistic practitioners or patients. With rising healthcare costs, healthcare fraud is a major contributor to these increasing healthcare costs.

```
# Filtering the warnings
import warnings
warnings.filterwarnings("ignore")
```

Data Loading

```
import pandas as pd

# Loading the dataset
data = pd.read_csv("/content/Healthcare Providers.csv")
```

```
# Display the first few rows of the dataset
data.head()
```

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Middle Initial of the Provider	Credentials of the Provider	Gend of t Provid
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	
1	3354385	1346202256	JONES	WENDY	P	M.D.	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	

5 rows × 27 columns

```
# Check the shape of the dataset
data.shape
```

```
(100000, 27)
```

```
# Check for missing values
data.isnull().sum()
```

```
index                                          0
National Provider Identifier                   0
Last Name/Organization Name of the Provider    0
First Name of the Provider                  4255
Middle Initial of the Provider             29331
Credentials of the Provider                 7209
Gender of the Provider                      4254
Entity Type of the Provider                    0
```

```
Street Address 1 of the Provider                                  0
Street Address 2 of the Provider                              59363
City of the Provider                                              0
Zip Code of the Provider                                          0
State Code of the Provider                                        0
Country Code of the Provider                                      0
Provider Type                                                     0
Medicare Participation Indicator                                  0
Place of Service                                                  0
HCPCS Code                                                        0
HCPCS Description                                                 0
HCPCS Drug Indicator                                              0
Number of Services                                                0
Number of Medicare Beneficiaries                                  0
Number of Distinct Medicare Beneficiary/Per Day Services          0
Average Medicare Allowed Amount                                   0
Average Submitted Charge Amount                                   0
Average Medicare Payment Amount                                   0
Average Medicare Standardized Amount                              0
dtype: int64
```

Inference: This helps us understand the size of the dataset and identify columns with missing values.

```
# Summary statistics of the dataset
data.describe()
```

	index	National Provider Identifier	Zip Code of the Provider
count	1.000000e+05	1.000000e+05	1.000000e+05
mean	4.907646e+06	1.498227e+09	4.163820e+08
std	2.839633e+06	2.874125e+08	3.082566e+08
min	2.090000e+02	1.003001e+09	6.010000e+02
25%	2.458791e+06	1.245669e+09	1.426300e+08
50%	4.901266e+06	1.497847e+09	3.633025e+08
75%	7.349450e+06	1.740374e+09	6.819881e+08
max	9.847440e+06	1.993000e+09	9.990166e+08

```
# Information about the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 27 columns):
 #   Column                                                    Non-Null Count   Dtype
---  ------                                                    --------------   -----
 0   index                                                     100000 non-null  int64
 1   National Provider Identifier                              100000 non-null  int64
 2   Last Name/Organization Name of the Provider               100000 non-null  object
 3   First Name of the Provider                                95745 non-null   object
 4   Middle Initial of the Provider                            70669 non-null   object
 5   Credentials of the Provider                               92791 non-null   object
 6   Gender of the Provider                                    95746 non-null   object
 7   Entity Type of the Provider                               100000 non-null  object
 8   Street Address 1 of the Provider                          100000 non-null  object
 9   Street Address 2 of the Provider                          40637 non-null   object
 10  City of the Provider                                      100000 non-null  object
 11  Zip Code of the Provider                                  100000 non-null  float64
 12  State Code of the Provider                                100000 non-null  object
 13  Country Code of the Provider                              100000 non-null  object
 14  Provider Type                                             100000 non-null  object
 15  Medicare Participation Indicator                          100000 non-null  object
 16  Place of Service                                          100000 non-null  object
 17  HCPCS Code                                                100000 non-null  object
 18  HCPCS Description                                         100000 non-null  object
 19  HCPCS Drug Indicator                                      100000 non-null  object
 20  Number of Services                                        100000 non-null  object
 21  Number of Medicare Beneficiaries                          100000 non-null  object
 22  Number of Distinct Medicare Beneficiary/Per Day Services  100000 non-null  object
 23  Average Medicare Allowed Amount                           100000 non-null  object
 24  Average Submitted Charge Amount                           100000 non-null  object
 25  Average Medicare Payment Amount                           100000 non-null  object
 26  Average Medicare Standardized Amount                      100000 non-null  object
dtypes: float64(1), int64(2), object(24)
memory usage: 20.6+ MB
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Data Preprocessing

```
data.isnull().sum()
```

```
index                                                         0
National Provider Identifier                                  0
Last Name/Organization Name of the Provider                   0
First Name of the Provider                                 4255
Middle Initial of the Provider                            29331
Credentials of the Provider                                7209
Gender of the Provider                                     4254
Entity Type of the Provider                                   0
Street Address 1 of the Provider                              0
Street Address 2 of the Provider                          59363
City of the Provider                                          0
Zip Code of the Provider                                      0
State Code of the Provider                                    0
Country Code of the Provider                                  0
Provider Type                                                 0
Medicare Participation Indicator                              0
Place of Service                                              0
HCPCS Code                                                    0
HCPCS Description                                             0
HCPCS Drug Indicator                                          0
Number of Services                                            0
Number of Medicare Beneficiaries                              0
Number of Distinct Medicare Beneficiary/Per Day Services      0
Average Medicare Allowed Amount                               0
Average Submitted Charge Amount                               0
Average Medicare Payment Amount                               0
Average Medicare Standardized Amount                          0
dtype: int64
```

```
# Step 1: Remove columns related to address, zip codes, etc.
columns_to_remove = [
    'Street Address 1 of the Provider', 'Street Address 2 of the Provider',
    'City of the Provider', 'Zip Code of the Provider', 'State Code of the Provider',
    'Country Code of the Provider'
]
data.drop(columns=columns_to_remove, inplace=True)
```

```
data.head()
```

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Middle Initial of the Provider	Credentials of the Provider	Gend of t Provid
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	
1	3354385	1346202256	JONES	WENDY	P	M.D.	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	

5 rows × 21 columns

```
# Convert columns with string representations of numbers to numerical values
numeric_columns = [
    'Number of Services', 'Number of Medicare Beneficiaries', 'Number of Distinct Medicare Beneficiary/Per Day Services',
    'Average Medicare Allowed Amount', 'Average Submitted Charge Amount', 'Average Medicare Payment Amount',
    'Average Medicare Standardized Amount'
]
for col in numeric_columns:
    data[col] = data[col].str.replace(',', '').astype(float)
```

```
data.head()
```

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Middle Initial of the Provider	Credentials of the Provider	Gend of t Provid
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	
1	3354385	1346202256	JONES	WENDY	P	M.D.	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	

5 rows × 21 columns

```
# Merging the name columns into a single column
data['Full Name'] = data['First Name of the Provider'].fillna('') + ' ' + \
                    data['Middle Initial of the Provider'].fillna('') + ' ' + \
                    data['Last Name/Organization Name of the Provider'].fillna('')
data['Full Name'] = data['Full Name'].str.strip()

data = data.drop(columns=['Last Name/Organization Name of the Provider',
                          'First Name of the Provider',
                          'Middle Initial of the Provider'])

full_name = data.pop('Full Name')

data.insert(1, 'Full Name', full_name)
```

```
data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	Provider Type	P
0	8774979	SATYASREE UPADHYAYULA	1891106191	M.D.	F	I	Internal Medicine	
1	3354385	WENDY P JONES	1346202256	M.D.	F	I	Obstetrics & Gynecology	
2	3001884	RICHARD W DUROCHER	1306820956	DPM	M	I	Podiatry	
3	7594822	JASPER FULLARD	1770523540	MD	M	I	Internal Medicine	
4	746159	ANTHONY E PERROTTI	1073627758	DO	M	I	Internal Medicine	

Next steps: Generate code with data | View recommended plots

```
# Imputation of categorical values with mode
data["Credentials of the Provider"] = data["Credentials of the Provider"].fillna(data["Credentials of the Provider"].mode()[0])
data["Gender of the Provider"] = data["Gender of the Provider"].fillna(data["Gender of the Provider"].mode()[0])
```

```
data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	Provider Type	P
0	8774979	SATYASREE UPADHYAYULA	1891106191	M.D.	F	I	Internal Medicine	
1	3354385	WENDY P JONES	1346202256	M.D.	F	I	Obstetrics & Gynecology	
2	3001884	RICHARD W DUROCHER	1306820956	DPM	M	I	Podiatry	
3	7594822	JASPER FULLARD	1770523540	MD	M	I	Internal Medicine	
4	746159	ANTHONY E PERROTTI	1073627758	DO	M	I	Internal Medicine	

Next steps: Generate code with data View recommended plots

```
data.isnull().sum()
```

```
index                                                   0
Full Name                                               0
National Provider Identifier                            0
Credentials of the Provider                             0
Gender of the Provider                                  0
Entity Type of the Provider                             0
Provider Type                                           0
Medicare Participation Indicator                        0
Place of Service                                        0
HCPCS Code                                              0
HCPCS Description                                       0
HCPCS Drug Indicator                                    0
Number of Services                                      0
Number of Medicare Beneficiaries                        0
Number of Distinct Medicare Beneficiary/Per Day Services 0
Average Medicare Allowed Amount                         0
Average Submitted Charge Amount                         0
Average Medicare Payment Amount                         0
Average Medicare Standardized Amount                    0
dtype: int64
```

```
# Step 2: One-hot encode categorical values with binary options
binary_columns = ['Gender of the Provider', 'Medicare Participation Indicator', 'Entity Type of the Provider', 'HCPCS Drug Indicator']
data = pd.get_dummies(data, columns=binary_columns, drop_first=True)
```

```
data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Provider Type	Place of Service	HCPCS Code	Des
0	8774979	SATYASREE UPADHYAYULA	1891106191	M.D.	Internal Medicine	F	99223	Initi inpa typi
1	3354385	WENDY P JONES	1346202256	M.D.	Obstetrics & Gynecology	O	G0202	mam b vie
2	3001884	RICHARD W DUROCHER	1306820956	DPM	Podiatry	O	99348	E pat visi
3	7594822	JASPER FULLARD	1770523540	MD	Internal Medicine	O	81002	I m
4	746159	ANTHONY E PERROTTI	1073627758	DO	Internal Medicine	O	96372	be s mu

Next steps: Generate code with data View recommended plots

```
# Step 3: Frequency encode categorical values with more than two unique values
frequency_columns = [
    'Full Name', 'Credentials of the Provider', 'Provider Type', 'Place of Service', 'HCPCS Code', 'HCPCS Description'
]
for col in frequency_columns:
    freq_encoding = data[col].value_counts().to_dict()
    data[col] = data[col].map(freq_encoding)


data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Provider Type	Place of Service	HCPCS Code	HCPCS Description	N Ser
0	8774979	1	1891106191	32757	11366	38384	1297	1297	
1	3354385	1	1346202256	32757	1028	61616	243	243	
2	3001884	1	1306820956	1330	2027	61616	44	44	
3	7594822	1	1770523540	40083	11366	61616	460	460	
4	746159	1	1073627758	2478	11366	61616	732	732	

Next steps: Generate code with data View recommended plots

```
# Step 4: Apply Standard Scaler on the encoded dataset

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)


# Convert the scaled data back to a DataFrame
scaled_df = pd.DataFrame(scaled_data, columns=data.columns)

# Save the processed dataset to a new CSV file
scaled_df.to_csv('processed_dataset.csv', index=False)


prep_data = scaled_df


prep_data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Provider Type	Place of Service	HCPCS Code	Descri
0	1.361920	-0.092857	1.366960	0.349313	1.336743	-1.266985	0.397579	0.3
1	-0.546996	-0.092857	-0.528945	0.349313	-0.940500	0.789275	-0.439989	-0.4
2	-0.671133	-0.092857	-0.665966	-1.595350	-0.720441	0.789275	-0.598126	-0.6
3	0.946316	-0.092857	0.947412	0.802637	1.336743	0.789275	-0.267549	-0.2
4	-1.465509	-0.092857	-1.477323	-1.524313	1.336743	0.789275	-0.051402	-0.0

Next steps: Generate code with prep_data View recommended plots

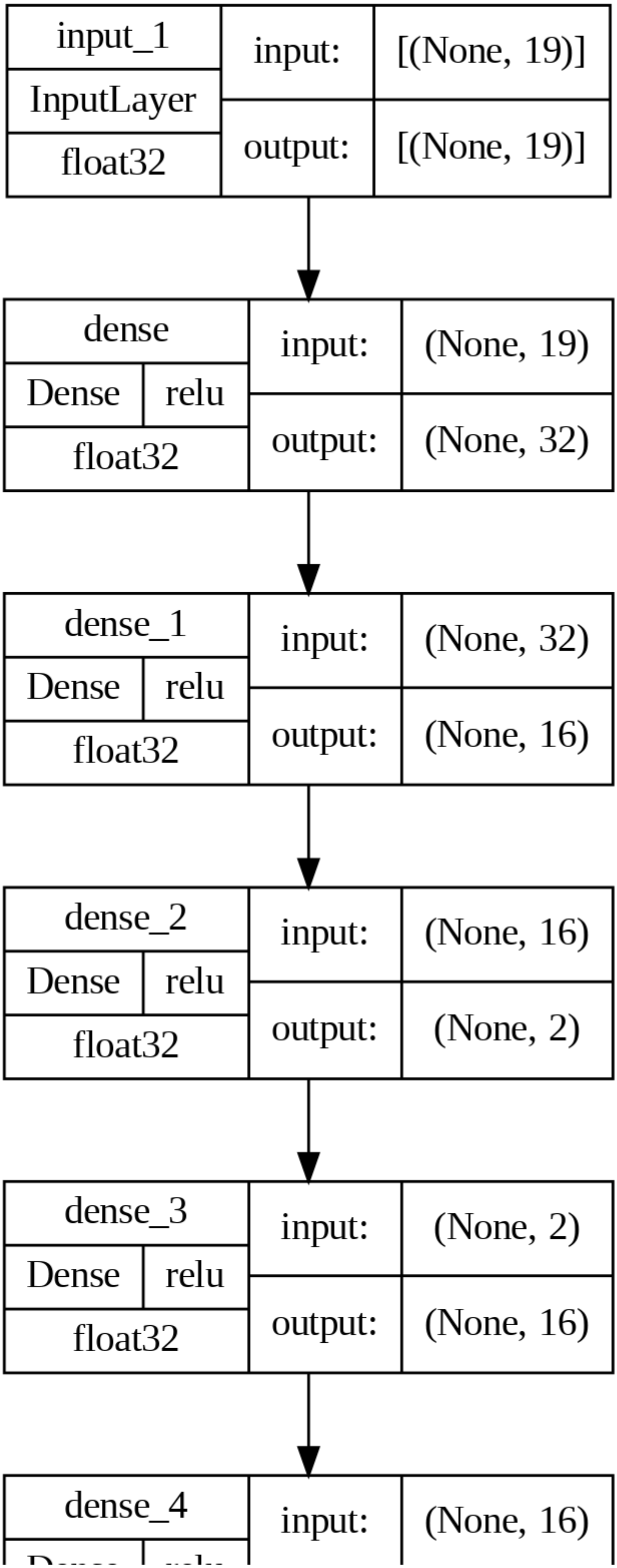
Autoencoders

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers


# Define the Autoencoder Model
input_dim = prep_data.shape[1]
encoding_dim = 32  # Number of neurons in the bottleneck layer


# Define the input layer
input_layer = Input(shape=(input_dim,))


# Define the encoder layers
encoder = Dense(encoding_dim, activation="relu", activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(16, activation="relu")(encoder)
encoder = Dense(2, activation="relu")(encoder)
encoder = Dense(16, activation="relu")(encoder)


# Define the decoder layers
decoder = Dense(32, activation="relu")(encoder)
decoder = Dense(input_dim, activation="relu")(decoder)


# Combine the encoder and decoder into an autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)


# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mean_squared_error')


autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 19)]              0

 dense (Dense)               (None, 32)                640

 dense_1 (Dense)             (None, 16)                528

 dense_2 (Dense)             (None, 2)                 34

 dense_3 (Dense)             (None, 16)                48

 dense_4 (Dense)             (None, 32)                544

 dense_5 (Dense)             (None, 19)                627

=================================================================
Total params: 2421 (9.46 KB)
Trainable params: 2421 (9.46 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
tf.keras.utils.plot_model(
```

```
    autoencoder,
    to_file='autoencoder_model.png',
    show_shapes=True,
    show_dtype=True,
    show_layer_names=True,
    rankdir='TB',
    expand_nested=False,
    dpi=200,
    show_layer_activations=True,
    show_trainable=False,
)
```

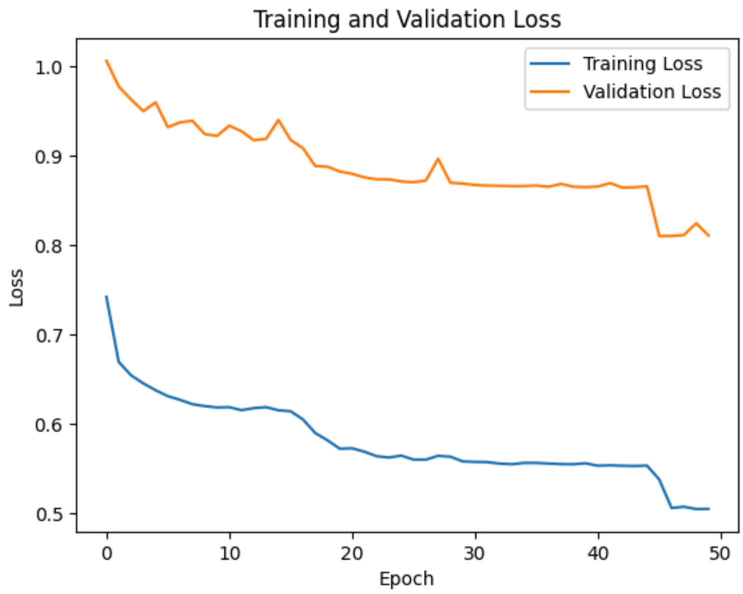

input_1
InputLayer
float32
input:
[(None, 19)]
output:
[(None, 19)]
dense
Dense
relu
float32
input:
(None, 19)
output:
(None, 32)
dense_1
Dense
relu
float32
input:
(None, 32)
output:
(None, 16)
dense_2
Dense
relu
float32
input:
(None, 16)
output:
(None, 2)
dense_3
Dense
relu
float32
input:
(None, 2)
output:
(None, 16)
dense_4
input:
(None, 16)

Dense	relu	

```
# Train the Autoencoder
history = autoencoder.fit(prep_data, prep_data, epochs=50, batch_size=32, shuffle=True, validation_split=0.2)
```

```
Epoch 1/50
2500/2500 [==============================] - 13s 4ms/step - loss: 0.7423 - val_loss: 1.0067
Epoch 2/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.6696 - val_loss: 0.9780
Epoch 3/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6546 - val_loss: 0.9636
Epoch 4/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.6455 - val_loss: 0.9503
Epoch 5/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6378 - val_loss: 0.9602
Epoch 6/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6312 - val_loss: 0.9324
Epoch 7/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6270 - val_loss: 0.9378
Epoch 8/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.6222 - val_loss: 0.9395
Epoch 9/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.6200 - val_loss: 0.9247
Epoch 10/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6185 - val_loss: 0.9227
Epoch 11/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6188 - val_loss: 0.9341
Epoch 12/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6155 - val_loss: 0.9278
Epoch 13/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.6178 - val_loss: 0.9177
Epoch 14/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6189 - val_loss: 0.9194
Epoch 15/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.6153 - val_loss: 0.9405
Epoch 16/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.6143 - val_loss: 0.9180
Epoch 17/50
2500/2500 [==============================] - 9s 3ms/step - loss: 0.6050 - val_loss: 0.9089
Epoch 18/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5899 - val_loss: 0.8889
Epoch 19/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5817 - val_loss: 0.8881
Epoch 20/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.5722 - val_loss: 0.8827
Epoch 21/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5728 - val_loss: 0.8802
Epoch 22/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5690 - val_loss: 0.8762
Epoch 23/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5639 - val_loss: 0.8740
Epoch 24/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.5625 - val_loss: 0.8740
Epoch 25/50
2500/2500 [==============================] - 9s 4ms/step - loss: 0.5646 - val_loss: 0.8716
Epoch 26/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5601 - val_loss: 0.8708
Epoch 27/50
2500/2500 [==============================] - 10s 4ms/step - loss: 0.5601 - val_loss: 0.8726
Epoch 28/50
2500/2500 [==============================] - 11s 4ms/step - loss: 0.5643 - val_loss: 0.8971
Epoch 29/50
2500/2500 [==============================] - 9s 3ms/step - loss: 0.5634 - val_loss: 0.8702
```

```
# Plot Training Loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
labels = autoencoder.predict(prep_data)
mse = np.mean(np.power(prep_data - labels, 2), axis=1)
```

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

```
# Anomaly Detection
threshold = np.percentile(mse, 99)
anomalies = mse > threshold

print(f"Threshold: {threshold}")
print(f"Number of anomalies detected: {np.sum(anomalies)}")
```

```
Threshold: 1.363107010093677
Number of anomalies detected: 1000
```

```
data['Anomaly'] = anomalies
```

```
data['Anomaly'] = data['Anomaly'].apply(lambda x: 1 if x == True else 0)
```

```
list(data['Anomaly']).count(1)
```

```
1000
```

```
list(data['Anomaly']).count(0)
```

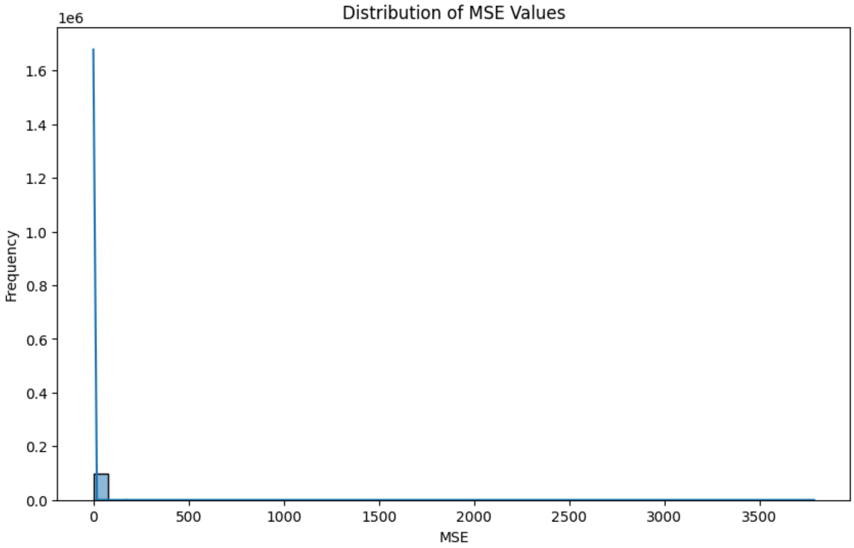
```
99000
```

```
data.head()
```

	index	Full Name	National Provider Identifier	Credentials of the Provider	Provider Type	Place of Service	HCPCS Code	HCPCS Description	N Ser
0	8774979	1	1891106191	32757	11366	38384	1297	1297	
1	3354385	1	1346202256	32757	1028	61616	243	243	
2	3001884	1	1306820956	1330	2027	61616	44	44	
3	7594822	1	1770523540	40083	11366	61616	460	460	
4	746159	1	1073627758	2478	11366	61616	732	732	

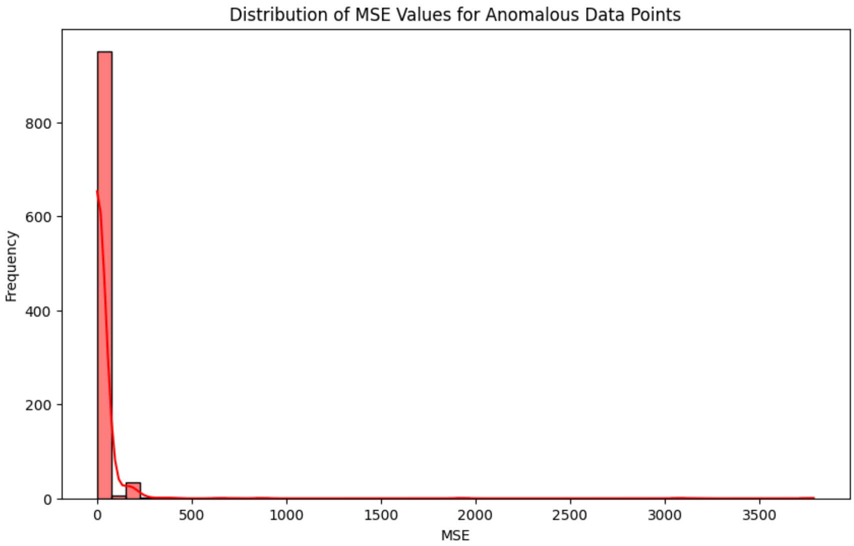
Next steps: Generate code with data | View recommended plots

```
# Add the MSE values to the dataframe for visualization
data['MSE'] = mse


# Plot distribution of MSE values
plt.figure(figsize=(10, 6))
sns.histplot(data['MSE'], bins=50, kde=True)
plt.title('Distribution of MSE Values')
plt.xlabel('MSE')
plt.ylabel('Frequency')
plt.show()
```




```
# Plot distribution of MSE values for normal data points
plt.figure(figsize=(10, 6))
sns.histplot(data[data['Anomaly'] == 0]['MSE'], bins=50, kde=True)
plt.title('Distribution of MSE Values for Normal Data Points')
plt.xlabel('MSE')
plt.ylabel('Frequency')
plt.show()
```

```
# Plot distribution of MSE values for anomalous data points
plt.figure(figsize=(10, 6))
sns.histplot(data[data['Anomaly'] == 1]['MSE'], bins=50, kde=True, color='red')
plt.title('Distribution of MSE Values for Anomalous Data Points')
plt.xlabel('MSE')
plt.ylabel('Frequency')
plt.show()
```




```
# Boxplot comparison of normal and anomalous data MSE values
plt.figure(figsize=(10, 6))
sns.boxplot(x='Anomaly', y='MSE', data=data)
plt.title('Boxplot of MSE Values for Normal and Anomalous Data Points')
plt.xlabel('Anomaly')
plt.ylabel('MSE')
plt.show()
```