INFOSYS SPRINGBOARD INTERNSHIP

MileStone 4

Name: Rudrani Ghosh

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

data = pd.read_csv("/Users/rudranighosh/Downloads/Healthcare Providers.csv")
data.head()
```

Out[1]:

	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	Gender of the Provider	Entity Type of the Provider	Street Address 1 of the Provider	Street Address 2 of the Provider	...	HCPCS Code	HCPCS Description	I In
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	F	I	1402 S GRAND BLVD	FDT 14TH FLOOR	...	99223	Initial hospital inpatient care, typically 70 ...	
1	3354385	1346202256	JONES	WENDY	P	M.D.	F	I	2950 VILLAGE DR	NaN	...	G0202	Screening mammography, bilateral (2-view study...	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	M	I	20 WASHINGTON AVE	STE 212	...	99348	Established patient home visit, typically 25 m...	
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	M	I	5746 N BROADWAY ST	NaN	...	81002	Urinalysis, manual test	
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	M	I	875 MILITARY TRL	SUITE 200	...	96372	Injection beneath the skin or into muscle for ...	

5 rows × 27 columns

In [2]:
```
# 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
```

In [3]:
```
irrelevant_columns=['Entity Type of the Provider',
                    'Street Address 1 of the Provider',
                    'Street Address 2 of the Provider',
                    'Zip Code of the Provider',
                    'Medicare Participation Indicator',
                    'Place of Service',
                    'HCPCS Code',
                    'HCPCS Description',
                    'HCPCS Drug Indicator',
                    'Country Code of the Provider']

data=data.drop(columns=irrelevant_columns)
```

Columns that have no relevance in our assignment have been dropped

In [4]:
```
data.head()
```

Out[4]:

	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	Gender of the Provider	City of the Provider	State Code of the Provider	Provider Type	Number of Services	Number of Medicare Beneficiaries	Ber D
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	F	SAINT LOUIS	MO	Internal Medicine	27	24	
1	3354385	1346202256	JONES	WENDY	P	M.D.	F	FAYETTEVILLE	NC	Obstetrics & Gynecology	175	175	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	M	NORTH HAVEN	CT	Podiatry	32	13	
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	M	KANSAS CITY	MO	Internal Medicine	20	18	
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	M	JUPITER	FL	Internal Medicine	33	24	

Data Preprocessing

In [5]:

```python
# 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_column = data.pop('Full Name')

data.insert(1, 'Full Name', full_name_column)


data.head()
```

Out[5]:

	index	Full Name	National Provider Identifier	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Provider Type	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount
0	8774979	SATYASREE UPADHYAYULA	1891106191	M.D.	F	SAINT LOUIS	MO	Internal Medicine	27	24	27	200.58777778
1	3354385	WENDY P JONES	1346202256	M.D.	F	FAYETTEVILLE	NC	Obstetrics & Gynecology	175	175	175	123.73
2	3001884	RICHARD W DUROCHER	1306820956	DPM	M	NORTH HAVEN	CT	Podiatry	32	13	32	90.65
3	7594822	JASPER FULLARD	1770523540	MD	M	KANSAS CITY	MO	Internal Medicine	20	18	20	3.5
4	746159	ANTHONY E PERROTTI	1073627758	DO	M	JUPITER	FL	Internal Medicine	33	24	31	26.52

A new column "Full Name" has been created to merge first name, middle name and last name

In [6]:

```python
# Uniform format of credentials
data['Credentials of the Provider'] = data['Credentials of the Provider'].str.replace(r'\.', '', regex=True).str.upper()
data.head()
```

Out[6]:

	index	Full Name	National Provider Identifier	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Provider Type	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount
0	8774979	SATYASREE UPADHYAYULA	1891106191	MD	F	SAINT LOUIS	MO	Internal Medicine	27	24	27	200.58777778
1	3354385	WENDY P JONES	1346202256	MD	F	FAYETTEVILLE	NC	Obstetrics & Gynecology	175	175	175	123.73
2	3001884	RICHARD W DUROCHER	1306820956	DPM	M	NORTH HAVEN	CT	Podiatry	32	13	32	90.65
3	7594822	JASPER FULLARD	1770523540	MD	M	KANSAS CITY	MO	Internal Medicine	20	18	20	3.5
4	746159	ANTHONY E PERROTTI	1073627758	DO	M	JUPITER	FL	Internal Medicine	33	24	31	26.52

"Credentials of the Provider" column now follows a uniform format. Such that MD and M.D and M.D. are all treated as the same unit

Converting Object to Numeric Type

In [7]:

```python
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 column in numeric_columns:
    data[column] = pd.to_numeric(data[column], errors='coerce')


data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 15 columns):
 #   Column                                                 Non-Null Count   Dtype  
---  ------                                                 --------------   -----  
 0   index                                                  100000 non-null  int64  
 1   Full Name                                              100000 non-null  object 
 2   National Provider Identifier                           100000 non-null  int64  
 3   Credentials of the Provider                            92791 non-null   object 
 4   Gender of the Provider                                 95746 non-null   object 
 5   City of the Provider                                   100000 non-null  object 
 6   State Code of the Provider                             100000 non-null  object 
 7   Provider Type                                          100000 non-null  object 
 8   Number of Services                                     97347 non-null   float64
 9   Number of Medicare Beneficiaries                       99595 non-null   float64
 10  Number of Distinct Medicare Beneficiary/Per Day Services  98500 non-null   float64
 11  Average Medicare Allowed Amount                        99255 non-null   float64
 12  Average Submitted Charge Amount                        93277 non-null   float64
 13  Average Medicare Payment Amount                        99534 non-null   float64
 14  Average Medicare Standardized Amount                   99530 non-null   float64
dtypes: float64(7), int64(2), object(6)
memory usage: 11.4+ MB
```

Looking for Missing Values and imputing them with Mean

In [8]:
```
# missing values
print(data.isnull().sum())
```

```
index                                                     0
Full Name                                                 0
National Provider Identifier                              0
Credentials of the Provider                            7209
Gender of the Provider                                 4254
City of the Provider                                      0
State Code of the Provider                                0
Provider Type                                             0
Number of Services                                     2653
Number of Medicare Beneficiaries                        405
Number of Distinct Medicare Beneficiary/Per Day Services    1500
Average Medicare Allowed Amount                         745
Average Submitted Charge Amount                        6723
Average Medicare Payment Amount                         466
Average Medicare Standardized Amount                    470
dtype: int64
```

In [9]:
```
# Imputation of numeric missing values with mean
data[numeric_columns] = data[numeric_columns].fillna(data[numeric_columns].mean())

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

```
index                                                     0
Full Name                                                 0
National Provider Identifier                              0
Credentials of the Provider                            7209
Gender of the Provider                                 4254
City of the Provider                                      0
State Code of the Provider                                0
Provider Type                                             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
```

Imputation of categorical columns with mode

In [10]:
```
categorical_columns = ['Credentials of the Provider',
                   'Gender of the Provider',
                   'City of the Provider',
                  'State Code of the Provider']

for column in categorical_columns:
    data[column].fillna(data[column].mode()[0], inplace=True)

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

```
index                                                     0
Full Name                                                 0
National Provider Identifier                              0
Credentials of the Provider                               0
Gender of the Provider                                    0
City of the Provider                                      0
State Code of the Provider                                0
Provider Type                                             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
```

Looking for Duplicate Values

In [11]:
```
# Check for duplicates
print(data.duplicated().sum())
```

```
0
```

In [12]:
```
data.head()
```

Out[12]:

	index	Full Name	National Provider Identifier	Credentials of the Provider	Gender of the Provider	City of the Provider	State Code of the Provider	Provider Type	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount	
0	8774979	SATYASREE UPADHYAYULA	1891106191	MD	F	SAINT LOUIS	MO	Internal Medicine	27.0	24.0	27.0	200.587778	
1	3354385	WENDY P JONES	1346202256	MD	F	FAYETTEVILLE	NC	Obstetrics & Gynecology	175.0	175.0	175.0	123.730000	5
2	3001884	RICHARD W DUROCHER	1306820956	DPM	M	NORTH HAVEN	CT	Podiatry	32.0	13.0	32.0	90.650000	1
3	7594822	JASPER FULLARD	1770523540	MD	M	KANSAS CITY	MO	Internal Medicine	20.0	18.0	20.0	3.500000	
4	746159	ANTHONY E PERROTTI	1073627758	DO	M	JUPITER	FL	Internal Medicine	33.0	24.0	31.0	26.520000	

Encoding some Categorical Columns using Frequency Encoder

In [13]:
```python
def frequency_encode(df, columns):
    for column in columns:
        freq_encoding = df[column].value_counts() / len(df)
        new_column_name = column + '_Freq'
        df.insert(df.columns.get_loc(column) + 1, new_column_name, df[column].map(freq_encoding))
    return df

columns_to_encode=['Credentials of the Provider',
                   'Gender of the Provider',
                   'Provider Type',
                  'State Code of the Provider']

data = frequency_encode(data, columns_to_encode)

df=data

data.head()
```

Out[13]:

	index	Full Name	National Provider Identifier	Credentials of the Provider	Credentials of the Provider_Freq	Gender of the Provider	Gender of the Provider_Freq	City of the Provider	State Code of the Provider	State Code of the Provider_Freq	Provider Type	Provide Type_Fre
0	8774979	SATYASREE UPADHYAYULA	1891106191	MD	0.73827	F	0.29105	SAINT LOUIS	MO	0.01997	Internal Medicine	0.1136
1	3354385	WENDY P JONES	1346202256	MD	0.73827	F	0.29105	FAYETTEVILLE	NC	0.03725	Obstetrics & Gynecology	0.0102
2	3001884	RICHARD W DUROCHER	1306820956	DPM	0.01915	M	0.70895	NORTH HAVEN	CT	0.01403	Podiatry	0.0202
3	7594822	JASPER FULLARD	1770523540	MD	0.73827	M	0.70895	KANSAS CITY	MO	0.01997	Internal Medicine	0.1136
4	746159	ANTHONY E PERROTTI	1073627758	DO	0.06176	M	0.70895	JUPITER	FL	0.07263	Internal Medicine	0.1136

In [14]:
```python
df.columns
```

Out[14]:
```
Index(['index', 'Full Name', 'National Provider Identifier',
       'Credentials of the Provider', 'Credentials of the Provider_Freq',
       'Gender of the Provider', 'Gender of the Provider_Freq',
       'City of the Provider', 'State Code of the Provider',
       'State Code of the Provider_Freq', 'Provider Type',
       'Provider Type_Freq', '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'],
      dtype='object')
```

Performing Standardization on Numerical Columns

In [15]:
```python
from sklearn.preprocessing import StandardScaler

data_copy=data.copy()

standardization_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',
                         'Credentials of the Provider_Freq',
                         'Gender of the Provider_Freq',
                         'State Code of the Provider_Freq' ]

# Standardization
standard_scaler = StandardScaler()
data[standardization_columns] = standard_scaler.fit_transform(data[standardization_columns])



print("Standardized DataFrame:")
data.head()
```

```
Standardized DataFrame:
```

Out[15]:

	index	Full Name	National Provider Identifier	Credentials of the Provider	Credentials of the Provider_Freq	Gender of the Provider	Gender of the Provider_Freq	City of the Provider	State Code of the Provider	State Code of the Provider_Freq	Provider Type	Provide Type_Fre
0	8774979	SATYASREE UPADHYAYULA	1891106191	MD	0.594983	F	-1.560716	SAINT LOUIS	MO	-0.737342	Internal Medicine	0.1136
1	3354385	WENDY P JONES	1346202256	MD	0.594983	F	-1.560716	FAYETTEVILLE	NC	-0.004973	Obstetrics & Gynecology	0.0102
2	3001884	RICHARD W DUROCHER	1306820956	DPM	-1.684316	M	0.640731	NORTH HAVEN	CT	-0.989093	Podiatry	0.0202
3	7594822	JASPER FULLARD	1770523540	MD	0.594983	M	0.640731	KANSAS CITY	MO	-0.737342	Internal Medicine	0.1136
4	746159	ANTHONY E PERROTTI	1073627758	DO	-1.549260	M	0.640731	JUPITER	FL	1.494517	Internal Medicine	0.1136

FINAL DATASET

In [16]:

```python
anomaly_detection_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',
    'Credentials of the Provider_Freq',
    'Gender of the Provider_Freq',
    'State Code of the Provider_Freq',
    'Provider Type_Freq'
]

X = data[anomaly_detection_columns]

X
```

Out[16]:

	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	Credentials of the Provider_Freq	Gender of the Provider_Freq	State Code of the Provider_Freq	Provider Type_Freq
0	-0.497577	-0.444753	-0.482232	1.098226	0.621012	0.972452	1.003321	0.594983	-1.560716	-0.737342	0.11366
1	0.503328	1.040098	0.554599	0.352134	1.940981	0.549955	0.722789	0.594983	-1.560716	-0.004973	0.01028
2	-0.463762	-0.552921	-0.447204	0.031012	-0.192958	-0.047975	-0.096209	-1.684316	0.640731	-0.989093	0.02027
3	-0.544917	-0.503753	-0.531272	-0.814992	-1.005784	-0.718674	-0.722804	0.594983	0.640731	-0.737342	0.11366
4	-0.456999	-0.444753	-0.454210	-0.591527	-0.816125	-0.541578	-0.551510	-1.549260	0.640731	1.494517	0.11366
...	...	...	...	...	...	...	...	...	...	...	...
99995	-0.544917	-0.484087	-0.531272	-0.020219	0.126753	-0.088807	-0.078095	-1.709831	-1.560716	0.142517	0.02780
99996	0.239576	0.371423	0.281380	-0.254193	-0.252286	-0.426514	-0.354403	-1.729577	-1.560716	-1.140399	0.05713
99997	-0.605783	-0.572588	-0.594322	-0.674428	-0.439269	-0.601485	-0.600151	0.594983	0.640731	-0.737342	0.04602
99998	-0.599020	-0.562754	-0.587316	-0.552503	-0.680654	-0.427351	-0.482868	0.594983	-1.560716	1.112228	0.11366
99999	3.303156	0.066586	3.440912	-0.474250	-0.778910	-0.429474	-0.476378	0.594983	-1.560716	1.112228	0.02780

100000 rows × 11 columns

Using Auto Encoders

Autoencoders can be a powerful tool for anomaly detection

In [17]:

```python
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from keras.models import Model
from keras.layers import Input, Dense

X_scaled = X  # since X is already standardized

# Define the Autoencoder Model
input_dim = X_scaled.shape[1]
encoding_dim = 11  # Number of nodes in the encoded layer
input_layer = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_layer)
decoded = Dense(input_dim, activation='sigmoid')(encoded)
autoencoder = Model(inputs=input_layer, outputs=decoded)

autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

In [18]:

```python
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from keras.models import Model
from keras.layers import Input, Dense

X_scaled = X

# Defining the Autoencoder Model with 6 Dense Layers
input_dim = X_scaled.shape[1]

input_layer = Input(shape=(input_dim,))

# Encoding layers
encoded = Dense(64, activation='relu')(input_layer)
```

```
encoded = Dense(32, activation='relu')(encoded)
encoded = Dense(16, activation='relu')(encoded)
encoded = Dense(encoding_dim, activation='relu')(encoded)

# Decoding layers
decoded = Dense(16, activation='relu')(encoded)
decoded = Dense(32, activation='relu')(decoded)
decoded = Dense(64, activation='relu')(decoded)
decoded = Dense(input_dim, activation='sigmoid')(decoded)

# Creating the Autoencoder Model
autoencoder = Model(inputs=input_layer, outputs=decoded)

# Compiling the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

In [19]:
```
autoencoder.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 11)	0
dense_2 (Dense)	(None, 64)	768
dense_3 (Dense)	(None, 32)	2,080
dense_4 (Dense)	(None, 16)	528
dense_5 (Dense)	(None, 11)	187
dense_6 (Dense)	(None, 16)	192
dense_7 (Dense)	(None, 32)	544
dense_8 (Dense)	(None, 64)	2,112
dense_9 (Dense)	(None, 11)	715

Total params: 7,126 (27.84 KB)
Trainable params: 7,126 (27.84 KB)
Non-trainable params: 0 (0.00 B)

Interpretation of Model Architecture

Model: "functional_7"

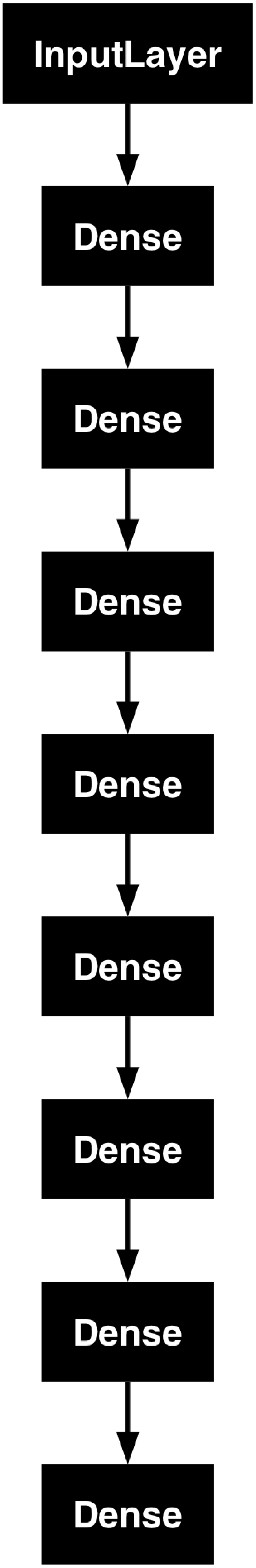
The autoencoder model consists of the following layers:

- **Input Layer:** The input layer takes in data with 11 features.
- **Dense Layers:**
  - The first dense layer has 64 units and 768 parameters.
  - The second dense layer has 32 units and 2,080 parameters.
  - The third dense layer has 16 units and 528 parameters.
  - The fourth dense layer has 11 units and 187 parameters.
  - The fifth dense layer has 16 units and 192 parameters.
  - The sixth dense layer has 32 units and 544 parameters.
  - The seventh dense layer has 64 units and 2,112 parameters.
  - The final dense layer has 11 units and 715 parameters.
- **Total Parameters:** The model has a total of 7,126 parameters, all of which are trainable.

The autoencoder consists of an encoding part (first four dense layers) and a decoding part (last four dense layers). The encoding layers reduce the input data to a lower-dimensional representation, while the decoding layers reconstruct the data back to its original dimensions. This structure helps the model learn an efficient representation of the input data, which can be useful for anomaly detection.

In [22]:
```
import tensorflow as tf
plot_path = 'autoencoder_model.png'
tf.keras.utils.plot_model(
    autoencoder,
    to_file=plot_path,
    show_shapes=False,
    show_dtype=False,
    show_layer_names=False,
    rankdir='TB',
    expand_nested=False,
    dpi=200,
    show_layer_activations=False,
    show_trainable=False
)

from IPython.display import Image
Image(filename=plot_path)
```

Out[22]:


InputLayer
Dense
Dense
Dense
Dense
Dense
Dense
Dense
Dense

In [23]:
```
import numpy as np
import pandas as pd
# Train the Autoencoder
history = autoencoder.fit(X_scaled, X_scaled, epochs=50, batch_size=32, shuffle=True, validation_split=0.1, verbose=1)
```

```
Epoch 1/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 2s 426us/step - loss: 0.6662 - val_loss: 0.5536
Epoch 2/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 402us/step - loss: 0.5584 - val_loss: 0.5480
Epoch 3/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 403us/step - loss: 0.5567 - val_loss: 0.5456
Epoch 4/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 405us/step - loss: 0.5496 - val_loss: 0.5485
Epoch 5/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 406us/step - loss: 0.5551 - val_loss: 0.5443
Epoch 6/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 408us/step - loss: 0.5496 - val_loss: 0.5439
Epoch 7/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 399us/step - loss: 0.5501 - val_loss: 0.5437
Epoch 8/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 405us/step - loss: 0.5480 - val_loss: 0.5433
Epoch 9/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 396us/step - loss: 0.5554 - val_loss: 0.5432
Epoch 10/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 392us/step - loss: 0.5544 - val_loss: 0.5434
Epoch 11/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5518 - val_loss: 0.5437
Epoch 12/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 392us/step - loss: 0.5510 - val_loss: 0.5430
Epoch 13/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5558 - val_loss: 0.5428
Epoch 14/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5535 - val_loss: 0.5437
Epoch 15/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 392us/step - loss: 0.5483 - val_loss: 0.5425
Epoch 16/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5519 - val_loss: 0.5424
Epoch 17/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5562 - val_loss: 0.5452
Epoch 18/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5467 - val_loss: 0.5424
Epoch 19/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5476 - val_loss: 0.5427
Epoch 20/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5494 - val_loss: 0.5423
Epoch 21/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5526 - val_loss: 0.5423
Epoch 22/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5485 - val_loss: 0.5422
Epoch 23/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 388us/step - loss: 0.5473 - val_loss: 0.5421
Epoch 24/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5564 - val_loss: 0.5422
Epoch 25/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5541 - val_loss: 0.5421
Epoch 26/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5507 - val_loss: 0.5423
Epoch 27/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 389us/step - loss: 0.5565 - val_loss: 0.5444
Epoch 28/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5499 - val_loss: 0.5419
Epoch 29/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 392us/step - loss: 0.5498 - val_loss: 0.5425
Epoch 30/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5516 - val_loss: 0.5420
Epoch 31/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5553 - val_loss: 0.5422
Epoch 32/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5497 - val_loss: 0.5420
Epoch 33/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 389us/step - loss: 0.5522 - val_loss: 0.5419
Epoch 34/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5426 - val_loss: 0.5418
Epoch 35/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5474 - val_loss: 0.5423
Epoch 36/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 390us/step - loss: 0.5460 - val_loss: 0.5418
Epoch 37/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5538 - val_loss: 0.5418
Epoch 38/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5542 - val_loss: 0.5417
Epoch 39/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5515 - val_loss: 0.5418
Epoch 40/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5500 - val_loss: 0.5417
Epoch 41/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 391us/step - loss: 0.5494 - val_loss: 0.5418
Epoch 42/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 395us/step - loss: 0.5488 - val_loss: 0.5420
Epoch 43/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5518 - val_loss: 0.5442
Epoch 44/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 393us/step - loss: 0.5571 - val_loss: 0.5419
Epoch 45/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 395us/step - loss: 0.5506 - val_loss: 0.5429
Epoch 46/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 394us/step - loss: 0.5516 - val_loss: 0.5423
Epoch 47/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 395us/step - loss: 0.5531 - val_loss: 0.5418
Epoch 48/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 403us/step - loss: 0.5495 - val_loss: 0.5418
Epoch 49/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 407us/step - loss: 0.5451 - val_loss: 0.5420
Epoch 50/50
2813/2813 ━━━━━━━━━━━━━━━━━━━━ 1s 416us/step - loss: 0.5504 - val_loss: 0.5418
```

In [24]:
```
# Reconstruct the Data and Calculate Reconstruction Error
X_reconstructed = autoencoder.predict(X_scaled)
reconstruction_errors = np.mean(np.square(X_scaled - X_reconstructed), axis=1)

# Detect Anomalies Based on Reconstruction Error
threshold = np.percentile(reconstruction_errors, 99)  # Set threshold at 99 percentile
```

```
data['Autoencoder_Anomaly'] = (reconstruction_errors > threshold).astype(int)

# Identify anomalies
anomalies = reconstruction_errors > threshold
num_anomalies = np.sum(anomalies)

print(f'Number of anomalies: {num_anomalies}')
```

```
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 1s 214us/step
Number of anomalies: 1000
```

Model Training and Anomaly Detection Results

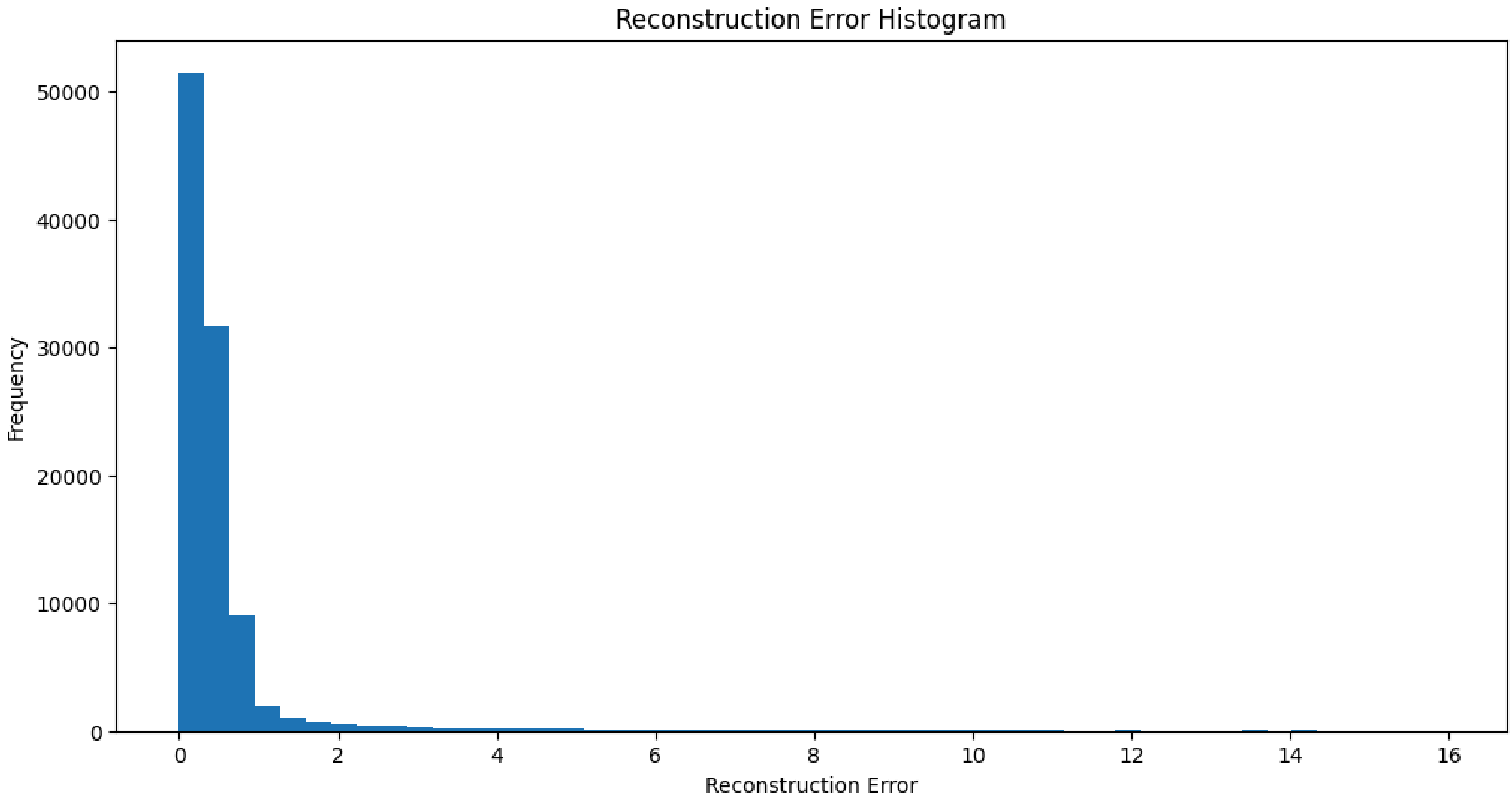
Training Progress:

- **Epochs Completed:** 3125

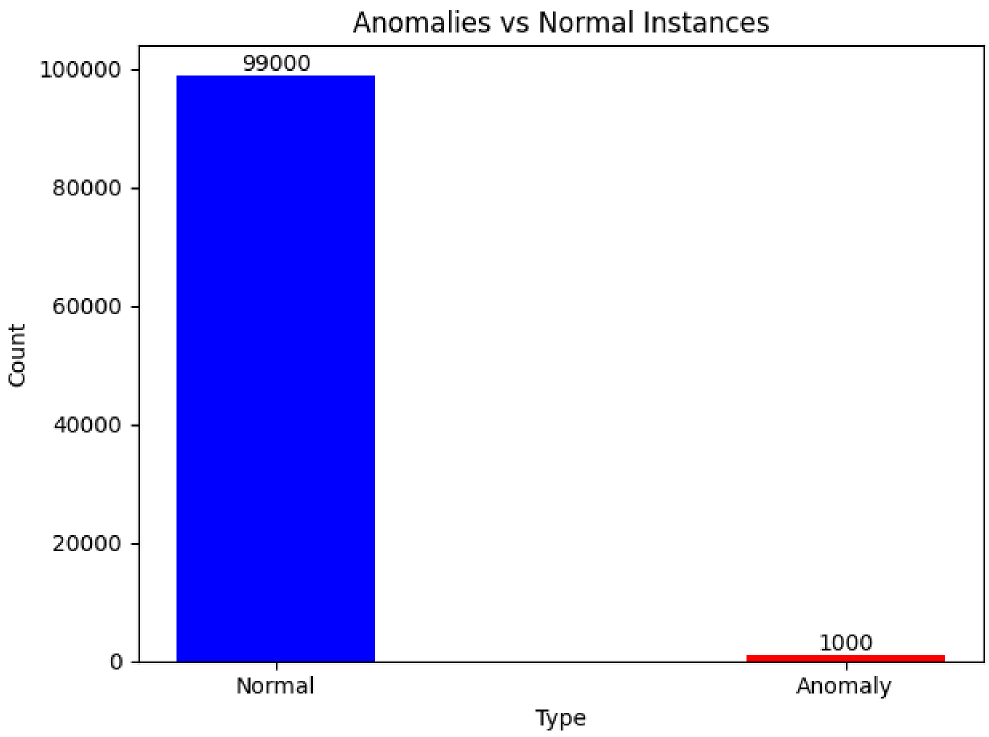
Anomaly Detection:

- **Number of Anomalies Detected:** 1000

In [25]:
```
# Plot the Reconstruction Error
plt.figure(figsize=(12, 6))
plt.hist(reconstruction_errors, bins=50)
plt.xlabel('Reconstruction Error')
plt.ylabel('Frequency')
plt.title('Reconstruction Error Histogram')
plt.show()
```

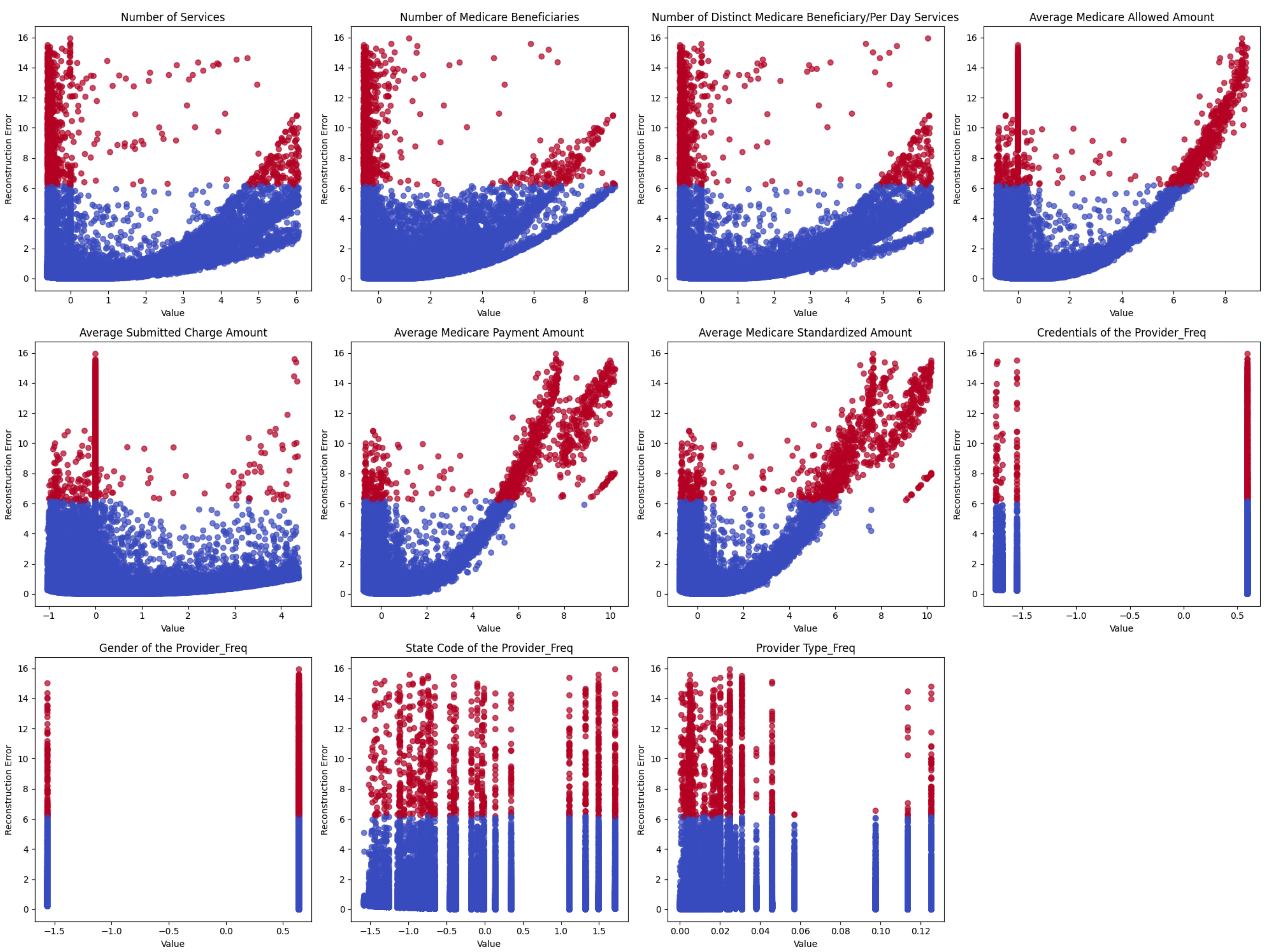


In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

threshold = np.percentile(reconstruction_errors, 99)
anomalies = reconstruction_errors > threshold

# Bar Graph for anomalies vs. normal instances
normal_count = sum(~anomalies)
anomaly_count = sum(anomalies)

bar_width = 0.35
index = np.arange(2)

bars = plt.bar(index, [normal_count, anomaly_count], bar_width, color=['blue', 'red'])
plt.title('Anomalies vs Normal Instances')
plt.xlabel('Type')
plt.ylabel('Count')
plt.xticks(index, ['Normal', 'Anomaly'])

# counts on top of the bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2.0, height, '%d' % int(height), ha='center', va='bottom')

plt.tight_layout()
plt.show()
```



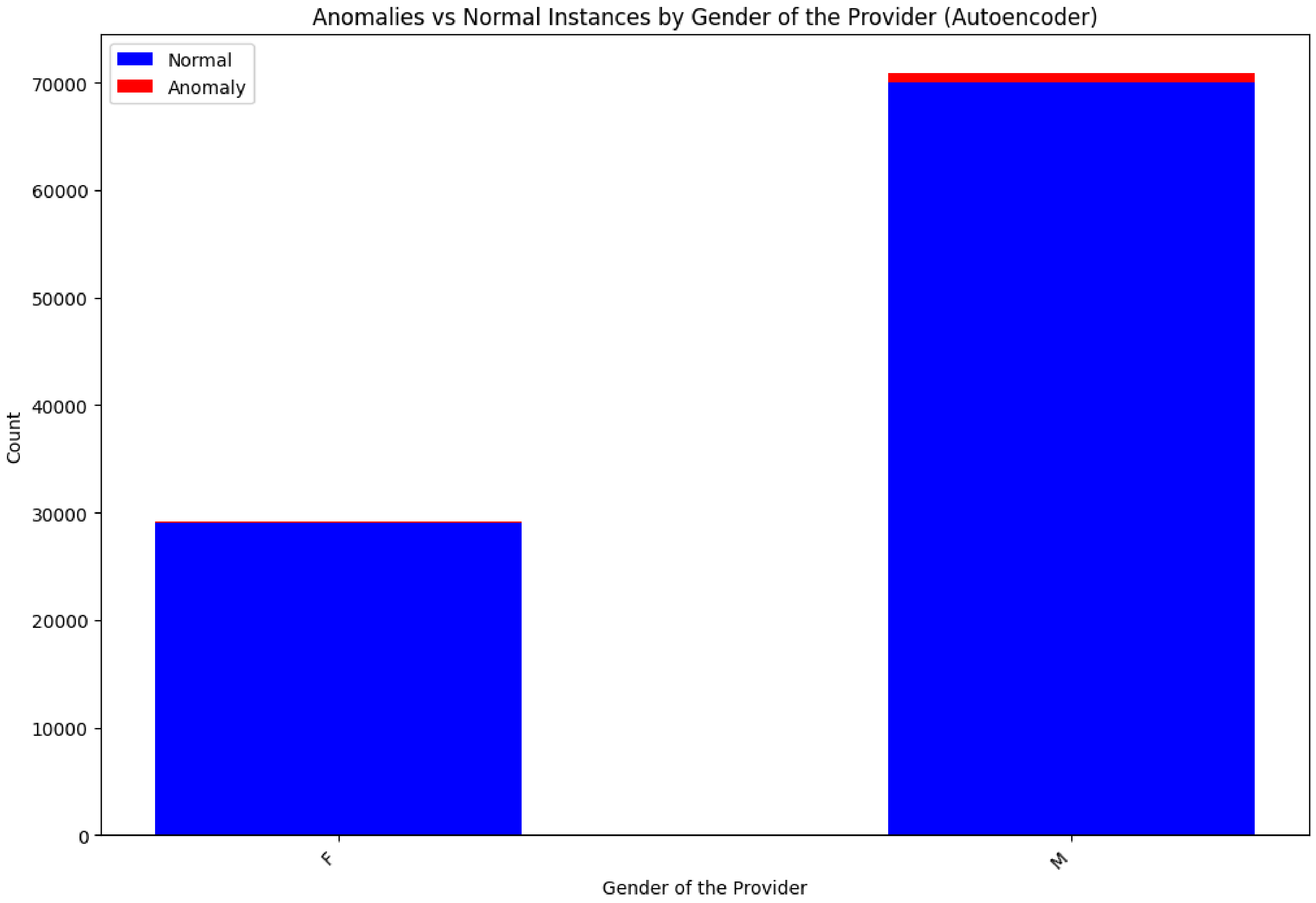
Number of Anomalies Detected: 1000

In [27]:
```python
# Scatter Plots of anomalies in each column
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(20, 15))
axes = axes.flatten()

for i, col in enumerate(X.columns):
    axes[i].scatter(X[col], reconstruction_errors, c=anomalies, cmap='coolwarm', alpha=0.7)
    axes[i].set_title(col)
    axes[i].set_xlabel('Value')
    axes[i].set_ylabel('Reconstruction Error')

# Hide any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



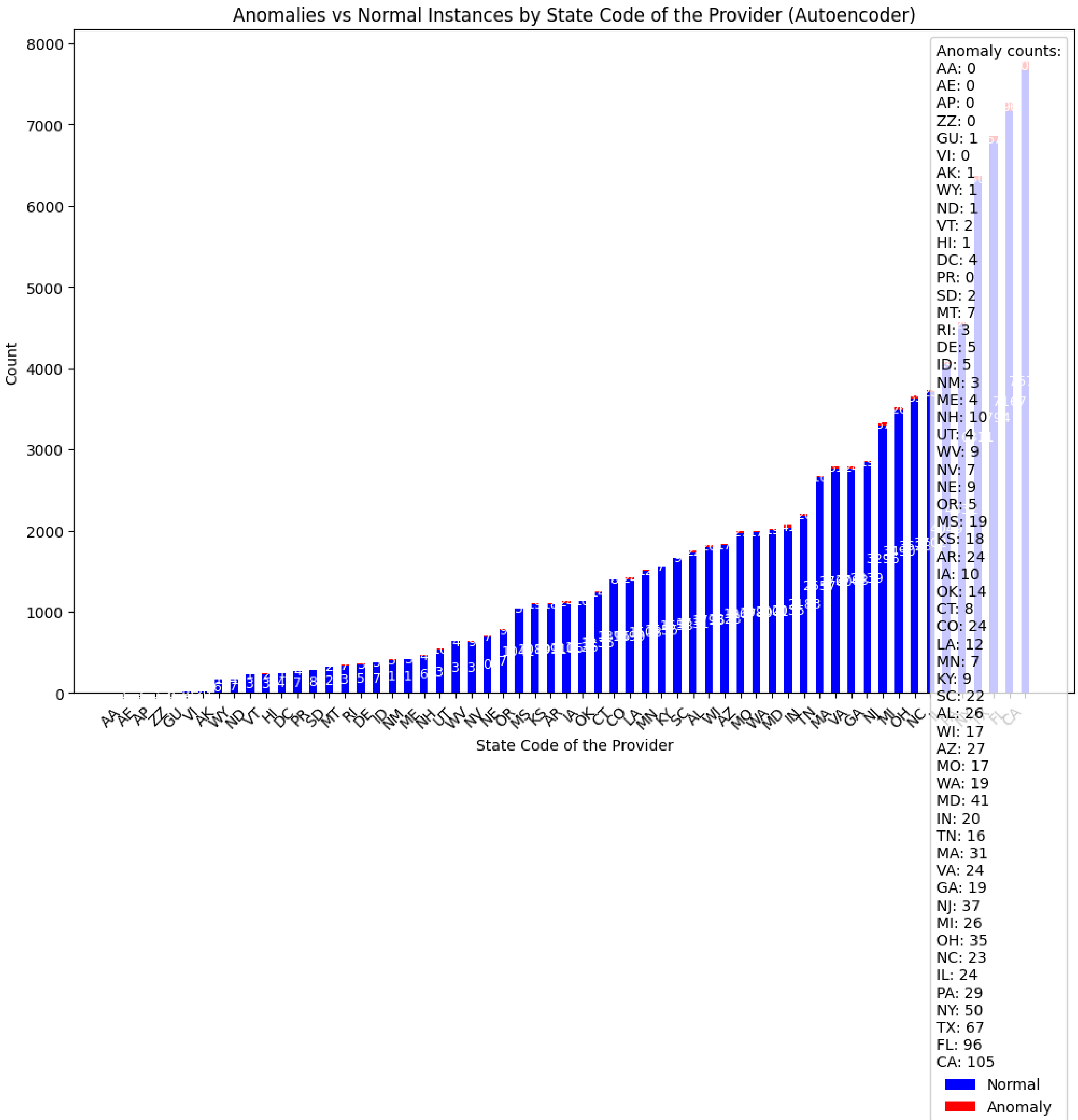
In [28]:
```python
original_gender_column = df['Gender of the Provider'].unique()
freq_encoded_gender_column = df['Gender of the Provider_Freq'].unique()

gender_mapping_dict = {}
for original_value in original_gender_column:
    freq_encoded_value = df.loc[df['Gender of the Provider'] == original_value, 'Gender of the Provider_Freq'].values[0]
    gender_mapping_dict[freq_encoded_value] = original_value


#Stacked Bar Plot for 'Gender of the Provider'
gender_grouped_autoencoder = data.groupby(['Gender of the Provider_Freq', 'Autoencoder_Anomaly']).size().unstack().fillna(0)
gender_grouped_autoencoder.columns = ['Normal', 'Anomaly'] if gender_grouped_autoencoder.shape[1] == 2 else (['Normal'] if 0 in gender_gr
gender_grouped_autoencoder = gender_grouped_autoencoder.reset_index()

# Mapping the frequency-encoded values back to the original values for labels
gender_grouped_autoencoder['Gender of the Provider'] = gender_grouped_autoencoder['Gender of the Provider_Freq'].map(gender_mapping_dict)
```

```
# Plot stacked bar plot for 'Gender of the Provider'
plt.figure(figsize=(12, 8))
bar_width = 0.5
bars1 = plt.bar(gender_grouped_autoencoder['Gender of the Provider'], gender_grouped_autoencoder['Normal'], color='blue', label='Normal',
bars2 = plt.bar(gender_grouped_autoencoder['Gender of the Provider'], gender_grouped_autoencoder['Anomaly'], bottom=gender_grouped_autoen
plt.xlabel('Gender of the Provider')
plt.ylabel('Count')
plt.title('Anomalies vs Normal Instances by Gender of the Provider (Autoencoder)')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.show()
```



We can see that more anamolies are present among Males than Females

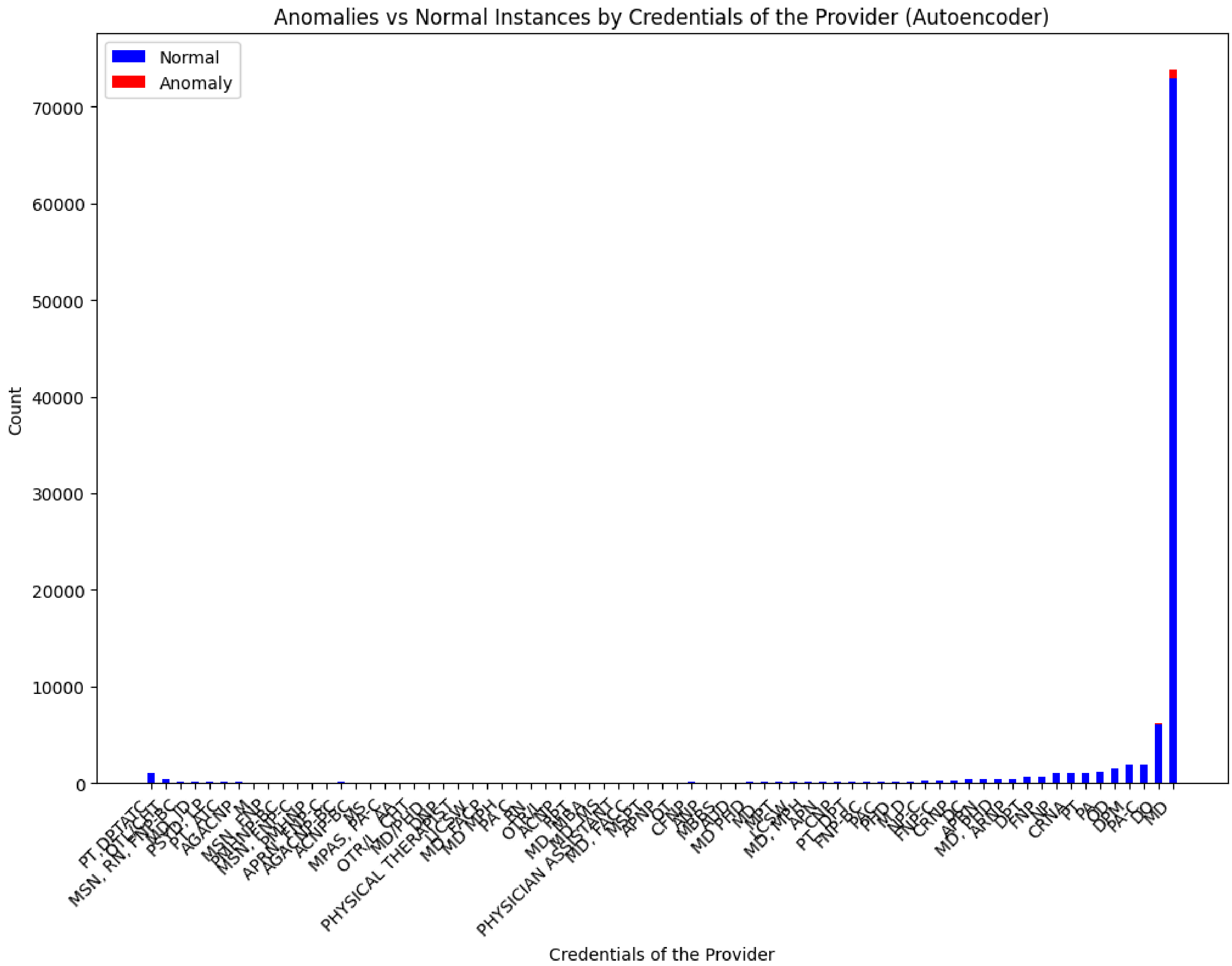
In [29]:
```
# Mapping frequency-encoded values back to original values for State Code of the Provider
original_state_code_column = df['State Code of the Provider'].unique()
freq_encoded_state_code_column = df['State Code of the Provider_Freq'].unique()

state_code_mapping_dict = {}
for original_value in original_state_code_column:
    freq_encoded_value = df.loc[df['State Code of the Provider'] == original_value, 'State Code of the Provider_Freq'].values[0]
    state_code_mapping_dict[freq_encoded_value] = original_value

# Plot Stacked Bar Plot for 'State Code of the Provider_Freq'
state_code_grouped_autoencoder = data.groupby(['State Code of the Provider_Freq', 'Autoencoder_Anomaly']).size().unstack().fillna(0)
state_code_grouped_autoencoder.columns = ['Normal', 'Anomaly'] if state_code_grouped_autoencoder.shape[1] == 2 else (['Normal'] if 0 in s
state_code_grouped_autoencoder = state_code_grouped_autoencoder.reset_index()

# Map the frequency-encoded values back to the original values for labels
state_code_grouped_autoencoder['State Code of the Provider'] = state_code_grouped_autoencoder['State Code of the Provider_Freq'].map(stat

# Calculate total counts for each state code
state_code_grouped_autoencoder['Total'] = state_code_grouped_autoencoder['Normal'] + state_code_grouped_autoencoder['Anomaly']

# Plot stacked bar plot for 'State Code of the Provider'
plt.figure(figsize=(12, 8))
bar_width = 0.5
bars1 = plt.bar(state_code_grouped_autoencoder['State Code of the Provider'], state_code_grouped_autoencoder['Normal'], color='blue', lab
bars2 = plt.bar(state_code_grouped_autoencoder['State Code of the Provider'], state_code_grouped_autoencoder['Anomaly'], bottom=state_cod

# Annotate counts on the bars
for i in range(state_code_grouped_autoencoder.shape[0]):
    plt.text(i, state_code_grouped_autoencoder['Normal'][i] / 2, int(state_code_grouped_autoencoder['Normal'][i]), ha='center', va='cente
    if state_code_grouped_autoencoder['Anomaly'][i] > 0:
        plt.text(i, state_code_grouped_autoencoder['Normal'][i] + state_code_grouped_autoencoder['Anomaly'][i] / 2, int(state_code_groupe

plt.xlabel('State Code of the Provider')
plt.ylabel('Count')
plt.title('Anomalies vs Normal Instances by State Code of the Provider (Autoencoder)')
plt.xticks(rotation=45, ha='right')
plt.legend(loc='upper right', title=f'Anomaly counts:\n' + '\n'.join([f"{row['State Code of the Provider']}: {int(row['Anomaly'])}" for _
plt.show()
```



We can see that most anomolies occur in the state of California - 105, followed by Florida- 96 and Texas- 67

In [30]:

```
original_credentials_column = df['Credentials of the Provider'].unique()
freq_encoded_credentials_column = df['Credentials of the Provider_Freq'].unique()

credentials_mapping_dict = {}
for original_value in original_credentials_column:
    freq_encoded_value = df.loc[df['Credentials of the Provider'] == original_value, 'Credentials of the Provider_Freq'].values[0]
    credentials_mapping_dict[freq_encoded_value] = original_value

credentials_grouped_autoencoder = data.groupby(['Credentials of the Provider_Freq', 'Autoencoder_Anomaly']).size().unstack().fillna(0)
credentials_grouped_autoencoder.columns = ['Normal', 'Anomaly'] if credentials_grouped_autoencoder.shape[1] == 2 else (['Normal'] if 0 in
credentials_grouped_autoencoder = credentials_grouped_autoencoder.reset_index()

# Mapping the frequency-encoded values back to the original values for labels
credentials_grouped_autoencoder['Credentials of the Provider'] = credentials_grouped_autoencoder['Credentials of the Provider_Freq'].map(

# stacked bar plot for 'Credentials of the Provider'
plt.figure(figsize=(12, 8))
bar_width = 0.5
bars1 = plt.bar(credentials_grouped_autoencoder['Credentials of the Provider'], credentials_grouped_autoencoder['Normal'], color='blue',
bars2 = plt.bar(credentials_grouped_autoencoder['Credentials of the Provider'], credentials_grouped_autoencoder['Anomaly'], bottom=creden
plt.xlabel('Credentials of the Provider')
plt.ylabel('Count')
plt.title('Anomalies vs Normal Instances by Credentials of the Provider (Autoencoder)')
plt.xticks(rotation=45, ha='right')
plt.legend()
plt.show()
```



We can see that most anamolies are present for credential MD, followed by DO

In [31]:

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Get the reconstructed data from the autoencoder
X_reconstructed = autoencoder.predict(X_scaled)

# Calculate the reconstruction errors
reconstruction_errors = np.mean(np.abs(X_scaled - X_reconstructed), axis=1)

# Calculate RMSE for the entire dataset
rmse = np.sqrt(mean_squared_error(X_scaled, X_reconstructed))

# Calculate MAE for the entire dataset
mae = mean_absolute_error(X_scaled, X_reconstructed)

print(f'RMSE: {rmse}')
print(f'MAE: {mae}')
```

```
3125/3125 ━━━━━━━━━━━━━━━━━━━━ 1s 232us/step
RMSE: 0.7415441297229188
MAE: 0.42007666479939965
```

Interpretation of Results

Root Mean Squared Error (RMSE):

- **Value:** 0.744
- **Interpretation:** The RMSE measures the average magnitude of reconstruction errors. An RMSE of 0.744 suggests a moderate level of error, so for our medical dataset of healthcare providers with 100,000 entries, this indicates a moderate reconstruction accuracy.

Mean Absolute Error (MAE):

- **Value:** 0.426
- **Interpretation:** The MAE measures the average absolute difference between the original and reconstructed data. An MAE of 0.426 indicates a moderate average error. This suggests the model's performance is fairly reasonable but may need improvement for critical applications.

In [ ]:

In [ ]: