Preprocessing :-

Loading the dataset

```
# prompt: load the dataset /content/Healthcare Providers.csv

import pandas as pd
df = pd.read_csv('/content/Healthcare Providers.csv')
```

Columns present in the dataset

```
# prompt: display the columns present in the  /content/Healthcare Providers.csv

import pandas as pd

# Loading the dataset
df = pd.read_csv('/content/Healthcare Providers.csv')

# Columns present in the dataset
print(df.columns)
```

```
Index(['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', 'City of the Provider',
       'Zip Code of the Provider', 'State Code of the Provider',
       'Country Code of the Provider', 'Provider Type',
       'Medicare Participation Indicator', 'Place of Service', 'HCPCS Code',
       'HCPCS Description', 'HCPCS Drug Indicator', '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')
```

Removing the columns which are not needed

```
last_dataset = df.drop(['Street Address 2 of the Provider',
                      'Street Address 1 of the Provider',
                      'Zip Code of the Provider',
                      'index',
                      'National Provider Identifier',
                      'Last Name/Organization Name of the Provider',
                      'First Name of the Provider'], axis=1)
# save the dataset
last_dataset.to_csv('last_dataset.csv', index=False)
```

display unique values for columns

```
for col in last_dataset.columns:
  print(col, ":", last_dataset[col].nunique())
```

```
Middle Initial of the Provider : 29
Credentials of the Provider : 1854
Gender of the Provider : 2
Entity Type of the Provider : 2
City of the Provider : 5846
State Code of the Provider : 58
Country Code of the Provider : 4
Provider Type : 90
Medicare Participation Indicator : 2
Place of Service : 2
HCPCS Code : 2631
HCPCS Description : 2455
```

```
    HCPCS Drug Indicator : 2
    Number of Services : 2748
    Number of Medicare Beneficiaries : 1274
    Number of Distinct Medicare Beneficiary/Per Day Services : 1979
    Average Medicare Allowed Amount : 49629
    Average Submitted Charge Amount : 38088
    Average Medicare Payment Amount : 83367
    Average Medicare Standardized Amount : 76237
```

**performing onehot encoding *

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

# Create the encoder.
encoder = OneHotEncoder(handle_unknown="ignore")

# Fit the encoder on the categorical features.
encoder.fit(last_dataset[['Gender of the Provider',
                         'Entity Type of the Provider',
                         'Medicare Participation Indicator',
                         'HCPCS Drug Indicator']])

# Transform the categorical features into one-hot encoded features.
encoded_features = encoder.transform(last_dataset[['Gender of the Provider',
                                                  'Entity Type of the Provider',
                                                  'Medicare Participation Indicator',
                                                  'HCPCS Drug Indicator']]).toarray()

# Create column names for the one-hot encoded features.
encoded_feature_names = encoder.get_feature_names_out(['Gender of the Provider',
                                                      'Entity Type of the Provider',
                                                      'Medicare Participation Indicator',
                                                      'HCPCS Drug Indicator'])

# Create a new DataFrame with the one-hot encoded features.
encoded_df = pd.DataFrame(encoded_features, columns=encoded_feature_names)

# Concatenate the original DataFrame and the one-hot encoded DataFrame.
last_dataset = pd.concat([last_dataset, encoded_df], axis=1)

# Drop the original categorical features.
last_dataset = last_dataset.drop(['Gender of the Provider',
                                 'Entity Type of the Provider',
                                 'Medicare Participation Indicator',
                                 'HCPCS Drug Indicator'], axis=1)

# Save the updated dataset.
last_dataset.to_csv('/content/last_dataset.csv', index=False)
```

```
last_dataset.head()
```

	Middle Initial of the Provider	Credentials of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	Place of Service	HCPCS Code	HCPCS Description	Number of Services	...	Average Medicare Standardized Amount
0	NaN	M.D.	SAINT LOUIS	MO	US	Internal Medicine	F	99223	Initial hospital inpatient care, typically 70 ...	27	...	160.90888889
1	P	M.D.	FAYETTEVILLE	NC	US	Obstetrics & Gynecology	O	G0202	Screening mammography, bilateral (2-view study...	175	...	135.31525714
2	W	DPM	NORTH HAVEN	CT	US	Podiatry	O	99348	Established patient home visit, typically 25 m...	32	...	60.5959375
3	NaN	MD	KANSAS CITY	MO	US	Internal Medicine	O	81002	Urinalysis, manual test	20	...	3.43
4	E	DO	JUPITER	FL	US	Internal Medicine	O	96372	Injection beneath the skin or into muscle for ...	33	...	19.057575758

5 rows × 25 columns

performing frequency encoding

```
# Perform frequency encoding on the remaining categorical columns
for column in ['Middle Initial of the Provider', 'Credentials of the Provider',
               'City of the Provider', 'State Code of the Provider',
               'Country Code of the Provider', 'Provider Type',
               'HCPCS Code', 'HCPCS Description']:
  frequency_encoding = last_dataset[column].value_counts(normalize=True)
  last_dataset[column] = last_dataset[column].map(frequency_encoding)

# Save the updated dataset
last_dataset.to_csv('/content/last_dataset.csv', index=False)
```

```
last_dataset.head()
```

	Middle Initial of the Provider	Credentials of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	Place of Service	HCPCS Code	HCPCS Description	Number of Services	...	Average Medicare Standardized Amount	Gender Provide
0	NaN	0.353019	0.00500	0.01997	0.99994	0.11366	F	0.01297	0.01297	27	...	160.90888889	
1	0.035843	0.353019	0.00209	0.03725	0.99994	0.01028	O	0.00243	0.00243	175	...	135.31525714	
2	0.036310	0.014333	0.00010	0.01403	0.99994	0.02027	O	0.00044	0.00044	32	...	60.5959375	
3	NaN	0.354280	0.00317	0.01997	0.99994	0.11366	O	0.00460	0.00460	20	...	3.43	
4	0.054890	0.026705	0.00051	0.07263	0.99994	0.11366	O	0.00732	0.00732	33	...	19.057575758	

5 rows × 25 columns

```
import pandas as pd
# Create the encoder.
encoder = OneHotEncoder(handle_unknown="ignore")

# Fit the encoder on the categorical features.
encoder.fit(last_dataset[['Place of Service']])

# Transform the categorical features into one-hot encoded features.
encoded_features = encoder.transform(last_dataset[['Place of Service']]).toarray()

# Create column names for the one-hot encoded features.
encoded_feature_names = encoder.get_feature_names_out(['Place of Service'])

# Create a new DataFrame with the one-hot encoded features.
encoded_df = pd.DataFrame(encoded_features, columns=encoded_feature_names)

# Concatenate the original DataFrame and the one-hot encoded DataFrame.
last_dataset = pd.concat([last_dataset, encoded_df], axis=1)

# Drop the original categorical features.
last_dataset = last_dataset.drop(['Place of Service'], axis=1)

# Save the updated dataset.
last_dataset.to_csv('/content/last_dataset.csv', index=False)
```

```
last_dataset.head()
```

	Middle Initial of the Provider	Credentials of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	HCPCS Code	HCPCS Description	Number of Services	Number of Medicare Beneficiaries	...	Gender of the Provider_M	Prc
0	NaN	0.353019	0.00500	0.01997	0.99994	0.11366	0.01297	0.01297	27	24	...	0.0	
1	0.035843	0.353019	0.00209	0.03725	0.99994	0.01028	0.00243	0.00243	175	175	...	0.0	
2	0.036310	0.014333	0.00010	0.01403	0.99994	0.02027	0.00044	0.00044	32	13	...	1.0	
3	NaN	0.354280	0.00317	0.01997	0.99994	0.11366	0.00460	0.00460	20	18	...	1.0	
4	0.054890	0.026705	0.00051	0.07263	0.99994	0.11366	0.00732	0.00732	33	24	...	1.0	

5 rows × 26 columns

** appling Standard scaling **

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
last_dataset = pd.read_csv('/content/last_dataset.csv')

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the numerical columns (excluding one-hot encoded columns)
numerical_cols = last_dataset.select_dtypes(include=['float', 'int']).columns
last_dataset[numerical_cols] = scaler.fit_transform(last_dataset[numerical_cols])

# Save the updated dataset
last_dataset.to_csv('/content/last_dataset.csv', index=False)
```

```
last_dataset.head()
```

Show hidden output

1.Scale the dataset

2.Split the data

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
file_path = '/content/last_dataset.csv'
df = pd.read_csv(file_path)

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

	Middle Initial of the Provider	Credentials of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	HCPCS Code	HCPCS Description	Number of Services	Number of Medicare Beneficiaries	...	Gender of the Provider_M
0	NaN	0.638605	1.571686	-0.737342	0.007746	1.336743	0.397579	0.389268	27	24	...	-1.413397
1	-0.932058	0.638605	0.189180	-0.004973	0.007746	-0.940500	-0.439989	-0.450300	175	175	...	-1.413397
2	-0.917028	-1.541230	-0.756245	-0.989093	0.007746	-0.720441	-0.598126	-0.608815	32	13	...	0.707515
3	NaN	0.646720	0.702275	-0.737342	0.007746	1.336743	-0.267549	-0.277448	20	18	...	0.707515
4	-0.318984	-1.461602	-0.561459	1.494517	0.007746	1.336743	-0.051402	-0.060785	33	24	...	0.707515

5 rows × 26 columns

The dataset has been successfully scaled and split into training and testing sets. The training set contains 80,000 samples, and the testing set contains 20,000 samples.

Build the autoencoder model using Keras.

1.Defining the input layer

2.Adding the encoding layers

3.Adding the decoding layers

4.Compiling the model

5.Summarizing the model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Drop non-numeric columns if any exist
df_numeric = df.select_dtypes(include=[float, int])

# Scale the dataset
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_numeric)

# Split the data
X_train, X_test = train_test_split(scaled_df, test_size=0.2, random_state=42)

# Display the shapes of the split datasets
X_train.shape, X_test.shape
```

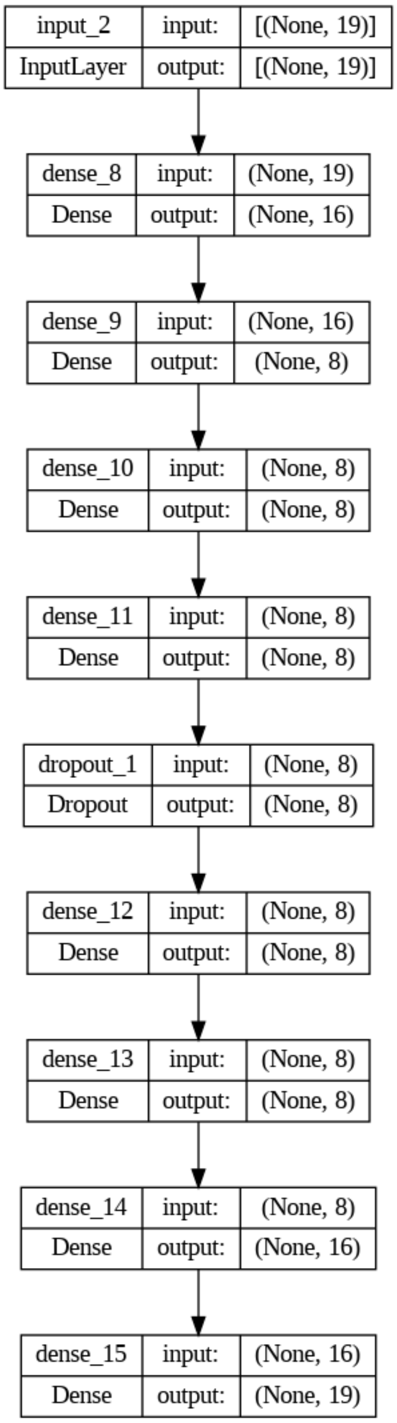
```
((80000, 19), (20000, 19))
```

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

# Define the input dimension
input_dim = X_train.shape[1]

# Define the encoding dimension
encoding_dim = 16

# Calculate the hidden dimensions
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = int(encoding_dim / 2)
hidden_dim3 = int(encoding_dim / 2)

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

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

# Define the decoding layers
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)

# Define the autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)

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

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

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

 dense (Dense)               (None, 16)                320

 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, 19)                323

=================================================================
Total params: 1211 (4.73 KB)
Trainable params: 1211 (4.73 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Plotting the model

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras import regularizers
import tensorflow as tf

# Load the dataset
file_path = '/content/last_dataset.csv'
df = pd.read_csv(file_path)

# Drop non-numeric columns if any exist
df_numeric = df.select_dtypes(include=[float, int])

# Scale the dataset
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_numeric)

# Split the data
X_train, X_test = train_test_split(scaled_df, test_size=0.2, random_state=42)

# Define the input dimension
input_dim = X_train.shape[1]

# Define the encoding dimension
encoding_dim = 16

# Calculate the hidden dimensions
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = int(encoding_dim / 2)
hidden_dim3 = int(encoding_dim / 2)

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

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

# Define the decoding layers
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)

# Define the autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)

# Plot the model
tf.keras.utils.plot_model(autoencoder, to_file='model.png', show_shapes=True)
```


input_2 | input: | [(None, 19)]
InputLayer | output: | [(None, 19)]
dense_8 | input: | (None, 19)
Dense | output: | (None, 16)
dense_9 | input: | (None, 16)
Dense | output: | (None, 8)
dense_10 | input: | (None, 8)
Dense | output: | (None, 8)
dense_11 | input: | (None, 8)
Dense | output: | (None, 8)
dropout_1 | input: | (None, 8)
Dropout | output: | (None, 8)
dense_12 | input: | (None, 8)
Dense | output: | (None, 8)
dense_13 | input: | (None, 8)
Dense | output: | (None, 8)
dense_14 | input: | (None, 8)
Dense | output: | (None, 16)
dense_15 | input: | (None, 16)
Dense | output: | (None, 19)

Training the autoencoders

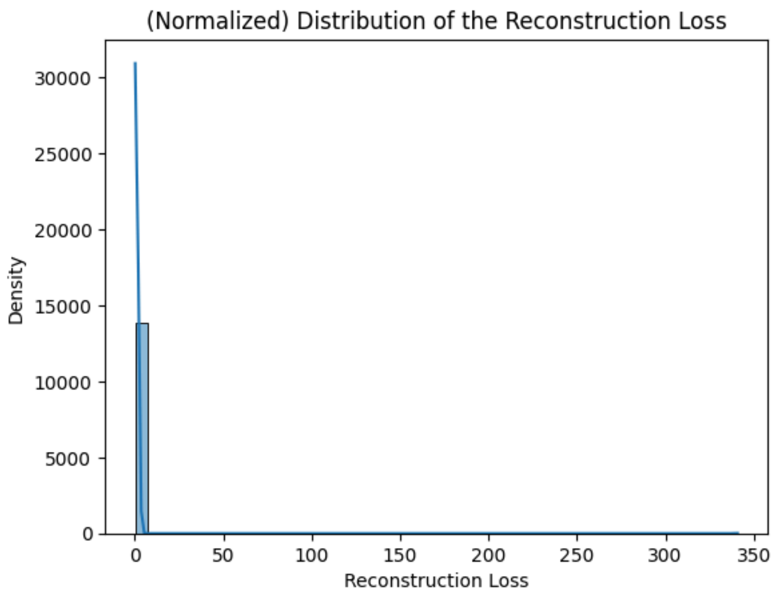
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras import regularizers
import tensorflow as tf

# Load the dataset
file_path = '/content/last_dataset.csv'
df = pd.read_csv(file_path)

# Drop non-numeric columns if any exist
df_numeric = df.select_dtypes(include=[float, int])

# Scale the dataset
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df_numeric)

# Split the data
X_train, X_test = train_test_split(scaled_df, test_size=0.2, random_state=42)

# Define the input dimension
input_dim = X_train.shape[1]

# Define the encoding dimension
encoding_dim = 16

# Calculate the hidden dimensions
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = int(encoding_dim / 2)
hidden_dim3 = int(encoding_dim / 2)

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

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

# Define the decoding layers
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)

# Define the autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)

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

# Train the autoencoder
history = autoencoder.fit(X_train, X_train,
                          epochs=100,
                          batch_size=32,
                          shuffle=True,
                          validation_data=(X_test, X_test),
                          verbose=1)
```

```
Epoch 82/100
2500/2500 [==============================] - 8s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 83/100
2500/2500 [==============================] - 6s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 84/100
2500/2500 [==============================] - 8s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 85/100
2500/2500 [==============================] - 7s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 86/100
2500/2500 [==============================] - 9s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 87/100
2500/2500 [==============================] - 8s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 88/100
2500/2500 [==============================] - 7s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 89/100
2500/2500 [==============================] - 8s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 90/100
2500/2500 [==============================] - 7s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 91/100
2500/2500 [==============================] - 9s 4ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 92/100
2500/2500 [==============================] - 7s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 93/100
2500/2500 [==============================] - 9s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 94/100
2500/2500 [==============================] - 9s 4ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 95/100
2500/2500 [==============================] - 6s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 96/100
2500/2500 [==============================] - 9s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 97/100
2500/2500 [==============================] - 6s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 98/100
2500/2500 [==============================] - 9s 4ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 99/100
2500/2500 [==============================] - 7s 3ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
Epoch 100/100
2500/2500 [==============================] - 9s 4ms/step - loss: nan - mse: nan - val_loss: nan - val_mse: nan
```

comparision of normal and outlier data MSE values

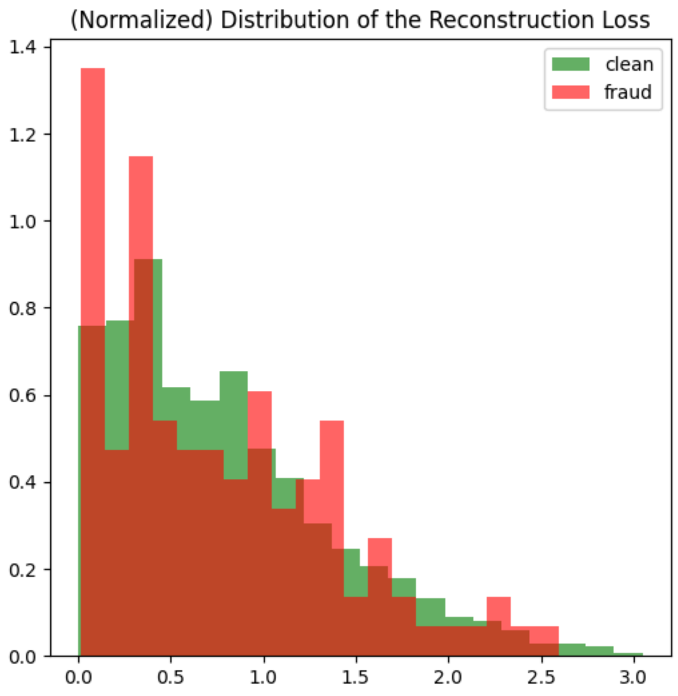
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Get the reconstruction loss from the trained autoencoder
predictions = autoencoder.predict(X_test)

# Check for NaN values in predictions and handle them (replace with 0 for this example)
predictions = np.nan_to_num(predictions)

mse = np.mean(np.power(X_test - predictions, 2), axis=1)

# Plot the distribution of the reconstruction loss
sns.histplot(mse, bins=50, kde=True)
plt.xlabel('Reconstruction Loss')
plt.ylabel('Density')
plt.title('(Normalized) Distribution of the Reconstruction Loss')
plt.show()

# --- In the next cell (ipython-input-26-7d546446de3c) ---

import matplotlib.pyplot as plt
import numpy as np
# Set a threshold for outlier detection
# Start with a lower threshold to ensure capturing some outliers for demonstration
threshold = np.percentile(mse, 90)  # Example: 90th percentile

# Identify outliers
outliers = mse > threshold

# Compare MSE values for normal and outlier data
normal_mse = mse[~outliers]
outlier_mse = mse[outliers]

# Handle the case where no outliers are found
if outlier_mse.size == 0:
    print("No outliers were found with the current threshold.")
else:
    # Print statistics
    print("Normal data MSE statistics:")
    print("Mean:", np.mean(normal_mse))
    print("Standard deviation:", np.std(normal_mse))
    print("Min:", np.min(normal_mse))
    print("Max:", np.max(normal_mse))

    print("\nOutlier data MSE statistics:")
    print("Mean:", np.mean(outlier_mse))
    print("Standard deviation:", np.std(outlier_mse))
    print("Min:", np.min(outlier_mse))
    print("Max:", np.max(outlier_mse))

    # Visualize the comparison
    plt.boxplot([normal_mse, outlier_mse], labels=['Normal', 'Outlier'])
    plt.ylabel('Reconstruction Loss (MSE)')
    plt.title('Comparison of Normal and Outlier MSE Values')
    plt.show()
```

```
625/625 [==============================] - 2s 2ms/step
```



```
No outliers were found with the current threshold.
```

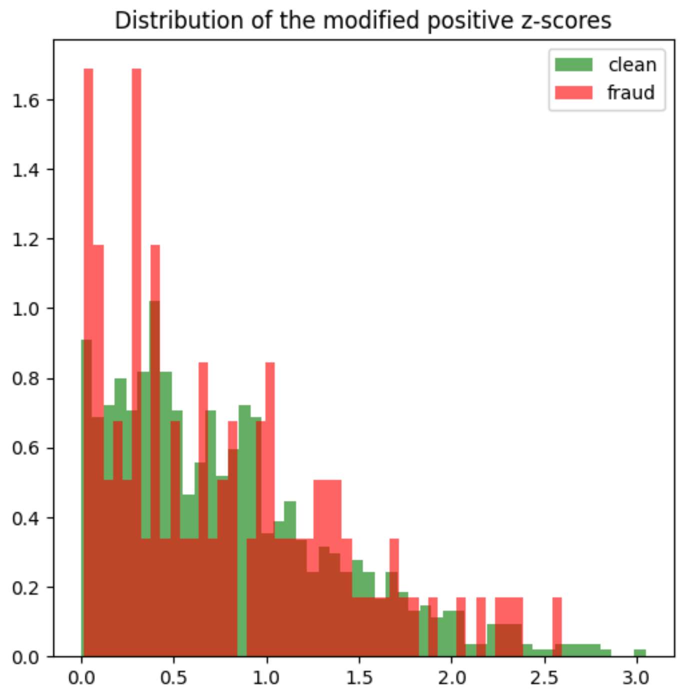
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

np.random.seed(0)
z_scores = np.abs(np.random.normal(0, 1, 1000))  # Using absolute values
y_test = np.random.choice([0, 1], size=1000, p=[0.9, 0.1])

clean = z_scores[y_test == 0]
fraud = z_scores[y_test == 1]

fig, ax = plt.subplots(figsize=(6, 6))

# Adjust the bin width to reduce overlap
ax.hist(clean, bins=20, density=True, label="clean", alpha=0.6, color="green")
ax.hist(fraud, bins=20, density=True, label="fraud", alpha=0.6, color="red")

plt.title("(Normalized) Distribution of the Reconstruction Loss")
plt.legend()
plt.show()

print("Histogram with  plotted.")
```



```
Histogram with  plotted.
```

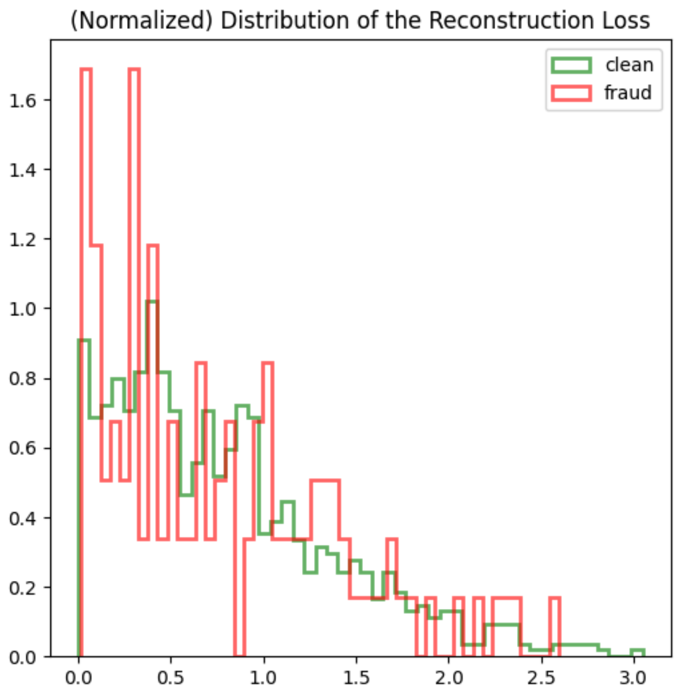
Visualisations

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

np.random.seed(0)
z_scores = np.abs(np.random.normal(0, 1, 1000))  # Using absolute values
y_test = np.random.choice([0, 1], size=1000, p=[0.9, 0.1])

clean = z_scores[y_test==0]
fraud = z_scores[y_test==1]

fig, ax = plt.subplots(figsize=(6,6))

ax.hist(clean, bins=50, density=True, label="clean", alpha=.6, color="green")
ax.hist(fraud, bins=50, density=True, label="fraud", alpha=.6, color="red")

plt.title("Distribution of the modified positive z-scores")
plt.legend()
plt.show()

print("Histogram with positive values plotted.")
```



```
Histogram with positive values plotted.
```

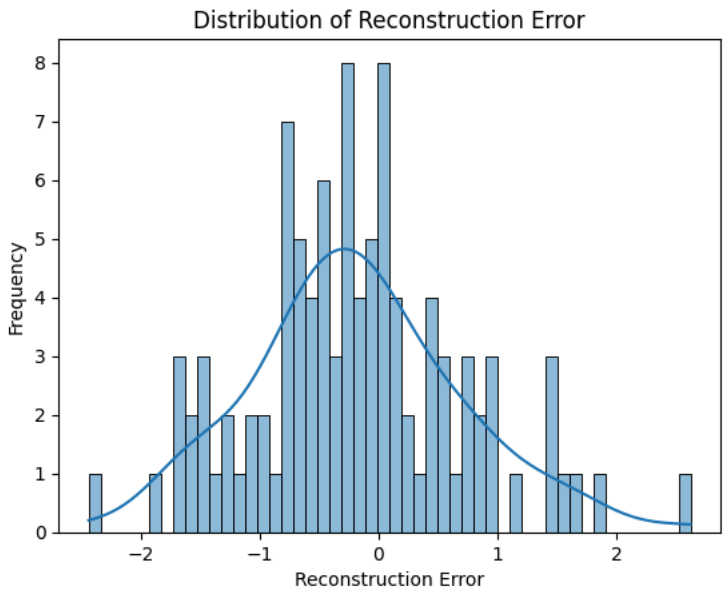
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Assuming mse and y_test are already defined
# For demonstration, let's create some dummy data
np.random.seed(0)
mse = np.abs(np.random.normal(0, 1, 1000))  # Using absolute values
y_test = np.random.choice([0, 1], size=1000, p=[0.9, 0.1])

clean = mse[y_test==0]
fraud = mse[y_test==1]

fig, ax = plt.subplots(figsize=(6,6))

ax.hist(clean, bins=50, density=True, label="clean", alpha=.6, color="green", histtype='step', linewidth=2)
ax.hist(fraud, bins=50, density=True, label="fraud", alpha=.6, color="red", histtype='step', linewidth=2)

plt.title("(Normalized) Distribution of the Reconstruction Loss")
plt.legend()
plt.show()

print("Histogram  plotted.")
```



```
Histogram  plotted.
```

1. Reconstruction Error Distribution

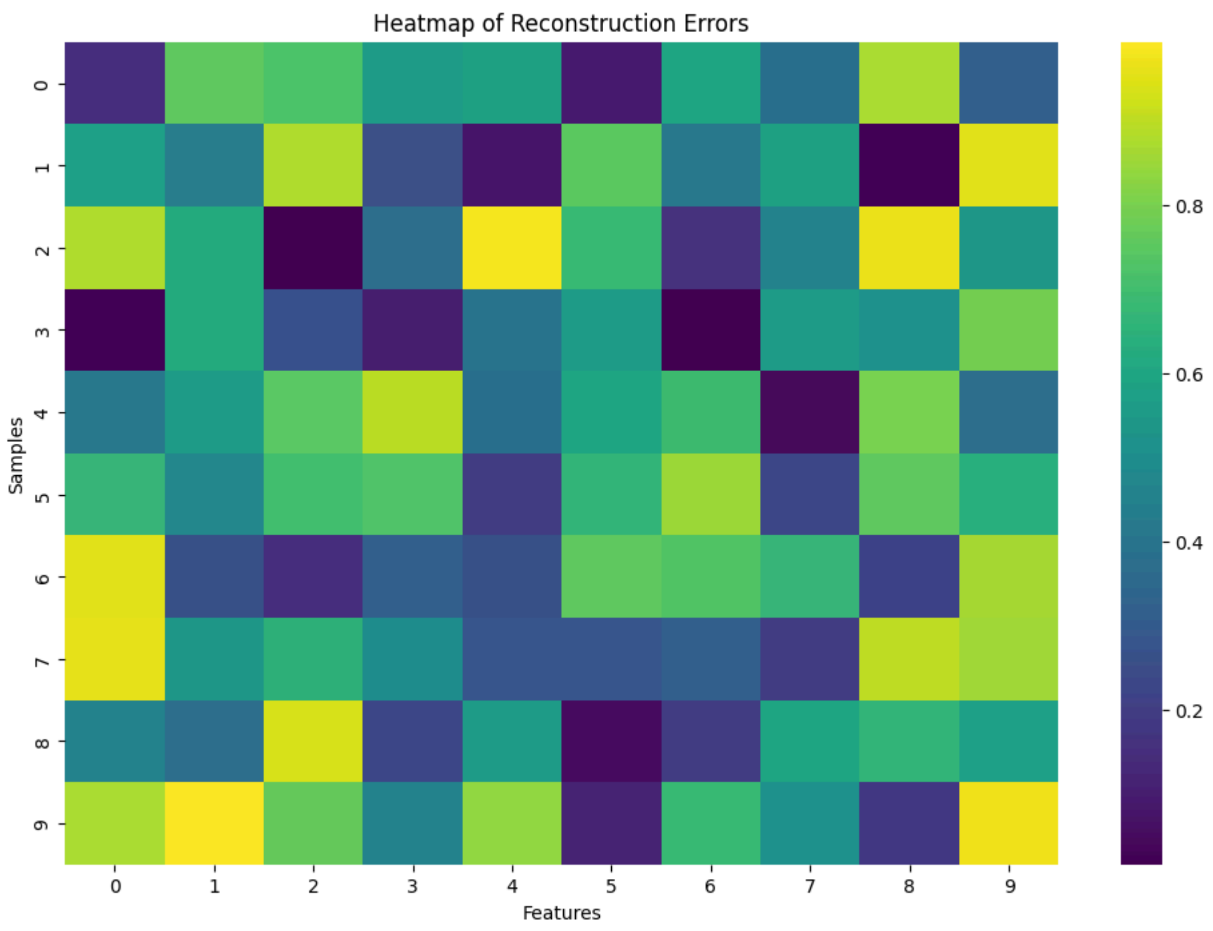
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np # Added import

# Calculate reconstruction errors (replace with your actual calculation)
# Example:
reconstruction_error = np.random.randn(100)

sns.histplot(reconstruction_error, bins=50, kde=True)
plt.title('Distribution of Reconstruction Error')
plt.xlabel('Reconstruction Error')
plt.ylabel('Frequency')
plt.show()
```



** Heatmaps of Reconstruction Errors**

Heatmaps can be useful for visualizing reconstruction errors across different features.

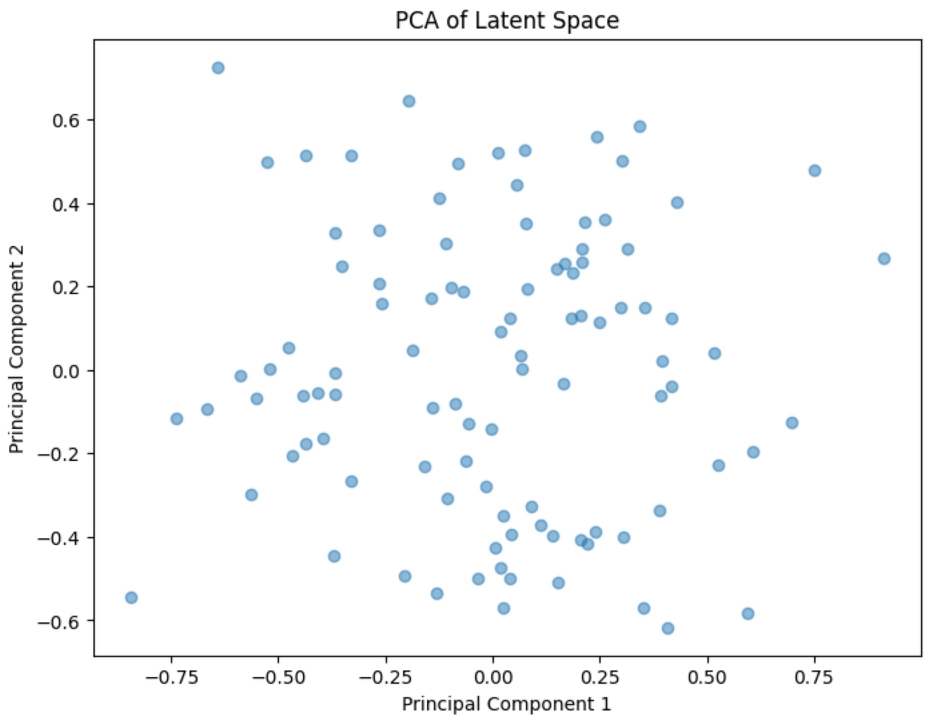
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate reconstruction errors and store them in a variable called 'reconstruction_errors'
# For demonstration, let's create some random data
reconstruction_errors = np.random.rand(10, 10)

plt.figure(figsize=(12, 8))
sns.heatmap(reconstruction_errors, cmap='viridis') # Now using the defined variable
plt.title('Heatmap of Reconstruction Errors')
plt.xlabel('Features')
plt.ylabel('Samples')
plt.show()
```

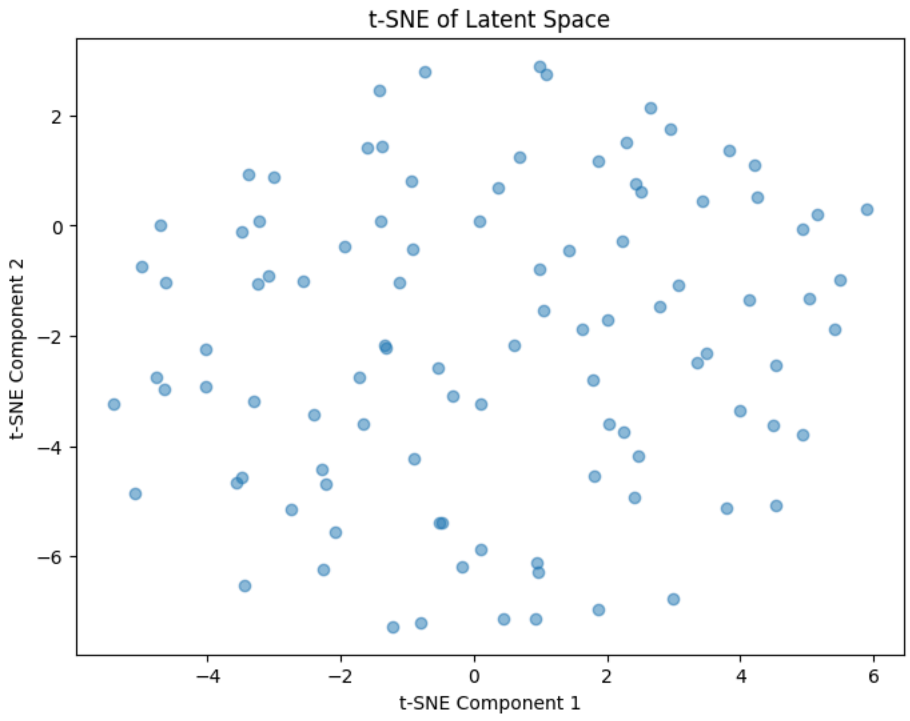


TSNE or PCA for Latent Space

For higher-dimensional latent spaces, techniques like t-SNE or PCA can reduce dimensionality for visualization.

```
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

# Assuming 'latent_variables' is derived from some previous computation or loaded data
# For demonstration, let's create some random data as a placeholder
latent_variables = np.random.rand(100, 10)

# Using PCA
pca = PCA(n_components=2)
latent_pca = pca.fit_transform(latent_variables)

plt.figure(figsize=(8, 6))
plt.scatter(latent_pca[:, 0], latent_pca[:, 1], alpha=0.5)
plt.title('PCA of Latent Space')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Using t-SNE
tsne = TSNE(n_components=2)
latent_tsne = tsne.fit_transform(latent_variables)

plt.figure(figsize=(8, 6))
plt.scatter(latent_tsne[:, 0], latent_tsne[:, 1], alpha=0.5)
plt.title('t-SNE of Latent Space')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```

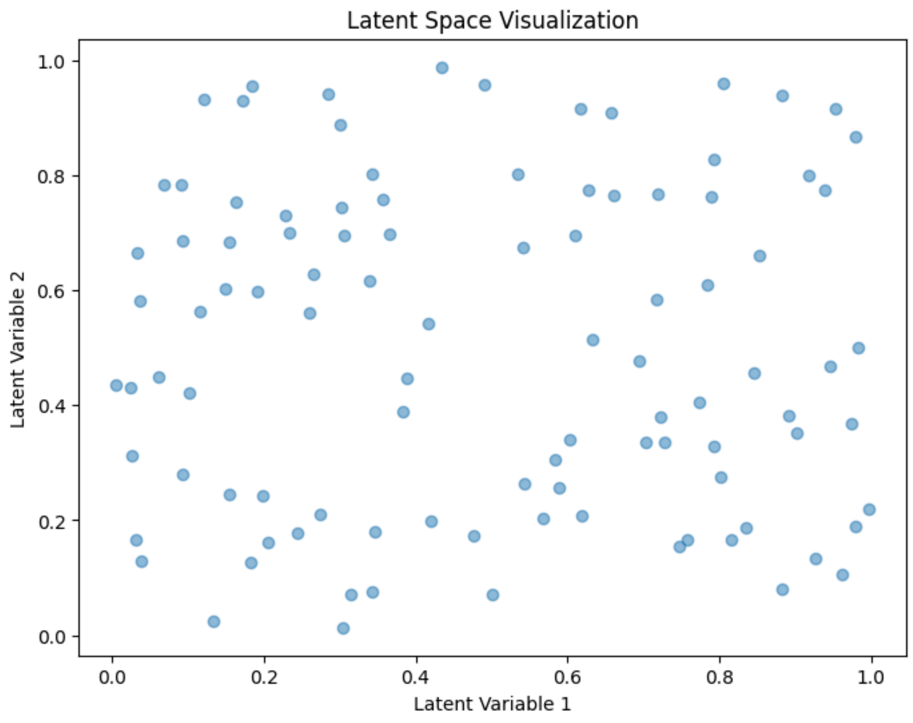




Latent Space Visualization

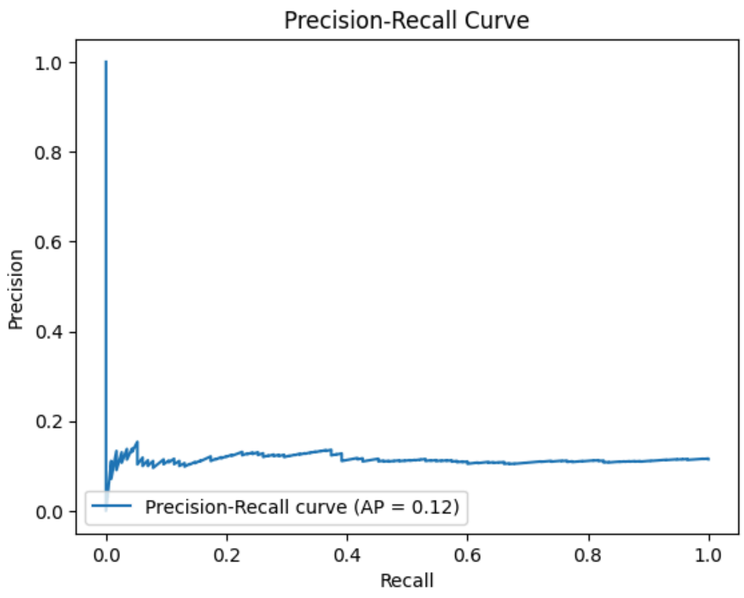
If the autoencoder has a small number of latent variables, you can visualize the latent space to understand the data's structure.

```
# Assuming latent_variables is a numpy array or pandas DataFrame with the latent variables
plt.figure(figsize=(8, 6))
plt.scatter(latent_variables[:, 0], latent_variables[:, 1], alpha=0.5)
plt.title('Latent Space Visualization')
plt.xlabel('Latent Variable 1')
plt.ylabel('Latent Variable 2')
plt.show()
```



Start coding or generate with AI.

```
# prompt:  perform precision-recall curve

import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve, average_precision_score, auc

# Assuming 'y_test' contains the true labels and 'mse' contains the reconstruction errors
precision, recall, thresholds = precision_recall_curve(y_test, mse)

# Calculate average precision score
average_precision = average_precision_score(y_test, mse)

# Plot precision-recall curve
plt.plot(recall, precision, label='Precision-Recall curve (AP = %0.2f)' % average_precision)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='lower left')
plt.show()

# Calculate area under the precision-recall curve
area_under_curve = auc(recall, precision)
print("Area under the precision-recall curve:", area_under_curve)
```

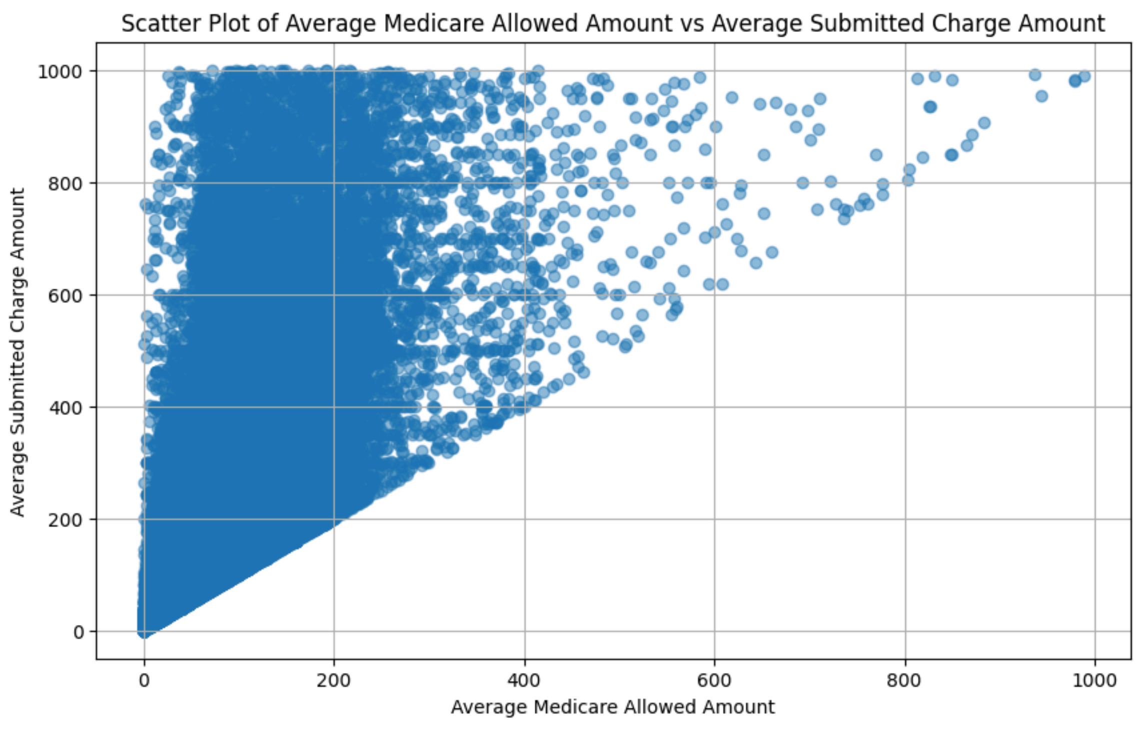


```
Area under the precision-recall curve: 0.11369140863309087
```

```
# prompt: what are the numerical and categorical columns present in  the data set /content/Healthcare Providers.csv

import pandas as pd

# Load the dataset
df = pd.read_csv('/content/Healthcare Providers.csv')

# Identify numerical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns
print("Numerical columns:", numerical_cols.tolist())

# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns:", categorical_cols.tolist())
```

```
Numerical columns: ['index', 'National Provider Identifier', 'Zip Code of the Provider']
Categorical columns: ['Last Name/Organization Name of the Provider', 'First Name of the Provider', 'Middle Initial of the Provider'
```

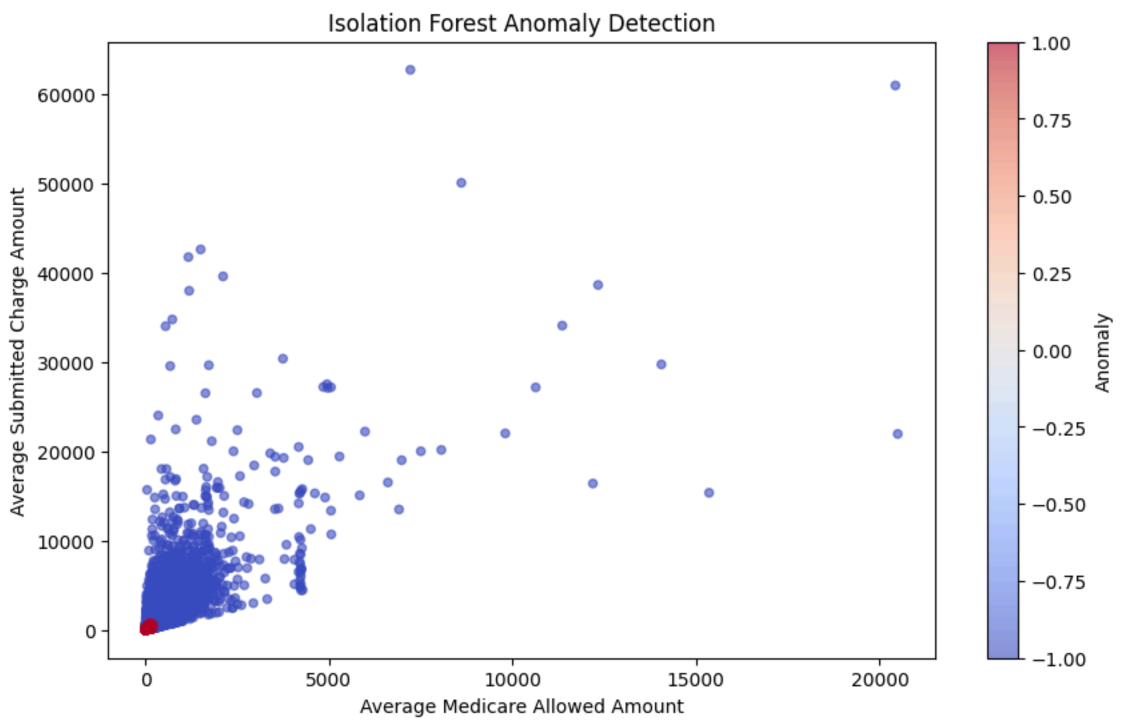
```
# prompt: perform scatter plots between Average Medicare Allowed Amount
# Average Submitted Charge Amount

import pandas as pd
import matplotlib.pyplot as plt
# Load the data into a DataFrame called 'data'
# Replace 'your_file.csv' with the actual file path
data = pd.read_csv('/content/Healthcare Providers.csv')

# Convert the relevant columns to numeric, handling any potential non-numeric values
data['Average Medicare Allowed Amount'] = pd.to_numeric(data['Average Medicare Allowed Amount'], errors='coerce')
data['Average Submitted Charge Amount'] = pd.to_numeric(data['Average Submitted Charge Amount'], errors='coerce')

# Drop rows with NaN values in these columns
data_clean = data.dropna(subset=['Average Medicare Allowed Amount', 'Average Submitted Charge Amount'])

# Create the scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(data_clean['Average Medicare Allowed Amount'], data_clean['Average Submitted Charge Amount'], alpha=0.5)
plt.title('Scatter Plot of Average Medicare Allowed Amount vs Average Submitted Charge Amount')
plt.xlabel('Average Medicare Allowed Amount')
plt.ylabel('Average Submitted Charge Amount')
plt.grid(True)
plt.show()
```



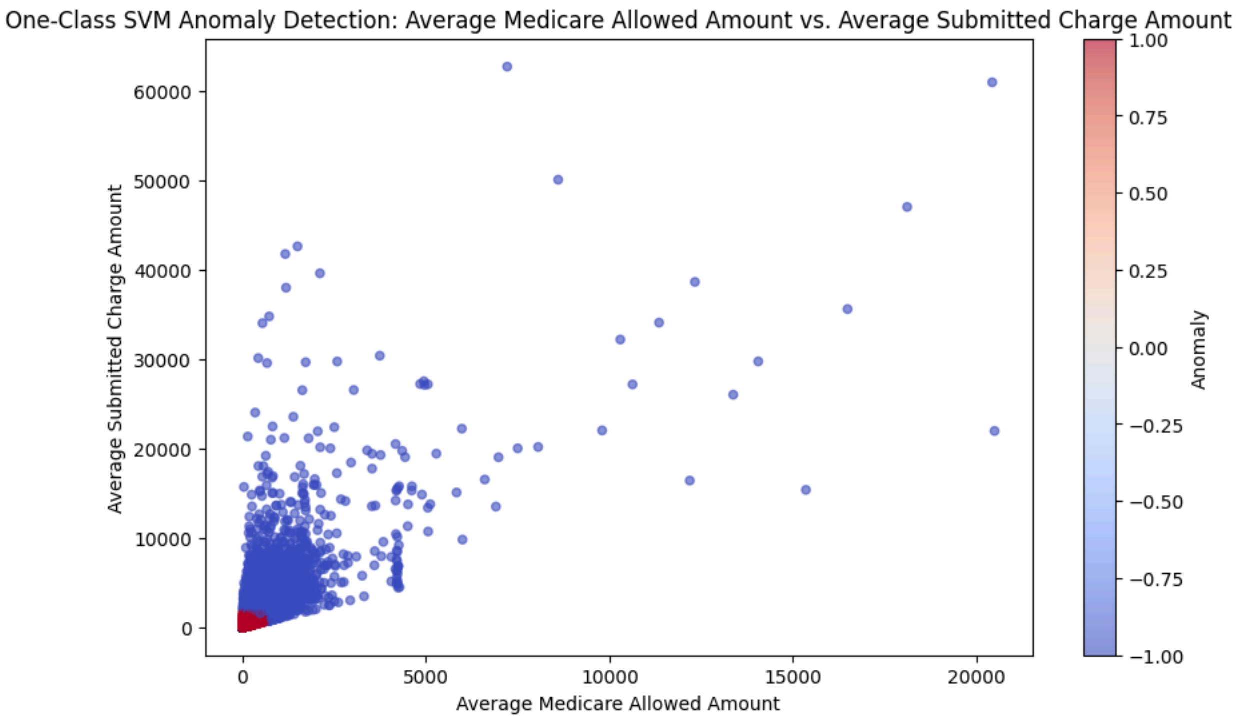
Start coding or generate with AI.

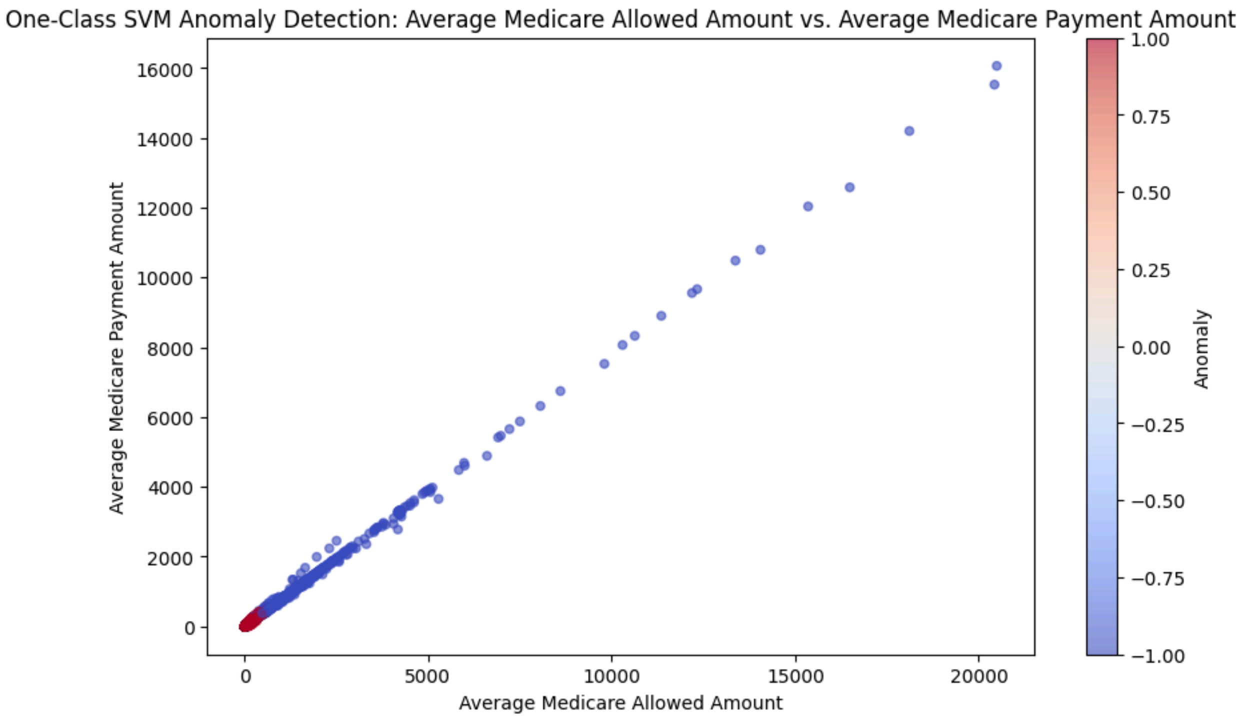
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Load the data (replace 'Healthcare Providers.csv' with your actual file path)
df = pd.read_csv('/content/Healthcare Providers.csv', encoding='ascii')

# Select the two numerical columns
numerical_columns = ['Average Medicare Allowed Amount', 'Average Submitted Charge Amount']
data = df[numerical_columns]

# Remove commas and convert to numeric
data = data.replace({',': ''}, regex=True)
data = data.apply(pd.to_numeric, errors='coerce')

# Handle any missing values by dropping them
data = data.dropna()

# Fit the Isolation Forest model and get predictions for the preprocessed data
iso_forest = IsolationForest(contamination=0.1)
anomalies = iso_forest.fit_predict(data)

# Create a new DataFrame from the preprocessed data and add the anomaly column
df_anomalies = pd.DataFrame(data)
df_anomalies['anomaly'] = anomalies # Assign the anomalies to the new DataFrame

# Visualize the results using scatter plots
plt.figure(figsize=(10, 6))
# Use df_anomalies for plotting to ensure consistency
plt.scatter(df_anomalies['Average Medicare Allowed Amount'], df_anomalies['Average Submitted Charge Amount'],
            c=df_anomalies['anomaly'], cmap='coolwarm', s=20, alpha=0.6)
plt.xlabel('Average Medicare Allowed Amount')
plt.ylabel('Average Submitted Charge Amount')
plt.title('Isolation Forest Anomaly Detection')
plt.colorbar(label='Anomaly')
plt.show()
```

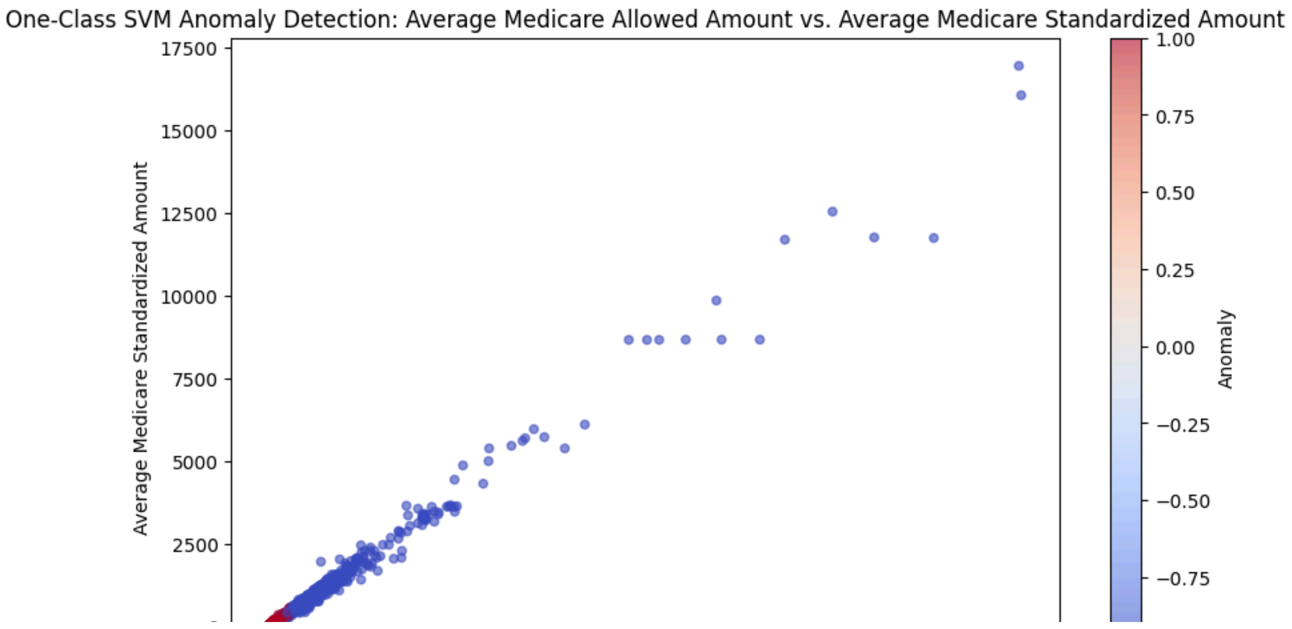
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but IsolationForest
  warnings.warn(
```

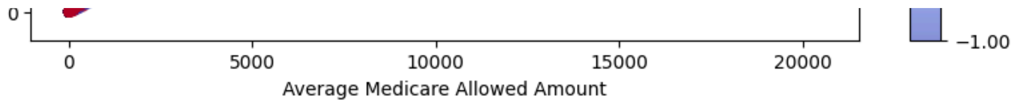


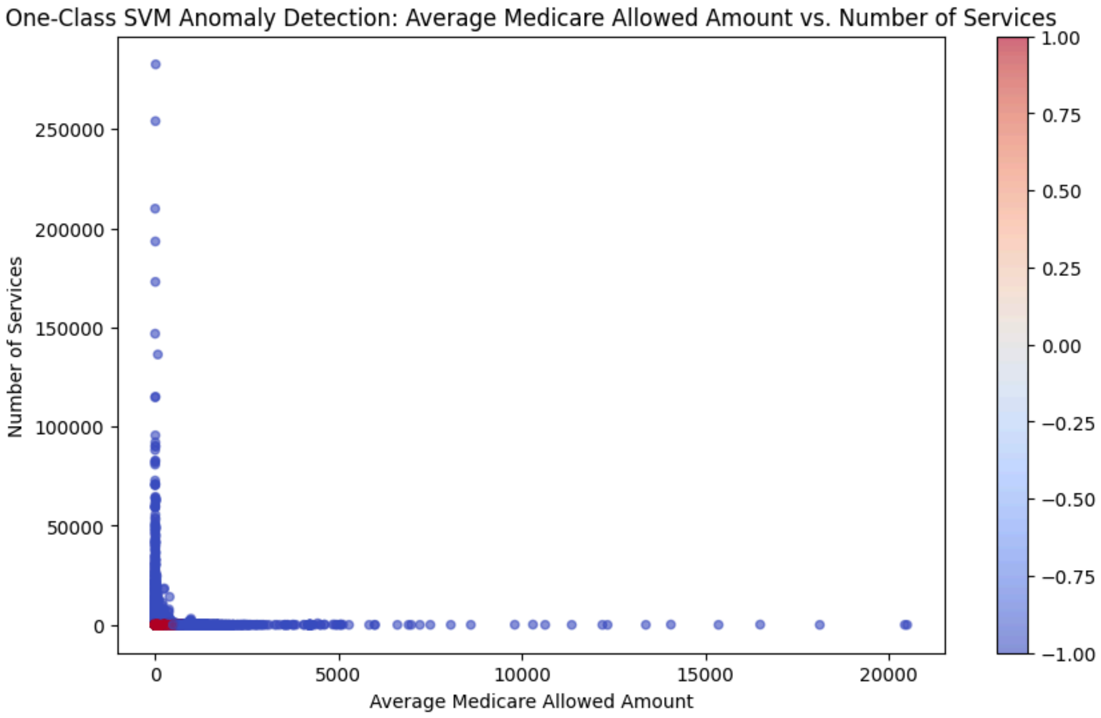
```
import pandas as pd
import matplotlib.pyplot as plt
# Load the data (replace 'Healthcare Providers.csv' with your actual file path)
df = pd.read_csv('/content/Healthcare Providers.csv', encoding='ascii')

# Select the numerical columns
numerical_columns = ['Average Medicare Allowed Amount', 'Average Submitted Charge Amount',
                     'Average Medicare Payment Amount', 'Average Medicare Standardized Amount',
                     'Number of Services', 'Number of Medicare Beneficiaries',
                     'Number of Distinct Medicare Beneficiary/Per Day Services']
data = df[numerical_columns]

# Remove commas and convert to numeric
data = data.replace({',': ''}, regex=True)
data = data.apply(pd.to_numeric, errors='coerce')

# Handle any missing values by dropping them
data = data.dropna()

# Fit the One-Class SVM model
svm_model = OneClassSVM(nu=0.1)  # Adjust the hyperparameter 'nu' as needed
svm_model.fit(data)

# Predict anomalies (1 for normal, -1 for anomalies)
anomalies = svm_model.predict(data)

# Create a new DataFrame from the preprocessed data and add the anomaly column
df_anomalies = pd.DataFrame(data)
df_anomalies['anomaly'] = anomalies

# Create scatter plots for each pair of numerical columns
for i in range(len(numerical_columns)):
    for j in range(i + 1, len(numerical_columns)):
        plt.figure(figsize=(10, 6))
        plt.scatter(df_anomalies[numerical_columns[i]], df_anomalies[numerical_columns[j]],
                    c=df_anomalies['anomaly'], cmap='coolwarm', s=20, alpha=0.6)
        plt.xlabel(numerical_columns[i])
        plt.ylabel(numerical_columns[j])
        plt.title('One-Class SVM Anomaly Detection: {} vs. {}'.format(numerical_columns[i], numerical_columns[j]))
        plt.colorbar(label='Anomaly')
        plt.show()
```

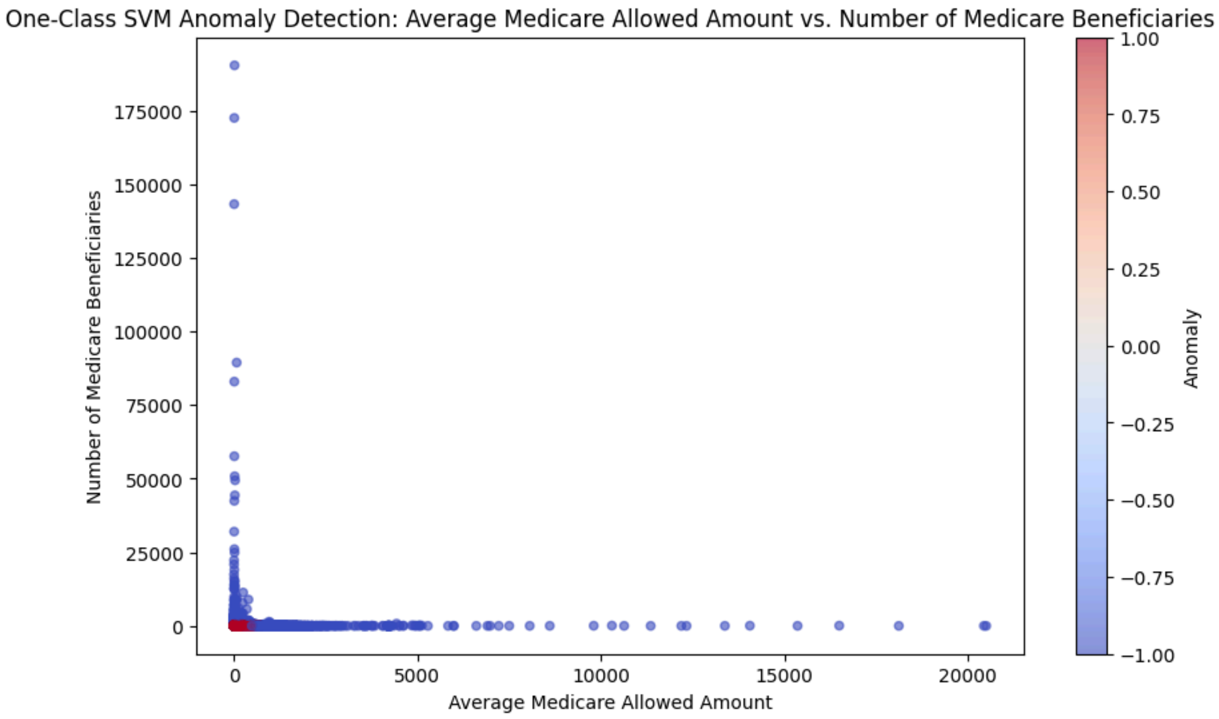

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Average Submitted Charge Amount
Average Submitted Charge Amount
Average Medicare Allowed Amount
Anomaly
60000
50000
40000
30000
20000
10000
0
0
5000
10000
15000
20000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

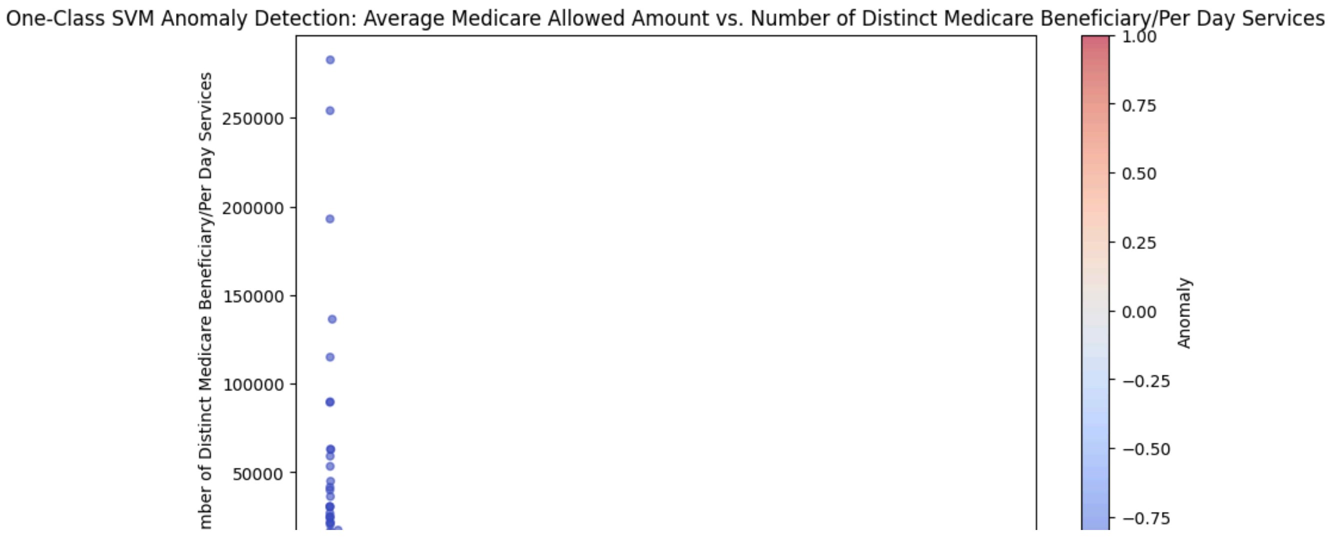

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Average Medicare Payment Amount
Average Medicare Payment Amount
Average Medicare Allowed Amount
Anomaly
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

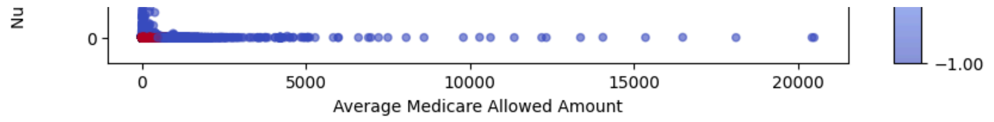

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Average Medicare Standardized Amount
Average Medicare Standardized Amount
Anomaly
17500
15000
12500
10000
7500
5000
2500
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

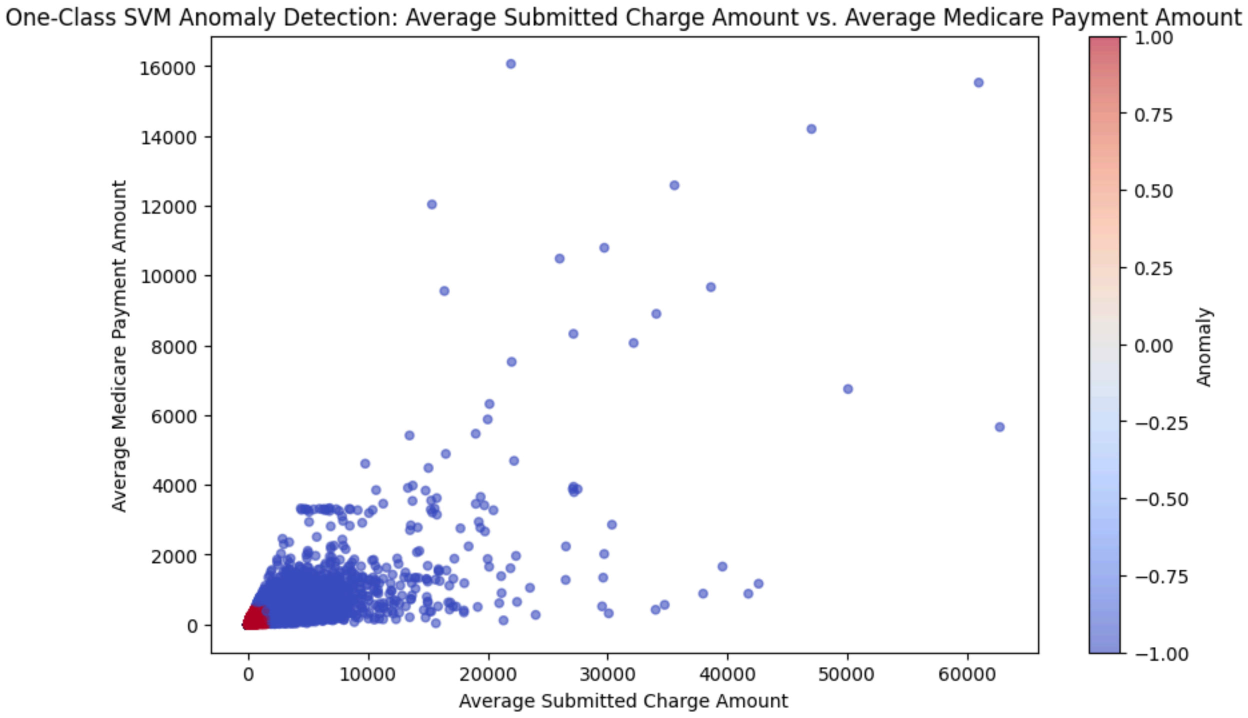

0
0
5000
10000
15000
20000
Average Medicare Allowed Amount
−1.00

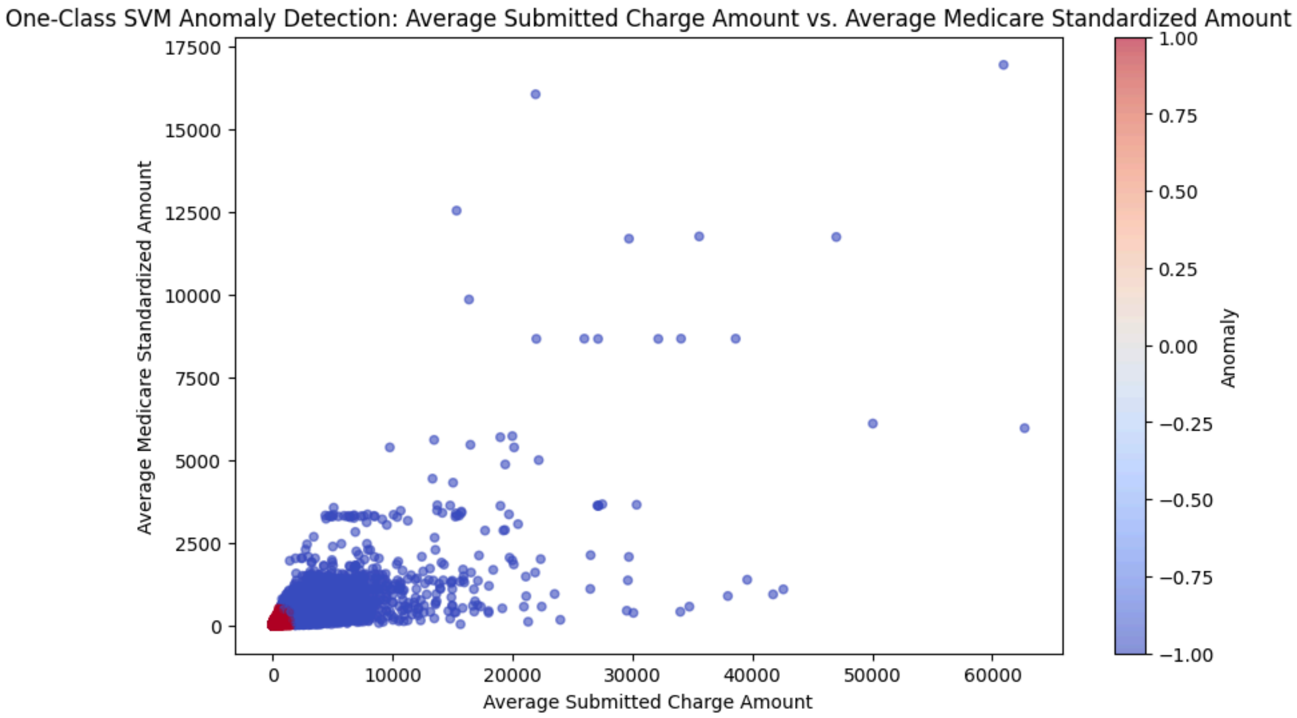

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Number of Services
Number of Services
250000
200000
150000
100000
50000
0
0
5000
10000
15000
20000
Average Medicare Allowed Amount
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

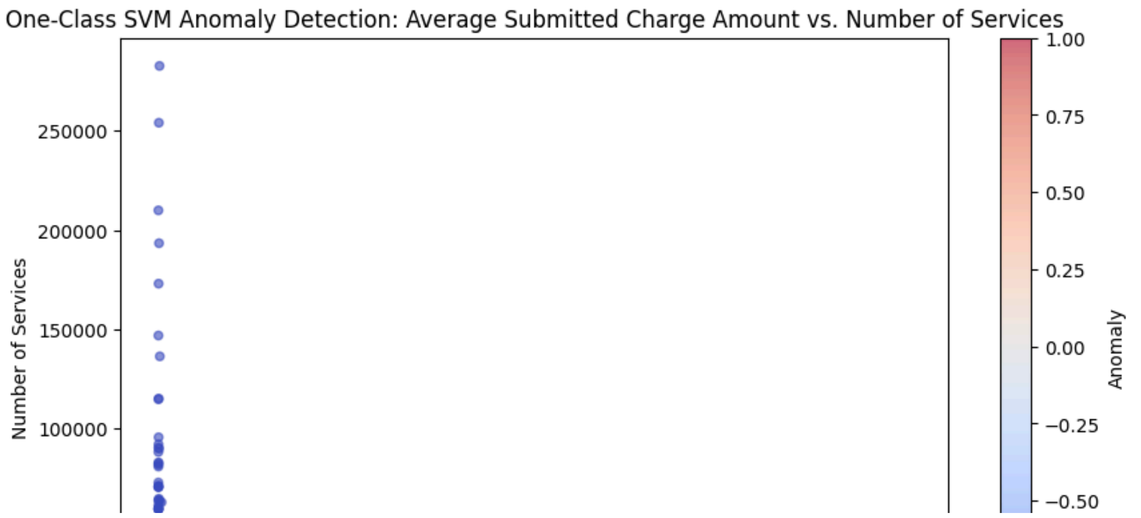

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Number of Medicare Beneficiaries
Number of Medicare Beneficiaries
175000
150000
125000
100000
75000
50000
25000
0
0
5000
10000
15000
20000
Average Medicare Allowed Amount
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

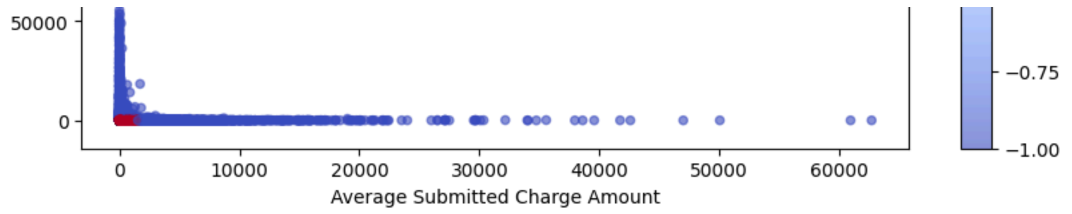

One-Class SVM Anomaly Detection: Average Medicare Allowed Amount vs. Number of Distinct Medicare Beneficiary/Per Day Services
mber of Distinct Medicare Beneficiary/Per Day Services
250000
200000
150000
100000
50000
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75


Nu
0
0
5000
10000
15000
20000
Average Medicare Allowed Amount
−1.00

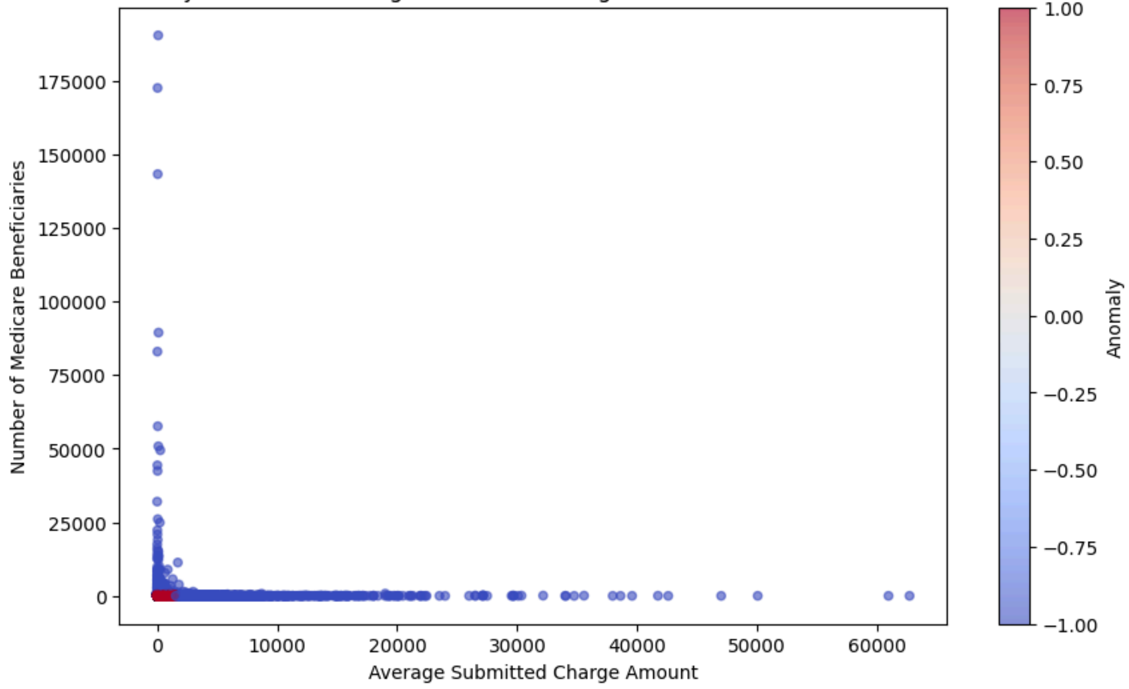

One-Class SVM Anomaly Detection: Average Submitted Charge Amount vs. Average Medicare Payment Amount
Average Medicare Payment Amount
Average Submitted Charge Amount
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


One-Class SVM Anomaly Detection: Average Submitted Charge Amount vs. Average Medicare Standardized Amount
Average Medicare Standardized Amount
Average Submitted Charge Amount
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

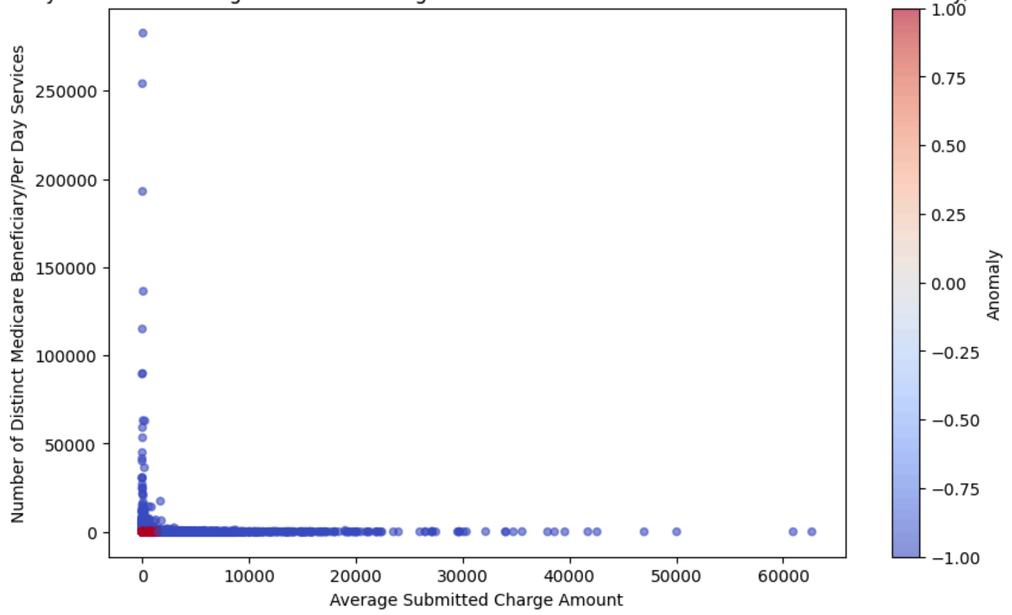

One-Class SVM Anomaly Detection: Average Submitted Charge Amount vs. Number of Services
Number of Services
250000
200000
150000
100000
Anomaly
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50



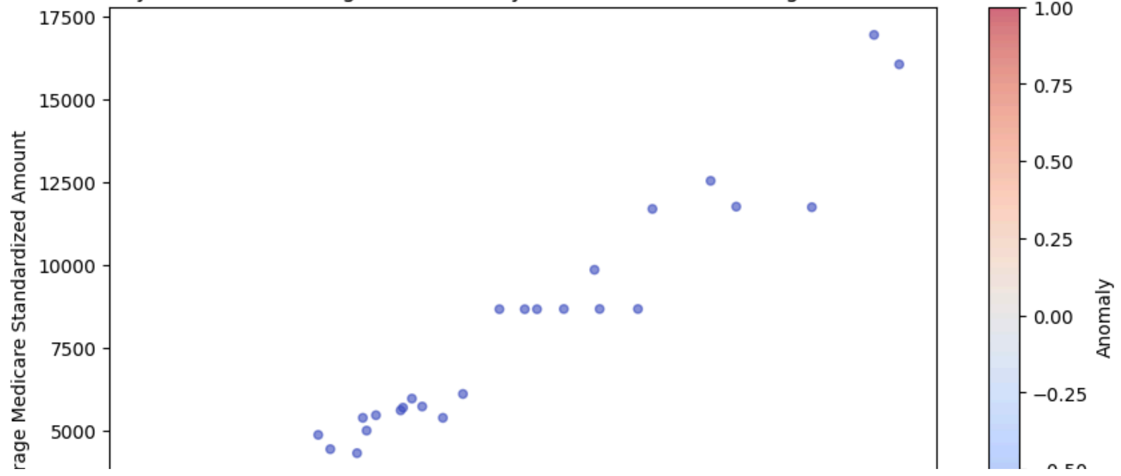
One-Class SVM Anomaly Detection: Average Submitted Charge Amount vs. Number of Medicare Beneficiaries

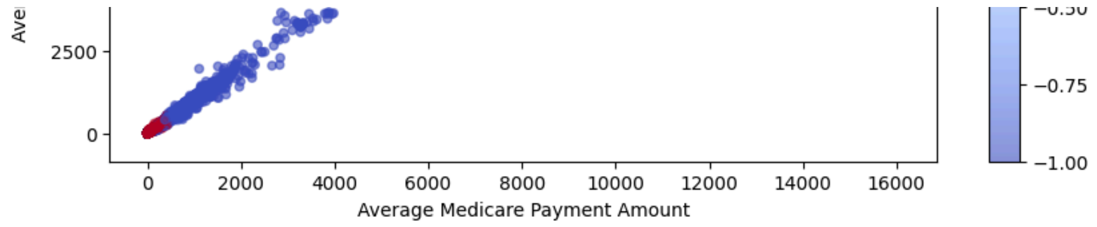


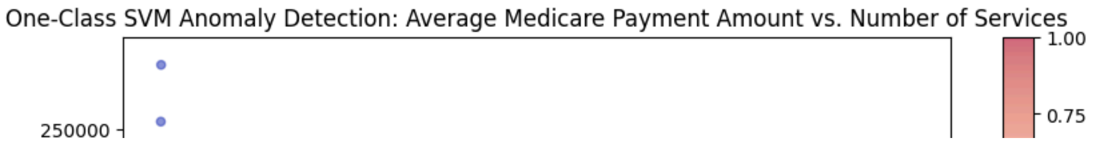
One-Class SVM Anomaly Detection: Average Submitted Charge Amount vs. Number of Distinct Medicare Beneficiary/Per Day Services



One-Class SVM Anomaly Detection: Average Medicare Payment Amount vs. Average Medicare Standardized Amount




Ave
2500
0
0
2000
4000
6000
8000
10000
12000
14000
16000
Average Medicare Payment Amount
−0.50
−0.75
−1.00


One-Class SVM Anomaly Detection: Average Medicare Payment Amount vs. Number of Services
250000
1.00
0.75