Outlier Detection on Titanic Dataset

Overview

Outlier detection is a crucial step in data preprocessing, especially in datasets like the Titanic dataset, where outliers can significantly impact the performance of machine learning models. This analysis focuses on identifying outliers using various methods, including the Interquartile Range (IQR) method, Z-Score method, Isolation Forest, Local Outlier Factor (LOF), Elliptic Envelope, and DBSCAN. Each method has its strengths and is suitable for different types of data distributions.

Why Perform Outlier Detection?

- **Improve Model Performance**: Outliers can skew the results of statistical analyses and machine learning models.
- **Data Quality**: Identifying and handling outliers can improve the overall quality of the dataset.
- **Insight Generation**: Outliers can sometimes provide valuable insights into anomalies or rare events in the data.

Prerequisites

- **Python 3.x**
- **Pandas**
- **NumPy**
- **Seaborn**
- **Matplotlib**
- **Scipy**
- **Scikit-learn**

Files Included

1. **Outlier Detection Code**: The main script for performing outlier detection on the Titanic dataset.

2. **Visualizations**: Various plots including boxplots, scatterplots, density plots, and more to visualize outliers.
3. **Outlier Detection Methods**: Implementation of IQR, Z-Score, Isolation Forest, Local Outlier Factor, Elliptic Envelope, and DBSCAN.

Code Description

1. Data Loading and Preprocessing

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.cluster import DBSCAN
from sklearn.covariance import EllipticEnvelope

# Load the dataset
df = pd.read_csv("Titanic-Dataset.csv")

# Display basic information
df.info()
df.head()

# Handling missing values
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)
df.drop(columns=['PassengerId', 'Name', 'Ticket', 'Cabin'], inplace=True)

# Convert categorical columns to numerical
df['Sex'] = df['Sex'].map({'male': 0, 'female': 1})
df['Embarked'] = df['Embarked'].map({'C': 0, 'Q': 1, 'S': 2})

# Display cleaned data info
df.info()
```

2. Interquartile Range (IQR) Method

```
# IQR Method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Outliers detected using IQR
outliers_iqr = (df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))

# Visualizing Outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['Age', 'Fare', 'SibSp', 'Parch']])
plt.title("Boxplot for Detecting Outliers using IQR Method")
plt.show()
```

3. Isolation Forest

```
# Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
df['IsoForest_Outlier'] = iso_forest.fit_predict(df[['Age', 'Fare']])

# Visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['Age'], y=df['Fare'], hue=df['IsoForest_Outlier'], p
plt.title("Outlier Detection using Isolation Forest")
plt.show()
```

4. Z-Score Method

```
# Z-Score Method
z_scores = np.abs(stats.zscore(df.select_dtypes(include=[np.number])))
outliers = np.where(z_scores > 3)

# Visualizing Outliers using Boxplots
plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['Age', 'Fare', 'SibSp', 'Parch']])
plt.title("Boxplot for Detecting Outliers using Z-Score")
plt.show()
```

5. Local Outlier Factor (LOF)

```
# Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20)
df['LOF_Outlier'] = lof.fit_predict(df[['Age', 'Fare']])

# Visualization
plt.figure(figsize=(8, 6))
sns.kdeplot(data=df, x='Age', y='Fare', hue='LOF_Outlier', fill=True, pal
plt.title("Density Plot for Outliers using Local Outlier Factor")
plt.show()
```

6. Elliptic Envelope

```
# Elliptic Envelope
ell_env = EllipticEnvelope(contamination=0.05)
df['Elliptic_Outlier'] = ell_env.fit_predict(df[['Age', 'Fare']])

# Visualization
plt.figure(figsize=(8, 6))
sns.kdeplot(x=df['Age'], y=df['Fare'], hue=df['Elliptic_Outlier'], fill=T
plt.title("Elliptic Envelope Outlier Detection")
plt.show()
```

7. DBSCAN

```
# DBSCAN
dbscan = DBSCAN(eps=1.5, min_samples=5)
df['DBSCAN_Outlier'] = dbscan.fit_predict(df[['Age', 'Fare']])

# Visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['Age'], y=df['Fare'], hue=df['DBSCAN_Outlier'], pale
plt.title("DBSCAN-based Outlier Detection")
plt.show()
```

Expected Outputs

1. **Visualizations**: Boxplots, scatterplots, density plots, and more to visualize outliers detected by different methods.

2. **Outlier Labels**: Columns added to the DataFrame indicating whether each data point is an outlier according to each method.

Use Cases

- **Data Cleaning**: Identify and handle outliers to improve data quality.
- **Anomaly Detection**: Detect anomalies in the data that may indicate rare events or errors.
- **Model Preparation**: Ensure that the data is clean and ready for machine learning models.

Advantages

- **Multiple Methods**: Provides a comprehensive approach to outlier detection using various techniques.
- **Visual Insights**: Visualizations help in understanding the distribution and identifying outliers.
- **Flexibility**: Different methods can be applied depending on the nature of the data and the specific requirements of the analysis.

Future Enhancements

- **Automated Outlier Detection**: Use automated tools or libraries for outlier detection to streamline the process.
- **Advanced Visualizations**: Incorporate more advanced visualizations like 3D plots or interactive plots.
- **Feature Engineering**: Use insights from outlier detection to create new features or transform existing ones.
- **Modeling**: Use the cleaned data to build predictive models and evaluate the impact of outlier detection on model performance.

References

- Pandas Documentation
- Seaborn Documentation
- Matplotlib Documentation
- Scipy Documentation
- Scikit-learn Documentation
- Titanic Dataset on Kaggle

This README provides a comprehensive overview of the outlier detection performed on the Titanic dataset, including the purpose, implementation, and potential applications. It also includes instructions for running the code and interpreting the results.