Source Code

1. Importing the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, RidgeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from google.colab import drive
drive.mount('/content/drive')
df=pd.read_csv(r"/content/drive/MyDrive/medical_aid_claims.csv")
df
df.columns = df.columns.str.strip()
df.info()
print(df.columns.tolist())
df["gender"].unique()
df["location"].unique()
df["employer"].unique()
df["cause"].unique()
df["relationship"].unique()
df = df.drop(['member-name', 'email', 'location', 'employer', 'patient_name', 'patient_suffix']
, axis=1)
df
```

2. Convert patient_dob to age

```
import datetime
current_year = datetime.datetime.now().year # Use datetime.datetime.now()
df['age'] = pd.to_datetime(df['patient_dob'], format='%m/%d/%Y').apply(lambda x:
current_year - x.year)
```

```
df = df.drop('patient_dob', axis=1)
df
df.info()
df.isnull().sum()
df.describe(include='all')
```

3. Converting categorical (non-numeric) columns in a DataFrame into numerical form

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
# List of columns that are of type 'object' (categorical features)
object_cols = df.select_dtypes(include=['object']).columns
# Convert all object-type columns to string (to handle mixed types)
for col in object_cols:
df[col] = df[col].astype(str)  # Ensure all data in the column is treated as string
# Apply label encoding to each object-type column
for col in object_cols:
df[col] = le.fit_transform(df[col])
# Check the result (first few rows)
print(df.head())
df.corr()
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True)
plt.title('Heatmap of Correlations',fontsize=15)
plt.show()
df["label"].value_counts()
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = df[df['label']== 0]
df_minority = df[df['label']== 1]
# Downsample majority class and upsample the minority class
df_minority_upsampled = resample(df_minority,
replace=True,n_samples=10000,random_state=100)
```

```
df_majority_downsampled = resample(df_majority,
replace=True,n_samples=10000,random_state=100)
# Combine minority class with downsampled majority class
df_balanced = pd.concat([df_minority_upsampled,df_majority_downsampled])
# Display new class counts
df_balanced['label'].value_counts()
```

4**. Balance an imbalanced dataset**

```
from sklearn.utils import resample
```

5. Data splitting and feature scaling

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=100)
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
# Save the scaler to a pickle file
with open('scaler.pkl', 'wb') as f:
 pickle.dump(scaler, f)
```

6.Performing CatBoost Algorithm

```
import pandas as pd
from catboost import CatBoostClassifier # This line should now work without error
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import pickle
# Create a CatBoost Classifier model
catboost_model = CatBoostClassifier(verbose=0, random_state=42)

# Train the CatBoost model using the selected features
catboost_model.fit(x_train, y_train)
# Saving the model
```

```
filename = r'catboost_model.pkl'
pickle.dump(catboost_model, open(filename, 'wb'))
# Make predictions on the test set
y_pred = catboost_model.predict(x_test)
# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
# Define class names (replace with your actual class names if needed)
class_names = ['No', 'Yes']
# Display confusion matrix with correct display_labels
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=class_names)
cm_display.plot(cmap=plt.cm.Blues)
plt.title('CatBoost Confusion Matrix')
plt.show()

# calculate and print accuracy
accuracy = catboost_model.score(x_test, y_test)
print(f"CatBoost Model Accuracy: {accuracy:.2f}")

import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import pickle



# Create an Isolation Forest model
iso_forest_model = IsolationForest(random_state=42, contamination='auto')

# Train the Isolation Forest model using the selected features
iso_forest_model.fit(x_train)

# Saving the model
filename = r'iso_forest_model.pkl'
```

```
pickle.dump(iso_forest_model, open(filename, 'wb'))

# Make predictions on the test set
# Isolation Forest labels anomalies as -1 and normal points as 1, so map accordingly
y_pred = iso_forest_model.predict(x_test)
y_pred = [0 if pred == -1 else 1 for pred in y_pred]

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Define class names (replace with your actual class names if needed)
class_names = ['No', 'Yes']

# Display confusion matrix with correct display_labels
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=class_names)
cm_display.plot(cmap=plt.cm.Blues)
plt.title('Isolation Forest Confusion Matrix')
plt.show()

# Calculate and print accuracy
accuracy = (y_test == y_pred).mean()
print(f"Isolation Forest Model Accuracy: {accuracy:.2f}")
```

7.Classification Report

```
from sklearn.metrics import accuracy_score, classification_report
class_names = ['No','Yes']
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=class_names))
from sklearn.metrics import accuracy_score, classification_report
class_names = ['No','Yes']
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=class_names))
```

8. Show actual vs predicted values

```
actual_vs_predicted = pd.DataFrame({
'Actual': ['No' if val == 0 else 'Yes' for val in y_test[:10].values],
'Predicted': ['No' if val == 0 else 'Yes' for val in y_pred[:10]]
})
print(actual_vs_predicted)
import pandas as pd
# Show actual vs predicted values for 10 samples with 'No' and 'Yes' labels
actual_vs_predicted = pd.DataFrame({
    'Actual': ['No' if val == 0 else 'Yes' for val in y_test[:10].values],
    'Predicted': ['No' if val == 0 else 'Yes' for val in y_pred[:10]]
})
print(actual_vs_predicted)
```