CPCS433 Artificial Intelligence Topics
Assignment 1

Shahad Hatem K. Baalkhashir
ID: 2006676

May 8, 2024

1 Problem: Linear Regression

1.1 Load the dataset file.

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
from google.colab import drive
import math
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import train_test_split,
    GridSearchCV
from sklearn import metrics
drive.mount('/content/drive')
!ls -l "/content/drive/MyDrive/VehiclesPr1/vehicles.csv"
file_path = "/content/drive/MyDrive/VehiclesPr1/vehicles.csv"
data = pd.read_csv(file_path)
```

1.2 Preprocess the dataset by handling missing values, outliers, and categorical variables.

```
data.head()  # View the first few rows
data.info()  # Get information about data types and missing
    values
data.describe()  # Get summary statistics for numerical columns

fig, ax = plt.subplots(figsize=(10,5))
ax.set_title('Distribution of the prices')
sns.distplot(data['price'], bins=30, kde=False)
```

1.2.1 Graphs to detect any null values, outlier, etc.

```
# To detect any outlier, we are going to use plots like boxplot
    and scatter plot to help us visualize the data better
fig, axs = plt.subplots(1, 2, figsize=(15, 6))

# Boxplot for 'odometer'
axs[0].boxplot(data['odometer'])
axs[0].set_title('Boxplot of Odometer')
axs[0].set_ylabel('Odometer')

# Boxplot for 'price'
axs[1].boxplot(data['price'])
axs[1].set_title('Boxplot of Price')
axs[1].set_ylabel('Price')

plt.show()

plt.figure(figsize=(10, 6))
plt.scatter(data['odometer'], data['price'])
plt.xlabel('Odometer')
plt.ylabel('Price')
plt.title('Scatter Plot of Odometer vs Price')
plt.grid(True)
plt.show()
```

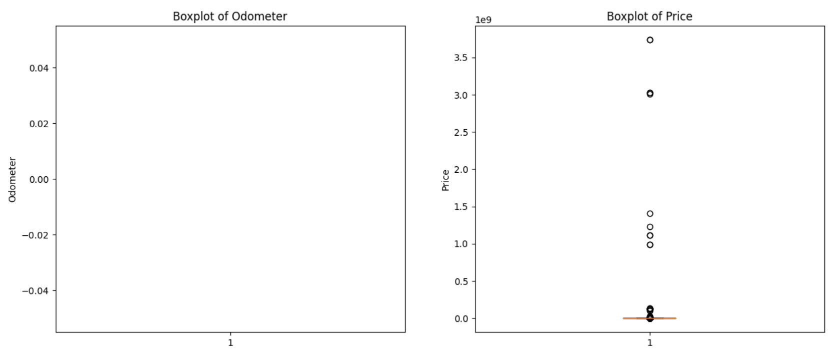


Figure 1: Box-plot for outliers

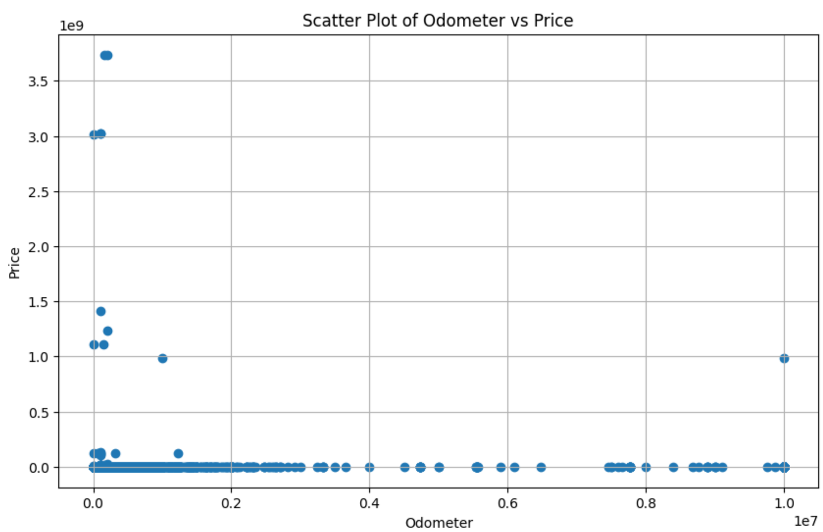


Figure 2: Scatter plot for outliers

```
# Removing outliers for both 'odometer' and 'price' columns,
    also using the function to handle missing values.
# Remove outliers for 'odometer'
Q1 = data['odometer'].quantile(0.25)
Q3 = data['odometer'].quantile(0.75)
IQR = Q3 - Q1
filter_odometer = (data['odometer'] <= Q3 + 3 * IQR)
data = data.loc[filter_odometer]

# Remove outliers for 'price'
```

```
Q1 = data['price'].quantile(0.25)
Q3 = data['price'].quantile(0.75)
IQR = Q3 - Q1
filter_price = (data['price'] >= Q1 - 1.5 * IQR) & (data['price'] <= Q3 + 1.5 * IQR)
data = data.loc[filter_price]

# Calculate and print the number of outliers removed for each column
outliers_removed_odometer = filter_odometer.sum()
outliers_removed_price = filter_price.sum()
print(outliers_removed_odometer, '(', '{:.2f}'.format(100 * outliers_removed_odometer / len(data)), '%', ')', 'outliers removed from "odometer"')
print(outliers_removed_price, '(', '{:.2f}'.format(100 * outliers_removed_price / len(data)), '%', ')', 'outliers removed from "price"')
```

Figure 3: Output of removing outliers

1.2.2 Drop all the columns that caused me troubles and I found them unnecessary, also dealing with categorical data

```
data = data.drop(columns = [ 'id', 'region','url', 'region_url', 'VIN', 'paint_color',
        'image_url', 'county', 'lat', 'long', 'posting_date', 'model', 'description', 'manufacturer', 'size'])
        rm_rows = ['year', 'fuel', 'transmission', 'drive', 'type']
for column in rm_rows:
    data = data[~data[column].isnull()]
    data = data.replace(np.nan, 'null', regex=True)  # Replace missing values with 'null' for better handling later on

    # Dealing With Categorical Variables, creating dummie
col = ['condition', 'cylinders', 'fuel', 'transmission', 'drive', 'type', 'title_status', 'state']
data_new = pd.concat([data]+[pd.get_dummies(data[i],drop_first=True,prefix=i+'_') for i in col],axis=1)
```

```
data_new.drop(['condition', 'cylinders', 'fuel', 'transmission',
     'drive', 'type', 'title_status', 'state'],axis=1,inplace=
    True)
data_new.head()
```

	price	year	odometer	condition__fair	condition__good	condition__like new	condition__new	condition__null	condition__salvage	cylinders__12 cylinders	...	state__sd	state__tn	state__tx	state__ut	state__va	state__vt
31	15000	2013.0	128000.0	False	False	False	False	False	False	False	...	False	False	False	False	False	False
32	27990	2012.0	68696.0	False	True	False	False	False	False	False	...	False	False	False	False	False	False
33	34590	2016.0	29499.0	False	True	False	False	False	False	False	...	False	False	False	False	False	False
34	35000	2019.0	43000.0	False	False	False	False	False	False	False	...	False	False	False	False	False	False
35	29990	2016.0	17302.0	False	True	False	False	False	False	False	...	False	False	False	False	False	False

5 rows × 93 columns

Figure 4: Example of the data after

1.2.3 Scaling out numerical data

```
# Scale out the the numerical data except year, because I tried
    before standrizing year and got really bad results.

std_scaler = StandardScaler()

for column in ['price', 'odometer']:
    data_new[column] = std_scaler.fit_transform(data_new[column
        ].values.reshape(-1,1))
```

1.3 Divide the dataset into training and testing (use splitting ratio 80:20).

```
# Select all columns except 'price' for X
X = data_new.drop('price', axis=1)

# Select the 'price' column for y and reshape it
y = data_new['price'].values.reshape(-1, 1)

X = X.to_numpy()
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=141)

X_train shape: (182260, 92)
y_train shape: (182260, 1)
```

1.4 Train a linear regression model using a gradient descent MSE cost-function, lr=0.1, and stopping condition number of iterations at least equals to 10.

```
def cost(X, y, w):
    """Calculates the cost function efficiently.

    Args:
        X: Training data (n_samples, n_features).
        y: Target labels (n_samples,).
        w: Model weights (n_features, 1).

    Returns:
        The cost value.
    """
    prediction = X @ w
    return (1 / (2 * X.shape[0])) * np.sum((prediction - y) **
       2)  # Vectorized sum

def permutation_importance(X, y, model, cost_function,
   n_features):
    importance = np.zeros(n_features)
    for i in range(n_features):
        X_shuffled = X.copy()
        X_shuffled[:, i] = np.random.permutation(X_shuffled[:, i
           ])  # Shuffle feature values
        original_cost = cost_function(X, y, model)
        shuffled_cost = cost_function(X_shuffled, y, model)
        importance[i] = original_cost - shuffled_cost  #
           Importance based on cost change

    # Get the indices of the most significant features
    sorted_indices = np.argsort(importance)[::-1]

    return sorted_indices

n = X_train.shape[1]
theta = np.ones((n, 1))

# Precompute X_train transpose for efficient matrix
   multiplication
XTX = np.transpose(X_train) @ X_train
XTy = np.transpose(X_train) @ y_train
```

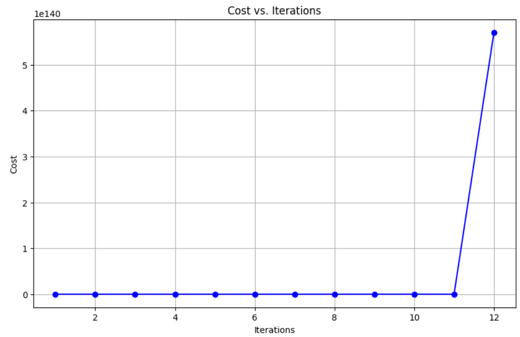


Figure 5: Enter Caption

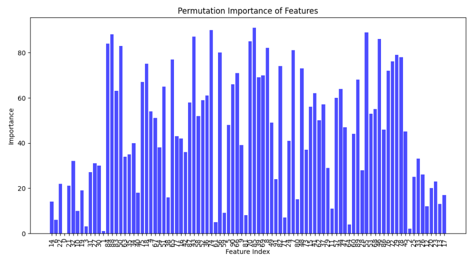
```
learning_rate = 0.1
num_iterations = 12
cost_history = np.zeros(num_iterations)

for i in range(num_iterations):
    prediction = X_train @ theta
    gradient = (1 / X_train.shape[0]) * XTX @ theta - XTy  #
        gradient calculation
    theta = theta - learning_rate * gradient
    cost_history[i] = cost(X_train, y_train, theta)  # Calculate
         cost after update theta = learning_rate - gradient

# Check for convergence
tolerance = 1e-5
if num_iterations >= 10 and np.all(np.abs(cost_history[-1] -
    cost_history[-5:]) < tolerance):
    print("Converged after", i + 1, "iterations.")
```

1.5 Test the performance of the model by computing and reporting the MSE on the testset.

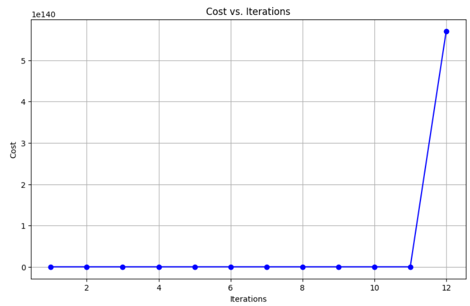
1.5.1 Keep a record of the cost at each iteration and plot the cost at each iteration against iterations (epochs).

1.5.2 Find and report the most significant features in the regression model.

```python
# Reshape theta and flatten cost_history for plotting
theta = theta.reshape(-1, 1)
print("The history of the cost: " ,cost_history)
cost_history = cost_history.flatten()

# Plot the cost vs iterations
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_iterations + 1), cost_history, marker='o',
     color='b')
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost vs. Iterations')
plt.grid(True)
plt.show()

# Assuming you have trained your model and have X_train and
    y_train
n_features = X_train.shape[1]
most_significant_features = permutation_importance(X_train,
    y_train, theta, cost, n_features)
print("Most significant features:", most_significant_features)

y_pred = X_test @ theta # Compute predictions on the test set

mse_test = np.mean((y_pred - y_test) ** 2) # Calculate MSE on
    the test set

print(f"MSE on test set: {mse_test}")

# Calculate feature importances using permutation importance
feature_importances = permutation_importance(X_train, y_train,
    theta, cost, n_features)

# Display a bar chart for the most important features
plt.figure(figsize=(12, 6))
plt.bar(range(n_features), feature_importances, color='b', alpha
    =0.7)
plt.xticks(range(n_features), most_significant_features,
    rotation=90)
plt.xlabel('Feature Index')
plt.ylabel('Importance')
plt.title('Permutation Importance of Features')
plt.show()
```



(a) Most significant features bar chart



(b) Cost plot

Figure 6: Comparison of two plots

```
The history of the cost:  [2.53637272e+017 4.15004344e+028 6.79034594e+039 1.11104374e+051
 1.81790179e+062 2.97447056e+073 4.86686088e+084 7.96321039e+095
 1.30294909e+107 2.13189939e+118 3.48823684e+129 5.70749082e+140]
```

Figure 7: History List

```
MSE on test set: 1.1415141361316664e+141
```

Figure 8: MSE on Test Data

```
Most significant features: [14  6 22  0 21 32 10 19  3 31 27 30  1 84 88 83 50 63 35 34 40 75 18  4
 67 54 51 38 65 77 16 42 87 43 58 52 36 74 61 56 59  5 90 66  9 81 70 85
 39 69  8 49 91 82 71 24  7 80 48 73 15 41 62 37 29 11 57 44 47 64 60 89
 28 55 53 68 46 86 76 72 79 78 45  2 25 33 26 12 20 23 13 17]
```

Figure 9: Most significant features

2 Problem: SVM Classifier

2.1 Load the dataset file

```python
data = pd.read_csv('/content/WA_Fn-UseC_-Telco-Customer-Churn.
    csv')
```

2.2 Preprocess the dataset by handling missing values, outliers, and categorical variables

```python
data.drop(["customerID", "gender"], axis=1, inplace=True)
data.head()
null_counts = data.isna().sum()
print("The Number of Missing Values in the Dataset (Nan)\n")
print(null_counts)
data['TotalCharges'] = pd.to_numeric(data.TotalCharges, errors='
    coerce')
data.isnull().sum()
```

```
The Number of Missing Values in the Dataset (Nan)

SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

(a) Missing values

```
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
StreamingTV          0
StreamingMovies      0
Contract             0
PaperlessBilling     0
PaymentMethod        0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

(b) Null values

```python
# Handle missing values add mean isntead of null
data[np.isnan(data['TotalCharges'])]
data['TotalCharges'] = data['TotalCharges'].replace(' ', np.nan)
data['TotalCharges'] = data['TotalCharges'].astype(float)
data.fillna(data["TotalCharges"].mean())
# if tenture is zero drop
```

```python
data.drop(labels=data[data['tenure'] == 0].index, axis=0,
    inplace=True)
data[data['tenure'] == 0].index

# From the plots, as we can see there are no Outliers to handle.

# Handling Categorical Variables
def encode_categorical_variables(df):
    for column in df.select_dtypes(include='object'):
        df[column] = pd.Categorical(df[column]).codes

# Apply the functions
encode_categorical_variables(data)

# Print the preprocessed data
data.head()

import plotly.express as px
fig = px.histogram(data, x="Churn", color="Contract", barmode="
    group", title="<b>Customer contract distribution<b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

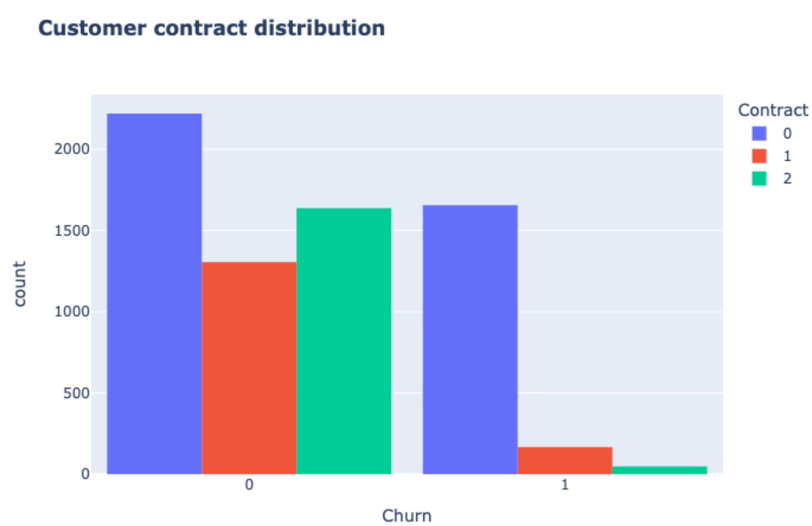


Figure 11: Customer contract distribution

2.3 Divide the dataset randomly into testing and training with a splitting ration of (70:30).

```
from sklearn.model_selection import train_test_split

# Separate the features and target
features = data.drop('Churn', axis=1)
target = data['Churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features,
    target, test_size=0.3, random_state=42)

# Divide the columns into 3 categories, one for standardisation,
     one for label encoding and one for one hot encoding
num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
cat_cols_ohe =['PaymentMethod', 'Contract', 'InternetService'] #
     those that need one-hot encoding
cat_cols_le = list(set(X_train.columns)- set(num_cols) - set(
    cat_cols_ohe)) #those that need label encoding

from sklearn.preprocessing import StandardScaler
# Scale numerical features
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

```
Shape of X_train: (4922, 18)
Shape of X_test: (2110, 18)
Shape of y_train: (4922,)
Shape of y_test: (2110,)
```

2.4 Train the algorithms.

```
from sklearn.svm import SVC
svc_model = SVC(random_state = 1)
svc_model.fit(X_train,y_train)
```

```
from sklearn.metrics import classification_report
# Predict the labels of the test data
predictions = svc_model.predict(X_test)
print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.82	0.92	0.87	1549
1	0.68	0.46	0.55	561
accuracy			0.80	2110
macro avg	0.75	0.69	0.71	2110
weighted avg	0.79	0.80	0.78	2110

Figure 12: classification report SVM Before gs

2.5 Use grid search to find the best hyperparameters for the SVM model.

```
from sklearn.model_selection import GridSearchCV
# Hyperparameter tuning with GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['linear']}

grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train, y_train)
# Get and Print the best hyperparameters
best_params = grid.best_params_
print("Best hyperparameters:", best_params)

''' Best hyperparameters: {'C': 0.1, 'gamma': 1, 'kernel': '
   linear'} '''
grid_predictions = grid.predict(X_test)
# print classification report
print(classification_report(y_test, grid_predictions))
```

	precision	recall	f1-score	support
0	0.83	0.89	0.86	1549
1	0.62	0.52	0.56	561
accuracy			0.79	2110
macro avg	0.73	0.70	0.71	2110
weighted avg	0.78	0.79	0.78	2110

Figure 13: classification report SVM After gs

2.6 Plot the two classes in different colors, along with the decision boundary for each classifier.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
# Transforming the dataset using PCA
X_pca = pca.fit_transform(X_train)
y = y_train
X_pca.shape, y.shape
# min and max values
xmin, xmax = X_pca[:, 0].min() - 2, X_pca[:, 0].max() + 2
ymin, ymax = X_pca[:, 1].min() - 2, X_pca[:, 1].max() + 2

# Creating a mesh region where the boundary will be plotted
xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.2),
                     np.arange(ymin, ymax, 0.2))

# Fitting SVM model on 2 features
grid.fit(X_pca, y)
# Plotting decision boundary for SVM
z2 = grid.predict(np.c_[xx.ravel(), yy.ravel()])
z2 = z2.reshape(xx.shape)

# Displaying the result
plt.contour(xx, yy, z2, alpha=0.4, colors='purple') # SVM
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y_train, s=50,
    alpha=0.8)
plt.title('SVM')
```

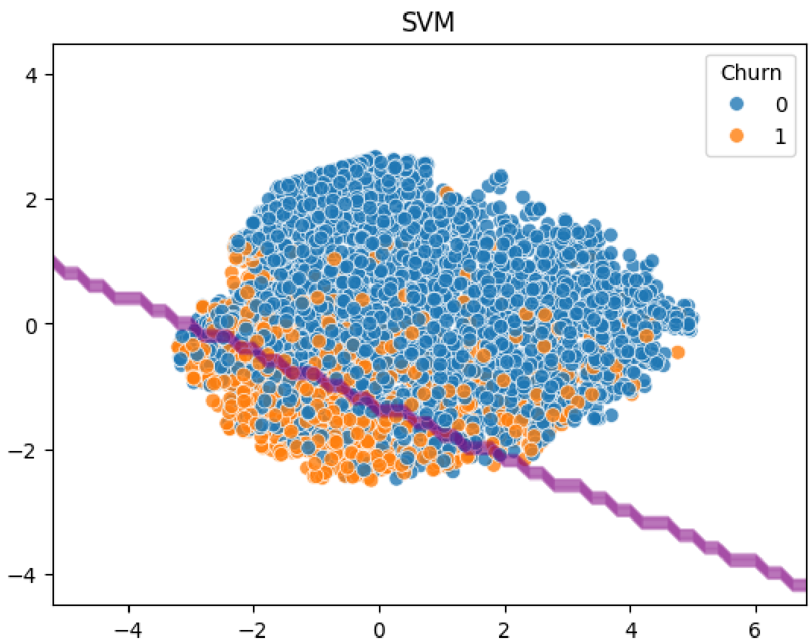
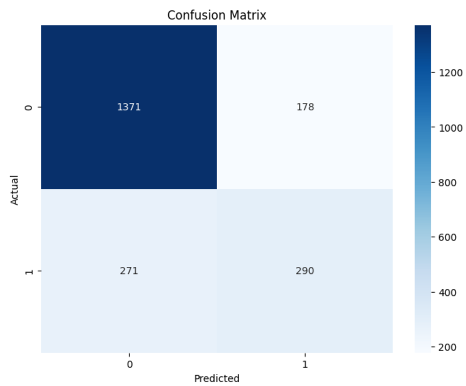


Figure 14: Churn class and Decision boundary

2.7 Report the results of the classifier and evaluate the performance of the classifier

```
grid_predictions = grid.predict(X_test)
# print classification report
print(classification_report(y_test, grid_predictions))


# Print the accuracy score
print("\nAccuracy Score:", accuracy_score(y_test,
    grid_predictions) * 100, "%")

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, grid_predictions))

# Calculate the confusion matrix
cm = confusion_matrix(y_test, grid_predictions)

# Create a DataFrame from the confusion matrix
```

```python
cm_df = pd.DataFrame(cm, index=[0, 1], columns=[0, 1])

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm_df, annot=True, fmt='g', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



(a) Confusion Matrix

```
Accuracy Score: 78.72037914691943 %

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1549
           1       0.62      0.52      0.56       561

    accuracy                           0.79      2110
   macro avg       0.73      0.70      0.71      2110
weighted avg       0.78      0.79      0.78      2110
```

(b) Report the results

3 Datasets Employed:

1. **Used Cars Dataset:** This dataset is available on Kaggle and provides extensive data on used cars. It can be accessed via the following link: `https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data`

2. **Telco Customer Churn Dataset:** Also available on Kaggle, this dataset provides valuable insights into customer churn in the telecommunications industry. It can be accessed via the following link: `https://www.kaggle.com/datasets/blastchar/telco-customer-churn`