```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

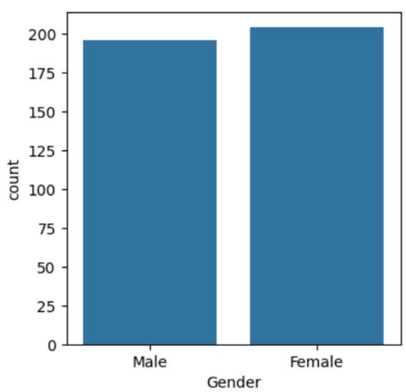
```
path = "/content/drive/MyDrive/Social_Network_Ads.csv"
df = pd.read_csv(path)
df.head(5)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

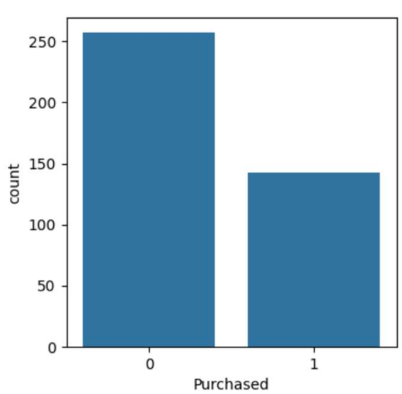
```
df.shape
```

```
(400, 5)
```

```
ax = plt.subplots(figsize = (4,4))
ax = sns.countplot(x=df['Gender'])
plt.show()
```




```
ax = plt.subplots(figsize = (4,4))
ax = sns.countplot(x=df['Purchased'])
plt.show()
```



```
[ ] # Separate features (X) and target variable (y)
    X = df.iloc[:, [1, 2, 3]].values  # Considering Gender, Age, and Estimated Salary as features
    y = df.iloc[:, 4].values  # Assuming 'Purchased' is the target variable
```

```
[ ] #use label encoder as 'Gender' is not numeric
    label_encoder = LabelEncoder()
    X[:, 0] = label_encoder.fit_transform(X[:, 0])
```

```
[ ] #Split dataset
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[ ] #feature scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
[ ] #create a KNN classifier with a value of k
    k_value = 5  # This value can be adjusted based on preference
    knn_classifier = KNeighborsClassifier(n_neighbors=k_value)
```

```
[ ] #fit the model with the training data
    knn_classifier.fit(X_train, y_train)
```

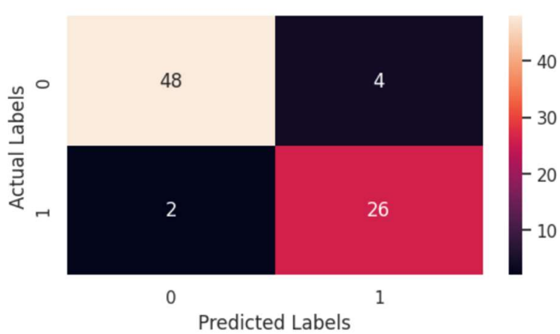
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
[ ] #prediction on test case
    y_pred = knn_classifier.predict(X_test)
```

```
[ ] #evaluate the performance of the classifier
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    classification_report_str = classification_report(y_test, y_pred)
```

```
#print the result
print(f'KNN Accuracy: {accuracy}')
print(f'KNN Confusion Matrix:\n{conf_matrix}')
sns.set(rc={'figure.figsize':(6,3)})
sns.heatmap(confusion_matrix(y_test,y_pred),annot = True,fmt = 'd')
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
print(f'KNN Classification Report:\n{classification_report_str}')
```

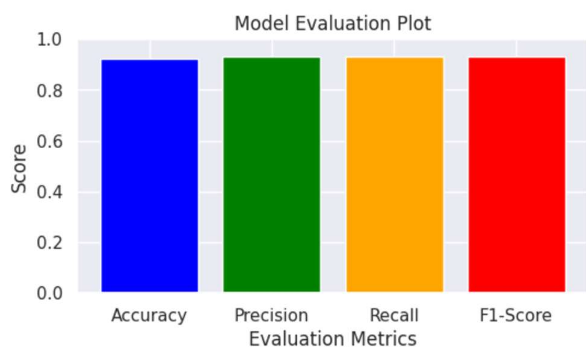
```
KNN Accuracy: 0.925
KNN Confusion Matrix:
[[48  4]
 [ 2 26]]
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        52
           1       0.87      0.93      0.90        28

    accuracy                           0.93        80
   macro avg       0.91      0.93      0.92        80
weighted avg       0.93      0.93      0.93        80
```



```
[ ] # Assuming we have a new set of feature values for prediction
    new_data = np.array([[0, 30, 50000]])  # Example: Gender (0 for Female, 1 for Male), Age, Estimated Salary

    # Use the trained KNN model to make predictions
    predicted_purchase = knn_classifier.predict(scaler.transform(new_data))

    # Print the predicted outcome
    if predicted_purchase[0] == 1:
        print("The targeted audience is predicted to purchase the product.")
    else:
        print("The targeted audience is predicted not to purchase the product.")
```

```
The targeted audience is predicted not to purchase the product.
```

```
[ ] # Assuming we have already evaluated the model and obtained these metrics, hence plotting the same in a bar plot
    accuracy = 0.925
    precision = 0.93
    recall = 0.93
    f1_score = 0.93

    # Plotting the bar plot
    metrics_names = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
    metrics_values = [accuracy, precision, recall, f1_score]

    plt.bar(metrics_names, metrics_values, color=['blue', 'green', 'orange', 'red'])
    plt.ylim([0, 1])  # Set the y-axis limit between 0 and 1
    plt.title('Model Evaluation Plot')
    plt.xlabel('Evaluation Metrics')
    plt.ylabel('Score')
    plt.show()
```



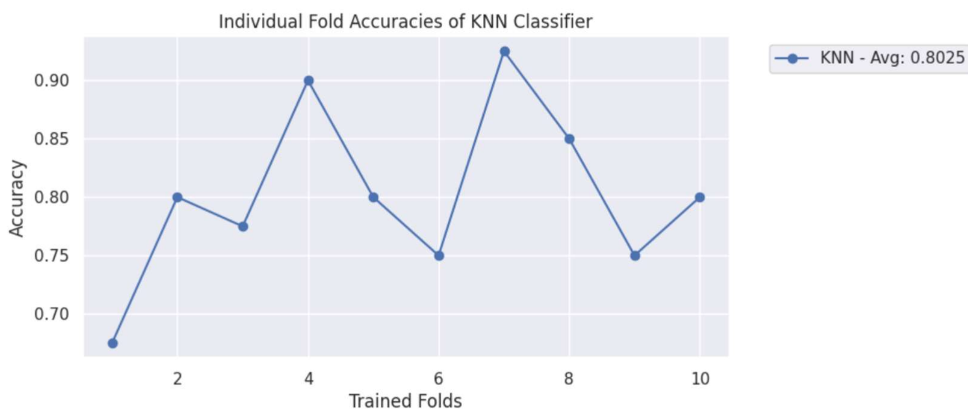
10 Fold Cross Validation

```
[ ] from sklearn.model_selection import cross_val_score, StratifiedKFold
    from sklearn.metrics import make_scorer

    # Define stratified 10-fold cross-validation
    cross_val = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    # Define accuracy as the evaluation metric
    scoring = make_scorer(accuracy_score)

    # Perform cross-validation on KNN
    cv_results = cross_val_score(knn_classifier, X, y, cv=cross_val, scoring=scoring)

    # Display results
    print("Cross-Validation Results:")
    print("Individual Accuracies:", cv_results)
    print("Average Accuracy:", np.mean(cv_results))
```

```
Cross-Validation Results:
Individual Accuracies: [0.675 0.8   0.775 0.9   0.8   0.75  0.925 0.85  0.75  0.8  ]
Average Accuracy: 0.8025
```

Cross validation Result visualization

```
[ ] # Cross-Validation Result
    model = ['KNN']
    accuracies = {
        'KNN': [0.675, 0.8, 0.775, 0.9, 0.8, 0.75, 0.925, 0.85, 0.75, 0.8],
    }

    # Plotting
    plt.figure(figsize=(8, 4))

    for model in model:
        plt.plot(range(1, 11), accuracies[model], marker='o', label=f'{model} - Avg: {sum(accuracies[model])/10:.4f}')

    plt.title('Individual Fold Accuracies of KNN Classifier')
    plt.xlabel('Trained Folds')
    plt.ylabel('Accuracy')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')  # Placing the legend outside the plot area
    plt.show()
```



ROC-Curve Plotting

```
from sklearn.metrics import roc_curve, auc

# Assuming y_test is the actual labels
label_encoder = LabelEncoder()
y_test_binary = label_encoder.fit_transform(y_test)

# Get predicted probabilities for the positive class
knn_predicted_scores = knn_classifier.predict_proba(X_test)[:, 1]

# Compute ROC curve and AUC for KNN model
knn_fpr, knn_tpr, _ = roc_curve(y_test_binary, knn_predicted_scores)

# Compute AUC for KNN model
knn_roc_auc = auc(knn_fpr, knn_tpr)

# Plot ROC curves for each model
plt.figure(figsize=(8, 6))
sns.set(style='darkgrid')
```

```
plt.plot(knn_fpr, knn_tpr, color='red', lw=2, label=f'KNN (AUC = {knn_roc_auc:.2f})')

plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('AUC-ROC Curve for KNN Model')
plt.legend(loc='lower right')
plt.show()
```