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

```
df = pd.read_csv('/content/drive/MyDrive/Social_Network_Ads.csv')
```

```
df.head()
```

	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

Next steps: Generate code with df View recommended plots

```
df.columns
```

```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

```
df.isnull().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

```
numeric_df = df.select_dtypes(include=["int64", "float64"])
```

```
covariance_matrix = numeric_df.cov()
```

```
most_promising_features = covariance_matrix.index[covariance_matrix.abs().sum().argsort()[::-1]]
```

```
print(most_promising_features)
```

```
Index(['User ID', 'EstimatedSalary', 'Age', 'Purchased'], dtype='object')
```

```
X = df.drop('EstimatedSalary', axis=1)
```

```
Y = df['Age']
```

```
print(X.head())
```

```
   User ID  Gender  Age  Purchased
0  15624510    Male   19          0
1  15810944    Male   35          0
2  15668575  Female   26          0
3  15603246  Female   27          0
4  15804002    Male   19          0
```

```
print(Y.head())
```

```
0    19
1    35
2    26
3    27
4    19
Name: Age, dtype: int64
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)


from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()


import pandas as pd


print(X_train.head())
```

```
      User ID  Gender  Age  Purchased
3    15603246  Female   27          0
18   15704583    Male   46          1
202  15735549  Female   39          1
250  15810075  Female   44          0
274  15692819  Female   57          1
```

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
X_train["Gender"] = label_encoder.fit_transform(X_train["Gender"])


X_train_scaled = scaler.fit_transform(X_train)


X_test.dtypes
```

```
User ID       int64
Gender       object
Age           int64
Purchased     int64
dtype: object
```

```
X_test["Gender"].unique()
```

```
array(['Female', 'Male'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
X_test["Gender"] = le.fit_transform(X_test["Gender"])


X_test_scaled = scaler.transform(X_test)


X_test_scaled
```

```
       [-1.34280661,  1.        ,  0.89565505,  1.3351437 ],
       [-0.01244574, -1.        ,  2.07309956,  1.3351437 ],
       [-0.89834178,  1.        , -1.85171546, -0.74898305],
       [ 0.2366299 ,  1.        ,  1.28813655,  1.3351437 ],
       [ 0.69708377,  1.        ,  0.40505317, -0.74898305],
```

```
       [-1.49470949, -1.        , -0.37990983, -0.74898305],
       [ 1.52770699,  1.        , -1.26299321, -0.74898305],
       [-1.08368389, -1.        ,  1.4843773 ,  1.3351437 ],
       [ 1.69399862, -1.        ,  0.01257167, -0.74898305],
       [ 0.94439213,  1.        , -1.26299321, -0.74898305],
       [ 1.45868552, -1.        , -0.0855487 , -0.74898305],
       [ 1.30444482, -1.        , -1.06675246, -0.74898305],
       [-1.32687323,  1.        ,  2.17121993,  1.3351437 ],
       [ 1.46941446, -1.        , -1.16487283, -0.74898305],
       [-1.50105501,  1.        , -0.67427095, -0.74898305],
       [ 0.14166973,  1.        , -0.67427095, -0.74898305],
       [-0.15165758, -1.        ,  0.3069328 , -0.74898305],
       [-1.70295671, -1.        , -0.28178945, -0.74898305],
       [ 0.84870834, -1.        ,  1.38625693,  1.3351437 ],
       [-0.30337955, -1.        , -0.96863208, -0.74898305],
       [-1.33662808,  1.        , -0.96863208, -0.74898305],
       [ 0.55645254, -1.        , -1.06675246,  1.3351437 ],
       [-1.41448596, -1.        ,  0.40505317,  1.3351437 ],
       [ 0.79509147, -1.        ,  0.89565505,  1.3351437 ],
       [ 0.95413306, -1.        ,  0.11069205, -0.74898305],
       [ 1.30060411, -1.        , -0.4780302 , -0.74898305],
       [ 1.19403831, -1.        ,  1.38625693,  1.3351437 ],
       [ 1.65376857,  1.        , -1.85171546, -0.74898305],
       [ 0.55254225,  1.        , -1.06675246, -0.74898305],
       [-0.69284986,  1.        , -1.45923396, -0.74898305],
       [-0.23289695,  1.        ,  0.89565505,  1.3351437 ],
       [-0.71057836, -1.        , -0.28178945, -0.74898305],
       [-0.05177127,  1.        ,  1.77873843,  1.3351437 ],
       [-0.89493246, -1.        ,  1.58249768,  1.3351437 ],
       [-1.10508611, -1.        , -0.28178945, -0.74898305],
       [ 0.34574226, -1.        , -0.0855487 ,  1.3351437 ]])
```

```
X_train_scaled
```

```
array([[-1.19424348, -1.        , -1.06675246, -0.74898305],
       [ 0.21592346,  1.        ,  0.79753468,  1.3351437 ],
       [ 0.64683448, -1.        ,  0.11069205,  1.3351437 ],
       ...,
       [-0.11569962, -1.        ,  0.50317355, -0.74898305],
       [-1.52869143,  1.        ,  0.11069205, -0.74898305],
       [-1.45447944, -1.        , -0.57615058, -0.74898305]])
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled = scaler.fit_transform(X_train)

# Transform the testing data
X_test_scaled = scaler.transform(X_test)

# Initialize and fit the logistic regression model
logreg = LogisticRegression()
logreg.fit(X_train_scaled, Y_train)

# Predict using the fitted model
y_pred = logreg.predict(X_test_scaled)
```

```
logreg = LogisticRegression(max_iter=1000)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

logreg = LogisticRegression()
logreg.fit(X_train_scaled, Y_train)

Y_train_pred = logreg.predict(X_train_scaled)
Y_test_pred = logreg.predict(X_test_scaled)
```

```
logreg = LogisticRegression(solver='saga')
```

```
import warnings
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.exceptions import UndefinedMetricWarning

# Ignore the warning messages
warnings.filterwarnings("ignore", category=UndefinedMetricWarning)

# Calculate accuracy for training set
train_accuracy = accuracy_score(Y_train, Y_train_pred)

# Calculate accuracy for testing set
test_accuracy = accuracy_score(Y_test, y_pred)

# Calculate precision for training set
train_precision = precision_score(Y_train, Y_train_pred, average='weighted', zero_division=1)

# Calculate precision for testing set
test_precision = precision_score(Y_test, y_pred, average='weighted', zero_division=1)

# Calculate recall for training set
train_recall = recall_score(Y_train, Y_train_pred, average='weighted', zero_division=1)

# Calculate recall for testing set
test_recall = recall_score(Y_test, y_pred, average='weighted', zero_division=1)

# Calculate F1 score for training set
train_f1_score = f1_score(Y_train, Y_train_pred, average='weighted', zero_division=1)

# Calculate F1 score for testing set
test_f1_score = f1_score(Y_test, y_pred, average='weighted', zero_division=1)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision
precision = precision_score(Y_test, y_pred, average='micro')

# Calculate recall
recall = recall_score(Y_test, y_pred, average='micro')

# Calculate F1 score
f1_score = f1_score(Y_test, y_pred, average='micro')
```

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test, y_pred)
```

```
cm
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 2, 0, 0],
       [0, 0, 0, ..., 0, 0, 1]])
```