Kerem Dogan

901438964

Screenshots

There are two types of supervised learning—classification and regression. Binary classification is used to predict a target variable that has only two labels, typically represented numerically with a zero or a one.

The `.head()` of a dataset, `churn_df`, is shown below. You can expect the rest of the data to contain similar values.

```
_length  total_day_charge  total_eve_charge  total_night_charge  total_intl_charge  customer_service_calls  churn
    101             45.85             17.65                9.64               1.22                       3      1
     73             22.30              9.05                9.98               2.75                       2      0
     86             24.62             17.53               11.49               3.13                       4      0
     59             34.73             21.02                9.66               3.24                       1      0
    129             27.42             18.75               10.11               2.59                       1      0
```

Looking at this data, which column could be the target variable for binary classification?

Answer the question 50XP

Possible Answers

Select one answer

- ( ) `"customer_service_calls"` PRESS 1
- ( ) `"total_night_charge"` PRESS 2
- (•) `"churn"` PRESS 3
- ( ) `"account_length"` PRESS 4

Drag the items below into order

```
from sklearn.module import Model
```

```
model = Model()
```

```
model.fit(X, y)
```

```
model.predict(X_new)
```

script.py

```python
# Import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

y = churn_df["churn"].values
X = churn_df[["account_length", "customer_service_calls"]].values

# Create a KNN classifier with 6 neighbors
knn = KNeighborsClassifier(n_neighbors=6)

# Fit the classifier to the data
knn.fit(X, y)
```

script.py

```python
# Predict the labels for the X_new
y_pred = knn.predict(X_new)

# Print the predictions
print("Predictions: {}".format(y_pred))
```

```python
# Import the module
from sklearn.model_selection import train_test_split

X = churn_df.drop("churn", axis=1).values
y = churn_df["churn"].values

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the classifier to the training data
knn.fit(X_train, y_train)

# Print the accuracy
print(knn.score(X_test, y_test))
```

```python
# Create neighbors
neighbors = np.arange(1, 13)
train_accuracies = {}
test_accuracies = {}

for neighbor in neighbors:

    # Set up a KNN Classifier
    knn = KNeighborsClassifier(n_neighbors=neighbor)

    # Fit the model
    knn.fit(X_train, y_train)

    # Compute accuracy
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)
print(neighbors, '\n', train_accuracies, '\n', test_accuracies)
```

```python
# Add a title
plt.title("KNN: Varying Number of Neighbors")
# Plot training accuracies
plt.plot(neighbors, train_accuracies.values(), label="Training Accuracy")
# Plot test accuracies
plt.plot(neighbors, test_accuracies.values(), label="Testing Accuracy")
plt.legend()
plt.xlabel("Number of Neighbors")
plt.ylabel("Accuracy")
# Display the plot
plt.show()
```

```python
import numpy as np
# Create X from the radio column's values
X = sales_df["radio"].values
# Create y from the sales column's values
y = sales_df["sales"].values
# Reshape X
X = X.reshape(-1, 1)
# Check the shape of the features and targets
print(X.shape, y.shape)
```

```python
# Import LinearRegression
from sklearn.linear_model import LinearRegression
# Create the model
reg = LinearRegression()
# Fit the model to the data
reg.fit(X, y)
# Make predictions
predictions = reg.predict(X)
print(predictions[:5])
```

```python
# Import matplotlib.pyplot
import matplotlib.pyplot as plt

# Create scatter plot
plt.scatter(X, y, color="blue")
# Create line plot
plt.plot(X, predictions, color="red")
plt.xlabel("Radio Expenditure ($)")
plt.ylabel("Sales ($)")
# Display the plot
plt.show()
```

```python
# Create X and y arrays
X = sales_df.drop("sales", axis=1).values
y = sales_df["sales"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Instantiate the model
reg = LinearRegression()
# Fit the model to the data
reg.fit(X_train, y_train)
# Make predictions
y_pred = reg.predict(X_test)
print("Predictions: {}, Actual Values: {}".format(y_pred[:2], y_test[:2]))
```

```python
# Import mean_squared_error
from sklearn.metrics import mean_squared_error
# Compute R-squared
r_squared = reg.score(X_test, y_test)
# Compute RMSE
rmse = mean_squared_error(y_test, y_pred, squared=False)
# Print the metrics
print("R^2: {}".format(r_squared))
print("RMSE: {}".format(rmse))
```

```python
# Import the necessary modules
from sklearn.model_selection import KFold, cross_val_score
# Create a KFold object
kf = KFold(n_splits=6, shuffle=True, random_state=5)
reg = LinearRegression()
# Compute 6-fold cross-validation scores
cv_scores = cross_val_score(reg, X, y, cv=kf)
# Print scores
print(cv_scores)
```

```python
# Print the mean
print(np.mean(cv_results))
# Print the standard deviation
print(np.std(cv_results))
# Print the 95% confidence interval
print(np.quantile(cv_results, [0.025, 0.975]))
```

```python
# Import Ridge
from sklearn.linear_model import Ridge
alphas = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
ridge_scores = []
for alpha in alphas:
  # Create a Ridge regression model
  ridge = Ridge(alpha=alpha)
  # Fit the data
  ridge.fit(X_train, y_train)
  # Obtain R-squared
  score = ridge.score(X_test, y_test)
  ridge_scores.append(score)
print(ridge_scores)
```

```python
# Import Lasso
from sklearn.linear_model import Lasso
# Instantiate a lasso regression model
lasso = Lasso(alpha=0.3)
# Fit the model to the data
lasso.fit(X, y)
# Compute and print the coefficients
lasso_coef = lasso.coef_
print(lasso_coef)
plt.bar(sales_columns, lasso_coef)
plt.xticks(rotation=45)
plt.show()
```

```python
# Import confusion matrix
from sklearn.metrics import confusion_matrix, classification_report
knn = KNeighborsClassifier(n_neighbors=6)
# Fit the model to the training data
knn.fit(X_train, y_train)
# Predict the labels of the test data: y_pred
y_pred = knn.predict(X_test)
# Generate the confusion matrix and classification report
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```python
# Import LogisticRegression
from sklearn.linear_model import LogisticRegression
# Instantiate the model
logreg = LogisticRegression()
# Fit the model
logreg.fit(X_train, y_train)
# Predict probabilities
y_pred_probs = logreg.predict_proba(X_test)[:, 1]
print(y_pred_probs[:10])
```

```python
from sklearn.metrics import roc_curve
# Generate ROC curve values: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_pred_probs)
plt.plot([0, 1], [0, 1], 'k--')
# Plot tpr against fpr
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Diabetes Prediction')
plt.show()
```

```python
# Import roc_auc_score
from sklearn.metrics import roc_auc_score
# Calculate roc_auc_score
print(roc_auc_score(y_test, y_pred_probs))
# Calculate the confusion matrix
print(confusion_matrix(y_test, y_pred))
# Calculate the classification report
print(classification_report(y_test, y_pred))
```

```python
# Import GridSearchCV
from sklearn.model_selection import GridSearchCV
# Set up the parameter grid
param_grid = {"alpha": np.linspace(0.00001, 1, 20)}
# Instantiate lasso_cv
lasso_cv = GridSearchCV(lasso, param_grid, cv=kf)
# Fit to the training data
lasso_cv.fit(X_train, y_train)
print("Tuned lasso paramaters: {}".format(lasso_cv.best_params_))
print("Tuned lasso score: {}".format(lasso_cv.best_score_))
```

```python
# Create the parameter space
params = {"penalty": ["l1", "l2"],
          "tol": np.linspace(0.0001, 1.0, 50),
          "C": np.linspace(0.1, 1.0, 50),
          "class_weight": ["balanced", {0:0.8, 1:0.2}]}
# Instantiate the RandomizedSearchCV object
logreg_cv = RandomizedSearchCV(logreg, params, cv=kf)
# Fit the data to the model
logreg_cv.fit(X_train, y_train)
# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters: {}".format(logreg_cv.best_params_))
print("Tuned Logistic Regression Best Accuracy Score: {}".format(logreg_cv.best_score_))
```

Run Code Submit Answer

```python
# Create music_dummies
music_dummies = pd.get_dummies(music_df, drop_first=True)
# Print the new DataFrame's shape
print("Shape of music_dummies: {}".format(music_dummies.shape))
```

```python
# Create X and y
X = music_dummies.drop("popularity", axis=1).values
y = music_dummies["popularity"].values
# Instantiate a ridge model
ridge = Ridge(alpha=0.2)
# Perform cross-validation
scores = cross_val_score(ridge, X, y, cv=kf, scoring="neg_mean_squared_error")
# Calculate RMSE
rmse = np.sqrt(-scores)
print("Average RMSE: {}".format(np.mean(rmse)))
print("Standard Deviation of the target array: {}".format(np.std(y)))
```

```python
# Print missing values for each column
print(music_df.isna().sum().sort_values())
# Remove values where less than 5% are missing
music_df = music_df.dropna(subset=["genre", "popularity", "loudness",
"liveness", "tempo"])
# Convert genre to a binary feature
music_df["genre"] = np.where(music_df["genre"] == "Rock", 1, 0)
print(music_df.isna().sum().sort_values())
print("Shape of the `music_df`: {}".format(music_df.shape))
```

```python
# Import modules
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
# Instantiate an imputer
imputer = SimpleImputer()
# Instantiate a knn model
knn = KNeighborsClassifier(n_neighbors=3)
# Build steps for the pipeline
steps = [("imputer", imputer),
         ("knn", knn)]
```

```python
steps = [("imputer", imp_mean),
         ("knn", knn)]
# Create the pipeline
pipeline = Pipeline(steps)
# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)
# Make predictions on the test set
y_pred = pipeline.predict(X_test)
# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```python
# Import StandardScaler
from sklearn.preprocessing import StandardScaler
# Create pipeline steps
steps = [("scaler", StandardScaler()),
         ("lasso", Lasso(alpha=0.5))]
# Instantiate the pipeline
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)
# Calculate and print R-squared
print(pipeline.score(X_test, y_test))
```

```python
steps = [("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]
pipeline = Pipeline(steps)
# Create the parameter space
parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)
# Instantiate the grid search object
cv = GridSearchCV(pipeline, param_grid=parameters)
# Fit to the training data
cv.fit(X_train, y_train)
print(cv.best_score_, "\n", cv.best_params_)
```

```python
models = {"Linear Regression": LinearRegression(), "Ridge": Ridge(alpha=0.1), "Lasso": Lasso(alpha=0.1)}
results = []
# Loop through the models' values
for model in models.values():
  kf = KFold(n_splits=6, random_state=42, shuffle=True)
  # Perform cross-validation
  cv_scores = cross_val_score(model, X_train, y_train, cv=kf)
  # Append the results
  results.append(cv_scores)
# Create a box plot of the results
plt.boxplot(results, labels=models.keys())
plt.show()
```

```python
# Import mean_squared_error
from sklearn.metrics import mean_squared_error
for name, model in models.items():
  # Fit the model to the training data
  model.fit(X_train_scaled, y_train)
  # Make predictions on the test set
  y_pred = model.predict(X_test_scaled)
  # Calculate the test_rmse
  test_rmse = mean_squared_error(y_test, y_pred, squared=False)
  print("{} Test Set RMSE: {}".format(name, test_rmse))
```

```python
# Create models dictionary
models = {"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(), "Decision Tree Classifier": DecisionTreeClassifier()}
results = []
# Loop through the models' values
for model in models.values():
    # Instantiate a KFold object
    kf = KFold(n_splits=6, random_state=12, shuffle=True)
    # Perform cross-validation
    cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
    results.append(cv_results)
plt.boxplot(results, labels=models.keys())
plt.show()
```

```python
# Create steps
steps = [("imp_mean", SimpleImputer()),
         ("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]
# Set up pipeline
pipeline = Pipeline(steps)
params = {"logreg__solver": ["newton-cg", "saga", "lbfgs"],
          "logreg__C": np.linspace(0.001, 1.0, 10)}
# Create the GridSearchCV object
tuning = GridSearchCV(pipeline, param_grid=params)
tuning.fit(X_train, y_train)
y_pred = tuning.predict(X_test)
# Compute and print performance
print("Tuned Logistic Regression Parameters: {}, Accuracy: {}".format(tuning.best_params_, tuning.score(X_test, y_test)))
```