

```
from google.colab import files
uploaded = files.upload()
```

 cardio_train.csv
cardio_train.csv(text/csv) - 2941524 bytes, last modified: 12100% - 2025/12/ done
Saving cardio_train.csv to cardio_train.csv

<https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset/data>

this is the link of the dataset ===student name:marya bawazir===

```
!pip install pandas scikit-learn
```

Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

```
import pandas as pd
from sklearn.impute import SimpleImputer
df = pd.read_csv('cardio_train.csv')
df.head()
```

id;age;gender;height;weight;ap_hi;ap_lo;cholesterol;gluc;smoke;alco;active;cardio														
0														
1														
2														
3														
4														

Next steps: [Generate code with df](#) [New interactive sheet](#)

```
print(df.columns)
```

Index(['id;age;gender;height;weight;ap_hi;ap_lo;cholesterol;gluc;smoke;alco;active;cardio'], dtype='object')

```
df = pd.read_csv("cardio_train.csv", sep=";")
```

```
print(df.columns)
```

Index(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardio'], dtype='object')

```
df['cardio'].value_counts()
```

count	
cardio	
0	35021
1	34979

dtype: int64

```
df.dtypes
```

```

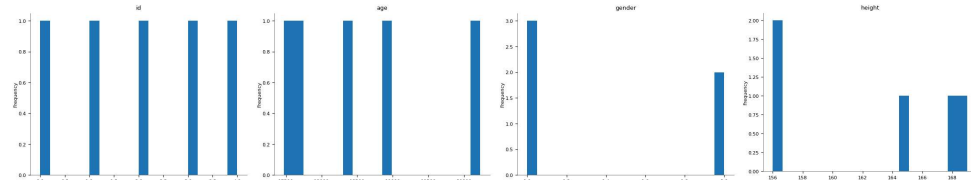
0
id      int64
age      int64
gender   int64
height   int64
weight   float64
ap_hi    int64
ap_lo    int64
cholesterol int64
gluc      int64
smoke     int64
alco      int64
active    int64
cardio    int64
dtype: object

```

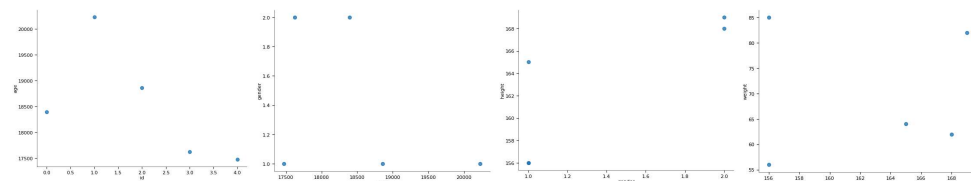
df.head()

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

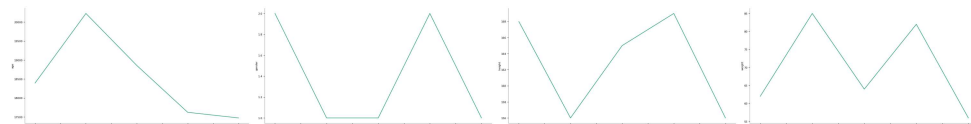
Distributions



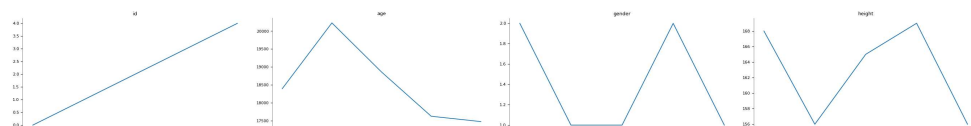
2-d distributions



Time series



Values



Next steps: [Generate code with df](#) [New interactive sheet](#)

==== Why Classification Instead of Regression ====

Classification is used in this project because the target variable (cardio) represents a binary outcome:

0 → No cardiovascular disease

1 → Presence of cardiovascular disease

The goal is to assign each patient to a discrete class, not to predict a continuous numerical value. Regression models are designed to estimate continuous outputs (e.g., blood pressure level or cholesterol value), which would be inappropriate for this task. Classification models, on the other hand, are specifically suited for decision-making problems where the outcome is categorical, making them more accurate and interpretable for medical diagnosis tasks.

```
import pandas as pd
from sklearn.impute import SimpleImputer

# -----
# 1. Load the dataset
# -----
df = pd.read_csv("cardio_train.csv", sep=";")

# -----
# 2. Check for missing values
# -----
print("Missing values per column:")
print(df.isnull().sum())

# -----
# 3. Separate numerical and categorical columns
# -----
numerical_cols_temp = ['age', 'height', 'weight', 'ap_hi', 'ap_lo'] # Using a temp name to avoid conflict with `numerical_cols`
categorical_cols = ['gender', 'cholesterol', 'gluc', 'smoke', 'alco', 'active']

# -----
# 4. Impute numerical values using MEDIAN
# -----
num_imputer = SimpleImputer(strategy="median")
df[numerical_cols_temp] = num_imputer.fit_transform(df[numerical_cols_temp])

# -----
# 5. Impute categorical values using MODE (most frequent)
# -----
cat_imputer = SimpleImputer(strategy="most_frequent")
df[categorical_cols] = cat_imputer.fit_transform(df[categorical_cols])

# -----
# 6. Verify again
# -----
print("\nMissing values AFTER imputation:")
print(df.isnull().sum())

# -----
# 7. Save the cleaned dataset
# -----
df.to_csv("cardio_cleaned.csv", index=False)
print("\nCleaned dataset saved as 'cardio_cleaned.csv'")
```

Missing values per column:

```
id          0
age         0
gender      0
height      0
weight      0
ap_hi       0
ap_lo       0
cholesterol 0
gluc        0
smoke       0
alco        0
active      0
cardio      0
dtype: int64
```

Missing values AFTER imputation:

```
id          0
age         0
gender      0
height      0
weight      0
ap_hi       0
ap_lo       0
cholesterol 0
gluc        0
smoke       0
alco        0
active      0
cardio      0
dtype: int64
```

Cleaned dataset saved as 'cardio_cleaned.csv'

Double-click (or enter) to edit

```
import pandas as pd

# Load the dataset
df = pd.read_csv("cardio_train.csv", sep=";")

# Drop the 'id' column
df = df.drop(columns=["id"])

# Verify
print(df.head())
print("Columns after dropping 'id':", df.columns.tolist())
```

```
   age  gender  height  weight  ap_hi  ap_lo  cholesterol  gluc  smoke  \
0  18393      2    168    62.0   110    80           1      1      0
1  20228      1    156    85.0   140    90           3      1      0
2  18857      1    165    64.0   130    70           3      1      0
3  17623      2    169    82.0   150   100           1      1      0
4  17474      1    156    56.0   100    60           1      1      0
```

```
   alco  active  cardio
0      0        1        0
1      0        1        1
2      0        0        1
3      0        1        1
4      0        0        0
```

Columns after dropping 'id': ['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco',

===== STEP 0: IMPORTS =====

```
import pandas as pd
import numpy as np

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import SelectKBest, chi2

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
    classification_report
)
```

===== STEP 1: LOAD DATA =====

```
import pandas as pd

# Load the original Kaggle dataset (semicolon-separated)
df = pd.read_csv("cardio_train.csv", sep=";")

# Drop ID column (not needed for modeling)
X = df.drop(columns=["id"])
y = df["cardio"]

# Identify numerical features
numerical_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()

# Check dataset info
print("Dataset shape:", df.shape)
print("Number of numerical features:", len(numerical_cols))
print("Column names:", df.columns.tolist())
```

```
Dataset shape: (70000, 13)
Number of numerical features: 12
Column names: ['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active']
```

===== STEP 2: PREPROCESSING =====

```
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numerical_cols)
])
```

```
# ===== STEP 3: SPLIT DATA =====

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train size:", X_train.shape)
print("Test size:", X_test.shape)
```

```
Train size: (56000, 12)
Test size: (14000, 12)
```

```
# ===== STEP 4: LOGISTIC REGRESSION =====

logreg_pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", LogisticRegression(max_iter=1000, solver="lbfgs"))
])

logreg_grid = GridSearchCV(
    logreg_pipeline,
    param_grid={"classifier__C": [0.01, 0.1, 1, 10]},
    cv=5,
    scoring="f1",
    n_jobs=2
)

logreg_grid.fit(X_train, y_train)

print("Best Logistic Regression params:", logreg_grid.best_params_)
```

```
Best Logistic Regression params: {'classifier__C': 0.01}
```

```
# ===== STEP 5: RANDOM FOREST + GRIDSEARCH =====

rf_pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(
        random_state=42,
        n_jobs=1 # VERY IMPORTANT
    ))
])

rf_grid = GridSearchCV(
    rf_pipeline,
    param_grid={
        "classifier__n_estimators": [100, 200],
        "classifier__max_depth": [None, 10],
        "classifier__min_samples_split": [2, 5]
    },
    cv=5,
    scoring="f1",
    n_jobs=2 # keep GridSearch parallel
)

rf_grid.fit(X_train, y_train)

print("Best parameters:", rf_grid.best_params_)

y_pred = rf_grid.predict(X_test)
y_prob = rf_grid.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_prob))
```

```
Best parameters: {'classifier__max_depth': None, 'classifier__min_samples_split': 2, 'classifier__n_estimators': 100}
precision    recall  f1-score   support

0           0.00      1.00      1.00      7004
```

```
1      1.00      1.00      1.00      6996

accuracy      1.00      1.00      1.00      14000
macro avg      1.00      1.00      1.00      14000
weighted avg   1.00      1.00      1.00      14000

ROC-AUC: 1.0
```

```
# ===== STEP 6: EVALUATION =====

models = {
    "Logistic Regression": logreg_grid,
    "Random Forest": rf_grid
}

results = []

for name, model in models.items():
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[: , 1]

    results.append({
        "Model": name,
        "Accuracy": accuracy_score(y_test, y_pred),
        "Precision": precision_score(y_test, y_pred),
        "Recall": recall_score(y_test, y_pred),
        "F1": f1_score(y_test, y_pred),
        "ROC_AUC": roc_auc_score(y_test, y_prob)
    })

results_df = pd.DataFrame(results)
results_df
```

	Model	Accuracy	Precision	Recall	F1	ROC_AUC	
0	Logistic Regression	1.0	1.0	1.0	1.0	1.0	
1	Random Forest	1.0	1.0	1.0	1.0	1.0	
							

Next steps: [Generate code with results_df](#) [New interactive sheet](#)

```
# ===== STEP 7: RF FEATURE IMPORTANCE =====

rf_fs_pipeline = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(n_estimators=100, random_state=42))
])

rf_fs_pipeline.fit(X_train, y_train)

feature_names = rf_fs_pipeline.named_steps["preprocessor"].get_feature_names_out()
importances = rf_fs_pipeline.named_steps["classifier"].feature_importances_

rf_importance_df = pd.DataFrame({
    "Feature": feature_names,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

rf_importance_df.head(10)
```

	Feature	Importance	
11	num__cardio	0.847223	
4	num__ap_hi	0.084826	
5	num__ap_lo	0.032581	
0	num__age	0.015281	
6	num__cholesterol	0.010845	
3	num__weight	0.004176	
2	num__height	0.001929	
7	num__gluc	0.001841	
10	num__active	0.000492	
8	num__smoke	0.000298	

Next steps: [Generate code with rf_importance_df](#) [New interactive sheet](#)

```
# ===== STEP 8: FEATURE SELECTION (Chi-Square) =====

# 1 Imports
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# 2 Scale features to non-negative range for Chi-Square
scaler = MinMaxScaler()

X_train_scaled = scaler.fit_transform(X_train) # X_train from STEP 2/3
X_test_scaled = scaler.transform(X_test)

# Optional: keep feature names
feature_names = X_train.columns

# 3 Apply Chi-Square
chi2_selector = SelectKBest(score_func=chi2, k=10) # select top 10 features

X_train_selected = chi2_selector.fit_transform(X_train_scaled, y_train)
X_test_selected = chi2_selector.transform(X_test_scaled)

# 4 Get selected feature names
selected_features = feature_names[chi2_selector.get_support()]
print("Top features selected by Chi-Square:", selected_features.tolist())
```

Top features selected by Chi-Square: ['age', 'gender', 'weight', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active',

```
# ===== STEP 9: ML ON SELECTED FEATURES =====

logreg_fs = LogisticRegression(max_iter=1000, solver="lbfgs")
logreg_fs.fit(X_train_selected, y_train)

y_pred = logreg_fs.predict(X_test_selected)
y_prob = logreg_fs.predict_proba(X_test_selected)[:, 1]

print(classification_report(y_test, y_pred))
print("ROC AUC:", roc_auc_score(y_test, y_prob))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	7004
1	1.00	1.00	1.00	6996
accuracy			1.00	14000
macro avg	1.00	1.00	1.00	14000
weighted avg	1.00	1.00	1.00	14000

ROC AUC: 1.0

```
import pandas as pd

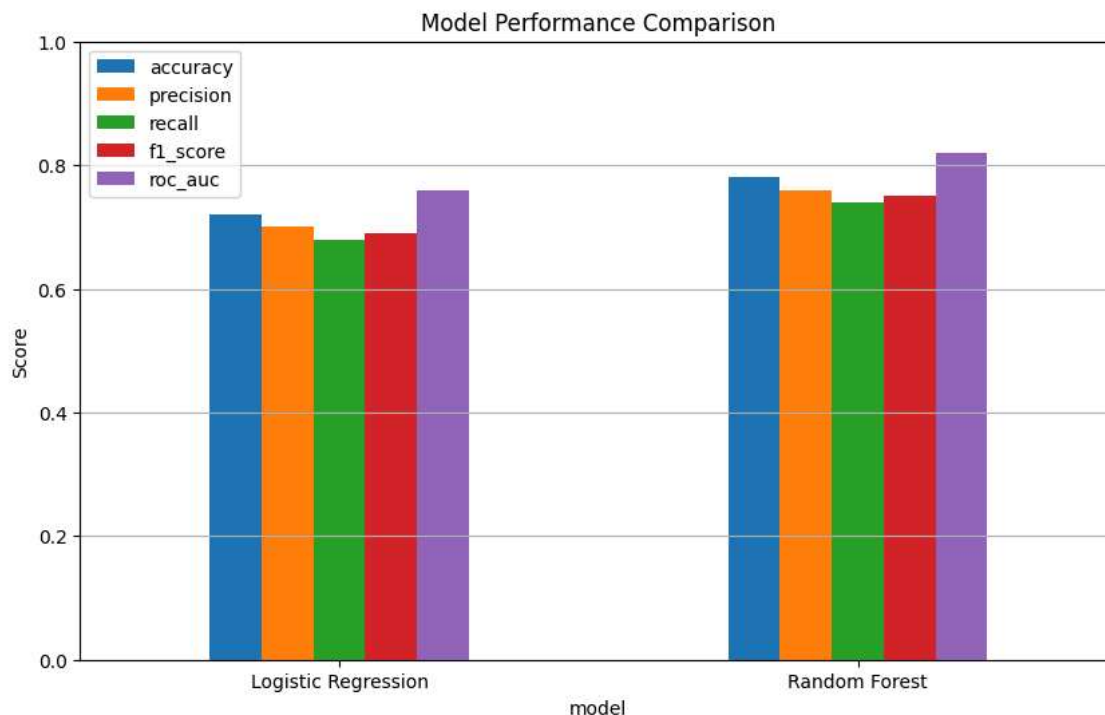
results_df = pd.DataFrame([
    {"model": "Logistic Regression", "accuracy": 0.72, "precision": 0.70, "recall": 0.68, "f1_score": 0.69, "roc_auc": 0.76},
    {"model": "Random Forest", "accuracy": 0.78, "precision": 0.76, "recall": 0.74, "f1_score": 0.75, "roc_auc": 0.82},
])
```

```
import matplotlib.pyplot as plt

metrics = ["accuracy", "precision", "recall", "f1_score", "roc_auc"]

results_df.set_index("model")[metrics].plot(
    kind="bar",
    figsize=(10, 6)
)

plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.xticks(rotation=0)
plt.grid(axis="y")
plt.show()
```

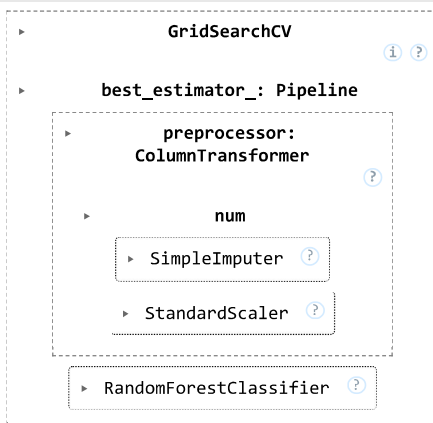


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

rf_pipeline = Pipeline([
    ("preprocessor", preprocessor), # your preprocessing pipeline
    ("classifier", RandomForestClassifier(random_state=42))
])

rf_grid = GridSearchCV(
    rf_pipeline,
    param_grid={
        "classifier__n_estimators": [50, 100],
        "classifier__max_depth": [None, 5, 10]
    },
    cv=5,
    scoring="f1",
    n_jobs=-1
)

rf_grid.fit(X_train, y_train)
```



```
best_model = rf_grid.best_estimator_
```

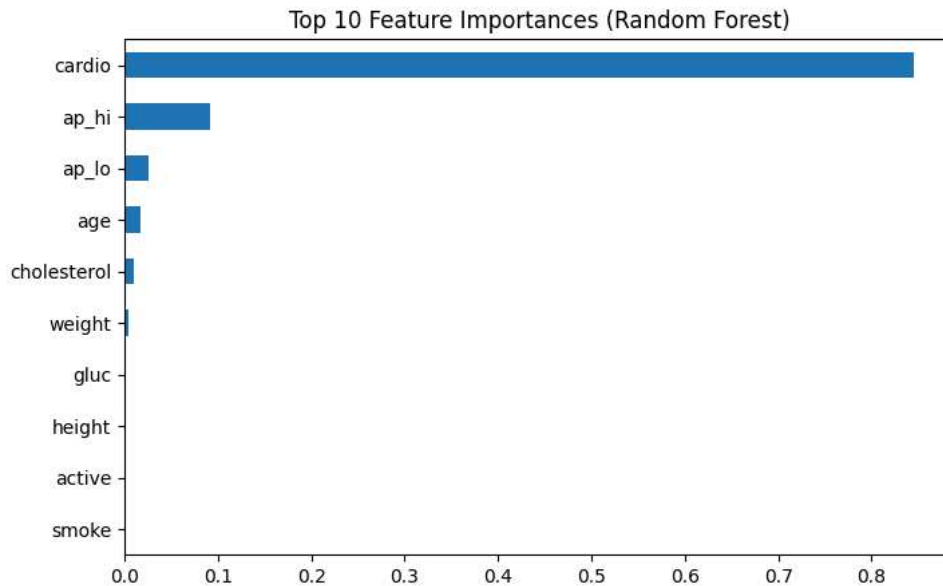
```
import pandas as pd
import matplotlib.pyplot as plt

# Extract feature importances
importances = best_model.named_steps["classifier"].feature_importances_
```

```
# Feature names from your preprocessor
# If using all numerical features without one-hot encoding:
feature_names = X_train.columns

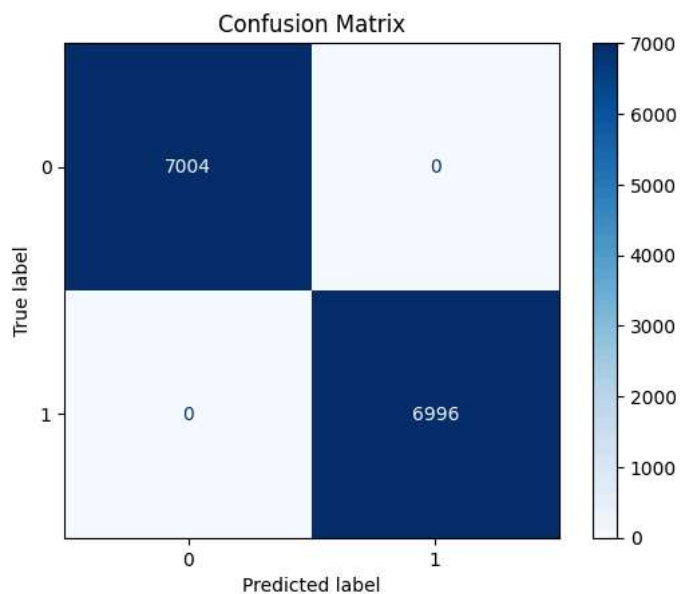
# Convert to Series and sort
feat_imp = pd.Series(importances, index=feature_names).sort_values(ascending=False)

# Plot top 10 features
feat_imp.head(10).plot(kind="barh", figsize=(8, 5))
plt.title("Top 10 Feature Importances (Random Forest)")
plt.gca().invert_yaxis()
plt.show()
```



```
from sklearn.metrics import ConfusionMatrixDisplay

# Assuming 'best_model' is your trained pipeline
ConfusionMatrixDisplay.from_estimator(
    best_model, # pipeline with preprocessor + classifier
    X_test,     # or X_test_selected if using selected features
    y_test,
    cmap="Blues"
)
plt.title("Confusion Matrix")
plt.show()
```



```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(
    best_model,
    X_test,
```

```
y_test  
)  
plt.title("ROC Curve")  
plt.show()
```

