smoker

December 12, 2025

[ ]:
```
# --- Cell 1: Environment & Imports ---
import sys
import os
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

RANDOM_STATE = 42

# 1. Force TensorFlow Backend (Fixes Keras 3 errors)
os.environ["KERAS_BACKEND"] = "tensorflow"

# 2. Machine Learning Imports
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix, roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTE

# 3. Deep Learning Imports (Robust Aliasing)
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam

# Settings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
sns.set(style="whitegrid")
```

[ ]:
```
# Load the dataset
df = pd.read_csv('smoker/train_dataset.csv')

print("--- Dataset Shape ---")
print(df.shape)
display(df.head())

# ---------- 2) Quick EDA ----------
print("\n--- Data types ---")
print(df.dtypes)

print("\n--- Duplicate Check ---")
duplicates = df.duplicated().sum()
print(f"Duplicate rows found: {duplicates}")

# print("\n--- Missing Values ---")
print("\n--- Missing values (counts) ---")
print(df.isna().sum().sort_values(ascending=False).head(30))

if duplicates > 0:
    df = df.drop_duplicates()
    print("Duplicates dropped.")

print("\n--- Basic statistics (numerical) ---")
display(df.describe().T)
```

```
--- Dataset Shape ---
(38984, 23)
```

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	\
0	35	170	85	97.0	0.9	0.9	
1	20	175	110	110.0	0.7	0.9	
2	45	155	65	86.0	0.9	0.9	
3	45	165	80	94.0	0.8	0.7	
4	20	165	60	81.0	1.5	0.1	

	hearing(left)	hearing(right)	systolic	relaxation	fasting blood sugar	\
0	1	1	118	78	97	
1	1	1	119	79	88	
2	1	1	110	80	80	
3	1	1	158	88	249	
4	1	1	109	64	100	

	Cholesterol	triglyceride	HDL	LDL	hemoglobin	Urine protein	\
0	239	153	70	142	19.8	1	
1	211	128	71	114	15.9	1	
2	193	120	57	112	13.7	3	
3	210	366	46	91	16.9	1	

```
4          179           200   47   92        14.9              1

   serum creatinine   AST   ALT  Gtp  dental caries  smoking
0               1.0    61   115  125              1        1
1               1.1    19    25   30              1        0
2               0.6  1090  1400  276              0        0
3               0.9    32    36   36              0        0
4               1.2    26    28   15              0        0


--- Data types ---
age                      int64
height(cm)               int64
weight(kg)               int64
waist(cm)              float64
eyesight(left)         float64
eyesight(right)        float64
hearing(left)            int64
hearing(right)           int64
systolic                 int64
relaxation               int64
fasting blood sugar      int64
Cholesterol              int64
triglyceride             int64
HDL                      int64
LDL                      int64
hemoglobin             float64
Urine protein            int64
serum creatinine       float64
AST                      int64
ALT                      int64
Gtp                      int64
dental caries            int64
smoking                  int64
dtype: object

--- Duplicate Check ---
Duplicate rows found: 5517

--- Missing values (counts) ---
age                    0
height(cm)             0
weight(kg)             0
waist(cm)              0
eyesight(left)         0
eyesight(right)        0
hearing(left)          0
hearing(right)         0
```

```
systolic               0
relaxation             0
fasting blood sugar    0
Cholesterol            0
triglyceride           0
HDL                    0
LDL                    0
hemoglobin             0
Urine protein          0
serum creatinine       0
AST                    0
ALT                    0
Gtp                    0
dental caries          0
smoking                0
dtype: int64
Duplicates dropped.

--- Basic statistics (numerical) ---
```

	count	mean	std	min	25%	50%	\
age	33467.0	44.153943	12.071768	20.0	40.0	40.0	
height(cm)	33467.0	164.684465	9.195867	130.0	160.0	165.0	
weight(kg)	33467.0	65.930319	12.877955	30.0	55.0	65.0	
waist(cm)	33467.0	82.081501	9.310533	51.0	76.0	82.0	
eyesight(left)	33467.0	1.013849	0.496245	0.1	0.8	1.0	
eyesight(right)	33467.0	1.009553	0.497867	0.1	0.8	1.0	
hearing(left)	33467.0	1.025368	0.157243	1.0	1.0	1.0	
hearing(right)	33467.0	1.026056	0.159303	1.0	1.0	1.0	
systolic	33467.0	121.498730	13.671019	71.0	112.0	120.0	
relaxation	33467.0	76.017599	9.672070	40.0	70.0	76.0	
fasting blood sugar	33467.0	99.261511	20.484366	46.0	89.0	96.0	
Cholesterol	33467.0	196.964562	36.416775	55.0	172.0	195.0	
triglyceride	33467.0	126.806048	71.765510	8.0	75.0	108.0	
HDL	33467.0	57.257537	14.598021	4.0	47.0	55.0	
LDL	33467.0	115.182090	43.159159	1.0	92.0	113.0	
hemoglobin	33467.0	14.624463	1.562414	4.9	13.6	14.8	
Urine protein	33467.0	1.086533	0.403008	1.0	1.0	1.0	
serum creatinine	33467.0	0.886467	0.222038	0.1	0.8	0.9	
AST	33467.0	26.195536	18.760580	6.0	19.0	23.0	
ALT	33467.0	27.139929	31.613159	1.0	15.0	21.0	
Gtp	33467.0	39.952401	49.965736	2.0	17.0	26.0	
dental caries	33467.0	0.214689	0.410613	0.0	0.0	0.0	
smoking	33467.0	0.366271	0.481792	0.0	0.0	0.0	

	75%	max
age	55.0	85.0
height(cm)	170.0	190.0

```
weight(kg)              75.0   135.0
waist(cm)               88.0   129.0
eyesight(left)           1.2     9.9
eyesight(right)          1.2     9.9
hearing(left)            1.0     2.0
hearing(right)           1.0     2.0
systolic               130.0   233.0
relaxation              82.0   146.0
fasting blood sugar    104.0   423.0
Cholesterol            220.0   445.0
triglyceride           160.0   999.0
HDL                     66.0   359.0
LDL                    136.0  1860.0
hemoglobin              15.7    21.1
Urine protein            1.0     6.0
serum creatinine         1.0    11.6
AST                     29.0  1090.0
ALT                     31.0  2914.0
Gtp                     44.0   999.0
dental caries            0.0     1.0
smoking                  1.0     1.0
```

[ ]:
```python
# Target check
target_col = "smoking"
if target_col not in df.columns:
    raise ValueError(f"Target column '{target_col}' not found in dataset␣
 ↪columns: {df.columns.tolist()}")

print("\n--- Target distribution ---")
print(df[target_col].value_counts(dropna=False))
display(df[target_col].value_counts(normalize=True).rename("proportion").
 ↪to_frame())

# If smoking is not binary, map it (common cases:␣
 ↪'Non-smoker','Ex-smoker','Current')
unique_vals = df[target_col].unique()
print("\nUnique target values:", unique_vals)

# Simple histogram for numeric features (medium-sized)
num_cols = df.select_dtypes(include=["int64","float64"]).columns.tolist()
num_cols = [c for c in num_cols if c != target_col]
print("\nNumerical columns:", num_cols)
```

```
--- Target distribution ---
smoking
0    21209
1    12258
```

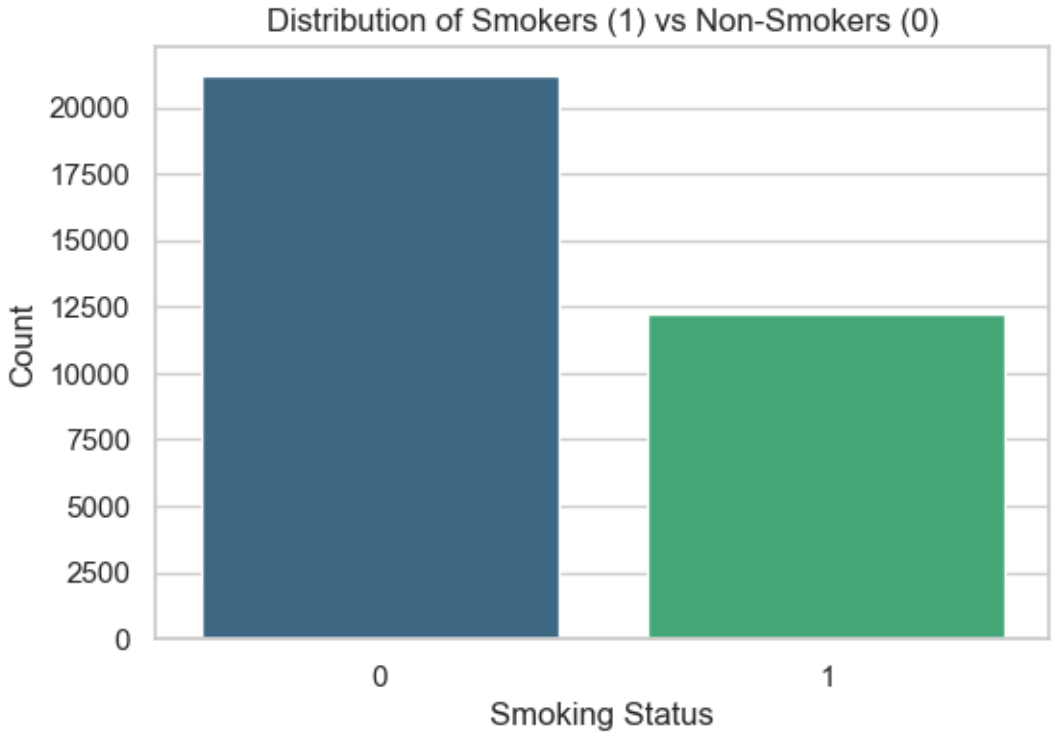
```
Name: count, dtype: int64
```

```
          proportion
smoking
0           0.633729
1           0.366271
```

```
Unique target values: [1 0]
```

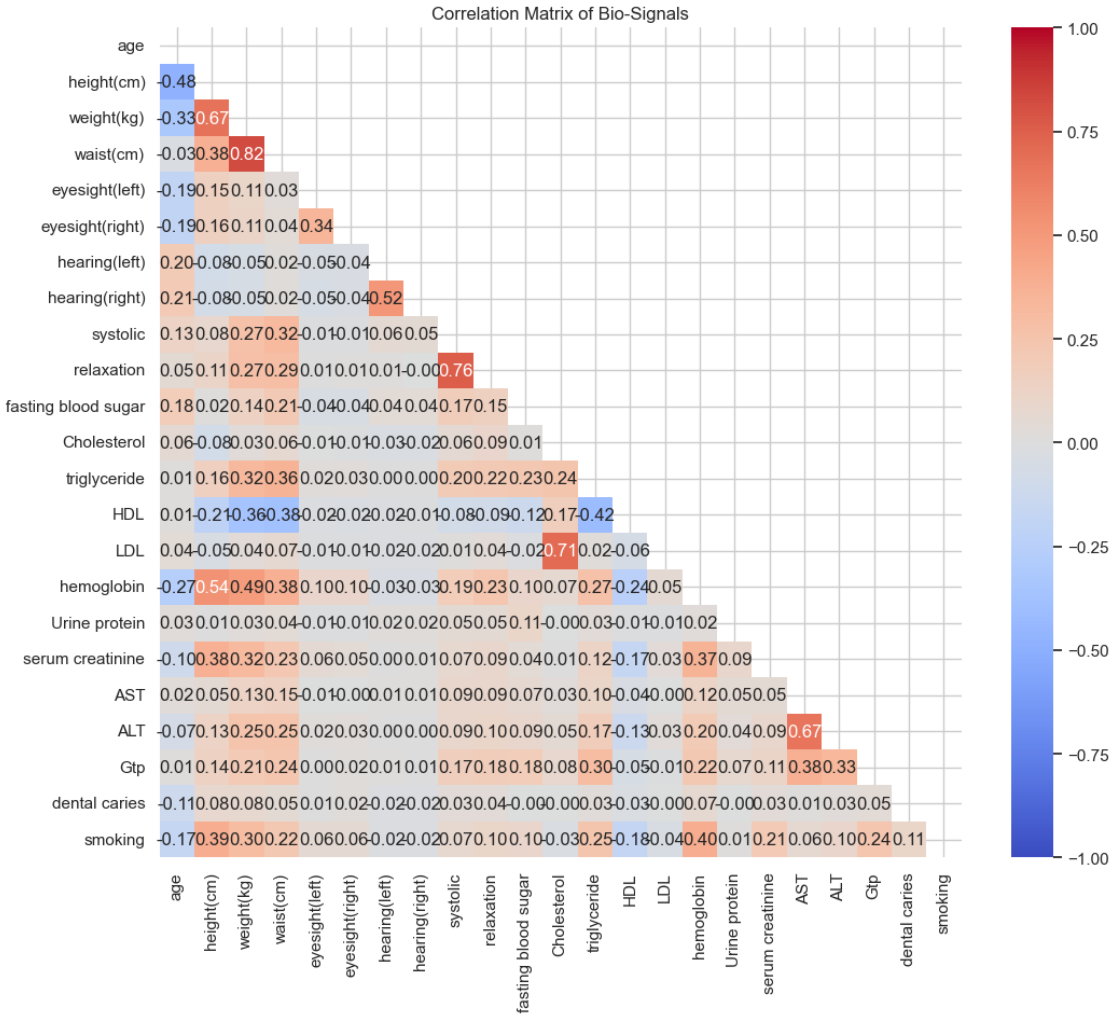
```
Numerical columns: ['age', 'height(cm)', 'weight(kg)', 'waist(cm)',
'eyesight(left)', 'eyesight(right)', 'hearing(left)', 'hearing(right)',
'systolic', 'relaxation', 'fasting blood sugar', 'Cholesterol', 'triglyceride',
'HDL', 'LDL', 'hemoglobin', 'Urine protein', 'serum creatinine', 'AST', 'ALT',
'Gtp', 'dental caries']
```

[ ]:
```python
# 2. Class Balance Analysis
plt.figure(figsize=(6, 4))
sns.countplot(x='smoking', data=df, palette='viridis')
plt.title('Distribution of Smokers (1) vs Non-Smokers (0)')
plt.xlabel('Smoking Status')
plt.ylabel('Count')
plt.show()

ratio = df['smoking'].value_counts(normalize=True)
print(f"Class Ratio:\n{ratio}")
```

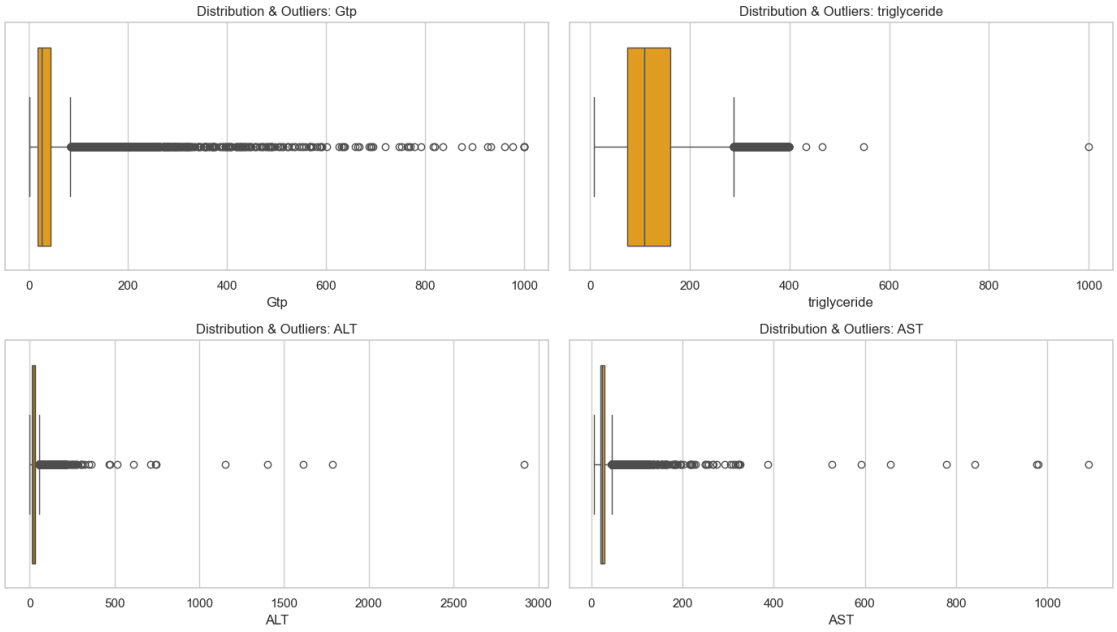


```
Class Ratio:
smoking
0    0.633729
1    0.366271
Name: proportion, dtype: float64
```

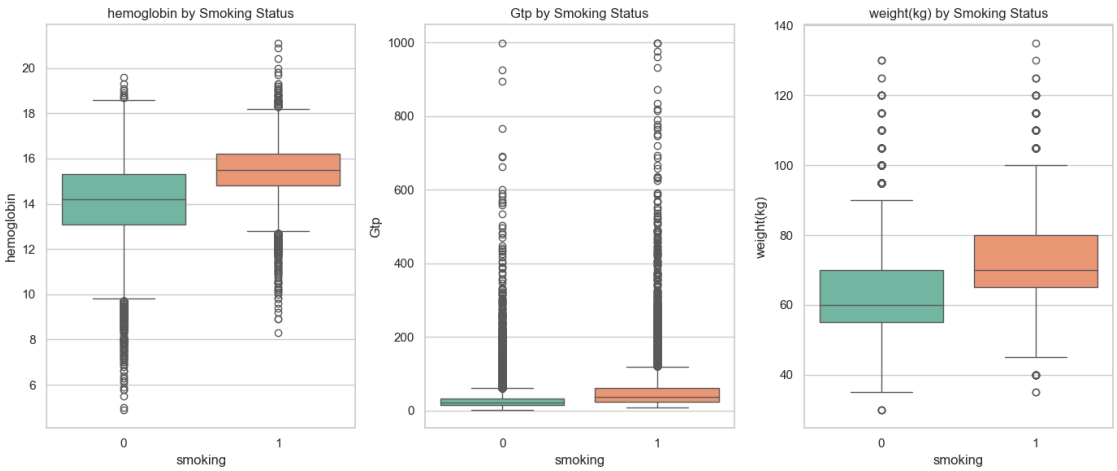
[ ]:
```
# --- A. Correlation Heatmap ---
plt.figure(figsize=(12, 10))

# Calculate correlation only on numeric columns
numeric_df = df.select_dtypes(include=[np.number])
corr = numeric_df.corr()

# Create a mask to hide the upper triangle (it's symmetrical/redundant)
mask = np.triu(np.ones_like(corr, dtype=bool))

# Plot heatmap
sns.heatmap(corr, mask=mask, annot=True, fmt=".2f", cmap='coolwarm', vmin=-1,␣
 ↪vmax=1)
plt.title('Correlation Matrix of Bio-Signals')
plt.show()
```



[ ]:
```
# --- B. Outlier Visualization (Boxplots) ---
# We focus on features known to have extreme values in this dataset
outlier_features = ['Gtp', 'triglyceride', 'ALT', 'AST']

plt.figure(figsize=(14, 8))
for i, col in enumerate(outlier_features):
    plt.subplot(2, 2, i+1)
    sns.boxplot(x=df[col], color='orange')
    plt.title(f'Distribution & Outliers: {col}')
plt.tight_layout()
plt.show()
```



[ ]:
```
# --- C. Compare "Signals" (Smoker vs Non-Smoker) ---
# This proves if the data actually contains predictive signals
signal_features = ['hemoglobin', 'Gtp', 'weight(kg)']

plt.figure(figsize=(14, 6))
for i, col in enumerate(signal_features):
    plt.subplot(1, 3, i+1)
    sns.boxplot(x='smoking', y=col, data=df, palette='Set2')
    plt.title(f'{col} by Smoking Status')
plt.tight_layout()
plt.show()
```



[ ]:
```
# --- 2. Feature Engineering (Crucial for High Accuracy) ---
# Create BMI (Standard medical metric better than raw height/weight)
# Height is in cm, so convert to meters
df['BMI'] = df['weight(kg)'] / ((df['height(cm)'] / 100) ** 2)

# Create Pulse Pressure (Captures the gap between systolic/relaxation)
df['Pulse_Pressure'] = df['systolic'] - df['relaxation']

# Drop highly correlated/redundant columns to help Logistic Regression
# We drop 'waist(cm)' because it correlates 0.82 with weight
# We drop 'systolic' and 'relaxation' because we captured their relationship in␣
 ↪Pulse_Pressure
drop_cols = ['waist(cm)', 'systolic', 'relaxation']
df_clean = df.drop(columns=drop_cols)

print(f"New Feature Set: {df_clean.columns.tolist()}")
print(f"Total columns: {len(df_clean.columns)}")
```

```
New Feature Set: ['age', 'height(cm)', 'weight(kg)', 'eyesight(left)',
'eyesight(right)', 'hearing(left)', 'hearing(right)', 'fasting blood sugar',
'Cholesterol', 'triglyceride', 'HDL', 'LDL', 'hemoglobin', 'Urine protein',
'serum creatinine', 'AST', 'ALT', 'Gtp', 'dental caries', 'smoking', 'BMI',
'Pulse_Pressure']
Total columns: 22
```

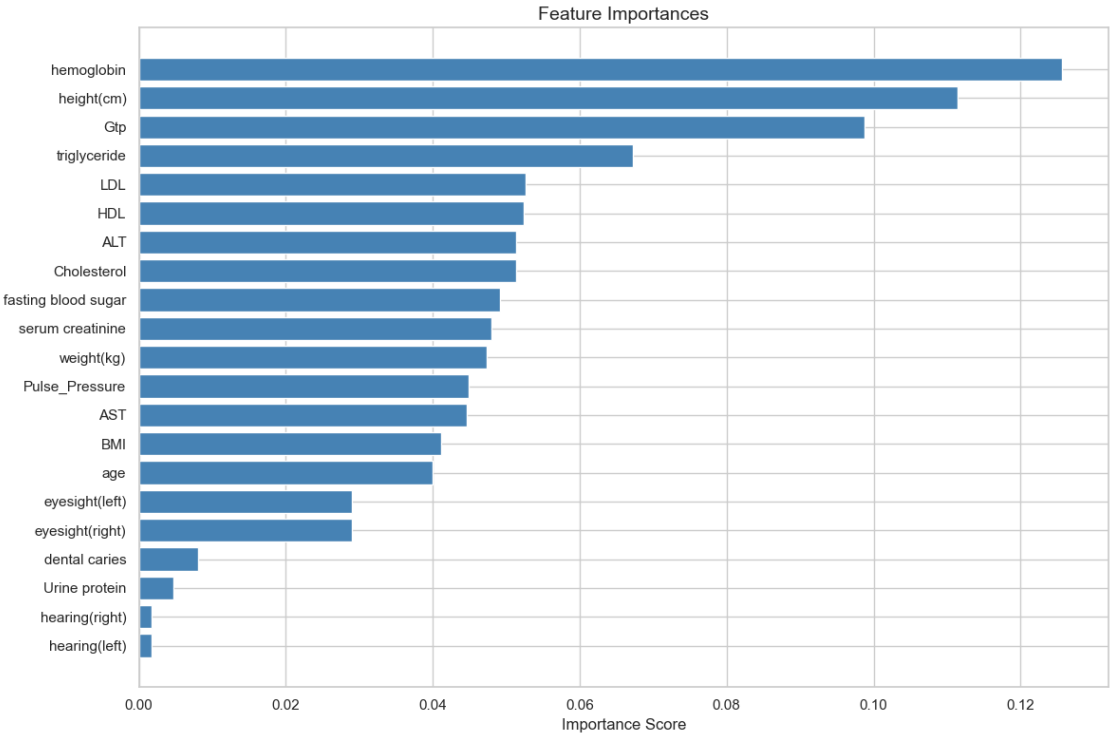
[ ]:
```
# --- 3. Outlier Handling (Log Transformation) ---
# The outliers in Gtp, ALT, and AST were massive. Log transform pulls them in.
# We add +1 to avoid log(0) errors
outlier_cols = ['Gtp', 'ALT', 'AST', 'triglyceride']
for col in outlier_cols:
    df_clean[col] = np.log1p(df_clean[col])
```

[ ]:
```
# --- 4. Train/Test Split ---
X = df_clean.drop('smoking', axis=1)
y = df_clean['smoking']

# Stratify ensures we keep the same smoker/non-smoker ratio in test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# --- 5. Scaling ---
# We use StandardScaler now that we have fixed the outliers with Log transform
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[31]:
```
# --- Feature Importance Analysis ---
# This helps identify which features are most predictive for smoking status

from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance

# Train a quick Random Forest to get feature importances
print("Training Random Forest for feature importance analysis...")
rf_temp = RandomForestClassifier(n_estimators=100, random_state=RANDOM_STATE,␣
 ↪class_weight='balanced')
rf_temp.fit(X_train_scaled, y_train)

# Get feature importances
feature_names = X_train.columns
importances = rf_temp.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(12, 8))
plt.title("Feature Importances", fontsize=14)
plt.barh(range(len(importances)), importances[indices], color='steelblue')
plt.yticks(range(len(importances)), [feature_names[i] for i in indices])
plt.xlabel('Importance Score')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

# Print top 10 most important features
print("\nTop 10 Most Important Features:")
for i in range(min(10, len(indices))):
    print(f"{i+1}. {feature_names[indices[i]]}: {importances[indices[i]]:.4f}")

# Permutation importance (more reliable, but slower)
print("\nComputing permutation importance (this may take a minute)...")
perm_importance = permutation_importance(rf_temp, X_test_scaled, y_test,␣
 ↪n_repeats=10,
                                         random_state=RANDOM_STATE, n_jobs=-1)

# Plot permutation importance
perm_indices = np.argsort(perm_importance.importances_mean)[::-1]
plt.figure(figsize=(12, 8))
plt.title("Permutation Feature Importances", fontsize=14)
plt.barh(range(len(perm_importance.importances_mean)),
         perm_importance.importances_mean[perm_indices],
         xerr=perm_importance.importances_std[perm_indices],
         color='coral')
plt.yticks(range(len(perm_indices)), [feature_names[i] for i in perm_indices])
```

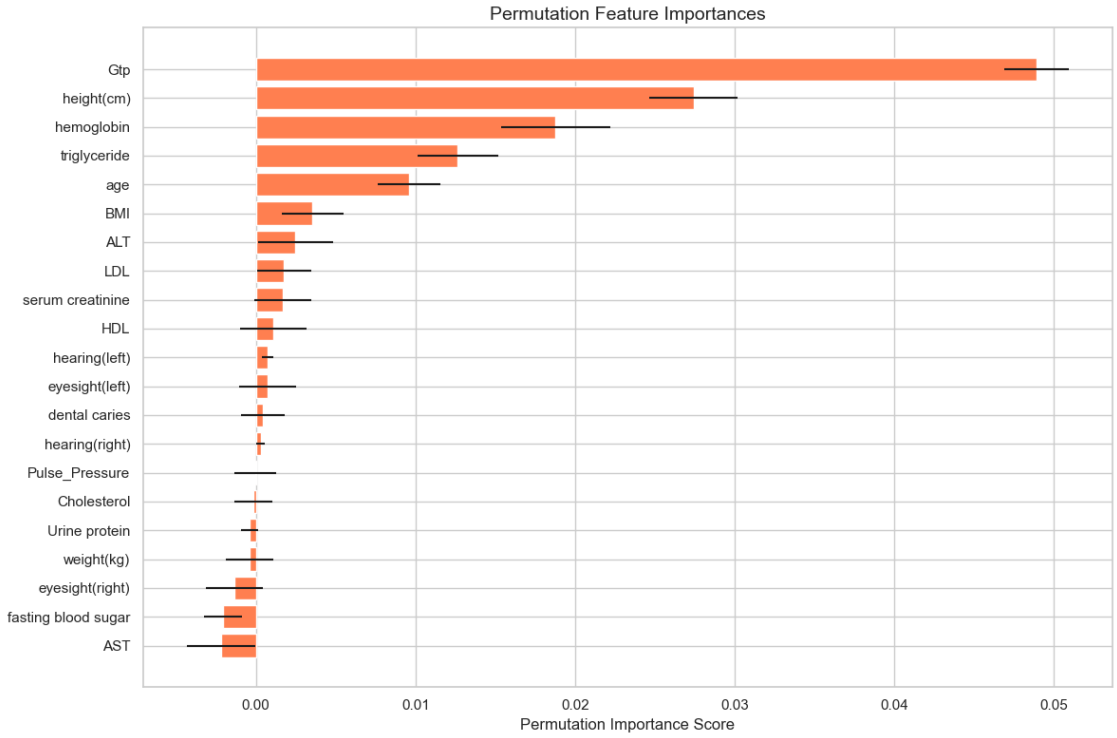
```
plt.xlabel('Permutation Importance Score')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

print("\nTop 10 Features by Permutation Importance:")
for i in range(min(10, len(perm_indices))):
    idx = perm_indices[i]
    print(f"{i+1}. {feature_names[idx]}: {perm_importance.importances_mean[idx]:
 ↪.4f} "
          f"(±{perm_importance.importances_std[idx]:.4f})")

# Optional: You can use this information to select top features if needed
top_features = [feature_names[i] for i in perm_indices[:15]]  # Top 15 features
print(f"\nSuggested top features: {top_features}")
```

```
Training Random Forest for feature importance analysis…
```




```
Top 10 Most Important Features:
1. hemoglobin: 0.1257
2. height(cm): 0.1114
3. Gtp: 0.0988
4. triglyceride: 0.0672
```

```
5. LDL: 0.0527
6. HDL: 0.0524
7. ALT: 0.0514
8. Cholesterol: 0.0513
9. fasting blood sugar: 0.0491
10. serum creatinine: 0.0480

Computing permutation importance (this may take a minute)…
```




```
Top 10 Features by Permutation Importance:
1. Gtp: 0.0489 (±0.0020)
2. height(cm): 0.0274 (±0.0028)
3. hemoglobin: 0.0188 (±0.0034)
4. triglyceride: 0.0127 (±0.0025)
5. age: 0.0096 (±0.0020)
6. BMI: 0.0036 (±0.0019)
7. ALT: 0.0025 (±0.0023)
8. LDL: 0.0018 (±0.0017)
9. serum creatinine: 0.0017 (±0.0018)
10. HDL: 0.0011 (±0.0021)

Suggested top features: ['Gtp', 'height(cm)', 'hemoglobin', 'triglyceride',
'age', 'BMI', 'ALT', 'LDL', 'serum creatinine', 'HDL', 'hearing(left)',
```

```
'eyesight(left)', 'dental caries', 'hearing(right)', 'Pulse_Pressure']
```

[32]:
```
# --- Logistic Regression ---
print("Training Logistic Regression...")

# Balance Data
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

# Grid Search
param_grid = {'C': [0.1, 1, 10], 'solver': ['liblinear', 'lbfgs']}
grid_lr = GridSearchCV(
    LogisticRegression(max_iter=3000),
    param_grid, cv=5, scoring='accuracy'
)
grid_lr.fit(X_train_smote, y_train_smote)

best_lr = grid_lr.best_estimator_
acc_lr = accuracy_score(y_test, best_lr.predict(X_test_scaled))

print(f"Best Params: {grid_lr.best_params_}")
print(f"LogReg Accuracy: {acc_lr:.4f}")
```

```
Training Logistic Regression…
Best Params: {'C': 10, 'solver': 'liblinear'}
LogReg Accuracy: 0.7257
```

[33]:
```
# ------------------ SVM with RandomizedSearchCV (FAST) ------------------
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils import resample

print("Applying SMOTE to training data for SVM (fast version)...")
smote_svm_fast = SMOTE(random_state=RANDOM_STATE)
X_train_svm_smote_fast, y_train_svm_smote_fast = smote_svm_fast.
 ↪fit_resample(X_train_scaled, y_train)
print(f"Training shape after SMOTE: {X_train_svm_smote_fast.shape}")

# Subsample for faster tuning
print("\nSampling 50% of SMOTE data for tuning...")
X_tune, y_tune = resample(
    X_train_svm_smote_fast, y_train_svm_smote_fast,
    n_samples=len(X_train_svm_smote_fast) // 2,
    random_state=RANDOM_STATE,
    stratify=y_train_svm_smote_fast,
)
print(f"Tuning on sample: {X_tune.shape}")
```

```
# Lean search space
param_grid_svm_fast = {
    'C': [0.1, 1, 10, 50],
    'gamma': ['scale', 0.01, 0.001],
    'kernel': ['rbf']
}

# RandomizedSearchCV for speed
random_svm = RandomizedSearchCV(
    SVC(probability=True, random_state=RANDOM_STATE, class_weight='balanced'),
    param_distributions=param_grid_svm_fast,
    n_iter=12,
    scoring='roc_auc',
    cv=3,
    n_jobs=-1,
    verbose=1,
    random_state=RANDOM_STATE,
)
random_svm.fit(X_tune, y_tune)

print("\nBest SVM Parameters (sample tuning):")
print(random_svm.best_params_)
print(f"Best CV ROC AUC (sample): {random_svm.best_score_:.4f}")

# Retrain on full SMOTE data
print("\nRetraining best model on FULL SMOTE data...")
best_svm = SVC(probability=True, random_state=RANDOM_STATE,␣
 ↪class_weight='balanced',
               **random_svm.best_params_)
best_svm.fit(X_train_svm_smote_fast, y_train_svm_smote_fast)
print("Final SVM trained on full data.")

# Evaluate
y_pred_svm = best_svm.predict(X_test_scaled)
y_proba_svm = best_svm.predict_proba(X_test_scaled)[:, 1]

print("\n--- SVM Results (FAST) ---")
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}")
print(f"ROC AUC: {roc_auc_score(y_test, y_proba_svm):.4f}")
print(classification_report(y_test, y_pred_svm))

# Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d',␣
 ↪cmap='Greens')
plt.title(f'SVM Confusion Matrix (Acc={accuracy_score(y_test, y_pred_svm):.
 ↪4f})')
```

```
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# ROC Curve
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_proba_svm)
plt.figure(figsize=(6, 5))
plt.plot(fpr_svm, tpr_svm, label=f'ROC AUC = {roc_auc_score(y_test,␣
 ↪y_proba_svm):.4f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('SVM ROC Curve (FAST)')
plt.legend()
plt.show()
```

```
Applying SMOTE to training data for SVM (fast version)…
Training shape after SMOTE: (33934, 21)

Sampling 50% of SMOTE data for tuning…
Tuning on sample: (16967, 21)
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[33], line 39
     28 # RandomizedSearchCV for speed
     29 random_svm = RandomizedSearchCV(
     30     SVC(probability=True, random_state=RANDOM_STATE,␣
 ↪class_weight='balanced'),
     31     param_distributions=param_grid_svm_fast,
   (…)     37     random_state=RANDOM_STATE,
     38 )
---> 39 random_svm.fit(X_tune, y_tune)
     41 print("\nBest SVM Parameters (sample tuning):")
     42 print(random_svm.best_params_)

File d:\IIITB\Sem-1\Machine Learning\ML-2\.venv\Lib\site-packages\sklearn\base.
 ↪py:1365, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args␣
 ↪**kwargs)
   1358     estimator._validate_params()
   1360 with config_context(
   1361     skip_parameter_validation=(
   1362         prefer_skip_nested_validation or global_skip_validation
   1363     )
   1364 ):
-> 1365     return fit_method(estimator, *args, **kwargs)
```

```
File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\sklearn\model_selection\_search.py:1051, in␣
 ↪BaseSearchCV.fit(self, X, y, **params)
   1045     results = self._format_results(
   1046         all_candidate_params, n_splits, all_out, all_more_results
   1047     )
   1049     return results
-> 1051 self._run_search(evaluate_candidates)
   1053 # multimetric is determined here because in the case of a callable
   1054 # self.scoring the return type is only known after calling
   1055 first_test_score = all_out[0]["test_scores"]

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\sklearn\model_selection\_search.py:1992, in␣
 ↪RandomizedSearchCV._run_search(self, evaluate_candidates)
   1990 def _run_search(self, evaluate_candidates):
   1991     """Search n_iter candidates from param_distributions"""
-> 1992     evaluate_candidates(
   1993         ParameterSampler(
   1994␣
 ↪            self.param_distributions, self.n_iter, random_state=self.random_state
   1995         )
   1996     )

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\sklearn\model_selection\_search.py:997, in BaseSearchCV.
 ↪fit.<locals>.evaluate_candidates(candidate_params, cv, more_results)
    989 if self.verbose > 0:
    990     print(
    991         "Fitting {0} folds for each of {1} candidates,"
    992         " totalling {2} fits".format(
    993             n_splits, n_candidates, n_candidates * n_splits
    994         )
    995     )
--> 997 out = parallel(
    998     delayed(_fit_and_score)(
    999         clone(base_estimator),
   1000         X,
   1001         y,
   1002         train=train,
   1003         test=test,
   1004         parameters=parameters,
   1005         split_progress=(split_idx, n_splits),
   1006         candidate_progress=(cand_idx, n_candidates),
   1007         **fit_and_score_kwargs,
   1008     )
   1009     for (cand_idx, parameters), (split_idx, (train, test)) in product(
   1010         enumerate(candidate_params),
```

```
   1011         enumerate(cv.split(X, y, **routed_params.splitter.split)),
   1012     )
   1013 )
   1015 if len(out) < 1:
   1016     raise ValueError(
   1017         "No fits were performed. "
   1018         "Was the CV iterator empty? "
   1019         "Were there no candidates?"
   1020     )

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\sklearn\utils\parallel.py:82, in Parallel.
 ↪__call__(self, iterable)
     73 warning_filters = warnings.filters
     74 iterable_with_config_and_warning_filters = (
     75     (
     76         _with_config_and_warning_filters(delayed_func, config,␣
 ↪warning_filters),
   (…)     80     for delayed_func, args, kwargs in iterable
     81 )
---> 82 return super().__call__(iterable_with_config_and_warning_filters)

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\joblib\parallel.py:2072, in Parallel.__call__(self,␣
 ↪iterable)
   2066 # The first item from the output is blank, but it makes the interpreter
   2067 # progress until it enters the Try/Except block of the generator and
   2068 # reaches the first `yield` statement. This starts the asynchronous
   2069 # dispatch of the tasks to the workers.
   2070 next(output)
-> 2072 return output if self.return_generator else list(output)

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\joblib\parallel.py:1682, in Parallel._get_outputs(self␣
 ↪iterator, pre_dispatch)
   1679     yield
   1681     with self._backend.retrieval_context():
-> 1682         yield from self._retrieve()
   1684 except GeneratorExit:
   1685     # The generator has been garbage collected before being fully
   1686     # consumed. This aborts the remaining tasks if possible and warn
   1687     # the user if necessary.
   1688     self._exception = True

File d:\IIITB\Sem-1\Machine Learning\ML-2\.
 ↪venv\Lib\site-packages\joblib\parallel.py:1800, in Parallel._retrieve(self)
   1789 if self.return_ordered:
   1790     # Case ordered: wait for completion (or error) of the next job
```

```
   1791     # that have been dispatched and not retrieved yet. If no job
   (…)   1795     # control only have to be done on the amount of time the next
   1796     # dispatched job is pending.
   1797     if (nb_jobs == 0) or (
   1798         self._jobs[0].get_status(timeout=self.timeout) == TASK_PENDING
   1799     ):
-> 1800         time.sleep(0.01)
   1801         continue
   1803 elif nb_jobs == 0:
   1804     # Case unordered: jobs are added to the list of jobs to
   1805     # retrieve `self._jobs` only once completed or in error, which
   (…)   1811     # timeouts before any other dispatched job has completed and
   1812     # been added to `self._jobs` to be retrieved.

KeyboardInterrupt:
```

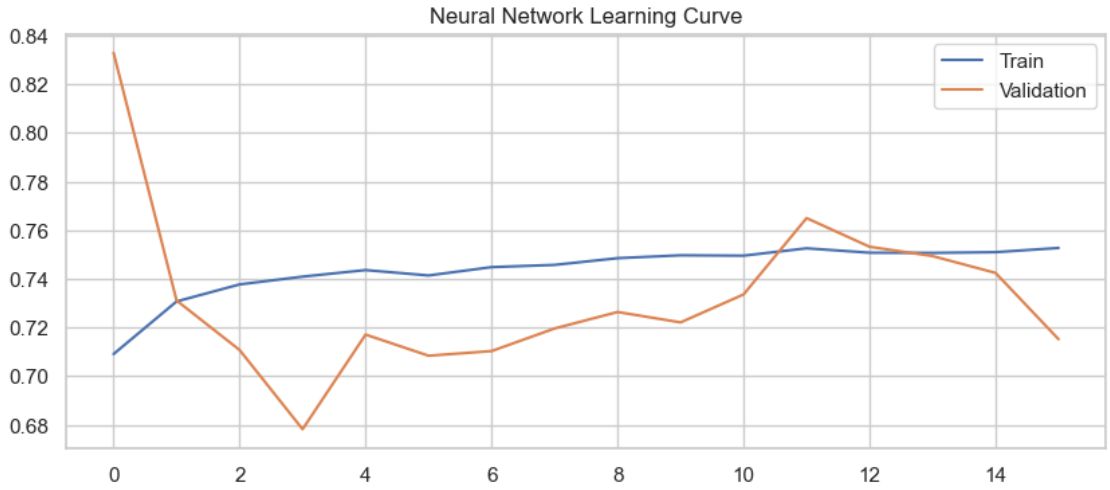
[35]:
```python
# --- Cell 6: Neural Network ---
print("Training Neural Network...")

# 1. Balance Data for NN
smote = SMOTE(random_state=42)
X_train_nn, y_train_nn = smote.fit_resample(X_train_scaled, y_train)

# 2. Build Model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_nn.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),

    Dense(32, activation='relu'),
    BatchNormalization(),

    Dense(1, activation='sigmoid')
])

# 3. Compile
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```python
# 4. Callbacks (Auto-Tuning)
callbacks = [
    EarlyStopping(monitor='val_loss', patience=15, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5, verbose=1)
]

# 5. Train
history = model.fit(
    X_train_nn, y_train_nn,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

# 6. Evaluate
loss, acc_nn = model.evaluate(X_test_scaled, y_test, verbose=0)
print(f"Neural Network Accuracy: {acc_nn:.4f}")

# Plot History
plt.figure(figsize=(10, 4))
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.title("Neural Network Learning Curve")
plt.legend()
plt.show()
```

```
Training Neural Network…
Epoch 1/100
849/849             4s 2ms/step -
accuracy: 0.7091 - loss: 0.5474 - val_accuracy: 0.8329 - val_loss: 0.5341 -
learning_rate: 0.0010
Epoch 2/100
849/849             2s 2ms/step -
accuracy: 0.7307 - loss: 0.5042 - val_accuracy: 0.7311 - val_loss: 0.6035 -
learning_rate: 0.0010
Epoch 3/100
849/849             2s 2ms/step -
accuracy: 0.7377 - loss: 0.4953 - val_accuracy: 0.7108 - val_loss: 0.6280 -
learning_rate: 0.0010
Epoch 4/100
849/849             2s 2ms/step -
accuracy: 0.7409 - loss: 0.4915 - val_accuracy: 0.6782 - val_loss: 0.6372 -
learning_rate: 0.0010
Epoch 5/100
849/849             2s 2ms/step -
accuracy: 0.7436 - loss: 0.4893 - val_accuracy: 0.7171 - val_loss: 0.6121 -
```

```
learning_rate: 0.0010
Epoch 6/100
825/849             0s 2ms/step -
accuracy: 0.7411 - loss: 0.4895
Epoch 6: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
849/849             2s 2ms/step -
accuracy: 0.7414 - loss: 0.4895 - val_accuracy: 0.7084 - val_loss: 0.6186 -
learning_rate: 0.0010
Epoch 7/100
849/849             2s 2ms/step -
accuracy: 0.7448 - loss: 0.4831 - val_accuracy: 0.7103 - val_loss: 0.6096 -
learning_rate: 5.0000e-04
Epoch 8/100
849/849             2s 2ms/step -
accuracy: 0.7458 - loss: 0.4826 - val_accuracy: 0.7196 - val_loss: 0.6061 -
learning_rate: 5.0000e-04
Epoch 9/100
849/849             2s 2ms/step -
accuracy: 0.7485 - loss: 0.4815 - val_accuracy: 0.7264 - val_loss: 0.5934 -
learning_rate: 5.0000e-04
Epoch 10/100
849/849             2s 2ms/step -
accuracy: 0.7497 - loss: 0.4827 - val_accuracy: 0.7221 - val_loss: 0.6079 -
learning_rate: 5.0000e-04
Epoch 11/100
828/849             0s 2ms/step -
accuracy: 0.7544 - loss: 0.4741
Epoch 11: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
849/849             2s 2ms/step -
accuracy: 0.7495 - loss: 0.4799 - val_accuracy: 0.7336 - val_loss: 0.5902 -
learning_rate: 5.0000e-04
Epoch 12/100
849/849             2s 2ms/step -
accuracy: 0.7526 - loss: 0.4806 - val_accuracy: 0.7650 - val_loss: 0.5748 -
learning_rate: 2.5000e-04
Epoch 13/100
849/849             2s 2ms/step -
accuracy: 0.7508 - loss: 0.4772 - val_accuracy: 0.7532 - val_loss: 0.5803 -
learning_rate: 2.5000e-04
Epoch 14/100
849/849             2s 2ms/step -
accuracy: 0.7507 - loss: 0.4791 - val_accuracy: 0.7494 - val_loss: 0.5858 -
learning_rate: 2.5000e-04
Epoch 15/100
849/849             2s 2ms/step -
accuracy: 0.7510 - loss: 0.4781 - val_accuracy: 0.7424 - val_loss: 0.5899 -
learning_rate: 2.5000e-04
Epoch 16/100
```

```
829/849              0s 2ms/step -
accuracy: 0.7558 - loss: 0.4723
Epoch 16: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
849/849              2s 2ms/step -
accuracy: 0.7527 - loss: 0.4787 - val_accuracy: 0.7152 - val_loss: 0.6020 -
learning_rate: 2.5000e-04
Neural Network Accuracy: 0.7345
```



[36]:
```python
def smooth(values, weight=0.85):
    """Smooth curves for cleaner visualization."""
    smoothed = []
    last = values[0]
    for v in values:
        smoothed_val = last * weight + (1 - weight) * v
        smoothed.append(smoothed_val)
        last = smoothed_val
    return smoothed

acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]
loss = history.history["loss"]
val_loss = history.history["val_loss"]

# Smooth curves
acc_s = smooth(acc)
val_acc_s = smooth(val_acc)
loss_s = smooth(loss)
val_loss_s = smooth(val_loss)

plt.figure(figsize=(14, 6), dpi=120)
```

```python
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(acc_s, label="Train Accuracy", linewidth=2)
plt.plot(val_acc_s, label="Validation Accuracy", linewidth=2)
plt.title("Neural Network Accuracy Curve", fontsize=14)
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Accuracy", fontsize=12)
plt.grid(alpha=0.3)
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(loss_s, label="Train Loss", linewidth=2)
plt.plot(val_loss_s, label="Validation Loss", linewidth=2)
plt.title("Neural Network Loss Curve", fontsize=14)
plt.xlabel("Epoch", fontsize=12)
plt.ylabel("Binary Crossentropy", fontsize=12)
plt.grid(alpha=0.3)
plt.legend()

plt.suptitle("Training Curves for Improved Deep Neural Network", fontsize=16)
plt.tight_layout()
plt.show()
```