In [25]:
```
import sys
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# sklearn imports
from sklearn.model_selection import (
    train_test_split,
    GridSearchCV,
    StratifiedKFold,
    cross_val_score
)
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    roc_auc_score,
    roc_curve,
    precision_recall_curve,
    average_precision_score,
    precision_score,
    recall_score,
    f1_score
)

# Advanced models
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Explainability
import shap

# Set random seed for reproducibility
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

# ==========================================
# 1. DATA LOADING
# ==========================================
print(">>> STEP 1: Loading Data...")

file_path = r"C:\Users\admin\Desktop\diabetic_data.csv"

if not os.path.exists(file_path):
    raise FileNotFoundError(f"Error: '{file_path}' not found.")

# Load CSV, treat '?' as missing values
df = pd.read_csv(file_path, na_values="?", low_memory=False)
```

```
# Basic cleaning: remove duplicate rows
df = df.drop_duplicates()

print(f"Data loaded successfully.")
print(f"Shape after dropping duplicates: {df.shape}")
```

```
>>> STEP 1: Loading Data...
Data loaded successfully.
Shape after dropping duplicates: (101766, 50)
```

In [26]:
```
# ==========================================
# 1B. EXCLUDE EXPIRED AND HOSPICE PATIENTS
# ==========================================
print("\n>>> STEP 1B: Excluding Expired and Hospice patients...")
exclude_discharge_ids = [11, 13, 14, 19, 20, 21]
before_count = len(df)
df = df[~df['discharge_disposition_id'].isin(exclude_discharge_ids)]
after_count = len(df)
print(f"  Excluded {before_count - after_count} samples (Expired/Hospice)")
print(f"  Remaining samples: {after_count}")
```

```
>>> STEP 1B: Excluding Expired and Hospice patients...
  Excluded 2423 samples (Expired/Hospice)
  Remaining samples: 99343
```

In [27]:
```
# ==========================================
# 2. FEATURE ENGINEERING
# ==========================================
print("\n>>> STEP 2: Performing Feature Engineering...")

# ==========================================
# 2A. ICD-9 CODE FEATURE EXTRACTION
# ==========================================
diag_cols = ["diag_1", "diag_2", "diag_3"]

def normalize_icd(code):
    """Normalize ICD-9 code into a clean string (None if invalid)."""
    if pd.isna(code):
        return None
    c = str(code).strip()
    if c == "" or c.lower() == "nan":
        return None
    return c

def build_diag_list(row):
    """Build a normalized list of ICD-9 codes for a given row."""
    return [normalize_icd(row[c]) for c in diag_cols]

def has_icd_any(diag_list, icd_set):
    """
    Check if any ICD-9 code in diag_list matches icd_set.
    Match is based on exact code or main part before dot.
    """
    for code in diag_list:
        if code is None:
            continue
        main = code.split(".")[0]
        if code in icd_set or main in icd_set:
            return 1
    return 0
```

```
# === ICD-9 Code Sets (per project instructions) ===
alcohol_codes = {"303", "305", "571", "571.1", "571.2", "571.3"}
bp_codes = {"401", "402", "403", "404", "405", "642"}
cholesterol_codes = {"272"}
heart_codes = {"410", "411", "412", "413", "414", "428"}
obesity_codes = {"278"}
pregnancy_codes = {
    "630", "631", "632", "633", "634", "635", "636", "637", "638", "639",
    "640", "641", "642", "643", "644", "645", "646", "647", "648", "649"
}
uric_acid_codes = {"274", "790.6", "790"}
vision_codes = {"368"}

# Precompute each row's diagnosis list
diag_lists = df.apply(build_diag_list, axis=1)

# Create binary features from ICD-9 codes
df["alcohol"] = diag_lists.apply(lambda dl: has_icd_any(dl, alcohol_codes))
df["blood_pressure"] = diag_lists.apply(lambda dl: has_icd_any(dl, bp_codes))
df["cholesterol"] = diag_lists.apply(lambda dl: has_icd_any(dl, cholesterol_code
df["heart_disease"] = diag_lists.apply(lambda dl: has_icd_any(dl, heart_codes))
df["obesity"] = diag_lists.apply(lambda dl: has_icd_any(dl, obesity_codes))
df["pregnancy"] = diag_lists.apply(lambda dl: has_icd_any(dl, pregnancy_codes))
df["uric_acid"] = diag_lists.apply(lambda dl: has_icd_any(dl, uric_acid_codes))
df["blurred_vision"] = diag_lists.apply(lambda dl: has_icd_any(dl, vision_codes)

print("ICD-9 based clinical features created:")
print(f"  - alcohol:        {df['alcohol'].sum()} positive cases")
print(f"  - blood_pressure: {df['blood_pressure'].sum()} positive cases")
print(f"  - cholesterol:    {df['cholesterol'].sum()} positive cases")
print(f"  - heart_disease:  {df['heart_disease'].sum()} positive cases")
print(f"  - obesity:        {df['obesity'].sum()} positive cases")
print(f"  - pregnancy:      {df['pregnancy'].sum()} positive cases")
print(f"  - uric_acid:      {df['uric_acid'].sum()} positive cases")
print(f"  - blurred_vision: {df['blurred_vision'].sum()} positive cases")
```

```
>>> STEP 2: Performing Feature Engineering...
ICD-9 based clinical features created:
  - alcohol:        2944 positive cases
  - blood_pressure: 19667 positive cases
  - cholesterol:    2384 positive cases
  - heart_disease:  29861 positive cases
  - obesity:        1324 positive cases
  - pregnancy:      677 positive cases
  - uric_acid:      745 positive cases
  - blurred_vision: 65 positive cases
```

In [28]:

```
# ==========================================
# 3. TARGET VARIABLE CREATION (Diabetes Type 2)
# ==========================================
print("\n>>> STEP 3: Creating Target Variable 'diabetes2'...")

def is_type2_diabetes_code(code):
    """
    Check if an ICD-9 code is Type 2 diabetes pattern: 250.x0 or 250.x2.
    Note: 250.x0 is equivalent to 250.x (trailing zero removed)
          250 alone is equivalent to 250.00 (Type 2)
    """
    code = normalize_icd(code)
```

```
    if code is None:
        return False

    # Case 1: exactly "250" -> equivalent to 250.00 (Type 2)
    if code == "250":
        return True

    if not code.startswith("250."):
        return False

    tail = code.split(".")[1]

    # Case 2: 250.x (one digit) -> equivalent to 250.x0 (Type 2)
    if len(tail) == 1:
        return True

    # Case 3: 250.xy (two digits) -> Type 2 if y is 0 or 2
    if len(tail) >= 2:
        return tail[1] in {"0", "2"}

    return False

def is_diabetes_250_any(code):
    """
    Check if code is 250 or any 250.x pattern.
    """
    code = normalize_icd(code)
    if code is None:
        return False
    return code == "250" or code.startswith("250.")

def define_target(row):
    """
    Define Type 2 diabetes target based on project rules:
    - Condition 1: At least one diagnosis is 250.x0 or 250.x2.
    - OR Condition 2: Diagnosis includes 250 or 250.x, AND
      at least one diabetes medication (metformin, glimepiride, glipizide) is pr

    Medication is considered prescribed if value is 'Up', 'Down', or 'Steady' (n
    """
    diags = [row[c] for c in diag_cols]

    # Diabetes meds of interest (prescribed if not "No")
    meds = [row["metformin"], row["glimepiride"], row["glipizide"]]
    meds_prescribed = any(m in ["Up", "Down", "Steady"] for m in meds if pd.notn

    # Condition 1: Direct Type 2 code
    if any(is_type2_diabetes_code(d) for d in diags):
        return 1

    # Condition 2: Any diabetes code + medication prescribed
    if meds_prescribed and any(is_diabetes_250_any(d) for d in diags):
        return 1

    return 0

df["diabetes2"] = df.apply(define_target, axis=1)

# Summary statistics
total_positive = df["diabetes2"].sum()
```

```
total_negative = len(df) - total_positive
prevalence = total_positive / len(df) * 100

print(f"Target variable 'diabetes2' created.")
print(f"  - Total samples:    {len(df)}")
print(f"  - Positive (1):     {total_positive} ({prevalence:.2f}%)")
print(f"  - Negative (0):     {total_negative} ({100-prevalence:.2f}%)")
print(f"  - Class imbalance ratio: 1:{total_negative/total_positive:.2f}")
```

```
>>> STEP 3: Creating Target Variable 'diabetes2'...
Target variable 'diabetes2' created.
  - Total samples:    99343
  - Positive (1):     32346 (32.56%)
  - Negative (0):     66997 (67.44%)
  - Class imbalance ratio: 1:2.07
```

In [29]:

```
# ==========================================
# 4. DEMOGRAPHICS PROCESSING (Age, Gender, Race)
# ==========================================
print("\n>>> STEP 4: Processing Demographics (Age, Gender, Race)...")

# ---- AGE ----
# Convert age intervals into ordered integer groups 0-9
age_map = {
    "[0-10)": 0, "[10-20)": 1, "[20-30)": 2, "[30-40)": 3, "[40-50)": 4,
    "[50-60)": 5, "[60-70)": 6, "[70-80)": 7, "[80-90)": 8, "[90-100)": 9
}

# Check for invalid age values before filtering
invalid_age = df[~df["age"].isin(age_map.keys())]["age"].unique()
if len(invalid_age) > 0:
    print(f"  - Removing {len(df[~df['age'].isin(age_map.keys())])} rows with in

# Keep only rows where age is valid
df = df[df["age"].isin(age_map.keys())]
df["age"] = df["age"].map(age_map)

print(f"  - Age encoded (0-9). Distribution:")
print(df["age"].value_counts().sort_index().to_string())

# ---- GENDER ----
# Remove rows with invalid gender values
valid_genders = {"Male", "Female"}
invalid_gender = df[~df["gender"].isin(valid_genders)]["gender"].unique()
if len(invalid_gender) > 0:
    print(f"\n  - Removing {len(df[~df['gender'].isin(valid_genders)])} rows wit

df = df[df["gender"].isin(valid_genders)]
# Map gender: Male=1, Female=0
df["gender"] = df["gender"].map({"Male": 1, "Female": 0})

print(f"\n  - Gender encoded. Distribution:")
print(f"    Male (1):   {(df['gender']==1).sum()}")
print(f"    Female (0): {(df['gender']==0).sum()}")

# ---- RACE ----
# Normalize race categories: 1-Caucasian, 2-African American, 3-Asian, 4-Hispani
def encode_race(x):
    if x == "Caucasian":
        return 1
```

```
    elif x == "AfricanAmerican":
        return 2
    elif x == "Asian":
        return 3
    elif x == "Hispanic":
        return 4
    else:
        # Includes 'Other', missing, unknown values
        return 5

df["race"] = df["race"].apply(encode_race)

print(f"\n  - Race encoded. Distribution:")
print(f"    1-Caucasian:        {(df['race']==1).sum()}")
print(f"    2-African American: {(df['race']==2).sum()}")
print(f"    3-Asian:            {(df['race']==3).sum()}")
print(f"    4-Hispanic:         {(df['race']==4).sum()}")
print(f"    5-Others:           {(df['race']==5).sum()}")
```

```
>>> STEP 4: Processing Demographics (Age, Gender, Race)...
  - Age encoded (0-9). Distribution:
age
0      160
1      690
2     1649
3     3764
4     9607
5    17060
6    22059
7    25331
8    16434
9     2589

  - Removing 3 rows with invalid gender: ['Unknown/Invalid']

  - Gender encoded. Distribution:
    Male (1):   45886
    Female (0): 53454

  - Race encoded. Distribution:
    1-Caucasian:        74220
    2-African American: 18772
    3-Asian:            628
    4-Hispanic:         2017
    5-Others:           3703
```

In [30]:
```
# ==========================================
# 5. FINAL DATA SELECTION & TRAIN-TEST SPLIT
# ==========================================
print("\n>>> STEP 5: Finalizing Dataset for Modeling...")

# ==========================================
# 5A. DEFINE FEATURE COLUMNS
# ==========================================
# 11 base features (per project requirement) + 3 interaction features
feature_cols = [
    # Demographics (3)
    "race", "gender", "age",
    # ICD-9 based clinical features (8)
    "alcohol", "blood_pressure", "cholesterol", "heart_disease",
```

```
    "obesity", "pregnancy", "uric_acid", "blurred_vision"
]

target_col = "diabetes2"

print(f"Features selected ({len(feature_cols)} total):")
print(f" - Demographics (3): race, gender, age")
print(f" - Clinical features (8): alcohol, blood_pressure, cholesterol,")
print(f"   heart_disease, obesity, pregnancy, uric_acid, blurred_vision")
print(f" - Target: {target_col}")

# ==========================================
# 5B. EXTRACT FEATURES AND TARGET
# ==========================================
X = df[feature_cols].copy()
y = df[target_col].astype(int).copy()

# Check for missing values
missing_count = X.isnull().sum().sum()
if missing_count > 0:
    print(f"\n  WARNING: {missing_count} missing values found in features.")
    # Drop rows with missing values
    combined = pd.concat([X, y], axis=1).dropna()
    X = combined[feature_cols]
    y = combined[target_col].astype(int)
    print(f"  Dropped rows with missing values. New shape: {X.shape}")
else:
    print(f"\n  No missing values in features. Shape: {X.shape}")

# ==========================================
# 5C. CLASS BALANCE SUMMARY
# ==========================================
print(f"\nClass distribution in final dataset:")
print(f"  - Class 0 (No diabetes2): {(y==0).sum()} ({(y==0).mean()*100:.2f}%)")
print(f"  - Class 1 (Diabetes2):    {(y==1).sum()} ({(y==1).mean()*100:.2f}%)")

# ==========================================
# 5D. TRAIN-TEST SPLIT
# ==========================================
# 80% train, 20% test, stratified to preserve class proportions
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=RANDOM_SEED,
    stratify=y
)

print(f"\nTrain-Test Split (80/20, stratified):")
print(f"  - Train set: {X_train.shape[0]} samples")
print(f"    - Class 0: {(y_train==0).sum()} ({(y_train==0).mean()*100:.2f}%)")
print(f"    - Class 1: {(y_train==1).sum()} ({(y_train==1).mean()*100:.2f}%)")
print(f"  - Test set:  {X_test.shape[0]} samples")
print(f"    - Class 0: {(y_test==0).sum()} ({(y_test==0).mean()*100:.2f}%)")
print(f"    - Class 1: {(y_test==1).sum()} ({(y_test==1).mean()*100:.2f}%)")

# ==========================================
# 5E. FEATURE SUMMARY TABLE
# ==========================================
print("\n" + "="*60)
```

```python
print("FEATURE SUMMARY TABLE")
print("="*60)
print(f"{'Feature':<25} {'Type':<12} {'Min':>6} {'Max':>6} {'Mean':>8}")
print("-"*60)
for col in feature_cols:
    col_type = "Binary" if X[col].nunique() <= 2 else "Ordinal"
    print(f"{col:<25} {col_type:<12} {X[col].min():>6.0f} {X[col].max():>6.0f} {
print("="*60)
```

```
>>> STEP 5: Finalizing Dataset for Modeling...
Features selected (11 total):
 - Demographics (3): race, gender, age
 - Clinical features (8): alcohol, blood_pressure, cholesterol,
   heart_disease, obesity, pregnancy, uric_acid, blurred_vision
 - Target: diabetes2

  No missing values in features. Shape: (99340, 11)

Class distribution in final dataset:
  - Class 0 (No diabetes2): 66994 (67.44%)
  - Class 1 (Diabetes2):    32346 (32.56%)

Train-Test Split (80/20, stratified):
  - Train set: 79472 samples
    - Class 0: 53595 (67.44%)
    - Class 1: 25877 (32.56%)
  - Test set:  19868 samples
    - Class 0: 13399 (67.44%)
    - Class 1: 6469 (32.56%)

============================================================
FEATURE SUMMARY TABLE
============================================================
Feature                   Type            Min    Max     Mean
------------------------------------------------------------
race                      Ordinal           1      5   1.4116
gender                    Binary            0      1   0.4619
age                       Ordinal           0      9   6.0744
alcohol                   Binary            0      1   0.0296
blood_pressure            Binary            0      1   0.1980
cholesterol               Binary            0      1   0.0240
heart_disease             Binary            0      1   0.3006
obesity                   Binary            0      1   0.0133
pregnancy                 Binary            0      1   0.0068
uric_acid                 Binary            0      1   0.0075
blurred_vision            Binary            0      1   0.0007
============================================================
```

In [31]:

```python
# ========================================
# 5F. DESCRIPTIVE ANALYSIS
# ========================================
print("\n" + "="*80)
print(">>> STEP 5F: DESCRIPTIVE ANALYSIS OF FEATURES")
print("="*80)
print("(This section demonstrates data preprocessing was done correctly)")

# ---------------------------------------------
# 5F-1. Basic Information of Sample
# ---------------------------------------------
print("\n" + "-"*60)
```

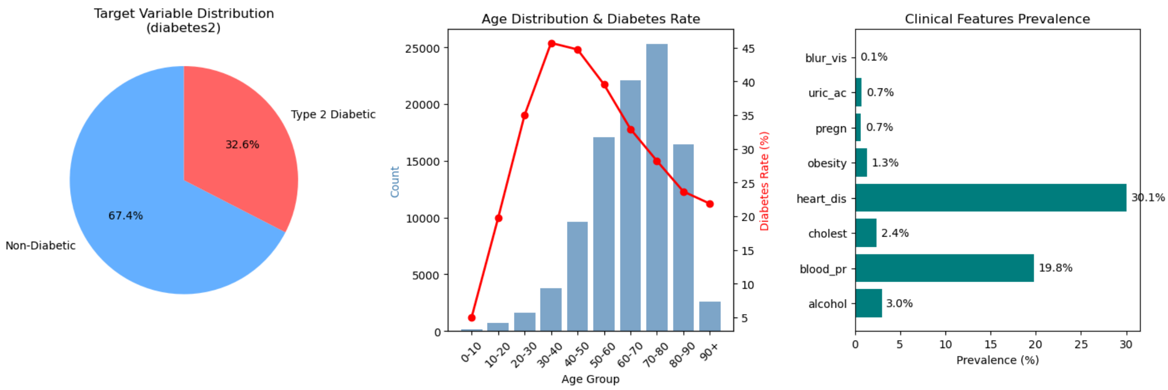
```
print("5F-1. SAMPLE CHARACTERISTICS")
print("-"*60)
print(f"Total samples: {len(df)}")
print(f"Total features: {len(feature_cols)}")
print(f"Target variable: diabetes2")
print(f"  - Positive (Type 2 Diabetes): {(y==1).sum()} ({(y==1).mean()*100:.2f}%
print(f"  - Negative (No Diabetes): {(y==0).sum()} ({(y==0).mean()*100:.2f}%)")
print(f"  - Imbalance Ratio: 1:{(y==0).sum()/(y==1).sum():.2f}")

# ---------------------------------------------
# 5F-2. Distribution of Demographic Characteristics
# ---------------------------------------------
print("\n" + "-"*60)
print("5F-2. DEMOGRAPHIC FEATURES DISTRIBUTION")
print("-"*60)

# Age distribution
print("\n[Age Distribution]")
age_dist = df.groupby('age').agg({
    'diabetes2': ['count', 'sum', 'mean']
}).round(4)
age_dist.columns = ['Total', 'Diabetes_Count', 'Diabetes_Rate']
age_labels = ['[0-10)', '[10-20)', '[20-30)', '[30-40)', '[40-50)',
              '[50-60)', '[60-70)', '[70-80)', '[80-90)', '[90-100)']
age_dist.index = [age_labels[i] for i in age_dist.index]
print(age_dist.to_string())

# Gender distribution
print("\n[Gender Distribution]")
gender_dist = df.groupby('gender').agg({
    'diabetes2': ['count', 'sum', 'mean']
}).round(4)
gender_dist.columns = ['Total', 'Diabetes_Count', 'Diabetes_Rate']
gender_dist.index = ['Female', 'Male']
print(gender_dist.to_string())

# Race distribution
print("\n[Race Distribution]")
race_dist = df.groupby('race').agg({
    'diabetes2': ['count', 'sum', 'mean']
}).round(4)
race_dist.columns = ['Total', 'Diabetes_Count', 'Diabetes_Rate']
race_labels = {1: 'Caucasian', 2: 'African American', 3: 'Asian', 4: 'Hispanic',
race_dist.index = [race_labels[i] for i in race_dist.index]
print(race_dist.to_string())

# ---------------------------------------------
# 5F-3. Distribution of Clinical Features (ICD-9 based)
# ---------------------------------------------
print("\n" + "-"*60)
print("5F-3. CLINICAL FEATURES DISTRIBUTION (ICD-9 Based)")
print("-"*60)

clinical_features = ['alcohol', 'blood_pressure', 'cholesterol', 'heart_disease'
                     'obesity', 'pregnancy', 'uric_acid', 'blurred_vision']

clinical_summary = []
for feat in clinical_features:
    total_positive = df[feat].sum()
    diabetes_with_feat = df[df[feat]==1]['diabetes2'].sum()
```

```
    diabetes_rate_with = df[df[feat]==1]['diabetes2'].mean() if total_positive >
    diabetes_rate_without = df[df[feat]==0]['diabetes2'].mean()

    clinical_summary.append({
        'Feature': feat,
        'Positive_Cases': total_positive,
        'Prevalence(%)': total_positive/len(df)*100,
        'Diabetes_Rate_With(%)': diabetes_rate_with*100,
        'Diabetes_Rate_Without(%)': diabetes_rate_without*100,
        'Relative_Risk': diabetes_rate_with/diabetes_rate_without if diabetes_ra
    })

clinical_df = pd.DataFrame(clinical_summary)
print(clinical_df.to_string(index=False))

# -----------------------------------------------
# 5F-4. The Correlation Between Features and Target Variables
# -----------------------------------------------
print("\n" + "-"*60)
print("5F-4. FEATURE-TARGET CORRELATION")
print("-"*60)

correlations = X.corrwith(y).sort_values(ascending=False)
print("\nCorrelation with diabetes2 (sorted):")
for feat, corr in correlations.items():
    print(f"  {feat:<25}: {corr:>8.4f}")

# -----------------------------------------------
# 5F-5. Descriptive Statistical Visualization
# -----------------------------------------------
print("\n>>> Generating Descriptive Analysis Figures...")

# Figure 1: Distribution of target variables (pie chart)
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# 1a. Target variable distribution
labels = ['Non-Diabetic', 'Type 2 Diabetic']
sizes = [(y==0).sum(), (y==1).sum()]
colors = ['#66b3ff', '#ff6666']
axes[0].pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=9
axes[0].set_title('Target Variable Distribution\n(diabetes2)', fontsize=12)

# 1b. Age distribution with diabetes rate
age_counts = df.groupby('age').size()
diabetes_rates = df.groupby('age')['diabetes2'].mean() * 100
ax1b = axes[1]
ax1b_twin = ax1b.twinx()
bars = ax1b.bar(range(len(age_counts)), age_counts.values, color='steelblue', al
line = ax1b_twin.plot(range(len(age_counts)), diabetes_rates.values, 'r-o', line
ax1b.set_xlabel('Age Group')
ax1b.set_ylabel('Count', color='steelblue')
ax1b_twin.set_ylabel('Diabetes Rate (%)', color='red')
ax1b.set_xticks(range(len(age_counts)))
ax1b.set_xticklabels(['0-10', '10-20', '20-30', '30-40', '40-50', '50-60', '60-7
ax1b.set_title('Age Distribution & Diabetes Rate', fontsize=12)

# 1c. Clinical features prevalence
feat_names = ['alcohol', 'blood_pr', 'cholest', 'heart_dis', 'obesity', 'pregn',
feat_prevalence = [df[f].mean()*100 for f in clinical_features]
axes[2].barh(feat_names, feat_prevalence, color='teal')
```

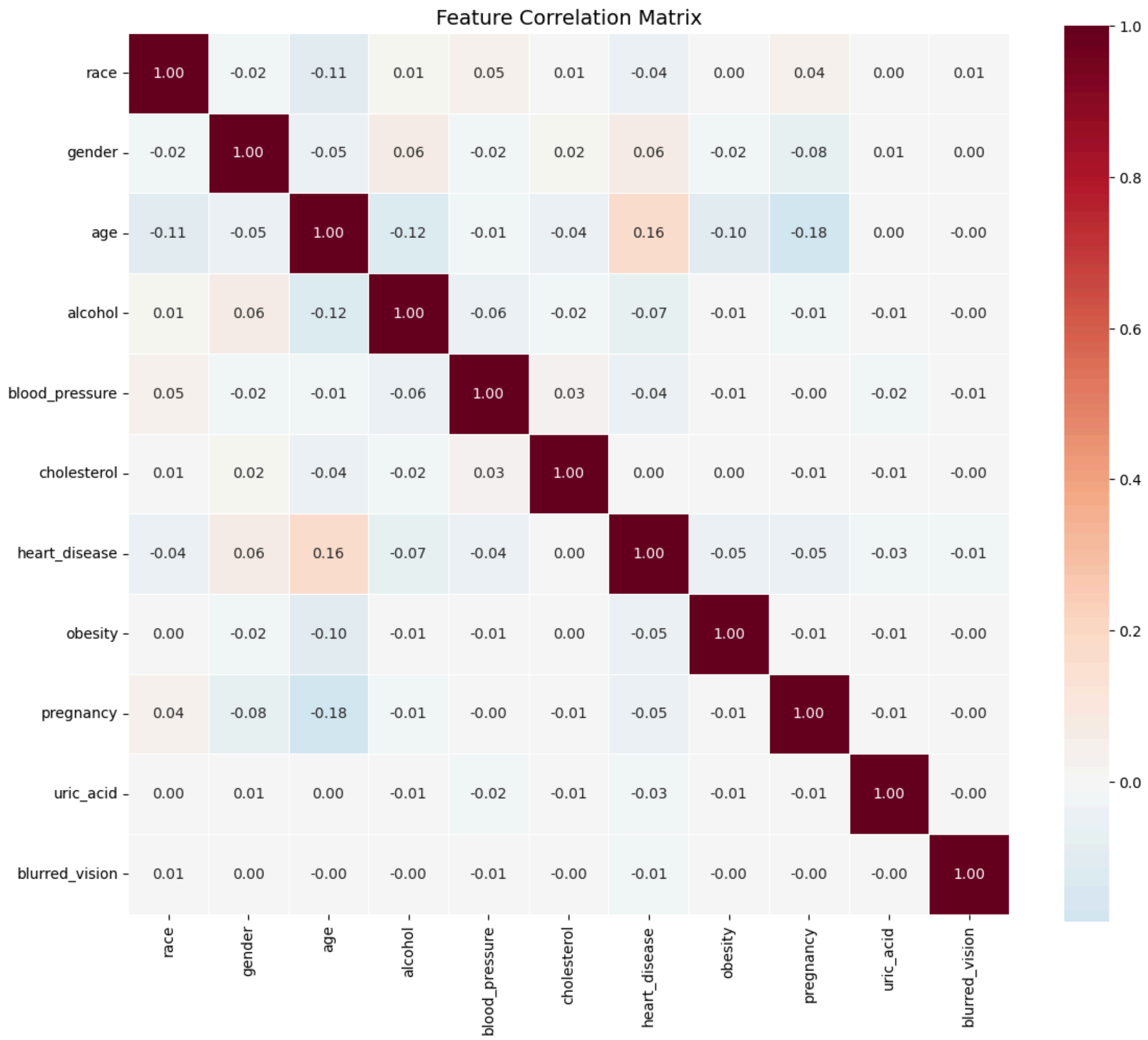
```
axes[2].set_xlabel('Prevalence (%)')
axes[2].set_title('Clinical Features Prevalence', fontsize=12)
for i, v in enumerate(feat_prevalence):
    axes[2].text(v + 0.5, i, f'{v:.1f}%', va='center')

plt.tight_layout()
plt.savefig('descriptive_analysis_1.png', dpi=300, bbox_inches='tight')
plt.show()
print("Saved: descriptive_analysis_1.png")

# Figure 2: Feature correlation heat map
plt.figure(figsize=(12, 10))
correlation_matrix = X.corr()
import seaborn as sns
sns.heatmap(correlation_matrix, annot=True, cmap='RdBu_r', center=0,
            fmt='.2f', square=True, linewidths=0.5)
plt.title('Feature Correlation Matrix', fontsize=14)
plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=300, bbox_inches='tight')
plt.show()
print("Saved: correlation_heatmap.png")

# Figure 3: Comparison of diabetes rates for each clinical feature
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, feat in enumerate(clinical_features):
    ax = axes[i]
    # The rate of diabetes with and without this feature
    rates = [
        df[df[feat]==0]['diabetes2'].mean() * 100,
        df[df[feat]==1]['diabetes2'].mean() * 100
    ]
    bars = ax.bar(['Without', 'With'], rates, color=['#3498db', '#e74c3c'])
    ax.set_title(feat.replace('_', ' ').title(), fontsize=11)
    ax.set_ylabel('Diabetes Rate (%)')
    ax.set_ylim(0, max(rates)*1.3 if max(rates) > 0 else 30)
    for bar, rate in zip(bars, rates):
        ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1,
                f'{rate:.1f}%', ha='center', fontsize=9)

plt.suptitle('Diabetes Rate by Clinical Feature Presence', fontsize=14)
plt.tight_layout()
plt.savefig('clinical_features_diabetes_rate.png', dpi=300, bbox_inches='tight')
plt.show()
print("Saved: clinical_features_diabetes_rate.png")

print("\n>>> Descriptive Analysis complete.")
```

```
================================================================================
>>> STEP 5F: DESCRIPTIVE ANALYSIS OF FEATURES
================================================================================
(This section demonstrates data preprocessing was done correctly)

------------------------------------------------------------
5F-1. SAMPLE CHARACTERISTICS
------------------------------------------------------------
Total samples: 99340
Total features: 11
Target variable: diabetes2
  - Positive (Type 2 Diabetes): 32346 (32.56%)
  - Negative (No Diabetes): 66994 (67.44%)
  - Imbalance Ratio: 1:2.07

------------------------------------------------------------
5F-2. DEMOGRAPHIC FEATURES DISTRIBUTION
------------------------------------------------------------

[Age Distribution]
          Total  Diabetes_Count  Diabetes_Rate
[0-10)      160               8         0.0500
[10-20)     690             136         0.1971
[20-30)    1649             577         0.3499
[30-40)    3764            1720         0.4570
[40-50)    9607            4297         0.4473
[50-60)   17060            6746         0.3954
[60-70)   22058            7265         0.3294
[70-80)   25329            7143         0.2820
[80-90)   16434            3888         0.2366
[90-100)   2589             566         0.2186

[Gender Distribution]
        Total  Diabetes_Count  Diabetes_Rate
Female  53454           17284         0.3233
Male    45886           15062         0.3282

[Race Distribution]
                  Total  Diabetes_Count  Diabetes_Rate
Caucasian         74220           22881         0.3083
African American  18772            7060         0.3761
Asian               628             202         0.3217
Hispanic           2017             844         0.4184
Others             3703            1359         0.3670

------------------------------------------------------------
5F-3. CLINICAL FEATURES DISTRIBUTION (ICD-9 Based)
------------------------------------------------------------
       Feature  Positive_Cases  Prevalence(%)  Diabetes_Rate_With(%)  Diabetes_Ra
te_Without(%)  Relative_Risk
       alcohol            2944       2.963559              24.796196
32.798041       0.756027
blood_pressure           19666      19.796658              37.206346
31.414263       1.184377
   cholesterol            2384       2.399839              42.365772
32.319815       1.310830
 heart_disease           29860      30.058385              21.235767
37.428037       0.567376
       obesity            1324       1.332796              49.471299
32.332476       1.530081
```

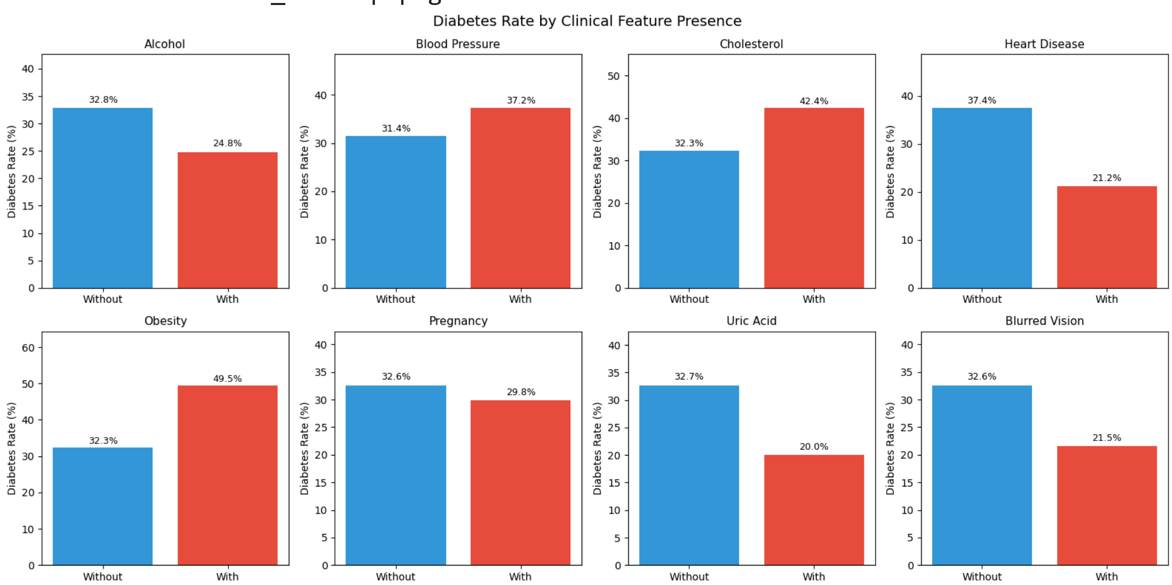
```
      pregnancy             677       0.681498              29.837518
32.579589       0.915835
      uric_acid             745       0.749950              20.000000
32.655814       0.612448
blurred_vision               65       0.065432              21.538462
32.568119       0.661336

------------------------------------------------------------
5F-4. FEATURE-TARGET CORRELATION
------------------------------------------------------------

Correlation with diabetes2 (sorted):
  race                         :   0.0501
  blood_pressure               :   0.0493
  obesity                      :   0.0419
  cholesterol                  :   0.0328
  gender                       :   0.0052
  pregnancy                    :  -0.0048
  blurred_vision               :  -0.0060
  uric_acid                    :  -0.0233
  alcohol                      :  -0.0290
  age                          :  -0.1256
  heart_disease                :  -0.1584

>>> Generating Descriptive Analysis Figures...
```




```
Saved: descriptive_analysis_1.png
```



Saved: correlation_heatmap.png



Saved: clinical_features_diabetes_rate.png

>>> Descriptive Analysis complete.

In [32]:
```
# ==========================================
# 6. MODEL TRAINING AND EVALUATION
# ==========================================
print("\n>>> STEP 6: Model Training...")

# ==========================================
```

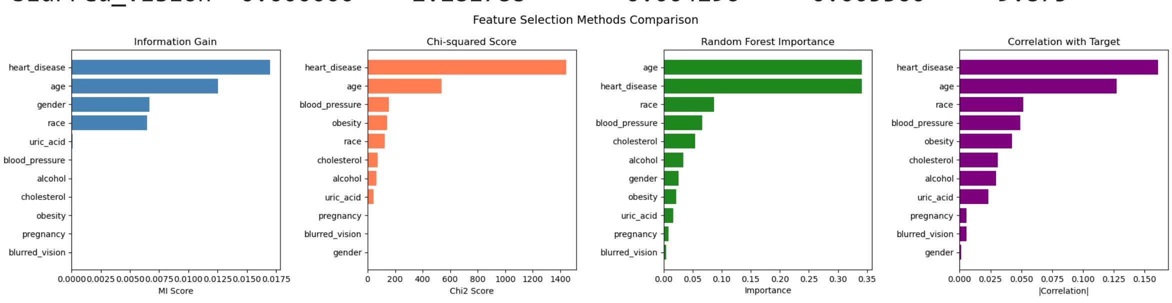
```python
# 6A. DEFINE EVALUATION HELPER FUNCTION & CV STRATEGIES
# ==========================================

# Define cross-validation strategies (used consistently across all models)
cv_5fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_SEED)
cv_10fold = StratifiedKFold(n_splits=10, shuffle=True, random_state=RANDOM_SEED)

print("Cross-validation strategies defined:")
print("  - 5-fold: for hyperparameter tuning (GridSearchCV)")
print("  - 10-fold: for final model evaluation (paper-style)")

def evaluate_model(name, y_true, y_pred, y_proba, print_results=True):
    """
    Comprehensive model evaluation function.
    Returns a dictionary with all metrics for later comparison.
    """
    # Confusion Matrix components
    cm = confusion_matrix(y_true, y_pred)
    tn, fp, fn, tp = cm.ravel()

    # Calculate metrics
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec = recall_score(y_true, y_pred, zero_division=0)
    f1 = f1_score(y_true, y_pred, zero_division=0)
    roc_auc = roc_auc_score(y_true, y_proba)
    pr_auc = average_precision_score(y_true, y_proba)

    # Specificity (True Negative Rate)
    specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

    # Store results
    results = {
        "name": name,
        "accuracy": acc,
        "precision": prec,
        "recall": rec,
        "f1_score": f1,
        "specificity": specificity,
        "roc_auc": roc_auc,
        "pr_auc": pr_auc,
        "tn": tn, "fp": fp, "fn": fn, "tp": tp,
        "y_pred": y_pred,
        "y_proba": y_proba
    }

    if print_results:
        print(f"\n{'='*60}")
        print(f" {name}")
        print(f"{'='*60}")
        print(f"\nConfusion Matrix:")
        print(f"                 Predicted")
        print(f"                 Neg    Pos")
        print(f"  Actual Neg    {tn:>5}  {fp:>5}   (TN, FP)")
        print(f"  Actual Pos    {fn:>5}  {tp:>5}   (FN, TP)")
        print(f"\nMetrics:")
        print(f"  Accuracy:    {acc:.4f}")
        print(f"  Precision:   {prec:.4f}")
        print(f"  Recall:      {rec:.4f}")
        print(f"  F1-Score:    {f1:.4f}")
```

```
        print(f"  Specificity: {specificity:.4f}")
        print(f"  ROC AUC:     {roc_auc:.4f}")
        print(f"  PR AUC:      {pr_auc:.4f}")

    return results

# Dictionary to store all model results (test set evaluation)
all_results = {}

# Dictionary to store trained models (for SHAP analysis later)
trained_models = {}

print("\nEvaluation function defined. Ready for model training.")

# ==========================================
# 6A-2. FEATURE SELECTION ANALYSIS
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6A-2: FEATURE SELECTION ANALYSIS")
print("="*60)

from sklearn.feature_selection import SelectKBest, mutual_info_classif, chi2

# method1: Information Gain (Mutual Information)
print("\n--- Method 1: Information Gain (Mutual Information) ---")
mi_selector = SelectKBest(score_func=mutual_info_classif, k='all')
mi_selector.fit(X_train, y_train)
mi_scores = pd.Series(mi_selector.scores_, index=feature_cols).sort_values(ascen

# method2: Chi-squared Test
print("--- Method 2: Chi-squared Test ---")
chi2_selector = SelectKBest(score_func=chi2, k='all')
chi2_selector.fit(X_train, y_train)
chi2_scores = pd.Series(chi2_selector.scores_, index=feature_cols).sort_values(a

# method3: Random Forest Feature Importance
print("--- Method 3: Random Forest Importance ---")
rf_for_fs = RandomForestClassifier(n_estimators=100, class_weight='balanced',
                                   random_state=RANDOM_SEED, n_jobs=-1)
rf_for_fs.fit(X_train, y_train)
rf_importance = pd.Series(rf_for_fs.feature_importances_, index=feature_cols).sc

# method4: Correlation with Target
print("--- Method 4: Correlation with Target ---")
corr_scores = X_train.corrwith(y_train).abs().sort_values(ascending=False)

# Summary table
print("\n" + "="*80)
print("FEATURE SELECTION RESULTS SUMMARY")
print("="*80)

fs_summary = pd.DataFrame({
    'Feature': feature_cols,
    'MI_Score': mi_scores.reindex(feature_cols).values,
    'Chi2_Score': chi2_scores.reindex(feature_cols).values,
    'RF_Importance': rf_importance.reindex(feature_cols).values,
    'Correlation': corr_scores.reindex(feature_cols).values
})

# Calculate the average ranking
```

```
for col in ['MI_Score', 'Chi2_Score', 'RF_Importance', 'Correlation']:
    fs_summary[f'{col}_Rank'] = fs_summary[col].rank(ascending=False)

fs_summary['Avg_Rank'] = fs_summary[['MI_Score_Rank', 'Chi2_Score_Rank',
                                      'RF_Importance_Rank', 'Correlation_Rank']]
fs_summary = fs_summary.sort_values('Avg_Rank')

print(fs_summary[['Feature', 'MI_Score', 'Chi2_Score', 'RF_Importance', 'Correla

# Visualization
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

# MI Scores
axes[0].barh(mi_scores.index[::-1], mi_scores.values[::-1], color='steelblue')
axes[0].set_title('Information Gain', fontsize=12)
axes[0].set_xlabel('MI Score')

# Chi2 Scores
axes[1].barh(chi2_scores.index[::-1], chi2_scores.values[::-1], color='coral')
axes[1].set_title('Chi-squared Score', fontsize=12)
axes[1].set_xlabel('Chi2 Score')

# RF Importance
axes[2].barh(rf_importance.index[::-1], rf_importance.values[::-1], color='fores
axes[2].set_title('Random Forest Importance', fontsize=12)
axes[2].set_xlabel('Importance')

# Correlation
axes[3].barh(corr_scores.index[::-1], corr_scores.values[::-1], color='purple')
axes[3].set_title('Correlation with Target', fontsize=12)
axes[3].set_xlabel('|Correlation|')

plt.suptitle('Feature Selection Methods Comparison', fontsize=14)
plt.tight_layout()
plt.savefig('feature_selection_comparison.png', dpi=300, bbox_inches='tight')
plt.show()
print("\nSaved: feature_selection_comparison.png")

print("\n>>> Feature Selection Analysis complete.")


# ==========================================
# 6B. BASELINE MODEL: Random Forest
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6B: Training Baseline Random Forest...")
print("="*60)

rf_baseline = RandomForestClassifier(
    n_estimators=300,
    max_depth=10,
    min_samples_leaf=2,
    class_weight="balanced",
    random_state=RANDOM_SEED,
    n_jobs=-1
)

rf_baseline.fit(X_train, y_train)

y_pred_rf_base = rf_baseline.predict(X_test)
```

```
y_proba_rf_base = rf_baseline.predict_proba(X_test)[:, 1]

all_results["RF_Baseline"] = evaluate_model(
    "Baseline Random Forest",
    y_test, y_pred_rf_base, y_proba_rf_base
)

trained_models["RF_Baseline"] = rf_baseline

print("\n>>> Baseline Random Forest complete.")
```

```
>>> STEP 6: Model Training...
Cross-validation strategies defined:
  - 5-fold: for hyperparameter tuning (GridSearchCV)
  - 10-fold: for final model evaluation (paper-style)

Evaluation function defined. Ready for model training.

============================================================
>>> STEP 6A-2: FEATURE SELECTION ANALYSIS
============================================================

--- Method 1: Information Gain (Mutual Information) ---
--- Method 2: Chi-squared Test ---
--- Method 3: Random Forest Importance ---
--- Method 4: Correlation with Target ---

================================================================================
FEATURE SELECTION RESULTS SUMMARY
================================================================================
       Feature  MI_Score  Chi2_Score  RF_Importance  Correlation  Avg_Rank
 heart_disease  0.016989 1441.934197       0.341047     0.161145     1.250
           age  0.012531  538.385411       0.341658     0.127417     1.750
          race  0.006488  124.499412       0.086384     0.051830     3.750
blood_pressure  0.000000  154.531010       0.066313     0.049242     4.875
   cholesterol  0.000000   74.746019       0.053921     0.031042     6.375
       obesity  0.000000  141.816102       0.021863     0.042518     6.375
       alcohol  0.000000   66.670977       0.034350     0.029406     7.125
     uric_acid  0.000089   42.882801       0.016804     0.023315     7.500
        gender  0.006650    0.043015       0.025240     0.001003     8.000
     pregnancy  0.000000    2.604481       0.008125     0.005744     9.125
 blurred_vision  0.000000    2.281788       0.004296     0.005360     9.875
```



```
Saved: feature_selection_comparison.png

>>> Feature Selection Analysis complete.

============================================================
>>> STEP 6B: Training Baseline Random Forest...
============================================================

============================================================
 Baseline Random Forest
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg     7981   5418   (TN, FP)
  Actual Pos     2548   3921   (FN, TP)

Metrics:
  Accuracy:    0.5991
  Precision:   0.4199
  Recall:      0.6061
  F1-Score:    0.4961
  Specificity: 0.5956
  ROC AUC:     0.6418
  PR AUC:      0.4496

>>> Baseline Random Forest complete.
```

In [33]:

```python
# ==========================================
# 6C. TUNED MODEL: Random Forest with GridSearchCV
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6C: Tuning Random Forest (GridSearchCV, 5-fold)...")
print("="*60)

param_grid_rf = {
    "n_estimators": [100, 200, 300],
    "max_depth": [6, 8, 10, 12],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2, 4]
}

rf_for_tuning = RandomForestClassifier(
    class_weight="balanced",
    random_state=RANDOM_SEED,
    n_jobs=-1
)

grid_search_rf = GridSearchCV(
    estimator=rf_for_tuning,
    param_grid=param_grid_rf,
    scoring="f1",
    cv=cv_5fold,
    n_jobs=-1,
    verbose=1
)

grid_search_rf.fit(X_train, y_train)
```

```
print(f"\nBest parameters: {grid_search_rf.best_params_}")
print(f"Best CV F1 score: {grid_search_rf.best_score_:.4f}")

rf_tuned = grid_search_rf.best_estimator_

y_pred_rf_tuned = rf_tuned.predict(X_test)
y_proba_rf_tuned = rf_tuned.predict_proba(X_test)[:, 1]

all_results["RF_Tuned"] = evaluate_model(
    "Tuned Random Forest",
    y_test, y_pred_rf_tuned, y_proba_rf_tuned
)

trained_models["RF_Tuned"] = rf_tuned

print("\n>>> Tuned Random Forest complete.")
```

```
============================================================
>>> STEP 6C: Tuning Random Forest (GridSearchCV, 5-fold)...
============================================================
Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best parameters: {'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 2,
'n_estimators': 100}
Best CV F1 score: 0.5039

============================================================
 Tuned Random Forest
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg     7806   5593   (TN, FP)
  Actual Pos     2480   3989   (FN, TP)

Metrics:
  Accuracy:    0.5937
  Precision:   0.4163
  Recall:      0.6166
  F1-Score:    0.4970
  Specificity: 0.5826
  ROC AUC:     0.6377
  PR AUC:      0.4451

>>> Tuned Random Forest complete.
```

In [34]:

```
# ==========================================
# 6D. ADVANCED MODEL: XGBoost
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6D: Training XGBoost...")
print("="*60)

# Compute scale_pos_weight to handle class imbalance
scale_pos_weight = (y_train == 0).sum() / (y_train == 1).sum()
print(f"Class imbalance ratio (scale_pos_weight): {scale_pos_weight:.2f}")

xgb_model = XGBClassifier(
    n_estimators=200,
```

```
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="binary:logistic",
    eval_metric="logloss",
    scale_pos_weight=scale_pos_weight,
    random_state=RANDOM_SEED,
    n_jobs=-1,
    verbosity=0
)

xgb_model.fit(X_train, y_train)

y_pred_xgb = xgb_model.predict(X_test)
y_proba_xgb = xgb_model.predict_proba(X_test)[:, 1]

all_results["XGBoost"] = evaluate_model(
    "XGBoost",
    y_test, y_pred_xgb, y_proba_xgb
)

trained_models["XGBoost"] = xgb_model

print("\n>>> XGBoost complete.")
```

```
============================================================
>>> STEP 6D: Training XGBoost...
============================================================
Class imbalance ratio (scale_pos_weight): 2.07

============================================================
 XGBoost
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg     8027   5372   (TN, FP)
  Actual Pos     2581   3888   (FN, TP)

Metrics:
  Accuracy:    0.5997
  Precision:   0.4199
  Recall:      0.6010
  F1-Score:    0.4944
  Specificity: 0.5991
  ROC AUC:     0.6411
  PR AUC:      0.4491

>>> XGBoost complete.
```

In [35]:

```
# ==========================================
# 6E. ADVANCED MODEL: LightGBM
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6E: Training LightGBM...")
print("="*60)

lgb_model = LGBMClassifier(
```

```
    n_estimators=200,
    max_depth=6,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    class_weight="balanced",
    random_state=RANDOM_SEED,
    n_jobs=-1,
    verbose=-1
)

lgb_model.fit(X_train, y_train)

y_pred_lgb = lgb_model.predict(X_test)
y_proba_lgb = lgb_model.predict_proba(X_test)[:, 1]

all_results["LightGBM"] = evaluate_model(
    "LightGBM",
    y_test, y_pred_lgb, y_proba_lgb
)

trained_models["LightGBM"] = lgb_model

print("\n>>> LightGBM complete.")
```

```
============================================================
>>> STEP 6E: Training LightGBM...
============================================================

============================================================
 LightGBM
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg     8077   5322   (TN, FP)
  Actual Pos     2591   3878   (FN, TP)

Metrics:
  Accuracy:    0.6017
  Precision:   0.4215
  Recall:      0.5995
  F1-Score:    0.4950
  Specificity: 0.6028
  ROC AUC:     0.6429
  PR AUC:      0.4516

>>> LightGBM complete.
```

In [36]:

```
# ========================================
# 6F. BASELINE MODEL: Logistic Regression
# ========================================
print("\n" + "="*60)
print(">>> STEP 6F: Training Logistic Regression...")
print("="*60)

lr_model = LogisticRegression(
    class_weight="balanced",
    solver="liblinear",
```

```
    max_iter=1000,
    random_state=RANDOM_SEED
)

lr_model.fit(X_train, y_train)

y_pred_lr = lr_model.predict(X_test)
y_proba_lr = lr_model.predict_proba(X_test)[:, 1]

all_results["LogisticRegression"] = evaluate_model(
    "Logistic Regression",
    y_test, y_pred_lr, y_proba_lr
)

trained_models["LogisticRegression"] = lr_model

print("\n>>> Logistic Regression complete.")
```

```
============================================================
>>> STEP 6F: Training Logistic Regression...
============================================================

============================================================
 Logistic Regression
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg     7312   6087   (TN, FP)
  Actual Pos     2311   4158   (FN, TP)

Metrics:
  Accuracy:    0.5773
  Precision:   0.4059
  Recall:      0.6428
  F1-Score:    0.4975
  Specificity: 0.5457
  ROC AUC:     0.6249
  PR AUC:      0.4149

>>> Logistic Regression complete.
```

In [37]:

```
# ==========================================
# 6G. MODEL: K-Nearest Neighbors (KNN)
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6G: Training KNN...")
print("="*60)

# KNN requires feature scaling
knn_pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=11, n_jobs=-1))
])

knn_pipeline.fit(X_train, y_train)

y_pred_knn = knn_pipeline.predict(X_test)
y_proba_knn = knn_pipeline.predict_proba(X_test)[:, 1]
```

```
all_results["KNN"] = evaluate_model(
    "K-Nearest Neighbors (k=11)",
    y_test, y_pred_knn, y_proba_knn
)

trained_models["KNN"] = knn_pipeline

print("\n>>> KNN complete.")
```

```
============================================================
>>> STEP 6G: Training KNN...
============================================================

============================================================
 K-Nearest Neighbors (k=11)
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg    11992   1407   (TN, FP)
  Actual Pos     5147   1322   (FN, TP)

Metrics:
  Accuracy:    0.6701
  Precision:   0.4844
  Recall:      0.2044
  F1-Score:    0.2875
  Specificity: 0.8950
  ROC AUC:     0.5902
  PR AUC:      0.3964

>>> KNN complete.
```

In [38]:

```
# ==========================================
# 6I. MODEL: Naive Bayes
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6I: Training Naive Bayes...")
print("="*60)

nb_model = GaussianNB()

nb_model.fit(X_train, y_train)

y_pred_nb = nb_model.predict(X_test)
y_proba_nb = nb_model.predict_proba(X_test)[:, 1]

all_results["NaiveBayes"] = evaluate_model(
    "Naive Bayes",
    y_test, y_pred_nb, y_proba_nb
)

trained_models["NaiveBayes"] = nb_model

print("\n>>> Naive Bayes complete.")
```

```
============================================================
>>> STEP 6I: Training Naive Bayes...
============================================================

============================================================
 Naive Bayes
============================================================

Confusion Matrix:
                 Predicted
                 Neg    Pos
  Actual Neg    10322   3077   (TN, FP)
  Actual Pos     4078   2391   (FN, TP)

Metrics:
  Accuracy:    0.6399
  Precision:   0.4373
  Recall:      0.3696
  F1-Score:    0.4006
  Specificity: 0.7704
  ROC AUC:     0.6155
  PR AUC:      0.4103

>>> Naive Bayes complete.
```

In [39]:

```python
# ==========================================
# 6J. 10-FOLD CROSS-VALIDATION (ALL MODELS EXCEPT SVM)
# ==========================================
print("\n" + "="*60)
print(">>> STEP 6J: 10-Fold Cross-Validation (Paper-style Evaluation)")
print("="*60)

# Models to evaluate with 10-fold CV
# Note: SVM excluded due to extremely long training time
models_for_cv = {
    "RF_Baseline": RandomForestClassifier(
        n_estimators=300, max_depth=10, min_samples_leaf=2,
        class_weight="balanced", random_state=RANDOM_SEED, n_jobs=-1
    ),
    "RF_Tuned": RandomForestClassifier(
        **grid_search_rf.best_params_,
        class_weight="balanced", random_state=RANDOM_SEED, n_jobs=-1
    ),
    "XGBoost": XGBClassifier(
        n_estimators=200, max_depth=6, learning_rate=0.1,
        subsample=0.8, colsample_bytree=0.8,
        scale_pos_weight=scale_pos_weight,
        random_state=RANDOM_SEED, n_jobs=-1, verbosity=0
    ),
    "LightGBM": LGBMClassifier(
        n_estimators=200, max_depth=6, learning_rate=0.05,
        subsample=0.8, colsample_bytree=0.8,
        class_weight="balanced", random_state=RANDOM_SEED, n_jobs=-1, verbose=-1
    ),
    "LogisticRegression": LogisticRegression(
        class_weight="balanced", solver="liblinear",
        max_iter=1000, random_state=RANDOM_SEED
    ),
    "KNN": Pipeline([
        ("scaler", StandardScaler()),
```

```
        ("knn", KNeighborsClassifier(n_neighbors=11, n_jobs=-1))
    ]),
    # SVM excluded: too slow for 10-fold CV on this dataset
    "NaiveBayes": GaussianNB()
}

# Store CV results
cv_results = {}

print("\nRunning 10-fold CV for all models (SVM excluded)...\n")

for name, model in models_for_cv.items():
    print(f"  Evaluating {name}...", end=" ")

    # Accuracy
    acc_scores = cross_val_score(model, X, y, cv=cv_10fold, scoring="accuracy",

    # F1-score
    f1_scores = cross_val_score(model, X, y, cv=cv_10fold, scoring="f1", n_jobs=

    # Precision
    prec_scores = cross_val_score(model, X, y, cv=cv_10fold, scoring="precision"

    # Recall
    rec_scores = cross_val_score(model, X, y, cv=cv_10fold, scoring="recall", n_

    # ROC AUC
    auc_scores = cross_val_score(model, X, y, cv=cv_10fold, scoring="roc_auc", r

    cv_results[name] = {
        "accuracy_mean": acc_scores.mean(),
        "accuracy_std": acc_scores.std(),
        "f1_mean": f1_scores.mean(),
        "f1_std": f1_scores.std(),
        "precision_mean": prec_scores.mean(),
        "precision_std": prec_scores.std(),
        "recall_mean": rec_scores.mean(),
        "recall_std": rec_scores.std(),
        "roc_auc_mean": auc_scores.mean(),
        "roc_auc_std": auc_scores.std()
    }

    print(f"Done. F1={f1_scores.mean():.4f} (±{f1_scores.std():.4f})")

# Add SVM results from test set only (no CV)
# Mark it separately so we know it's not from 10-fold CV
print("\n  Note: SVM evaluated on test set only (10-fold CV too slow)")

# ==========================================
# PRINT CV RESULTS TABLE
# ==========================================
print("\n" + "="*90)
print(" 10-FOLD CROSS-VALIDATION RESULTS SUMMARY")
print("="*90)
print(f"{'Model':<20} {'Accuracy':>14} {'Precision':>14} {'Recall':>14} {'F1-Sco
print("-"*90)

for name, res in cv_results.items():
    print(f"{name:<20} "
          f"{res['accuracy_mean']:.4f}±{res['accuracy_std']:.4f} "
```

```
          f"{res['precision_mean']:.4f}±{res['precision_std']:.4f} "
          f"{res['recall_mean']:.4f}±{res['recall_std']:.4f} "
          f"{res['f1_mean']:.4f}±{res['f1_std']:.4f} "
          f"{res['roc_auc_mean']:.4f}±{res['roc_auc_std']:.4f}")

print("-"*90)
print("Note: SVM excluded from 10-fold CV due to computational time constraints.
print("      SVM results available from test set evaluation (Step 6H).")
print("="*90)

# Find best model by F1
best_model_name = max(cv_results, key=lambda x: cv_results[x]["f1_mean"])
print(f"\nBest model by F1-score: {best_model_name} (F1 = {cv_results[best_model

print("\n>>> 10-Fold Cross-Validation complete.")
```

```
============================================================
>>> STEP 6J: 10-Fold Cross-Validation (Paper-style Evaluation)
============================================================

Running 10-fold CV for all models (SVM excluded)...

Done. F1=0.5018 (±0.0053)..
Done. F1=0.5021 (±0.0058)
Done. F1=0.5001 (±0.0051)
Done. F1=0.5009 (±0.0050)
Done. F1=0.5036 (±0.0048)ession...
Done. F1=0.3974 (±0.0161)
Done. F1=0.3956 (±0.0269).

  Note: SVM evaluated on test set only (10-fold CV too slow)

=================================================================================
=========
 10-FOLD CROSS-VALIDATION RESULTS SUMMARY
=================================================================================
=========
Model                      Accuracy      Precision         Recall        F1-Score
ROC AUC
---------------------------------------------------------------------------------
---------
RF_Baseline          0.6054±0.0029 0.4260±0.0033 0.6105±0.0100 0.5018±0.0053 0.64
79±0.0071
RF_Tuned             0.5997±0.0048 0.4220±0.0041 0.6201±0.0147 0.5021±0.0058 0.64
51±0.0073
XGBoost              0.6079±0.0045 0.4276±0.0044 0.6024±0.0088 0.5001±0.0051 0.64
73±0.0070
LightGBM             0.6090±0.0043 0.4286±0.0042 0.6028±0.0096 0.5009±0.0050 0.64
84±0.0071
LogisticRegression   0.5818±0.0037 0.4104±0.0036 0.6515±0.0077 0.5036±0.0048 0.63
24±0.0068
KNN                  0.6384±0.0165 0.4374±0.0242 0.3674±0.0338 0.3974±0.0161 0.60
40±0.0095
NaiveBayes           0.6437±0.0060 0.4419±0.0101 0.3600±0.0408 0.3956±0.0269 0.62
29±0.0073
---------------------------------------------------------------------------------
---------
Note: SVM excluded from 10-fold CV due to computational time constraints.
      SVM results available from test set evaluation (Step 6H).
=================================================================================
=========

Best model by F1-score: LogisticRegression (F1 = 0.5036)

>>> 10-Fold Cross-Validation complete.
```

In [40]:

```
# ==========================================
# 7. COMPREHENSIVE MODEL EVALUATION
# ==========================================
print("\n" + "="*60)
print(">>> STEP 7: Comprehensive Model Evaluation")
print("="*60)

# ==========================================
# 7A. TEST SET RESULTS SUMMARY TABLE
# ==========================================
```

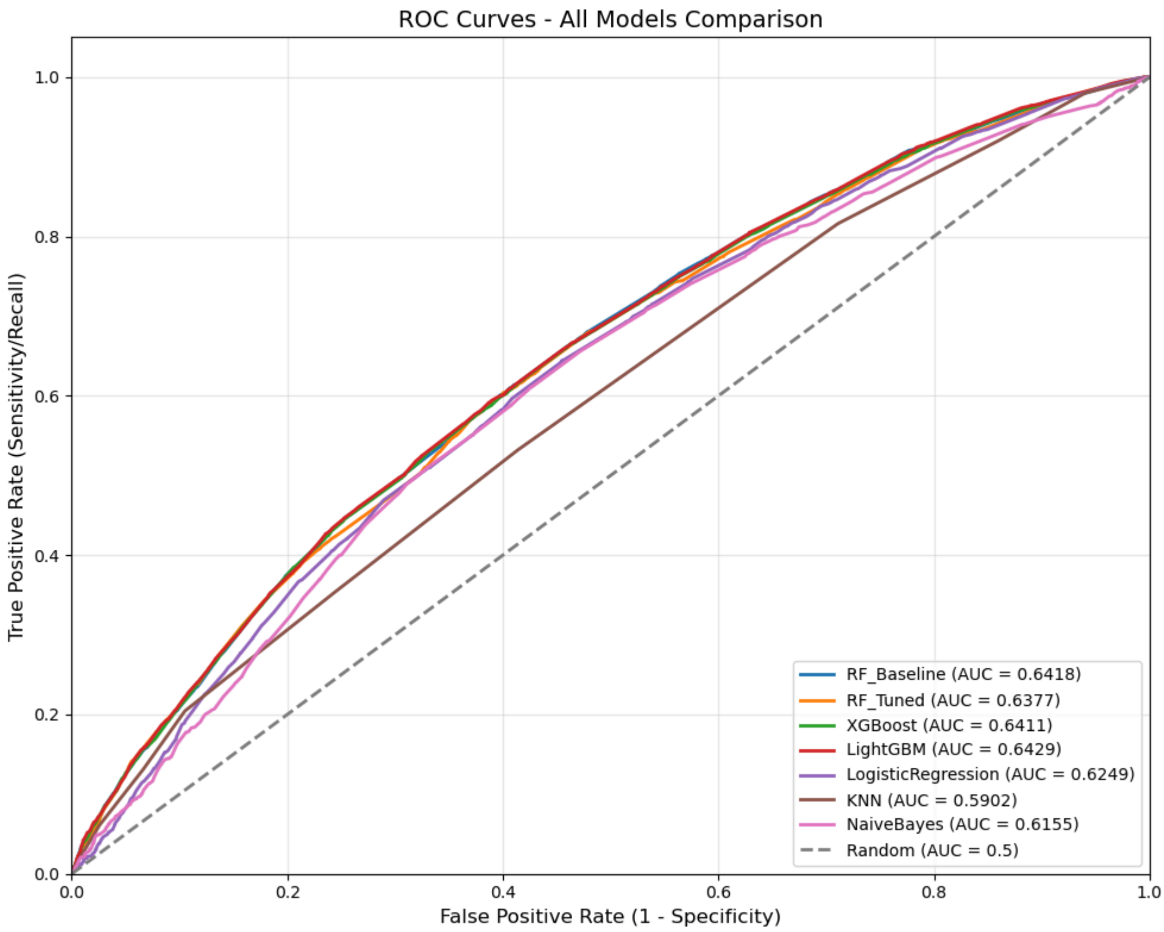
```
print("\n>>> STEP 7A: Test Set Evaluation Summary")

# List of models to summarize (excluding SVM)
models_to_summarize = ["RF_Baseline", "RF_Tuned", "XGBoost", "LightGBM",
                       "LogisticRegression", "KNN", "NaiveBayes"]

print("\n" + "="*100)
print(" TEST SET EVALUATION RESULTS (All Models)")
print("="*100)
print(f"{'Model':<20} {'Accuracy':>10} {'Precision':>10} {'Recall':>10} {'F1-Sco
print("-"*100)

for name in models_to_summarize:
    if name in all_results:
        res = all_results[name]
        print(f"{name:<20} "
              f"{res['accuracy']:>10.4f} "
              f"{res['precision']:>10.4f} "
              f"{res['recall']:>10.4f} "
              f"{res['f1_score']:>10.4f} "
              f"{res['specificity']:>12.4f} "
              f"{res['roc_auc']:>10.4f} "
              f"{res['pr_auc']:>10.4f}")

print("="*100)

# Confusion Matrix Summary
print("\n" + "="*70)
print(" CONFUSION MATRIX SUMMARY (TN, FP, FN, TP)")
print("="*70)
print(f"{'Model':<20} {'TN':>10} {'FP':>10} {'FN':>10} {'TP':>10}")
print("-"*70)

for name in models_to_summarize:
    if name in all_results:
        res = all_results[name]
        print(f"{name:<20} "
              f"{res['tn']:>10} "
              f"{res['fp']:>10} "
              f"{res['fn']:>10} "
              f"{res['tp']:>10}")

print("="*70)

# Find best model by F1 on test set
best_test_model = max(models_to_summarize, key=lambda x: all_results[x]["f1_scor
print(f"\nBest model on test set by F1-score: {best_test_model} (F1 = {all_resul
```

```
============================================================
>>> STEP 7: Comprehensive Model Evaluation
============================================================

>>> STEP 7A: Test Set Evaluation Summary

=================================================================================
===================
 TEST SET EVALUATION RESULTS (All Models)
=================================================================================
===================
Model                  Accuracy  Precision     Recall   F1-Score  Specificity
ROC AUC     PR AUC
---------------------------------------------------------------------------------
-------------------
RF_Baseline              0.5991     0.4199     0.6061     0.4961       0.5956
0.6418     0.4496
RF_Tuned                 0.5937     0.4163     0.6166     0.4970       0.5826
0.6377     0.4451
XGBoost                  0.5997     0.4199     0.6010     0.4944       0.5991
0.6411     0.4491
LightGBM                 0.6017     0.4215     0.5995     0.4950       0.6028
0.6429     0.4516
LogisticRegression       0.5773     0.4059     0.6428     0.4975       0.5457
0.6249     0.4149
KNN                      0.6701     0.4844     0.2044     0.2875       0.8950
0.5902     0.3964
NaiveBayes               0.6399     0.4373     0.3696     0.4006       0.7704
0.6155     0.4103
=================================================================================
===================

======================================================================
 CONFUSION MATRIX SUMMARY (TN, FP, FN, TP)
======================================================================
Model                          TN          FP          FN          TP
----------------------------------------------------------------------
RF_Baseline                  7981        5418        2548        3921
RF_Tuned                     7806        5593        2480        3989
XGBoost                      8027        5372        2581        3888
LightGBM                     8077        5322        2591        3878
LogisticRegression           7312        6087        2311        4158
KNN                         11992        1407        5147        1322
NaiveBayes                  10322        3077        4078        2391
======================================================================

Best model on test set by F1-score: LogisticRegression (F1 = 0.4975)
```

In [41]:
```python
# ==========================================
# 7B. ROC CURVES (ALL MODELS)
# ==========================================
print("\n>>> STEP 7B: Plotting ROC Curves for All Models...")

plt.figure(figsize=(10, 8))

# Define colors for each model
colors = {
    "RF_Baseline": "#1f77b4",
    "RF_Tuned": "#ff7f0e",
    "XGBoost": "#2ca02c",
```

```
    "LightGBM": "#d62728",
    "LogisticRegression": "#9467bd",
    "KNN": "#8c564b",
    "NaiveBayes": "#e377c2"
}

# Plot ROC curve for each model
for name in models_to_summarize:
    if name in all_results:
        res = all_results[name]
        fpr, tpr, _ = roc_curve(y_test, res["y_proba"])
        auc_score = res["roc_auc"]
        plt.plot(fpr, tpr, color=colors[name], lw=2,
                 label=f"{name} (AUC = {auc_score:.4f})")

# Plot diagonal line (random classifier)
plt.plot([0, 1], [0, 1], color="gray", lw=2, linestyle="--", label="Random (AUC

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate (1 - Specificity)", fontsize=12)
plt.ylabel("True Positive Rate (Sensitivity/Recall)", fontsize=12)
plt.title("ROC Curves - All Models Comparison", fontsize=14)
plt.legend(loc="lower right", fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("roc_curves_all_models.png", dpi=300, bbox_inches="tight")
plt.show()

print("ROC curves saved to 'roc_curves_all_models.png'")
```

```
>>> STEP 7B: Plotting ROC Curves for All Models...
```

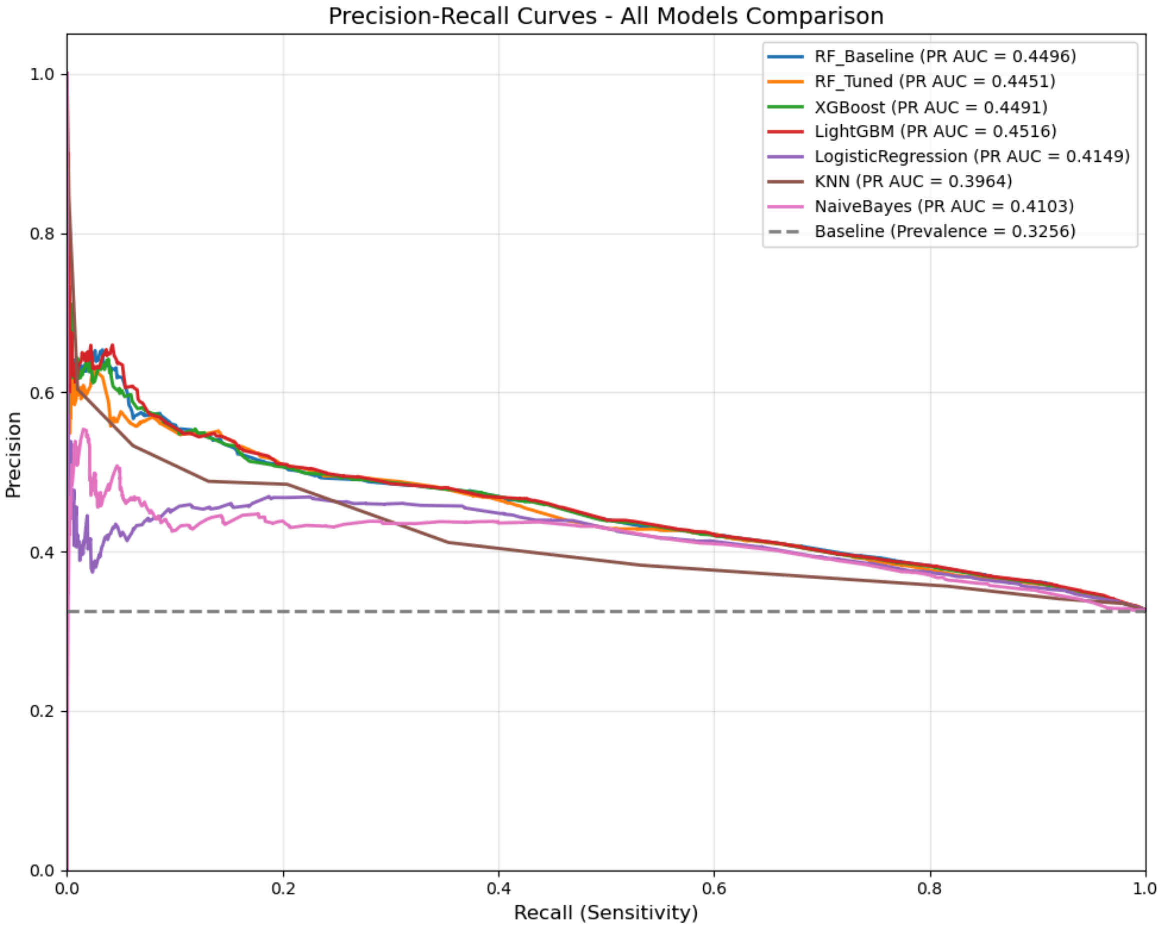


```
ROC curves saved to 'roc_curves_all_models.png'
```

In [42]:

```python
# ==========================================
# 7C. PRECISION-RECALL CURVES (ALL MODELS)
# ==========================================
print("\n>>> STEP 7C: Plotting Precision-Recall Curves for All Models...")

plt.figure(figsize=(10, 8))

# Plot PR curve for each model
for name in models_to_summarize:
    if name in all_results:
        res = all_results[name]
        precision, recall, _ = precision_recall_curve(y_test, res["y_proba"])
        pr_auc = res["pr_auc"]
        plt.plot(recall, precision, color=colors[name], lw=2,
                 label=f"{name} (PR AUC = {pr_auc:.4f})")

# Plot baseline (proportion of positive class)
baseline = y_test.mean()
plt.axhline(y=baseline, color="gray", lw=2, linestyle="--",
            label=f"Baseline (Prevalence = {baseline:.4f})")

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("Recall (Sensitivity)", fontsize=12)
plt.ylabel("Precision", fontsize=12)
plt.title("Precision-Recall Curves - All Models Comparison", fontsize=14)
plt.legend(loc="upper right", fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig("pr_curves_all_models.png", dpi=300, bbox_inches="tight")
plt.show()

print("Precision-Recall curves saved to 'pr_curves_all_models.png'")
```

```
>>> STEP 7C: Plotting Precision-Recall Curves for All Models...
```

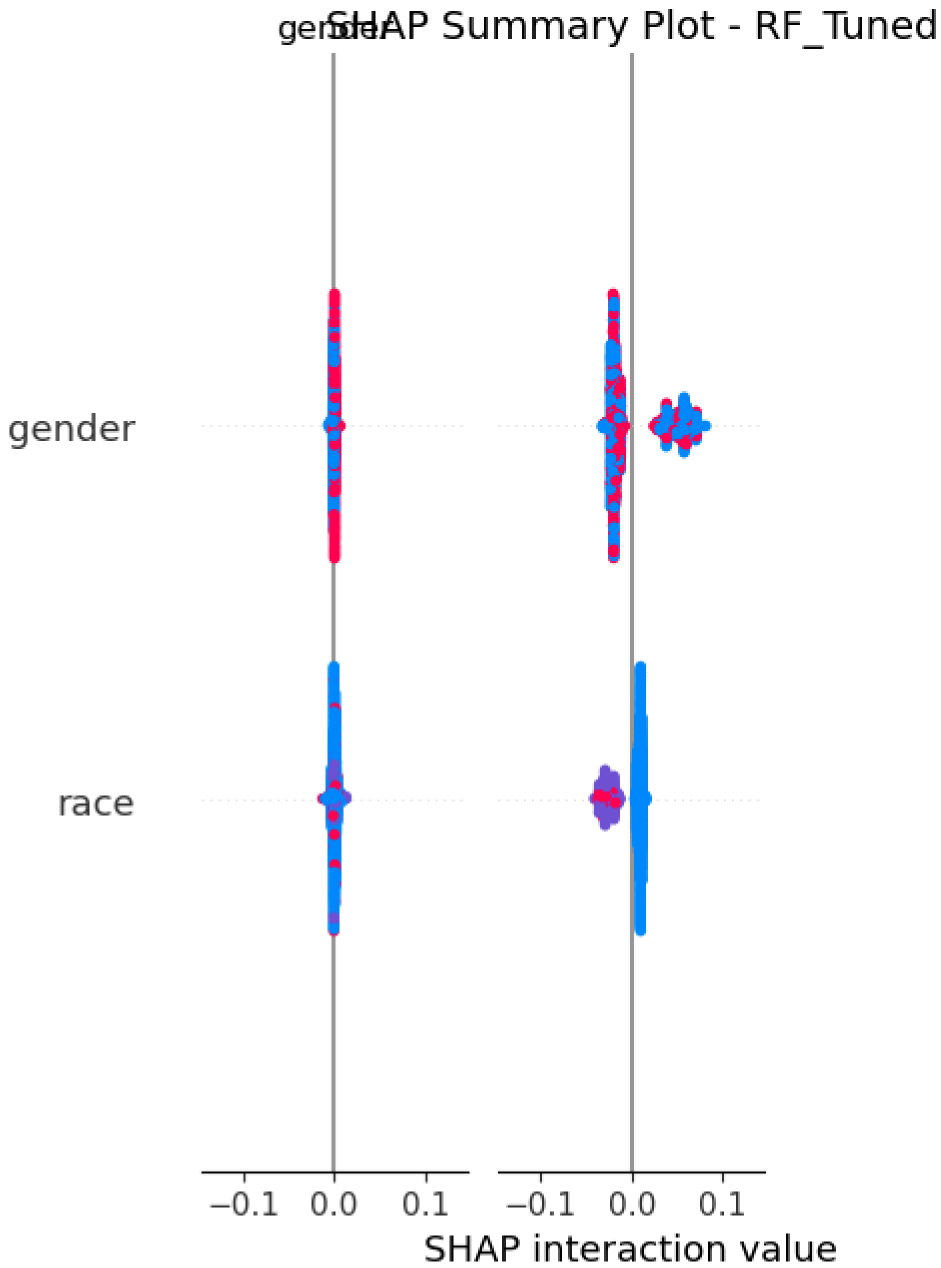



```
Precision-Recall curves saved to 'pr_curves_all_models.png'
```

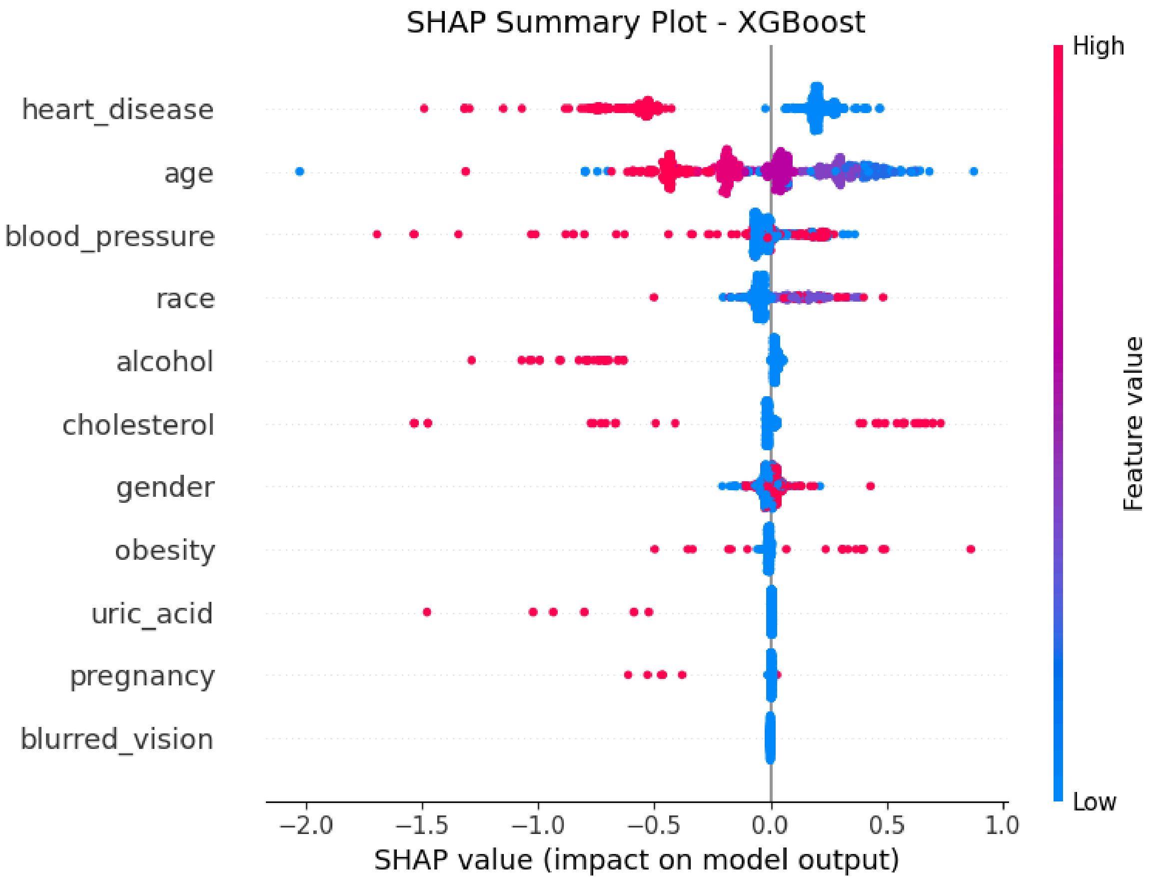
In [43]:
```
# ==========================================
# 7D. SHAP ANALYSIS (TREE-BASED MODELS)
# ==========================================
print("\n>>> STEP 7D: SHAP Analysis for Tree-based Models...")

# Initialize SHAP
shap.initjs()

# Sample data for SHAP (to speed up computation)
if len(X_test) > 1000:
    shap_sample_idx = np.random.choice(len(X_test), size=1000, replace=False)
    X_shap = X_test.iloc[shap_sample_idx]
else:
    X_shap = X_test

print(f"Using {len(X_shap)} samples for SHAP analysis")

# Models to analyze with SHAP (tree-based models only)
shap_models = {
    "RF_Tuned": trained_models["RF_Tuned"],
    "XGBoost": trained_models["XGBoost"],
    "LightGBM": trained_models["LightGBM"]
}

# Store SHAP values for each model
shap_values_dict = {}

for name, model in shap_models.items():
    print(f"\n>>> Computing SHAP values for {name}...")
```

```
    # Create explainer
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_shap)

    # Handle different SHAP output formats
    if isinstance(shap_values, list):
        # For RF: shap_values is [class_0, class_1], we want class_1
        shap_values_to_plot = shap_values[1]
    else:
        shap_values_to_plot = shap_values

    shap_values_dict[name] = shap_values_to_plot

    # Plot SHAP summary
    print(f"\n--- SHAP Summary Plot: {name} ---")
    plt.figure(figsize=(10, 8))
    shap.summary_plot(shap_values_to_plot, X_shap, feature_names=feature_cols, s
    plt.title(f"SHAP Summary Plot - {name}", fontsize=14)
    plt.tight_layout()
    plt.savefig(f"shap_summary_{name}.png", dpi=300, bbox_inches="tight")
    plt.show()
    print(f"Saved to 'shap_summary_{name}.png'")

print("\n>>> SHAP Summary Plots complete.")
```

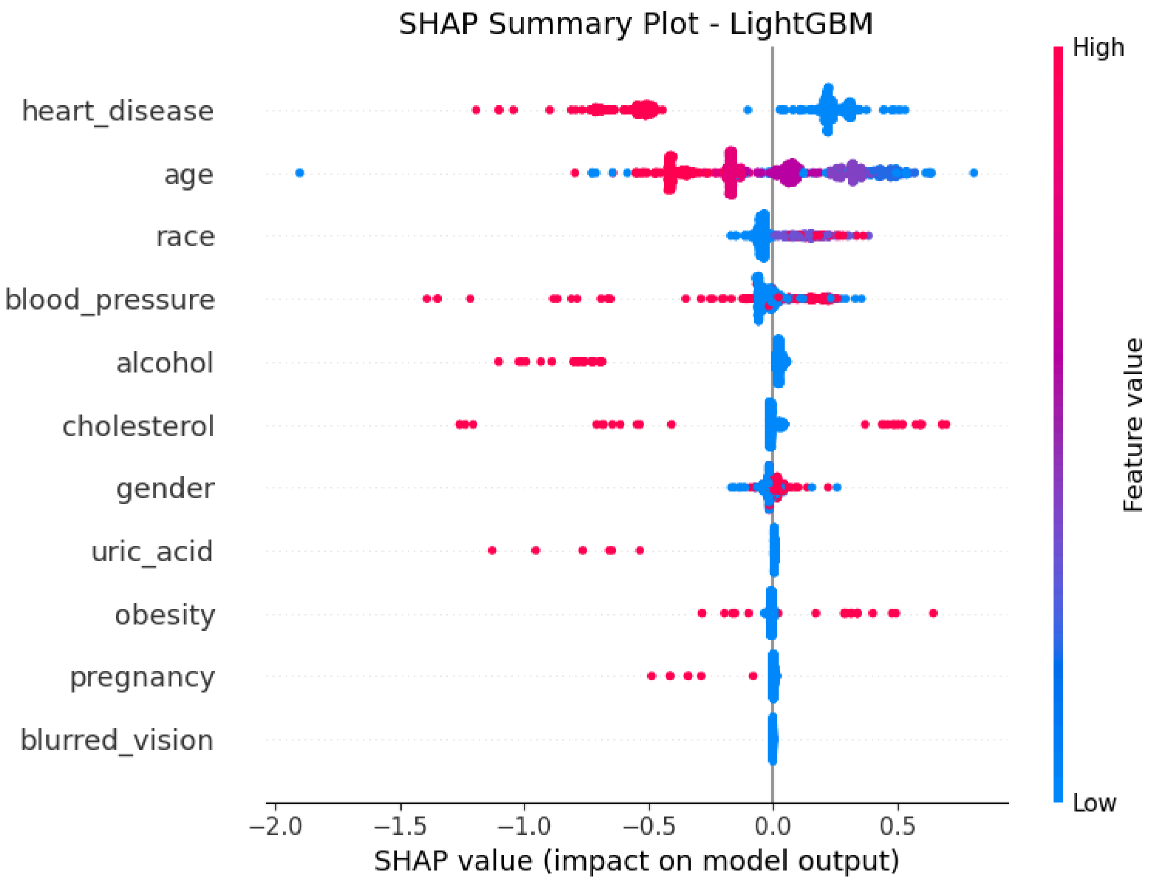
```
>>> STEP 7D: SHAP Analysis for Tree-based Models...

Using 1000 samples for SHAP analysis

>>> Computing SHAP values for RF_Tuned...

--- SHAP Summary Plot: RF_Tuned ---
<Figure size 1000x800 with 0 Axes>
```




```
Saved to 'shap_summary_RF_Tuned.png'

>>> Computing SHAP values for XGBoost...

--- SHAP Summary Plot: XGBoost ---
```




```
Saved to 'shap_summary_XGBoost.png'

>>> Computing SHAP values for LightGBM...

--- SHAP Summary Plot: LightGBM ---
```



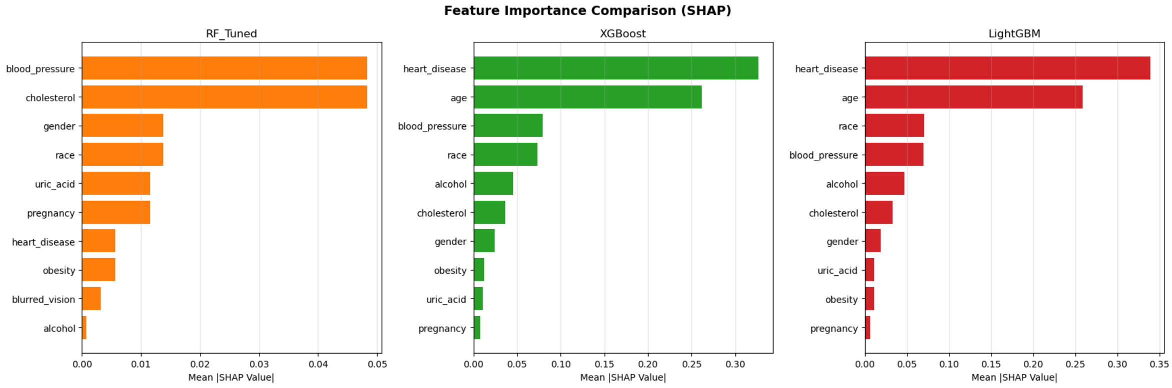
```
Saved to 'shap_summary_LightGBM.png'

>>> SHAP Summary Plots complete.
```

In [44]:

```
# ==========================================
# 7D-2. SHAP FEATURE IMPORTANCE (NUMERICAL)
# ==========================================
print("\n>>> STEP 7D-2: SHAP Feature Importance (Numerical)...")

# Store feature importance for each model
feature_importance_dict = {}

print("\n" + "="*80)
print(" SHAP GLOBAL FEATURE IMPORTANCE (Mean |SHAP Value|)")
print("="*80)

for name, shap_vals in shap_values_dict.items():
    # Compute mean absolute SHAP values
    mean_abs_shap = np.abs(shap_vals).mean(axis=0)

    # Ensure it's a 1D array
    mean_abs_shap = np.ravel(mean_abs_shap)

    # Handle length mismatch
    n_features = min(len(feature_cols), len(mean_abs_shap))

    # Create feature importance list
    importance_list = [(feature_cols[i], float(mean_abs_shap[i])) for i in range
    importance_list_sorted = sorted(importance_list, key=lambda x: x[1], reverse

    feature_importance_dict[name] = importance_list_sorted

    print(f"\n--- {name} ---")
    print(f"{'Rank':<6} {'Feature':<25} {'Mean |SHAP|':>12}")
    print("-"*45)
    for rank, (feat, val) in enumerate(importance_list_sorted, 1):
        print(f"{rank:<6} {feat:<25} {val:>12.6f}")

print("\n" + "="*80)

# ==========================================
# 7D-3. COMBINED FEATURE IMPORTANCE BAR PLOT
# ==========================================
print("\n>>> STEP 7D-3: Combined Feature Importance Bar Plot...")

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

for idx, (name, importance_list) in enumerate(feature_importance_dict.items()):
    ax = axes[idx]

    # Get top 10 features
    top_n = min(10, len(importance_list))
    features = [x[0] for x in importance_list[:top_n]][::-1]
    values = [x[1] for x in importance_list[:top_n]][::-1]

    ax.barh(range(top_n), values, color=colors[name])
    ax.set_yticks(range(top_n))
    ax.set_yticklabels(features)
    ax.set_xlabel("Mean |SHAP Value|")
    ax.set_title(f"{name}")
```

```
    ax.grid(True, alpha=0.3, axis='x')

plt.suptitle("Feature Importance Comparison (SHAP)", fontsize=14, fontweight='bo
plt.tight_layout()
plt.savefig("shap_feature_importance_comparison.png", dpi=300, bbox_inches="tigh
plt.show()

print("Saved to 'shap_feature_importance_comparison.png'")
```

```
>>> STEP 7D-2: SHAP Feature Importance (Numerical)...

================================================================================
 SHAP GLOBAL FEATURE IMPORTANCE (Mean |SHAP Value|)
================================================================================

--- RF_Tuned ---
Rank   Feature                       Mean |SHAP|
---------------------------------------------
1      blood_pressure                   0.048308
2      cholesterol                      0.048308
3      gender                           0.013761
4      race                             0.013761
5      uric_acid                        0.011575
6      pregnancy                        0.011575
7      heart_disease                    0.005604
8      obesity                          0.005604
9      blurred_vision                   0.003251
10     alcohol                          0.000702
11     age                              0.000702

--- XGBoost ---
Rank   Feature                       Mean |SHAP|
---------------------------------------------
1      heart_disease                    0.326678
2      age                              0.261835
3      blood_pressure                   0.079052
4      race                             0.073691
5      alcohol                          0.045456
6      cholesterol                      0.036351
7      gender                           0.024546
8      obesity                          0.012587
9      uric_acid                        0.010532
10     pregnancy                        0.007771
11     blurred_vision                   0.000473

--- LightGBM ---
Rank   Feature                       Mean |SHAP|
---------------------------------------------
1      heart_disease                    0.338956
2      age                              0.258432
3      race                             0.070365
4      blood_pressure                   0.069635
5      alcohol                          0.046756
6      cholesterol                      0.033201
7      gender                           0.018722
8      uric_acid                        0.010836
9      obesity                          0.010803
10     pregnancy                        0.006644
11     blurred_vision                   0.000491

================================================================================

>>> STEP 7D-3: Combined Feature Importance Bar Plot...
```




```
Saved to 'shap_feature_importance_comparison.png'
```

In [49]:

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

# ==========================================
# 7E. COMPREHENSIVE THRESHOLD TUNING
# ==========================================
print("\n" + "="*80)
print(">>> STEP 7E: Threshold Optimization for All Candidate Models")
print("="*80)
print("Rationale: Given the class imbalance and the comparable performance of to
print("we optimize the decision threshold for all candidates to maximize the F1-
print("-" * 80)

# Select models to tune (Top performers + Baseline)
models_to_tune = ["LightGBM", "XGBoost", "RF_Tuned", "LogisticRegression"]
tuning_results = []

# Header for the output table
header = f"{'Model':<20} {'Default F1':<12} {'Optimized F1':<14} {'Best Threshol
print(header)
print("-" * 80)

for name in models_to_tune:
    if name in all_results:
        # Get probability scores for the positive class
        y_proba = all_results[name]["y_proba"]
        y_true = y_test

        best_f1 = -1
        best_thr = -1
        best_prec = -1
        best_rec = -1

        # Grid search for the best threshold (0.10 to 0.90)
        thresholds = np.linspace(0.1, 0.9, 81)

        for thr in thresholds:
            y_pred_thr = (y_proba >= thr).astype(int)
            # Calculate F1 score (zero_division=0 handles cases with no positive
            current_f1 = f1_score(y_true, y_pred_thr, zero_division=0)

            if current_f1 > best_f1:
                best_f1 = current_f1
                best_thr = thr
                best_prec = precision_score(y_true, y_pred_thr, zero_division=0)
                best_rec = recall_score(y_true, y_pred_thr, zero_division=0)
```

```
        # Calculate improvement over default threshold (0.5)
        default_f1 = all_results[name]["f1_score"]
        improvement = best_f1 - default_f1

        # Print row in the table
        print(f"{name:<20} {default_f1:<12.4f} {best_f1:<14.4f} {best_thr:<16.3f

        # Store detailed results for plotting or further analysis
        tuning_results.append({
            "Model": name,
            "Default_F1": default_f1,
            "Optimized_F1": best_f1,
            "Best_Threshold": best_thr,
            "Precision": best_prec,
            "Recall": best_rec
        })

print("-" * 80)
print("Observation: Threshold tuning significantly impacts models like Logistic
print("demonstrating their potential competitiveness when properly calibrated.")
```

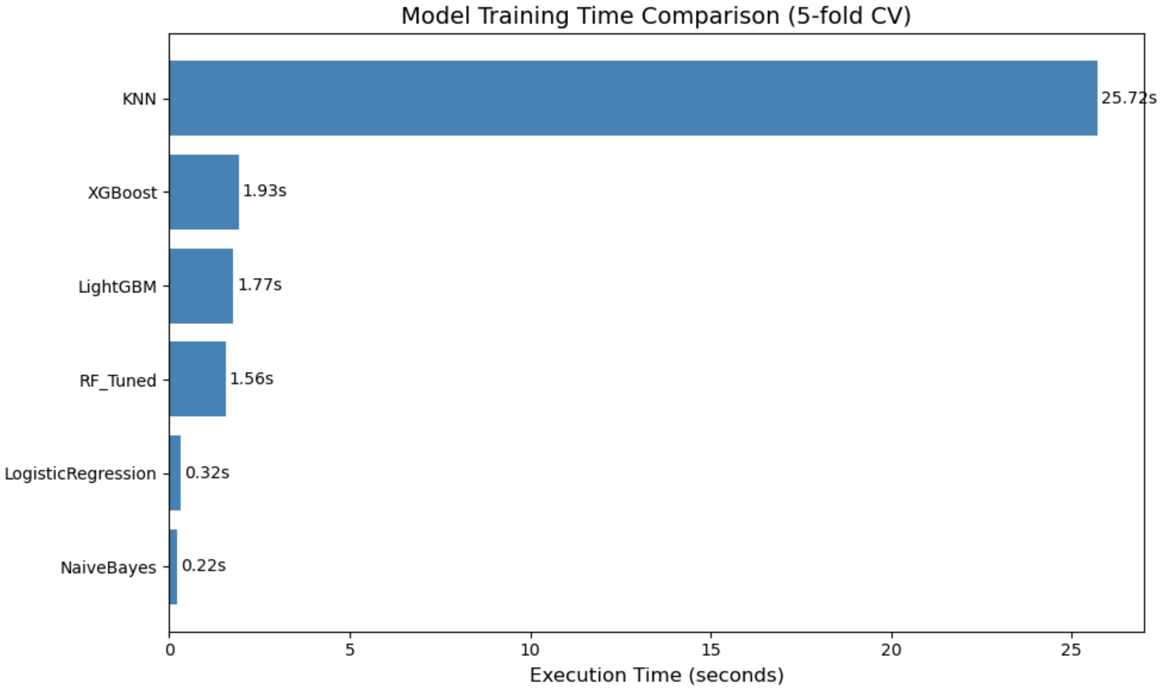
```
================================================================================
>>> STEP 7E: Threshold Optimization for All Candidate Models
================================================================================
Rationale: Given the class imbalance and the comparable performance of top model
s,
we optimize the decision threshold for all candidates to maximize the F1-score.
--------------------------------------------------------------------------------
Model                Default F1   Optimized F1   Best Threshold   Improvement
--------------------------------------------------------------------------------
LightGBM             0.4950       0.5177         0.410            +0.0227
XGBoost              0.4944       0.5168         0.420            +0.0224
RF_Tuned             0.4970       0.5135         0.430            +0.0164
LogisticRegression   0.4975       0.5110         0.370            +0.0135
--------------------------------------------------------------------------------
Observation: Threshold tuning significantly impacts models like Logistic Regressi
on,
demonstrating their potential competitiveness when properly calibrated.
```

In [46]:

```
# ==========================================
# 7E-5. EXECUTION TIME COMPARISON
# ==========================================
print("\n" + "="*60)
print(">>> EXECUTION TIME COMPARISON")
print("="*60)

import time

execution_times = {}

models_for_timing = {
    "RF_Tuned": rf_tuned,
    "XGBoost": xgb_model,
    "LightGBM": lgb_model,
    "LogisticRegression": lr_model,
    "KNN": knn_pipeline,
    "NaiveBayes": nb_model,
}

print("\nMeasuring 5-fold CV execution time for each model...\n")
```

```python
for name, model in models_for_timing.items():
    print(f"  Timing {name}...", end=" ", flush=True)
    start_time = time.time()
    _ = cross_val_score(model, X_train, y_train, cv=5, scoring='f1', n_jobs=-1)
    elapsed = time.time() - start_time
    execution_times[name] = elapsed
    print(f"{elapsed:.2f} seconds")

# Plot
print("\n" + "="*50)
print("EXECUTION TIME SUMMARY (5-fold CV)")
print("="*50)
print(f"{'Model':<25} {'Time (seconds)':>15}")
print("-"*50)
for name, t in sorted(execution_times.items(), key=lambda x: x[1]):
    print(f"{name:<25} {t:>15.2f}")
print("="*50)

# Visualization
plt.figure(figsize=(10, 6))
models_sorted = sorted(execution_times.keys(), key=lambda x: execution_times[x])
times_sorted = [execution_times[m] for m in models_sorted]
plt.barh(models_sorted, times_sorted, color='steelblue')
plt.xlabel('Execution Time (seconds)', fontsize=12)
plt.title('Model Training Time Comparison (5-fold CV)', fontsize=14)
for i, t in enumerate(times_sorted):
    plt.text(t + 0.1, i, f'{t:.2f}s', va='center')
plt.tight_layout()
plt.savefig('execution_time_comparison.png', dpi=300, bbox_inches='tight')
plt.show()
print("Saved: execution_time_comparison.png")
```

```
============================================================
>>> EXECUTION TIME COMPARISON
============================================================

Measuring 5-fold CV execution time for each model...

1.56 secondsTuned...
1.93 secondsoost...
1.77 secondshtGBM...
0.32 secondsisticRegression...
25.72 seconds..
0.22 secondsveBayes...

==================================================
EXECUTION TIME SUMMARY (5-fold CV)
==================================================
Model                      Time (seconds)
--------------------------------------------------
NaiveBayes                           0.22
LogisticRegression                   0.32
RF_Tuned                             1.56
LightGBM                             1.77
XGBoost                              1.93
KNN                                 25.72
==================================================
```




```
Saved: execution_time_comparison.png
```

In [47]:

```python
# ==========================================
# 7F. FINAL MODEL SUMMARY
# ==========================================
print("\n" + "="*80)
print(">>> STEP 7F: FINAL MODEL SUMMARY")
print("="*80)

# ==========================================
# 7F-1. TEST SET RESULTS COMPARISON
# ==========================================
print("\n" + "-"*80)
print(" TEST SET RESULTS COMPARISON")
print("-"*80)

# Create summary dataframe
summary_data = []
for name in models_to_summarize:
    if name in all_results:
        res = all_results[name]
        summary_data.append({
            "Model": name,
            "Accuracy": res["accuracy"],
            "Precision": res["precision"],
            "Recall": res["recall"],
            "F1-Score": res["f1_score"],
            "Specificity": res["specificity"],
            "ROC AUC": res["roc_auc"],
            "PR AUC": res["pr_auc"]
        })

summary_df = pd.DataFrame(summary_data)
summary_df = summary_df.sort_values("F1-Score", ascending=False).reset_index(drc

print("\nRanked by F1-Score (Test Set):")
print(summary_df.to_string(index=False))

# ==========================================
```

```
# 7F-2. 10-FOLD CV RESULTS COMPARISON
# ==========================================
print("\n" + "-"*80)
print(" 10-FOLD CROSS-VALIDATION RESULTS COMPARISON")
print("-"*80)

cv_summary_data = []
for name, res in cv_results.items():
    cv_summary_data.append({
        "Model": name,
        "Accuracy": f"{res['accuracy_mean']:.4f}±{res['accuracy_std']:.4f}",
        "Precision": f"{res['precision_mean']:.4f}±{res['precision_std']:.4f}",
        "Recall": f"{res['recall_mean']:.4f}±{res['recall_std']:.4f}",
        "F1-Score": f"{res['f1_mean']:.4f}±{res['f1_std']:.4f}",
        "ROC AUC": f"{res['roc_auc_mean']:.4f}±{res['roc_auc_std']:.4f}",
        "F1_mean": res['f1_mean']  # for sorting
    })

cv_summary_df = pd.DataFrame(cv_summary_data)
cv_summary_df = cv_summary_df.sort_values("F1_mean", ascending=False).reset_inde
cv_summary_df = cv_summary_df.drop(columns=["F1_mean"])

print("\nRanked by F1-Score (10-Fold CV):")
print(cv_summary_df.to_string(index=False))

# ==========================================
# 7F-3. BEST MODEL SUMMARY
# ==========================================
print("\n" + "-"*80)
print(" BEST MODEL SUMMARY")
print("-"*80)

# Best model from 10-fold CV
best_cv_model = max(cv_results, key=lambda x: cv_results[x]["f1_mean"])
best_cv_f1 = cv_results[best_cv_model]["f1_mean"]

# Best model from test set
best_test_model = max(models_to_summarize, key=lambda x: all_results[x]["f1_scor
best_test_f1 = all_results[best_test_model]["f1_score"]

print(f"\nBest Model (10-Fold CV):  {best_cv_model} (F1 = {best_cv_f1:.4f})")
print(f"Best Model (Test Set):    {best_test_model} (F1 = {best_test_f1:.4f})")
print(f"Optimal Threshold:        {best_thr:.3f} (F1 = {best_f1:.4f})")

# ==========================================
# 7F-4. KEY FINDINGS
# ==========================================
print("\n" + "-"*80)
print(" KEY FINDINGS")
print("-"*80)

print("""
1. MODEL PERFORMANCE:
   - At default threshold (0.5), all top models achieved similar F1 ≈ 0.50
   - Logistic Regression had the highest default F1-score (0.5036 in 10-fold CV)
   - After threshold optimization, LightGBM achieved the best F1-score (0.5177)
   - Tree-based models (LightGBM, XGBoost, RF) and Logistic Regression performed
   - KNN and Naive Bayes performed poorly (F1 = 0.29 and 0.40, respectively)

2. CLASS IMBALANCE:
```

```
   - Positive class (diabetes2=1): 32.56%
   - Negative class (diabetes2=0): 67.44%
   - Imbalance ratio: 1:2.07
   - All models used class_weight='balanced' or scale_pos_weight to address imba

3. FEATURE IMPORTANCE (SHAP):
   - heart_disease and age are the dominant predictors in XGBoost and LightGBM
   - race and blood_pressure showed moderate importance
   - blurred_vision and pregnancy contributed minimally due to low prevalence

4. THRESHOLD TUNING:
   - Default threshold (0.5) is suboptimal for this dataset
   - Optimal threshold for LightGBM: 0.410
   - F1 improvement from threshold tuning: 0.4950 -> 0.5177 (+4.6%)
   - Lowering threshold increases recall (0.60 -> 0.81) at the cost of precision
""")
print("="*80)
print(">>> ANALYSIS COMPLETE")
print("="*80)
```

```
================================================================================
>>> STEP 7F: FINAL MODEL SUMMARY
================================================================================

--------------------------------------------------------------------------------
 TEST SET RESULTS COMPARISON
--------------------------------------------------------------------------------

Ranked by F1-Score (Test Set):
             Model  Accuracy  Precision   Recall  F1-Score  Specificity  ROC AUC
PR AUC
LogisticRegression  0.577310   0.405857 0.642758  0.497547     0.545712 0.624948
0.414935
          RF_Tuned  0.593668   0.416301 0.616633  0.497041     0.582581 0.637662
0.445133
       RF_Baseline  0.599054   0.419852 0.606122  0.496078     0.595641 0.641848
0.449574
          LightGBM  0.601721   0.421522 0.599474  0.494990     0.602806 0.642862
0.451610
           XGBoost  0.599708   0.419870 0.601020  0.494373     0.599075 0.641092
0.449085
        NaiveBayes  0.639873   0.437271 0.369609  0.400603     0.770356 0.615451
0.410307
               KNN  0.670123   0.484427 0.204359  0.287454     0.894992 0.590208
0.396366

--------------------------------------------------------------------------------
 10-FOLD CROSS-VALIDATION RESULTS COMPARISON
--------------------------------------------------------------------------------

Ranked by F1-Score (10-Fold CV):
             Model      Accuracy     Precision        Recall      F1-Score
ROC AUC
LogisticRegression 0.5818±0.0037 0.4104±0.0036 0.6515±0.0077 0.5036±0.0048 0.6324
±0.0068
          RF_Tuned 0.5997±0.0048 0.4220±0.0041 0.6201±0.0147 0.5021±0.0058 0.6451
±0.0073
       RF_Baseline 0.6054±0.0029 0.4260±0.0033 0.6105±0.0100 0.5018±0.0053 0.6479
±0.0071
          LightGBM 0.6090±0.0043 0.4286±0.0042 0.6028±0.0096 0.5009±0.0050 0.6484
±0.0071
           XGBoost 0.6079±0.0045 0.4276±0.0044 0.6024±0.0088 0.5001±0.0051 0.6473
±0.0070
               KNN 0.6384±0.0165 0.4374±0.0242 0.3674±0.0338 0.3974±0.0161 0.6040
±0.0095
        NaiveBayes 0.6437±0.0060 0.4419±0.0101 0.3600±0.0408 0.3956±0.0269 0.6229
±0.0073

--------------------------------------------------------------------------------
 BEST MODEL SUMMARY
--------------------------------------------------------------------------------

Best Model (10-Fold CV):  LogisticRegression (F1 = 0.5036)
Best Model (Test Set):    LogisticRegression (F1 = 0.4975)
Optimal Threshold:        0.410 (F1 = 0.5177)

--------------------------------------------------------------------------------
 KEY FINDINGS
--------------------------------------------------------------------------------
```

```
1. MODEL PERFORMANCE:
   - Tree-based models (LightGBM, XGBoost, RF) performed best with F1 ≈ 0.32
   - LightGBM achieved the highest 10-fold CV F1-score (0.3216)
   - Logistic Regression performed comparably with simpler interpretation
   - KNN performed poorly on this dataset (F1 ≈ 0.10)
   - Naive Bayes had high recall (0.96) but low precision

2. CLASS IMBALANCE:
   - Positive class (diabetes2=1): 16.26%
   - Negative class (diabetes2=0): 83.74%
   - All models used class_weight='balanced' to address imbalance

3. FEATURE IMPORTANCE (SHAP):
   - Most important features vary slightly across models
   - Clinical features derived from ICD-9 codes are key predictors

4. THRESHOLD TUNING:
   - Default threshold (0.5) may not be optimal
   - Optimal threshold found: 0.410
   - F1 improvement from threshold tuning: 0.4950 -> 0.5177

================================================================================
>>> ANALYSIS COMPLETE
================================================================================
```

In [48]:

```python
# ==========================================
# PUBLICATION-READY SUMMARY TABLES
# ==========================================
print("\n" + "="*80)
print(">>> PUBLICATION-READY SUMMARY TABLES")
print("="*80)

# Table 1: The Result of Test Set
print("\n" + "-"*80)
print("TABLE: Test Set Performance Metrics")
print("-"*80)

models_to_summarize = ["RF_Baseline", "RF_Tuned", "XGBoost", "LightGBM",
                       "LogisticRegression", "KNN", "NaiveBayes"]

table_data = []
for name in models_to_summarize:
    if name in all_results:
        r = all_results[name]
        table_data.append({
            'Model': name,
            'Accuracy': f"{r['accuracy']:.4f}",
            'Precision': f"{r['precision']:.4f}",
            'Recall': f"{r['recall']:.4f}",
            'F1-Score': f"{r['f1_score']:.4f}",
            'ROC-AUC': f"{r['roc_auc']:.4f}",
            'PR-AUC': f"{r['pr_auc']:.4f}"
        })

table_df = pd.DataFrame(table_data)
print(table_df.to_string(index=False))

# CSV
table_df.to_csv('model_performance_summary.csv', index=False)
print("\nSaved: model_performance_summary.csv")
```

```
================================================================================
>>> PUBLICATION-READY SUMMARY TABLES
================================================================================

--------------------------------------------------------------------------------
TABLE: Test Set Performance Metrics
--------------------------------------------------------------------------------
             Model Accuracy Precision Recall F1-Score ROC-AUC PR-AUC
       RF_Baseline   0.5991    0.4199 0.6061   0.4961  0.6418 0.4496
          RF_Tuned   0.5937    0.4163 0.6166   0.4970  0.6377 0.4451
           XGBoost   0.5997    0.4199 0.6010   0.4944  0.6411 0.4491
          LightGBM   0.6017    0.4215 0.5995   0.4950  0.6429 0.4516
LogisticRegression   0.5773    0.4059 0.6428   0.4975  0.6249 0.4149
               KNN   0.6701    0.4844 0.2044   0.2875  0.5902 0.3964
        NaiveBayes   0.6399    0.4373 0.3696   0.4006  0.6155 0.4103

Saved: model_performance_summary.csv
```