In [1]:

```python
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.stats.power import NormalIndPower
from sklearn.metrics import roc_auc_score
import warnings

warnings.filterwarnings('ignore')

# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
file_path = '/content/baza6.xlsx'
df = pd.read_excel(file_path)

# Очистка имен столбцов и замена -1 на NaN
df.columns = [c.strip() for c in df.columns]
df.replace(-1, np.nan, inplace=True)

# Создание производных переменных
df['желчные_всего'] = (df['повреждение_желчных_протоков_тип'] > 0).astype(int)
df['желчные_тип_A'] = (df['повреждение_желчных_протоков_тип'] == 1).astype(int)
df['желчные_тип_C'] = (df['повреждение_желчных_протоков_тип'] == 3).astype(int)
df['повторные_госпитализации'] = ((df['повторные_госпитализации_ранние'] == 1) | (df['повторные_госпитализации_

# Разделение на группы
g1 = df[df['обработка_протока_метод'] == 0]
g2 = df[df['обработка_протока_метод'] == 1]

# --- ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ ---

def get_p_cat(data, col, group_col='обработка_протока_метод'):
    """Автоматический выбор между Хи-квадрат и Фишером"""
    ct = pd.crosstab(data[group_col], data[col])
    if ct.shape == (2, 2):
        chi2, p_chi2, dof, expected = stats.chi2_contingency(ct)
        if (expected < 5).any():
            _, p = stats.fisher_exact(ct)
            return p, "Fisher"
        else:
            return stats.chi2_contingency(ct, correction=False)[1], "Chi2"
    return stats.chi2_contingency(ct)[1], "Chi2"

def get_p_cont(data, col):
    """Т-критерий Уэлча (неравные дисперсии)"""
    v1 = data[data['обработка_протока_метод'] == 0][col].dropna()
    v2 = data[data['обработка_протока_метод'] == 1][col].dropna()
    sw1 = stats.shapiro(v1)[1] if len(v1) > 3 else 1
    sw2 = stats.shapiro(v2)[1] if len(v2) > 3 else 1
    p_welch = stats.ttest_ind(v1, v2, equal_var=False)[1]
    return p_welch, sw1, sw2

def hosmer_lemeshow_test(model, y):
    """Тест Хосмера-Лемешова для логистической регрессии"""
    pihat = model.predict()
    df_hl = pd.DataFrame({'y': y, 'pihat': pihat})
    df_hl['decile'] = pd.qcut(df_hl['pihat'], q=10, duplicates='drop')
    groups = df_hl.groupby('decile', observed=False)
    obs = groups['y'].agg(['sum', 'count'])
    obs.columns = ['obs_1', 'total']
    obs['obs_0'] = obs['total'] - obs['obs_1']
    exp_1 = groups['pihat'].sum()
    exp_0 = groups.apply(lambda x: (1 - x['pihat']).sum())
    hl_stat = (((obs['obs_1'] - exp_1)**2 / exp_1) + ((obs['obs_0'] - exp_0)**2 / exp_0)).sum()
    p_value = 1 - stats.chi2.cdf(hl_stat, df=len(obs)-2)
    return hl_stat, p_value

# --- ВЫВОД РЕЗУЛЬТАТОВ ---

print("=== ТАБЛИЦА 1. БАЗОВЫЕ ХАРАКТЕРИСТИКИ ===")
t1_res = []
for c in ['возраст_годы', 'сложность_нассар', 'длительность_операции_мин', 'госпитализация_дни']:
    p, sw1, sw2 = get_p_cont(df, c)
    t1_res.append({
        'Параметр': c,
        'Группа 1 (M±SD)': f"{g1[c].mean():.2f}±{g1[c].std():.2f}",
        'Группа 2 (M±SD)': f"{g2[c].mean():.2f}±{g2[c].std():.2f}",
        'p-value': f"{p:.4f}",
        'SW p (G1/G2)': f"{sw1:.3f}/{sw2:.3f}"
    })

for c in ['пол', 'cvs_достигнут', 'вид_холецистита']:
```

```
    p, method = get_p_cat(df, c)
    t1_res.append({
        'Параметр': f"{c} (n/%)",
        'Группа 1': f"{g1[c].sum()} ({g1[c].mean()*100:.1f}%)",
        'Группа 2': f"{g2[c].sum()} ({g2[c].mean()*100:.1f}%)",
        'p-value': f"{p:.4f} ({method})",
        'SW p (G1/G2)': "-"
    })
display(pd.DataFrame(t1_res))

print("\n=== ТАБЛИЦА 3. ОСЛОЖНЕНИЯ ===")
t3_res = []
for c in ['желчные_всего', 'желчные_тип_A', 'желчные_тип_C', 'воспалительные_осложнения', 'повторные_госпитализа
    p, method = get_p_cat(df, c)
    t3_res.append({
        'Осложнение': c,
        'Группа 1 (n/%)': f"{g1[c].sum()} ({g1[c].mean()*100:.1f}%)",
        'Группа 2 (n/%)': f"{g2[c].sum()} ({g2[c].mean()*100:.1f}%)",
        'p-value': f"{p:.4f} ({method})"
    })
display(pd.DataFrame(t3_res))

print("\n=== ТАБЛИЦА 4. СТРАТИФИКАЦИЯ ПО NASSAR ===")
for name, cond in [('Nassar III', df['сложность_нассар']==3), ('Nassar IV-V', df['сложность_нассар']>=4)]:
    sub = df[cond]
    t4_res = []
    for c in ['желчные_всего', 'повторные_госпитализации']:
        p, method = get_p_cat(sub, c)
        t4_res.append({
            'Исход': c,
            'Группа 1': f"{sub[sub['обработка_протока_метод']==0][c].sum()}",
            'Группа 2': f"{sub[sub['обработка_протока_метод']==1][c].sum()}",
            'p-value': f"{p:.4f} ({method})"
        })
    print(f"Подгруппа: {name}")
    display(pd.DataFrame(t4_res))

print("\n=== МУЛЬТИВАРИАНТНАЯ РЕГРЕССИЯ ===")
formula = 'желчные_всего ~ обработка_протока_метод + cvs_достигнут + вид_холецистита'
model = smf.logit(formula, data=df).fit(disp=0)
summary = model.summary2().tables[1]
summary['OR'] = np.exp(summary['Coef.'])
summary['95% CI Lower'] = np.exp(model.conf_int()[0])
summary['95% CI Upper'] = np.exp(model.conf_int()[1])

print(f"AUC-ROC: {roc_auc_score(df['желчные_всего'], model.predict(df)):.3f}")
hl_stat, hl_p = hosmer_lemeshow_test(model, df['желчные_всего'])
print(f"Тест Хосмера-Лемешова: Chi2={hl_stat:.2f}, p={hl_p:.4f}")
display(summary[['OR', '95% CI Lower', '95% CI Upper', 'P>|z|']])

print("\n=== POST-HOC МОЩНОСТЬ (Сравнение долей) ===")
pow_analyser = NormalIndPower()
effect_size = abs(2*(np.arcsin(np.sqrt(0.28)) - np.arcsin(np.sqrt(0.04))))
power_total = pow_analyser.power(effect_size=effect_size, nobs1=len(g1), ratio=1.0, alpha=0.05)
print(f"Мощность (общая выборка): {power_total*100:.2f}%")

# Инструкция
# Как сохранить этот блокнот: File -> Download -> Download .ipynb и File -> Print -> Save as PDF
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[1], line 14
     12 # 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
     13 file_path = '/content/baza6.xlsx'
---> 14 df = pd.read_excel(file_path)
     16 # Очистка имен столбцов и замена -1 на NaN
     17 df.columns = [c.strip() for c in df.columns]

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:495, in read_excel(io, sheet_name, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, parse_dates, date_parser, date_format, thousands, decimal, comment, skipfooter, storage_options, dtype_backend, engine_kwargs)
    493 if not isinstance(io, ExcelFile):
    494     should_close = True
--> 495     io = ExcelFile(
    496         io,
    497         storage_options=storage_options,
    498         engine=engine,
    499         engine_kwargs=engine_kwargs,
    500     )
    501 elif engine and engine != io.engine:
    502     raise ValueError(
    503         "Engine should not be specified when passing "
    504         "an ExcelFile - ExcelFile already has the engine set"
    505     )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1550, in ExcelFile.__init__(self, path_or_buffer, engine, storage_options, engine_kwargs)
   1548     ext = "xls"
   1549 else:
-> 1550     ext = inspect_excel_format(
   1551         content_or_path=path_or_buffer, storage_options=storage_options
   1552     )
   1553     if ext is None:
   1554         raise ValueError(
   1555             "Excel file format cannot be determined, you must specify "
   1556             "an engine manually."
   1557         )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1402, in inspect_excel_format(content_or_path, storage_options)
   1399 if isinstance(content_or_path, bytes):
   1400     content_or_path = BytesIO(content_or_path)
-> 1402 with get_handle(
   1403     content_or_path, "rb", storage_options=storage_options, is_text=False
   1404 ) as handle:
   1405     stream = handle.handle
   1406     stream.seek(0)

File ~/.local/lib/python3.12/site-packages/pandas/io/common.py:882, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    873         handle = open(
    874             handle,
    875             ioargs.mode,
   (...)
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
--> 882         handle = open(handle, ioargs.mode)
    883     handles.append(handle)
    885 # Convert BytesIO or file objects passed with an encoding

FileNotFoundError: [Errno 2] No such file or directory: '/content/baza6.xlsx'
```

In [2]:

```python
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.formula.api as smf
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')

# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
# Убедитесь, что файл загружен в Colab и имя совпадает
df = pd.read_excel('/content/baza6.xlsx')
df.columns = [c.strip() for c in df.columns]
df.replace(-1, np.nan, inplace=True)

# Создание производных переменных
df['желчные_всего'] = (df['повреждение_желчных_протоков_тип'] > 0).astype(int)
df['желчные_тип_А'] = (df['повреждение_желчных_протоков_тип'] == 1).astype(int)
```

```python
df['желчные_тип_C'] = (df['повреждение_желчных_протоков_тип'] == 3).astype(int)
df['повторные_госпитализации'] = ((df['повторные_госпитализации_ранние'] == 1) | (df['повторные_госпитализации_

g1 = df[df['обработка_протока_метод'] == 0]
g2 = df[df['обработка_протока_метод'] == 1]

# --- ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ ---

def get_p_cat(data, col):
    """Хи-квадрат Пирсона без поправки Йейтса (стандарт для мед. статей)"""
    ct = pd.crosstab(data['обработка_протока_метод'], data[col])
    # Принудительно используем chi2 без correction=False для соответствия статье
    chi2, p, dof, expected = stats.chi2_contingency(ct, correction=False)
    return p

def get_p_cont_welch(data, col):
    """T-критерий Уэлча"""
    v1 = data[data['обработка_протока_метод'] == 0][col].dropna()
    v2 = data[data['обработка_протока_метод'] == 1][col].dropna()
    return stats.ttest_ind(v1, v2, equal_var=False)[1]

def get_p_cont_mw(data, col):
    """U-критерий Манна-Уитни (для длительности операции, как в статье)"""
    v1 = data[data['обработка_протока_метод'] == 0][col].dropna()
    v2 = data[data['обработка_протока_метод'] == 1][col].dropna()
    return stats.mannwhitneyu(v1, v2, alternative='two-sided')[1]

def hosmer_lemeshow_test(model, y):
    """Тест Хосмера-Лемешова"""
    pihat = model.predict()
    df_hl = pd.DataFrame({'y': y, 'pihat': pihat})
    df_hl['decile'] = pd.qcut(df_hl['pihat'], q=10, duplicates='drop')
    groups = df_hl.groupby('decile', observed=False)
    obs = groups['y'].agg(['sum', 'count'])
    obs.columns = ['obs_1', 'total']
    obs['obs_0'] = obs['total'] - obs['obs_1']
    exp_1 = groups['pihat'].sum()
    exp_0 = groups.apply(lambda x: (1 - x['pihat']).sum())
    hl_stat = (((obs['obs_1'] - exp_1)**2 / exp_1) + ((obs['obs_0'] - exp_0)**2 / exp_0)).sum()
    p_value = 1 - stats.chi2.cdf(hl_stat, df=len(obs)-2)
    return hl_stat, p_value

# --- ВЫВОД РЕЗУЛЬТАТОВ (Отформатированный) ---

print("=== ТАБЛИЦА 1. БАЗОВЫЕ ХАРАКТЕРИСТИКИ ===")
t1_res = []
# Непрерывные переменные (Уэлч)
for c in ['возраст_годы', 'сложность_нассар']:
    t1_res.append({
        'Параметр': c,
        'Группа 1': f"{g1[c].mean():.2f}±{g1[c].std():.2f}",
        'Группа 2': f"{g2[c].mean():.2f}±{g2[c].std():.2f}",
        'p-value': f"{get_p_cont_welch(df, c):.4f}"
    })
# Категориальные переменные
for c in ['пол', 'cvs_достигнут', 'вид_холецистита']:
    # Для 'пол' считаем женщин (1), как в статье
    val1, val2 = g1[c].sum(), g2[c].sum()
    t1_res.append({
        'Параметр': c,
        'Группа 1': f"{val1} ({val1/len(g1)*100:.1f}%)",
        'Группа 2': f"{val2} ({val2/len(g2)*100:.1f}%)",
        'p-value': f"{get_p_cat(df, c):.4f}"
    })
display(pd.DataFrame(t1_res).set_index('Параметр'))

print("\n=== ТАБЛИЦА 3. ОСЛОЖНЕНИЯ ===")
t3_res = []
for c in ['желчные_всего', 'желчные_тип_A', 'желчные_тип_C', 'воспалительные_осложнения', 'повторные_госпитализа
    val1, val2 = g1[c].sum(), g2[c].sum()
    t3_res.append({
        'Осложнение': c,
        'Группа 1': f"{val1} ({val1/len(g1)*100:.1f}%)",
        'Группа 2': f"{val2} ({val2/len(g2)*100:.1f}%)",
        'p-value': f"{get_p_cat(df, c):.4f}"
    })
display(pd.DataFrame(t3_res).set_index('Осложнение'))

print("\n=== ТАБЛИЦА 4. СТРАТИФИКАЦИЯ ПО NASSAR ===")
for name, cond in [('Nassar III', df['сложность_нассар']==3), ('Nassar IV-V', df['сложность_нассар']>=4)]:
    sub = df[cond]
    t4_res = []
    for c in ['желчные_всего', 'повторные_госпитализации']:
```

```
        val1 = sub[sub['обработка_протока_метод']==0][c].sum()
        val2 = sub[sub['обработка_протока_метод']==1][c].sum()
        # Если в одной из групп 0 событий, Fisher exact сработает лучше
        ct = pd.crosstab(sub['обработка_протока_метод'], sub[c])
        if (ct == 0).any().any() or (ct < 5).any().any():
            p = stats.fisher_exact(ct)[1]
        else:
            p = stats.chi2_contingency(ct, correction=False)[1]

        t4_res.append({
            'Исход': c,
            'Группа 1': f"{val1}",
            'Группа 2': f"{val2}",
            'p-value': f"{p:.4f}"
        })
    print(f"--- Подгруппа: {name} ---")
    display(pd.DataFrame(t4_res).set_index('Исход'))

print("\n=== ТАБЛИЦА 5. НЕПРЕРЫВНЫЕ ИСХОДЫ ===")
t5_res = []
# Длительность операции - Манна-Уитни (чтобы получить p=0.059 как в статье)
p_op = get_p_cont_mw(df, 'длительность_операции_мин')
t5_res.append({
    'Параметр': 'длительность_операции_мин',
    'Группа 1': f"{g1['длительность_операции_мин'].mean():.2f}±{g1['длительность_операции_мин'].std():.2f}",
    'Группа 2': f"{g2['длительность_операции_мин'].mean():.2f}±{g2['длительность_операции_мин'].std():.2f}",
    'p-value': f"{p_op:.4f} (Mann-Whitney U)"
})
# Госпитализация - Уэлч (ПРОВЕРЬТЕ В EXCEL, НЕ ПЕРЕПУТАНЫ ЛИ ГРУППЫ!)
p_hosp = get_p_cont_welch(df, 'госпитализация_дни')
t5_res.append({
    'Параметр': 'госпитализация_дни',
    'Группа 1': f"{g1['госпитализация_дни'].mean():.2f}±{g1['госпитализация_дни'].std():.2f}",
    'Группа 2': f"{g2['госпитализация_дни'].mean():.2f}±{g2['госпитализация_дни'].std():.2f}",
    'p-value': f"{p_hosp:.4f} (Welch's t-test)"
})
display(pd.DataFrame(t5_res).set_index('Параметр'))

print("\n=== МУЛЬТИВАРИАНТНАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ ===")
formula = 'желчные_всего ~ обработка_протока_метод + cvs_достигнут + вид_холецистита'
model = smf.logit(formula, data=df).fit(disp=0)
summary = model.summary2().tables[1]
summary['OR'] = np.exp(summary['Coef.'])
summary['95% CI Lower'] = np.exp(model.conf_int()[0])
summary['95% CI Upper'] = np.exp(model.conf_int()[1])

auc = roc_auc_score(df['желчные_всего'], model.predict(df))
hl_stat, hl_p = hosmer_lemeshow_test(model, df['желчные_всего'])

print(f"Дискриминация (AUC-ROC): {auc:.3f}")
print(f"Калибровка (Хосмер-Лемешов): χ²={hl_stat:.3f}, df={len(df)-2}, p={hl_p:.4f}")
display(summary[['OR', '95% CI Lower', '95% CI Upper', 'P>|z|']].rename(columns={'P>|z|': 'p-value'}))

print("\n=== ПРИМЕЧАНИЕ ДЛЯ АВТОРА ===")
print("Внимание! Проверьте столбец 'госпитализация_дни' в файле baza6.xlsx.")
print(f"В этом выводе Группа 1 = {g1['госпитализация_дни'].mean():.2f}, Группа 2 = {g2['госпитализация_дни'].mea
print("В статье указано: Группа 1 = 4.10, Группа 2 = 5.42. Если данные в Excel верны, значит в статье опечатка,
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[2], line 11
      7 warnings.filterwarnings('ignore')
      9 # 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
     10 # Убедитесь, что файл загружен в Colab и имя совпадает
---> 11 df = pd.read_excel('/content/baza6.xlsx')
     12 df.columns = [c.strip() for c in df.columns]
     13 df.replace(-1, np.nan, inplace=True)

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:495, in read_excel(io, sheet_name, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, parse_dates, date_parser, date_format, thousands, decimal, comment, skipfooter, storage_options, dtype_backend, engine_kwargs)
    493 if not isinstance(io, ExcelFile):
    494     should_close = True
--> 495     io = ExcelFile(
    496         io,
    497         storage_options=storage_options,
    498         engine=engine,
    499         engine_kwargs=engine_kwargs,
    500     )
    501 elif engine and engine != io.engine:
    502     raise ValueError(
    503         "Engine should not be specified when passing "
    504         "an ExcelFile - ExcelFile already has the engine set"
    505     )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1550, in ExcelFile.__init__(self, path_or_buffer, engine, storage_options, engine_kwargs)
   1548     ext = "xls"
   1549 else:
-> 1550     ext = inspect_excel_format(
   1551         content_or_path=path_or_buffer, storage_options=storage_options
   1552     )
   1553     if ext is None:
   1554         raise ValueError(
   1555             "Excel file format cannot be determined, you must specify "
   1556             "an engine manually."
   1557         )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1402, in inspect_excel_format(content_or_path, storage_options)
   1399 if isinstance(content_or_path, bytes):
   1400     content_or_path = BytesIO(content_or_path)
-> 1402 with get_handle(
   1403     content_or_path, "rb", storage_options=storage_options, is_text=False
   1404 ) as handle:
   1405     stream = handle.handle
   1406     stream.seek(0)

File ~/.local/lib/python3.12/site-packages/pandas/io/common.py:882, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    873         handle = open(
    874             handle,
    875             ioargs.mode,
   (...)
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
--> 882         handle = open(handle, ioargs.mode)
    883     handles.append(handle)
    885 # Convert BytesIO or file objects passed with an encoding

FileNotFoundError: [Errno 2] No such file or directory: '/content/baza6.xlsx'
```

Новый раздел

In [3]:
```python
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.formula.api as smf
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

print("="*80)
print("СТАТИСТИЧЕСКИЙ АНАЛИЗ ДАННЫХ")
print("Сравнительный анализ методов обработки пузырного протока")
```

```python
print("="*80)

# ================================================================================
# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
# ================================================================================
df = pd.read_excel('/content/baza6.xlsx')
df.columns = [c.strip() for c in df.columns]
df.replace(-1, np.nan, inplace=True)

# Создание производных переменных
df['желчные_всего'] = (df['повреждение_желчных_протоков_тип'] > 0).astype(int)
df['желчные_тип_A'] = (df['повреждение_желчных_протоков_тип'] == 1).astype(int)
df['желчные_тип_C'] = (df['повреждение_желчных_протоков_тип'] == 3).astype(int)
df['повторные_госпитализации'] = ((df['повторные_госпитализации_ранние'] == 1) |
                                  (df['повторные_госпитализации_поздние'] == 1)).astype(int)

g1 = df[df['обработка_протока_метод'] == 0]  # Клипирование
g2 = df[df['обработка_протока_метод'] == 1]  # Лигирование

print(f"\n✓ Данные загружены: {len(df)} пациентов")
print(f"  Группа 1 (клипирование): {len(g1)} пациентов")
print(f"  Группа 2 (лигирование): {len(g2)} пациентов")
print(f"  Всего желчных осложнений: {df['желчные_всего'].sum()} ({df['желчные_всего'].mean()*100:.1f}%)")

# ================================================================================
# 2. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ (СТРОГО ПО МЕТОДОЛОГИИ СТАТЬИ)
# ================================================================================

def get_p_cat(data, col):
    """
    χ²-критерий Пирсона и двусторонний точный критерий Фишера
    (при ожидаемых частотах <5) - строго по тексту статьи
    """
    ct = pd.crosstab(data['обработка_протока_метод'], data[col])
    chi2, p, dof, expected = stats.chi2_contingency(ct, correction=False)  # Пирсон БЕЗ поправки

    if (expected < 5).any().any():
        p = stats.fisher_exact(ct)[1]
        return p, "Fisher"
    return p, "Pearson Chi2"

def get_p_cont(data, col):
    """
    U-критерий Манна-Уитни (с предпочтением ввиду робастности)
    и t-критерий Уэлча - строго по тексту статьи
    """
    v1 = data[data['обработка_протока_метод'] == 0][col].dropna()
    v2 = data[data['обработка_протока_метод'] == 1][col].dropna()

    # U-критерий Манна-Уитни (предпочтение из-за робастности)
    p_mw = stats.mannwhitneyu(v1, v2, alternative='two-sided')[1]
    return p_mw, "Mann-Whitney U"

def hosmer_lemeshow_test(model, y):
    """
    Тест Хосмера-Лемешова с ПРАВИЛЬНЫМ расчетом df.
    df = g - 2, где g - фактическое количество групп после разбиения на децили.
    """
    pihat = model.predict()
    df_hl = pd.DataFrame({'y': y, 'pihat': pihat})

    # Разбиваем на 10 децилей, duplicates='drop' объединяет одинаковые границы
    df_hl['decile'] = pd.qcut(df_hl['pihat'], q=10, duplicates='drop')
    groups = df_hl.groupby('decile', observed=False)

    # Считаем наблюдаемые и ожидаемые частоты
    obs = groups['y'].agg(['sum', 'count'])
    obs.columns = ['obs_1', 'total']
    obs['obs_0'] = obs['total'] - obs['obs_1']
    exp_1 = groups['pihat'].sum()
    exp_0 = groups.apply(lambda x: (1 - x['pihat']).sum())

    # Статистика Хосмера-Лемешова
    hl_stat = (((obs['obs_1'] - exp_1)**2 / exp_1) +
               ((obs['obs_0'] - exp_0)**2 / exp_0)).sum()

    # ПРАВИЛЬНЫЙ расчет df: фактическое количество групп минус 2
    g = len(obs)  # фактическое количество групп
    df = g - 2

    p_value = 1 - stats.chi2.cdf(hl_stat, df=df)

    return hl_stat, df, p_value, g
```

```python
# ============================================================================
# 3. ТАБЛИЦА 1. БАЗОВЫЕ ХАРАКТЕРИСТИКИ
# ============================================================================
print("\n" + "="*80)
print("ТАБЛИЦА 1. БАЗОВЫЕ ДЕМОГРАФИЧЕСКИЕ И ОПЕРАЦИОННЫЕ ХАРАКТЕРИСТИКИ")
print("="*80)

t1_res = []

# Непрерывные переменные
for c in ['возраст_годы', 'сложность_нассар']:
    p, method = get_p_cont(df, c)
    t1_res.append({
        'Параметр': c,
        'Группа 1': f"{g1[c].mean():.2f}±{g1[c].std():.2f}",
        'Группа 2': f"{g2[c].mean():.2f}±{g2[c].std():.2f}",
        'p-value': f"{p:.4f} ({method})"
    })

# Категориальные переменные
for c in ['пол', 'cvs_достигнут', 'вид_холецистита']:
    val1, val2 = g1[c].sum(), g2[c].sum()
    p, method = get_p_cat(df, c)
    t1_res.append({
        'Параметр': c,
        'Группа 1': f"{val1} ({val1/len(g1)*100:.1f}%)",
        'Группа 2': f"{val2} ({val2/len(g2)*100:.1f}%)",
        'p-value': f"{p:.4f} ({method})"
    })

df_table1 = pd.DataFrame(t1_res).set_index('Параметр')
display(df_table1)

# ============================================================================
# 4. ТАБЛИЦА 3. ОСЛОЖНЕНИЯ
# ============================================================================
print("\n" + "="*80)
print("ТАБЛИЦА 3. ЧАСТОТА ОСЛОЖНЕНИЙ В ГРУППАХ СРАВНЕНИЯ")
print("="*80)

t3_res = []
for c in ['желчные_всего', 'желчные_тип_A', 'желчные_тип_C',
          'воспалительные_осложнения', 'повторные_госпитализации']:
    val1, val2 = g1[c].sum(), g2[c].sum()
    p, method = get_p_cat(df, c)
    t3_res.append({
        'Осложнение': c,
        'Группа 1': f"{val1} ({val1/len(g1)*100:.1f}%)",
        'Группа 2': f"{val2} ({val2/len(g2)*100:.1f}%)",
        'p-value': f"{p:.4f} ({method})"
    })

df_table3 = pd.DataFrame(t3_res).set_index('Осложнение')
display(df_table3)

# ============================================================================
# 5. ТАБЛИЦА 4. СТРАТИФИКАЦИЯ ПО НАССАРУ
# ============================================================================
print("\n" + "="*80)
print("ТАБЛИЦА 4. СРАВНЕНИЕ МЕТОДОВ В ЗАВИСИМОСТИ ОТ КЛАССА СЛОЖНОСТИ")
print("="*80)

for name, cond in [('III класс (сложный)', df['сложность_нассар']==3),
                   ('IV-V классы (очень сложный/экстремальный)', df['сложность_нассар']>=4)]:
    sub = df[cond]
    t4_res = []

    for c in ['желчные_всего', 'повторные_госпитализации']:
        val1 = sub[sub['обработка_протока_метод']==0][c].sum()
        val2 = sub[sub['обработка_протока_метод']==1][c].sum()

        ct = pd.crosstab(sub['обработка_протока_метод'], sub[c])
        chi2, p, dof, expected = stats.chi2_contingency(ct, correction=False)

        if (expected < 5).any().any() or (ct == 0).any().any():
            p = stats.fisher_exact(ct)[1]
            method = "Fisher"
        else:
            method = "Pearson Chi2"

        t4_res.append({
            'Исход': c,
```

```
            'Группа 1': f"{val1}",
            'Группа 2': f"{val2}",
            'p-value': f"{p:.4f} ({method})"
        })

    print(f"\n--- {name} ---")
    df_table4 = pd.DataFrame(t4_res).set_index('Исход')
    display(df_table4)

# ================================================================================
# 6. ТАБЛИЦА 5. ДЛИТЕЛЬНОСТЬ ОПЕРАЦИИ И ГОСПИТАЛИЗАЦИИ
# ================================================================================
print("\n" + "="*80)
print("ТАБЛИЦА 5. ДЛИТЕЛЬНОСТЬ ОПЕРАТИВНОГО ВМЕШАТЕЛЬСТВА И ГОСПИТАЛИЗАЦИИ")
print("="*80)

t5_res = []
for c in ['длительность_операции_мин', 'госпитализация_дни']:
    p, method = get_p_cont(df, c)
    t5_res.append({
        'Параметр': c,
        'Группа 1': f"{g1[c].mean():.2f}±{g1[c].std():.2f}",
        'Группа 2': f"{g2[c].mean():.2f}±{g2[c].std():.2f}",
        'p-value': f"{p:.4f} ({method})"
    })

df_table5 = pd.DataFrame(t5_res).set_index('Параметр')
display(df_table5)

# ================================================================================
# 7. МУЛЬТИВАРИАНТНАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ
# ================================================================================
print("\n" + "="*80)
print("МУЛЬТИВАРИАНТНАЯ ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ")
print("="*80)

formula = 'желчные_всего ~ обработка_протока_метод + cvs_достигнут + вид_холецистита'
model = smf.logit(formula, data=df).fit(disp=0)

summary = model.summary2().tables[1]
summary['OR'] = np.exp(summary['Coef.'])
summary['95% CI Lower'] = np.exp(model.conf_int()[0])
summary['95% CI Upper'] = np.exp(model.conf_int()[1])

# AUC-ROC
auc = roc_auc_score(df['желчные_всего'], model.predict(df))

# Тест Хосмера-Лемешова (ПРАВИЛЬНЫЙ с корректным df)
hl_stat, df_hl, hl_p, g = hosmer_lemeshow_test(model, df['желчные_всего'])

print(f"\nДискриминация (AUC-ROC): {auc:.3f}")
print(f"Калибровка (Хосмер-Лемешов): χ²={hl_stat:.3f}, g={g} групп, df={df_hl}, p={hl_p:.4f}")
print(f"\nВажно: df = g - 2 = {g} - 2 = {df_hl} (фактическое количество групп после разбиения на децили)")

# Таблица OR
df_or = summary[['OR', '95% CI Lower', '95% CI Upper', 'P>|z|']].copy()
df_or.columns = ['OR', '95% CI Lower', '95% CI Upper', 'p-value']
df_or.rename(columns={'P>|z|': 'p-value'}, inplace=True)
display(df_or)

# ================================================================================
# 8. КАЛИБРОВОЧНЫЙ ГРАФИК
# ================================================================================
print("\n" + "="*80)
print("КАЛИБРОВОЧНЫЙ ГРАФИК")
print("="*80)

probs = model.predict(df)
df_calib = pd.DataFrame({'pred': probs, 'obs': df['желчные_всего']})
df_calib['decile'] = pd.qcut(df_calib['pred'], q=10, duplicates='drop')

calib_data = df_calib.groupby('decile', observed=False).agg({
    'pred': 'mean',
    'obs': 'mean'
}).reset_index()

plt.figure(figsize=(8, 6))
plt.plot(calib_data['pred'], calib_data['obs'], 's-', label='Модель', markersize=8)
plt.plot([0, 1], [0, 1], 'k:', label='Идеально калиброванная')
plt.xlabel('Среднее предсказанное значение (вероятность)')
plt.ylabel('Доля положительных исходов')
plt.title('Калибровочный график')
plt.legend()
```

```
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# ================================================================================
# 9. ROC КРИВАЯ
# ================================================================================
print("\n" + "="*80)
print("ROC КРИВАЯ")
print("="*80)

fpr, tpr, _ = roc_curve(df['желчные_всего'], probs)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.3f})')
plt.plot([0, 1], [0, 1], 'r--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# ================================================================================
# 10. ИТОГОВАЯ ИНФОРМАЦИЯ
# ================================================================================
print("\n" + "="*80)
print("✅ АНАЛИЗ ЗАВЕРШЕН")
print("="*80)
print("\nОсновные результаты:")
print(f"1. Скорректированное OR для лигирования: {summary.loc['обработка_протока_метод', 'OR']:.3f} "
      f"(95% ДИ: {summary.loc['обработка_протока_метод', '95% CI Lower']:.3f}-"
      f"{summary.loc['обработка_протока_метод', '95% CI Upper']:.3f}; "
      f"p={summary.loc['обработка_протока_метод', 'P>|z|']:.4f})")
print(f"2. AUC-ROC: {auc:.3f}")
print(f"3. Тест Хосмера-Лемешова: χ²={hl_stat:.3f}, df={df_hl}, p={hl_p:.4f}")
print(f"\nКод строго соответствует методологии, описанной в разделе 'Статистическая обработка' статьи.")
print("="*80)
```

```
================================================================================
СТАТИСТИЧЕСКИЙ АНАЛИЗ ДАННЫХ
Сравнительный анализ методов обработки пузырного протока
================================================================================
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 18
     13 print("="*80)
     15 # ============================================================================
     16 # 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
     17 # ============================================================================
---> 18 df = pd.read_excel('/content/baza6.xlsx')
     19 df.columns = [c.strip() for c in df.columns]
     20 df.replace(-1, np.nan, inplace=True)

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:495, in read_excel(io, sheet_name, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, parse_dates, date_parser, date_format, thousands, decimal, comment, skipfooter, storage_options, dtype_backend, engine_kwargs)
    493 if not isinstance(io, ExcelFile):
    494     should_close = True
--> 495     io = ExcelFile(
    496         io,
    497         storage_options=storage_options,
    498         engine=engine,
    499         engine_kwargs=engine_kwargs,
    500     )
    501 elif engine and engine != io.engine:
    502     raise ValueError(
    503         "Engine should not be specified when passing "
    504         "an ExcelFile - ExcelFile already has the engine set"
    505     )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1550, in ExcelFile.__init__(self, path_or_buffer, engine, storage_options, engine_kwargs)
   1548     ext = "xls"
   1549 else:
-> 1550     ext = inspect_excel_format(
   1551         content_or_path=path_or_buffer, storage_options=storage_options
   1552     )
   1553     if ext is None:
   1554         raise ValueError(
   1555             "Excel file format cannot be determined, you must specify "

   1556             "an engine manually."
   1557         )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1402, in inspect_excel_format(content_or_path, storage_options)
   1399 if isinstance(content_or_path, bytes):
   1400     content_or_path = BytesIO(content_or_path)
-> 1402 with get_handle(
   1403     content_or_path, "rb", storage_options=storage_options, is_text=False
   1404 ) as handle:
   1405     stream = handle.handle
   1406     stream.seek(0)

File ~/.local/lib/python3.12/site-packages/pandas/io/common.py:882, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    873         handle = open(
    874             handle,
    875             ioargs.mode,
   (...)
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
--> 882         handle = open(handle, ioargs.mode)
    883     handles.append(handle)
    885 # Convert BytesIO or file objects passed with an encoding

FileNotFoundError: [Errno 2] No such file or directory: '/content/baza6.xlsx'
```

Новый раздел

In [4]:

```python
import pandas as pd
import numpy as np
from scipy import stats
import statsmodels.formula.api as smf
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')

# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
df = pd.read_excel('/content/baza6.xlsx')
```

```
df.columns = [c.strip() for c in df.columns]
df.replace(-1, np.nan, inplace=True)

# Создание переменных
df['желчные_всего'] = (df['повреждение_желчных_протоков_тип'] > 0).astype(int)
df['желчные_тип_A'] = (df['повреждение_желчных_протоков_тип'] == 1).astype(int)
df['желчные_тип_C'] = (df['повреждение_желчных_протоков_тип'] == 3).astype(int)
df['повторные_госпитализации'] = ((df['повторные_госпитализации_ранние'] == 1) | (df['повторные_госпитализации_

def get_p_cat(data, col):
    ct = pd.crosstab(data['обработка_протока_метод'], data[col])
    if ct.shape[0] < 2 or ct.shape[1] < 2: return np.nan
    chi2, p, dof, expected = stats.chi2_contingency(ct, correction=False)
    if (expected < 5).any().any(): p = stats.fisher_exact(ct)[1]
    return p

print("=== ТАБЛИЦА 4. СТРАТИФИКАЦИЯ ПО NASSAR ===")
targets = [
    ('Желчные осложнения (всего)', 'желчные_всего'),
    ('Стриктура ОЖП (Тип C)', 'желчные_тип_C'),
    ('Желчеистечение (Тип A)', 'желчные_тип_A'),
    ('Воспалительные осложнения', 'воспалительные_осложнения')
]

for name, cond in [('Nassar III (Сложный)', df['сложность_нассар']==3),
                   ('Nassar IV-V (Экстремальный)', df['сложность_нассар']>=4)]:
    sub = df[cond]
    t4_data = []
    for label, col in targets:
        p = get_p_cat(sub, col)
        g1_val = sub[sub['обработка_протока_метод']==0][col].sum()
        g2_val = sub[sub['обработка_протока_метод']==1][col].sum()
        t4_data.append({'Параметр': label, 'Клипирование': g1_val, 'Лигирование': g2_val, 'p-value': f"{p:.4f}"

    print(f"\nПодгруппа: {name}")
    print(pd.DataFrame(t4_data).to_string(index=False))

print("\n=== РЕГРЕССИОННАЯ МОДЕЛЬ (Качество) ===")
model = smf.logit('желчные_всего ~ обработка_протока_метод + cvs_достигнут + вид_холецистита', data=df).fit(disp
auc = roc_auc_score(df['желчные_всего'], model.predict(df))

def hl_p(model, y):
    pihat = model.predict()
    df_hl = pd.DataFrame({'y': y, 'pihat': pihat})
    df_hl['decile'] = pd.qcut(df_hl['pihat'], q=10, duplicates='drop')
    groups = df_hl.groupby('decile', observed=False)
    obs = groups['y'].agg(['sum', 'count'])
    obs['obs_0'] = obs['count'] - obs['sum']
    exp_1, exp_0 = groups['pihat'].sum(), groups.apply(lambda x: (1 - x['pihat']).sum())
    hl_stat = (((obs['sum'] - exp_1)**2 / exp_1) + ((obs['obs_0'] - exp_0)**2 / exp_0)).sum()
    return 1 - stats.chi2.cdf(hl_stat, df=len(obs)-2)

print(f"AUC-ROC: {auc:.3f}")
print(f"Hosmer-Lemeshow p: {hl_p(model, df['желчные_всего']):.4f}")

summary = model.summary2().tables[1]
summary['OR'] = np.exp(summary['Coef.'])
print("\nРезультаты регрессии (OR):")
print(summary[['OR', 'P>|z|']])
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[4], line 10
      7 warnings.filterwarnings('ignore')
      9 # 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
---> 10 df = pd.read_excel('/content/baza6.xlsx')
     11 df.columns = [c.strip() for c in df.columns]
     12 df.replace(-1, np.nan, inplace=True)

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:495, in read_excel(io, sheet_name, header, names, index_col, usecols, dtype, engine, converters, true_values, false_values, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, parse_dates, date_parser, date_format, thousands, decimal, comment, skipfooter, storage_options, dtype_backend, engine_kwargs)
    493 if not isinstance(io, ExcelFile):
    494     should_close = True
--> 495     io = ExcelFile(
    496         io,
    497         storage_options=storage_options,
    498         engine=engine,
    499         engine_kwargs=engine_kwargs,
    500     )
    501 elif engine and engine != io.engine:
    502     raise ValueError(
    503         "Engine should not be specified when passing "
    504         "an ExcelFile - ExcelFile already has the engine set"
    505     )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1550, in ExcelFile.__init__(self, path_or_buffer, engine, storage_options, engine_kwargs)
   1548     ext = "xls"
   1549 else:
-> 1550     ext = inspect_excel_format(
   1551         content_or_path=path_or_buffer, storage_options=storage_options
   1552     )
   1553     if ext is None:
   1554         raise ValueError(
   1555             "Excel file format cannot be determined, you must specify "
   1556             "an engine manually."
   1557         )

File ~/.local/lib/python3.12/site-packages/pandas/io/excel/_base.py:1402, in inspect_excel_format(content_or_path, storage_options)
   1399 if isinstance(content_or_path, bytes):
   1400     content_or_path = BytesIO(content_or_path)
-> 1402 with get_handle(
   1403     content_or_path, "rb", storage_options=storage_options, is_text=False
   1404 ) as handle:
   1405     stream = handle.handle
   1406     stream.seek(0)

File ~/.local/lib/python3.12/site-packages/pandas/io/common.py:882, in get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text, errors, storage_options)
    873         handle = open(
    874             handle,
    875             ioargs.mode,
   (...)
    878             newline="",
    879         )
    880     else:
    881         # Binary mode
--> 882         handle = open(handle, ioargs.mode)
    883     handles.append(handle)
    885 # Convert BytesIO or file objects passed with an encoding

FileNotFoundError: [Errno 2] No such file or directory: '/content/baza6.xlsx'
```

Новый раздел

In [5]:

```
import sys
import pandas as pd
import numpy as np
import scipy
import statsmodels
import sklearn
import matplotlib

print("=== ВЕРСИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ===")
print(f"Python: {sys.version.split()[0]}")
print(f"pandas: {pd.__version__}")
print(f"numpy: {np.__version__}")
print(f"scipy: {scipy.__version__}")
```

```
print(f"statsmodels: {statsmodels.__version__}")
print(f"scikit-learn: {sklearn.__version__}")
print(f"matplotlib: {matplotlib.__version__}")
print("=======================================")
```

```
=== ВЕРСИИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ===
Python: 3.12.2
pandas: 2.2.2
numpy: 2.0.2
scipy: 1.17.1
statsmodels: 0.14.6
scikit-learn: 1.6.1
matplotlib: 3.10.0
=======================================
```