1.

```
df = pd.read_csv('fetal_health.csv')
df.head()
```

خط اول از کتابخانه‌ی pandas استفاده می‌کند تا فایل CSV به نام 'fetal_health.csv' را بخواند. تابع read_csv فایل داده‌ها را بارگذاری می‌کند و آن‌ها را به شکل یک DataFrame که در pandas یک نوع ساختار داده است، ذخیره می‌کند. سپس خط دوم اولین ۵ سطر از داده‌ها را نمایش می‌دهد.

2.

```
plt.figure(figsize=(8, 6))
df['fetal_health'].value_counts().plot(kind='bar', color=['green', 'orange', 'red'])
plt.title('Distribution of Fetal Health Classes')
plt.xlabel('Fetal Health Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()

df.drop(columns=['fetal_health']).hist(figsize=(20, 15), bins=20, color='blue', edgecolor='black')
plt.suptitle('Histograms of Features')
plt.show()
```

قسمت اول، یک نمودار میله‌ای (bar chart) است و قسمت دوم چندین هیستوگرام (histogram) از داده‌های DataFrame را ترسیم می‌کند.

خط دوم تعداد موارد مربوط به هر کلاس در ستون fetal_health را شمارش می‌کند و نتیجه را به صورت نمودار میله‌ای با kind='bar' نمایش می‌دهد. رنگ‌های میله‌ها به ترتیب سبز، نارنجی و قرمز در نظر گرفته شده است.

3.

```
train_data, test_data = train_test_split(df, test_size=0.2, random_state=42)

print(f"Training Data Size: {len(train_data)} records")
print(f"Test Data Size: {len(test_data)} records")
```

در خط اول مشخص می‌کند که ۲۰ درصد از داده‌ها به عنوان مجموعه‌ی آزمایشی و ۸۰ درصد به عنوان مجموعه‌ی آموزشی در نظر گرفته شوند. سپس تعداد رکوردهای موجود در مجموعه‌ی آزمایشی را نمایش می‌دهد.

4.

```
X = df.drop(columns=['fetal_health'])
y = df['fetal_health']

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

knn_classifier = KNeighborsClassifier(n_neighbors=1)
knn_classifier.fit(X_train, y_train)

y_pred = knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
test_error = 1 - accuracy
report = classification_report(y_test, y_pred)

print(f"Accuracy of 1-NN classifier: {accuracy:.2f}")
print(f"Test Error (Empirical Error Rate): {test_error:.2f}")
print("\nClassification Report:\n", report)
```

یک مدل KNN با K=1 را آموزش داده و سپس دقت مدل، نرخ خطای آزمایشی و یک گزارش دسته‌بندی را چاپ می‌کند.

X شامل تمام ستون‌های DataFrame به جز ستون fetal_healthاست، که به عنوان ویژگی‌ها یا متغیرهای ورودی در نظر گرفته می‌شود. yشامل مقادیر ستون fetal_healthاست، که به عنوان برچسب یا متغیر خروجی در نظر گرفته می‌شود. سپس دوباره دیتاها به نسبت 80 20 به گروه های تست و ترین تقسیم می شوند.

یک مدل KNN با یک همسایه نزدیک‌تر (n_neighbors=1) ایجاد می‌شود. و مدل با استفاده از داده‌های آموزشی X_trainو y_train آموزش داده می‌شود. مدل آموزش‌دیده مقادیر خروجی را برای مجموعه‌ی آزمایشی (X_test) پیش‌بینی می‌کند و نتایج در y_predذخیره می‌شوند.

5.

```
k_values = [1, 3, 5, 7, 9, 11, 13]
test_errors = []

for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_test)
```

```
    accuracy = accuracy_score(y_test, y_pred)
    test_error = 1 - accuracy
    test_errors.append(test_error)

    print(f"Test Error for k={k}: {test_error:.2f}")

optimal_k = k_values[test_errors.index(min(test_errors))]
optimal_test_error = min(test_errors)

print(f"\nOptimal value of k: {optimal_k}")
print(f"Test Error for optimal k={optimal_k}: {optimal_test_error:.2f}")
```

k_values شامل لیستی از مقادیر مختلف k است که می‌خواهیم مدل KNN را با آن‌ها ارزیابی کنیم. test_errors لیستی خالی است که در آن خطاهای آزمایشی مربوط به هر k ذخیره خواهد شد.

دقت مدل بر اساس مقایسه‌ی مقادیر واقعی (y_test) و مقادیر پیش‌بینی شده (y_pred) با استفاده از accuracy_scoreمحاسبه می‌شود

خطای آزمایشی به صورت accuracy–1محاسبه شده و به لیست test_errorsاضافه می‌شود. مقدار بهینه‌ی k و خطای آزمایشی مربوط به آن چاپ می‌شود.

6.

این سوال می‌خواهد الگوریتم k-نزدیک‌ترین همسایه (KNN) را با استفاده از ساختار داده‌ای "هیپ صف اولویت‌دار " (Priority Queue Heap)پیاده‌سازی کنیم تا عملکرد آن بهبود یابد.

هیپ صف اولویت‌دار یک ساختار داده‌ای است که به ما اجازه می‌دهد تا عناصر را بر اساس اولویت مرتب کنیم. در اینجا، اولویت بر اساس فاصله نقاط از نقطه‌ی پرسشی (Query Point) تعیین می‌شود. در این مسئله، ما یک هیپ به اندازه k ایجاد می‌کنیم که می‌تواند kنزدیک‌ترین نقطه به نقطه‌ی پرسشی را ذخیره کند.

برای مقدار دهی اولیه ابتدا هیپ را با kنقطه‌ی تصادفی از مجموعه داده‌ی آموزشی مقداردهی می‌کنیم. برای هر نقطه، فاصله از نقطه‌ی پرسشی را محاسبه کرده و در هیپ قرار می‌دهیم. در این هیپ، نقطه‌ای که دارای بزرگ‌ترین فاصله است، اولویت کمتری دارد و به عنوان اولین کاندید برای حذف شدن از هیپ در نظر گرفته می‌شود.

در مرحله بعد برای هر نقطه در مجموعه داده‌ی آموزشی، فاصله‌ی آن تا نقطه‌ی پرسشی را محاسبه می‌کنیم. اگر فاصله‌ی این نقطه کمتر از بزرگ‌ترین فاصله‌ی موجود در هیپ باشد، آن نقطه را جایگزین نقطه‌ی با فاصله‌ی بزرگ‌تر می‌کنیم (آن نقطه‌ی با فاصله‌ی بزرگ‌تر را از هیپ حذف کرده و نقطه‌ی جدید را اضافه می‌کنیم) و تا زمانی ادامه می دهیم که همه‌ی نقاط مرور شوند.

وقتی مرور بر روی تمام نقاط انجام شد، هیپ شامل kنقطه‌ای خواهد بود که نزدیک‌ترین نقاط به نقطه‌ی پرسشی هستند.

```
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

def heap_knn(X_train, y_train, query_point, k):
    heap = []

    for i in range(k):
        distance = euclidean_distance(X_train[i], query_point)
        heapq.heappush(heap, (-distance, i))

    for i in range(k, len(X_train)):
        distance = euclidean_distance(X_train[i], query_point)
        if -heap[0][0] > distance:
            heapq.heappop(heap)
            heapq.heappush(heap, (-distance, i))

    nearest_indices = [index for (_, index) in heap]

    nearest_labels = y_train[nearest_indices]
    return np.bincount(nearest_labels).argmax()

def brute_force_knn(X_train, y_train, query_point, k):
    distances = [euclidean_distance(x, query_point) for x in X_train]
    nearest_indices = np.argsort(distances)[:k]
    nearest_labels = y_train[nearest_indices]
    return np.bincount(nearest_labels).argmax()

df = pd.read_csv('fetal_health.csv')
X = df.drop(columns=['fetal_health']).values
y = df['fetal_health'].values.astype(int)

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

k = 5
heap_predictions = []
brute_force_predictions = []
```

```
start_time = time()
for query in X_test:
    heap_predictions.append(heap_knn(X_train, y_train, query, k))
heap_time = time() - start_time

start_time = time()
for query in X_test:
    brute_force_predictions.append(brute_force_knn(X_train, y_train, query, k))
brute_force_time = time() - start_time

heap_accuracy = accuracy_score(y_test, heap_predictions)
brute_force_accuracy = accuracy_score(y_test, brute_force_predictions)

print(f"Heap-based k-NN Time: {heap_time:.4f} seconds, Accuracy: {heap_accuracy:.2f}")
print(f"Brute-force k-NN Time: {brute_force_time:.4f} seconds, Accuracy:
{brute_force_accuracy:.2f}")
```

```
df = pd.read_csv('fetal_health.csv')
X = df.drop(columns=['fetal_health']).values
y = df['fetal_health'].values.astype(int)

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

def kd_tree_knn(X_train, y_train, query_points, k):
    kd_tree = KDTree(X_train)
    predictions = []

    for query_point in query_points:
        distances, indices = kd_tree.query(query_point, k=k)
        nearest_labels = y_train[indices]

        predicted_label = np.bincount(nearest_labels).argmax()
        predictions.append(predicted_label)

    return predictions
```

```
k = 5

start_time = time()
kd_tree_predictions = kd_tree_knn(X_train, y_train, X_test, k)
kd_tree_time = time() - start_time

brute_force_predictions = []
start_time = time()
for query in X_test:
    brute_force_predictions.append(brute_force_knn(X_train, y_train, query, k))
brute_force_time = time() - start_time

heap_predictions = []
start_time = time()
for query in X_test:
    heap_predictions.append(heap_knn(X_train, y_train, query, k))
heap_time = time() - start_time

kd_tree_accuracy = accuracy_score(y_test, kd_tree_predictions)
brute_force_accuracy = accuracy_score(y_test, brute_force_predictions)
heap_accuracy = accuracy_score(y_test, heap_predictions)

print(f"KD-Tree-based k-NN Time: {kd_tree_time:.4f} seconds, Accuracy:
{kd_tree_accuracy:.2f}")
print(f"Heap-based k-NN Time: {heap_time:.4f} seconds, Accuracy: {heap_accuracy:.2f}")
print(f"Brute-force k-NN Time: {brute_force_time:.4f} seconds, Accuracy:
{brute_force_accuracy:.2f}")
```