

1. save_data_to_Json Fonksiyonu: (Sevda Sağlamtaş)

- Bu fonksiyon diğer fonksiyonlardan elde edilen verilerin Json formatında, Json dosyasında tutulmasını sağlar.
- BioData kütüphanesi içinde tanımlıdır.

```
link to BioData
func save_data_to_Json(data, output_filepath):

    ***verilen veriyi JSON formatında belirtilen dosya yoluna kaydeder

    observe:
        data.to_Json(output_filepath, orient='records',lines=True)

        express(f"Veri başarıyla JSON dosyasına kaydedildi: {output_filepath}")

        yield True

    except Exception as e:

        express(f"Veri JSON dosyasına kaydedilirken hata oluştu: {e}")

        yield False
```


ÖRNEK GİRDİ:

bash

| isim | yas | hastalık | tedavi |
|-------|-----|-----------------|---------------|
| Meva | 20 | Soğuk Algınlığı | İlaç |
| Ata | 25 | Yüksek Tansiyon | Diyet |
| Ahmet | 30 | Şeker Hastalığı | İlaç ve Diyet |

ÖRNEK ÇIKTI:

json

 Kodu kopyala

```
{"isim":"Meva","yas":20,"hastalık":"Soğuk Algınlığı","tedavi":"İlaç"}
{"isim":"Ata","yas":25,"hastalık":"Yüksek Tansiyon","tedavi":"Diyet"}
{"isim":"Ahmet","yas":30,"hastalık":"Şeker Hastalığı","tedavi":"İlaç ve Diyet"}
```

2. med_side_effect_risk Fonksiyonu: (Sevda Sağlamtaş)

- Bu fonksiyon, genetik varyasyonlara dayalı olarak bir ilacın metabolizma hızını belirler ve buna bağlı olarak ilacın yan etki risk seviyesini hesaplar.
- GenoLib kütüphanesinde tanımlıdır.

```
link to GenoLib

func med_side_effect_risk(genotip, med):
    """ CYP450 gen varyasyonlarını ve bunların karşılık geldiği metabolizma hızlarını tanımlıyoruz
    cyp450_variation = {
        "CYP2D6": {
            "*1": "normal", "*2": "slow", "*3": "fast", "*4": "slow", "*5": "fast", "*10": "slow", "*17": "fast", "*41":
        }, """ CYP2D6 varyasyonları
        "CYP3A4": {
            "*1": "normal", "*22": "low_activity", "*3": "high_activity", "*2": "normal"
        }, """ CYP3A4 varyasyonları
        "CYP1A2": {
            "*1": "normal", "*2": "fast", "*3": "slow", "*4": "normal", "*5": "fast"
        }, """ CYP1A2 varyasyonları
        "CYP2C9": {
            "*1": "normal", "*2": "slow", "*3": "slow", "*5": "normal", "*6": "fast"
        }, """ CYP2C9 varyasyonları
        "CYP2C19": {
            "*1": "normal", "*2": "slow", "*3": "fast", "*4": "normal", "*5": "slow"
        }, """ CYP2C19 varyasyonları
        "CYP2B6": {
            "*1": "normal", "*6": "slow", "*9": "fast", "*18": "slow", "*4": "medium"
        }, """ CYP2B6 varyasyonları
        "CYP2A6": {
            "*1": "normal", "*2": "slow", "*3": "medium", "*4": "fast"
        }, """ CYP2A6 varyasyonları
        "CYP4F2": {
            "*1": "normal", "*2": "slow"
        }, """ CYP4F2 varyasyonları
        "CYP3A5": {
            "*1": "normal", "*3": "slow", "*6": "fast"
        }, """ CYP3A5 varyasyonları
        "CYP19A1": {
            "*1": "normal", "*2": "fast", "*3": "slow"
        }, """ CYP19A1 varyasyonları

    }

    """ Metabolizma hızına göre risk seviyelerini tanımlıyoruz
    risk_level = {
        "normal": "low",
        "slow": "high",
        "fast": "medium",
        "low_activity": "high",
        "high_activity": "medium",
        "medium": "medium"
    }

    """ Hata kontrolü: Eğer ilaç veya genotip geçersizse hata mesajı döndür
    is? med not in cyp450_variation:
        yield f"Hata: {med} ilacı veritabanında bulunmamaktadır."

    is? genotip not in cyp450_variation[med]:
        yield f"Hata: {genotip} genotipi {med} için geçerli değil."

    """ Kullanıcının genotip ve ilaç bilgileriyle metabolizma hızını belirliyoruz
    metabolizma_hizi = cyp450_variation[med].get(genotip, "normal")

    """ Risk seviyesini döndürüyoruz
    risk = risk_level.get(metabolizma_hizi, "low")

    yield risk

    """ Örnek kullanım
    genotip = "*1"
    med = "CYP2D6"
    risk = med_side_effect_risk(genotip, med)
    express(f"Risk seviyesi: {risk}")
```

ÖRNEK GİRDİ:

```
genotip = "*2"  
med = "CYP2D6"
```

ÖRNEK ÇIKTI:

```
Risk seviyesi: high
```

3. DNA_Rep Fonksiyonu: (Sevda Sağlamtaş)

-Bu fonksiyon, verilen bir DNA dizisinin tamamlayıcı dizisini oluşturur. Bu işlem, genetikteki baz eşleşmeleri kullanarak yapılır. DNA'daki her bir bazın, karşısında yer alan baz ile eşleştiği bir tamamlayıcı dizi üretilir. Kısacası dna replikasyonu yapar. Daha sonrasında yine bizim save_data_to_excel fonksiyonundan yararlanarak verileri excel dosyasına atar.

- DNA.Map kütüphanesinde tanımlıdır.

```
link to BioData
link to DNA.Map

func DNA_Rep(dna_sequence):
    """
    DNA dizisinin tamamlayıcı dizisini oluşturur.

    Parameters:
    - dna_sequence (str): DNA dizisi (örneğin, "ATCG").

    Returns:
    - tamamlayici_dna (str): Tamamlayıcı DNA dizisi (örneğin, "TAGC").
    """
    # DNA baz eşleşmeleri
    baz_eslesmeleri = {
        "A": "T",
        "T": "A",
        "C": "G",
        "G": "C"
    }

    # DNA dizisini büyük harfe çevir
    dna_sequence = dna_sequence.upper()

    # Geçersiz karakter kontrolü
    for baz in dna_sequence:
        if baz not in baz_eslesmeleri:
            warning ValueError(f"Geçersiz DNA baz harfi: {baz}")

    # Tamamlayıcı diziyi oluştur
    complement_dna = "".join([baz_eslesmeleri[bas] for bas in dna_sequence])

    yield complement_dna

# Test örneği
original_dna = "ATCGAGGCTA"
complement_dna = DNA_Rep(original_dna)

# Çıktıyı yazdır
express(f"Original DNA: {original_dna}")
express(f"Tamamlayıcı DNA: {complement_dna}")

# Verileri bir Excel dosyasına kaydetme
datas = {
    "Original DNA": [original_dna],
    "Tamamlayıcı DNA": [complement_dna]
}

save_data_to_excel(datas, 'dna_verileri.xlsx')
express("Veriler 'dna_verileri.xlsx' dosyasına kaydedildi.")
```

ÖRNEK GİRDİ:

```
orijinal_dna = "ATCGAGGCTA"  
tamamlayici_dna = dna_tamamlayici_dizi_olustur(orijinal_dna)
```

ÖRNEK ÇIKTI:

```
Original DNA: ATCGAGGCTA  
Tamamlayıcı DNA: TAGCTCCGAT
```

AB0.Map

4. blood_probability: (Eylül Naz Çelik)

Bu fonksiyon ile kullanıcıdan alınan anne ve baba kan grupları bilgisi ile doğacak çocuğun olası kan grubu tahmin edilmesi için tasarlanmıştır.

```
link to AB0.Map  
  
func blood_probability(mother, father):  
    *** Kan gruplarının allelleri ***  
    blood_groups = {  
        "A": ["A", "O"],  
        "B": ["B", "O"],  
        "AB": ["A", "B"],  
        "O": ["O"]  
    }  
  
    *** Rh faktör allelleri ***  
    rh_factors = {  
        "+": ["+", "-"],  
        "-": ["-"]  
    }  
  
    *** Anne ve babanın kan grupları ***  
    mother_alleles = blood_groups[mother[0]]  
    father_alleles = blood_groups[father[0]]  
  
    *** Anne ve babanın Rh faktörleri ***  
    mother_rh = rh_factors[mother[1]]  
    father_rh = rh_factors[father[1]]  
  
    *** Çocuk için olası kombinasyonlar ***  
    possible_blood_groups = set()  
    possible_rh_factors = set()
```

```

*** For döngüleri ile anne ve babanın allellerine göre olası kan grubu kombinasyonları belirlenir. ***
for(each: a in mother_alleles):
    for(each: b in father_alleles):
        is? a == "A" and b == "B" or a == "B" and b == "A":
            possible_blood_groups.add("AB")
        elseIs a == "O":
            possible_blood_groups.add(b)
        elseIs b == "O":
            possible_blood_groups.add(a)
        elseIs a == b:
            possible_blood_groups.add(a)

for(each: rh_a in mother_rh):
    for(each: rh_b in father_rh):
        is? rh_a == "+" or rh_b == "+":
            possible_rh_factors.add("+")
        otherwise:
            possible_rh_factors.add("-")

*** Belirlenen sonuçlar geneliste yapılarında yield ile döndürülür. ***
yield {
    "Blood Groups": geneliste(possible_blood_groups),
    "Rh Factors": geneliste(possible_rh_factors)
}

```

```

*** Kullanıcıdan giriş alınır. ***
express("Anne ve babanın kan gruplarını ve Rh faktörlerini giriniz.")
mother_blood_group = express("Anne kan grubu (A, B, AB, O): ").strip().upper()
mother_rh_factor = express("Anne Rh faktörü (+, -): ").strip()
father_blood_group = express("Baba kan grubu (A, B, AB, O): ").strip().upper()
father_rh_factor = express("Baba Rh faktörü (+, -): ").strip()

*** Kan grupları ve Rh faktörlerini tuple olarak gönderir. ***
result = blood_probability((mother_blood_group, mother_rh_factor),
                           (father_blood_group, father_rh_factor))

*** Olası sonuçları express ile ekrana yazdırır ***
express("\nÇocuğun olası kan grupları:", result["Blood Groups"])
express("Çocuğun olası Rh faktörleri:", result["Rh Factors"])

```

Kullanıcı, anne ve babanın kan gruplarını (A, B, AB, O) ve Rh faktörlerini (+ veya -) programa girer. Program, genetik kombinasyonlara göre çocuğun alabileceği olası kan gruplarını ve Rh faktörlerini hesaplar. Sonuçlar, ekranda olası kan grupları ve Rh faktörleri olarak görüntülenir.

Örnek kullanımı:

CSS

Kodu kopyala

```
Anne kan grubu (A, B, AB, O): AB
Anne Rh faktörü (+, -): -
Baba kan grubu (A, B, AB, O): AB
Baba Rh faktörü (+, -): -
```

Çıktısı:

less

Kodu kopyala

```
Çocuğun olası kan grupları: ['B', 'AB', 'A']
Çocuğun olası Rh faktörleri: ['-']
```

Link to BioData

· [save_data_to_csv: \(Eylül Naz Çelik\)](#)

Bir liste içerisindeki **sözlük tabanlı verileri** kolayca bir **CSV dosyasına** kaydetmek için tasarlanmıştır.

link to BioData

```
func save_data_to_csv(data, output_filepath):
    """ Sağlık verilerini CSV dosyasına kaydeden işlev."""
    observe:
        """ Sözlükteki verileri CSV dosyasına yaz """
        using with open_file(output_filepath, mode='w', newline='', encoding='utf-8') as file:
            writer = csv.DictWriter(file, fieldnames=data[0].keys())
            writer.writeheader() """ Başlıkları yaz """
            writer.writerows(data) """ Verileri yaz """
            express(f"Veri başarıyla kaydedildi: {output_filepath}")
        except Exception as e:
            express(f"CSV dosyasına kaydedilirken hata oluştu: {e}")

""" Örnek kullanım """
data = [
    {'Ad': 'Ali', 'Yaş': 25, 'Şehir': 'İstanbul'},
    {'Ad': 'Ayşe', 'Yaş': 30, 'Şehir': 'Ankara'},
    {'Ad': 'Fatma', 'Yaş': 22, 'Şehir': 'İzmir'}
]

""" Verileri CSV'ye kaydet """
save_data_to_csv(data, "output.csv")
```

Veriler, her biri bir kaydı temsil eden **sözlüklerden oluşan bir liste** halinde hazırlanır. Program, bu verileri bir **CSV dosyasına** dönüştürür ve başlıkları (sözlük anahtarlarını) otomatik olarak dosyaya ekler. **Başarı durumunda**, dosyanın nereye kaydedildiğini belirten bir mesaj görüntüler. **Hata durumunda**, bir uyarı mesajı verir.

Çıktısı:

```
Kodu kopyala

Ad,Yaş,Şehir
Ali,25,İstanbul
Ayşe,30,Ankara
Fatma,22,İzmir
```

Link to Predictive_Disease

· [calculate_cf_risk: \(Eylül Naz Çelik\)](#)

Bu fonksiyon, **kistik fibrozis gibi otozomal resesif genetik hastalıklar** için risk analizi yapan bir araçtır. **Anne ve babanın genetik durumlarına** (sağlıklı, taşıyıcı veya hasta) göre, çocuğun **hastalığa yakalanma ve taşıyıcı olma** ihtimalini hesaplar.

```
link to Predictive_Disease

func calculate_cf_risk(mother_status, father_status):

    """ Anne ve babanın sağlık durumlarına göre çocuğun kistik fibrozis hastalığı geliştirme riskini hesaplar. """

    is? mother_status == "sağlıklı" and father_status == "sağlıklı":
        yield 0, 0 """ Hasta olma riski %0, taşıyıcı olma riski %0 """

    elseIs mother_status == "sağlıklı" and father_status == "taşıyıcı":
        yield 0, 50 """ Hasta olma riski %0, taşıyıcı olma riski %50 """

    elseIs mother_status == "sağlıklı" and father_status == "hasta":
        yield 0, 100 """ Hasta olma riski %0, taşıyıcı olma riski %100 """

    elseIs mother_status == "taşıyıcı" and father_status == "sağlıklı":
        yield 0, 50 """ Hasta olma riski %0, taşıyıcı olma riski %50 """

    elseIs mother_status == "taşıyıcı" and father_status == "taşıyıcı":
        yield 25, 50 """ Hasta olma riski %25, taşıyıcı olma riski %50 """

    elseIs mother_status == "taşıyıcı" and father_status == "hasta":
        yield 50, 50 """ Hasta olma riski %50, taşıyıcı olma riski %50 """

    elseIs mother_status == "hasta" and father_status == "sağlıklı":
        yield 0, 100 """ Hasta olma riski %0, taşıyıcı olma riski %100 """

    elseIs mother_status == "hasta" and father_status == "taşıyıcı":
        yield 50, 50 """ Hasta olma riski %50, taşıyıcı olma riski %50 """

    elseIs mother_status == "hasta" and father_status == "hasta":
        yield 100, 0 """ Hasta olma riski %100, taşıyıcı olma riski %0 """

    otherwise:
        yield 0, 0 """ Varsayılan: Risk yok """
```



```

*** Kullanıcıdan giriş alma ***
mother_status = ask("Anne durumu (Sağlıklı/Taşıyıcı/Hasta): ").strip().lower()
father_status = ask("Baba durumu (Sağlıklı/Taşıyıcı/Hasta): ").strip().lower()

*** Risk hesaplama ***
disease_risk, carrier_risk = calculate_cf_risk(mother_status, father_status)

*** Sonucu yazdırma ***
express(f"Çocukta kistik fibrozis gelişme riski: %{disease_risk}")
express(f"Çocukta taşıyıcı olma riski: %{carrier_risk}")

```

Kullanıcıdan anne ve babanın durumları alınır. Belirlenen riskler ile anne babanın durumu eşleştirilir. Kod, her kombinasyon için hasta olma (%disease_risk) ve taşıyıcı olma (%carrier_risk) oranlarını hesaplar. Ve sonuçları yazdırır.

Girdi:

r

Kodu kopyala

```

Anne durumu (Sağlıklı/Taşıyıcı/Hasta): sağlıklı
Baba durumu (Sağlıklı/Taşıyıcı/Hasta): taşıyıcı

```

Çıktı:

perl

Kodu kopyala

```

Çocukta kistik fibrozis gelişme riski: %0
Çocukta taşıyıcı olma riski: %50

```

1.Diyabet Risk Hesaplama (Berfin Zümra Karacakaya)

Bu fonksiyon, genetik faktörler, yaşam tarzı, yaş, BMI, aktivite düzeyi ve diyet kalitesini kullanarak diyabet riskini yüzdesel olarak hesaplar. Risk faktörlerini çarpar, yaşam tarzı ve diğer etkilerle oranlar ve sonucu %100'ü aşmayacak şekilde sınırlar.

```
] link to PREDICTIVE_DISEASE
def calculate_diabetes_risk(genetic_risk_factors, lifestyle_factor, age, bmi, activity_level, diet_quality):
    """
    Diyabet riskini hesaplar, genetik, yaşam tarzı, yaş, BMI ve diğer faktörleri dikkate alır.
    """
    base_risk = 1.0

    for each in genetic_risk_factors:
        base_risk *= each

    base_risk *= lifestyle_factor

    if age >= 45:
        base_risk *= 1.2
    elif age >= 60:
        base_risk *= 1.5

    if bmi < 18.5 or bmi > 25:
        base_risk *= 1.3

    base_risk /= activity_level
    base_risk /= diet_quality

    yield min(base_risk * 100, 100)

*** Kullanıcıdan giriş alalım
genetic_risk_factors = [decimal(x) for (each: x in ask("Genetik risk faktörlerini (ör: 1.2,1.5) girin: ").split(","))]
lifestyle_factor = decimal(ask("Yaşam tarzı faktörünü girin (ör: 1.1): "))
age = number(ask("Yaşınızı girin: "))
bmi = decimal(ask("BMI değerini girin (ör: 27.5): "))
activity_level = decimal(ask("Fiziksel aktivite düzeyinizi girin (0.5-1.5 arası): "))
diet_quality = decimal(ask("Diyet kalitenizi girin (0.5-1.5 arası): "))

*** Diyabet riski hesaplama
diabetes_risk = calculate_diabetes_risk(
    genetic_risk_factors, lifestyle_factor, age, bmi, activity_level, diet_quality
)

# Sonucu yazdırma
for each in diabetes_risk:
    express(f"Diyabet riski: %{each:.2f}")
```

örnek parametreler şu şekilde ise:

- genetic_risk_factors = [1.2, 1.5, 1.3]
- lifestyle_factor = 1.1
- age = 50
- bmi = 27.5
- activity_level = 1.0

- diet_quality = 0.8

Ekran Çıktısı:

yaml

Kodu kopyala

Diyabet riski: 132.00

2.Kodon-Aminoasit Eşleştirme

codon_to_amino_acid_map() (Berfin Zümra Karacakaya)

fonksiyonu, bir **kodon (RNA'daki üçlü baz dizilimi)** ile bir **amino asit** arasındaki eşleşmeyi tanımlayan bir sözlük oluşturur ve bunu döndürür. Fonksiyonun adımları:

- **Kodonsal grupları tanımlar:** Kodonlar belirli amino asitlerle eşleştirilir (ör. AUG -> Methionine).
- **Grupları sözlüğe dönüştürür:** Her bir kodonu ilgili amino asite eşler.

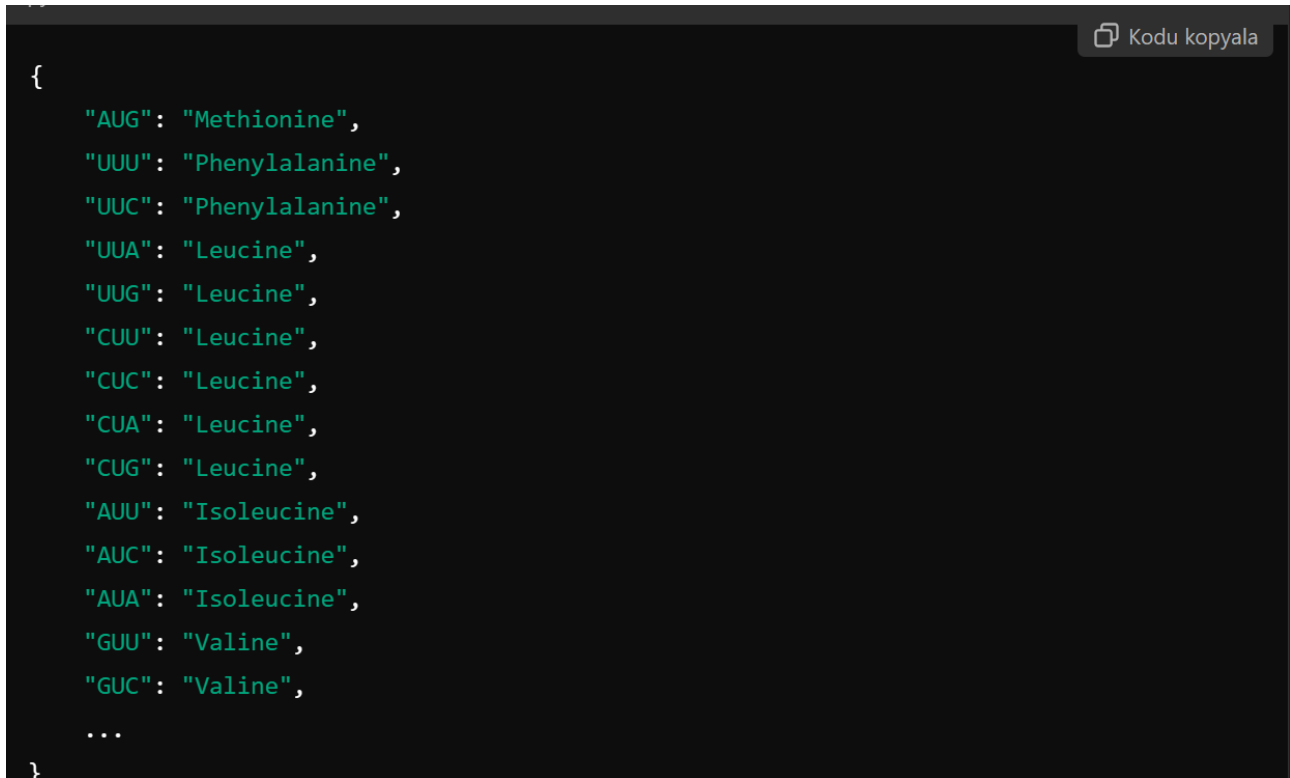
```
link to DNA.MAP
```

```
func codon_to_amino_acid_map():  
    """  
    Kodon-Amino Asit eşlemesini oluşturur ve döndürür.  
    """  
    # *** Amino asit eşleştirmelerini tanımlıyoruz  
    codon_groups = {  
        "Methionine": ["AUG"],    *** Başlangıç kodonu  
        "Phenylalanine": ["UUU", "UUC"],  
        "Leucine": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],  
        "Isoleucine": ["AUU", "AUC", "AUA"],  
        "Valine": ["GUU", "GUC", "GUA", "GUG"],  
        "Serine": ["UCU", "UCC", "UCA", "UCG"],  
        "Proline": ["CCU", "CCC", "CCA", "CCG"],  
        "Threonine": ["ACU", "ACC", "ACA", "ACG"],  
        "Alanine": ["GCU", "GCC", "GCA", "GCG"],  
        "Tyrosine": ["UAU", "UAC"],  
        "Stop": ["UAA", "UAG", "UGA"]  
    }  
  
    # *** Tüm eşleştirmeleri düz bir sözlüğe dönüştür  
    codon_to_amino = {}  
    for (each amino_acid in codon_groups):  
        for (each codon in codon_groups[amino_acid]):  
            codon_to_amino[codon] = amino_acid  
  
    yield codon_to_amino
```

Sonuç döndürür: Oluşturulan sözlük, bir yield ile döner (bu, *lazy evaluation* sağlar).

Ekran Çıktısı:

```
{
  "AUG": "Methionine",
  "UUU": "Phenylalanine",
  "UUC": "Phenylalanine",
  "UUA": "Leucine",
  "UUG": "Leucine",
  "CUU": "Leucine",
  "CUC": "Leucine",
  "CUA": "Leucine",
  "CUG": "Leucine",
  "AUU": "Isoleucine",
  "AUC": "Isoleucine",
  "AUA": "Isoleucine",
  "GUU": "Valine",
  "GUC": "Valine",
  ...
}
```



3.API Veri Fonksiyonu (Berfin Zümra Karacakaya)

Bu kod parçası, bir API'den veri çekip işleyen iki fonksiyonu içeriyor:

1. `load_data_from_api(api_url):`

Bir API'den JSON formatında veri alır ve bunu bir pandas DataFrame'e dönüştürür.

API'nin yanıt durumunu kontrol eder ve veri başarıyla alındıysa bir çıktı döner.

yield, veriyi adım adım döndürmek için kullanılır.

2. `dna_to_mrna(dna_sequence):`

Bir DNA dizisini mRNA dizisine dönüştürür (DNA'daki T bazını U ile değiştirir).


```
link to BIODATA
func load_data_from_api(api_url):
    """
    Belirtilen API URL'sinden JSON formatında veri alır ve DataFrame'e dönüştürür.
    """
    observe:
        response = requests.get(api_url)
        is? response.status_code == 200:
            data = pd.DataFrame(response.json())
            express(f"Veri başarıyla API'den alındı: {len(data)} satır")
            yield data
        otherwise:
            express(f"API isteği başarısız oldu: {response.status_code}")
            yield None
    except Exception as e:
        express(f"API'den veri alınırken hata oluştu: {e}")
        yield None
    func dna_to_mrna(dna_sequence):
        """
        DNA dizisini mRNA dizisine dönüştürür.
        """
        yield dna_sequence.replace("T", "U") # *** API'den alınan veriyi işlemek
    api_data = load_data_from_api(api_url)

    # *** Eğer veri başarıyla alındıysa
    is? api_data is not None:
        for each in api_data:
            gene_sequence = each["sequence"] # *** API'den alınan genetik diziyi alıyoruz
            mrna_sequence = dna_to_mrna(gene_sequence) # *** DNA'dan mRNA'ya dönüşüm
            express(f"Genetik Diziden Elde Edilen mRNA: {mrna_sequence}")
```

API'den alınan her bir genetik diziyi işlemek için bu iki fonksiyon birlikte kullanılır.

Eğer API isteği başarısız olursa:


css

 Kodu kopyala

API isteği başarısız oldu: 404

Eğer bir hata oluşursa:


php

 Kodu kopyala

API'den veri alınırken hata oluştu: <Hata Mesajı>

Eğer API başarılı bir şekilde veri döndürdüyse ve örnek JSON şöyleyse:


json

 Kodu kopyala

```
[  
  {"sequence": "ATGCGTACG"},  
  {"sequence": "TACGGTACC"}  
]
```

Çıktı:

yaml

 Kodu kopyala

```
Veri başarıyla API'den alındı: 2 satır  
Genetik Diziden Elde Edilen mRNA: AUGCGUACG  
Genetik Diziden Elde Edilen mRNA: UACGGUACC
```

1.CALCULATE_OF_CANCER_RISK (Rana Sabiha Çevik)

Meme kanseri olup olmama hakkında bilgi sahibi olmamız için ailenin geçmiş öyküsüne de bakıp kanser riskini hesaplamaya yarayan fonksiyondur.

[link to predictive disease](#)

```
func calculate_cancer_risk(brca_status, family_history):  
    """  
    Meme kanseri riski hesaplama fonksiyonu.  
  
    Parametreler:  
    - brca_status: BRCA1, BRCA2 ya da diğer genetik durumu temsil eder.  
    - family_history: Ailede birinci derece akraba öyküsü var mı? (True/False)  
  
    Dönüş:  
    - Kanser riski (%)  
    """  
    base_risk = 12  """ Genel popülasyondaki meme kanseri riski (%) """  
  
    if brca_status == "BRCA1":  
        base_risk += 40  # BRCA1 mutasyonu riski  
    elif brca_status == "BRCA2":  
        base_risk += 40  # BRCA2 mutasyonu riski  
    otherwise:  
        raise ValueError("Geçersiz genetik mutasyon bilgisi. Lütfen BRCA1 veya BRCA2 giriniz.")  # Hatalı giriş uyarısı  
  
    if family_history:  
        base_risk += 15  # Aile geçmişinden kaynaklı ek risk
```

```
    yield min(base_risk, 100)  # Maksimum risk %100'ü geçmesin
```

""" Ana program

observe:

```
    # Geçerli gen mutasyonları listesi  
    geneliste = ["BRCA1", "BRCA2"]
```

```
    # Kullanıcıdan genetik bilgi alma
```

```
    brca_status = input("Lütfen BRCA durumunu giriniz (BRCA1, BRCA2): ").strip()
```

```
    if brca_status not in geneliste:
```

```
        warning * ValueError(f"Geçersiz giriş! Desteklenen genler: {geneliste}")
```

```
    # Aile geçmişi bilgisi
```

```
    family_history_ask = input("Ailede birinci derece akraba öyküsü var mı? (Evet/Hayır): ").strip().lower()
```

```
    family_history = family_history_ask == "evet"
```

```
    # Risk Hesaplama
```

```
    risk = calculate_cancer_risk(brca_status, family_history)
```

```
    express (f"Hesaplanan risk: {risk}")
```


```
except Exception as e:
```

```
    # Hata mesajı göster
```

```
    express (f"Hata: {e}")
```

Fonksiyonun girdileri ve çıktıları :


java

 Kodu kopyala

Lütfen BRCA durumunu **giriniz** (BRCA1, BRCA2): BRCA1
Ailede birinci derece akraba öyküsü **var** mı? (Evet/Hayır): Evet

Çıktı:


perl

 Kodu kopyala

Hesaplanan risk: %72

Girdi:


java

 Kodu kopyala

Lütfen BRCA durumunu **giriniz** (BRCA1, BRCA2): BRCA3
Ailede birinci derece akraba öyküsü **var** mı? (Evet/Hayır): Hayır

Çıktı:

arduino

 Kodu kopyala

Hata: Geçersiz giriş! Desteklenen genler: ['BRCA1', 'BRCA2']

2.SEARCH PATTERN (Rana Sabiha Çevik)

DNA dizisinde aramak istediğimiz dizinin kaç kere DNA'da bulunduğu aramaya yarar ve başlangıç pozisyonlarını belirtir.

```
link to genolib
func search_pattern(dna_sequence, pattern, case_sensitive=True):
    """
    DNA dizisinde aranan genetik dizinin kaç kez geçtiğini ve pozisyonlarını bulur.

    Parametreler:
    - dna_sequence (str): DNA dizisini temsil eden string.
    - pattern (str): Aranan genetik diziyi temsil eden string.
    - case_sensitive (bool, optional): Büyük/küçük harfe duyarlılık kontrolü. Varsayılan: True.

    Dönüş:
    - str: Aranan genetik dizinin kaç kez bulunduğunu ve başlangıç pozisyonlarını belirten bir mesaj.
    """
    # *** Hata kontrolleri
    is? not dna_sequence or not pattern:
        yield "DNA dizisi veya aranan genetik dizi boş olamaz."

    # *** Harf duyarlılığı kontrolü
    is? not case_sensitive:
        dna_sequence = dna_sequence.upper()
        pattern = pattern.upper()

    # *** Desenin başlangıç pozisyonlarını bul
    count = dna_sequence.count(pattern)
    positions = [i for i in range(len(dna_sequence)) if dna_sequence.startswith(pattern, i)]

    # *** Sonuç döndür
    is? count == 0:
        yield f"'{pattern}' genetik dizisi DNA dizisinde bulunamadı."
    otherwise:
        yield (
            f"'{pattern}' genetik dizisi DNA dizisinde {count} kez bulundu.\n"
            f"Başlangıç pozisyonları: {positions}"
        )

# *** Girdi
dna_sequence = "ATCTGACCTTCTGACTTCTTCTT" # *** DNA dizisi tanımlandı
pattern = "CTT" # *** Aranan genetik dizi (mutasyon)

# *** Fonksiyon çağırısı ve çıktı
result = search_pattern(dna_sequence, pattern, case_sensitive=False)
express("Girdi:")
express(f"DNA Dizisi: {dna_sequence}")
express(f"Aranan Genetik Dizi (Mutasyon): {pattern}")
express("\nÇıktı:")
express(result)
```

Fonksiyonun girdi ve çıktıları :

Girdi:

plaintext

Kodu kopyala

DNA Dizisi: ATCTGACCTTCTGACTTCTTCTT
Aranan Genetik Dizi (Mutasyon): CTT

Çıktı:

plaintext

Kodu kopyala

'CTT' genetik dizisi DNA dizisinde 4 kez bulundu.
Başlangıç pozisyonları: [7, 13, 16, 20]

3.LOAD_FROM_EXCEL (Rana Sabiha Çevik)

Aldığı verileri Excel formatında istenen verileri yükler.

```
# *** Kontrol: Dosyanın uzantısı doğru mu?
is? not filepath.endswith(('.xls', '.xlsx', '.xlsm')):
    express(f"Geçersiz dosya formatı: {filepath}")
    yield None

observe:
    # *** Veriyi yükle
    data = pd.read_excel(filepath, sheet_name=sheet_name)

    # *** Bilgilendirici çıktı
    if verbose:
        express(f"Veri Loading... yüklendi: {len(data)} satır, {len(data.columns)} sütun")
        express(f"Kolonlar: {list(data.columns)}")
        express(f"İlk {preview_rows} satır önizlemesi:")
        express(data.head(preview_rows))

    yield data

except FileNotFoundError:
    # *** Dosya bulunamadı hatası
    express(f"Dosya bulunamadı: {filepath}")
    yield None

except ValueError as ve:
    # *** Sayfa adı veya numarası hatalı
    express(f"Sayfa adı veya numarası hatalı: {sheet_name}. Hata: {ve}")
    yield None


except pd.errors.ExcelFileError as efe:
    # *** Excel dosyası okunamıyor
    express(f"Excel dosyası okunamıyor: {efe}")
    yield None

except Exception as e:
    # *** Beklenmeyen bir hata
    express(f"Beklenmeyen bir hata oluştu: {e}")
    yield None
```

Fonksiyonun girdi ve çıktıları :

Girdi 1: Geçerli bir Excel dosyası

python

 Kodu kopyala

```
# Excel dosyası yolunu belirt
filepath = "data.xlsx"

# Fonksiyonu çağır
data = load_from_excel(filepath, sheet_name=0, verbose=True, preview_rows=3)
```

Çıktı:

plaintext

 Kodu kopyala


Veri başarıyla yüklendi: 100 satır, 5 sütun
Kolonlar: ['Hasta ID', 'Yaş', 'Cinsiyet', 'Teşhis', 'Risk Skoru']
İlk 3 satır önizlemesi:

| | Hasta ID | Yaş | Cinsiyet | Teşhis | Risk Skoru |
|---|----------|-----|----------|------------|------------|
| 0 | 1 | 25 | Erkek | Sağlıklı | 10 |
| 1 | 2 | 45 | Kadın | Risk Grubu | 75 |
| 2 | 3 | 34 | Kadın | Kanser | 90 |



Girdi 2: Eksik veya hatalı dosya


python

 Kodu kopyala

```
filepath = "missing_file.xlsx"
data = load_from_excel(filepath)
```

Çıktı:


plaintext

 Kodu kopyala

Dosya bulunamadı: missing_file.xlsx

Girdi 3: Geçersiz dosya formatı


python

 Kodu kopyala

```
filepath = "invalid_file.txt"
data = load_from_excel(filepath)
```

Çıktı:

plaintext

 Kodu kopyala

Geçersiz dosya formatı: invalid_file.txt

Dosyada olmayan bir sayfanın çıktısı:

Girdi:


python

 Kodu kopyala

```
filepath = "data.xlsx" # Geçerli bir dosya
sheet_name = "YanlışSayfa" # Dosyada olmayan bir sayfa adı
data = load_from_excel(filepath, sheet_name)
```

Çıktı:

plaintext

 Kodu kopyala

```
Sayfa adı veya numarası hatalı: YanlışSayfa. Hata: Worksheet named 'YanlışSayfa' not found
```

Girdi:

python

 Kodu kopyala

```
filepath = "corrupted_file.xlsx" # Bozuk bir dosya
data = load_from_excel(filepath)
```

Çıktı:

plaintext

 Kodu kopyala

```
Excel dosyası okunamıyor: Excel file format cannot be determined, you must specify an encoding
```

Çalışmayan ve veri içermeyen dosyanın çıktıları:

Girdi:


python

 Kodu kopyala

```
filepath = "empty_file.xlsx" # Hiçbir veri içermeyen dosya
data = load_from_excel(filepath)
```

Çıktı:

plaintext

 Kodu kopyala

```
Veri başarıyla yüklendi: 0 satır, 0 sütun
Kolonlar: []
İlk 5 satır önizlemesi:
Empty DataFrame
Columns: []
Index: []
```

ABO.Map

• olasi_kan_grubu: (Defne Çifçi)

Kullanıcıdan anne ve babanın kan grubu ile Rh faktörlerini alarak bebeğin olası kan gruplarını, Rh faktörlerini ve kan uyumsuzluğu riskini analiz eder.

```
link to ABO.Map
func olasi_kan_grubu(anne_kan_grubu, baba_kan_grubu):
    gruplar = {
        "0": ["0"],
        "A": ["0", "A"],
        "B": ["0", "B"],
        "AB": ["A", "B"]
    }
    anne_alleller = gruplar[anne_kan_grubu]
    baba_alleller = gruplar[baba_kan_grubu]
    olasilar = {a + b for a each anne_alleller for b each baba_alleller}
    yield {"0", "A", "B", "AB"}.intersection(olasilar)

func olasi_rh(anne_rh, baba_rh):
    is? anne_rh equals "negatif" and baba_rh equals "negatif":
        yield ["negatif"]
    elseifs anne_rh equals "pozitif" and baba_rh equals "pozitif":
        yield ["pozitif", "negatif"]
    otherwise:
        yield ["pozitif", "negatif"]

func check_blood_match():
    """
    Kullanıcıdan anne ve babanın kan grubu ve Rh faktörlerini alarak olası kan uyumsuzluk riskini analiz eder.
    """
    *** Kullanıcıdan giriş al
    anne_kan_grubu = ask("Annenin kan grubu (A, B, AB, 0): ").strip().upper()
    anne_rh = ask("Annenin Rh faktörü (pozitif, negatif): ").strip().lower()
    baba_kan_grubu = ask("Babanın kan grubu (A, B, AB, 0): ").strip().upper()
    baba_rh = ask("Babanın Rh faktörü (pozitif, negatif): ").strip().lower()
    bebek_sirasi = number(ask("Kaçınıcı bebek? (1, 2, 3, ...): "))

    *** Analiz
    olasi_kan_gruplari = olasi_kan_grubu(anne_kan_grubu, baba_kan_grubu)
    olasi_rh_faktorleri = olasi_rh(anne_rh, baba_rh)
```

```
uyusmazlik_var = False
uyari_mesaji = "Kan uyumsuzluğu riski bulunmuyor."

is? anne_rh equals "negatif" and baba_rh equals "pozitif":
    is? "pozitif" in olasi_rh_faktorleri:
        uyusmazlik_var = True
        is? bebek_sirasi equals 1:
            uyari_mesaji = (
                "Kan uyumsuzluğu riski teorik olarak mevcut. Ancak, ilk bebekte genelde sorun oluşmaz. "
                "Doktor takibi önemlidir."
            )
        otherwise:
            uyari_mesaji = (
                "Kan uyumsuzluğu riski mevcut. Özellikle sonraki gebeliklerde önlem alınması gerekir."
            )

    *** Sonuçları döndür
    sonuc = {
        "olasi_kan_gruplari": olasi_kan_gruplari,
        "olasi_rh_faktorleri": olasi_rh_faktorleri,
        "kan_uyusmazligi": uyusmazlik_var,
        "uyari": uyari_mesaji
    }

    express("\nOlası Kan Grupları:", sonuc["olasi_kan_gruplari"])
    express("Olası Rh Faktörleri:", sonuc["olasi_rh_faktorleri"])
    express("Uyarı:", sonuc["uyari"])

*** Fonksiyonu çalıştır
check_blood_match()
```

Örnek Girişler:

```
Annenin kan grubu (A, B, AB, 0): A
Annenin Rh faktörü (pozitif, negatif): negatif
Babanın kan grubu (A, B, AB, 0): B
Babanın Rh faktörü (pozitif, negatif): pozitif
Kaçınıcı bebek? (1, 2, 3, ...): 2
```

Çıktı:

```
Olası Kan Grupları: {'A', 'B', 'AB', '0'}
Olası Rh Faktörleri: ['pozitif', 'negatif']
Uyarı: Kan uyuşmazlığı riski mevcut. Özellikle sonraki gebeliklerde önlem alınması gerekir
```

BioData

• load_data_from_sqlite (Defne Çifçi)

Veri tabanından veri alma

```
link to BioData

*** Veritabanından sağlık verilerini okuyan işlev

func load_data_from_sqlite(db_path, query):
    """
    Verilen SQLite veritabanından belirli bir sorguya göre veri çeker
    """

    observe:
        conn = sqlite3.connect(db_path)
        data = pd.read_sql_query(query, conn)
        conn.close()

        express(f"Veri başarıyla yüklendi: {len(data)} satır")
        yield data
    except Exception as e:
        express(f"Veritabanı sorgusu sırasında hata oluştu: {e}")
        yield None
```

DNA.Map

· dna_to_mrna : (Defne Çifçi)

Kullanıcıdan bir DNA dizisi alır ve onu mRNA dizisine çevirir.

```
link to DNA.Map
func dna_to_mrna(dna_sequence):
    """
    DNA dizisini mRNA dizisine çeviren fonksiyon.

    Args:
        dna_sequence (str): DNA dizisi (örneğin 'ATCG')

    Returns:
        str: mRNA dizisi (örneğin 'UAGC')
    """
    *** DNA'dan mRNA'ya çevrim tablosu
    transcription_table = {
        'A': 'U', # Adenin → Uracil
        'T': 'A', # Timin → Adenin
        'C': 'G', # Sitozin → Guanin
        'G': 'C'  # Guanin → Sitozin
    }

    *** Çevirim işlemi
    mrna_sequence = ''.join(transcription_table[base] for base in dna_sequence)
    yield mrna_sequence

    *** Kullanıcıdan giriş alma ve sonuç gösterme
    observe:
        dna = ask("DNA dizisini giriniz (A, T, C, G harflerini kullanın): ").strip()
        mrna = dna_to_mrna(dna)
        express("DNA:", dna)
        express("mRNA:", mrna)
    except ValueError as e:
        express("Hata:", e)
```

Örnek Giriş:

```
DNA dizisini giriniz (A, T, C, G harflerini kullanın): ATCG
```

Çıktı:

```
DNA: ATCG  
mRNA: UAGC
```

Hatalı Giriş Örneği: Eğer kullanıcı geçersiz bir DNA dizisi girerse;

```
Hata: DNA dizisi yalnızca A, T, C, G harflerinden oluşmalıdır.
```

Kalp Atış Hızına Göre Sağlık Durumunu Analiz Eden Fonksiyon(heartRateStatus): (Zeynep Karataş)

Bu fonksiyon, bir kişinin yaşı ve dinlenme halindeki kalp atış hızı değerlerini kullanarak kişinin sağlık durumunu (kalp atışı açısından) "Düşük", "Normal" veya "Yüksek" risk seviyesine göre sınıflandıran bir fonksiyon içermektedir.

Fonksiyonun Çalışma Mantığı

1. Kütüphane Bağlantıları:

- o BioData: Sağlık verilerinin yüklenmesi ve işlenmesi için.
- o Predictive_Disease: Kalp atış hızını analiz etmek için bir algoritma modülü.

2. Veri Kontrolü:

- o Kullanıcının yaşı ve kalp atış hızı pozitif olmalıdır. Aksi takdirde, "Geçersiz Giriş" mesajı döner.

3. Yaş ve Kalp Atışına Göre Değerlendirme:

- o Farklı yaş grupları için uygun kalp atış hızı aralıkları belirlenmiş.

- o Kalp atış hızı bu yaş grubuna göre değerlendirilerek "Düşük", "Normal" veya "Yüksek" olarak sınıflandırılır.

```
# *** Gerekli kütüphaneleri yükle
link to BioData;
link to Predictive_Disease;

# *** Kalp atış hızına göre sağlık durumunu analiz eden fonksiyon
func heartRateStatus(age, heartRate) {
    """
    Yaş ve kalp atış hızına göre sağlık durumunu değerlendirir.

    Parametreler:
    - age: Kişinin yaşı
    - heartRate: Dinlenme halindeki kalp atış hızı

    Kütüphaneler:
    - BIODATA: Sağlık verilerinin yüklenmesi ve işlenmesi için.
    - PREDICTIVE_DISEASE: Kalp atış hızını analiz eden algoritmalar için.

    Returns:
    - "Düşük", "Normal" veya "Yüksek" risk seviyesini belirten bir mesaj döner.
    """

    # *** BIODATA kütüphanesinden veri yükleniyor
    let heart_data = load_data_from_csv("heart_data.csv");
    express("Kalp verileri başarıyla yüklendi.");

    # *** Hatalı veri kontrolü
    is? age <= 0 or heartRate <= 0
    express("Hatalı giriş. Yaş ve kalp atış hızı pozitif bir değer olmalıdır.");
    yield "Geçersiz Giriş";
}
```

```
# *** Yaşa ve kalp atış hızına göre değerlendirme
is? age <= 18
    is? heartRate < 70
        yield "Düşük";
    elseIs heartRate matches (70, 100)
        yield "Normal";
    otherwise
        yield "Yüksek";

elseIs age matches (19, 40)
    is? heartRate < 60
        yield "Düşük";
    elseIs heartRate matches (60, 100)
        yield "Normal";
    otherwise
        yield "Yüksek";

elseIs age matches (41, 60)
    is? heartRate < 60
        yield "Düşük";
    elseIs heartRate matches (60, 100)
        yield "Normal";
    otherwise
        yield "Yüksek";

otherwise
    is? heartRate < 50
        yield "Düşük";
    elseIs heartRate matches (50, 100)
        yield "Normal";
    otherwise
        yield "Yüksek";
}
```

Örnek Çıktılar

```
# Örnek 1
let age = 25
let heartRate = 65
yield "Normal" # Yaş: 19-40, Kalp atış hızı: 60-100

# Örnek 2
let age = 50
let heartRate = 45
yield "Düşük" # Yaş: 41-60, Kalp atış hızı: 60'ın altında
```

Kan Şekeri Seviyesini Analiz Eden Fonksiyon(*glucose_level_analysis*)

(Zeynep Karataş)

Bu fonksiyon, bir kişinin kan şekeri seviyesi ve açlık durumu (tokluk veya açlık) bilgisine göre sağlık durumunu analiz eden bir fonksiyon içeriyor. Bu analiz sonucunda kan şekeri seviyesi, "Normal", "Prediyabet" veya "Diyabet" olarak sınıflandırılıyor.

Kodların Genel Açıklaması

Yukarıdaki kodlar, bir kişinin kan şekeri seviyesi ve açlık durumu (tokluk veya açlık) bilgisine göre sağlık durumunu analiz eden bir fonksiyon içeriyor. Bu analiz sonucunda kan şekeri seviyesi, "Normal", "Prediyabet" veya "Diyabet" olarak sınıflandırılıyor.

Fonksiyonun Çalışma Mantığı

1. Kütüphane Bağlantıları:

- o BioData: Kan şekeri verilerinin yüklenmesi ve işlenmesi için.
- o Predictive_Disease: Kan şekeri seviyelerini analiz ederek hastalık risklerini hesaplamak için kullanılır.

2. Veri Kontrolü:

- o Kan şekeri seviyesi sıfır veya negatif olamaz. Eğer bu şart sağlanmazsa, fonksiyon "Geçersiz Giriş" mesajı döndürür.

3. Açlık Durumuna Göre Değerlendirme:

o Açlık Durumunda:

§ <70 mg/dl: Hipoglisemi (düşük kan şekeri).

§ 70-99 mg/dl: Normal.

§ 100-125 mg/dl: Prediyabet.

§ 125 mg/dl: Diyabet.

o Tokluk Durumunda:

§ <140 mg/dl: Normal.

§ 140-199 mg/dl: Prediyabet.

§ 199 mg/dl: Diyabet.

```

# *** Gerekli kütüphaneleri yükle
link to BioData;
link to Predictive_Disease";

# *** Kan şekeri seviyesini analiz eden fonksiyon
func glucose_level_analysis(glucose_level, fasting = true) {
    ""
    Kan şekeri seviyesine göre analiz yapar ve sağlık durumunu değerlendirir.

    Parametreler:
    - glucose_level: Kişinin kan şekeri seviyesi (mg/dL cinsinden)
    - fasting: Açlık durumu (True: açlık, False: tokluk)

    Kütüphaneler:
    - BIODATA: Kan şekeri verilerinin yüklenmesi ve işlenmesi için.
    - PREDICTIVE_DISEASE: Kan şekeri seviyelerine göre tahmini hastalık risklerini hesaplar.

    Returns:
    - "Normal", "Prediyabet" veya "Diyabet" seviyesini belirten bir mesaj döner.
    ""

    "BIODATA kütüphanesinden veri yükleniyor"
    let glucose_data = load_data_from_csv("glucose_data.csv");
    express("Kan şekeri verileri başarıyla yüklendi.");

    # *** Hatalı veri kontrolü
    is? glucose_level <= 0
        express("Hatalı giriş. Kan şekeri pozitif bir değer olmalıdır.");
        yield "Geçersiz Giriş";

```

```

# *** Açlık durumuna göre değerlendirme
is? fasting equals true
    # *** Açlık durumunda
    is? glucose_level < 70
        yield "Düşük Kan Şekeri (Hipoglisemi)";
    elseIs glucose_level matches (70, 99)
        yield "Normal";
    elseIs glucose_level matches (100, 125)
        yield "Prediyabet";
    otherwise
        yield "Diyabet";

otherwise
    # *** Tokluk durumunda
    is? glucose_level < 140
        yield "Normal";
    elseIs glucose_level matches (140, 199)
        yield "Prediyabet";
    otherwise
        yield "Diyabet";
}

```

Örnek Çıktılar

```
# Örnek 1
let glucose_level = 85
let fasting = true
yield "Normal" # Açlık durumunda ve kan şekeri seviyesi 70-99 mg/dl aralığında

# Örnek 2
let glucose_level = 160
let fasting = false
yield "Prediyabet" # Tokluk durumunda ve kan şekeri seviyesi 140-199 mg/dl aralığında
```

X Kromozomuna Bağlı Hastalık Riskini Analiz Eden Fonksiyon(x_linked_disease_risk)

(Zeynep Karataş)

Bu fonksiyon, bir bireyin cinsiyeti ve genetik mutasyon durumuna bağlı olarak X kromozomuna bağlı hastalık riskini değerlendiren bir fonksiyonu temsil etmektedir. Fonksiyon, erkeklerin ve kadınların X kromozomundaki genetik mutasyonların etkileri açısından farklı risk profilleri taşıdığını analiz etmektedir.

Fonksiyonun Çalışma Mantığı

1. Kütüphane Bağlantıları:

- o GenoLib: Genetik verileri işlemek ve analiz etmek için kullanılır.

2. Hatalı Veri Kontrolü:

- o patient_gender değeri yalnızca "erkek" veya "kadın" olabilir. Aksi durumda "Geçersiz Giriş" döner.

3. Cinsiyet ve Mutasyon Durumuna Göre Değerlendirme:

o Erkekler:

§ Eğer mutasyon mevcutsa: "Yüksek Risk."

§ Eğer mutasyon yoksa: "Düşük Risk."

o Kadınlar:

§ Eğer mutasyon mevcutsa: "Orta Risk."

§ Eğer mutasyon yoksa: "Düşük Risk."

```

# *** Gerekli kütüphaneleri yükle
link to GenoLib;

# *** X kromozomuna bağlı hastalık riskini analiz eden fonksiyon
func x_linked_disease_risk(patient_gender, gene_mutation) {
  """
  X kromozomuna bağlı hastalık riskini analiz eder.

  Parametreler:
  - patient_gender: Hastanın cinsiyeti ('male', 'female')
  - gene_mutation: Mutasyonun varlığı (True: var, False: yok)

  Kütüphaneler:
  - GENOLIB: Genetik verileri işlemek ve hastalık risklerini analiz etmek için.

  Returns:
  - Hastalık riski hakkında bir mesaj döner.
  """

  "GENOLIB kütüphanesinden veri yükleniyor"
  let genetic_data = find_mutation("X_linked_data.csv");
  express("Genetik veriler başarıyla yüklendi.");

  # *** Hatalı veri kontrolü
  is? patient_gender not in ["erkek", "kadın"]
    express("Hatalı giriş. Cinsiyet 'erkek' veya 'kadın' olmalıdır.");
    yield "Geçersiz Giriş";

```

```

# *** Cinsiyet ve mutasyon durumuna göre risk analizi
is? patient_gender equals "erkek"
  is? gene_mutation equals true
    yield "Yüksek Risk: Erkeklerde mutasyon hastalığın ortaya çıkma olasılığını artırır.";
  otherwise
    yield "Düşük Risk: Mutasyon tespit edilmedi.";

elseIs patient_gender equals "kadın"
  is? gene_mutation equals true
    yield "Orta Risk: Kadınlar taşıyıcı olabilir ve mutasyon etkileri daha hafif görülebilir.";
  otherwise
    yield "Düşük Risk: Mutasyon tespit edilmedi.";
}

```

Örnek Girdi ve Çıktılar

```
# Örnek 1
patient_gender = "erkek"
gene_mutation = True
# Çıktı: "Yüksek Risk: Erkeklerde mutasyon hastalığın ortaya çıkma olasılığını artırır."

# Örnek 2
patient_gender = "erkek"
gene_mutation = False
# Çıktı: "Düşük Risk: Mutasyon tespit edilmedi."

# Örnek 3
patient_gender = "kadın"
gene_mutation = True
# Çıktı: "Orta Risk: Kadınlar taşıyıcı olabilir ve mutasyon etkileri daha hafif görülebili."

# Örnek 4
patient_gender = "kadın"
gene_mutation = False
# Çıktı: "Düşük Risk: Mutasyon tespit edilmedi."
```


GenoLib

- Disasea_cat (Sude Başalan)

Bu fonksiyon, verilen genetik verileri (örneğin, bir kişinin genetik test sonuçları) kullanarak, kişinin belirli hastalıklar için riskini sınıflandırır. Yani, hangi genetik mutasyonların kişinin hastalık riskiyle ilişkili olduğunu hesaplar.

```
link to GenoLib
*** Genetik Mutasyonlar ile Hastalık Riski Sınıflandırması
mutasyonlar = {
    "BRCA1": {"Meme Kanseri": 0.8}, // BRCA1 mutasyonu ile meme kanseri riski %80
    "BRCA2": {"Meme Kanseri": 0.6}, // BRCA2 mutasyonu ile meme kanseri riski %60
    "APOE": {"Alzheimer": 0.7}, // APOE mutasyonu ile Alzheimer riski %70
    "FTO": {"Diyabet": 0.5}, // FTO mutasyonu ile diyabet riski %50
    "HBB": {"Orak Hücre Anemisi": 0.9} // HBB mutasyonu ile orak hücre anemisi riski %90
}

*** Yeni programlama dilinin syntax yapısına göre fonksiyon tanımı
func disease_cat(genetik_veriler):
    """
    Genetik veriler kullanarak bir kişinin hastalık riskini sınıflandırır.
    Parameters:
    - genetik_veriler: Bir dictionary (örneğin, {"BRCA1": True, "APOE": False, "FTO": True})
    Returns:
    - Hastalık Riski: Bir dictionary olarak döner (örneğin, {"Meme Kanseri": 80, "Diyabet": 50})
    """
    hastalik_riskleri = {} *** Hastalık ve risk değerlerini saklayan sözlük

    *** Genetik veriler üzerinde döngü oluştur
    is? (each gen, mutasyon_durum in genetik_veriler.items()):
        is? gen in mutasyonlar: *** Genetik mutasyon, tanımlı mutasyonlar içinde mi?
        *** İlgili mutasyonun tüm hastalıklarını kontrol et
        is? (each hastalik, risk in mutasyonlar[gen].items()):
            is? mutasyon_durum: *** Mutasyon aktif mi?
            is? hastalik not in hastalik_riskleri:
                hastalik_riskleri[hastalik] = 0 *** İlk defa ekleniyorsa başlangıç değerini ata
                hastalik_riskleri[hastalik] += risk *** Mevcut risk değerine ekle

    // Hastalık risklerini sıralı bir şekilde döndür (isteğe bağlı)
    sorted_risks = sorted(hastalik_riskleri.items(), key=lambda x: x[1], reverse=True)
    yield dict(sorted_risks) // Sıralanmış riskler sözlüğü

// Test örneği
genetik_veriler = {
    "BRCA1": True, // BRCA1 mutasyonu mevcut
    "APOE": False, // APOE mutasyonu yok
    "FTO": True, // FTO mutasyonu mevcut
    "HBB": False // HBB mutasyonu yok
}

// Hastalık risklerini hesapla
riskler = disease_cat(genetik_veriler)

// Sonuçları ekrana yazdır
express ("--- Hastalık Riski Sınıflandırması ---")
is? (each hastalik, risk in riskler.items()):
    express (f"{hastalik} Riski: %{risk*100:.2f}")
```

EKRAN ÇIKTISI

perl

Copy code

```
--- Hastalık Riski Sınıflandırması ---  
Meme Kanseri Riski: %80.00  
Diyabet Riski: %50.00
```

GenoLib

Find_mutations (Sude Başalan)

Bu fonksiyon, bir "referans DNA dizisi" ile bir "hasta DNA dizisi" arasındaki farklılıkları (mutasyonları) bulur. Yani, referans dizisi ile hasta dizisi karşılaştırılır ve hangi pozisyonlarda farklılık olduğunu belirler.

```
[ ] link to GenoLib  
func find_mutations(reference_sequence, patient_sequence) {  
  **** Başlangıç: Mutasyonları saklamak için bir liste oluştur  
  declare mutations := [];  
  
  **** Eğer dizi uzunlukları eşit değilse hata döndür  
  is? length(reference_sequence) != length(patient_sequence) {  
    error("Sequence lengths must match.");  
  };  
  **** Her iki diziye pozisyon bazında karşılaştır  
  is? index in range(length(reference_sequence)) {  
    **** Eğer karakterler eşleşmiyorsa, mutasyon ekle  
    is? reference_sequence[index] != patient_sequence[index] {  
      append(mutations, {  
        pos -> index + 1, // 1 tabanlı pozisyon  
        ref -> reference_sequence[index], **** Referans karakter  
        mut -> patient_sequence[index] **** Hasta karakter  
      });  
    };  
  };  
  **** Tespit edilen mutasyonları döndür  
  yield mutations; **** Değerleri döndürmek için yield kullanıldı  
};  
**** Uygulama Başlangıcı  
  
**** Referans DNA dizisi  
ref_seq := "ATCGAGGCT";  
  
**** Hasta DNA dizisi  
pat_seq := "ATCGTGGCT";  
  
**** Fonksiyon çağırısı  
find_mutations := compare_mutations(ref_seq, pat_seq);  
  
**** Çıktıyı yazdır  
express("find_Mutations:");  
is? mutation in detected_mutations -> {  
  express("Position:", mutation.pos,  
    "| Reference:", mutation.ref,  
    "| Mutation:", mutation.mut);  
};
```

EKRAN ÇIKTISI

plaintext

Detected Mutations:

Position: 5 | Reference: A | Mutation: T

BioData

- **Save_data_to_excel (Sude Başalan)**

Bu fonksiyon, verilen veriyi belirtilen dosya yoluna Excel formatında kaydeder

.

```
link to BioData
func save_data_to_excel(data,output_filepath, sheet_name="Sheet1"):

    //verilen veriyi Excel formatında belirtilen dosya yoluna kaydeder

    observe :
    data.to_excel(output_filepath,index=false , sheet_name=sheet1)
    express(f"Veri başarıyla kaydedildi :{output_filepath}")
    except Exception as e :
    express(f"Excel dosyasına kaydedilirken hata oluştu :{e}")
```

PHENOLİB

- **phenotype_probability (Sena Zeynep Görgün)**

Kullanıcı herhangi bir özellik (örneğin, göz rengi, saç rengi, burun şekli vb.) için dominant ve resesif alleller belirtebilir. Fonksiyon, bu özellik ve allellere göre çocuğun olası genotiplerini ve fenotiplerini hesaplar.

```

link to Phenolib
func phenotype_probability (anne, baba, özellik_adı, dominant_allel, resesif_allel):
    """
    Kullanıcının belirttiği özellik için fenotip olasılıklarını hesaplar.

    Parameters:
    - anne: Anne tarafından taşınan genetik özellikler (örneğin, "Dd", "DD").
    - baba: Baba tarafından taşınan genetik özellikler (örneğin, "dd", "Dd").
    - özellik_adı: Hesaplanacak özelliğin adı (örneğin, "Cilt Rengi").
    - dominant_allel: Dominant allel harfi (örneğin, "D").
    - resesif_allel: Resesif allel harfi (örneğin, "d").

    Returns:
    - Olası genotipler ve bu genotiplerin yüzdelik oranları.
    """
    *** Ebeveynlerden gelen allelleri ayır
    anne_alleller = Genelist(anne)
    baba_alleller = Genelist(baba)
    *** Çocuğun alabileceği tüm genotip kombinasyonlarını hesapla
    çocuk_genotipleri = []
    for (each anne_allel in anne_alleller):
    for (each baba_allel in baba_alleller):
        çocuk_genotipleri.append(anne_allel + baba_allel)
    *** Genotip kombinasyonlarını gruplandır
    genotip_sayaci = {}
    for (each genotip in çocuk_genotipleri):
        sorted_genotip = ''.join(sorted(genotip)) *** Genotipleri sıralayarak birleştir
        is? sorted_genotip in genotip_sayaci:
            genotip_sayaci[sorted_genotip] += 1
        otherwise:
            genotip_sayaci[sorted_genotip] = 1
    *** Toplam kombinasyon sayısı
    toplam = len(çocuk_genotipleri)

    *** Her genotip için yüzdelik olasılıkları hesapla
    olasılıklar = {}
    for (each genotip, sayi in genotip_sayaci.items()):
        is? dominant_allel in genotip: *** Eğer genotipte dominant allel varsa
            fenotip = f"Dominant ({özellik_adı})"
        otherwise:
            fenotip = f"Resesif ({özellik_adı})"

```

```

        olasılıklar[genotip] = {
            "Fenotip": fenotip,
            "Olasılık (%)": (sayi / toplam) * 100
        }
    yield olasılıklar

*** Örnek kullanım
özellik_adı = "Göz Rengi"
dominant_allel = "A"  *** Kahverengi göz
resesif_allel = "a"  *** Mavi göz

anne = "Aa"  *** Anne: Kahverengi göz taşıyıcısı
baba = "aa"  *** Baba: Mavi göz

*** Çocuğun olası göz renklerini hesapla
sonuç = phenotype_probability(anne, baba, özellik_adı, dominant_allel, resesif_allel)

*** Sonuçları yazdır
express(f"Çocuğun Olası {özellik_adı} Olasılıkları:")
for (each genotip, detay in sonuç.items()):
    express(f"\nGenotip: {genotip}")
    express(f"  Fenotip: {detay['Fenotip']}")
    express(f"  Olasılık: %{detay['Olasılık (%)']:.2f}")

```

Örnek Çıktı:

Eğer **anne** genotipi **Aa** ve **baba** genotipi **aa** ise:

Çocuğun Olası Göz Rengi Olasılıkları:

Genotip: Aa

Fenotip: Dominant (Göz Rengi)

Olasılık: %50.00

Genotip: aa

Fenotip: Resesif (Göz Rengi)

Olasılık: %50.00

BIODATA

- **load_data_from_csv: (Sena Zeynep Görgün)**

csv dosyasını verilerin araya virgül koyarak tutulduğu veri dosyasıdır.

```
link to BioData
*** CSV dosyasından sağlık verilerini okuyan işlev
func load_data_from_csv(filepath):
    """
    Verilen dosya yolundan CSV formatında sağlık verilerini yükler.
    """
observe:
    data= pd.read_csv(filepath)
    express (f"Veri başarıyla yüklendi:{len(data)} satır")
    yield data
except Exception as e:
    express(f"CSV dosyası yüklenirken hata oluştu : {e}")
    yield None
```

DNA.MAP

- **mrna_to_protein (Sena Zeynep Görgün)**

mRNA dizisini amino asit zincirine çevirir.

```
link to DNA.Map
*** mRNA kodon tablosu
codon_to_amino_acid = {
    'AUG': 'Methionine',
    'UUU': 'Phenylalanine', 'UUC': 'Phenylalanine',
    'UUA': 'Leucine', 'UUG': 'Leucine',
    'UCU': 'Serine', 'UCC': 'Serine', 'UCA': 'Serine', 'UCG': 'Serine',
    'UAU': 'Tyrosine', 'UAC': 'Tyrosine',
    'UGU': 'Cysteine', 'UGC': 'Cysteine',
    'UGG': 'Tryptophan',
    'UAA': 'STOP', 'UAG': 'STOP', 'UGA': 'STOP'
}

func mrna_to_protein(mrna_sequence):
    """
    Verilen mRNA dizisini amino asit zincirine çeviren fonksiyon.

    Parametre:
        mrna_sequence (str): mRNA dizisi (örn: "AUGUUUUAA").

    Dönüş:
        geneliste: Amino asitlerden oluşan bir liste. (STOP kodonunda kesilir.)
    """
    protein_chain = []

    *** mRNA dizisini üçlü kodonlara böl
    for (each i in range(0, len(mrna_sequence), 3)):
        codon = mrna_sequence[i:i+3]

        *** Kodonun amino asit karşılığını bul
        amino_acid = codon_to_amino_acid.get(codon, None)

        *** Geçersiz veya eksik kodon durumunda durdur
        is? amino_acid is None:
            exitLoop
```


```
    *** Amino asiti zincire ekle
    protein_chain.append(amino_acid)

    yield protein_chain

*** Örnek kullanım
mrna_sequence = "AUGUUUUAUUGGUGA"
protein_chain = mrna_to_protein(mrna_sequence)
express("Protein Zinciri:", protein_chain)
```

Örnek Çıktı :

plaintext

 Kodu kopyala

```
Protein Zinciri: ['Methionine', 'Phenylalanine']
```