SUNUM notları vize zamanı:

Sunum notları:

Uygulamayı yaparken geliştirirken sürekli yeni kütüphaneler geldi ve bazı kütüphaneler python’ın bazı sürümlerine uymuyordu, bu yüzden değiştire değiştire en son python 3.11 de karar kıldım.

Uygulamayı geliştirirken bazı yerlerde ek yüklemeler istedi mesela NVIDIa sitesinden bi şeyler indirdim, sonra visual studio indirme yöneticisinden c++ kodlamalar ile ilgili birşeyler indirmem gerekti sonra vscode da ki pythonda kod çalıştı.

uygulamada şu modeller ve yöntemler mevcut:

1. **LSTM (Long Short-Term Memory)**:
   * Derin öğrenme tabanlı bir yapay sinir ağı modeli.
   * Ana tahmin modeli olarak kullanılıyor ve zaman serisi verilerindeki bağımlılıkları öğrenmede başarılı.
2. **Topluluk (Ensemble) Modelleri**:
   * **Random Forest**: Karar ağaçlarından oluşan bir topluluk modelidir.
   * **XGBoost**: Gradyan artırma yöntemiyle çalışan güçlü bir topluluk modeli.
   * **LightGBM**: Hızlı ve hafif bir gradyan artırma modeli.

**1. LSTM Modeli**

* **Fonksiyon Adı**: build\_lstm\_model
* **Yer**: BitcoinPredictorApp sınıfında, build\_lstm\_model fonksiyonu.
* **Açıklama**: LSTM modeli, Sequential modeli olarak tanımlanıyor ve LSTM katmanları eklenerek yapılandırılıyor. Modelin derin öğrenme tabanlı bir yapay sinir ağı olduğu için zaman serilerindeki uzun vadeli bağımlılıkları öğrenmede kullanılıyor.  
  metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

  Açıklama otomatik olarak oluşturuldu

**2. Topluluk (Ensemble) Modelleri**

* **Fonksiyon Adı**: train\_models ve EnsembleModel sınıfı.
* **Yer**: TrainingThread sınıfında run fonksiyonunda çağrılır ve EnsembleModel sınıfında tanımlanır.
* **Modeller**:
  + **Random Forest**: self.ensemble\_model.rf\_model
  + **XGBoost**: self.ensemble\_model.xgb\_model
  + **LightGBM**: self.ensemble\_model.lgbm\_model
* **Açıklama**: Bu modeller EnsembleModel sınıfında tanımlanmıştır ve train\_models fonksiyonunda eğitilir. Topluluk modeli, LSTM ile birlikte tahmin doğruluğunu artırmak için kullanılır.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yaptığım müdahaleler:

Dip fiyat , zirve fiyatın en fazla beşte biri olabilir. Yani bitcoin düşüş trendine girince en fazla beşte bir fiyatına düşebilir sonra da düşüş durur. Geçmiş fiyat hareketleri buna yaklaşık sonuçlar veriyor.

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yükseliş trendleri zamanla gücü azalacak, çünkü piyasa değeri artıyor, dünyada para arzı belli. Bitcoinin sayısı aşırı derece azalmadan fiyatı 100 katrilyon dolar olamaz çünkü dünyadaki para arzı ve doların enflasyonu belli bir oranda ilerliyor. Böyle bir müdahale yaptım.

Accumulation (birikim) aşamasında, belirli bir mini ralli olasılığı tanımlandı. Kodda %20 olasılıkla mini ralliler tetikleniyor ve bu ralliler 2-6 hafta sürebiliyor:

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Birikim aşaması için fiyat aralığı sınırı (%15 fiyat aralığı) ve volatilite sınırı tanımlandı. Böylece bu aşamada fiyat belirli bir aralıkta dalgalanıyor:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ralli aşamalarında ralli gücünün zamanla azaldığı bir yapı oluşturuldu. Logaritmik bir azalma ile belirli bir taban seviyeye kadar düşüyor:

metin, ekran görüntüsü, yazı tipi, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu müdahaleler bitcoin fiyatının gerçeklerden ve realiteden kopmaması için gerçekleşen küçük dokunuşlar oldu.

* Bir **model**, belirli bir algoritmanın verileri analiz edip öğrenmesiyle oluşturulan matematiksel bir yapı veya fonksiyondur. Model, verilerdeki örüntüleri öğrenir ve bu öğrendiği örüntüler sayesinde yeni verilere tahmin veya sınıflandırma yapabilir.

**2. Topluluk (Ensemble) Modelleri ve Çalışma Mantığı**

* Bu projede **Random Forest**, **XGBoost**, ve **LightGBM** modelleri **Ensemble (Topluluk) Modelleri** olarak kullanılıyor. Ensemble, birden fazla modelin çıktısını birleştirerek daha doğru tahminler elde etmeyi amaçlar.
* Bu modeller, verilerdeki karmaşık ilişkileri öğrenerek fiyat tahmininde bulunmak için eğitiliyor.

**3. Modellerin Çalışma Prensibi**

* **Random Forest**, **XGBoost**, ve **LightGBM** gibi algoritmalar **denetimli öğrenme** yöntemleri kullanır. Bu yöntem, modelin geçmiş verilerden öğrenmesi ve gelecekteki sonuçları tahmin etmesi anlamına gelir.
* **Bu modeller nasıl çalışır?**:
  1. **Veri Hazırlığı**: İlk olarak, train\_models gibi bir fonksiyon içinde modelin öğreneceği veriler hazırlanır. Bu verilere, tahmin yapılacak özellikler (örneğin, Bitcoin fiyatı, işlem hacmi, volatilite gibi) ve hedef değerler (örneğin, gelecekteki Bitcoin fiyatı) dahildir.
  2. **Model Eğitimi**: Modeller bu verileri kullanarak eğitilir. Eğitim sırasında model, verilerdeki örüntüleri (örneğin, fiyat ve trend arasındaki ilişki) öğrenir. Her model, verilerdeki farklı yönleri öğrenerek fiyat tahminini daha doğru hale getirir.
  3. **Tahmin Yapma**: Eğitim tamamlandığında, bu modeller yeni veri üzerinde tahmin yapabilir. Örneğin, self.ensemble\_model.rf\_model.predict(yeni\_veri) çağrısı, Random Forest modelini kullanarak yeni veriye göre bir fiyat tahmini yapar.
* self.rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42, n\_jobs=-1)
* Bu satırda, RandomForestRegressor adlı bir model self.rf\_model olarak tanımlanır. Bu model, RandomForestRegressor sınıfını kullanarak bir **Random Forest** algoritması oluşturur.
* **self.ensemble\_model.rf\_model.fit(X, y)** komutu, modeli X ve y verilerini kullanarak eğitir. Burada:
  + X: Özellikler (bağımsız değişkenler) yani modelin tahmin yaparken bakacağı veriler.
  + y: Hedef değerler (bağımlı değişkenler), yani modelin tahmin etmeye çalıştığı asıl değerlerdir.

**5. Yapay Zeka ve Model Tanımlarıyla Ne Elde Ediliyor?**

* **Öğrenme Süreci**: Model, veri örüntülerini ve ilişkileri öğrenir. Örneğin, Random Forest modelinin fit fonksiyonunu çağırdığınızda, model eğitim verilerindeki ilişkilere göre kendini optimize eder.
* **Tahmin Süreci**: Eğitimden sonra, model yeni veriler üzerinde predict fonksiyonu ile tahmin yapabilir. Örneğin, self.ensemble\_model.rf\_model.predict(yeni\_veri) komutuyla Random Forest modeli yeni bir tahmin üretir.

**6. Neden Farklı Modeller Kullanılıyor?**

* **Model Çeşitliliği**: Her modelin öğrenme şekli farklıdır ve verilerin farklı yönlerini yakalayabilir. Örneğin, Random Forest karar ağaçları kullanarak veri içindeki belirgin kalıpları öğrenirken, XGBoost ve LightGBM gradyan artırma yöntemiyle daha ince detayları öğrenir.
* **Ensemble (Topluluk) Yaklaşımı**: Farklı modellerin tahminleri birleştirildiğinde (örneğin, bu projede LSTM ile ensemble modellerin birleşimi), tek bir modelin başarısız olabileceği durumlarda daha sağlam tahminler elde edilebilir.

**LSTM'nin Önemi**

* kodda en öne çıkan yapay zeka modeli **LSTM**. Bunun nedeni, LSTM'nin zaman serisi verilerindeki örüntüleri ve bağımlılıkları öğrenmede çok başarılı olmasıdır. Bitcoin fiyatları gibi zamanla değişen verilere sahip bir durumda, LSTM'nin geçmiş değerleri öğrenerek geleceği tahmin etme yeteneği çok önemlidir.
* **Neden LSTM öncelikli?** LSTM modelini öncelikli olarak kullanmış olmamın sebebi, zaman serisi analizlerinde uzun vadeli bağımlılıkları modellemede güçlü olmasıdır. Diğer modeller, verideki kısa vadeli veya karmaşık ilişkileri öğrenmede etkili olabilir; ancak, asıl tahmin sürecinde uzun vadeli eğilimleri modellemek için LSTM tercih edildi.

**TrainingThread Nedir?**

* **TrainingThread** ifadesi, Türkçe’de "Eğitim İş Parçacığı" anlamına gelir.
* Bu sınıf, modelin arka planda eğitilmesini sağlamak için kullanılır. Kodun çalışması sırasında, kullanıcı arayüzünün donmasını engellemek için eğitim işlemleri farklı bir iş parçacığında (thread) yapılır. Yani, **TrainingThread** sınıfı, model eğitimi sırasında kullanıcı arayüzünün akıcılığını korur ve eğitim işlemini sorunsuzca gerçekleştirir.

**Koddaki Yapay Zeka Modelleri ve İşlevleri**

* **LSTM (Long Short-Term Memory)**:
  + LSTM, zaman serilerindeki uzun vadeli bağımlılıkları öğrenir. Bitcoin fiyatlarının geçmiş hareketlerinden gelecekteki değerleri tahmin etmek için kullanılır.
  + **Uygulamadaki yeri**: build\_lstm\_model fonksiyonunda tanımlanır. Eğitimi ise train\_models fonksiyonunda yapılır.
* **Random Forest**:
  + Random Forest, veriyi birçok karar ağacı üzerinden analiz ederek kısa vadeli bağımlılıkları öğrenir. Bu model, verinin belirli örüntülerini yakalayarak daha istikrarlı tahminler yapabilir.
  + **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

**Random Forest Nedir?**

* **Random Forest**, temel olarak birçok **Karar Ağacı** modelinin bir araya gelerek oluşturduğu bir **topluluk (ensemble) modeli**dir.
* Her bir karar ağacı, veriyi farklı bir açıdan analiz eder ve bağımsız bir tahmin yapar. Random Forest, bu ağaçların tahminlerini birleştirerek daha güçlü ve güvenilir sonuçlar elde eder.
* **“Random” (rastgele)** ismi, her ağacın eğitiminde kullanılan verilerin ve özelliklerin rastgele seçilmesinden gelir.

**Random Forest Nasıl Çalışır?**

Random Forest’ın nasıl çalıştığını basitçe adım adım açıklayalım:

1. **Veri Kümesinin Rastgele Alt Kümelere Bölünmesi**:
   * Random Forest, eğitim verisini birçok küçük alt kümeye böler. Bu alt kümeler, orijinal verinin rastgele seçilmiş örneklerinden oluşur. Bu yöntem **Bootstraping** olarak adlandırılır.
   * Her ağaç farklı bir alt küme üzerinde eğitildiği için, ağaçlar birbirinden bağımsız hale gelir ve fazla öğrenme (overfitting) riski azalır.
2. **Her Ağaç İçin Rastgele Özellik Seçimi**:
   * Her ağacın karar verirken kullanacağı özellikler de rastgele seçilir. Örneğin, bir ağaç fiyat ve işlem hacmi gibi özelliklere bakarken, diğer bir ağaç volatilite ve RSI gibi farklı özelliklere bakabilir.
   * Bu özellik seçim süreci, her ağacın farklı açılardan öğrenmesini sağlar ve modelin çeşitliliğini artırır.
3. **Karar Ağacı Yapısı**:
   * Her bir karar ağacı, veriyi sınıflandırmak veya tahmin etmek için bir dizi "karar" oluşturur. Bu kararlar, **“if-else”** mantığıyla ilerler.
   * Örneğin, bir karar ağacı Bitcoin fiyatının son 7 günlük ortalamasına göre bir tahmin yapabilir: Eğer fiyat ortalaması belirli bir değerden büyükse, fiyatın yükseleceğine dair bir tahminde bulunabilir.
4. **Tahminlerin Birleştirilmesi**:
   * Her bir ağaç bağımsız olarak bir tahmin yapar. Tüm ağaçların yaptığı tahminler toplandıktan sonra bir ortalama alınır (regresyon problemlerinde) veya en çok tekrarlanan tahmin seçilir (sınıflandırma problemlerinde).
   * Bu toplama süreci, modelin daha sağlam olmasını sağlar. Tek bir ağaç hatalı bir tahmin yapsa bile, diğer ağaçların tahminleriyle bu hata dengelenir.

**Neden Random Forest Kullanılır?**

* **Güçlü ve Güvenilir**: Farklı karar ağaçlarının tahminleri bir araya getirildiği için, model daha güvenilir ve sağlam hale gelir. Bu nedenle, Random Forest daha düşük hata oranlarına sahiptir.
* **Aşırı Öğrenme (Overfitting) Riskini Azaltır**: Her ağaç rastgele bir veri alt kümesi üzerinde eğitildiğinden, model veriye fazla uymaz. Bu da modelin yeni veriler üzerinde daha iyi performans göstermesini sağlar.
* Uygulamada, Random Forest modeli EnsembleModel sınıfı içinde self.rf\_model olarak tanımlanmış. train\_models fonksiyonunda, bu model eğitilir:
* self.ensemble\_model.rf\_model.fit(X\_2d, y\_2d)
* **Ne İşe Yarıyor?** Random Forest, uygulamada verinin kısa vadeli bağımlılıklarını öğrenir. Örneğin, son günlerdeki fiyat hareketlerine ve işlem hacmine bakarak gelecekteki fiyat hakkında kısa vadeli tahminlerde bulunabilir.

Özetle:

Random Forest, birçok küçük karar ağacından oluşan bir yapay zeka modelidir. Her bir ağaç kendi başına öğrenir ve tahmin yapar; daha sonra bu tahminler birleştirilerek güçlü bir sonuç elde edilir.

Uygulamanızda, kısa vadeli eğilimleri ve örüntüleri yakalamak için kullanılır ve diğer modellerle birlikte tahmin doğruluğunu artırır.

**Random Forest** ve **LSTM** (Long Short-Term Memory) modelleri, farklı yapay zeka algoritmalarıdır ve farklı görevlerde öne çıkarlar. Bu iki modelin farklarını anlamak için temel özelliklerini ve kullanım alanlarını karşılaştıralım.

**1. Model Türü ve Kullanım Amacı**

* **Random Forest**:
  + **Model Türü**: Karar ağaçlarına dayalı bir **ensemble (topluluk)** yöntemidir.
  + **Kullanım Amacı**: Kısa vadeli bağımlılıkları ve doğrusal olmayan ilişkileri öğrenmekte iyidir. Daha çok **sınıflandırma** ve **regresyon** problemlerinde kullanılır.
  + **Özellikleri**: Verideki örüntüleri ve kısa vadeli ilişkileri yakalayabilir. Zamanla sırayla olan veri bağımlılıklarını dikkate almaz.
* **LSTM**:
  + **Model Türü**: Bir tür **Recurrent Neural Network (RNN)**, yani geri beslemeli sinir ağıdır.
  + **Kullanım Amacı**: **Zaman serisi verileri** gibi ardışık verilerdeki uzun vadeli bağımlılıkları öğrenmekte iyidir. Fiyat tahmini, metin analizi gibi zamanla sırayla gelen verilerde kullanılır.
  + **Özellikleri**: Zaman içinde değişen örüntüleri yakalayabilir. Zaman serisi verilerindeki uzun vadeli bağımlılıkları öğrenebilir, geçmiş veriye dayalı tahminler yapabilir.

**2. Zamanla İlişkili Verilerdeki Performans**

* **Random Forest**:
  + Zamanla ilişkili verilerde doğrudan etkili değildir, çünkü geçmiş verilerle geleceği ilişkilendirmek için özel bir mekanizması yoktur.
  + Her bir veri noktasını bağımsız olarak işler, dolayısıyla veri sıralaması veya zaman bilgisi önemli değildir.
* **LSTM**:
  + Zamanla ilişkili verilerde çok etkilidir, çünkü veri sıralamasını dikkate alarak geçmiş verilere göre geleceği tahmin eder.
  + "Hafıza" hücreleri sayesinde uzun vadeli bağımlılıkları öğrenir ve geçmişteki olayların gelecekteki tahminlerde nasıl bir etkisi olacağını anlamaya çalışır.

**3. Çalışma Prensibi**

* **Random Forest**:
  + Çok sayıda karar ağacı oluşturarak çalışır ve her ağaç, verideki bağımsız örnekleri öğrenir. Her ağacın yaptığı tahminler birleştirilerek nihai sonuç elde edilir.
  + Karar ağaçlarının bağımsızlığı nedeniyle, verinin belirli kısımlarındaki örüntüleri öğrenir, ancak ardışık verileri (örneğin zaman serileri) dikkate almaz.
* **LSTM**:
  + Geri beslemeli sinir ağı olarak çalışır ve verinin zaman sırasına göre işlenmesini sağlar.
  + "Unutma" ve "hatırlama" kapıları sayesinde geçmiş verilerin gelecekte nasıl bir etkisi olacağını anlamaya çalışır. Örneğin, Bitcoin’in fiyatındaki geçmiş dalgalanmalar, LSTM tarafından öğrenilerek gelecekteki tahminlerde kullanılır.

**4. Uygulama Alanları**

* **Random Forest**:
  + Genelde zaman sırasına göre bağımlılık gerektirmeyen veri analizlerinde (örneğin sınıflandırma ve regresyon) kullanılır. Finans, tıp, biyoloji gibi alanlarda, özellikler arasındaki ilişkileri analiz etmek için uygundur.
* **LSTM**:
  + Zaman serisi verilerinde, dil işleme (metin analiz), finansal tahminler, ses tanıma, ve video analizi gibi sıralı verilerle çalışılan durumlarda tercih edilir.

**5. Uygulamadaki Örnek Kullanımlar**

* **Random Forest**: Eğer Bitcoin fiyatının belirli teknik göstergelerle ilişkisini analiz etmek ve bu göstergelerden kısa vadeli fiyat tahmini yapmak istiyorsanız, Random Forest iyi bir seçimdir.
* **LSTM**: Bitcoin fiyatının geçmiş verilerine dayalı olarak gelecekteki fiyatı tahmin etmek istiyorsanız, LSTM daha iyi bir seçenektir. Çünkü fiyat dalgalanmalarındaki zamanla ilişkili örüntüleri öğrenebilir.

**Özet:**

* **Random Forest**: Zaman sırasını dikkate almadan bağımsız örüntüleri öğrenir, kısa vadeli tahminlerde ve sınıflandırma görevlerinde iyidir.
* **LSTM**: Zaman sırasını ve geçmiş bağımlılıkları öğrenir, ardışık verilerdeki örüntüleri yakalayarak uzun vadeli tahminlerde başarılıdır.

**XGBoost**:

* XGBoost, veri içindeki karmaşık ve doğrusal olmayan örüntüleri öğrenmede oldukça etkilidir. Gradyan artırma yöntemiyle çalışarak hata oranını azaltır.
* **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

**XGBoost Nedir?**

* **XGBoost** (Extreme Gradient Boosting), gradyan artırma yöntemiyle çalışan güçlü ve hızlı bir makine öğrenmesi algoritmasıdır.
* Özellikle karmaşık verilerdeki doğrusal olmayan ilişkileri yakalamakta çok başarılıdır. XGBoost, birçok karar ağacını art arda oluşturarak her adımda modeli daha iyi hale getirir.

**Gradyan Artırma Yöntemi Nedir?**

Gradyan artırma yöntemi, bir modeli art arda geliştirerek hataları en aza indirme tekniğidir. Bu yöntem, birden çok zayıf modelin (genellikle karar ağaçları) birleşiminden güçlü bir model oluşturur.

**Gradyan artırmanın nasıl çalıştığını adım adım anlatalım:**

1. **Başlangıç Modeli Oluşturulur**:
   * Gradyan artırma yöntemi, başlangıçta oldukça basit bir tahmin modeliyle başlar. Bu genellikle her veri noktası için ortalama bir değer tahmin eden bir modeldir.
   * İlk model oluşturulduktan sonra, her veri noktası için tahminler yapılır ve hatalar (gerçek değer ile tahmin arasındaki fark) hesaplanır.
2. **Hataların Öğrenilmesi**:
   * İlk modelin yaptığı hatalar analiz edilir. Gradyan artırma yöntemi, bu hataları en aza indirmek için çalışır.
   * Hataları anlamak için **gradyan** (eğim) hesaplanır. Bu eğim, modelin her veri noktası için ne kadar yanlış tahmin yaptığını gösterir. Yani, her bir veri noktası için "Bu tahmin ne kadar yanlış?" sorusunun cevabını verir.
3. **Yeni Ağaç Eklenmesi**:
   * Gradyan artırma yöntemi, hataları düzeltecek şekilde yeni bir karar ağacı ekler. Bu yeni ağaç, bir önceki modelin yaptığı hataları "öğrenmeye" çalışır.
   * Örneğin, ilk model bazı veri noktalarında çok yüksek, bazı veri noktalarında çok düşük tahminler yapmışsa, eklenen yeni ağaç bu hataları dengelemeye çalışır.
4. **Ağaçların Birleştirilmesi**:
   * Eklenen her yeni ağaç, bir önceki modelin hatalarını düzeltmeye yönelik olduğundan, zamanla model daha doğru tahminler yapmaya başlar.
   * Gradyan artırma süreci, her bir ağacın katkısını küçük adımlarla ekleyerek hataları kademeli olarak azaltır. Böylece model, her yeni eklenen ağaçla birlikte daha hassas hale gelir.
5. **Tüm Ağaçların Toplanması**:
   * Gradyan artırma, belirli sayıda ağaç ekleyene veya hatalar çok küçük hale gelene kadar devam eder.
   * Sonuç olarak, tüm ağaçların tahminleri birleştirilir ve daha güçlü, daha doğru bir model elde edilir.

**XGBoost ve Gradyan Artırma Yöntemi Arasındaki İlişki**

* **XGBoost**, gradyan artırma yönteminin gelişmiş ve optimize edilmiş bir versiyonudur. Standart gradyan artırma algoritmalarına göre daha hızlı ve verimli çalışır.
* XGBoost, **daha iyi genel performans**, **hızlı eğitim süresi**, ve **daha düşük hata oranı** sağlamak için optimizasyon teknikleri (örneğin, daha hızlı hesaplama için paralel işleme) kullanır.

**Neden Gradyan Artırma ve XGBoost Kullanıyoruz?**

* **Doğrusal Olmayan Örüntüleri Yakalar**: XGBoost, verideki karmaşık ilişkileri öğrenme konusunda çok iyidir. Bitcoin fiyat tahmini gibi doğrusal olmayan yapıya sahip verilerde iyi sonuç verir.
* **Düşük Hata Oranı Sağlar**: Her adımda hataları azaltarak daha doğru tahminler yapar. Bu yüzden finansal verilerde çok kullanılır.

**Uygulamadaki Yeri ve Kullanımı**

* XGBoost, EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda fit fonksiyonu ile eğitilir:
* self.ensemble\_model.xgb\_model.fit(X\_2d, y\_2d)

Bu kod, modelin verilerdeki örüntüleri öğrenmesini ve tahminler yapmasını sağlar. Yani XGBoost, veriyi analiz ederek kısa vadeli tahminler yapar ve tahmin doğruluğunu artırır.

**LightGBM**:

* LightGBM, hızlı bir gradyan artırma modelidir ve büyük veri kümelerinde bile yüksek hızda çalışabilir. Verideki detayları hızlı bir şekilde öğrenir.
* **Uygulamadaki yeri**: EnsembleModel sınıfında tanımlanır ve train\_models fonksiyonunda eğitilir.

Sunum kilit noktalar:

Uygulamayı yaparken geliştirirken sürekli yeni kütüphaneler geldi ve bazı kütüphaneler python’ın bazı sürümlerine uymuyordu, bu yüzden değiştire değiştire en son python 3.11 de karar kıldım.

Uygulamayı geliştirirken bazı yerlerde ek yüklemeler istedi mesela NVIDIa sitesinden bi şeyler indirdim, sonra visual studio indirme yöneticisinden c++ kodlamalar ile ilgili birşeyler indirmem gerekti sonra vscode da ki pythonda kod çalıştı.

Finalde neler olmasını planlıyorum? Uygulamayı yapmaya başlamadan önce plan şu şekildeydi:

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şuanki hali ise şu şekilde:

metin, ekran görüntüsü, diyagram, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Tahmin et e basınca tarih aynı olsa bile farklı tahminler yapıyor.

Finale neler eklemeyi planlıyorum:

1- Haberleri çekme ve bu haberlerin yapay zeka yorumu, haber iyiyse trendin iyileşme yönünde desteklenmesi, haber kötü ise trendin düşüş yönünde desteklenmesi.

Haber aşırı kötü ise yükseliş trendinin düşüş trendine geçmesi.

Haber aşırı iyi ise düşüş trendinin yükseliş trendine evrilmesi, dönüşmesi, geçiş yapması.

2- RSI indikatörünün düşeni kırması ile al sinyali vermesi.

ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

Mesela alttaki grafik RSI indikatörüdür, hareketli ortalamaların birleştirilmiş halidir, RSI hisse veya coin fiyatından genelde önce gider. Hisse fiyatı yatay giderken, RSI düşeni kırarsa yani düşüş trendinin ivmesi azaldığı anda, alım yönlü bir pozisyon alınabilir. RSI normalde 70 ‘in üzeri sat çünkü fiyat çok şişti, 30’un altı al çünkü fiyat çok dipte demektir ama bu indikatörün al ve sat sinyallerinden fakrlı olarak borsacılar düşeni kırmasına da göz hesabı ile bakmaktadır, RSI’da da hisse fiyatını çizer gibi yükseliş düşüş trendi çizip bu trendi aşağı yukarı kırmasına göre, RSI İNDİKATÖRÜNÜN KENDİSİ AL VE YA SAT SİNYALİ ÜRETMEDİĞİ HALDE göz hesabı ile ve RSI’nın asıl amacından farklı olarak borsacılar al ve sat sinyalleri üretebilmektedir ama bunu göz hesabı ile yapmaktadırlar ve bunun kodlamaya dökülmesi iyi olur, artık göz hesabı yerine RSI’nın düşeni kırma tespit uygulaması, RSI düşeni kırma al sat robotu geliştirilse oldukça faydalı olur, işe yarar.

3- Koddaki tahmin yapay zeka mekanizmalarının geliştirilmesi, hisse coin tahmin uygulaması tezlerini taradığımda normalde bir canlı yayın anı yakalama tespit etme yapay zeka geliştirmesi olan yolo v3’ün hisse senedi/ coin gelecek tahmini yaparken kullanıldığında başarı oranının yüksek olduğu yazıyordu, ama yolo v3 eklemeden önce mevcut uygulamamın başarı oranının tespiti ve ardından yolo v3’ün eklenmesi düşünülecektir. Yolo v3 dışında farklı yapay zeka modelleri de gelişmeler ve araştırmalar ışığında uygulamaya eklenebilir.

4- Başarı oranı eklenecek

Uygulama tahmin yapıyor fakat tahmin et’e tekrar basınca farklı bir tahmin yapıyor. Bu program hakkında neyi gösteriyor? Programın tahminleri iddialı değil mi? Akademik tezleri incelerken herkes programının başarı oranından bahsediyordu. Benim uygulamam her seferinde farklı bir tahmin yapıyor. Peki böyle bir durumda bu uygulamanın bir başarı oranı şudur budur diyebilir miyiz? Uygulamayı tek bir tahmin yapar hale mi dönüştürelim yoksa böyle kalsın mı sence?