Ev Fiyat Tahmini
Proje Raporu

Teslim Tarihi: 6 Haziran 2021

Ekip:
Ayşenaz BALİOĞLU
Beste ŞENGÜL
Sıla KÖLELİ

Ders:
INF 103 | Algoritma ve İleri Bilgisayar Programlama

İçindekiler Tablosu

Giriş

Projenin Amacı

Ev fiyat tahmini projesi, hâlihazırda pek çok ev bilgisini barındıran bir veri setinden kullanıcının talep ettiği bilgileri kullanıcıya en anlaşılır şekilde ulaştırır. Aynı zamanda kullanıcıya yeni bir evin fiyat tahmini konusunda kolay erişilebilir hizmet sunmaktadır. Bunlara paralel olarak projenin amacı kullanıcıya talep edilen ev bilgilerini en verimli ve anlaşılır şekilde vermek ve fiyat bilgisi olmayan evler için en uygun/doğru fiyat tahmininde bulunmaktır.

Gerçek Hayat Uygulamaları

Yazılan bu program herkese farklı amaçlar çerçevesinde fayda sağlayabilir. Emlakçılar veya ev almak, satmak, ekspertiz yapmak, yatırım yapmakla ilgilenen herhangi bir insan grubu bu programdan yararlanabilir. Programın çalışmasındaki en önemli unsur veri setinin doğru seçilmesidir. Bu noktada- veri setinin seçimi kullanıcıya kaldığından- bir emlakçı, ev almak isteyen herhangi bir kişiden daha fazla veriyi daha kısa sürede toplayabilir. Veriye ulaşmanın kolaylığı, emlakçıların herhangi bir vatandaşa göre bu programdan daha çok verim almasını sağlayabilir.

Emlakçıların en büyük sıkıntılarından biri ellerinde bulunan evlerin bilgilerine ulaşmak ve verilerin analizini yapmaktır. Evlerin bilgileri depolamak amacıyla sıklıkla Access, Excel gibi uygulamaları veya CSV dosyalarını kullanmak durumunda kalan emlakçıların, sonrasında aradıkları ölçütlere uygun evleri bulabilmek için bu uygulamalara hâkim olmaları gerekmektedir. Bu, herkesin üstesinden kolaylıkla gelebileceği süreç olmayabilir. Yazdığımız emlak programının yaralarından biri de programın, ilgilenen bilgiye ulaşımı kolaylaştırarak emlakçıları bu zorunluluktan kurtarabilecek potansiyele sahip olmasıdır.

Kullanılan Teknolojiler

Projemiz GitHub üzerinden yürütüldüğünden proje yönetiminde Git teknolojisinden faydalanılmıştır. Kodun yazılması için Sublime Text text editörü ve Code::Bloks IDE'si kullanılmıştır. Kod optimizasyonu hata kontrolleri için Valgrind, Code::Bloks ve GDB(GNU debugger) hata ayıklayıcıları kullanılmıştır. Rapor yazımı Word'de yazılmıştır.

Uzaktan çalışan bir ekibin ürünü olan projemiz için GitHub görev dağılımını ve takibini hızlandırmış, düzene sokmuştur. Sublime Text kodumuzu daha rahat yazmamızı ve Code::Bloks IDE'si ile GDB ise daha verimli bir şekilde segmentation fault'ları kontrol etmemizi sağlamıştır. Valgrind, bellek sızıntılarının kontrol edilmesi ve sızıntı varsa önlenmesi bakımından kod yazım sürecinde büyük rol oynamıştır.

Programda kullanılan `mean_sale_prices()` fonksiyonu yazımında doğru değerlerin olduğu bir kontrol verisi oluşturabilmek için Excel üzerinden CSV dosyalarından gerekli filtrelemelerle her struct elemanın çeşidi için ev fiyat ortalamaları elde edildi. Bu fonksiyonun çıktısıyla Excel'den elde edilen veriler kıyaslanarak `mean_sale_prices()`'ın doğruluğu kontrol edildi. Ayrıca fiyat tahmini için oluşturduğumuz iki modelimizin performansını kıyaslamak için yine Excel üzerinden modellemeler sonucu oluşan çıktılar görselleştirildi.

Projemizin Diğer Çözümlerden Farkı

Grup olarak, veri setinin içerdiği dosyalarda bulunan bilgileri saklamak üzere Hash Table veri yapısını seçmiş ve ev bilgilerini, temelde anahtar indeksli bir tablo olan bu veri yapısına, her ev farklı bir anahtar değeri alacak şekilde eklemiş olmamız; olası başka çözümlerden bizi ayıran özelliklerden biridir.

Bu özelliklerden bir başkası, programımızın piyasada ve internet ortamında karşımıza çıkan birçok ev fiyatı tahmin programının aksine, tahmin yapmak için kullanıcıya iki farklı metot arasında seçim olanağı sunuyor olmasıdır. Buna ek olarak yazdığımız program, ev tahmini yapmanın ötesinde başka işlevler de görmektedir. Programımızın en önemli özelliklerinden biri, veri analizi için yazdığımız fonksiyonların istenen bilgiye kolayca ulaşımı sağlıyor olmalarıdır.

Proje Mimarisi/Yapısı

Süreç

Kod Yapısı

Seçtiğimiz projeyle gelen dosyalarda belirli bir iskelet kodu verilmemiştir. Ancak, proje açıklama dosyasında bizden beklenen temel fonksiyonların açıklamaları ve bu fonksiyonları kurarken bize yardımcı olması amacıyla konulmuş bazı örnekler açıklanmıştır. Kurduğumuz algoritmaya bağlı olarak bu örnek fonksiyonları yenileyip bazı yerlerde parametrelere eklemelerde bulunmuştur. Yaptığımız bu eklemelerdeki genel amaç, fonksiyonların en verimli şekilde çalışmasını sağlamaktır.

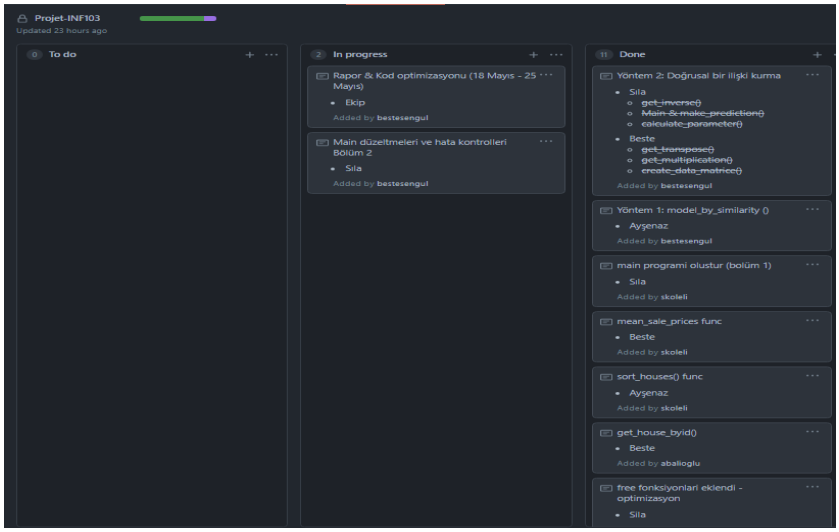
Ayrıca, bizden beklenen fonksiyonlar dışında, projenin hem veri analizi hem de model oluşturma bölümlerinde kullanmak üzere projemize birçok fonksiyon eklenmiştir. Raporumuzun fonksiyon açıklamalarının bulunduğu kısmında eklediğimiz fonksiyonların programda aldığı görevler üzerine detaylı bilgi bulmak mümkündür.

İzlenen Yol Haritası

Projenin tamamlanması amacıyla verilen sürenin neredeyse hepsi kullanılmıştır. Bu süre zarfında iş yükünün belirli bölümlere ayrılması ve bu işlerin tamamlanması için zaman kısıtlamalarının konulması projenin düzenli bir program çerçevesinde bitirilmesini sağlamıştır. Projenin ikinci kısmı olan “model oluşturmanın” temelinde “veri analizi” yattığı için projenin başlangıcında bu bölüm üzerine dikkatlice çalışılmıştır. Main programı (kullanıcı arayüzü) ise, bütün süreç boyunca gerekli eklemeler yapılarak oluşturulmuştur.

Ekip Çalışması Süreci

Microsoft Teams üzerinden yapılan haftalık toplantılar, proje kodunun GitHub platformunda tutulmuş olması ve iş bölümü kontrolleri için GitHub projesi açılması, pandemi koşullarına rağmen grupça sorunsuz bir şekilde çalışmamıza büyük katkıları olmuştur.

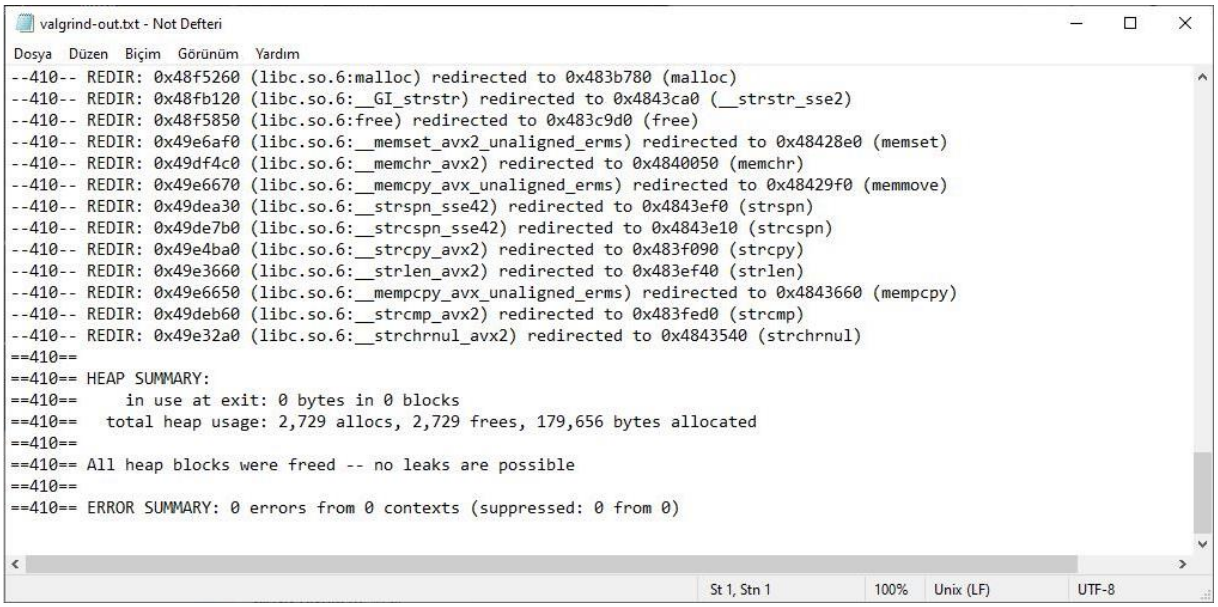


Resim_1: GitHub İş Bölümü ve Planlama

Toplantılar sırasında oluşturulan haftalık programlar ve görevler, projede düzenli bir şekilde ilerlenmesine yardımcı olurken; Git teknolojisi ve sunduğu özellikler sayesinde kodda kimin nasıl değişikliklerde bulunduğunu kolayca gözlemleme fırsatımız olmuştur.

Hata Ayıklama Süreci

Programlama sürecinde, hafıza hatalarıyla oldukça sık karşılaşılmıştır. Derleme sürecini terminal üzerinden "make" komutu ile gerçekleştirildiği için fark edilmesi güç olan bu hataların varlığı, bellek hata ve sızıntılarını algılama amacıyla kullanılan Valgrind isimli programlama aracı sayesinde saptanmıştır. Hataları ayıklama sürecinde ise Valgrind'in oluşturduğu çıktıları detaylı bir şekilde incelenmiş ve internet ortamında kapsamlı araştırmalar yaparak hataların temeline ulaşılmıştır.



Resim_2: Valgrind kontrolleri

Veri Analizi Bölümü (Birinci Bölüm)

Dosya Sistemi & Veriye Kısa Bir Bakış

Tüm kaynak kod dosyaları `src` adı verilen klasör altında tutulmaktadır. CSV tipinde iki adet veri dosyamız ise `dataset` adı altındaki dosyada yer almaktadır. Verilerimizden ilki `house_price_data.csv`'de toplam 9, ikincisi `house_price_test.csv`'de toplam 8 sütun bulunmaktadır. `house_price_test.csv`'de dosyasında eksik olan kısım `Sale Price` adında bir kolondur.

Verideki kolonlar:

1. `ID`: Her evin sahip olduğu eşsiz kimlik numarasıdır. house_price_data.csv dosyasında bulunan evlerin ID'leri 101 ile 1460 ; `house_price_test.csv` dosyasının içerdiği evlerin ID'leri ise 1 ve 100 değerleri arasındadır.
2. `Lot Area`: Evlerin tüm alanlarının ne kadar büyüklükte olduğunun bilgisini saklayan kolondur. `house_price_data.csv`'de 1007 farklı alan bilgisi tutulmaktadır.
3. `Sale Price`: Evlerin satış fiyatlarını tutar.
4. Street: Her evin bulunduğu sokağın adı yer alır. `house_price_data.csv`'de iki farklı sokak bulunmaktadır.
5. `Overall Qual`: Her evin genel kalitesinin 10 üstünden kaç olduğunu tutan kolondur.
6. `Overall Cond`: Her evin genel durumunun 9 üstünden puanını tutar.
7. `Kitchen Qual`: Evlerin mutfak kalitesini tutan kolondur. Derecelendirme; `Ex` (Excellent), `Gd` (Good), `TA` (Typical/Average) ve `Fa`(Fair) değerleri arasındadır.
8. `Year Buil`t: Her evin inşa edildiği yılın bilgisini tutar. `house_price_data.csv`'de 112 farklı inşa yılı bulunmaktadır.
9. `Neighborhood`: Evlerin bulunduğu mahalleyi saklayan kolondur. `house_price_data.csv`'de 25 faklı mahalle bilgisi bulunmaktadır.

Program Derlenmesi & Kullanıcı Arayüzü

Kullanıcı programı önce Linux tabanlı bir işletim sistemi üzerinde, terminalde `make` komutu ile derlemelidir. Program 3 argüman alır (Bkz.):

1. çalıştırılabilir dosya,
2. veri toplamak için uygun bir CSV dosyası,
3. fiyat tahmini için fiyat bilgileri olmayan evlerden oluşan bir CSV dosyası.

Programı denemek isteyen bir kişi, program kodlarının yanında verilen veri dosyalarını kullanarak `./main house_price_data.csv house_price_test.csv` şeklinde programı çalıştırabilir. Eğer parametre olarak verilen dosyalar açılabilir ve veriler programa doğru şeklide aktarılabilirse ekrana ana menü basılacaktır. Bu menüde 8 farklı işleme izin verilmektedir:

1. Evleri listele: print_houseList() fonksiyonu sayesinde tüm evler ve özellikleri ekrana basılır.
2. ID değeri ile ev bul: `get_house_byid()` fonksiyonu sayesinde ID numarası verilen bir evin bilgileri ekrana basılır.
3. ID değeri verilen evin komşu evlerini bul: Önce `get_house_byid()` fonksiyonu ile istenilen evin olup olmadığı kontrol edilir ve varsa o eve ulaşılır. Ardından `count_neighborhood()` ve `get_neighborhoods()` fonksiyonları sayesinde verilen ID numarasına sahip evin aynı komşuluğunda bulunan evler ekrana bastırılır.
4. Kritere göre ortalama fiyatları listele: Ekrana 7 farklı ev özelliğini bulunduran bir seçenek menüsü gelir ve bu menüde gösterilen ev özelliklerinden birinin girilmesi istenir. Ardından `mean_sale_prices()` fonksiyonu sayesinde istenen kritere göre fiyat ortalamaları ekrana basılır.
5. En yüksek fiyata sahip ilk 10 evi göster: `quicksort()` fonksiyonu ile evler sıralandıktan sonra en yüksek fiyata sahip ilk on ev ekrana basılır.
6. Evleri sırala & kaydet: Önce ekranda gösterilen 8 seçenekli kriter değerinin numarası girilir. Ardından `sort_houses()` fonksiyonu sayesinde evler istenen kritere göre sıralanır ve ayrı bir `*.csv` uzantılı dosyaya kaydedilir.
7. Evler için fiyat tahmini yap (model_by_similarity): `model_by_similartiy()` fonksiyonu burada kullanılarak birinci metot uygulanır.
8. Evler için fiyat tahmini (doğrusal ilişki): Matris yöntemi hesaplamaları kullanılan ikinci metot burada kullanılır.

Kullanıcı programdan çıkmak istiyorsa 0'a basması ya da terminal penceresini kapatması gerekir. Aksi taktirde her işlem sonunda ana menüye geri dönülecektir. Ana menüyü tekrar görmek için ise 9'a basılmalıdır.

```
sila@silaK-X555LN:~$ cd Desktop
sila@silaK-X555LN:~/Desktop$ cd Proje
sila@silaK-X555LN:~/Desktop/Proje$ cd src
sila@silaK-X555LN:~/Desktop/Proje/src$ make
gcc -I . -c main.c -Wall -ggdb3
gcc -I . -c dataset.c -Wall -ggdb3
gcc -I . -c models.c -Wall -ggdb3
gcc main.o dataset.o models.o -o main -Wall -ggdb3
sila@silaK-X555LN:~/Desktop/Proje/src$ ./main ~/Desktop/Proje/dataset/
house_price_data.csv ~/Desktop/Proje/dataset/house_price_test.csv

Emlak Programina Hos Geldiniz!
_____________________________________________________________

Yapmak istediginiz islemin numarasini asagidan secebilirsiniz:

1 - Evleri listele
2 - ID degeri ile ev bul
3 - ID degeri verilen evin komsu evlerini bul
4 - Kritere gore ortalama fiyatlari listele
5 - En yuksek fiyata sahip ilk 10 evi goster
6 - Evleri sirala & kaydet
7 - Evler icin fiyat tahmini yap (model_by_similarity)
8 - Evler icin fiyat tahmini (dogrusal iliski)
Programdan cikmak icin 0'a basiniz.
Yapmak istediginiz islem:
2
Yonlendiriliyorsunuz..
_____________________________________________________________
Aramak istediginiz evin ID numarasini giriniz:
45
Araniyor...
45 ID nolu bir ev bulunamadi.
Ana menuye donuluyor..
_____________________________________________________________

(Ana menu seceneklerini tekrar gormek icin 9, programdan cikmak icin 0
 tuslayiniz.)
Yapmak istediginiz islem:
```

Resim_3_1: Kullanıcı Arayüzü – Parametreler ve Menü Görünümü

```
4 - Kritere gore ortalama fiyatlari listele
5 - En yuksek fiyata sahip ilk 10 evi goster
6 - Evleri sirala & kaydet
7 - Evler icin fiyat tahmini yap (model_by_similarity)
8 - Evler icin fiyat tahmini (dogrusal iliski)
Programdan cikmak icin 0'a basiniz.
Yapmak istediginiz islem:
4
Yonlendiriliyorsunuz..
_____________________________________________________________
Kriterler:
1 - Ev alani
2 - Sokak adi
3 - Mahalle adi
4 - Imar yili
5 - Genel niteligi
6 - Genel durumu
7 - Mutfak kalitesi

Fiyat ortalamarini gormek istediginiz kriterin numarasini seciniz:
2
Sokaklara gore evlerin fiyat ortalamasi:
capacity: 2
(    1 ) Mean =           134228.594    Criteria type:            Grvl
(    2 ) Mean =           181617.438    Criteria type:            Pave
 *mean_sale_prices()              street adina basarili*
Ana menuye donuluyor..
_____________________________________________________________

(Ana menu seceneklerini tekrar gormek icin 9, programdan cikmak icin 0
 tuslayiniz.)
Yapmak istediginiz islem:
0
Yonlendiriliyorsunuz..
_____________________________________________________________
Cikis yapiliyor
Programi kullandiginiz icin tesekkurler!
sila@silaK-X555LN:~/Desktop/Proje/src$
```

Resim_3_2: Kullanıcı Arayüzü – Çıktı Görünümü ve Programdan

Kullanılan Structure'lar

Projemizde iki tane struct bulunmaktadır:

```
typedef struct house {
  int id;
  int lotarea ;
  char street[15] ;
  int saleprice ;
  char neighborhood[15] ;
  int yearbuilt ;
  int overallqual ;
  int overallcond ;
  char kitchenqual[5] ;
 }House;
```

```
typedef struct count{
  int row;
  int column;
 }Count;
```

Programda kullanılan veri yapısı hash tablosudur. Bize sağladığı en büyük avantaj hızlı bir şekilde istenilen evlere, evlerin sahip olduğu ID'ler ile ulaşabilmemizdir. House struct'ı sayesinde bir mainde `House **table` formatında bir Hash Tablosu oluşturulmuştur. `House` ve `Count` struct'ları dataset.h header dosyasında tanımlanmıştır. `Count` struct'ı veri analizinde ihtiyaç duyulduğu için sadece dataset.c dosyasında daha çok kullanılmıştır. models.h header dosyasına dataset.h header dosyası eklendiği için iki struct da models.h'ta tanımlıdır. `House` struct'ını models.h ve models.c dosyalarında kullanılmıştır.

Veri Analizi Fonksiyonları (DATASET.H)

- `int hash_compute(int id, int birinciid)`

Veri setinin içerdiği dosyalardaki bilgileri saklamak üzere seçtiğimiz Hash Table (Karma Tablo) veri yapısı, anahtar indeksli bir tablo yapısıdır. Bu fonksiyon ile her eleman, indeksini belirleyen bir anahtar ile eşleşir. Veri setinin içeriği incelendiğinde her evin ID değerinin birbirinden olduğu görülür. Bu, her ev için farklı birer anahtar değeri atamayı mümkün kılar. Bu, çakışmalara mâni olmanın yanı sıra aranan evin ID değeri sayesinde kolayca bulunmasına yardımcı olur. `id`, anahtar değerini bulmak istediğimizi evin; `birinciid` ise bu evin bulunduğu CSV dosyasındaki ilk evin ID'sidir.

- `House** create_table(int size)`

`size` değeri kadar evi taşıyacak bir hash tablosu oluşturur ve döndürür. `malloc()` fonksiyonu ile hafızada yer açar ve açılan yeri boşaltır. `get_neighborhoods()` ve `model_by_similarity()` fonksiyonlarında kullanılmıştır.

- `void read_house_data(char* dosya_adi, House** houses, Count size)`

`void` olarak çağırılan bu fonksiyon, `dosya_adi` ismli dosyayı okur ve dosyadaki evleri `houses` hash tablosuna `hash compute()` fonksiyonu ile anahtar değerini belirleyerek yerleştirir. CSV (Comma Separated Values) formatlı dosyalarındaki veriyi böler, ayrılmış değerleri `House` yapımızın doğru alt yapısına atar. `Count size` parametre olarak `house_price_data.csv` dosyasını almış olan `count_house()` fonksiyonun döndürdüğü değerdir. Bu değerin `row` alt yapısı dosyanın içerdiği ev sayısına eş değerken, `column` alt yapısı okunan dosyada satış değerlerine ait bir kolonun var olup olmamasına göre değişir.

- `Count count_house(char dosya_adi[30])`

(Bkz.: `read_house_data()`)

- `int count_neighborhood(House* house, House** houses, int size)`

`house` ile `houses`'da bulunan evlerden kaç tanesinin aynı komşulukta bulunduğunu belirler ve döndürür. Bunu, bütün tabloyu tarayıp karşılaştırmalar yaparak gerçekleştirir. `get_neighborhoods()` fonksiyonunu kolaylaştırmak için yazılmıştır, bu fonksiyon içerisindeki `create_table()` fonksiyonuna parametre olarak yazılır. `size`, `houses`'ın taşıdığı ev sayısına eşdeğerdir.

- `void print_house(House* house)`

Verilen `house` evinin sahip olduğu id, alan, sokak, mahalle, fiyat, inşa edildiği sene, genel durumu, genel kalitesi ve mutfak kalitesi bilgilerini ekrana basar.

- `void print_houseList(House** houses, int size)`

`houses` listesini ekrana basar. Bu işlemleri `print_house()` sayesinde gerçekleştirir.

- `House* get_house_byid(House** houses, int id, int sizeofhouses)`

`houses` hash tablosunun içinde bulunan evlerden `id`'yi ve `sizeofhouses`'ı `hash_compute()`'a vererek istenen evin hash değerine ulaşılır. Bu değer sayesinde `id`'si verilen ev döndürülür.

- `House** get_neighborhoods(House* house, House** houses, int size)`

`house` aynı komşulukta bulanan evleri döndürür. Bunu yapmak için önce `houses` hash tablosunun içinde bulunan evleri tarar ve aranan komşulukta bulunan evleri yeni bir hash tablosuna ekler. Fonksiyon, yeni oluşturulan hash tablosunu döndürür. `size`, `houses`'da bulunan ev sayısına eşdeğerdir.

- `void swap(House* a, House* b)`

`void` olarak çağırılan bu fonksiyon, `a` ve `b` evlerinin yerlerini değiş tokuş eder. `quicksort()` fonksiyonu için yazılmıştır.

- `void sort_houses(House**houses, int first, int last, char* criter_name)`

`houses` hash tablosunu `criter_name`'e göre büyükten küçüğe veya alfabetik olarak "Quick Sort" mantığına göre sıralar ve sıralı listeyi istenen kriterin adında bir CSV dosyası açıp kaydeder. Sıralama, `quicksort()` fonksiyonu çağrılarak yapılır. İki fonksiyon yazmayı seçmemizin nedeni, gerekli olmayan yerlerde dosya açarak programın hızının yavaşlatılmasını engellemektir.

- `void quicksort(House**houses, int first, int last, char* criter_name)`

`houses` hash tablosunu `criter_name`'e göre büyükten küçüğe veya alfabetik olarak, "Quick Sort" mantığını kullanarak sıralar. Quick Sort algoritması temelde; diziyi bölme ve oluşan alt dizileri rekürsif olarak sıralama (Divide and Conquer) adımlarını izler. `first` ve `last` parametreleri sırasıyla `House** houses`'ın ilk ve son indeksleridir.

- `void mean_sale_prices(House ** houses, int size, char* criter_name)`

`houses` hash tablosundan verilen `criter_name`'e göre ilgili hash tablosu elemanının tüm farklı değerleri için ortalama hesabı yapar ve bu hesaplamaları ekrana basar. En başta `criter_name`'e göre o struct elemanından kaç çeşit bulunduğu saptanır ve çeşit miktarına göre ortalama hesabı için ihtiyaç duyulan üç dizi için yeterli alan açar: her çeşide göre ev fiyatlarının toplamını tutan `toplam`, struct elemanın her çeşidinden kaç tane olduğunu tutan `miktar` ve he çeşidin toplam değerini miktar değerine bölüp ortalamalarını tutan `mean`. Tüm işlemlerden sonra bu üç listenin yeri `free()` fonksiyonu ile heap belleğine iade edilir.

- `void delete_table(House** table, int size)`

İçine gönderilen `House` tipindeki hash table için ayrılan yerleri `free()` fonksiyonu yardımıyla iade eder.

- `void delay(double dly)`

Çağırıldığı yerde `dly` kadar saniye boyunca programın ilerleyişini durdurur.

Model Tasarımı (İkinci Bölüm)

Modeller & Fiyat Tahmini Fonksiyonları(MODELS.H)

➢ Metot 1

- `int model_by_similarity(House**houses, int size_houses, House* new_house)`

`new_house`'un satış değerini, `houses` hash tablosundaki diğer evlerle karşılaştırmalar yapıp, `new_house`'a en çok benzeyen `houses` elemanların fiyatlarının ortalamasını alarak tahmin eder. `size_houses`, `houses`'ın taşıdığı ev sayısına eşittir. Bu fonksiyon emlak programımızda `house_price_test.csv` dosyasındaki evlerin `saleprice` değerlerini tahmin etmek ve bu değerleri `prices_by_similarity.txt` bastırmak için kullanılır.

➢ Metot 2

- `void create_data_matrice(House** houses, float* X, int Xpitch, int Xrow, float* Y, int Ypitch, int Yrow)`

`houses` hash tablosu içinde yer alan tüm evlerin alan ve fiyat bilgilerini iki adet tek boyutlu dizi olarak geri döndürür. `X` matrisi içinde evlerin alan bilgilerini taşır ve boyutu $[evMiktarı]\ x\ 1$'dir. `Y` matrisi evlerin fiyat bilgilerini tutar ve boyutu $[evMiktarı]\ x\ 2$ 'dir. İlk sütunu 1 değeri ile doludur. İkinci sütunda ise fiyat bilgileri yer alır. Bunun amacı ise `X` ile `W` matrisleri arasında gerçekleşmesini istediğimiz çarpımın kolaylıkla yapılabilmesidir.

- `void print_data_matrice(float* matrice, int matricePitch, int row)`

Parametre olarak verilen `matrice`'i ekrana basar. Bu işlemi gerçekleştirebilmek için `matricePitch` (verilen `matrice`'in sütun sayısı) ve `row` (verilen `matrice`'in satır sayısı) kullanılır. Basılan her satır için satır sayısı da ekrana basılır.

- `void get_multiplication_normal(float* A,int Apitch, int Arow, float* B,int Bpitch, int Brow, float* S,int* Spitch, int* Srow)`

`A` ve `B` matrislerinin matris çarpımı `S` matrisi içinde saklanır. `Apitch` (birinci matrisin sütun sayısı) ile `Brow` (ikinci matrisin satır sayısı) kıyaslanarak matris çarpımı için boyut uygunluğu kontrol ediliyor. Eğer birbirilerine eşitlerse klasik matris çarpımı algoritması kullanılarak `S` yeni matris elde edilir. Çarpma işlemi öncesi `Spitch` ve `Srow` değerleri de güncellendiğinden `S` matrisinin her türlü bilgisi bu fonksiyondan sonra güncellenmiş olur.

- `void get_transpose(float* A, int Apitch, int Arow, float* S, int* Spitch, int* Srow)`

`A` matrisinin devriği `S` matrisinde saklanır. Fonksiyon sonunda transpoze sonucu oluşan S matrisinin sütun ve satır değerleri güncellenir. Guncellenebilmeleri icin `Srow` ve `Spitch` adresleri ile alınıyorlar. S matrisinin sütun değerini taşıyan değişken `Srow`'dur ve bu değişkeninin değeri işlem sonunda `Apitch`'e eşittir. `Spitch` ise `S` matrisinin satır sayısıdır ve işlem sonunda yeni değeri `Arow`'a eşitlenir.

- `int get_inverse(float *A, int Apitch, int Arow,`
  `float *I, int *Ipitch, int *Irow)`

$Arow\ x\ Apitch$ boyutlarındaki bir `A` matrisinin önce tersi alınıp alınamayacağını kontrol eder, alınamıyorsa 0 gönderir. Eğer tersi alınabiliyorsa formülüyle $inverse(A) = adjoint(A)/determinant(A)$ fonksiyonun tersini hesaplayıp `I` matrisine gönderir. Bütün işlemler bittikten sonra 1 gönderir.

- `void adjoint(float* A, int Apitch, int Arow,`
  `float* S, int Spitch, int Srow)`

`Apitch` kadar sütun ve `Arow` kadar satıra sahip `A` matrisinin kofaktör matrisinin(ek matrisinin) tersini hesaplayıp `S` matrisine yazar.

- `void getCofactor(float* A, int Apitch, int Arow,`
  `float* C, int *Cpitch, int *Crow)`

`Apitch` kadar sütun ve `Arow` kadar satıra sahip `A` matrisinin kofaktör matrisini(ek matrisini) hesaplayıp `C` matrisine yazar.

- `int is_invertible(float* A, int Apitch, int Arow)`

`Apitch` sütunu, `Arow` satıra olan `A` matrisinin ters çevrilebilir olup olmadığını hesaplar. Çevrilebiliyorsa 1, çevrilemiyorsa 0 döndürür.

- `float getDeterminant(float* A, int Apitch, int Arow)`

`Apitch` sütunu ve `Arow` satırı olan bir `A` matrisinin determinant değerini hesaplayıp geri döndürür.

- `void make_prediction(House** test_list,int listSize,float* coefficients, int Cpitch, int Crow)`

`test_list`'ten aldığı fiyat bilgisi olmayan evlerin tahmini fiyatını, `coefficients` matrisini kullanarak oluşturduğu yeni bir dosyaya yazdırır.

- `float* calculate_parameter(float* X, int Xpitch, int Xrow,`
  `float* Y, int Ypitch, int Yrow)`

$Xrow\ x\ Xpitch$ boyutlarındaki `X` matrisini ve $Yrow\ x\ Ypitch$ boyutlarındaki Y matrislerini kullanarak hesaplar yapar ve oluşturduğu yeni bir `W` matrisine yaptığı hesapların sonucunu yazar. Bu `W` matrisini döndürür.

Model 1 VS. Model 2

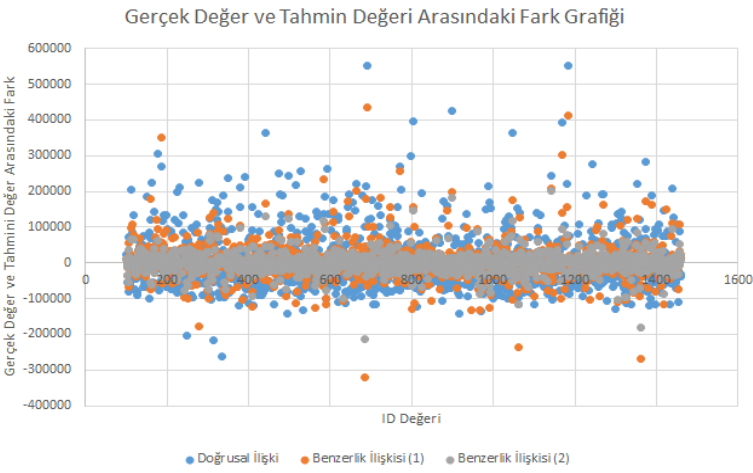
Ev fiyatı tahmin modellerinin yazım sürecinde kullanmamız için bize verilen `house_price_test.csv` dosyasında bulunan evlerin model performansını ölçebilmemiz için verilmiş bir gerçek fiyat listesi bulunmaması nedeniyle, bütün denemelerimiz `house_price_data.csv`'de bulunan ve gerçek `saleprice` değerleri bulunan evler üzerinde yapılmıştır. Bu noktadan sonra yazılan her çıkarım, bu dosya için yapılan tahminler üzerinedir. Eklenen bütün grafikler- aksi belirtilmedikçe- bu dosya üzerinden elde edilen verileri kapsar. Denemelerimizde çıkan sonuçların görselleştirilmesiyle elde ettiğimiz veriye/verilere göre; benzerlik ilişkisine bağlı olarak oluşturulan ev fiyatı tahmin modeli (`model_by_similarity()`), doğrusal ilişkiyi baz alan modelimize göre daha iyi bir sonuç vermiştir.

Sonucun bu şekilde olmasında birkaç farklı etken rol oynamaktadır. Bunlardan biri: benzerliğe dayalı olan modelin oluşturulma sürecinde birçok farklı algoritmanın denenmiş ve bütün bu denemeler arasında her parametrenin en iyi sonuç veren halinin seçilmiş olmasıdır. `AREA` ve `YEAR` değişkenlerin aldıkları değerler ve genel kalite değerlerinin (`overallqual`, `overallcond`, `kitchenqual`) koda dahil ediliş şeklini en optimal hale getirme üzerinde olan çalışmalarımız modelimizin başarısını arttırmıştır.

Doğrusal ilişki üzerine kurulu olan modelin doğru tahmin yapmada daha düşük performans göstermiş olmasının bir diğer nedeni de aslında temelde birinci nedenle aynıdır: optimizasyon. Bu; doğrusal ilişki modeli kurulum sürecinde model optimizasyonuna her ne kadar `model_by_similarity()`'de ayrılmış olana eş değerde zaman ve efor ayrılmış olsa da güncel bilgi birikimimizi aşan bazı faktörlerin devreye girmiş olmasından kaynaklanmaktadır. Örnek olarak, normal şartlarda doğrusal ilişki kurulurken verideki "gürültü"nün de hesaba katılması gerekmektedir. Veriler genellikle bir hata içermektedir ve bu hatalar toplu olarak gürültü olarak adlandırılmaktadır. Tipik olarak verilerde ne kadar çok oranda gürültü varsa, güvenilir sonuçlara ulaşmak o kadar güçleşecektir. Makine öğrenim yöntemleri bu gürültüyle çalışmayı mümkün kılar ancak bu proje için bunu göz ardı etmekteyiz. Bu modelin optimizasyonu hakkında daha fazla bilgiye raporun Sonuçlar: Veri Analizi ve Model Çıkarımları erişilebilir.

Aşağıda verilen grafik bizlere `house_price_data.csv` dosyasında bulunan evlerin, yine bu dosyada verilmiş olan gerçek satış değerleri ve kurduğumuz modellerle elde ettiğimiz tahmini satış değerleri arasındaki farkı gösterir. Sıfır hattından en az sapma gösteren model en gerçekçi tahminleri yapmaktadır- yani en başarılı modeldir.

Doğrusal ilişkiye dayalı modeli (mavi) benzerlik ilişkisine dayalı olandan (turuncu ve gri) daha düşük performans gösterdiği grafikte net olarak görülmektedir. Grafiğe bakıldığında benzerlik ilişkisine dayalı olan model adına iki ayrı alanın varlığı görülebilir. Bunun nedeni fiyatı tahmin edilmeye çalışılan eve `house_price_data.csv` 'deki evlerden en benzeyenleri bularak çalışan bu algoritmanın, fiyatı tahmin edilmek istenen evin de `house_price_data.csv`'deki ev listesinin bir parçası olduğu çoğu durumda "en benzer ev" olarak direkt kendisini seçmesidir. Bu durumda çıkan sonuçlar, `model_by_similarity()`'nin gerçek tahmin yapma performansını yansıtmaz. Fiyatı aranan evin kendisini seçmesini engellendiğinde (ID değerleriyle) problem çözümlenir. `model_by_similarity()`'nin tahmin yetisinin gerçek göstergesi turuncuyla gösterilen kısımdır (Benzerlik İlişkisi (1)).



Resim_4: Gerçek Değer & Tahmini Değerler Fark Grafiği

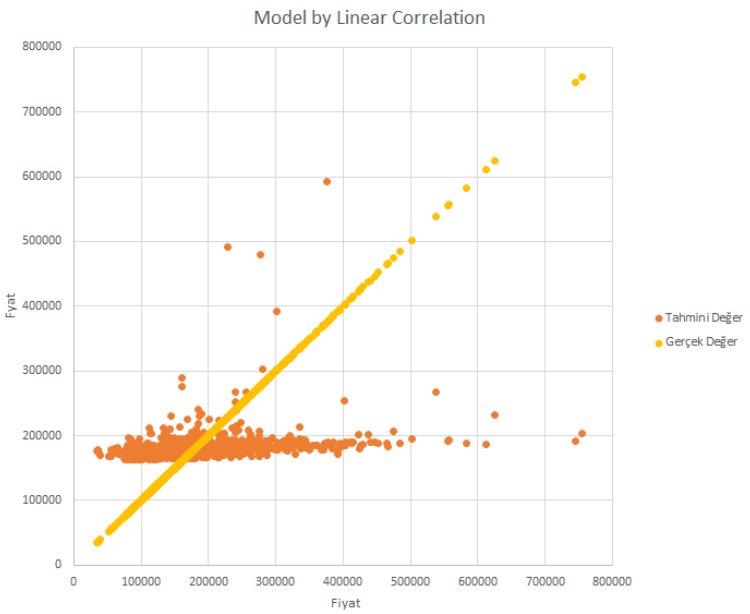
Sonuçlar

Veri Analizi & Model Çıkarımları

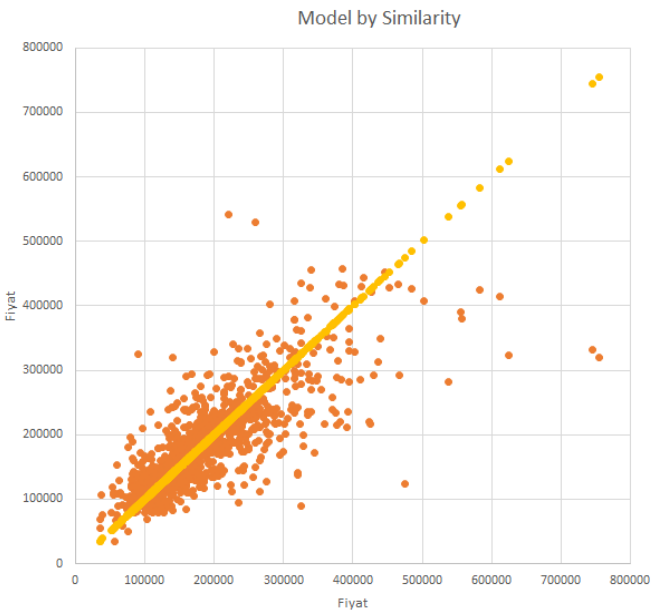
Veri analizinde elimizde bulunan `house_price_data.csv` ve `house_price_test.csv` dosyalarında bulunan verileri daha yakından tanımamıza yardımcı olmuştur. Veri belirli bir sokak ve komşuluklardaki evleri ve bu evlere ait ev alanı, imar yılı, genel durum, genel kalite ve mutfak kalitesi bilgilerini kapsamaktadır. Bu bilgiler üzerinde projenin veri analizi kısmında yaptığımız çalışmalardan model tasarımı bölümünde yararlanılmıştır.

Model tasarımında ise -daha önceki bölümde de anlatıldığı gibi- `house_price_data.csv` üstünde çalışınca `model_by_simlarity()`'nin diğer modele oranla çok daha yakın tahminlerde bulunduğunu gözlemledik. Doğrusal ilişki kullanılan modellemede `model_by_similarity()`'nin sonuçlarıyla orantılı değerler elde edilmiştir. Ancak, bu modelden elde edilen tahminlerin genelde 200 bin bandında yığılma gösterdiği ve bu nedenle, gerçek fiyatın en yüksek olduğu yerlerde çok büyük oranda bir sapma yaşandığı gözlenmektedir. Doğrusal ilişki modelinin belli bir değerde (150-200 bin bandında) yığılma yaratma eğilimini aşağıda bulunan grafiklerden hem Model by Linear Correlation'da hem de Test Dosyası Fiyat Tahminlerinde gözlemlememiz mümkündür. Bu yığılmanın nedenlerinden bazıları: hesaplanan W matrisinin değeri ve verimizde göz ardı ettiğimiz "gürültü"dür. Daha stabil fonksiyonlar kullanılarak; verimizi, bir veri ön işleme tekniği olan veri temizleme işlemine tabii tutarak veya daha iyi bir veri seti kullanarak bu sıkıntıyı minimale indirgememiz mümkündür.

Aşağıda bulunan Model by Linear Correlation ve Model by Similarity ismli grafikler bizlere, bu modellerle elde edilen değerlerin gerçek değerlere göre oranını gösterir. Sonuçlar, gerçek değerleri üzerinde barındıran $x = y$ doğrusundan ne kadar uzaklaşırsa sapma da o kadar fazladır. En çok sapma Model by Linear Correlation'da gözlemlenir.

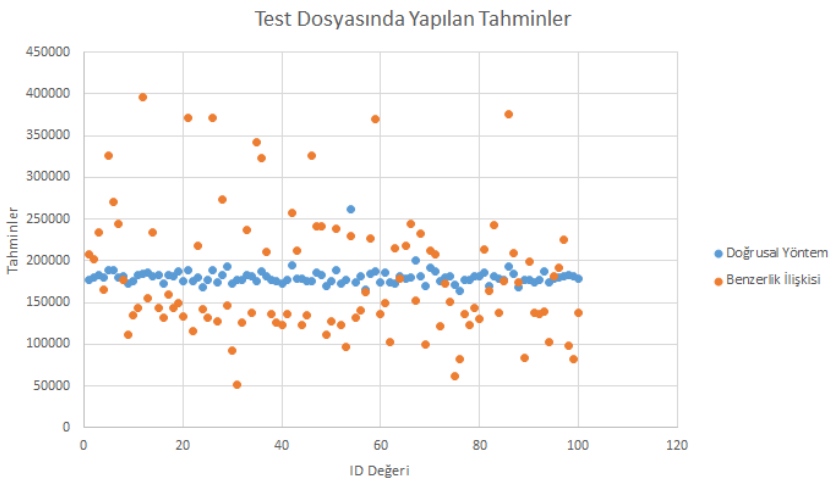


Resim_5: Model by Linear Correlation



Resim_6: Model by Similarity

Test Dosyası Fiyat Tahminleri grafiği bizlere sadece `house_price_test.csv` dosyası üzerinden yapılan fiyat tahmini sonuçlarını verir. Bu tahminlerin doğruluğunu karşılaştırabileceğimiz gerçek değerler bulunmamaktadır, bu nedenle bu grafik bize modellerin çalışma performansı üzerine bir bilgi vermez.



Resim_7: Test Dosyası Fiyat Tahminleri

Başarabildiklerimiz & Başaramadıklarımız

Bu çalışma bizlere üniversite hayatımızda ilk defa bir bölüm dersinde grup çalışması yapma deneyimini yaşatmış ve birinci sınıfı pandemi koşullarında geçirmiş olan bizlere bölümdeki insanları biraz da olsa tanıma fırsatı sunmuştur. Ayrıca, ilk defa farklı insanlarla beraber kodlama yaparak çalışma tarzlarımız arasındaki farkları gözlemleme şansı yaratmıştır. Bu süreç, hepimize grup arkadaşlarımızdan yeni bilgiler edinme olanağı tanımıştır.

Bütün bu kazanımlara rağmen pandemi nedeniyle grup arkadaşlarımızla yüz yüze çalışamamanın getirdiği eksiklikler bulunmaktadır. Arada sürekli iletişimimizi kısıtlayan ekranların bulunmadığı bir senaryoda belki çok daha farklı kazanımlar elde edilebilirdi.

Yazılım kısmında ise `model_by_similarity()` olabildiğince optimize edilmeye çalışılmıştır. Doğrusal ilişki mantığının kullanıldığı modellemede elde edilen sonuçlar genel olarak `model_by_similarity()` ile orantılı olsa da bazı değerlerin beklenenden yüksek çıkmasının önüne geçilememiştir. Veri analizi kısmında hafızayı iyi kullanabilen ve hızlı çalışabilecek şekilde istenen fonksiyonları kodladığımıza inanıyoruz.

Kapanış & Future Work

Kapanış

Projenin tüm süreçlerinde en verimli şekilde çalışması amaçlanan ve fiyatı bilinen evler üzerinden fiyat tahmini yapabilen bir program oluşturulmuştur. Programın iki ana bölümünden biri olan Veri Analizi kısmında bir veri başarılı bir şeklide;

- ✓ Verilen `*.csv` tipindeki dataset'lerden okunmuştur.
- ✓ Her filtrelemede bir kritere göre filtreleme yapılmıştır.
- ✓ Bu filtrelemelere göre fiyat ortalamaları hesaplanmıştır.
- ✓ İstenen kritere göre veri sıralanmıştır.

Projenin ikinci bölümü olan Model Tasarımı kısmında fiyat bilgilerini barındıran veri ve veri analizi kısmındaki çalışmalar sayesinde;

- ✓ İlk metot tarafından fiyatı tahmin edilmek istenen evin sahip olduğu tüm diğer kriterlerini taşıyan evleri, fiyat bilgilerini barındıran veri içerinden erişerek benzer evlerin fiyat ortalaması tahmini ev fiyatı olarak elde edilmiştir.
- ✓ İkinci metot tarafından fiyat bilgilerini barındıran verideki `Lot Area` kriteri ile `Sale Price` kriteri arasında doğrusal bir ilişki kurularak fiyat bilgisi tahmin edilecek evlere aynı ilişki `Lot Area` değerleri sayesinde uygulanmıştır.

Teorik Olası Çözümler & Yöntemler

Daha İyi Fiyat Tahmini için Alternatif Metot : Rating Sistemi

Puanlama sisteminin temelinde bir evin her özelliği için -bu noktada bir Enum yapısı ya da bir Priority Queue kullanılarak- belirli bir puan üstünden `house_price_test.csv`'teki ev fiyatını tahmin etmektir. Bunun için önceden önceliklendirme yapılması çok önemlidir. Bilindiği üzere evler için `Sale Price` dışında 8 farklı kriter bulunmaktadır. Evlerin fiyatlarının aşırı yüksek ya da düşük çıkmasını önlemek için rating sistemini, kullanıcıdan (emlakçı ya da konut eksperi) en yüksek ev fiyatının alınıp daha önceden belirlenen önceliklendirmeye göre her kriterin maksimum fiyatı belirlenir. Bu önceliklendirme ve kriter miktarı, programın başında kullanıcı tarafından ayarlanabilmesi önemlidir. Yapılabilecek önceliklendirmeye örnek olarak şöyle bir sıralama yapılabilir:

([öncelik]. [kriter] – [genel fiyata etkisi]) → Kriterler Dataset İyileştirmelerine uygun alınmıştır.

1. İl & ilçe - %15
2. Lot Area (m^2) - %15
3. Muhit tahlili - %10
4. Neighborhood & Street - %10
5. Built Year - %10
6. Site içinde olup olmaması - %10
7. Overall Qual - %5
8. Overall Cond - %5
9. Oda Sayısı - %5
10. Baktığı cehpe - %5
11. Isıtma sistemi - %5
12. Kitchen Qual - %5

Mesela en pahalı ev ya da evler 1.000.000 TL/dolar olarak verilirse her kriterin en yüksek değeri girilen en yüksek fiyatla önceden tanımlanmış genel fiyatına etki yüzdeleri ile çarpılarak oluşturulur. Bu durumda Lot Area kriteri için puanlama yapılırken en yüksek Lot Area değeri (100.000 TL/dolar) bulunur. Fiyat tahmin yapılması istenen her evin alan değeri en büyük Lot Area değeriyle oranlanarak 100.000 TL/dolar ile çarpılır. Overall Cond ve Overall Qual kriterleri 10 üstünden derecelendirmeler yapıldığı için sahip olunan değer 10'a bölünüp 50.000 ile çarpılır. Kitchen Qual ise 4 değere sahip. Bu durumda 50.000 TL/dolar; 0.25, 0.5, 0.75 ya da 1 ile çarpılır. Bu şeklide ortaya çıkan tüm puanların toplanması sonucunda tahmini ev fiyatlarına erişebilmemiz mümkündür.

Dataset İyileştirmeleri

Hem bu yeni (üçüncü) metot hem de diğer metotların performansını arttırabilmek için evlerin hangi ilde ve ülkede bulunduğu (lokasyon bilgilerinin daha da açık olması), cepheye baktığı, site içinde olup olmadığı, kaçıncı katta olduğu, ekspertiz yılı, daha önce yenilenmişse yenilenme yılı, evlerin kaç oda kaç salon olduğu, ısıtma sistemi ve bulunduğu muhitteki en yakın markete, alışveriş merkezine, ulaşım noktalarına uzaklığı gibi evler hakkındaki kriter / bilgi miktarı arttırılmalıdır.

Kullanıcı Tiplerine Özel İyileştirmeler

Emlakçılar & Konut Eksperleri

Evlerin kriter sayısı yanında hedef müşteri kitlesi de fiyat tahmini konusunda büyük bir rol oynamaktadır. Kişilerin gelir miktarlarının aşağı yukarı tahmini sayesinde ya da direkt alınması yoluyla potansiyel ev sahibi ile uygun ev eşleştirilebilir. Bu durum, aynı zamanda satış miktarını oldukça olumlu bir biçimde etkileyecektir. Bu tip bir değişkenin işin içine katılması demek potansiyel müşterilerin fiyat konusunda bir çıta yukarıya esnemek istemesi yani kredi çekmesi durumunun da sınırlarının araştırılmasına imkân verir. Bir kişinin kredi puanının ilgili gerekli kuruluşlardan bilgi alınarak ve geliri, üstüne kayıtlı gayrimenkullerin sayısı, değeri, durumu, ailesi olup olmaması gibi etkenlerin analizinin yapılması satıcı tarafındaki riskin azaltılmasına inanılmaz faydası olur.

Ev Almak & Yatırım Yapmak İsteyenler

Potansiyel olsun olmasın her müşterinin gelir vs. gibi kısıtlamaları olsa da satın almak istediği ev ile ilgili belli istekleri de bulunmaktadır. Bu istekleri de karşılamak amacıyla bir filtreleme fonksiyonu yazılabilir. Bu fonksiyon sayesinde kullanıcı tahmini ev fiyatları arasından isteğine uyan evleri listeyebilecektir.

Bu hizmet, emlakçı ya da konut bilirkişileri tarafından ev almak isteyenlere bir web sitesi üzerinden ulaştırılabilirse hem önceliklendirme yüzdeleri ve kriterler piyasaya göre güncel tutulabilir hem de kullanıcıların fiyatta uzlaşabileceği ya da belirli konularda danışabileceği bir mercii de olmuş olacaktır.