Popis

Cílem projektu je vytvoření aplikace umožňující doporučovat uživateli produkty nad zvolenou databází produktů. Uživatel bude s aplikací interagovat pomocí webového rozhraní, ve kterém si zvolí dvojici atributu, na základě kterých aplikace nalezne zajímavé produkty. U každého atributu půjde zvolit jestli je pro uživatele důležitější nižší nebo vyšší hodnota. Následně aplikace pomocí operátoru skyline nalezne produkty, které dominují ostatní v dané dvojici atributů.

Výstup bude realizován jako 2D graf, kde budou jednotlivé produkty reprezentovány jako body a výsledná skyline jako křivka spojující jednotlivé dominující produkty. Ostatní body budou pod nebo nad skyline křivkou, podle toho, kde na osách leží optimum.

Způsob řešení

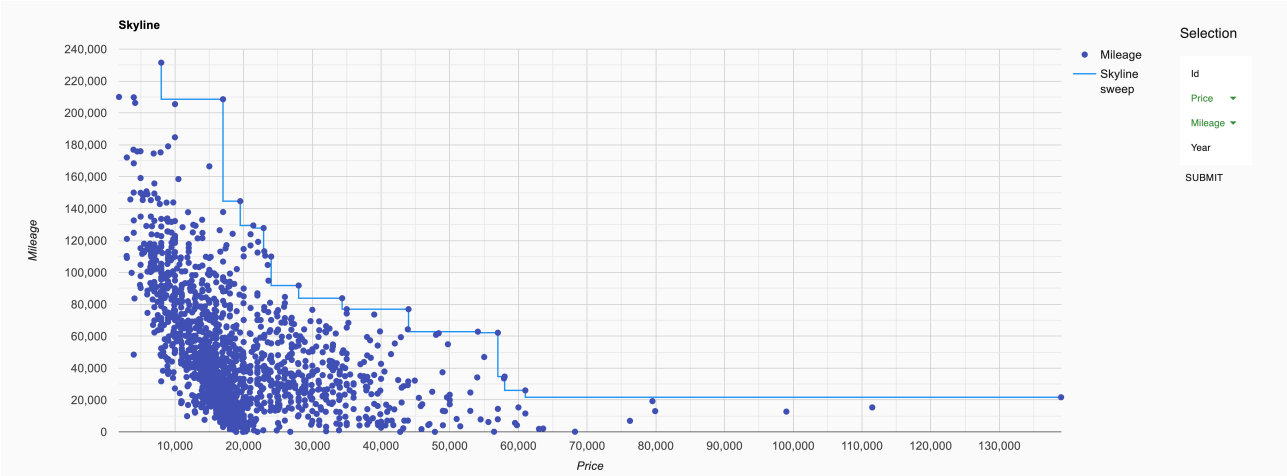
Pro implementaci jsem se rozhodl pro dvě řešení. První jako Desktop aplikaci napsanou v Pythonu a knihovnou PySide. Druhé jako webovou aplikaci v jazyce Javascript s knihovnou React. Dvě řešení proto, že jsem chtěl porovnat odlišnosti dvou jazyků a jejich vliv na samotnou implementaci jednotlivých algoritmů.

Pro výpočet skyline byly implementovány 3 algoritmy: “Brute force”, “Divide and conquer” a jako nejpokročilejší “Sweep plane”.

Řešení desktopové aplikace je díky použití pythonu cross-platform. Prvotně jsem využil pro načtení dat knihovnu pandas, ale to se ukázalo jako zbytečný over-head, protože jsem nakonec nevyužil žádné optimalizované metody, které pandas nabízí a přístup po jednotlivých řádcích (metoda iterrows) byl mnohonásobně pomalejší než nativní python List. Překvapení pro mne bylo, že ačkoliv PySide je rozhraní pro QT knihovnu v C++, tak rychlost aplikaci již při 2000 prvků byla velmi nízka a pouhý re-render grafu trvá řádově vteřiny.

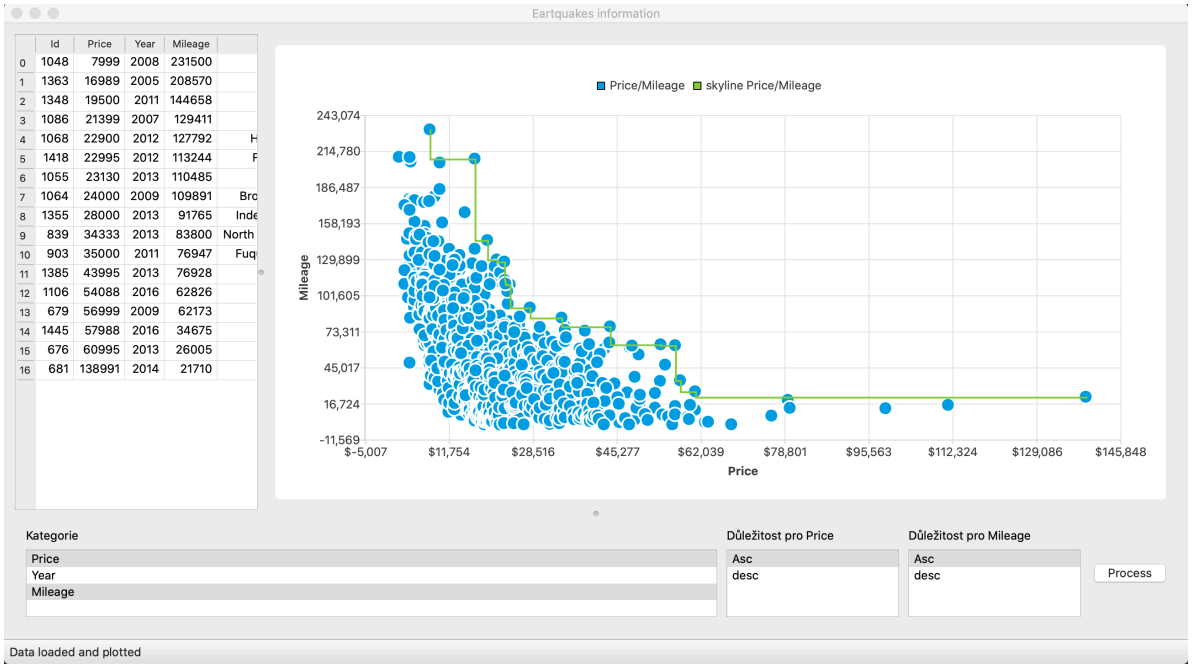
Pro řešení webové aplikace jsem využil knihovnu React a material-ui (nastylované komponenty pro React). Pro vývoj využívám standartní stack: webpack (jako builder) + babel (jako transpiler). V Javascriptu je celkem zajímavé, že nemá definovaný standartní řadící algoritmus a tedy implementace se velmi liší prohlížeč od prohlížeče. Proto jsem využil modul “timsort” s implementací merge-sortu se složitostí O(nlogn). Samotné rozhraní se skládá ze tří částí. První je 2D graf se zobrazenými body a skyline křivkou. Vpravo od grafu je možnost výběru atributů a jejich důležitosti. A ve spodní částí je tabulka se všemi prvky - zde jsem pro optimální výkon využil modul “react-virtualized”, který bez problému zvládne v prohlížeči pracovat s tabulkou o deseti tisících řádcích.

Příklad výstupu



Id	Price	Mileage	Year	City	State	Vin	Make	Model
1	16472	18681	2015	Jefferson City	MO	KL4CJBSBXFB267643	Buick	EncoreConvenie
2	15749	27592	2015	Highland	IN	KL4CJASB5FB245057	Buick	EncoreFWD
3	16998	13650	2015	Boone	NC	KL4CJCSB0FB264921	Buick	EncoreLeather
4	15777	25195	2015	New Orleans	LA	KL4CJASB4FB217542	Buick	EncoreFWD
5	16784	22800	2015	Las Vegas	NV	KL4CJBSB3FB166881	Buick	EncoreConvenie
6	17020	16877	2016	Grand Island	NY	KL4CJASB7GB536760	Buick	EncoreFWD
7	15950	27885	2015	West Covina	CA	KL4CJASB3FB241802	Buick	EncoreFWD

Jako vstup jsem využil staženou databázi (odkaz) o prodeji automobilů. Každý záznam obsahuje atributy: Id, cenu, rok výroby, počet najetých kilometrů, město prodeje, stát, název výrobce a model auta. Ukázka výstupu zobrazuje čistě spuštěnou aplikaci s dvěma tisíci auty a vypočtenou skyline křivku pro cenu/počet najetých kilometrů, kde důležitost je kladena na vysokou cenu a vysoký počet najetých kilometrů. V sekci “Selection” lze měnit výběr atributů a pomocí malé šipku u atributu i jeho důležitost (vyšší/nižší má být lepší). Tabulku pod grafem lze procházet posunem a řadit dle jednotlivých atributů při kliknutí na daný nadpis sloupce.

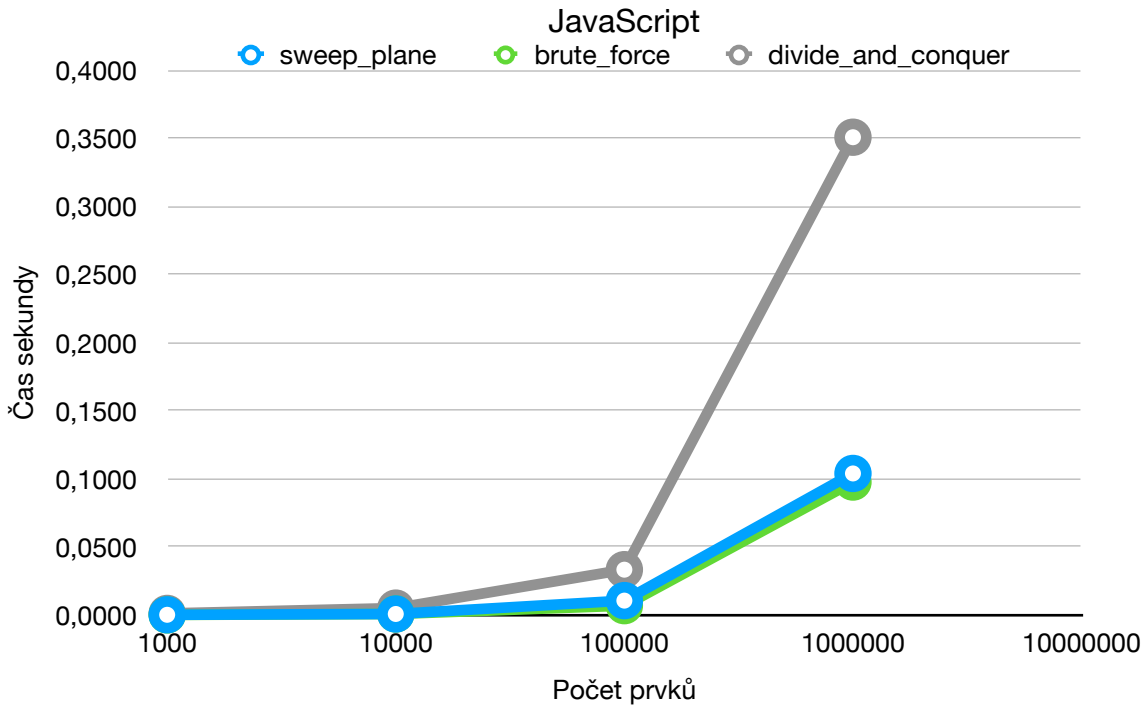


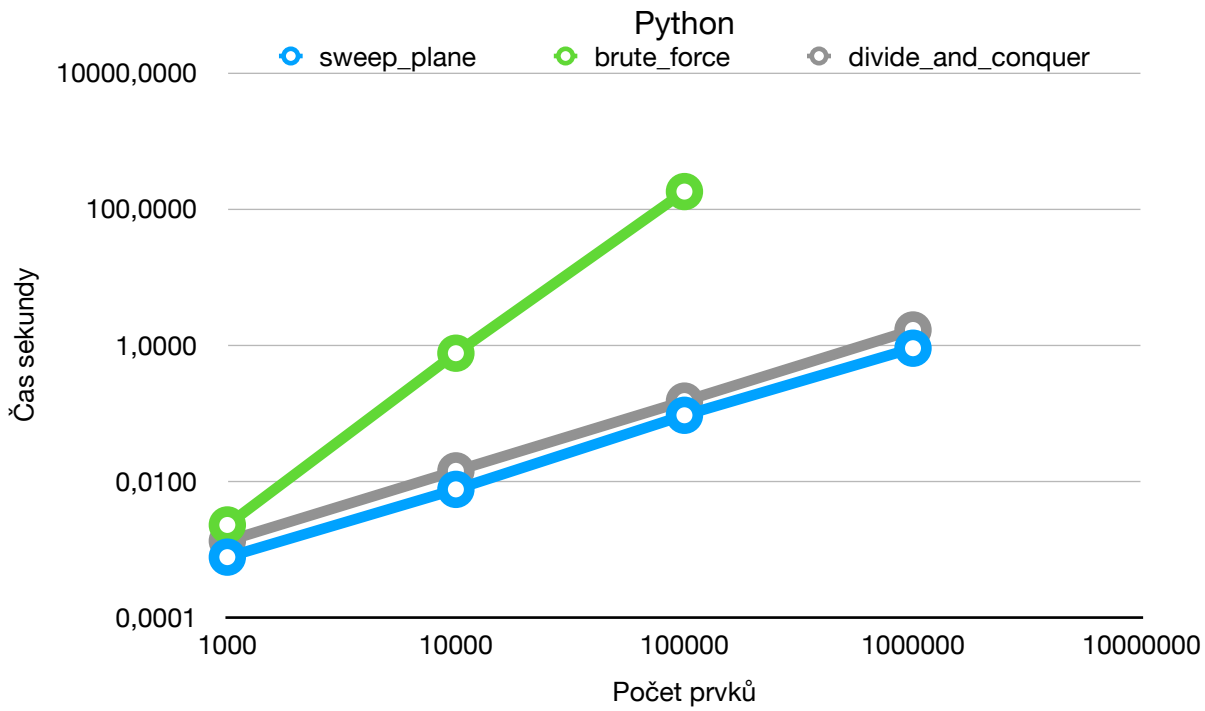
Experimentální sekce - zkoumání rychlosti a přesnosti

Implementovány jsou 3 algoritmy a všechny by měli spočítat stejný výsledek, protože jsou založeny na striktním porovnávání numerických hodnot. Nejpomalejší algoritmus je “Brute force” nebo-li výpočet hrubou silou. Klasický brute-force by měl složitost $O(n^2)$, já jsem ho trochu vylepšil, tak že jsem data nejprve seřadil a díky tomu se složitost snížila na O(n.log(n)+n.n/2).

Středně výkonný algoritmus je “Divide and conquer”, který je rychlejší než brute-force, ale zase je náročnější na paměť, kvůli rekurzivnímu dělení dat na poloviny (vždy se vytvoří kopie). Jeho složitost je podle Master theoremu O(n log n).

Jako nejlepší řešení jsem implementoval “Sweep plane” algoritmus, který má stejnou složitost jako “Divide and conquer” O(n log n), ale nižší paměťovou náročnost.





Diskuze - rozbor nedostatků a způsob řešení

Naměřené časové údaje v případě Pythonu odpovídají předpokládané složitosti, kde u “Brute force” již při deseti tisíci prvků je čas přes 100 sekund. Ovšem u Javascript je výsledek úplně jiný, než se předpokládalo. “Brute force” je podle časové náročnosti téměř identický s algoritmem “Sweep plane”, přestože jejich odhad složitosti byl O(n.log(n)+n.n/2) a O(n log n). Vzhledem k téměř identické implementaci algoritmů v obou jazycích, se jako nejpravděpodobnější příčina výsledného rozdílu jeví v Javascript enginu V8 funkce JIT (Just in time compilation). Oba jazyky jsou sice interpretované, ale Javascript díky JIT dokáže za běhu vysoce optimalizovat kód, který se spouští opakovaně, což je právě případ “Brute force” řešení. JIT za běhu programu kompiluje bytecode na vysoce efektivní strojový kód, který způsobil v našem případě drastické zrychlení oproti python interpreteru, který tuto funkci nemá.

// TODO - kouknout do prezentace a rozebrat nedostatky - asi ve více dimenzich…

Závěr

Cílem projektu bylo naimplementovat aplikaci s uživatelským rozhraním, která mu bude doporučovat produkty z databáze produktů. Toto doporučení je realizováno pomocí 2D grafu se zobrazenou skyline křivkou. Výpočet skyline je řešení pomocí tří různých algoritmů: “Brute force”, “Divide and conquer” a “Sweep plane” u kterých jsem porovnával rychlost v závislosti na velikosti vstupu. Pro zajímavější porovnání jsem aplikaci implementovat jako webovou stránku v jazyce Javascript a potom jako desktopovou aplikaci v jazyce Python. U řešení v Javascript jsem objevil velice zajímavé chování v případě “Brute force” algoritmu, který vůči předpokladům se ukázal stejně rychlý jako “Sweep plane”, který by měl být mnohem výkonnější. Projevila se zde odlišnost ve výkonosti různých programovacích jazyků v různých případech.