Programming Assignment Artificial Intelligence

Wiktor Biesiadecki

551458 – Artificial Intelligence

Dr Koorosh Aslansefat

University of Hull

07/01/2026

Objective

Your goal is to devise a strategy that considers all these features to estimate a traversal cost for each cell of the grid and use this estimate to determine the most efficient and cost-effective delivery routes.

Task 1

Task 1.1 - Why is this an important problem to solve?

Delivery efficiency is an important topic that can be overlooked. According to Wikipedia, distribution corresponds to 16.4% of total GVA in United Kingdom. (Wikipedia, 2019) Seemingly it doesn't affect us, as we are used to just click order now button and we are expecting our delivery to show up at our door in timely manner. Reality is with growing reliance on online shopping and food deliveries this industry is affecting us more each day.

Let's consider some of the key factors. First Amazon alone (Butler, 2022) delivers 2 million parcels a day in UK alone. Each driver delivers between 250 – 300 parcels each day and that results in approximately 7000 delivery vans driving each day on UK roads just to deliver parcels from this one vendor. This number shows how important optimisation could be. Without proper approach company can lose valuable revenue delivering less goods to customers, who expect their orders to arrive as soon as possible. Proper planning can mitigate entering congested areas in rush hours, delivering parcels in hours when most customers are at home to avoid reattempting deliveries. Optimising for more efficient routes with shorter travel time from one address to another. Optimisation thus reducing cost of delivery will directly benefit companies, it might as well benefit customers if companies choose to share that benefit with them. Recently new delivery methods worth consideration are emerging like use of autonomous vehicles both ground and air as well as delivering to lockers and stores.

The big factor here is sustainability. Most of the vans in fleet are diesel with smaller percentage being electric. When diesel vehicles add up to pollution in populated areas while being powered by not renewable source of energy, electric vehicles require considerable amounts of rare elements procured from all over the world to be produced. (RAC, 2021) Using electricity to power them has a lot of advantages but this energy comes at a cost. According to International Energy Agency in 2024 almost half energy production in UK came from fossil fuels and other "not clean" sources like burning waste. (IEA, 2024) Both types have certain lifespan after which they will have to be replaced for newer models and scrapped.

As we can see from this brief introduction this problem ranges from convenience, increased traffic, pollution, customer satisfaction all the way to government regulations and working conditions in remote lithium mine halfway across the world where people work for $3+ per day. (Zheng, 2023)

Task 1.2 - Can AI help solve this problem, and why?

While AI isn't and probably in the near future won't be able to remedy all problems discussed above, it can definitely help in some areas, especially optimizing and increasing efficiency of certain tasks and processes like problems described in this assignment based on available historical data. Processing large quantities of information, finding patterns, predicting outcomes and supporting us in decision making process based on tangible results is where AI truly stands out, surpassing human abilities. Making use of datasets we can identify bottlenecks, patterns and areas for possible improvement. Using this tool, we can easily simulate different scenarios to predict how changes would affect real life performance. We can expose algorithms to various parameters and variables to better understand how they will affect outcome, and which ones are most influential on end result. Advancements in computer hardware allow us to use machine learning in increasingly complex data, making any predictions more accurate, while being able to obtain results faster.

Combining historical data that model is already trained on with real-time information can help optimize decision making process even further giving us the best solution for current situation. This approach applied to delivery efficiency can bring noticeable improvements. In dynamically changing environment and monitoring real time parameters routes of delivery vehicles can be adjusted to avoid congestion. Sequence of deliveries can be altered, for example in case of traffic accidents, visiting other areas first and coming back when obstructions are cleared. Knowing weather forecasts and historical data we would be able to make predictions if, for example, sending more couriers with less parcels would be solution to finish deliveries in timely manner. In 2019 28% of e-commerce purchases were repeat customers (Metrilo Blog, 2019) and seeing increased growth of e-commerce after pandemic purchases it's safe to assume this number only grew (International Trade Administration, 2024). Collecting data from deliveries (for example delivery success ratio, time of delivery, time spent at the address) to this 28% we could determine the best conditions for deliveries to these addresses.

Task 1.3 - Is this problem best viewed as a prediction task, a search task, or a combination of both?

This task is best viewed as combination of both:

- devise a strategy that considers all these features to estimate a traversal cost for each cell of the grid – **prediction task - regression**
- use this estimate to determine the most efficient and cost-effective delivery routes – **search task**

Task 1.4 - Justify your answer with clear reasoning and provide examples of suitable AI techniques. If you treat it as a prediction problem, is it regression or classification?

The main objective later separated into tasks consists of two main parts.

- devise a strategy that considers all these features to estimate a traversal cost for each cell of the grid
- use this estimate to determine the most efficient and cost-effective delivery routes

Let's break down first part of the objective. My goal is to estimate the cost of each cell based on provided data. This means we will use machine learning to solve this problem. Below description from IBM confirms this assumption.

"Predictive artificial intelligence (AI) involves using statistical analysis and machine learning (ML) to identify patterns, anticipate behaviours and forecast upcoming events. Organizations use predictive AI to predict potential future outcomes, causation, risk exposure and more." (Mucci, 2024)

When examining provided dataset we can clearly notice that we are working with data that's clearly labelled which in conclusion tells us we can use supervised learning. Because our dataset is numerical it needs to be treated as regression task.

In chapter 7 3 "*Artificial intelligence: foundations of computational agents"* we can find below examples

"Figure 7.1 […] shows training examples typical of a classification task. The aim is to predict whether a person reads an article posted to a threaded discussion website given properties of the article."

"Figure 7.2 […] shows some data for a regression task, where the aim is to predict the value of feature Y on examples for which the value of feature X is provided. This is a regression task because Y is a real valued feature."

Example	Author	Thread	Length	Where_read	User_action
e_1	known	new	long	home	skips
e_2	unknown	new	short	work	reads
e_3	unknown	followup	long	work	skips
e_4	known	followup	long	home	skips
e_5	known	new	short	home	reads
e_6	known	followup	long	work	skips
e_7	unknown	followup	short	work	skips
e_8	unknown	new	short	work	reads
e_9	known	followup	long	home	skips
e_{10}	known	new	long	work	skips
e_{11}	unknown	followup	short	home	skips
e_{12}	known	new	long	work	skips
e_{13}	known	followup	short	home	reads
e_{14}	known	new	short	work	reads
e_{15}	known	new	short	home	reads
e_{16}	known	followup	short	work	reads
e_{17}	known	new	short	home	reads
e_{18}	unknown	new	short	work	reads
e_{19}	unknown	new	long	work	?
e_{20}	unknown	followup	short	home	?

Figure 7.1: Example data of a user's behavior. These are fictitious examples obtained from observing a user deciding whether to read articles posted to a threaded discussion website depending on whether the author is known or not, whether the article started a new thread or was a follow-up, the length of the article, and whether it is read at home or at work. $e_1, \ldots, e_{18}$ are the training examples. The aim is to make a prediction for the user action on e_{19}, e_{20}, and other, currently unseen, examples

Example	X	Y
e_1	0.7	1.7
e_2	1.1	2.4
e_3	1.3	2.5
e_4	1.9	1.7
e_5	2.6	2.1
e_6	3.1	2.3
e_7	3.9	7
e_8	2.9	?
e_9	5.0	?

Figure 7.2: Examples for a toy regression task

(Poole and Mackworth, 2010)

These definitions clearly show differences between regression and classification and affirms choice of regression for this example.

elevation	avg_traffic_speed	pollution_level	population_density	road_quality_index	proximity_main_roads	weather_condition_index	type_of_terrain	zone_classification	time_of_day	traversal_cost
37.45401	72.99983	89.6736	3707.776016	8.36348007	580.7790434	2.167785749	Park	Commercial	Morning	569.1064
95.07143	18.4512	28.4453	4405.509375	2.30742786	526.9716496	5.857533154	Park	Residential	Afternoon	378.2459
73.19939	34.66397	37.9078	2315.899377	9.51817416	351.0369496	6.538476947	Residential Are	Residential	Morning	534.4027
59.86585	66.32806	54.2013	1445.893658	8.58901704	493.2126579	5.56934744	Road	Residential	Evening	826.2173

Example data from dataset showing similarities to regression example.

Now I need to examine second part of the objective. After estimating the cost of each cell, I need to use this information to find optimal route.

At the beginning of chapter 3 "*Artificial intelligence: foundations of computational agents"* reads:

"Finding the best route from a current location to a destination is a search problem. The state includes mode of transportation (e.g., on foot or on a bus), the location of the traveller, and the direction of travel. A legal route will include the roads (going the correct way down one-way streets) and intersections the traveller will traverse. The best route could mean

- the shortest (least distance) route
- the quickest route
- the route that uses the least energy
- the lowest-cost route, where the cost takes into account time, money (e.g., fuel and tolls), and the route's attractiveness." (Poole and Mackworth, 2010)

This explanation perfectly demonstrates that second part of the objective is clearly a search task. I will be looking for quickest and shortest routes characterized by lowest cost in pursue of optimisation. To achieve this informed search would be ideal choice as uninformed search doesn't take node weight into consideration. It will find shortest path, but not the most optimal path.

Task 2

2.1 Data Preprocessing

To make sure my model predicts desired target value, input values need to be in correct format and all missing values need to be handled, all non-numerical values need to be appropriately encoded, and finally prepared dataset needs to be split into test set and training set.

2.1.1 Non-Numerical Features

First step in data preprocessing is importing dataset using pandas, data analysis and manipulation library for python.

```
df = pd.read_csv('/kaggle/input/dataset/traversal_cost_data.csv')
```

When the document is loaded, preprocessing can continue. Using built in pandas' function. isnull() I can quickly check data set for missing values.

```
df.isnull().sum()
```

```
elevation                0
avg_traffic_speed        0
pollution_level          0
population_density       0
road_quality_index       0
proximity_main_roads     0
weather_condition_index  0
type_of_terrain          0
zone_classification      0
time_of_day              0
traversal_cost           0
dtype: int64
```

In this case it appears there's no missing data in any of the columns. Next step is to encode categorical data which means to convert categorical data to numerical.

In this situation One-Hot is better choice as there's no ordinal relation between categorical values in dataset.

Provided dataset has 3 columns with categorical values. We can identify them using .dtypes function. Categorical data is marked as "object".

type_of_terrain	zone_classification	time_of_day
Park	Commercial	Morning
Park	Residential	Afternoon
Residential Area	Residential	Morning
Road	Residential	Evening
Park	Residential	Morning
Residential Area	Industrial	Morning

```
print(df.dtypes)
```

```
elevation                float64
avg_traffic_speed        float64
pollution_level          float64
population_density       float64
road_quality_index       float64
proximity_main_roads     float64
weather_condition_index  float64
type_of_terrain           object
zone_classification       object
time_of_day               object
traversal_cost           float64
dtype: object
```

Each column has its own categories.

```
df["type_of_terrain"].unique()
```

```
array(['Park', 'Residential Area', 'Road', 'Commercial Area'],
      dtype=object)
```

```
df["zone_classification"].unique()
```

```
array(['Commercial', 'Residential', 'Industrial'], dtype=object)
```

```
df["time_of_day"].unique()
```

```
array(['Morning', 'Afternoon', 'Evening', 'Night'], dtype=object)
```

To easier encode this data, we can use scikit-learn library for python which has OneHotEncoder function built in.

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
categoricalColumns = ["type_of_terrain", "zone_classification", "time_of_day"]
oneHot = encoder.fit_transform(df[categoricalColumns])
```

```
featureNames = encoder.get_feature_names_out(categoricalColumns)
```

```
dfOneHot = pd.DataFrame(oneHot, columns=featureNames)
```

Encoding done on categorical columns only.

```
numericalColumns = ['elevation', 'avg_traffic_speed', 'pollution_level',
                    'population_density', 'road_quality_index',
                    'proximity_main_roads', 'weather_condition_index', 'traversal_cost']
```

```
dfProcessed = pd.concat([df[numericalColumns].reset_index(drop=True), dfOneHot], axis=1)
```

Both numerical and encoded categorical combined into one set again.

The result of encoding:

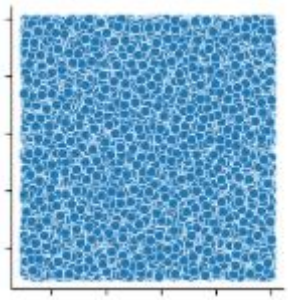
type_of_te	type_of_te	type_of_te	type_of_te	zone_clas	zone_clas	zone_clas	time_of_d	time_of_d	time_of_d	time_of_
0	1	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	1	1	0	0	0
0	0	1	0	0	0	1	0	0	1	0
0	0	0	1	0	0	1	0	1	0	0
0	1	0	0	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	1	0	1	0	0	0
0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	0	0	1	0	0	1	0

Using .dtype function again we can see that all categorical data was successfully encoded as numerical.

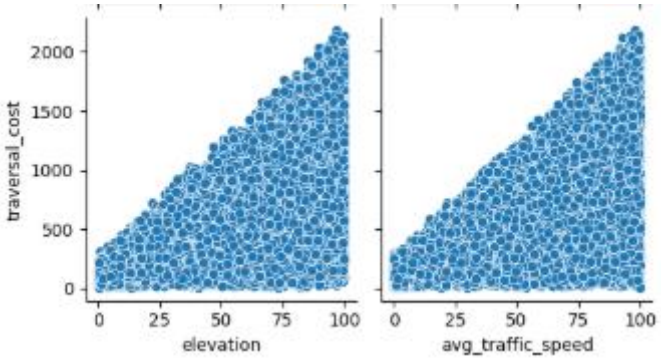
```
print(dfProcessed.dtypes)
```

```
elevation                           float64
avg_traffic_speed                   float64
pollution_level                     float64
population_density                  float64
road_quality_index                  float64
proximity_main_roads                float64
weather_condition_index             float64
traversal_cost                      float64
type_of_terrain_Commercial Area     float64
type_of_terrain_Park                float64
type_of_terrain_Residential Area    float64
type_of_terrain_Road                float64
zone_classification_Commercial      float64
zone_classification_Industrial      float64
zone_classification_Residential     float64
time_of_day_Afternoon               float64
time_of_day_Evening                 float64
time_of_day_Morning                 float64
time_of_day_Night                   float64
dtype: object
```

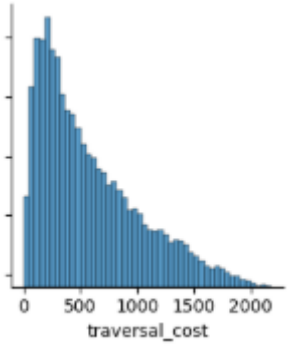
I used matplotlib and seaborn to visualize data and correlation between each column. In this case it did not bring any effect as points are distributed across the entire value range as there appears to be no obvious linear or nonlinear relationship.



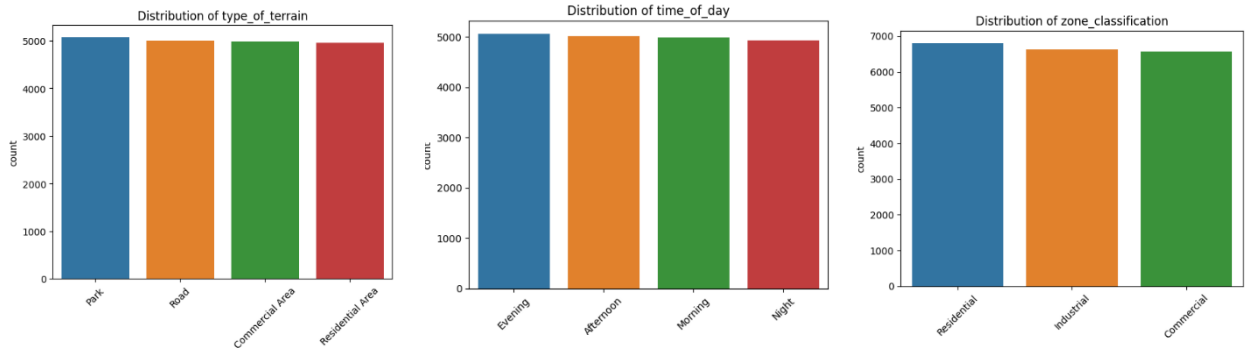
The only difference is correlation between travel cost and elevation, travel cost and average traffic speed. Plot implies that higher speed and elevation allow for higher travel cost, which especially in case of average traffic speed seems very counterintuitive suggesting congestion as preferred as no high travel cost inputs ended up in low average speed category.

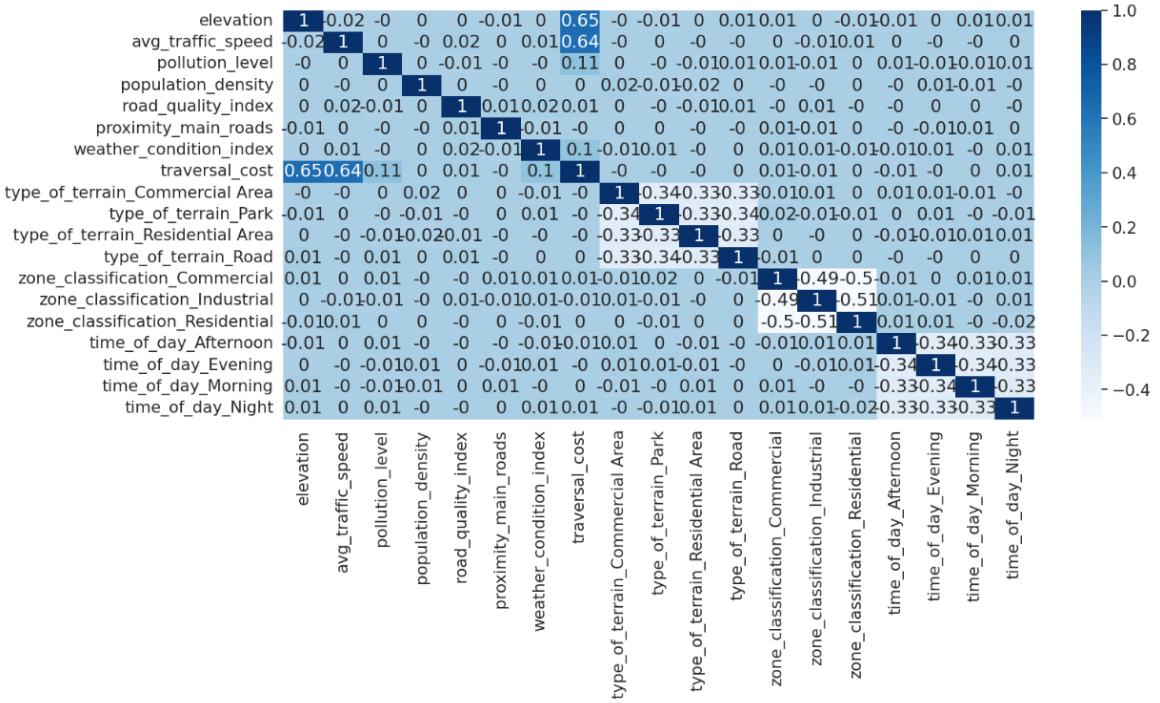


We can also notice that most of cells have travel cost in 0 -500 value with higher travel cost cells being more sparce.



Data in type of terrain, type of day and zone classification seems to be spread out evenly between each category.





Features corelation heatmap shows this very clearly as well. Visible correlation between traversal cost, elevation and average traffic speed. Negative correlation seen between categories in time of day, zone classification and type of terrain which is to be expected as they replace each other.

2.1.2 Data Splitting

Next step is to perform data splitting into training set and test set. I'll train the algorithms on 80% on available data and test it on remaining 20%.

```
from sklearn.model_selection import train_test_split

Y = dfProcessed['traversal_cost']
X = dfProcessed.drop('traversal_cost', axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

```
print(X_train.shape)
print(Y_train.shape)
print(X_test.shape)
print(Y_test.shape)
```

```
(16000, 18)
(16000,)
(4000, 18)
(4000,)
```

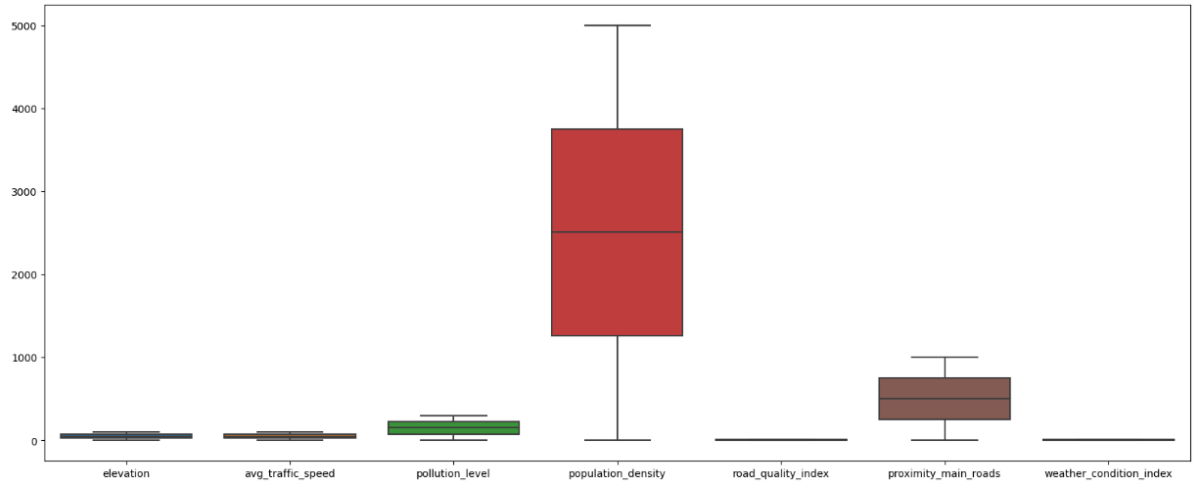
I used scikit-learn for that as well, it has a build in function train_test_split that takes four arguments:

- X – Input feature
- Y – target
- test_size – part of total data used as training set
- random_state – random seed

And returns training and test sets of desired size. We can see that from initial 20000 rows, training set has 16000 (80%) and test set has 4000 (20%).

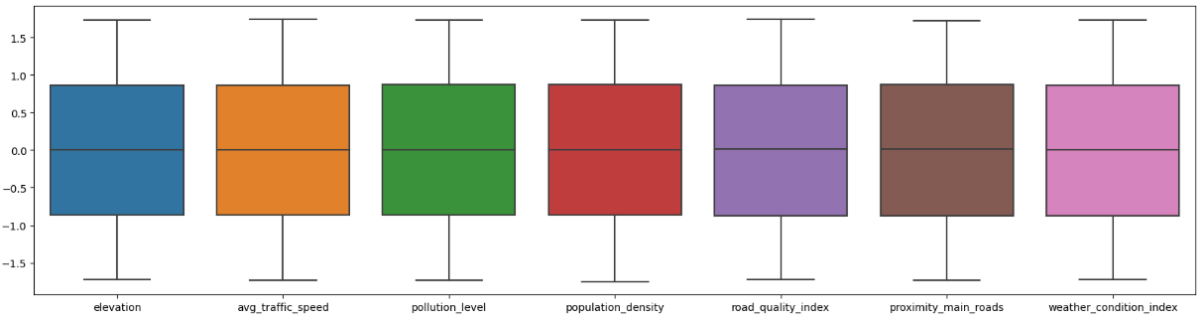
2.1.3 Data Scaling

Next important part of data preprocessing is scaling.



Looking at a graph, we can how disproportionate values of data set are. This is caused by comparing values of different scale like average traffic speed and population density. Data scaling is a process that bring all numeric data to a common scale without distorting their relations. Scaling is important for machine learning for several reasons:

- Some algorithms line neural networks and linear regression are sensitive to scale and perform better with scaled data
- Different scales can disproportionately influence model
- Common scale enables direct comparison



Result of data scaling. Data previously encoded omitted here as it already falls in required range. Target values are not scaled as most algorithms don't require that.

2.2 Model Implementation and Evaluation

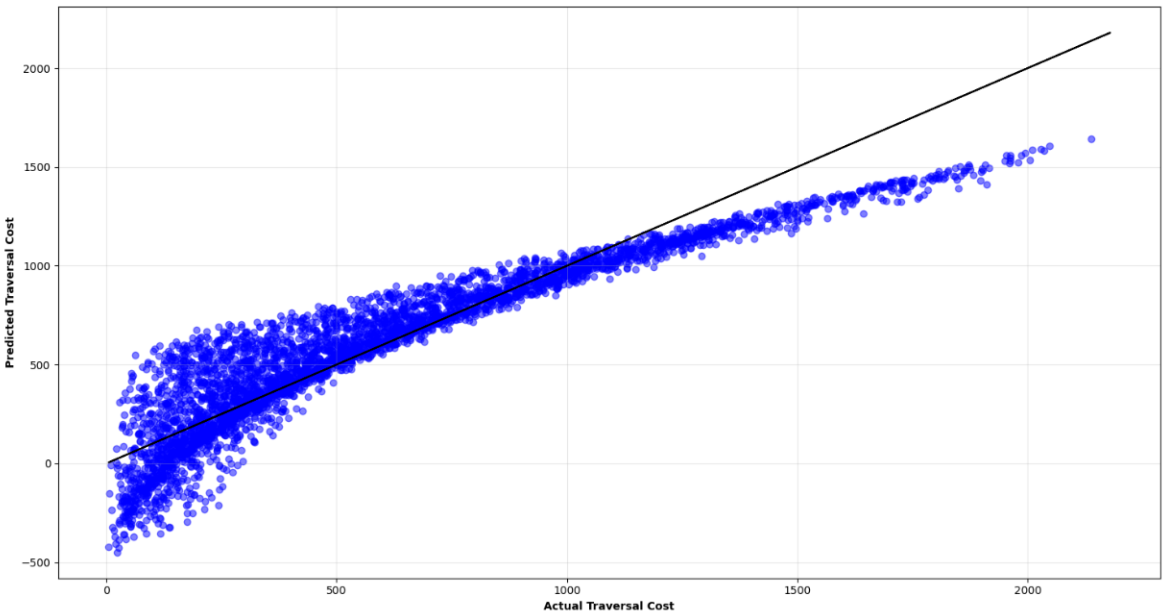
2.2.1 Linear Regression

Training

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

regressor.fit(X_train, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```



Evaluation

- Mean Absolute Error: 124.31001764922824
- Mean Squared Error: 27446.31014130236
- Root Mean Squared Error: 165.66927941324053

2.2.2 Polynomial Regression

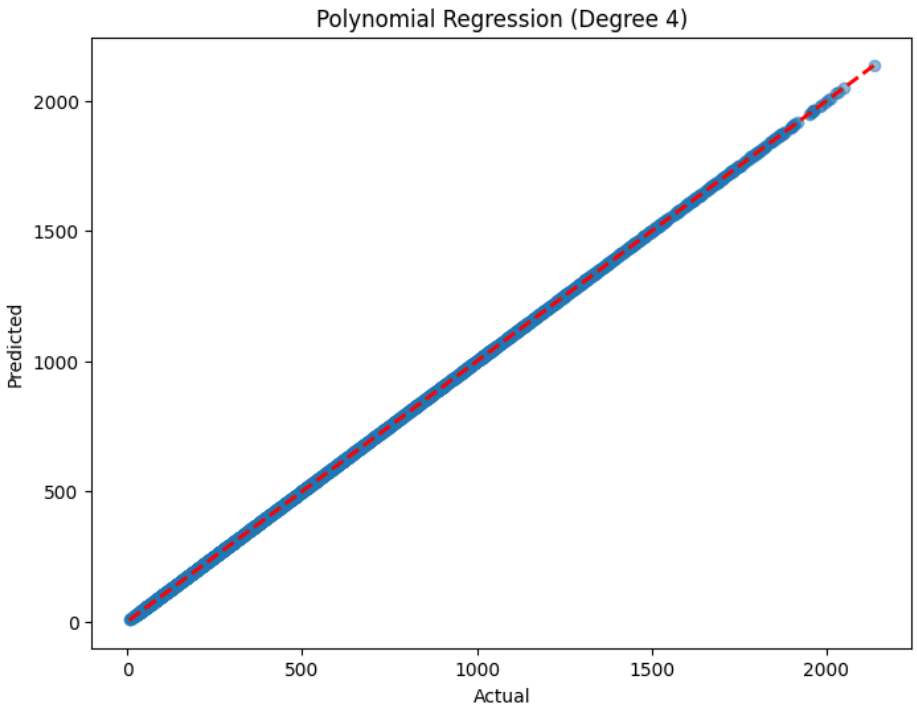
Training

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=4, include_bias=False)
X_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
```

```
regressor = LinearRegression()
regressor.fit(X_poly, Y_train)
print(regressor.intercept_)
print(regressor.coef_)
```

```
1099565377735.887
[-1.78034910e+10  1.76219619e+12 -1.22374742e+12 ...  0.00000000e+00
  0.00000000e+00 -4.53021296e+10]
```

```
y_pred_test_poly = regressor.predict(X_test_poly)
```



Evaluation

- Mean Absolute Error: 0.4422038717226541
- Mean Squared Error: 0.30972503726788614
- Root Mean Squared Error: 0.5565294576820585

2.2.3 Neural Network

Training

Sequential deep neural network model was implemented using Tensor Flow python library. Model I with some handpicked parameters was serving as a baseline to compare with tuned Model II.

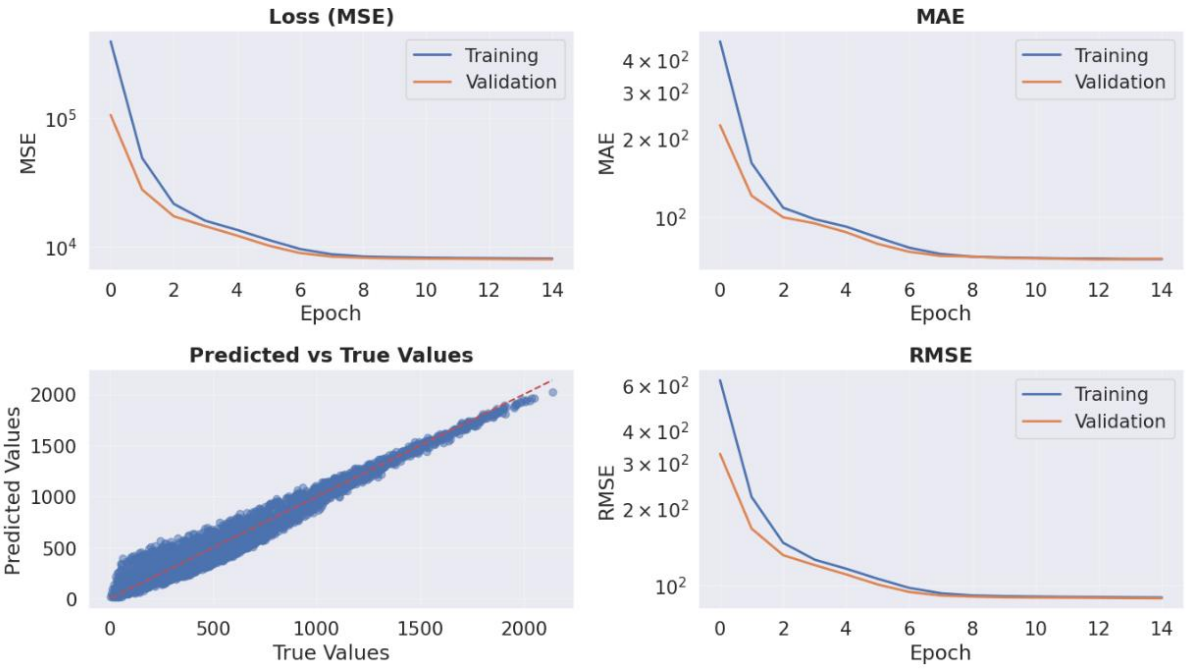
Model I

```
model = Sequential([
    Input(shape=(18,)),
    Dense(10, activation='relu'),
    Dense(10, activation='relu'),
    Dense(1)
])

optimizer = Adam(learning_rate=0.001)
model.compile(
    optimizer=optimizer,
    loss='mse',
    metrics=['mae']
)

model.summary()

hist = model.fit(
    X_train_nn, Y_train_nn,
    epochs=15,
    batch_size=20,
    validation_data=(X_val_nn, Y_val_nn),
    verbose=1
)
```



- Mean Absolute Error: 69.37408449883834
- Mean Squared Error: 7983.026326286996
- Root Mean Squared Error: 89.34778299592551

Model II

This model is modification of model I. Hyperparameter optimisation was done using Bayesian optimisation. Model was improved with additional layer of neurons, and 3 dropout layers.

Parameter range:

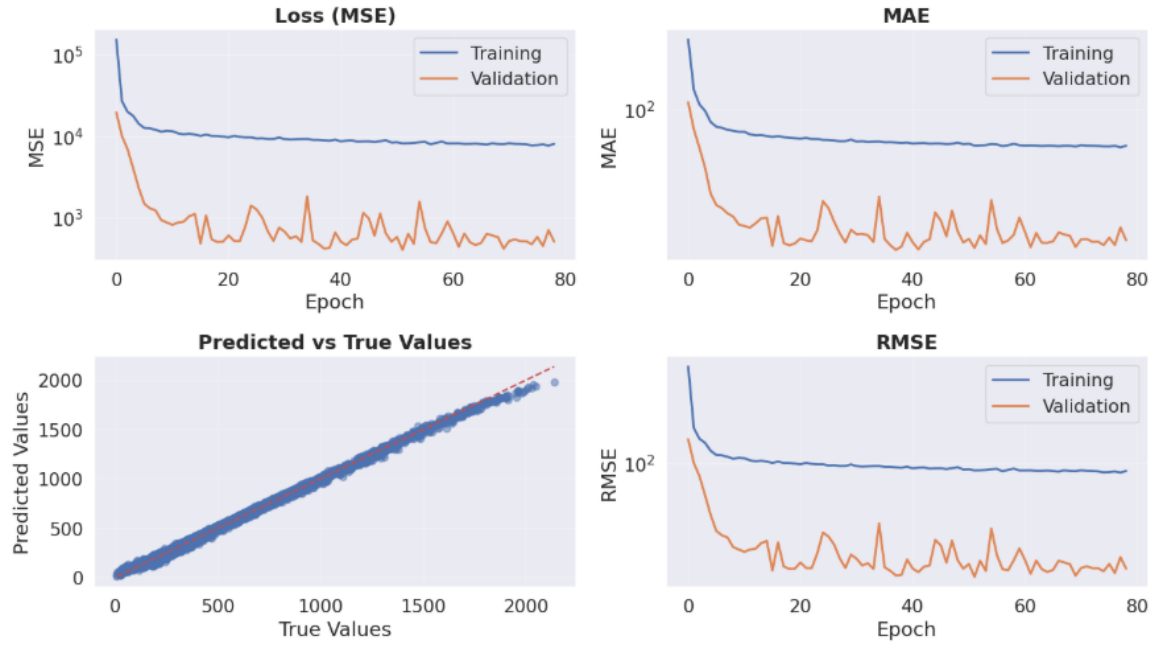
```
'neurons1': (32, 256),
'neurons2': (32, 256),
'neurons3': (32, 64),
'activation': (0, 4),
'learning_rate': (0.0001, 0.01),
'batch_size': (20, 50),
'epochs': (10, 100),
'dropout_rate': (0.1, 0.7),
```

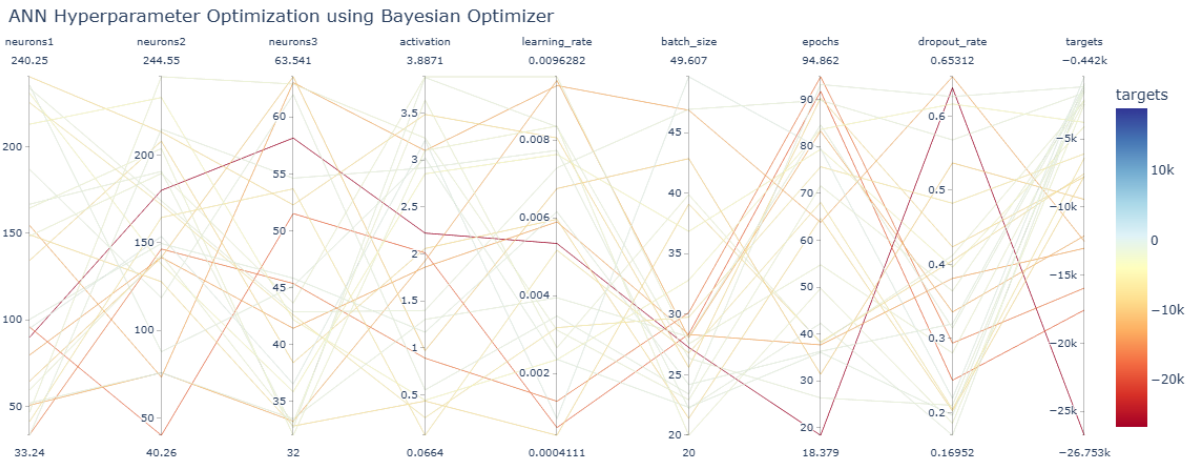
Optimisation:

iter	target	neurons1	neurons2	neurons3	activa...	learni...	batch_...	epochs	dropou...
1	-130484.9	115.89698	244.96000	55.423806	2.3946339	0.0016445	24.679835	15.227525	0.6197056
2	-4666.785	166.64976	190.60825	32.658703	3.8796394	0.0083411	26.370173	26.364247	0.2100427
3	-840.1311	100.15026	149.54544	45.822240	1.1649165	0.0061573	24.184815	36.293018	0.3198171
4	-6432.181	134.15967	207.87941	38.389561	2.0569377	0.0059649	21.393512	64.679036	0.2023144
5	-2868.820	46.571556	244.55036	62.900225	3.2335893	0.0031156	22.930163	71.580972	0.3640914
6	-1526.976	59.336564	142.91962	33.100432	3.6372816	0.0026619	39.875668	38.053996	0.4120408
7	-8801.403	154.46310	73.407398	63.026708	3.1005312	0.0094010	46.844820	63.810998	0.6531245
8	-1200.183	51.822320	75.900161	33.447273	1.3013213	0.0039479	28.140470	84.586375	0.3140519
9	-463.9158	94.929330	153.56392	36.509575	3.2087879	0.0008380	49.606608	79.502029	0.2192294
10	-1811.580	33.236954	214.66335	54.619435	2.9160286	0.0077355	22.221339	42.261915	0.1695214
11	-4598.536	225.33516	171.61878	42.588736	0.2542334	0.0031787	29.755499	75.664556	0.4825344
12	-4439.807	230.73565	137.77614	35.827015	2.8529791	0.0076317	36.838315	79.387046	0.3962773
13	-3231.944	149.09215	127.76918	32.813412	0.4315657	0.0004111	39.092312	38.292038	0.4051424
14	-698.5721	235.29489	87.841459	45.132253	3.0222045	0.0023651	22.309397	36.077630	0.1967327
15	-4653.644	240.25227	213.01896	52.268920	3.4858423	0.0080563	25.597101	90.330309	0.4236053
16	-4214.659	212.86659	232.72445	42.176111	0.4402076	0.0023565	32.813233	83.621328	0.6164383
17	-16144.27	33.557277	146.40739	45.357152	0.8884312	0.0012866	30.128455	94.861873	0.2939217
18	-2839.262	148.20909	189.47624	43.636147	3.8871283	0.0096282	27.553468	54.752365	0.2805269
19	-17618.19	95.804270	40.262676	51.506058	2.0107160	0.0006096	28.359393	91.743929	0.2437371
20	-13575.13	64.456451	141.63741	63.540814	0.9682210	0.0067541	42.848588	31.387378	0.5369298
21	-142359.9	114.38342	173.63650	52.272950	2.1430987	0.0009938	45.059074	38.870205	0.2119111
22	-12107.47	41.133631	164.36001	53.682059	0.0663513	0.0051697	26.794873	68.065551	0.2046198
23	-1171.725	186.77005	118.62871	61.975359	0.5500837	0.0034765	23.404205	93.222425	0.6264036
24	-26314.84	89.778924	179.83642	58.151110	2.2208032	0.0053435	27.255568	18.379249	0.6383294
25	-1021.807	233.69364	173.81472	42.848953	1.3968382	0.0072869	46.913307	89.837778	0.5679253
26	-17322.67	224.21185	155.45110	45.883656	1.9362772	0.0099215	39.843828	89.782535	0.5395206
27	-12260.78	81.036933	137.94904	37.096034	1.8178837	0.0062338	23.333147	40.847465	0.3689866
28	-4840.730	232.90159	172.78447	43.134456	2.5870502	0.0089642	46.412013	89.692494	0.5040521

Best parameters:

- 'neurons1': 94.9293301699733
- 'neurons2': 153.56392262744765
- 'neurons3': 36.509575199192405
- 'activation': 'selu'
- 'learning_rate': 0.0008380513724297313
- 'batch_size': 49.60660809801552
- 'epochs': 79.50202923669917
- 'dropout_rate': 0.21922940892050344



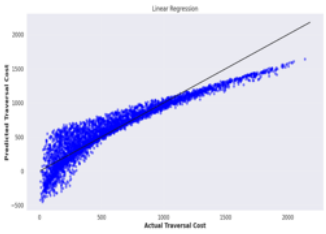
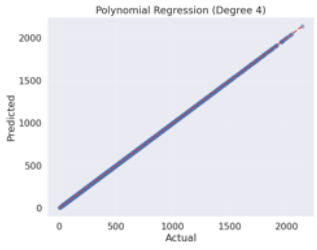
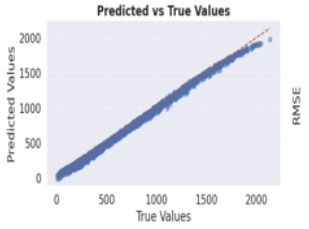


Evaluation

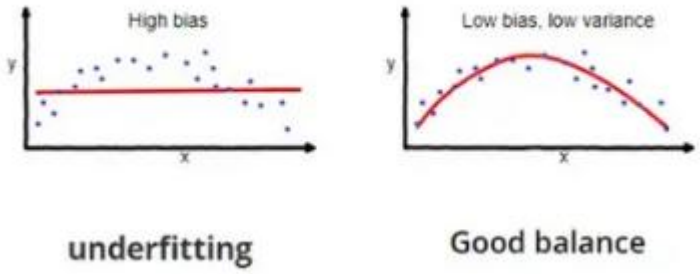
- Mean Absolute Error: 15.743532041503444
- Mean Squared Error: 432.8008487755675
- Root Mean Squared Error: 20.803866197790438

2.3 Analysis and Summary

Differences in models performance and complexity are clearly visible. Each model produced noticeably different output and performance metrics differ significantly as well.

Model / Metric	Linear Regression	Polynomial Regression	Neural Network
MAE	124.3100176	0.442203872	15.74353204
MSE	27446.31014	0.309725037	432.8008488
RMSE	165.6692794	0.556529458	20.8038662
Graph			

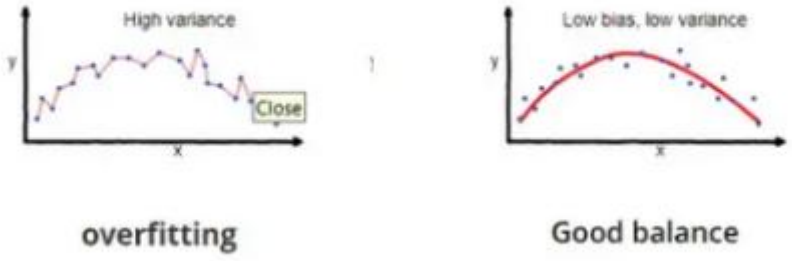
First tested model was linear regression. This model performed poorly. Predicted data is laid out in shape deviating from expected line. It's spread in conical shape, and some predictions had negative value which is unacceptable when predicting travel costs. In this case simple model was not capable to capture patterns in data causing underfitting. Linear regression assumes linear relationship between input and output, but data in provided in dataset is not linear as a result high bias and large prediction errors occurred.



(Rakesh, 2023)

Polynomial regression might look like the best choice, with predicted data exactly matching expected line and small errors. This is in fact misleading. Pretended perfect fit is in fact sign of underlying issues with how this model worked with provided dataset causing overfitting. This occurs when the model fits to random variations and patterns in the training

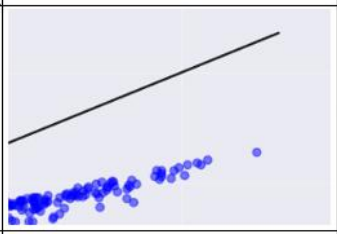
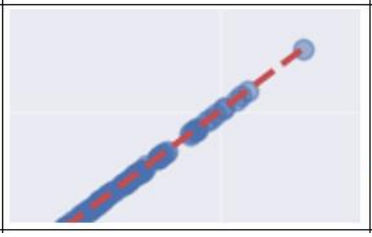
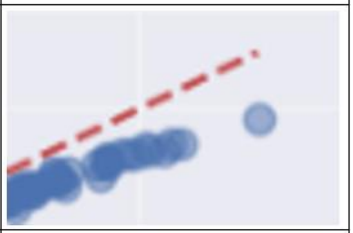
data instead of capturing the actual underlying structure. In effect it achieves high accuracy in training but fails on actual data.



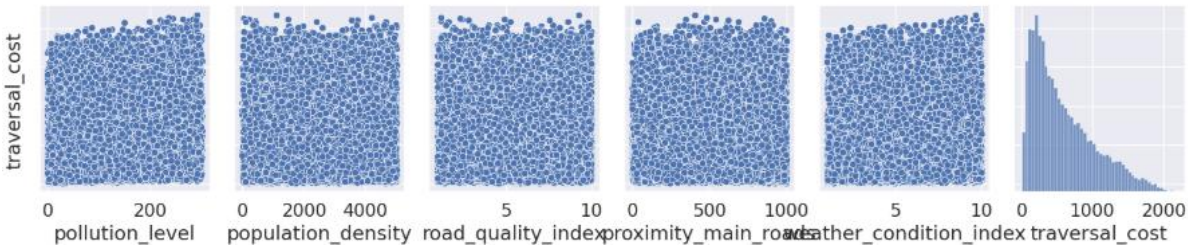
(Rakesh, 2023)

The best performing model in lineup was undoubtedly neural network. At cost of higher complexity and increased training time I was able to achieve good results. The success of neural network is hidden it's its complexity, where given proper hyperparameter tuning we can prevent issues troubling linear and polynomial regression. Hidden layers automatically learn complex patterns. Dropout layers preventing reliance on the same neurons preventing overfitting.

What all 3 models had in common is loss of fidelity in top range of outputs.

Model / Metric	Linear Regression	Polynomial Regression	Neural Network
Graph			

This was caused by discovered in exploratory data analysis unequal data distribution.



Majority of data is concentrated in lower ranges, giving less data in upper ranges to train model on.

Task 3

3.1 Load the Provided Grid Data

After uploading provided_grid.csv file Kaggle I had to load it to a variable.

```
grid = pd.read_csv('/kaggle/input/provided-grid/provided_grid.csv')
```

Before estimation of travel costs could be done, I had to perform the same data manipulation as was done on training set. That includes encoding ordinal features of time_of_day, zone_classification and type_of_terrain columns into numerical values using the same techniques and combining all data back into one dataset.

```
categoricalColumns = ["type_of_terrain", "zone_classification", "time_of_day"]
oneHot_grid = encoder.fit_transform(grid[categoricalColumns])
featureNames_grid = encoder.get_feature_names_out(categoricalColumns)
dfOneHot_grid = pd.DataFrame(oneHot_grid, columns=featureNames_grid)
```

```
numerical_columns = ['elevation', 'avg_traffic_speed', 'pollution_level',
                     'population_density', 'road_quality_index',
                     'proximity_main_roads', 'weather_condition_index']
grid_processed = pd.concat([grid[numerical_columns].reset_index(drop=True), dfOneHot_grid], axis=1)
```

As a last step I scaled numerical columns using the same scaler as I used for training set.

```
grid_scaled = grid_processed.copy()
grid_scaled[numerical_columns] = scaler.transform(grid_processed[numerical_columns])
```

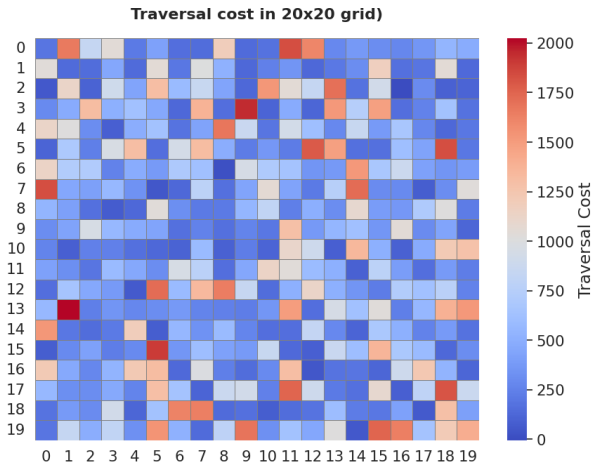
This time I didn't do dataset splitting as this action is only required in training phase. Now I'm using trained model on actual data.

3.2 Estimate Traversal Costs

Estimating travel costs on provided grid only required me to use trained previously model on provided grid data.

```
nn_predictions = model.predict(grid_scaled, verbose=0).flatten()
```

The achieved result can be seen below in form of a 20x20 grid. Heatmap applied on graph helps visualise travel cost of each cell. Warmer colours represent high travel cost and are easily distinguishable from cooler low-cost cells.



Saving grid as csv file.

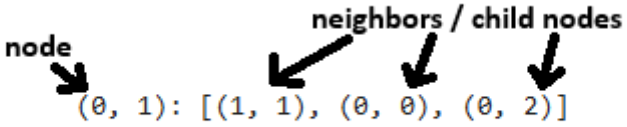
```
grid_save = pd.DataFrame(cost_grid,
                         columns=[f'Col_{i}' for i in range(grid_size)],
                         index=[f'Row_{i}' for i in range(grid_size)])

grid_save.to_csv('/kaggle/working/Estimated_grid.csv', index=False)
```

Next to find the most optimal path and most efficient searching algorithm for this task I've implemented and compared performance of four search algorithms:

- Depth First Search (DFS)
- Breadth First Search (BFS)
- Dijkstra's Algorithm
- A* Search Algorithm

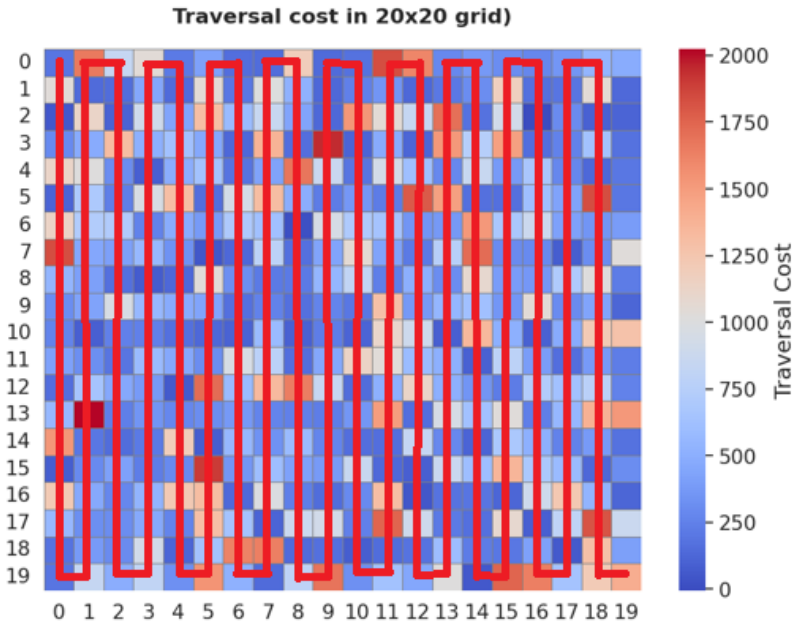
Before that could happen, traversal cost grid data had to be converted into graph to fit search algorithms.



Coordinates of the node are (row, column).

Depth First Search

Depth-First Search (DFS) is classic graph traversal algorithm. Due to the way it works it explores each path all the way to the end before backtracking instead of exploring neighbours adjacent to nodes along the way. This way algorithm will find a path, but necessarily shortest path. This issue exaggerated in this example as graph in form of a grid has more connections than a tree, where this algorithm finds its uses. Also being uninformed search algorithm, it visits nodes without consideration of weight.

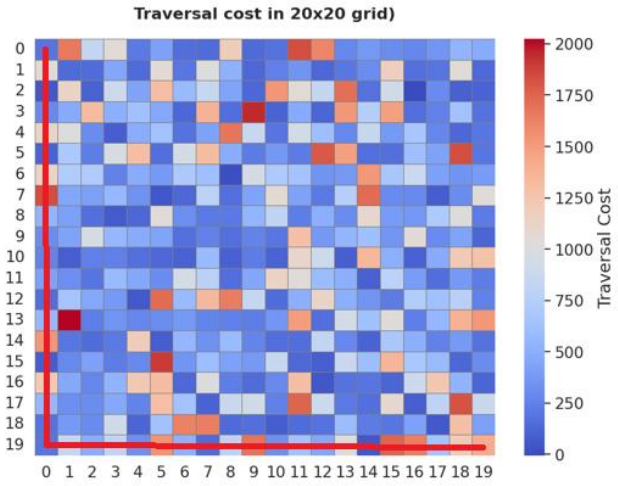


[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0), (19, 0), (19, 1), (18, 1), (17, 1), (16, 1), (15, 1), (14, 1), (13, 1), (12, 1), (11, 1), (10, 1), (9, 1), (8, 1), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1), (2, 1), (1, 1), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (5, 2), (6, 2), (7, 2), (8, 2), (9, 2), (10, 2), (11, 2), (12, 2), (13, 2), (14, 2), (15, 2), (16, 2), (17, 2), (18, 2), (19, 2), (19, 3), (18, 3), (17, 3), (16, 3), (15, 3), (14, 3), (13, 3), (12, 3), (11, 3), (10, 3), (9, 3), (8, 3), (7, 3), (6, 3), (5, 3), (4, 3), (3, 3), (2, 3), (1, 3), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4), (11, 4), (12, 4), (13, 4), (14, 4), (15, 4), (16, 4), (17, 4), (18, 4), (19, 4), (19, 5), (18, 5), (17, 5), (16, 5), (15, 5), (14, 5), (13, 5), (12, 5), (11, 5), (10, 5), (9, 5), (8, 5), (7, 5), (6, 5), (5, 5), (4, 5), (3, 5), (2, 5), (1, 5), (0, 5), (0, 6), (1, 6), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (7, 6), (8, 6), (9, 6), (10, 6), (11, 6), (12, 6), (13, 6), (14, 6), (15, 6), (16, 6), (17, 6), (18, 6), (19, 6), (19, 7), (18, 7), (17, 7), (16, 7), (15, 7), (14, 7), (13, 7), (12, 7), (11, 7), (10, 7), (9, 7), (8, 7), (7, 7), (6, 7), (5, 7), (4, 7), (3, 7), (2, 7), (1, 7), (0, 7), (0, 8), (1, 8), (2, 8), (3, 8), (4, 8), (5, 8), (6, 8), (7, 8), (8, 8), (9, 8), (10, 8), (11, 8), (12, 8), (13, 8), (14, 8), (15, 8), (16, 8), (17, 8), (18, 8), (19, 8), (19, 9), (18, 9), (17, 9), (16, 9), (15, 9), (14, 9), (13, 9), (12, 9), (11, 9), (10, 9), (9, 9), (8, 9), (7, 9), (6, 9), (5, 9), (4, 9), (3, 9), (2, 9), (1, 9), (0, 9), (0, 10), (1, 10), (2, 10), (3, 10), (4, 10), (5, 10), (6, 10), (7, 10), (8, 10), (9, 10), (10, 10), (11, 10), (12, 10), (13, 10), (14, 10), (15, 10), (16, 10), (17, 10), (18, 10), (19, 10), (19, 11), (18, 11), (17, 11), (16, 11), (15, 11), (14, 11), (13, 11), (12, 11), (11, 11), (10, 11), (9, 11), (8, 11), (7, 11), (6, 11), (5, 11), (4, 11), (3, 11), (2, 11), (1, 11), (0, 11), (0, 12), (1, 12), (2, 12), (3, 12), (4, 12), (5, 12), (6, 12), (7, 12), (8, 12), (9, 12), (10, 12), (11, 12), (12, 12), (13, 12), (14, 12), (15, 12), (16, 12), (17, 12), (18, 12), (19, 12), (19, 13), (18, 13), (17, 13), (16, 13), (15, 13), (14, 13), (13, 13), (12, 13), (11, 13), (10, 13), (9, 13), (8, 13), (7, 13), (6, 13), (5, 13), (4, 13), (3, 13), (2, 13), (1, 13), (0, 13), (0, 14), (1, 14), (2, 14), (3, 14), (4, 14), (5, 14), (6, 14), (7, 14), (8, 14), (9, 14), (10, 14), (11, 14), (12, 14), (13, 14), (14, 14), (15, 14), (16, 14), (17, 14), (18, 14), (19, 14), (19, 15), (18, 15), (17, 15), (16, 15), (15, 15), (14, 15), (13, 15), (12, 15), (11, 15), (10, 15), (9, 15), (8, 15), (7, 15), (6, 15), (5, 15), (4, 15), (3, 15), (2, 15), (1, 15), (0, 15), (0, 16), (1, 16), (2, 16), (3, 16), (4, 16), (5, 16), (6, 16), (7, 16), (8, 16), (9, 16), (10, 16), (11, 16), (12, 16), (13, 16), (14, 16), (15, 16), (16, 16), (17, 16), (18, 16), (19, 16), (19, 17), (18, 17), (17, 17), (16, 17), (15, 17), (14, 17), (13, 17), (12, 17), (11, 17), (10, 17), (9, 17), (8, 17), (7, 17), (6, 17), (5, 17), (4, 17), (3, 17), (2, 17), (1, 17), (0, 17), (0, 18), (1, 18), (2, 18), (3, 18), (4, 18), (5, 18), (6, 18), (7, 18), (8, 18), (9, 18), (10, 18), (11, 18), (12, 18), (13, 18), (14, 18), (15, 18), (16, 18), (17, 18), (18, 18), (19, 18), (19, 19)]

Path length - 381

Breadth First Search

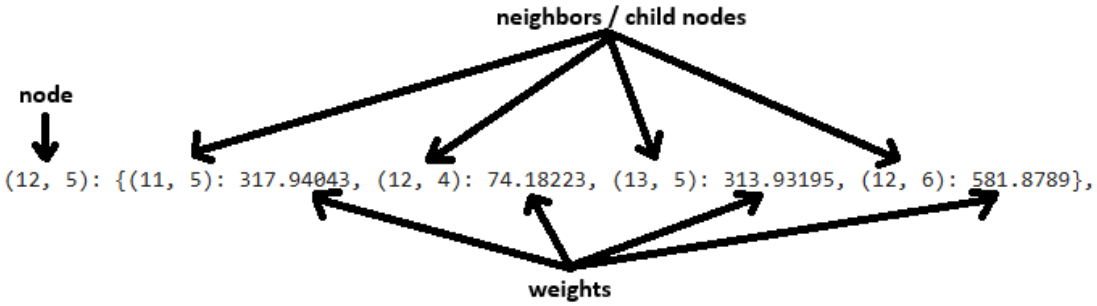
BFS like DFS is another example of uninformed search. Unlike DFS this algorithm doesn't go deep into each path. I it explores each path gradually visiting neighbours first. This way algorithm is meant to find the shortest path not just any path.



[(0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7, 0), (8, 0), (9, 0), (10, 0), (11, 0), (12, 0), (13, 0), (14, 0), (15, 0), (16, 0), (17, 0), (18, 0), (19, 0), (19, 1), (19, 2), (19, 3), (19, 4), (19, 5), (19, 6), (19, 7), (19, 8), (19, 9), (19, 10), (19, 11), (19, 12), (19, 13), (19, 14), (19, 15), (19, 16), (19, 17), (19, 18), (19, 19)]

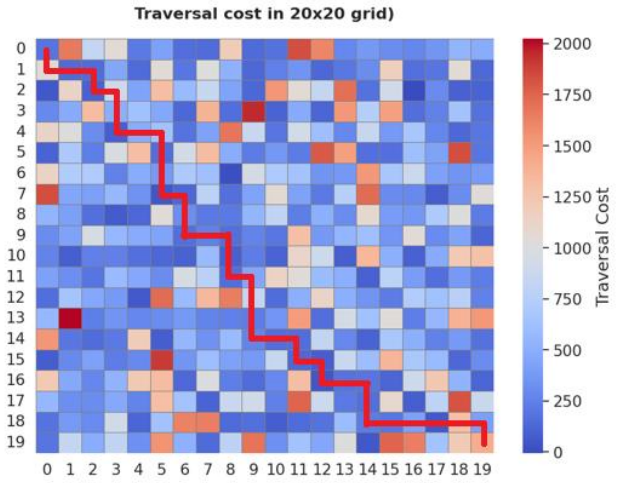
Path length – 39

Before I could implement Dijkstra and A* algorithms graph had to modified again to include weights. BFS and DFS are uninformed search algorithms therefore have no capacity to influence path based on weight.



Dijkstra's Algorithm

Dijkstra is first example of greedy algorithm. It finds shortest path between starting node and all other nodes in weighted graph. It explores all possible paths and chooses one with smallest weight.



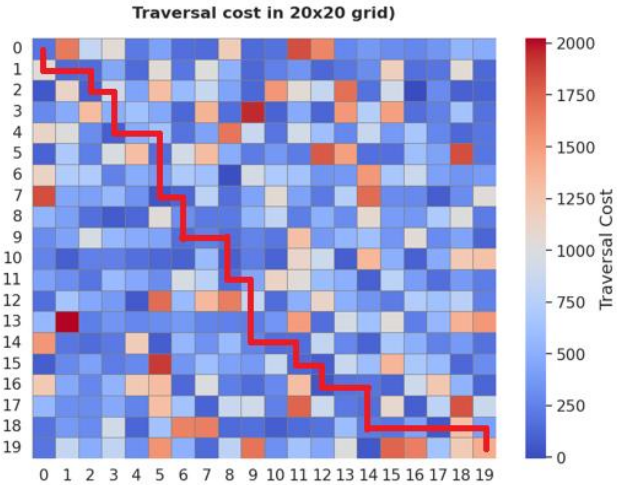
[(0, 0), (1, 0), (1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5), (6, 5), (7, 5), (7, 6), (8, 6), (9, 6), (9, 7), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (13, 9), (14, 9), (14, 10), (14, 11), (15, 11), (15, 12), (16, 12), (16, 13), (16, 14), (17, 14), (18, 14), (18, 15), (18, 16), (18, 17), (18, 18), (18, 19), (19, 19)]

Path length – 39

Total weight - 12916.667999267578

A* Search Algorithm

A* is an informed search algorithm. It's used to find shortest path between two nodes on weighted graph. It works similarly to Dijkstra but with greater efficiency due to use of heuristic function.



[(0, 0), (1, 0), (1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5), (6, 5), (7, 5), (7, 6), (8, 6), (9, 6), (9, 7), (9, 8), (10, 8), (11, 8), (11, 9), (12, 9), (13, 9), (14, 9), (14, 10), (14, 11), (15, 11), (15, 12), (16, 12), (16, 13), (16, 14), (17, 14), (18, 14), (18, 15), (18, 16), (18, 17), (18, 18), (18, 19), (19, 19)]

Path length - 39

Total weight - 12916.667999267578

Recommendation:

Based on achieved results I can with most certainty not recommend BFS and DFS algorithm. When DFS is obvious based on obtained results, BFS despite finding shortest path completely neglected weight of the nodes. Since my goal was to find most optimal path, lack of consideration for weight disqualifies this algorithm as well.

This leaves two algorithms to choose from. Dijkstra and A*. Despite similarities and the same results achieved, there are key differences to consider.

Characteristics	Dijkstra's Algorithm	A* Search Algorithm
Algorithm Type	Greedy	Informed search (uses heuristics)
Initialization	Tentative distance: 0 for source, infinity for others	Tentative distance and heuristic function
Heuristic Use	None	Uses heuristic to estimate remaining cost to goal
Efficiency	Less efficient for large graphs	More efficient with a well-chosen heuristic
Complexity	O(V^2) or O((V + E) log V) with priority queue	Depends on heuristic; typically better than Dijkstra's
Use Case	Network routing protocols	Navigation systems, games, AI pathfinding
Strengths	Guarantees shortest path for all nodes	Efficient pathfinding with good heuristic
Weaknesses	Can be slow for large graphs	Performance depends on heuristic quality

(GeeksforGeeks, 2024)

Taking this into consideration, my recommendation would be A* with Dijkstra in close 2nd place. In my opinion A* has advantage in efficiency and should provide better scalability for example if there's a need to expand grid.

Code availability:

Code available at: https://www.kaggle.com/code/wiktorbiesiadecki/programming-assignment-ai-202421412

Downloaded version of notebook provided as a redundancy measure:

programming-assignment-ai-202421412.ipynb

Note: Graphs and values will be slightly different every time notebook runs.

References:

Wikipedia. (2019). *Economy of the United Kingdom*. [online] Available at: https://en.wikipedia.org/wiki/Economy_of_the_United_Kingdom.

Butler, S. (2022). *Amazon delivery drivers in UK raise alarm over real-terms pay cut*. [online] The Guardian. Available at: https://www.theguardian.com/technology/2022/mar/27/amazon-delivery-drivers-raise-alarm-over-real-terms-pay-cut.

IEA (2024). *United Kingdom - Countries & Regions*. [online] IEA. Available at: https://www.iea.org/countries/united-kingdom/electricity.

Poole, D.L. and Mackworth, A.K. (2010). *Artificial intelligence : foundations of computational agents*. Cambridge: Cambridge University Press.

Mucci, T. (2024). *What Is Predictive AI?* [online] IBM. Available at: https://www.ibm.com/think/topics/predictive-ai.

GeeksforGeeks (2024). *Difference Between Dijkstra's Algorithm and A* Search Algorithm*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/dsa/difference-between-dijkstras-algorithm-and-a-search-algorithm/.

RAC (2021). *Are electric cars really better for the environment? | RAC Drive*. [online] www.rac.co.uk. Available at: https://www.rac.co.uk/drive/electric-cars/choosing/are-electric-cars-really-better-for-the-environment/.

Zheng, M. (2023). *The Environmental Impacts of Lithium and Cobalt Mining*. [online] earth.org. Available at: https://earth.org/lithium-and-cobalt-mining/.

Metrilo Blog. (2019). *Repeat purchase rate for ecommerce brands*. [online] Available at: https://www.metrilo.com/blog/repeat-purchase-rate.

International Trade Administration (2024). *Impact of COVID Pandemic on eCommerce*. [online] International Trade Administration. Available at: https://www.trade.gov/impact-covid-pandemic-ecommerce.

kooaslansefat (2022). *Fairness Evaluation for Boston House Price Pred.* [online] Kaggle.com. Available at: https://www.kaggle.com/code/kooaslansefat/fairness-evaluation-for-boston-house-price-pred/notebook [Accessed 13 Dec. 2025].

kooaslansefat (2021). *DL Hypertuning and Parallel Plots*. [online] Kaggle.com. Available at: https://www.kaggle.com/code/kooaslansefat/dl-hypertuning-and-parallel-plots [Accessed 15 Dec. 2025].

Rakesh (2023). *The Art of Balance: Tackling Overfitting and Underfitting in Linear Regression Models*. [online] Medium. Available at: https://medium.com/@rakeshandugala/the-art-of-balance-tackling-overfitting-and-underfitting-in-linear-regression-models-bc1517b892c8 [Accessed 16 Dec. 2025].

Pandas (n.d.). *pandas.DataFrame.loc — pandas 1.3.3 documentation*. [online] pandas.pydata.org. Available at: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html.

Waskom, M. (2024). *seaborn.heatmap — seaborn 0.10.1 documentation*. [online] seaborn.pydata.org. Available at: https://seaborn.pydata.org/generated/seaborn.heatmap.html.