Uber Dataset: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

Load Dataset:

In [570…

```
import pandas as pd

df = pd.read_csv('./uber.csv')
df.head()
```

Out[570…

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	drop
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	

Clean the Data (deal with missing values)

In [571…

```
df = df.drop(columns=['Unnamed: 0'])
df.dropna()
```

Out[571…

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_lo
0	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73
1	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73
2	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73
3	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73
4	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73
…	…	…	…	…	…	
199995	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_lo
199996	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74
199997	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73
199998	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-73
199999	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-73

199999 rows × 8 columns

According to the df.info(), there is 1 row with a null instance. To correct this, df.dropna() deletes any rows with null values. We will also be deleting the useless ID column (unamed: 0).

Also we need to ensure that the longitude and latitude is validated so our "distance" calculation is accurate.

In [486…

```
# cdoe from kaggle dataset website
df = df[(df.pickup_latitude<90) & (df.dropoff_latitude<90) &
        (df.pickup_latitude>-90) & (df.dropoff_latitude>-90) &
        (df.pickup_longitude<180) & (df.dropoff_longitude<180) &
        (df.pickup_longitude>-180) & (df.dropoff_longitude>-180)]
df
```

Out[486…

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_lo
0	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73
1	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73
2	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73
3	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73
4	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73
…	…	…	…	…	…	
199995	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739367	-73
199996	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736837	-74
199997	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756487	-73

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_lo
199998	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725452	-7
199999	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720077	-7

199987 rows × 8 columns

Compute the distance of the ride

The total distance of the uber ride may be useful when modeling the dataset. The example below shows using the geopy library to calculate the distance between two coordinates in miles. Using a list comphresion we can add a "total distance column"."

In [487…
```
from geopy import distance
# coordinates
coords_1 = (df.iloc[0]['pickup_latitude'], df.iloc[0]['pickup_longitude'])
coords_2 = (df.iloc[0]['dropoff_latitude'], df.iloc[0]['dropoff_longitude'])
# compute the distance in miles:
x = float(distance.distance(coords_1, coords_2).miles) # distance in miles
x
```

Out[487… 1.0445937861491572

In [488…
```
total_distance = [float(distance.distance((df.iloc[x]['pickup_latitude'], df.iloc[x]['p
                                          (df.iloc[x]['dropoff_latitude'], df.ilo
                                          for x in range(0,len(df))]
# append to df
df['total_distance'] = total_distance
```

In [489…
```
df['total_distance'] = total_distance
```

In [491…
```
df.head()
```

Out[491…

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longituc
0	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.9995
1	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.9947
2	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.9625
3	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.9653

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitud
4	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.9730

Implement a custom transformer (to make a Train and Test split)

- x = distance
- y = fare amount

We use .iloc and .reshape to make a custom transform to split our train and test sets properly.

In [492…

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2)

x_train, y_train = train.iloc[:,8].values.reshape(-1,1), train.iloc[:,1].values.reshape
x_test, y_test = test.iloc[:,8].values.reshape(-1,1), test.iloc[:,1].values.reshape(-1,
```

Use sklearn.linear_model.LinearRegression

In [493…

```
import numpy as np
from sklearn.linear_model import LinearRegression

lrg_model = LinearRegression().fit(x_train, y_train)
```

In [494…

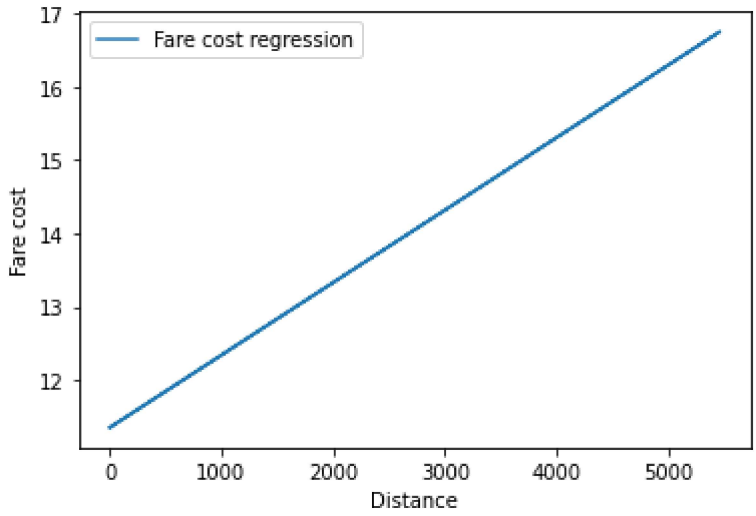
```
predictions = lrg_model.predict(x_test)
```

In [495…

```
import matplotlib.pyplot as plt

# create data
x = x_test
y = np.array(predictions).T.tolist()[0]
x2 = x_test
y2 = y_test

# plot lines
plt.plot(x, y, label = "Fare cost regression")
plt.xlabel('Distance')
plt.ylabel('Fare cost')
plt.legend()
plt.show()
```



First we used a simple regression model where the distance modesl the fare cost. We graph the distance over the fare. This line can now be used to make predictions about the fare. Examining the predictions array alot of the values were similar.

Decesion Tree Regression

A DecisionTreeRegressor will be used to build a tree with leafs and nodes which will give us more accurate results than a linear regression line.

In [640…
```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

dtr_model = DecisionTreeRegressor().fit(x_train, y_train)
```

In [641…
```
predictions_dtr = dtr_model.predict(x_test)
```

In [642…
```
y_predict = predictions_dtr
y_true = y_test
predictions_dtr
```

Out[642… array([12.9 , 18. , 18.65, ..., 15. , 11.5 , 16.])

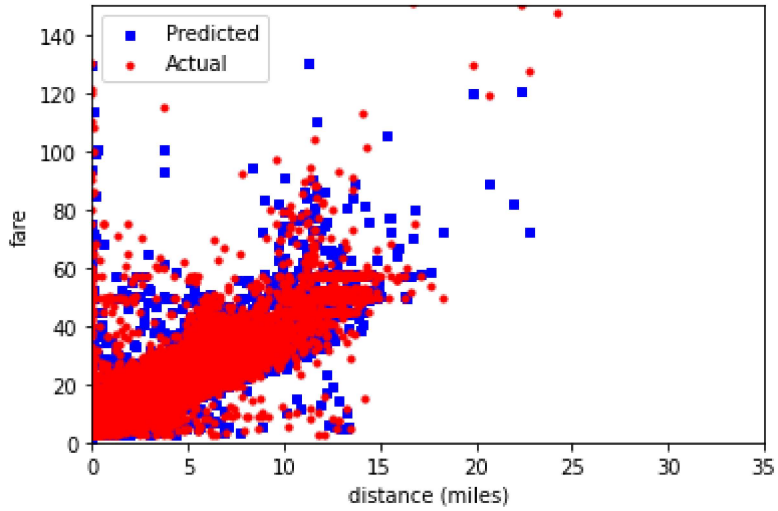
Midsize Plot (0,35 miles)

In [500…
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x_test, y_predict, s=10, c='b', marker="s", label='Predicted') #
ax1.scatter(x_test, y_true, s=10, c='r', marker="o", label='Actual')
plt.xlim([0, 35])
plt.ylim([0, 150])
plt.xlabel("distance (miles)")
plt.ylabel("fare")
```

```
plt.legend(loc='upper left');
plt.show()
```



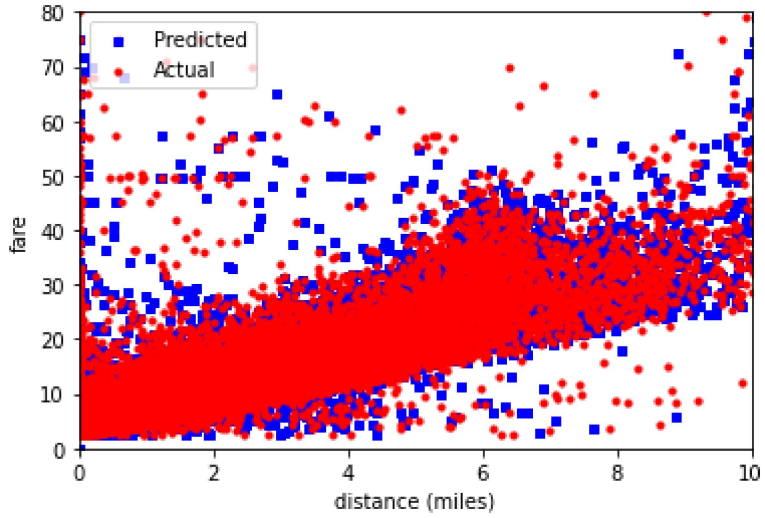
Zoomed In Plot (0-10 miles):

In [501…

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x_test, y_predict, s=10, c='b', marker="s", label='Predicted') #
ax1.scatter(x_test, y_true, s=10, c='r', marker="o", label='Actual')
plt.xlim([0, 10])
plt.ylim([0, 80])
plt.xlabel("distance (miles)")
plt.ylabel("fare")
plt.legend(loc='upper left');
plt.show()
```



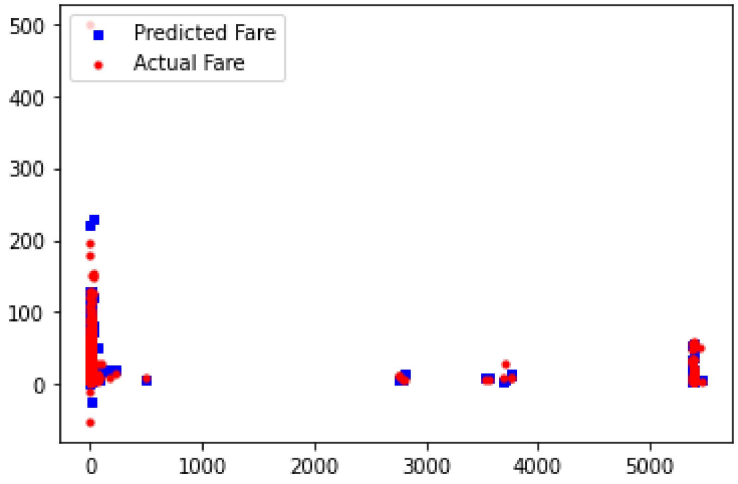
Zoomed Out Plot (No limits):

In [502…

```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x_test, y_predict, s=10, c='b', marker="s", label='Predicted Fare') #
ax1.scatter(x_test, y_true, s=10, c='r', marker="o", label='Actual Fare')
plt.legend(loc='upper left');
plt.show()
```



Examining the distribution results, the model was close to predicting the actual cost. The paramaters could be hypertuned to adjust the number of nodes/leaf for greater accuracy.

Use sklearn.metrics.mean_squared_error and at least one other sklearn.metrics option to evaluate model performance

In [643…

```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_true, y_predict, squared=False)
```

Out[643… 7.001136993391676

In [644…

```
from sklearn.metrics import r2_score

r2_score(y_true, y_predict)
```

Out[644… 0.5165019297439098

root mean error:

- Perfect model = 0.00
- Our model = 7.00

This score means there is a difference in estimates and actual. This higher means indicates our model may predict "fare values" higher than the actual values. This may be due to the model being confused by outliers.
https://www.mygreatlearning.com/blog/mean-square-error-explained/

r2 score

- Perfect score = 1.0
- Our model - 0.52

The r2 score means the distance and fare values are somewhat correlated. We can use a better model (like a neural network) or hypertuning to increase the r2 score.

Scale/normalize/standardize features using sklearn.preprocessing

We will reproduce the exact same decesion regression process only with data preprocessing. We will compare the new preprocessed results with the old results.

Using sklearn.preprocessing, we need to normalize our data due to the data distribution and outliers. The minmax scalar will be used to normalize our data. The MinMax scalar puts all the X values between [0,1]- which works to make our model more accurate during training.

In [604…]
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

#scaler = StandardScaler() # MinMax Scaler nomralizes
scaler = MinMaxScaler() # MinMax Scaler nomralizes
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)
```

In [605…]
```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

dtr_model_scaled = DecisionTreeRegressor().fit(x_train_scaled, y_train)
```

In [606…]
```
predictions_dtr = dtr_model_scaled.predict(x_test_scaled)

y_predict = predictions_dtr
y_true = y_test
```

Compute Metrics:

In [607…]
```
from sklearn.metrics import r2_score

r2_score(y_test, predictions_dtr)
```

Out[607… 0.6011915598197504

In [608…]
```
from sklearn.metrics import mean_squared_error

mean_squared_error(y_true, y_predict, squared= False)
```

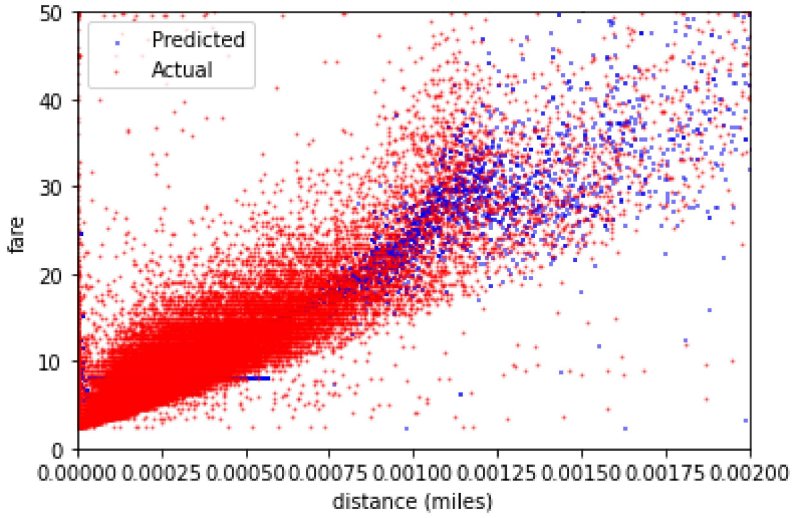
Out[608… 6.358480937710742

Updated Graph:

In [609…]
```
import matplotlib.pyplot as plt
```

```
fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(x_test_scaled, y_predict, s=1, c='b', marker="s", alpha=0.5, label='Predict
ax1.scatter(x_test_scaled, y_true, s=1, c='r', marker="o", alpha=0.5, label='Actual')
plt.xlabel("distance (miles)")
plt.ylabel("fare")
plt.xlim([0,.002])
plt.ylim([0,50])
plt.legend(loc='upper left');
plt.show()
```



As expected our model is now more accurate (better r2 and MNSE values)

r2 score

- Before scaling: 0.5165019297439098
- After scaling: 0.6011915598197504

Use Grid Search CV or RandomizedSearch CV to tune hyperparameters for a model

Using grid serach CV, we can get a dictionary of paramaters that will increase our decesion tree model's performance

In [610…

```
parameters={"splitter":["best","random"],
            "max_depth" : [1,5,10],
           "min_samples_leaf":[1,2,3,4],
           "min_weight_fraction_leaf":[0.1,0.2,0.3,0.4],
           "max_features":["auto","log2","sqrt",None],
            }


tuning_model=GridSearchCV(dtr_model,param_grid=parameters,scoring='neg_mean_squared_err
```

In [611…

```
tuning_model.fit(x_train_scaled,y_train)
```

Out[611…
```
GridSearchCV(cv=3, estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [1, 5, 10],
                         'max_features': ['auto', 'log2', 'sqrt', None],
                         'min_samples_leaf': [1, 2, 3, 4],
                         'min_weight_fraction_leaf': [0.1, 0.2, 0.3, 0.4],
                         'splitter': ['best', 'random']},
             scoring='neg_mean_squared_error')
```

In [612…
```
tuning_model.best_params_
```

Out[612…
```
{'max_depth': 5,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.1,
 'splitter': 'best'}
```

In [613…
```
dtr_hypertuned= DecisionTreeRegressor(max_depth=3,max_features='auto',max_leaf_nodes=No
dtr_hypertuned.fit(x_train_scaled, y_train)
```

Out[613…
```
DecisionTreeRegressor(max_depth=3, max_features='auto',
                      min_weight_fraction_leaf=0.1)
```

In [614…
```
predictions_dtr = dtr_hypertuned.predict(x_test)

y_predict = predictions_dtr
y_true = y_test
```

Using tuning_model.best*params*, we can see the output of our paramaters for hypertuning. We can feed this paramaters back into the model for more accurate results.

Create a single pipeline that does full process from data preparation to final prediction.

Using a pipeline and the .fit() function, we can do all data preperation, model creation, and model fitting all in one line.

In [634…
```
from sklearn.pipeline import Pipeline

pipe = Pipeline([('scaler', MinMaxScaler()), ('dtr', DecisionTreeRegressor())]).fit(x_t
```

In [635…
```
pipe.score(x_test_scaled, y_test)
```

Out[635…
```
0.6011915598197504
```

In [636…
```
pipe.predict([[0.00050359]])
```

Out[636…
```
array([8.21484461])
```

The score for our pipeline is the same as our decesion tree regressor model (as expceted). The flow allows the data to be scaled with the minmax scaler and the creation of the decesion tree regression model.