LASSONDE SCHOOL OF ENGINEERING | YORK U

Final Project

EECS3401

Introduction to Artificial Intelligence and Logic Programming

Predicting Uber Fares with the Uber Fares Dataset

Authors:	*Student Numbers:*
Group 36	
Kanwar Partap Pannu	218918938
Gabriel Rincon	218624734
Abdullah Sajjad	217250945

Wednesday 22nd November, 2023

Contents

1 Framing the Problem and Looking at the Big Picture

Uber took the world by storm with it's founding in 2009 and quickly became one of the most used forms of transportation across North America and much of rest of the world. It appeals to many thanks to its convenience and accessibility. When calculating the price of an Uber ride many factors are considered such as time, distance etc. but no clear formula can tell you your fare ahead of time. The ultimate goal of the Uberfares data set is to more accurately predict and understand the Uber pricing model. For Uber it is very important to understand how they're pricing models work to help Uber coordinate any initiatives relating to the pricing model and to ensure better compliance with local regulations. Better fare predictors can also allow drivers to better asses their potential earnings from a certain ride and ensure they can make the best decisions. Most importantly however the user experience would vastly improve, with a more accurate fare the user can plan for trips ahead of time and perform cost-benefit analyses with other options. Framing the problem from the dataset is quite clear and straight forward. This dataset is a supervised learning problem because the data is labelled with the target variable being the fare amount. Since the target variable is a continuous numerical value which we are trying to predict this is a regression task. This dataset is not updated live with a continuous flow of data thus we can train the the model on the whole dataset at once with batch learning.

2 Exploratory Data Analysis

2.1 Description of the Data Set

The Uber Fares data set includes the fares for many different unique trips mostly centered around New York although some data points are in very far away regions. It contains 200000 entries before data cleaning. Analysis done in this section is done on the cleaned dataset, the details of this cleanup are in section 3. The following fields are included in the orginial data set:

- key - A unique identifier for each trip
- fare_amount - fare cost in USD
- pickup_datetime - Date and time at the start of the trip
- passenger_count - Number of passengers inside the vehicle according to the driver
- pickup_longitude - Longitude at the start of the trip
- pickup_latitude - Latitude at the start of the trip
- dropoff_longitude - Longitude at the end of the trip
- dropoff_latitude - Latitude at the end of the trip

Below is some information on the cleaned data set where the key field has been removed.

Field	Non-Null Count	Data Type
fare_amount	199259	Float64
pickup_datetime	199259	Int64
pickup_longitude	199259	Float64
pickup_latitude	199259	Float64
dropoff_longitude	199259	Float64
dropoff_latitude	199259	Float64
passenger_count	199259	Int64

Using the head function we can observe the first five entries of the dataset shown below to understand the dataset better.

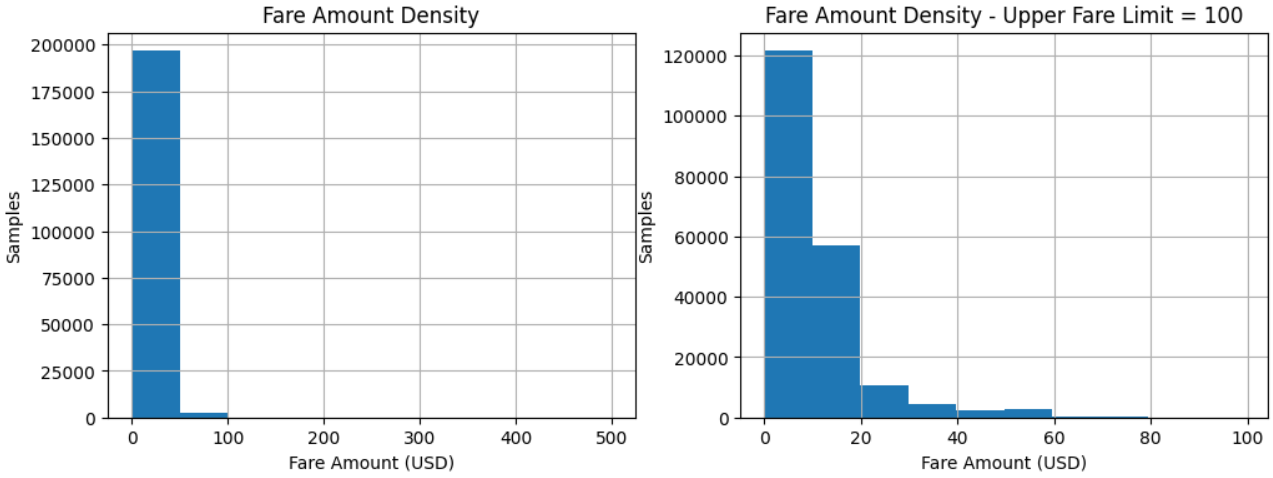
	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	1431028326000000000	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	1247861096000000000	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	1251150300000000000	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	1246004541000000000	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	1409248020000000000	-73.925023	40.744085	-73.973082	40.761247	5

The describe function gives a numerical description of the data, for example 25% of the Uber fares are below six dollars usd.

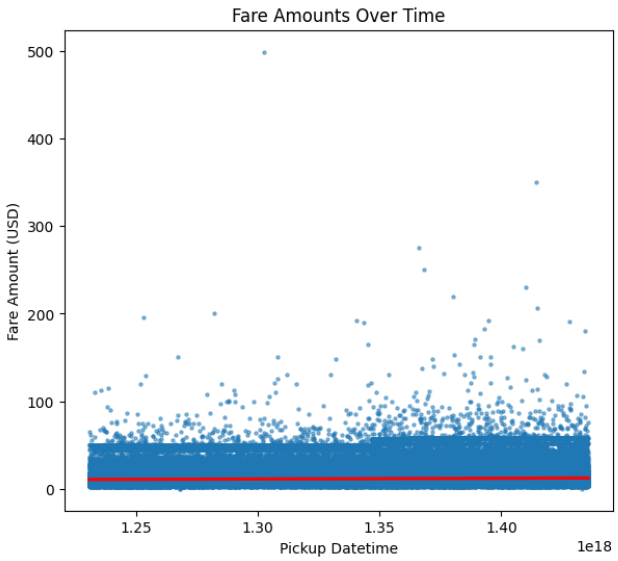
	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	199259.000000	1.992590e+05	199259.000000	199259.000000	199259.000000	199259.000000	199259.000000
mean	11.369411	1.332473e+18	-72.504195	39.919184	-72.517762	39.925768	1.690488
std	9.905917	5.830927e+16	10.442166	6.127711	10.509222	6.198131	1.384791
min	0.010000	1.230773e+18	-93.824668	-74.015515	-737.916665	-74.015750	1.000000
25%	6.000000	1.282489e+18	-73.992064	40.734792	-73.991409	40.733828	1.000000
50%	8.500000	1.332573e+18	-73.981825	40.752582	-73.980094	40.753042	1.000000
75%	12.500000	1.382305e+18	-73.967161	40.767155	-73.963663	40.767995	2.000000
max	499.000000	1.435708e+18	40.808425	48.018760	40.831932	493.533332	208.000000

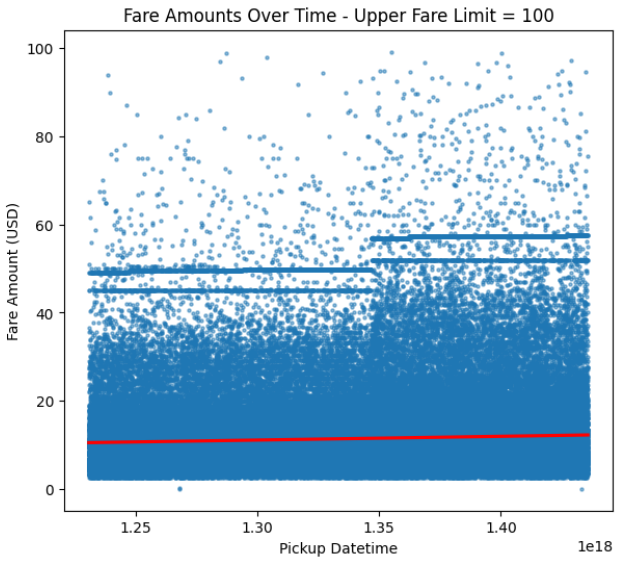
2.2 Graph Analysis

While these tables give us an understanding of the dataset structure and values included graphs will help us better understand the correlations and interesting relationships between the data points. As we can see from the resultant graph displaying all the fare amounts as a histogram, majority of the fares are distributed in the range $0 - $25.

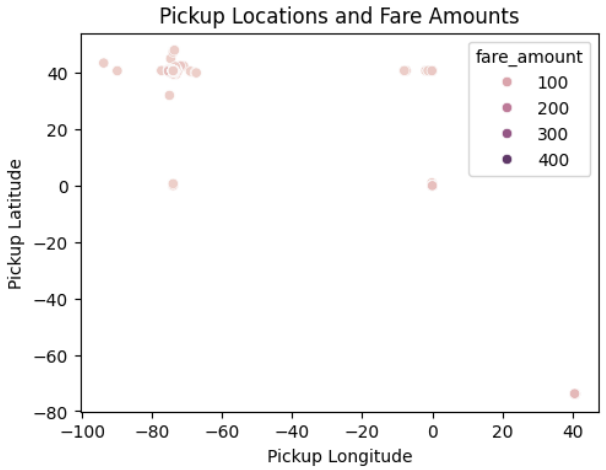


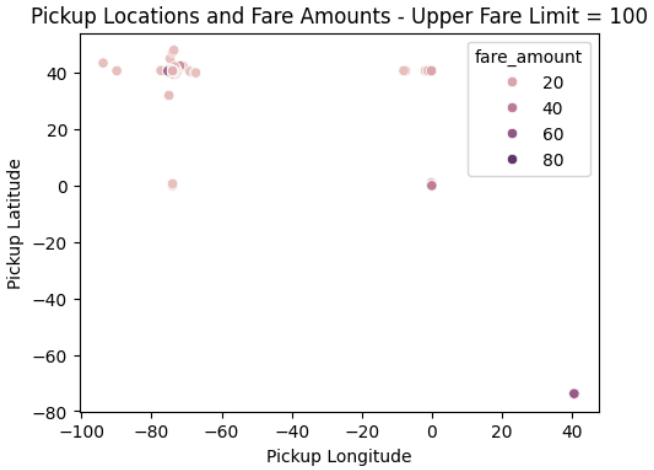
The next following graphs show the fare amount in respect to date-time with the original dataset and one restricted to fare amounts lower than 100 dollars for better visualisation. From these graphs, it's a little hard to visually see any overall correlation between date-time and fare amount. The main observation that can be drawn is the increase in fare amounts near data-time 1.35e18 which corresponds to some time in 2012 where there was likely a mark up in service costs. Other observations will likely need to be left to be seen in the correlation matrix.



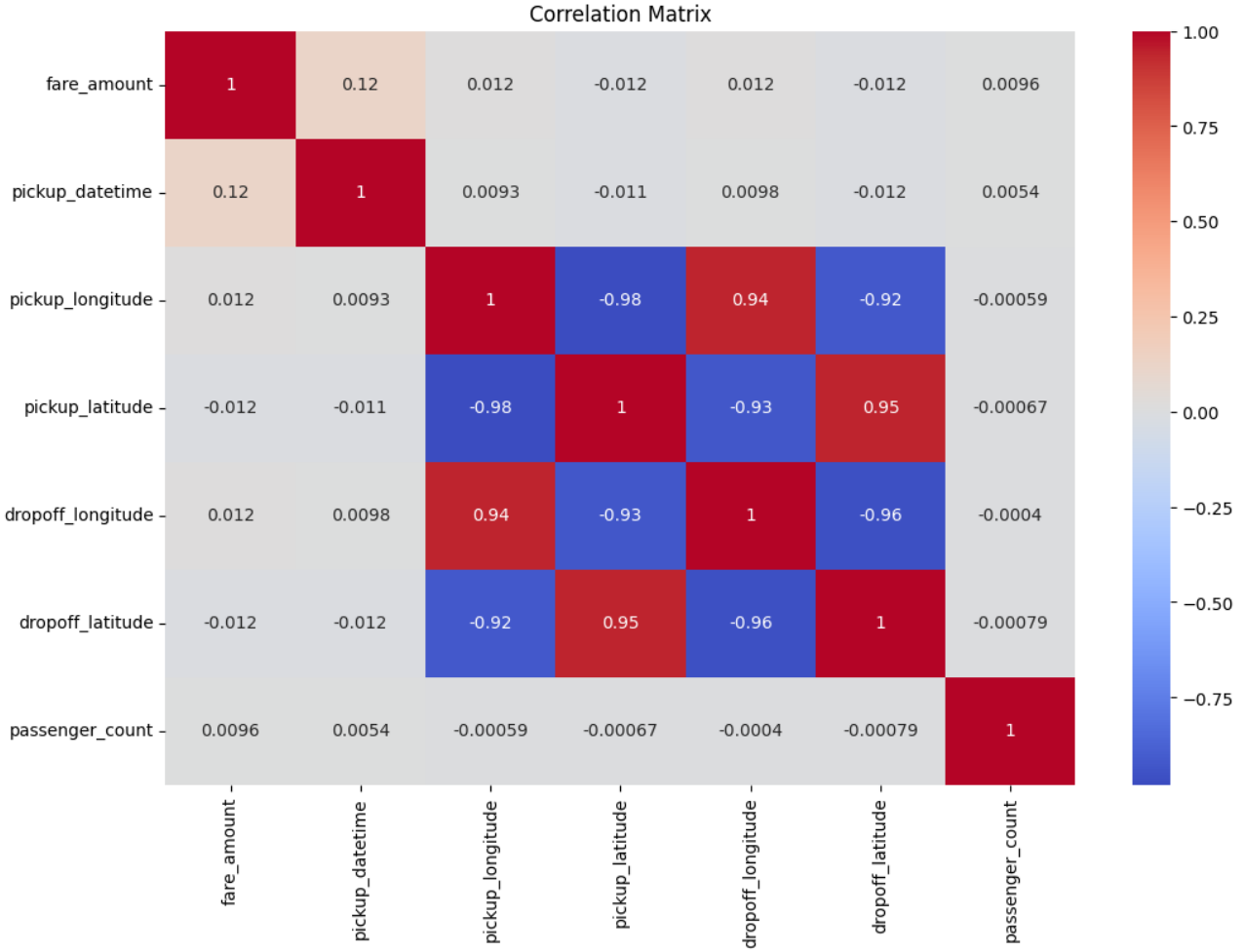


The last graphs compare fare amount and pickup location with the original dataset and one restricted to fare amounts lower than 100 dollars for better visualisation. It helps to identify any geographical patterns or areas with higher fares. There doesn't seem to be a clear correlation between geographical location and price. It should be noted however that in the second graph, there is a significantly higher cost fare in the bottom right however those coordinates are in Antarctica so it's legitimacy is questionable.





Finally from the correlation matrix we can conclude that the fare seems to be reliant more on pickup date and time as compared to other variables, with pickup longitude as the runner-up.



3 Data Preparation

3.1 Data Cleaning

To make the data suitable for preparation, several data points were removed. Firstly all NA values were removed, then the key field was removed for all entries as it is an unrelated feature in predicting the price of the ride. Additionally all entries with negative fares and zero passenger counts were removed as these signified data entry errors or other discrepancies. Then the data set was restricted to include only valid coordinates as many coordinates went beyond the limit of 180 or -180 thus these points needed to be removed, this was done by limiting the longitude and latitude values. Finally all dates were converted to integer representations.

3.2 Pipe lining and Preparation for Regression

Before pipe lining the target feature being the fare amount is separated. For pipe lining a modified version of the end to end ML project sample on the course website is used. First the missing numerical values are filled with the mean using a SimpleImputer then the numerical columns scaled using StandardScaler without scaling the target. Then the data set is split into 80% training set and 20% test.

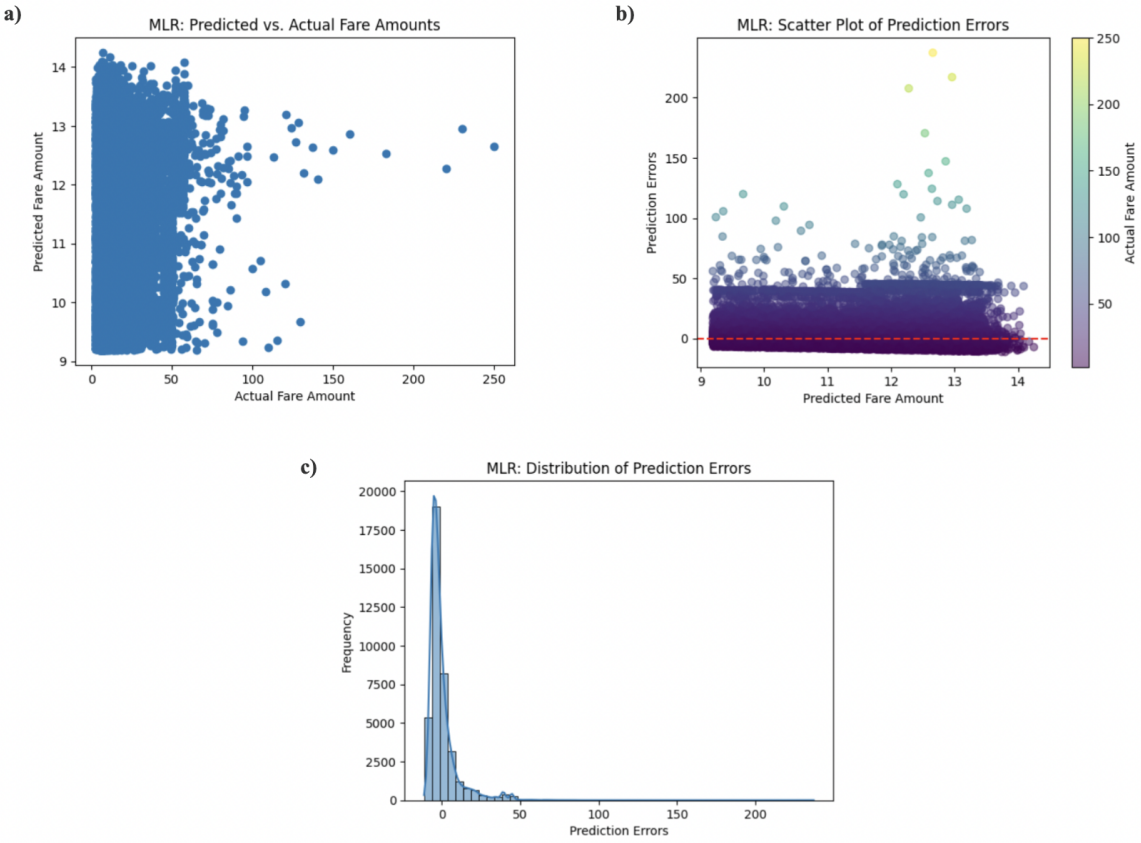
4 Training and Comparison

As mentioned in section 1 this task is a regression task so three different models were used which are best suited for regression tasks. The models used were multiple linear regression, ridge regression and elastic net regression. For each model the mean squared error and the mean absolute error were calculated and evaluated based on the scoring from sklearn. Below is a table comparing these results of the different regression models.

	Multiple Linear	Ridge	Elastic Net
Mean Squared Error	9.999959878142283	9.999959958084101	10.032650021093067
Mean Absolute Error	6.033518385262153	6.033518453483443	6.061790508287913
Sklearn Score	0.0158666835743817	0.015866667839637305	0.009421849019126127

5 Best Performing Algorithm

Based on the scores provided in the previous section, the multiple linear regression(MLR) model is the best preforming algorithm with ridge regression being a close second. Below there are three graphs displaying the results from the MLR model. Graph a displays the predicted values in respect to the actual values, the extreme outliers were not accurately predicted by the model. It is unclear whether this is due to the model not being able to generalize or if these points are data errors that seem legitimate and were not cleaned out during the data cleaning step. Graph b plots predicted values versus error values and uses a colour gradient for the actual fare amount and shows similar trends to graph A. Graph C plots the distribution in errors for the model. Based on the graphs shown below there does not seem to be a clear correlation between the features and the prediction errors which is good in the sense that this means that the model isn't over or under valuing any feature but bad in the sense that this means it is difficult to associate any problematic features in terms of consistency.



6 Limitations

Even the best model of the three trained did not achieve the best results possible which may be due to many reasons. Firstly the dataset includes many points which are valid points able to be passed through to the model but make no sense to the overall dataset or big picture. For example the datasets are mostly centered around New York however there are points in Antarctica, Europe, Africa and other points far away from New York which can be passed to the network but are hard to determine as being legitimate. While we decided to include all data points whose coordinates lied within the bounds of the globe, there were other factors that maybe

should have been considered but were not to prevent input bias. For example, since the extreme majority of data points were within New York, should we have restricted the data to New York since for any prediciton outside of New York, the accuracy will fall and if we are predicting within the bounds of the city, then restricting the training to the city should improve performance. There were some coordinates as well that while they were valid coordinates, they were in the ocean however these would prove very complicated to clean without the help of an external library so these were left as is. The last consideration was in the fare cost. The extreme majority of the fare costs didn't exceed 25$ and looked believable up to 100$. The fare costs that went into the hundreds, even as high as $500 seem suspicious and if there was more evidence of these being faulty data points, they would have been removed. Since there was no firm evidence of that, they were kept and as a result, are partly to blame for the fact that the best performing model's MAE to be as high as 6 when 75% of trips are under $12.5.

A

Source Code

```
# %% [markdown]
# # Uber Fares Predictions

# %% [markdown]
# ## Preparing The Data

# %% [markdown]
# ### Import Relevant Libraries

# %%
import sklearn
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from IPython.core.interactiveshell import InteractiveShell
from IPython.display import Markdown, display
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, ElasticNet

# %% [markdown]
# ### Configuring Libraries

# %%
pd.options.mode.chained_assignment = None

# Use a fixed seed to make results reproducible
np.random.seed(873409)

# %% [markdown]
# ### Loading The Dataset
#
# Note: Kaggle does not support importing directly so the .csv must be in the same directory as
↪  the notebook

# %%
df = pd.read_csv("uber.csv")

# %% [markdown]
# ### Cleaning The Dataset

# %%
# Define coordinate bounds
longitude_min = -180
longitude_max = 180
latitude_min = -90
latitude_max = 90

# Dropping unrelated features
df.drop(['Unnamed: 0', 'key'], inplace=True, axis=1)

# Remove na values
df.dropna(inplace=True)

# Remove faulty data
```

```
df = df[df.fare_amount > 0]

df = df[
    (df.pickup_longitude >= longitude_min) &
    (df.pickup_longitude <= longitude_max) &
    (df.pickup_latitude >= latitude_min) &
    (df.pickup_latitude <= latitude_max)
]

df = df[df.passenger_count > 0]

# Convert date to int
df.pickup_datetime = pd.to_numeric(pd.to_datetime(df.pickup_datetime))

# %% [markdown]
# #### Creating A Slightly Stricter Dataframe For EDA Graph Pairs

# %%
filtered_df = df[df.fare_amount < 100]

# %% [markdown]
# ## Exploratory Data Analysis
#
# Note: For all graphs, a second one is generated with an upper limit of £100USD so that the patterns are more easily observable at values closer to the mean.

# %% [markdown]
# ### Exploring The Data

# %%
display(df.info())

# %%
display(df.head())

# %%
display(df.describe())

# %%
display(Markdown("> Data Shape: {}".format(df.shape)))

# %% [markdown]
# ### Histogram of Fare Amounts
#
# As we can see from the resultant graph, majority of the fares are distributed in the range £0 - £25.

# %%
plt.figure(figsize=(12,4))
plt.subplot(121)
df.fare_amount.hist()
plt.title('Fare Amount Density')
plt.xlabel('Fare Amount (USD)')
plt.ylabel('Samples')
plt.subplot(122)
plt.title('Fare Amount Density - Upper Fare Limit = 100')
plt.xlabel('Fare Amount (USD)')
plt.ylabel('Samples')
filtered_df.fare_amount.hist()
plt.show()

# %% [markdown]
# ### Scatter Plot With Line of Best Fit of Fare Amounts Over Time
```

```
#
# From these graphs, it's a little hard to visually see any overall correlation between Datetime and fare amount. The main observation that can be drawn is the increase in fare amounts near datatime 1.35e18 which corresponds to some time in 2012 where there was likely a mark up in service costs. Other observations will likely need to be left to be seen in the correlation matrix

# %%
plt.figure(figsize=(15, 6))
plt.subplot(121)
sns.regplot(x='pickup_datetime', y='fare_amount', data=df, scatter_kws={'s': 5, 'alpha': 0.5}, line_kws={'color': 'red'})
plt.title('Fare Amounts Over Time')
plt.xlabel('Pickup Datetime')
plt.ylabel('Fare Amount (USD)')
plt.subplot(122)
sns.regplot(x='pickup_datetime', y='fare_amount', data=filtered_df, scatter_kws={'s': 5, 'alpha': 0.5}, line_kws={'color': 'red'})
plt.title('Fare Amounts Over Time - Upper Fare Limit = 100')
plt.xlabel('Pickup Datetime')
plt.ylabel('Fare Amount (USD)')
plt.show()

# %% [markdown]
# ### Scatter Plot of Fare Amounts Against Pickup Locations
#
# The following graph displays the distribution of fares with respect to location.  It helps to identify any geographical patterns or areas with higher fares. There doesn't seem to be a clear correlation between geographical location and price. It should be noted however that in the second graph, there is a siginificantly higher cost fare in the bottom right however those coordinates are in Antartica so it's legitemacy is questionable.

# %%
plt.figure(figsize=(12, 4))
plt.subplot(121)
sns.scatterplot(x='pickup_longitude', y='pickup_latitude', hue='fare_amount', data=df)
plt.title('Pickup Locations and Fare Amounts')
plt.xlabel('Pickup Longitude')
plt.ylabel('Pickup Latitude')
plt.subplot(122)
sns.scatterplot(x='pickup_longitude', y='pickup_latitude', hue='fare_amount', data=filtered_df)
plt.title('Pickup Locations and Fare Amounts - Upper Fare Limit = 100')
plt.xlabel('Pickup Longitude')
plt.ylabel('Pickup Latitude')
plt.show()

# %% [markdown]
# ### Correlation

# %%
# Calculate the correlation matrix
correlation_matrix = df.corr()

# Visualize the correlation matrix as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# %% [markdown]
# From this matrix we can conclude that the fare seems to be reliant more on pickup date and time as compared to other variables, with pickup longitude as the runner-up.

```

```
# %% [markdown]
# ## Preparing Data For Regression

# %% [markdown]
# ### Separate Target Feature

# %%
dft = df.fare_amount
df.drop(['fare_amount'], inplace=True, axis=1)
display(Markdown("> Feature Set Shape: {}".format(df.shape)))
display(Markdown("> Target Set Shape: {}".format(dft.shape)))

# %% [markdown]
# ### Apply Pipeline
#
# Note: This code is an adapted version of the code in the end-to-end ML samples on the course website

# %%
num_cols = df.select_dtypes(include='number').columns.to_list()

# Create pipelines for numeric and categorical columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'), StandardScaler())

# Use ColumnTransformer to set the estimators and transformations

preprocessing = ColumnTransformer([('num', num_pipeline, num_cols)],
                                  remainder='passthrough'
                                 )

display(preprocessing)

df_prepared = preprocessing.fit_transform(df)

feature_names = preprocessing.get_feature_names_out()
df_prepared = pd.DataFrame(data=df_prepared, columns=feature_names)

display(df_prepared.shape)

# %% [markdown]
# ### Split the datasets

# %%
df_train, df_test, dft_train, dft_test = train_test_split(df_prepared, dft, test_size=0.2, random_state=42)
display(Markdown("> %s %s %s %s" % (df_train.shape, dft_train.shape, df_test.shape, dft_test.shape)))

# %% [markdown]
# ## Training 3 Different Regression Models

# %% [markdown]
# ### Multiple Linear Regression

# %%
mlr = LinearRegression()
mlr.fit(df_train, dft_train)

# %% [markdown]
# ### Ridge Regression

# %%
rr = Ridge()
```

```
rr.fit(df_train, dft_train)

# %% [markdown]
# ### Elastic Net Regression

# %%
enr = ElasticNet()
enr.fit(df_train, dft_train)

# %% [markdown]
# ## Comparing Performances Between The Models

# %% [markdown]
# ### Multiple Linear Regression

# %%
mlr_dft_predict = mlr.predict(df_test)

mlr_RMSE = mean_squared_error(dft_test, mlr_dft_predict, squared=False)
mlr_MAE = mean_absolute_error(dft_test, mlr_dft_predict)

display(Markdown("> Multiple Linear Regression RMSE: %s" % mlr_RMSE))
display(Markdown("> Multiple Linear Regression MAE: %s" % mlr_MAE))

# %% [markdown]
# ### Ridge Regression

# %%
rr_dft_predict = rr.predict(df_test)

rr_RMSE = mean_squared_error(dft_test, rr_dft_predict, squared=False)
rr_MAE = mean_absolute_error(dft_test, rr_dft_predict)

display(Markdown("> Ridge Regression RMSE: %s" % rr_RMSE))
display(Markdown("> Ridge Regression MAE: %s" % rr_MAE))

# %% [markdown]
# ### Elastic Net Regression

# %%
enr_dft_predict = enr.predict(df_test)

enr_RMSE = mean_squared_error(dft_test, enr_dft_predict, squared=False)
enr_MAE = mean_absolute_error(dft_test, enr_dft_predict)

display(Markdown("> Elastic Net Regression RMSE: %s" % enr_RMSE))
display(Markdown("> Elastic Net Regression MAE: %s" % enr_MAE))

# %% [markdown]
# ### Comparing The Performances using sklearn's scoring

# %%
display(Markdown("> Multiple Linear Regression Score: %s" % (mlr.score(df_test, dft_test))))
display(Markdown("> Ride Regression Score: %s" % (rr.score(df_test, dft_test))))
display(Markdown("> Elastic Net Regression: %s" % (enr.score(df_test, dft_test))))

# %% [markdown]
# ## Analysing The Best Performing Model - Multiple Linear Regression

# %% [markdown]
# ### Calulating Prediction Errors

# %%
```

```
prediction_errors = dft_test - mlr_dft_predict

# %% [markdown]
# ### Graphing Predicted vs Actual Values

# %%
plt.scatter(dft_test, mlr_dft_predict)
plt.xlabel("Actual Fare Amount")
plt.ylabel("Predicted Fare Amount")
plt.title("MLR: Predicted vs. Actual Fare Amounts")
plt.show()

# %% [markdown]
# ### Graphing Scatter Plot of Prediction Errors

# %%
# Scatter Plot of Prediction Errors
plt.scatter(mlr_dft_predict, prediction_errors, c=dft_test, cmap='viridis', alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')
plt.colorbar(label="Actual Fare Amount")
plt.xlabel("Predicted Fare Amount")
plt.ylabel("Prediction Errors")
plt.title("MLR: Scatter Plot of Prediction Errors")
plt.show()

# %% [markdown]
# ### Graphing Distribution of Prediction Errors

# %%
sns.histplot(prediction_errors, bins=50, kde=True)
plt.xlabel("Prediction Errors")
plt.ylabel("Frequency")
plt.title("MLR: Distribution of Prediction Errors")
plt.show()

# %% [markdown]
# There does not seem to be a clear correlation between the features and the prediction errors
↪  which is good in the sense that this means that the model isn't over or under valuing any
↪  feature but bad in the sense that this means it is difficult to associate any problematic
↪  features in terms of consistency.



```

B Links

B.1 Video

https://youtu.be/nywGmpOWxcU

B.2 Dataset

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

B.3 GitHub

https://github.com/GabrielERincon/IntrotoAI-UberFares