STAT 525 : *Intermediate Statistical Methodology*

Analysis of City of Chicago's Rideshare Data
Using Neural Networks for Statistical Modeling

Shiraz Khan

Abstract

This project aims to fulfill two complementary objectives. The first is to analyze the data of rideshare trips in Chicago in 2018-19. Specifically, it models the consumer behavior over the course of a day, and categorically for different days of the week. The concepts introduced in STAT525 are employed for inference and assessment of regression models.

The second objective of this project is to use neural networks for regression. This allows us to use a unified approach that can fit both linear and non-linear models of varying degrees of complexity, and to arbitrary accuracy. The advantages and disadvantages and notable characteristics of a neural network based approach are noted.

I. INTRODUCTION

A. CHICAGO RIDESHARE DATA

The *City of Chicago* recently published data[1] consisting of all the rideshare (presumable Uber/Lyft) trips within the city since November 2018. The information includes approximate pickup/dropoff locations, times and fares. The full 26.6GB dataset has 101 million rows, each corresponding to a single rideshare trip, and can be downloaded from the listed reference. The description of the data, as given on the City of Chicago website[1],

> *"All trips, starting November 2018, reported by Transportation Network Providers (sometimes called rideshare companies) to the City of Chicago as part of routine reporting required by ordinance."*

The dataset was published after several privacy considerations were addressed, such as

- Trip start/stop times are rounded to the nearest 15-minute interval.
- Tips are rounded off to the nearest \$1.00; Fares to the nearest \$2.50.
- For about one-third rows of the data, only the community areas of pickup/dropoff are provided, and each spans about 3 sq miles.

Such modifications should not interfere with our analysis, but are put in place so one cannot reverse-engineer the identity and personal information of a single person.

B. NEURAL NETWORKS

A Neural Network (NN) is essentially a continuous function approximator. It accepts a numerical vector input and returns a vector output. We will use such an approximation to arrive at the mapping $f_{w,b} : \mathcal{R}^i \to \mathcal{R}^o$, such that

$$\hat{Y}_i = f_{w,b}(X_i) \tag{1}$$

where $w \in \mathcal{R}^n$ and $b \in \mathcal{R}^m$ are parameters of f (called the **weights** and **biases** respectively). $\hat{Y}_i$ is (as we will demonstrate) an unbiased estimate of Y_i. When the NN is chosen to be a linear function approximator, the values of w and b are equal to the ordinary least squares regression coefficients.

The function f can be linear or non-linear depending on the structure of the NN. The structure of an NN can be discussed in terms of **layers** and **activation functions**. Each NN has exactly one input and one output layer. The other layers (called hidden layers) increase the complexity and fidelity of the function approximation. The choice of activation function further increases the complexity of the approximation. A network with 3 linear layers looks like

```
HiddenLayer = LinearActivation(InputLayer)
OutputLayer = LinearActivation(HiddenLayer)
```

Hence, for using an NN as a regression model, we may choose the following structures,

- **For Linear Regression :** Input Layer $\xrightarrow{\text{Linear}}$ Output Layer
- **For Non-Linear Regression :** Input Layer $\xrightarrow{\text{Non-Linear}}$ Hidden Layer $\xrightarrow{\text{Linear}}$ Output Layer

The reason why we insist on having a linear layer at the end, is that many activation functions have a limited range (such as $[0, 1]$). A detailed explanation of weights and biases will be omitted here. The weights and biases$w$ and b are updated by solving the following optimization problem. Defining the mean squared loss (l_i) as,

$$l_i = (\hat{Y}_i - Y_i)^2 \tag{2}$$

Our regression approach is then,

$$[w, b] = \underset{[w,b]}{\text{argmin}} \sum_{i=0}^{N} (\hat{Y}_i - Y_i)^2 \tag{3}$$

where $\hat{Y}_i$ is given by eq. 1. The pseudo-code for this algorithm is,

```
for epoch 1 to 1000:
    for row in data:
        output = NN.predict(row.input)
        loss = mean_squared_error(output, row.output)
        NN.optimize(loss)
```

This process is analogous to the least squares regression method, which minimizes the square of residuals.

II. OBJECTIVES

The main objectives, as well as conclusions and inferences sought from them, are as follows. For the analysis (Sec. III and IV), we will

1) DATA PREPARATION:

- Prepare the data to extract useful information
- Convert desired data into numerical formats (quantitative and categorical variables)
- Generate scatterplots to detect possible trends in the data

2) MODEL DESIGN:

- Build a neural network regression model
- Verify the model's consistency with ordinary least squares linear regression
- Extend the model to do non-linear regression

3) STATISTICAL ANALYSIS:

- Use different linear and non-linear neural networks to fit the data
- Compare model inferences and report trends

III. PRE-ANALYSIS

In the dataset, all timestamp values were of the `string` datatype. Python's *datetime* package can be used to get numerical data as shown in Fig. 2. This gives us a **quantitative predictor**, which we call *"Start Time in Minutes* or simply *"Minutes"*, which is the number of minutes passed during the course of the day,

$$X_1 = \text{data}[\textit{"Start Time in Minutes"}] \in [0, 1440) \tag{4}$$

Conveniently, Python `datetime` objects also have the `weekday` method, which returns the day of week corresponding to the date. We use this to construct our categorical predictor. Let's define the mapping,

$$\textit{Weekday}(x) = \begin{cases} 0 & \text{if } x \text{ is on a Monday} \\ 1 & \text{if } x \text{ is on a Tuesday} \\ \vdots & \\ 6 & \text{if } x \text{ is on a Sunday} \end{cases} \tag{5}$$

Trip Start Timestamp	Fare
08/03/2019 08:15:00 PM	10.0
09/14/2019 04:30:00 PM	10.0
09/11/2019 07:15:00 AM	30.0
07/04/2019 06:00:00 PM	7.5
07/01/2019 07:15:00 PM	5.0

(a) Timestamp columns in original data

Trip Start Timestamp	Start Time in Minutes
08/03/2019 08:15:00 PM	1215
09/14/2019 04:30:00 PM	990
09/11/2019 07:15:00 AM	435
07/04/2019 06:00:00 PM	1080
07/01/2019 07:15:00 PM	1155

(b) Data extracted from timestamps

Fig. 1: Preparing the Quantitative Variable

But we cannot have $X_2 = \textit{Weekday}(x)$, as then our model would interpret weekdays as a continuous variable. This is especially a problem in linear regression - The variations from Sunday to Monday need not be the same as those from Monday to Tuesday. Rather, we use the following **categorical variables**,

$$X_i' = \begin{cases} 1 & \text{if } \textit{Weekday}(x) = \text{Weekdays}_i \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where Weekdays $= \{$Monday, ... Sunday$\}$.

Trip Start Timestamp	Weekday
08/03/2019 08:15:00 PM	5
09/14/2019 04:30:00 PM	5
09/11/2019 07:15:00 AM	2
07/04/2019 06:00:00 PM	3
07/01/2019 07:15:00 PM	0

(a) Weekdays extracted from timestamp

	Start Time in Minutes	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Total Fare per Minute
0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.500000
1	15	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.666667
2	30	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.666667
3	45	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.666667
4	60	0.0	0.0	0.0	0.0	0.0	0.0	1.0	4.666667

(b) Final ordered columns in training data

Fig. 2: Preparing the Categorical Veriables

Lastly, our quantity to be regressed and predicted is the *Total Fare per Minute*, or the total amount of money being spent on rideshare across Chicago, including tips. This is obtained by simply summing the fares for every 15-minute interval and dividing by 15[1].

[1]In the dataset, pickup and dropoff times are reported to the nearest 15 minute interval.

IV. ANALYSIS

A. Building the Regression Model

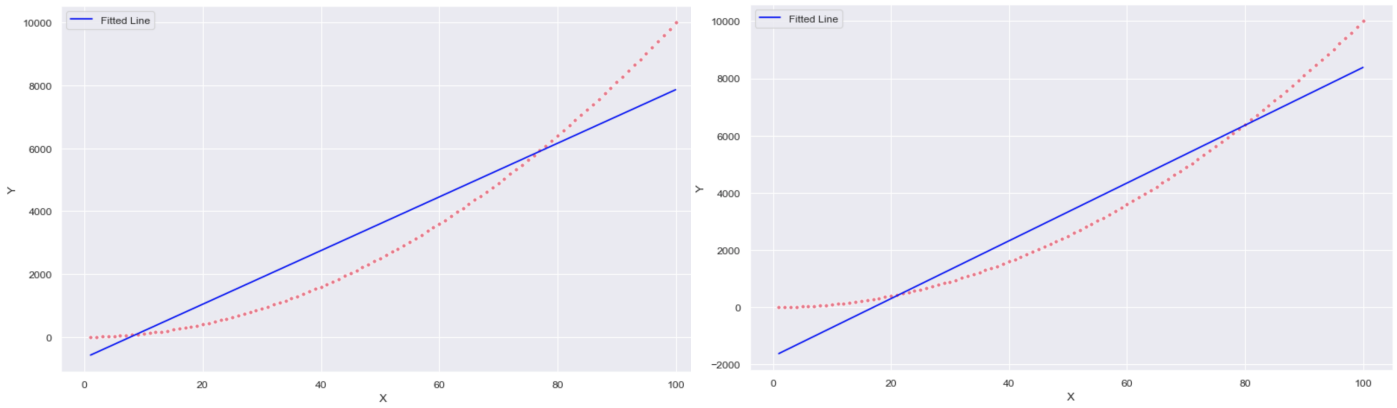
To test our regression model, we attempt to fit the function $y = x^2$ for $x \in [0, 100]$ using ordinary least squares. For unbiased LS regression,

$$\hat{Y}_h = \bar{Y} \quad \text{where } h : \hat{X}_h = \bar{X} \tag{7}$$

i.e. the regression line passes through the mean of observations. However, we find that even after sufficiently many epochs, $\hat{Y}_h$ is 3642.108, whereas $\bar{Y}$ is 3383.5. This is because unlike classical least squares regression, performance of iterative methods such as neural networks are sensitive to the relative scales of the variables. When we standardize and normalize the variables according to,

$$X_i = \frac{X_i - \mu_i}{\sigma_i} \tag{8}$$

When we use this scaling, we are able to regress the data with a line that passes through the mean, 3383.5. For the rest of this project, unless otherwise mentioned, it may be assumed that the data was scaled before fitting.



(a) Unscaled Features (b) Scaled Features

Fig. 3: Testing the regression model : $y = x^2$

B. *Scatter-plot of Fares against Time*

We first analyze a small fraction of the data, since the full dataset of 26GB takes long to process.

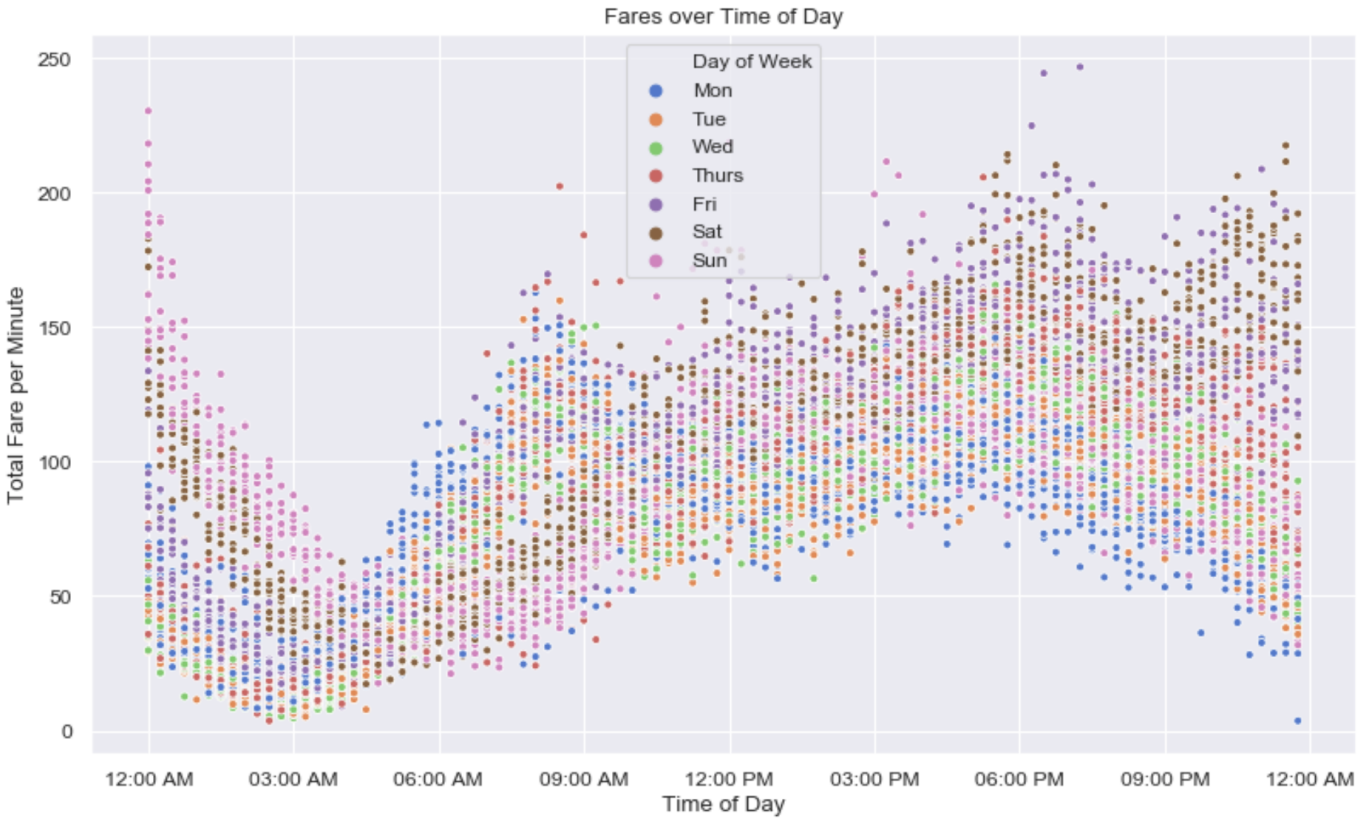


Fig. 4: Scatter Plot of 10,000 Rows

C. *Linear Regression (of 10,000 observations) with One Quantitative Predictor*

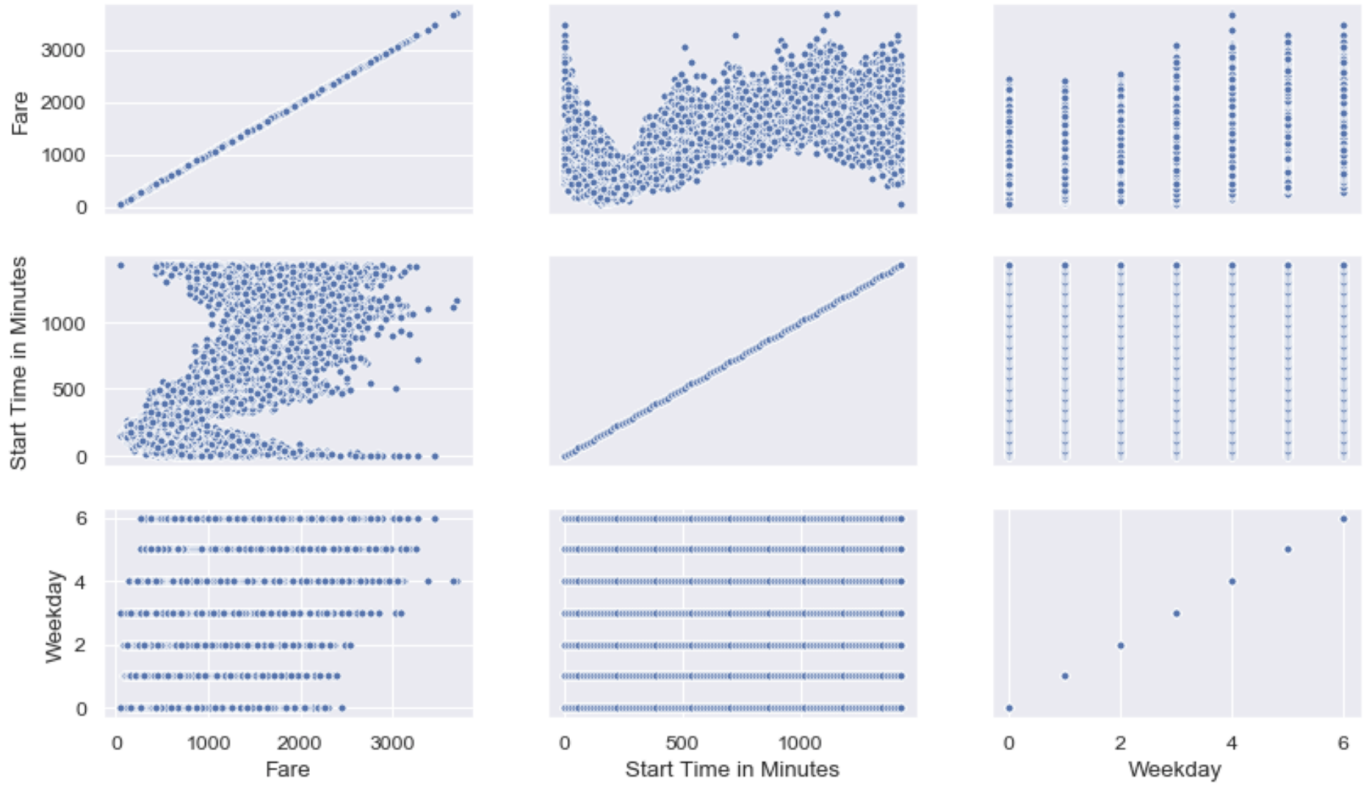
We begin by noting that the correlation matrix and individual scatterplots do not give much information. The fitted linear regression line is shown in Fig. **??**.

Once again, it may be noted that unscaled variables lead to a $\hat{Y}_h$ value of 1.3801122 whereas scaled variables lead to the true value of $\hat{Y}_h = \bar{Y} = 1.4530145$. The reason for this is evident if we look at the plot of loss defined in Eq. 2 against the training epochs (Fig. 6a and 6b). From Fig. 6d it is evident that the residuals exhibit a systematic departure from the normal distribution.

Possible resolutions for improving the QQ plot include removing outliers and including more data. We will show

	Weekday	Start Time in Minutes	Fare
Weekday	1.000000	0.000000	0.209461
Start Time in Minutes	0.000000	1.000000	0.596779
Fare	0.209461	0.596779	1.000000

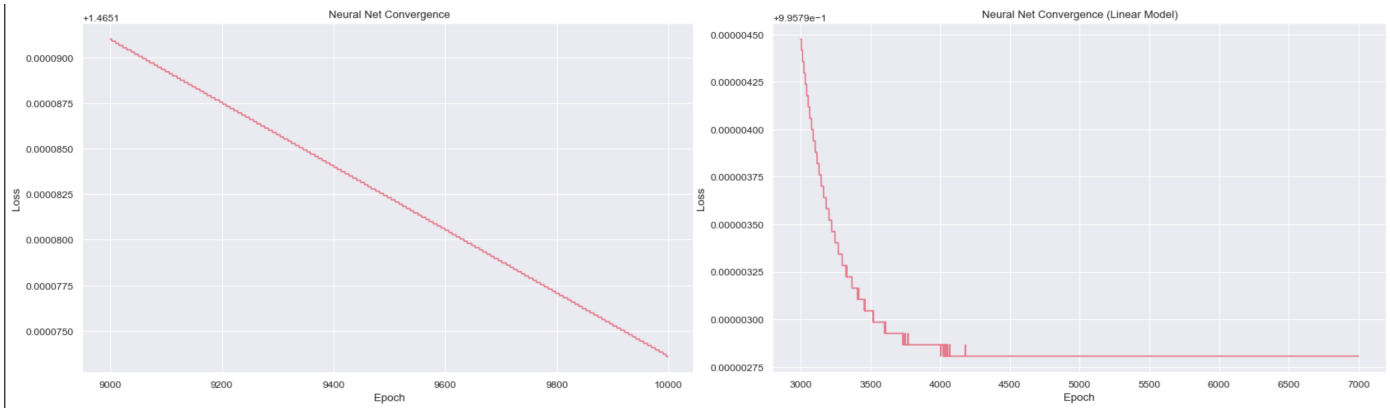
(a) Correlation Matrix



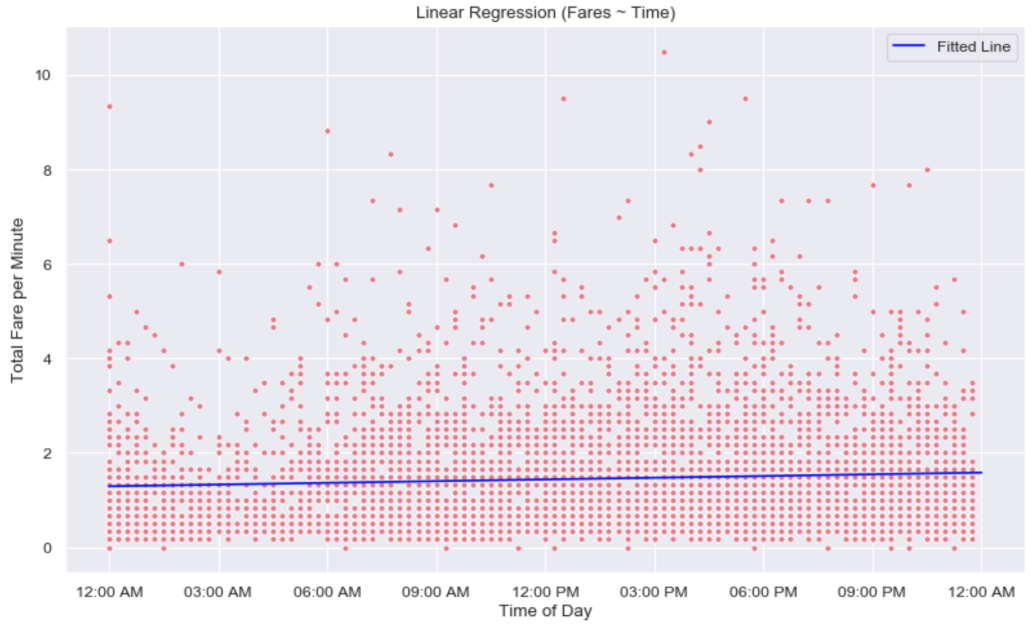
(b) Scatter Plots

Fig. 5: Correlation between Variables

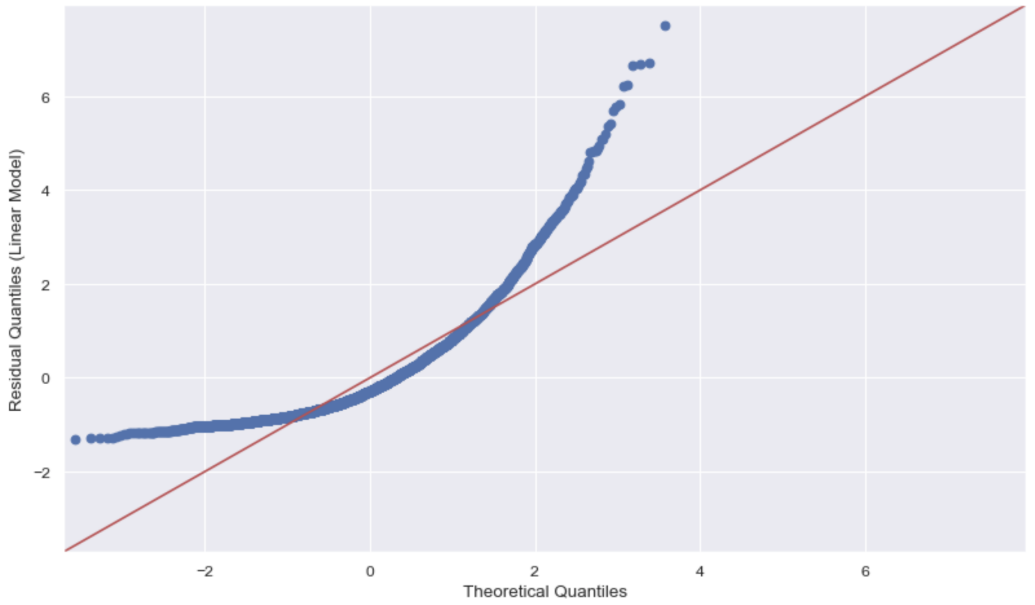
that by including more data, the QQ plot can better fit the theoretical line.



(a) Training Loss : Unscaled Features (b) Training Loss : Scaled Features



(c) Regression Line : Scaled Features

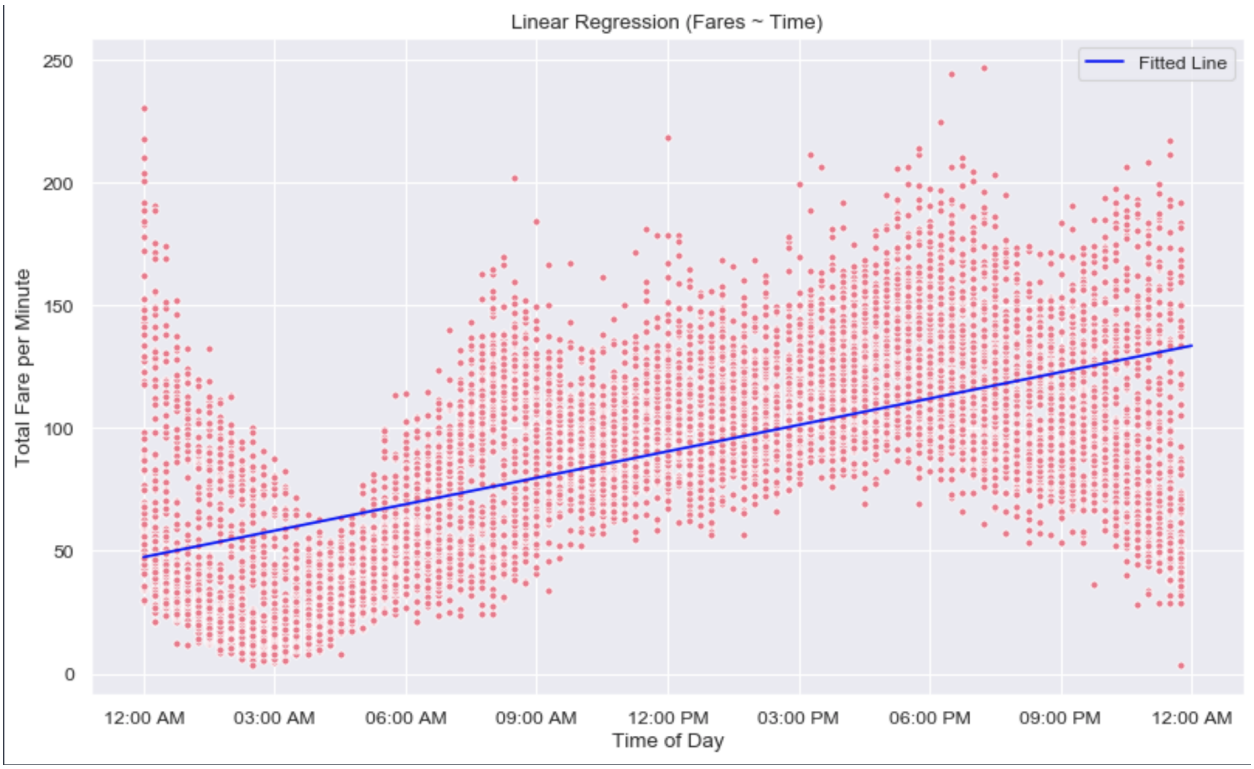


(d) Residuals against Theoretical (Normal) Quantiles

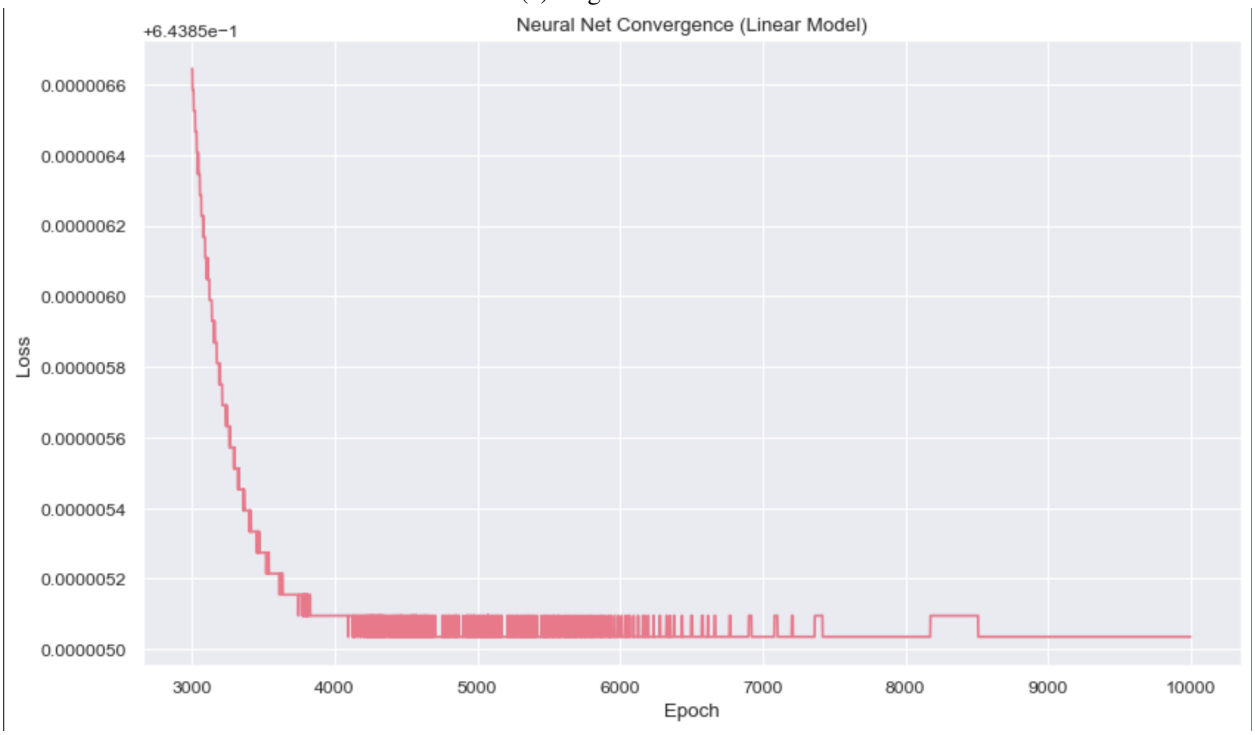
Fig. 6: Linear Regression (10,000 obs.) with One Q.V.

D. Linear Regression (of 1,000,000 observations) with One Quantitative Predictor

See Fig. 7. Note that the QQ plot now fits the theoretical quantiles better.



(a) Regression Line



(b) Loss

Fig. 7: Linear Regression (1,000,000 obs.) with One Q.V.

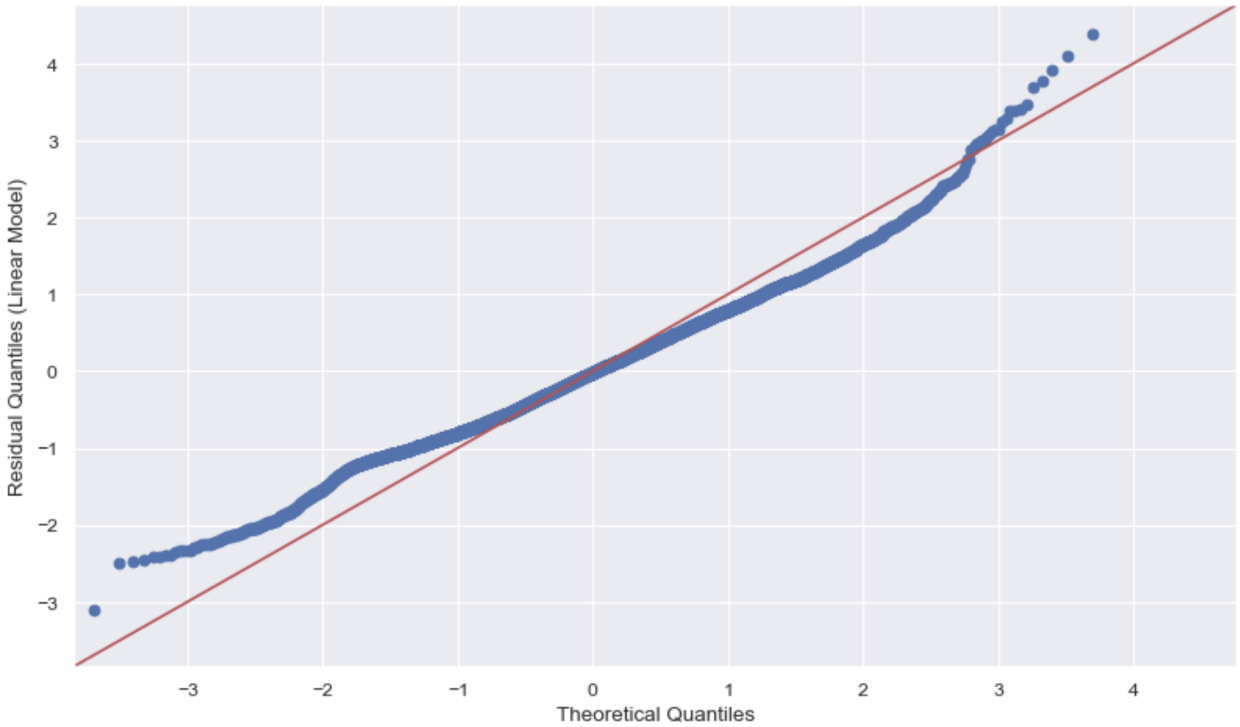
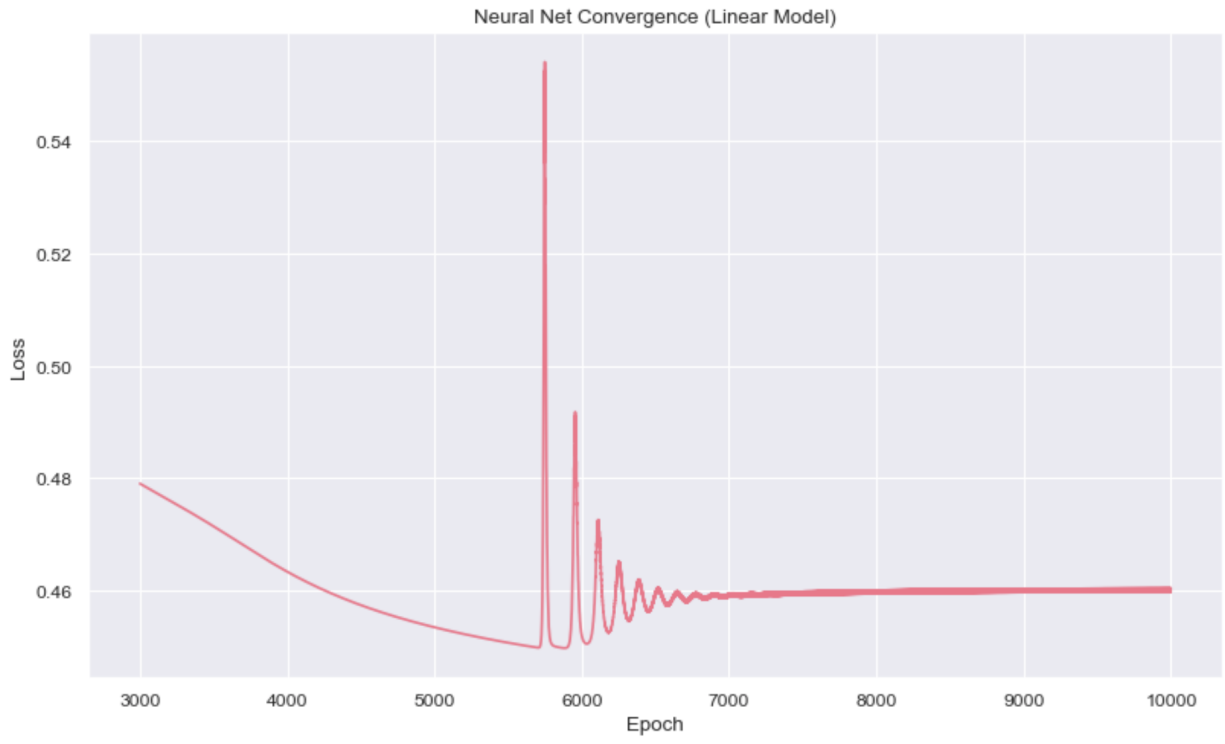


Fig. 8: QQ Plot

E. Non-Linear Regression (of 1,000,000 observations) with One Quantitative Predictor

See Fig. 9.

(a) Regression Line



(b) Loss

Fig. 9: Non-Linear Regression (1,000,000 obs.) with One Q.V.

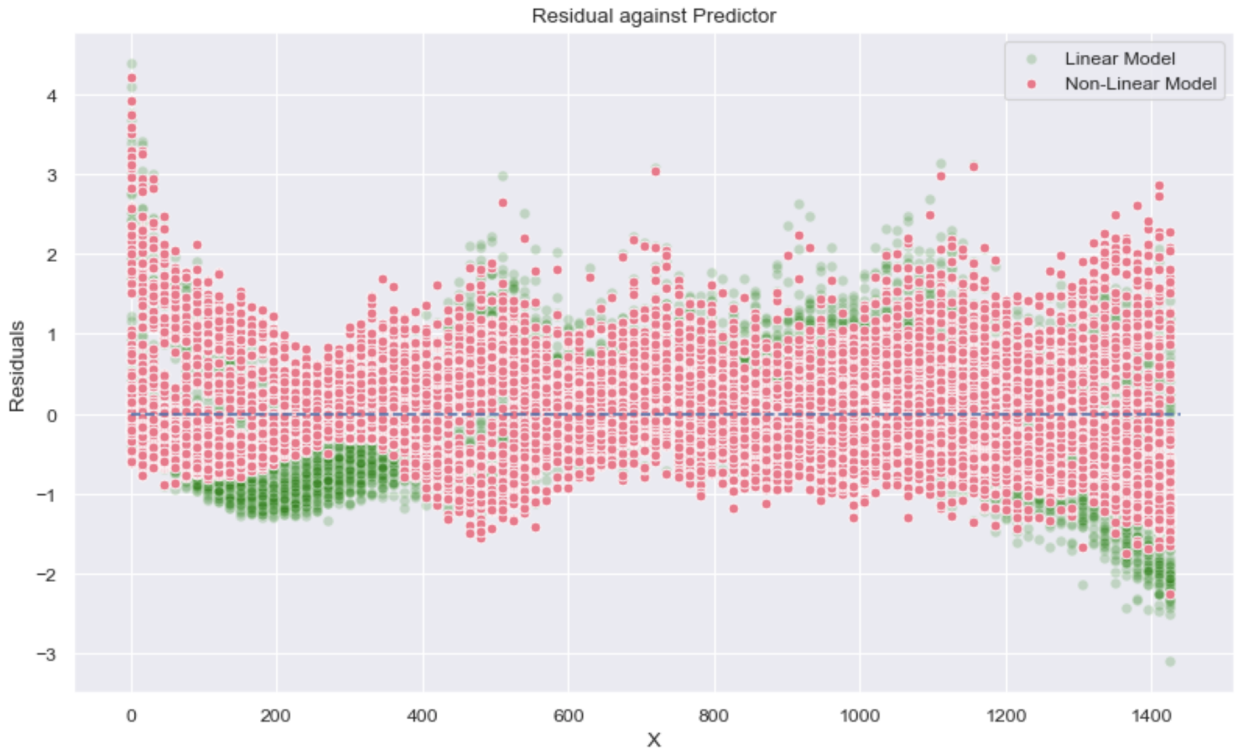
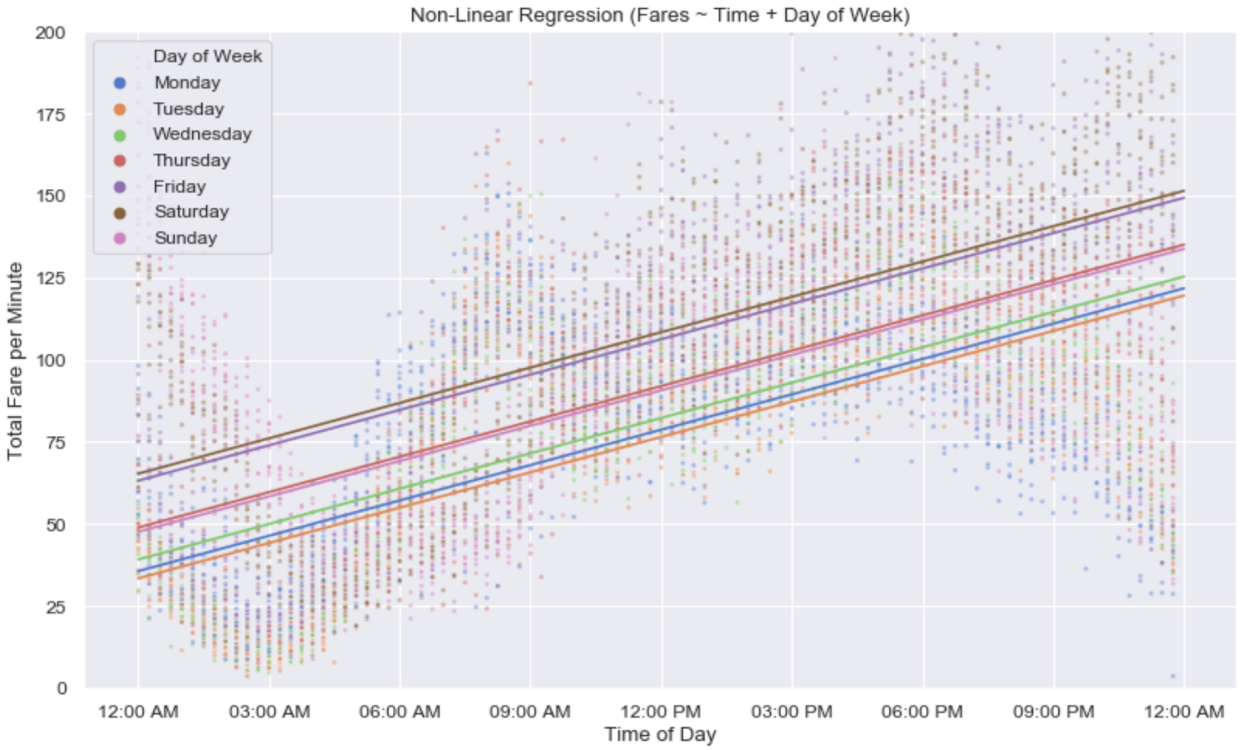


Fig. 10: Comparing Residuals

F. Linear Regression with Quantitative and Categorical Variables

See Fig. 11.

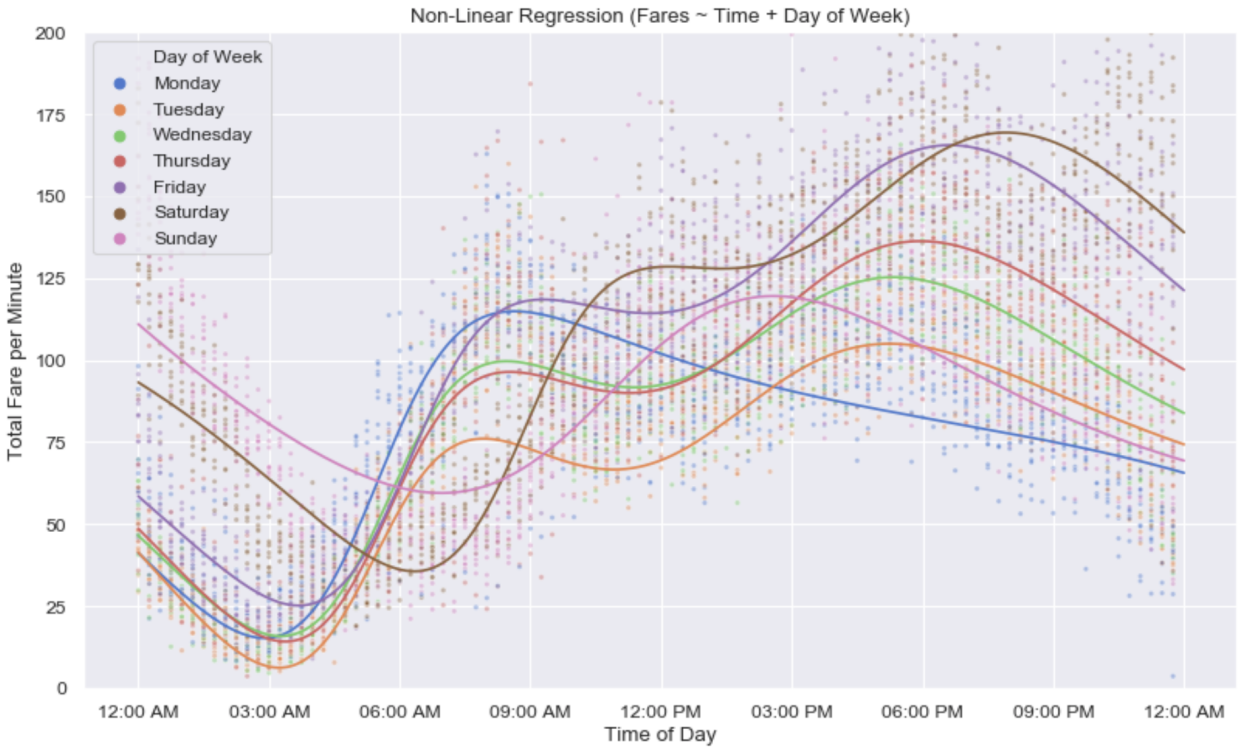


(a) Regression Lines

Fig. 11: Linear Regression with 1 QV and 7CVs

G. Non-Linear Regression with Quantitative and Categorical Variables

See Fig. 12.



(a) Regression Lines

Fig. 12: Non-Linear Regression with 1 QV and 7CVs

V. RESULTS

Some common statistical inference measures are reported below.

Stat. Measure	**Linear** (without CVs)	**Non-Linear** (without CVs)	**Linear** (with CVs)	**Non-Linear** (with CVs)
SSE	5748.3	4104.0	5040.4	1917.1
SSR	3179.7	4824.0	3887.5	7010.8
Parameters (p)	2	10	9	41
R^2	0.35614	0.5403	0.43543	0.7852

Note that the sample size is so huge, that *Adjusted* $R^2 \simeq R^2$.

VI. CONCLUSIONS

Before we introduced the categorical variables, 28.6% of the error was due to non-linearity of the trend. After we introduced the categorical variable, the non-linearity accounted for 62% of the error. This can also be noted in the increase of R^2. Since our *Adjusted* R^2 values were almost equal to R^2, we are unable to make comments regarding data-overfitting.

However, the non-linear model with categorical variables has as many as 41 predictors, so it is a fairly complex model. Moreover, the coefficients of a neural network do not have an easily interpretable meaning, i.e. in terms of slopes and intercepts. The benefit of the neural network approach is that the model can be easily extended to more predictor variables, and the non-linearity of the fitting can be varied very easily.

Another advantage of the neural network approach is that one does not need to wait for the entire dataset to be fitted, in order to use the model. Since the regression is done iteratively by going through the dataset multiple

times, the model can be switched between training and evaluation modes as desired. Optimization-based approaches also do not have the large matrix inversions that are involved in linear algebraic least-squares fitting.

VII. CODE

Note : There might be a few functions in the code which were not used while generating this report. A functional Jupyter Notebook file can be found on my github profile, 'github.com/shirazkn'.

A. FILE : pre_analysis.py

```
# General statistical analysis methods to pre-analyze the data
# such as scatter-plots, correlations...

import json
from datetime import datetime, time
import numpy as np

# The oldest date in data
DATA_START_DATE = None


# *------------------------* DATA PREPARATION *------------------------* #


def show_null_columns(data):
    """
    Prints the columns in the dataset which have Null values
    :param data: Pandas DataFrame obj
    NOTE : There might be other columns with Null-like values,
    such as a string value "Unknown".
    These will not be counted.
    """
    null_entries = {c: data.isnull().sum()[c] for c in data.keys()}
    null_entries = {key: value for key, value in null_entries.items()
                            if null_entries[key]}
    print(f"The following columns have Null values: ")
    print(json.dumps(null_entries, indent=4, sort_keys=True, default=str), "\n")


# *--------------------* DATE/TIME MANIPULATION *--------------------* #

def set_start_date(data_dates):
    global DATA_START_DATE
    DATA_START_DATE = data_dates.min()


def get_datetime_from_str(timestamp):
    """
    Returns datetime.datetime object from timestamp
    string formatted as 08/03/2019 08:15:00 PM
    """
    return datetime.strptime(timestamp, "%m/%d/%Y %I:%M:%S %p")
```

```
def get_days_in_data(dt):
    """
    Returns no. of days passed since start of data-set
    So <DATA_START_DATE, DATA_START_DATE+1, ... > are mapped to <0, 1, ... >
    """
    return (dt - DATA_START_DATE).days


def get_minutes_in_day(dt):
    """
    Returns the no. of minutes that transpired in the day
    """
    return dt.hour*60 + dt.minute


def get_day_in_week(dt):
    """
    Returns day of week, with 0: Monday and 6: Sunday
    """
    return dt.weekday()


def get_str_from_minutes(minutes):
    """
    Useful for plotting
    :param minutes: Output of get_minutes_in_day
    :return: str ("11:59 PM")
    """
    dt = time(int(np.floor(minutes / 60)), int(minutes % 60), 0)
    return time.strftime(dt, "%I:%M %p")


def change_xticks_to_time(plot, n_ticks=6):
    """
    Maps x-axis of a plot from Time (Minutes) to Time (%H:%M %p)
    """
    ticks = np.linspace(0, 1440, n_ticks)
    tick_labels = [get_str_from_minutes(t) for t in ticks[:-1]]
    tick_labels.append(tick_labels[0])
    plot.set_xticks(ticks)
    plot.set_xticklabels(tick_labels)


def minutes_in_day(points=1440):
    return np.linspace(0, 1440, points)


# *------------------------------* OTHER *------------------------------* #
```

```
def correlation_matrix(data, col_names):
    """
    Returns the correlation matrix between variables in <col_names>
    :param data: Pandas DataFrame obj
    :param col_names: Names of columns which you want to Analyze
    """
    corr_matrix = data.corr()
    for c in corr_matrix.columns:
        if c not in col_names:
            corr_matrix = corr_matrix.drop(c, axis=0).drop(c, axis=1)

    return corr_matrix
```

B. FILE : analysis.py

```
import numpy as np
import pandas as pd
import torch

from plotting import line_xy
from pre_analysis import minutes_in_day
from sklearn.preprocessing import StandardScaler


# Loss function used for training
criterion = torch.nn.MSELoss()
torch.manual_seed(0)


# *-----------------* NEURAL NETWORK DEFINITIONS *-----------------* #

# Linear Regression Model
class RegModel(torch.nn.Module):
    """
    Neural network which stores
    """

    def __init__(self, inputs, outputs, learning_rate=0.1,
                         scale_features=True, linear=True):
        super(RegModel, self).__init__()

        self.inputs = inputs
        self.outputs = outputs
        self.input_scaler = StandardScaler() if scale_features else NoScaler()
        self.output_scaler = StandardScaler() if scale_features else NoScaler()

        n_inputs = len(inputs)
        n_outputs = len(outputs)
```

```
    # Linear Regression Model
    if linear:
        self.Linear = torch.nn.Linear(n_inputs, n_outputs)
        self.Linear.weight.data.uniform_(-1, 1)
        self.forward = self.linear_forward

    # Non-linear Regression Model
    else:
                # No. of neurons in the hidden layer
        n_hidden = int(np.max([n_inputs + n_outputs - 5, 3]))
        self.Input = torch.nn.Linear(n_inputs, n_hidden)
        self.Output = torch.nn.Linear(n_hidden, n_outputs)
        self.forward = self.nonlinear_forward

    self.optimizer = torch.optim.SGD(self.parameters(), lr=learning_rate)
    self.fitted = "Not Calculated"

    self.sse = None
    self.sst = None

def linear_forward(self, x):
    x = self.Linear(x)
    return x

def nonlinear_forward(self, x):
    x = self.Input(x)
    x = torch.tanh(x)  # Non-linear Activation
    x = self.Output(x)
    return x

def fit_data(self, data, epochs=100, outliers=None):
    """
    Learn from each row in data (Pandas DataFrame)
    :param data: Training Data
    :param epochs: Training epochs (int)
    :param outliers: Indices of outlier points
    """
    self.train()
    X, Y = self.XY_from_data(data, outliers)
    losses = []
    for epoch in range(epochs):
        self.optimizer.zero_grad()
        Y_pred = self(X)
        loss = criterion(Y_pred, Y)
        loss.backward()
        self.optimizer.step()
        losses.append(loss)
    self.eval()
    return losses
```

```
def XY_from_data(self, data, outliers=None):
    """
    :param data: Pandas DataFrame
    :param outliers: Data points to be omitted from the returned lists
    :return: list(Tensor), list(Tensor)
    """
    if outliers:
        data = data.drop(data.index[outliers])

    X_array = self.input_scaler.fit_transform(
                                    np.array(data[self.inputs], dtype=float)
                                )
    Y_array = self.output_scaler.fit_transform(
                                    np.array(data[self.outputs], dtype=float)
                                )
    X = torch.from_numpy(X_array).float()
    Y = torch.from_numpy(Y_array).float()
    for x, y in zip(X, Y):
        x.requires_grad = True
        y.requires_grad = False

    return X, Y

def predict(self, x):
    """
    Uses NN to make prediction
    :param x: Float, unscaled input
    :return: Float, unscaled output
    """
    x = self.input_scaler.transform([x])[0]
    y = self(torch.tensor(x, requires_grad=False).float()).detach().numpy()
    return self.output_scaler.inverse_transform([y])[0]

def regression_line_1(self, label="Fitted Line", color="blue", **kwargs):
    """
    Plot regression line for the single predictor case
    """
    plot_data = pd.DataFrame()
    plot_data["Start Time in Minutes"] = minutes_in_day(points=4000)
    plot_data["Total Fare per Minute"] = [self.predict([x])[0]
                                for x in plot_data["Start Time in Minutes"]]
    return line_xy(x="Start Time in Minutes", y="Total Fare per Minute",
                           data=plot_data,
                    label=label, color=color, **kwargs)

def regression_lines_2(self, weekdays, palette, **kwargs):
    """
    Plot regression line for the multiple predictors case
    """
    plot_data = pd.DataFrame()
```

```
        plot_data["Start Time in Minutes"] = minutes_in_day(points=4000)
        for weekday, color in zip(weekdays, palette):
            inputs = [[min, 0, 0, 0, 0, 0, 0, 0]
                              for min in plot_data["Start Time in Minutes"]]
            for x in inputs:
                x[weekday + 1] = 1

            plot_data["Total Fare per Minute"] = [self.predict(x)[0]
                                                  for x in inputs]
            line_xy(x="Start Time in Minutes", y="Total Fare per Minute",
                           data=plot_data, color=color, **kwargs)

    def compute_residuals(self, data, outliers=None):
        self.eval()
        X, Y = self.XY_from_data(data, outliers)
        Y_pred = self(X)

        self.fitted = data.copy()
        if outliers:
            self.fitted = data.drop(data.index[outliers])

        self.fitted["Predictions"] = [t.detach().numpy()[0] for t in Y_pred]
        self.fitted["Residuals"] = [t.detach().numpy()[0]*(-1)
                                    for t in Y_pred - Y]
        self.fitted["Residuals_Sq"] = [t**2 for t in self.fitted["Residuals"]]
        self.sse = self.fitted["Residuals_Sq"].sum()
        total_errors = np.array([t.detach().numpy()[0]
                                 - Y.mean().detach().numpy() for t in Y])
        self.sst = np.sum(total_errors**2)

    def get_outliers_from_residuals(self, limits):
            # Optionally, outliers can be removed from the dataset.
        # Was not used for this report.
        indices = []
        for i, e in enumerate(self.fitted["Residuals"]):
            if e < limits[0] or e > limits[1]:
                indices.append(i)
        return indices


class NoScaler:
    """
    Dummy 'scaler' class, just does identity transformation for all elements
    Using this to compare effect of scaling variables for neural nets
    """
    def __init__(self):
        pass

    def fit_transform(self, x):
        return x
```

```
    def transform(self, x):
        return x

    def inverse_transform(self, x):
        return x
```

C. FILE : plotting.py

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
colors = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
sns.palplot(sns.color_palette(colors))


# *------------------------* PLOTTING FUNCTIONS *------------------------* #

def line_xy(x, y, **kwargs):
    return sns.lineplot(x, y, **kwargs)


def scatter_xy(data, x: str, y: str, axis=None, position=None, **kwargs):
    """
    Makes a scatter-plot between data[x] and data[y]
    :param data: Pandas DataFrame
    :param x: str
    :param y: str
    :param axis: Used by func. scatter_grid
    :param position: Used by func. scatter_grid
    :return plt
    """
    sns.set_palette("husl", 3)
    plot_obj = sns.scatterplot(x=x, y=y, data=data, ax=axis, **kwargs)
    axis.set_xlabel(x) if axis else None
    axis.set_ylabel(y) if axis else None

    # <position> is used for generating nice-looking subplots
    if not position:
        plot_obj.set_title(f"{y} against {x}")

    elif axis:
        if position == "inner":
            axis.xaxis.label.set_visible(False)
            axis.yaxis.label.set_visible(False)
            axis.set_xticks([])
            axis.set_yticks([])

        elif position == "left":
```

```
            axis.xaxis.label.set_visible(False)
            axis.set_xticks([])

        elif position == "bottom":
            axis.yaxis.label.set_visible(False)
            axis.set_yticks([])

        elif position == "corner":
            pass

    return plot_obj


def scatter_grid(data, col_names, **kwargs):
    """
    Makes a pairwise correlation grid as subplots
    :param data: DataFrame
    :param col_names: list [str]
    """
    fig, axes = plt.subplots(nrows=len(col_names), ncols=len(col_names))
    axis = [0, 0]
    for v1 in col_names:
        axis[1] = 0
        for v2 in col_names:
            position = "inner"
            if axis[0] == len(col_names) - 1:
                position = "bottom" if axis[1] else "corner"
            elif axis[1] == 0:
                position = "left"
            scatter_xy(data, y=v1, x=v2, axis=axes[axis[0], axis[1]],
                               position=position, **kwargs)
            axis[1] += 1
        axis[0] += 1


def change_labels(plot_obj, labels):
    """
    Change labels of plot
    len(._legend.texts) == len(labels)
    """
    for text, label in zip(plot_obj.legend_.texts, labels):
        text.set_text(label)

# ------------------------------------------------------------------------ #
```

D. FILE : notebook.ipynb

```
from analysis import *
from pre_analysis import *
from plotting import *
...
```

REFERENCES

[1] "Rideshare data published by City of Chicago," https://data.cityofchicago.org/Transportation/Transportation-Network-Providers-Trips/m6dm-c72p, accessed: 2019-11-10.