Data Science Salary Forecaster

Rich Kirschenheiter, Stefany Lima, Timothy Kelley,
Sabyasachi Das, James Bruning, & Arielle Eagan

As future Data Science and STEM professionals, graduates of the NU Data Science Bootcamp are curious about the ***job market*** and ***earning potential*** in new roles available after the Bootcamp….

And for those roles, what **salaries** can we ask for?

As an applicant, salary negotiations are daunting.

No one wants to go into negotiations without a sense of what others are being paid and what they can ask for.

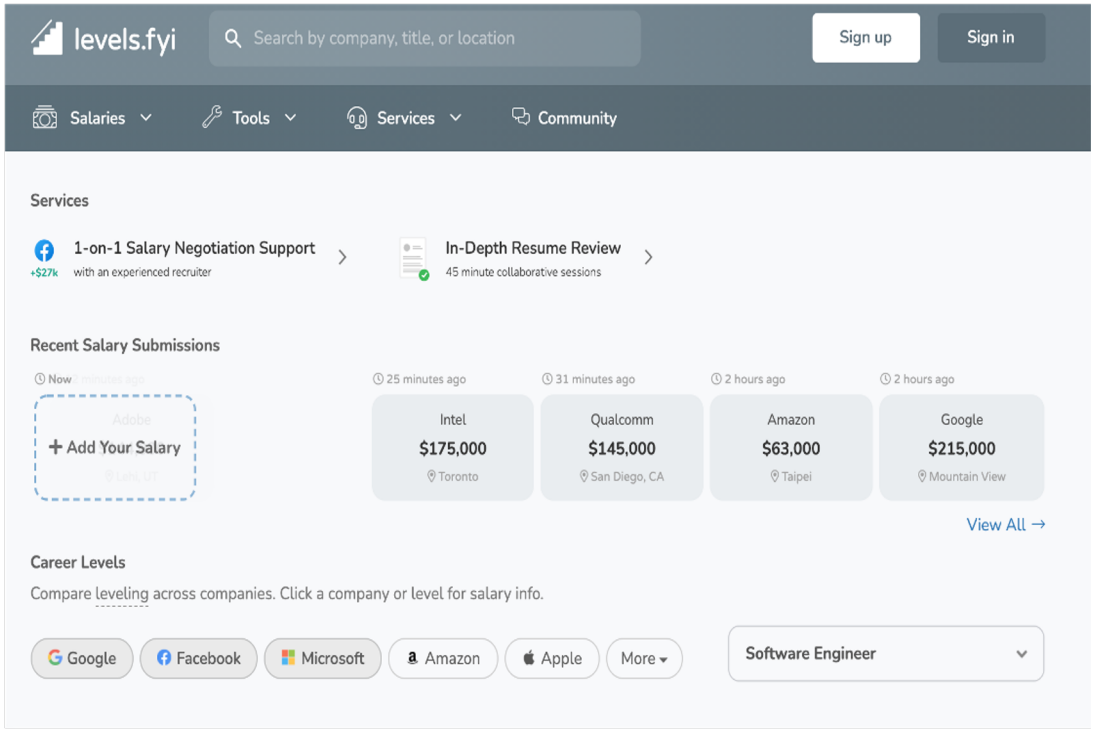
So, we created a

Data Science Salary Forecaster

We started with our existing interactive online tool, **which maps Data Science salaries across 5 leading companies.**


Salaries by Experience and Company
Apple
Amazon
Facebook
Google
Microsoft
600
550
500
450
400
350
300
250
200
150
100
Salary in $1,000
0
5
10
15
20
25
Years of Experience

Data source of our interactive tool:



Data scraped from a site called levels.fyi : https://www.levels.fyi/?compare=Google,Facebook,Microsoft&track=Software%20Engineer

API in JSON format allowed us to pull more updated data, running only about 6 days behind current present day.

Data set included:

- company name
- job title, location (city and state)
- compensation
  - total yearly compensation
  - base salary
  - bonus
  - stock grants
- demographics

Original data cleaning process for our interactive tool:

```
In [16]: #exploring title data
         salary_df['title'].value_counts()

Out[16]: Software Engineer                41231
         Product Manager                   4673
         Software Engineering Manager      3569
         Data Scientist                    2578
         Hardware Engineer                 2200
         Product Designer                  1516
         Technical Program Manager         1381
         Solution Architect                1157
         Management Consultant              976
         Business Analyst                   885
         Marketing                          710
         Mechanical Engineer                490
         Sales                              461
         Recruiter                          451
         Human Resources                    364
         Name: title, dtype: int64
```

Steps we took to clean the data:

- Analyzed data first
- Dropped non-relevant columns (ex: kept city name but dropped ”cityID”
- Began to analyze salary data
- Began to analyze different companies
- Began to interpret “levels” of roles at each company
- Created bins for “entry”, “mid”, and “senior” levels

Original python script for interactive tool

```python
#Import dependencies
import pandas as pd
import requests

#API pull for latest salary info from www.levels.fyi
salaryData = requests.get('https://www.levels.fyi/js/salaryData.json').json()
salary_df = pd.DataFrame(salaryData)

#dropping columns that are not relevant to project
salary_df = salary_df.drop(['cityid', 'dmaid','rowNumber','otherdetails','tag', 'basesalary', 'stockgrantvalue', 'bonus', 'gender

#converting to float to allow for summary stats
salary_df["totalyearlycompensation"] = pd.to_numeric(salary_df["totalyearlycompensation"])
salary_df["yearsofexperience"] = pd.to_numeric(salary_df["yearsofexperience"])
salary_df["yearsatcompany"] = pd.to_numeric(salary_df["yearsatcompany"])

#coverting timestamp from object to datetime
salary_df['timestamp'] =  pd.to_datetime(salary_df['timestamp'], infer_datetime_format=True)

# Create separate cols for city, state and country
def split_location(location):
    items = location.split(', ')
    city = items[0]
    state = items[1]

    if len(items)==2:
        country = 'US'
    elif len(items)==3:
        country = items[2].strip()
    elif len(items)==4:
        country = ', '.join([i.strip() for i in items[2:]])
    else:
        country = None
        print(location)
```

We used Chart.js library to create our interactive tool

Building off of this interactive tool, we created a machine learning predictive model to predict Data Science salaries.

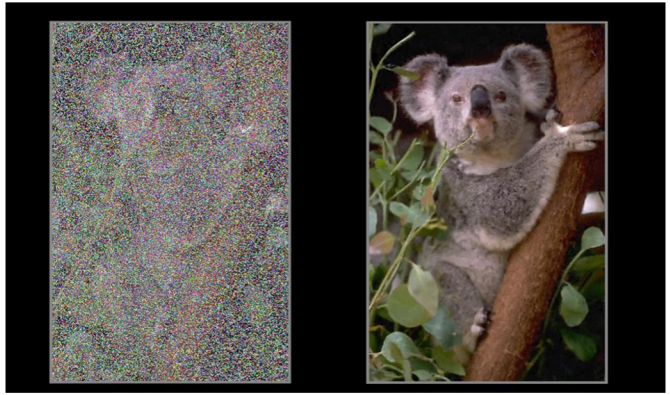
Steps

Cleaning	Feature Engineering	Model Selection	Hyperparameter Tuning	Final Model / Demo
Reduce noise by converting levels to numeric values, resolved null value issues, and mismatched formatting.	Determined what features to use. Started with five but our absolute mean error was too high to be useable. Added two more features which dramatically improved our model's accuracy.	Deciding which models, Neural Net or Random Forest, best suit our data.	Running models multiple times to find best results with the best parameters possible, using .	Combining original features and predicted features as inputs into final model for prediction. Final model demo.

Data Cleaning

Even though data had already been cleaned, we went through it once again to make it accessible to models used later on.

Focus on reducing noise



In contrast to our original project, where we only used data from the top 5 companies, we expanded to include all companies in dataset.

- Removed blank spaces and capitalization
- Assigned level numbers across companies for similar jobs

```
salary_df['level'] = salary_df.level.apply(lambda x: x.lower() )
salary_df['level'] = salary_df.level.apply(lambda x: x.strip() )
```

```
In [15]:
def clean_string(x):
    try:
        num_list = [i for i in x if i.isdigit()]
        if len(num_list)<1:
            return None
        else:
            return int(''.join(num_list))
    except:
        None
```

```
In [16]:
us_df.level = us_df.level.apply(lambda x: clean_string(x))
```

Reduced noise in the levels data by as much data as we could to a numeric value.

In future iterations, we would benefit from a domain expert to confirm numeric levels assigned are proportionate to job title.

In [3]:

```
#normalize formating inconsistency in levels
us_df.level.replace('senior associate', '4', inplace=True)
us_df.level.replace('principal associate', '5', inplace=True)
us_df.level.replace('manager', '5', inplace=True)
us_df.level.replace('director', '7', inplace=True)
us_df.level.replace('senior manager', '6', inplace=True)
us_df.level.replace('senior', '5', inplace=True)
us_df.level.replace('senior software engineer', '6', inplace=True)
us_df.level.replace('senior data scientist', '5', inplace=True)
us_df.level.replace('data scientist', '4', inplace=True)
us_df.level.replace('associate', '4', inplace=True)
us_df.level.replace('senior consultant', '5', inplace=True)
us_df.level.replace('lead associate', '5', inplace=True)
us_df.level.replace('staff', '4', inplace=True)
us_df.level.replace('staff software engineer', '6', inplace=True)
us_df.level.replace('group manager', '6', inplace=True)
us_df.level.replace('analyst', '4', inplace=True)
us_df.level.replace('vice president', '8', inplace=True)
us_df.level.replace('senior consultant', '5', inplace=True)
us_df.level.replace('consultant', '4', inplace=True)
us_df.level.replace('executive director / vice-president', '8', inplace=True)
us_df.level.replace('senior mts', '6', inplace=True)
us_df.level.replace('lead mts', '5', inplace=True)
us_df.level.replace('ii', '5', inplace=True)
us_df.level.replace('data scientist ii', '5', inplace=True)
us_df.level.replace('data scientist i', '4', inplace=True)
us_df.level.replace('only one level across netflix', '5', inplace=True)
```

```
In [19]: us_df['stockgrantvalue'] = us_df['stockgrantvalue'].fillna(0)
         us_df['bonus'] = us_df['bonus'].fillna(0)

In [20]: us_df[us_df.level.isna()]
```

Out[20]:

	timestamp	company	level	title	totalyearlycompensation	yearsofexperience	yearsatcompany	stockgrantvalue	bonus	city	state	cou
897	2018-06-25 08:45:29	Tesla	NaN	Data Scientist	168.0	8.0	3.0	50.0	0.0	Palo Alto	CA	
1150	2018-08-03 15:28:13	Qualcomm	NaN	Data Scientist	220.0	6.0	6.0	45.0	20.0	San Diego	CA	
1297	2018-08-15 06:10:00	Amazon	NaN	Data Scientist	200.0	5.0	1.0	0.0	0.0	Boston	MA	
1301	2018-08-15 11:57:44	Booz Allen Hamilton	NaN	Data Scientist	120.0	1.0	0.0	0.0	0.0	Washington	DC	

- Replaced missing values with 0.
- Dropped NaN levels that didn't fit our cleaning loop.
- Corrected inconsistent data formatting.

```
In [32]: x=[]
         y=[]
         for idx, row in us_df.iterrows():
             if row['stockgrantvalue']>row['totalyearlycompensation']:
                 x.append(row['stockgrantvalue']/1000)
             else:
                 x.append(row['stockgrantvalue'])

             if row['bonus']>row['totalyearlycompensation']:
                 y.append(row['bonus']/1000)
             else:
                 y.append(row['bonus'])

         us_df['bonus']=y
         us_df['stockgrantvalue']=x
```

Feature Selection

We had a variety of avenues we could try and approach with our feature selection

- Location
- Company specific
- Experience
- Country of Origin
- Title

6 Main Features

1. Years at company
2. Years of experience
3. Level at company
4. Distance from 3 major cities
5. Bonus
6. Equity

In [15]:

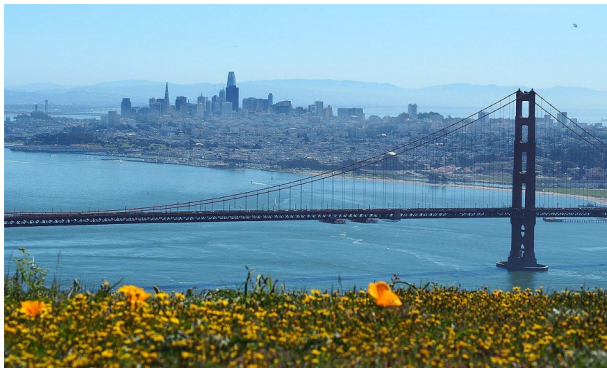
```
base_url = "https://maps.googleapis.com/maps/api/distancematrix/json?"
distances=[]
for i in range(len(city_states)):
    origin_city=city_states[i]

    destination_city="Seattle%20CWA"


    try:

        url=base_url+"origins="+origin_city+"&destinations="+destination_city+"%20C"+"&key="+g_key

        payload={}
        headers = {}

        response = requests.request("GET", url, headers=headers, data=payload)

        x=response.json()

        y=x["rows"][0]["elements"][0]["distance"]["value"]

        distances.append(y)

    except:
        distances.append('None')
        print(city_states[i])
```

We decided to focus on 3 major cities for Data Science jobs in the US.

Seattle, WA

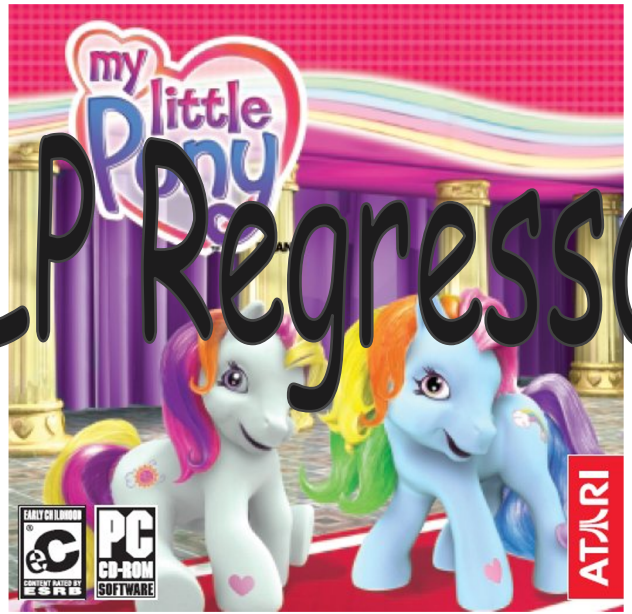
New York City, NY

San Francisco, CA



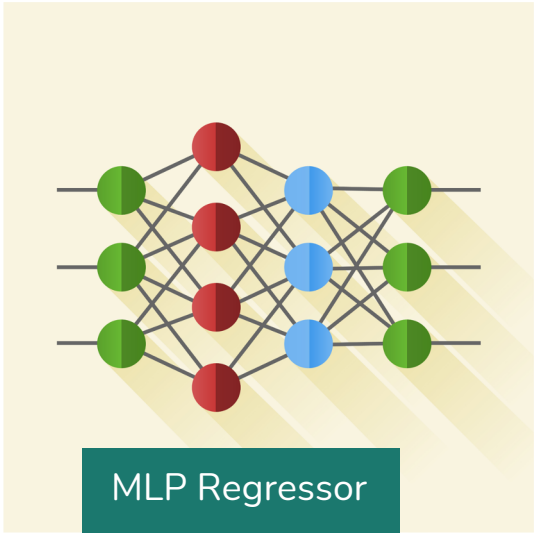
Model Selection and Structure

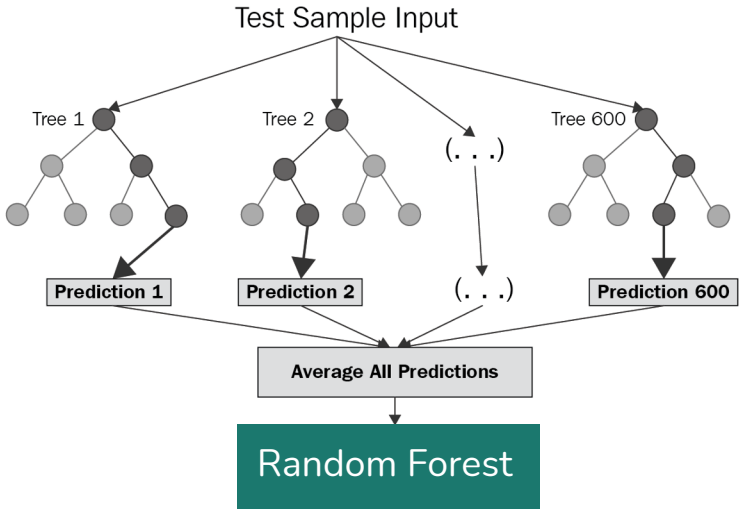
First Step: Trial and Error



MLP Regressor

First Step: Trial and Error





Variables

MLPRegressor:

```
{'solver': 'adam',
 'max_iter': 2000,
 'learning_rate': 'adaptive',
 'hidden_layer_sizes': (100,),
 'alpha': 0.01,
 'activation': 'relu'}
```

Random Forest:

- max_depth

```
{'min_samples_leaf': 1, 'max_depth': 500}
```

- min_samples_leaf

Second Step: Adding Predictions (and features)

Original model:

- Would take in 3-4 data points
  - Job Level
  - Job Location
  - Years of Experience
  - Years at Company (optional)
- Would return a total yearly compensation

Potential upgraded model:

- Would take in same 3-4 features
  - Level, Location, Experience and Seniority
- Would return a total yearly compensation, and expectations for bonuses and equity or stock options.

Hyper-Parameters Tuning:

We used Randomized Search CV to determine the best parameters for our model in order to have the lowest possible mean absolute error for each model.

In [23]:
```
reg1 = MLPRegressor(hidden_layer_sizes=(200,100,50),activation="relu" ,
                    random_state=42, learning_rate='adaptive', max_iter=2000, solver='adam', early_stopping=True)
```

In [24]:
```
from sklearn.model_selection import RandomizedSearchCV

rs = RandomizedSearchCV(reg1, param_distributions = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.01, 0.001, 0.0001, .1, 0.05],
    'learning_rate': ['constant','adaptive'],
    'max_iter': [1000,2000]},
    random_state=42
)

result = rs.fit(X_trainscaled, y_train)
```

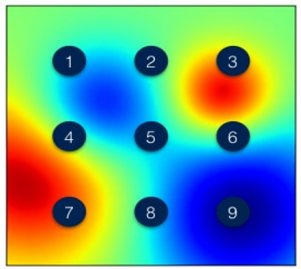
In [28]:
```
best_params = result.best_params_
```

In [29]:
```
best_params
```

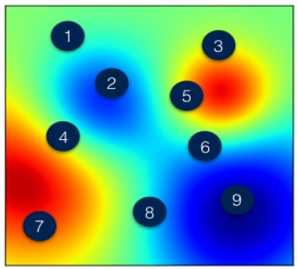
Out[29]:
```
{'solver': 'adam',
 'max_iter': 2000,
 'learning_rate': 'adaptive',
 'hidden_layer_sizes': (100,),
 'alpha': 0.01,
 'activation': 'relu'}
```

In [30]:
```
y_test_preds = rs.predict(X_testscaled)
mean_absolute_error(y_test_preds, y_test)
```

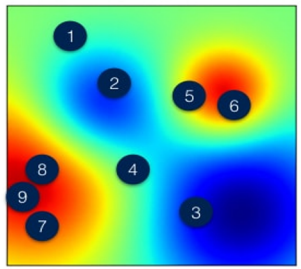
Out[30]: 26.628635055208914



Grid Search



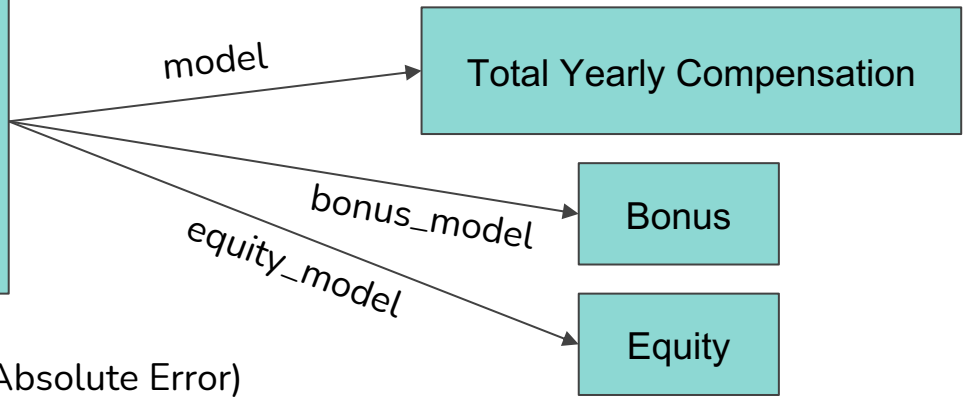
Random Search



Adaptive Selection

Neural Network Models Built

Features:
1. Years at Company
2. Years of Experience
3. Level
4. Distance of Job location from NYC
5. Distance of Job location from SF
6. Distance of Job location from Seattle



Model Accuracy (calculated in terms of Mean Absolute Error)

Model for	Baseline Error of Means for Model	MAE with Random Forest	MAE with Neural Network	MAE with Neural Network and hyperparameter tuning
Total Yearly Compensation	75.23	37.46	27.06	26.63
Bonus	69.01		11.70	11.31
Equity	594.50		39.39	38.25

Saving the models and the scalars

```
from pickle import dump
model = 'model_artifacts/model.pkl'
scalar = 'model_artifacts/scalar.pkl'
dump(result, open(model, 'wb'))
dump(sc_X, open(scalar, 'wb'))
```

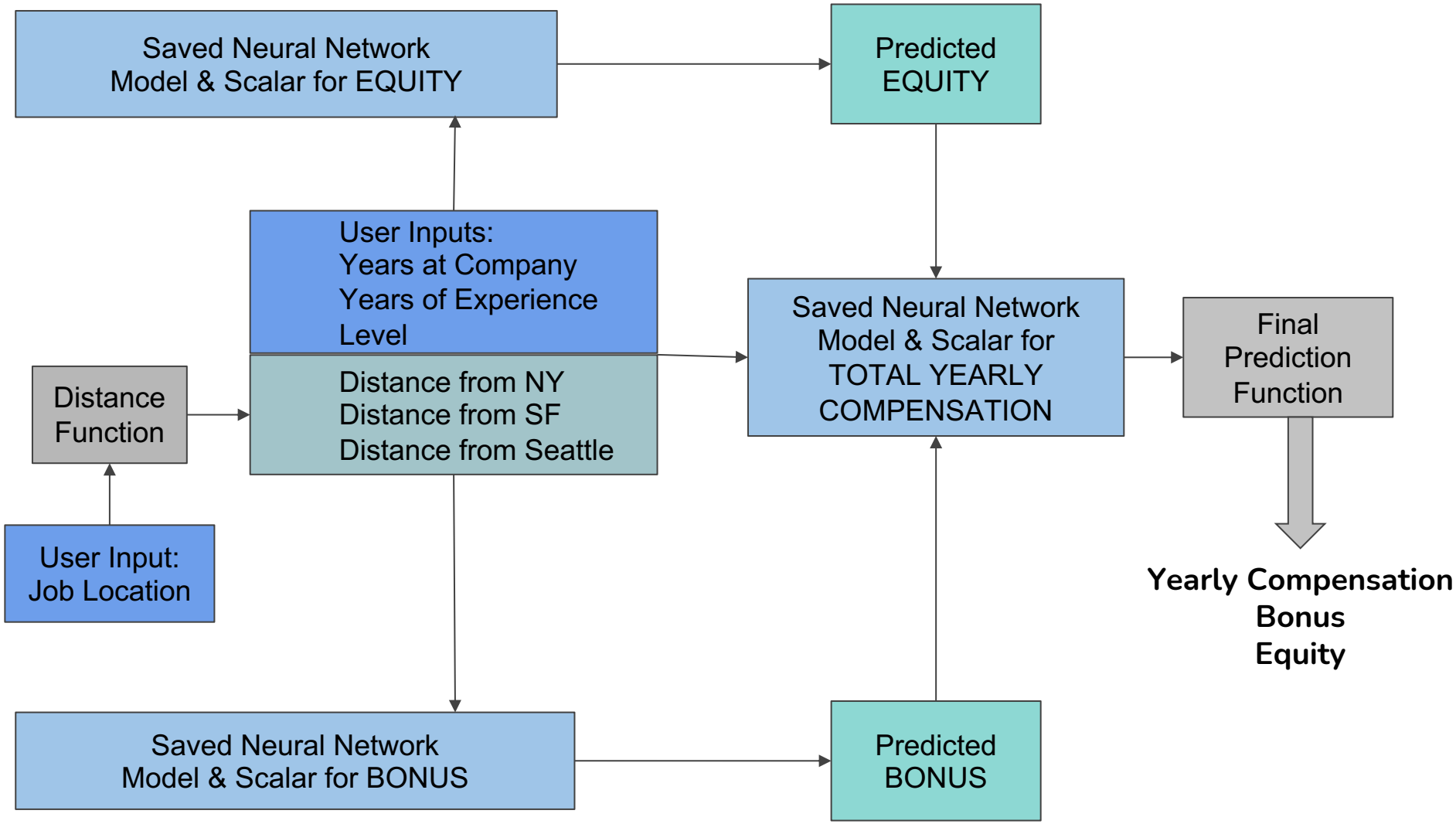
```
from pickle import dump
bonus_model = 'model_artifacts/bonus_model.pkl'
bonus_scalar = 'model_artifacts/bonus_scalar.pkl'
dump(result, open(bonus_model, 'wb'))
dump(sc_X, open(bonus_scalar, 'wb'))
```

```
from pickle import dump
equity_model = 'model_artifacts/equity_model.pkl'
equity_scalar = 'model_artifacts/equity_scalar.pkl'
dump(result, open(equity_model, 'wb'))
dump(sc_X, open(equity_scalar, 'wb'))
```

Loading the trained models and the scalars into the final prediction notebook

```
model = load(open('model_artifacts/model.pkl', 'rb'))
scalar = load(open('model_artifacts/scalar.pkl', 'rb'))

bonus_model = load(open('model_artifacts/bonus_model.pkl', 'rb'))
bonus_scalar = load(open('model_artifacts/bonus_scalar.pkl', 'rb'))

equity_model = load(open('model_artifacts/equity_model.pkl', 'rb'))
equity_scalar = load(open('model_artifacts/equity_scalar.pkl', 'rb'))
```

The models can now be used to predict total yearly compensation, bonus and equity for a set of user inputs


Saved Neural Network
Model & Scalar for EQUITY
Predicted
EQUITY
User Inputs:
Years at Company
Years of Experience
Level
Distance from NY
Distance from SF
Distance from Seattle
Distance
Function
User Input:
Job Location
Saved Neural Network
Model & Scalar for
TOTAL YEARLY
COMPENSATION
Final
Prediction
Function
Yearly Compensation
Bonus
Equity
Saved Neural Network
Model & Scalar for BONUS
Predicted
BONUS

Ready for some predictions?

Conclusion

Limitations, lessons learned, and next steps

Limitations:

- Data seems to have been collected via a survey with write-in sections, which allowed people to fill in data points themselves. A survey with drop-down options would have helped input data more cleanly and would have helped us feel more confident in our binning of entry, mid, and senior level positions at companies.
- Own our skills - we're still learning!

Lessons learned:

- Involve experts in the content area to ensure that data represents the truth accurately
    - Ex: have experts in content area on team, conduct focus groups, conduct market research
- Work as a team - it's helps encourage creativity, encouragement, and to catch errors in code
- We are gaining valuable (literally $$) skills in this Data Science Bootcamp

Next steps:

- Continued testing of the model
- Using the model to help us negotiate our own future jobs! We hope you use it too :)

Thank you for our instructional team for their support in this project!