NFT Trends, Project Report of ELEN 6883

Group member: Victoria Fang (yf2647), Yusheng Chen (yc4157), Tianyu Qin(tq2155), Mingyang Song (ms6525), Ruoyu Chen(rc3506), Xueji Zhao(xz3159)
Department of Electrical Engineering
Columbia University
New York, 10027

I. Introduction

NFT stands for non-fungible token, a unique digital certificate of ownership for any assigned digital asset stored on the blockchain. It matches each item with a tamper-proof ID number, making it impossible to generate identical digital works. NFT has 10 popular use cases, including music, gaming, art, and fashion[1]. It initially made its debut in gaming, providing ownership verification and authentication of game items. NFTs have gained traction as a viable option for artists to secure their rights to digital art, curbing piracy and counterfeiting. Tokenizing digital art streamlines the process of creating, trading, and earning royalties. There are multiple NFT market and trading platforms available, such as Opensea, Nifty Gateway, Rabile, and NBA Top Shot.

OpenSea is the largest NFT trading platform, with 2.4 million active users and a daily trading volume of $6.03 million as of November 2022[2]. It runs on the Ethereum blockchain and allows creators to mint, buy, sell, and trade digital assets such as collectibles, game items, and art. OpenSea's secure and user-friendly platform enables creators to monetize their digital assets and buyers to access a vast array of verified NFTs, highlighting the increasing demand for NFTs and blockchain's potential to transform digital asset trading.

IPFS, or InterPlanetary File System, is a P2P distributed file system proposed by Protocol Lab. It is a protocol and network for storing and sharing files using content-addressing to uniquely identify each file in a global namespace. The goal of IPFS is to replace the location-based HTTP protocol, which has drawbacks such as low performance efficiency and reliance on central servers and backbone networks. While HTTP has benefits such as simplicity and a wide application, the proposal of IPFS highlights the potential for decentralized file sharing and a more efficient and resilient web infrastructure.

However, for IPFS, it provides a multi-centralized solution. Instead of relying on domain names and IPs for content addressing, it uses unique HASH keys to facilitate data search. IPFS serves as a distributed file storage system where file data is not stored in a single centralized server but is rather distributed across all suitable computers on the network[3]. For HTTP in terms of security, whether it is physical or personnel problems, data will be deleted or lost. The cost of IPFS distributed storage, especially in the initial stage, can be reduced to about 1/10 of the cost of centralization. More importantly, because it is a multi-point backup mechanism, there is no possibility of loss, unless all nodes in the IPFS ecosystem are down, but this situation is not realistic.[4]

HTTP	IPFS
Data is not persistent in HTTP Data persistent in IPFS	Data persistent in IPFS
It uses a centralized client server approach It uses a decentralized peer to peer approach	It uses a decentralized peer to peer approach
Data is requested using the address on which data is hosted	Data is requested using the cryptographic hash of that data
One has to set up a hosting server or pay for one, in order to make content publicly available	Uploading content on the IPFS network does not require a host server, every node hosts the data on the network

Figure 1. Difference between HTTP and IPFS

II. Summary of our work

A. Data Collection

To analyze the trend of NFTs, we have pulled a large amount of data through the NFTPort API for analysis purposes.

The Python script called Data_Collection is used to pull the top selling NFT contracts in the last 24 hours, 7 days, 30 days from the NFTport API. It retrieves the sale statistics for each contract and the metadata of the NFTs within them. The script sends HTTP GET requests to the API endpoints for contract statistics and metadata, using the contract addresses obtained from the top selling contract API endpoint. The obtained JSON data is then dumped into separate files for later analysis. The script uses the requests module to send requests and the json module to convert the response data into JSON format. The time module is also used to insert delays between requests to avoid exceeding the API rate limit.

Overall, this script is useful for collecting data on the top selling NFT contracts and their corresponding NFT metadata, which can be used for market analysis and research purposes. The data pulled for the top selling contracts, including their name, contract address, and metadata such as description and image URLs. The sales statistics for the top selling contract includes volume, sales, average price, market cap, floor price, and historic floor prices over different time periods such as one day, seven days, and thirty days.

B. Cleaning and Extraction of Contract Data

Since the data collected from the NFTPort API are not ready for use, we will need to clean and extract the data from the json file.

The "fix_json.py" code is used to correct a JSON file that has formatting issues. It first reads the original file using the 'open' function in Python and stores the contents in the 'data' variable. The 'replace' function is then used to replace any instances of '}{', which indicate an error in the JSON formatting, with ',{'. The result is then wrapped in square brackets to make it a valid JSON list using the 'fixed_data' variable. The 'json.loads' function is then used to parse the corrected JSON data into a Python dictionary, which is stored in the 'parsed_data' variable. Finally, the 'json.dump' function is used to save the corrected data to a new file with the name 'fixed_contract_stat_30d.json' and an indent of 4 spaces. This ensures that the JSON data is properly formatted and ready for use.

```python
import json

# Read the original file
with open('contract_stat_30d.json', 'r') as f:
    data = f.read()

# Fix the JSON
fixed_data = '[' + data.replace('}{', '},{') + ']'
parsed_data = json.loads(fixed_data)

# Save the fixed JSON to a new file
with open('fixed_contract_stat_30d.json', 'w') as f:
    json.dump(parsed_data, f, indent=4)
```

fix_json.py

Figure 1. Code for Fixing Data Format

Then "extract_top_stat.py" code is used to extract certain properties and specific data from two JSON file and write it to one CSV file. For further analysis of NFT trends, these properties are selected for extraction: “contract_address” and “name” from “top_contract” files, and “statistics” (e.g. “one_day_sales”, “one_day_average_price) from “contract_stat” files.

Take data for 30 days as an example. First, the code uses the 'json.load' function to open and read a JSON file named 'top_contract_30d.json'. The code then loops through each contract in the JSON data and extracts the contract's address and name, which are stored in a new list called 'contracts'. This data is then written to a new CSV file named 'top_stat_30d.csv' using the 'csv.writer' function. The writer writes a header row with the column names "contract_address" and "name", followed by each contract's address and name in their respective columns.

The code then appends additional data to the 'top_stat_30d.csv' file using the 'csv.writer' function again. This time, it first opens and reads a JSON file named 'fixed_contract_stat.json'. The code then extracts the statistics for each contract in the JSON data and writes them to the CSV file as additional columns. The writer writes a header row with the new column names, which are extracted from the statistics data in the JSON file. The code then loops through each contract in the JSON data, extracts its statistics, and writes them to the CSV file row by row.

```python
import json
import csv

# Open and read the JSON file
with open('top_contract_30d.json') as json_file:
    data = json.load(json_file)

# Extract "contract_address" and "name" for each contract
contracts = []
for contract in data['contracts']:
    address = contract['contract_address']
    name = contract['name']
    contracts.append({'address': address, 'name': name})

# Write the extracted data to a CSV file
with open('top_stat_30d.csv', mode='w', newline='') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow(['contract_address', 'name'])
    for contract in contracts:
        writer.writerow([contract['address'], contract['name']])
```

Figure 2. Exacting Certain Property

```python
# Open the output file in append mode
with open('top_stat_30d.csv', mode='a', newline='') as output_file:
    output_writer = csv.writer(output_file)

    # Open the contract_stat.json file and parse it as JSON
    with open('fixed_contract_stat.json') as json_file:
        data = json.load(json_file)

        # Write the headers for the new columns to the output file
        headers = list(data[0]['statistics'].keys())
        output_writer.writerow(['', ''] + headers)

        # Loop through each contract in the JSON data
        for contract in data:
```

Figure 3. Write in the Header

```python
    # Get the statistics for the contract
    stats = contract['statistics']

    # Write the statistics to the output file
    row = ['', '']
    for key in headers:
        row.append(stats[key])
    output_writer.writerow(row)
```

extract_top_stat.py

Figure 4. Loops Through Each Contract

Then code from "extract_meta.py" reads data from a JSON file called "fixed_meta.json". It then iterates over each NFT (non-fungible token) within the data, and extracts its contract address, token ID, and attributes. It then writes this information to a new CSV file named after the contract address, with one row per attribute for each NFT. If there is an error encountered while processing an NFT, the code will skip that NFT and move on to the next. Once all NFTs have been processed, the code reads in each CSV file it has created, combines the attributes for each token ID into a dictionary, and writes this information to a new CSV file with each trait_type as a column header and each token ID as a row. The resulting CSV file will have one row per token ID and one column for each trait_type, with the value for each attribute listed under its corresponding trait_type.

```python
import json
import csv

# read the meta.json file
with open('fixed_meta.json') as f:
    data = json.load(f)

#Contracts
for j in range(50):
    try:
        # code that may raise an error
        for nft in data[j]['nfts']:
            contract_address = nft['contract_address']
            attributes = nft['metadata']['attributes']
            token_id = nft['token_id']
            with open(f'{contract_address}.csv', 'a', newline='') as csvfile:
                writer = csv.writer(csvfile)
                writer.writerow(['token_id', 'trait_type', 'value'])
                for attribute in attributes:
                    writer.writerow([token_id,  attribute['trait_type'],
attribute['value']])
```

Figure 5. Part of "extract_meta.py" Code

C. Data Analysis and Trend Discovery

1. Data Importing

First of all, we will need to import all the CSV files that were clear and organized before to the jupyter notebook. In addition we import the functions and plot for further analysis.

```
In [1]: import os
        import numpy as np
        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import LabelEncoder
        from sklearn.preprocessing import StandardScaler
        from sklearn.neural_network import MLPRegressor
        from sklearn.model_selection import GridSearchCV
        from sklearn.metrics import mean_squared_error,r2_score
        from sklearn.inspection import permutation_importance
        import matplotlib.pyplot as plt
        os.chdir('/Users/zoechan/Desktop/blockchain/data_cleaned_extracted/after')
        df_24h=pd.read_csv('top_stat_extract/top_stat_24h.csv')
        df_30d=pd.read_csv('top_stat_extract/top_stat_30d.csv')
        df_7d=pd.read_csv('top_stat_extract/top_stat_7d.csv')
        df=pd.concat([df_24h,df_30d,df_7d])
```

Figure 6. Importing CSV Files

2. Data Processing

Categorical variables: To process the data, we use the Label Encoding method to convert the categorical variables 'contract_address' and 'name' into numerical labels.

```
In [2]: X = df.drop(columns=['updated_date'])
        # use Label Encoding method to convert categorical variables
        X['contract_address']=LabelEncoder().fit_transform(X['contract_address'])
        X['name']=LabelEncoder().fit_transform(X['name'])
```

Figure 7. Label Encoding

Missing values: We fill in missing values in the DataFrame using the mean value of the rows with the same 'contract_address'. Then, for the remaining missing values, we choose to delete the corresponding rows from the DataFrame.

```
In [3]: # fill in missing value with mean of the same contract_address
        for col in X.columns:
            if X[col].isnull().sum() > 0:
                X[col] = X.groupby('contract_address')[col].transform(lambda x: x.f
        X=X.dropna()
```

Figure 8. Fill in Missing Values

Training/test dataset split: Selecting 'one_day_average_price' as the target variable for prediction,we then split the data into a training set and a test set using an 80/20 split.

Standardization: Fit and transform scalar(z-score normalization method) on X_train and X_test

Features: 'contract_address', 'name', 'one_day_volume', 'one_day_change', 'one_day_sales', 'seven_day_volume', 'seven_day_change', 'seven_day_sales', 'seven_day_average_price', 'thirty_day_volume', 'thirty_day_change', 'thirty_day_sales', 'thirty_day_average_price', 'total_volume', 'total_sales', 'total_supply', 'total_minted', 'num_owners', 'average_price', 'market_cap', 'floor_price', 'floor_price_historic_one_day', 'floor_price_historic_seven_day', 'floor_price_historic_thirty_day', 'updated_date'

```
In [4]: # Split x and y
y = X['one_day_average_price'].to_numpy()
y = y.reshape(y.shape[0],1)
X =X.drop(columns=['one_day_average_price'])
# Split train and test dataset
X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

In [5]: # Fit and transform scalar on X_train and X_test
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Add a column of ones to the feature metrices
# X_train = np.hstack([np.ones((X_train.shape[0], 1)), X_train])
# X_test = np.hstack([np.ones((X_test.shape[0], 1)), X_test])
```

Figure 9. Data Splitting and Transform

3. Data Modelling

Model selection: To predict the 'one_day_average_price' for each contract, we applied the neural network method to build a model. Although the dataset for this project is not very large, neural networks can still work well with smaller datasets if proper regularization techniques are applied to prevent overfitting. Using a neural network for regression problems offers the advantages of flexibility, scalability, and the ability to model complex non-linear relationships between the input features and the target variable.

Adjust parameters: Based on the results of our experimentation using grip research, we adjusted the parameters 'hidden_layer_sizes', 'alpha', and 'learning_rate_init' in order to find the combination that produced the best training results. We then trained the neural network model, called best_mlp, using these optimal parameter values.

```
In [6]: #construct neural network model and use grid search to find best parameters
mlp = MLPRegressor(random_state=42)
param_grid = {
    'hidden_layer_sizes': [(10,), (20,), (30,),(10,2), (20,2), (30,2)],
    'alpha': [0.001, 0.01, 0.1],
    'learning_rate_init': [0.001, 0.01, 0.1]
}
grid_search = GridSearchCV(mlp, param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)
# train model with best parameters
best_mlp = MLPRegressor(random_state=42, **grid_search.best_params_)
best_mlp.fit(X_train, y_train)
```

Figure 10. Adjusting Parameter

Evaluation models: To evaluate the models, we calculated the MSE and R-squared values for both the training set and the test set.

```
# use mse and R square to evaluate models
y_train_pred = best_mlp.predict(X_train)
y_test_pred = best_mlp.predict(X_test)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
print("MSE for training set: ",round(mse_train,4))
print("MSE for test set: ",round(mse_test,4))
print("R square for training set: ",round(r2_train,4))
print("R square test set: ",round(r2_test,4))
```

Figure 11. MSE & R-squared Calculation

III. Showcase of Our work

1. Evaluate MSE & R squared

MSE (Mean Squared Error) and R-squared are two common metrics used to evaluate the performance of a regression model.

By evaluating both MSE and R-squared, we can gain a better understanding of the model's predictive power and how well it fits the data.

The MSE values for the training and test sets were 0.0106 and 0.0488, respectively. The R-squared values for the training and test sets were 0.9999 and 0.9995, respectively.The small MSE values and the R-squared values close to 1 indicate that the model has performed very well and has accurately predicted the target variable.

```
In [7]: # use mse and R square to evaluate models
y_train_pred = best_mlp.predict(X_train)
y_test_pred = best_mlp.predict(X_test)
mse_train = mean_squared_error(y_train, y_train_pred)
mse_test = mean_squared_error(y_test, y_test_pred)
r2_train = r2_score(y_train, y_train_pred)
r2_test = r2_score(y_test, y_test_pred)
print("MSE for training set: ",round(mse_train,4))
print("MSE for test set: ",round(mse_test,4))
print("R square for training set: ",round(r2_train,4))
print("R square test set: ",round(r2_test,4))

MSE for training set:  0.0106
MSE for test set:  0.0488
R square for training set:  0.9999
R square test set:  0.9995
```

Figure 12. MSE & R-squared Results

2. Ranking Feature

```
In [8]: # rank feature importance
result = permutation_importance(best_mlp, X_train, y_train, n_repeats=10, r
sorted_idx = result.importances_mean.argsort()
plt.barh(range(X_train.shape[1]), result.importances_mean[sorted_idx])
print(result.importances_mean[sorted_idx])
plt.yticks(range(X_train.shape[1]), X.columns[sorted_idx])
plt.xlabel("Permutation Importance")
plt.show()

[0.00012182 0.00013284 0.00056021 0.00060454 0.0006074  0.00065428
 0.00077582 0.00096673 0.00107837 0.00221481 0.00269304 0.00369244
 0.00592162 0.00619088 0.00701622 0.00745477 0.01503798 0.0164005
 0.03676156 0.03718142 0.04052977 0.04482619 0.06256485 0.06871042]
```

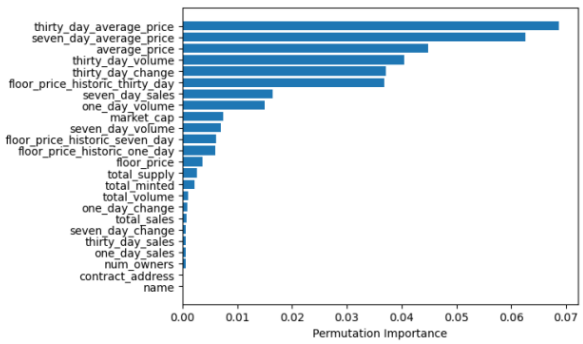

Figure 13. Ranking Feature

3. Top NFT price with different artist

The X-axis represents the artist names, while the y-axis represents the NFT prices. This graph shows a significant price gap between the top artist and the rest of them.

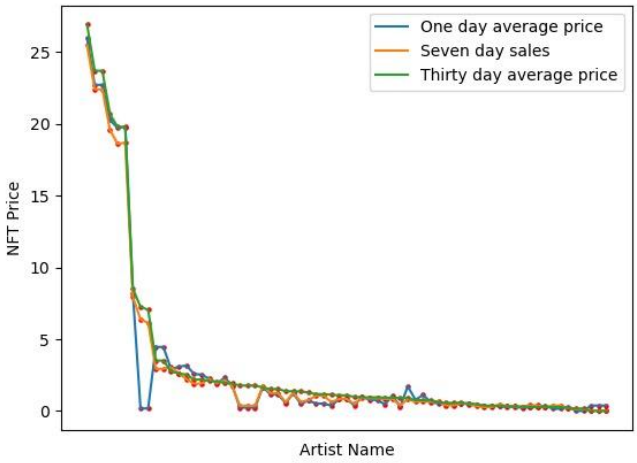

Figure 14. Top Artist VS NFT Price

4. Top 30 Artists NFT Price

Our code has the ability to display top 30 artists that produce NFT by their average price in 30 days. It will provide the average price for a day, a week, and a month.

```
In [10]: # Top 30 artists with high nft thirty_day_average_price
         df3 = df2.reset_index()[['name','one_day_average_price','seven_day_average_
         df3
```

Out[10]:

	name	one_day_average_price	seven_day_average_price	thirty_day_average_price
0	Azuki	25.951447	25.472240	26.911870
1	Wrapped Cryptopunks	22.710144	22.400138	23.692012
2	MutantApeYachtClub	22.710144	22.400138	23.692012
3	BoredApeKennelClub	20.265496	19.570104	20.672733
4	Beanz	19.702313	18.600017	19.784315
5	PudgyPenguins	19.803098	18.688494	19.774501
6	BoredApeYachtClub	8.165408	7.947592	8.539467
7	Sandbox's ASSETs	0.163750	6.368622	7.239723
8	Grails III Mint Pass	0.190917	6.150855	7.057725
9	Consortium Key	4.460000	2.931262	3.513031
10	MetaNinjago	4.460000	2.931262	3.513031
11	HV-MTL	2.914355	3.044166	2.749238
12	Captainz	3.083827	2.631754	2.618409
13	DeGods	3.166878	2.227214	2.532828
14	Kanpai Pandas	2.595867	1.909718	2.189529

Figure 15. Top 30 Artist List

IV. Conclusion

There are three main conclusions that can be drawn from our code and the plot we demonstrated.

1. NFT price prediction: Considering the MSE and R-squared values,we can conclude that the neural network model we trained using the optimal parameter values obtained from grid research has excellent performance in predicting the 'one_day_average_price' for each contract.
2. The features 'thirty_day_average_price', 'seven_day_average_price', 'average_price', 'thirty_day_volume', 'thirty_day_change', and 'floor_price_historic_thirty_day' have the greatest impact on the price of a series of NFTs.NFT series with a thirty-day average price higher than the seven-day average price have the potential to appreciate in value, especially those with prices higher than the historical average price, high trading volume, and significant price changes in the last thirty days. These are worth considering for investment.
3. The prices of different NFT series minted by different artists can vary greatly. The most popular 6 NFT series are from the artists 'Azuki', 'Wrapped Cryptopunks', 'MutantApeYachtClub', 'BoredApeKennelClub', 'Beanz', and 'PudgyPenguins'. These 6 NFT series are priced much higher, up to even 10 times more, compared to other NFT series.

CONTRIBUTION OF WORK	
Background: Introduction to NFTs, OpenSea, and IPFS	Tianyu Qin (tq2155)
Data Collection and Storage	Mingyang Song (ms6525)
Data Preprocessing and Feature Extraction	Victoria Fang (yf2647) Xueji Zhao (xz3159)
Data Analysis and Trend Discovery	Ruoyu Chen (rc3506)
Documentation and Presentation	Yusheng Chen (yc4157)

References

[1] Builtin. "10 Surprising Use Cases for NFTs Beyond Digital Art." Builtin, 8 Apr. 2021, https://builtin.com/nft-non-fungible-token/nft-use-cases.

[2] Investopedia. "What Is OpenSea? A Guide to the NFT Marketplace." Investopedia, 21 Oct. 2021, https://www.investopedia.com/what-is-opensea-6362477.

[3] Wikipedia. "InterPlanetary File System." Wikipedia, Wikimedia Foundation, 28 Apr. 2022, https://en.wikipedia.org/wiki/InterPlanetary_File_System.

[4] GeeksforGeeks. "Difference Between HTTP and IPFS." GeeksforGeeks, n.d., https://www.geeksforgeeks.org/difference-between-http-and-ipfs/.