**Appendices**

**Appendix A. Selected ERC20 tokens for the analysis.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Token symbol** | **Token Name** | **CoinMarketCap Link** | **Link to the Token's smart contract** | **Data available from** | **Number of observations** |
| UNI | Uniswap | <https://coinmarketcap.com/currencies/uniswap/> | <https://etherscan.io/address/0x1f9840a85d5af5bf1d1762f925bdaddc4201f984#code> | 15.10.2020 | 5586 |
| LINK | Chainlink | <https://coinmarketcap.com/currencies/chainlink/> | <https://etherscan.io/address/0x514910771af9ca656af840dff83e8264ecf986ca#code> | 28.09.2017 | 32840 |
| AAVE | Aave | <https://coinmarketcap.com/currencies/aave/> | <https://etherscan.io/address/0x7Fc66500c84A76Ad7e9c93437bFc5Ac33E2DDaE9#code> | 15.10.2020 | 6176 |
| MKR | Maker | <https://coinmarketcap.com/currencies/maker/> | <https://etherscan.io/address/0x9f8f72aa9304c8b593d555f12ef6589cc3a579a2#code> | 22.01.2018 | 25297 |
| LEO | UNUS SED LEO | <https://coinmarketcap.com/currencies/unus-sed-leo/> | <https://etherscan.io/address/0x2af5d2ad76741191d15dfe7bf6ac92d4bd912ca3#code> | 20.05.2019 | 5505 |
| COMP | Compund | <https://coinmarketcap.com/currencies/compound/> | <https://etherscan.io/address/0xc00e94cb662c3520282e6f5717214004a7f26888#code> | 15.07.2020 | 6859 |
| GRT | The Graph | <https://coinmarketcap.com/currencies/the-graph/> | <https://etherscan.io/address/0xc944e90c64b2c07662a292be6244bdf05cda44a7#code> | 17.12.2020 | 4644 |
| HT | Huobi Token | <https://coinmarketcap.com/currencies/huobi-token/> | <https://etherscan.io/address/0x6f259637dcd74c767781e37bc6133cd6a68aa161#code> | 27.02.2018 | 26017 |
| CEL | Celsius | <https://coinmarketcap.com/currencies/celsius/> | <https://etherscan.io/address/0xaaaebe6fe48e54f431b0c390cfaf0b017d09d42d#code> | 08.10.2019 | 5267 |
| CHZ | Chiliz | <https://coinmarketcap.com/currencies/chiliz/> | <https://etherscan.io/address/0x3506424f91fd33084466f402d5d97f05f8e3b4af#code> | 30.09.2020 | 6496 |
| TEL | Telcoin | <https://coinmarketcap.com/currencies/telcoin/> | <https://etherscan.io/address/0x467bccd9d29f223bce8043b84e8c8b282827790f#code> | 25.01.2018 | 27462 |
| YFI | yearn.finance | <https://coinmarketcap.com/currencies/yearn-finance/> | <https://etherscan.io/address/0x0bc529c00c6401aef6d220be8c6ea1667f6ad93e#code> | 15.10.2020 | 4622 |
| HOT | Holo | <https://coinmarketcap.com/currencies/holo/> | <https://etherscan.io/address/0x6c6ee5e31d828de241282b9606c8e98ea48526e2#code> | 24.07.2018 | 25627 |
| ENJ | Enjin Coin | <https://coinmarketcap.com/currencies/enjin-coin/> | <https://etherscan.io/address/0xf629cbd94d3791c9250152bd8dfbdf380e2a3b9c#code> | 03.11.2017 | 21862 |
| MANA | Decentraland | <https://coinmarketcap.com/currencies/decentraland/> | <https://etherscan.io/address/0x0f5d2fb29fb7d3cfee444a200298f468908cc942#code> | 24.11.2017 | 31290 |
| QNT | Quant | <https://coinmarketcap.com/currencies/quant/> | <https://etherscan.io/address/0x4a220e6096b25eadb88358cb44068a3248254675#code> | 28.09.2018 | 6782 |
| BAT | Basic Attention Token | <https://coinmarketcap.com/currencies/basic-attention-token/> | <https://etherscan.io/address/0x0d8775f648430679a709e98d2b0cb6250d2887ef#code> | 13.11.2017 | 31708 |
| SNX | Synthetix | <https://coinmarketcap.com/currencies/synthetix-network-token/> | <https://etherscan.io/address/0xc011a73ee8576fb46f5e1c5751ca3b9fe0af2a6f#code> | 17.09.2020 | 5379 |
| NEXO | Nexo | <https://coinmarketcap.com/currencies/nexo/> | <https://etherscan.io/address/0xb62132e35a6c13ee1ee0f84dc5d40bad8d815206#code> | 06.07.2018 | 23829 |
| BNT | Bancor | <https://coinmarketcap.com/currencies/bancor/> | <https://etherscan.io/address/0x1f573d6fb3f13d689ff844b4ce37794d79a7ff1c#code> | 27.07.2017 | 31979 |
| CRV | Curve DAO Token | <https://coinmarketcap.com/currencies/curve-dao-token/> | <https://etherscan.io/address/0xD533a949740bb3306d119CC777fa900bA034cd52#code> | 17.09.2020 | 5406 |
| CHSB | SwissBorg | <https://coinmarketcap.com/currencies/swissborg/> | <https://etherscan.io/address/0xba9d4199fab4f26efe3551d490e3821486f135ba#code> | 07.02.2018 | 28161 |
| KCS | KuCoin Token | <https://coinmarketcap.com/currencies/kucoin-token/> | <https://etherscan.io/address/0xf34960d9d60be18cc1d5afc1a6f012a723a28811#code> | 24.01.2018 | 28262 |
| ZRX | 0x Token | <https://coinmarketcap.com/currencies/0x/> | <https://etherscan.io/address/0xe41d2489571d322189246dafa5ebde1f4699f498#code> | 13.09.2017 | 32886 |
| UMA | Uma | <https://coinmarketcap.com/currencies/uma/> | <https://etherscan.io/address/0x04Fa0d235C4abf4BcF4787aF4CF447DE572eF828#code> | 03.07.2020 | 8682 |
| ANKR | Ankr | <https://coinmarketcap.com/currencies/ankr/> | <https://etherscan.io/address/0x8290333cef9e6d528dd5618fb97a76f268f3edd4#code> | 26.03.2019 | 6656 |
| VGX | Voyager Token | <https://coinmarketcap.com/currencies/voyager-token/> | <https://etherscan.io/address/0x5af2be193a6abca9c8817001f45744777db30756#code> | 22.09.2017 | 32251 |
| 1INCH | 1INCH | <https://coinmarketcap.com/currencies/1inch/> | <https://etherscan.io/address/0x111111111117dc0aa78b770fa6a738034120c302#code> | 25.12.2020 | 4491 |

**Appendix B1**. **Results for 24 lags.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Token | RMSE | | MAE | | MAPE | |
| ARIMA | LSTM | ARIMA | LSTM | ARIMA | LSTM |
| UNI | 9.06404E-05 | 2.67926E-04 | 5.73061E-05 | 2.02419E-04 | 6.08580E-01 | 2.14260E+00 |
| LINK | 1.79011E-04 | 5.22499E-03 | 1.17702E-04 | 4.97856E-03 | 6.79774E-01 | 2.88102E+01 |
| AAVE | 1.71168E-03 | 2.83228E-03 | 9.77731E-04 | 2.03150E-03 | 7.13180E-01 | 1.47895E+00 |
| MKR | 1.98077E-02 | 3.97226E-02 | 1.16298E-02 | 2.29970E-02 | 9.20166E-01 | 1.90687E+00 |
| LEO | 3.36795E-05 | 2.05373E-04 | 2.31220E-05 | 2.02315E-04 | 2.17993E+00 | 1.99179E+01 |
| COMP | 2.46043E-03 | 1.59488E-02 | 1.66468E-03 | 1.51799E-02 | 9.21362E-01 | 9.57930E+00 |
| GRT | 3.86526E-06 | 1.75880E-05 | 2.68551E-06 | 1.70209E-05 | 9.29903E-01 | 6.02689E+00 |
| HT | 2.12909E-04 | 2.06984E-04 | 7.53810E-05 | 1.17345E-04 | 1.06948E+00 | 1.79579E+00 |
| CEL | 2.34849E-04 | 2.30451E-04 | 1.13950E-04 | 1.16851E-04 | 3.78446E+00 | 3.91990E+00 |
| CHZ | 2.49000E-06 | 7.64000E-06 | 1.36000E-06 | 6.91000E-06 | 1.16918E+00 | 5.94614E+00 |
| TEL | 2.13297E-07 | 8.24210E-07 | 9.56808E-08 | 6.17160E-07 | 2.59303E+00 | 5.12857E+01 |
| YFI | 2.95495E-01 | 1.43839E+00 | 1.63596E-01 | 7.88750E-01 | 9.42779E-01 | 4.50522E+00 |
| HOT | 1.33555E-07 | 2.23518E-07 | 5.57198E-08 | 1.59440E-07 | 1.64637E+00 | 1.35819E+01 |
| ENJ | 1.85110E-05 | 2.22516E-05 | 9.86118E-06 | 1.43454E-05 | 1.47131E+00 | 2.33141E+00 |
| MANA | 6.97098E-06 | 3.03288E-05 | 3.58309E-06 | 1.16307E-05 | 1.31614E+00 | 3.50638E+00 |
| QNT | 2.87547E-01 | 2.84693E-01 | 2.07803E-02 | 1.89096E-02 | 4.69933E+01 | 3.92608E+01 |
| BAT | 6.32356E-06 | 1.74275E-05 | 3.49581E-06 | 1.46467E-05 | 9.01583E-01 | 4.44443E+00 |
| SNX | 8.97874E-05 | 4.35969E-04 | 5.34372E-05 | 4.09287E-04 | 1.06222E+00 | 9.40280E+00 |
| NEXO | 1.81876E-05 | 3.22200E-05 | 1.19565E-05 | 2.52099E-05 | 1.28075E+00 | 2.70857E+00 |
| BNT | 2.68782E-05 | 1.06514E-04 | 1.64259E-05 | 9.59690E-05 | 7.23598E-01 | 5.35325E+00 |
| CRV | 1.95912E-05 | 3.57869E-05 | 1.27500E-05 | 3.12639E-05 | 1.50356E+00 | 3.90893E+00 |
| CHSB | 9.73155E-06 | 9.79767E-05 | 5.80047E-06 | 7.96358E-05 | 1.49482E+00 | 1.82276E+01 |
| KCS | 6.57268E-05 | 2.26387E-03 | 3.25546E-05 | 1.25420E-03 | 1.14222E+00 | 4.73961E+01 |
| ZRX | 1.11431E-05 | 1.51653E-05 | 6.76727E-06 | 1.09594E-05 | 9.94104E-01 | 1.53209E+00 |
| UMA | 1.08953E-04 | 3.70172E-04 | 6.73017E-05 | 3.52457E-04 | 9.97224E-01 | 6.19860E+00 |
| ANKR | 2.20527E-06 | 2.76349E-06 | 1.29156E-06 | 1.89732E-06 | 2.28955E+00 | 3.15845E+00 |
| VGX | 4.19615E-05 | 3.75392E-04 | 2.22734E-05 | 3.17836E-04 | 2.02218E+00 | 4.50527E+01 |
| 1INCH | 1.85317E-05 | 2.77286E-05 | 1.19965E-05 | 1.99223E-05 | 9.32958E-01 | 1.52222E+00 |

**Appendix B2**. **Results for 48 lags.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Token | RMSE | | MAE | | MAPE | |
| ARIMA | LSTM | ARIMA | LSTM | ARIMA | LSTM |
| UNI | 8.56710E-05 | 2.33801E-04 | 5.46636E-05 | 1.48715E-04 | 5.80705E-01 | 1.59824E+00 |
| LINK | 1.75312E-04 | 8.94015E-03 | 1.15813E-04 | 8.28925E-03 | 6.68420E-01 | 4.56319E+01 |
| AAVE | 1.66228E-03 | 2.29220E-03 | 9.57466E-04 | 1.57358E-03 | 6.98750E-01 | 1.15440E+00 |
| MKR | 1.94174E-02 | 6.45744E-02 | 1.13252E-02 | 4.03930E-02 | 8.96384E-01 | 3.57375E+00 |
| LEO | 3.18728E-05 | 3.67453E-04 | 2.14406E-05 | 3.65335E-04 | 2.01072E+00 | 3.58532E+01 |
| COMP | 2.34937E-03 | 1.29116E-02 | 1.59287E-03 | 1.20911E-02 | 8.82359E-01 | 7.50740E+00 |
| GRT | 3.72636E-06 | 4.89193E-06 | 2.61109E-06 | 3.48275E-06 | 9.05723E-01 | 1.21375E+00 |
| HT | 1.90975E-04 | 2.05816E-04 | 7.12413E-05 | 1.13300E-04 | 1.01424E+00 | 1.69263E+00 |
| CEL | 2.30904E-04 | 3.19865E-04 | 1.12983E-04 | 2.42882E-04 | 3.74524E+00 | 8.18690E+00 |
| CHZ | 2.34376E-06 | 1.96084E-05 | 1.31347E-06 | 1.91486E-05 | 1.13027E+00 | 1.65773E+01 |
| TEL | 1.98785E-07 | 1.16833E-06 | 8.77344E-08 | 7.61114E-07 | 2.40361E+00 | 5.17223E+01 |
| YFI | 2.86305E-01 | 1.30799E+00 | 1.56908E-01 | 6.86864E-01 | 9.03044E-01 | 3.93371E+00 |
| HOT | 1.22628E-07 | 2.42769E-07 | 5.21007E-08 | 1.33099E-07 | 1.53808E+00 | 9.03227E+00 |
| ENJ | 1.75836E-05 | 1.90855E-05 | 9.43488E-06 | 1.16998E-05 | 1.41429E+00 | 1.99758E+00 |
| MANA | 6.31827E-06 | 2.55749E-05 | 3.32228E-06 | 1.01389E-05 | 1.23139E+00 | 3.00008E+00 |
| QNT | 2.77677E-01 | 2.92121E-01 | 1.95510E-02 | 1.97252E-02 | 4.44757E+01 | 4.18974E+01 |
| BAT | 6.15305E-06 | 2.71476E-05 | 3.40032E-06 | 2.30426E-05 | 8.76802E-01 | 7.35729E+00 |
| SNX | 8.73123E-05 | 6.15012E-04 | 5.22191E-05 | 5.86350E-04 | 1.03825E+00 | 1.35928E+01 |
| NEXO | 1.73635E-05 | 4.86451E-05 | 1.15192E-05 | 4.18879E-05 | 1.23498E+00 | 4.86547E+00 |
| BNT | 2.56782E-05 | 4.26645E-05 | 1.57035E-05 | 3.14903E-05 | 6.90186E-01 | 1.47706E+00 |
| CRV | 1.89430E-05 | 2.09929E-05 | 1.20886E-05 | 1.49898E-05 | 1.42802E+00 | 1.81757E+00 |
| CHSB | 9.06030E-06 | 1.28877E-04 | 5.47888E-06 | 1.04538E-04 | 1.40851E+00 | 2.35154E+01 |
| KCS | 6.38368E-05 | 6.17732E-04 | 3.17850E-05 | 4.23328E-04 | 1.11790E+00 | 2.38608E+01 |
| ZRX | 1.08569E-05 | 3.11016E-05 | 6.57220E-06 | 2.94100E-05 | 9.65315E-01 | 4.88970E+00 |
| UMA | 1.05768E-04 | 1.17972E-04 | 6.58770E-05 | 8.58564E-05 | 9.74489E-01 | 1.38291E+00 |
| ANKR | 1.99927E-06 | 2.33066E-06 | 1.17036E-06 | 1.50536E-06 | 2.07537E+00 | 2.52774E+00 |
| VGX | 3.99050E-05 | 3.43867E-04 | 2.11302E-05 | 2.61656E-04 | 1.96498E+00 | 3.56536E+01 |
| 1INCH | 1.74175E-05 | 4.61766E-05 | 1.13751E-05 | 3.86333E-05 | 8.86684E-01 | 2.95998E+00 |

**Appendix B3. Results for 72 lags.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Token | RMSE | | MAE | | MAPE | |
| ARIMA | LSTM | ARIMA | LSTM | ARIMA | LSTM |
| UNI | 8.40874E-05 | 2.21339E-04 | 5.41629E-05 | 1.12962E-04 | 5.75143E-01 | 1.22044E+00 |
| LINK | 1.73544E-04 | 1.43212E-02 | 1.14638E-04 | 1.41150E-02 | 6.62427E-01 | 8.62388E+01 |
| AAVE | 1.62268E-03 | 2.61891E-03 | 9.54035E-04 | 1.85741E-03 | 6.96903E-01 | 1.36389E+00 |
| MKR | 1.92647E-02 | 3.18721E-02 | 1.12997E-02 | 1.56645E-02 | 8.93513E-01 | 1.27057E+00 |
| LEO | 3.09418E-05 | 2.70204E-04 | 2.10236E-05 | 2.67542E-04 | 1.97132E+00 | 2.63333E+01 |
| COMP | 2.30363E-03 | 1.53005E-02 | 1.56217E-03 | 1.45013E-02 | 8.67044E-01 | 9.22645E+00 |
| GRT | 3.69673E-06 | 7.88430E-06 | 2.57068E-06 | 6.50492E-06 | 8.91072E-01 | 2.26806E+00 |
| HT | 1.72882E-04 | 2.04260E-04 | 7.01552E-05 | 1.09328E-04 | 9.98876E-01 | 1.62656E+00 |
| CEL | 2.31870E-04 | 2.37943E-04 | 1.11750E-04 | 1.11444E-04 | 3.70913E+00 | 3.68055E+00 |
| CHZ | 2.25442E-06 | 2.20623E-05 | 1.28707E-06 | 2.16870E-05 | 1.10763E+00 | 1.88382E+01 |
| TEL | 1.84298E-07 | 8.31364E-07 | 8.41189E-08 | 6.33915E-07 | 2.29560E+00 | 5.88544E+01 |
| YFI | 2.84574E-01 | 1.17612E+00 | 1.56484E-01 | 6.03514E-01 | 9.00886E-01 | 3.45192E+00 |
| HOT | 1.19673E-07 | 2.25802E-07 | 5.05913E-08 | 1.64098E-07 | 1.51105E+00 | 1.40261E+01 |
| ENJ | 1.71546E-05 | 2.11936E-05 | 9.23889E-06 | 1.34666E-05 | 1.38139E+00 | 2.18429E+00 |
| MANA | 6.18810E-06 | 2.36270E-05 | 3.26682E-06 | 5.95956E-06 | 1.20914E+00 | 2.15482E+00 |
| QNT | 2.75146E-01 | 2.83774E-01 | 1.89958E-02 | 1.74291E-02 | 4.39413E+01 | 3.52162E+01 |
| BAT | 6.09633E-06 | 2.83874E-05 | 3.38068E-06 | 2.40831E-05 | 8.73688E-01 | 7.71202E+00 |
| SNX | 8.73632E-05 | 4.00240E-04 | 5.21882E-05 | 3.55005E-04 | 1.03648E+00 | 8.07914E+00 |
| NEXO | 1.69863E-05 | 3.84098E-05 | 1.13082E-05 | 3.07968E-05 | 1.21303E+00 | 3.55816E+00 |
| BNT | 2.51207E-05 | 6.39418E-05 | 1.53220E-05 | 5.33223E-05 | 6.71250E-01 | 2.65422E+00 |
| CRV | 1.83011E-05 | 2.18932E-05 | 1.19197E-05 | 1.54367E-05 | 1.40842E+00 | 1.85987E+00 |
| CHSB | 8.91383E-06 | 3.07893E-05 | 5.38190E-06 | 2.41543E-05 | 1.38185E+00 | 5.44907E+00 |
| KCS | 6.40177E-05 | 1.00299E-03 | 3.17196E-05 | 6.23739E-04 | 1.11011E+00 | 2.30079E+01 |
| ZRX | 1.07389E-05 | 1.31500E-05 | 6.51441E-06 | 9.58005E-06 | 9.56385E-01 | 1.67187E+00 |
| UMA | 1.04286E-04 | 3.42271E-04 | 6.53984E-05 | 3.29165E-04 | 9.66581E-01 | 5.70591E+00 |
| ANKR | 1.92700E-06 | 2.04911E-06 | 1.14577E-06 | 1.25131E-06 | 2.04319E+00 | 2.16285E+00 |
| VGX | 4.44316E-05 | 4.23437E-04 | 2.11255E-05 | 3.47864E-04 | 2.03085E+00 | 4.83918E+01 |
| 1INCH | 1.71360E-05 | 2.53758E-05 | 1.11447E-05 | 1.83619E-05 | 8.69029E-01 | 1.40788E+00 |

**Appendix C1. Min-max natural logarithms of RMSE results for 24 lags.**

**Chart, bar chart

Description automatically generated**

**Appendix C2. Min-max natural logarithms of MAE results for 24 lags.**

Chart, bar chart

Description automatically generated

**Appendix C3. Min-max natural logarithms of MAPE results for 24 lags.**

Chart, bar chart

Description automatically generated

**Appendix C4. Min-max natural logarithms of RMSE results for 48 lags.**

Chart, bar chart

Description automatically generated

**Appendix C5. Min-max natural logarithms of MAE results for 48 lags.**

Chart, bar chart

Description automatically generated

**Appendix C6. Min-max natural logarithms of MAPE results for 48 lags.**

Chart, bar chart

Description automatically generated

**Appendix C7. Min-max natural logarithms of RMSE results for 72 lags.**

Chart, bar chart

Description automatically generated

**Appendix C8. Min-max natural logarithms of MAE results for 72 lags.**

Chart, bar chart

Description automatically generated

**Appendix C9. Min-max natural logarithms of MAPE results for 72 lags.**

Chart, bar chart

Description automatically generated

**Appendix D. JavaScript program for retrieving data for the selected ERC20 tokens.**

**// Improting required libraries for working with the filesystem and HTTP requests**

**const fs = require('fs');**

**const axios = require('axios');**

**// List of ERC20 tokens for which the data should be collected**

**const TOKENS = ['UNI', 'LINK', 'AAVE', 'MKR', 'FTT', 'AMP', 'LEO', 'COMP', 'GRT', 'HT', 'CEL', 'CHZ', 'TEL', 'YFI', 'HOT', 'ENJ', 'SUSHI', 'MANA', 'QNT', 'BAT', 'SNX', 'NEXO', 'BNT', 'CRV', 'CHSB', 'KCS', 'ZRX', 'UMA', 'ANKR', 'VGX', '1INCH'];**

**// API Key settings**

**const API\_KEYS = ['4250F621-E075-46FC-8DA9-646D89B3489C', '92877E56-7AE5-44E9-9861-9D0388301E11'];**

**// Index that tracks which API key is used**

**let currAPIKeyIndex = 0;**

**// An http options object**

**const httpOptions = {**

**method: 'GET',**

**hostname: 'rest.coinapi.io',**

**headers: { 'X-CoinAPI-Key': API\_KEYS[0] },**

**};**

**// A method that sends an HTTP request to the CoinAPI**

**const retriveData = (token, startDate) => {**

**console.log(`Fetching data for ${token}`);**

**const url = `http://rest.coinapi.io/v1/ohlcv/${token}/ETH/history?period\_id=1HRS&time\_start=${startDate}&time\_end=2021-07-01T00:00:00`;**

**return axios.get(url, httpOptions).catch(error => {**

**console.error('ERROR when sending a request');**

**return;**

**});**

**};**

**// A method that parses response data in the required format and saves it as a text file**

**const parseAndSaveData = (file, rawData) => {**

**const data = rawData.data;**

**let timePeriodEnd;**

**for (let i = 0; i < data.length; i++) {**

**const currData = data[i];**

**let row = `${currData.time\_period\_end}\t${currData.price\_open}\t${currData.price\_high}\t${currData.price\_low}\t${currData.price\_close}\t${currData.volume\_traded}\t${currData.trades\_count}\n`;**

**fs.appendFileSync(file, row, error => {**

**if (error) {**

**console.error('ERROR when writing a file');**

**return;**

**}**

**});**

**timePeriodEnd = currData.time\_period\_end;**

**}**

**return timePeriodEnd;**

**};**

**// A method that launches the proecss of retrieving the data of the tokens and writing it to a file**

**const startRetrievingData = async () => {**

**const START\_DATE = '2017-01-01T00:00:00.0000000Z';**

**const i = 30;**

**//start loop**

**const token = TOKENS[i];**

**for (let i=0; i<TOKENS.length; i++) {**

**const token = TOKENS[i];**

**let currentStartDate = START\_DATE;**

**let start=true;**

**// Sending requests until all data till 1st of July 2021 is retrievid**

**while (currentStartDate !== START\_DATE || start) {**

**// Retrieving data**

**const result = await retriveData(token, currentStartDate);**

**if (result) {**

**// Parsing and saving the results in a file**

**currentStartDate = await parseAndSaveData(`DataFiles/${token}.txt`, result);**

**start = false;**

**// Quiting the program if an error occured**

**if (!currentStartDate) {**

**return;**

**}**

**} else {**

**// Selecting new API key**

**currAPIKeyIndex++;**

**httpOptions.headers = { 'X-CoinAPI-Key': API\_KEYS[currAPIKeyIndex] };**

**}**

**}**

**}**

**};**

**// Script launcher**

**startRetrievingData();**

**Appendix E. Python program for performing the main analysis.**

**"""**

**Automatically generated by Colaboratory.**

**Original file is located at**

**https://colab.research.google.com/drive/1u65OMZCRYt2\_aEZIwBOrSDjdKvHAWq3-**

**"""**

**# importing required libraries**

**import sys**

**import numpy as numpy**

**from pandas import read\_csv, DataFrame**

**import matplotlib.pyplot as pyplot**

**! pip install scikit-learn**

**from sklearn.preprocessing import MinMaxScaler**

**from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error**

**# libraries for ARIMA model**

**! pip install pmdarima**

**import pmdarima as pmdarima**

**from statsmodels.tsa.arima.model import ARIMA**

**from statsmodels.tsa.stattools import adfuller**

**# libraries for LSTM model**

**from keras.models import Sequential**

**from keras.layers import LSTM, Dense**

**from keras.callbacks import EarlyStopping**

**"""# Importing and pre-processing data"""**

**# array of lags that will be analysed**

**lags\_arr = [24, 48, 72]**

**# array of token symbols that will be analysed**

**token\_symbols = ['UNI', 'LINK', 'AAVE', 'MKR', 'LEO', 'COMP', 'GRT', 'HT', 'CEL',**

**'CHZ', 'TEL', 'YFI', 'HOT', 'ENJ', 'MANA', 'QNT', 'BAT', 'SNX', 'NEXO',**

**'BNT', 'CRV', 'CHSB', 'KCS', 'ZRX', 'UMA', 'ANKR', 'VGX', '1INCH']**

**number\_of\_tokens = len(token\_symbols)**

**# importing data from csv files (before executing this step, csv data files should be uploaded)**

**raw\_datasets = []**

**for symbol in token\_symbols:**

**raw\_datasets.append(read\_csv('{}.csv'.format(symbol)))**

**# scaling data and splitting datasets to training and test datasets**

**training\_set\_ratio = 0.80**

**data\_columns = ['Open', 'High', 'Low', 'Close', 'Volume', 'Trades']**

**scalers = []**

**training\_datasets, test\_datasets = [], []**

**for i in range(number\_of\_tokens):**

**# separating dates from values, converting data to float type**

**raw\_dataset\_without\_dates = raw\_datasets[i][data\_columns].astype('float')**

**# scaling data**

**scaler = MinMaxScaler()**

**training\_set\_size = int(len(raw\_datasets[i])\*training\_set\_ratio)**

**raw\_training\_dataset = raw\_dataset\_without\_dates[:training\_set\_size]**

**scaler.fit(raw\_training\_dataset)**

**scaled\_dataset = DataFrame(scaler.transform(raw\_dataset\_without\_dates), columns=data\_columns)**

**scalers.append(scaler)**

**# splitting scaled dataset to training and test datasets**

**training\_datasets.append(scaled\_dataset[:training\_set\_size])**

**test\_datasets.append(scaled\_dataset[training\_set\_size:])**

**"""#ARIMA"""**

**# determining optimal ARIMA parameters for each dataset**

**ARIMA\_params = []**

**for i in range(number\_of\_tokens):**

**model = pmdarima.auto\_arima(training\_datasets[i]['Close'],**

**test='adf', # using Augmented Dickey-Fuller (ADF) test to find optimal 'd'**

**d=None, # stating that the model should determine 'd' parameter**

**start\_p=1, max\_p=3, # setting range for potential 'p' parameters**

**start\_q=1, max\_q=3, # setting range for potential 'q' parameters**

**error\_action='ignore', # stating that erros should not be printed**

**suppress\_warnings=True) # stating that warnings should not be printed**

**# recording calculated optimal ARIMA parameters**

**ARIMA\_params.append(model.order)**

**# printing out the parameters for visibility**

**print(token\_symbols[i] + ' - ' + str(model.order))**

**# stating that warnings should not be printed**

**import warnings**

**warnings.filterwarnings("ignore")**

**# predicting future prices with ARIMA**

**ARIMA\_predictions\_for\_all\_lags = []**

**# for each lags option (i.e. for 24 lags, 48 lags, and 72 lags)**

**for lags in lags\_arr:**

**print('{} lags:'.format(lags))**

**ARIMA\_predictions = []**

**# for each token**

**for i in range(number\_of\_tokens):**

**# Extracting closing prices data**

**test\_dataset\_close\_prices = test\_datasets[i]['Close'].to\_numpy().tolist()**

**training\_dataset\_close\_prices = training\_datasets[i]['Close'].to\_numpy().tolist()**

**# creating batches for testing**

**training\_X = []**

**for j in range(len(test\_dataset\_close\_prices)):**

**curr\_data = []**

**if j >= lags:**

**curr\_data.extend(test\_dataset\_close\_prices[(j-lags):j])**

**else:**

**curr\_data.extend(training\_dataset\_close\_prices[-(lags-j):])**

**curr\_data.extend(test\_dataset\_close\_prices[0:j])**

**training\_X.append(curr\_data)**

**# forecasting future prices from the test batches with ARIMA**

**curr\_predictions = []**

**for j in range(len(test\_dataset\_close\_prices)):**

**sys.stdout.write('\r' + '{} - {}/{}'.format(token\_symbols[i], j, len(test\_dataset\_close\_prices)))**

**sys.stdout.flush()**

**model = ARIMA(training\_X[j], order=ARIMA\_params[i])**

**model\_fit = model.fit()**

**output = model\_fit.forecast()**

**yhat = output[0]**

**curr\_predictions.append(yhat)**

**# recording predictions for a token**

**ARIMA\_predictions.append(curr\_predictions)**

**print(' - done\n')**

**# recording predictions for all tokens**

**ARIMA\_predictions\_for\_all\_lags.append(ARIMA\_predictions)**

**"""#LSTM"""**

**# creating batches for training and testing**

**training\_y\_arr, training\_X\_arr, test\_X\_arr = [], [], []**

**for lags in lags\_arr:**

**# arrays for storing batches for trianing and testing**

**training\_X, test\_X = [], []**

**# array for storing dependent variables of the training dataset**

**training\_y = []**

**for i in range(number\_of\_tokens):**

**# extracting dependent variables of the training dataset**

**training\_y.append(training\_datasets[i][lags:]['Close'].to\_numpy())**

**# creating batches for training**

**curr\_training\_X = []**

**for j in range(0, len(training\_datasets[i]) - lags):**

**curr\_training\_X.append(training\_datasets[i][j:j+lags].to\_numpy().tolist())**

**training\_X.append(numpy.array(curr\_training\_X))**

**# creating batches for testing**

**curr\_test\_X = []**

**for j in range(len(test\_datasets[i])):**

**curr\_data = []**

**if j >= lags:**

**curr\_data.extend(test\_datasets[i][(j-lags):j].to\_numpy().tolist())**

**else:**

**curr\_data.extend(training\_datasets[i][-(lags-j):].to\_numpy().tolist())**

**curr\_data.extend(test\_datasets[i][0:j].to\_numpy().tolist())**

**curr\_test\_X.append(curr\_data)**

**test\_X.append(numpy.array(curr\_test\_X))**

**# recording dependent variables of the training datase**

**training\_y\_arr.append(training\_y)**

**# recording training and testing batches**

**training\_X\_arr.append(training\_X)**

**test\_X\_arr.append(test\_X)**

**# array for storing all LSTM predictions results**

**LSTM\_predictions\_for\_all\_lags = []**

**# for each lag option**

**for l in range(len(lags\_arr)):**

**LSTM\_predictions = []**

**print('{} lags:'.format(lags\_arr[l]))**

**# for each token building LSTM models and predicting test values**

**for i in range(number\_of\_tokens):**

**# building an LSTM model**

**training\_data = training\_X\_arr[l][i]**

**model = Sequential()**

**model.add(LSTM(64, input\_shape=(training\_data.shape[1], training\_data.shape[2]), return\_sequences=False))**

**model.add(Dense(1))**

**model.compile(loss='mae', optimizer='adam')**

**# training the LSTM model with the test dataset (test batches and dependent variables of the training datase)**

**earlyStopping = EarlyStopping(monitor='val\_loss', mode='min', verbose=1, patience=50)**

**model.fit(training\_data, training\_y\_arr[l][i], epochs=1000, batch\_size=64,**

**validation\_split=0.9, verbose=2, shuffle=False, callbacks=[earlyStopping])**

**# making predictions by the LSTM model**

**yhat = model.predict(test\_X\_arr[l][i])**

**# recording predictions**

**LSTM\_predictions.append(yhat.flatten('C'))**

**print('{} - done\n\n'.format(token\_symbols[i]))**

**# recording all predictions**

**LSTM\_predictions\_for\_all\_lags.append(LSTM\_predictions)**

**"""#Calculate performance of ARIMA and LSTM"""**

**def mean\_absolute\_percentage\_error(test\_data, predicted\_data):**

**return numpy.mean(numpy.abs((test\_data - predicted\_data) / test\_data)) \* 100**

**def calculate\_performance(predictions, test\_data, scaler):**

**# rescaling the predictions to normal scale**

**dataframe\_for\_rescaling = DataFrame({"col1":predictions, "col2":predictions, "col3":predictions,**

**"col4":predictions, "col5":predictions, "col6":predictions})**

**dataframe\_scaled\_back = scaler.inverse\_transform(dataframe\_for\_rescaling)**

**predictions\_scaled\_back = DataFrame(dataframe\_scaled\_back)[3].to\_numpy()**

**# rescaling test dataset back to normal scale**

**dataframe\_for\_rescaling = DataFrame({"col1":test\_data, "col2":test\_data, "col3":test\_data,**

**"col4":test\_data, "col5":test\_data, "col6":test\_data})**

**dataframe\_scaled\_back = scaler.inverse\_transform(dataframe\_for\_rescaling)**

**test\_y\_scaled\_back = DataFrame(dataframe\_scaled\_back)[3].to\_numpy()**

**# calculating performance metrics**

**RMSE = mean\_squared\_error(test\_y\_scaled\_back, predictions\_scaled\_back, squared=False) # Root mean square error**

**MAE = mean\_absolute\_error(test\_y\_scaled\_back, predictions\_scaled\_back) # Mean absolute error**

**MAPE = mean\_absolute\_percentage\_error(test\_y\_scaled\_back, predictions\_scaled\_back) # Mean absolute percentage error**

**return [RMSE, MAE, MAPE]**

**# array holding all results**

**all\_results = []**

**# calculating and saving performance results for each lag option**

**for l in range(len(lags\_arr)):**

**lags = lags\_arr[l]**

**print('{} lags:'.format(lags))**

**results = DataFrame(columns=['Token', 'ARIMA\_RMSE','LSTM\_RMSE', 'ARIMA\_MAE', 'LSTM\_MAE', 'ARIMA\_MAPE', 'LSTM\_MAPE'])**

**# calculating and saving performance results for each token**

**for i in range(number\_of\_tokens):**

**# calculating performance for a token**

**test\_data = test\_datasets[i]['Close'].to\_numpy()**

**ARIMA\_performance = calculate\_performance(ARIMA\_predictions\_for\_all\_lags[l][i], test\_data, scalers[i])**

**LSTM\_performance = calculate\_performance(LSTM\_predictions\_for\_all\_lags[l][i], test\_data, scalers[i])**

**# recording results for a token**

**results = results.append({'Token': token\_symbols[i],**

**'ARIMA\_RMSE': ARIMA\_performance[0], 'LSTM\_RMSE': LSTM\_performance[0],**

**'ARIMA\_MAE': ARIMA\_performance[1], 'LSTM\_MAE': LSTM\_performance[1],**

**'ARIMA\_MAPE': ARIMA\_performance[2], 'LSTM\_MAPE': LSTM\_performance[2]},**

**ignore\_index=True)**

**# recording all results**

**all\_results.append(results)**

**print(results)**

**print()**

**# downloading results as a file (optional)**

**from google.colab import drive**

**from google.colab import files**

**drive.mount('/drive')**

**for i in range(len(lags\_arr)):**

**fileName = 'results\_ARIMA\_LSTM\_{}\_lags.csv'.format(lags\_arr[i])**

**all\_results[i].to\_csv(fileName)**

**files.download(fileName)**

**# a method for plotting results**

**def plotting\_results(token\_symbols, ARIMA\_series, LSTM\_series, metrics\_name, number\_of\_lags):**

**# locations of the labels**

**x = numpy.arange(len(token\_symbols))**

**# width of a bars**

**width = 0.35**

**# passing data for the chart**

**fig, ax = pyplot.subplots()**

**series1 = ax.bar(x - width/2, ARIMA\_series, width, label='ARIMA')**

**series2 = ax.bar(x + width/2, LSTM\_series, width, label='LSTM')**

**# adding labels, title, and formatting**

**ax.set\_ylabel('Scaled logarithms of {}'.format(metrics\_name))**

**ax.set\_title('{} results for {} lags'.format(metrics\_name, number\_of\_lags))**

**ax.set\_xticks(x)**

**ax.set\_xticklabels(token\_symbols)**

**ax.legend()**

**pyplot.rcParams["figure.figsize"] = (12,3)**

**pyplot.xticks(rotation = 90)**

**pyplot.show()**

**# plotting charts with results for each lag option**

**for i in range(len(all\_results)):**

**# selecting list of tokens from the results**

**token\_symbols = all\_results[i]['Token']**

**number\_of\_tokens = len(token\_symbols)**

**# concatenating ARIMA and LSTM results for scaling for better data visualisation**

**arima = all\_results[i][['ARIMA\_RMSE', 'ARIMA\_MAE', 'ARIMA\_MAPE']].to\_numpy()**

**lstm = all\_results[i][['LSTM\_RMSE', 'LSTM\_MAE', 'LSTM\_MAPE']].to\_numpy()**

**concatendated\_results = DataFrame(numpy.concatenate((arima, lstm)), columns=['RMSE', 'MAE', 'MAPE'])**

**# computing logarithm of the results, so the scale is less diverge**

**log\_results = numpy.log(concatendated\_results)**

**# scaling the results data so it can be visualised on the same chart**

**scaler = MinMaxScaler()**

**scaledResults = DataFrame(scaler.fit\_transform(log\_results), columns=['RMSE', 'MAE', 'MAPE'])**

**# separating ARIMA and LSTM scaled results**

**ARIMA\_scaled\_results = scaledResults[:number\_of\_tokens]**

**LSTM\_scaled\_results = scaledResults[number\_of\_tokens:]**

**# Plotting charts**

**print('Results for {} lags:'.format(lags\_arr[i]))**

**plotting\_results(token\_symbols, ARIMA\_scaled\_results['RMSE'], LSTM\_scaled\_results['RMSE'], 'RMSE', lags\_arr[i])**

**plotting\_results(token\_symbols, ARIMA\_scaled\_results['MAE'], LSTM\_scaled\_results['MAE'], 'MAE', lags\_arr[i])**

**plotting\_results(token\_symbols, ARIMA\_scaled\_results['MAPE'], LSTM\_scaled\_results['MAPE'], 'MAPE', lags\_arr[i])**

**print('\n\n')**